Course: Machine Learning

Lab 1: Working with Regression Models

Overview

In this lab, you will learn how to construct and evaluate regression machine learning models using Azure ML and Python.

Regression is one of the fundamental machine learning methods used in data science. Regression enables you to predict values of a label variable given data from the past. Regression, like classification, is a supervised machine learning technique, wherein models are trained from labeled cases. In this case, you will train and evaluate a nonlinear regression model that produces improved predictions of building energy efficiency.

What You'll Need

To complete this lab, you will need the following:

- An Azure ML account
- A web browser and Internet connection
- The lab files for this lab

Regression Modeling with Azure ML

You will create and evaluate an improved regression model using Azure ML. The steps in this process include:

- Testing and evaluating a new model type.
- Pruning the features for the new model.
- Using the Tune Model Hyperparameters module improved model performance.
- Cross validating the model to ensure it generalizes well.
- Evaluating the performance of the model in depth with Python.

Build a New Model

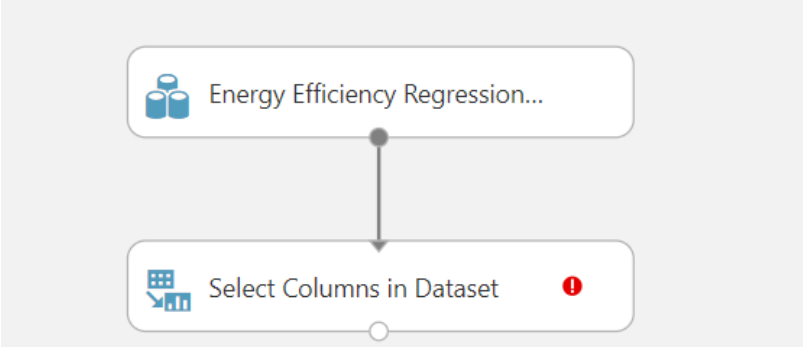
In this procedure, you will construct and evaluate a nonlinear regression model. There are a number of nonlinear regression models supported in Azure ML. In this case, you will work with the **Decision Forest Regression** method. This approach uses majority voting among an ensemble (group) of regression tree models. The splits in tree models exhibit nonlinear behavior. Using an ensemble averages out errors.

By following these steps, you will add and evaluate a nonlinear regression module in your experiment:

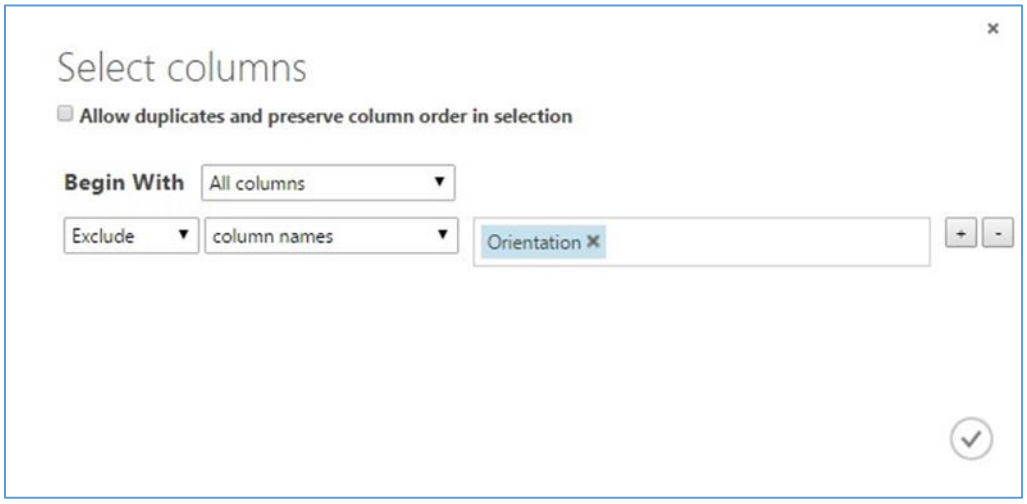
1. If you have not already done so, open a browser and browse to https://studio.azureml.net. Then sign in using the Microsoft account associated with your Azure ML account.
2. Create New Experiment.

3. Search for the **Select Columns in Dataset** module and drag it onto the canvas. Then make the following connections:

   - connect the **Results dataset** output port of the **Energy Efficiency Regression data** to the **Dataset** input port of the **Select Columns in Dataset** module. Including the data preparation steps (upper part) of your experiment, it should resemble the following diagram.



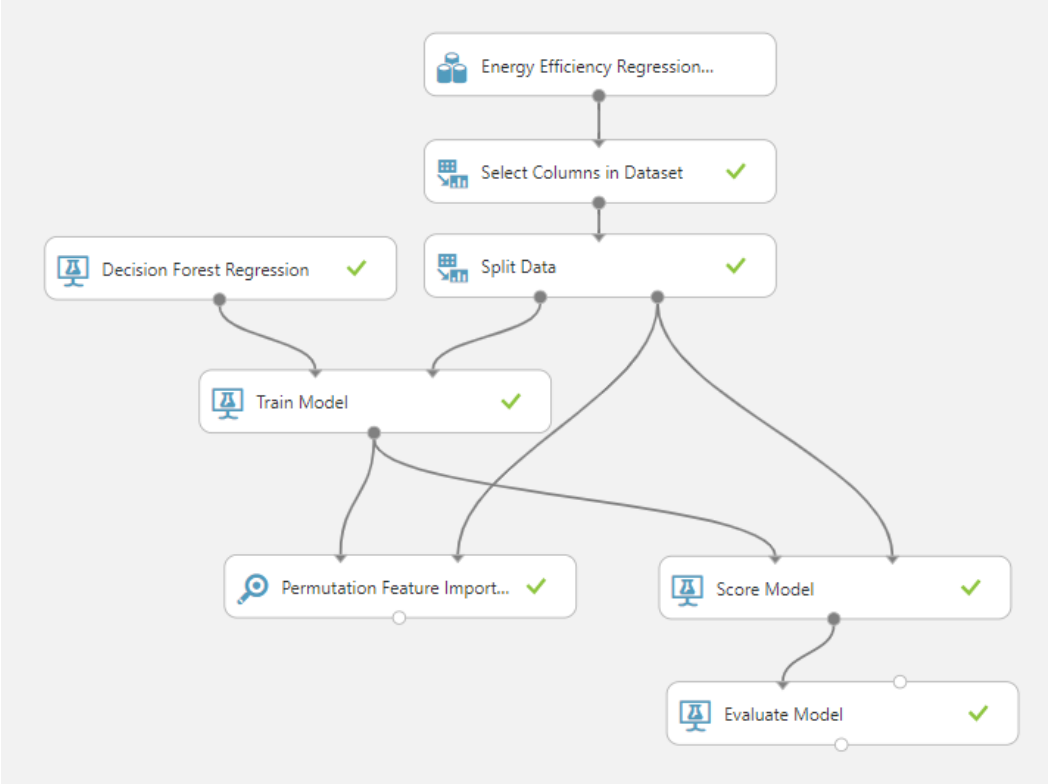
4. After the connections are made, on the **Properties** pane for the **Select Columns in Dataset** module, launch the column selector. Begin with all columns, and exclude **Orientation**, a feature known to have no predictive power, as shown in the following image.



5. Search for the **Split Data** module. Drag this module onto your experiment canvas. Connect the **Results dataset** output port of the **Select Columns in Dataset** module to the **Dataset** input port of the **Split Data** module. Set the **Properties** of the **Split Data** module as follows:

   - **Splitting mode:** Split Rows
   - **Fraction of rows in the first output:** 0.6
   - **Randomized split:** Checked
   - **Random seed:** 5416
   - **Stratified split:** False

6. Search for the **Decision Forest Regression** module. Make sure you have selected the regression model version of this algorithm. Drag this module onto the canvas. Set the properties of this module as follows:
   - **Resampling method:** Bagging
   - **Create trainer mode:** Single Parameter
   - **Number of decision trees:** 40
   - **Maximum depth of the decision trees:** 32
   - **Number of random splits per node:** 128
   - **Minimum number of samples per leaf node:** 4
   - **Allow unknown values for categorized features:** Checked
7. Search for the **Train Model** module. Drag this module onto the canvas.
8. Connect the **Untrained Model** output port of the **Decision Forest Regression** module to the **Untrained Model** input port of the **Train Model** module.
9. Connect the **Results dataset1** (left) output port of the **Split Data** module to the **Dataset input** port of the **Train model** module.
10. Select the **Train Model** module. Then, on the **Properties** pane, launch the column selector and select the **Heating Load** column.
11. Search for the **Score Model** module and drag it onto the canvas.
12. Connect the **Trained Model** output port of the of the **Train Model** module to the **Trained Model** input port of the **Score Model** module. Then connect the **Results dataset2** (right) output port of the **Split Data** module to the **Dataset** port of the **Score Model** module.
13. Search for the **Permutation Feature Importance** module and drag it onto the canvas.
14. Connect the **Trained Model** output port of the **Train Model** module to the **Trained model** input port of the **Permutation Feature Importance** module. Then connect the **Results dataset2** (right) output port of the **Split Data** module to the **Dataset** port of the **Test data** input port of the **Permutation Feature Importance** module.
15. Select the **Permutation Feature Importance** module, and in the **Properties** pane, set the following parameters:
    - **Random Seed:** 4567
    - **Metric for measuring performance:** Regression – Root Mean Squared Error
16. Search for the **Evaluate Model** module and drag it onto the canvas. Connect the **Scored Dataset** output port of the **Score Model** module to the left **Scored dataset** input port of the **Evaluate Model** module. The new portion of your experiment should now look like the following.



17. Save and run the experiment. When the experiment is finished, visualize the **Evaluation Result** port of the **Evaluate Model** module and review the performance statistics for the model.
18. Can you draw any conclusions from the results?

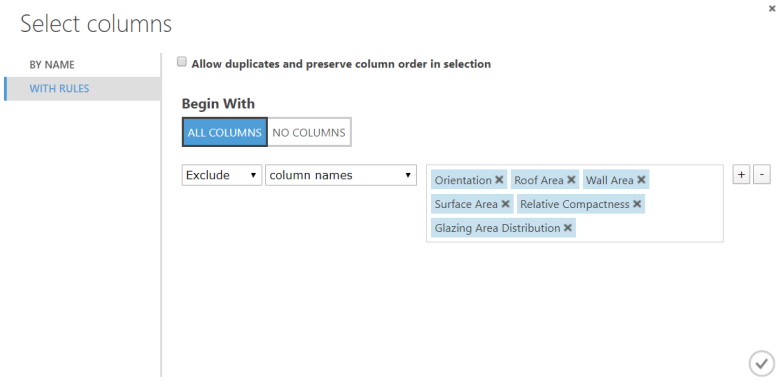
Prune Features

1. Visualize the **Feature importance** output port of the **Permutation Feature Importance** module, and note that there are some columns with low scores (less than 1), indicating that these columns have little importance in predicting the label. You can optimize your model and make it more generalizable by removing (or *pruning*) some of these features.
2. Make a note of the feature with the lowest importance score, and close the feature importance dataset.
3. Select the **Select Columns in Dataset** module you added at the beginning of this exercise, and on the **Properties** pane, click **Launch column selector**. Add the feature you identified as the least important to the list of columns to be excluded.
4. Save and run the experiment. When the experiment has finished running, click the **Evaluation results** output port of the **Evaluate Model** module and select **Visualize**. Note that these performance measures have been changed very little by pruning the least important feature. This result indicates that removing this feature was a good idea. In general, if removing a feature makes little difference in model performance, you are better off removing it. This approach simplifies the model and reduces the chances your model will not generalize well to new input values when the model is placed in production.

5. Select the **Select Columns in Dataset** module again, and launch the column selector. In a real experiment, you would remove features one by one and re-evaluate the model at each stage until its accuracy starts to decrease. However, in this lab, go ahead and configure the **Select Columns in Dataset** module to exclude the following features, which do not change the model accuracy metrics significantly:

   - **Orientation**
   - **Glazing Area Distribution**
   - **Surface Area**
   - **Relative Compactness**
   - **Wall Area**

   At this point the **Column Selector** of the **Select Columns in Dataset** module should be set to exclude the columns shown in the following image.



   At the end of the pruning process, you are left with the following four features:

   - **Overall Height**
   - **Wall Area**
   - **Glazing Area**
   - **Roof Area**

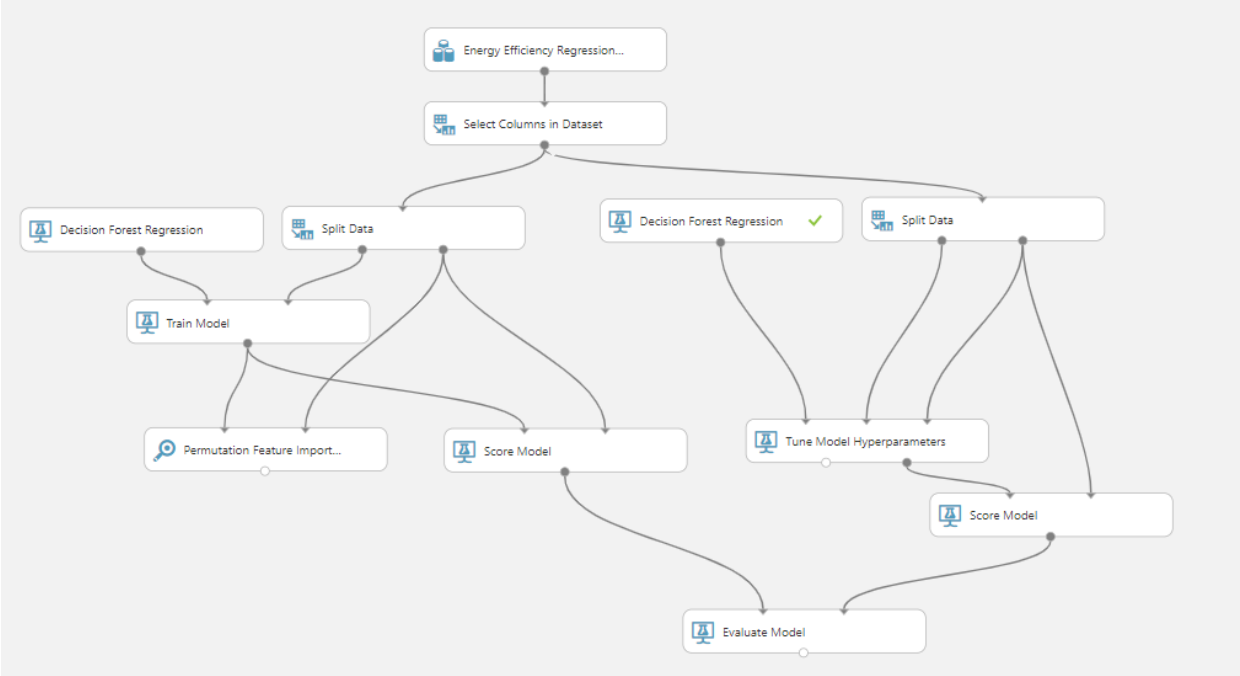
   Removing any of these features will cause the accuracy metrics to be degraded. Evidently, these are all you need for good model performance.

Sweep Parameters to Improve the Model

You will now use the **Tune Model Hyperparameters** module to optimize the performance of the **Decision Forest** model. The **Tune Model Hyperparameters** module searches a number of parameter combinations to find the combination producing the best model performance.

1. Select all of the modules below the **Select Columns in Dataset** module you added at the beginning of this lab. Then copy and paste these modules onto the canvas and drag the copies to one side.
2. Connect the **Results dataset** output of the **Select Columns in Dataset** module to the **Dataset input** of the new **Split Data** module.

3. Remove the copied **Permutation Feature Importance**, **Train Model**, and **Evaluate Model** modules.
4. Search for the **Tune Model Hyperparameters** module. Drag this module onto the canvas in place of the **Train Model** module you removed.
5. Connect the **Untrained model** output port of the new **Decision Forest Model** module to the **Untrained model** (left) input port of the **Tune Model Hyperparameters** module.
6. Connect the **Results dataset1** (left) output port of the new **Split Data** module to the **Training dataset** (middle) input port of the **Tune Model Hyperparameters** module.
7. Connect the **Results dataset2** (right) output port of the **Split Data** module to the **Optional test dataset** (right) input port of the **Tune Model Hyperparameters** module.
8. Connect the **Trained model** (right) output of the **Tune Model Hyperparameters** module to the **Trained model** (left) input of the new **Score Model** module.
9. Click the **Tune Model Hyperparameters** module to expose the **Properties** pane. Set the properties as follows so that 20 combinations of parameters are randomly tested:
    - **Specify parameter sweeping mode: Random sweep**
    - **Maximum number of runs on random sweep: 20**
    - **Random seed: 4567**
    - **Column Selector: Heating Load**
    - **Metric for measuring performance for classification: Accuracy**
    - **Metric for measuring performance for regression: Coefficient of determination**
10. Connect the **Scored dataset** output port of the copied **Score Model** module to the **Scored dataset to compare** (right) input port of the original **Evaluate Model** module for the first **Decision Forest Regression** model. The lower part of your experiment should now look like the following.



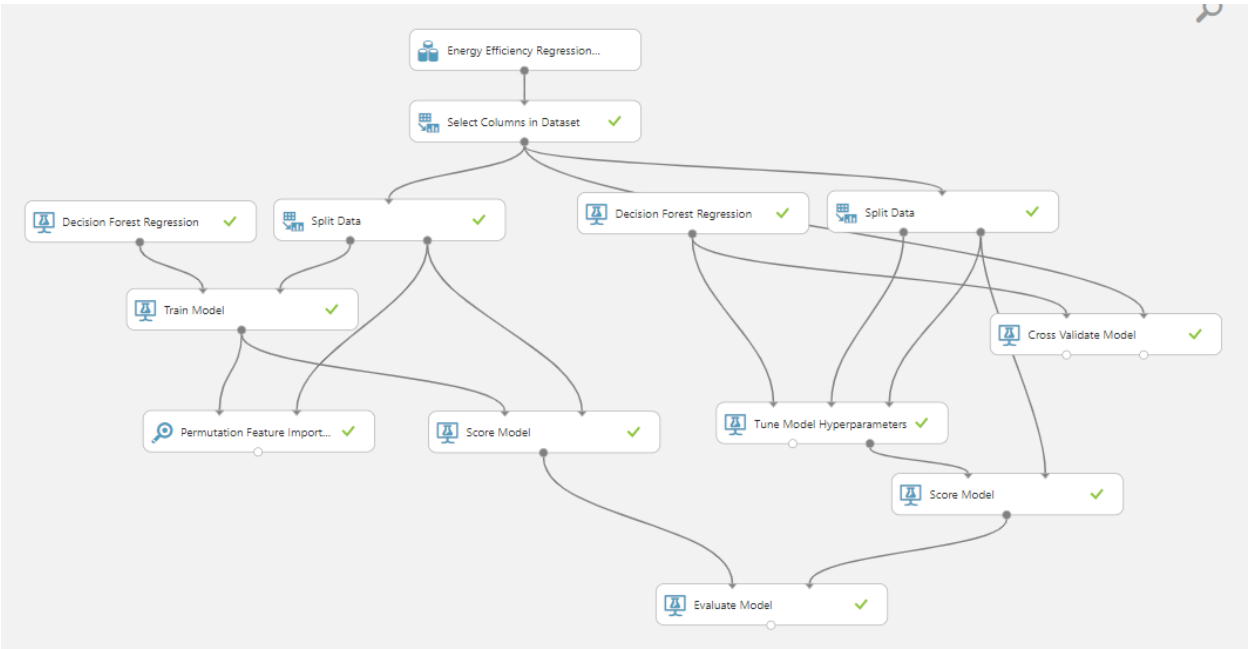
11. Save and run the experiment. When the experiment is finished, visualize the **Evaluation results** output port of the **Evaluate Model** module and compare the **Coefficient of Determination** and **Relative Squared Error** values for the two models.
12. Can you draw any conclusions from the results?

Cross Validate the Model

Cross validation or a machine learning model uses resampling of the dataset to test the performance on a number of training and testing data subsets. Each training and testing data subset sampled from the complete dataset is called a *fold*. Ideally, a good machine learning model should work well regardless of the test data used. When cross validated, a good model will have similar performance across the folds. This property of good machine learning models is known as *generalization*. A model that generalizes produces good results for any possible set of valid input values. Models that generalize can be expected to work well in production.

1. Search for the **Cross Validate Model** module. Drag this module onto the canvas.
2. Connect the **Untrained model** output from the most recently added **Decision Forest Model** module (the one connected to the **Tune Model Hyperparameters** module) to the **Untrained model** input port of the **Cross Validate Model** module.
3. Connect the **Results dataset** output port of the **Select Columns in Dataset** module to the **Dataset input** port of the **Cross Validate Model** module.
4. Select the **Cross Validate Model** module, and set its properties as follows:
   - **Column Selector**: Heating Load
   - **Random seed**: 3467

   Your experiment should resemble the following.



5. Save and run the experiment. When the experiment has finished, visualize the **Evaluation Results by Fold** (right) output port of the **Cross Validate Model** module. Scroll to the right and note the **Relative Squared Error** and **Coefficient of Determination** columns. Scroll to the bottom of the page, past the results of the 10 folds of the cross validation in the first 10 rows, and examine the **Mean** row toward the bottom. These results look like the following.

rows	columns
12	9

5	76	es.Gemini.Dll.GeminiDecisionForestRegressor	175.569194	1.267792	1.735611	0.147654
6	77	Microsoft.Analytics.Modules.Gemini.Dll.GeminiDecisionForestRegressor	124.963905	1.14664	1.621733	0.119527
7	77	Microsoft.Analytics.Modules.Gemini.Dll.GeminiDecisionForestRegressor	314.145116	1.360657	2.116276	0.156862
8	76	Microsoft.Analytics.Modules.Gemini.Dll.GeminiDecisionForestRegressor	142.112427	1.292251	1.816558	0.133412
9	77	Microsoft.Analytics.Modules.Gemini.Dll.GeminiDecisionForestRegressor	110.947165	0.946714	1.346069	0.102507
Mean	768	Microsoft.Analytics.Modules.Gemini.Dll.GeminiDecisionForestRegressor	165.929537	1.166598	1.718625	0.128643
Standard Deviation	768	Microsoft.Analytics.Modules.Gemini.Dll.GeminiDecisionForestRegressor	76.080484	0.139892	0.222327	0.017096

Notice that the **Relative Squared Error** and **Coefficient of Determination** values in the folds (along the two rightmost columns) are not that different from each other. The values in the folds are close to the values shown in the **Mean** row. Finally, the values in the **Standard Deviation** row are much

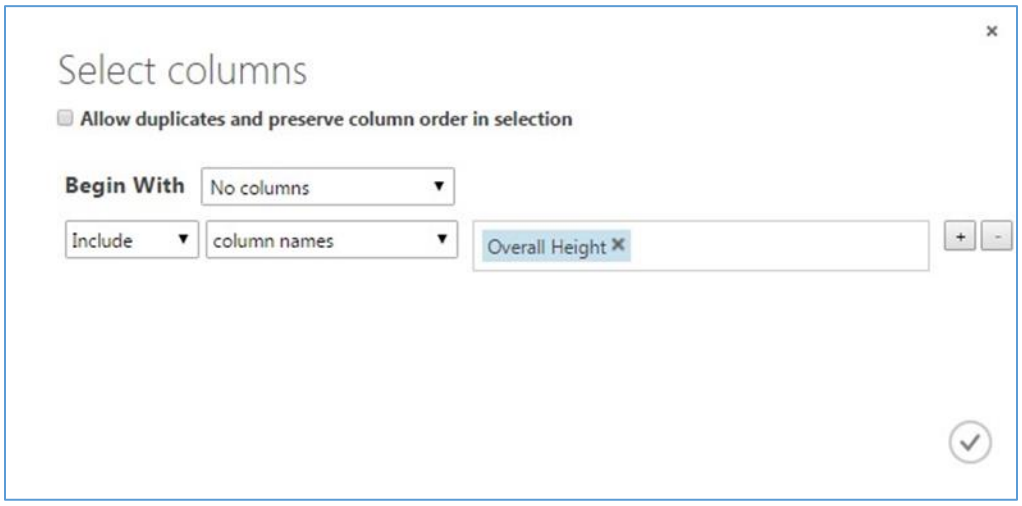
smaller than the corresponding values in the **Mean** row. These consistent results across the folds indicate that the model is insensitive to the training and test data chosen, and is likely to generalize well.

Evaluating Model Errors with Python

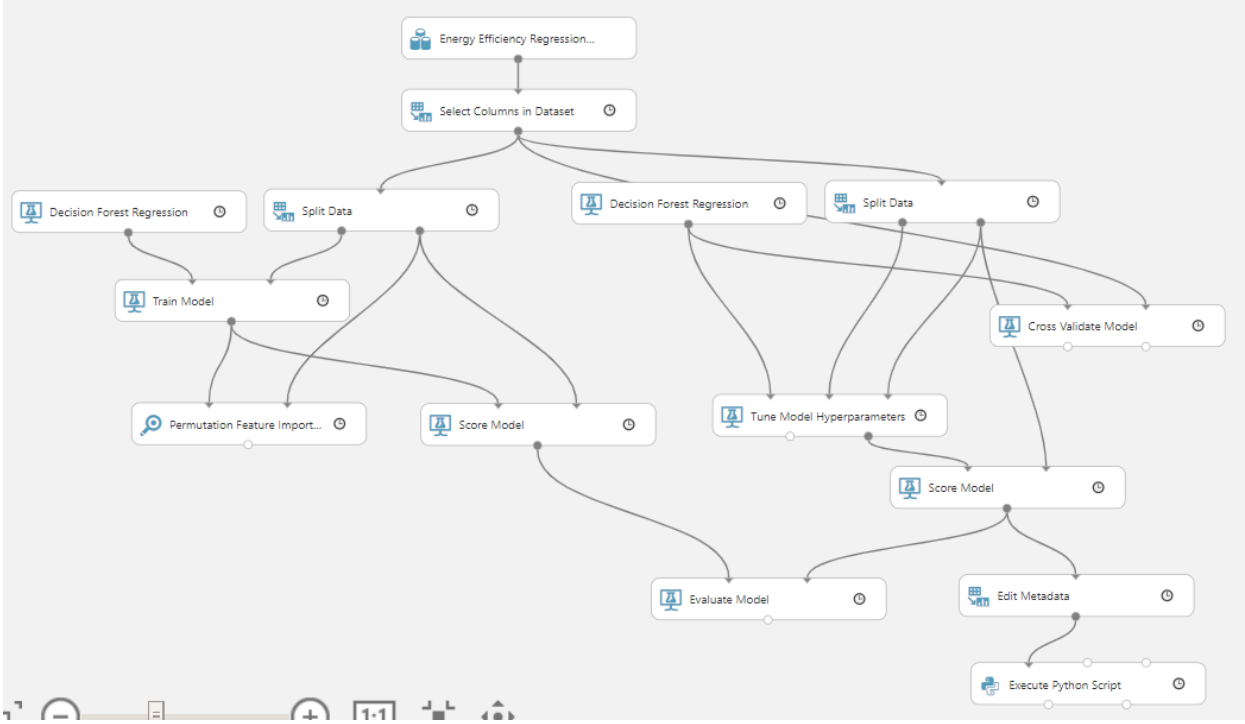
In this exercise, you will evaluate the model errors using custom Python code

The summary statistics for the nonlinear **Decision Forest Regression** model look quite promising. However, summary statistics can hide some significant problems one should understand. To investigate the residuals, or model errors, you will use some custom Python code.

1. Search for and locate the **Edit Metadata** module. Drag this module onto your canvas.
2. Connect the **Scored Dataset** output of the newest **Score Model** module (the one connected to the **Tune Model Hyperparameters** module) to the input of the **Metadata Editor** module.
3. Select the **Edit Metadata** model, and in the **Properties** pane, click **Launch Column Selector**.
4. Choose the **Overall Height** column as shown in the following image.



5. In the **Categorical** box, select **Make non-categorical**. The output from this **Metadata Editor** model will show the **Overall Height** column as a string type, which we can work with in Python.
6. Search for and locate the **Execute Python Script** module. Drag this module onto your canvas.
7. Connect the **Results Dataset** output of the **Edit Metadata** module to the **Dataset1** (left) input of the **Execute Python Script** module. Your experiment should resemble the following image.



8. With the **Execute Python Script** module selected, in the **Properties** pane, replace the existing Python script with the code from **VisResiduals.py.**
9. Save and run the experiment. Then, when the experiment is finished, visualize the **Python device port** of the **Execute Python Script** module.
10. Review the scatter plot that shows **Heating Load** against residuals conditioned by **Overall Height**.
11. Review the conditioned scatter plots that have been created.
12. Review the created histograms.
13. Review the pair of Q-Q normal plots.
14. Close the Python device output.
15. Visualize the **Result Dataset** output of the **Execute Python Script** module, and review the root squared mean error results returned by the **rsme** function, as shown in the following image.
16. Can you draw any conclusions from the results?

Summary

In this lab, you have constructed and evaluated a nonlinear regression model. Highlights from the results of this lab are:

- The nonlinear regression model fits the building energy efficiency data rather well. The residual structure is improved when compared to the linear regression model used in the Module 3 labs.
- The nonlinear model requires only a small feature set to achieve these results.
- Using the Tune Model Hyperparameters module improved model performance.

- Cross validation indicates the model generalizes well.