Microsoft

Data Science Essentials

Lab 5 – Transforming Data

Overview

In this lab, you will learn how to use tools in Azure Machine Learning along with either Python or R to integrate, clean and transform data. Collectively, data scientists refer to these processes as 'data munging'. You will work with a version of the automotive price dataset which has been modified to illustrate some of the points in this lab.

Datasets rarely arrive in the form required for serious exploration and analysis. Preparing data for exploration and analysis is an essential step in the data science process. Thus, data scientists must develop skills in data integration, cleaning and transformation.

Note: This lab provides instructions to perform each task required to ingest, join, and manipulate the automobile price data. In many cases, the tasks can be accomplished using one of a variety of techniques; including built-in Azure ML modules, SQL, R, or Python. You can choose to perform these tasks using any (or all) of the techniques described.

The built-in modules in Azure ML provide an easy to use approach to performing some of the most common data transformations, and if you intend to use an Azure ML experiment to build and publish a predictive web service, you should generally try to use the built-in modules wherever possible as they are re-configured for production automatically as part of the web service creation process.

You can use custom scripts to perform the same operations supported by the built-in modules, and in some cases you can write scripts to make data transformations that are not readily supported by the built-in modules. However, you should be cautious when using custom code in an experiment that you intend to publish as a predictive web service – especially if the web service will be required to work with single row inputs. During the web service creation process, Azure ML converts built-in modules that perform aggregation operations on multiple rows so that they will work with single rows in production based on statistical constants derived from training data in the experiment. This conversion cannot be automated for custom code.

What You'll Need

To complete this lab, you will need the following:

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

Note: To set up the required environment for the lab, follow the instructions in the Setup Guide for this course.

Ingesting and Joining Data

Most data science experiments require you to ingest data from one or more sources, and potentially join data from diverse sources based on a common key field. In this exercise, you will explore data from two source files, and then create an Azure Machine Learning experiment in which you read data from the sources and implement a join to merge the two datasets into a single table.

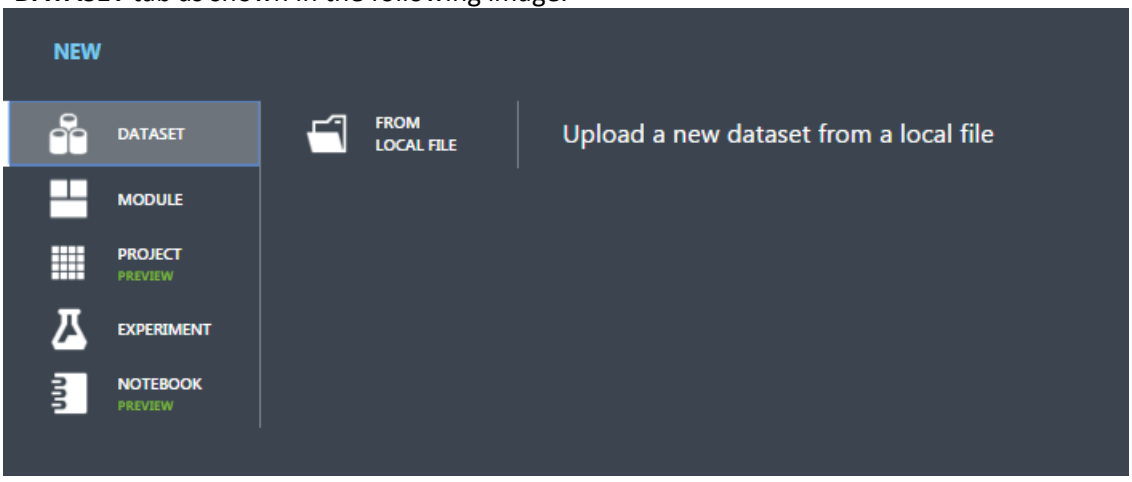
Examine the Datasets

Use a text editor or spreadsheet application such as Microsoft Excel to open the **autos.csv** file in the directory where you unpacked the lab files for this module (for example, C:\DAT203.1x\Mod5) and examine the data it contains.

1. Note the following:
    - The column names are in the first row.
    - There are numeric and character data columns.
    - The **make-id** column contains a numeric code that represents the automobile manufacturer.
2. Close the file, **being sure not to save any changes**.
3. Open the **makes.csv** file in the same directory, and examine the columns in this data. The **make-id** column matches the key values in the **autos.csv** file, and the manufacturer name for each **make-id** is listed in the **make** column.
4. Close the data file, **being sure not to save any changes**.

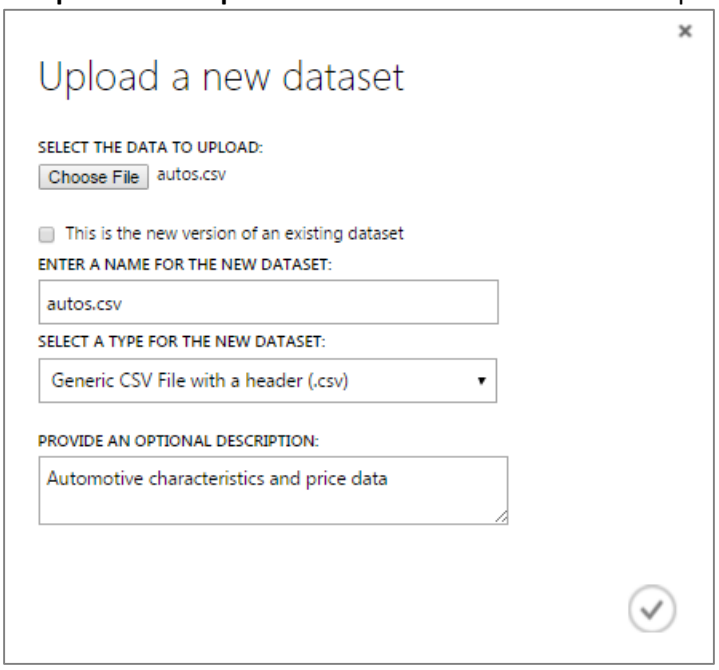
Upload Data into Azure Machine Learning

1. 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 name it **Autos**.
3. With your experiment open, at the bottom left, click **NEW**. Then in the **NEW** dialog box, click the **DATASET** tab as shown in the following image.

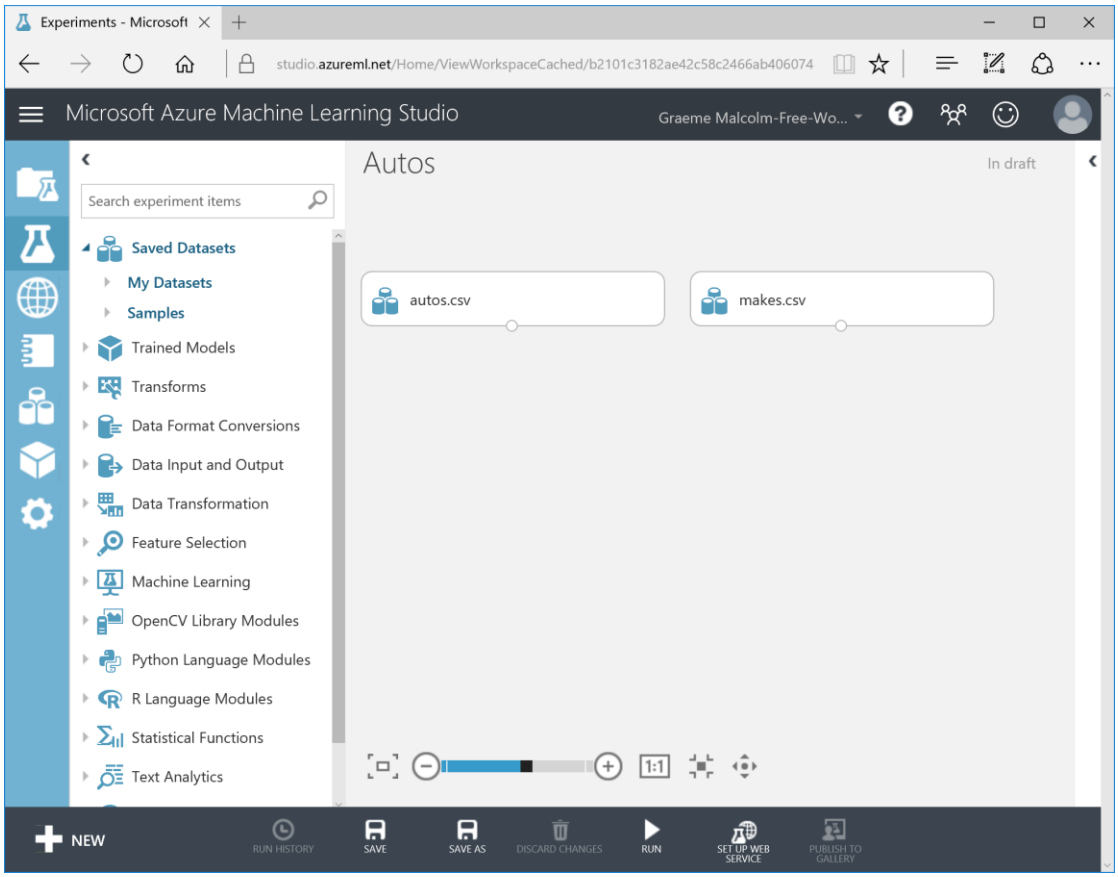


4. Click **FROM LOCAL FILE**. Then in the **Upload a new dataset** dialog box, browse to the **autos.csv** file from the folder where you extracted the lab files on your local computer and enter the following details as shown in the image below, and then click the OK icon.

- **This is a new version of an existing dataset**: Unselected
- **Enter a name for the new dataset**: autos.csv
- **Select a type for the new dataset**: Generic CSV file with a header (.csv)
- **Provide an optional description**: Automotive characteristics and price data



5. Wait for the upload of the dataset to be completed, and then on the experiment items pane, expand **Saved Datasets** and **My Datasets** to verify that the **autos.csv** dataset is listed.
6. Repeat the process to upload the **makes.csv** file with the following settings.
    - **This is a new version of an existing dataset**: Unselected
    - **Enter a name for the new dataset**: makes.csv
    - **Select a type for the new dataset**: Generic CSV file with a header (.csv)
    - **Provide an optional description**: Automotive manufacturer data
7. In the **Autos** experiment, drag the both the **autos.csv** and **makes.csv** datasets to the canvas, as shown here:



Join Data

The procedures in this section describe how to join the two datasets by using the following techniques:

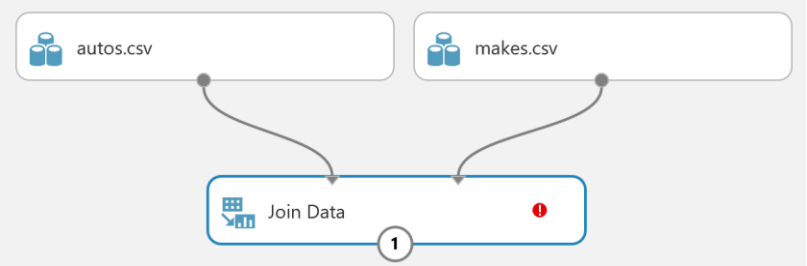
- The built-in Azure ML **Join Data** module
- A SQL script in an **Apply SQL Transformation** module
- An R script in an **Execute R Script** module
- A Python script in an **Execute Python Script** module

Note: You need only complete one of the following procedures, but you can try multiple alternative techniques if you wish!

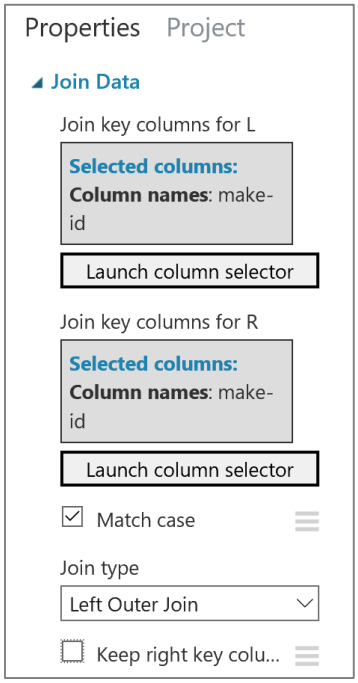
Join Data with the Join Data Module

If you want to join the datasets using the built-in Join Data module, follow this procedure.

1. In the **Autos** experiment, search for the **Join Data** module and drag it onto the canvas beneath the datasets.
2. Connect the output of the **autos.csv** dataset to the **Dataset1** (left) input of the **Join Data** module. Then connect the output of the **makes.csv** dataset to the **Dataset2** (right) input of the **Join Data** module as shown here:



3. Select the **Join Data** module, and in the properties pane on the right, set the following properties:
    - Launch the column selector to select the join key columns for L, and in the list of columns from the **autos.csv** dataset, select **make-id**.
    - Launch the column selector to select the join key columns for R, and in the list of columns from the **makes.csv** dataset, select **make-id**.
    - Note the **Match case** checkbox, which in this case makes no difference as the key is numeric.
    - In the **Join type** list, select **Left Outer Join**. This will ensure that the results retain all records from the **autos.csv** dataset, even if there is no matching record in the **makes.csv** dataset.
    - Clear the **Keep right key column** checkbox. This will avoid including a duplicate **make-id** column in the results
4. Ensure that the **Properties** pane for the **Join Data** module looks like this:

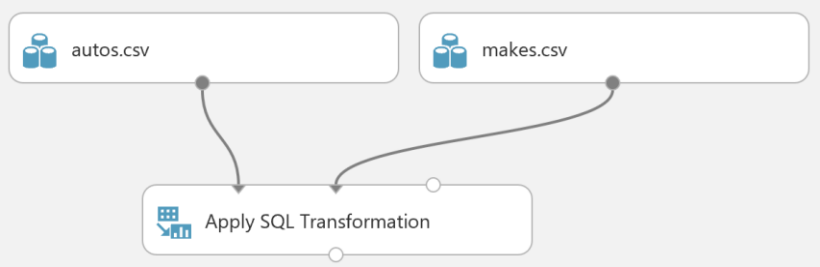


5. **Save** and **run** the experiment.
6. When the experiment has finished running, visualize the output of the **Join Data** module and verify that it contains all of the columns from the original **autos.csv** dataset, and a column named **make** containing the manufacturer of each automobile from the **makes.csv** dataset.

Join Data with SQL

If you want to join the datasets using a SQL script, follow this procedure.

1. In the **Autos** experiment, search for the **Apply SQL Transformation** module and drag it onto the canvas beneath the datasets.
2. Connect the output of the **autos.csv** dataset to the **Table1** (left) input of the **Apply SQL Transformation** module. Then connect the output of the **makes.csv** dataset to the **Table2** (middle) input of the **Apply SQL Transformation** module as shown here:



3. Select the **Apply SQL Transformation** module, and in the **Properties** pane, replace the existing SQL Query Script with the following code (which you can copy and paste from **Join.sql** in the lab files folder for this module):

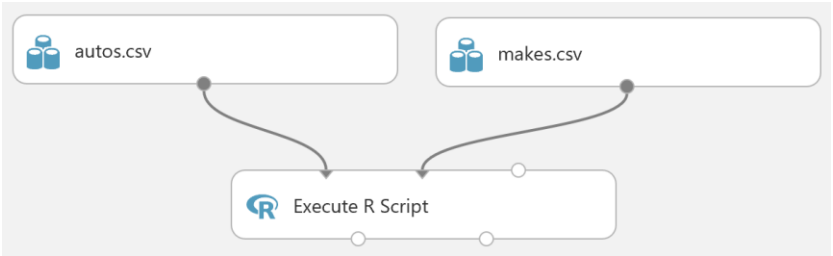
```
SELECT t1.*, t2.[make]
FROM t1 LEFT OUTER JOIN t2
ON t1.[make-id] = t2.[make-id];
```

4. **Save** and **run** the experiment.
5. When the experiment has finished running, visualize the output of the **Apply SQL Transformation** module and verify that it contains all of the columns from the original **autos.csv** dataset, and a column named **make** containing the manufacturer of each automobile from the **makes.csv** dataset.

Join Data with R

If you want to join the datasets using an R script, follow this procedure.

1. In the **Autos** experiment, search for the **Execute R Script** module and drag it onto the canvas beneath the datasets.
2. Connect the output of the **autos.csv** dataset to the **Dataset1** (left) input of the **Execute R Script** module. Then connect the output of the **makes.csv** dataset to the **Dataset2** (middle) input of the **Execute R Script** module as shown here:



3. Select the **Execute R Script** module, and in the **Properties** pane, replace the existing R Script with the following code (which you can copy and paste from **Join.R** in the lab files folder for this module):

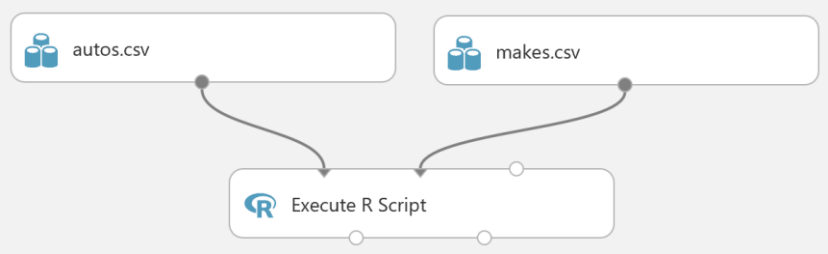
```
## Define a function to join the data frames
join.auto <- function(autos, makes){
  require(dplyr) ## Make sure dplyr is loaded
  left_join(autos, makes, by = 'make-id')
}

## R code to join the two input tables
autos <- maml.mapInputPort(1) # read autos data frame from port 1
makes <- maml.mapInputPort(2) # read makes data frame from port 2
out <- join.auto(autos, makes) ## Join the data frames
maml.mapOutputPort("out") ## Output the joined data frame
```

4. **Save** and **run** the experiment.
5. When the experiment has finished running, visualize the output of the **Execute R Script** module and verify that it contains all of the columns from the original **autos.csv** dataset, and a column named **make** containing the manufacturer of each automobile from the **makes.csv** dataset.

Join Data with Python

If you want to join the datasets using a Python script, follow this procedure.

1. In the **Autos** experiment, search for the **Execute Python Script** module and drag it onto the canvas beneath the datasets.
2. Connect the output of the **autos.csv** dataset to the **Dataset1** (left) input of the **Execute R Script** module. Then connect the output of the **makes.csv** dataset to the **Dataset2** (middle) input of the **Execute Python Script** module as shown here:



3. Select the **Execute Python Script** module, and in the **Properties** pane, replace the existing R Script with the following code (which you can copy and paste from **Join.py** in the lab files folder for this module):

```
## Function to join automotive data
def auto_join(autos, makes):
    import pandas as pd
    key = ['make-id'] ## Define key column
    ## Return the joined dataframe
    return pd.merge(autos, makes, on = key, how = 'left')

## Perform the join in AML
def azureml_main(autos, makes):
    return auto_join(autos, makes)
```

4. **Save** and **run** the experiment.
5. When the experiment has finished running, visualize the output of the **Execute Python Script** module and verify that it contains all of the columns from the original **autos.csv** dataset, and a

column named **make** containing the manufacturer of each automobile from the **makes.csv** dataset.

Manipulating Data and Metadata

In this exercise you will start to prepare the joined automotive dataset so that it is ready for meaningful exploration and analysis.

The procedures in this section describe how to prepare the dataset by using the following techniques:

- Built-in Azure ML modules
- Custom code; specifically:
  - A SQL script in an **Apply SQL Transformation** module
  - An R script in an **Execute R Script** module
  - A Python script in an **Execute Python Script** module

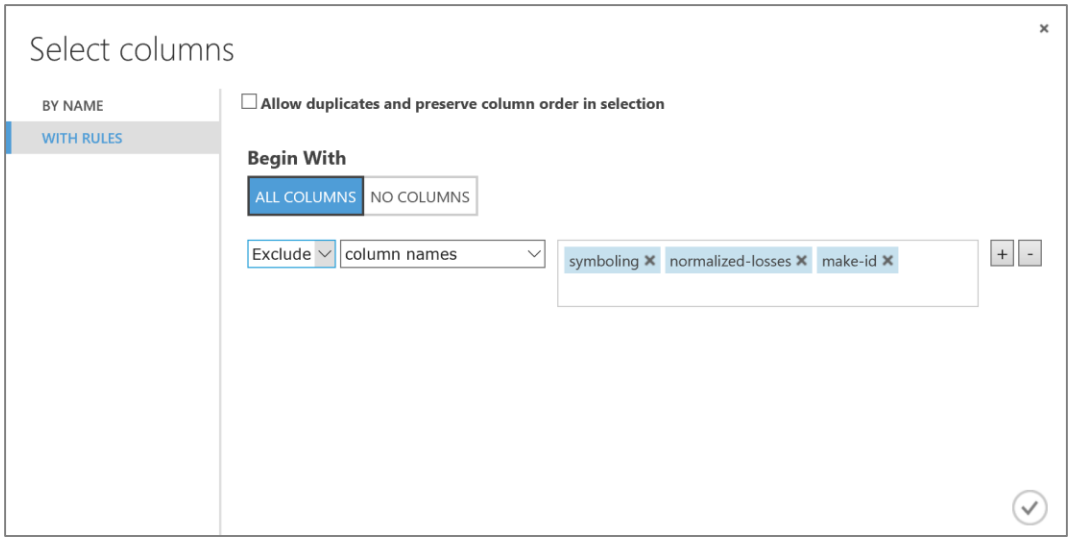
Note: You must complete at least one of the following procedures, but you can try multiple alternative techniques if you wish!

Manipulate Data and Metadata with Azure ML

To manipulate the data and metadata for the automobile dataset by using built-in Azure ML modules, complete the following tasks.

Remove Unnecessary Columns

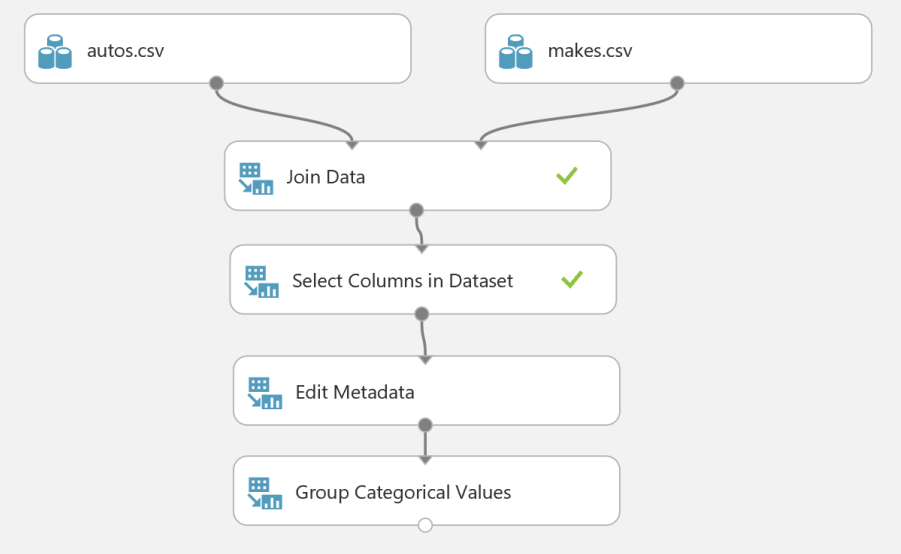
1. In the **Autos** experiment, visualize the output of the module you used to join the **autos.csv** and **makes.csv** datasets, and note that it contains columns named **symboling**, **normalized-losses**, and **make-id**. You have determined that these columns are not useful in building a predictive model for automobile prices, so you have decided to remove them.
2. Search for a **Select Columns in Dataset** module and drag it to the canvas under the module you used to join the **autos.csv** and **makes.csv** datasets.
3. Connect the output from the module that joins the datasets to the input of the **Select Columns in Dataset** module.
4. Select the **Select Columns in Dataset** module, and in the **Properties** pane, launch the column selector and on the **WITH RULES** page configure the module to start with all columns and exclude the **symboling**, **normalized-losses**, and **make-id** columns as shown here:



5. Save and run the experiment, and when the experiment has finished running, visualize the output of the **Select Columns in a Dataset** module to verify that the **symboling**, **normalized-losses**, and **make-id** columns are no longer included in the dataset. Then select the column header for the **num-of-cylinders** column and note that it is a string feature with seven unique values. The histogram shows that the frequency of these values is spread unevenly, with very few instances of some values – so it may be more useful to group these into a smaller number of categories.

Create a Grouped Categorical Feature

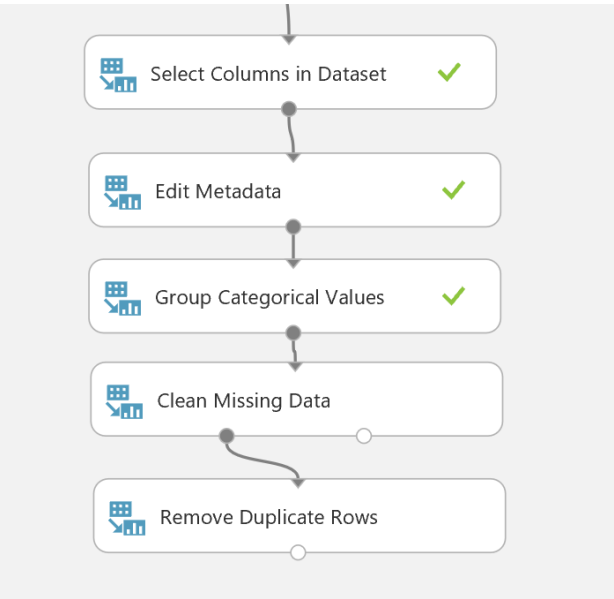
1. Search for the **Edit Metadata** module, drag it to the experiment beneath the **Select Columns in a Dataset** module, and connect the output from the first **Select Columns in a Dataset** module to its input.
2. Set the properties of the new **Edit Metadata** module as follows:
    - **Column:** Launch the column select and include only the **num-of-cylinders** column.
    - **Data type:** Unchanged
    - **Categorical:** Make categorical
    - **Fields:** Unchanged
    - **New column names:** blank
3. Search for a **Group Categorical Values** module and drag it to the canvas beneath the **Edit Metadata** module.
4. Connect the output of the **Edit Metadata** module to the input of the new **Group Categorical Values** module.
5. Set the properties of the new **Group Categorical Values** module as follows:
    - **Selected columns**: num-of-cylinders
    - **Output mode**: Inplace
    - **Default level name**: other
    - **New number of levels**: 4
    - **Name of new level 1**: four-or-less
    - **Comma separated list of old levels to map to new level 1**: two,three,four
    - **Name of new level 2**: five-six
    - **Comma separated list of old levels to map to new level 2**: five,six
    - **Name of new level 3**: eight-twelve
    - **Comma separated list of old levels to map to new level 3**: eight,twelve
6. Ensure that your experiment looks similar to this:



7. Save and run your experiment.
8. When the experiment has finished running, visualize the output of the **Group Categorical Values** module select the **num-of-ylinders** column heading, and note that it now contains three unique values based on the categorical levels you specified.
9. Select the column headings for the **stroke**, **horsepower**, and **peak-rpm** columns in turn, noting the number of missing values in these columns. Note the total number of rows in the dataset, which includes the rows containing missing values and potentially also includes rows that are duplicated.

Remove Rows with Missing or Repeated Values

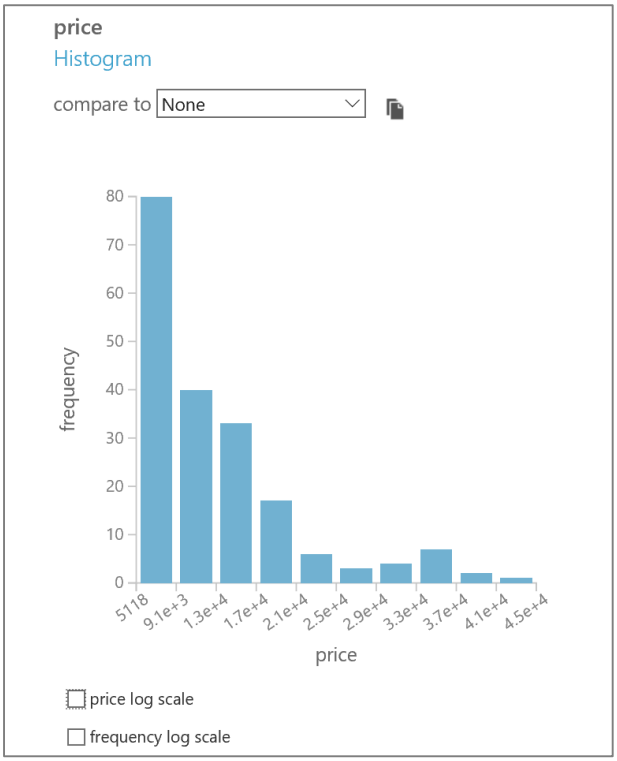
1. Drag a **Clean Missing Data** module onto the canvas beneath the **Group Categorical Values** module.
2. Connect the output of the **Group Categorical Values** module to the input of the **Clean Missing Data** module.
3. On the properties pane of the **Clean missing data** module set the following properties:
    - **Column selector**: All columns
    - **Minimum missing value ratio**: 0
    - **Maximum missing value ratio**: 1
    - **Clearing mode**: Remove entire row
4. Drag a **Remove Duplicate Rows** module onto the canvas beneath the **Clean Missing Data** module.
5. Connect the **Results** dataset (left) output of the **Clean Missing Data** module to the input of the **Remove Duplicate Rows** module.
6. Set the properties of the **Remove Duplicate Rows** module as follows:
    - **Key column selection**: Include all features
    - **Retain first duplicate row**: Checked
7. Ensure that the lower part of your experiment resembles the figure below:



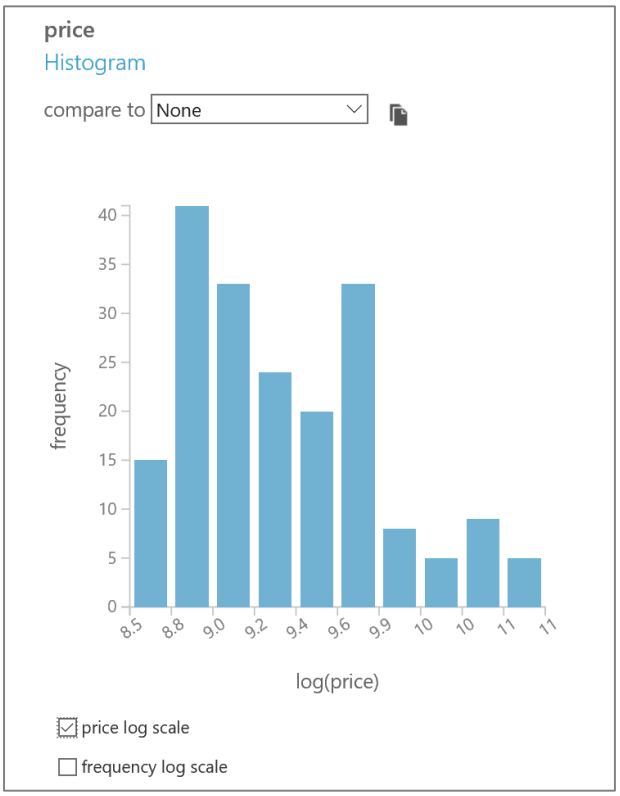
8. Save and run your experiment.
9. Visualize the output of the **Group Categorical Values**, **Clean Missing Data**, and **Remove Duplicate Rows** modules; noting the number of rows returned by each module.

Create a Calculated Column

1. Visualize the output of the **Remove Duplicate Rows** module, and select the **price** column. Then note the histogram for price shows that the data is skewed so that there are proportionally many more low-price cars than medium or high-price cars, as shown here:

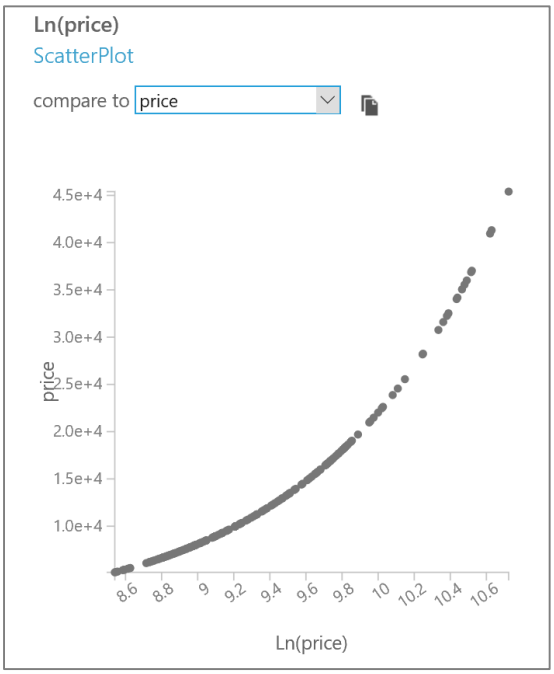


2. Select the price log scale checkbox to view the distribution when the log of price is calculated, and note that the data is more evenly distributed, as shown here:



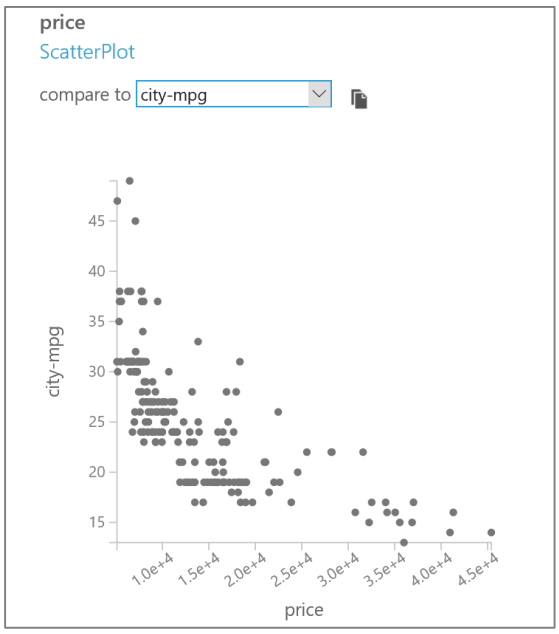
The overall goal of the data exploration you are conducting is to find a way to predict a car's price based on its features, but the skewed nature of the price distribution will make this difficult. It may be easier to fit the more balanced logarithmic distribution price values to a predictive model. To support this hypothesis, you will create a new calculated column in the dataset that contains the natural log of the price.

3. Close the visualization and drag an **Apply Math Operation** module onto the canvas beneath the **Remove Duplicate Rows** module.
4. Connect the output of the **Remove Duplicate Rows** module to the input of the **Apply Math Operation** module.
5. Set the properties of the **Apply Math Operation** module as follows:
    - **Category:** Basic
    - **Basic math function**: Ln
    - **Column set, Column names**: price
    - **Output mode**: Append
6. Save and run the experiment, and when it has finished, visualize the output of the **Apply Math Operation** module to verify that a new calculated column named **Ln(price)** has been created. Select the **Ln(price)** column, and in the **Visualizations** area, verify that the histogram shows a more even distribution than that of the original price column. Then, in the **compare to** list select the **price** column. The visualization should resemble the following:



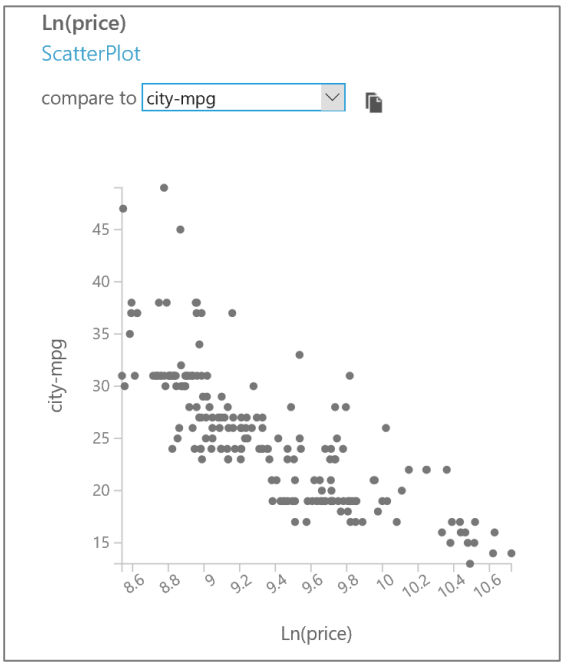
Note the clear numeric relationship between the **Ln(price)** and **price** columns, which indicates that a model to predict the log of price is useful in predicting price.

7. Select the **price** column header, and in the **compare to** drop-down list select **city-mpg**. You will see a scatter plot as shown below.



Note that the relationship between **price** and **city-mpg** does not appear to be linear.

8. Select the **Ln(price)** column, and in the **compare to** drop-down list select **city-mpg**. You will see a scatter plot as shown below:



Note that, with the exception of some outliers at the high end of **city-mpg**, the relationship between log of price and **city-mpg** appears to be roughly linear. The logarithmic transformation has 'flattened' the curvature of **price** vs. **city-mpg**, which is the reason for applying this transformation.

Note: The logarithmic transformation applied to price was an initial guess. There is no reason this transformation is the best possible. You may wish to try some other transformations and compare the results to the logarithmic transformation.

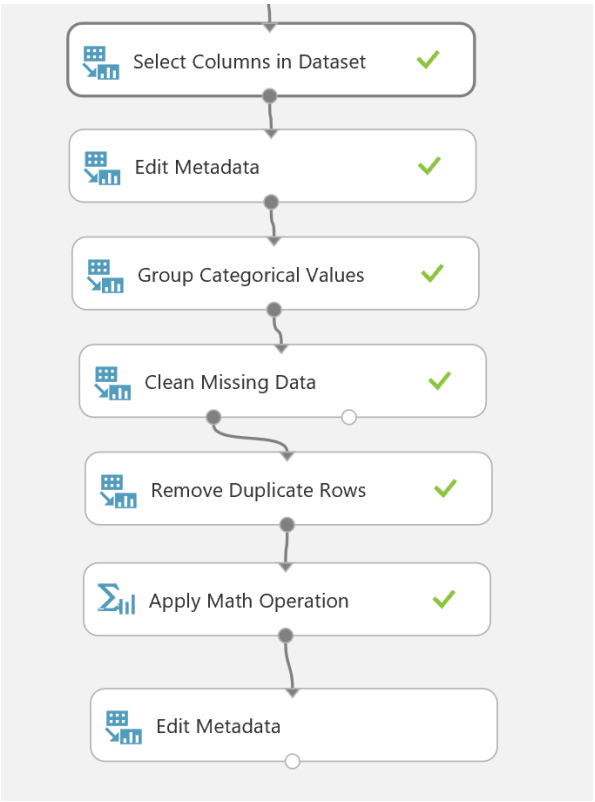
The name of the **Ln(price)** column, and other columns that include a "-" character can cause confusion in some scripting languages, so you plan to rename these columns.

Rename Columns

1. Add another **Edit Metadata** module to the experiment beneath the **Apply Math Operation** module, and connect the output of the **Apply Math Operation** module to its input.
2. Set the properties of the **Edit Metadata** module as follows:
    - **Column:** Launch the column select and include the following columns (in this order):
        - fuel-type
        - num-of-doors
        - body-style
        - drive-wheels
        - engine-location
        - wheel-base
        - curb-weight
        - engine-type
        - num-of-cylinders
        - engine-size
        - fuel-system
        - compression-ratio

    - peak-rpm
    - city-mpg
    - highway-mpg
    - Ln(price)
- **Data type:** Unchanged
- **Categorical:** Unchanged
- **Fields:** Unchanged
- **New column names** (*comma-delimited on a single line*) : fueltype,doors,body,drive,engineloc,wheelbase,weight,enginetype,cylinders,enginesize, fuelsystem,compression,rpm,citympg,highwaympg,lnprice

3. Ensure that the lower part of your experiment looks similar to this:



4. Save and run the experiment.
5. When the experiment has finished running, visualize the output of the **Edit Metadata** module and verify that the columns have been renamed.

Manipulate Data and Metadata with Custom Code

You may already be familiar with a scripting language such as SQL, R, or Python; and you may be using this language to explore data outside of Azure ML. You can use your custom data manipulation code within Azure ML experiments, making it easy to switch between local data analysis environments and Azure ML.

Transform Data with SQL

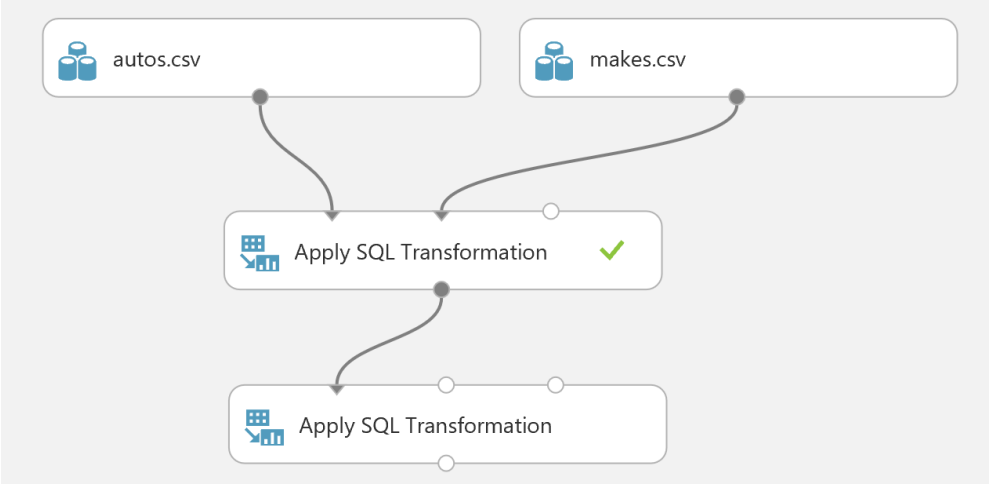
To manipulate the data and metadata for the automobile dataset by using SQL, complete the following tasks.

1. In the **Autos** experiment, visualize the output of the module you used to join the **autos.csv** and **makes.csv** datasets, and note the following:
    - The dataset contains columns named **symboling**, **normalized-losses**, and **make-id**. You have determined that these columns are not useful in building a predictive model for automobile prices, so you have decided to remove them.
    - The **num-of-cylinders** column is a string feature with seven unique values. The histogram shows that the frequency of these values is spread unevenly, with very few instances of some values – so it may be more useful to group these into a smaller number of categories.
    - The **stroke**, **horsepower**, and **peak-rpm** columns contains missing values. Note the total number of rows in the dataset, which includes the rows containing missing values and potentially also includes rows that are duplicated.
2. Search for an **Apply SQL Transformation** module and drag it to the canvas under the module you used to join the **autos.csv** and **makes.csv** datasets.
3. Connect the output from the module that joins the datasets to the left-most input of the **Apply SQL Transformation** module.
4. Select the **Apply SQL Transformation** module, and in the **Properties** pane, replace the existing SQL Query Script with the following code (which you can copy and paste from **PrepData.sql** in the lab files folder for this module):

```
SELECT DISTINCT [fuel-type] AS fueltype,
       [aspiration],
       [num-of-doors] AS doors,
       [body-style] AS body,
       [drive-wheels] AS drive,
       [engine-location] AS engineloc,
       [wheel-base] AS wheelbase,
       [length],
       [width],
       [height],
       [curb-weight] AS weight,
       [engine-type] AS enginetype,
       CASE
        WHEN [num-of-cylinders] IN ('two', 'three', 'four')
          THEN 'four-or-less'
        WHEN [num-of-cylinders] IN ('five', 'six')
          THEN 'five-six'
        WHEN [num-of-cylinders] IN ('eight', 'twelve')
          THEN 'eight-twelve'
        ELSE 'other'
       END
       AS cylinders,
       [engine-size] AS enginesize,
       [fuel-system] AS fuelsystem,
       [bore],
       [stroke],
       [compression-ratio] AS compression,
       [horsepower],
       [peak-rpm] AS rpm,
       [city-mpg] AS citympg,
       [highway-mpg] AS highwaympg,
       [price],
```

```
        [make],
        log([price]) AS lnprice
FROM T1
WHERE   [fuel-type] IS NOT NULL AND
        [aspiration] IS NOT NULL AND
        [num-of-doors] IS NOT NULL AND
        [body-style] IS NOT NULL AND
        [drive-wheels] IS NOT NULL AND
        [engine-location] IS NOT NULL AND
        [wheel-base] IS NOT NULL AND
        [length] IS NOT NULL AND
        [width] IS NOT NULL AND
        [height] IS NOT NULL AND
        [curb-weight] IS NOT NULL AND
        [engine-type] IS NOT NULL AND
        [num-of-cylinders] IS NOT NULL AND
        [engine-size] IS NOT NULL AND
        [fuel-system] IS NOT NULL AND
        [bore] IS NOT NULL AND
        [stroke] IS NOT NULL AND
        [compression-ratio] IS NOT NULL AND
        [horsepower] IS NOT NULL AND
        [peak-rpm] IS NOT NULL AND
        [city-mpg] IS NOT NULL AND
        [highway-mpg] IS NOT NULL AND
        [price] IS NOT NULL AND
        [make] IS NOT NULL;
```

5. Verify that your experiment resembles this:



6. **Save** and **run** the experiment.
7. When the experiment has finished running, visualize the output of the **Apply SQL Transformation** module and note that:
    - The **symboling**, **normalized-losses**, and **make-id** columns are no longer included in the dataset.
    - Some remaining columns have been renamed.
    - The **cylinders** column now contains three unique values based on the categorical levels you specified.

- The total number of rows has been reduced by eliminating rows that contain missing values and duplicate rows.
- A new column named **lnprice**, containing the log of the **price** column value has been created.

Transform Data with R

To manipulate the data and metadata for the automobile dataset by R, complete the following tasks.

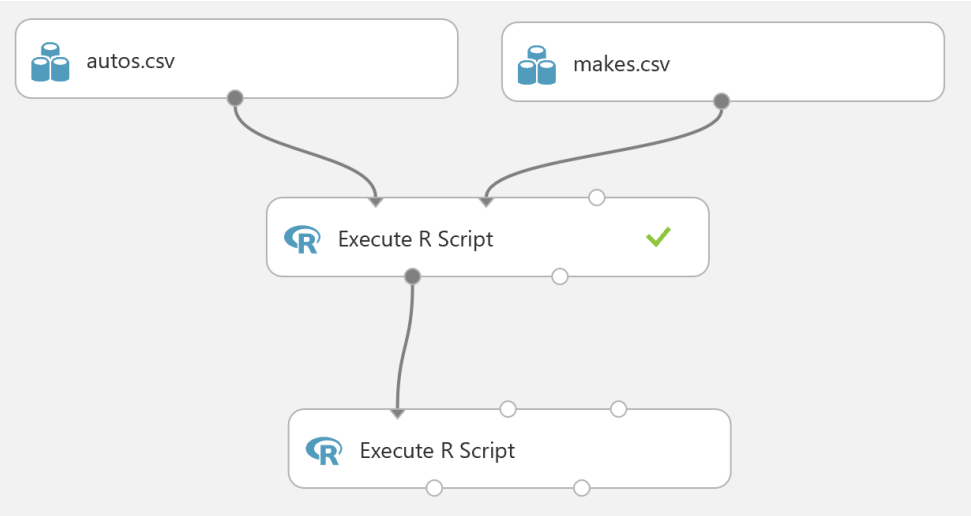
R is designed to support statistical data exploration and modeling, and is a popular choice of language for data scientists. If you prefer to work with SQL or Python, you can skip this procedure and complete the equivalent SQL procedure above or Python procedure below.

1. In the **Autos** experiment, visualize the output of the module you used to join the **autos.csv** and **makes.csv** datasets, and note the following:
    - The dataset contains columns named **symboling**, **normalized-losses**, and **make-id**. You have determined that these columns are not useful in building a predictive model for automobile prices, so you have decided to remove them.
    - The **num-of-cylinders** column is a string feature with seven unique values. The histogram shows that the frequency of these values is spread unevenly, with very few instances of some values – so it may be more useful to group these into a smaller number of categories.
    - The **stroke**, **horsepower**, and **peak-rpm** columns contains missing values. Note the total number of rows in the dataset, which includes the rows containing missing values and potentially also includes rows that are duplicated.
2. Search for an **Execute R Script** module and drag it to the canvas under the module you used to join the **autos.csv** and **makes.csv** datasets.
3. Connect the output from the module that joins the datasets to the left-most input of the **Execute R Script** module.
4. Select the **Execute R Script** module, and in the **Properties** pane, replace the existing R Script with the following code (which you can copy and paste from **PrepData.R** in the lab files folder for this module):

```
## Function to clean and prepare the auto data
prep.auto <- function(df, col.names){
  require(dplyr) ## Make sure dplyr is loaded

  ## set the column names.
  names(df) <- col.names

  ## Eliminate unneeded columns
  df <- df[,!(names(df) %in%
                c('symboling', 'normalizedlosses', 'makeid'))]

  ## Coerce some character columns to numeric
  ## Uncomment if NOT in Azure ML
  # cols <- c('price', 'bore', 'stroke', 'horsepower', 'rpm')
  # df[, cols] <- lapply(df[, cols], as.numeric)

  ## Add a log transformed column for price using dplyr mutate
  df <- df %>% mutate(lnprice = log(price))
```

```
  ## Remove rows with NAs
  df <- df[complete.cases(df), ]

  ## Remove duplicate rows
  df <- df %>% filter (! duplicated(df,))

  ## Consolidate the number of cylinders
  df <- df %>%
    mutate(cylinders = ifelse(cylinders %in%
        c("four", "three", "two"), "three-four",
           ifelse(cylinders %in%
             c("five", "six"), "five-six", "eight-twelve")))
  df
}

## Define column names
col.names <- c('symboling', 'normalizedlosses', 'makeid', 'fueltype',
'aspiration', 'doors',
               'body', 'drive', 'engineloc', 'wheelbase',
               'length', 'width', 'height', 'weight', 'enginetype',
               'cylinders', 'enginesize', 'fuelsystem', 'bore',
'stroke',
               'compression', 'horsepower', 'rpm', 'citympg',
               'highwaympg', 'price', 'make')

## R code to prep the auto data
autos.price <- maml.mapInputPort(1) # read autos data frame from port 1
out <- prep.auto(autos.price, col.names) # Transform the data
maml.mapOutputPort("out") ## Output the prepared data frame
```

5. Verify that your experiment resembles this:



6. **Save** and **run** the experiment.
7. When the experiment has finished running, visualize the output of the **Execute R Script** module and note that:
    - The **symboling**, **normalized-losses**, and **make-id** columns are no longer included in the dataset.
    - Some remaining columns have been renamed.

- The **cylinders** column now contains three unique values based on the categorical levels you specified.
- The total number of rows has been reduced by eliminating rows that contain missing values and duplicate rows.
- A new column named **lnprice**, containing the log of the **price** column value has been created.

Transform Data with Python

To manipulate the data and metadata for the automobile dataset by using Python, complete the following tasks.

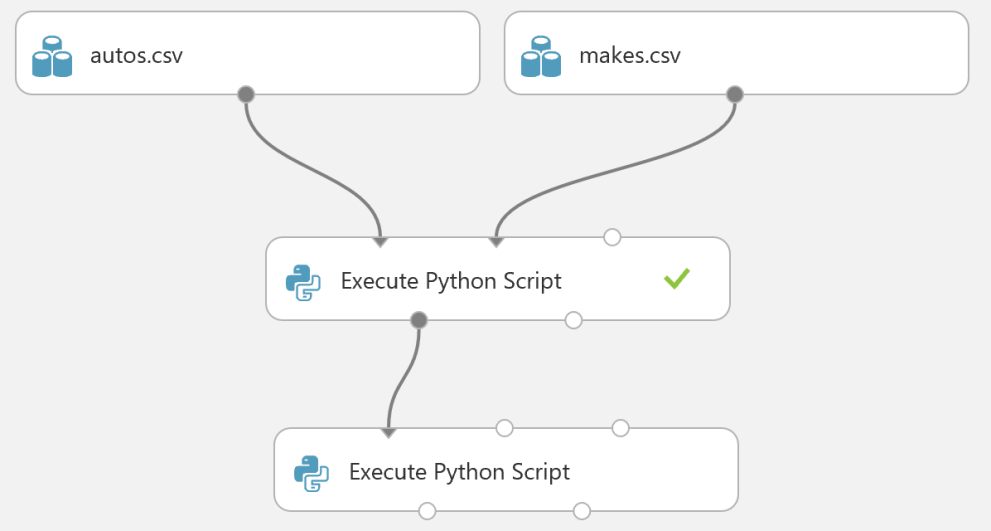
Python is a general purpose programming language that has comprehensive support for data manipulation, analysis, and modeling. If you prefer to work with SQL or R, you can skip this procedure and complete the equivalent SQL or R procedure above.

1. In the **Autos** experiment, visualize the output of the module you used to join the **autos.csv** and **makes.csv** datasets, and note the following:
    - The dataset contains columns named **symboling**, **normalized-losses**, and **make-id**. You have determined that these columns are not useful in building a predictive model for automobile prices, so you have decided to remove them.
    - The **num-of-cylinders** column is a string feature with seven unique values. The histogram shows that the frequency of these values is spread unevenly, with very few instances of some values – so it may be more useful to group these into a smaller number of categories.
    - The **stroke**, **horsepower**, and **peak-rpm** columns contains missing values. Note the total number of rows in the dataset, which includes the rows containing missing values and potentially also includes rows that are duplicated.
2. Search for an **Execute Python Script** module and drag it to the canvas under the module you used to join the **autos.csv** and **makes.csv** datasets.
3. Connect the output from the module that joins the datasets to the left-most input of the **Execute Python Script** module.
4. Select the **Execute Python Script** module, and in the **Properties** pane, replace the existing Python Script with the following code (which you can copy and paste from **PrepData.py** in the lab files folder for this module):

```
## Function to prepare the automotive data
def prep_auto(df, col_names):
    import pandas as pd
    import numpy as np

    ## Assign names to columns
    df.columns = col_names

    ## Drop unneeded columns
    drop_list = ['symboling', 'normalizedlosses', 'makeid']
    df.drop(drop_list, axis = 1, inplace = True)

    ## Remove rows with missing values
    df = df[~pd.isnull(df).any(axis=1)]
```

```
    ## Add a log transformed column for price
    df['lnprice'] = np.log(df['price'].as_matrix())

    ## Remove duplicate rows
    df.drop_duplicates(inplace = True)

    ## Create a column with new levels for the number of cylinders
    df['cylinders'] = ['four-or-less' if x in ['two', 'three', 'four']
else
                                   ('five-six' if x in ['five', 'six']
else
                                   'eight-twelve') for x in
df['cylinders']]
    return df

def azureml_main(df):
    ## Define column names
    col_names = ['symboling', 'normalizedlosses', 'makeid', 'fueltype',
'aspiration', 'doors',
       'body', 'drive', 'engineloc', 'wheelbase',
       'length', 'width', 'height', 'weight', 'enginetype',
       'cylinders', 'enginesize', 'fuelsystem', 'bore', 'stroke',
       'compression', 'horsepower', 'rpm', 'citympg',
       'highwaympg', 'price', 'make']

    ## Call function to prep auto data and return
    return prep_auto(df, col_names)
```

5. Verify that your experiment resembles this:



6. **Save** and **run** the experiment.
7. When the experiment has finished running, visualize the output of the **Execute Python Script** module and note that:
    - The **symboling**, **normalized-losses**, and **make-id** columns are no longer included in the dataset.
    - Some remaining columns have been renamed.

- The **cylinders** column now contains three unique values based on the categorical levels you specified.
- The total number of rows has been reduced by eliminating rows that contain missing values and duplicate rows.
- A new column named **lnprice**, containing the log of the **price** column value has been created.

Finding and Filtering Outliers

In these exercises, you will apply methods for detecting and filtering outliers in the automotive dataset. Detecting and filtering of outliers is often a necessary step in preparing a dataset for analysis. These exercises will give you hands-on experience with this process.

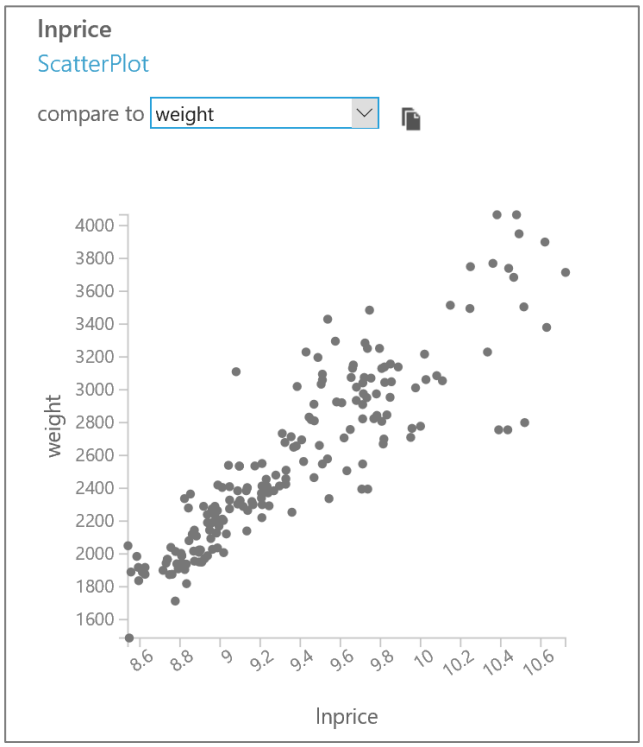
Detecting and classifying outliers can be challenging. Typically, several iterations are needed to identify and confirm the presence of outliers. Further, when identifying outliers, some caution is required. Be sure you are filter outliers, rather than interesting or unusual values that may be important in to your analysis.

Find and Visualize Outliers

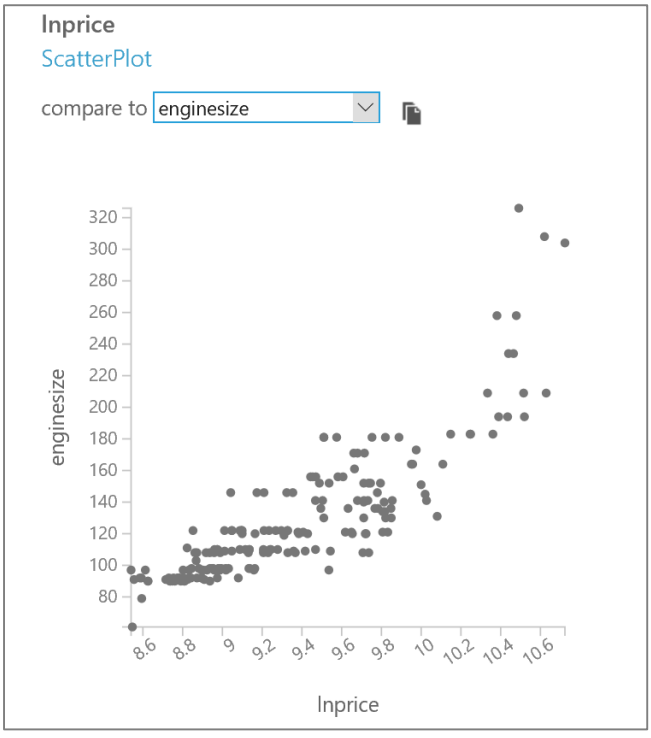
In this procedure, you will use native Azure ML modules and R or Python to find and visualize outliers. Visualization is a useful way to explore your data and detect potential outliers. While the native data visualization capabilities in Azure ML provide a useful starting point for identifying potential outliers, you will typically need to use R or Python code to verify them.

Identify Potential Outliers with Azure ML

1. In the **Autos** experiment, visualize the output of the final module (which should show the dataset after all metadata and data cleansing transformations have been performed).
2. Select the **lnprice** column, and in the **compare to** list, select the **weight** column. Then review the resulting scatter-plot, which should look like this:

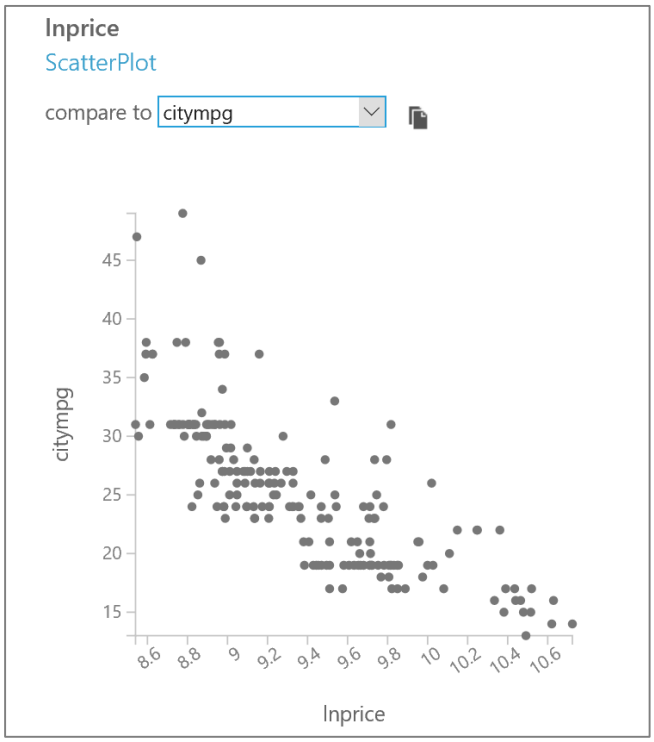


3. Note that autos with a **weight** of greater than 3500 seem to be in a group by themselves, when plotted against **lnprice**. On the same plot notice the three unusual points around an **lnprice** of 10.5 and **weight** of 2750.
4. In the **compare to** list, select **enginesize** and review the resulting scatter-plot, which should look like this:



5. Note that autos with an **enginesize** over 190 appear to be in a group by themselves when plotted against **lnprice**.

6. In the **compare to** list, select **citympg** and review the resulting scatter-plot, which should look like this:



7. Note that autos with a **citympg** over 40 appear to be in a group by themselves when plotted against **lnprice**.

Note: For the sake of brevity you will only work with three features of the dataset identified in this procedure. These three features were found to be important by a lengthy iterative process. If you are interested in an additional challenge, try searching other features for outliers.

Visualize Outliers with R

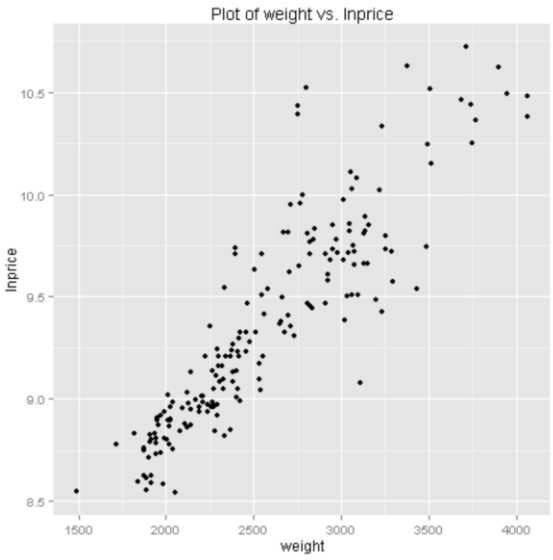
Note: If you prefer to work with Python, skip this procedure and complete the next procedure, *Visualize Outliers with Python*.

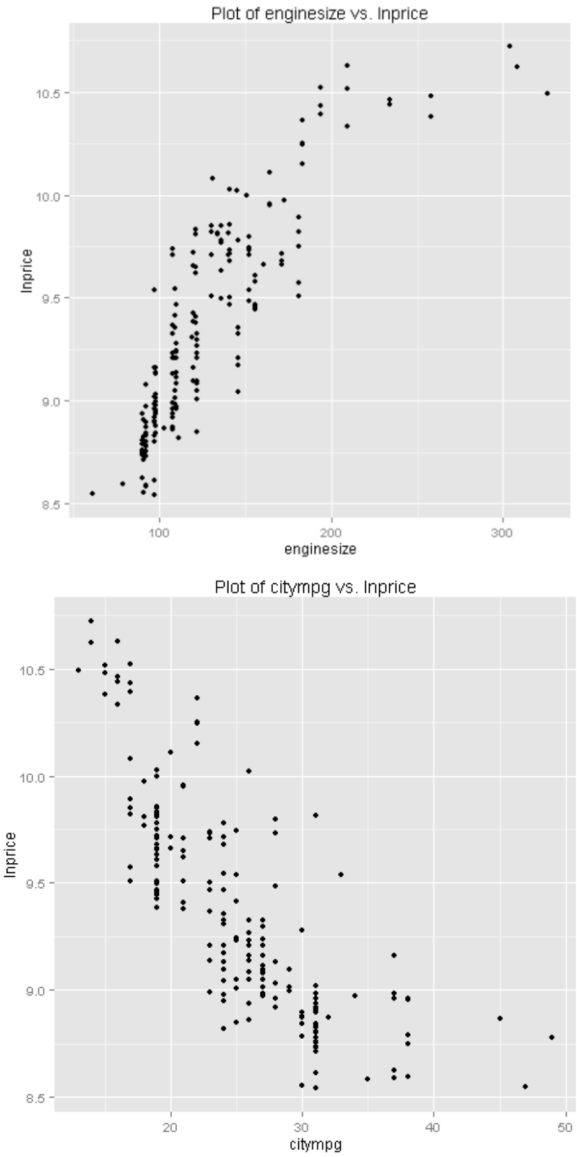
1. In the **Autos** experiment, drag an **Execute R Script** module onto the canvas and connect the output of the data flow so far to the left-most input of the new **Execute R Script** module.
2. Select the **Execute R Script** module, and in the **Properties** pane, replace the existing R Script with the following code (which you can copy and paste from **PlotCols.R** in the lab files folder for this module):

```
vis.simple <- function(col1, df = auto.price, col2 = 'lnprice'){
  require(ggplot2) # make sure ggplot2 is loaded
  title = paste('Plot of', col1, 'vs.', col2) # create the title text
string
  ggplot(df, aes_string(col1, col2)) +
    geom_point() +
    ggtitle(title)
}

## Scatter plots of select columns
auto.price <- maml.mapInputPort(1) # read autos data frame from port 1
```

```
plot.cols <- c("weight",
               "enginesize",
               "citympg")
lapply(plot.cols, vis.simple) # apply the plot columns to the function
maml.mapOutputPort("auto.price") ## Output the data frame
```

3. Save and run the experiment.
4. When your experiment has finished running, visualize the **R Device (Dataset)** (right-most) output of the **Execute R Script** module containing the code to plot the columns. Scroll down to the plots which should appear as shown here:





Examine these plots, noting that they match the plots you visualized in Azure ML previously.

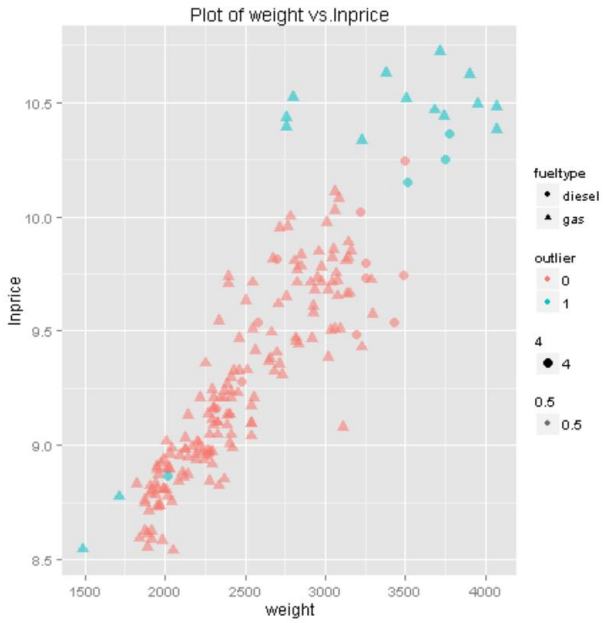
Up to this point you have examined the relationships in the data and identified some potential outliers. You now need to verify that the aforementioned thresholds are indeed filtering outliers.

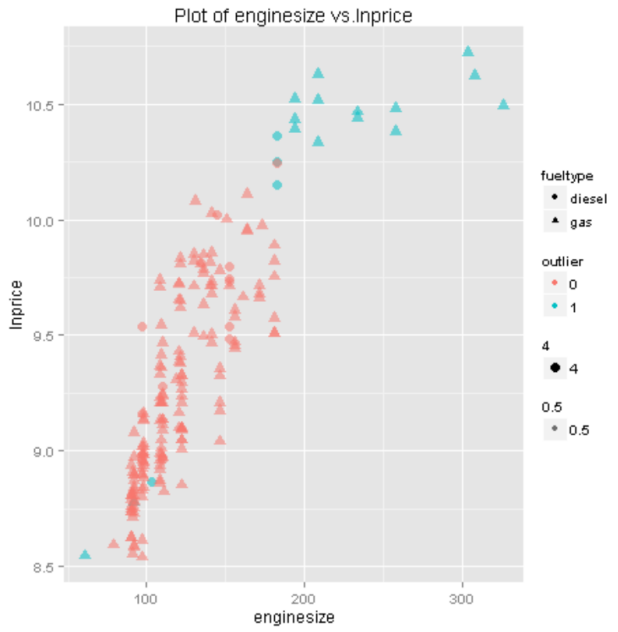
5. Select the **Execute R Script** module containing the code to plot the columns, and replace the existing code with the following code (which you can copy and paste from **PlotOutliers.R** in the lab files folder for this module):

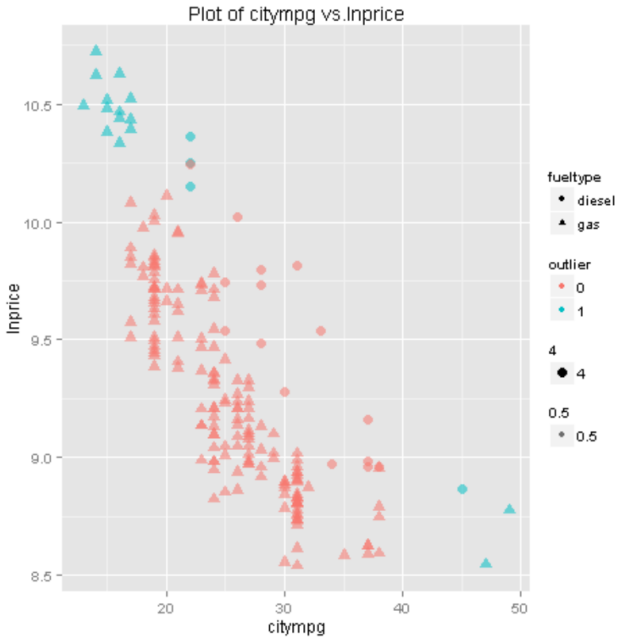
```
vis.outlier <- function(col1 = 'citympg'){
  require(ggplot2) # make sure ggplot2 is loaded
  ## convert character columns to factors for plotting
  auto.price[, "outlier"] <- as.factor(auto.price[, "outlier"])
  auto.price[, "fueltype"] <- as.factor(auto.price[, "fueltype"])
  title = paste('Plot of', col1, 'vs.lnprice') # character string title
  ggplot(auto.price, aes_string(col1, 'lnprice')) +
    geom_point(aes(color = outlier,
```

```
                    shape = fueltype,
                    alpha = 0.5, size = 4)) +
    ggtitle(title)
}

id.outlier <- function(df){
  ## Use ifelse to filter for outliers
  df[, "outlier"] <- ifelse(df[,"enginesize"] > 190 |
                              df[, "weight"] > 3500|
                              df[, "citympg"] > 40, '1', '0')
  df
}

## ID and plot outliers
auto.price <- maml.mapInputPort(1) # read autos data frame from port 1
require(dplyr) # make sure dplyr is loaded
## set the columns to plot
plot.cols <- c("weight",
               "enginesize",
               "citympg")
auto.price <- id.outlier(auto.price) # id outliers
lapply(plot.cols, vis.outlier)  # plot the results
out <- auto.price %>% filter(outlier == 1) # return the outliers
maml.mapOutputPort("out") ## Output the data frame
```

6. Examine this code, and note that the **vis.outlier** function plots **col** vs**. lnprice**, with the point color determined by **outlier**, and the shape determined by the **fueltype**. The **id.outlier** function marks outliers using the R **ifelse** function. Outliers are extracted using the dplyr **filter** function and output. Review the comments in the code to understand the details.
7. Save and run the experiment.
8. When your experiment has finished running, visualize the **R Device (Dataset)** (right-most) output of the **Execute R Script** module containing the code to plot the outliers. Scroll down to the plots which should appear as shown here:







9. Examine these plots, paying attention to the values marked as outliers. In particular notice:
    - Outliers on the plots of **weight** vs. **lnprice** and **enginesize** vs. **lnprice** cluster at the upper right of the plot. Some other points are in the lower left of these plots stand out, when the shape (**fueltype**) is taken into account.
    - The outliers are nicely clustered in the upper left and lower right of the plot of **citympg** vs. **lnprice**.

    These observations help to confirm the hypothesis that the points marked are indeed outliers.

10. To further investigate these potential outliers, visualize the **Results dataset** (left) output of the **Execute R Script** module. Scroll across until you see the **make** of each car.

Examine these results, paying particular attention to **make**. It appears these autos fall into two groups. One group are luxury brands. The other group comprises autos built for exceptional fuel economy. Once again, this observation helps confirm the hypothesis that these autos are outliers. Any analytic or machine learning model may need to account for these observations to produce accurate predictions of auto price.

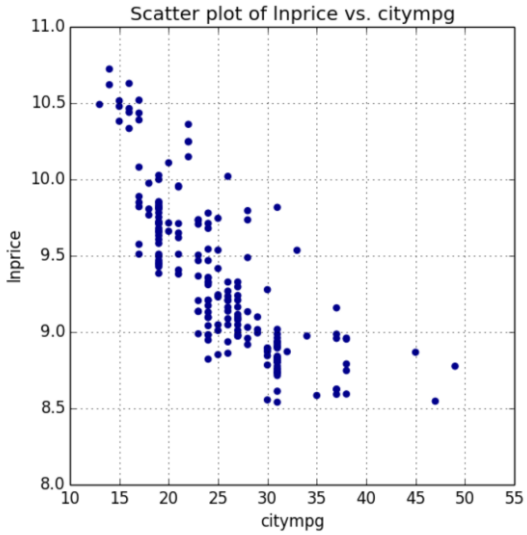
Visualize Outliers with Python

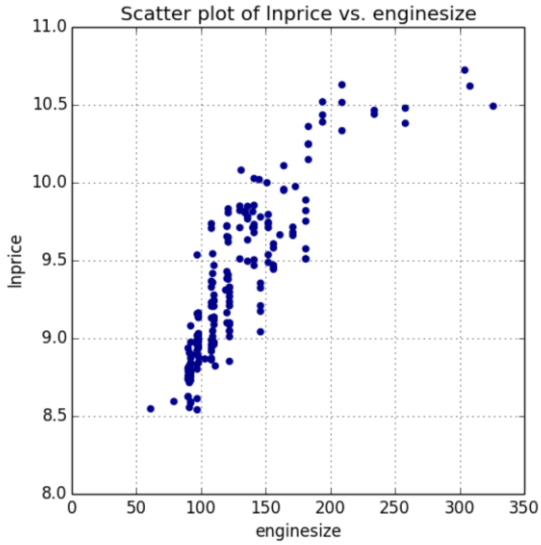
Note: If you prefer to work with R, skip this procedure and complete the previous procedure, *Visualize Outliers with R*.

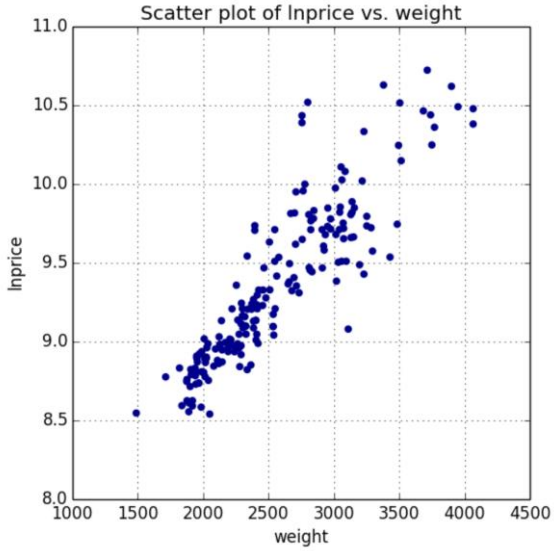
1. In the **Autos** experiment, drag an **Execute Python Script** module onto the canvas and connect the output of the data flow so far to the left-most input of the new **Execute Python Script** module.
2. Select the **Execute Python Script** module, and in the **Properties** pane, replace the existing Python Script with the following code (which you can copy and paste from **PlotCols.py** in the lab files folder for this module):

```
def auto_scatter_simple(df, plot_cols):
    import matplotlib.pyplot as plt
    ## Loop over the columns
    for col in plot_cols:
        fig = plt.figure(figsize=(6, 6))
        ax = fig.gca()
        ## simple scatter plot
        df.plot(kind = 'scatter', x = col, y = 'lnprice',
                            ax = ax, color = 'DarkBlue')
        ax.set_title('Scatter plot of price vs. ' + col)
        fig.savefig('scatter_' + col + '.png')
    return plot_cols

def azureml_main(df):
    import matplotlib
    matplotlib.use('agg')
    ## Define plot columns
    plot_cols = ["weight",
                 "enginesize",
                 "citympg"]
    auto_scatter_simple(df, plot_cols) ## Create plots
    return df
```

3. Save and run the experiment.
4. When your experiment has finished running, visualize the **Python Device (Dataset)** (right-most) output of the **Execute Python Script** module containing the code to plot the columns. Scroll down to the plots which should appear as shown here:







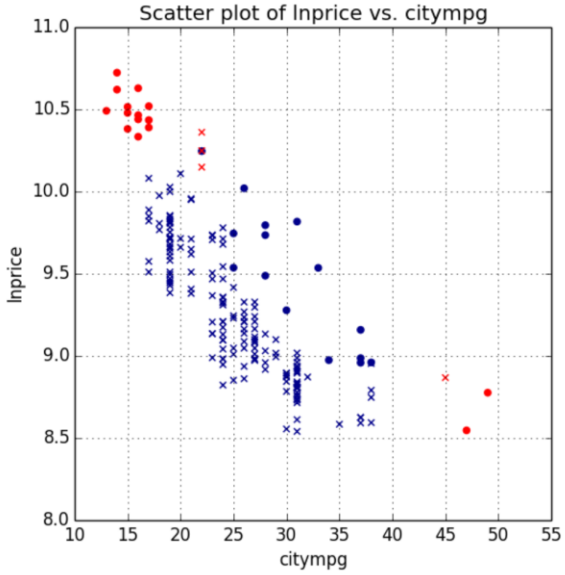
5. Examine these plots, noting that they match the plots you visualized in Azure ML previously.

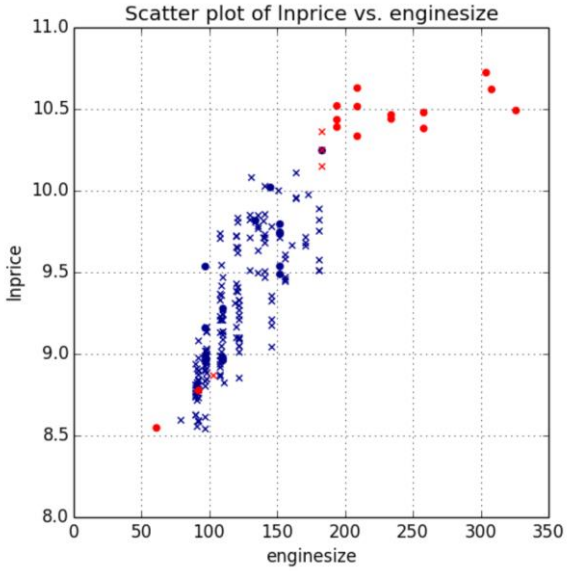
Up to this point you have examined the relationships in the data and identified some potential outliers. You now need to verify that the aforementioned thresholds are indeed filtering outliers.

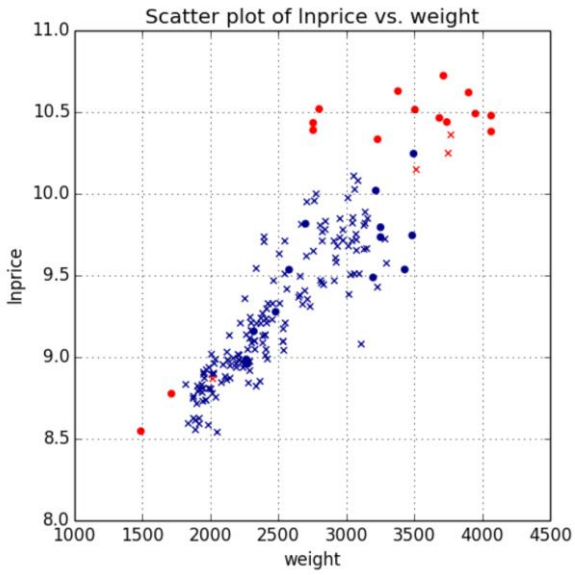
6. Select the **Execute Python Script** module containing the code to plot the columns, and replace the existing code with the following code (which you can copy and paste from **PlotOutliers.py** in the lab files folder for this module):

```
def id_outlier(df):
    ## Create a vector of 0 of length equal to the number of rows
    temp = [0] * df.shape[0]
    ## test each outlier condition and mark with a 1 as required
    for i, x in enumerate(df['enginesize']):
        if (x > 190): temp[i] = 1
    for i, x in enumerate(df['weight']):
        if (x > 3500): temp[i] = 1
    for i, x in enumerate(df['citympg']):
        if (x > 40): temp[i] = 1
    df['outlier'] = temp # append a column to the data frame
    return df

def auto_scatter_outlier(df, plot_cols):
    import matplotlib.pyplot as plt
    outlier = [0,0,1,1] # Vector of outlier indicators
    fuel = ['gas','diesel','gas','diesel']  # vector of fuel types
    color = ['DarkBlue','DarkBlue','Red','Red'] # vector of color
choices for plot
    marker = ['x','o','o','x'] # vector of shape choices for plot
    for col in plot_cols: # loop over the columns
        fig = plt.figure(figsize=(6, 6))
        ax = fig.gca()
        ## Loop over the zip of the four vectors an subset the data and
        ## create the plot using the aesthetics provided
        for o, f, c, m in zip(outlier, fuel, color, marker):
            temp = df.ix[(df['outlier'] == o) & (df['fueltype'] == f)]
            if temp.shape[0] > 0:
                temp.plot(kind = 'scatter', x = col, y = 'lnprice' ,
                          ax = ax, color = c, marker = m)
        ax.set_title('Scatter plot of lnprice vs. ' + col)
        fig.savefig('scatter_' + col + '.png')
    return plot_cols

def azureml_main(df):
    import matplotlib
    matplotlib.use('agg')
    ## Define the plot columns
    plot_cols = ["weight",
                 "enginesize",
                 "citympg"]
    df = id_outlier(df)  # mark outliers
    auto_scatter_outlier(df, plot_cols) # create plots
    df = df[df.outlier == 1] # filter for outliers
    return df
```

7. Examine this code, and note that the **auto_scatter_outlier** function plots **col** vs**. lnprice**, with the point color determined by **outlier**, and the shape determined by the **fueltype**. The **id_outlier** function marks outliers, using the nested **ifelse** statements in a Python list comprehension. Outliers are extracted using a Pandas filter. Review the comments in the code to understand the details.
8. Save and run the experiment.
9. When your experiment has finished running, visualize the **Python Device (Dataset)** (right-most) output of the **Execute Python Script** module containing the code to plot the outliers. Scroll down to the plots which should appear as shown here:







Examine these plots, paying attention to the values marked as outliers. In particular notice:

- Outliers on the plots of **weight** vs. **lnprice** and **enginesize** vs. **lnprice** cluster at the upper right of the plot. Some other points are in the lower left of these plots stand out, when the shape (**fueltype**) is taken into account.
- The outliers are nicely clustered in the upper left and lower right of the plot of **citympg** vs. **lnprice**.

These observations help to confirm the hypothesis that the points marked are indeed outliers.

10. To further investigate these potential outliers, visualize the **Results dataset** (left) output of the **Execute Python Script** module. Scroll across until you see the **make** of each car.

    Examine these results, paying particular attention to **make**. It appears these autos fall into two groups. One group are luxury brands. The other group comprises autos built for exceptional fuel economy. Once again, this observation helps confirm the hypothesis that these autos are outliers. Any analytic or machine learning model may need to account for these observations to produce accurate predictions of auto price.

Filter Outliers

In this procedure, you will filter rows containing outliers from the dataset. You can use any of the following techniques to filter outliers.

The procedures in this section describe how to filter outliers by using the following techniques:

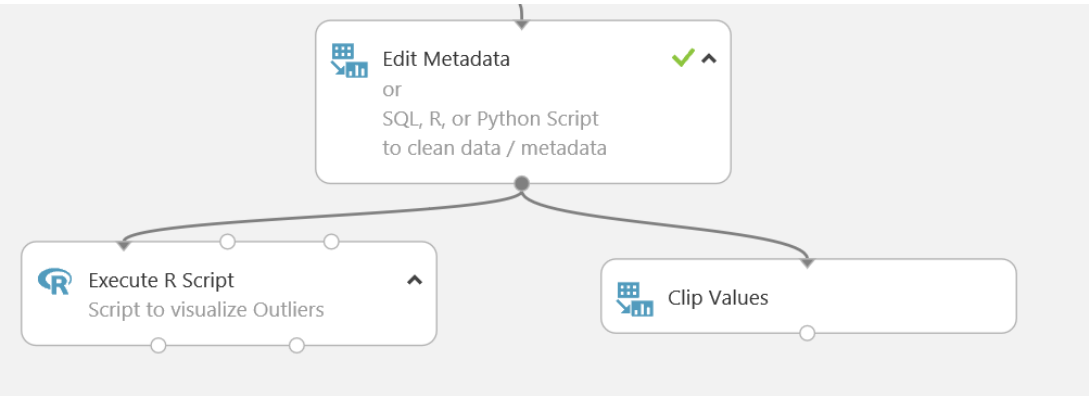
- Built-in Azure ML modules
- A SQL script in an Apply SQL Transformation module
- An R script in an Execute R Script module
- A Python script in an Execute Python module

Note: You need only complete one of the following procedures, but you can try multiple alternative techniques if you wish!

Filter Outliers in Azure ML

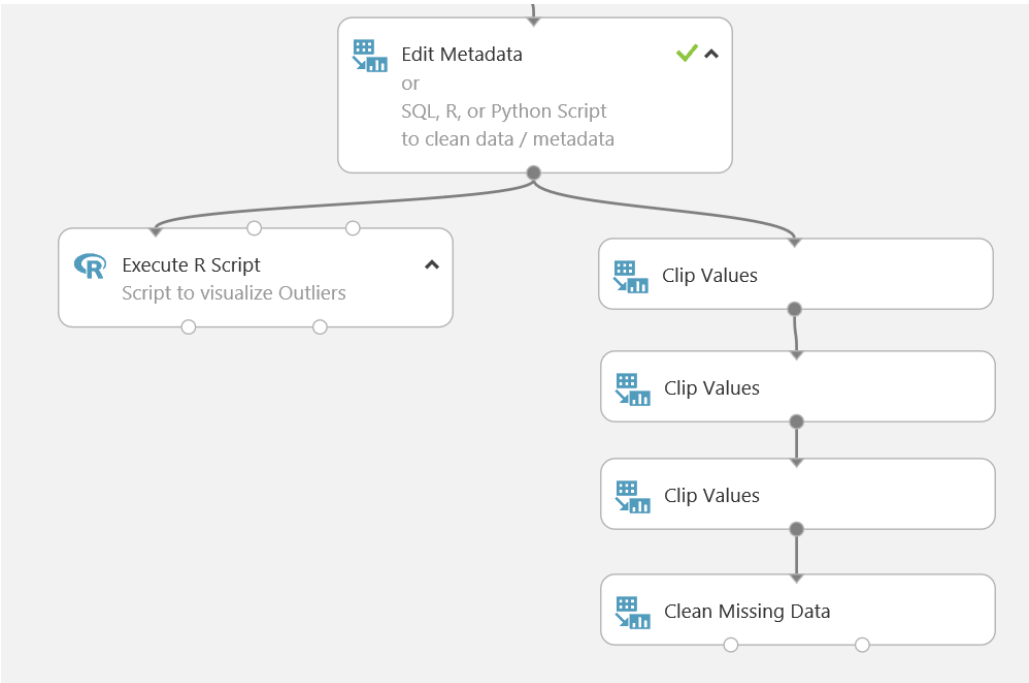
To filter outliers by using built-in Azure ML modules, complete the following procedure.

1. In the **Autos** experiment, drag a **Clip Values** module onto the canvas and connect the output of the last module you used to transform the autos data (*not* the **Execute R Script** or **Execute Python Script** module used to visualize outliers) to the input of the **Clip Values** module as shown here:



2. Set the following properties of the **Clip Values** module:
    - **Set of thresholds**: ClipPeaks
    - **Upper threshold**: Constant
    - **Constant value for upper threshold**: 190
    - **Upper substitute value**: Missing
    - **List of columns**: Use the column selector to select **enginesize**
    - **Overwrite flag**: Checked
    - **Add indicator columns**: unchecked
3. Add another **Clip Values** module and connect the output of the first **Clip Values** module to the input of the second **Clip Values** module.
4. Set the following properties of the second **Clip Values** module:
    - **Set of thresholds**: ClipPeaks
    - **Upper threshold**: Constant
    - **Constant value for upper threshold**: 3500
    - **Upper substitute value**: Missing
    - **List of columns**: Use the column selector to select **weight**
    - **Overwrite flag**: Checked
    - **Add indicator columns**: unchecked
5. Add a third **Clip Values** module and connect the output of the second **Clip Values** module to the input of the third **Clip Values** module.
6. Set the following properties of the third **Clip Values** module:
    - **Set of thresholds**: ClipPeaks
    - **Upper threshold**: Constant
    - **Constant value for upper threshold**: 40
    - **Upper substitute value**: Missing
    - **List of columns**: Use the column selector to select **citympg**
    - **Overwrite flag**: Checked
    - **Add indicator columns**: unchecked

7. To remove the rows, which now have missing values rather than outliers, drag a **Clean Missing Data** module onto the canvas and connect the output of the third **Clip Values** module to the input of the **Clean Missing Data** module.
8. On the properties pane of the **Clean Missing Data** module configure the following parameters:
    - **Columns to be cleaned: Selected columns**: All columns
    - **Minimum missing value ratio**: 0
    - **Maximum missing value ratio**: 1
    - **Cleaning mode**: remove entire row
9. Verify that the bottom section of the experiment looks similar to this:

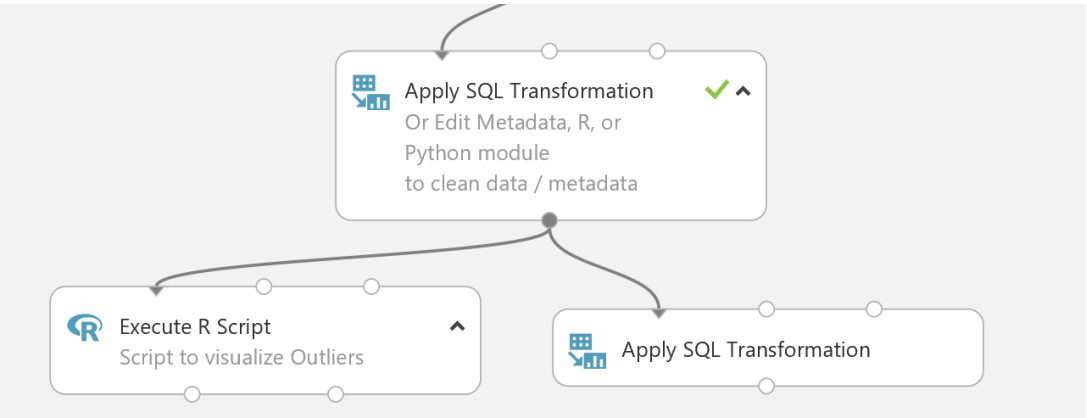


10. Save and run the experiment.
11. When the experiment as finished visualize the **Cleaned Dataset** (left) output of the **Clean Missing Data** module, and note the number of rows in the dataset. Then select the **Inprice** column header, and use the compare to list to visualize the relationship between **Inprice** and the **enginesize**, **weight**, and **citympg** columns to verify that the outliers are no longer present.

Filter Outliers Using SQL

To filter outliers by using SQL, complete the following procedure.

1. In the **Autos** experiment, drag an **Apply SQL Transform** module onto the canvas and connect the output of the last module you used to transform the autos data (*not* the **Execute R Script** or **Execute Python Script** module used to visualize outliers) to the input of the **Apply SQL Transform** module as shown here:



2. Select the **Apply SQL Transformation** module you just added, and in the **Properties** pane, replace the existing SQL Query Script with the following code (which you can copy and paste from **FilterOutliers.sql** in the lab files folder for this module):

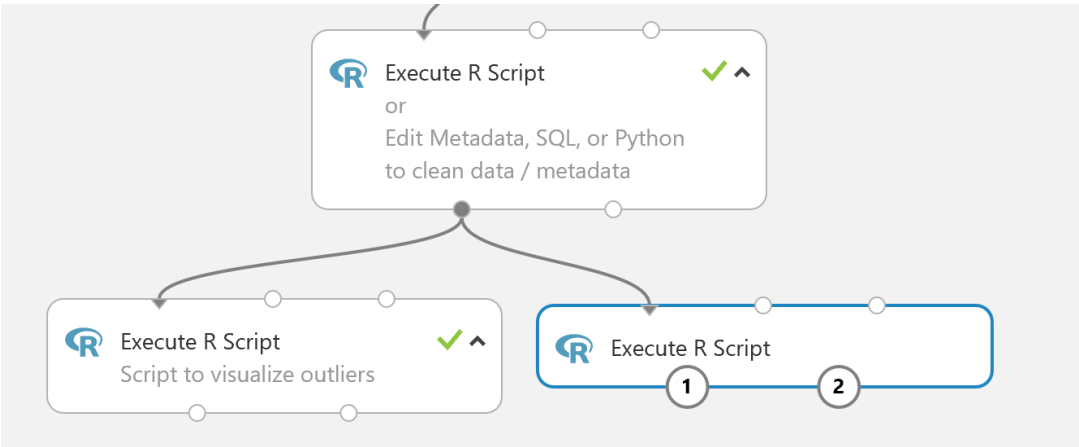
```
SELECT *
FROM t1
WHERE enginesize < 190
AND weight < 3500
AND citympg < 40;
```

3. Save and run the experiment.
4. When the experiment as finished visualize the output of the **Apply SQL Transformation** module, and note the number of rows in the dataset. Then select the **lnprice** column header, and use the compare to list to visualize the relationship between **lnprice** and the **enginesize**, **weight**, and **citympg** columns to verify that the outliers are no longer present.

Filter Outliers Using R

To filter outliers by using R, complete the following procedure.

1. In the **Autos** experiment, drag an **Execute R Script** module onto the canvas and connect the output of the last module you used to transform the autos data (*not* the **Execute R Script** or **Execute Python Script** module used to visualize outliers) to the input of the new **Execute R Script** module as shown here:



2. Select the **Execute R Script** module you just added, and in the **Properties** pane, replace the existing R Script with the following code (which you can copy and paste from **FilterOutliers.R** in the lab files folder for this module):

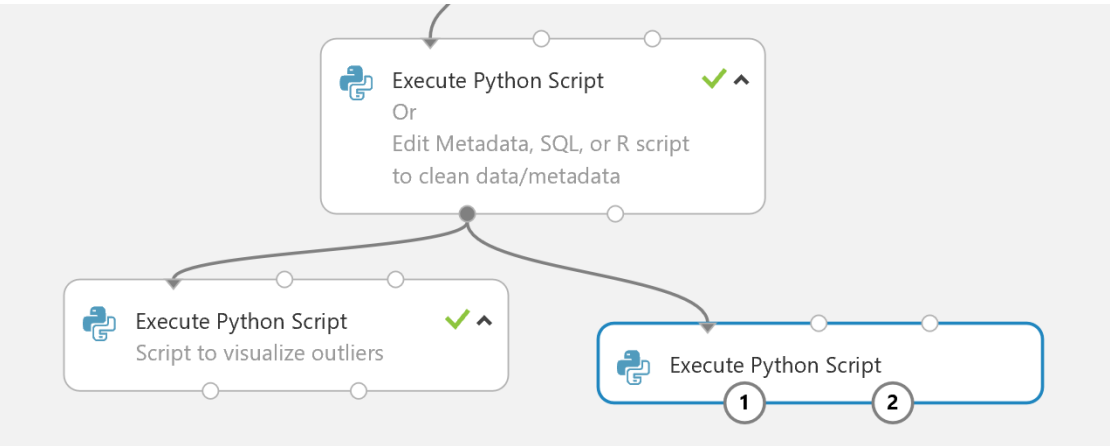
```
id.outlier <- function(df){
  ## Use ifelse to filter for outliers
  df[, "outlier"] <- ifelse(df[,"enginesize"] > 190 |
                              df[, "weight"] > 3500|
                              df[, "citympg"] > 40, '1', '0')
  df
}

## ID and plot outliers
auto.price <- maml.mapInputPort(1) # read autos data frame from port 1
require(dplyr) # make sure dplyr is loaded
auto.price <- id.outlier(auto.price) # mark the outliers
## remove rows where outlier is 1, then remove outlier column
out <- auto.price %>% filter(outlier == 0) %>% subset(select=-outlier)
# filter outliers
maml.mapOutputPort("out") ## Output the data frame
```

3. Save and run the experiment.
4. When the experiment as finished visualize the output of the **Execute R Script** module, and note the number of rows in the dataset. Then select the **Inprice** column header, and use the compare to list to visualize the relationship between **Inprice** and the **enginesize**, **weight**, and **citympg** columns to verify that the outliers are no longer present.

Filter Outliers Using Python

To filter outliers by using Python, complete the following procedure.

1. In the **Autos** experiment, drag an **Execute Python Script** module onto the canvas and connect the output of the last module you used to transform the autos data (*not* the **Execute R Script** or **Execute Python Script** module used to visualize outliers) to the input of the new **Execute Python Script** module as shown here:



2. Select the **Execute Python Script** module you just added, and in the **Properties** pane, replace the existing Python Script with the following code (which you can copy and paste from **FilterOutliers.py** in the lab files folder for this module):

```
def id_outlier(df):
    ## Create a vector of 0 of length equal to the number of rows
    temp = [0] * df.shape[0]
    ## test each outlier condition and mark with a 1 as required
    for i, x in enumerate(df['enginesize']):
        if (x > 190): temp[i] = 1
    for i, x in enumerate(df['weight']):
        if (x > 3500): temp[i] = 1
    for i, x in enumerate(df['citympg']):
        if (x > 40): temp[i] = 1
    df['outlier'] = temp # append a column to the data frame
    return df


def azureml_main(df):
    ## Define the plot columns
    df = id_outlier(df)  # mark outliers
    df = df[df.outlier == 0] # filter for outliers
    df.drop('outlier', axis = 1, inplace = True)
    return df
```

3. Save and run the experiment.
4. When the experiment as finished visualize the output of the **Execute Python Script** module, and note the number of rows in the dataset. Then select the **lnprice** column header, and use the compare to list to visualize the relationship between **lnprice** and the **enginesize**, **weight**, and **citympg** columns to verify that the outliers are no longer present.

Scaling Numeric Columns

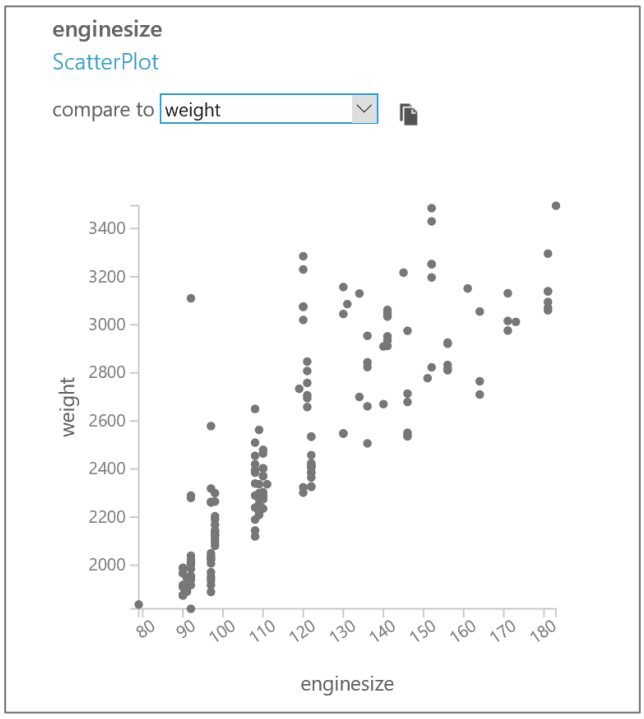
In this exercise, you will scale the numeric columns in the dataset. With the outliers in the numeric features removed, the dataset is now ready to scale. Scaling insures that features with large numeric values will not dominate the training of a machine learning model.

Note: You can use R or Python to scale features by invoking the R **scale** function or the Python Scikit-Learn preprocessing **scale** function. However, in a later lab you will expand this experiment to create a predictive model that will be published as an Azure ML web service. The **Normalize Data** module used in this exercise will be automatically reconfigured to use constants derived from the training data in the experiment when the web service is called, so the built-in module is the preferred approach in this scenario.

Scale Numeric Columns Using Azure ML

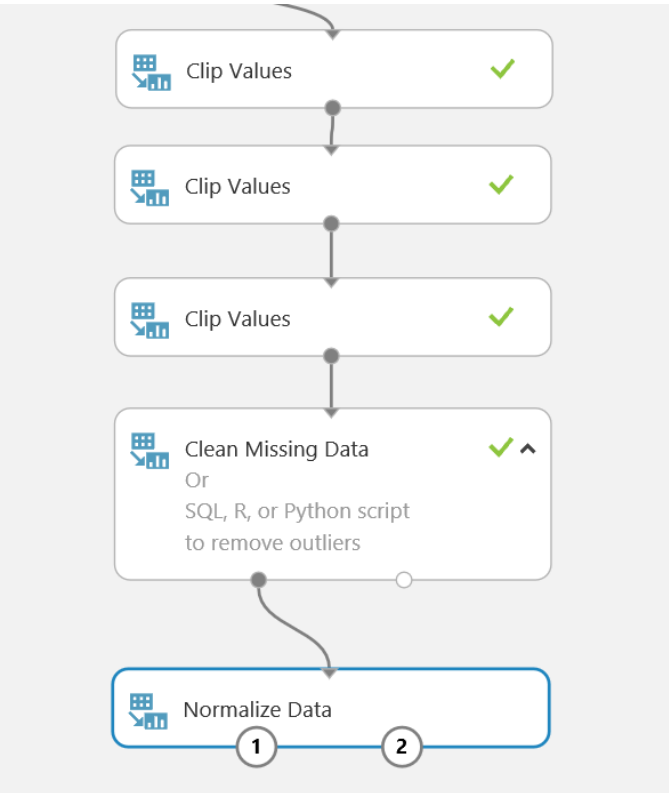
In this exercise you will scale the numeric columns of the dataset using the **Normalize Data** module in Azure ML.

1. In the **Autos** experiment, visualize the output of the last module in the data flow (which should show the automobiles data with rows containing outliers removed).
2. Select the **enginesize** column, and in the compare to list, select the **weight** column to generate a scatter-plot that compares these columns as shown here:

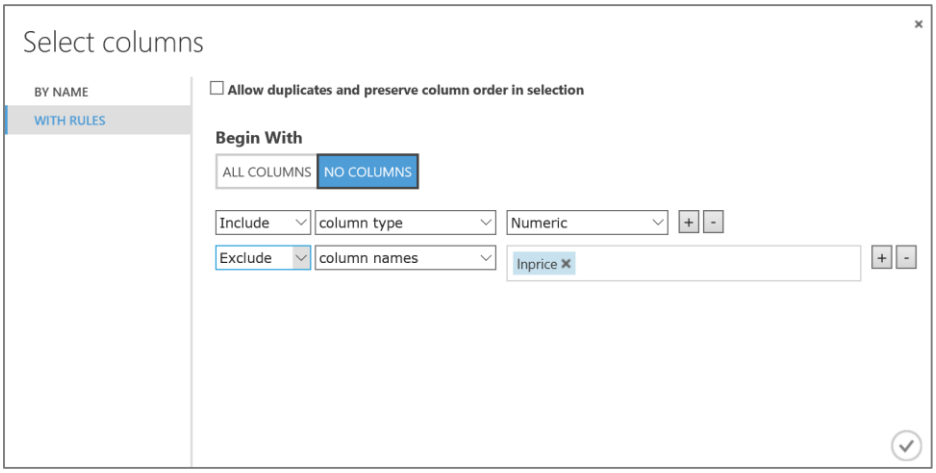


Note that the **weight** and **enginesize** values are on different scales – engine sizes are typically between 80 and 180; curb weights are between 1800 and 3600.

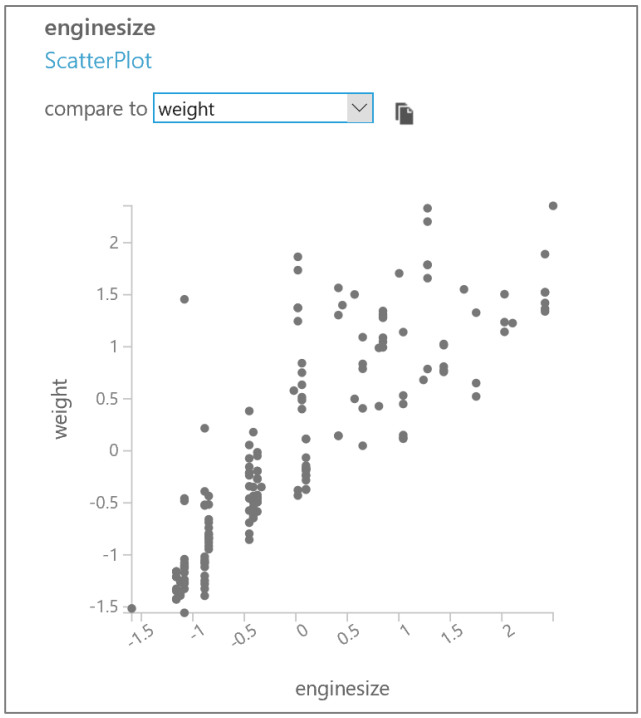
3. Drag a **Normalize Data** module onto the canvas and connect the output from the module you used to remove outliers from the dataset to the input of the **Normalize Data** module, as shown here:



4. On the properties pane of the **Normalize Data** module select the following settings:
    - **Transformation method**: ZScore
    - **Use 0 for constant columns when checked**: Unchecked
    - **Columns to transform**: Include all numeric columns except for **Inprice**, as shown here:



5. Save and run the experiment.
6. When the experiment has finished running visualize the output of the **Normalize Data** module. Select the **enginesize** column, and in the compare to list, select the **weight** column to generate a scatter-plot that compares these columns as shown here:



Note that both engine-size and curb-weight are at approximately the same scale, in the range of about -1.5 to 2.5. Prior to normalization the numerical values of these two columns differed by about an order of magnitude.

Summary

In this lab you have performed common data integration, cleaning, and transformation steps. Specifically, you have:

- Ingested and joined data from multiple sources.
- Deleted unnecessary and redundant columns.
- Consolidated the number of categories of a categorical feature.
- Treated missing values.
- Removed duplicate rows.
- Generated a calculated column
- Located and treated outliers.
- Scaled numeric values.