Data Science and Machine Learning Essentials

Lab 3A – Visualizing Data

By Stephen Elston and Graeme Malcolm

Overview

In this lab, you will learn how to use R or Python to visualize data. If you intend to work with R, complete the *Visualizing Data with R* exercise. If you plan to work with Python, complete the *Visualizing Data with Python* exercise. Unless you need to work in both languages, you do not need to try both exercises.

Note: This lab builds on knowledge and skills developed in the preceding labs in this course. If you have little experience with Azure ML, and you did not complete the previous labs, you are advised to do so before attempting this lab.

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
- Python Anaconda and Spyder or R and RStudio

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.

Visualizing Data with R

R includes some basic native data visualization functionality, and also supports the **ggplot2** library; which provides extensive graphical capabilities. This makes R a useful language with which to create visualizations of your data in order to explore relationships between the data fields and identify features that may be useful for predicting labels in machine learning projects.

In this exercise, you will use R to visualize data from a dataset of metrics for buildings. Specifically, you will try to identify data fields that influence the *heating load* of a building, which is a measure of its energy efficiency.

Note: If you prefer to work with Python, skip this exercise and complete the next exercise, *Visualizing Data with Python*.

Install the ggplot2 Library

1. Start RStudio, and close any open script files from previous sessions.
2. In the **Console** pane, enter the following command to install the **ggplot2** library. If you are prompted to use a personal library, click **Yes**.

```
install.packages('ggplot2', dep = TRUE)
```

Load the Dataset

1. In RStudio, open the **PrepEE.R** file in the folder where you extracted the lab files for this course.
2. In the **PrepEE.R** pane, in the following code, change ***C:\\DAT203xLabfiles*** to the path to the folder where you extracted the lab files for this course.

```
## Load the data
dirName <- 'C:\\DAT203xLabfiles'
fileName <- "EnergyEfficiencyRegressiondata.csv"
infile <- file.path(dirName, fileName)
eeframe <- read.csv( infile, header = TRUE, stringsAsFactors =
FALSE)

## Remove dots from column names.
names(eeframe) <- gsub("\\.", "", names(eeframe))

## Remove columns we are not going to use.
eeframe$CoolingLoad <- NULL

## Convert some columns to factors/categorical.
catList <- c("OverallHeight", "Orientation")
eeframe[, catList] <- lapply(eeframe[, catList],
                             function(x)
as.factor(as.character(x)))

## Scale the numeric features.
scaleList <- c("RelativeCompactness", "SurfaceArea",
               "WallArea", "RoofArea", "GlazingArea",
               "GlazingAreaDistribution")
eeframe[, scaleList] <- lapply(eeframe[, scaleList], function(x)
as.numeric(scale(x)))
```

3. Select the code listed above (with the modified path), and on the **PrepEE.R** toolbar, click **Run**. The code performs the following actions:
   a. Loads a data frame named **eeframe** with data from a text file named **EnergyEfficiencyRegressiondata.txt**.
   b. Cleans the data. Specifically:
      - In the column names, periods (".") are removed (spaces in column names are replaced with periods when a data frame is loaded in R).
      - The **Cooling Load** (now renamed to **CoolingLoad**) column is removed, because it contains similar data to the **Heating Load** column, and is not required for the visualizations you will create in this exercise.
      - The **Overall Height** and **Orientation** columns are converted to categorical features (their numerical values indicate categories rather than scalar measurements).

- The **Relative Compactness**, **Surface Area**, **Wall Area**, **Roof Area**, **Glazing Area**, and **Glazing Distribution** columns are scaled so they can be easily compared.

4. In the **Console** pane, enter the following command:

```
head (eeframe)
```

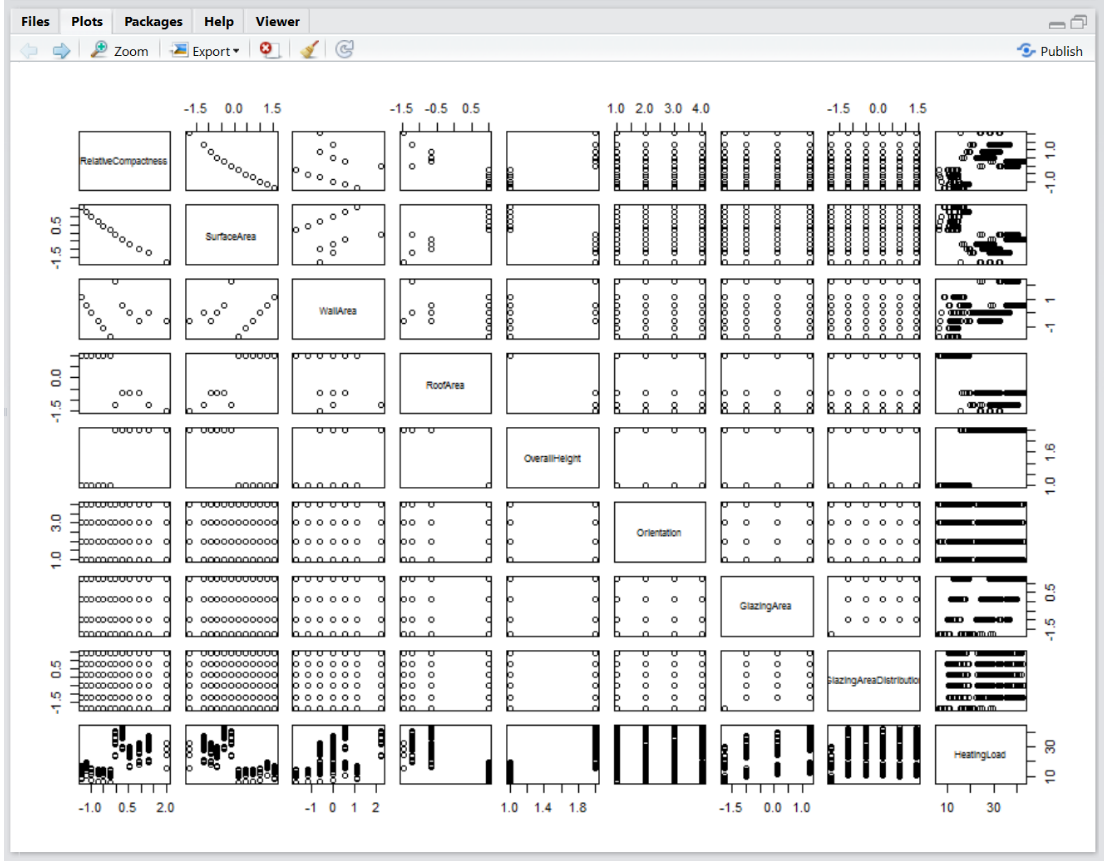
5. View the results, which show the first few rows of the data frame. Note that the data includes the following columns, which describe the physical attributes of a building and its Heating Load measurement:
   - **RelativeCompactness**
   - **SurfaceArea**
   - **WallArea**
   - **RoofArea**
   - **OverallHeight**
   - **Orientation**
   - **GlazingArea**
   - **GlazingAreaDistribution**
   - **HeatingLoad**

Create a Pair-Wise Scatter Plot

1. In RStudio, open the **VisualizeEE.R** file in the folder where you extracted the lab files for this course.
2. In the **VisualizeEE.R** pane, under the comment **## Use basic R graphics to create a pair-wise scatter plot**, select the following code.

```
Azure = FALSE
if(Azure){
  eeframe <- maml.mapInputPort(1)
  maml.mapOutputPort('eeframe')
}
pairs(~ ., data = eeframe)
```

3. With the code above selected, on the **VisualizeEE.R** toolbar, click **Run**. When running in Azure, the code loads data from the first input port of the **Execute R Script** module; but when running locally, it uses the data frame you loaded in the **PrepEE.r** script. The `pairs(~ ., data = eeframe)` statement creates a scatter plot matrix visualization that compares all columns in the dataset.
4. In the **Plots** pane, view the scatter plot matrix that has been generated, as shown below.



5. Note that this plot is comprised of a number of scatter plots. For each variable there is both a row and a column. The variable is plotted on the vertical axis in the row, and on the horizontal axis in the column. In this way, every combination of cross plots for all variables are displayed in both possible orientations. This type of plot allows you to examine the relationships between many variables in one view. The alternative of examining a large number of scatter plot combinations one at a time is both tedious and in all likelihood difficult since you need to remember relationships you have already viewed to understand the overall structure of the data.

   Some particular features you can notice include:

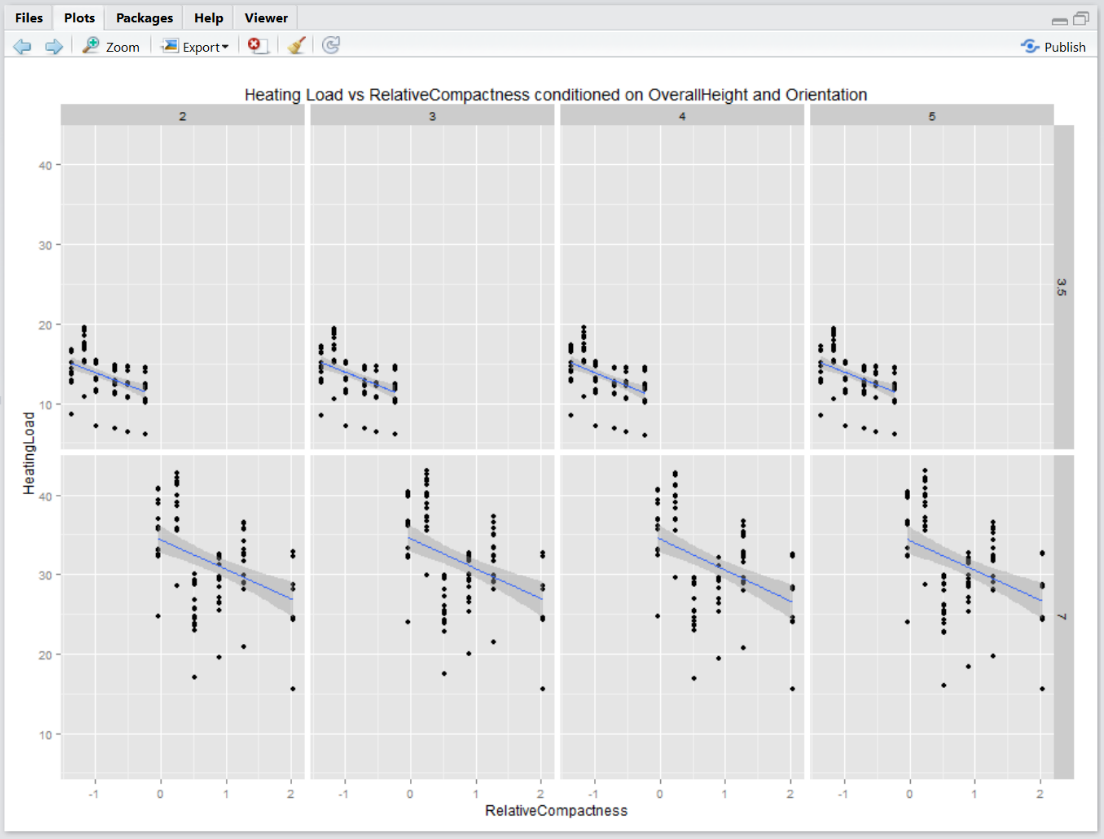
   - You can see plots of **RelativeCompactness** vs. **SurfaceArea**; second from left in the top row and second from top in the left most column. These two variable appear to be highly correlated. We may not want to use both features in a model.
   - Many of the other variables seem to cluster into two groups based on the value of **OverallHeight**.
   - **HeatingLoad** clusters into two groups for certain variables. For example, look at the cross plots between **HeatingLoad** and **RelativeCompactness** and **SurfaceArea.**

Create Conditioned Scatter Plots

1. In the **VisualizeEE.R** pane, under the comment **## Use ggplot2 to create conditioned scatter plots**, select the following code.

```
library(ggplot2)
plotCols <- c("RelativeCompactness",
              "SurfaceArea",
              "WallArea",
              "RoofArea",
              "GlazingArea",
              "GlazingAreaDistribution")
plotEE <- function(x){
  title <- paste("Heating load vs", x, "\n conditioned on
OverallHeight and Orientation")
  ggplot(eeframe, aes_string(x, "HeatingLoad")) +
    geom_point() +
    facet_grid(OverallHeight ~ Orientation) +
    ggtitle(title) +
    stat_smooth(method = "lm")
}
lapply(plotCols, plotEE)
```

2. With the code above selected, on the **VisualizeEE.R** toolbar, click **Run**. The code performs the following actions:
    - Loads the **ggplot2** library.
    - Creates scatter plot charts that compare **Heating Load** with:
        - **Relative Compactness**
        - **Surface Area**
        - **Wall Area**
        - **Roof Area**
        - **Glazing Area**
        - **Glazing Area Distribution**
    - Adds conditions for **Overall Height** and **Orientation** to each of these scatter plots.
3. In the **Plots** pane, use the **Previous Plot** and **Next Plot** toolbar buttons to view each of the scatter plots, as shown below.



4. Note that each chart includes eight scatter plots. There are four shaded tiles horizontally across the top, one for each level (unique value) of **Orientation**. Two shaded tiles, arranged vertically, on

the right represent the two levels (unique values) of **OverallHeight**. Each of these scatter plots has **RelativeCompactness** on the vertical (x) axis and **HeatingLoad** on the vertical (y) axis. The data are grouped by, or condition on **Overall Height** value (3.5 and 7) and **Orientation** value (2, 3, 4, and 5).

5. Examine this chart and note the following interesting features in these scatter plots:
    - The range of values of **HeatingLoad** are quite different between the upper and lower rows; **OverallHeight** of 3.5 and 7, respectively. In fact, there is very little overlap in these values, indicating that **OverallHeight** is an important feature in these data.
    - The distribution of these data does not change significantly with the levels of **`Orientation`**, indicating it is not a significant feature.
    - There is a notable trend of **HeatingLoad** with respect to **RelativeCompactness**, indicating **RelativeCompactness** is a significant feature.

Create Histograms

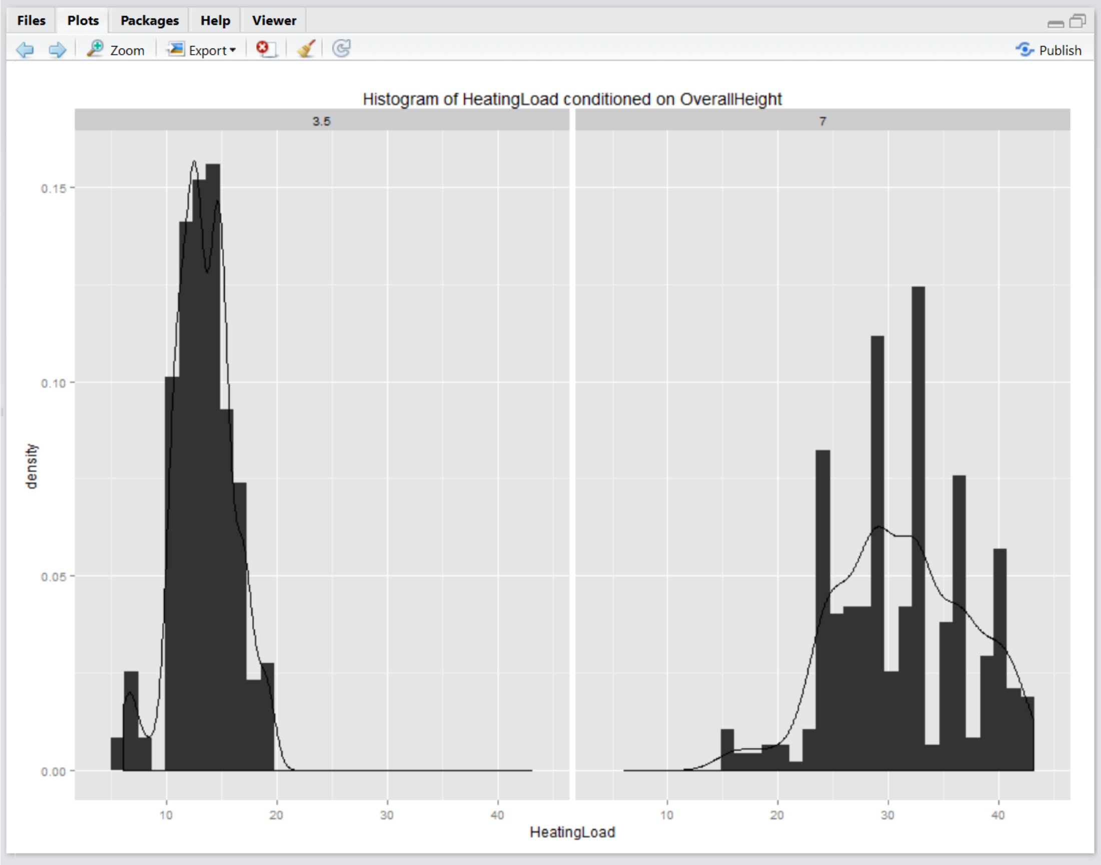
1. In the **VisualizeEE.R** pane, under the comment **## Create histograms**, select the following code.

```
plotCols4 <- c("RelativeCompactness",
               "SurfaceArea",
               "WallArea",
               "RoofArea",
               "GlazingArea",
               "GlazingAreaDistribution",
               "HeatingLoad")
library(gridExtra)
eeHist <- function(x) {
  title <- paste("Histogram of", x, "conditioned on
OverallHeight")
  ggplot(eeframe, aes_string(x)) +
    geom_histogram(aes(y = ..density..)) +
    facet_grid(. ~ OverallHeight) +
    ggtitle(title) +
    geom_density()
}
lapply(plotCols4, eeHist)
```

2. With the code above selected, on the **VisualizeEE.R** toolbar, click **Run**. The code performs the following actions:
    a. Creates histograms for each of the following columns:
        - **Relative Compactness**
        - **Surface Area**
        - **Wall Area**
        - **Roof Area**
        - **Glazing Area**
        - **Glazing Area Distribution**
        - **Heating Load**

    b. Adds a condition for **Overall Height** to each of these histogram.

    c. Adds a line to indicate density.
3. In the **Plots** pane, use the **Previous Plot** and **Next Plot** toolbar buttons to view each of the histograms, as shown below.



Note: A histogram plots the density of a distribution of the vertical axis, vs. bins of values on the horizontal axis. The values of a continuous variable are placed into one of several equal width bins. The density for each bin is the count of values in that bin. As the number or width of the bins changes, the details of the histogram will change. Histograms provide an empirical view of the distribution of the data being plotted.

4. Examine the pairs of histograms created. In most cases, the range of values on the horizontal azis are quite different for the two values of **Overall Height**; 7 and 3.5. A few of histogram pairs show little difference between the two values of **Overall Height**.

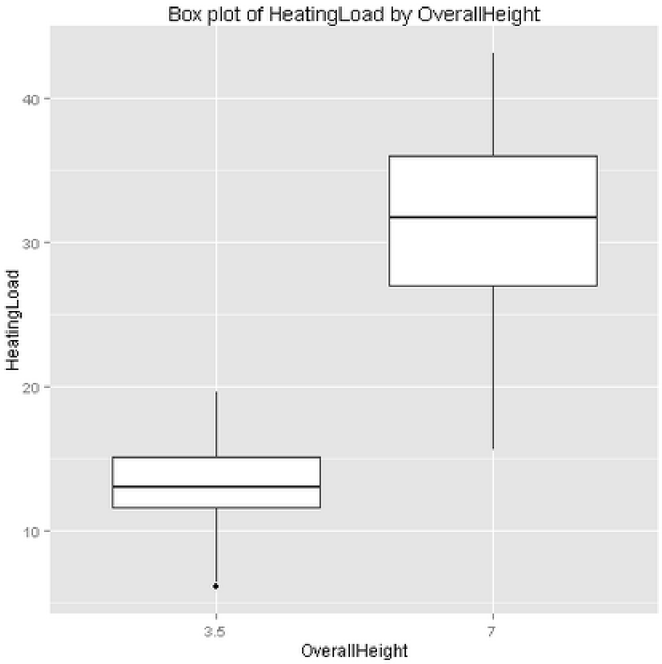
   Direct your attention to the histograms of **Heating Load**. Examine this chart and note how different the distribution of **Heating Load** is for the two values of **Overall Height**. In fact, there is very little overlap in the range of values for the two levels of **Overall Height**. Additionally, note the outliers in both distributions shown.

Create Box Plots

1. In the **VisualizeEE.R** pane, under the comment **## Create box plots**, select the following code.

```
eebox <- function(x) {
  title <- paste("Box plot of", x, "by OverallHeight")
  ggplot(eeframe, aes_string('OverallHeight', x)) +
    geom_boxplot() +
    ggtitle(title)
}
lapply(plotCols4, eebox)
```

2. With the code above selected, on the **VisualizeEE.R** toolbar, click **Run**. The code creates a box plot of each of the following columns conditioned on **OverallHeight**:
   - **Relative Compactness**
   - **Surface Area**
   - **Wall Area**
   - **Roof Area**
   - **Glazing Area**
   - **Glazing Area Distribution**
   - **Heating Load**
3. In the **Plots** pane, use the **Previous Plot** and **Next Plot** toolbar buttons to view each of the box plots, as shown below.



Note: The most prominent feature of a boxplot is the box. The box encloses the inner two quartiles of the distribution; first upper and first lower quartile. The line inside the box is placed at the median value. The lines or whiskers coming from the top and bottom of the box represent the outer upper and lower most quartiles of the distribution; second upper and second lower quartile. Any outliers are represented by symbols beyond the end of the end of the whiskers.

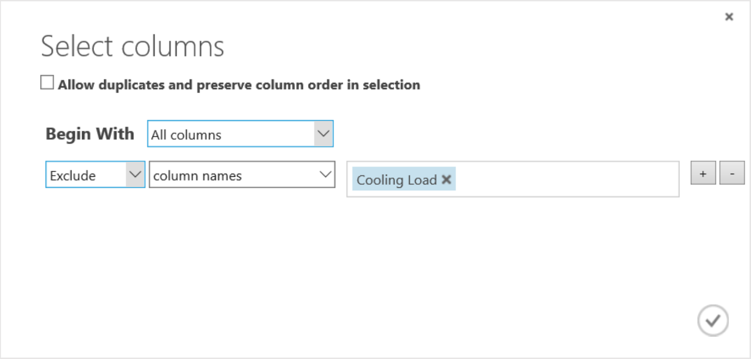
4. Examine the boxplots created for each of the variables. These boxplots show a different view of similar distribution information as the histograms.

   In most cases, there is little overlap in the values of the variable when grouped by the two levels (values) of **Overall Height**; 7 and 3.5. A few of boxplots show little difference between the two values of **Overall Height**.

   Direct your attention to the boxplot of **Heating Load**. Examine this chart and note how different the distribution of **Heating Load** is for the two values of **Overall Height**. In fact, there is very little overlap in the range of values for the two levels of **Overall Height**.

Create Visualizations in an Azure Machine Learning 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 a new blank experiment, and give it the title **Visualize Data (R)**.
3. Search for the **Energy Efficiency Regression data** dataset, and drag it to the experiment canvas.
4. Search for the **Project Columns** module and drag it to the canvas below the **Energy Efficiency Regression data** dataset. Then connect the output port from the **Energy Efficiency Regression data** dataset to the first input port of the **Project Columns** module.
5. Select the **Project Columns** module, and in the **Properties** pane, launch the column selector. Then configure the column selector to begin with all columns and exclude the **Cooling Load** column as shown in the following image.



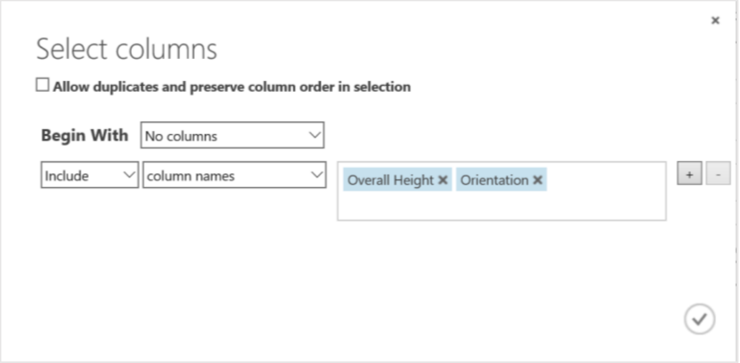
Select columns

☐ Allow duplicates and preserve column order in selection

Begin With: All columns

Exclude | column names | Cooling Load ✕

6. Search for the **Metadata Editor** module, and drag it to the canvas below the **Project Columns** module. Then connect the **Results dataset** output port from the **Project Columns** module to the input port of the **Metadata Editor** module.
7. Select the **Metadata Editor** module, and in the **Properties** pane, launch the column selector. Then configure the column selector to begin with no columns and include the **Overall Height** and **Orientation** column names as shown in the following image.



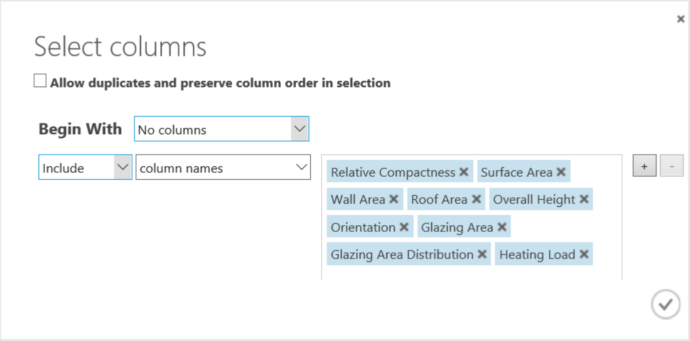
Select columns

☐ Allow duplicates and preserve column order in selection

Begin With: No columns

Include | column names | Overall Height ✕ Orientation ✕

8. With the **Metadata Editor** module selected, in the **Properties** pane, in the **Categorical** list, select **Make Categorical**.
9. Drag a second **Metadata Editor** module to the canvas below first one. Then connect the **Results dataset** output port from the first **Metadata Editor** module to the input port of the second one.
10. Select the second **Metadata Editor** module, and in the **Properties** pane, launch the column selector. Then configure the column selector to begin with no columns and include all column names as shown in the following image.



Select columns

□ Allow duplicates and preserve column order in selection

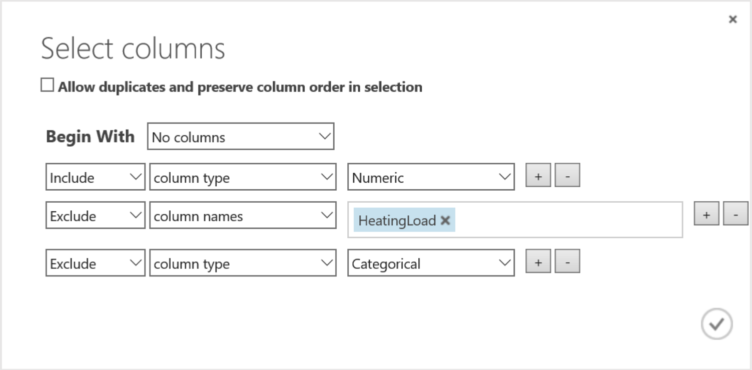
Begin With: No columns

Include | column names | Relative Compactness ✖ Surface Area ✖ Wall Area ✖ Roof Area ✖ Overall Height ✖ Orientation ✖ Glazing Area ✖ Glazing Area Distribution ✖ Heating Load ✖

11. With the second **Metadata Editor** module selected, in the **Properties** pane, note the list of selected column names. Then in the **New column names** text box, enter the column names in the same order as they are listed, with spaces removed, separated by commas. For example:

```
RelativeCompactness,SurfaceArea,WallArea,RoofArea,OverallHeight,Orienta
tion,GlazingArea,GlazingAreaDistribution,HeatingLoad
```

12. Search for the **Normalize Data** module, and drag it to the canvas below the **Metadata Editor** module. Then connect the output port from the **Metadata Editor** module to the input port of the **Normalize Data** module.
13. Select the **Normalize Data** module, and in the **Properties** pane, in the **Transformation Method** list, select **MinMax**. Then launch the column selector and configure the module to begin with no columns, include all **Numeric** columns, exclude the **HeatingLoad** column, and exclude all **Categorical** columns; as shown in the following image.



Select columns

□ Allow duplicates and preserve column order in selection

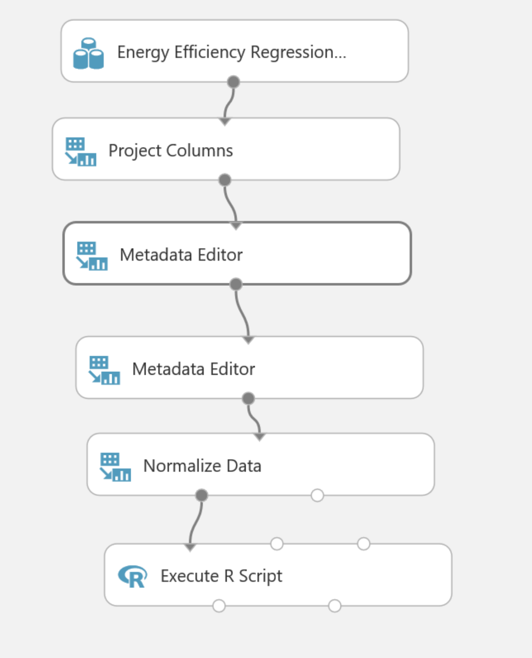
Begin With: No columns

Include | column type | Numeric

Exclude | column names | HeatingLoad ✖

Exclude | column type | Categorical

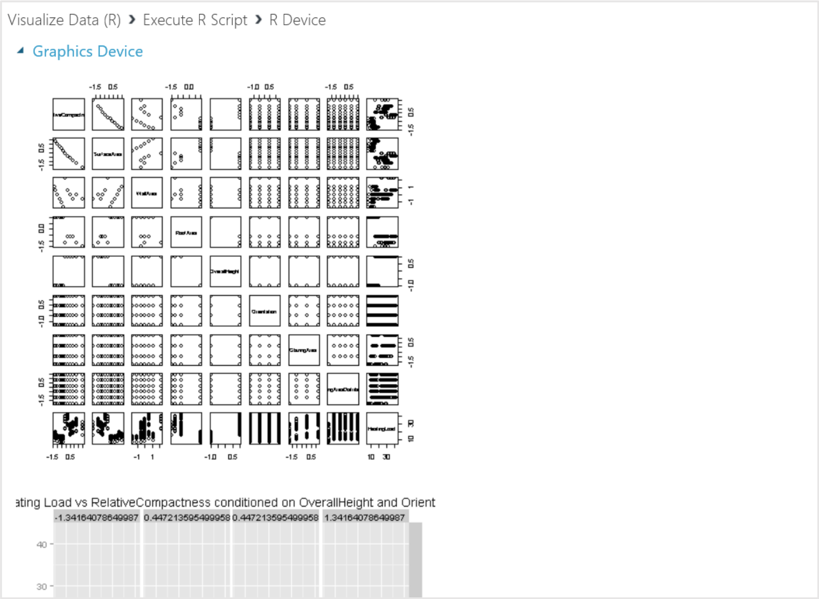
14. Search for the **Execute R Script** module, and drag it to the experiment canvas under the **Metadata Editor** module. Then connect the **Transformed dataset** output port from the **Normalize Data** module to the **Dataset1** input port of the **Execute R Script** module. Your experiment should now look like the following figure:



15. Select the **Execute R Script** module, and in the **Properties** pane, replace the default R script with the code from the **VisualizeEE.R** script you ran in RStudio (ensure you copy and paste the entire script).

> **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.

16. Edit the R script in the **Properties** pane to change the statement `Azure = FALSE` to `Azure = TRUE`. This is required to use the data from the dataset instead of loading it from a local variable.
17. Save and run the experiment.
18. When the experiment has finished running, visualize the output from the **R Device dataset** output port of the **Execute R Script** module (the output on the right), and in the **Graphics Device** area, view the data visualizations generated by the R script, as shown in the following image.



19. Close the R Device output.

Visualizing Data with Python

Python supports the **matplotlib** library; which provides extensive graphical capabilities. This makes Python a useful language with which to create visualizations of your data in order to explore relationships between the data fields and identify features that may be useful for predicting labels in machine learning projects.

In this exercise, you will use Python to visualize data from a dataset of metrics for buildings. Specifically, you will try to identify data fields that influence the *heating load* of a building, which is a measure of its energy efficiency.

Note: If you prefer to work with R, skip this exercise and complete the previous exercise, *Visualizing Data with R*.

Load the Dataset

1. Start Spyder, and open the **PrepEE.py** file in the folder where you extracted the lab files for this course.
2. In the **PrepEE.py** pane, in the following code, change ***C://DAT203xLabfiles*** to the path to the folder where you extracted the lab files for this course.

```
## Load the data
import pandas as pd
import os
from sklearn import preprocessing

pathName = "c://DAT203xLabfiles"
fileName = "EnergyEfficiencyRegressiondata.csv"
filePath = os.path.join(pathName, fileName)
eeframe = pd.read_csv(filePath)

## Remove columns we're not going to use
eeframe = eeframe.drop('Cooling Load', 1)
```

```
## scale numeric features
scaleList = ["Relative Compactness", "Surface Area",
       "Wall Area", "Roof Area", "Glazing Area",
       "Glazing Area Distribution"]
arry = eeframe[scaleList].as_matrix()
eeframe[scaleList] = preprocessing.scale(arry)
```

3. Select the code listed above (with the modified path) and on the toolbar, click **Run current cell**. The code performs the following actions:
    a. Loads a data frame named **eeframe** with data from a text file named **EnergyEfficiencyRegressiondata.txt**.
    b. Cleans the data. Specifically:
        - The **Cooling Load** column is removed, because it contains similar data to the **Heating Load** column, and is not required for the visualizations you will create in this exercise.
        - The **Relative Compactness**, **Surface Area**, **Wall Area**, **Roof Area**, **Glazing Area**, and **Glazing Distribution** columns are scaled so they can be easily compared.
4. In the **IPython console** pane, enter the following command:

```
eeframe
```

6. View the results, which show the first few rows of the data frame. Note that the data includes the following columns, which describe the physical attributes of a building and its Heating Load measurement:
    - **Relative Compactness**
    - **Surface Area**
    - **Wall Area**
    - **Roof Area**
    - **Overall Height**
    - **Orientation**
    - **Glazing Area**
    - **Glazing Area Distribution**
    - **Heating Load**

Import the matplotlib Library

1. In Spyder, open the **VisualizeEE.py** file in the folder where you extracted the lab files for this course.
2. In the **VisualizeEE.py** pane, under the comment **## Import Libraries**, select the following code.

```
import matplotlib
matplotlib.use('agg')  # Set backend

from pandas.tools.plotting import scatter_matrix
import pandas.tools.rplot as rplot
import matplotlib.pyplot as plt
import numpy as np
```

3. With the code above selected, on the toolbar, click **Run current cell**. The code imports the **matplotlib** library, and some other libraries used in the script. Please ignore the warning about `rplot` being deprecated.

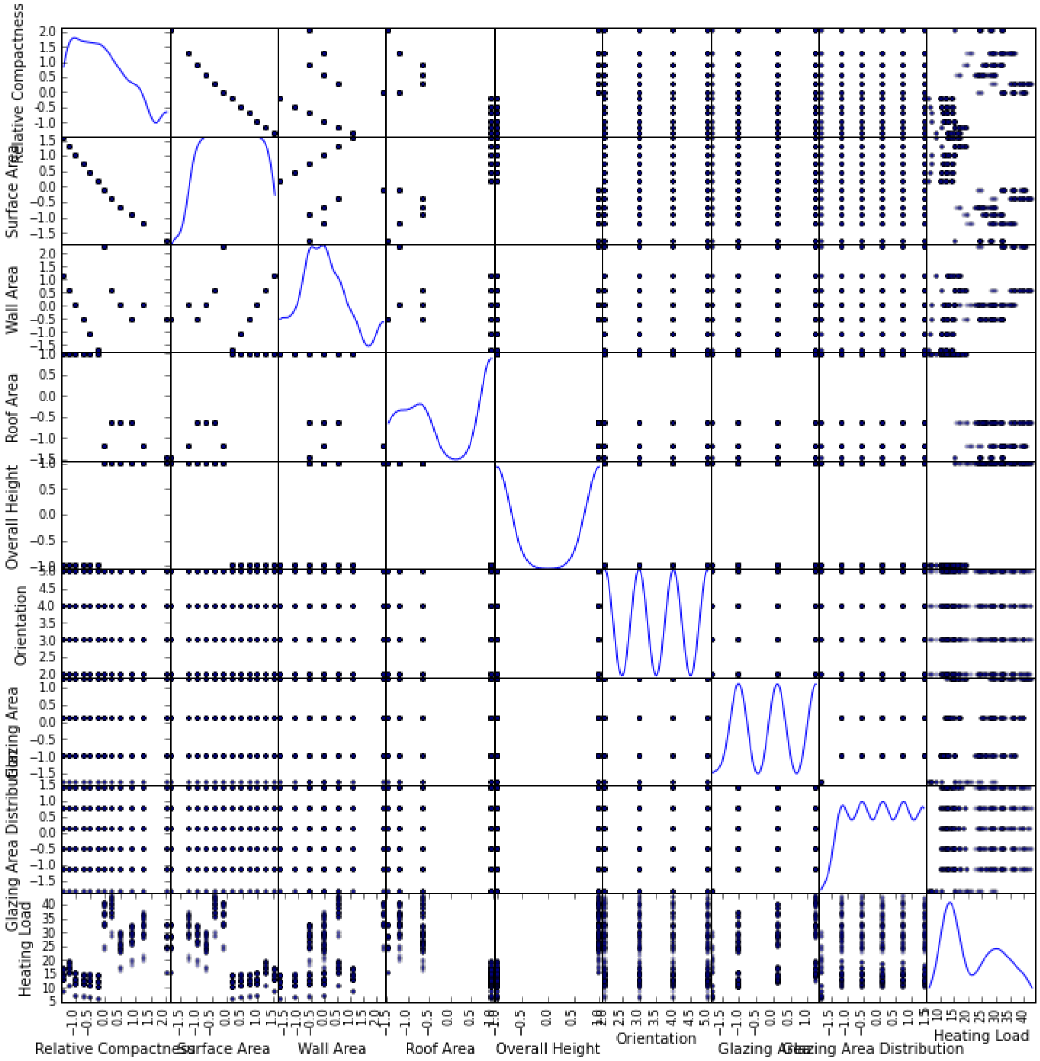
Note: When running in an IPython console on your desktop machine the above code may generate several warnings. Setting a backend may be ignored. Further, the `rplot` library will generate warnings that it has been deprecated. Please ignore these warnings. This code will operate without warnings in the Azure ML **Execute Python Script** module.

Create a Pair-Wise Scatter Plot

1. In the **VisualizeEE.py** pane, under the comment **## Create a pair-wise scatter plot**, select the following code.

```
Azure = False
## If in Azure, frame1 is passed to function
if(Azure == False):
    frame1 = eeframe

fig1 = plt.figure(1, figsize=(10, 10))
ax = fig1.gca()
scatter_matrix(frame1, alpha=0.3,
               diagonal='kde', ax = ax)
plt.show()
if(Azure == True): fig1.savefig('scatter1.png')
```

2. With the code above selected, on the toolbar, click **Run current cell**. When running in Azure, a data frame parameter named **frame1** is passed to the first input port of the **Execute Python Script** module; but when running locally, this code loads **frame1** with the **eeframe** data frame you loaded in the **PrepEE.py** script. The code then creates a scatter plot matrix visualization that compares all columns in the dataset, and if running in Azure, saves the resulting image so that it can be included in the output.
3. In the **IPython console** pane, view the scatter plot matrix that has been generated, as shown below.



Note: This plot is comprised of a number of scatter plots. For each variable there is both a row and a column. The variable is plotted on the vertical axis in the row, and on the horizontal axis in the column. In this way, every combination of cross plots for all variables are displayed in both possible orientations. This type of plot allows you to examine the relationships between many variables in one view. The alternative of examining a large number of scatter plot combinations one at a time is both tedious and in all likelihood difficult since you need to remember relationships you have already viewed to understand the overall structure of the data.

4. Some particular features you can notice include:
    - You can see plots of **Relative Compactness** vs. **Surface Area**; second from left in the top row and second from top in the left most column. These two variable appear to be highly correlated. We may not want to use both features in a given model.
    - Many of the other variables seem to cluster into two groups based on the value of **Overall Height**.
    - **Heating Load** clusters into two groups for certain variables. For example, look at the cross plots between **Heating Load** and **Relative Compactness** and **Surface Area.**

Create Conditioned Scatter Plots

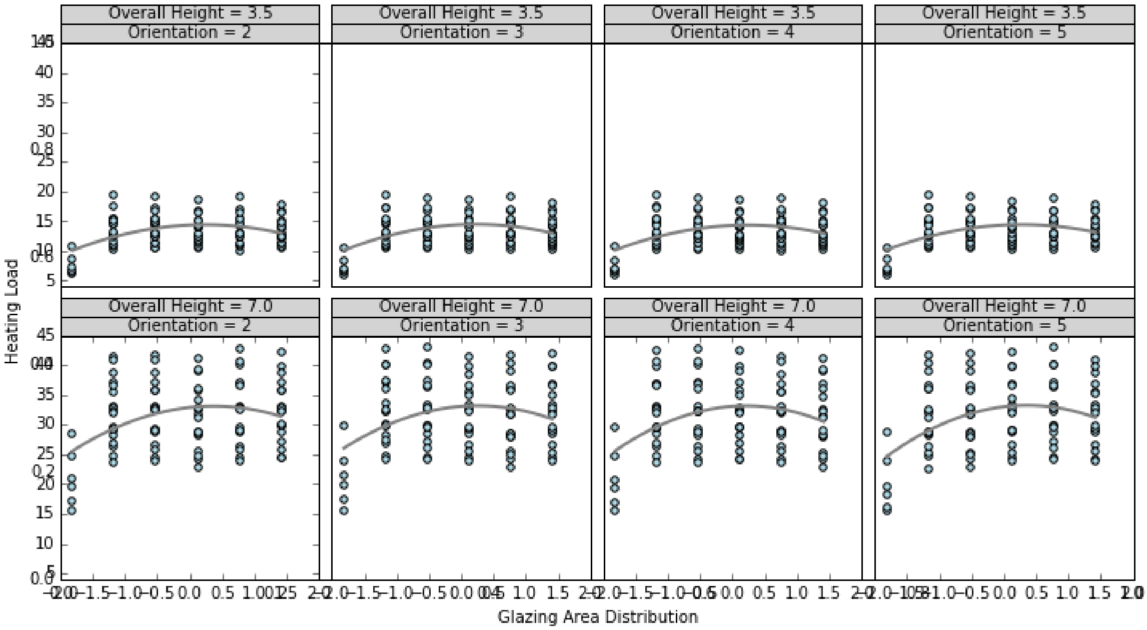
1. In the **VisualizeEE.py** pane, under the comment **## Create conditioned scatter plots**, select the following code.

```
## Create conditioned scatter plots.
    col_list = ["Relative Compactness",
               "Surface Area",
               "Wall Area",
               "Roof Area",
               'Glazing Area',
               "Glazing Area Distribution"]

    indx = 0
    for col in col_list:
        if(frame1[col].dtype in [np.int64, np.int32, 
np.float64]):
            indx += 1

            fig = plt.figure(figsize = (12,6))
            fig.clf()
            ax = fig.gca()
            plot = rplot.RPlot(frame1, x = col, y = 'Heating 
Load')
            plot.add(rplot.TrellisGrid(['Overall 
Height','Orientation']))
            plot.add(rplot.GeomScatter())
            plot.add(rplot.GeomPolyFit(degree=2))
            ax.set_xlabel(col)
            ax.set_ylabel('Heating Load')
            plot.render(plt.gcf())

            if(Azure == True): fig.savefig('scatter' + col + 
'.png')
```

2. With the code above selected, on the toolbar click **Run current cell**. The code creates scatter plot charts that compare **Heating Load** with:
    - **Relative Compactness**
    - **Surface Area**
    - **Wall Area**
    - **Roof Area**
    - **Glazing Area**
    - **Glazing Area Distribution**

   The code adds conditions for **Overall Height** and **Orientation** to each of these scatter plots.

3. In the **IPython console** pane, view the scatter plots, as shown below.



4. Note that each chart includes eight scatter plots. There are two rows of shaded tiles horizontally across the top, one for each level (unique value) of **Orientation**, and one for each of the two levels (unique values) of **Overall Height**. Each of these scatter plots has **Relative Compactness** on the vertical (x) axis and **Heating Load** on the vertical (y) axis. The data are grouped by, or condition on **Overall Height** value (3.5 and 7) and **Orientation** value (2, 3, 4, and 5).
5. Examine this chart and note the following interesting features in these scatter plots:
    - The range of values of `Heating Load` are quite different between the upper and lower rows; **Overall Height** of 3.5 and 7, respectively. In fact, there is very little overlap in these values, indicating that **Overall Height** is an important feature in these data.
    - The distribution of these data does not change significantly with the levels of **Orientation,** indicating it is not a significant feature.
    - There is a notable trend of **Heating Load** with respect to **Relative Compactness**, indicating **Relative Compactness** is a significant feature.

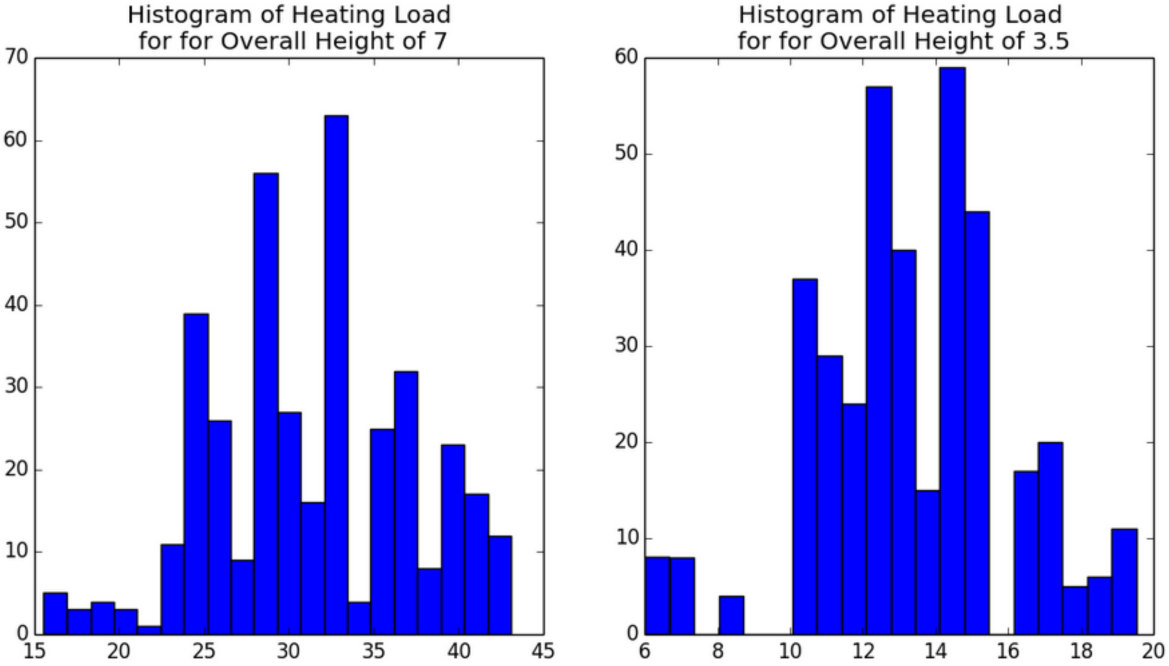
Create Histograms

1. In the **VisualizeEE.py** pane, under the comment **## Histograms of features by Overall Height**, select the following code.

```
## Create histograms
    col_list = ["Relative Compactness",
                "Surface Area",
                "Wall Area",
                "Roof Area",
                'Glazing Area',
                "Glazing Area Distribution",
                "Heating Load"]
    for col in col_list:
        temp7 = frame1.ix[frame1['Overall Height'] == 7,
                                 col].as_matrix()
        temp35 = frame1.ix[frame1['Overall Height'] == 3.5,
                               col].as_matrix()
        fig = plt.figure(figsize = (12,6))
```

```
fig.clf()
ax7 = fig.add_subplot(1, 2, 1)
ax35 = fig.add_subplot(1, 2, 2)
ax7.hist(temp7, bins = 20)
ax7.set_title('Histogram of ' +col +
               '\n for for Overall Height of 7')
ax35.hist(temp35, bins = 20)
ax35.set_title('Histogram of ' +col +
               '\n for for Overall Height of 3.5')
```

2. With the code above selected, on the toolbar click **Run current cell**. The code creates histogram pairs for each of the following columns:
    - **Relative Compactness**
    - **Surface Area**
    - **Wall Area**
    - **Roof Area**
    - **Glazing Area**
    - **Glazing Area Distribution**
    - **Heating Load**
3. In the **IPython console** pane, view each of the histograms, as shown below.



Note: A histogram plots the density of a distribution of the vertical axis, vs. bins of values on the horizontal axis. The values of a continuous variable are placed into one of several equal width bins. The density for each bin is the count of values in that bin. As the number or width of the bins changes, the details of the histogram will change. Histograms provide an empirical view of the distribution of the data being plotted.

4. Examine the pairs of histograms created. In most cases, the range of values on the horizontal axis are quite different for the two values of **Overall Height**; 7 and 3.5. A few of histogram pairs show little difference between the two values of **Overall Height**.

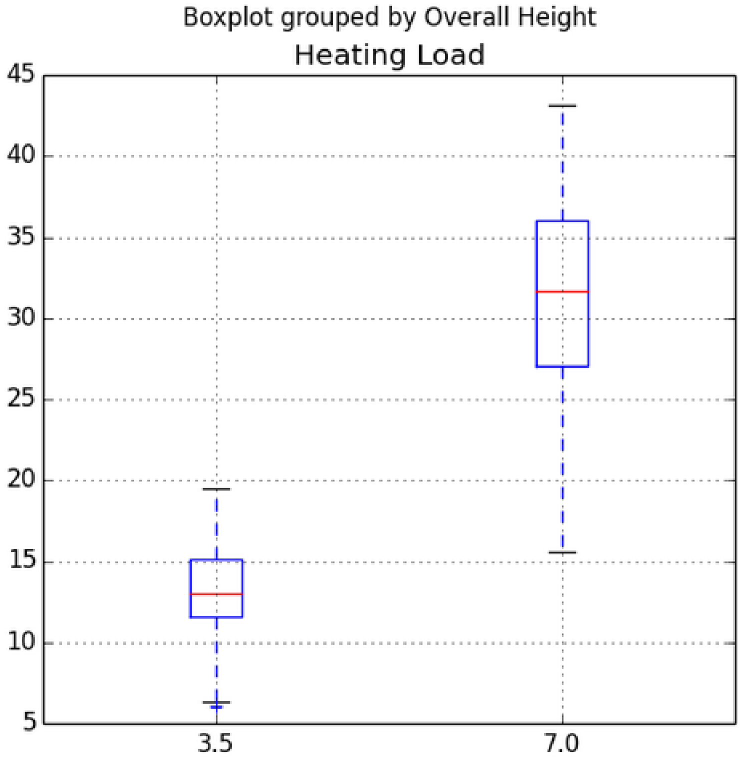
    Direct your attention to the histograms of **Heating Load**. Examine this chart and note how different the distribution of **Heating Load** is for the two values of **Overall Height**. In fact, there is very little overlap in the range of values for the two levels of **Overall Height**. Additionally, note the outliers in both distributions shown.

Create Boxplots

1. In the **VisualizeEE.py** pane, under the comment **## Create boxplots**, select the following code.

```
for col in col_list:
    if(frame1[col].dtype in [np.int64, np.int32, np.float64]):
        fig = plt.figure(figsize = (6,6))
        fig.clf()
        ax = fig.gca()
        frame1[[col, 'Overall Height']].boxplot(column = [col],
                       ax = ax, by = ['Overall Height'])
        ax.set_xlabel('')
        if(Azure == True): fig.savefig('box_' + col + '.png')
```

2. With the code above selected, on the toolbar click **Run current cell**. The code uses the **col_list** list you just created to make boxplots for each of the following columns:
    - **Relative Compactness**
    - **Surface Area**
    - **Wall Area**
    - **Roof Area**
    - **Glazing Area**
    - **Glazing Area Distribution**
    - **Heating Load**
3. In the **IPython console** pane, view each of the boxplot pairs as shown below.



Note: The most prominent feature of a boxplot is the box. The box encloses the inner two quartiles of the distribution; first upper and first lower quartile. The line inside the box is placed at the median value. The lines or whiskers coming from the top and bottom of the box represent the outer upper and lower most quartiles of the distribution; second upper and second lower quartile. Any outliers are represented by symbols beyond the end of the end of the whiskers.

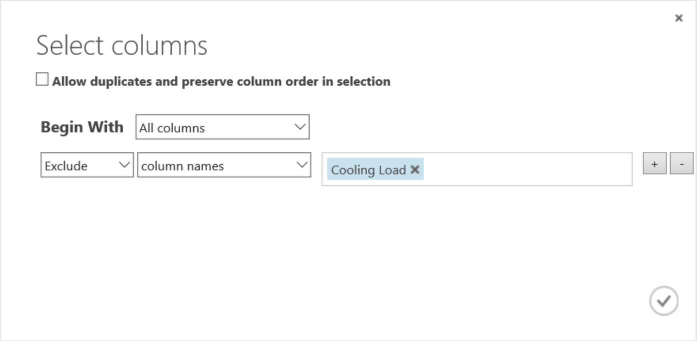
5. Examine the boxplots created for each of the variables. These boxplots show a different view of similar distribution information as the histograms.

   In most cases, there is little overlap in the values of the variable when grouped by the two levels (values) of **Overall Height**; 7 and 3.5. A few of boxplots show little difference between the two values of **Overall Height**.

   Direct your attention to the boxplot of **Heating Load**. Examine this chart and note how different the distribution of **Heating Load** is for the two values of **Overall Height**. In fact, there is very little overlap in the range of values for the two levels of **Overall Height**.

Create Visualizations in an Azure Machine Learning 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 a new blank experiment, and give it the title **Visualize Data (Python)**.
3. Search for the **Energy Efficiency Regression data** dataset, and drag it to the experiment canvas.
4. Search for the **Project Columns** module and drag it to the canvas below the **Energy Efficiency Regression data** dataset. Then connect the output port from the **Energy Efficiency Regression data** dataset to the first input port of the **Project Columns** module.
5. Select the **Project Columns** module, and in the **Properties** pane, launch the column selector. Then configure the column selector to begin with all columns and exclude the **Cooling Load** column as shown in the following image.



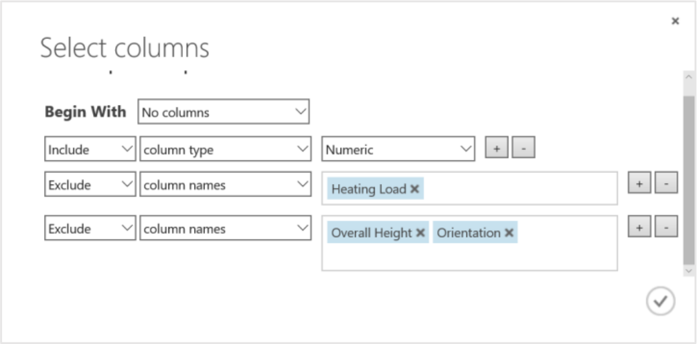
Select columns

☐ Allow duplicates and preserve column order in selection

Begin With: All columns

Exclude | column names | Cooling Load ✖

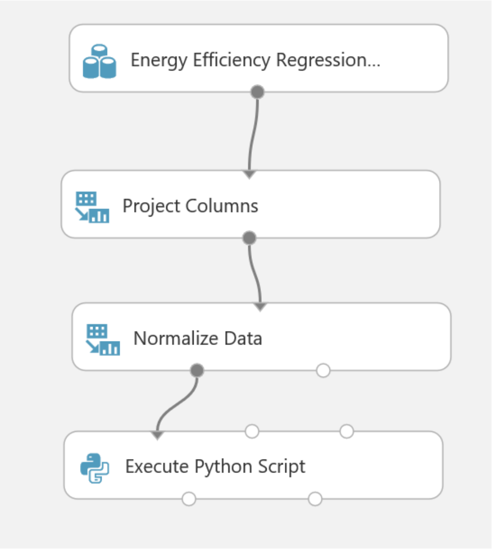
6. Search for the **Normalize Data** module, and drag it to the canvas below the **Project Columns** module. Then connect the output port from the **Project Columns** module to the input port of the **Normalize Data** module.
7. Select the **Normalize Data** module, and in the **Properties** pane, in the **Transformation Method** list, select **MinMax**. Then launch the column selector and configure the module to begin with no columns, include all **Numeric** columns, exclude the **Heating Load** column (which is the label we hope to predict), and exclude the **Overall Height,** and **Orientation** columns (which are categorical), as shown in the following image.



Select columns

Begin With	No columns		
Include	column type	Numeric	+ -
Exclude	column names	Heading Load ✖	+ -
Exclude	column names	Overall Height ✖ Orientation ✖	+ -

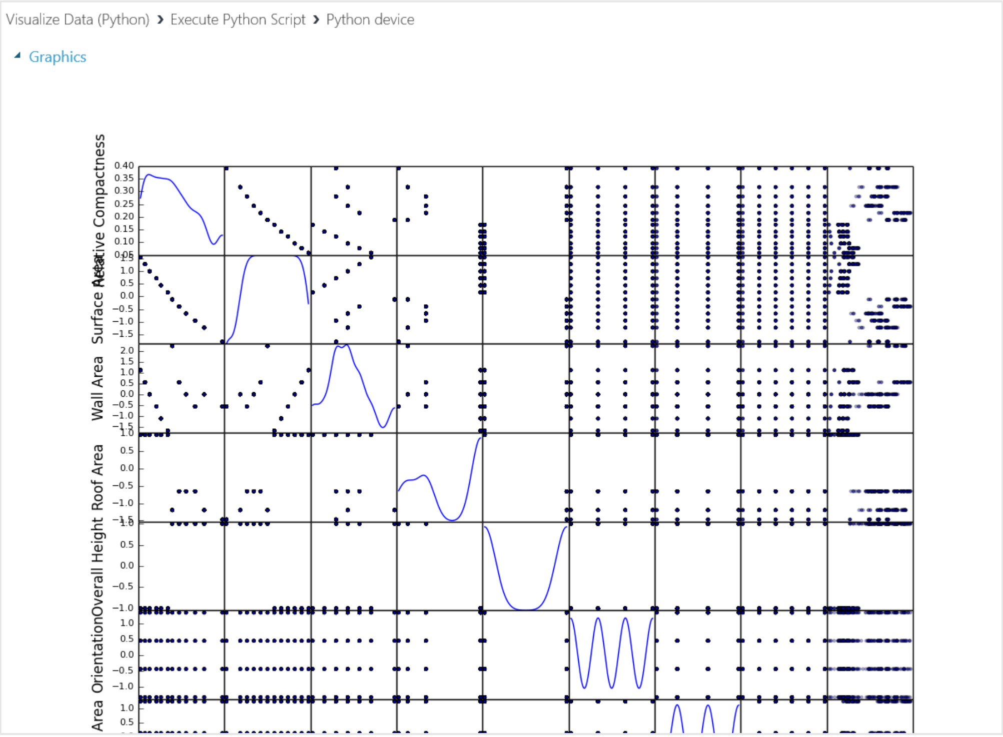
8. Search for the **Execute Python Script** module, and drag it to the experiment canvas under the **Normalize Data** module. Then connect the **Transformed dataset** output port from the **Normalize Data** module to the first input port of the **Execute Python Script** module. At this point your experiment should look like the following figure:



9. Select the **Execute Python Script** module, and in the **Properties** pane, replace the default Python script with the code from the **VisualizeEE.py** script you ran in Spyder (ensure you copy and paste the entire script, including the function definition and the return statement!).

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.

10. Edit the Python script in the **Properties** pane to change the statement `Azure = False` to `Azure = True`. This is required to use the data from the dataset instead of loading it from a local file.
11. Save and run the experiment.
12. When the experiment has finished running, visualize the output from the **Python Device dataset** output port of the **Execute Python Script** module (the output on the right), and in the **Graphics** area, view the data visualizations generated by the Python script, as shown in the following image.



13. Close the Python device output.

Summary

In this lab, you used custom Python or R code to visualize data. This is an important technique for exploring data in order to identify relationships between data fields when planning a machine learning model.

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