Using H2O Driverless AI

Patrick Hall, Megan Kurka & Angela Bartz

http://docs.h2o.ai

May 2025: Version 2.1.0

Using H2O Driverless AI
by Patrick Hall, Megan Kurka & Angela Bartz

Published by H2O.ai, Inc.
2307 Leghorn St.
Mountain View, CA 94043

©2017-2023 H2O.ai, Inc. All rights reserved.

May 2025: Version 2.1.0

Photos by ©H2O.ai, Inc.

All copyrights belong to their respective owners.
While every precaution has been taken in the
preparation of this book, the publisher and
authors assume no responsibility for errors or
omissions, or for damages resulting from the
use of the information contained herein.

Printed in the United States of America.

Contents

1 Overview **7**
1.1 Citation 7
1.2 Have Questions? 7

2 Why Driverless AI? **8**

3 Key Features **9**

4 Supported Algorithms **11**

5 Installing and Upgrading Driverless AI **13**

6 Launching Driverless AI **14**
6.1 Messages 14

7 The Datasets Page **15**
7.1 Adding Datasets 15
7.2 Dataset Details 18
7.2.1 Dataset Details Page 18
7.2.2 Dataset Rows Page 20
7.2.3 Modify by Recipe 20
7.2.4 Downloading Datasets 21
7.3 Splitting Datasets 22
7.4 Visualizing Datasets 23
7.4.1 The Visualization Page 23

8 Running an Experiment **27**
8.1 Before You Begin 27
8.2 New Experiment 27
8.3 Completed Experiment 31
8.3.1 Completed Experiment Actions 31
8.3.2 Experiment Insights and Scores 32
8.4 Model Insights 33
8.5 Model Scores 33
8.5.1 Experiment Summary 35
8.6 Viewing Experiments 38
8.6.1 Checkpointing, Rerunning, and Retraining 38
8.6.2 Deleting Experiments 41

9 Diagnosing a Model **41**

10 Project Workspace **43**
10.1 Linking Datasets 44

10.1.1 Selecting Datasets 45
10.2 Linking Experiments 45
10.2.1 New Experiments 46
10.2.2 Checkpointing Experiments 46
10.3 The Experiments Leaderboard 46
10.3.1 Leaderboard Scoring 47
10.3.2 Comparing Experiments 48
10.4 Unlinking Data on a Projects Page 50
10.5 Deleting Projects 50

11 Leaderboards **51**
11.1 Creating a Leaderboard 51

12 Interpreting a Model **53**
12.1 Interpret this Model button - Regular Experiments 53
12.2 Interpret this Model button - Time-Series Experiments 54
12.2.1 Multi-Group Time Series MLI 54
12.2.2 Single Time Series MLI 56
12.3 Model Interpretation - Driverless AI Models 58
12.4 Model Interpretation - External Models 59
12.5 Interpretation Expert Settings 59
12.6 Understanding the Model Interpretation Page 60
12.6.1 Summary Tab 60
12.6.2 DAI Model Tab 60
12.6.3 Surrogate Model Tab 61
12.6.4 Dashboard and Actions Buttons 61
12.6.5 DAI Model Tab Plots 63
12.6.6 Surrogate Model Tab Plots 77
12.7 General Considerations 85
12.7.1 Machine Learning and Approximate Explanations . . . 85
12.7.2 The Multiplicity of Good Models in Machine Learning . 86
12.7.3 Expectations for Consistency Between Explanatory Techniques 86

13 Viewing Explanations **87**

14 Score on Another Dataset **88**

15 Transform Another Dataset **88**

16 The Driverless AI Scoring Pipelines **90**
16.1 Visualize the Scoring Pipeline 90
16.2 Which Pipeline Should I Use? 92
16.3 Driverless AI Standalone Python Scoring Pipeline 93

16.3.1 Python Scoring Pipeline Files 93
16.3.2 Quick Start - Recommended Method 93
16.3.3 Quick Start - Alternative Method 94
16.3.4 The Python Scoring Module 97
16.3.5 The Scoring Service 98
16.3.6 Python Scoring Pipeline FAQ 100
16.3.7 Troubleshooting Python Environment Issues 101
16.4 Driverless AI MLI Standalone Scoring Package 102
16.4.1 MLI Python Scoring Package Files 102
16.4.2 Quick Start - Recommended Method 103
16.4.3 Quick Start - Alternative Method 104
16.4.4 Prerequisites 104
16.4.5 MLI Python Scoring Module 105
16.4.6 K-LIME vs Shapley Reason Codes 106
16.4.7 MLI Scoring Service Overview 106
16.5 Driverless AI MOJO Scoring Pipeline 109
16.5.1 Prerequisites 109
16.5.2 MOJO Scoring Pipeline Files 110
16.5.3 Quickstart 110
16.5.4 Execute the MOJO from Java 111
16.5.5 MOJO Scoring Pipeline - C++ Solution 113
16.5.5.1 Downloading the Scoring Pipeline Runtimes . 113

17 Deployment 116
17.1 Additional Resources 117
17.2 Deployments Overview Page 117
17.3 AWS Lambda Deployment 117
17.3.1 Driverless AI Prerequisites 117
17.3.2 AWS Access Permissions Prerequisites 117
17.3.3 Deploying the Lambda 119
17.3.4 Testing the Lambda Deployment 120
17.3.5 AWS Deployment Issues 121
17.4 REST Server Deployment 122
17.4.1 Prerequisites 122
17.4.2 Deploying on REST Server 122
17.4.3 Testing the REST Server Deployment 124
17.4.4 REST Server Deployment Issues 125

18 About Driverless AI Transformations 126
18.1 Numeric Transformers 126
18.2 Time Series Experiments Transformers 127
18.3 Categorical Transformers (String) 128
18.4 Text Transformers (String) 129

18.5 Time Transformers (Date, Time) 130

19 Monitoring and Logging **130**
19.1 Monitoring Pending Jobs 130
19.1.1 Failed Jobs 131
19.2 Logs 131
19.3 Sending Logs to H2O 134

20 References **135**

21 Authors **136**

1 Overview

H2O Driverless AI is an artificial intelligence (AI) platform for automatic machine learning. Driverless AI automates some of the most difficult data science and machine learning workflows such as feature engineering, model validation, model tuning, model selection and model deployment. It aims to achieve highest predictive accuracy, comparable to expert data scientists, but in much shorter time thanks to end-to-end automation. Driverless AI also offers automatic visualizations and machine learning interpretability (MLI). Especially in regulated industries, model transparency and explanation are just as important as predictive performance. Modeling pipelines (feature engineering and models) are exported (in full fidelity, without approximations) both as Python modules and as pure Java standalone scoring artifacts.

Driverless AI runs on commodity hardware. It was also specifically designed to take advantage of graphical processing units (GPUs), including multi-GPU workstations and servers such as IBM's Power9-GPU AC922 server and the NVIDIA DGX-1 for order-of-magnitude faster training.

This document describes how to use H2O Driverless AI UI and is updated periodically. To view the latest Driverless AI User Guide, please go to `http://docs.h2o.ai`. To view an example for running the Driverless AI Python Client, please refer to Appendix A: The Python Client in the Driverless AI User Guide.

For more information about Driverless AI, please see `https://www.h2o.ai/driverless-ai/`. For a third-party review, please see `https://www.infoworld.com/article/3236048/machine-learning/review-h2oai-automates-machine-learning.html`.

1.1 Citation

To cite this booklet, use the following: Hall, P., Kurka, M., and Bartz, A. (Sept 2018). *Using H2O Driverless AI*. `http://docs.h2o.ai`

1.2 Have Questions?

If you have questions about using Driverless AI, we recommend reviewing the FAQ in the Driverless AI User Guide. If after reviewing the FAQ you have additional questions, then you can post them on Stack Overflow using the **driverless-ai** tag at `http://stackoverflow.com/questions/tagged/driverless-ai`.

2 Why Driverless AI?

Over the last several years, machine learning has become an integral part of many organizations' decision-making processes at various levels. With not enough data scientists to fill the increasing demand for data-driven business processes, H2O.ai offers Driverless AI, which automates several time consuming aspects of a typical data science workflow, including data visualization, feature engineering, predictive modeling, and model explanation.

H2O Driverless AI is a high-performance, GPU-enabled computing platform for automatic development and rapid deployment of state-of-the-art predictive analytics models. It reads tabular data from plain text sources, Hadoop, or S3 buckets and automates data visualization and building predictive models. Driverless AI targets business applications such as loss-given-default, probability of default, customer churn, campaign response, fraud detection, anti-money-laundering, demand forecasting, and predictive asset maintenance models. (Or in machine learning parlance: common regression, binomial classification, and multinomial classification problems.)

How do you frame business problems in a data set for Driverless AI?

The data that is read into Driverless AI must contain one entity per row, like a customer, patient, piece of equipment, or financial transaction. That row must also contain information about what you will be trying to predict using similar data in the future, like whether that customer in the row of data used a promotion, whether that patient was readmitted to the hospital within thirty days of being released, whether that piece of equipment required maintenance, or whether that financial transaction was fraudulent. (In data science speak, Driverless AI requires "labeled" data.) Driverless AI runs through your data many, many times looking for interactions, insights, and business drivers of the phenomenon described by the provided dataset. Driverless AI can handle simple data quality problems, but it currently requires all data for a single predictive model to be in the same dataset, and that dataset must have already undergone standard ETL, cleaning, and normalization routines before being loaded into Driverless AI.

How do you use Driverless AI results to create commercial value?

Commercial value is generated by Driverless AI in a few ways.

- Driverless AI empowers data scientists or data analysts to work on projects faster and more efficiently by using automation and state-of-the-art computing power to accomplish tasks in just minutes or hours instead of the weeks or months that it can take humans.

- Like in many other industries, automation leads to standardization of business processes, enforces best practices, and eventually drives down the cost of delivering the final product - in this case a predictive model.
- Driverless AI makes deploying predictive models easy - typically a difficult step in the data science process. In large organizations, value from predictive modeling is typically realized when a predictive model is moved from a data analyst's or data scientist's development environment into a production deployment setting. In this setting, the model is running on live data and making quick and automatic decisions that make or save money. Driverless AI provides both Java- and Python-based technologies to make production deployment simpler.

Moreover, the system was designed with interpretability and transparency in mind. Every prediction made by a Driverless AI model can be explained to business users, so the system is viable even for regulated industries.

Visit https://www.h2o.ai/products/h2o-driverless-ai/ to download your free 21-day evaluation copy.

3 Key Features

Below are some of the key features available in Driverless AI.

Flexibility of Data and Deployment: Driverless AI works across a variety of data sources including Hadoop HDFS, Amazon S3, and more. Driverless AI can be deployed everywhere including all clouds (Microsoft Azure, AWS, Google Cloud) and on premises on any system, but it is ideally suited for systems with GPUs, including IBM Power 9 with GPUs built in.

NVIDIA GPU Acceleration: Driverless AI is optimized to take advantage of GPU acceleration to achieve up to 40X speedups for automatic machine learning. It includes multi-GPU algorithms for XGBoost, GLM, K-Means, and more. GPUs allow for thousands of iterations of model features and optimizations.

Automatic Data Visualization (Autovis): For datasets, Driverless AI automatically selects data plots based on the most relevant data statistics, generates visualizations, and creates data plots that are most relevant from a statistical perspective based on the most relevant data statistics. These visualizations help users get a quick understanding of their data prior to starting the model building process. They are also useful for understanding the composition of very large datasets and for seeing trends or even possible issues, such as large numbers of missing values or significant outliers that could impact modeling results.

Automatic Feature Engineering: Feature engineering is the secret weapon that advanced data scientists use to extract the most accurate results from

algorithms. H2O Driverless AI employs a library of algorithms and feature transformations to automatically engineer new, high value features for a given dataset. Included in the interface is an easy-to-read variable importance chart that shows the significance of original and newly engineered features.

Automatic Model Documentation: To explain models to business users and regulators, data scientists and data engineers must document the data, algorithms, and processes used to create machine learning models. Driverless AI provides an AutoDoc (Autodoc) for each experiment, relieving the user from the time-consuming task of documenting and summarizing their workflow used when building machine learning models. The AutoDoc includes details about the data used, the validation schema selected, model and feature tuning, and the final model created. With this capability in Driverless AI, practitioners can focus more on drawing actionable insights from the models and save weeks or even months in development, validation, and deployment process. Driverless AI also provides a number of `autodoc_` configuration options, giving users even more control over output of the AutoDoc.

Time Series Forecasting: Time series forecasting is one of the biggest challenges for data scientists. These models address key use cases, including demand forecasting, infrastructure monitoring, and predictive maintenance. Driverless AI delivers superior time series capabilities to optimize for almost any prediction time window. Driverless AI incorporates data from numerous predictors, handles structured character data and high-cardinality categorical variables, and handles gaps in time series data and other missing values.

NLP with TensorFlow: Text data can contain critical information to inform better predictions. Driverless AI automatically converts short text strings into features using powerful techniques like TFIDF. With TensorFlow, Driverless AI can also process larger text blocks and build models using all available data to solve business problems like sentiment analysis, document classification, and content tagging.

Automatic Scoring Pipelines: For completed experiments, Driverless AI automatically generates both Python scoring pipelines and new ultra-low latency automatic scoring pipelines. The new automatic scoring pipeline is a unique technology that deploys all feature engineering and the winning machine learning model in a highly optimized, low-latency, production-ready Java code that can be deployed anywhere.

Machine Learning Interpretability (MLI): Driverless AI provides robust interpretability of machine learning models to explain modeling results in a human-readable format. In the MLI view, Driverless AI employs a host of different techniques and methodologies for interpreting and explaining the results of its models. A number of charts are generated automatically, including K-LIME, Shapley, Variable Importance, Decision Tree Surrogate, Partial Dependence,

Individual Conditional Expectation (ICE) and more. Additionally, you can download a CSV of LIME and Shapley reasons codes from this view.

Automatic Reason Codes: In regulated industries, an explanation is often required for significant decisions relating to customers (for example, credit denial). Reason codes show the key positive and negative factors in a model's scoring decision in a simple language. Reasons codes are also useful in other industries, such as healthcare, because they can provide insights into model decisions that can drive additional testing or investigation.

Custom Recipe Support: Driverless AI allows you to import custom recipes for MLI algorithms, feature engineering (transformers), scorers, and configuration. You can use your custom recipes in combination with or instead of all built-in recipes. This allows you to have greater influence over the Driverless AI Automatic ML pipeline and gives you control over the optimization choices that Driverless AI makes.

4 Supported Algorithms

Constant Model

A Constant Model predicts the same constant value for any input data. The constant value is computed by optimizing the given scorer. For example, for MSE/RMSE, the constant is the (weighted) mean of the target column. For MAE, it is the (weighted) median. For other scorers like MAPE or custom scorers, the constant is found with an optimization process. For classification problems, the constant probabilities are the observed priors.

A constant model is meant as a baseline reference model. If it ends up being used in the final pipeline, a warning will be issued because that would indicate a problem in the dataset or target column (e.g., when trying to predict a random outcome).

Decision Tree

A Decision Tree is a single (binary) tree model that splits the training data population into sub-groups (leaf nodes) with similar outcomes. No row or column sampling is performed, and the tree depth and method of growth (depth-wise or loss-guided) is controlled by hyper-parameters.

FTRL

Follow the Regularized Leader (FTRL) is a DataTable implementation ([13]) of the FTRL-Proximal online learning algorithm proposed in "Ad click prediction: a view from the trenches" ([10]). This implementation uses a hashing trick and Hogwild approach ([11]) for parallelization. FTRL can do binomial and

multinomial classification, binomial and multinomial regressions, as well as regression for continuous targets.

GLM

Generalized Linear Models (GLM) estimate regression models for outcomes following exponential distributions. GLMs are an extension of traditional linear models. They have gained popularity in statistical data analysis due to:

- the flexibility of the model structure unifying the typical regression methods (such as linear regression and logistic regression for binary classification)
- the recent availability of model-fitting software
- the ability to scale well with large datasets

Isolation Forest

Isolation Forest is useful for identifying anomalies or outliers in data. Isolation Forest isolates observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of that selected feature. This split depends on how long it takes to separate the points. Random partitioning produces noticeably shorter paths for anomalies. When a forest of random trees collectively produces shorter path lengths for particular samples, they are highly likely to be anomalies.

LightGBM

LightGBM is a gradient boosting framework developed by Microsoft that uses tree based learning algorithms. It was specifically designed for lower memory usage and faster training speed and higher efficiency. Similar to XGBoost, it is one of the best gradient boosting implementations available. It is also used for fitting Random Forest models inside of Driverless AI.

RuleFit

The RuleFit ([3]) algorithm creates an optimal set of decision rules by first fitting a tree model, and then fitting a Lasso (L1-regularized) GLM model to create a linear model consisting of the most important tree leaves (rules).

TensorFlow

TensorFlow is an open source software library for performing high performance numerical computation. Driverless AI includes a TensorFlow NLP recipe based on CNN Deeplearning models.

XGBoost

XGBoost is a supervised learning algorithm that implements a process called boosting to yield accurate models. Boosting refers to the ensemble learning technique of building many models sequentially, with each new model attempting

to correct for the deficiencies in the previous model. In tree boosting, each new model that is added to the ensemble is a decision tree. XGBoost provides parallel tree boosting (also known as GBDT, GBM) that solves many data science problems in a fast and accurate way. For many problems, XGBoost is one of the best gradient boosting machine (GBM) frameworks today.

5 Installing and Upgrading Driverless AI

Installation and upgrade steps are provided in the Driverless AI User Guide. The installation steps vary based on your platform and require a license key. Contact sales@h2o.ai for information on how to purchase a Driverless AI license. Or visit `https://www.h2o.ai/download/` to obtain a free 21-day trial license.

For the best (and intended-as-designed) experience, install Driverless AI on modern data center hardware with GPUs and CUDA support. Use Pascal or Volta GPUs with maximum GPU memory for best results. (Note the older K80 and M60 GPUs available in EC2 are supported and very convenient, but not as fast.)

To simplify cloud installation, Driverless AI is provided as an AMI. To simplify local installation, Driverless AI is provided as a Docker image. For the best performance, including GPU support, use the Docker '–gpus all' flag. For a lower-performance experience without GPUs, use regular docker (with the same docker image). RPM and Tar.sh options are available for native Driverless AI installations on RedHat 7, CentOS 7, and SLES 12 operating systems. And finally, a DEB installation is available for Ubuntu 16.04 environments and on Windows 10 using Windows Subsystem for Linux (WSL).

For native installs (rpm, deb, tar.sh), Driverless AI requires a minimum of 5 GB of system memory in order to start experiments and a minimum of 5 GB of disk space in order to run a small experiment. Note that these limits can be changed in the config.toml file. We recommend that you have lots of system CPU memory (64 GB or more) and 1 TB of free disk space available.

For Docker installs, we recommend 1TB of free disk space. Driverless AI uses approximately 38 GB. In addition, the unpacking/temp files require space on the same Linux mount /var during installation. Once DAI runs, the mounts from the Docker container can point to other file system mount points.

If you are running Driverless AI with GPUs, be sure that your GPU has compute capability of at least 3.5 and at least 4GB of RAM. If these requirements are not met, then Driverless AI will switch to CPU-only mode.

Driverless AI supports unvalidated, none, local, Client Certificate, LDAP, mTLS, OpenID, and PAM authentication. Authentication can be configured by setting

environment variables or via a config.toml file. Refer to the Setting Environment Variables section in the User Guide.

Driverless AI also supports HDFS, S3, Azure Blob Store, BlueData DataTap, Google Cloud Storage, Google Big Query, JDBC, KDB+, Minio, and Snowflake access. Support for these data sources can be configured by setting environment variables for the data connectors or via a config.toml file. Refer to the Data Connectors section in the User Guide for more information.

6 Launching Driverless AI

Driverless AI is tested most extensively on Chrome and Firefox. For the best user experience, we recommend using the latest version of Chrome. You may encounter issues if you use other browsers or earlier versions of Chrome and/or Firefox.

1. After Driverless AI is installed and started, open a browser and navigate to `<driverless-ai-host-machine>:12345`.

2. The first time you log in to Driverless AI, you will be prompted to read and accept the Evaluation Agreement. You must accept the terms before continuing. Review the agreement, then click **I agree to these terms** to continue.

3. Log in by entering unique credentials. For example:

   ```
   Username: h2oai
   Password: h2oai
   ```

   Note that these credentials do not restrict access to Driverless AI; they are used to tie experiments to users. If you log in with different credentials, for example, then you will not see any previously run experiments.

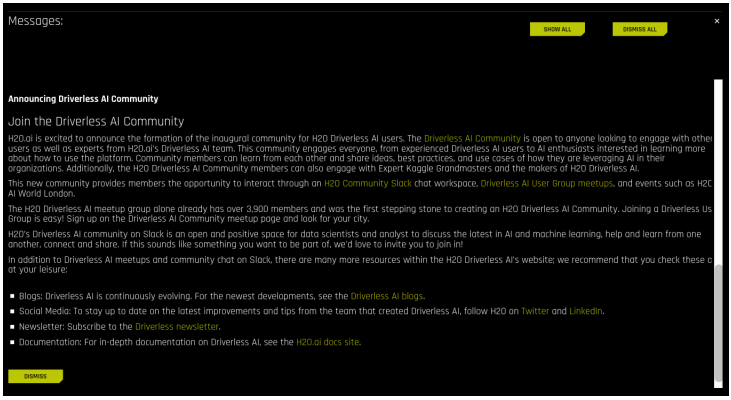
4. As with accepting the Evaluation Agreement, the first time you log in, you will be prompted to enter your License Key. Paste the License Key into the **License Key** entry field, and then click **Save** to continue. This license key will be saved in the host machine's **/license** folder.

   Note: Contact sales@h2o.ai for information on how to purchase a Driverless AI license.

Upon successful completion, you will be ready to add datasets and run experiments.

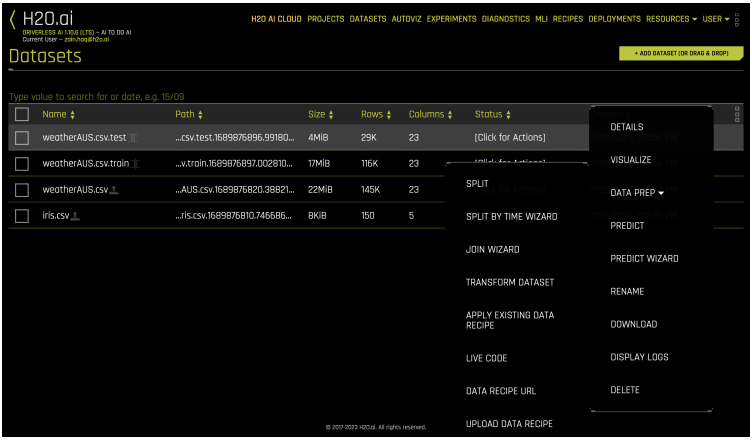
6.1 Messages

A **Messages** menu option is available when you launch Driverless AI. Click this to view news and upcoming events regarding Driverless AI.



7 The Datasets Page

The Datasets Overview page is the Driverless AI Home page. This shows all datasets that have been imported. Note that the first time you log in, this list will be empty.



7.1 Adding Datasets

Driverless AI supports the following dataset file formats: arff, avro, bin, bz2, csv, dat, feather, gz, jay, orc, parquet, pkl, tgz, tsv, txt, xls, xlsx, xz, and zip (see notes for csv, jay, orc, and parquet below).

Notes

- CSV in UTF-16 encoding is only supported when implemented with a byte order mark (BOM). If a BOM is not present, the dataset is read as UTF-8.
- For Parquet file formats, if you select to import multiple Parquet files, those files will be imported as multiple datasets. If you select a folder of Parquet files, the folder will be imported as a single dataset. Tools like Spark/Hive export data as multiple Parquet files that are stored in a directory with a user-defined name. For example, if you export with `Spark dataFrame.write.parquet("/data/big_parquet_dataset")`, Spark creates a folder **/data/big_parquet_dataset**, which will contain multiple ORC or Parquet files (depending on the number of partitions in the input dataset) and metadata. Exporting ORC files produces a similar result.
- For ORC and Parquet file formats, you may receive a "Failed to ingest binary file with ORC / Parquet: lists with structs are not supported" error when ingesting an ORC or Parquet file that has a struct as an element of an array. This is because PyArrow cannot handle a struct that's an element of an array.
- A workaround to flatten Parquet files is provided in Sparkling Water. Refer to our Sparkling Water solution for more information.
- You can create new datasets from Python script files (custom recipes) by selecting **Data Recipe URL** or **Upload Data Recipe** from the **Add Dataset (or Drag & Drop)** dropdown menu. If you select the **Data Recipe URL** option, the URL must point to either an HTML or raw version of the file, a GitHub repository or tree, or a local file. In addition, you can create a new dataset by modifying an existing dataset with a custom recipe. Refer to the **Modify by Recipe** section below for more information. Note that datasets created or added from recipes will be saved as .jay files.

You can add datasets using one of the following methods:

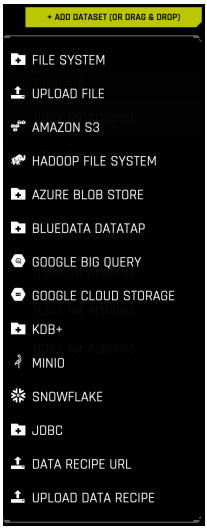
- Drag and drop files from your local machine directly onto this page. Note that this method currently works for files that are less than 10 GB.

or

1. Click the **Add Dataset or Drag and Drop** button to upload or add a dataset.

   Notes:

- Upload File, File System, HDFS, S3, Data Recipe URL, and Upload Data Recipe are enabled by default. These can be disabled by removing them from the `enabled_file_systems` setting in the config.toml file.
- If File System is disabled, Driverless AI will open a local filebrowser by default.
- If Driverless AI was started with data connectors enabled for Azure Blob Store, BlueData Datatap, Google Big Query, Google Cloud Storage, Hive, KDB+, Minio, Snowflake, or JDBC, then these options will appear in the **Add Dataset (or Drag & Drop)** dropdown menu Refer to the Data Connectors section in the Driverless AI User Guide for more information.
- When specifying to add a dataset using **Data Recipe URL**, the URL must point to either an HTML or raw version of the file, a GitHub repository or tree, or a local file. When adding or uploading datasets via recipes, the dataset will be saved as a .jay file.

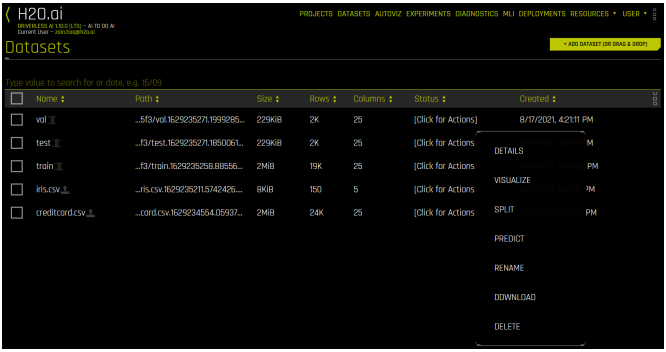


Notes:

- Datasets must be in delimited text format.
- Driverless AI can detect the following separators: ,|;t
- When importing a folder, the entire folder and all of its contents are read into Driverless AI as a single file.

- When importing a folder, all of the files in the folder must have the same columns.

Upon completion, the datasets will appear in the Datasets Overview page. Click on a dataset or click the **[Click for Actions]** button to open a submenu. From this menu, you can specify to view Details, Split, Visualize, Predict, or Delete a dataset. You can also delete an unused dataset by hovering over it, clicking the **X** button, and then confirming the delete. **Note**: You cannot delete a dataset that was used in an active experiment. You have to delete the experiment first.



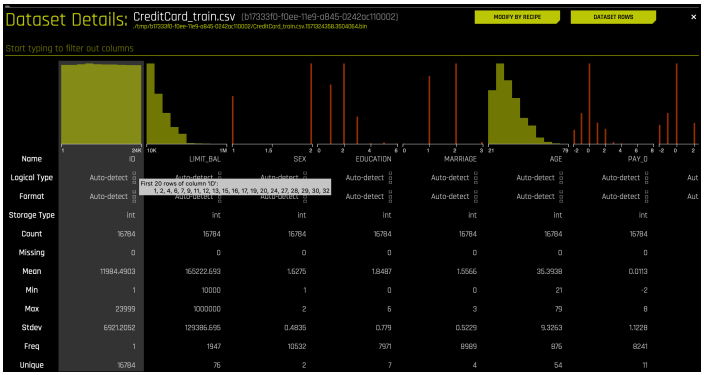
7.2 Dataset Details

To view a summary of a dataset or to preview the dataset, click on the dataset or select the **[Click for Actions]** button beside the dataset that you want to view, and then click **Details** from the submenu that appears. This opens the Dataset Details page.

7.2.1 Dataset Details Page

The Dataset Details page provides a summary of the dataset. This summary lists each column that is included in the dataset along with the type, the count, the mean, minimum, maximum, standard deviation, frequency, and the number of unique values. **Note**: Driverless AI recognizes the following column types: integer, string, real, and boolean. Date columns are given a `str` type.

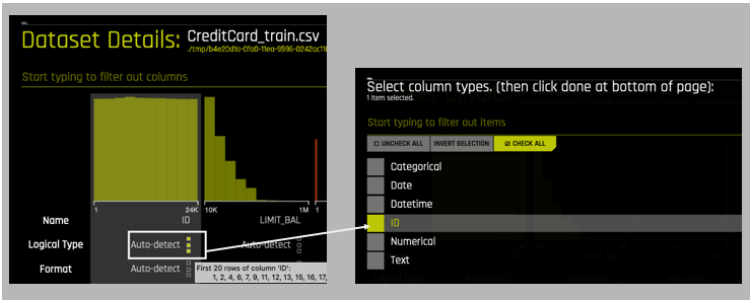
Hover over the top of a column to view a summary of the first 20 rows of that column.



To view information for a specific column, type the column name in the field above the graph.

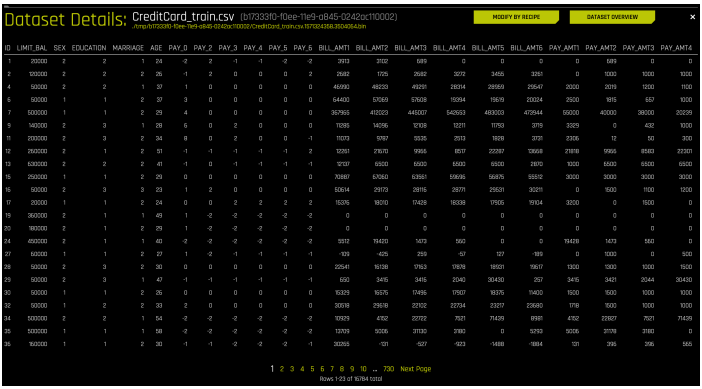
Changing a Column Type

Driverless AI also allows you to change a column type. If a column's data type or distribution does not match the manner in which you want the column to be handled during an experiment, changing the **Logical Type** can help to make the column fit better. For example, an integer zip code can be changed into a categorical so that it is only used with categorical-related feature engineering. For Date and Datetime columns, use the **Format** option. To change the Logical Type or Format of a column, click on the group of square icons located to the right of the words **Auto-detect**. (The squares light up when you hover over them with your cursor.) Then select the new column type for that column.



7.2.2 Dataset Rows Page

To switch the view and preview the dataset, click the **Dataset Rows** button in the top right portion of the UI. Then click the **Dataset Overview** button to return to the original view.



7.2.3 Modify by Recipe

The option to create a new dataset by modifying an existing dataset with custom recipes is also available from this page. Scoring pipelines can be created on the new dataset by building an experiment. This feature is useful when you want to make changes to the training data that you would not need to make on the new data you are predicting on. For example, you can change the target column from regression to classification, add a weight column to mark specific training rows as being more important, or remove outliers that you do not want to model on.

Click the **Modify by Recipe** button in the top right portion of the UI and select from the following options:

- **Data Recipe URL**: Load a custom recipe from a URL to use to modify the dataset. The URL must point to either an HTML or raw version of the file, a GitHub repository or tree, or a local file.

- **Upload Data Recipe**: If you have a custom recipe available on your local system, click this button to upload that recipe.

- **Live Code**: Manually enter custom recipe code to use to modify the dataset. Click the **Get Preview** button to preview the code's effect on the dataset, then click **Save** to create a new dataset.

Notes:

- These options are enabled by default. You can disable them by removing `recipe_file` and `recipe_url` from the `enabled_file_systems` configuration option.

- Modifying a dataset with a recipe will not overwrite the original dataset. The dataset that is selected for modification will remain in the list of available datasets in its original form, and the modified dataset will appear in this list as a new dataset.

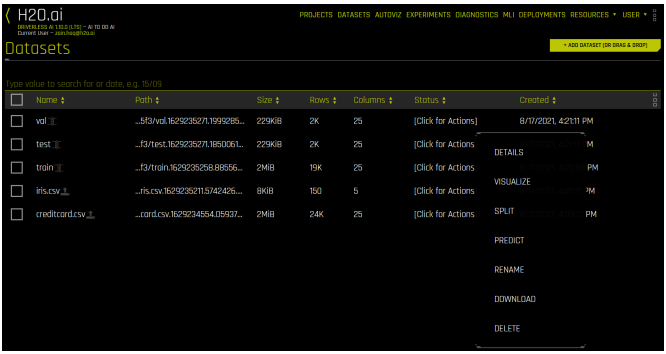
- Changes made to the original dataset through this feature will not be applied to new data that is scored.

7.2.4 Downloading Datasets

In Driverless AI, you can download datasets from the **Datasets Overview** page.

To download a dataset, click on the dataset or select the **[Click for Actions]** button beside the dataset that you want to download, and then select **Download** from the submenu that appears.

Note: The option to download datasets will not be available if the `enable_dataset_do` option is set to `false` when starting Driverless AI. This option can be specified in the config.toml file.

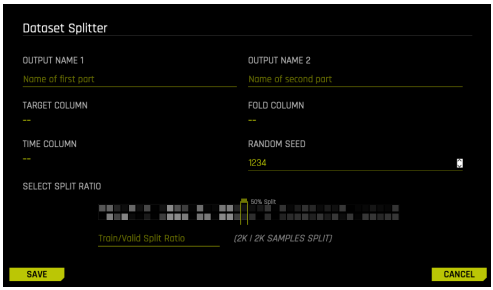


7.3 Splitting Datasets

In Driverless AI, you can split a training dataset into test and validation datasets.

Perform the following steps to split a dataset.

1. On the Datasets page, select the **[Click for Actions]** button beside the dataset that you want to split, and then select **Split** from the submenu that appears.
2. The Dataset Splitter form displays. Specify an Output Name 1 and an Output Name 2 for the first and second part of the split. (For example, you can name one test and one valid.)
3. Optionally specify a Target column (for stratified sampling), a Fold column (to keep rows belonging to the same group together), and/or a Time column.
4. Use the slider to select a split ratio, or enter a value in the Train/Valid Split Ratio field.
5. Click **Save** when you are done.



Upon completion, the split datasets will be available on the Datasets page.

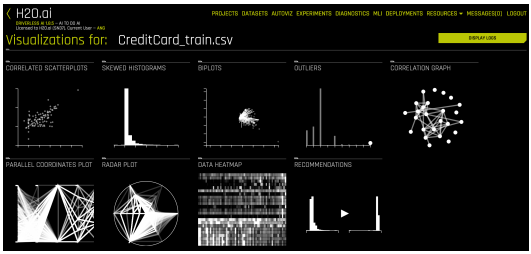
7.4 Visualizing Datasets

Perform one of the following steps to visualize a dataset:

- On the Datasets page, select the **[Click for Actions]** button beside the dataset that you want to view, and then click **Visualize** from the submenu that appears.
- Click the **Autoviz** top menu link to go to the Visualizations list page, click the **New Visualization** button, then select or import the dataset that you want to visualize.

7.4.1 The Visualization Page

The Visualization page shows all available graphs for the selected dataset. Note that the graphs on the Visualization page can vary based on the information in your dataset. You can also view and download logs that were generated during the visualization.



The following is a complete list of available graphs.

- **Correlated Scatterplots**: Correlated scatterplots are 2D plots with large values of the squared Pearson correlation coefficient. All possible scatterplots based on pairs of features (variables) are examined for correlations. The displayed plots are ranked according to the correlation. Some of these plots may not look like textbook examples of correlation. The only criterion is that they have a large value of squared Pearson's r (greater than .95). When modeling with these variables, you may want to leave out variables that are perfectly correlated with others.

  Note that points in the scatterplot can have different sizes. Because Driverless AI aggregates the data and does not display all points, the bigger the point is, the bigger number of exemplars (aggregated points) the plot covers.

- **Spikey Histograms**: Spikey histograms are histograms with huge spikes. This often indicates an inordinate number of single values (usually zeros)

or highly similar values. The measure of "spikeyness" is a bin frequency that is ten times the average frequency of all the bins. You should be careful when modeling (particularly regression models) with spikey variables.

- **Skewed Histograms**: Skewed histograms are ones with especially large skewness (asymmetry). The robust measure of skewness is derived from Groeneveld, R.A. and Meeden, G. (1984), "Measuring Skewness and Kurtosis." The Statistician, 33, 391-399 ([6]). Highly skewed variables are often candidates for a transformation (e.g., logging) before use in modeling. The histograms in the output are sorted in descending order of skewness.

- **Varying Boxplots**: Varying boxplots reveal unusual variability in a feature across the categories of a categorical variable. The measure of variability is computed from a robust one-way analysis of variance (ANOVA). Sufficiently diverse variables are flagged in the ANOVA. A boxplot is a graphical display of the fractiles of a distribution. The center of the box denotes the median, the edges of a box denote the lower and upper quartiles, and the ends of the "whiskers" denote that range of values. Sometimes outliers occur, in which case the adjacent whisker is shortened to the next lower or upper value. For variables (features) having only a few values, the boxes can be compressed, sometimes into a single horizontal line at the median.

- **Heteroscedastic Boxplots**: Heteroscedastic boxplots reveal unusual variability in a feature across the categories of a categorical variable. Heteroscedasticity is calculated with a Brown-Forsythe test: Brown, M. B. and Forsythe, A. B. (1974), "Robust tests for equality of variances." Journal of the American Statistical Association, 69, 364-367. Plots are ranked according to their heteroscedasticity values. A boxplot is a graphical display of the fractiles of a distribution. The center of the box denotes the median, the edges of a box denote the lower and upper quartiles, and the ends of the "whiskers" denote that range of values. Sometimes outliers occur, in which case the adjacent whisker is shortened to the next lower or upper value. For variables (features) having only a few values, the boxes can be compressed, sometimes into a single horizontal line at the median.

- **Biplots**: A Biplot is an enhanced scatterplot that uses both points and vectors to represent structure simultaneously for rows and columns of a data matrix. Rows are represented as points (scores), and columns are represented as vectors (loadings). The plot is computed from the first two principal components of the correlation matrix of the variables (features). You should look for unusual (non-elliptical) shapes in the points that

might reveal outliers or non-normal distributions. And you should look for purple vectors that are well-separated. Overlapping vectors can indicate a high degree of correlation between variables.

- **Outliers**: Variables with anomalous or outlying values are displayed as red points in a dot plot. Dot plots are constructed using an algorithm in Wilkinson, L. (1999). "Dot plots." The American Statistician, 53, 276–281 ([15]). Not all anomalous points are outliers. Sometimes the algorithm will flag points that lie in an empty region (i.e., they are not near any other points). You should inspect outliers to see if they are miscodings or if they are due to some other mistake. Outliers should ordinarily be eliminated from models only when there is a reasonable explanation for their occurrence.

- **Correlation Graph**: The correlation network graph is constructed from all pairwise squared correlations between variables (features). For continuous-continuous variable pairs, the statistic used is the squared Pearson correlation. For continuous-categorical variable pairs, the statistic is based on the squared intraclass correlation (ICC). This statistic is computed from the mean squares from a one-way analysis of variance (ANOVA). The formula is (MSbetween - MSwithin)/(MSbetween + (k - 1)MSwithin), where k is the number of categories in the categorical variable. For categorical-categorical pairs, the statistic is computed from Cramer's V squared. If the first variable has k1 categories and the second variable has k2 categories, then a k1 x k2 table is created from the joint frequencies of values. From this table, we compute a chi-square statistic. Cramer's V squared statistic is then (chi-square / n) / min(k1,k2), where n is the total of the joint frequencies in the table. Variables with large values of these respective statistics appear near each other in the network diagram. The color scale used for the connecting edges runs from low (blue) to high (red). Variables connected by short red edges tend to be highly correlated.

- **Parallel Coordinates Plot**: A Parallel Coordinates Plot is a graph used for comparing multiple variables. Each variable has its own vertical axis in the plot. Each profile connects the values on the axes for a single observation. If the data contain clusters, these profiles will be colored by their cluster number.

- **Radar Plot**: A Radar Plot is a two-dimensional graph that is used for comparing multiple variables. Each variable has its own axis that starts from the center of the graph. The data are standardized on each variable between 0 and 1 so that values can be compared across variables. Each profile, which usually appears in the form of a star, connects the values on the axes for a single observation. Multivariate outliers are represented

by red profiles. The Radar Plot is the polar version of the popular Parallel Coordinates plot. The polar layout enables us to represent more variables in a single plot.

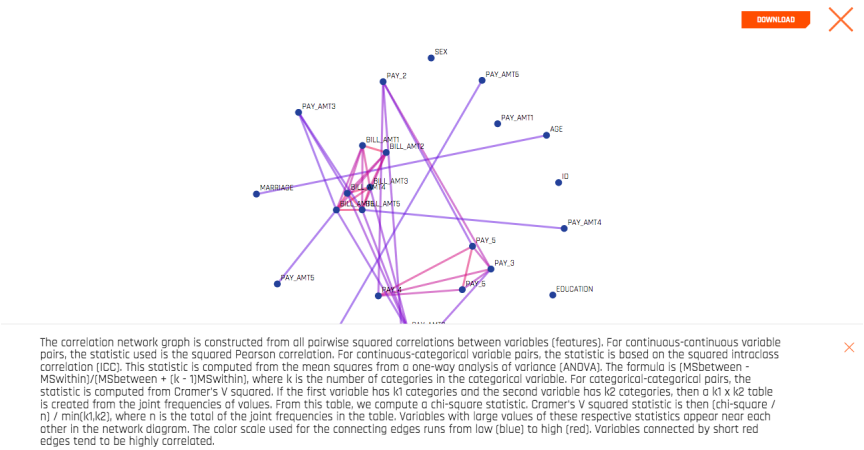
- **Data Heatmap**: The heatmap graphic is constructed from the transposed data matrix. Rows of the heatmap represent variables, and columns represent cases (instances). The data are standardized before display so that small values are yellow and large values are red. The rows and columns are permuted via a singular value decomposition (SVD) of the data matrix so that similar rows and similar columns are near each other.

- **Recommendations**: The recommendations graphic implements the Tukey ladder of powers collection of log, square root, and inverse data transformations, as well as extensions of these three transformers that handle negative values (Yeo-Johnson). For each transformer, transformations are selected by comparing the robust skewness of the transformed column with the robust skewness of the original raw column. When a transformation leads to a relatively low value of skewness, it is recommended.

- **Missing Values Heatmap**: The missing values heatmap graphic is constructed from the transposed data matrix. Rows of the heatmap represent variables and columns represent cases (instances). The data are coded into the values 0 (missing) and 1 (nonmissing). Missing values are colored red and nonmissing values are left blank (white). The rows and columns are permuted via a singular value decomposition (SVD) of the data matrix so that similar rows and similar columns are near each other.

- **Gaps Histogram**: The gaps index is computed using an algorithm of Wainer and Schacht based on work by John Tukey. (Wainer, H. and Schacht, Psychometrika, 43, 2, 203-12.) Histograms with gaps can indicate a mixture of two or more distributions based on possible subgroups not necessarily characterized in the dataset.

The images on this page are thumbnails. You can click on any of the graphs to view and download a full-scale image.



8 Running an Experiment

8.1 Before You Begin

This section describes how to run an experiment using the Driverless AI UI. Before you begin, it is best that you understand the available options that you can specify. Note that only a dataset and a target column are required to be specified, but Driverless AI provides a variety of experiment and expert settings that you can use to build your models. Hover over each option in the UI, or review the Experiments section in the Driverless AI User Guide for information about these options.

After you have a comfortable working knowledge of these options, you are ready to start your own experiment.

8.2 New Experiment

1. Run an experiment by selecting **[Click for Actions]** button beside the dataset that you want to use. Click **Predict** to begin an experiment.

2. The Experiment Settings form displays and auto-fills with the selected dataset. Optionally enter a custom name for this experiment. If you do not add a name, Driverless AI will create one for you.

3. Optionally specify a validation dataset and/or a test dataset.

    - The validation set is used to tune parameters (models, features, etc.). If a validation dataset is not provided, the training data is used (with holdout splits). If a validation dataset is provided, training data is not used for parameter tuning - only for training. A validation dataset

can help to improve the generalization performance on shifting data distributions.

- The test dataset is used for the final stage scoring and is the dataset for which model metrics will be computed against. Test set predictions will be available at the end of the experiment. This dataset is not used during training of the modeling pipeline.

These datasets must have the same number of columns as the training dataset. Also note that if provided, the validation set is not sampled down, so it can lead to large memory usage, even if accuracy=1 (which reduces the train size).

4. Specify the target (response) column. Note that not all explanatory functionality will be available for multinomial classification scenarios (scenarios with more than two outcomes). When the target column is selected, Driverless AI automatically provides the target column type and the number of rows. If this is a classification problem, then the UI shows unique and frequency statistics for numerical columns. If this is a regression problem, then the UI shows the dataset mean and standard deviation values.

   Notes Regarding Frequency:

   - For data imported in versions <= 1.0.19, TARGET FREQ and MOST FREQ both represent the count of the least frequent class for numeric target columns and the count of the most frequent class for categorical target columns.
   - For data imported in versions 1.0.20-1.0.22, TARGET FREQ and MOST FREQ both represent the frequency of the target class (second class in lexicographic order) for binomial target columns; the count of the most frequent class for categorical multinomial target columns; and the count of the least frequent class for numeric multinomial target columns.
   - For data imported in version 1.0.23 (and later), TARGET FREQ is the frequency of the target class for binomial target columns, and MOST FREQ is the most frequent class for multinomial target columns.

5. The next step is to set the parameters and settings for the experiment. (Hover over each option and/or refer to the Experiment Settings section in the Driverless AI User Guide for detailed information about these settings.) You can set the parameters individually, or you can let Driverless AI infer the parameters and then override any that you disagree with. Note that Driverless AI will automatically infer the best settings for Accuracy, Time,

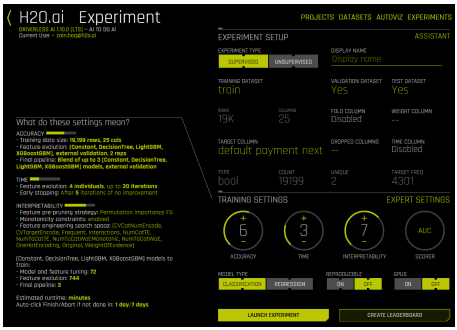
and Interpretability and provide you with an experiment preview based on those suggestions. If you adjust these knobs, the experiment preview will automatically update based on the new settings.

Expert settings (optional):

Optionally specify additional expert settings for the experiment. Refer to the Expert Settings section in the Driverless AI User Guide for detailed information about these settings. Note that the default values for these options are derived from the environment variables in the config.toml file.

Additional settings (optional):

- **Classification** or **Regression** button. Driverless AI automatically determines the problem type based on the response column. Though not recommended, you can override this setting by clicking this button.
- **Reproducible**: Click this button to build this with a random seed.
- **Enable GPUs**: Specify whether to enable GPUs. (Note that this option is ignored on CPU-only systems.)



6. Click **Launch Experiment** to start the experiment.

   The experiment launches with a randomly generated experiment name. You can change this name at anytime during or after the experiment. Mouse over the name of the experiment to view an edit icon, then type in the desired name.

   As the experiment runs, a running status displays in the upper middle portion of the UI. First Driverless AI figures out the backend and determines whether GPUs are running. Then it starts parameter tuning, followed by feature engineering. Finally, Driverless AI builds the scoring pipeline.

   In addition to the status, the UI also displays:

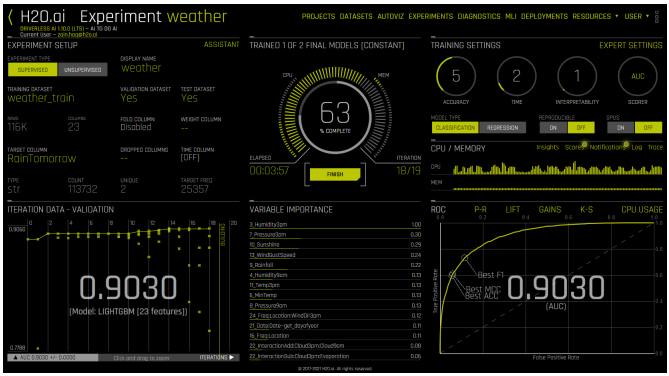
- Details about the dataset.
- The iteration data (internal validation) for each cross validation fold along with the specified scorer value. Click on a specific iteration or drag to view a range of iterations. Double click in the graph to reset the view.
- The variable importance values. To view variable importance for a specific iteration, just select that iteration in the Iteration Data graph. The Variable Importance list will automatically update to show variable importance information for that iteration. Hover over an entry to view more info. **Note**: When hovering over an entry, you may notice the term "Internal[...] specification." This label is used for features that do not need to be translated/explained and ensures that all features are uniquely identified.
- CPU/Memory information including Notifications, Logs, and Trace info. (Note that Trace is used for development/debugging and to show what the system is doing at that moment.)

For classification problems, the lower right section includes a toggle between an ROC curve, Precision-Recall graph, Lift chart, Gains chart, Kolmogorov-Smirnov chart, and GPU Usage information (if GPUs are available). For regression problems, the lower right section includes a toggle between a Residuals chart, an Actual vs. Predicted chart, and GPU Usage information (if GPUs are available). Refer to the Experiment Graphs section in the User Guide for more information about these graphs.

Upon completion, an Experiment Summary section will populate in the lower right section.

You can stop experiments that are currently running. Click the **Finish** button to stop the experiment. This jumps the experiment to the end and completes the ensembling and the deployment package. You can also click **Abort** to terminate the experiment. (You will be prompted to confirm the abort.) Aborted experiments will display on the Experiments page as Failed. You can restart aborted experiments by clicking the right side of the experiment, then selecting **Restart from Last Checkpoint**. This will start a new experiment based on the aborted one. Alternatively, you can started a new experiment based on the aborted one by selecting **New Experiment with Same Settings**.

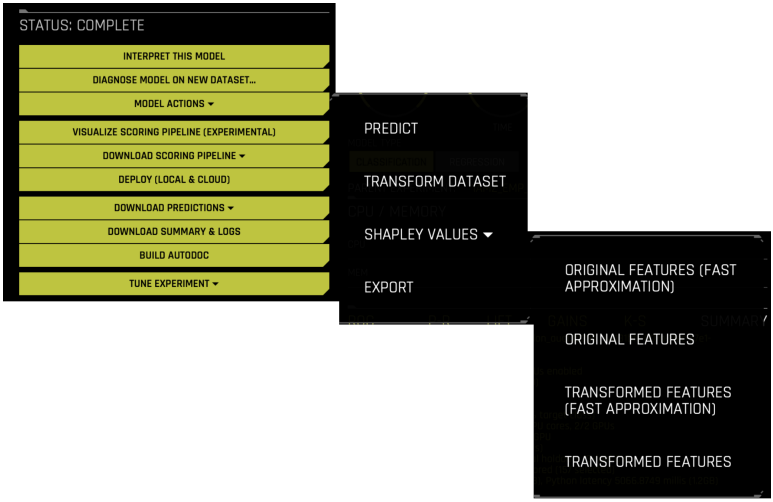
The final step that Driverless AI performs during an experiment is to complete the experiment report. During this step, you can click **Abort** to skip this report.



8.3 Completed Experiment

8.3.1 Completed Experiment Actions

After an experiment status changes from RUNNING to COMPLETE, the UI provides you with several options:

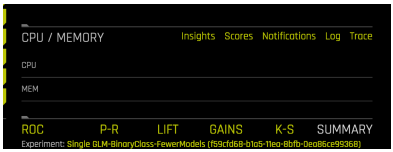


- Deploy (Local and Cloud). Refer to the Deployment section.
- Interpret this Model. Refer to the Interpreting a Model section. (Not supported for Image or multiclass Time Series experiments.)
- Diagnose Model on New Dataset. Refer to the Diagnosing a Model section.

- Score on Another Dataset. Refer to the Score on Another Dataset section.
- Transform Another Dataset. Refer to the Transform Another Dataset section. (Not available for time series experiments.)
- Download Predictions dropdown:
  - Training Holdout Predictions. In csv format, available if a validation set was NOT used.
  - Validation Set Predictions. In csv format, available if a validation set was used.
  - Test Set Predictions. In csv format, available if a test dataset is used.
- Download Python Scoring Pipeline. A standalone Python scoring pipeline for H2O Driverless AI.
- Download MOJO Scoring Pipeline. A standalone Model Object, Optimized scoring pipeline. (Not available for TensorFlow or RuleFit models.)
- Visualize Scoring Pipeline: Opens an experiment pipeline visualization page.
- Download Summary and Logs. A zip file containing the following:
  - Experiment logs (regular and anonymized)
  - A summary of the experiment
  - The experiment features along with their relative importance
  - Ensemble information
  - An experiment preview
  - Word version of an auto-generated report for the experiment
  - A target transformations tuning leaderboard
  - A tuning leaderboard
- Download AutoDoc: A Word version of an auto-generated report for the experiment. This file is also available in the Experiment Summary zip file. Note that this option is not available for deprecated models.

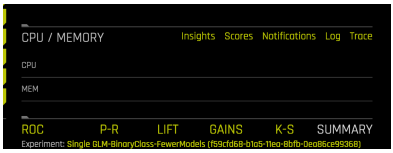
8.3.2 Experiment Insights and Scores

While an experiment is running, the UI provides you with options for viewing model insights (for time-series experiments) and model scores. Refer to the Model Insights and Model Scores sections for more information.



8.4 Model Insights

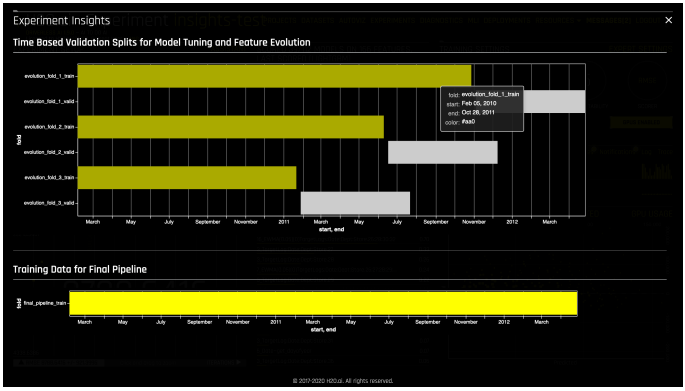
For time-series experiments, you can view detailed information about model scores while an experiment is running or after an experiment is complete by clicking on the **Insights** option.



The Insights page includes the following graphs:

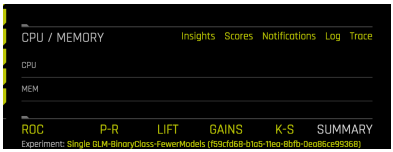
- Time Based Validation Splits for Model Tuning and Feature Evolution
- Training Data for Final Pipeline
- Time Based Back Testing of Final Pipeline on Training Data

You can hover over a point in the graphs to view the dates in the graphs.



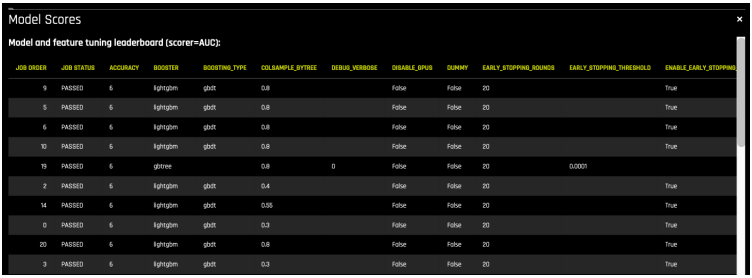
8.5 Model Scores

You can view detailed information about model scores after an experiment is complete by clicking on the **Scores** option.



The Model Scores page that opens includes the following tables:

- **Model and feature tuning leaderboard**: This leaderboard shows scoring information based on the scorer that was selected in the experiment. This information is also available in the **tuning_leaderboard.json** file of the experiment summary. You can download that file directly from the bottom of this table.

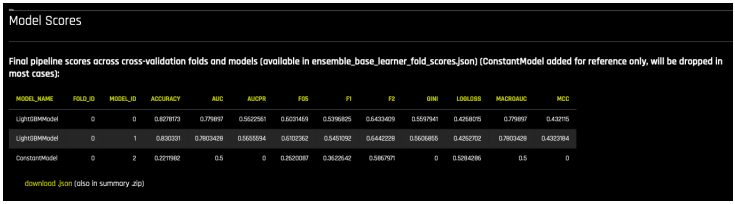


Model Scores

Model and feature tuning leaderboard (scorer=AUC):

JOB ORDER	JOB STATUS	ACCURACY	BOOSTER	BOOSTING_TYPE	COLSAMPLE_BYTREE	DEBUG_VERBOSE	DISABLE_GPUS	DUMMY	EARLY_STOPPING_ROUNDS	EARLY_STOPPING_THRESHOLD	ENABLE_EARLY_STOPPING
9	PASSED	6	lightgbm	gbdt	0.8		False	False	20		True
5	PASSED	6	lightgbm	gbdt	0.8		False	False	20		True
6	PASSED	6	lightgbm	gbdt	0.8		False	False	20		True
10	PASSED	6	lightgbm	gbdt	0.8		False	False	20		True
19	PASSED	6	gbtree		0.8	0	False	False	20	0.0001	
2	PASSED	6	lightgbm	gbdt	0.4		False	False	20		True
14	PASSED	6	lightgbm	gbdt	0.55		False	False	20		True
0	PASSED	6	lightgbm	gbdt	0.3		False	False	20		True
20	PASSED	6	lightgbm	gbdt	0.8		False	False	20		True
3	PASSED	6	lightgbm	gbdt	0.3		False	False	20		True

- **Final pipeline scores across cross-validation folds and models**: This table shows the final pipeline scores across cross-validation folds and models. Note that if Constant Model was enabled (default), then that model is added in this table as a baseline (reference) only and will be dropped in most cases. This information is also included in the **ensemble_base_learner_fold_scores.json** file of the experiment summary. You can download that file directly from a link at the bottom of this table.



Model Scores

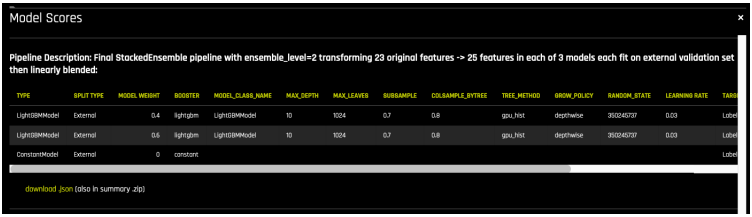
Final pipeline scores across cross-validation folds and models (available in ensemble_base_learner_fold_scores.json) (ConstantModel added for reference only, will be dropped in most cases):

MODEL_NAME	FOLD_ID	MODEL_ID	ACCURACY	AUC	AUCPR	F05	F1	F2	GINI	LOGLOSS	MACROAUC	MCC
LightGBMModel	0	0	0.8278173	0.779897	0.5592561	0.6031469	0.5396825	0.6433409	0.5597941	0.4258015	0.779897	0.432115
LightGBMModel	0	1	0.830331	0.7803428	0.5655594	0.6102362	0.5451092	0.6442228	0.5606855	0.4252702	0.7803428	0.4332984
ConstantModel	0	2	0.2271962	0.5	0	0.2620087	0.3629542	0.5967571	0	0.5284286	0.5	0

download .json (also in summary .zip)

- **Pipeline Description**: This shows how the final Stacked Ensemble pipeline was calculated. This information is also included in the **ensem-**

ble_model_description.json file of the experiment summary. You can download that file directly from a link at the bottom of this table.



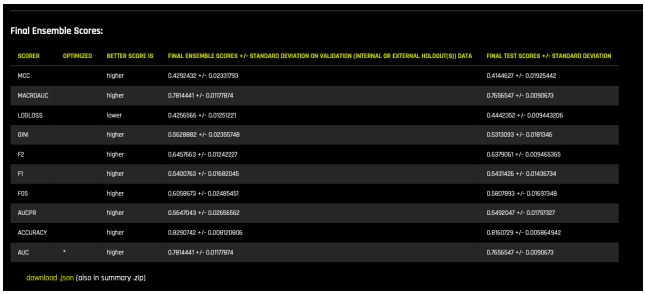
Model Scores

Pipeline Description: Final StackedEnsemble pipeline with ensemble_level=2 transforming 23 original features -> 25 features in each of 3 models each fit on external validation set then linearly blended:

TYPE	SPLIT TYPE	MODEL WEIGHT	BOOSTER	MODEL_CLASS_NAME	MAX_DEPTH	MAX_LEAVES	SUBSAMPLE	COLSAMPLE_BYTREE	TREE_METHOD	GROW_POLICY	RANDOM_STATE	LEARNING RATE	TARGET
LightGBMModel	External	0.4	lightgbm	LightGBMModel	10	1024	0.7	0.8	gpu_hist	depthwise	350245737	0.03	Label
LightGBMModel	External	0.6	lightgbm	LightGBMModel	10	1024	0.7	0.8	gpu_hist	depthwise	350245737	0.03	Label
ConstantModel	External	0	constant										Label

download .json (also in summary .zip)

- **Final Ensemble Scores**: This shows the final scores for each scorer in DAI. If a custom scorer was used in the experiment, that scorer will also appear here. This information is also included in the **ensemble_scores.json** file of the experiment summary. You can download that file directly from a link at the bottom of this table.



Final Ensemble Scores:

SCORER	OPTIMIZED	BETTER SCORE IS	FINAL ENSEMBLE SCORES +/- STANDARD DEVIATION ON VALIDATION (INTERNAL OR EXTERNAL HOLDOUT(S)) DATA	FINAL TEST SCORES +/- STANDARD DEVIATION
MCC		higher	0.4292432 +/- 0.02337793	0.4144627 +/- 0.01925442
MACROAUC		higher	0.7814441 +/- 0.01177874	0.7656547 +/- 0.0050673
LOGLOSS		lower	0.4256566 +/- 0.07251221	0.4442352 +/- 0.0039443206
GINI		higher	0.5628882 +/- 0.02355748	0.5313093 +/- 0.0181346
F2		higher	0.6407653 +/- 0.01242227	0.6375061 +/- 0.009466365
F1		higher	0.5400763 +/- 0.01682045	0.5431425 +/- 0.01436734
F05		higher	0.5056673 +/- 0.02485451	0.5107693 +/- 0.01597348
AUCPR		higher	0.5547043 +/- 0.02655562	0.5492047 +/- 0.01797327
ACCURACY		higher	0.8290742 +/- 0.008121806	0.8160709 +/- 0.005864542
AUC	*	higher	0.7814441 +/- 0.01177874	0.7656547 +/- 0.0050673

download .json (also in summary .zip)

8.5.1 Experiment Summary

An experiment summary is available for each completed experiment. Click the **Download Experiment Summary** button to download the **h2oai_experiment_summary_<experiment>.zip** file.

The files within the experiment summary zip provide textual explanations of the graphical representations that are shown on the Driverless AI UI. For example, the **preview.txt** file provides the same information that was included on the UI before starting the experiment; the **summary.txt** file provides the same summary that appears in the lower-right portion of the UI for the experiment; the **features.txt** file provides the relative importance values and descriptions for the top features.

Experiment Report

A **report** file is included in the experiment summary. This report provides insight into the training data and any detected shifts in distribution, the validation

schema selected, model parameter tuning, feature evolution and the final set of features chosen during the experiment.

- **report.docx**: The report available in Word format

Experiment Overview Artifacts

The Experiment Summary contains artifacts that provide overviews of the experiment.

- **preview.txt**: Provides a preview of the experiment. (This is the same information that was included on the UI before starting the experiment.)

- **summary.txt**: Provides the same summary that appears in the lower-right portion of the UI for the experiment.

Tuning Artifacts

During the Driverless AI experiment, model tuning is performed to determined the optimal algorithm and parameter settings for the provided dataset. For regression problems, target tuning is also performed to determine the best way to represent the target column (i.e. does taking the log of the target column improve results). The results from these tuning steps are available in the Experiment Summary.

- **tuning_leaderboard**: A table of the model tuning performed along with the score generated from the model and training time. (Available in txt or json.)

- **target_transform_tuning_leaderboard.txt**: A table of the transforms applied to the target column along with the score generated from the model and training time. (This will be empty for binary and multiclass use cases.)

Features Artifacts

Driverless AI performs feature engineering on the dataset to determine the optimal representation of the data. The top features used in the final model can be seen in the GUI. The complete list of features used in the final model is available in the Experiment Summary artifacts.

The Experiment Summary also provides a list of the original features and their estimated feature importance. For example, given the features in the final Driverless AI model, we can estimate the feature importance of the original features.

Feature	Feature Importance
NumToCatWoE:PAY_AMT2	1
PAY_3	0.92
ClusterDist9:BILL_AMT1:LIMIT_BAL:PAY_3	0.90

To calculate the feature importance of PAY_3, we can aggregate the feature importance for all variables that used PAY_3:

- NumToCatWoE:PAY_AMT2: 1 * 0 (PAY_3 not used.)
- PAY_3: 0.92 * 1 (PAY_3 is the only variable used.)
- ClusterDist9:BILL_AMT1:LIMIT_BAL:PAY_3: 0.90 * 1/3 (PAY_3 is one of three variables used.)

Estimated Feature Importance = (1*0) + (0.92*1) + (0.9*(1/3)) = 1.22

Note: The feature importance is converted to relative feature importance.

- **features**: A complete list of all features used in the final model, a description of the feature, and the feature importance. (Available in txt or json.)
- **features_orig**: A list of the original features provided and an estimate of the relative feature importance of that original feature in the final model. (Available in txt or json.)

Final Model Artifacts

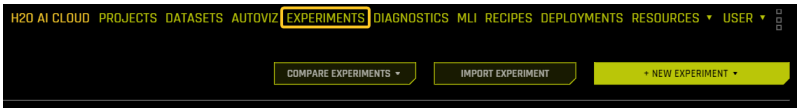
The Experiment Summary includes artifacts that describe the final model. This is the model that is used to score new datasets and create the MOJO scoring pipeline. The final model may be an ensemble of models depending on the Accuracy setting.

- **ensemble.txt**: A summary of the final model which includes a description of the model(s), gains/lifts table, confusion matrix, and scores of the final model for our list of scorers.
- **ensemble_description.txt**: A sentence describing the final model. (For example: Final TensorFlowModel pipeline with ensemble_level=0 transforming 21 original features -¿ 54 features in each of 1 models each fit on full training data (i.e. no hold-out).)
- **ensemble_model_description.json**: A json file describing the model(s) and for ensembles how the model predictions are weighted.

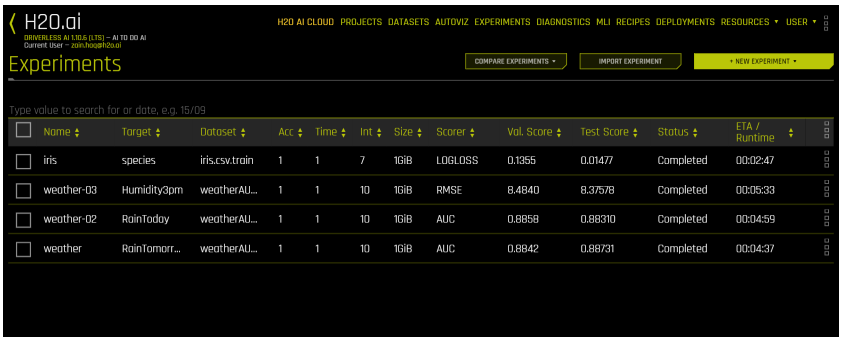
- **ensemble_model_params.json**: A json file decribing the parameters of the model(s).
- **ensemble_scores.json**: The scores of the final model for our list of scorers.
- **ensemble_confusion_matrix**: The confusion matrix for the internal validation and test data if test data is provided.
- **ensemble_confusion_matrix_stats_test.json**: Confusion matrix statistics on the test data. (Only available if test data provided)
- **ensemble_gains**: The lift and gains table for the internal validation and test data if test data is provided. (Visualization of lift and gains can be seen in the UI.)

8.6 Viewing Experiments

The upper-right corner of the Driverless AI UI includes an **Experiments** link.



Click this link to open the Experiments page. From this page, you can rename an experiment, view previous experiments, begin a new experiment, rerun an experiment, and delete an experiment.



8.6.1 Checkpointing, Rerunning, and Retraining

In Driverless AI, you can retry an experiment from the last checkpoint, you can run a new experiment using an existing experiment's settings, and you can retrain an experiment's final pipeline.

Checkpointing Experiments

In real-world scenarios, data can change. For example, you may have a model currently in production that was built using 1 million records. At a later date, you may receive several hundred thousand more records. Rather than building a new model from scratch, Driverless AI includes H2O.ai Brain, which enables caching and smart re-use of prior models to generate features for new models.

You can configure one of the following Brain levels in the experiment's Expert Settings.

- Level -1: Don't use any brain cache
- Level 0: Don't use any brain cache but still write to cache
- Level 1: Smart checkpoint if an old experiment_id is passed in (for example, via running "resume one like this" in the GUI)
- Level 2: Smart checkpoint if the experiment matches all column names, column types, classes, class labels, and time series options identically. (default)
- Level 3: Smart checkpoint like level 1, but for the entire population. Tune only if the brain population is of insufficient size.
- Level 4: Smart checkpoint like level 2, but for the entire population. Tune only if the brain population is of insufficient size.
- Level 5: Smart checkpoint like level 4, but will scan over the entire brain cache of populations (starting from resumed experiment if chosen) in order to get the best scored individuals.

If you chooses Level 2 (default), then Level 1 is also done when appropriate.

To make use of smart checkpointing, be sure that the new data has:

- The same data column names as the old experiment
- The same data types for each column as the old experiment. (This won't match if, e.g,. a column was all int and then had one string row.)
- The same target as the old experiment
- The same target classes (if classification) as the old experiment
- For time series, all choices for intervals and gaps must be the same

When the above conditions are met, then you can:

- Start the same kind of experiment, just rerun for longer.
- Use a smaller or larger data set (i.e. fewer or more rows).
- Effectively do a final ensemble re-fit by varying the data rows and starting an experiment with a new accuracy, time=1, and interpretability. Check the experiment preview for what the ensemble will be.
- Restart/Resume a cancelled, aborted, or completed experiment

To run smart checkpointing on an existing experiment, click the right side of the experiment that you want to retry, then select **Restart from Last Checkpoint**. The experiment settings page opens. Specify the new dataset. If desired, you can also change experiment settings, though the target column must be the same. Click **Launch Experiment** to resume the experiment from the last checkpoint and build a new experiment.

The smart checkpointing continues by adding a prior model as another model used during tuning. If that prior model is better (which is likely if it was run for more iterations), then that smart checkpoint model will be used during feature evolution iterations and final ensemble.

Notes:

- Driverless AI does not guarantee exact continuation, only smart continuation from any last point.
- The directory where the H2O.ai Brain meta model files are stored is **tmp/H2O.ai_brain**. In addition, the default maximum brain size is 20GB. Both the directory and the maximum size can be changed in the config.toml file.

Rerunning Experiments

To run a new experiment using an existing experiment's settings, click the right side of the experiment that you want to use as the basis for the new experiment, then select **New Experiment with Same Settings**. This opens the experiment

settings page. From this page, you can rerun the experiment using the original settings, or you can specify to use new data and/or specify different experiment settings. Click **Launch Experiment** to create a new experiment with the same options.

Retrain Final Pipeline

To retrain an experiment's final pipeline, click the right side of the experiment that you want to use as the basis for the new experiment, then select **Retrain Final Pipeline**. This opens the experiment settings page with the same settings as the original experiment except that Time is set to 0. This retrain mode is equivalent to setting feature brain level 3 with time 0 (no tuning or feature evolution iterations).

8.6.2 Deleting Experiments

To delete an experiment, hover over the experiment that you want to delete. An "X" option displays. Click this to delete the experiment. A confirmation message will display asking you to confirm the delete. Click **OK** to delete the experiment or **Cancel** to return to the experiments page without deleting.

9 Diagnosing a Model

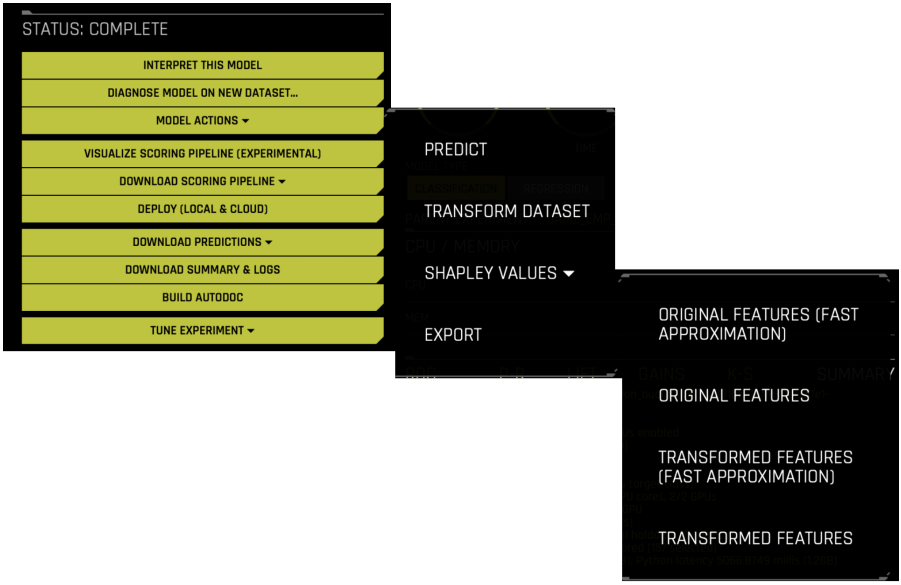
The **Diagnosing Model on New Dataset** option allows you to view model performance for multiple scorers based on existing model and dataset.

On the completed experiment page, click the **Diagnose Model on New Dataset** button.

Notes:

- You can also diagnose a model by selecting **Diagnostic** from the top menu, then selecting an experiment and test dataset.

- The Model Diagnostics page also automatically populates with any experiments that were scored from the Project Leaderboard on the Projects page.

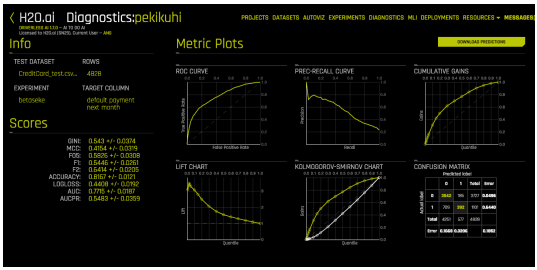


Select a dataset to use when diagnosing this experiment. At this point, Driverless AI will begin calculating all available scores for the experiment.

When the diagnosis is complete, it will be available on the **Model Diagnostics** page. Click on the new diagnosis. From this page, you can download predictions. You can also view scores and metric plots. The plots are interactive. Click a graph to enlarge. In the enlarged view, you can hover over the graph to view details for a specific point. You can also download the graph.

Classification metric plots include the following graphs:

- ROC Curve
- Precision-Recall Curve
- Cumulative Gains
- Lift Chart
- Kolmogorov-Smirnov Chart
- Confusion Matrix



Regression metric plots include the following graphs:

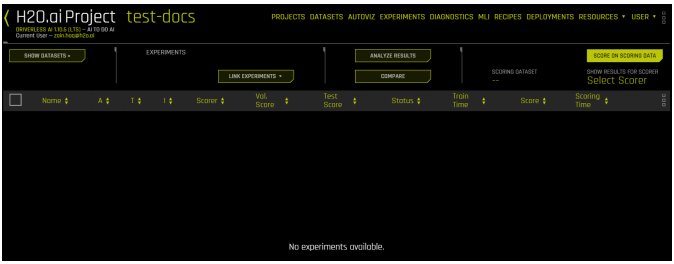
- Actual vs Predicted
- Residual Plot with LOESS curve
- Residual Histogram

10 Project Workspace

Driverless AI provides a Project Workspace for managing datasets and experiments related to a specific business problem or use case. Whether you are trying to detect fraud or predict user retention, datasets and experiments can be stored and saved in the individual projects. A Leaderboard on the Projects page allows you to easily compare performance and results and identify the best solution for your problem.

To create a Project Workspace:

1. Click the **Projects** option on the top menu.
2. Click **New Project**.
3. Specify a name for the project and provide a description.
4. Click **Create Project**. This creates an empty Project page.

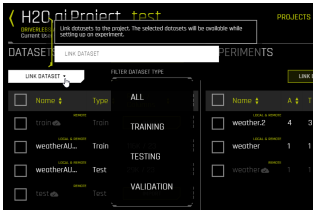


From the Projects page, you can link datasets and/or experiments, and you can run new experiments. When you link an existing experiment to a Project, the datasets used for the experiment will automatically be linked to this project (if not already linked).

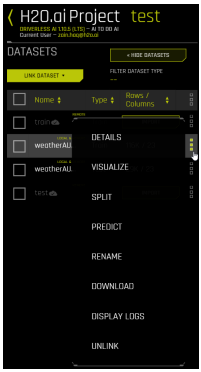
10.1 Linking Datasets

Any dataset that has been added to Driverless AI can be linked to a project. In addition, datasets used for experiments are also automatically linked when an experiment is linked to the project.

You can link a Training, Validation, or Test dataset by selecting the **Training**, **Validation**, or **Test** tab, clicking **Link Dataset**, and then selecting the dataset(s) to include. The list available datasets include those that were added on :ref:'Datasets', or you can browse datasets in your file system. Be sure to select the correct tab before linking a training, validation, or test dataset. This is because, when you run a new experiment in the project, the training data, validation data, and test data options for that experiment come from list of datasets linked here. You will not be able to, for example, select any datasets from within the Training tab when specifying a test dataset on the experiment.

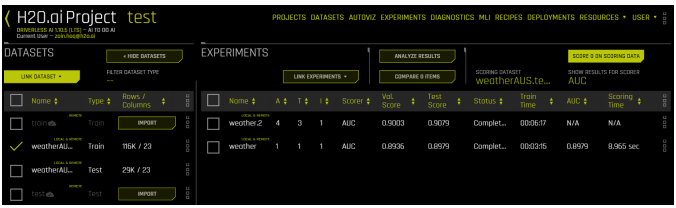


When datasets are linked, the same menu options are available here as on the Datasets page.



10.1.1 Selecting Datasets

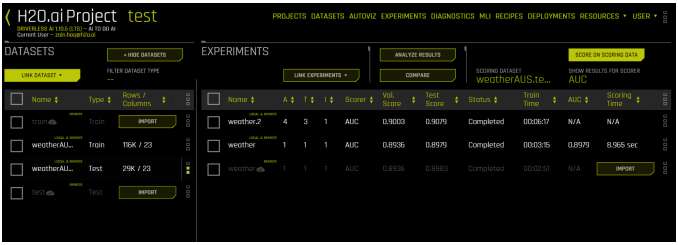
In the Datasets section, you can select a training, validation, or testing dataset. The Experiments section will show experiments in the Project that use the selected dataset.



10.2 Linking Experiments

Existing experiments can be selected and linked to a Project. Additionally, you can run a new experiment or checkpointing an existing experiment from this page, and those experiments will automatically be linked to this Project.

Link an existing experiment to the project by clicking **Link Experiment** and then selecting the experiment(s) to include. When you link experiments, the datasets used to create the experiments are also automatically linked.



10.2.1 New Experiments

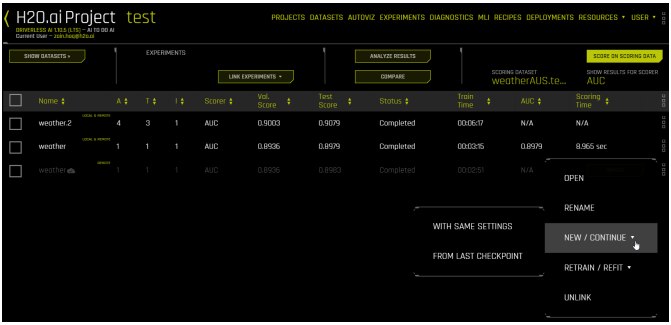
When experiments are run from within a Project, only linked datasets can be used.

1. Click the **New Experiment** link to begin a new experiment.
2. Select your training data and optionally your validation and/or testing data.
3. Specify your desired experiment settings, and then click **Launch Experiment**.

As the experiment is running, it will be listed at the top of the Experiments Leaderboard until it is completed. It will also be available on the **Experiments** page.

10.2.2 Checkpointing Experiments

When experiments are linked to a Project, the same checkpointing options for experiments are available here as on the Experiments page.

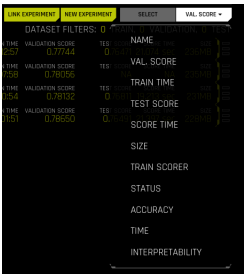


10.3 The Experiments Leaderboard

When attempting to solve a business problem, a normal workflow will include running multiple experiments, either with different/new data or with a variety of settings, and the optimal solution can vary for different users and/or business problems. For some users, the model with the highest accuracy for validation and test data could be the most optimum one. Other users might be willing to make an acceptable compromise on the accuracy of the model for a model with greater performance (faster prediction). For some, it could also mean how quickly the model could be trained with acceptable levels of accuracy. The Experiments Leaderboard makes it easy for you to find the best solution for your business problem.

The Leaderboard is organized by showing running experiments first, then completed experiments (sorted by validation score by default), then canceled experiments. You can change the sorting of completed experiments by selecting the sort dropdown menu.

Hover over experiments in the Leaderboard to view additional information about the experiment, including the problem type, datasets used, and the target column.



10.3.1 Leaderboard Scoring

The Leaderboard allows you to view scoring information for a variety of scorers. You can change the scorer used by clicking the **Scorer** link and then selecting a scorer.

Experiments linked to projects do not automatically include a test score. To view Test Scores in the Leaderboard, you must first complete the scoring step for a particular dataset and experiment combination. Without the scoring step, no scoring data is available to populate in the Test Score and Score Time columns of the Leaderboard. Experiments that do not include a test score or that have an invalid scorer (for example, if the R2 scorer is selected for classification experiments) show **N/A** in the Leaderboard. Also, if **None** is selected for the scorer, then all experiments will show **N/A**.

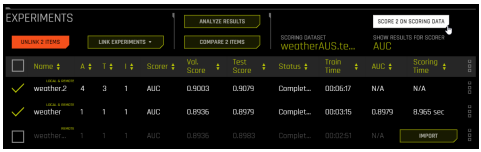
To score the experiment:

1. Click **Select Scoring Dataset** at the top of the Experiments list and select a linked Test Dataset or a test dataset available on the file system.

2. Select the model or models that you want to score.

3. Click the **Select Scorer** button at the top of the Experiments list and select a scorer.

4. Click the **Score** n **Items** button.



Notes:

- If an experiment has already scored a dataset, it will not score it again. The scoring step is deterministic, so for a particular scorer dataset and experiment combination, the score will be same regardless of how many times you repeat it.

- The scorer dataset absolutely needs to have all the columns that are expected by the various experiments you are scoring it on. However, the columns of the scorer dataset need not be exactly the same as input features expected by the experiment. There can be additional columns in the scorer dataset. If these columns were not used for training, they will be ignored. This feature gives you the ability to train experiments on different training datasets (i.e., having different features), and if you have an "uber test dataset" that includes all these feature columns, then you can use the same dataset to score these experiments.

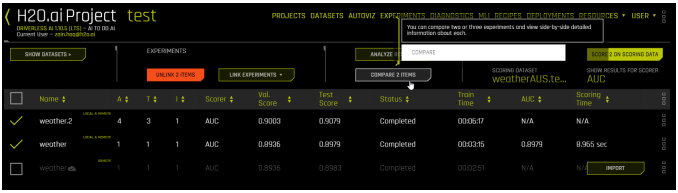
- You will notice a Score Time in the Experiments Leaderboard. This values shows the total time (in seconds) that it took for calculating the experiment scores for all applicable scorers for the experiment type. This is valuable to users who need to estimate the runtime performance of an experiment.

10.3.2 Comparing Experiments

You can compare two or three experiments and view side-by-side detailed information about each.

1. Click the **Select** button at the top of the Leaderboard and select either two or three experiments that you want to compare. You cannot compare more than three experiments.

2. Click the **Compare n Items** button.



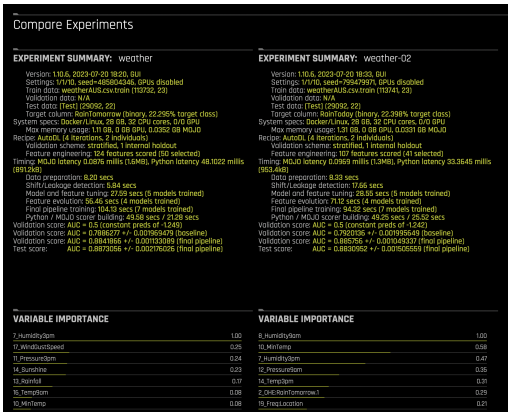
This opens the **Compare Experiments** page. This page includes the experiment summary for each experiment as well as metric plots. The metric plots vary depending on whether this is a classification or regression experiment.

For classification experiments, this page includes:

- Variable Importance list
- Confusion Matrix
- ROC Curve
- Precision Recall Curve
- Lift Chart
- Gains Chart
- Kolmogorov-Smirnov Chart

For regression experiments, this page includes:

- Variable Importance list
- Actual vs. Predicted Graph



10.4 Unlinking Data on a Projects Page

Unlinking datasets and/or experiments does not delete that data from Driverless AI. The datasets and experiments will still be available on the **Datasets** and **Experiments** pages.

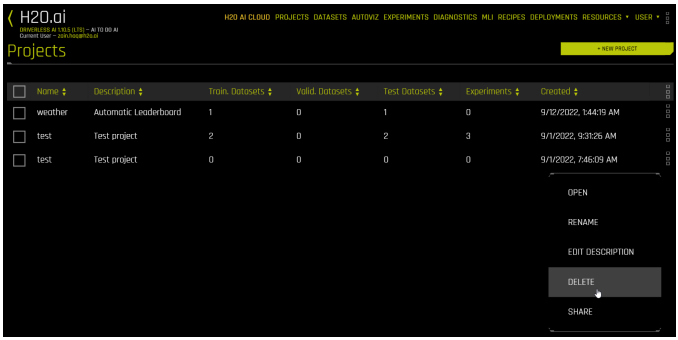
- Unlink a dataset by clicking on the dataset and selecting **Unlink** from the menu. textbfNote: You cannot unlink datasets that are tied to experiments in the same project.

- Unlink an experiment by clicking on the experiment and selecting **Unlink** from the menu. Note that this will not automatically unlink datasets that were tied to the experiment.

10.5 Deleting Projects

To delete a project, click the **Projects** option on the top menu to open the main Projects page. Hover over the project that you want to delete and click the red **X** button.

Note that deleting projects does not delete datasets and experiments from Driverless AI. Any datasets and experiments from deleted projects will still be available on the **Datasets** and **Experiments** pages.



11 Leaderboards

Although Driverless AI makes it easy for non-experts to experiment with machine learning, there is still a fair bit of knowledge and background in data science that is required to produce high-performing machine learning models. To make machine learning tasks even more accessible to non-experts, Driverless AI provides an Automatic Machine Learning (AutoML) feature that automatically trains and tunes models using different configuration settings and algorithms. AutoML also creates a "leaderboard" of models that were trained in the process and places those in a Project.

The models are scored using the scorer specified when creating the leaderboard. For binary classification problems, that scorer defaults to AUC; for multiclass classification problems, the scorer defaults to Logloss; and for regression problems, the scorer defaults to RMSE.

Leaderboards are supported for all Driverless AI experiments. When creating a leaderboard, the models that are built will vary based on whether you are running a regular experiment or a time-series experiment. More information is available in the Leaderboards section in the User Guide.

11.1 Creating a Leaderboard

Creating a Leaderboard is similar to running a new experiment:

1. On the **Datasets** page, select the dataset that you want to use for the experiment, then click **Predict**; or on the **Experiments** page, click **New Experiment**, then select the dataset that you want to use.

2. Specify whether to include dropped columns, a validation dataset, and a testing dataset.

3. Specify the Target column and optionally a fold column, weight column, and time column.

4. Optionally specify Expert Settings.

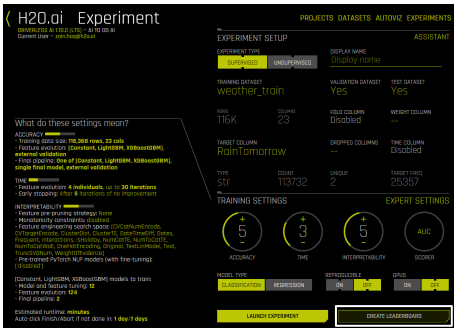
5. Optionally adjust the Accuracy/Time/Interpretability knobs.

6. Optionally override the default scorer.

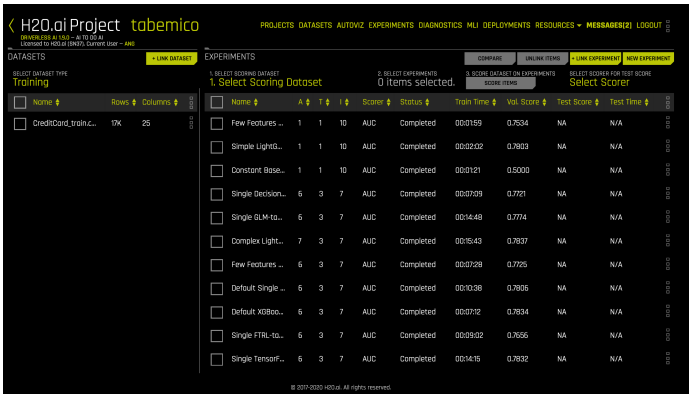
7. Optionally override the Classification/Regression setting.

8. Optionally specify to make the experiments reproducible and/or whether to enable GPUs.

9. Click the **Create Leaderboard** button.



Driverless AI will create a new, randomly named project and begin automatically training models. After all models have been built, you can score each experiment and compare experiments as described in the **Projects** chapter.



12 Interpreting a Model

Driverless AI provides robust interpretability of machine learning models to explain modeling results in a human-readable format. In the Machine Learning Interpetability (MLI) view, Driverless AI employs a host of different techniques and methodologies for interpreting and explaining the results of its models. A number of charts are generated automatically, including K-LIME, Shapley, Variable Importance, Decision Tree Surrogate, Partial Dependence, Individual Conditional Expectation, and more. Additionally, you can download a CSV of LIME, Shapley, and Original (Kernel SHAP) Shapley reason codes as well as text and Python files of Decision Tree Surrogate model rules from this view.

This chapter describes Machine Learning Interpretability (MLI) in Driverless AI for both regular and time-series experiments. There are two methods you can use for interpreting models:

- Using the **Interpret this Model** button on a completed experiment page to interpret a Driverless AI model on original and transformed features.
- Using the **MLI** link in the upper right corner of the UI to interpret either a Driverless AI model or an external model.

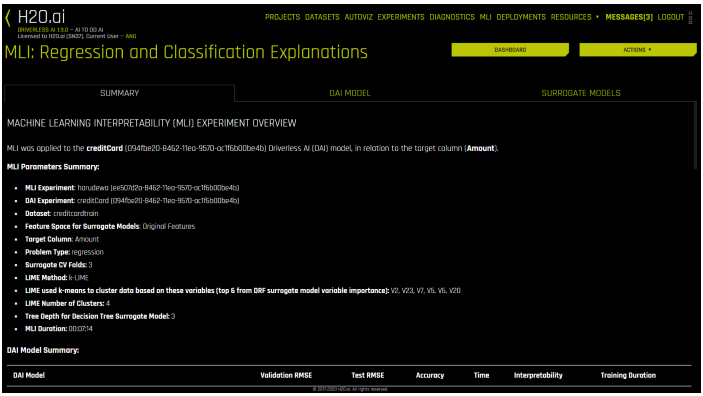
Notes:

- This release deprecates experiments and MLI models from 1.7.0 and earlier.
- MLI does not require an Internet connection to run on current models.
- MLI is not available for Image or multiclass Time Series experiments.
- For time series experiments, when the test set contains actuals, you will see the time series metric plot and the group metrics table. If there are no actuals, MLI will run, but you will see only the prediction value time series and a Shapley table.

12.1 Interpret this Model button - Regular Experiments

Clicking the **Interpret this Model** button on a completed experiment page launches the Model Interpretation for that experiment. Python and Java logs can be viewed while the interpretation is running.

For regular experiments, this page provides several visual explanations of the trained Driverless AI model and its results. More information about this page is available in the Understanding the Model Interpretation Page section later in this chapter.



12.2 Interpret this Model button - Time-Series Experiments

Driverless AI allows you to run MLI for multi-group and single-group time series experiments.

12.2.1 Multi-Group Time Series MLI

This section describes how to run MLI on time series data for multiple groups.

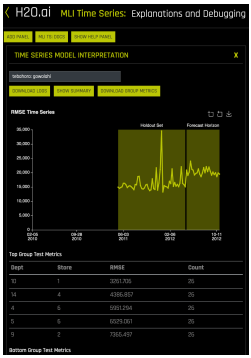
1. Click the **Interpret this Model** button on a completed time series experiment to launch Model Interpretation for that experiment. This page includes the following:

    - A Help panel describing how to read and use this page. Click the **Hide Help Button** to hide this text.

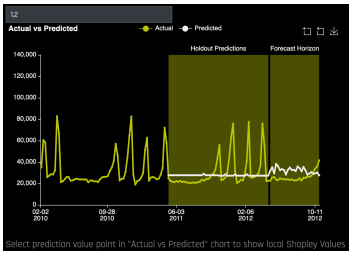
    - If a test set is provided and the test set includes actuals, then a panel will display showing a time series plot and the top and bottom group matrix tables based on the scorer that was used in the experiment. The metric plot will show the metric of interest per time point for holdout predictions and the test set. Likewise, the actual vs. predicted plot will show actuals vs. predicted values per time point for the holdout set and the test set. Note that this panel can be resized if necessary.

    - If a test set is not provided, then internal validation predictions will be used. The metric plot will only show the metric of interest per time point for holdout predictions. Likewise, the actual vs. predicted plot will only show actuals vs. predicted values per time point for the holdout set.

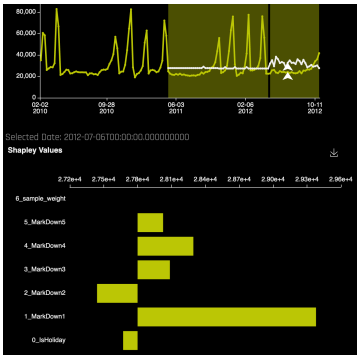
- A **Download Logs** button for retrieving logs that were generated when this interpretation was built.
- A **Download Group Metrics** button for retrieving the averages of each group's scorer, as well as each group's sample size.
- A **Show Summary** button that provide details about the experiment settings that were used.
- A **Group Search** entry field for selecting the groups to view.
- Use the zoom feature to magnify any portion of a graph by clicking the Enable Zoom icon near the top-right corner of a graph. While this icon is selected, click and drag to draw a box over the portion of the graph you want to magnify. Click the Disable Zoom icon to return to the default view.



2. Scroll to the bottom of the panel and select a grouping in the **Group Search** field to view a graph of Actual vs. Predicted values for the group. The outputted graph can be downloaded to your local machine.



3. Click on a prediction point in the plot (white line) to view Shapley values for that prediction point. The Shapley values plot can also be downloaded to your local machine.



4. Click **Add Panel** to add a new MLI Time Series panel. This allows you to compare different groups in the same model and also provides the flexibility to do a "side-by-side" comparison between different models.

12.2.2 Single Time Series MLI

Time Series MLI can also be run when only one group is available.

1. Click the **Interpret this Model** button on a completed time series experiment to launch Model Interpretation for that experiment. This page includes the following:

    - A Help panel describing how to read and use this page. Click the **Hide Help Button** to hide this text.

    - If a test set is provided and the test set includes actuals, then a panel will display showing a time series plot and the top and bottom group matrix tables based on the scorer that was used in the experiment. The metric plot will show the metric of interest per time point for holdout predictions and the test set. Likewise, the actual vs. predicted plot will show actuals vs. predicted values per time point for the holdout set and the test set. Note that this panel can be resized if necessary.

    - If a test set is not provided, then internal validation predictions will be used. The metric plot will only show the metric of interest per time point for holdout predictions. Likewise, the actual vs. predicted plot will only show actuals vs. predicted values per time point for the holdout set.

    - A **Download Logs** button for retrieving logs that were generated when this interpretation was built.

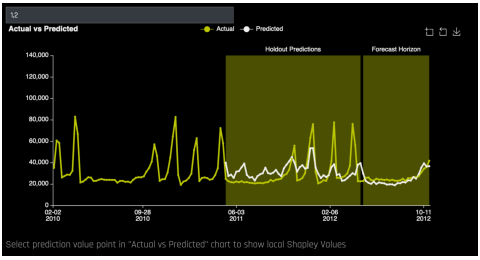
    - A **Download Group Metrics** button for retrieving the averages of each group's scorer, as well as each group's sample size.

- A **Show Summary** button that provide details about the experiment settings that were used.

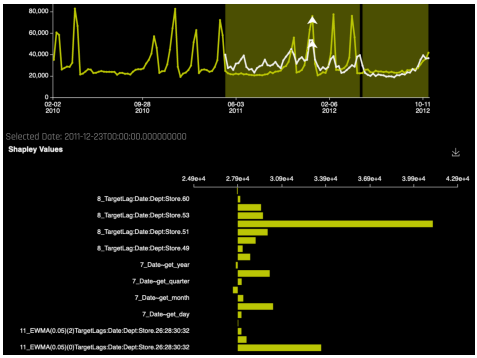
- A **Group Search** entry field for selecting the groups to view.

- Use the zoom feature to magnify any portion of a graph by clicking the Enable Zoom icon near the top-right corner of a graph. While this icon is selected, click and drag to draw a box over the portion of the graph you want to magnify. Click the Disable Zoom icon to return to the default view.

2. Scroll to the bottom of the panel and select an option in the **Group Search** field to view a graph of Actual vs. Predicted values for the group. (Note that for Single Time Series MLI, there will only be one option in this field.) The outputted graph can be downloaded to your local machine.



3. Click on a prediction point in the plot (white line) to view Shapley values for that prediction point. The Shapley values plot can also be downloaded to your local machine.

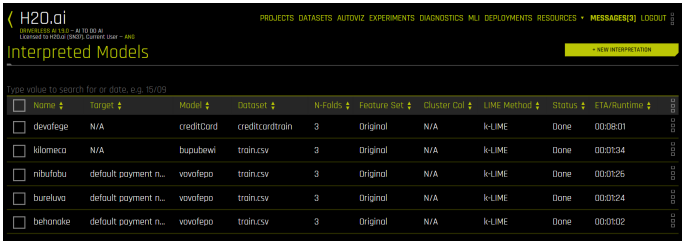


4. Click **Add Panel** to add a new MLI Time Series panel. This allows you to do a "side-by-side" comparison between different models.

12.3 Model Interpretation - Driverless AI Models

This method allows you to run model interpretation on a Driverless AI model. This method is similar to clicking one of the **Interpret This Model** buttons on an experiment summary page.

1. Click the **MLI** link in the upper-right corner of the UI to view a list of interpreted models.



2. Click the **New Interpretation** button.

3. Select the dataset that was used to train the model that you will use for interpretation.

4. Specify the Driverless AI model that you want to use for the interpretation. Once selected, the target column used for the model will be selected.

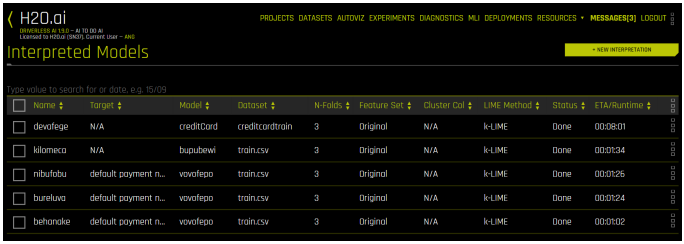
5. Optionally specify a weighted column and dropped columns.

6. Click the **Launch MLI** button.

12.4 Model Interpretation - External Models

Model Interpretation does not need to be run on a Driverless AI experiment. You can train an external model and run Model Interpretability on the predictions.

1. Click the **MLI** link in the upper-right corner of the UI to view a list of interpreted models.



2. Click the **New Interpretation** button.

3. Select the dataset that you want to use for the model interpretation. This must include a prediction column that was generated by the external model. If the dataset does not have predictions, then you can join the external predictions. An example showing how to do this using Python is available in the Driverless AI User Guide.

   Note: When running interpretations on an external model, leave the **Select Model** option empty. That option is for selecting a Driverless AI model.

4. Specify a Target Column (actuals) and the Prediction Column (scores from the model).

5. Optionally specify a weighted column and dropped columns.

6. Click the **Launch MLI** button.

12.5 Interpretation Expert Settings

Optionally specify additional expert settings for an interpretation. Refer to the Interpretation Expert Settings section in the Driverless AI User Guide for detailed information about these settings. Note that the default values for these options are derived from the environment variables in the config.toml file.

Notes:

- The selection of available expert settings is determined by the type of model you want to interpret and the specified LIME method.

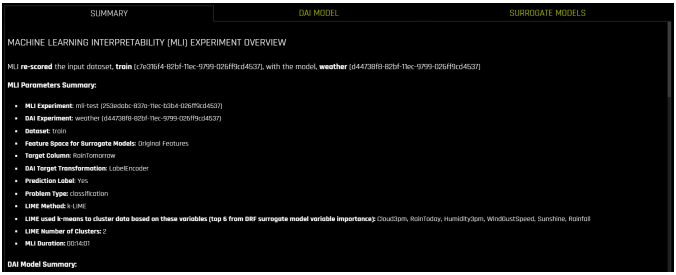
- Expert settings are not available for time-series models.

12.6 Understanding the Model Interpretation Page

The Model Interpretation page is organized into three tabs.

12.6.1 Summary Tab

The Summary tab provides an overview of the interpretation, including the dataset and Driverless AI experiment (if available) that were used for the interpretation along with the feature space (original or transformed), target column, problem type, and k-Lime information. If the interpretation was created from a Driverless AI model, then a table with the Driverless AI model summary is also included along with the top variables for the model.



12.6.2 DAI Model Tab

The DAI Model tab is organized into tiles for each interpretation method. To view a specific plot, click the tile for the plot that you want to view.

For binary classification and regression experiments, this tab includes Feature Importance and Shapley (not supported for RuleFit and TensorFlow models) plots for original and transformed features as well as Partial Dependence/ICE, Disparate Impact Analysis (DIA), Sensitivity Analysis, NLP Tokens and NLP LOCO (for text experiments), and Permutation Feature Importance (if the `autodoc_include_permutation_feature_importance` configuration option is enabled) plots. For multiclass classification experiments, this tab includes Feature Importance and Shapley plots for original and transformed features. See the DAI Model Tab Plots section for more information on these plots.

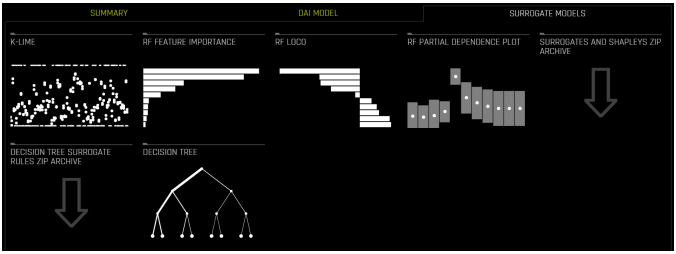
Note: Shapley plots are not supported for RuleFit and TensorFlow models. Shapley plots are also not supported for BYOR models that DO NOT implement the `has_pred_contribs` method and DO implement `pred_contribs=True` in `predict`.

12.6.3 Surrogate Model Tab

A surrogate model is a data mining and engineering technique in which a generally simpler model is used to explain another, usually more complex, model or phenomenon. For example, the decision tree surrogate model is trained to predict the predictions of the more complex Driverless AI model using the original model inputs. The trained surrogate model enables a heuristic understanding (i.e., not a mathematically precise understanding) of the mechanisms of the highly complex and nonlinear Driverless AI model.

The Surrogate Model tab is organized into tiles for each interpretation method. To view a specific plot, click the tile for the plot that you want to view. For binary classification and regression experiments, this tab includes K-LIME/LIME-SUP and Decision Tree plots as well as Feature Importance, Partial Dependence, and LOCO plots for the Random Forest surrogate model. See the Surrogate Model Tab Plots section for more information on these plots.

Note: For multiclass classification experiments, only the Decision Tree and Random Forest Feature Importance plots are available in this tab.



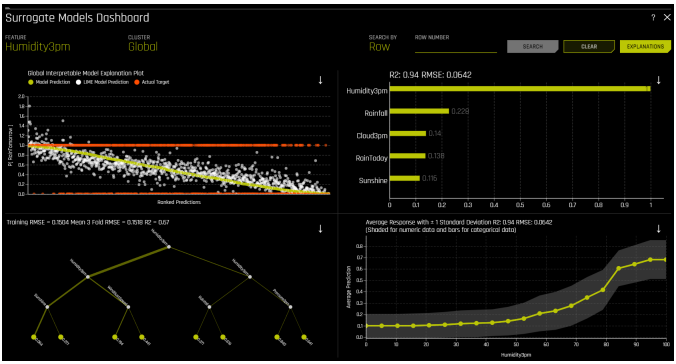
12.6.4 Dashboard and Actions Buttons

The Model Interpretation page also features the **Dashboard** and **Actions** buttons, which are located in the upper-right corner.

Dashboard Button

Click the **Dashboard** button to view the Dashboard page. For binary classification and regression experiments, the Dashboard page provides a single page with a Global Interpretable Model Explanations plot, a Feature Importance plot, a Decision Tree plot, and a Partial Dependence plot. You can also view explanations from this page by clicking the **Explanations** button located in the upper-right corner. Refer to the **Viewing Explanations** section for more information.

Note: The Dashboard is not available for multiclass classification experiments.



Actions Button

Click the **Actions** button to view the following options:

- **MLI Docs**: View the Machine Learning Interpetability section of the Driverless AI documentation.
- **Download MLI Logs**: Download a ZIP file of the logs that were generated during the interpretation.
- **Experiment**: View the experiment that was used to generate the interpretation.
- **Scoring Pipeline**: For binomial and regression experiments, download the scoring pipeline for the interpretation. This option is not available for multinomial experiments.
- **Download Reason Codes LIME**: For binomial experiments, download a CSV file of LIME reason codes.
- **Download Reason Codes Shapley**: For regression, binary, and multinomial experiments, download a CSV file of Shapley reason codes.
- **Display MLI Java Logs**: View MLI Java logs for the interpretation.

- **Display MLI Python Logs**: View MLI Python logs for the interpretation.
- **Download Decision Tree Surrogate Rules**: Download text and Python files of decision tree surrogate model rules for the interpretation.
- **Download Reason Codes Original Shapley (Kernel Shapley)**: For regression, binary, and multinomial experiments, download a CSV file of Original Shapley reason codes.

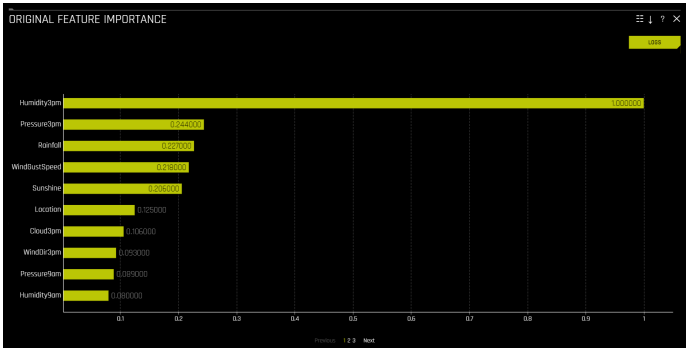
12.6.5 DAI Model Tab Plots

This section describes the plots that are available in the DAI Model Tab.

Feature Importance (Original and Transformed Features)

This plot is available for all models for binary classification, multiclass classification, and regression experiments.

This plot shows the Driverless AI feature importance. Driverless AI feature importance is a measure of the contribution of an input variable to the overall predictions of the Driverless AI model.



Shapley Plot (Original and Transformed Features)

This plot is not available for RuleFit or TensorFlow models. For all other models, this plot is available for binary classification, multiclass classification, and regression experiments.

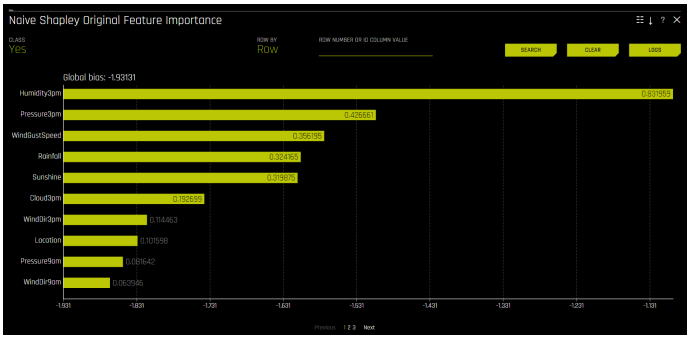
Shapley explanations are a technique with credible theoretical support that presents consistent global and local variable contributions. Local numeric Shapley values are calculated by tracing single rows of data through a trained tree ensemble and aggregating the contribution of each input variable as the row of data moves through the trained ensemble. For regression tasks, Shapley values sum to the prediction of the Driverless AI model. For classification

problems, Shapley values sum to the prediction of the Driverless AI model before applying the link function. Global Shapley values are the average of the absolute Shapley values over every row of a data set.

Notes:

- Shapley values for transformed features are calculated by tracing individual rows of data through a trained tree ensemble and then aggregating the contribution of each input variable as the row of data moves through the trained ensemble. More information about Shapley for tree-based models is available at `https://arxiv.org/abs/1706.06060`.

- Shapley values for original features are calculated with the Kernel Explainer method, which uses a special weighted linear regression to compute the importance of each feature. More information about Kernel SHAP is available at `http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf`. If Kernel Explainer is disabled, Shapley values for original features are approximated from the accompanying Shapley values for transformed features with the Naive Shapley method. For example, if the transformed feature `feature1_feature2` has a Shapley value of 0.5, then the Shapley value of the original features `feature1` and `feature2` will be 0.25.



Partial Dependence and Individual Conditional Expectation (ICE)

Partial dependence is a measure of the average model prediction with respect to an input variable. Partial dependence plots display how machine-learned response functions change based on the values of an input variable of interest, while taking nonlinearity into consideration and averaging out the effects of all other input variables. Partial dependence plots are well-known and described in the Elements of Statistical Learning (Hastie et all, 2001 [4]). Partial dependence plots enable increased transparency in Driverless AI models and the ability to

validate and debug Driverless AI models by comparing a variable's average predictions across its domain to known standards, domain knowledge, and reasonable expectations. The partial dependence plot is available for binary classification and regression models.

Individual conditional expectation (ICE) plots, a newer and less well-known adaptation of partial dependence plots, can be used to create more localized explanations for a single individual using the same basic ideas as partial dependence plots. ICE Plots were described by Goldstein et al (2015 [5]). ICE values are simply disaggregated partial dependence, but ICE is also a type of nonlinear sensitivity analysis in which the model predictions for a single row are measured while a variable of interest is varied over its domain. ICE plots enable a user to determine whether the model's treatment of an individual row of data is outside one standard deviation from the average model behavior, whether the treatment of a specific row is valid in comparison to average model behavior, known standards, domain knowledge, and reasonable expectations, and how a model will behave in hypothetical situations where one variable in a selected row is varied across its domain. The ICE plot is available for binary classification and regression models.

Given the row of input data with its corresponding Driverless AI and *K*-LIME predictions:

debt_to_income_ratio	credit_score	savings_acct_balance	observed_default	H2OAI_predicted_default	k-LIME_predicted_default
30	600	1000	1	0.85	0.9

Taking the Driverless AI model as F(**X**), assuming credit scores vary from 500 to 800 in the training data, and that increments of 30 are used to plot the ICE curve, ICE is calculated as follows:

$$\text{ICE}_{credit_score,500} = F(30, 500, 1000)$$

$$\text{ICE}_{credit_score,530} = F(30, 530, 1000)$$

$$\text{ICE}_{credit_score,560} = F(30, 560, 1000)$$

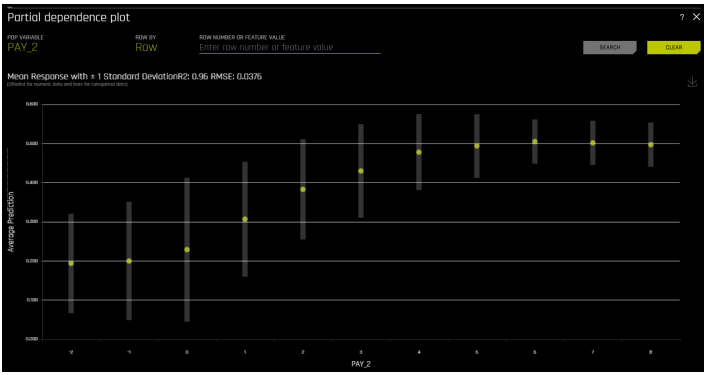
...

$$\text{ICE}_{credit_score,800} = F(30, 800, 1000)$$

The one-dimensional partial dependence plots displayed here do not take interactions into account. Large differences in partial dependence and ICE are an indication that strong variable interactions may be present. In this case partial dependence plots may be misleading because average model behavior may not accurately reflect local behavior.

Overlaying ICE plots onto partial dependence plots allow the comparison of the Driverless AI model's treatment of certain examples or individuals to the model's average predictions over the domain of an input variable of interest.

This plot shows the partial dependence when a variable is selected and the ICE values when a specific row is selected. Users may select a point on the graph to see the specific value at that point. Partial dependence (yellow) portrays the average prediction behavior of the Driverless AI model across the domain of an input variable along with +/- 1 standard deviation bands. ICE (grey) displays the prediction behavior for an individual row of data when an input variable is toggled across its domain. Currently, partial dependence and ICE is only available for the top ten most important original input variables. Categorical variables with 20 or more unique values are never included in these plots.



Disparate Impact Analysis

This plot is available for binary classification and regression models.

DIA is a technique that is used to evaluate fairness. Bias can be introduced to models during the process of collecting, processing, and labeling data—as a result, it is important to determine whether a model is harming certain users by making a significant number of biased decisions.

DIA typically works by comparing aggregate measurements of unprivileged groups to a privileged group. For instance, the proportion of the unprivileged group that receives the potentially harmful outcome is divided by the proportion of the privileged group that receives the same outcome—the resulting proportion is then used to determine whether the model is biased. Refer to the **Summary** section to determine if a categorical level (for example, Fairness Female) is fair in comparison to the specified reference level and user-defined thresholds. Fairness All is a true or false value that is only true if every category is fair in comparison to the reference level.

Disparate impact testing is best suited for use with constrained models in Driverless AI, such as linear models, monotonic GBMs, or RuleFit. The average group metrics reported in most cases by DIA may miss cases of local discrimination, especially with complex, unconstrained models that can treat individuals very differently based on small changes in their data attributes.

DIA allows you to specify a **disparate impact variable** (the group variable that is analyzed), a **reference level** (the group level that other groups are compared to), and **user-defined thresholds for disparity**. Several tables are provided as part of the analysis:

- **Group metrics**: The aggregated metrics calculated per group. For example, true positive rates per group.
- **Group disparity**: This is calculated by dividing the metric_for_group by the reference_group_metric. Disparity is observed if this value falls outside of the user-defined thresholds.
- **Group parity**: This builds on Group disparity by converting the above calculation to a true or false value by applying the user-defined thresholds to the disparity values.

In accordance with the established four-fifths rule, user-defined thresholds are set to 0.8 and 1.25 by default. These thresholds will generally detect if the model is (on average) treating the non-reference group 20% more or less favorably than the reference group. Users are encouraged to set the user-defined thresholds to align with their organization's guidance on fairness thresholds.

Fairness Metrics

DIA calculates Marginal Error (ME), Adverse Impact Ratio (AIR), and Standardized Mean Difference (SMD) for binary models and SMD for regression models.

- **ME** is the difference between the percent of the control group members receiving a favorable outcome and the percent of the protected class members receiving a favorable outcome:

  $$\text{ME} \equiv 100 \cdot (\text{PR}(\hat{y} = 1 | X_c = 1) - \text{Pr}(\hat{y} = 1 | X_p = 1))$$

  Where:

  - $\hat{y}$ is the model decisions.
  - (X_c) and X_p are binary markers created from some demographic attribute.
  - c is the control group.
  - p is the protected group.

  - $Pr(\cdot)$ is the operator for conditional probability.

- **AIR** is equal to the ratio of the proportion of the protected class that receives a favorable outcome and the proportion of the control class that receives a favorable outcome:

  AIR $\equiv \frac{Pr(\hat{y}=1|X_p=1)}{Pr(\hat{y}=1|X_c=1)}$

  Where:

  - $\hat{y}$ is the model decisions.
  - X_p and X_c are binary markers created from some demographic attribute.
  - c is the control group.
  - p is the protected group.
  - $Pr()$ is the operator for conditional probability.

- **SMD** is used to assess disparities in continuous features such as income differences in employment analyses or interest rate differences in lending:

  SMD $\equiv \frac{\bar{\hat{y}}_p - \bar{\hat{y}}_c}{\sigma_{\hat{y}}}$

  Where:

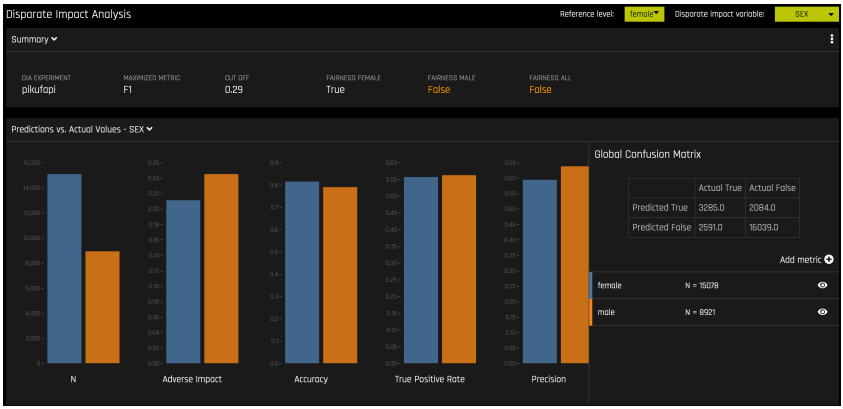
  - $\bar{\hat{y}}_p$ is the difference in the average protected class outcome.
  - $\bar{\hat{y}}_c$ is the control class outcome.
  - $\sigma_{\hat{y}}$ is a measure of the standard deviation of the population.

Notes:

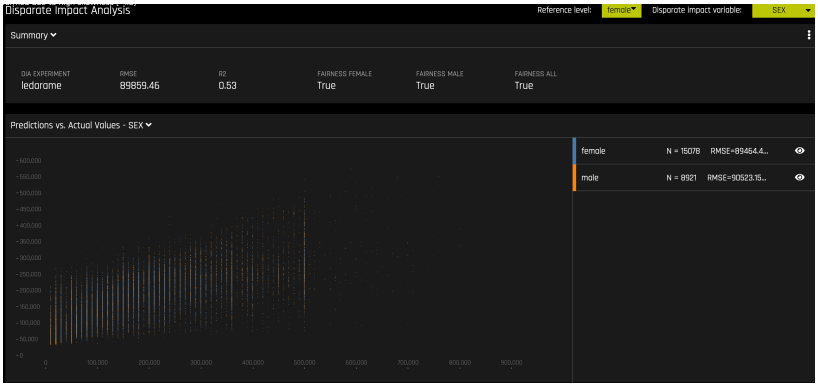
- Although the process of DIA is the same for both classification and regression experiments, the returned information is dependent on the type of experiment being interpreted. An analysis of a regression experiment returns an actual vs. predicted plot, while an analysis of a binary classification experiment returns confusion matrices. The above tables are provided for both types of experiments.

- Users are encouraged to consider the explanation dashboard to understand and augment results from disparate impact analysis. In addition to its established use as a fairness tool, users may want to consider disparate impact for broader model debugging purposes. For example, users can analyze the supplied confusion matrices and group metrics for important, non-demographic features in the Driverless AI model.

Classification Experiment



Regression Experiment



Sensitivity Analysis

Note: Sensitivity Analysis (SA) is not available for multiclass experiments.

Sensitivity Analysis (or "What if?") is a simple and powerful model debugging, explanation, fairness, and security tool. The idea behind SA is both direct and simple: Score your trained model on a single row, on multiple rows, or on an entire dataset of potentially interesting simulated values and compare the model's new outcome to the predicted outcome on the original data.

Beyond traditional assessment practices, sensitivity analysis of machine learning model predictions is perhaps the most important validation technique for machine learning models. Sensitivity analysis investigates whether model behavior and

outputs remain stable when data is intentionally perturbed or other changes are simulated in the data. Machine learning models can make drastically differing predictions for only minor changes in input variable values. For example, when looking at predictions that determine financial decisions, SA can be used to help you understand the impact of changing the most important input variables and the impact of changing socially sensitive variables (such as Sex, Age, Race, etc.) in the model. If the model changes in reasonable and expected ways when important variable values are changed, this can enhance trust in the model. Similarly, if the model changes to sensitive variables have minimal impact on the model, then this is an indication of fairness in the model predictions.

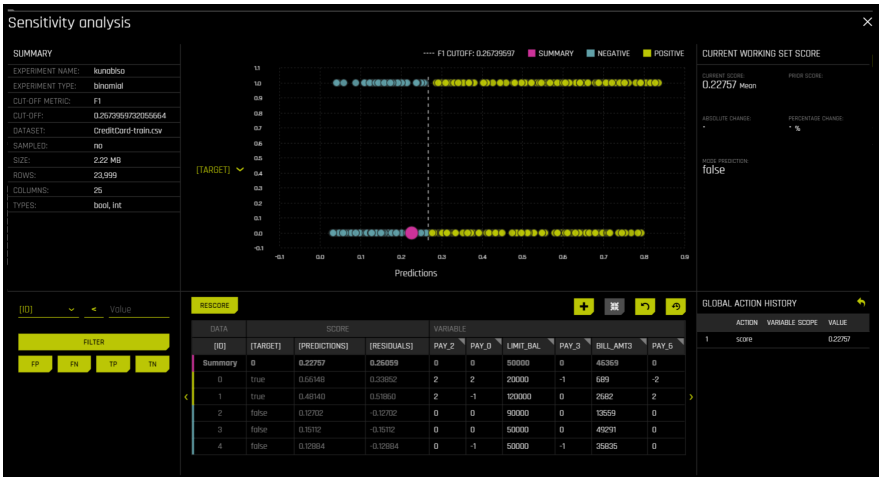
This page utilizes the What If Tool for displaying the SA information.

The top portion of this page includes:

- A summary of the experiment
- Predictions for a specified column. Change the column on the Y axis to view predictions for that column.
- The current working score set. This updates each time you rescore.

The bottom portion of this page includes:

- A filter tool for filtering the analysis. Choose a different column, predictions, or residuals. Set the filter type, then choose to filter by False Positive, False Negative, True Positive, or True Negative.
- Scoring chart. Click the **Rescore** button after applying a filter to update the scoring chart. This chart also allows you to add or remove variables, toggle the main chart aggregation, reset the data, and delete the global history while resetting the data.
- The current history of actions taken on this page. You can delete individual actions by selecting the action and then clicking the **Delete** button that appears.



Use Case 1: Using SA on a Single Row or on a Small Group of Rows

This section describes scenarios for using SA for explanation, debugging, security, or fairness when scoring a trained model on a single row or on a small group of rows.

- **Explanation**: Change values for a variable, and then rescore the model. View the difference between the original prediction and the new model prediction. If the change is big, then the changed variable is locally important.
- **Debugging**: Change values for a variable, and then rescore the model. View the difference between the original prediction and the new model prediction and determine whether the change to variable made the model more or less accurate.
- **Security**: Change values for a variable, and then rescore the model. View the difference between the original prediction and the new model prediction. If the change is big, then the user can, for example, inform their IT department that this variable can be used in an adversarial attack or inform the model makers that this variable should be more regularized.
- **Fairness**: Change values for a demographic variable, and then rescore the model. View the difference between the original prediction and the new model prediction. If change is big, then the user can consider using a different model, regularizing the model more, or applying post-hoc bias remediation techniques.
- **Random**: Set variables to random values, and then rescore the model. This can help you look for things the you might not have thought of.

Use Case 2: Using SA on an Entire Dataset and Trained Model

This section describes scenarios for using SA for explanation, debugging, security, or fairness when scoring a trained model for an entire dataset and trained predictive model.

- **Financial Stress Testing**: Assume the user wants to see how their loan default rates will change (according to their trained probability of default model) when they change an entire dataset to simulate that all their customers are under more financial stress (such as lower FICO scores, lower savings balances, higher unemployment, etc). Change the values of the variables in their entire dataset, and look at the **Percentage Change** in the average model score (default probability) on the original and new data. They can then use this discovered information along with external information and processes to understand whether their institution has enough cash on hand to be prepared for the simulated crisis.
- **Random**: Set variables to random values, and then rescore the model. This allows the user to look for things the user might not have thought of.

Additional Resources

Sensitivity Analysis on a Driverless AI Model: This ipynb uses the UCI credit card default data to perform sensitivity analysis and test model performance.

NLP Tokens

This plot is available for natural language processing (NLP) models. It is located in the **Dataset** tab on the Model Interpretation page (only visible for NLP models).

This plot shows both the global and local importance values of each token in a corpus (a large and structured set of texts). The corpus is automatically generated from text features used by Driverless AI models prior to the process of tokenization.

Local importance values are calculated by using the term frequency–inverse document frequency (TFIDF) as a weighting factor for each token in each row. The TFIDF increases proportionally to the number of times a token appears in a given document and is offset by the number of documents in the corpus that contain the token. Specify the row that you want to view, then click the **Search** button to see the local importance of each token in that row.

Global importance values are calculated by using the inverse document frequency (IDF), which measures how common or rare a given token is across all documents. (Default View)

Notes:

- MLI support for NLP is not available for multinomial experiments.
- MLI for NLP does not currently feature the option to remove stop words.
- By default, up to 10,000 tokens are created during the tokenization process. This value can be changed in the configuration.
- By default, Driverless AI uses up to 10,000 documents to extract tokens from. This value can be changed with the `config.mli_nlp_sample_limit` parameter. Downsampling is used for datasets that are larger than the default sample limit.
- Driverless AI does not currently generate a K-LIME scoring pipeline for MLI NLP problems.

NLP LOCO

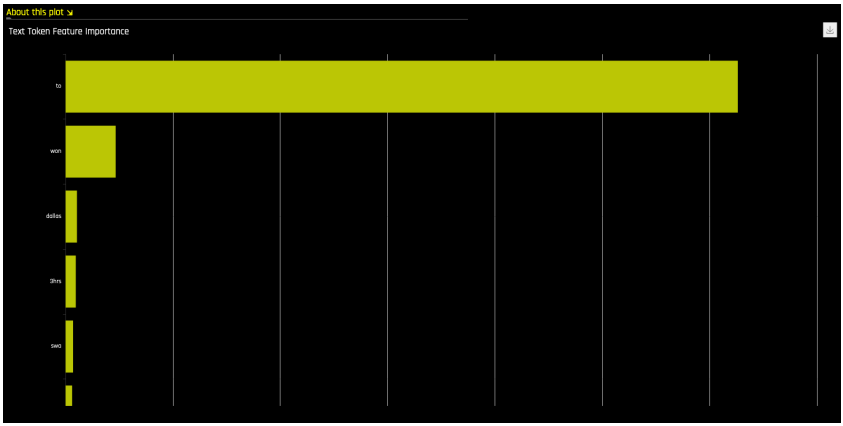
This plot is available for natural language processing (NLP) models.

This plot applies a leave-one-covariate-out (LOCO) styled approach to NLP models by removing a specific token from all text features in a record and predicting local importance without that token. The difference between the resulting score and the original score (token included) is useful when trying to determine how specific changes to text features alter the predictions made by the model.

Notes:

- MLI support for NLP is not available for multinomial experiments.
- Due to computational complexity, the global importance value is only calculated for N (20 by default) tokens. This value can be changed with the `mli_nlp_top_n` configuration option.

- A specific token selection method can be used by specifying one of the following options for the `mli_nlp_min_token_mode` configuration option:
  - `linspace`: Selects N evenly spaced tokens according to their IDF score (Default)
  - `top`: Selects top N tokens by IDF score
  - `bottom`: Selects bottom N tokens by IDF score
- Local values for NLP LOCO can take a significant amount of time to calculate depending on the specifications of your hardware.
- Driverless AI does not currently generate a K-LIME scoring pipeline for MLI NLP problems.



NLP Tokens

This plot is available for natural language processing (NLP) models.

This plot shows both the global and local importance values of each token in a corpus (a large and structured set of texts). The corpus is automatically generated from text features used by Driverless AI models prior to the process of tokenization.

Local importance values are calculated by using the term frequency–inverse document frequency (TFIDF) as a weighting factor for each token in each row. The TFIDF increases proportionally to the number of times a token appears in a given document and is offset by the number of documents in the corpus that contain the token. Specify the row that you want to view, then click the Search button to see the local importance of each token in that row.

Global importance values are calculated by using the inverse document frequency (IDF), which measures how common or rare a given token is across all documents. (Default View)

Notes:

- MLI support for NLP is not available for multinomial experiments.
- MLI for NLP does not currently feature the option to remove stop words.
- By default, up to 10,000 tokens are created during the tokenization process. This value can be changed in the configuration.
- By default, Driverless AI uses up to 10,000 documents to extract tokens from. This value can be changed with the `config.mli_nlp_sample_limit` parameter. Downsampling is used for datasets that are larger than the default sample limit.
- Driverless AI does not currently generate a K-LIME scoring pipeline for MLI NLP problems.

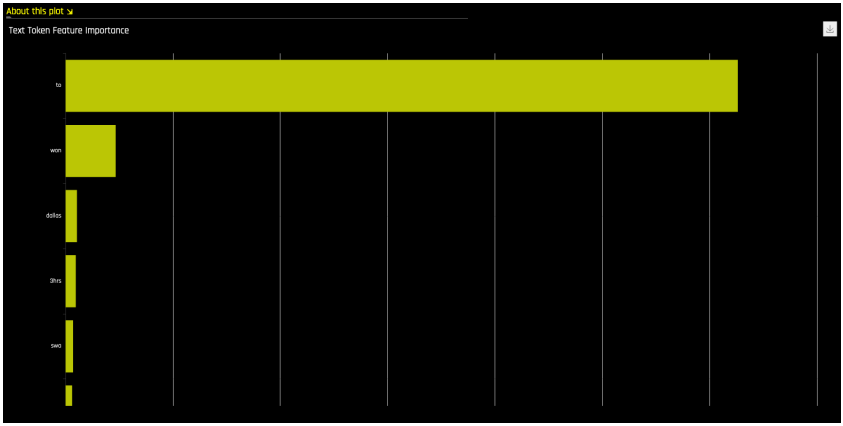
NLP LOCO

This plot is available for natural language processing (NLP) models.

This plot applies a leave-one-covariate-out (LOCO) styled approach to NLP models by removing a specific token from all text features in a record and predicting local importance without that token. The difference between the resulting score and the original score (token included) is useful when trying to determine how specific changes to text features alter the predictions made by the model.

Notes:

- MLI support for NLP is not available for multinomial experiments.
- Due to computational complexity, the global importance value is only calculated for N (20 by default) tokens. This value can be changed with the `mli_nlp_top_n` configuration option.
- A specific token selection method can be used by specifying one of the following options for the `mli_nlp_min_token_mode` configuration option:
  1. `linspace`: Selects N evenly spaced tokens according to their IDF score (Default)
  2. `top`: Selects top N tokens by IDF score
  3. `bottom`: Selects bottom N tokens by IDF score
- Local values for NLP LOCO can take a significant amount of time to calculate depending on the specifications of your hardware.
- Driverless AI does not currently generate a K-LIME scoring pipeline for MLI NLP problems.

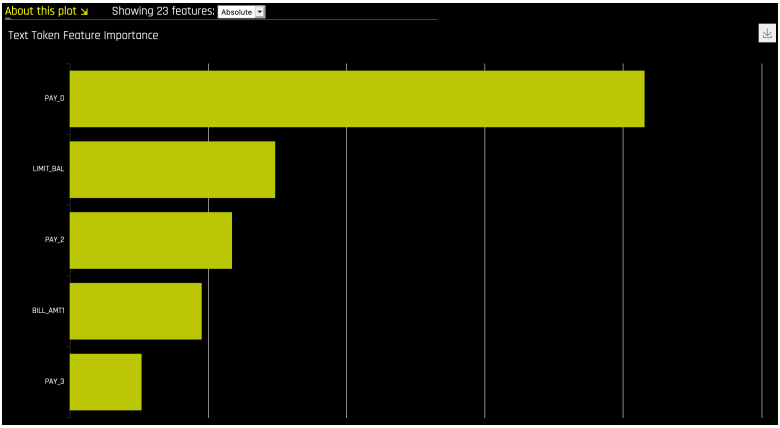


Permutation Feature Importance

Note: The Permutation-based feature importance plot is only available if the `autodoc_include_permutation_feature_importance` configuration option was enabled when starting Driverless AI or when starting the experiment. In addition, this plot is only available for binary classification and regression experiments.

Permutation-based feature importance shows how much a model's performance would change if a feature's values were permuted. If the feature has little

predictive power, shuffling its values should have little impact on the model's performance. If a feature is highly predictive, however, shuffling its values should decrease the model's performance. The difference between the model's performance before and after permuting the feature provides the feature's absolute permutation importance.



12.6.6 Surrogate Model Tab Plots

This section describes the plots that are available in the Surrogate Model Tab.

K-LIME and LIME-SUP

The MLI screen includes a K-LIME or LIME-SUP graph. A K-LIME graph is available by default when you interpret a model from the experiment page. When you create a new interpretation, you can instead choose to use LIME-SUP as the LIME method. Note that these graphs are essentially the same, but the K-LIME/LIME-SUP distinction provides insight into the LIME method that was used during model interpretation.

The K-LIME Technique

This plot is available for binary classification and regression models.

K-LIME is a variant of the LIME technique proposed by Ribeiro at al [12]. *K*-LIME generates global and local explanations that increase the transparency of the Driverless AI model, and allow model behavior to be validated and debugged by analyzing the provided plots, and comparing global and local explanations to one-another, to known standards, to domain knowledge, and to reasonable expectations.

K-LIME creates one global surrogate GLM on the entire training data and also creates numerous local surrogate GLMs on samples formed from K-means

clusters in the training data. The features used for K-means are selected from the Random Forest surrogate model's variable importance. The number of features used for K-means is the minimum of the top 25 percent of variables from the Random Forest surrogate model's variable importance and the max number of variables that can be used for K-means, which is set by the user in the config.toml setting for `mli_max_number_cluster_vars`. (Note, if the number of features in the dataset are less than or equal to 6, then all features are used for K-means clustering.) The previous setting can be turned off to use all features for k-means by setting `use_all_columns_klime_kmeans` in the config.toml file to `true`. All penalized GLM surrogates are trained to model the predictions of the Driverless AI model. The number of clusters for local explanations is chosen by a grid search in which the R2 between the Driverless AI model predictions and all of the local K-LIME model predictions is maximized. The global and local linear model's intercepts, coefficients, R2 values, accuracy, and predictions can all be used to debug and develop explanations for the Driverless AI model's behavior.

LIME-SUP explains local regions of the trained Driverless AI model in terms of the original variables. Local regions are defined by each leaf node path of the decision tree surrogate model instead of simulated, perturbed observation samples - as in the original LIME. For each local region, a local GLM model is trained on the original inputs and the predictions of the Driverless AI model. Then the parameters of this local GLM can be used to generate approximate, local explanations of the Driverless AI model.

The parameters of the global *K*-LIME model give an indication of overall linear feature importance and the overall average direction in which an input variable influences the Driverless AI model predictions. The global model is also used to generate explanations for very small clusters ($N < 20$) where fitting a local linear model is inappropriate.

The in-cluster linear model parameters can be used to profile the local region, to give an average description of the important variables in the local region, and to understand the average direction in which an input variable affects the Driverless AI model predictions. For a point within a cluster, the sum of the local linear model intercept and the products of each coefficient with their respective input variable value are the *K*-LIME prediction. By disaggregating the *K*-LIME predictions into individual coefficient and input variable value products, the local linear impact of the variable can be determined. This product is sometimes referred to as a reason code and is used to create explanations for the Driverless AI model's behavior.

In the following example, reason codes are created by evaluating and disaggregating a local linear model.

Given the row of input data with its corresponding Driverless AI and *K*-LIME predictions:

debt_to_income_ratio	credit_score	savings_acct_balance	observed_default	H2OAI_predicted_default	k-LIME_predicted_default
30	600	1000	1	0.85	0.9

And the local linear model:

$$y_{\text{K-LIME}} = 0.1 + 0.01 * debt_to_income_ratio + 0.0005 * credit_score + 0.0002 * savings_account_balance$$

It can be seen that the local linear contributions for each variable are:

- *debt_to_income_ratio*: 0.01 * 30 = 0.3
- *credit_score*: 0.0005 * 600 = 0.3
- *savings_acct_balance*: 0.0002 * 1000 = 0.2

Each local contribution is positive and thus contributes positively to the Driverless AI model's prediction of 0.85 for *H2OAI_predicted_default*. By taking into consideration the value of each contribution, reason codes for the Driverless AI decision can be derived. *debt_to_income_ratio* and *credit_score* would be the two largest negative reason codes, followed by *savings_acct_balance*.

The local linear model intercept and the products of each coefficient and corresponding value sum to the *K*-LIME prediction. Moreover it can be seen that these linear explanations are reasonably representative of the nonlinear model's behavior for this individual because the *K*-LIME predictions are within 5.5% of the Driverless AI model prediction. This information is encoded into English language rules which can be viewed by clicking the **Explanations** button.

Like all LIME explanations based on linear models, the local explanations are linear in nature and are offsets from the baseline prediction, or intercept, which represents the average of the penalized linear model residuals. Of course, linear approximations to complex non-linear response functions will not always create suitable explanations and users are urged to check the *K*-LIME plot, the local model R^2, and the accuracy of the *K*-LIME prediction to understand the validity of the *K*-LIME local explanations. When *K*-LIME accuracy for a given point or set of points is quite low, this can be an indication of extremely nonlinear behavior or the presence of strong or high-degree interactions in this local region of the Driverless AI response function. In cases where *K*-LIME linear models are not fitting the Driverless AI model well, nonlinear LOCO feature importance values may be a better explanatory tool for local model behavior. As *K*-LIME local explanations rely on the creation of *k*-means clusters, extremely wide input

data or strong correlation between input variables may also degrade the quality of K-LIME local explanations.

The LIME-SUP Technique

This plot is available for binary classification and regression models.

LIME-SUP explains local regions of the trained Driverless AI model in terms of the original variables. Local regions are defined by each leaf node path of the decision tree surrogate model instead of simulated, perturbed observation samples - as in the original LIME. For each local region, a local GLM model is trained on the original inputs and the predictions of the Driverless AI model. Then the parameters of this local GLM can be used to generate approximate, local explanations of the Driverless AI model.

The Global Interpretable Model Explanation Plot

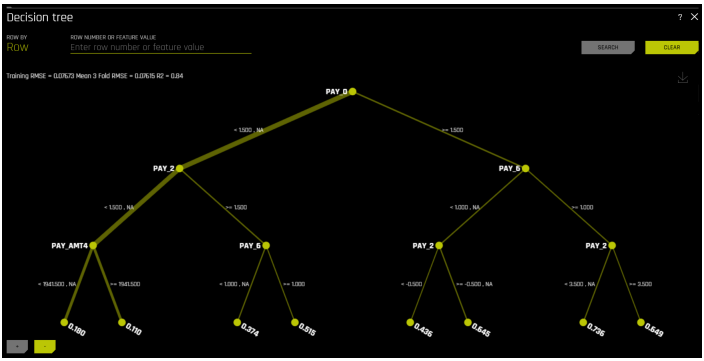
This plot shows Driverless AI model predictions and LIME model predictions in sorted order by the Driverless AI model predictions. This graph is interactive. Hover over the **Model Prediction**, **LIME Model Prediction**, or **Actual Target** radio buttons to magnify the selected predictions. Or click those radio buttons to disable the view in the graph. You can also hover over any point in the graph to view LIME reason codes for that value. By default, this plot shows information for the global LIME model, but you can change the plot view to show local results from a specific cluster. The LIME plot also provides a visual indication of the linearity of the Driverless AI model and the trustworthiness of the LIME explanations. The closer the local linear model approximates the Driverless AI model predictions, the more linear the Driverless AI model and the more accurate the explanation generated by the LIME local linear models.

Decision Tree

The decision tree surrogate model increases the transparency of the Driverless AI model by displaying an *approximate* flow-chart of the complex Driverless AI

model's decision making process. It also displays the most important variables in the Driverless AI model and the most important interactions in the Driverless AI model. The decision tree surrogate model can be used for visualizing, validating, and debugging the Driverless AI model by comparing the displayed decision-process, important variables, and important interactions to known standards, domain knowledge, and reasonable expectations. It is known to date back at least to 1996 (Craven and Shavlik).

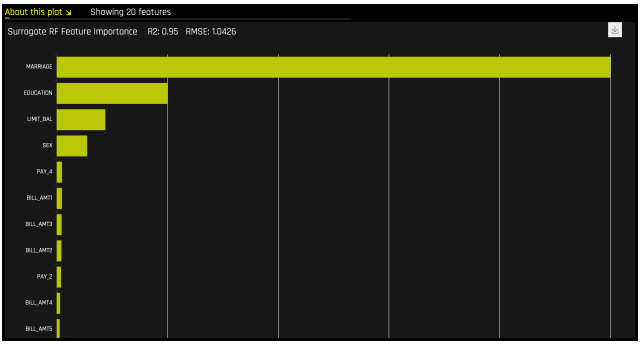
In the decision tree plot, the highlighted row shows the path to the highest probability leaf node and indicates the globally important variables and interactions that influence the Driverless AI model prediction for that row. The decision tree plot is available for binary classification and regression models.



Global Feature Importance vs Local Feature Importance

Global feature importance (yellow) is a measure of the contribution of an input variable to the overall predictions of the Driverless AI model. Global feature importance is calculated by aggregating the improvement in splitting criterion caused by a single variable across all of the decision trees in the Driverless AI model. Local feature importance (grey) is a measure of the contribution of an input variable to a single prediction of the Driverless AI model. Local feature importance is calculated by removing the contribution of a variable from every decision tree in the Driverless AI model and measuring the difference between the prediction with and without the variable. Both global and local variable importance are scaled so that the largest contributor has a value of 1.

Note: Engineered features are used for MLI when a time series experiment is built. This is because munged time series features are more useful features for MLI compared to raw time series features.



Random Forest LOCO

This plot is available for binary and multinomial classification models as well as regression models.

Local feature importance describes how the combination of the learned model rules or parameters and an individual row's attributes affect a model's prediction for that row while taking nonlinearity and interactions into effect. Local feature importance values reported in this plot are based on a variant of the leave-one-covariate-out (LOCO) method (Lei et al, 2017).

The LOCO-variant method for binary and regression models calculates each local feature importance by re-scoring the trained Driverless AI model for each feature in the row of interest, while removing the contribution to the model prediction of splitting rules that contain that feature throughout the ensemble. The original prediction is then subtracted from this modified prediction to find the raw, signed importance for the feature. All local feature importance values for the row are then scaled between 0 and 1 for direct comparison with global feature importance values.

The LOCO-variant method for multinomial models differs slightly in that it calculates row-wise local feature importance values by re-scoring the trained supervised model and measuring the impact of setting each variable to missing. The sum of the absolute value of differences across classes is then calculated for each dropped or replaced column.

Given the row of input data with its corresponding Driverless AI and *K*-LIME predictions:

debt_to_income_ratio	credit_score	savings_acct_balance	observed_default	H2OAI_predicted_default	k-LIME_predicted_default
30	600	1000	1	0.85	0.9

Taking the Driverless AI model as F(**X**), LOCO-variant feature importance values are calculated as follows.

First, the modified predictions are calculated:

$$F_{debt_to_income_ratio} = F(NA, 600, 1000) = 0.99$$

$$F_{credit_score} = F(30, NA, 1000) = 0.73$$

$$F_{savings_acct_balance} = F(30, 600, NA) = 0.82$$

Second, the original prediction is subtracted from each modified prediction to generate the unscaled local feature importance values:

$$\text{LOCO}_{debt_to_income_ratio} = F_{debt_to_income_ratio} - 0.85 = 0.99 - 0.85 = 0.14$$

$$\text{LOCO}_{credit_score} = F_{credit_score} - 0.85 = 0.73 - 0.85 = -0.12$$

$$\text{LOCO}_{savings_acct_balance} = F_{savings_acct_balance} - 0.85 = 0.82 - 0.85 = -0.03$$

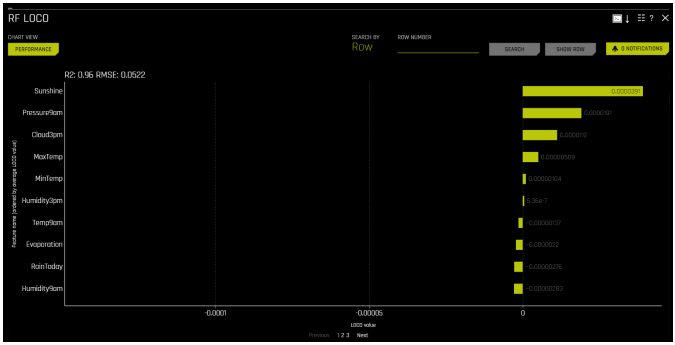
Finally LOCO values are scaled between 0 and 1 by dividing each value for the row by the maximum value for the row and taking the absolute magnitude of this quotient.

$$\text{Scaled}(\text{LOCO}_{debt_to_income_ratio}) = \text{Abs}(\text{LOCO}_{debt_to_income_ratio}/0.14) = 1$$

$$\text{Scaled}(\text{LOCO}_{credit_score}) = \text{Abs}(\text{LOCO}_{credit_score}/0.14) = 0.86$$

$$\text{Scaled}(\text{LOCO}_{savings_acct_balance}) = \text{Abs}(\text{LOCO}_{savings_acct_balance}/0.14) = 0.21$$

One drawback to these LOCO-variant feature importance values is, unlike *K*-LIME, it is difficult to generate a mathematical error rate to indicate when LOCO values may be questionable.



NLP Surrogate Models

These plots are available for natural language processing (NLP) models.

For NLP surrogate models, Driverless AI creates a TFIDF matrix by tokenizing all text features. The resulting frame is appended to numerical or categorical columns from the training dataset, and the original text columns are removed. This frame is then used for training surrogate models that have prediction columns consisting of tokens and the original numerical or categorical features.

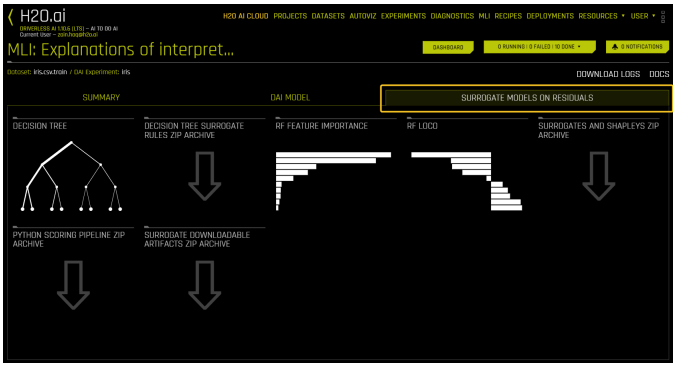
Notes:

- MLI support for NLP is not available for multinomial experiments.
- Each row in the TFIDF matrix contains N columns, where N is the total number of tokens in the corpus with values that are appropriate for that row (0 if absent).
- Driverless AI does not currently generate a K-LIME scoring pipeline for MLI NLP problems.

Running Surrogate Models on Residuals

In Driverless AI, residuals (differences between observed and predicted values) can be used as targets in MLI surrogate models for the purpose of debugging models. The method used to calculate residuals varies depending on the type of problem. For classification problems, logloss residuals are calculated for a specified class. For regression problems, residuals are determined by calculating the square of the difference between targeted and predicted values.

To run MLI surrogate models on residuals, enable the **Debug Model Residuals** interpretation expert setting. For classification experiments, specify a class to use as an outcome of interest with the **Class for Debugging Classification Model Logloss Residuals** interpretation expert setting (not visible for regression problems). You can view the models by clicking the **Surrogate Models on Residuals** tab once the interpretation is complete.



Partial Dependence and Individual Conditional Expectation

A Partial Dependence and ICE plot is available for both Driverless AI and surrogate models. Refer to the previous Partial Dependence and Individual Conditional Expectation section for more information about this plot.

12.7 General Considerations

12.7.1 Machine Learning and Approximate Explanations

For years, common sense has deemed the complex, intricate formulas created by training machine learning algorithms to be uninterpretable. While great advances have been made in recent years to make these often nonlinear, non-monotonic, and non-continuous machine-learned response functions more understandable (Hall et al, 2017 [7]), it is likely that such functions will never be as directly or universally interpretable as more traditional linear models.

Why consider machine learning approaches for inferential purposes? In general, linear models focus on understanding and predicting average behavior, whereas machine-learned response functions can often make accurate, but more difficult to explain, predictions for subtler aspects of modeled phenomenon. In a sense, linear models create very exact interpretations for approximate models. The approach here seeks to make approximate explanations for very exact models. It is quite possible that an approximate explanation of an exact model may have as much, or more, value and meaning than the exact interpretations of an approximate model. Moreover, the use of machine learning techniques for inferential or predictive purposes does not preclude using linear models for interpretation (Ribeiro et al, 2016 [12]).

12.7.2 The Multiplicity of Good Models in Machine Learning

It is well understood that for the same set of input variables and prediction targets, complex machine learning algorithms can produce multiple accurate models with very similar, but not exactly the same, internal architectures (Brieman, 2001 [1]). This alone is an obstacle to interpretation, but when using these types of algorithms as interpretation tools or with interpretation tools it is important to remember that details of explanations will change across multiple accurate models.

12.7.3 Expectations for Consistency Between Explanatory Techniques

- The decision tree surrogate is a global, nonlinear description of the Driverless AI model behavior. Variables that appear in the tree should have a direct relationship with variables that appear in the global feature importance plot. For certain, more linear Driverless AI models, variables that appear in the decision tree surrogate model may also have large coefficients in the global K-LIME model.

- K-LIME explanations are linear, do not consider interactions, and represent offsets from the local linear model intercept. LOCO importance values are nonlinear, do consider interactions, and do not explicitly consider a linear intercept or offset. LIME explanations and LOCO importance values are not expected to have a direct relationship but can align roughly as both are measures of a variable's local impact on a model's predictions, especially in more linear regions of the Driverless AI model's learned response function.

- ICE is a type of nonlinear sensitivity analysis which has a complex relationship to LOCO feature importance values. Comparing ICE to LOCO can only be done at the value of the selected variable that actually appears in the selected row of the training data. When comparing ICE to LOCO the total value of the prediction for the row, the value of the variable in the selected row, and the distance of the ICE value from the average prediction for the selected variable at the value in the selected row must all be considered.

- ICE curves that are outside the standard deviation of partial dependence would be expected to fall into less populated decision paths of the decision tree surrogate; ICE curves that lie within the standard deviation of partial dependence would be expected to belong to more common decision paths.

- Partial dependence takes into consideration nonlinear, but average, behavior of the complex Driverless AI model without considering interactions. Variables with consistently high partial dependence or partial dependence that swings widely across an input variable's domain will likely also have high global importance values. Strong interactions between input variables can cause ICE values to diverge from partial dependence values.

13 Viewing Explanations

Note: Not all explanatory functionality is available for multinomial classification scenarios.

Driverless AI provides easy-to-read explanations for a completed model. You can view these by clicking the **Explanations** button on the **Dashboard** page. Note that the **Dashboard** button is only available for completed experiments.

The UI allows you to view global, cluster-specific, and local reason codes. You can also export the explanations to CSV.

- **Global Reason Codes**: To view global reason codes, click **Cluster** and select **Global** from the list of options. With Global selected, click the **Explanations** button located in the upper-right corner.

- **Cluster Reason Codes**: To view reason codes for a specific cluster, click **Cluster** and select a specific cluster from the list of options. With a cluster selected, click the **Explanations** button.

- **Local Reason Codes by Row Number**: To view local reason codes for a specific row, select a point on the graph or type a value in the **Row Number or Feature Value** field. With a value selected, click the **Explanations** button.

- **Local Reason Codes by ID**: Identifier columns cannot be specified by the user - MLI makes this choice automatically by choosing columns whose values are unique (dataset row count equals the number of unique values in a column). To find a row by identifier column, choose **ID** from the drop-down menu (if it meets the logic of being an identifier column), and then specify a value. With a value selected, click the **Explanations** button.

14 Score on Another Dataset

After you generate a model, you can use that model to make predictions on another dataset.

1. Click the **Experiments** link in the top menu and select the experiment that you want to use.
2. On the Experiments page, click the **Score on Another Dataset** button.
3. Locate the new dataset that you want to score on. Note that this new dataset must include the same columns as the dataset used in selected experiment.
4. Click **Select** at the top of the screen. This immediately starts the scoring process.
5. Click the **Download Predictions** button after scoring is complete.

15 Transform Another Dataset

When a training dataset is used in an experiment, Driverless AI transforms the data into an improved, feature engineered dataset. (Refer to About Driverless AI Transformations for more information about the transformations that are provided in Driverless AI.) But what happens when new rows are added to your dataset? In this case, you can specify to transform the new dataset after adding it to Driverless AI, and the same transformations that Driverless AI applied to the original dataset will be applied to these new rows.

Follow these steps to transform another dataset. Note that this assumes the new dataset has been added to Driverless AI already.

Note: **Transform Another Dataset** is not available for Time Series experiments.

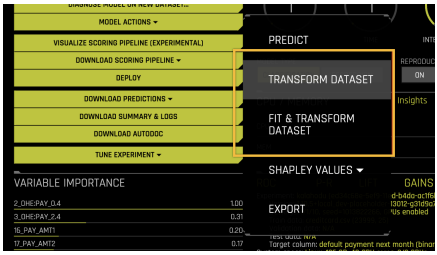
1. On the completed experiment page for the original dataset, click the **Transform Another Dataset** button.

2. Select the new training dataset that you want to transform. Note that this must have the same number columns as the original dataset.

3. In the **Select** drop down, specify a validation dataset to use with this dataset, or specify to split the training data. If you specify to split the data, then you also specify the split value (defaults to 25 percent) and the seed (defaults to 1234). **Note**: To ensure the transformed dataset respects the row order, choose a validation dataset instead of splitting the training data. Splitting the training data will result in a shuffling of the row order.

4. Optionally specify a test dataset. If specified, then the output also include the final test dataset for final scoring.

5. Click **Launch Transformation**.



The following datasets will be available for download upon successful completion:

- Training dataset (not for cross validation)
- Validation dataset for parameter tuning
- Test dataset for final scoring. This option is available if a test dataset was used.

16 The Driverless AI Scoring Pipelines

Driverless AI provides several Scoring Pipelines for experiments and/or interpreted models.

- A standalone Python Scoring Pipeline is available for experiments and interpreted models.

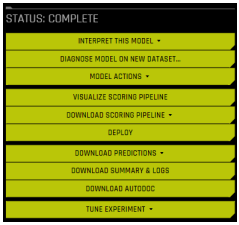
- A low-latency, standalone MOJO Scoring Pipeline is available for experiments, with both Java and C++ backends.

The Python Scoring Pipeline is implemented as a Python whl file. While this allows for a single process scoring engine, the scoring service is generally implemented as a client/server architecture and supports interfaces for TCP and HTTP.

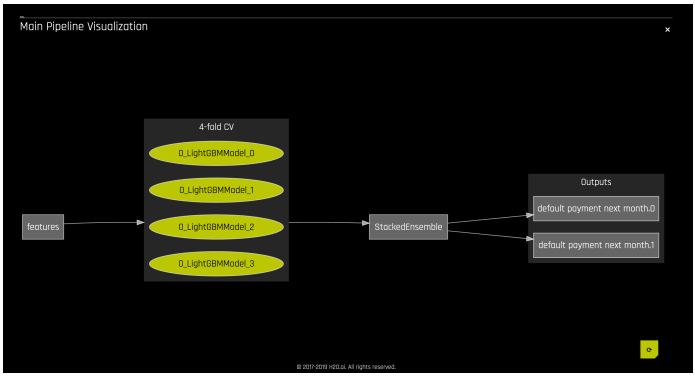
The MOJO Scoring Pipeline provides a standalone scoring pipeline that converts experiments to MOJOs, which can be scored in real time. The MOJO Scoring Pipeline is available as either a pure Java runtime or a C++ runtime. For the C++ runtime, both Python and R wrappers are provided.

16.1 Visualize the Scoring Pipeline

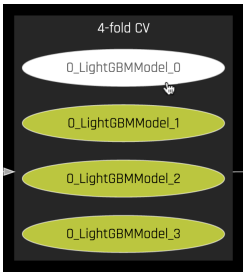
A visualization of the scoring pipeline is available for each completed experiment. **Note**: This pipeline is best viewed in the latest version of Chrome.

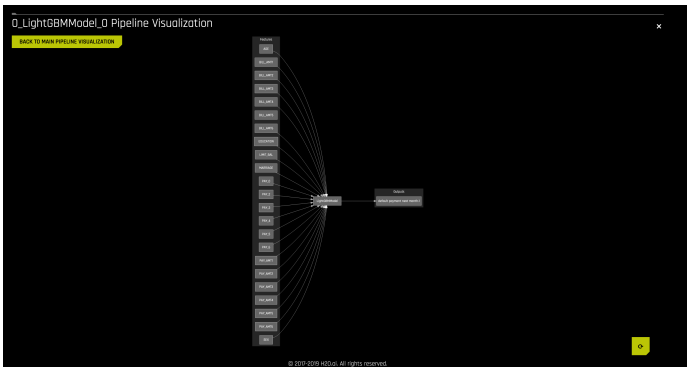


Click the **Visualize Scoring Pipeline (Experimental)** button on the completed experiment page to view the visualization.

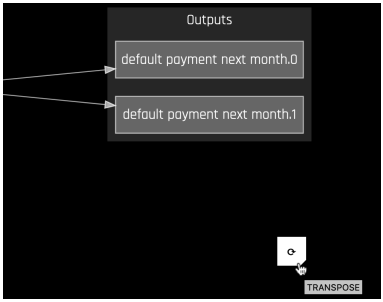


To view a visual representation of a specific model, click on the oval that corresponds with that model.





To change the orientation of the visualization, click the **Transpose** button in the bottom right-hand corner of the screen.



16.2 Which Pipeline Should I Use?

Driverless AI provides a Python Scoring Pipeline, an MLI Standalone Scoring Pipeline, and a MOJO Scoring Pipeline. Consider the following when determining the scoring pipeline that you want to use.

- For all pipelines, the higher the accuracy, the slower the scoring.
- The Python Scoring Pipeline is slower but easier to use than the MOJO scoring pipeline.
- When running the Python Scoring Pipeline:
  - HTTP is easy and is supported by virtually any language. HTTP supports RESTful calls via curl, wget, or supported packages in various scripting languages.
  - TCP is a bit more complex, though faster. TCP also requires Thrift, which currently does not handle NAs.
- The MOJO Scoring Pipeline is flexible and is faster than the Python Scoring Pipeline, but it requires a bit more coding. The MOJO Scoring Pipeline is available as either a pure Java runtime or a C++ runtime.
- The MLI Standalone Python Scoring Pipeline can be used to score interpreted models but only supports k-LIME reason codes.
  - For obtaining k-LIME reason codes from an MLI experiment, use the MLI Standalone Python Scoring Pipeline. k-LIME reason codes are available for all models.
  - For obtaining Shapley reason codes from an MLI experiment, use the DAI Standalone Python Scoring Pipeline. Shapley is only available for XGBoost and LightGBM models. Note that obtaining Shapley reason codes through the Python Scoring Pipeline can be time consuming.

16.3 Driverless AI Standalone Python Scoring Pipeline

As indicated earlier, a scoring pipeline is available after a successfully completed experiment. This package contains an exported model and Python 3.6 source code examples for productionizing models built using H2O Driverless AI.

The files in this package allow you to transform and score on new data in a couple of different ways:

- From Python 3.6, you can import a scoring module, and then use the module to transform and score on new data.
- From other languages and platforms, you can use the TCP/HTTP scoring service bundled with this package to call into the scoring pipeline module through remote procedure calls (RPC).

16.3.1 Python Scoring Pipeline Files

The **scoring-pipeline** folder includes the following notable files:

- **example.py**: An example Python script demonstrating how to import and score new records.
- **run_example.sh**: Runs example.py (also sets up a virtualenv with prerequisite libraries).
- **tcp_server.py**: A standalone TCP server for hosting scoring services.
- **http_server.py**: A standalone HTTP server for hosting scoring services.
- **run_tcp_server.sh**: Runs TCP scoring service (runs server.py).
- **run_http_server.sh**: Runs HTTP scoring service (runs server.py).
- **example_client.py**: An example Python script demonstrating how to communicate with the scoring server.
- **run_tcp_client.sh**: Demonstrates how to communicate with the scoring service via TCP (runs example_client.py).
- **run_http_client.sh**: Demonstrates how to communicate with the scoring service via HTTP (using curl).

16.3.2 Quick Start - Recommended Method

This is the recommended method for running the Python Scoring Pipeline. Use this method if:

- You have an air gapped environment with no access to the Internet.

- You are running Power.
- You want an easy quick start approach.

Prerequisites

- A valid Driverless AI license key
- A completed Driverless AI experiment
- Downloaded Python Scoring Pipeline

Running the Python Scoring Pipeline - Recommended

1. On https://www.h2o.ai/download/, download the TAR SH version of Driverless AI (for either Linux or IBM Power).
2. Use bash to execute the download. This creates a new dai-nnn folder.
3. Change directories into the new Driverless AI folder.

```
cd dai-nnn directory.
```

4. Run the following to install the Python Scoring Pipeline for your completed Driverless AI experiment:

```
./dai-env.sh pip install /path/to/your/scoring_experiment.whl
```

5. Run the following command to run the included scoring pipeline example:

```
DRIVERLESS_AI_LICENSE_KEY="pastekeyhere"
     SCORING_PIPELINE_INSTALL_DEPENDENCIES=0 ./dai-env.sh
```

16.3.3 Quick Start - Alternative Method

This section describes an alternative method for running the Python Scoring Pipeline. This version requires Internet access. It is also not supported on Power machines.

Prerequisites

The following are required in order to run the downloaded scoring pipeline.

- The scoring module and scoring service are supported only on Linux x86_64 with Python 3.6 and OpenBLAS.
- The scoring module and scoring service download additional packages at install time and require Internet access. Depending on your network environment, you might need to set up internet access via a proxy.
- Valid Driverless AI license. Driverless AI requires a license to be specified in order to run the Python Scoring Pipeline.

- Apache Thrift (to run the TCP scoring service)
- Linux x86_64 environment
- Python 3.6
- libopenblas-dev (required for H2O4GPU)
- Internet access to download and install packages. Note that depending on your environment, you may also need to set up proxy.
- OpenCL

Examples of how to install these prerequisites are below:

Installing Python 3.11 and OpenBlas Ubuntu 16.10+

```
$ sudo apt install python3.11 python3.11-dev python3-pip python3-dev \
    python-virtualenv python3-virtualenv libopenblas-dev
```

Installing Python 3.11 and OpenBLAS on Ubuntu 16.4

```
$ sudo add-apt-repository ppa:deadsnakes/ppa
$ sudo apt-get update
$ sudo apt-get install python3.11 python3.11-dev python3-pip python3-dev \
    python-virtualenv python3-virtualenv libopenblas-dev
```

Installing Conda 3.6

You can install Conda using either Anaconda or Miniconda. Refer to the links below for more information:

- Anaconda: https://docs.anaconda.com/anaconda/install.html
- Miniconda: https://conda.io/docs/user-guide/install/index.html

License Specification

Driverless AI requires a license to be specified in order to run the Python Scoring Pipeline. The license can be specified via an environment variable:

```
# Set DRIVERLESS_AI_LICENSE_FILE, the path to the Driverless AI license file
%env DRIVERLESS_AI_LICENSE_FILE="/home/ubuntu/license/license.sig"

# Set DRIVERLESS_AI_LICENSE_KEY, the Driverless AI license key (Base64
     encoded string)
%env DRIVERLESS_AI_LICENSE_KEY="oLqLZXMI0y..."
```

Installing the Thrift Compiler

Thrift is required to run the scoring service in TCP mode, but it is not required to run the scoring module. The following steps are available on the

Thrift documentation site at: https://thrift.apache.org/docs/BuildingFromSource.

```
$ sudo apt-get install automake bison flex g++ git libevent-dev \
    libssl-dev libtool make pkg-config libboost-all-dev ant
$ wget https://github.com/apache/thrift/archive/0.10.0.tar.gz
$ tar -xvf 0.10.0.tar.gz
$ cd thrift-0.10.0
$ ./bootstrap.sh
$ ./configure
$ make
$ sudo make install
```

Run the following to refresh the runtime shared after installing Thrift:

```
$ sudo ldconfig /usr/local/lib
```

Running the Python Scoring Pipeline - Alternative Method

1. On the completed Experiment page, click on the **Download Scoring Pipeline** button to download the **scorer.zip** file for this experiment onto your local machine.
2. Unzip the scoring pipeline.

After the pipeline is downloaded and unzipped, you will be able to run the scoring module and the scoring service.

Score from a Python Program

If you intend to score from a Python program, run the scoring module example. (Requires Linux x86_64 and Python 3.6.)

```
$ export DRIVERLESS_AI_LICENSE_FILE="/path/to/license.sig"
$ bash run_example.sh
```

Score Using a Web Service

If you intend to score using a web service, run the HTTP scoring server example. (Requires Linux x86_64 and Python 3.6.)

```
$ export DRIVERLESS_AI_LICENSE_FILE="/path/to/license.sig"
$ bash run_http_server.sh
$ bash run_http_client.sh
```

Score Using a Thrift Service

If you intend to score using a Thrift service, run the TCP scoring server example. (Requires Linux x86_64, Python 3.6 and Thrift.)

```
$ export DRIVERLESS_AI_LICENSE_FILE="/path/to/license.sig"
$ bash run_tcp_server.sh
$ bash run_tcp_client.sh
```

Note: By default, the `run_*.sh` scripts mentioned above create a virtual environment using virtualenv and pip, within which the Python code is executed. The scripts can also leverage Conda (Anaconda/Mininconda) to create Conda virtual environment and install required package dependencies. The package manager to use is provided as an argument to the script.

```
# to use conda package manager
$ export DRIVERLESS_AI_LICENSE_FILE="/path/to/license.sig"
$ bash run_example.sh --pm conda

# to use pip package manager
$ export DRIVERLESS_AI_LICENSE_FILE="/path/to/license.sig"
$ bash run_example.sh --pm pip
```

Note: If you experience errors while running any of the above scripts, please check to make sure your system has a properly installed and configured Python 3.6 installation. Refer to the Troubleshooting Python Environment Issues section at the end of this chapter to see how to set up and test the scoring module using a cleanroom Ubuntu 16.04 virtual machine.

16.3.4 The Python Scoring Module

The scoring module is a Python module bundled into a standalone wheel file (name scoring_*.whl). All the prerequisites for the scoring module to work correctly are listed in the requirements.txt file. To use the scoring module, all you have to do is create a Python virtualenv, install the prerequisites, and then import and use the scoring module as follows:

```
# See 'example.py' for complete example.
from scoring_487931_20170921174120_b4066 import Scorer
scorer = Scorer()       # Create instance.
score = scorer.score([  # Call score()
    7.416,              # sepal_len
    3.562,              # sepal_wid
    1.049,              # petal_len
    2.388,              # petal_wid
])
```

The scorer instance provides the following methods (and more):

- score(list): Score one row (list of values).
- score_batch(df): Score a Pandas dataframe.
- fit_transform_batch(df): Transform a Pandas dataframe.
- get_target_labels(): Get target column labels (for classification problems).

The process of importing and using the scoring module is demonstrated by the bash script run_example.sh, which effectively performs the following steps:

```
# See 'run_example.sh' for complete example.
$ virtualenv -p python3.11 env
$ source env/bin/activate
$ pip install --use-deprecated=legacy-resolver -r requirements.txt
$ export DRIVERLESS_AI_LICENSE_FILE="/path/to/license.sig"
$ python example.py
```

16.3.5 The Scoring Service

The scoring service hosts the scoring module as an HTTP or TCP service. Doing this exposes all the functions of the scoring module through remote procedure calls (RPC). In effect, this mechanism allows you to invoke scoring functions from languages other than Python on the same computer or from another computer on a shared network or on the Internet.

The scoring service can be started in two ways:

- In TCP mode, the scoring service provides high-performance RPC calls via Apache Thrift (https://thrift.apache.org/) using a binary wire protocol.
- In HTTP mode, the scoring service provides JSON-RPC 2.0 calls served by Tornado (http://www.tornadoweb.org).

Scoring operations can be performed on individual rows (row-by-row) or in batch mode (multiple rows at a time).

Scoring Service - TCP Mode (Thrift)

The TCP mode allows you to use the scoring service from any language supported by Thrift, including C, C++, C#, Cocoa, D, Dart, Delphi, Go, Haxe, Java, Node.js, Lua, perl, PHP, Python, Ruby and Smalltalk.

To start the scoring service in TCP mode, you will need to generate the Thrift bindings once, then run the server:

```
# See 'run_tcp_server.sh' for complete example.
$ thrift --gen py scoring.thrift
$ python tcp_server.py --port=9090
```

Note that the Thrift compiler is only required at build-time. It is not a run time dependency, i.e. once the scoring services are built and tested, you do not need to repeat this installation process on the machines where the scoring services are intended to be deployed.

To call the scoring service, simply generate the Thrift bindings for your language of choice, then make RPC calls via TCP sockets using Thrift's buffered transport in conjunction with its binary protocol.

```
# See 'run_tcp_client.sh' for complete example.
$ thrift --gen py scoring.thrift

# See 'example_client.py' for complete example.
socket = TSocket.TSocket('localhost', 9090)
transport = TTransport.TBufferedTransport(socket)
protocol = TBinaryProtocol.TBinaryProtocol(transport)
client = ScoringService.Client(protocol)
transport.open()
row = Row()
row.sepalLen = 7.416  # sepal_len
row.sepalWid = 3.562  # sepal_wid
row.petalLen = 1.049  # petal_len
row.petalWid = 2.388  # petal_wid
scores = client.score(row)
transport.close()
```

You can reproduce the exact same result from other languages, e.g. Java:

```
$ thrift --gen java scoring.thrift

// Dependencies:
// commons-codec-1.9.jar
// commons-logging-1.2.jar
// httpclient-4.4.1.jar
// httpcore-4.4.1.jar
// libthrift-0.10.0.jar
// slf4j-api-1.7.12.jar

import ai.h2o.scoring.Row;
import ai.h2o.scoring.ScoringService;
import org.apache.thrift.TException;
import org.apache.thrift.protocol.TBinaryProtocol;
import org.apache.thrift.transport.TSocket;
import org.apache.thrift.transport.TTransport;
import java.util.List;

public class Main {
  public static void main(String[] args) {
    try {
      TTransport transport = new TSocket("localhost", 9090);
      transport.open();

      ScoringService.Client client = new ScoringService.Client(
        new TBinaryProtocol(transport));

      Row row = new Row(7.642, 3.436, 6.721, 1.020);
      List<Double> scores = client.score(row);
      System.out.println(scores);

      transport.close();
    } catch (TException ex) {
      ex.printStackTrace();
    }
  }
}
```

Scoring Service - HTTP Mode (JSON-RPC 2.0)

The HTTP mode allows you to use the scoring service using plaintext JSON-RPC calls. This is usually less performant compared to Thrift, but has the

advantage of being usable from any HTTP client library in your language of choice, without any dependency on Thrift.

See http://www.jsonrpc.org/specification for JSON-RPC documentation.

To start the scoring service in HTTP mode:

```
# See 'run_http_server.sh' for complete example.
$ export DRIVERLESS_AI_LICENSE_FILE="/path/to/license.sig"
$ python http_server.py --port=9090
```

To invoke scoring methods, compose a JSON-RPC message and make a HTTP POST request to http://host:port/rpc as follows:

```
# See 'run_http_client.sh' for complete example.
$ curl http://localhost:9090/rpc \
  --header "Content-Type: application/json" \
  --data @- <<EOF
 {
  "id": 1,
  "method": "score",
  "params": {
    "row": [ 7.486, 3.277, 4.755, 2.354 ]
  }
 }
EOF
```

Similarly, you can use any HTTP client library to reproduce the above result. For example, from Python, you can use the requests module as follows:

```
import requests
row = [7.486, 3.277, 4.755, 2.354]
req = dict(id=1, method='score', params=dict(row=row))
res = requests.post('http://localhost:9090/rpc', data=req)
print(res.json()['result'])
```

16.3.6 Python Scoring Pipeline FAQ

Why am I getting a "TensorFlow is disabled" message when I run the Python Scoring Pipeline?

If you ran an experiment when TensorFlow was enabled and then attempt to run the Python Scoring Pipeline, you may receive a message similar to the following:

```
TensorFlow is disabled. To enable, export DRIVERLESS_AI_ENABLE_TENSORFLOW=1
     or set enable_tensorflow=true in config.toml.
```

This can be fixed by enabling the DRIVERLESS_AI_ENABLE_TENSORFLOW flag when running the Python Scoring Pipeline. For example:

```
$ export DRIVERLESS_AI_LICENSE_FILE="/path/to/license.sig"
$ DRIVERLESS_AI_ENABLE_TENSORFLOW=1 bash run_example.sh
```

16.3.7 Troubleshooting Python Environment Issues

The following instructions describe how to set up a cleanroom Ubuntu 16.04 virtual machine to test that this scoring pipeline works correctly.

Prerequisites:

- Install Virtualbox: sudo apt-get install virtualbox
- Install Vagrant: https://www.vagrantup.com/downloads.html

1. Create configuration files for Vagrant.
   - bootstrap.sh: contains commands to set up Python 3.6 and OpenBLAS.
   - Vagrantfile: contains virtual machine configuration instructions for Vagrant and VirtualBox.

```
----- bootstrap.sh -----

#!/usr/bin/env bash

sudo apt-get -y update
sudo apt-get -y install apt-utils build-essential python-software-
     properties software-properties-common zip libopenblas-dev
sudo add-apt-repository -y ppa:deadsnakes/ppa
sudo apt-get update -yqq
sudo apt-get install -y python3.11 python3.11-dev python3-pip python3-
     dev python-virtualenv python3-virtualenv

# end of bootstrap.sh

----- Vagrantfile -----

# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure(2) do |config|
  config.vm.box = "ubuntu/xenial64"
  config.vm.provision :shell, path: "bootstrap.sh", privileged: false
  config.vm.hostname = "h2o"
  config.vm.provider "virtualbox" do |vb|
    vb.memory = "4096"
  end
end

# end of Vagrantfile
```

2. Launch the VM and SSH into it. Note that we are also placing the scoring pipeline in the same directory so that we can access it later inside the VM.

```
cp /path/to/scorer.zip .
vagrant up
vagrant ssh
```

3. Test the scoring pipeline inside the virtual machine.

```
cp /vagrant/scorer.zip .
unzip scorer.zip
cd scoring-pipeline/
export DRIVERLESS_AI_LICENSE_FILE="/path/to/license.sig"
bash run_example.sh
```

At this point, you should see scores printed out on the terminal. If not, contact us at support@h2o.ai.

16.4 Driverless AI MLI Standalone Scoring Package

This package contains an exported model and Python 3.6 source code examples for productionizing models built using H2O Driverless AI Machine Learning Interpretability (MLI) tool. This is only available for interpreted models.

The files in this package allow you to obtain reason codes for a given row of data a couple of different ways:

- From Python 3.6, you can import a scoring module, and then use the module to transform and score on new data.
- From other languages and platforms, you can use the TCP/HTTP scoring service bundled with this package to call into the scoring pipeline module through remote procedure calls (RPC).

16.4.1 MLI Python Scoring Package Files

The **scoring-pipeline-mli** folder includes the following notable files:

- **example.py**: An example Python script demonstrating how to import and interpret new records.
- **run_example.sh**: Runs example.py (This also sets up a virtualenv with prerequisite libraries.)
- **run_example_shapley.sh**: Runs example_shapley.py. This compares K-LIME and Driverless AI Shapley reason codes.
- **tcp_server.py**: A standalone TCP server for hosting MLI services.
- **http_server.py**: A standalone HTTP server for hosting MLI services.
- **run_tcp_server.sh**: Runs the TCP scoring service (specifically, tcp_server.py).
- **run_http_server.sh**: Runs HTTP scoring service (runs http_server.py).
- **example_client.py**: An example Python script demonstrating how to communicate with the MLI server.

- **example_shapley.py**: An example Python script demonstrating how to compare K-LIME and Driverless AI Shapley reason codes.
- **run_tcp_client.sh**: Demonstrates how to communicate with the MLI service via TCP (runs example_client.py).
- **run_http_client.sh**: Demonstrates how to communicate with the MLI service via HTTP (using curl).

16.4.2 Quick Start - Recommended Method

This is the recommended method for running the MLI Scoring Pipeline. Use this method if:

- You have an air gapped environment with no access to the Internet.
- You are running Power.
- You want an easy quick start approach.

Prerequisites

- A valid Driverless AI license key.
- A completed Driverless AI experiment.
- Downloaded MLI Scoring Pipeline.

Running the MLI Scoring Pipeline - Recommended

1. Download the TAR SH version of Driverless AI from https://www.h2o.ai/download/ (for either Linux or IBM Power).
2. Use bash to execute the download. This creates a new dai-nnn folder.
3. Change directories into the new Driverless AI folder.
4. Run the following to install the Python Scoring Pipeline for your completed Driverless AI experiment:

```
./dai-env.sh pip install /path/to/your/scoring_experiment.whl
```

5. Run the following command to run the included scoring pipeline example:

```
DRIVERLESS_AI_LICENSE_KEY="pastekeyhere"
     SCORING_PIPELINE_INSTALL_DEPENDENCIES=0 ./dai-env.sh /path/to/
     your/run_example.sh
```

16.4.3 Quick Start - Alternative Method

This section describes an alternative method for running the MLI Standalone Scoring Pipeline. This version requires Internet access. It is also not supported on Power machines.

16.4.4 Prerequisites

- Valid Driverless AI license.
- The scoring module and scoring service are supported only on Linux with Python 3.6 and OpenBLAS.
- The scoring module and scoring service download additional packages at install time and require internet access. Depending on your network environment, you might need to set up internet access via a proxy.
- Apache Thrift (to run the scoring service in TCP mode)

Installing Python 3.6

```
# Installing Python 3.11 on Ubuntu 16.10+:
$ sudo apt install python3.11 python3.11-dev python3-pip python3-dev \
    python-virtualenv python3-virtualenv
```

```
# Installing python3.11 on Ubuntu 16.04
$ sudo add-apt-repository ppa:deadsnakes/ppa
$ sudo apt-get update
$ sudo apt-get install python3.11 python3.11-dev python3-pip python3-dev \
    python-virtualenv python3-virtualenv
```

```
# Installing Conda 3.6
You can install Conda using either Anaconda or Miniconda. Refer to the links
     below for more information:
- Anaconda - https://docs.anaconda.com/anaconda/install.html
- Miniconda - https://conda.io/docs/user-guide/install/index.html
```

Installing the Thrift Compiler

Refer to Thrift documentation at https://thrift.apache.org/docs/BuildingFromSource for more information.

```
$ sudo apt-get install automake bison flex g++ git libevent-dev \
    libssl-dev libtool make pkg-config libboost-all-dev ant
$ wget https://github.com/apache/thrift/archive/0.10.0.tar.gz
$ tar -xvf 0.10.0.tar.gz
$ cd thrift-0.10.0
$ ./bootstrap.sh
$ ./configure
$ make
$ sudo make install
```

Run the following to refresh the runtime shared after installing Thrift.

```
$ sudo ldconfig /usr/local/lib
```

Running the MLI Scoring Pipeline - Alternative Method

Before running the quickstart examples, be sure that the MLI Scoring Package is already downloaded and unzipped.

1. On the MLI page, click the **Scoring Pipeline** button.
2. Unzip the scoring pipeline, and run the following examples in the **scoring-pipeline-mli** folder.

Run the scoring module example. (This requires Linux x86_64 and Python 3.6.)

```
$ bash run_example.sh
```

Run the TCP scoring server example. Use two terminal windows. (This requires Linux x86_64, Python 3.6 and Thrift.)

```
$ bash run_tcp_server.sh
$ bash run_tcp_client.sh
```

Run the HTTP scoring server example. Use two terminal windows. (This requires Linux x86_64, Python 3.6 and Thrift.)

```
$ bash run_http_server.sh
$ bash run_http_client.sh
```

16.4.5 MLI Python Scoring Module

The MLI scoring module is a Python module bundled into a standalone wheel file (name scoring_*.whl). All the prerequisites for the scoring module to work correctly are listed in the **requirements.txt** file. To use the scoring module, all you have to do is create a Python virtualenv, install the prerequisites, and then import and use the scoring module as follows:

```
----- See 'example.py' for complete example. -----
from scoring_487931_20170921174120_b4066 import Scorer
scorer = KLimeScorer()       # Create instance.
score = scorer.score_reason_codes([  # Call score_reason_codes()
    7.416,              # sepal_len
    3.562,              # sepal_wid
    1.049,              # petal_len
    2.388,              # petal_wid
])
```

The scorer instance provides the following methods:

- `score_reason_codes(list)`: Get KLime reason codes for one row (list of values).

- `score_reason_codes_batch(dataframe)`: Takes and outputs a Pandas Dataframe
- `get_column_names()`: Get the input column names
- `get_reason_code_column_names()`: Get the output column names

The process of importing and using the scoring module is demonstrated by the bash script `run_example.sh`, which effectively performs the following steps:

```
----- See 'run_example.sh' for complete example. -----
$ virtualenv -p python3.11 env
$ source env/bin/activate
$ pip install --use-deprecated=legacy-resolver -r requirements.txt
$ python example.py
```

16.4.6 K-LIME vs Shapley Reason Codes

There are times when the K-LIME model score is not close to the Driverless AI model score. In this case it may be better to use reason codes using the Shapley method on the Driverless AI model. **Note**: The reason codes from Shapley will be in the transformed feature space.

To see an example of using both K-LIME and Driverless AI Shapley reason codes in the same Python session, run:

```
$ bash run_example_shapley.sh
```

For this batch script to succeed, MLI must be run on a Driverless AI model. If you have run MLI in standalone (external model) mode, there will not be a Driverless AI scoring pipeline.

If MLI was run with transformed features, the Shapley example scripts will not be exported. You can generate exact reason codes directly from the Driverless AI model scoring pipeline.

16.4.7 MLI Scoring Service Overview

The MLI scoring service hosts the scoring module as a HTTP or TCP service. Doing this exposes all the functions of the scoring module through remote procedure calls (RPC).

In effect, this mechanism allows you to invoke scoring functions from languages other than Python on the same computer, or from another computer on a shared network or the internet.

The scoring service can be started in two ways:

- In TCP mode, the scoring service provides high-performance RPC calls via Apache Thrift (https://thrift.apache.org/) using a binary wire protocol.

- In HTTP mode, the scoring service provides JSON-RPC 2.0 calls served by Tornado (http://www.tornadoweb.org).

Scoring operations can be performed on individual rows (row-by-row) using `score` or in batch mode (multiple rows at a time) using `score_batch`. Both functions allow you to specify `pred_contribs=[True|False]` to get MLI predictions (KLime/Shapley) on a new dataset. See the **example_shapley.py** file for more information.

MLI Scoring Service - TCP Mode (Thrift)

The TCP mode allows you to use the scoring service from any language supported by Thrift, including C, C++, C#, Cocoa, D, Dart, Delphi, Go, Haxe, Java, Node.js, Lua, perl, PHP, Python, Ruby and Smalltalk.

To start the scoring service in TCP mode, you will need to generate the Thrift bindings once, then run the server:

```
----- See 'run_tcp_server.sh' for complete example. -----
$ thrift --gen py scoring.thrift
$ python tcp_server.py --port=9090
```

Note that the Thrift compiler is only required at build-time. It is not a run time dependency, i.e. once the scoring services are built and tested, you do not need to repeat this installation process on the machines where the scoring services are intended to be deployed.

To call the scoring service, simply generate the Thrift bindings for your language of choice, then make RPC calls via TCP sockets using Thrift's buffered transport in conjunction with its binary protocol.

```
----- See 'run_tcp_client.sh' for complete example. -----
$ thrift --gen py scoring.thrift

----- See 'example_client.py' for complete example. -----
socket = TSocket.TSocket('localhost', 9090)
transport = TTransport.TBufferedTransport(socket)
protocol = TBinaryProtocol.TBinaryProtocol(transport)
client = ScoringService.Client(protocol)
transport.open()
row = Row()
row.sepalLen = 7.416  # sepal_len
row.sepalWid = 3.562  # sepal_wid
row.petalLen = 1.049  # petal_len
row.petalWid = 2.388  # petal_wid
scores = client.score_reason_codes(row)
transport.close()
```

You can reproduce the exact same result from other languages, e.g. Java:

```
$ thrift --gen java scoring.thrift

// Dependencies:
```

```
// commons-codec-1.9.jar
// commons-logging-1.2.jar
// httpclient-4.4.1.jar
// httpcore-4.4.1.jar
// libthrift-0.10.0.jar
// slf4j-api-1.7.12.jar

import ai.h2o.scoring.Row;
import ai.h2o.scoring.ScoringService;
import org.apache.thrift.TException;
import org.apache.thrift.protocol.TBinaryProtocol;
import org.apache.thrift.transport.TSocket;
import org.apache.thrift.transport.TTransport;
import java.util.List;

public class Main {
  public static void main(String[] args) {
    try {
      TTransport transport = new TSocket("localhost", 9090);
      transport.open();

      ScoringService.Client client = new ScoringService.Client(
        new TBinaryProtocol(transport));

      Row row = new Row(7.642, 3.436, 6.721, 1.020);
      List<Double> scores = client.score_reason_codes(row);
      System.out.println(scores);

      transport.close();
    } catch (TException ex) {
      ex.printStackTrace();
    }
  }
}
```

Scoring Service - HTTP Mode (JSON-RPC 2.0)

The HTTP mode allows you to use the scoring service using plaintext JSON-RPC calls. This is usually less performant compared to Thrift, but has the advantage of being usable from any HTTP client library in your language of choice, without any dependency on Thrift.

For JSON-RPC documentation, see http://www.jsonrpc.org/specification .

To start the scoring service in HTTP mode:

```
----- See 'run_http_server.sh' for complete example. -----
$ python http_server.py --port=9090
```

To invoke scoring methods, compose a JSON-RPC message and make a HTTP POST request to http://host:port/rpc as follows:

```
----- See 'run_http_client.sh' for complete example. -----
$ curl http://localhost:9090/rpc \
  --header "Content-Type: application/json" \
  --data @- <<EOF
 {
  "id": 1,
  "method": "score_reason_codes",
```

```
    "params": {
      "row": [ 7.486, 3.277, 4.755, 2.354 ]
    }
  }
EOF
```

Similarly, you can use any HTTP client library to reproduce the above result. For example, from Python, you can use the requests module as follows:

```
import requests
row = [7.486, 3.277, 4.755, 2.354]
req = dict(id=1, method='score_reason_codes', params=dict(row=row))
res = requests.post('http://localhost:9090/rpc', data=req)
print(res.json()['result'])
```

16.5 Driverless AI MOJO Scoring Pipeline

Note: MOJOs are currently not available for TensorFlow or RuleFit models.

For completed experiments, Driverless AI automatically converts models to MOJOs (Model Objects, Optimized), which can be deployed for scoring in real time. A MOJO is a scoring engine that can be deployed in any Java environment for scoring in real time. The MOJO Scoring Pipeline is available as either a pure Java solution or a C++-based solution.

Keep in mind that, similar to H2O-3, MOJOs are tied to experiments. Experiments and MOJOs are not automatically upgraded when Driverless AI is upgraded.

16.5.1 Prerequisites

The following are required in order to run the MOJO scoring pipeline.

- Java 7 runtime (JDK 1.7) or newer. **NOTE**: We recommend using Java 11+ due to a bug in Java. (See https://bugs.openjdk.java.net/browse/JDK-8186464.)
- Valid Driverless AI license. You can download the `license.sig` file from the machine hosting Driverless AI (usually in the **license** folder). Copy the license file into the downloaded **mojo-pipeline** folder.
- mojo2-runtime.jar file. This is available from the top navigation menu in the Driverless AI UI and in the downloaded mojo-pipeline.zip file for an experiment.

License Specification

Driverless AI requires a license to be specified in order to run the MOJO Scoring Pipeline. The license can be specified in one of the following ways:

- Via an environment variable:

  - `DRIVERLESS_AI_LICENSE_FILE`: Path to the Driverless AI license file, or
  - `DRIVERLESS_AI_LICENSE_KEY`: The Driverless AI license key (Base64 encoded string)
- Via a system property of JVM (`-D` option):
  - `ai.h2o.mojos.runtime.license.file`: Path to the Driverless AI license file, or
  - `ai.h2o.mojos.runtime.license.key`: The Driverless AI license key (Base64 encoded string)
- Via an application classpath:
  - The license is loaded from a resource called `/license.sig`.
  - The default resource name can be changed via the JVM system property `ai.h2o.mojos.runtime.license.filename`.

For example:

```
# Specify the license via a temporary environment variable
export DRIVERLESS_AI_LICENSE_FILE="path/to/license.sig"
```

16.5.2 MOJO Scoring Pipeline Files

The **mojo-pipeline** folder includes the following files:

- **run_example.sh**: A bash script to score a sample test set.
- **pipeline.mojo**: Standalone scoring pipeline in MOJO format.
- **mojo2-runtime.jar**: MOJO Java runtime.
- **example.csv**: Sample test set (synthetic, in the correct format).
- **DOT** files: Text files that can be rendered as graphs that provide a visual representation of the MOJO scoring pipeline (can be edited to change the appearance and structure of a rendered graph).
- **PNG** files: Image files that provide a visual representation of the MOJO scoring pipeline.

16.5.3 Quickstart

Before running the quickstart examples, be sure that the MOJO scoring pipeline is already downloaded and unzipped:

1. On the completed Experiment page, click on the **Download Scoring Pipeline** button to download the **scorer.zip** file for this experiment onto your local machine.

   Note: This button is **Build MOJO Scoring Pipeline** if the MOJO Scoring Pipeline is disabled.

2. In the pop-up menu that appears, click on the **Download MOJO Scoring Pipeline** button once again to download the scorer.zip file for this experiment onto your local machine. Refer to the provided instructions for Java, Python, or R.

3. Run the following to score all rows in the sample test set with the file paths to the test set (example.csv), MOJO pipeline (pipeline.mojo) and license (license.sig) stored in environment variables `TEST_SET_FILE`, `MOJO_PIPELINE_FILE`, `DRIVERLESS_AI_LICENSE_KEY`:

```
$ bash run_example.sh
```

4. Run the following to score a specific test set (example.csv) with MOJO pipeline (pipeline.mojo) and the license file (license.sig):

```
$ bash run_example.sh pipeline.mojo example.csv license.sig
```

5. To run Java application for data transformation directly:

```
$ java -Dai.h2o.mojos.runtime.license.file=license.sig -cp mojo2-
    runtime.jar ai.h2o.mojos.ExecuteMojo pipeline.mojo example.csv
```

16.5.4 Execute the MOJO from Java

1. Open a new terminal window, create an experiment folder, and change directories to that new folder:

```
$ mkdir experiment && cd experiment
```

2. Create your main program in the **experiment** folder by creating a new file called **Main.java** (for example, using `vim Main.java`). Include the following contents.

```
    import ai.h2o.mojos.runtime.MojoPipeline;
    import ai.h2o.mojos.runtime.frame.MojoFrame;
    import ai.h2o.mojos.runtime.frame.MojoFrameBuilder;
    import ai.h2o.mojos.runtime.frame.MojoRowBuilder;
    import ai.h2o.mojos.runtime.lic.LicenseException;
    import ai.h2o.mojos.runtime.utils.CsvWritingBatchHandler;
```

```
import com.opencsv.CSVWriter;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.Writer;

public class DocsExample {
    public static void main(String[] args) throws IOException,
         LicenseException {
        // Load model and csv
        final MojoPipeline model = MojoPipeline.loadFrom("pipeline.
             mojo");

        // Get and fill the input columns
        final MojoFrameBuilder frameBuilder = model.
             getInputFrameBuilder();
        final MojoRowBuilder rowBuilder = frameBuilder.
             getMojoRowBuilder();
        rowBuilder.setValue("AGE", "68");
        rowBuilder.setValue("RACE", "2");
        rowBuilder.setValue("DCAPS", "2");
        rowBuilder.setValue("VOL", "0");
        rowBuilder.setValue("GLEASON", "6");
        frameBuilder.addRow(rowBuilder);

        // Create a frame which can be transformed by MOJO pipeline
        final MojoFrame iframe = frameBuilder.toMojoFrame();

        // Transform input frame by MOJO pipeline
        final MojoFrame oframe = model.transform(iframe);
        // `MojoFrame.debug()` can be used to view the contents of
             a Frame
        // oframe.debug();

        // Output prediction as CSV
        final Writer writer = new BufferedWriter(new
             OutputStreamWriter(System.out));
        final CSVWriter csvWriter = new CSVWriter(writer);
        CsvWritingBatchHandler.csvWriteFrame(csvWriter, oframe,
             true);
    }
}
```

3. Compile the source code with the files of the MOJO runtime (mojo2-runtime.jar) and MOJO pipeline (pipeline.mojo) copied into the experiment:

```
$ javac -cp mojo2-runtime.jar -J-Xms2g -J-XX:MaxPermSize=128m Main.java
```

4. Run the MOJO example with the license (license.sig) copied into the experiment:

```
# Linux and OS X users
$ java -Dai.h2o.mojos.runtime.license.file=license.sig -cp .:mojo2-
     runtime.jar Main
# Windows users
$ java -Dai.h2o.mojos.runtime.license.file=license.sig -cp .;mojo2-
     runtime.jar Main
```

5. The following output is displayed:

```
CAPSULE.True
0.5442205910902282
```

16.5.5 MOJO Scoring Pipeline - C++ Solution

The C++ Scoring Pipeline is provided as R and Python packages for the protobuf-based MOJO2 protocol. The packages are self contained, so no additional software is required. Simply build the MOJO Scoring Pipeline and begin using your preferred method. To download the MOJO Scoring Pipeline onto your local machine, click the **Download MOJO Scoring Pipeline** button, then click the same button again in the pop-up menu that appears. Refer to the provided instructions for Java, Python, or R.

Notes:

- These scoring pipelines are currently not available for RuleFit models.
- The **Download MOJO Scoring Pipeline** button appears as **Build MOJO Scoring Pipeline** if the MOJO Scoring Pipeline is disabled.

16.5.5.1 Downloading the Scoring Pipeline Runtimes

The R and Python packages can be downloaded from within the Driverless AI application. To do this, click **Resources**, then click **MOJO2 R Runtime** and **MOJO2 Py Runtime** from the drop-down menu. In the pop-up menu that appears, click the button that corresponds to the OS you are using. Choose from Linux, Mac OS X, and IBM PowerPC.

Examples

The following examples show how to use the R and Python APIs of the C++ MOJO runtime.

R Example

Prerequisites

- Linux OS (x86 or PPC) or Mac OS X (10.9 or newer)
- Driverless AI License (either file or environment variable)
- Rcpp ($\geq$1.0.0)
- data.table

```
# Install the R MOJO runtime using one of the methods below

# Install the R MOJO runtime on PPC Linux
install.packages("./daimojo_2.2.0_ppc64le-redhat-linux-gnu.tar.gz")

# Install the R MOJO runtime on x86 Linux
```

```
install.packages("./daimojo_2.2.0_x86_64-redhat-linux-gnu.tar.gz")

#Install the R MOJO runtime on Mac OS X
install.packages("./daimojo_2.2.0_x86_64-darwin.tar.gz")


# Load the MOJO
{.r}
library(daimojo)
m <- load.mojo("../data/dai/pipeline.mojo")
create.time(m)
## [1] "2019-11-18 22:00:24 UTC"
uuid(m)
## [1] "65875c15-943a-4bc0-a162-b8984fe8e50d"

# Load data and make predictions
{.r}
library(data.table)
col_class <- setNames(feature.types(m), feature.names(m))
d <- fread("../data/dai/example.csv", colClasses=col_class)
predict(m, d)
##       label.B    label.M
## 1  0.08287659 0.91712341
## 2  0.77655075 0.22344925
## 3  0.58438434 0.41561566
## 4  0.10570505 0.89429495
## 5  0.01685609 0.98314391
## 6  0.23656610 0.76343390
## 7  0.17410333 0.82589667
## 8  0.10157948 0.89842052
## 9  0.13546191 0.86453809
## 10 0.94778244 0.05221756
```

Python Example

Prerequisites

- Python 3.6
- Linux OS (x86 or PPC) or Mac OS X (10.9 or newer)
- Driverless AI License (either file or environment variable)
- datatable. Run the following to install:

```
# Install on Linux PPC, Linux x86, or Mac OS X
pip install datatable
```

- - Python MOJO runtime. Run one of the following commands after downloading from the GUI:

```
# Install the MOJO runtime on Linux PPC
pip install daimojo-2.2.0-cp311-cp311-linux_ppc64le.whl

# Install the MOJO runtime on Linux x86
pip install daimojo-2.2.0-cp311-cp311-linux_x86_64.whl

# Install the MOJO runtime on Mac OS X
pip install daimojo-2.2.0-cp311-cp311-macosx_10_7_x86_64.whl
```

```
# import the daimojo model package
import daimojo.model

# specify the location of the MOJO
m = daimojo.model("../data/dai/pipeline.mojo")

# retrieve the full path of the MOJO
m.modelfile
'/home/<user>/Desktop/mojo2/data/dai/pipeline.mojo'

# retrieve the UUID
m.uuid

# verify the MOJO version
m.mojo_version
'2.19.3'

# view the creation time
m.created_time
'Mon May 6 14:00:24 2019'

# retrive a list of missing values
m.missing_values
['',
 '?',
 'None',
 'nan',
 'NA',
 'N/A',
 'unknown',
 'inf',
 '-inf',
 '1.7976931348623157e+308',
 '-1.7976931348623157e+308']

# retrieve the feature names
m.feature_names
['clump_thickness',
 'uniformity_cell_size',
 'uniformity_cell_shape',
 'marginal_adhesion',
 'single_epithelial_cell_size',
 'bare_nuclei',
 'bland_chromatin',
 'normal_nucleoli',
 'mitoses']

# retrieve the feature types
m.feature_types
['float32',
 'float32',
 'float32',
 'float32',
 'float32',
 'float32',
 'float32',
 'float32',
 'float32']

# retrieve the output names
m.output_names
['label.B', 'label.M']

# retrieve the output types
```

```
['float64', 'float64']


# import the datatable module
import datatable as dt

# parse the example.csv file
pydt = dt.fread("../data/dai/example.csv", na_strings=m.missing_values)

# retrieve the column types
pydt.stypes
(stype.float64,
 stype.float64,
 stype.float64,
 stype.float64,
 stype.float64,
 stype.float64,
 stype.float64,
 stype.float64,
 stype.float64)

# make predictions on the example.csv file
res = m.predict(pydt)

# retrieve the predictions
res
          label.B.   label.M
0       0.0828766       0.917123
1       0.776551        0.223449
2       0.584384        0.415616
3       0.105705        0.894295
4       0.0168561       0.983144
5       0.236566        0.763434
6       0.174103        0.825897
7       0.101579        0.898421
8       0.135462        0.864538
9       0.947782        0.0522176

[10 rows by 2 columns]

# retrieve the prediction column names
res.names
#     ('label.B', 'label.M')

# retrieve the prediction column types
res.stypes
(stype.float64, stype.float64)

# convert datatable results to common data types
# res.to_pandas()  # need pandas
# res.to_numpy()   # need numpy
res.to_list()
```

17 Deployment

Driverless AI can deploy the MOJO scoring pipeline for you to test and/or to integrate into a final product.

Notes:

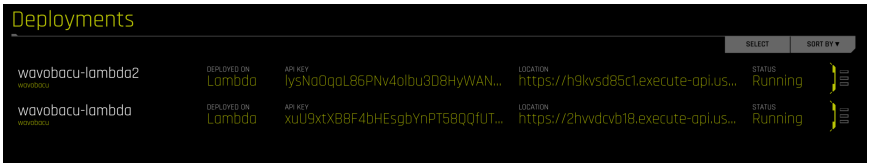
- This section describes how to deploy a MOJO scoring pipeline and assumes that a MOJO scoring pipeline exists.
- This is an early feature that will continue to support additional deployments.

17.1 Additional Resources

Refer to the https://github.com/h2oai/dai-deployment-templates repository to see different deployment templates for Driverless AI scorers, including AWS Lambda, KDB, and Local REST.

17.2 Deployments Overview Page

All of the MOJO scoring pipeline deployments are available in the Deployments Overview page, which is available from the top menu. This page lists all active deployments and the information needed to access the respective endpoints. In addition, it allows you to stop any deployments that are no longer needed.



17.3 AWS Lambda Deployment

17.3.1 Driverless AI Prerequisites

To deploy a MOJO scoring pipeline as an AWS lambda function, the MOJO pipeline archive has to be created first by choosing the **Build MOJO Scoring Pipeline** option on the completed experiment page.

17.3.2 AWS Access Permissions Prerequisites

The following AWS access permissions need to be provided to the role in order for Driverless AI Lambda deployment to succeed.

- AWSLambdaFullAccess
- IAMFullAccess
- AmazonAPIGatewayAdministrator

The policy can be further stripped down to restrict Lambda and S3 rights using the JSON policy definition as follows:

```
{
      "Version": "2012-10-17",
      "Statement": [
          {
              "Sid": "VisualEditor0",
              "Effect": "Allow",
              "Action": [
                  "iam:GetPolicyVersion",
                  "iam:DeletePolicy",
                  "iam:CreateRole",
                  "iam:AttachRolePolicy",
                  "iam:ListInstanceProfilesForRole",
                  "iam:PassRole",
                  "iam:DetachRolePolicy",
                  "iam:ListAttachedRolePolicies",
                  "iam:GetRole",
                  "iam:GetPolicy",
                  "iam:DeleteRole",
                  "iam:CreatePolicy",
                  "iam:ListPolicyVersions"
              ],
              "Resource": [
                  "arn:aws:iam::*:role/h2oai*",
                  "arn:aws:iam::*:policy/h2oai*"
              ]
          },
          {
              "Sid": "VisualEditor1",
              "Effect": "Allow",
              "Action": "apigateway:*",
              "Resource": "*"
          },
          {
              "Sid": "VisualEditor2",
              "Effect": "Allow",
              "Action": [
                  "lambda:CreateFunction",
                  "lambda:ListFunctions",
                  "lambda:InvokeFunction",
                  "lambda:GetFunction",
                  "lambda:UpdateFunctionConfiguration",
                  "lambda:DeleteFunctionConcurrency",
                  "lambda:RemovePermission",
                  "lambda:UpdateFunctionCode",
                  "lambda:AddPermission",
                  "lambda:ListVersionsByFunction",
                  "lambda:GetFunctionConfiguration",
                  "lambda:DeleteFunction",
                  "lambda:PutFunctionConcurrency",
```

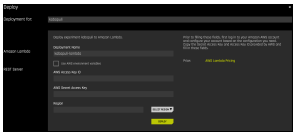
```
                "lambda:GetPolicy"
            ],
            "Resource": "arn:aws:lambda:*:*:function:h2oai*"
        },
        {
            "Sid": "VisualEditor3",
            "Effect": "Allow",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::h2oai*/*",
                "arn:aws:s3:::h2oai*"
            ]
        }
    ]
}
```

17.3.3 Deploying the Lambda

Once the MOJO pipeline archive is ready, Driverless AI provides a **Deploy** option on the completed experiment page.

Note: This button is not available on PPC64LE environments.

This option opens a new dialog for setting the AWS account credentials (or use those supplied in the Driverless AI configuration file or environment variables), AWS region, and the desired deployment name (which must be unique per Driverless AI user and AWS account used).



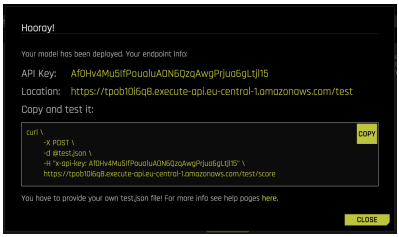
Amazon Lambda deployment parameters:

- **Deployment Name**: A unique name of the deployment. By default, Driverless AI offers a name based on the name of the experiment and the deployment type. This has to be unique both for Driverless AI user and the AWS account used.
- **Region**: The AWS region to deploy the MOJO scoring pipeline to. It makes sense to choose a region geographically close to any client code calling the endpoint in order to minimize request latency. (See also AWS Regions and Availability Zones.)
- **Use AWS environment variables**: If enabled, the AWS credentials are taken from the Driverless AI configuration file. This would usually be entered by the Driverless AI installation administrator.

- **AWS Access Key ID** and **AWS Secret Access Key**: Credentials to access the AWS account. This pair of secrets identifies the AWS user and the account and can be obtained from the AWS account console.

17.3.4 Testing the Lambda Deployment

On a successful deployment, all the information needed to access the new endpoint (URL and an API Key) is printed, and the same information is available in the **Deployments Overview Page** after clicking on the deployment row.



Note that the actual scoring endpoint is located at the path **/score**. In addition, to prevent DDoS and other malicious activities, the resulting AWS lambda is protected by an API Key, i.e., a secret that has to be passed in as a part of the request using the `x-api-key` HTTP header.

The request is a JSON object containing attributes:

- **fields**: A list of input column names that should correspond to the training data columns.
- **rows**: A list of rows that are in turn lists of cell values to predict the target values for.
- (optional) **includeFieldsInOutput**: A list of input columns that should be included in the output.

An example request providing 2 columns on the input and asking to get one column copied to the output looks as follows:

```
{
  "fields": [
    "age", "salary"
  ],
  "includeFieldsInOutput": [
    "salary"
  ],
  "rows": [
    [
      "48.0", "15000.0"
    ],
```

```
      [
        "35.0", "35000.0"
      ],
      [
        "18.0", "22000.0"
      ]
    ]
  }
```

Assuming the request is stored locally in a file named **test.json**, the request to the endpoint can be sent, e.g., using the `curl` utility, as follows:

```
$ URL={place the endpoint URL here}
$ API_KEY={place the endpoint API key here}
$ curl \
    -X POST \
    -H "x-api-key: ${API_KEY}" \
    -d @test.json ${URL}/score
```

The response is a JSON object with a single attribute "score", which contains the list of rows with the optional copied input values and the predictions.

For the example above with a two class target field, the result is likely to look something like the following snippet. The particular values would of course depend on the scoring pipeline:

```
  {
    "score": [
      [
        "48.0",
        "0.6240277982943945",
        "0.045458571508101536",
      ],
      [
        "35.0",
        "0.7209441819603676",
        "0.06299909138586585",
      ],
      [
        "18.0",
        "0.7209441819603676",
        "0.06299909138586585",
      ]
    ]
  }
```

17.3.5 AWS Deployment Issues

We create a new S3 bucket per AWS Lambda deployment. The bucket names have to be unique throughout AWS S3, and one user can create a maximum of 100 buckets. Therefore, we recommend setting the bucket name used for deployment with the `deployment_aws_bucket_name` config option.

17.4 REST Server Deployment

This section describes how to deploy the trained MOJO scoring pipeline as a local Representational State Transfer (REST) Server.

17.4.1 Prerequisites

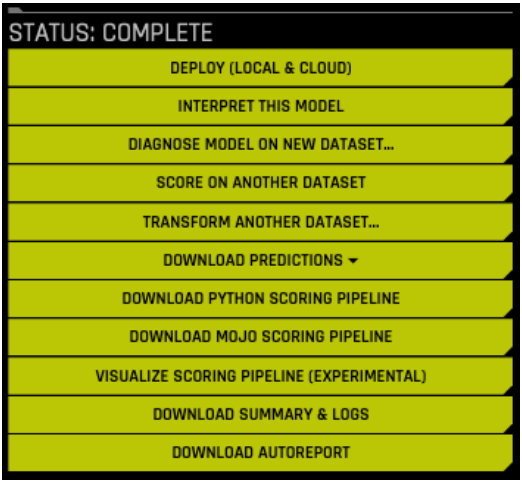
- Driverless AI MOJO Scoring Pipeline: To deploy a MOJO scoring pipeline as a Local REST Scorer, the MOJO pipeline archive has to be created first by choosing the Build MOJO Scoring Pipeline option on the completed experiment page.
- When using a firewall or a virtual private cloud (VPC), the ports that are used by the REST server must be exposed.
- Ensure that you have enough memory and CPUs to run the REST scorer. Typically, a good estimation for the amount of required memory is 12 times the size of the pipeline.mojo file. For example, a 100MB pipeline.mojo file will require approximately 1200MB of RAM. (**Note**: To conveniently view in-depth information about your system in Driverless AI, click on **Resources** at the top of the sceen, then click **System Info**.)
- When running Driverless AI in a Docker container, you must expose ports on Docker for the REST service deployment within the Driverless AI Docker container. For example, the following exposes the Driverless AI Docker container to listen to port 8094 for requests arriving at the host port at 18094.

```
docker run \
    -d \
    --pid=host \
    --init \
    --rm \
    --shm-size=256m \
    -u `id -u`:`id -g` \
    -p 12181:12345 \
    -p 18094:8094 \
    -v `pwd`/data:/data \
    -v `pwd`/log:/log \
    -v `pwd`/license:/license \
    -v `pwd`/tmp:/tmp \
    h2oai/<dai-image-name>:TAG
```

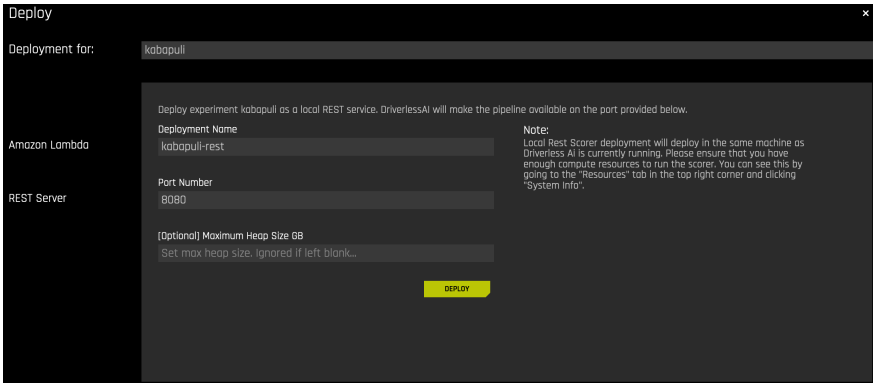
17.4.2 Deploying on REST Server

Once the MOJO pipeline archive is ready, Driverless AI provides a **Deploy (Local & Cloud)** option on the completed experiment page.

Notes: This button is only available after the MOJO Scoring Pipeline has been built.



This option opens a new dialog for setting the REST Server deployment name, port number, and maximum heap size (optional).



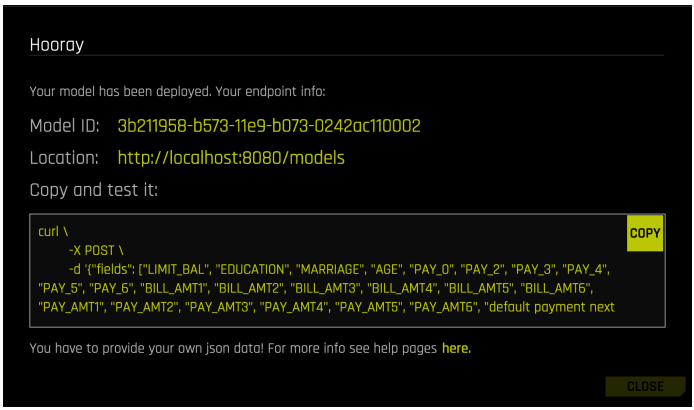
1. Specify a name for the REST scorer in order to help track the deployed REST scorers.

2. Provide a port number on which the REST scorer will run. For example, if port number 8081 is selected, the scorer will be available at http://my-ip-address:8081/models

3. Optionally specify the maximum heap size for the Java Virtual Machine (JVM) running the REST scorer. This can help constrain the REST scorer from overconsuming memory of the machine. Because the REST scorer is running on the same machine as Driverless AI, it may be helpful to limit the amount of memory that is allocated to the REST scorer. This

option will limit the amount of memory the REST scorer can use, but it will also produce an error if the memory allocated is not enough to run the scorer. (The amount of memory required is mostly dependent on the size of MOJO. See **Prerequisites** for more information.)

17.4.3 Testing the REST Server Deployment



The request is a JSON object containing attributes:

- **fields**: A list of input column names that should correspond to the training data columns.
- **rows**: A list of rows that are in turn lists of cell values to predict the target values for.
- (optional) **includeFieldsInOutput**: A list of input columns that should be included in the output.

An example request providing 2 columns on the input and asking to get one column copied to the output looks as follows:

```
{
"fields": [
  "age", "salary"
],
"includeFieldsInOutput": [
  "salary"
],
"rows": [
  [
    "48.0", "15000.0"
  ],
  [
    "35.0", "35000.0"
  ],
  [
```

```
      "18.0", "22000.0"
    ]
  ]
}
```

Assuming the request is stored locally in a file named `test.json`, the request to the endpoint can be sent, e.g., using the `curl` utility, as follows:

```
URL={place the endpoint URL here}
curl \
  -X POST \
  -d {"fields": ['age', 'salary', 'education'], "rows": [1, 2, 3], "
       includeFieldsInOutput": ["education"]}\
  -H "Content-Type: application/json" \
  ${URL}/score
```

The response is a JSON object with a single attribute `score`, which contains the list of rows with the optional copied input values and the predictions.

For the example above with a two class target field, the result is likely to look something like the following snippet (the exact values depend on the scoring pipeline):

```
 {
  "score": [
    [
      "48.0",
      "0.6240277982943945",
      "0.045458571508101536",
    ],
    [
      "35.0",
      "0.7209441819603676",
      "0.06299909138586585",
    ],
    [
      "18.0",
      "0.7209441819603676",
      "0.06299909138586585",
    ]
  ]
}
```

17.4.4 REST Server Deployment Issues

- Local REST server deployments are useful for determining the behavioral characteristics of a MOJO that is intended for deployment. However, we don't recommend using the REST Server deployment as a production level scoring service. The REST Server deployment runs in the same machine as the core of Driverless AI, and therefore has to share system resources with all other Driverless AI processes. This can lead to unexpected scenarios in which competition for compute resources causes the REST Server to fail. Additionally, given that the memory and CPU resource

requirements for the REST scorer and for scoring the MOJO are typically smaller than the requirements for training, it is typically much more cost effective to deploy the MOJO in a smaller instance/machine. Templates can be easily found/built from the repository: https://github.com/h2oai/dai-deployment-templates.

- The REST Server will be shut down if Driverless AI is restarted.

18 About Driverless AI Transformations

Transformations in Driverless AI are applied to columns in the data. The transformers create the engineered features in experiments.

Driverless AI provides a number of transformers. The downloaded experiment logs include the transformations that were applied to your experiment. Note that you can exclude transformations in the **config.toml** file, and that list of excluded transformers will also be available in the experiment log.

The following transformers are available for classification (multiclass and binary) and regression experiments.

18.1 Numeric Transformers

- **ClusterDist Transformer**: The Cluster Distance Transformer clusters selected numeric columns and uses the distance to a specific cluster as a new feature.
- **ClusterTE Transformer**: The Cluster Target Encoding Transformer clusters selected numeric columns and calculates the mean of the response column for each cluster. The mean of the response is used as a new feature. Cross Validation is used to calculate mean response to prevent overfitting.
- **Interactions Transformer**: The Interactions Transformer adds, divides, multiplies, and subtracts two numeric columns in the data to create a new feature. This transformation uses a smart search to identify which feature pairs to transform. Only interactions that improve the baseline model score are kept.
- **InteractionsSimple Transformer**: The InteractionsSimple Transformer adds, divides, multiplies, and subtracts two numeric columns in the data to create a new feature. This transformation randomly selects pairs of features to transform.
- **NumCatTE Transformer**: The Numeric Categorical Target Encoding Transformer calculates the mean of the response column for several selected columns. If one of the selected columns is numeric, it is first

converted to categorical by binning. The mean of the response column is used as a new feature. Cross Validation is used to calculate mean response to prevent overfitting.

- **NumToCatTE Transformer**: The Numeric to Categorical Target Encoding Transformer converts numeric columns to categoricals by binning and then calculates the mean of the response column for each group. The mean of the response for the bin is used as a new feature. Cross Validation is used to calculate mean response to prevent overfitting.
- **NumToCatWoEMonotonic Transformer**: The Numeric to Categorical Weight of Evidence Transformer converts a numeric column to categorical by binning and then calculates Weight of Evidence for each bin. The Weight of Evidence is used as a new feature. Weight of Evidence measures the "strength" of a grouping for separating good and bad risk and is calculated by taking the log of the ratio of distributions for a binary response column.
- **NumToCatWoE Transformer**: The Numeric to Categorical Weight of Evidence Transformer converts a numeric column to categorical by binning and then calculates Weight of Evidence for each bin. The Weight of Evidence is used as a new feature. Weight of Evidence measures the "strength" of a grouping for separating good and bad risk and is calculated by taking the log of the ratio of distributions for a binary response column.
- **Original Transformer**: The Original Transformer applies an identity transformation to a numeric column.
- **TruncSVDNum Transformer**: Truncated SVD Transformer trains a Truncated SVD model on selected numeric columns and uses the components of the truncated SVD matrix as new features.

18.2 Time Series Experiments Transformers

- **DateOriginal Transformer**: The Date Original Transformer retrieves date values such as year, quarter, month, day, day of the year, week, and weekday values.
- **DateTimeOriginal Transformer**: The Date Original Transformer retrieves date values such as year, quarter, month, day, day of the year, week, and weekday values.
- **EwmaLags Transformer**: The Exponentially Weighted Moving Average (EWMA) Transformer calculates the exponentially weighted moving average of target or feature lags.
- **LagsAggregates Transformer**: The Lags Aggregates Transformer calculates aggregations of target/feature lags like mean(lag7, lag14, lag21)

with support for mean, min, max, median, sum, skew, kurtosis, std. The aggregation is used as a new feature.

- **LagsInteraction Transformer**: The Lags Interaction Transformer creates target/feature lags and calculates interactions between the lags (lag2 - lag1, for instance). The interaction is used as a new feature.
- **Lags Transformer**: The Lags Transformer creates target/feature lags, possibly over groups. Each lag is used as a new feature. Lag transformers may apply to categorical (strings) features or binary/multiclass string valued targets after they have been internally numerically encoded.
- **LinearLagsRegression Transformer**: The Linear Lags Regression transformer trains a linear model on the target or feature lags to predict the current target or feature value. The linear model prediction is used as a new feature.

18.3 Categorical Transformers (String)

- **Cat Transformer**: The Cat Transformer sorts a categorical column in lexicographical order and uses the order index created as a new feature.
- **CatOriginal Transformer**: The Categorical Original Transformer applies an identity transformation that leaves categorical features as they are. This transformer works with models that can handle non-numeric feature values.
- **CVCatNumEncode Transformer**: The Cross Validation Categorical to Numeric Encoding Transformer calculates an aggregation of a numeric column for each value in a categorical column (ex: calculate the mean Temperature for each City) and uses this aggregation as a new feature.
- **CVTargetEncode Transformer**: The Cross Validation Target Encoding Transformer calculates the mean of the response column for each value in a categorical column and uses this as a new feature. Cross Validation is used to calculate mean response to prevent overfitting.
- **Frequent Transformer**: The Frequent Transformer calculates the frequency for each value in categorical column(s) and uses this as a new feature. This count can be either the raw count or the normalized count.
- **LexiLabelEncoder Transformer**: The Lexi Label Encoder sorts a categorical column in lexicographical order and uses the order index created as a new feature.
- **NumCatTE Transformer**: The Numeric Categorical Target Encoding Transformer calculates the mean of the response column for several selected columns. If one of the selected columns is numeric, it is first

converted to categorical by binning. The mean of the response column is used as a new feature. Cross Validation is used to calculate mean response to prevent overfitting.

- **OneHotEncoding Transformer**: The One-hot Encoding transformer converts a categorical column to a series of boolean features by performing one-hot encoding. The boolean features are used as new features.
- **SortedLE Transformer**: The Sorted Label Encoding Transformer sorts a categorical column by the response column and uses the order index created as a new feature.
- **WeightOfEvidence Transformer**: The Weight of Evidence Transformer calculates Weight of Evidence for each value in categorical column(s). The Weight of Evidence is used as a new feature. Weight of Evidence measures the "strength" of a grouping for separating good and bad risk and is calculated by taking the log of the ratio of distributions for a binary response column.

$$\text{WOE} = \ln\left(\frac{\text{Distribution of Goods}}{\text{Distribution of Bads}}\right)$$

This only works with a binary target variable. The likelihood needs to be created within a stratified kfold if a fit_transform method is used. More information can be found here: http://ucanalytics.com/blogs/information-value-and-weight-of-evidencebanking-case/.

18.4 Text Transformers (String)

- **TextBiGRU Transformer**: The Text Bidirectional GRU Transformer trains a bi-directional GRU TensorFlow model on word embeddings created from a text feature to predict the response column. The GRU prediction is used as a new feature. Cross Validation is used when training the GRU model to prevent overfitting.
- **TextCharCNN Transformer**: The Text Character CNN Transformer trains a CNN TensorFlow model on character embeddings created from a text feature to predict the response column. The CNN prediction is used as a new feature. Cross Validation is used when training the CNN model to prevent overfitting.
- **TextCNN Transformer**: The Text CNN Transformer trains a CNN TensorFlow model on word embeddings created from a text feature to predict the response column. The CNN prediction is used as a new a

feature. Cross Validation is used when training the CNN model to prevent overfitting.

- **TextLinModel Transformer**: The Text Linear Model Transformer trains a linear model on a TF-IDF matrix created from a text feature to predict the response column. The linear model prediction is used as a new feature. Cross Validation is used when training the linear model to prevent overfitting.
- **Text Transformer**: The Text Transformer tokenizes a text column and creates a TFIDF matrix (term frequency-inverse document frequency) or count (count of the word) matrix. This may be followed by dimensionality reduction using truncated SVD. Selected components of the TF-IDF/Count matrix are used as new features.

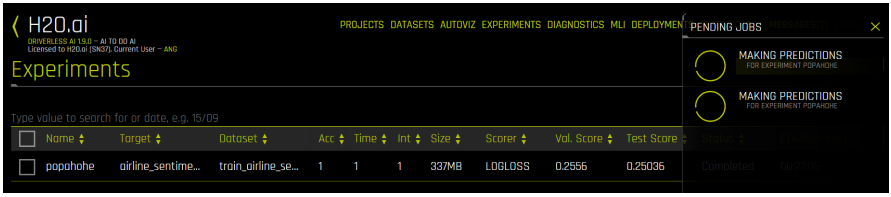
18.5 Time Transformers (Date, Time)

- **Dates Transformer**: The Dates Transformer retrieves any date values, including year, quarter, month, day, day of the year, week, weekday, hour, minute, and second values.
- **IsHoliday Transformer**: The Is Holiday Transformer determines if a date column is a holiday. A boolean column indicating if the date is a holiday is added as a new feature. Creates a separate feature for holidays in the United States, United Kingdom, Germany, Mexico, and the European Central Bank. Other countries available in the python Holiday package can be added via the configuration file.

19 Monitoring and Logging

19.1 Monitoring Pending Jobs

Driverless AI features a **Pending Jobs** panel that allows you to monitor the progress of various long-running jobs that can be started from the **Completed Experiment** page. To view this panel, click the group of square icons located in the upper-right corner. The circular icon next to the description of a pending job indicates its status.



The following jobs are monitored in this panel:

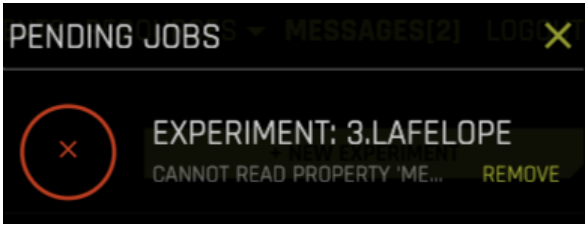
- Create AutoDoc
- Create MOJO Scoring Pipeline
- Create Python Scoring Pipeline
- Create Test Set Predictions
- Create Training Predictions
- Score Model
- Transform Data

Navigate to a completed job by clicking the Open icon. You can also clear a completed job from the panel by clicking **Remove** or cancel an ongoing job by clicking **Abort**.

Note: Certain jobs cannot be cancelled.

19.1.1 Failed Jobs

If a job fails, you can review the logs associated with that job type to determine what caused the failure. Refer to the Driverless AI Logs section for more information.

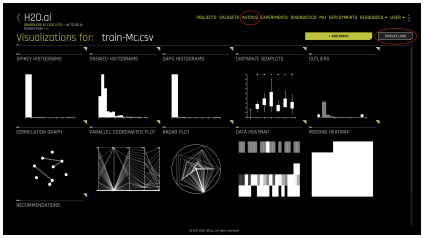


19.2 Logs

Driverless AI provides a number of logs that can be retrieved while visualizing datasets, while an experiment is running, and after an experiment is completed.

While Visualizing Datasets

When running Autovisualization, you can access the Autoviz logs by clicking the **Display Logs** button on the Visualize Datasets page.



This page presents logs created while the dataset visualization was being performed. You can download the **vis-data-server.log** file by clicking the **Download Logs** button on this page. This file can be used to troubleshoot any issues encountered during dataset visualization.

While an Experiment is Running

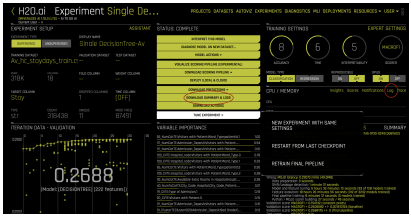
While the experiment is running, you can access the logs by clicking on the **Log** button on the experiment screen. The **Log** button can be found in the CPU/Memory section. Clicking on the **Log** button will present the experiment logs in real time. You can download these logs by clicking on the **Download Logs** button in the upper right corner.

Only the **h2oai_experiment.log** will be downloaded while the experiment is running (for example: **h2oai_experiment_tobosoru.log**). It will have the same information as the logs being presented in real time on the screen.

For troubleshooting purposes, view the complete **h2oai_experiment.log** (or **h2oai_experiment_anonymized.log**). This will be available after the experiment finishes, as described in the next section.

After an Experiment has Finished

If the experiment has finished, you can download the logs by clicking on the **Download Summary and Logs** button at the center of the experiment screen.



This will download a zip file which includes the following logs:

- **h2oai_experiment.log**: This is the log corresponding to the experiment.

- **h2oai_experiment_anonymized.log**: This is the log corresponding to the experiment where all data in the log is anonymized.
- **h2oai_server.log**: Contains the logs for all experiments and all users.
- **h2oai_server_anonymized.log**: Contains the logs for all experiments and all users where all data in the log is anonymized.
- **h2o.log**: This is the log corresponding to H2O-3. (H2O-3 is used internally for parts of Driverless AI.)

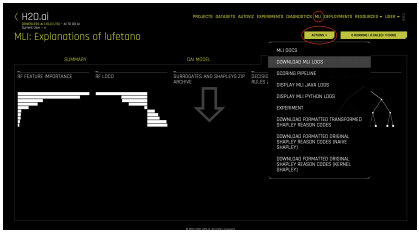
For troubleshooting purposes, view the complete **h2oai_experiment.log** or the **h2oai_experiment_anonymized.log**.

The following additional information about your particular experiment will also be included in the zip file:

- **tuning_leaderboard.txt**: The results of the parameter tuning stage. This contains the model parameters investigated and their performance.
- **gene_summary.txt**: A summary of the feature transformations available for each gene over the feature engineering iterations
- **features.txt**: The features used in the final Driverless AI model along with feature importance and feature description
- **details folder**: Contains standard streams for each of the subprocesses performed by Driverless A. This information is for debugging purposes

After Model Interpretation

You can view an MLI log for completed model interpretations by selecting the **Download MLI Logs** link on the MLI page.

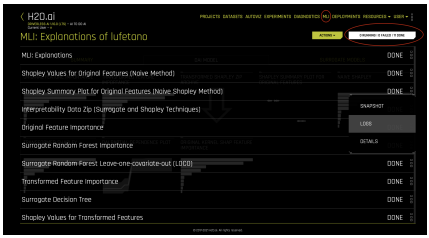


This will download a zip file which includes the following logs:

- h2oai_experiment_mli_key.log: This is the log corresponding to the model interpretation.
- h2oai_experiment_mli_key_anonymized.log: This is the log corresponding to the model interpretation where all data in the log is anonymized.

This file can be used to view logging information for successful interpretations. If MLI fails, then those logs are in **./tmp/h2oai_experiment_mli_key.log** and **./tmp/h2oai_experiment_mli_key_anonymized.log**.

The explainer or recipe logs are accessible from the task run button.



19.3 Sending Logs to H2O

This section describes the logs to send in the event of failures when running Driverless AI.

Dataset Failures

- **Adding Datasets**: If a dataset fails to import, a message on the screen should provide the reason for the failure. The logs to send are available in the Driverless AI **./tmp** folder.
- **Dataset Details**: If a failure occurs when attempting to view Dataset Details, the logs to send are available in the Driverless AI **./tmp** folder.
- **Autovisualization**: If a failure occurs when attempting to Visualize Datasets, a message on the screen should provide a reason for the failure. The logs to send are available in the Driverless AI **./tmp** folder.

Experiments

- **While Running an Experiment**: As indicated previously, a **Log** button is available on the Experiment page. Clicking on the **Log** button will present the experiment logs in real time. You can download these logs by clicking on the **Download Logs** button in the upper right corner. You can also retrieve the **h2oai_experiment.log** for the corresponding experiment in the Driverless AI **./tmp** folder.

MLI

- **During Model Interpretation**: If a failure occurs during model interpretation, then the logs to send are **./tmp/h2oai_experiment_mli_key.log** and **./tmp/h2oai_experiment_mli_key_anonymized.log**.

20 References

1. L. Breiman. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical Science*, 16(3), 2001. URL https://projecteuclid.org/euclid.ss/1009213726

2. M. W. Craven and J. W. Shavlik. Extracting tree-structured representations of trained networks. *Advances in Neural Information Processing Systems*, 1996. URL http://papers.nips.cc/paper/1152-extracting-tree-structured-representations-of-trained-networks.pdf

3. J. Friedman. **Predictive Learning via Rule Ensembles**, October 2005. URL http://statweb.stanford.edu/~jhf/ftp/RuleFit.pdf

4. J. Friedman, T. Hastie, and R. Tibshirani. **The Elements of Statistical Learning**. Springer, New York, 2001. URL https://web.stanford.edu/~hastie/ElemStatLearn/printings/ESLII_print12.pdf

5. A. Goldstein, A. Kapelner, J. Bleich, and E. Pitkin. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, 24(1), 2015

6. R. A. Groeneveld and G. Meeden. **Measuring Skewness and Kurtosis**. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 33 (4):391–399, December 1984

7. P. Hall, W. Phan, and S. S. Ambati. Ideas on interpreting machine learning. *O'Reilly Ideas*, 2017. URL https://www.oreilly.com/ideas/ideas-on-interpreting-machine-learning

8. J. Hartigan and S. Mohanty. **The RUNT test for Multimodality**. *Journal of Classification*, 9(1):63–70, January 1992

9. J. Lei, M. G'Sell, A. Rinaldo, R. J. Tibshirani, and L. Wasserman. Distribution-free predictive inference for regression. *Journal of the American Statistical Association just-accepted*, 2017. URL http://www.stat.cmu.edu/~ryantibs/papers/conformal.pdf

10. H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, S. Chikkerur, D. Liu, M. Wattenberg, A. M. Hrafnkelsson, T. Boulos, and J. Kubica. **Ad Click Prediction: A View from the Trenches**. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013.

URL https://research.google.com/pubs/archive/41159.pdf

11. F. Niu, B. Recht, C. Re, and S. J. Wright. **Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent**. In *NIPS*, 2011. URL https://people.eecs.berkeley.edu/~brecht/papers/hogwildTR.pdf

12. M. T. Ribeiro, S. Singh, and C. Guestrin. Why should I trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016. URL http://www.kdd.org/kdd2016/papers/files/rfp0573-ribeiroA.pdf

13. J. W. Tukey. **Exploratory Data Analysis**. Addison-Wesley, Reading, MA, 1977

14. L. Wilkinson. **Dot Plots**. *The American Statistician*, 53(3):276–281, 1999

15. L. Wilkinson, A. Anand, and R. Grossman. *"Graph-theoretic Scagnostics," in Proceedings of the IEEE Information Visualization*. INFOVIS '05. IEEE Computer Society, Washington, DC, USA, 2005

16. I. Yeo and R. Johnson. A new family of power transformations to improve normality or symmetry. *Biometrika*, 2000. URL https://www.jstor.org/stable/2673623?seq=1

17. H2O.ai Team. Datatable for python. URL https://github.com/h2oai/datatable

21 Authors

Patrick Hall

Patrick Hall is senior director for data science products at H2O.ai where he focuses mainly on model interpretability. Patrick is also currently an adjunct professor in the Department of Decision Sciences at George Washington University, where he teaches graduate classes in data mining and machine learning. Prior to joining H2O.ai, Patrick held global customer facing roles and research and development roles at SAS Institute.

Follow him on Twitter: @jpatrickhall

Megan Kurka

Megan is a customer data scientist at H2O.ai. Prior to working at H2O.ai, she worked as a data scientist building products driven by machine learning for B2B customers. Megan has experience working with customers across multiple

industries, identifying common problems, and designing robust and automated solutions.

Angela Bartz

Angela is the doc whisperer at H2O.ai. With extensive experience in technical communication, she brings our products to life by documenting the features and functionality of the entire suite of H2O products. Having worked for companies both large and small, she is an expert at understanding her audience and translating complex ideas into consumable documents. Angela has a BA degree in English from the University of Detroit Mercy.