# Contents

1 Overview ................................................. 6  
1.1 Citation ............................................. 6  
1.2 Have Questions? .................................... 6  

2 Why Driverless AI? ................................. 7  

3 Key Features ........................................ 8  

4 Supported Algorithms ......................... 9  

5 Installing and Upgrading Driverless AI ...... 10  

6 Launching Driverless AI ......................... 11  
6.1 Messages ......................................... 12  

7 The Datasets Page ................................. 13  
7.1 Adding Datasets ................................. 13  
7.2 Dataset Details ................................. 15  
7.2.1 Dataset Details Page ....................... 15  
7.2.2 Dataset Rows Page ......................... 16  
7.3 Splitting Datasets ............................. 17  
7.4 Visualizing Datasets ......................... 18  
7.4.1 The Visualization Page .................... 18  

8 Running an Experiment ......................... 22  
8.1 Before You Begin .............................. 22  
8.2 New Experiment ............................... 22  
8.3 Completed Experiment ....................... 26  
8.3.1 Experiment Summary ...................... 27  
8.4 Viewing Experiments ......................... 30  
8.4.1 Checkpointing, Rerunning, and Retraining ... 30  
8.4.2 Deleting Experiments ....................... 33  

9 Diagnosing a Model .............................. 33  

10 Interpreting a Model ......................... 35  
10.1 Interpret this Model button - Non-Time-Series ... 35  
10.2 Interpret this Model button - Time-Series ........ 36  
10.2.1 Multi-Group Time Series MLI ............... 36  
10.2.2 Single Time Series MLI .................... 38  
10.3 Model Interpretation - Driverless AI Models .... 40  
10.4 Model Interpretation - External Models ........ 42
10.5 Understanding the Model Interpretation Page .......................... 45
  10.5.1 Summary Page .............................................. 46
  10.5.2 DAI Model Dropdown ....................................... 46
  10.5.3 Surrogate Models Dropdown ................................. 48
  10.5.4 Random Forest Dropdown .................................. 52
  10.5.5 Dashboard Page ............................................. 56
10.6 General Considerations ............................................ 57
  10.6.1 Machine Learning and Approximate Explanations .......... 57
  10.6.2 The Multiplicity of Good Models in Machine Learning ... 57
  10.6.3 Expectations for Consistency Between Explanatory Tech-
          niques .......................................................... 58

11 Viewing Explanations .................................................. 59

12 Score on Another Dataset ............................................. 61

13 Transform Another Dataset ........................................... 61

14 The Python and MOJO Scoring Pipelines .............................. 62
  14.1 Which Pipeline Should I Use? .................................. 63
  14.2 Driverless AI Standalone Python Scoring Pipeline ............. 63
    14.2.1 Python Scoring Pipeline Files ........................... 64
    14.2.2 Quick Start - Recommended Method ....................... 64
    14.2.3 Quick Start - Alternative Method ......................... 65
    14.2.4 The Python Scoring Module ............................... 68
    14.2.5 The Scoring Service ..................................... 69
    14.2.6 Python Scoring Pipeline FAQ ............................ 71
    14.2.7 Troubleshooting Python Environment Issues ............... 72
  14.3 Driverless AI MLI Standalone Scoring Package .................. 73
    14.3.1 MLI Python Scoring Package Files ....................... 73
    14.3.2 Quick Start - Recommended Method ....................... 74
    14.3.3 Quick Start - Alternative Method ......................... 75
    14.3.4 Prerequisites ........................................... 75
    14.3.5 MLI Python Scoring Module ............................... 76
    14.3.6 K-LIME vs Shapley Reason Codes .......................... 77
    14.3.7 MLI Scoring Service Overview ............................ 77
  14.4 Driverless AI MOJO Scoring Pipeline ............................ 80
    14.4.1 Prerequisites ........................................... 80
    14.4.2 Enabling the MOJO Scoring Pipeline ...................... 81
    14.4.3 MOJO Scoring Pipeline Files ............................. 82
    14.4.4 Quickstart ............................................. 82
    14.4.5 Compile and Run the MOJO from Java ...................... 83

15 Deployment ............................................................ 84
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.

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

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.
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.

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.

Data Visualization: For datasets, Driverless AI can generate visualizations and creates data plots that are most relevant from a statistical perspective based on the most relevant data statistics in order to help users get a quick understanding of their data prior to starting the model building process.

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.
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.

Time Series: 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.

Supported Algorithms

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.

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.

**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

**TensorFlow**

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

**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).

**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.

## 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 nvidia-docker. 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, LDAP, 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, 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.

**Launching Driverless AI**

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.

### Messages

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

Adding Datasets

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

Parquet Notes

- 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 Parquet files (depending on the number of partitions in the input dataset) + metadata.

- You may receive a "Failed to ingest binary file with Parquet: lists with structs are not supported" error when ingesting a 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. In Sparkling Water, we provide a workaround to flatten the Parquet file. Refer to our Sparkling Water solution for more information.

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, and S3 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 local filebrowser by default.

- If Driverless AI was started with data connectors enabled for HDFS, S3, Azure Blob Store, BlueData DataTap, Google Cloud Storage, Google Big Query, KDB+, Minio, and/or Snowflake, then a drop-down will appear allowing you to specify where to begin browsing for the dataset. Refer to the **Data Connectors** section in the Driverless AI User Guide for more information.

**Notes:**

- 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.

![Datasets Overview](image)

**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.

**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.

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.
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.

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.

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. In most cases, all possible scatterplots based on pairs of features (variables) are examined for correlations. However, if there are more than 50 numerical columns inside the dataset, Driverless AI randomly selects 50 of them and only examines all pairs of these 50. 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 Pearson’s $r$. (Only variables with Pearson $R^2 > 0.95^2$ are displayed.)
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. 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 ([14]). 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 $(\text{MS}_{\text{between}} - \text{MS}_{\text{within}})/(\text{MS}_{\text{between}} + (k - 1)\text{MS}_{\text{within}})$, 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 $k_1$ categories and the second variable has $k_2$ categories, then a $k_1 \times k_2$ 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 $(\chi^2 / n) / \min(k_1,k_2)$, 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.

- **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.
Running an Experiment

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 Running an Experiment 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.

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 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).

3. 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 $\leq$ 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.

4. 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.)

5. 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.

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 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 Model with Same Params**.

Completed Experiment

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

- **Deploy.** Refer to the Deployment section. (Only available after building MOJO scoring pipelines. Not available for PPC64LE environments.)

- **Interpret this Model.** Refer to the Interpreting a Model section. (Not supported for NLP experiments. Please contact H2O support for assistance with interpreting NLP 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.
• Build MOJO Scoring Pipeline. A standalone Model Object, Optimized scoring pipeline. (Not available for TensorFlow, RuleFit, or FTRL models.)

• Download Experiment Summary. A zip file containing the following:
  – 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 Logs

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 determine 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.
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 describing 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.)

### 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.

### 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 choose 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 Model with Same Params. 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).

**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.

**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
Interpreting a Model

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:**

- MLI no longer runs on experiments from previous releases, but also does not require internet to run on current models.

- MLI is not supported for NLP experiments. Please contact H2O support for assistance with interpreting NLP experiments.

**Interpret this Model button - Non-Time-Series**

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 non-time-series 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.
Interpreting a Model

Interpret this Model button - Time-Series

Limitations

- MLI for time series will not run without a test set from a Driverless AI experiment. In future releases, internal validation predictions will be used if a test set is not provided.

- 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.

- Model interpretation for time-series experiments is currently in an alpha state and changes are imminent.

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 (alpha) 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 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. Note that this panel can be resized if necessary.
- A **Show Summary** button that provides details about the experiment settings that were used.

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

**Single Time Series MLI**

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

1. Click the **Interpret this Model (alpha)** 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 the test set includes actuals, then a panel will display showing a time series plot showing the single group’s matrix tables based on the scorer that was used in the experiment. Note that this panel can be resized if necessary.

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

   - A **Group Search** entry field for selecting the group to view. Note that for Single Time Series MLI, there will only be one option in this field.
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.

**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. Select a LIME method of either K-LIME (default) or LIME-SUP.
   - 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.

6. For K-LIME interpretations, specify the depth that you want for your decision tree surrogate model. The tree depth value can be a value from 2-5 and defaults to 3. For LIME-SUP interpretations, specify the LIME-SUP tree depth. This can be a value from 2-5 and defaults to 3.

7. Optionally specify whether to use original features or transformed features in the surrogate model for the new interpretation. Note: If this option is disabled, then the K-LIME clustering column option will not be available, and quantile binning will not be available.

8. Specify whether to perform the interpretation on a sample of the training data. By default, MLI will sample the training dataset if it is greater than 100k rows. (Note that this value can be modified in the config.toml setting for mli_sample_size.) Turn this toggle off to run MLI on the entire dataset.

9. Optionally specify weight and dropped columns.

10. For K-LIME interpretations, optionally specify a clustering column. Note that this column should be categorical. Also note that this is only available
when K-LIME is used as the LIME method and when **Use Original Features** is enabled. If the LIME method is changed to LIME-SUP, then this option is no longer available.

11. Optionally specify the number of surrogate cross-validation folds to use (from 0 to 10). When running experiments, Driverless AI automatically splits the training data and uses the validation data to determine the performance of the model parameter tuning and feature engineering steps. For a new interpretation, Driverless AI uses 3 cross-validation folds by default for the interpretation.

12. For K-LIME interpretations, optionally specify one or more columns to generate decile bins (uniform distribution) to help with MLI accuracy. Columns selected are added to top \( n \) columns for quantile binning selection. If a column is not numeric or not in the dataset (transformed features), then the column will be skipped. **Note**: This option is only available when **Use Original Features** is enabled.

13. For K-LIME interpretations, optionally specify the number of top variable importance numeric columns to run decile binning to help with MLI accuracy. (Note that variable importances are generated from a Random Forest model.) This defaults to 0, and the maximum value is 10. **Note**: This option is only available when **Use Original Features** is enabled.

14. Click the **Launch MLI** button.

---

**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. Select a LIME method of either K-LIME (default) or LIME-SUP.

   - **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.
Interpreting a Model 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.

6. For K-LIME interpretations, specify the depth that you want for your decision tree surrogate model. The tree depth value can be a value from 2-5 and defaults to 3. For LIME-SUP interpretations, specify the LIME-SUP tree depth. This can be a value from 2-5 and defaults to 3.

7. Specify whether to perform the interpretation on a sample of the training data. By default, MLI will sample the training dataset if it is greater than 100k rows. (Note that this value can be modified in the config.toml setting for mli_sample_size.) Turn this toggle off to run MLI on the entire dataset.

8. Optionally specify weight and dropped columns.

9. For K-LIME interpretations, optionally specify a clustering column. Also note that this is only available when K-LIME is used as the LIME method. If the LIME method is changed to LIME-SUP, then this option is no longer available.

10. Optionally specify the number of surrogate cross-validation folds to use (from 0 to 10). When running experiments, Driverless AI automatically splits the training data and uses the validation data to determine the performance of the model parameter tuning and feature engineering steps. For a new interpretation, Driverless AI uses 3 cross-validation folds by default for the interpretation.

11. For K-LIME interpretations, optionally specify one or more columns to generate decile bins (uniform distribution) to help with MLI accuracy. Columns selected are added to top \( n \) columns for quantile binning selection. If a column is not numeric or not in the dataset (transformed features), then the column will be skipped.

12. For K-LIME interpretations, optionally specify the number of top variable importance numeric columns to run decile binning to help with MLI accuracy. (Note that variable importances are generated from a Random
Interpreting a Model

Forest model.) This value is combined with any specific columns selected for quantile binning. This defaults to 0, and the maximum value is 10.

13. Click the **Launch MLI** button.

---

**Understanding the Model Interpretation Page**

The Model Interpretation page opens with a Summary of the interpretation. This page also provides left-hand navigation for viewing additional plots. This navigation includes:

- **Summary**: Provides a summary of the MLI experiment.
- **DAI Model**: Provides Feature Importance and Shapley plots for transformed features. (Shapley plots not supported for RuleFit and TensorFlow models.)
- **Surrogate Models**: Provides KLIME and Decision Tree plots. This also includes a Random Forest submenu, which includes Global and Local Feature Importance plots for original features and a Partial Dependence plot.
- **Dashboard**: A single page with a Global Interpretable Model Explanations plot, a Feature Importance plot, a Decision Tree plot, and a Partial Dependence plot.
- **MLI Docs**: A link to the "Machine Learning Interpretability with Driverless AI" booklet.
- **Download MLI Logs**: Downloads a zip file of the logs that were generated during this interpretation.
- **Experiment**: Provides a link back to the experiment that generated this interpretation.
- **Scoring Pipeline**: Downloads the scoring pipeline for this interpretation.
- **Download Reason Codes**: Download a CSV file of LIME and/or Shapley reason codes.

**Summary Page**

The Summary page is the first page that opens when you view an interpretation. This page 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.

**DAI Model Dropdown**

This menu provides a Feature Importance plot and a Shapley plot for transformed features. (Shapley plots not supported for RuleFit and TensorFlow models.)

**Feature Importance**

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. 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.
Shapley Plot

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


You can view a Shapley explanations plot by selecting the Interpret this Model on Transformed Features button in an experiment.
Surrogate Models Dropdown

The Surrogate Models dropdown includes KLIME/LIME-SUP and Decision Tree plots as well as a Random Forest submenu, which includes Global and Local Feature Importance plots for original features and a Partial Dependence plot.

K-LIME and LIME-SUP

The MLI screen includes a KLIME or LIME-SUP graph. A KLIME 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 KLIME/LIME-SUP distinction provides insight into the LIME method that was used during model interpretation.

The KLIME Technique

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
Interpreting a Model

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:

<table>
<thead>
<tr>
<th>debt_to_income_ratio</th>
<th>credit_score</th>
<th>savings_acct_balance</th>
<th>observed_default</th>
<th>H2OAI_predicted_default</th>
<th>k-LIME_predicted_default</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>600</td>
<td>1000</td>
<td>1</td>
<td>0.85</td>
<td>0.9</td>
</tr>
</tbody>
</table>

And the local linear model:

$$y_{K-LIME} = 0.1 + 0.01 \times debt\_to\_income\_ratio + 0.0005 \times credit\_score + 0.0002 \times savings\_acct\_balance$$

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

- $\text{debt\_to\_income\_ratio}: 0.01 \times 30 = 0.3$
- $\text{credit\_score}: 0.0005 \times 600 = 0.3$
- $\text{savings\_acct\_balance}: 0.0002 \times 1000 = 0.2$

Each local contribution is positive and thus contributes positively to the Driverless AI model’s prediction of 0.85 for $\text{H2OAI\_predicted\_default}$. 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**

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 flowchart of the complex Driverless AI model’s decision making process. The decision tree surrogate model 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.

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. The decision tree surrogate is known to date back at least to 1996 (Craven and Shavlik, [2]). The decision tree surrogate model here is trained to predict the predictions of the more complex Driverless AI model using the of 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.

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.
Random Forest Dropdown

The Random Forest dropdown provides a submenu that includes a Feature Importance plot, a Partial Dependence plot, and a LOCO plot. These plots are for original features rather than transformed features.

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.
LOCO

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 here are based on a variant of the leave-one-covariate-out (LOCO) method (Lei et al, 2017 [9]).

In the LOCO-variant method, each local feature importance is found 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.

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

<table>
<thead>
<tr>
<th>debt_to_income_ratio</th>
<th>credit_score</th>
<th>savings_acct_balance</th>
<th>observed_default</th>
<th>H2OAI_predicted_default</th>
<th>k-LIME_predicted_default</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>600</td>
<td>1000</td>
<td>1</td>
<td>0.85</td>
<td>0.9</td>
</tr>
</tbody>
</table>

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

First, the modified predictions are calculated:

$$F_{\text{debt_to_income_ratio}} = F(NA, 600, 1000) = 0.99$$
$$F_{\text{credit_score}} = F(30, NA, 1000) = 0.73$$
$$F_{\text{savings_acct_balance}} = F(30, 600, NA) = 0.82$$
Second, the original prediction is subtracted from each modified prediction to generate the unscaled local feature importance values:

\[
\begin{align*}
\text{LOCO}_{\text{debt to income ratio}} &= F_{\text{debt to income ratio}} - 0.85 = 0.99 - 0.85 = 0.14 \\
\text{LOCO}_{\text{credit score}} &= F_{\text{credit score}} - 0.85 = 0.73 - 0.85 = -0.12 \\
\text{LOCO}_{\text{savings acct balance}} &= F_{\text{savings acct balance}} - 0.85 = 0.82 - 0.85 = -0.03
\end{align*}
\]

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.

\[
\begin{align*}
\text{Scaled(LOCO}_{\text{debt to income ratio}} &= \text{Abs}(\text{LOCO}_{\text{debt to income ratio}}/0.14) = 1 \\
\text{Scaled(LOCO}_{\text{credit score}} &= \text{Abs}(\text{LOCO}_{\text{credit score}}/0.14) = 0.86 \\
\text{Scaled(LOCO}_{\text{savings acct balance}} &= \text{Abs}(\text{LOCO}_{\text{savings acct balance}}/0.14) = 0.21
\end{align*}
\]

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.

**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.

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.

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

<table>
<thead>
<tr>
<th>debt_to_income_ratio</th>
<th>credit_score</th>
<th>savings_acct_balance</th>
<th>observed_default</th>
<th>H2OAI_predicted_default</th>
<th>k-LIME_predicted_default</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>600</td>
<td>1000</td>
<td>1</td>
<td>0.85</td>
<td>0.9</td>
</tr>
</tbody>
</table>

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:

\[
\text{ICE}_{\text{credit\_score}, 500} = F(30, 500, 1000)
\]

\[
\text{ICE}_{\text{credit\_score}, 530} = F(30, 530, 1000)
\]

\[
\text{ICE}_{\text{credit\_score}, 560} = F(30, 560, 1000)
\]

\[
\vdots
\]

\[
\text{ICE}_{\text{credit\_score}, 800} = F(30, 800, 1000)
\]

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 less than 20 unique values are only included if in the top 10).

Dashboard Page

The Model Interpretation Dashboard includes the following information:

- Global interpretable model explanation plot
- Feature importance (Global for original features; LOCO for interpretations with predictions and when interpreting on raw features)
- Decision tree surrogate model
- Partial dependence and individual conditional expectation plots
General Considerations

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]).

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.

**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.

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 Model Interpretation page. Note that this button is only available for completed experiments. Click Close when you are done to return to the Model Interpretations page.

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, select the Global plot from the Cluster dropdown.

With Global selected, click the Explanations button beside the Cluster dropdown.

- **Cluster Reason Codes**: To view reason codes for a specific cluster, select a cluster from the Cluster dropdown.

With a cluster selected, click the Explanations button.
• **Local Reason Codes:** To view local reason codes, select a point on the graph or type a value in the Value field.

With a value selected, click the **Explanations** button in the upper-right corner.
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.

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.

The Python and MOJO Scoring Pipelines

Driverless AI provides a Python Scoring Pipeline for experiments and interpreted models and a MOJO (Java-based) Scoring Pipeline for experiments.

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.
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.
- Use the MOJO Scoring Pipeline for a pure Java solution. This solution is flexible and is faster than the Python Scoring Pipeline, but it requires a bit more coding.
- 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.

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).

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).

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

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 3.6 and OpenBlas Ubuntu 16.10+**

```
$ sudo apt install python3.6 python3.6-dev python3-pip python3-dev \
    python-virtualenv python3-virtualenv libopenblas-dev
```

**Installing Python 3.6 and OpenBLAS on Ubuntu 16.4**

```
$ sudo add-apt-repository ppa:deadsnakes/ppa
$ sudo apt-get update
$ sudo apt-get install python3.6 python3.6-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](https://docs.anaconda.com/anaconda/install.html)
- **Miniconda**: [https://conda.io/docs/user-guide/install/index.html](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](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
```
The Python and MOJO Scoring Pipelines

$ ./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/Miniconda) to create Conda virtual environment and install required package dependencies. The package manager to use is provided as an argument to the script.
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:

```python
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:

```bash
# See 'run_example.sh' for complete example.
$ virtualenv -p python3.6 env
$ source env/bin/activate
$ pip install -r requirements.txt
$ export DRIVERLESS_AI_LICENSE_FILE="/path/to/license.sig"
$ python example.py
```
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:

```
$ 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
$ python tcp_client.py

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:

```java
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.


To start the scoring service in HTTP mode:
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:

```python
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'])
```

**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
```
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`

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 python3.6 python3.6-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.

```bash
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.

**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).

**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).

**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:

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

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

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

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 3.6 on Ubuntu 16.10+
$ sudo apt install python3.6 python3.6-dev python3-pip python3-dev \ python-virtualenv python3-virtualenv

# Installing Python3.6 on Ubuntu 16.04
$ sudo add-apt-repository ppa:deadsnakes/ppa
$ sudo apt-get update
$ sudo apt-get install python3.6 python3.6-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
```

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
```

**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:

```python
----- 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 python3.6 env
$ source env/bin/activate
$ pip install -r requirements.txt
$ python example.py
```

**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.

**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:

```
$ thrust --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. ------
$ thrust --gen py scoring.thrift
$ python example_client.py
```

```
----- 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
```

```
----- See 'run_tcp_client.sh' for complete example. ------
$ thrust --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
```

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

```
----- See 'example_client.py' for complete example. ------
```
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:

```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:

```bash
----- 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'])
```

**Driverless AI MOJO Scoring Pipeline**

**Note:**

- The MOJO Scoring Pipeline is currently in a beta state. Updates and improvements will continue to be made in subsequent Driverless AI releases.

- MOJOs are currently not available for TensorFlow, RuleFit, or FTRL models.

For completed experiments, Driverless AI 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.

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.

**Prerequisites**

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

- Java 7 runtime (JDK 1.7) or newer.
• 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:

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

Enabling the MOJO Scoring Pipeline

The MOJO Scoring Pipeline is disabled by default. As a result, a MOJO will have to be built for each desired experiment by clicking on the Build MOJO Scoring Pipeline button:
To enable MOJO Scoring Pipelines for each experiment, stop Driverless AI, then restart using the `DRIVERLESS_AI_MAKE_MOJO_SCORING_PIPELINE=1` flag. (Refer to the Config.toml File section in the User Guide. for more information.) For example:

```bash
nvidia-docker run \
--add-host name.node:172.16.2.186 \
-e DRIVERLESS_AI_MAKE_MOJO_SCORING_PIPELINE=1 \
-p 12345:12345 \
--init -it --rm \
-v /tmp/dtmp:/tmp \
-v /tmp/dlog:/log \
-u $(id -u):$(id -g) \
opsh2oai/h2oai-runtime
```

Or you can change the value of `make_mojo_scoring_pipeline` to `true` in the config.toml file and specify that file when restarting Driverless AI.

**MOJO Scoring Pipeline Files**

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

- `run_example.sh`: An 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, of the correct format).

**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.

   ![DOWNLOAD MOJO SCORING PIPELINE]

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

2. To score all rows in the sample test set (`example.csv`) with the MOJO pipeline (`pipeline.mojo`) and license stored in the environment variable `DRIVERLESS_AI_LICENSE_KEY`:
3. To score a specific test set (example.csv) with the MOJO pipeline (pipeline.mojo) and the license file (license.sig):

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

4. To run Java application for data transformation directly:

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

**Compile and Run the MOJO from Java**

1. Open a new terminal window and change directories to the `experiment` folder:

```bash
$ 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.

```java
import java.io.IOException;
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.utils.SimpleCSV;

public class Main {

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

        // Get and fill the input columns
        MojoFrameBuilder frameBuilder = model.getInputFrameBuilder();
        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
        MojoFrame iframe = frameBuilder.toMojoFrame();

        // Transform input frame by MOJO pipeline
        MojoFrame oframe = model.transform(iframe);
        // 'MojoFrame.debug()' can be used to view the contents of a Frame
        // oframe.debug();
    }
}
```
3. Compile the source code:

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

4. Run the MOJO example:

```
# 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
```

## Deployment

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

**Note:** This is an early feature that will eventually support multiple different deployments. At this point, Driverless AI can deploy the trained MOJO scoring pipeline as an **AWS Lambda Function**, i.e., a server-less scorer running in Amazon Cloud and charged by the actual usage.

## 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.
AWS Lambda Deployment

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.

In addition, the Terraform tool has to be installed on the system running Driverless AI. The tool is included in the Driverless AI Docker images but not in native install packages. To install Terraform, please follow steps on Terraform installation. Note: Terraform is not available on every platform. In particular, there is no Power build, so AWS Lambda Deployment is currently not supported on Power installations of Driverless AI.

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:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "iam:GetPolicyVersion",
                "iam:DeletePolicy",
                "iam:CreateRole",
                "iam:AttachRolePolicy",
                "iam:ListInstanceProfilesForRole",
```
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.

**Testing the Lambda**

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:

```json
{
    "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:

```bash
$ URL={place the endpoint URL here}
$ API_KEY={place the endpoint API key here}
$ curl \n```
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:

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

---

**About Driverless AI Transformations**

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

- **Filter Transformer**: The Filter Transformer counts each numeric value in the dataset.

- **Frequent Transformer**: The Frequent Transformer counts each categorical value in the dataset. This count can be either the raw count or the normalized count.

- **Bulk Interactions Transformer**: The Bulk Interactions Transformer will add, divide, multiply, and subtract two columns in the data.

- **Truncated SVD Numeric Transformer**: Truncated SVD trains on a selected numeric of columns in the data. The components of the truncated SVD will be new features.
• **Cross Validation Target Encoding**: Cross validation target encoding is done on a categorical column.

• **Cross Validation Categorical to Numeric Encoding**: This transformer converts a categorical column to a numeric column. Cross validation target encoding is done on the categorical column.

• **Dates Transformer**: The Dates Transformer retrieves any date values, including:
  - Year
  - Quarter
  - Month
  - Day
  - Day of year
  - Week
  - Week day
  - Hour
  - Minute
  - Second

• **Text Transformer**: The Text Transform transforms a text column using TFIDF (term frequency-inverse document frequency) or count (count of the word). This may be followed by dimensionality reduction using truncated SVD.

• **Numeric to Categorical Target Encoding Transformer**: This transformer converts a numeric column to categorical by binning. Cross validation target encoding is done on the binned column.

• **Cluster Target Encoding Transformer**: Selected columns in the data are clustered, and target encoding is done on the cluster ID.

• **Cluster Distance Transformer**: Selected columns in the data are clustered, and the distance to a chosen cluster center is calculated.

• **Weight of Evidence**: Creates likelihood type of features using the Weights Of Evidence (WOE) transformation method. The weight of evidence tells the predictive power of an independent variable in relation to the dependent variable, for example, the measurement of good customers in relations to bad customers.
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/.

- **Numeric To Categorical Weight of Evidence Transformer**: This transformer converts a numeric column to categorical by binning and then creates the likelihood type of features using the WOE transformation method.

- **DateTime Label Encoder**: Time series frequency preserving label encoding of date times or relative times in [ns].

- **DateTime Label Normalizer**: Normalization of label encoded time-axis between train/valid or train/test.

- **Lags Transformer**: Creation of target or feature lags.

- **Lags Interaction Transformer**: Creation of interactions between target/feature lags (lag2 - lag1, for instance).

- **Lags Aggregates Transformer**: Aggregations of target/feature lags like mean(lag7, lag14, lag21) with support for mean, min, max, median, sum, skew, kurtosis, std.

- **Exponential Lags Smoothing Transformer**: Exponential smoothing of target/feature lags.

- **Linear Lags Regression Transformer**: Linear regression of lags.

### Example Transformations

In this section, we will describe the transformations using the example of predicting house prices on the example dataset.

<table>
<thead>
<tr>
<th>Date Built</th>
<th>Square Footage</th>
<th>Num Beds</th>
<th>Num Baths</th>
<th>State</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/01/1920</td>
<td>1700</td>
<td>3</td>
<td>2</td>
<td>NY</td>
<td>$700K</td>
</tr>
</tbody>
</table>

- **Frequent Transformer**
  - the count of each categorical value in the dataset
• the count can be either the raw count or the normalized count

<table>
<thead>
<tr>
<th>Date Built</th>
<th>Square Footage</th>
<th>Num Beds</th>
<th>Num Baths</th>
<th>State</th>
<th>Price</th>
<th>Freq_ State</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/01/1920</td>
<td>1700</td>
<td>3</td>
<td>2</td>
<td>NY</td>
<td>700,000</td>
<td>4.500</td>
</tr>
</tbody>
</table>

There are 4,500 properties in this dataset with state = NY.

**Bulk Interactions Transformer**

• add, divide, multiply, and subtract two columns in the data

<table>
<thead>
<tr>
<th>Date Built</th>
<th>Square Footage</th>
<th>Num Beds</th>
<th>Num Baths</th>
<th>State</th>
<th>Price</th>
<th>Interaction_NumBeds#subtract#NumBaths</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/01/1920</td>
<td>1700</td>
<td>3</td>
<td>2</td>
<td>NY</td>
<td>700,000</td>
<td>1</td>
</tr>
</tbody>
</table>

There is one more bedroom than there are number of bathrooms for this property.

**Truncated SVD Numeric Transformer**

• truncated SVD trained on selected numeric columns of the data
• the components of the truncated SVD will be new features

<table>
<thead>
<tr>
<th>Date Built</th>
<th>Square Footage</th>
<th>Num Beds</th>
<th>Num Baths</th>
<th>State</th>
<th>Price</th>
<th>TruncSVD_Price_NumBeds_NumBaths_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/01/1920</td>
<td>1700</td>
<td>3</td>
<td>2</td>
<td>NY</td>
<td>700,000</td>
<td>0.632</td>
</tr>
</tbody>
</table>

The first component of the truncated SVD of the columns Price, Number of Beds, Number of Baths.

**Dates Transformer**

• get year, get quarter, get month, get day, get day of year, get week, get week day, get hour, get minute, get second

<table>
<thead>
<tr>
<th>Date Built</th>
<th>Square Footage</th>
<th>Num Beds</th>
<th>Num Baths</th>
<th>State</th>
<th>Price</th>
<th>DateBuilt_Month</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/01/1920</td>
<td>1700</td>
<td>3</td>
<td>2</td>
<td>NY</td>
<td>700,000</td>
<td>1</td>
</tr>
</tbody>
</table>

The home was built in the month January.

**Text Transformer**

• transform text column using methods: TFIDF or count (count of the word)

• this may be followed by dimensionality reduction using truncated SVD

**Categorical Target Encoding Transformer**
• cross validation target encoding done on a categorical column

<table>
<thead>
<tr>
<th>Date Built</th>
<th>Square Footage</th>
<th>Num Beds</th>
<th>Num Baths</th>
<th>State</th>
<th>Price</th>
<th>CV_TE_State</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/01/1920</td>
<td>1700</td>
<td>3</td>
<td>2</td>
<td>NY</td>
<td>700,000</td>
<td>550,000</td>
</tr>
</tbody>
</table>

The average price of properties in NY state is $550,000*.

*In order to prevent overfitting, Driverless AI calculates this average on out-of-fold data using cross validation.

**Numeric to Categorical Target Encoding Transformer**

• numeric column converted to categorical by binning

• cross validation target encoding done on the binned numeric column

<table>
<thead>
<tr>
<th>Date Built</th>
<th>Square Footage</th>
<th>Num Beds</th>
<th>Num Baths</th>
<th>State</th>
<th>Price</th>
<th>CV_TE_SquareFootage</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/01/1920</td>
<td>1700</td>
<td>3</td>
<td>2</td>
<td>NY</td>
<td>700,000</td>
<td>345,000</td>
</tr>
</tbody>
</table>

The column **Square Footage** has been bucketed into 10 equally populated bins. This property lies in the **Square Footage** bucket 1,572 to 1,749. The average price of properties with this range of square footage is $345,000*.

*In order to prevent overfitting, Driverless AI calculates this average on out-of-fold data using cross validation.

**Cluster Target Encoding Transformer**

• selected columns in the data are clustered

• target encoding is done on the cluster ID

<table>
<thead>
<tr>
<th>Date Built</th>
<th>Square Footage</th>
<th>Num Beds</th>
<th>Num Baths</th>
<th>State</th>
<th>Price</th>
<th>ClusterTE_4_NumBeds_NumBaths_SquareFootage</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/01/1920</td>
<td>1700</td>
<td>3</td>
<td>2</td>
<td>NY</td>
<td>700,000</td>
<td>450,000</td>
</tr>
</tbody>
</table>

The columns: **Num Beds**, **Num Baths**, **Square Footage** have been segmented into 4 clusters. The average price of properties in the same cluster as the selected property is $450,000*.

*In order to prevent overfitting, Driverless AI calculates this average on out-of-fold data using cross validation.

**Cluster Distance Transformer**

• selected columns in the data are clustered

• the distance to a chosen cluster center is calculated
The columns: Num Beds, Num Baths, Square Footage have been segmented into 4 clusters. The difference from this record to Cluster 1 is 0.83.

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 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 AI. 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`.

**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`.

**References**


15. H2O.ai Team. **Datatable for python.** URL [https://github.com/h2oai/datatable](https://github.com/h2oai/datatable)

**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.