Data Science and Machine Learning Essentials

Lab 4A – Working with Regression Models

By Stephen Elston and Graeme Malcolm

Overview

In this lab, you will continue to learn how to construct and evaluated regression machine learning models using Azure ML and R or Python. If you intend to work with R, complete the *Evaluating Model Errors with R* exercise. If you plan to work with Python, complete the *Evaluating Model Errors with Python* exercise. Unless you need to work in both languages, you do not need to try both exercises.

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 which produces improved predictions of building energy efficiency.

Note: This lab builds on the experiment you completed in Lab 3C. If you have not completed Lab 3C, you can copy the experiment from the Cortana Analytics Gallery.

What You'll Need

To complete this lab, you will need the following:

- An Azure ML account
- A web browser and Internet connection
- Python Anaconda or R and RStudio
- The lab files for this lab

Note: To set up the required environment for the lab, follow the instructions in the **Setup** document for this course. Then download and extract 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 Sweep Parameters module improved model performance.
- Cross validating the model to ensure it generalizes well.

- Evaluating the performance of the model in depth with R or Python.

Build a New Model

In this procedure, you will construct and evaluate a nonlinear regression model.

In module 3 you performed the following tasks:

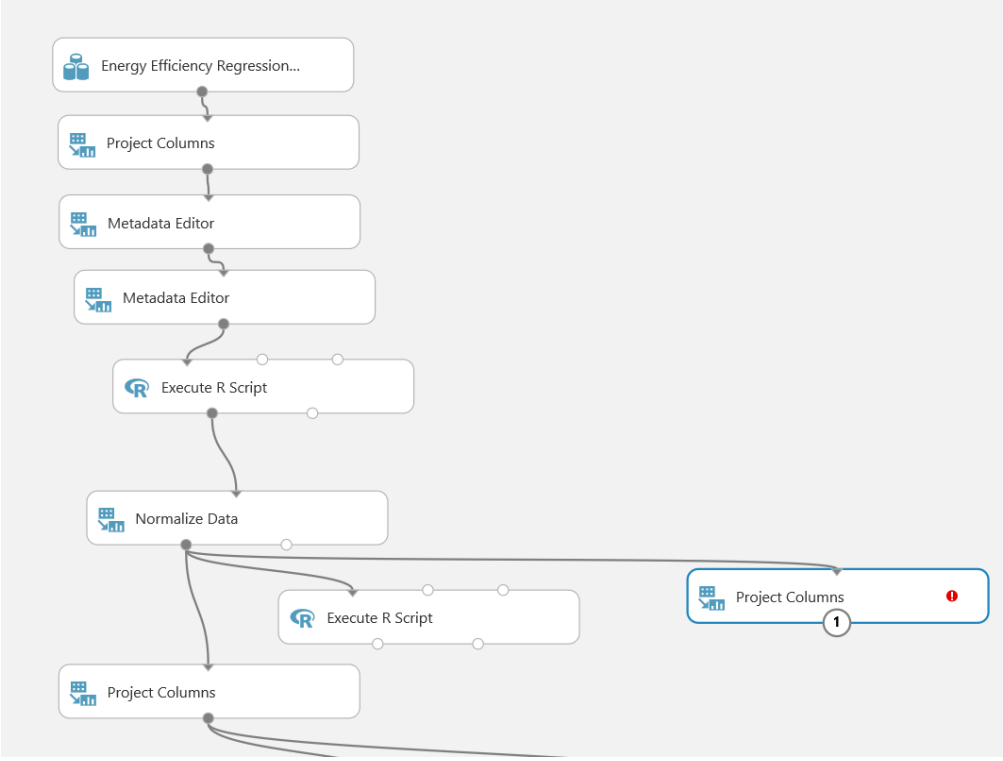
- Performed visualization of the data set to gain an understanding of the relationships between the features and the label.
- Build and evaluated a machine learning model using the linear regression method.
- Performed feature engineering for the linear regression model

In the module 3 labs, you should have noted that there was considerable structure in the residuals (errors). Further, feature pruning did not improve this situation. You should therefore conclude that there are fundamentally nonlinear relationships between the features and the labels in this data set.

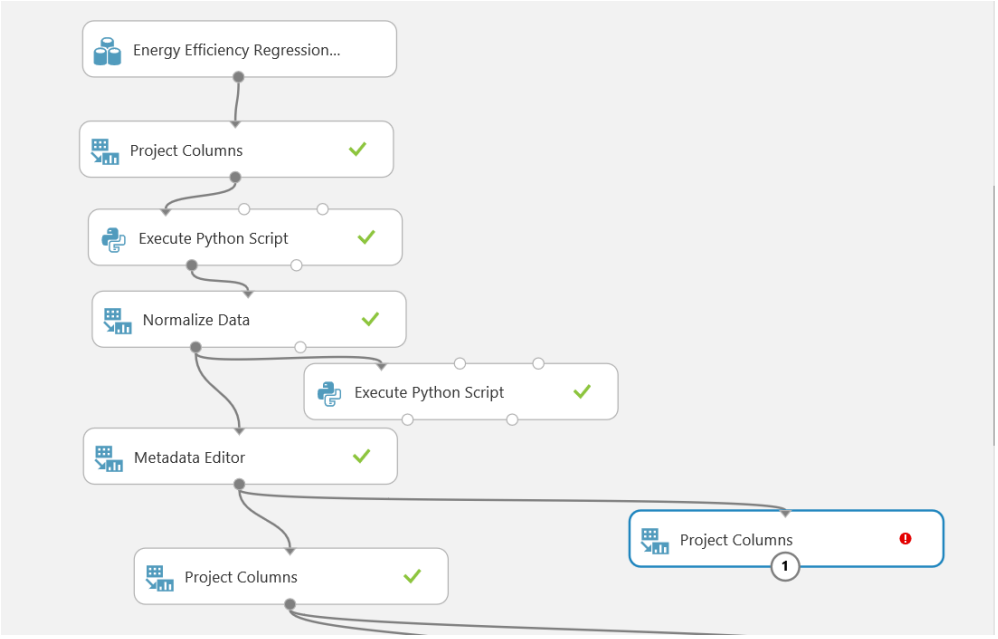
You will now construct and evaluated a nonlinear regression model. There are a number of nonlinear regression models supported in Azure ML. In this case, you will work with the **Decision Forest Regression** method. This approach uses majority voting among an ensemble (group) of regression tree models. 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:

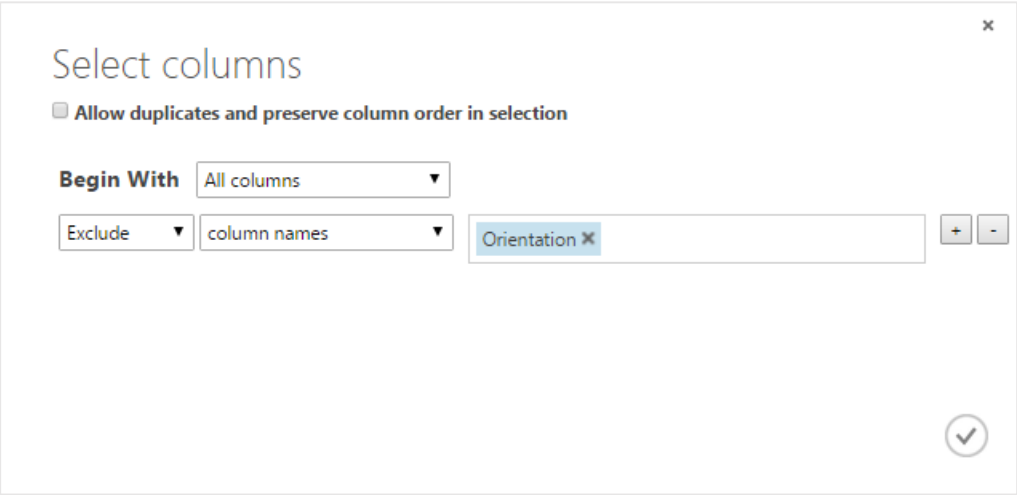
1. If you have not already done so, open a browser and browse to https://studio.azureml.net. Then sign in using the Microsoft account associated with your Azure ML account.
2. If you prefer to work with R, open the **Evaluation (R)** experiment, and save a copy as **Regression (R)**. Otherwise, save a copy of the **Evaluation (Python)** experiment as **Regression (Python)**. If you did not complete Lab 3C, you can copy the appropriate **Evaluation** experiment from the collection for this course in the Cortana Analytics Gallery at http://gallery.cortanaanalytics.com/Collection/5bfa7c8023724a29a41a4098d3fc3df9.
3. Search for the **Project Columns** module and drag it onto the canvas. Then, depending on the scripting language you are using, make the following connections:
    - If you are using R, connect the **Transformed dataset** output port of the **Normalize Data** module to the **Dataset** input port of the **Project Columns** module. Including the data preparation steps (upper part) of your experiment, it should resemble the diagram below:



- If you are using Python, connect the **Results dataset** output port of the first **Metadata Editor** module to the **Dataset** input port of the **Project Columns** module. Including the data preparation steps (upper part) of your experiment, it should resemble the diagram below:



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



5. Search for the **Split** module. Drag this module onto your experiment canvas. Connect the **Results dataset** output port of the **Project Columns** module to the **Dataset** input port of the **Split** module. Set the **Properties** of the **Split** 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

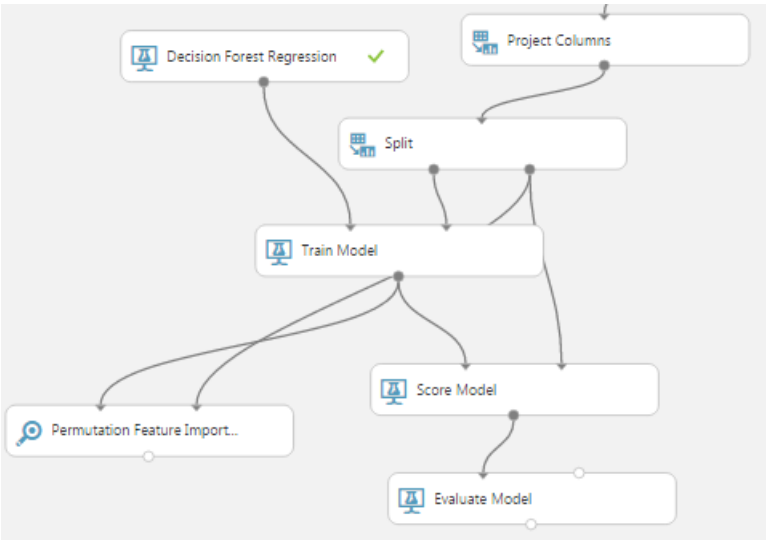
6. Search for the **Decision Forest Regression** module. Make sure you have selected the regression model version of this algorithm. Drag this module onto the canvas. Set the properties of this module as follows:
    - **Resampling method:** Bagging
    - **Create trainer mode:** Single Parameter
    - **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** 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** 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** 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 hand **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.

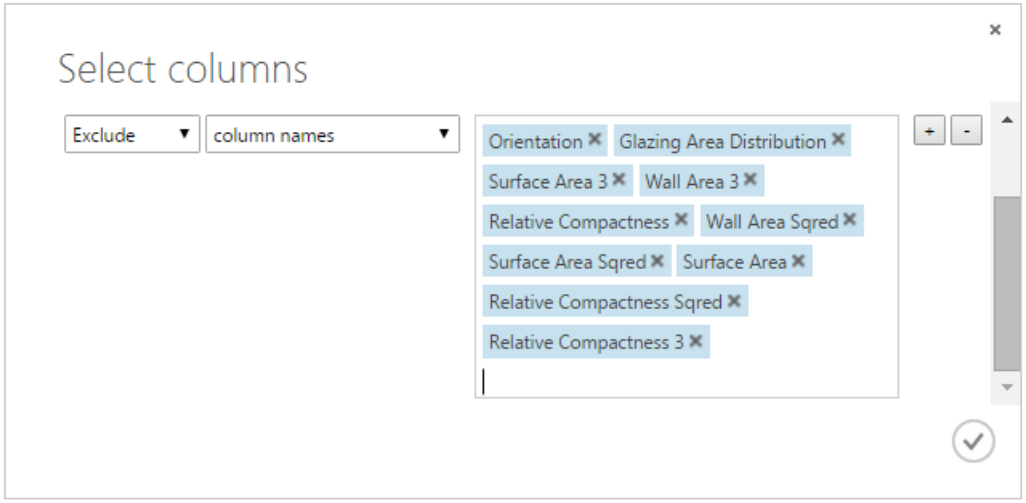
    Overall, these statistics are promising. The **Coefficient of Determination** is a measure of the reduction in the variance, between the raw label and the model error; squared error. This static is often referred to as R^2. A perfect model would have a **Coefficient of Determination** of 1.0, all the variance in the label is explained in the model. **Relative Squared Error** is the ratio of the variance or squared error of the model divided by the variance of the data. A perfect model would have a **Relative Squared Error** of 0.0, all model errors are zero. You should observe that these results from the nonlinear model are an improvement over those achieved with the linear model.

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 **Project Columns** 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 on 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 you model will not generalize well to new input values when the model is placed in production.
5. Select the **Project Columns** 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 **Project Columns** module to exclude the following features which do not change the model accuracy metrics significantly:
    - **Orientation**
    - **Glazing Area Distribution**
    - **Surface Area 3**
    - **Relative Compactness Sqred**
    - **Wall Area 3**
    - **Wall Area Sqred**
    - **Surface Area Sqred**
    - **Surface Area**
    - **Relative Compactness**
    - **Relative Compactness 3**

    At this point the **Column Selector** of the **Project Columns** module should be set to exclude the columns shown below:



    At the end of the pruning process, you are left with the following four features:
    - **Overall Height**
    - **Wall Area**
    - **Glazing Area**
    - **Roof Area**

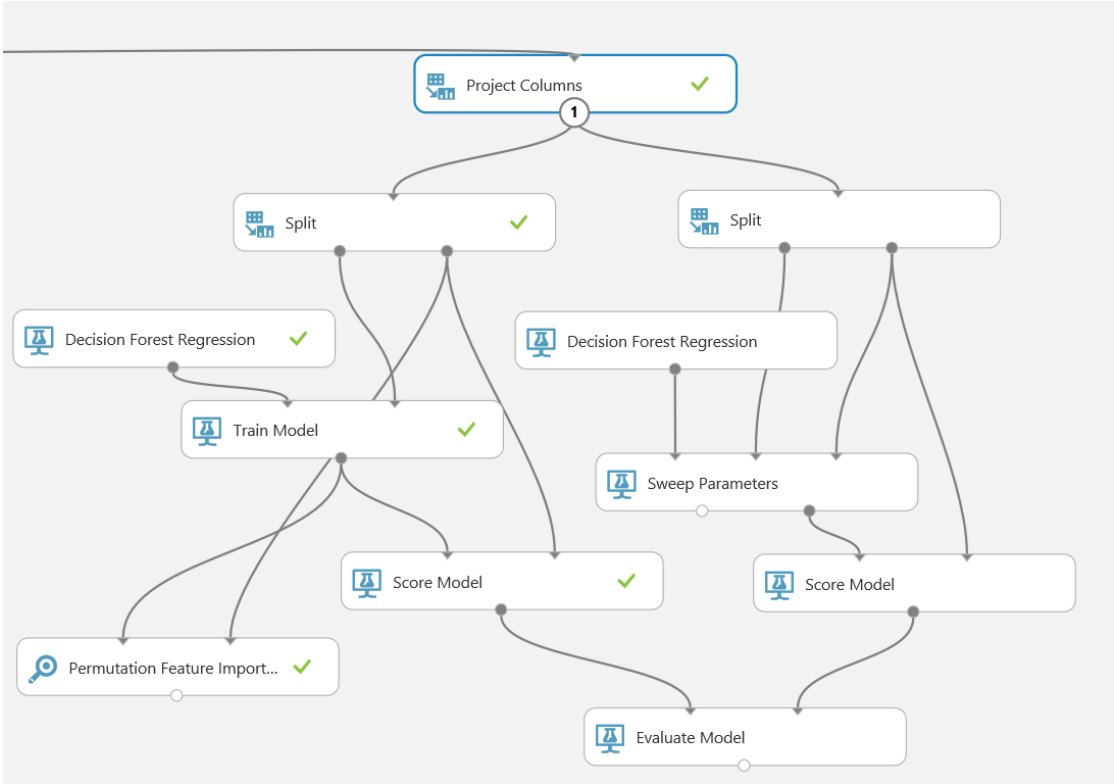
    Removing any of these features will cause the accuracy metrics to be degraded. Evidently, these are all you need for good model performance.

Sweep Parameters to Improve the Model

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

1. Select all of the modules below the **Project Columns** 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 **Project Columns** module to the **Dataset input** of the new **Split** module.
3. Remove the copied **Permutation Feature Importance**, **Train Model**, and **Evaluate Model** modules.
4. Search for the **Sweep Parameters** 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 **Sweep Parameters** module.
6. Connect the **Results dataset1** (left) output port of the new **Split** module to the **Training dataset** (middle) input port of the **Sweep Parameters** module.
7. Connect the **Results dataset2** (right) output port of the **Split** module to the **Optional test dataset** (right) input port of the **Sweep Parameters** module.
8. Connect the **Trained model** (right) output of the **Sweep Parameters** module to the **Trained model** (left) input of the new **Score Model** module.
9. Click the **Sweep Parameters** 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:



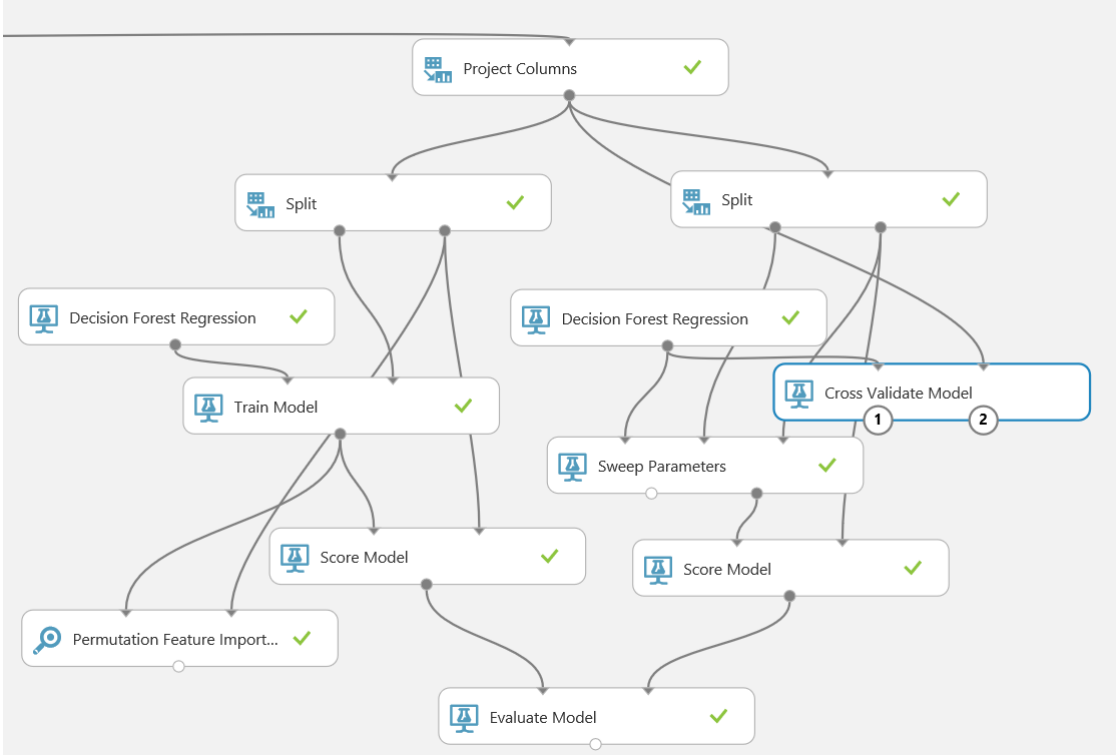
11. Save and run the experiment. When the experiment is finished, visualize the **Evaluation results** output port of the **Evaluate Model** module and compare the **Coefficient of Determination** and **Relative Squared Error** values for the two models.

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 data set 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 which 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 **Sweep Parameters** module) to the **Untrained model** input port of the **Cross Validate Model** module.
3. Connect the **Results dataset** output port of the **Project Columns** 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, passed 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:

0	76	Microsoft.Analytics.Modules.Gemini.Dll.GeminiDecisionForestRegressor	92.411708	0.439912	0.678375	0.049186	0.004617	0.995383
1	77	Microsoft.Analytics.Modules.Gemini.Dll.GeminiDecisionForestRegressor	64032.767724	0.439278	0.648732	0.048122	0.003974	0.996026
2	77	Microsoft.Analytics.Modules.Gemini.Dll.GeminiDecisionForestRegressor	20969000.02908	0.401314	0.5712	0.042855	0.003152	0.996848
3	77	Microsoft.Analytics.Modules.Gemini.Dll.GeminiDecisionForestRegressor	54.76422	0.401664	0.632591	0.04506	0.004105	0.995895
4	76	Microsoft.Analytics.Modules.Gemini.Dll.GeminiDecisionForestRegressor	529026.692535	0.35277	0.491689	0.039822	0.002526	0.997474
5	77	Microsoft.Analytics.Modules.Gemini.Dll.GeminiDecisionForestRegressor	14544.838732	0.429284	0.614163	0.046027	0.0037	0.9963
6	77	Microsoft.Analytics.Modules.Gemini.Dll.GeminiDecisionForestRegressor	13530.205284	0.39687	0.615753	0.042467	0.003478	0.996522
7	77	Microsoft.Analytics.Modules.Gemini.Dll.GeminiDecisionForestRegressor	54.898889	0.383518	0.625334	0.040463	0.003649	0.996351
8	77	Microsoft.Analytics.Modules.Gemini.Dll.GeminiDecisionForestRegressor	49.755101	0.358023	0.473653	0.042664	0.002487	0.997513
9	77	Microsoft.Analytics.Modules.Gemini.Dll.GeminiDecisionForestRegressor	47.056368	0.391701	0.560291	0.043495	0.003142	0.996858
Mean	768	Microsoft.Analytics.Modules.Gemini.Dll.GeminiDecisionForestRegressor	2159043.341964	0.399433	0.591178	0.044016	0.003483	0.996517
Standard Deviation	768	Microsoft.Analytics.Modules.Gemini.Dll.GeminiDecisionForestRegressor	6611174.418862	0.030437	0.066661	0.003069	0.000677	0.000677

Notice that the **Relative Squared Error** and **Coefficient of Determination** values in the folds (along the two right most 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.

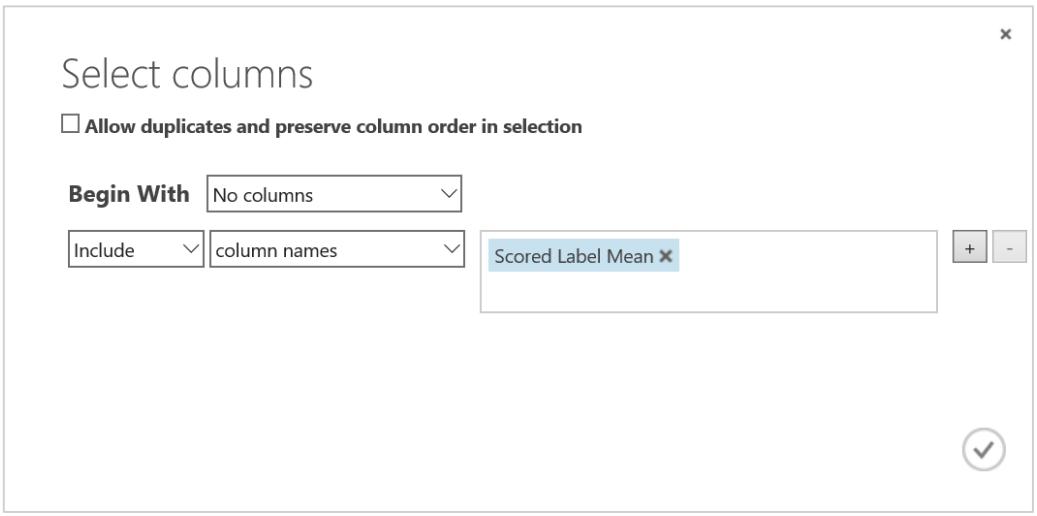
Evaluating Model Errors with R

In this exercise you will evaluate the model errors using custom R code.

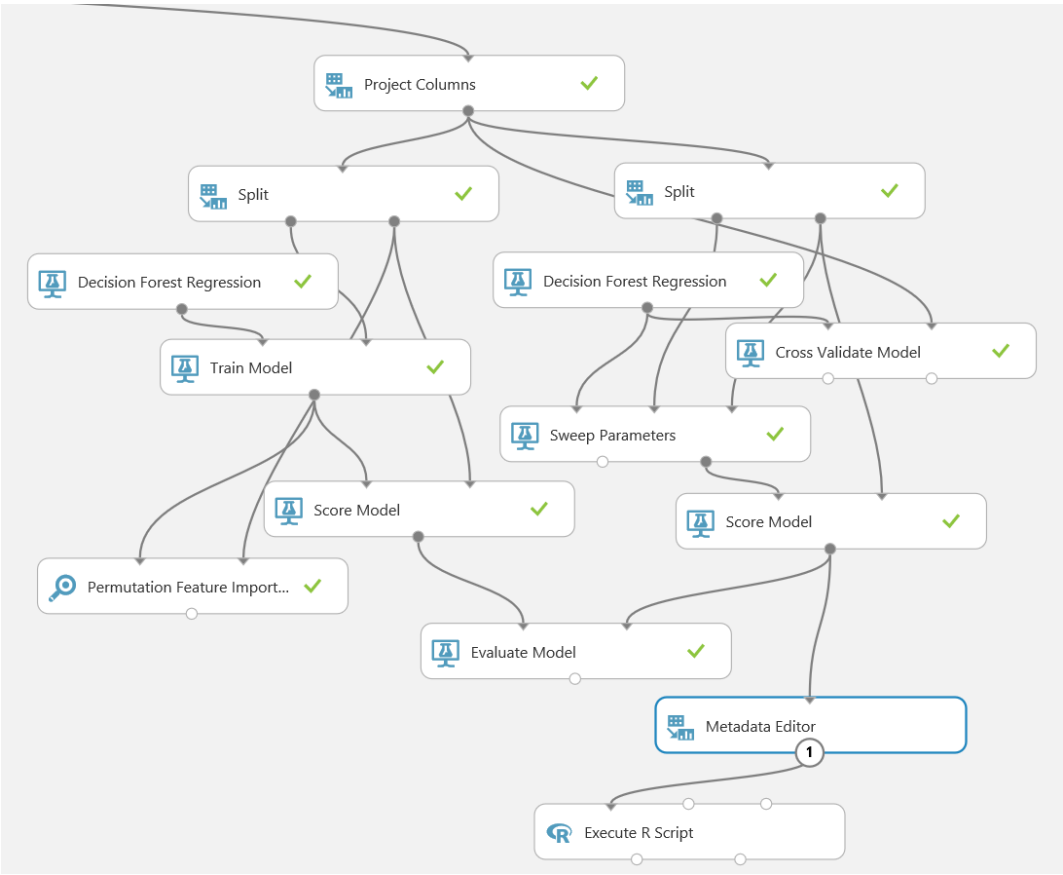
Note: If you prefer to work with Python, skip this exercise and complete the following exercise, *Evaluating Model Errors with Python*.

The summary performance statistics for the nonlinear **Decision Forest Regression** model look quite promising. However, summary statistics can hide some significant problems one needs to be aware of. To investigate the residuals, or model errors, you will use some custom R code. Much of this code is the same or similar to code you used for the labs in Module 3.

1. Search for the **Metadata Editor** module and drag it onto your canvas.
2. Connect the **Scored Dataset** output of the newest **Score Model** module (connected to the **Sweep Parameters** module) to the input of the **Metadata Editor** module.
3. Click the **Metadata Editor** module, and in the properties pane click **Launch Column Selector**. Choose the **Scored Label Mean** column as shown in the figure below.



4. In the **New column names** box type **ScoredLabels**, with no quotes. The output from this **Metadata Editor** model will now have a column name with no spaces, compatible with R data frame column names.
5. Search for the **Execute R Script** module, and drag it onto your canvas. Connect the **Results Dataset** output of the **Metadata Editor** module to the **Dataset1** (left) input of the **Execute R Script** module. Your experiment should resemble the figure below.



6. With the **Execute R Script** module selected, in the properties pane, replace the existing R script with the following code. You can copy this code from **VisResiduals.R** in the folder where you extracted the lab files:

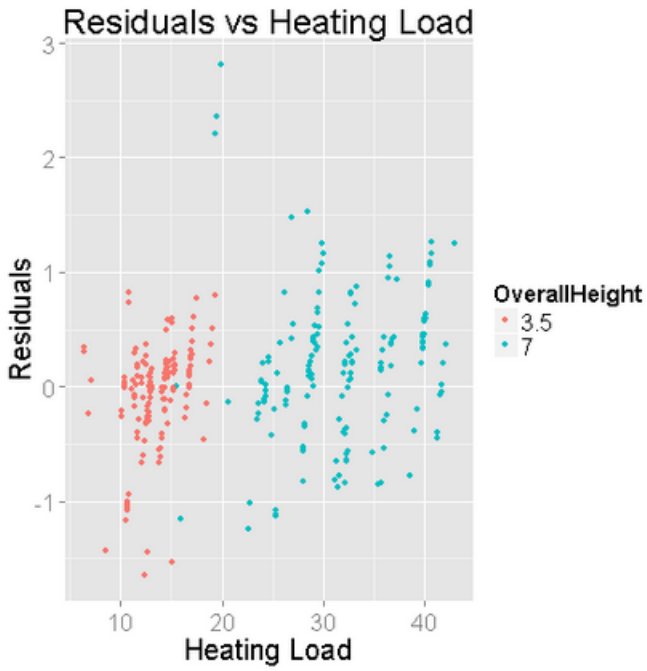
```
frame1 <- maml.mapInputPort(1)

frame1$Resids <- frame1$HeatingLoad - frame1$ScoredLabels

## Plot of residuals vs HeatingLoad.
library(ggplot2)
ggplot(frame1, aes(x = HeatingLoad, y = Resids , by =
OverallHeight)) +
  geom_point(aes(color = OverallHeight)) +
  xlab("Heating Load") + ylab("Residuals") +
  ggtitle("Residuals vs Heating Load") +
  theme(text = element_text(size=20))

  ## create some conditioned plots of the residuals
plotCols <- c("RelativeCompactnessSqred",
              "SurfaceArea",
              "GlazingArea")

plotEERes <- function(x){
  title <- paste("Residuals vs Heating Load conditioned by", x)
  facFormula <- paste("OverallHeight ~", x)
  ggplot(frame1, aes(Resids, HeatingLoad)) +
    geom_point() +
    facet_grid(facFormula) +
    ggtitle(title)
}

lapply(plotCols, plotEERes)

## Conditioned histograms of the residuals
ggplot(frame1, aes(Resids)) +
  geom_histogram(binwidth = 0.5) +
  facet_grid(. ~ OverallHeight) +
  ggtitle('Histogram of residuals conditioned by Overall Height')
+
  xlab('Residuals')

## Quantile-quantile normal plot of the residuals.
Resids35 <- frame1[frame1$OverallHeight == 3.5, ]$Resid
Resids7 <- frame1[frame1$OverallHeigh == 7, ]$Resid
par(mfrow = c(1,2))
qqnorm(Resids35)
qqnorm(Resids7)
par(mfrow = c(1,1))


rmse <- function(x){
  sqrt(sum(x^2)/length(x))
}
```

```
outFrame <- data.frame(
  rms_Overall = rmse(frame1$Resids),
  rms_35 = rmse(Resids35),
  rms_7 = rmse(Resids7))

maml.mapOutputPort('outFrame')
```

Tip: To copy code in a local code file to the clipboard, press **CTRL+A** to select all of the code, and then press **CTRL+C** to copy it. To paste copied code into the code editor in the Azure ML **Properties** pane, press **CTRL+A** to select the existing code, and then press **CTRL+V** to paste the code from the clipboard, replacing the existing code.

Note that this code is nearly identical to the model evaluation code already discussed in lab 3C.

7. Save and run and run the experiment. Then, when the experiment is finished, visualize the **R Device port** of the **Execute R Script** module.
8. Examine the scatter plot that shows heating load against residuals conditioned by **OverallHeight**, which should look similar to this:

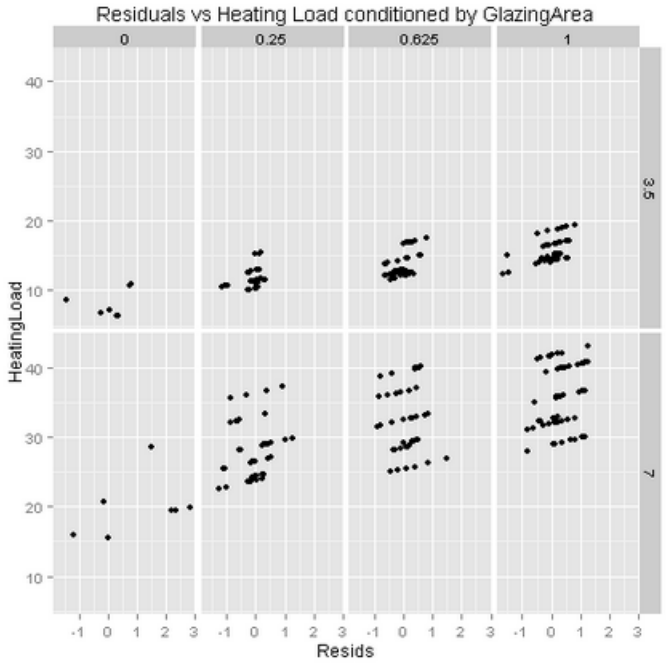


Examine the structure of these residuals with respect to the label, **Heating Load**. In an ideal case, the residuals should appear random with respect to the label **(Heating Load**). In fact, there is little structure in these residuals, and the distribution of these residuals does not change much with the value of the label.

If you compare this plot to the similar plot you created for the Module 3 labs, you can see that the linear structure in the residuals has disappeared. Further, the dispersion of the residuals is significantly reduced.

In summary, using a nonlinear regression model fits these data well. The residuals are reasonably well behaved.

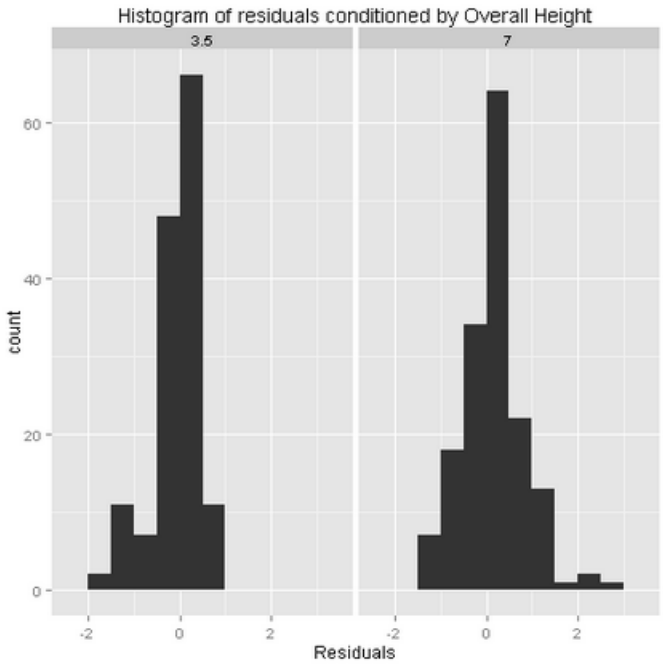
9. Review the conditioned scatter plots have been created. For example, look in detail at the scatter plot conditioned on **GlazingArea** and **OverallHeigh**t, as shown below.



Note the shaded conditioning level tiles across the top and right side of this chart. The four tiles across the top (horizontal) show the four levels (unique values) of **GlazingArea**. The two tiles on the right (vertical axis) show the two levels (unique values) of **OverallHeight**. Each scatter plot shows the data falling into the group by **GlazingArea** and **OverallHeight**, with the label (**Heating Load**) on the vertical axis and the residuals on the horizontal axis.

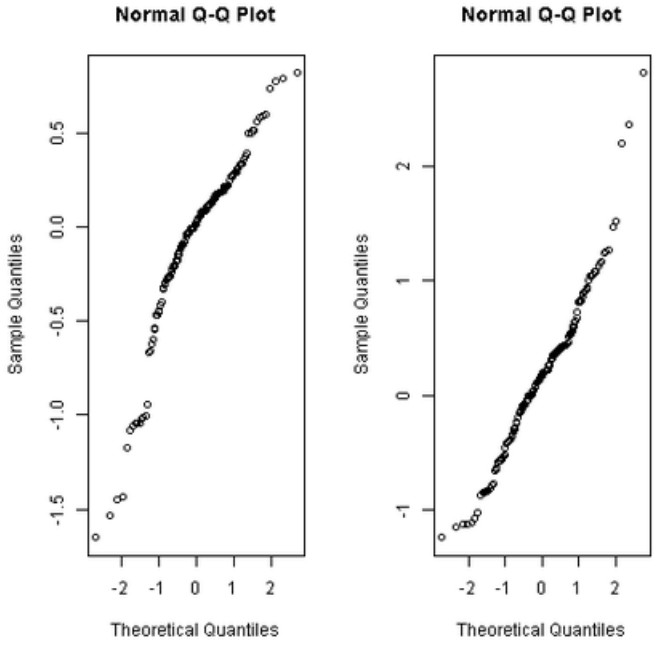
Examine this plot and notice that the residuals are not random across these plots. There is a slight linear structure visible in these subplots. However, there is not a notable change in the distribution of the residuals across the subplots. Further, the range of residual values is much less than for the linear regression model used in Module 3. These observations confirm that the nonlinear regression model is working reasonably well.

10. Examine the histogram, as shown below:



Examine these results, and note the differences in the histograms by **OverallHeight**. Further, there are some small outliers for the **OverallHeight** of 7. However, the range of these residuals is not great. These residuals are much reduced when compared to the equivalent pair of histograms discussed in Module 3. Again, we can conclude that the nonlinear model is working well.

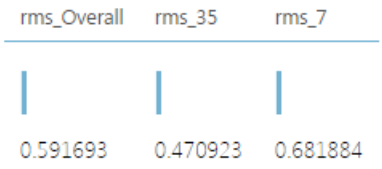
11. Review the pair of Q-Q normal plots, as shown below:



Note: A Q-Q normal plot uses the quantiles of a theoretical Normal distribution on the horizontal axis vs. the quantiles of the residuals on the vertical axis. For an ideal linear model, the residuals will be normally distributed and fall close to a straight line.

The data shown on both of this plots deviates from straight lines. Further, some outliers are noticeable. When compared to the plots for the linear model created for the Module 3 labs, these plots show improvement. Primarily, the range of the outliers is much reduced. Again, we can conclude that the nonlinear model is working well.

12. Close the R Device output.
13. Visualize the **Result Dataset** output of the **Execute R Script** module, and review the root squared mean error results returned by the **rsme** function, as shown below:



Compare these results to those obtained in Module 3. All three measures of RMS error have been reduced. Further the relative difference between **OverallHeight** of 3.5 and **OverallHeight** of 7 are reduced.

Using a nonlinear regression model has worked well for this problem. The residual measures are all satisfactory.

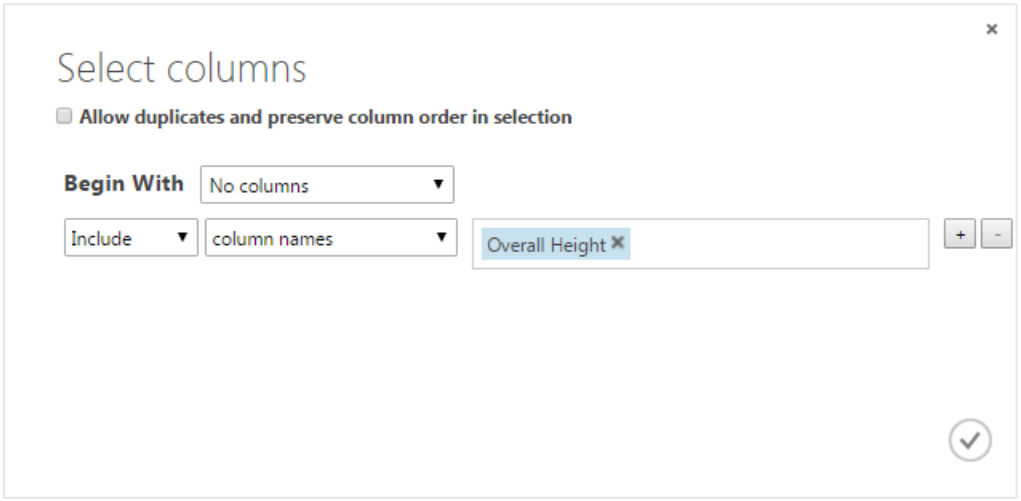
Evaluating Model Errors with Python

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

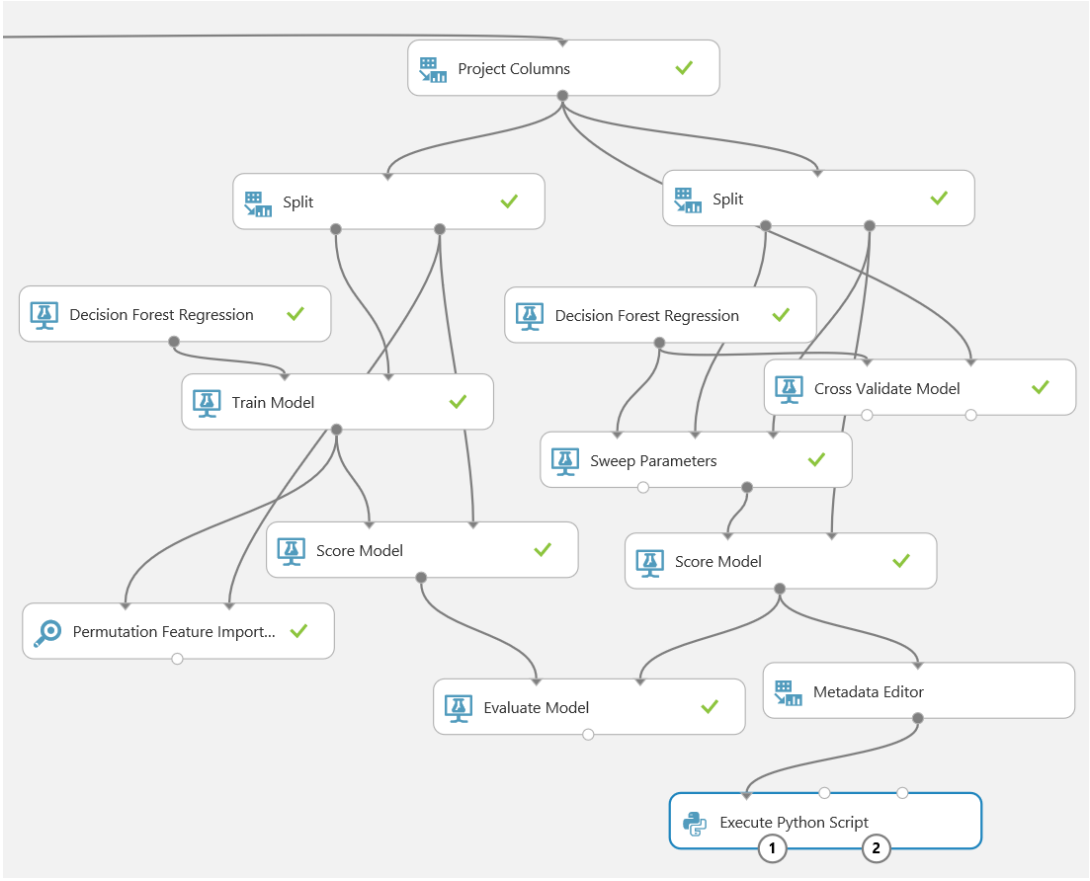
Note: If you prefer to work with R, complete the exercise *Evaluating Model Errors with R*.

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. Much of this code is the same or similar to code you used for the labs in Module 3.

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



4. 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.
5. Search for and locate the **Execute Python Script** module. Drag this module onto your canvas.
6. Connect the **Results Dataset** output of the **Metadata Editor** module to the **Dataset1** (left) input of the **Execute Python Script** module. Your experiment should resemble the following the figure below:



7. With the **Execute Python Script** module selected, in the properties pane, replace the existing Python script with the following code. You can copy this code from **VisResiduals.py** in the folder where you extracted the lab files:

```
def rmse(Resid):
    import numpy as np
    resid = Resid.as_matrix()
    length = Resid.shape[0]
    return np.sqrt(np.sum(np.square(resid)) / length)

def azureml_main(frame1):
    # Set graphics backend
    import matplotlib
    matplotlib.use('agg')

    import pandas as pd
    import pandas.tools.rplot as rplot
    import matplotlib.pyplot as plt
    import statsmodels.api as sm

## Compute the residuals
    frame1['Resids'] = frame1['Heating Load'] - frame1['Scored
Label Mean']

    ## Create data frames by Overall Height
    temp1 = frame1.ix[frame1['Overall Height'] == 7]
    temp2 = frame1.ix[frame1['Overall Height'] == 3.5]

## Create a scatter plot of residuals vs Heating Load.
    fig1 = plt.figure(1, figsize=(9, 9))
    ax = fig1.gca()
    temp1.plot(kind = 'scatter', x = 'Heating Load', \
                y = 'Resids', c = 'DarkBlue',
                alpha = 0.3, ax = ax)
    temp2.plot(kind = 'scatter', x = 'Heating Load', \
                y = 'Resids', c = 'Red', alpha = 0.3, ax = ax)
    ax.set_title('Heating load vs. model residuals')
    plt.show()
    fig1.savefig('plot1.png')

    ## Scatter plots of the residuals conditoned by
## several features.
    col_list = ["Relative Compactness Sqred",
              "Surface Area",
              "Glazing Area"]

    for col in col_list:
        ## First plot one value of Overall Height.
        fig = plt.figure(figsize=(10, 5))
        fig.clf()
        ax = fig.gca()
        plot = rplot.RPlot(temp1, x = 'Heating Load',
                                  y = 'Resids')
```

```
        plot.add(rplot.GeomScatter(alpha = 0.3,
                                   colour = 'DarkBlue'))
        plot.add(rplot.TrellisGrid(['.', col]))
        ax.set_title('Residuals by Heating Load and height = 7
conditioned on ' + col + '\n')
        plot.render(plt.gcf())
        fig.savefig('scater_' + col + '7' + '.png')

        ## Now plot the other value of Overall Height.
        fig = plt.figure(figsize=(10, 5))
        fig.clf()
        ax = fig.gca()
        plot = rplot.RPlot(temp2, x = 'Heating Load',
                                  y = 'Resids')
        plot.add(rplot.GeomScatter(alpha = 0.3, colour = 'Red'))
        plot.add(rplot.TrellisGrid(['.', col]))
        ax.set_title('Residuals by Heating Load and height = 3.5
conditioned on ' + col + '\n')
        plot.render(plt.gcf())
        fig.savefig('scater_' + col + '3.5' + '.png')

## Histograms of the residuals
    fig4 = plt.figure(figsize = (12,6))
    fig4.clf()
    ax1 = fig4.add_subplot(1, 2, 1)
    ax2 = fig4.add_subplot(1, 2, 2)
    ax1.hist(temp1['Resids'].as_matrix(), bins = 40)
    ax1.set_xlabel("Residuals for Overall Height = 3.5")
    ax1.set_ylabel("Density")
    ax1.set_title("Histogram of residuals")
    ax2.hist(temp2['Resids'].as_matrix(), bins = 40)
    ax2.set_xlabel("Residuals of model")
    ax2.set_ylabel("Density")
    ax2.set_title("Residuals for Overall Height = 7")
    fig4.savefig('plot4.png')

## QQ Normal plot of residuals
    fig3 = plt.figure(figsize = (12,6))
    fig3.clf()
    ax1 = fig3.add_subplot(1, 2, 1)
    ax2 = fig3.add_subplot(1, 2, 2)
    sm.qqplot(temp1['Resids'], ax = ax1)
    ax1.set_title('QQ Normal residual plot \n with Overall Height
= 3.5')
    sm.qqplot(temp2['Resids'], ax = ax2)
    ax2.set_title('QQ Normal residual plot \n with Overall Height
= 7')
    fig3.savefig('plot3.png')
```

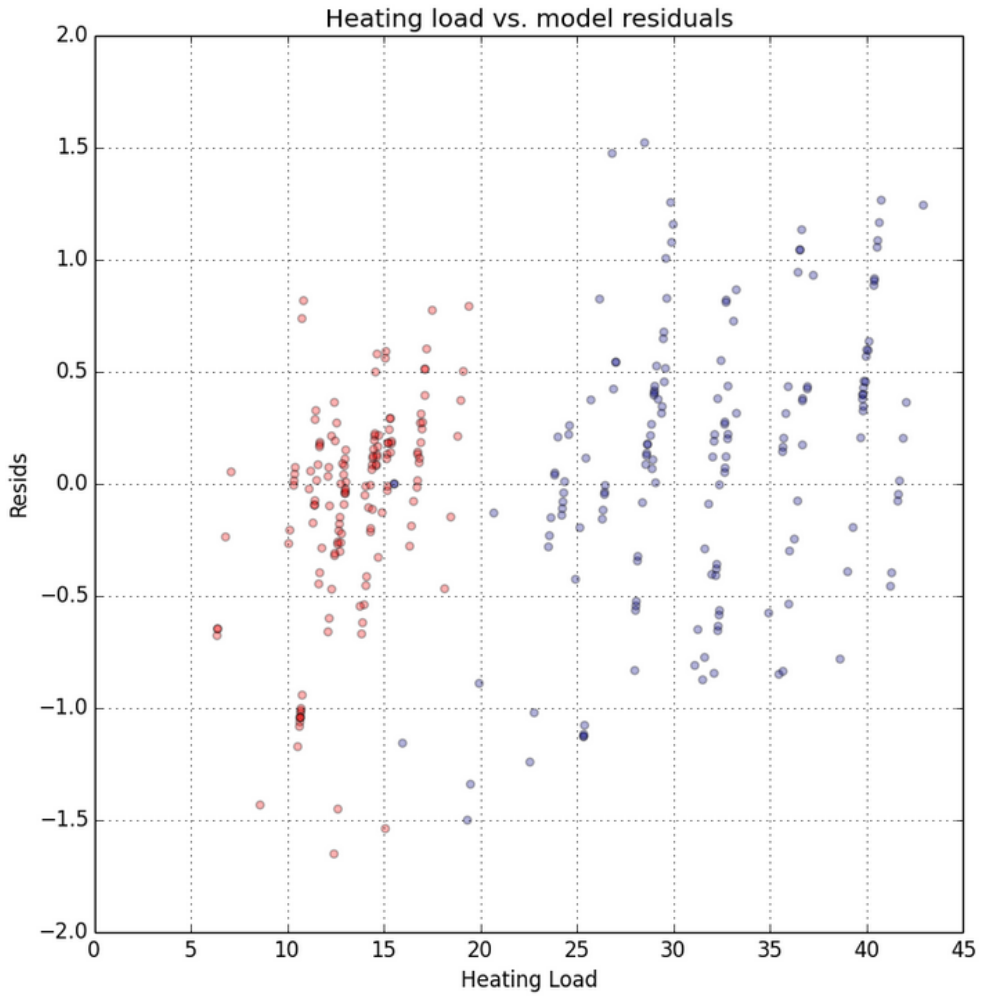
```
out_frame = pd.DataFrame({ \
  'rmse_Overall' : [rmse(frame1['Resids'])], \
  'rmse_35Height' : [rmse(temp1['Resids'])], \
  'rmse_70Height' : [rmse(temp2['Resids'])] })

return out_frame
```

Tip: To copy code in a local code file to the clipboard, press **CTRL+A** to select all of the code, and then press **CTRL+C** to copy it. To paste copied code into the code editor in the Azure ML **Properties** pane, press **CTRL+A** to select the existing code, and then press **CTRL+V** to paste the code from the clipboard, replacing the existing code.

WARNING!: Ensure you have a Python return statement at the end of your azureml_main function; for example, return frame1. Failure to include a return statement will prevent your code from running and may produce an inconsistent error message.

Note that most details of this code are described in the labs for Module 3. The predicted value column is now called **Scored Label Mean.**

8. Save and run the experiment. Then, when the experiment is finished, visualize the **Python device port** of the **Execute Python Script** module.
9. Experiment the scatter plot that shows **Heating Load** against residuals conditioned by **Overall Height**, which should look similar to this figure:

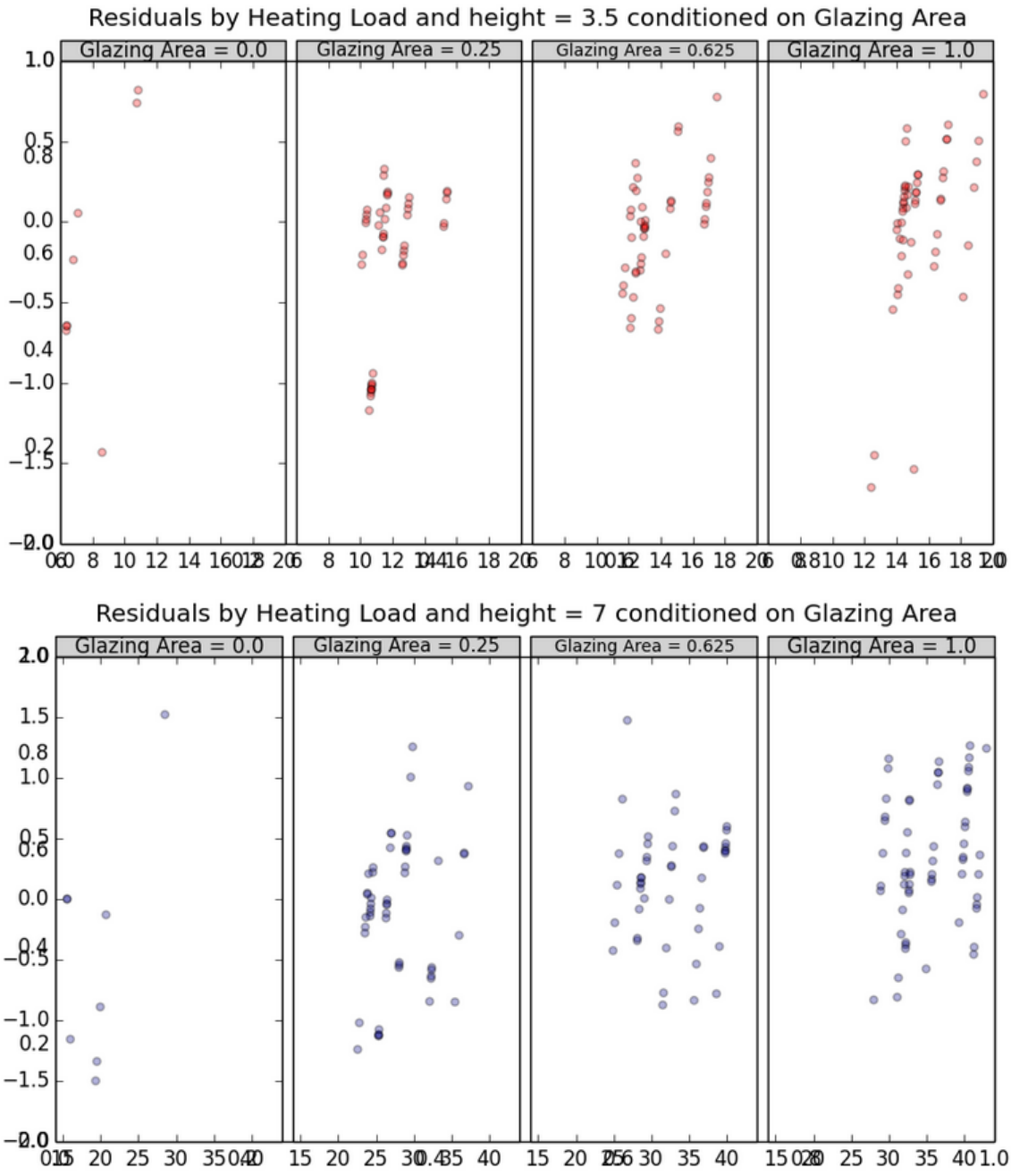


Examine the structure of these residuals with respect to the label, Heating Load. In an ideal case, the residuals should appear random with respect to the label (**Heating Load**). In fact, there is little structure in these residuals, and the distribution of these residuals does not change much with the value of the label.

If you compare this plot to the similar plot you created for the Module 3 labs, you can see that the linear structure in the residuals has disappeared. Further, the dispersion in the residuals is significantly reduced.

In summary, using a nonlinear regression model fits these data well. The residuals are reasonably well behaved.

10. Review the conditioned scatter plots have been created. For example, look in detail at the scatter plots by **Overall Height** and conditioned on **Glazing Area**, as shown below.

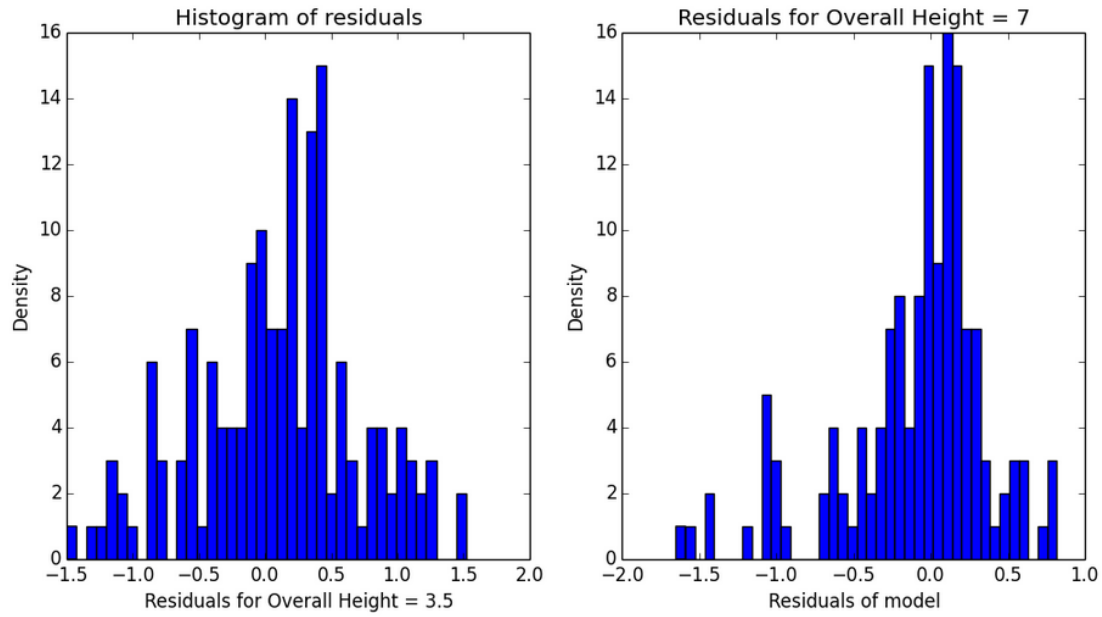


There is a pair of conditioned scatter plots; one for **Overall Height** of 7 and one for **Overall Height** of 3.5. Note the shaded conditioning level tiles across the top of these charts showing the four levels (unique values) of **Glazing Area**. Each scatter plot shows the data falling into the group by **Glazing Area** and **Overall Height**, with the label (**Heating Load**) on the vertical axis and the Residuals on the horizontal axis.

Examine this plot and notice that the residuals are not completely random across these plots. There is a slight linear structure visible in these subplots. However, there is not a notable change in the distribution of the residuals across the subplots. Further, the range of residual values is

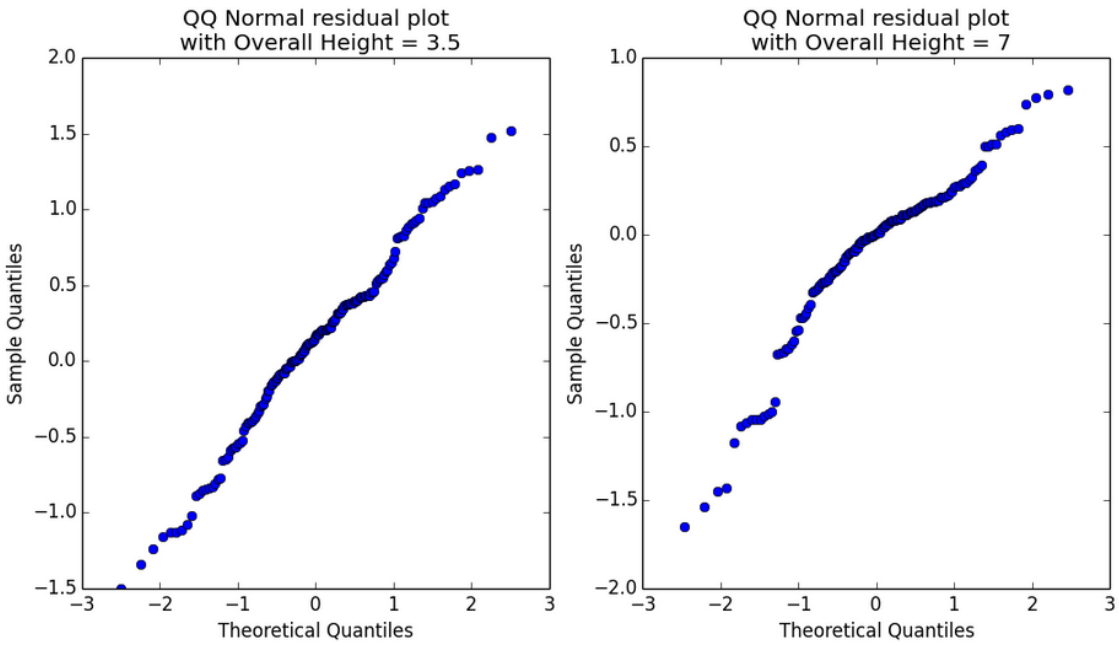
much less than for the linear regression model used in Module 3. These observations confirm that the nonlinear regression model is working reasonably well.

11. Examine the histogram, as shown below:



Examine these results, and note the differences in the histograms by **Overall Height**. Further, there are some small outliers for the **Overall Height** of 7. However, the range of these residuals is not great. These residuals are much reduced when compared to the equivalent pair of histograms discussed in Module 3. Again, we can conclude that the nonlinear model is working well.

12. Review the pair of Q-Q normal plots, as shown below:



Note: A Q-Q normal plot uses the quantiles of a theoretical Normal distribution on the horizontal axis vs. the quantiles of the residuals on the vertical axis. For an ideal linear model, the residuals will be normally distributed and fall close to a straight line.

The data shown on both of this plots deviates from straight lines. Further, some outliers are noticeable. When compared to the plots for the linear model created for the Module 3 labs, these plots show improvement. Primarily, the range of the outliers is much reduced. Again, we can conclude that the nonlinear model is working well.

13. Close the Python device output.

14. 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 below:

Compare these results to those obtained in Module 3. All three measures of RMS error have been reduced. Further the relative difference between **Overall Height** of 3.5 and **Overall Height** of 7 are reduced.

Using a nonlinear regression model has worked well for this problem. The residual measures are all satisfactory.

Summary

In this lab you have constructed and evaluated a nonlinear regression model. Highlight 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 only requires a small feature set to achieve these results.
- Using the Sweep Parameters module improved model performance.
- Cross validation indicates the model generalizes well.

Note: The experiment created in this lab is available in the Cortana Analytics library at http://gallery.cortanaanalytics.com/Collection/5bfa7c8023724a29a41a4098d3fc3df9.