Microsoft

Data Science Essentials

Lab 4 – Visualizing Data

Overview

In this lab, you will learn how to use R or Python to visualize and explore data. Before creating analytical models, a data scientist must develop an understanding of the properties and relationships in a dataset. There are two goals for data exploration and visualization. First to understand the relationships between the data columns. Second to identify features that may be useful for predicting labels in machine learning projects. Additionally, redundant, collinear features can be identified. Thus, visualization for data exploration is an essential data science skill.

In this lab you will explore two datasets. Your first goal is to explore a dataset that includes information about automobiles, which you want to use to create a solution that predicts the price of an automobile based on its characteristics. This type of predictive modeling, in which you attempt to predict a real numeric value, is known as *regression;* and it will be discussed in more detail later in the course – for now, the focus of this lab is on visually exploring the data to determine which features may be useful in predicting automobile prices.

After exploring the automobile data, you will turn your attention to some adult census data, which you plan to use to classify people as high or low income based information known about them. This technique of predicting whether data entities belong to one class or another is known as *classification*, and will be discussed later in the course.

If you intend to work with R, complete the R exercises. If you plan to work with Python, complete the Python exercises. 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 R or Python, and have not completed Lab 3, you are encouraged to do so now.

What You'll Need

To complete this lab, you will need the following:

- The files for this lab
- An R or Python development environment. The lab steps assume you are using R Studio (for R) or Spyder (for Python).

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

Visualizing Data with R

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

R includes some basic native data visualization functionality, and also supports the **ggplot2** library; which provides extensive graphical capabilities. This library adds a useful language to R for defining visualizations of your data.

Install the ggplot2 Library

1. Start RStudio, and close any open script files from previous sessions.
2. In the **Console** pane, type the following command to ensure **ggplot2** is installed:

```
library(ggplot)
```

3. If no error is displayed, skip to the next procedure. If **ggplot2** was not found, enter the following command to install it. If you are prompted to use a personal library, click **Yes**.

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

Load the Automobiles Dataset

1. In RStudio, open the **LoadAutos.R** file in the folder for this module where you extracted the lab files.
2. In the **LoadAutos.R** code editor pane, review the **read.auto** function shown below:

```
read.auto <- function(path = 'c:/dat203.1x/mod4/'){
  ## Read the csv file
  filePath <- file.path(path, 'Automobile price data _Raw_.csv')
  auto.price <- read.csv(filePath, header = TRUE,
                         stringsAsFactors = FALSE)

  ## Coerce some character columns to numeric
  cols <- c('price', 'bore', 'stroke',
          'horsepower', 'peak.rpm')
  auto.price[, cols] <- lapply(auto.price[, cols], as.numeric)

  ## remove rows with NAs
  auto.price <- auto.price[complete.cases(auto.price), ]

  ## Add a log transformed column for price
  auto.price$lnprice <- log(auto.price$price)

  ## Consolidate the number of cylinders
  auto.price$num.cylinders <-
    ifelse(auto.price$num.of.cylinders %in% c("four", "three"), "three-
four",
           ifelse(auto.price$num.of.cylinders %in% c("five", "six"),
"five-six", "eight-twelve"))

  auto.price
}
```

3. Click the **Source** icon in the upper right above the code editor window to reference the **LoadAutos.R** script in the R console.
4. In the Console window, enter the following code, making sure to set the path to the folder containing the lab files for this module (for example, *C:/dat203.1x/mod4*). Note the output which is a summary of the **auto.price** data frame:

```
auto.price <- read.auto(path = 'PATH-TO-YOUR-LAB-FILES')
str(auto.price)
```

5. Review the output, which includes details of the columns in the dataset as shown below:

```
'data.frame':     195 obs. of  28 variables:
 $ symboling        : int  3 3 1 2 2 2 1 1 1 2 ...
 $ normalized.losses: chr  "?" "?" "?" "164" ...
 $ make             : chr  "alfa-romero" "alfa-romero" "alfa-romero"
 $ fuel.type        : chr  "gas" "gas" "gas" "gas" ...
 $ aspiration       : chr  "std" "std" "std" "std" ...
 $ num.of.doors     : chr  "two" "two" "two" "four" ...
 $ body.style       : chr  "convertible" "convertible" "hatchback" ...
 $ drive.wheels     : chr  "rwd" "rwd" "rwd" "fwd" ...
 $ engine.location  : chr  "front" "front" "front" "front" ...
 $ wheel.base       : num  88.6 88.6 94.5 99.8 99.4 ...
 $ length           : num  169 169 171 177 177 ...
 $ width            : num  64.1 64.1 65.5 66.2 66.4 66.3 71.4 71.4 71.4
 $ height           : num  48.8 48.8 52.4 54.3 54.3 53.1 55.7 55.7 55.9
 $ curb.weight      : int  2548 2548 2823 2337 2824 2507 2844 2954 3086
 $ engine.type      : chr  "dohc" "dohc" "ohcv" "ohc" ...
 $ num.of.cylinders : chr  "four" "four" "six" "four" ...
 $ engine.size      : int  130 130 152 109 136 136 136 136 131 108 ...
 $ fuel.system      : chr  "mpfi" "mpfi" "mpfi" "mpfi" ...
 $ bore             : num  3.47 3.47 2.68 3.19 3.19 3.19 3.19 3.19 3.13
 $ stroke           : num  2.68 2.68 3.47 3.4 3.4 3.4 3.4 3.4 3.4 2.8
 $ compression.ratio: num  9 9 9 10 8 8.5 8.5 8.5 8.3 8.8 ...
 $ horsepower       : num  111 111 154 102 115 110 110 110 140 101 ...
 $ peak.rpm         : num  5000 5000 5000 5500 5500 5500 5500 5500 5500
 $ city.mpg         : int  21 21 19 24 18 19 19 19 17 23 ...
 $ highway.mpg      : int  27 27 26 30 22 25 25 25 20 29 ...
 $ price            : num  13495 16500 16500 13950 17450 ...
 $ lnprice          : num  9.51 9.71 9.71 9.54 9.77 ...
 $ num.cylinders    : chr  "three-four" "three-four" "five-six" ...
```

Note: The missing values or NAs in the dataset will likely produce some warning messages when you run this code. This is normal and can be safely explored.

6. Examine names and data types of the columns in this dataset. There are both numeric columns and character columns. In addition, you can see the first few values of each column.

Note: Character data is typically treated as categorical where each unique string value is a categorical value of the column. When exploring data sets, it is important to understand the relationships between each category and the label values.

Create a Pair-Wise Scatter Plot

In this exercise, you will apply a visualization technique known as a scatter plot matrix to the **auto.price** dataset. Scatter plot matrix methods are widely used to quickly produce a single overall view of the relationships in a dataset.

The scatter plot matrix allows you to examine the relationships between many variables in one view. The alternative to viewing a scatter plot matrix is an examining a large number of scatter plot combinations one at a time. This process 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.

1. In RStudio, open the **ExploreAutos.R** file from the folder containing the lab files for this module.
2. In the code editor window, under the comments, **## Numeric columns**, examine the following code.

```
## Numeric columns to plot
num.cols <- c("wheel.base",
              "width",
              "height",
              "curb.weight",
              "engine.size",
              "bore",
              "compression.ratio",
              "city.mpg",
              "price",
              "lnprice")
```

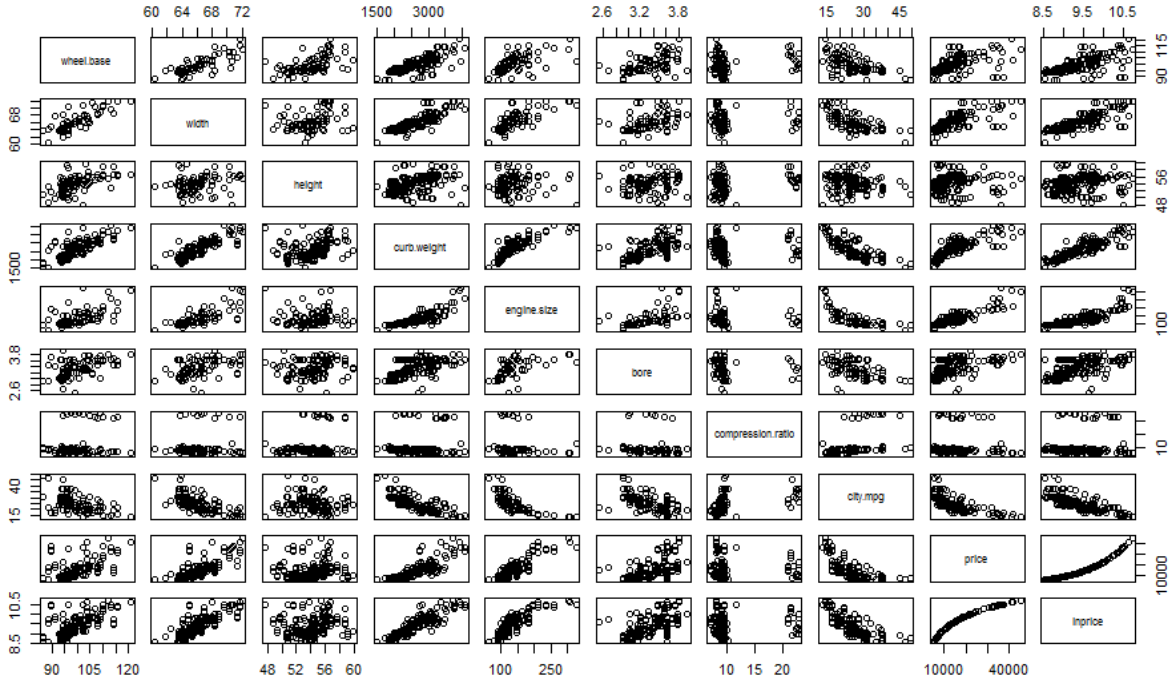
   The **num.cols** object defines the columns with numeric values.

3. Click the **Source** icon to reference the script file in the console.

   Note: When you **Source** a .R file, the code in the file is executed. In this case, objects like **num.cols** are created when the .R file is **Sourced.**

4. In the Console window, enter the following code and view the scatter plot matrix that is generated (shown below):

```
pairs(~ ., data = auto.price[, num.cols])
```



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 is displayed in both possible orientations. Examine scatter plot matrix, which shows plots of each numeric column verses every other numeric column, and note the following:

- Not surprisingly the features, **price** and **lnprice** (log of price) are closely related.
- Many features show significant collinearity, such as **wheel.base**, **width** and **curb.weight**, or **curb.weight** and **engine.size**, and **city,mpg**.
- A number of features show a strong relationship with the label, **price**, such as **city.mpg**, **engine.size**, and **curb.weight**.
- The feature, **compression.ratio**, has two distinct groupings of values, apparently corresponding to diesel (high compression ratio) and gasoline (low compression ratio) engines. These tightly grouped sets of values act very much like a categorical variable.

Note: The number of scatter plots and the memory required to compute and display them can be a bit daunting. You may wish to make a scatter plot matrix with fewer columns. For example, you can eliminate columns which are collinear with other columns such as highway.mpg, by using only city mpg.

Create Histograms

Histograms are a basic, yet powerful, tool for examining the distribution properties of a dataset.

Note: You have already worked with histograms in a previous lab, so here you will focus on conditioned histograms of certain columns. A conditioned histogram is a histogram of a subset of data conditioned on another variable in the dataset. Often the histogram of a numeric variable is conditioned on a categorical variable. It is also possible to condition a histogram on (generally overlapping) ranges of a numeric variable.

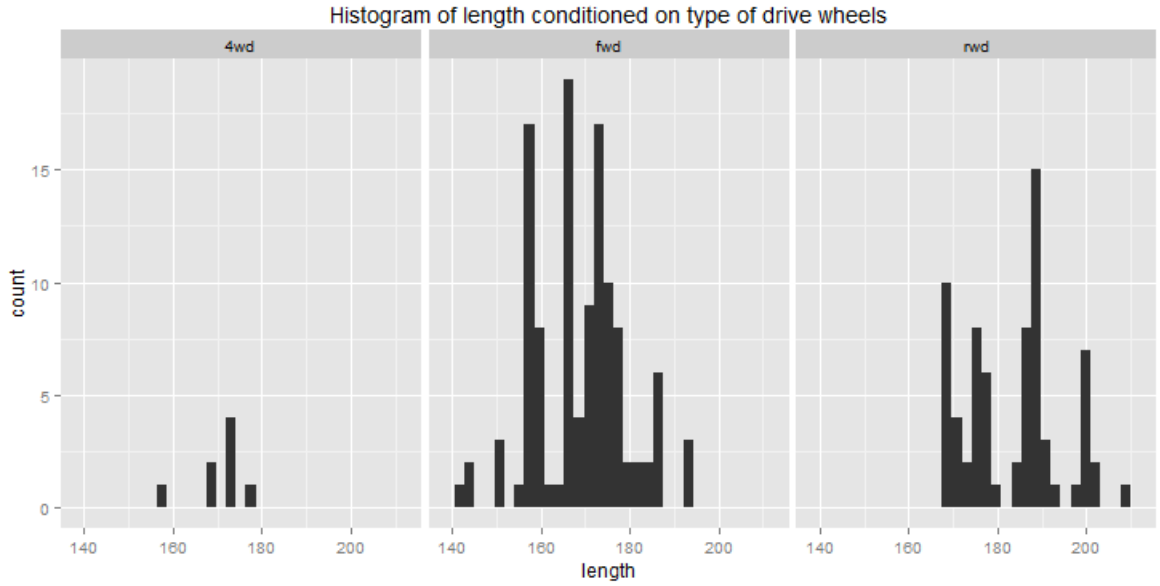
1. In code editor pane, under the comments **## Define columns for making a conditioned histogram** and **## Function to plot conditioned histograms**, examine the following code:

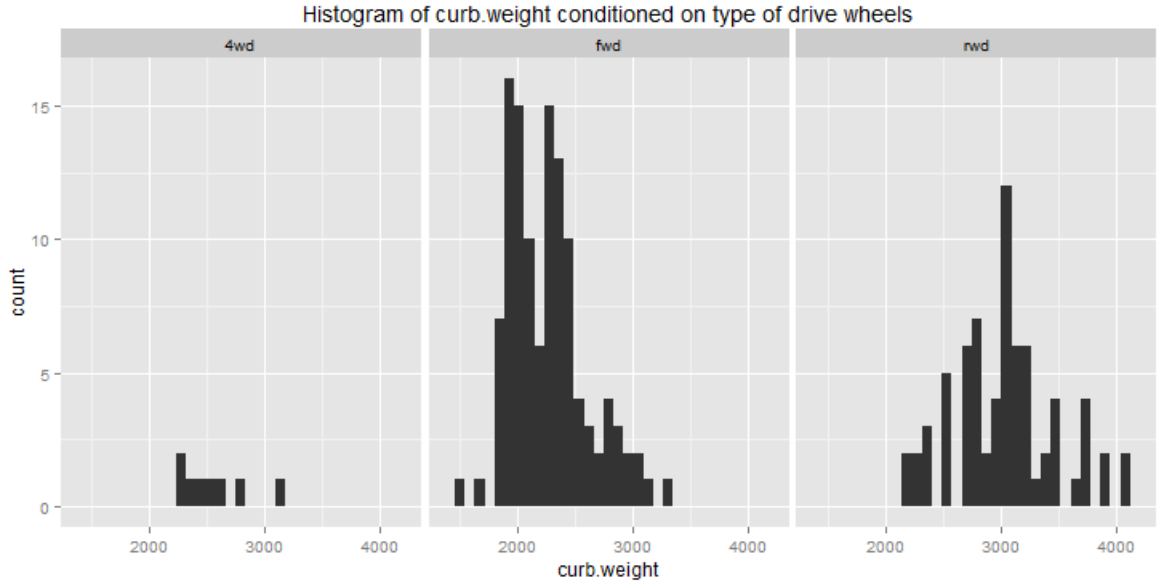
```
## Define columns for making a conditioned histogram
plot.cols2 <- c("length",
                "curb.weight",
                "engine.size",
                "city.mpg",
                "price")

## Function to plot conditioned histograms
auto.hist <- function(x) {
  library(ggplot2)
  library(gridExtra)
  ## Compute the bin width
  rg = range(auto.price[,x])
  bw = (rg[2] - rg[1])/30
  ## Define the title
  title <- paste("Histogram of", x, "conditioned on type of drive
wheels")
  ## Create the histogram
  ggplot(auto.price, aes_string(x)) +
    geom_histogram(aes(y = ..count..), binwidth = bw) +
    facet_grid(. ~ drive.wheels) +
    ggtitle(title)
}
```

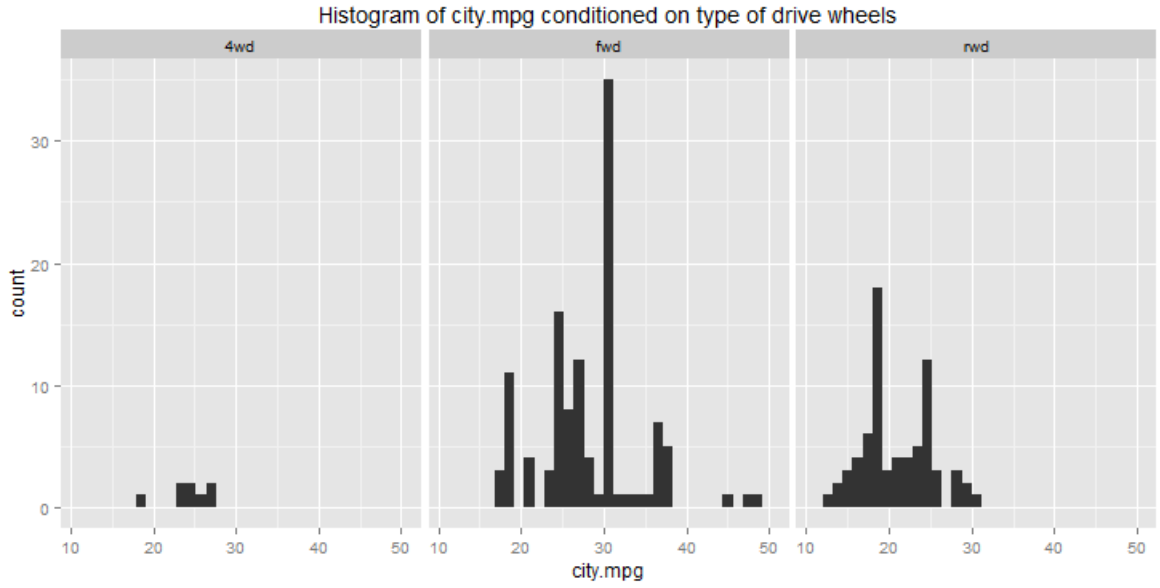
   The first part of this code defines a limited set of numeric columns, for which you will create conditioned histograms. The function **auto.hist** computes histograms, conditioned on the **drive.wheels** column, given the name of a numeric column from the **auto.price** dataset. Read this code, and note the comments, to understand the details of operation.

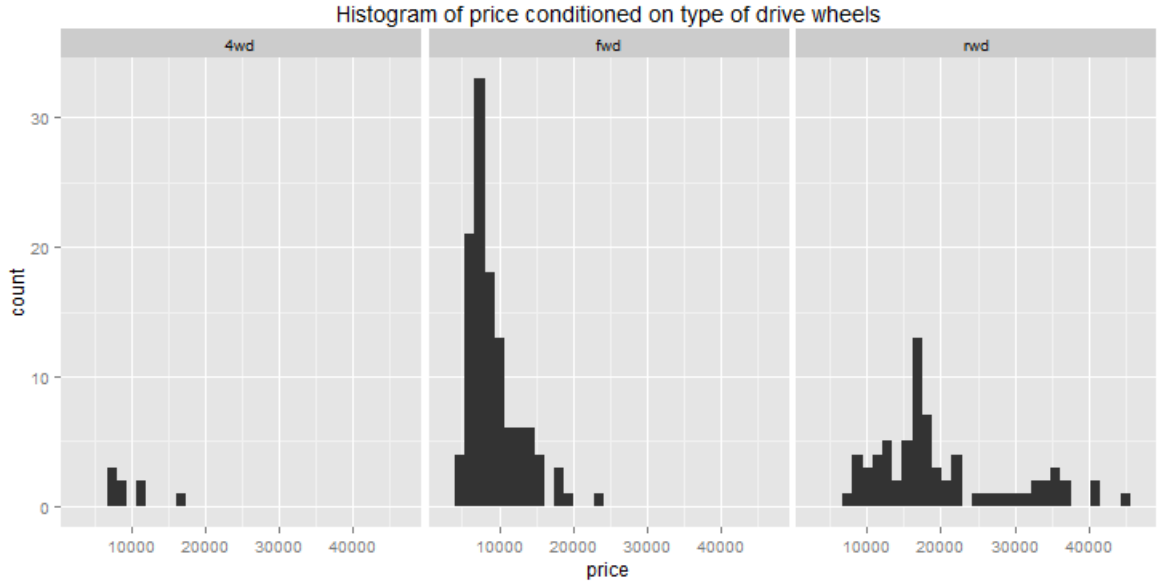
2. In the Console pane, enter the following code to view conditioned histograms for the columns you defined conditioned on the **drive-wheels** column (some of which are shown below – you will need to click the **Previous Plot** and **Next Plot** buttons in the **Plots** pane to view them all):

```
lapply(plot.cols2, auto.hist)
```


Histogram of length conditioned on type of drive wheels
4wd
fwd
rwd
count
0
5
10
15
140
160
180
200
length


Histogram of curb.weight conditioned on type of drive wheels
4wd
fwd
rwd
count
0
5
10
15
2000
3000
4000
curb.weight





3. In the **Plots** pane, use the **Previous Plot** and **Next Plot** toolbar buttons to view each of the conditioned histograms. Examine this series of conditioned plots, and note the following:
   - There is a consistent difference in the distributions of the numeric features conditioned on the categories of **drive.wheels**.
   - The distribution of the values generally increases for **length** and **curb.weight** for real wheel drive (rwd) cars, with the values for 4 wheel drive (4wd) and real wheel drive (rwd) overlapping.
   - Cars with fwd have the highest **city.mpg**, with 4wd and rwd in a similar range.
   - Generally, 4wd cars have the lowest **price**, with rwd cars having the widest range.

   Note: In a previous lab you worked with histograms conditioned on number of cylinders. In this exercise you have created histograms conditioned on the type of drive wheels. In each case, the conditioning has highlighted different aspects of the relations in these data. Exploring the

distributions in the dataset conditioned on other features will likely highlight other aspects of the structure of these data. You may wish to try this on your own.

Create Box Plots

In this exercise you will create conditioned box plots. Box plots are another tool for comparing distributions of conditioned numeric variables. Box plots allow comparison of summary statistics, median and quartiles, as well as to visualize outliers.

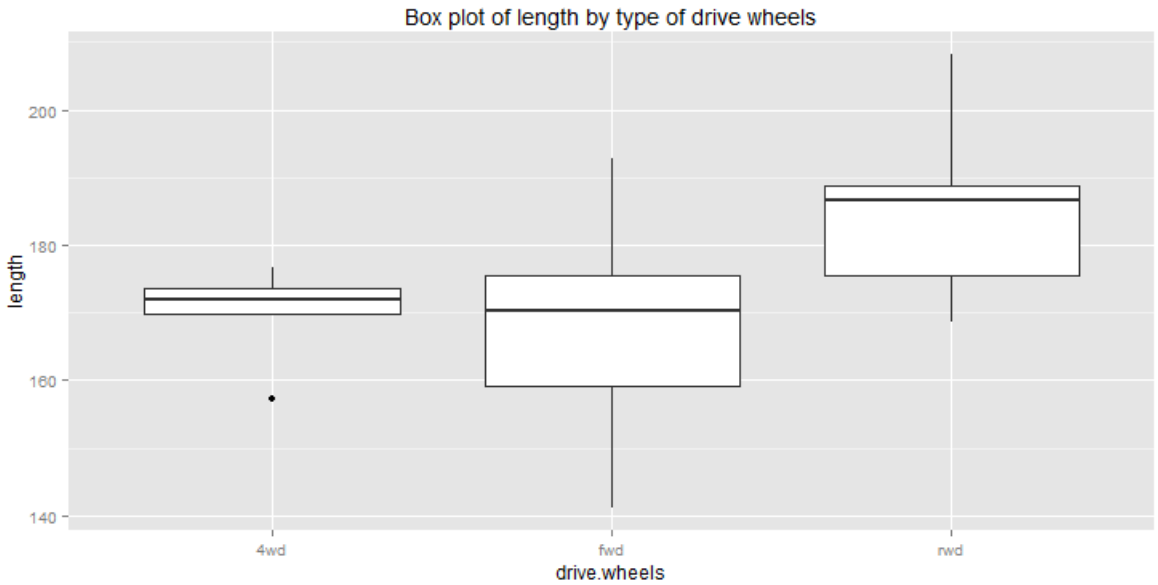
1. In the code editor pane, under the comments **## Function to create conditioned box plots**, examine the following code:

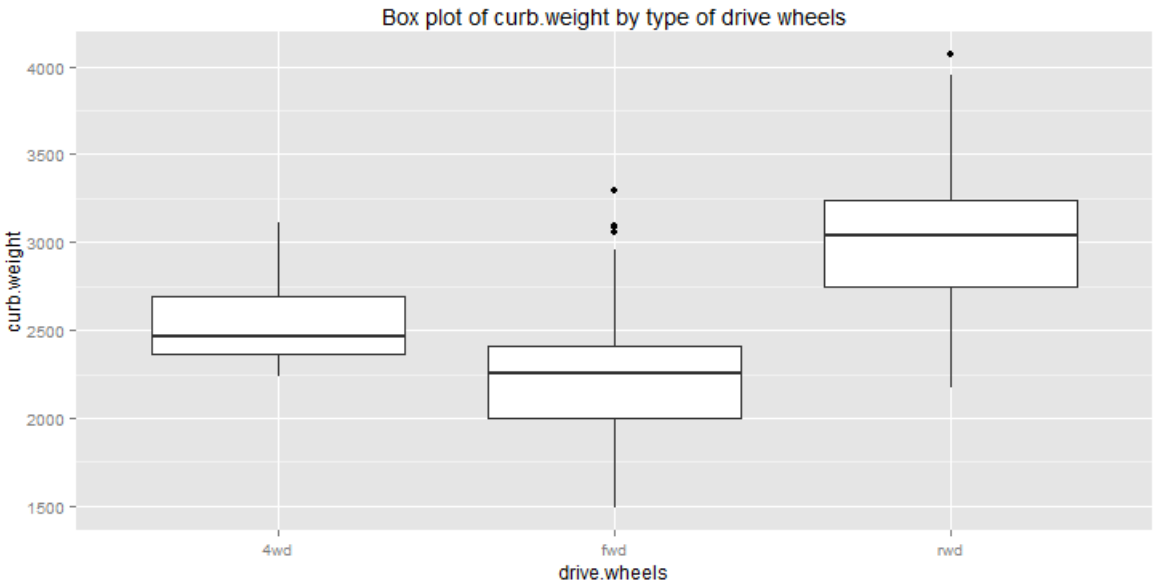
```
## Function to create conditioned box plots
auto.box <- function(x) {
  title <- paste("Box plot of", x, "by type of drive wheels")
  ggplot(auto.price, aes_string('drive.wheels', x)) +
    geom_boxplot() +
    ggtitle(title)
}
```

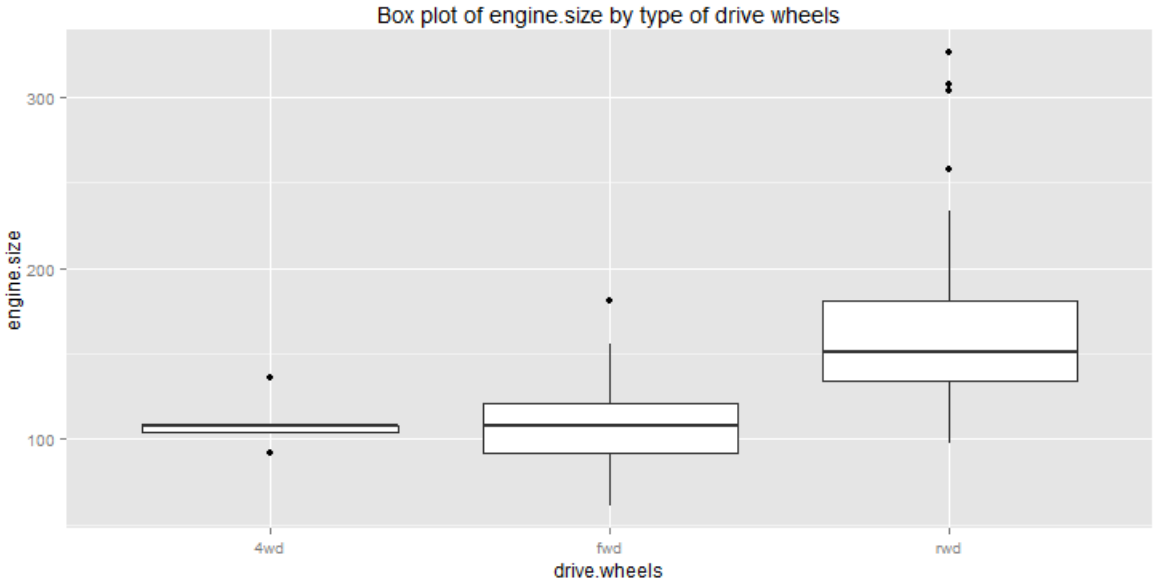
   The **auto.box** function computes box plots, conditioned on the **drive.wheels** column, given the name of a numeric column from the **auto.price** dataset.

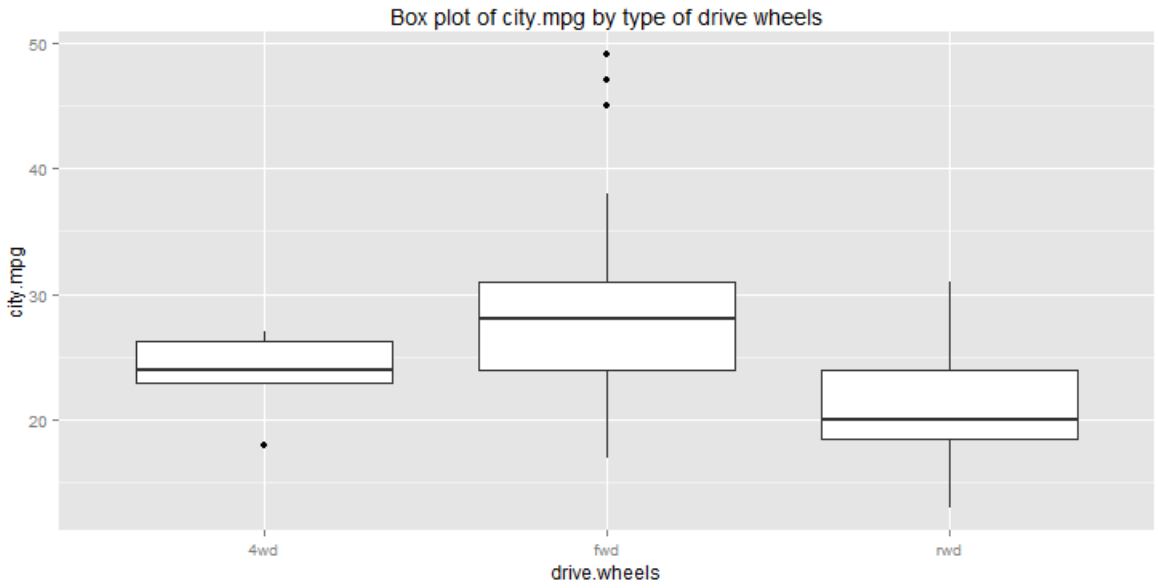
2. In the Console pane, enter the following code and view the box plots for the columns you defined conditioned on the **drive-wheels** column (shown below):

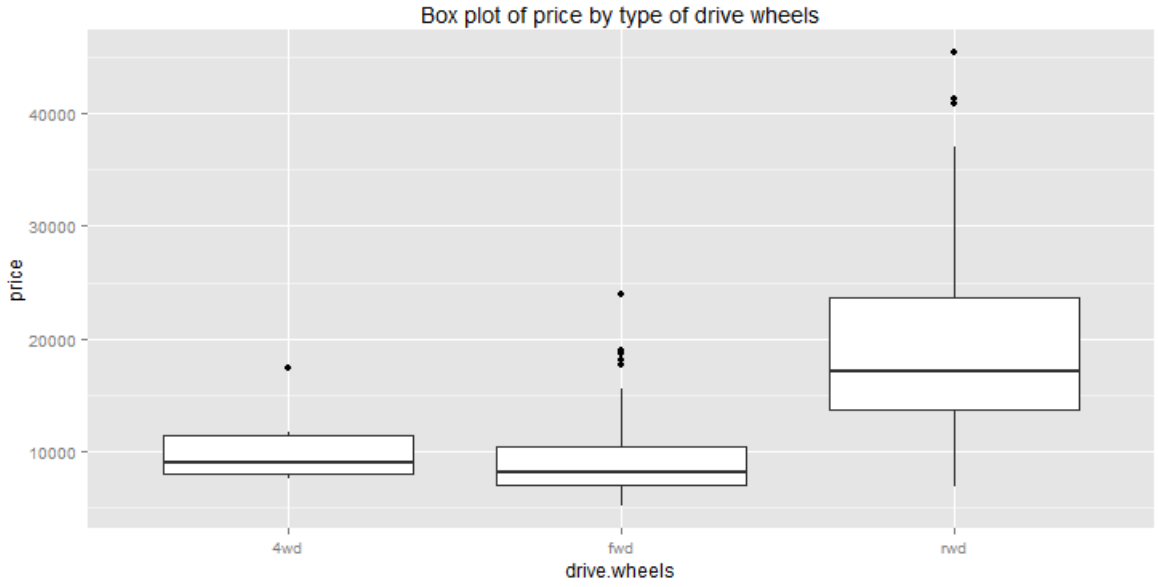
```
lapply(plot.cols2, auto.box)
```




Box plot of curb.weight by type of drive wheels
curb.weight
4000
3500
3000
2500
2000
1500
4wd
fwd
rwd
drive.wheels


Box plot of engine.size by type of drive wheels
engine.size
300
200
100
4wd
fwd
rwd
drive.wheels





3. Review each of the box plots. The conclusions you can draw from these plots are much the same as from the conditioned histograms. One detail now stands out; the **curb.weight** of 4wd cars is mostly between that of fwd and rwd cars. As is often the case, a different view or visualization for the same data results in finding new relationships.

Create Scatter Plots

Scatter plots are widely used to examine the relationship between two variables. In this procedure, you will explore methods to show the relationship between more than two variables on a two-dimensional scatter plot. First you will apply color as a method to examine multiple dimensions. By adding color to a scatter plot, you can effectively project three dimensions onto a two-dimensional plot. Next you will

create and examine conditioned scatter plots. By conditioning multiple dimensions, you can project several additional dimensions onto the two-dimensional plot.

You will not use point shape as a differentiator in this exercise, but keep in mind that shape can be as useful as color. Additionally, shape may be easier for the significant fraction of the population who are color blind.

Note: Be careful when combining methods for projecting multiple dimensions. You can easily end up with a plot that is not only hard to interpret, but even harder for you to communicate your observations to your colleagues. For example, if you use three conditioning variables, plus color and shape, you are projecting seven dimensions of your dataset. While this approach might reveal important relationships, it may just create a complex plot.

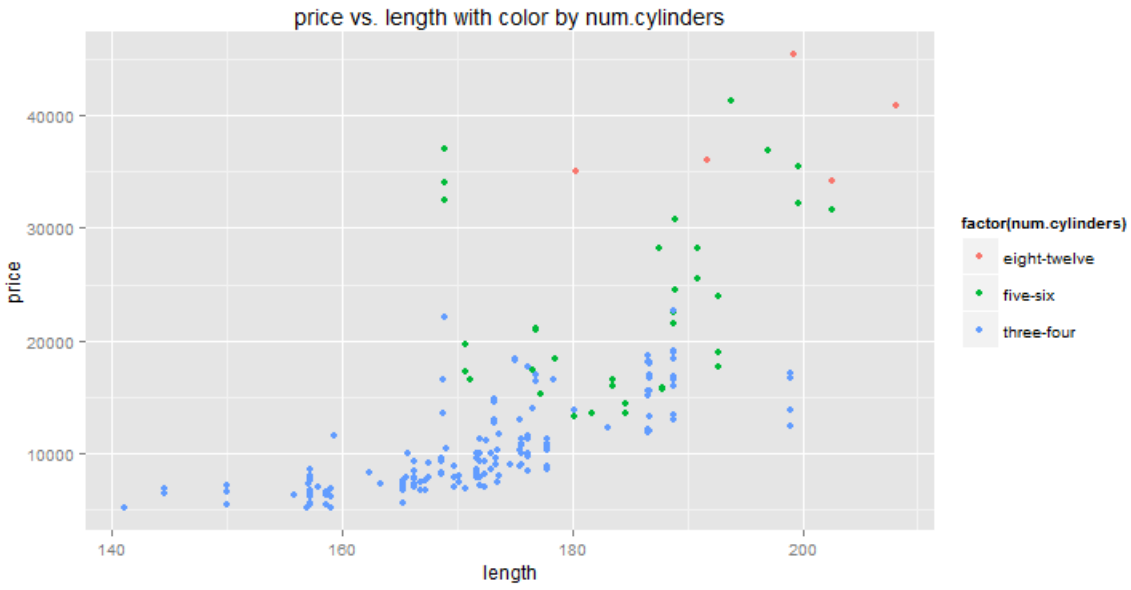
1. In the code editor pane, under the comments **## Define columns for making scatterplots** and **## Scatter plot using color to differentiate points**, examine the following code:
   code.

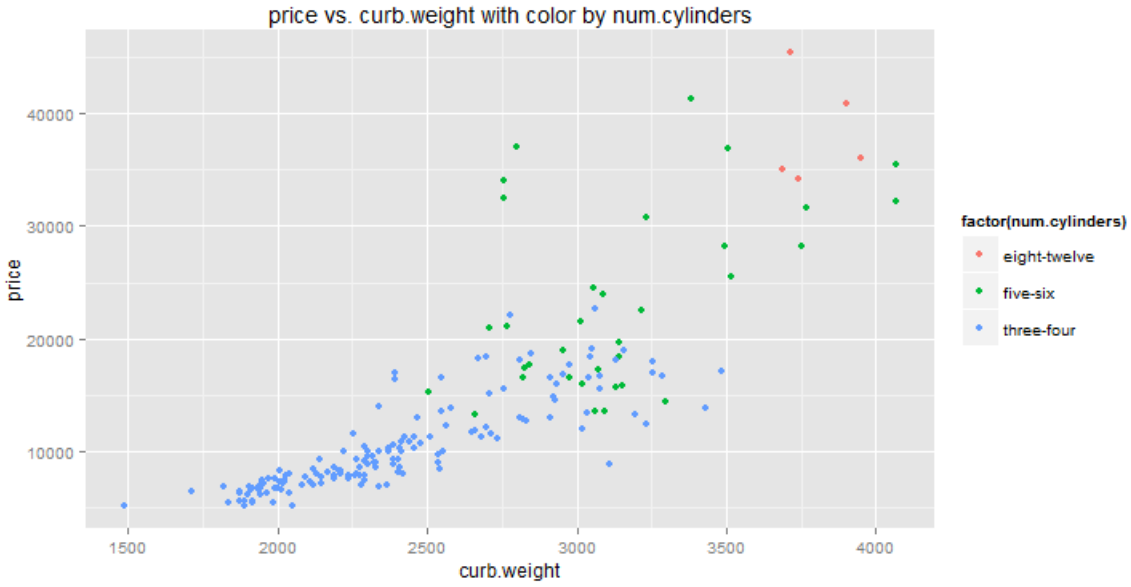
```
## Define columns for making scatter plots
plot.cols3 <- c("length",
                "curb.weight",
                "engine.size",
                "city.mpg")

## Scatter plot using color to differentiate points
scatter.auto <- function(x){
  require(ggplot2)
  title <- paste("price vs.", x, 'with color by num.cylinders')
  ggplot(auto.price, aes_string(x, 'price')) +
    geom_point(aes(color = factor(num.cylinders))) +
    ggtitle(title)
}
```

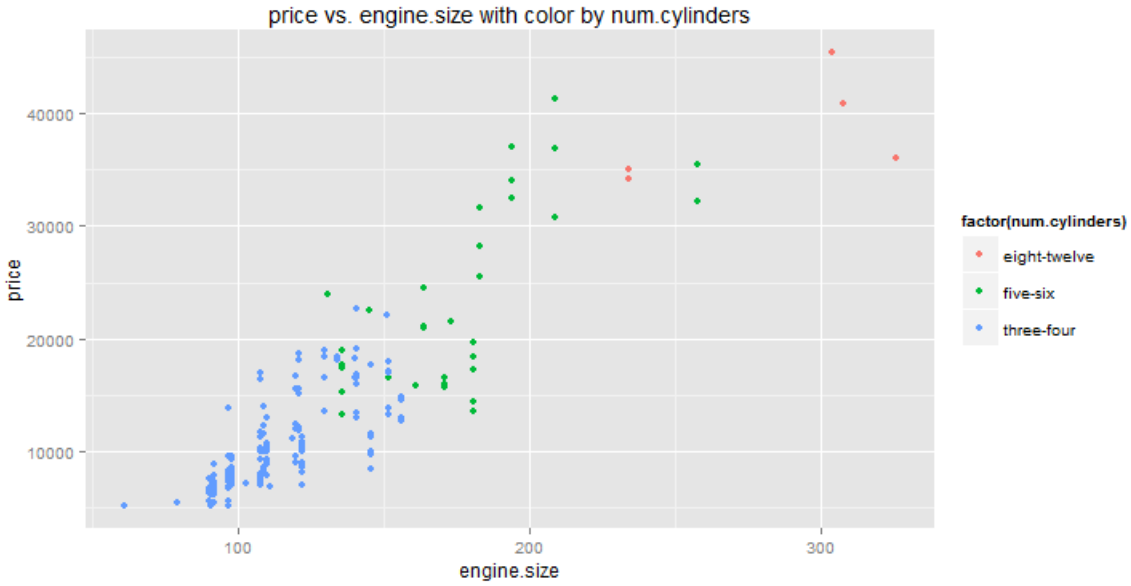
   The first part of this code defines a limited set of numeric columns, for which you will createscatter plots vs. **price**. The function **scatter.auto** computes a scatter plot with the color of the point determined by the **num.cylinders** column.

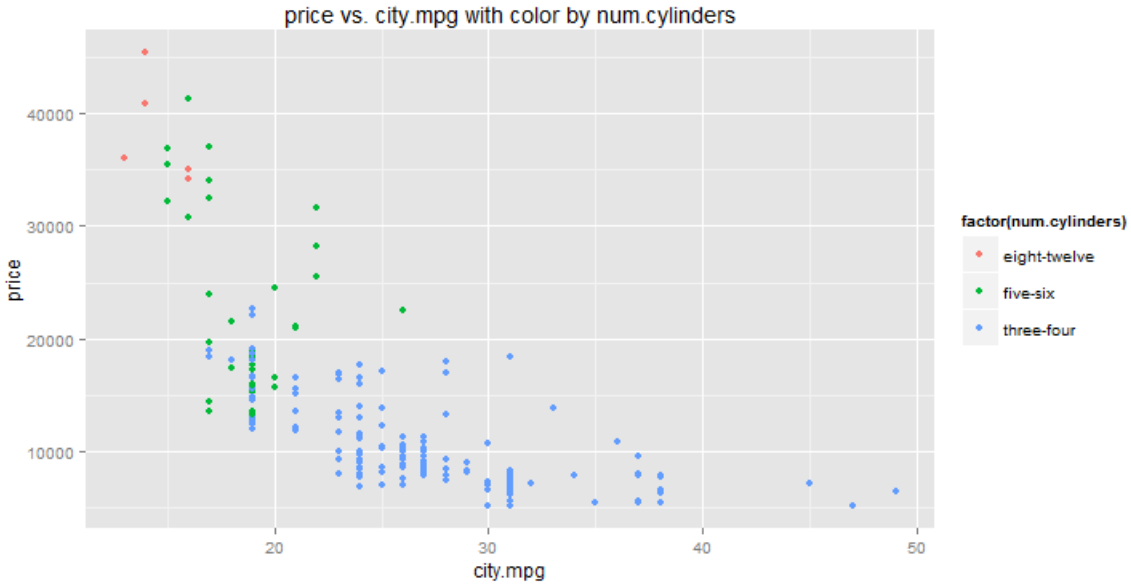
2. In the Console pane, enter the following code and view the scatter plots for the four columns vs. price that are generated (shown below):

```
lapply(plot.cols3, scatter.auto)
```


price vs. length with color by num.cylinders
price
length
40000
30000
20000
10000
140
160
180
200
factor(num.cylinders)
eight-twelve
five-six
three-four


price vs. curb.weight with color by num.cylinders
price
curb.weight
40000
30000
20000
10000
1500
2000
2500
3000
3500
4000
factor(num.cylinders)
eight-twelve
five-six
three-four





Examine these plots, and note the following:

- Each of the four variables plotted against price shows a strong trend, indicating they might be good predictors of price.
- In each plot, the trend and positioning of the points for by numbers of cylinders are different. Cars with different numbers of cylinders seem to form distinct but overlapping clusters.

3. Review the code under the comment **## Conditioned scatter plots**, as shown here:

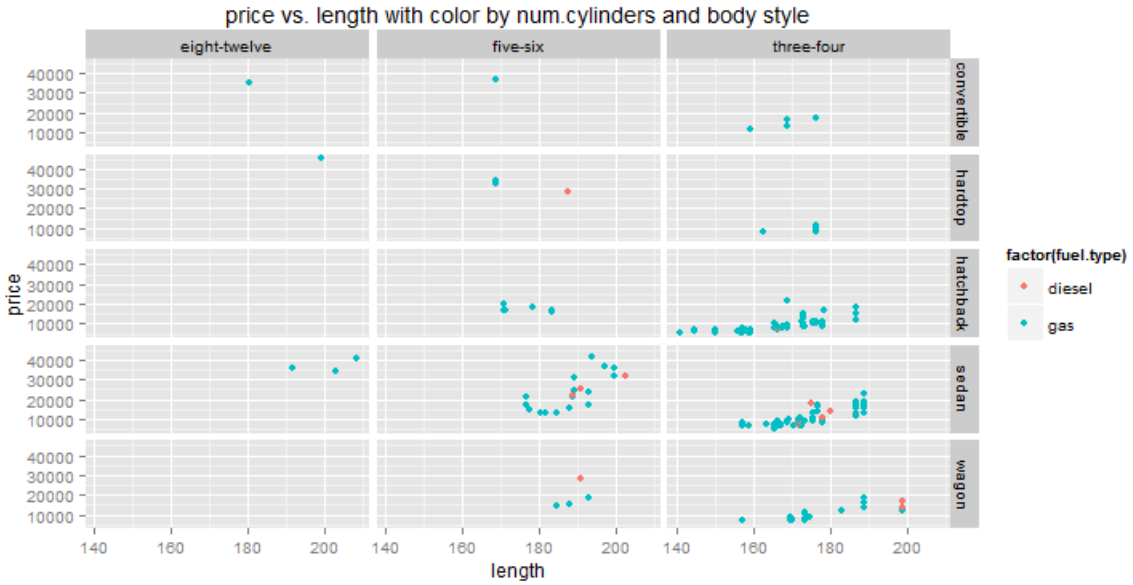
```
## Conditioned scatter plots
scatter.auto.cond <- function(x){
  require(ggplot2)
  library(gridExtra)
  title <- paste("price vs.", x, 'with color by num.cylinders and body
style')
```

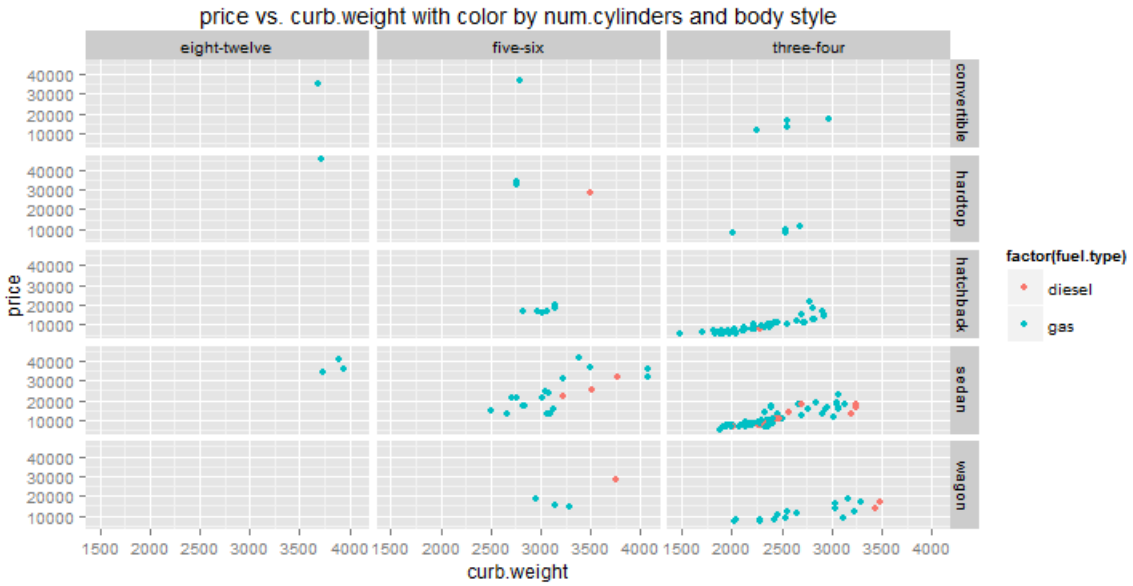
```
  ggplot(auto.price, aes_string(x, 'price')) +
    geom_point(aes(color = factor(fuel.type))) +
    facet_grid(body.style ~ num.cylinders) +
    ggtitle(title)
}
```

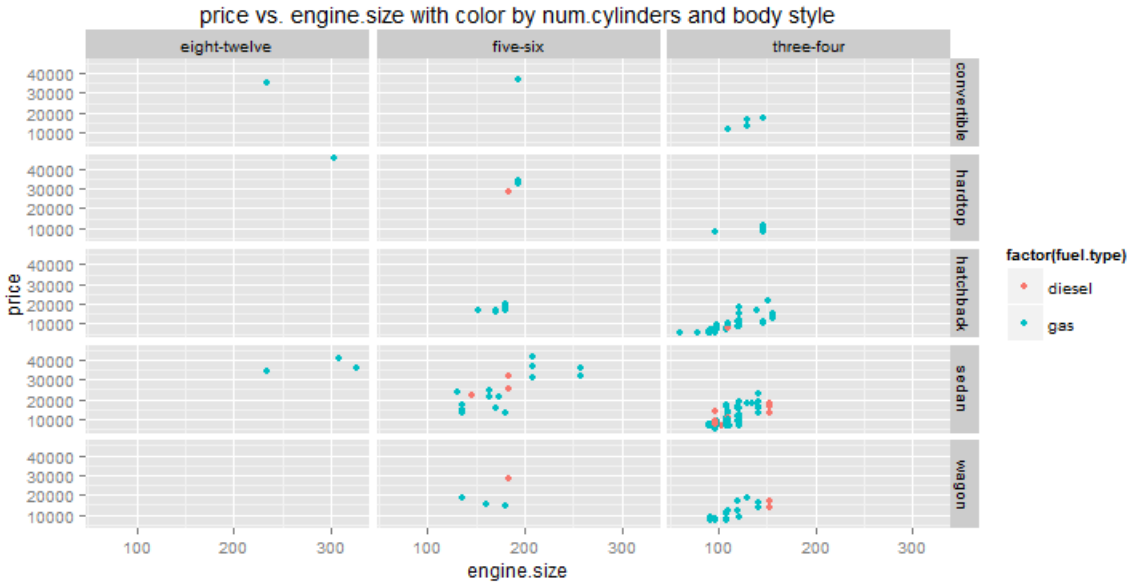
This code is similar to the pervious function. The **facet_grid** function has been added to condition the plots on **body.style** and **num.cylinders**. The color is now determined by the fuel type.

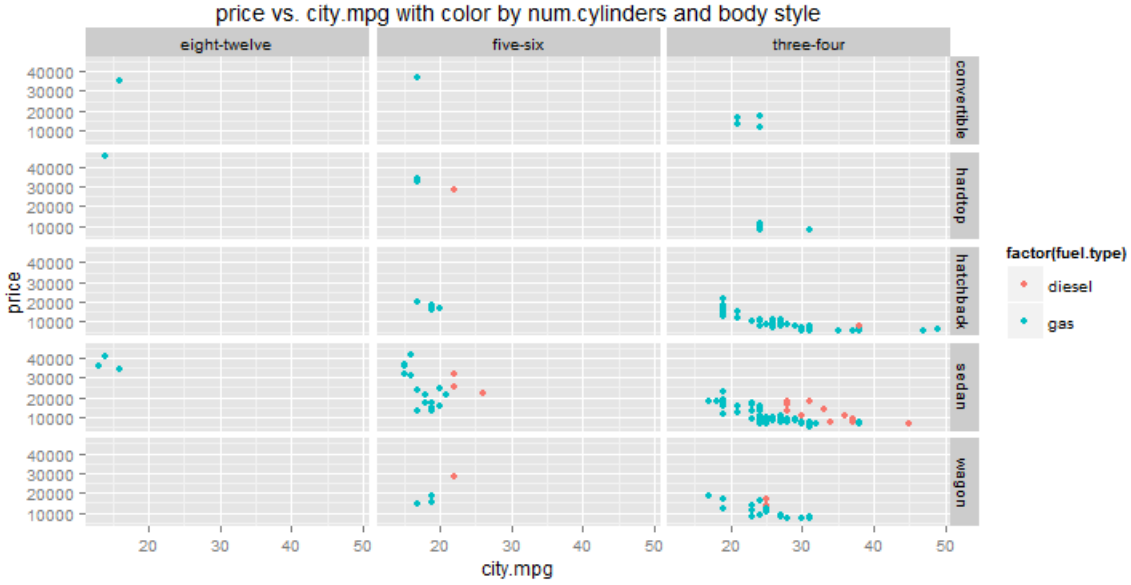
4. In the Console pane, enter the following code view the conditioned scatter plots for the four columns vs. price generated, as shown below:

```
scatter.auto.cond(plot.cols3)
```









Carefully examine the plots you have created. There is a lot of detail to understand. Generally, you should be able to conclude the following:

- There are no diesel cars for some of the conditioning combinations, such as convertables and cars with eight or more cylinders.
- There are no cars at all for some combinations of conditioning variables.
- Fuel type does not seem to be a good predictor of price in most cases, since the points for diesel cars are mostly on the same trend as gasoline cars.
- Diesel engines are generally used in heavier cars, which then tend to be more fuel efficient.
- As observed previously, the number of cylinders has a significant impact on **price**.
- There is considerable overlap in **price** conditioned by body style, indicating body style will be a weak predictor.

Note: By now, you should realize that exploring a dataset in detail is an open-ended and complex task. You may wish to try other combinations of conditioning variables, and color variable, to find some other interesting relationships in these data.

Load Adult Census Data

Now that you've explored the automobile data, it's time to turn your attention to the adult census data, which you plan to use for a classification solution in which you classify people has low-income if they earn $50,000 or less, or high-income if they earn more than $50,000.

1. In RStudio, open the **ExploreIncome.R** file from the folder containing the lab files for this module.
2. In the code editor pane, review the code shown below, which loads the adult census data file:

```
read.income <- function(path = 'c:/dat203.1x/mod4/'){
  ## Read the csv file
  filePath <- file.path(path, 'Adult Census Income Binary
Classification dataset.csv')
  read.csv(filePath, header = TRUE, stringsAsFactors = FALSE)
}
```

3. Click the **Source** button to reference the script file in the console.
4. In the Console pane, enter the following code, making sure to set the path to the folder containing your lab files:

```
Income <- read.income('SET-YOUR-PATH-HERE')
str(Income)
```

5. Note the output which is a summary of the **Income** data frame (shown below). There are both numeric columns and character columns. In addition, you can see the first few values of each column.

```
'data.frame':     32561 obs. of  15 variables:
 $ age           : int  39 50 38 53 28 37 49 52 31 42 ...
 $ workclass     : chr  " State-gov" " Self-emp-not-inc" " Private" ...
 $ fnlwgt        : int  77516 83311 215646 234721 338409 284582 ...
 $ education     : chr  " Bachelors" " Bachelors" " HS-grad" ...
 $ education.num : int  13 13 9 7 13 14 5 9 14 13 ...
 $ marital.status: chr  " Never-married" " Married-civ-spouse" ...
 $ occupation    : chr  " Adm-clerical" " Exec-managerial" ...
 $ relationship  : chr  " Not-in-family" " Husband" " Not-in-family"...
 $ race          : chr  " White" " White" " White" " Black" ...
 $ sex           : chr  " Male" " Male" " Male" " Male" ...
 $ capital.gain  : int  2174 0 0 0 0 0 0 0 14084 5178 ...
 $ capital.loss  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ hours.per.week: int  40 13 40 40 40 40 16 45 50 40 ...
 $ native.country: chr  " United-States" " United-States" ...
 $ income        : chr  " <=50K" " <=50K" " <=50K" " <=50K" ...
```

Note: Character data is typically treated as categorical where each unique string value is a categorical value of the column. When exploring data sets, it is important to understand the relationships between each category and the label values.

Bar plot the categorical features

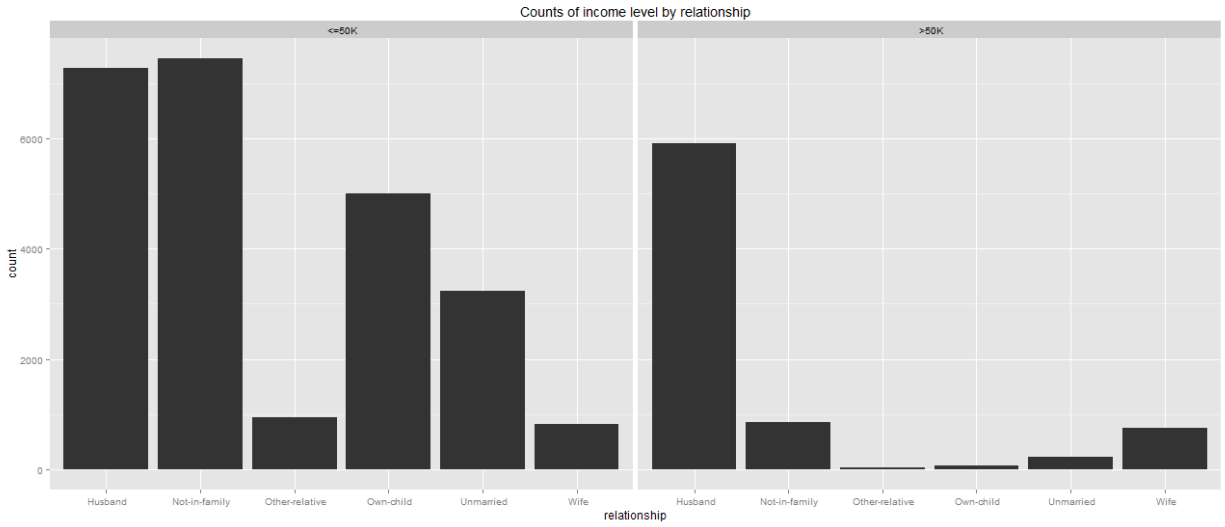
1. In the code editor pane, under the comments **## Features to plot and ## Bar plot of categorical features**, examine the following code:

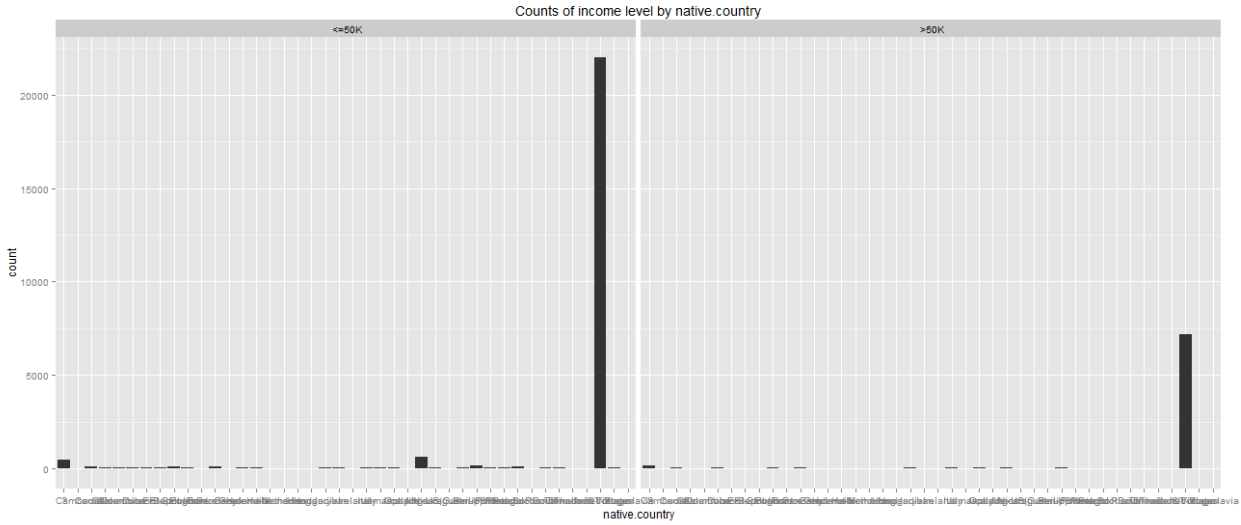
```
## Features to plot
name.list <- function(x) {
  names <- names(x)
  len <- length(names)
  names[-len]
}

## Bar plot of categorical features
bar.income <- function(x){
  library(ggplot2)
  if(!is.numeric(Income[,x])) {
    capture.output(
      plot( ggplot(Income, aes_string(x)) +
              geom_bar() +
              facet_grid(. ~ income) +
              ggtitle(paste("Counts of income level by",x))))
```

The first function defines a list of the features in the dataset, less the last column which is the label. The **bar.income** function creates a bar plot for any input column which is not numeric **(!is.numeric)** conditioned on the values of the label.

2. In the Console pane, enter the following command and view the series of bar plots generated (some examples are shown below):

```
feature.names <- name.list(Income)
lapply(feature.names, bar.income)
```





Examine the two plots shown above.

The first plot shows a feature, **marital.status**, which may have some power in predicting the label category. Note that the proportions of cases are quite different for people with income **<=50K** and people with income **>50K**. *Husband* dominates **>50K**, while *Not-in-family* is the largest fraction for **<=50K**. This feature is therefore said to separate the two label categories. However, the separation is not perfect as there is overlap in the cases. This is typical and it is rare to find a single feature which provides perfect separation.

The second plot shows a feature, **native.country**, which may not be that useful in predicting the label category. First, there are quite a few categories. Second, the only category with significant numbers of people is *United States*. There is only a small fraction of the people spread over the other 41 countries. There is no particular pattern noticeable between people with income **<=50K** and people with income **>50K**.

You may wish to step through the plots for other features. Consider if these features help to separate the label categories.

Note: When looking at the bar plots of the features conditioned on label value, keep in mind that there are about three times as many people with low income as high income. Therefore, it is the proportions between the categories of the label which are important, not the absolute values. In other words, are the particular values of a feature relatively more or less likely for the label categories?

Box plot the numeric features

There are two possible categories or label values for a person's income category, in this case, **<=50K** or **>50K**. In this exercise you will create box plots for each numeric feature, conditioned on the label value. By examining these box plots, you will determine which features are likely to be useful as predictors of the label categories.

1. In the code editor pane, under the comments **## Create Box plot of numeric features**, examine the following code:

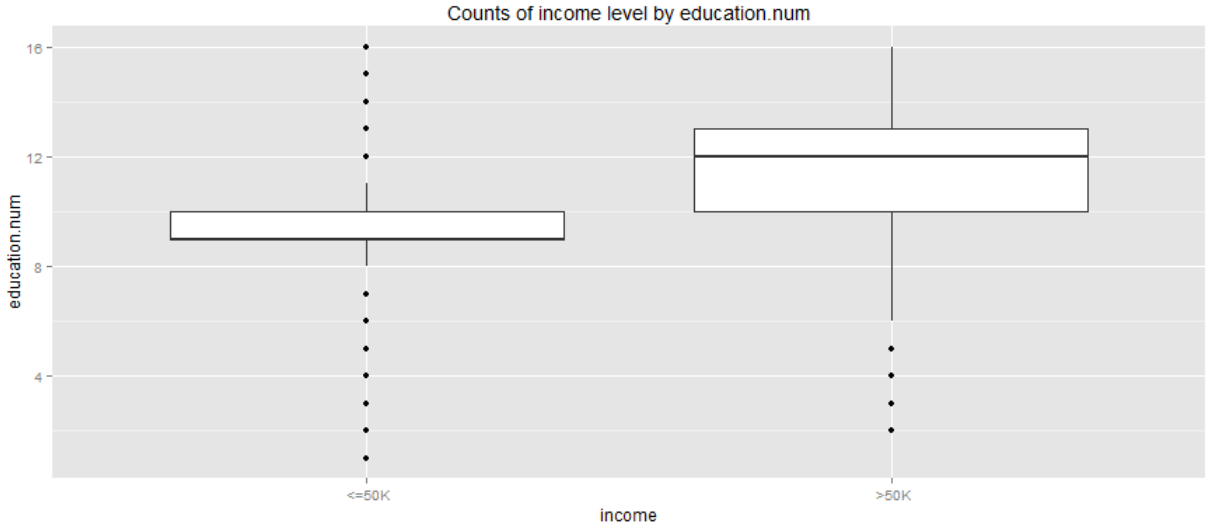
```
## Create Box plot of numeric features
box.income <- function(x){
```

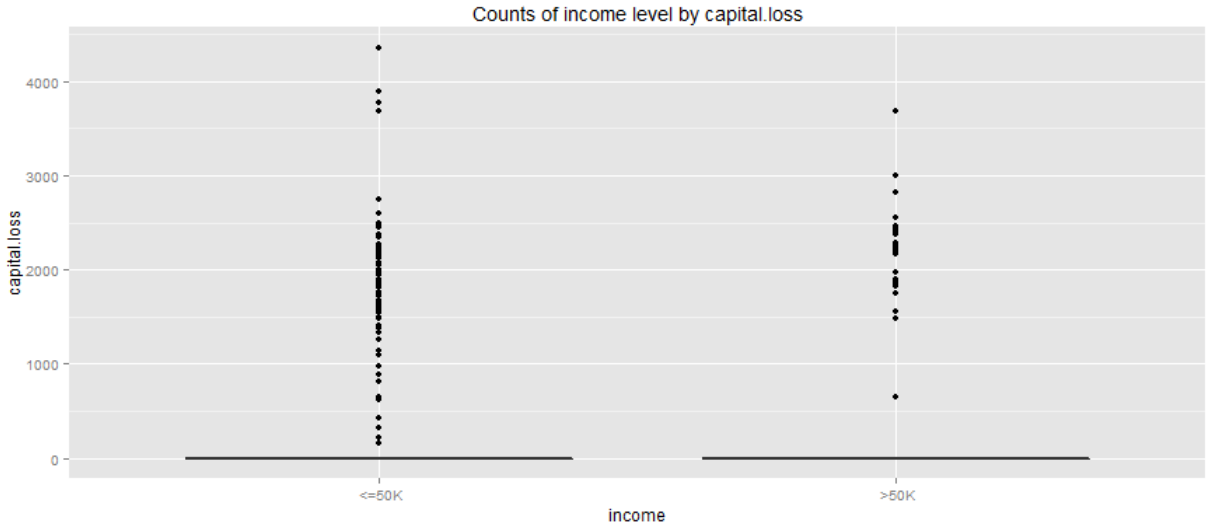
```
  library(ggplot2)
  if(is.numeric(Income[,x])) {
    capture.output(
      plot( ggplot(Income, aes_string('income', x)) +
              geom_boxplot() +
              ggtitle(paste("Counts of income level by",x))))
 }}
```

This function creates a box plot for any input column which is numeric. The **capture.output** function is used to capture and suppress voluminous and annoying output from ggplot.

2. In the Console pane, enter the following code and view the series of box plots generated (a few examples are shown below):

```
lapply(feature.names, box.income)
```





The first plot shows a feature, **Income**, which may have some power in predicting the label category. Note that people with income **>50K** tend to have consistently higher **education.num** values than people with income **<=50K**.

The second plot shows a feature, **capital.loss**, which is not likely to be that useful in predicting the label category. Almost all people in the sample have zero (0) capital loss for both label values. Note that the quartile markers in the box plot all fall on top of each other at 0.

You may wish to step thought the plots for other features and consider if these features help to separate the label categories.

Visualizing Data with Python

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

Python supports the **matplotlib** library; which provides extensive graphical capabilities. Additionally, the Python **Pandas** library adds higher level graphics capability. These features make Python a useful language to create visualizations of your data when exploring relationships between the data features. Further, you can identify features that may be useful for predicting labels in machine learning projects.

Load the Automobiles Dataset

1. Start Spyder, and open the **LoadAutos.py** file in the folder where you extracted the lab files for this course.
2. In the code editor pane, note the code listing which should appear as shown here:

```
def read_auto(pathName = "c:\dat203.1x\mod4", fileName = "Automobile
price data _Raw_.csv"):
    import pandas as pd
    import numpy as np
    import os

    ## Read the .csv file
    pathName = pathName
    fileName = fileName
    filePath = os.path.join(pathName, fileName)
    auto_price = pd.read_csv(filePath)

    ## Convert some columns to numerica values
    cols = ['price', 'bore', 'stroke',
          'horsepower', 'peak-rpm']
    auto_price[cols] = auto_price[cols].convert_objects(convert_numeric
= True)

    ## Compute the log of the auto price
    auto_price['lnprice'] = np.log(auto_price.price)

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

3. In the IPython console, enter the following code to change the directory of your IPython session and load the automobile data, being sure to specify the full path to the folder containing the lab files for this module (for example, *c:\dat203.1x\mod4*).

```
import os
os.chdir('PATH-TO-YOUR-LAB-FILES')

import LoadAutos as au
auto_price = au.read_auto('.')
```

7. In the IPython console window type the following code to display the columns in the automobiles data:

```
auto_price.dtypes
```

8. Examine names and data types of the columns in this dataset, as shown below. There are both numeric columns and character columns (shown with a dtype of 'object').

```
symboling              int64
normalized-losses     object
make                  object
fuel-type             object
aspiration            object
num-of-doors          object
body-style            object
drive-wheels          object
engine-location       object
wheel-base           float64
length               float64
width                float64
height               float64
curb-weight            int64
engine-type           object
num-of-cylinders      object
engine-size            int64
fuel-system           object
bore                 float64
stroke               float64
compression-ratio    float64
horsepower           float64
peak-rpm             float64
city-mpg               int64
highway-mpg            int64
price                float64
lnprice              float64
num-cylinders         object
```

Note: Character data is typically treated as categorical where each unique string value is a categorical value of the column. When exploring data sets, it is important to understand the relationships between each category and the label values.

Create a Pair-Wise Scatter Plot

In this exercise, you will apply a visualization technique known as a scatter plot matrix to the **auto.price** dataset. Scatter plot matrix methods quickly produce a single overall view of the relationships in a dataset.

The scatter plot matrix 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. This is especially the case, as you must remember relationships you have already viewed to understand the overall structure of the data.

1. In Spyder, open the **ExploreAutos.py** file in the folder where you extracted the lab files for this course.
9. In the code editor pane, under the comments **## Numeric columns** select the following code and on the **Run** menu, click **Run cell** (or click the **Run current cell** button on the toolbar) to run it. This specifies the columns you will plot:

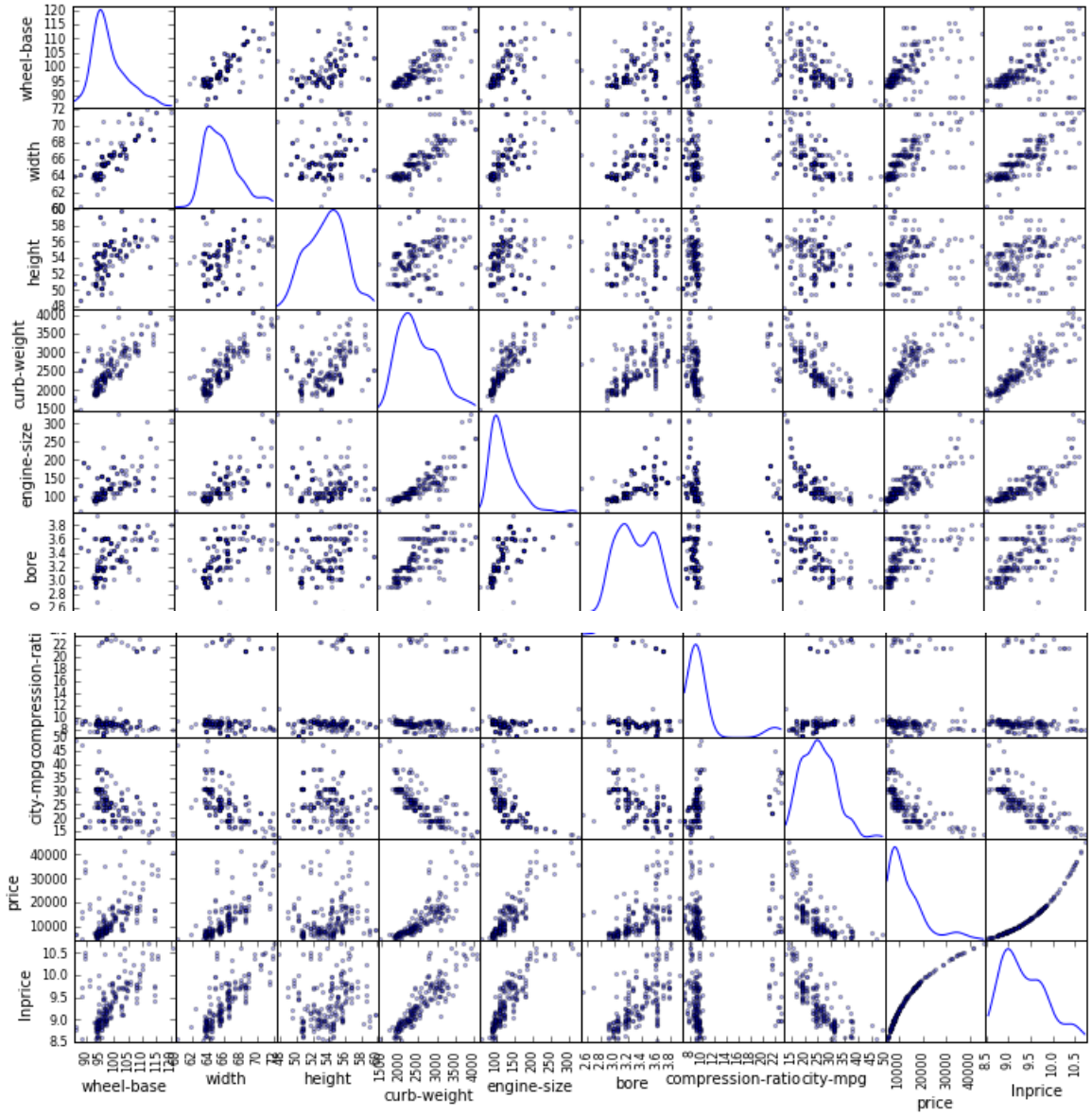
```
## Numeric columns
plot_cols = ["wheel-base",
             "width",
             "height",
             "curb-weight",
             "engine-size",
             "bore",
             "compression-ratio",
             "city-mpg",
             "price",
             "lnprice"]
```

2. In the code editor pane, review the code under the comments **## Create pair-wise scatter plots**. This code defines a function named **auto_pairs** that creates a scatter matrix plot.

```
## Create pair-wise scatter plots
def auto_pairs(plot_cols, df = auto_price):
    import matplotlib.pyplot as plt
    from pandas.tools.plotting import scatter_matrix
    fig = plt.figure(1, figsize=(12, 12))
    fig.clf()
    ax = fig.gca()
    scatter_matrix(df[plot_cols], alpha=0.3,
                   diagonal='kde', ax = ax)
    return('Done')
```

3. In the IPython console, enter the following code and view the scatter plot matrix that is generated (shown below):

```
import ExploreAutos as ea
ea.auto_pairs(plot_cols, auto_price)
```



4. 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 is displayed in both possible orientations. Examine scatter plot matrix, which shows plots of each numeric column verses every other numeric column, and note the following:
    - Not surprisingly the features, **price** and **lnprice** (log of price) are closely related.
    - Many features show significant collinearity, such as **wheel-base**, **width** and **curb-weight**, or **curb-weight** and **engine-size**, and **city-mpg**.
    - A number of features show a strong relationship with the label, **price**, such as **city-mpg**, **engine-size**, and **curb-weight**.
    - The feature, **compression-ratio**, has two distinct groupings of values, apparently corresponding to diesel (high compression ratio) and gasoline (low compression ratio) engines. These tightly grouped sets of values act very much like a categorical variable.

Note: The number of scatter plots and the memory required to compute and display them can be a bit daunting. You may wish to make a scatter plot matrix with fewer columns. For example,

you can eliminate columns which are collinear with other columns such as highway.mpg, using only city mpg.

Create Histograms

In this exercise, you will create histograms of certain numeric columns in the **auto_price** dataset. Histograms are a basic, yet powerful, tool for examining the distribution properties of a dataset.

Note: You have already worked with histograms in a previous lab, so here you will focus on conditioned histograms of certain columns. A conditioned histogram is a histogram of a subset of data conditioned on another variable in the dataset. Often the histogram of a numeric variable is conditioned on a categorical variable. It is also possible to condition a histogram on (generally overlapping) ranges of a numeric variable.

1. In the code editor pane, under the comments **## Define columns for making a conditioned histogram** select the following code and on the **Run** menu, click **Run cell** (or click the **Run current cell** button on the toolbar) to run it. This specifies the columns you will plot:

```
## Define columns for making a conditioned histogram
plot_cols2 = ["length",
              "curb-weight",
              "engine-size",
              "city-mpg",
              "price"]
```

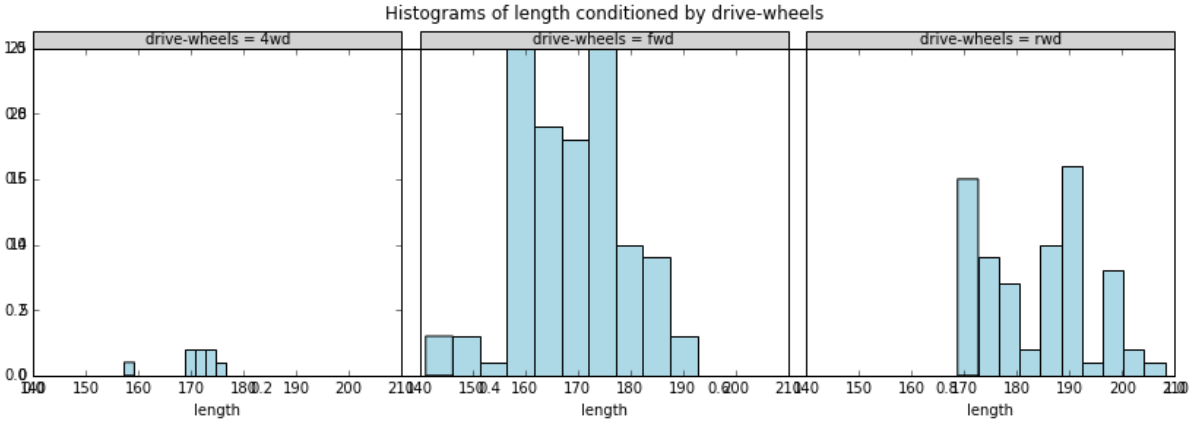
2. In the code editor pane, review the code under the comment **## Function to plot conditioned histograms**. This code defines a function named **cond_hists** that computes histograms, conditioned on a specified column and dataset:

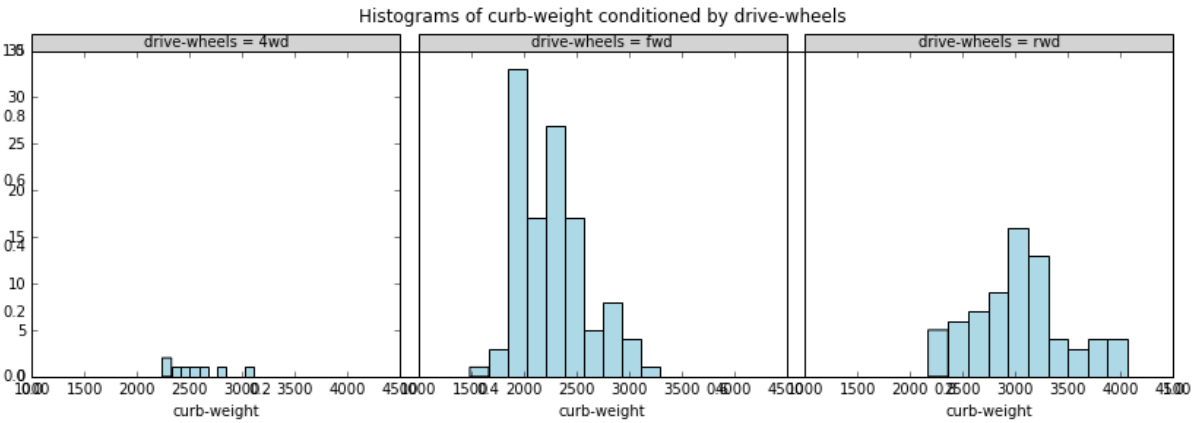
```
## Function to plot conditioned histograms
def cond_hists(df, plot_cols, grid_col):
    import matplotlib.pyplot as plt
    import pandas.tools.rplot as rplot
    ## Loop over the list of columns
    for col in plot_cols:
        ## Define figure
        fig = plt.figure(figsize=(14, 4))
        fig.clf()
        ax = fig.gca()
        ## Setup plot and grid and plot the data
        plot = rplot.RPlot(df, x = col,
                                  y = '.')
        plot.add(rplot.TrellisGrid(['.', grid_col]))
        plot.add(rplot.GeomHistogram())
        ax.set_title('Histograms of ' + col + ' conditioned by ' +
grid_col + '\n')
        plot.render()
    return grid_col
```

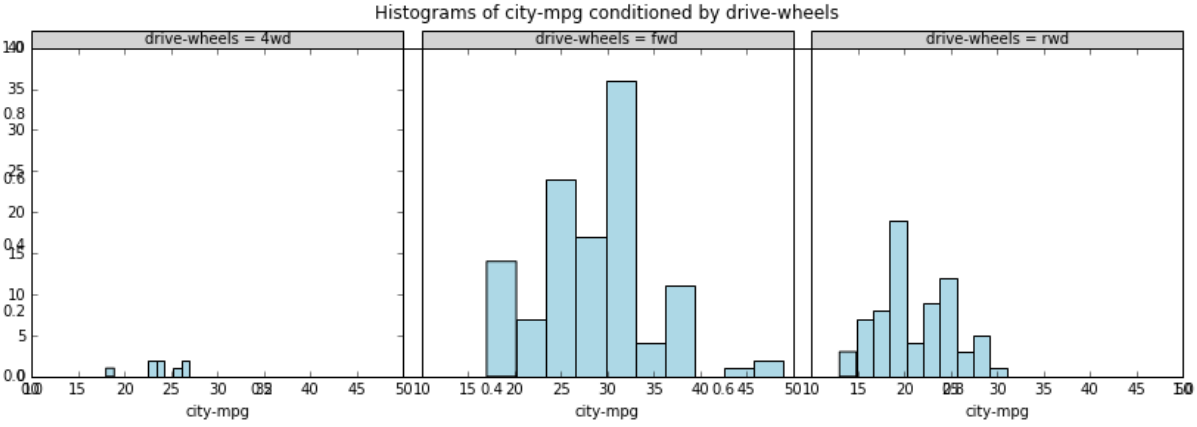
3. In the IPython console, enter the following command to view histograms for the columns you defined conditioned on the **drive-wheels** column (some of which are shown below):

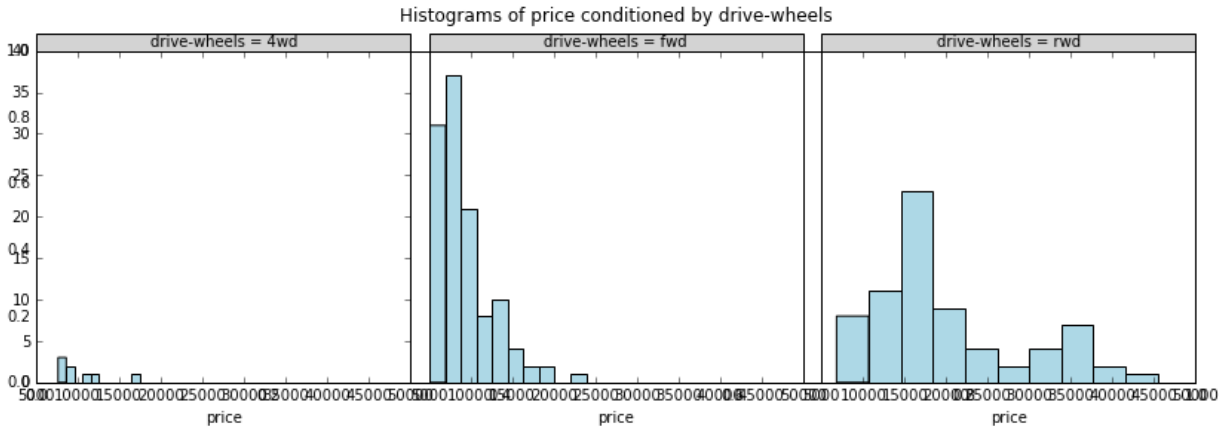
```
ea.cond_hists(auto_price, plot_cols2, 'drive-wheels')
```

Note: When you run this code you will likely see a warning that the rplot interface is deprecated. At present we are using this interface to ensure backward compatibility with Azure Machine Learning. Python users who want to do more with conditioned plotting methods should explore the newer seaborn package https://stanford.edu/~mwaskom/software/seaborn/.









4. Examine this series of conditioned plots, and note the following:
   - There is a consistent difference in the distributions of the numeric features conditioned on the categories of **drive-wheels**.
   - The distribution of the values generally increases for **length** and **curb-weight** for real wheel drive (rwd) cars, with the values for 4 wheel drive (4wd) and real wheel drive (rwd) overlapping.
   - Cars with fwd have the highest **city-mpg**, with 4wd and rwd in a similar range.
   - Generally, 4wd cars have the lowest **price**, with rwd cars having the widest range.

   Note: In a previous lab you worked with histograms conditioned on number of cylinders. In this exercise you have created histograms conditioned on the type of drive wheels. In each case, the conditioning has highlighted different aspects of the relations in these data. Exploring the distributions in the dataset conditioned on other features will likely highlight other aspects of the structure of these data. You may wish to try this on your own.

Create Box Plots

In this exercise you will create conditioned box plots. Box plots are another tool for comparing distributions of conditioned numeric variables. Box plots allow comparison of summary statistics, median and quartiles, as well as to visualize outliers.

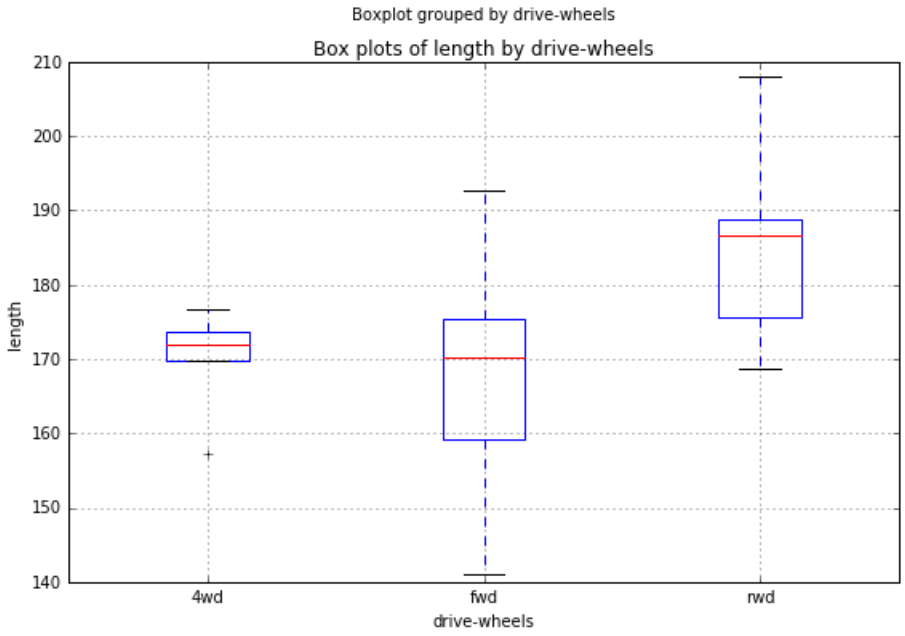
1. In the code editor pane, under the comments **## Create boxplots of data**, examine the following code:

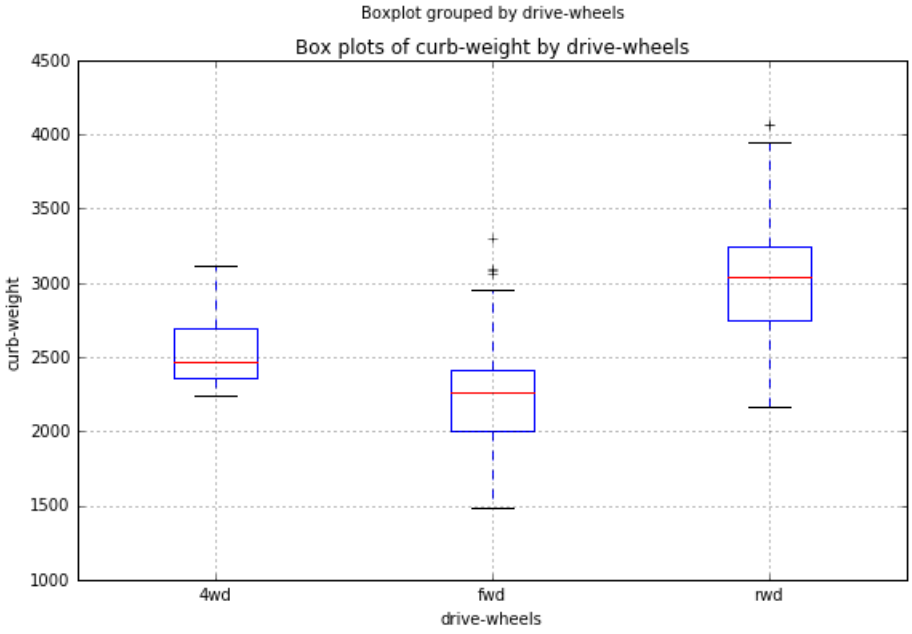
```
## Create Boxplots of data
def auto_boxplot(df, plot_cols, by):
    import matplotlib.pyplot as plt
    for col in plot_cols:
        fig = plt.figure(figsize=(9, 6))
        ax = fig.gca()
        df.boxplot(column = col, by = by, ax = ax)
        ax.set_title('Box plots of ' + col + ' by ' + by)
        ax.set_ylabel(col)
    return by
```

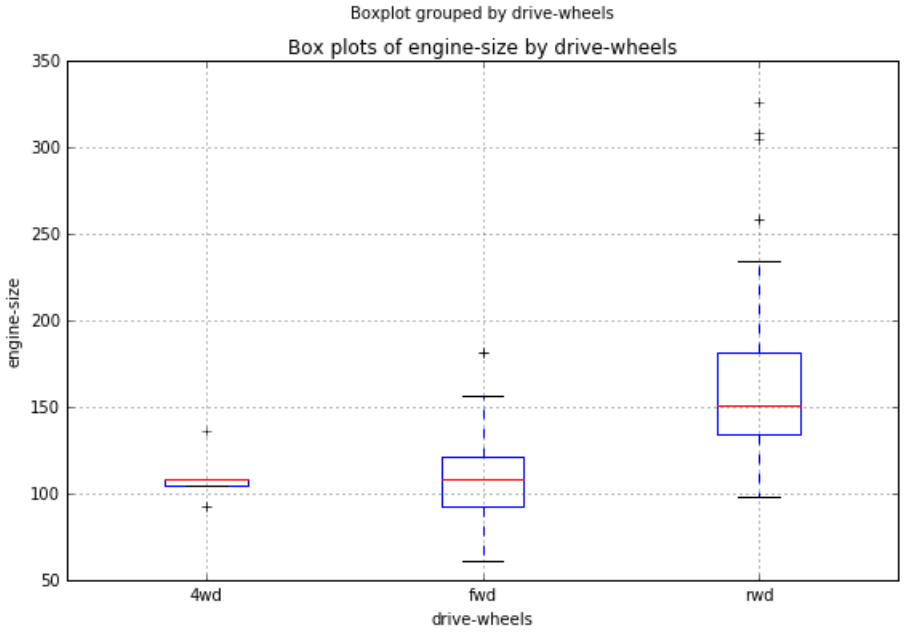
   The **auto_boxplot** function computes box plots, conditioned on the **drive-wheels** column, given the name of a numeric column from the **auto_price** dataset.

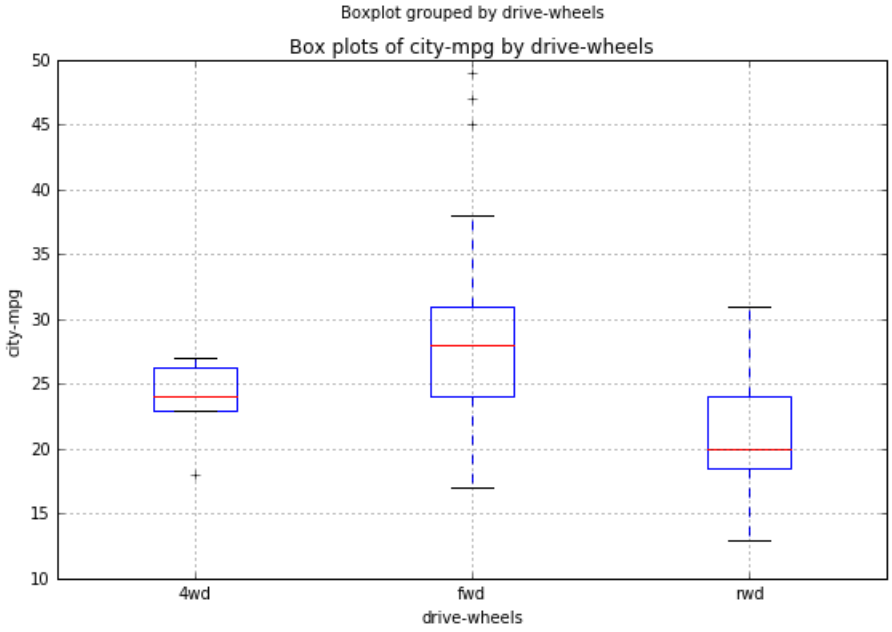
2. In the IPython console, enter the following code and view the box plots for the columns you defined conditioned on the **drive-wheels** column (shown below):

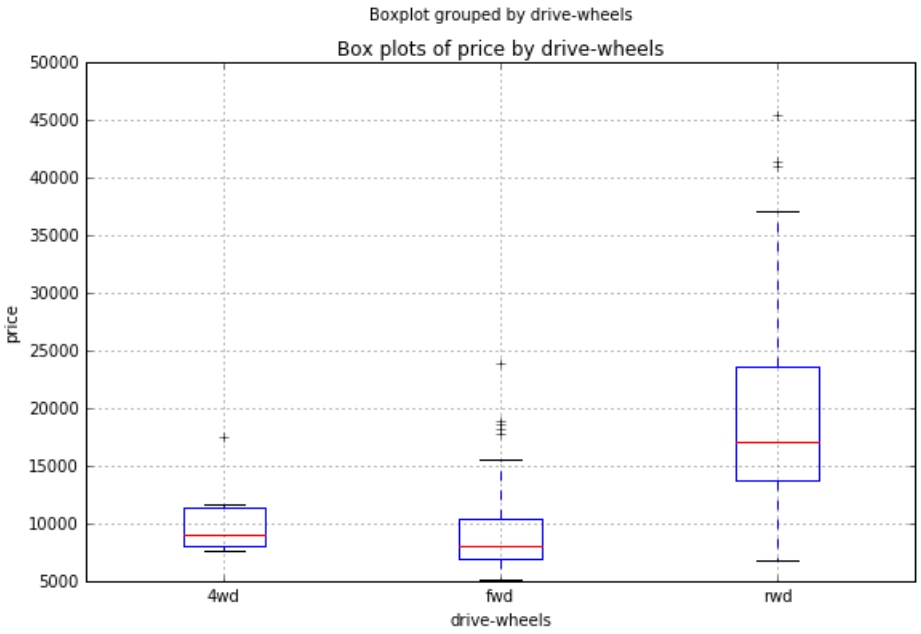
```
ea.auto_boxplot(auto_price, plot_cols2, 'drive-wheels')
```






Boxplot grouped by drive-wheels
Box plots of engine-size by drive-wheels
engine-size
350
300
250
200
150
100
50
4wd
fwd
rwd
drive-wheels


Boxplot grouped by drive-wheels
Box plots of city-mpg by drive-wheels
city-mpg
50
45
40
35
30
25
20
15
10
4wd
fwd
rwd
drive-wheels



3. Scroll though up and down in the IPython console window to view each of the box plots. The conclusions you can draw from these plots are much the same as from the conditioned histograms. One detail now stands out; the **curb-weight** of 4wd cars is mostly between that of fwd and rwd cars. As is often the case, a different view or visualization for the same data results in finding new relationships.

Create Scatter Plots

Scatter plots are widely used to examine the relationship between two variables. In this procedure, you will explore methods to show the relationship between more than two variables on a two-dimensional scatter plot. First you will apply color as a method to examine multiple dimensions. By adding color to a scatter plot, you can effectively project three dimensions onto a two-dimensional plot. Next you will create and examine conditioned scatter plots. By conditioning multiple dimensions, you can project several additional dimensions onto the two-dimensional plot.

You will not use point shape as a differentiator in this exercise, but keep in mind that shape can be as useful as color. Additionally, shape may be easier for the significant fraction of the population who are color blind.

Note: Be careful when combining methods for projecting multiple dimensions. You can easily end up with a plot that is not only hard to interpret, but even harder for you to communicate your observations to your colleagues. For example, if you use three conditioning variables, plus color and shape, you are projecting seven dimensions of your dataset. While this approach might reveal important relationships, it may just create a complex plot.

1. In the code editor pane, under the comments **## Define columns for making scatter plots** select the following code and on the **Run** menu, click **Run cell** (or click the **Run current cell** button on the toolbar) to run it. This specifies the columns you will plot:

```
## Define columns for making scatter plots
plot_cols3 = ["length",
              "curb-weight",
```

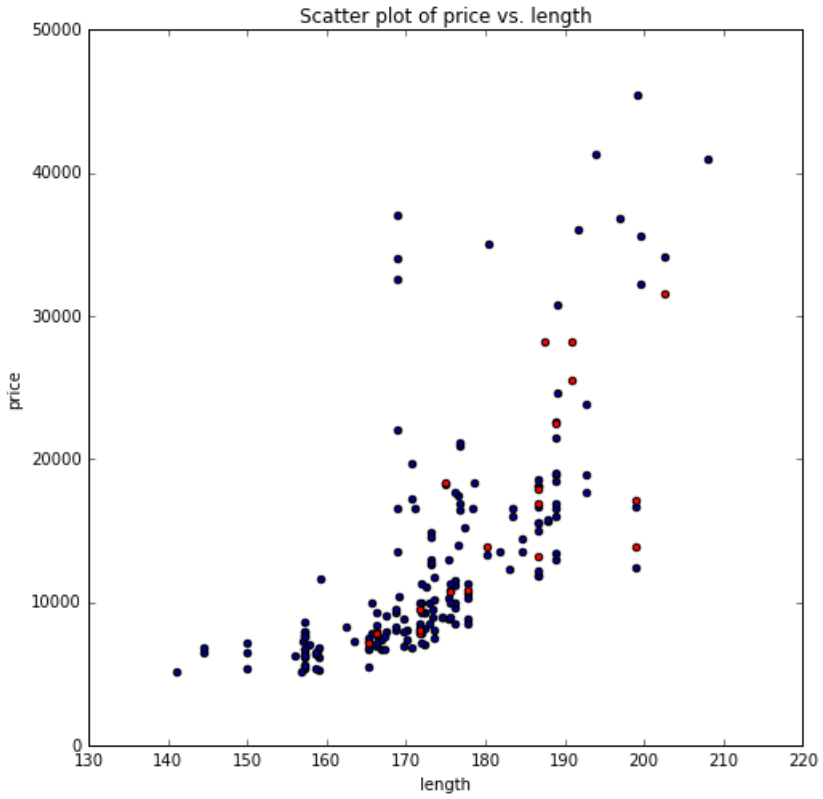
```
                "engine-size",
                "city-mpg"]
```

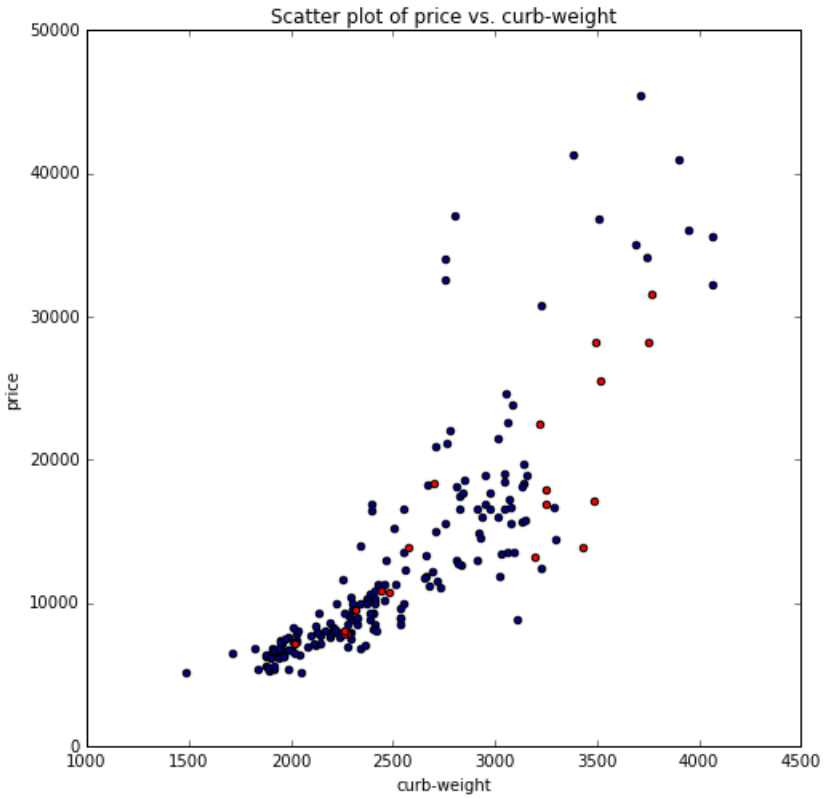
2. In the code editor pane, review the code under the comment **## Create scatter plot**. This code defines a function named **auto_scatter** that computes a scatter plot with the color of the point determined by the **fuel-type** column; DarkBlue for gas and Red for diesel:

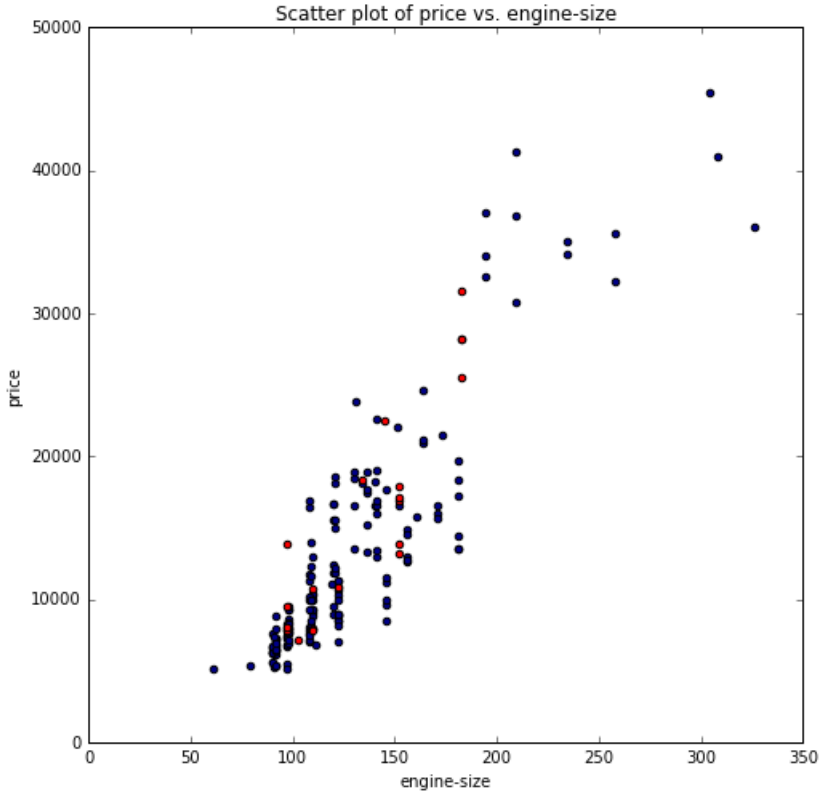
```
## Create scatter plot
def auto_scatter(df, plot_cols):
    import matplotlib.pyplot as plt
    for col in plot_cols:
        fig = plt.figure(figsize=(8, 8))
        ax = fig.gca()
        temp1 = df.ix[df['fuel-type'] == 'gas']
        temp2 = df.ix[df['fuel-type'] == 'diesel']
        if temp1.shape[0] > 0:
            temp1.plot(kind = 'scatter', x = col, y = 'price' ,
                       ax = ax, color = 'DarkBlue')
        if temp2.shape[0] > 0:
            temp2.plot(kind = 'scatter', x = col, y = 'price' ,
                       ax = ax, color = 'Red')
        ax.set_title('Scatter plot of price vs. ' + col)
    return plot_cols
```

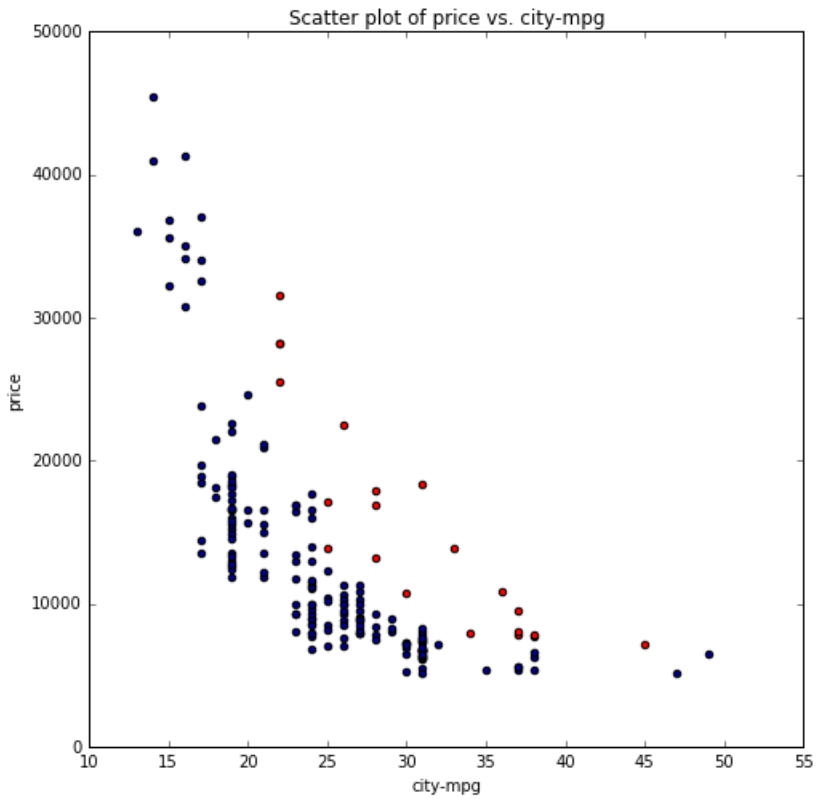
3. In the IPython console, enter the following code and view the scatter plots for the four columns vs. price generated (shown below):

```
ea.auto_scatter(auto_price, plot_cols3)
```




Scatter plot of price vs. curb-weight
price
curb-weight
50000
40000
30000
20000
10000
0
1000
1500
2000
2500
3000
3500
4000
4500


Scatter plot of price vs. engine-size
price
engine-size
50000
40000
30000
20000
10000
0
0
50
100
150
200
250
300
350



4. Examine these plots, and note the following:
    - Each of the four variables plotted against price shown a strong trend, indicating they might be good predictors of price.
    - For the first three plots, diesel cars appear to be on the same trend as gasoline cars. However, a diesel car will tend to be more expensive to achieve the same city-mpg.

5. Review the code under the comment **## Conditioned scatter plots**, as shown here:

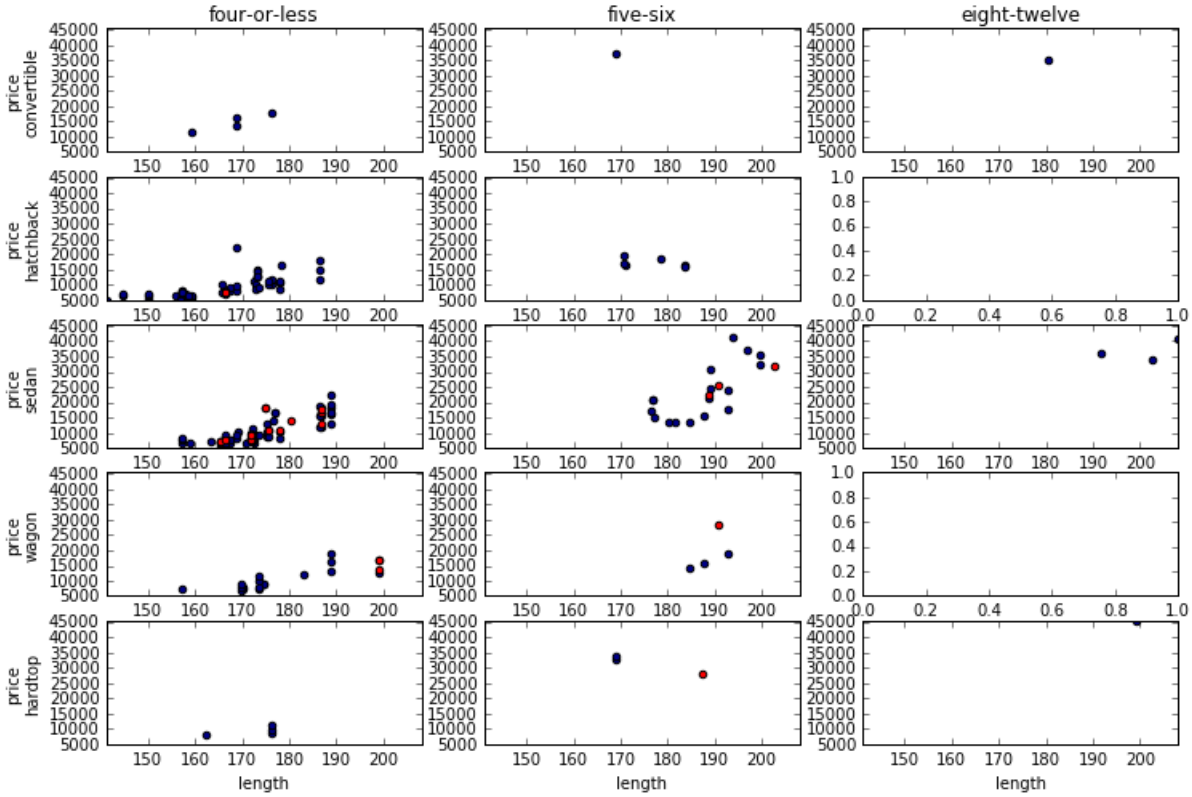
```
## Create conditioned scatter plots
def auto_scatter_cond(df, plot_cols, y, cond_var1, cond_var2):
    import CondPlots as cp
    for col in plot_cols:
        cp.condPltsCol(df, col, y, cond_var1, cond_var2)
    return cond_var1, cond_var2


def condPltsCol(df, col1, col2, var1, var2):
    import matplotlib.pyplot as plt

    ## Find the levels of the conditioning variables
    levs1 = df[var1].unique().tolist()
    num1 = len(levs1)
    levs2 = df[var2].unique().tolist()
    num2 = len(levs2)

    ## Determine the limits for the plots
    xlims = (df[col1].min(), df[col1].max())
    ylims = (df[col2].min(), df[col2].max())

    ## Define a figure and axes for the plot
    fig, ax = plt.subplots(num1, num2, figsize = (12, 8))
```

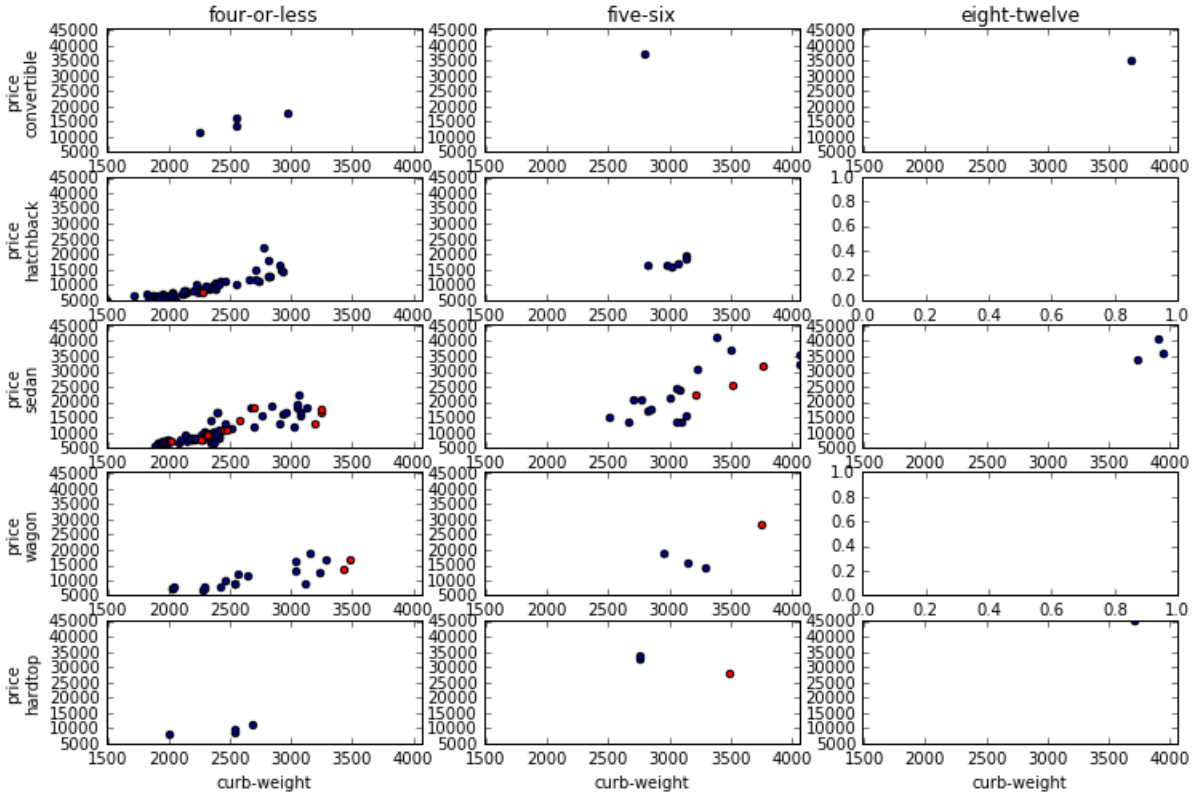
```
    ## Loop over conditioning variables subset the data
    ## and set create scatter plots for each conditioning
    ## variable pair and with data both gas and diesel cars
    for i, val1 in enumerate(levs1):
        for j, val2 in enumerate(levs2):
            temp1 = df.ix[(df[var1] == val1) & (df[var2] == val2) &
(df['fuel-type'] == 'gas')]
            temp2 = df.ix[(df[var1] == val1) & (df[var2] == val2) &
(df['fuel-type'] == 'diesel')]
            if temp1.shape[0] > 0:
                temp1.plot(kind = 'scatter', x = col1, y = col2 , ax =
ax[i,j],
                           xlim = xlims, ylim = ylims, color =
'DarkBlue')
            if temp2.shape[0] > 0:
                temp2.plot(kind = 'scatter', x = col1, y = col2 , ax =
ax[i,j],
                           xlim = xlims, ylim = ylims, color = 'Red')
            ax[i,j].set_title(val1 + ' and ' + val2 )
            ax[i,j].set_xlabel('')

    ## Some lables for the x axis
    ax[i,j].set_xlabel(col1)
    ax[i,(j-1)].set_xlabel(col1)
    return col1, col2
```

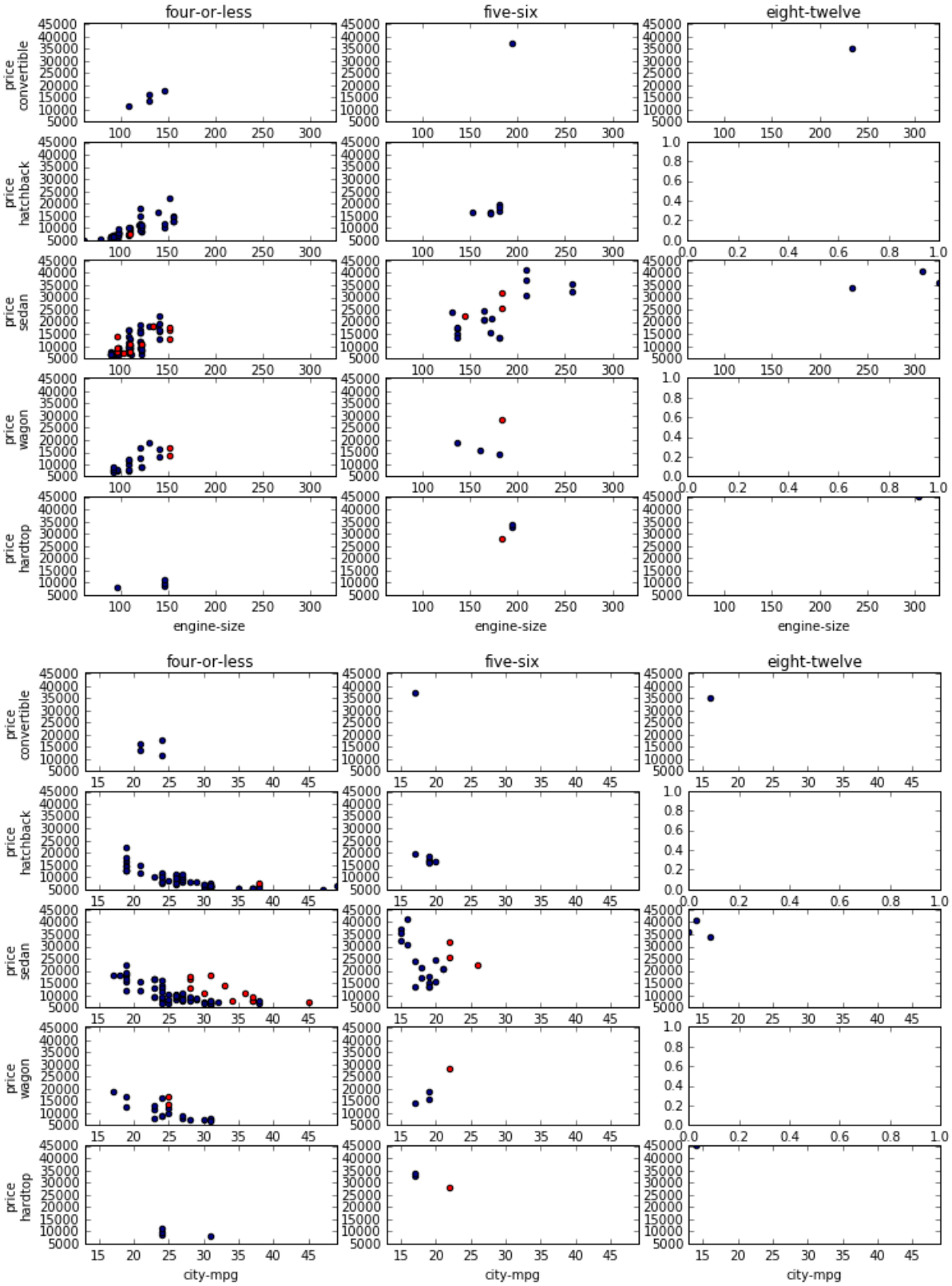
Note: The code defines a function named **auto_scatter_cond**, which is used to create conditioned scatter plots. The working details of computing and displaying these conditioned plots are in the **condPltsCol** function. The same plotting effects and more, can be created with the seaborn module, which is not used to ensure compatibility with Azure Machine Learning.

6. In the IPython console prompt, enter the following command and view the conditioned scatter plots for the four columns vs. price generated (shown below):

```
ea.auto_scatter_cond(auto_price, plot_cols3, 'price', 'body-style',
'num-cylinders')
```


four-or-less
five-six
eight-twelve
price
convertible
hatchback
sedan
wagon
hardtop
length


four-or-less
five-six
eight-twelve
price
convertible
hatchback
sedan
wagon
hardtop
curb-weight



Carefully examine the plots you have created. There is a lot of detail to understand. Generally, you should be able to conclude the following:

- There are no diesel cars for some of the conditioning combinations, such as convertables and cars with eight or more cylinders.
- There are no cars at all for some combinations of conditioning variables.
- Fuel type does not seem to be a good predictor of price in most cases, since the points for diesel cars mostly follow the same trend as gasoline cars.
- Diesel engines are generally used in heavier cars, which then tend to be more fuel efficient.
- As observed previously, the number of cylinders has a significant impact on **price**.
- There is considerable overlap in **price** conditioned by body style, indicating body style will be a weak predictor.

Note: By now, you should realize that exploring a dataset in detail is an open-ended and complex task. You may wish to try other combinations of conditioning variables, and color variable, to find some other interesting relationships in these data.

Load Adult Census Data

Now that you've explored the automobile data, it's time to turn your attention to the adult census data, which you plan to use for a classification solution in which you classify people has low-income if they earn $50,000 or less, or high-income if they earn more than $50,000.

Load the dataset

1. In Spyder, open the **ExploreIncome.py** file from the folder containing the lab files for this module.
2. In the code editor pane, review the code shown below, which loads the adult census data file:

```
def read_income(pathName = "c:\dat203.1x\mod4", fileName = "Adult
Census Income Binary Classification dataset.csv"):
    ## Load the data
    import pandas as pd
    import os

    ## Read the .csv file
    filePath = os.path.join(pathName, fileName)
    return pd.read_csv(filePath)
```

3. At the IPython console, enter the following code:

```
import ExploreIncome as ei
Income = ei.read_income('.')
Income.dtypes
```

4. Note the output which is a summary of the **Income** data frame (shown below). There are both numeric columns and character columns. Character columns have a data type of object.

```
age                int64
 workclass         object
 fnlwgt             int64
 education         object
 education-num      int64
 marital-status    object
 occupation        object
 relationship      object
 race              object
 sex               object
 capital-gain       int64
```

```
capital-loss       int64
hours-per-week     int64
native-country    object
income            object
```

Note: Character data is typically treated as categorical in which each unique string value is a categorical value of the column. When exploring data sets, it is important to understand the relationships between each category and the label values.

Bar plot the categorical features

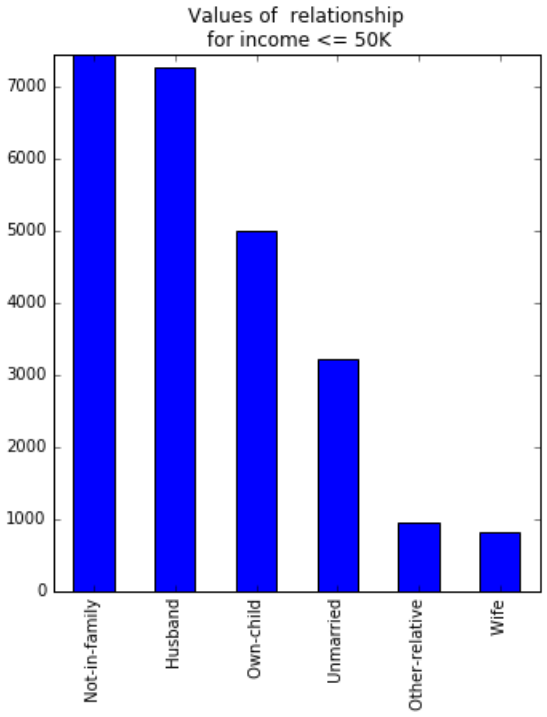
1. In the code editor pane, under the comment **## Plot categorical variables as bar plots**, examine the following code:

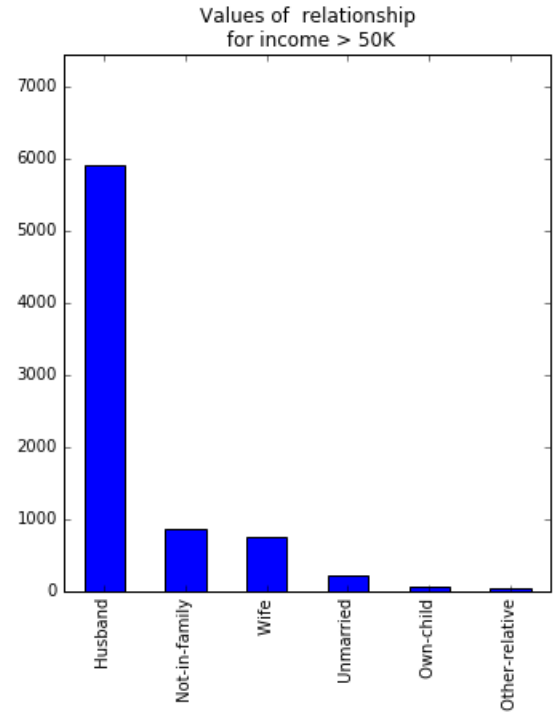
```
## Plot categorical variables as bar plots
def income_barplot(df):
    import numpy as np
    import matplotlib.pyplot as plt

    cols = df.columns.tolist()[:-1]
    for col in cols:
        if(df.ix[:, col].dtype not in [np.int64, np.int32,
np.float64]):
            temp1 = df.ix[df[' income'] == ' <=50K',
col].value_counts()
            temp0 = df.ix[df[' income'] == ' >50K', col].value_counts()

            ylim = [0, max(max(temp1), max(temp0))]
            fig = plt.figure(figsize = (12,6))
            fig.clf()
            ax1 = fig.add_subplot(1, 2, 1)
            ax0 = fig.add_subplot(1, 2, 2)
            temp1.plot(kind = 'bar', ax = ax1, ylim = ylim)
            ax1.set_title('Values of ' + col + '\n for income <= 50K')
            temp0.plot(kind = 'bar', ax = ax0, ylim = ylim)
            ax0.set_title('Values of ' + col + '\n for income > 50K')
    return('Done')
```

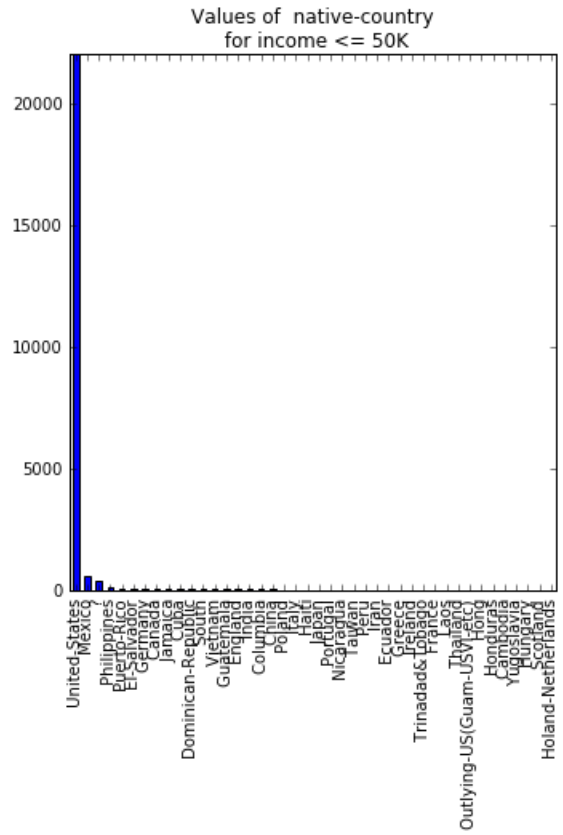
   For each non-numeric column in this data set, the code creates two Pandas series, one for each possible label value. Each series is plotted on the axis for side-by-side plots. To ensure the scales are the same on the two bar plots, ylim is computed in advance and set in the plot method calls.

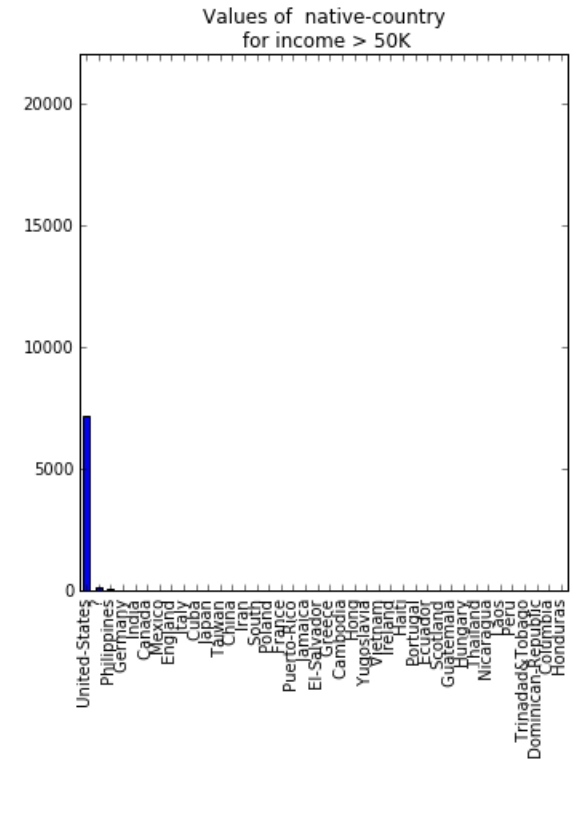
2. In the IPython console, enter the following command and view the series of bar plots generated, (some examples are shown below):

```
ei.income_barplot(income)
```









Examine the two plots shown above.

The first plot shows a feature, **marital-status**, which may have some power in predicting the label category. Note that the proportions of cases are quite different for people with income

<=50K and people with income **>50K**. *Husband* dominates **>50K**, while *Not-in-family* is the largest fraction for **<=50K**. This feature is therefore said to separate the two label categories. However, the separation is not perfect as there is overlap in the cases. This is typical and it is rare to find a single feature which provides perfect separation.

The second plot shows a feature, **native-country**, which may not be that useful in predicting the label category. First, there are quite a few categories. Second, the only category with significant numbers of people is *United States*. There is only a small fraction of the people spread over the other 41 countries. There is no particular pattern noticeable between people with income **<=50K** and people with income **>50K**.

You may wish to step through the plots for other features and consider if these features help to separate the label categories.

Note: When looking at the bar plots of the features conditioned on label value, keep in mind that the number of people with income label of **<=50K** exceeds the number with income **>50K**. Therefore, it is the proportions between the categories of the label which are important, not the absolute values. In other words, are particular values of a feature relatively more or less likely for the label categories?

Box plot the numeric features

There are two possible categories or label values for a person's income category, in this case, **<=50K** or **>50K**. In this exercise you will create box plots for each numeric feature, conditioned on the label value. By examining these box plots, you will determine which features are likely to be useful as predictors of the label categories.

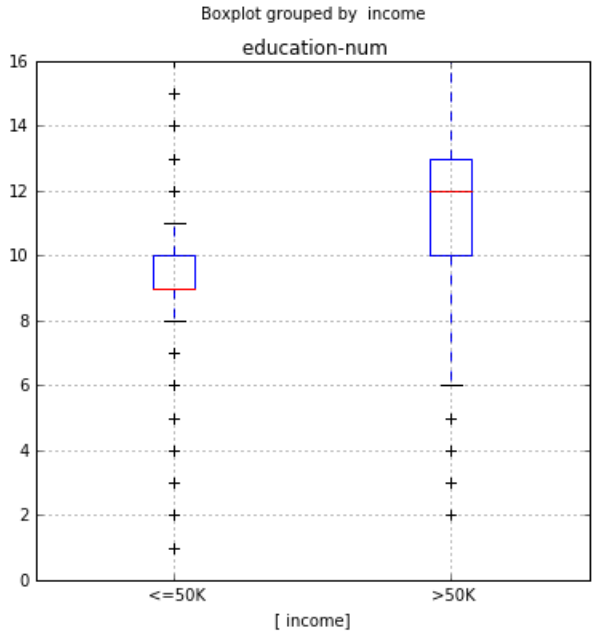
1. In the code editor pane, under the comments **## Plot categorical variables as box plots**, examine the following code:

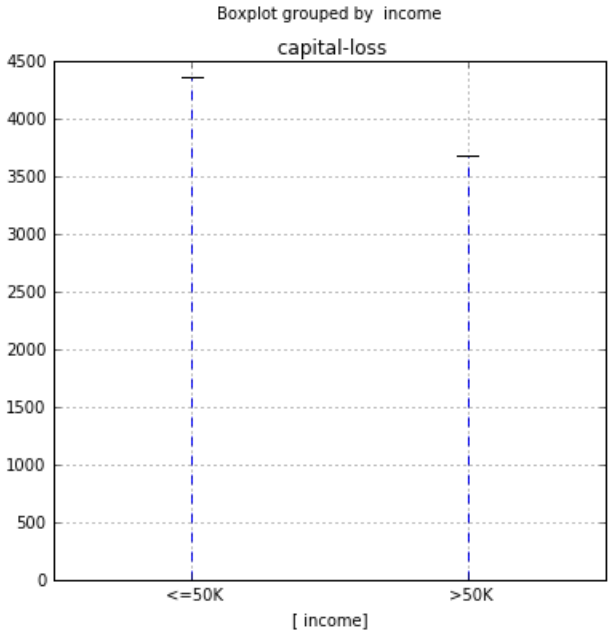
```
## Plot categorical variables as box plots
def income_boxplot(df):
    import numpy as np
    import matplotlib.pyplot as plt

    cols = df.columns.tolist()[:-1]
    for col in cols:
        if(df[col].dtype in [np.int64, np.int32, np.float64]):
            fig = plt.figure(figsize = (6,6))
            fig.clf()
            ax = fig.gca()
            df.boxplot(column = [col], ax = ax, by = [' income'])
    return('Done')
```

   This function creates a box plot for any input column which is of any numeric type (**in [np.int64, np.int32, np.float64]**).

2. In the IPython console, enter the following command and view the series of box plots generated (a few examples are shown below):

```
ei.income_boxplot(Income)
```





Examine the two plots shown above.

The first plot shows a feature, **Income**, which may have some power in predicting the label category. Note that people with income **>50K** tend to have consistently higher **education.num** values than people with income **<=50K**.

The second plot shows a feature, **capital-loss**, which is not likely to be that useful in predicting the label category. Almost all people in the sample have zero (0) capital loss for both label values.

You may wish to step thought the plots for other features and consider if these features help to separate the label categories.

Summary

In this lab, you have explored two datasets using visualization techniques with R or Python. In particular, you have applied the following techniques to explore a regression and a classification dataset:

- Scatter plot matrix
- Conditioned histograms
- Conditioned box plots
- Scatter plots using color to show the values of an additional variable
- Conditioned scatter plot matrices
- Conditioned bar plots