Python Tutorial

Table of Contents

[1 Reading in Data and Basic Statistical Functions 5](#_Toc486249313)

[1.1 Read in the data. 5](#_Toc486249314)

[a) Read the data in as a .csv file. 5](#_Toc486249315)

[b) Read the data in as a .xls file. 5](#_Toc486249316)

[c) Read the data in as a .json file. 5](#_Toc486249317)

[1.2 Find the dimensions of the data set. 5](#_Toc486249318)

[1.3 Find basic information about the data set. 5](#_Toc486249319)

[1.4 Look at the first 5 observations. 6](#_Toc486249320)

[1.5 Calculate mean of numeric variables. 6](#_Toc486249321)

[1.6 Compute summary statistics of the data set. 6](#_Toc486249322)

[1.7 Descriptive statistics functions applied to variables of the data set. 7](#_Toc486249323)

[1.8 Produce a one-way table to describe the frequency of a variable. 7](#_Toc486249324)

[a) Produce a one-way table of a discrete variable. 7](#_Toc486249325)

[b) Produce a one-way table of a categorical variable. 7](#_Toc486249326)

[1.9 Produce a two-way table to describe the frequency of two categorical or discrete variables. 8](#_Toc486249327)

[1.10 Select a subset of the data that meets a certain criterion. 8](#_Toc486249328)

[1.11 Determine the correlation between two continuous variables. 8](#_Toc486249329)

[2 Basic Graphing and Plotting Functions 9](#_Toc486249330)

[2.1 Visualize a single continuous variable by producing a histogram. 9](#_Toc486249331)

[2.2 Visualize a single continuous variable by producing a boxplot. 9](#_Toc486249332)

[2.3 Visualize two continuous variables by producing a scatterplot. 10](#_Toc486249333)

[2.4 Visualize a relationship between two continuous variables by producing a scatterplot and a plotted line of best fit. 11](#_Toc486249334)

[2.5 Visualize a categorical variable by producing a bar chart. 11](#_Toc486249335)

[2.6 Visualize a continuous variable, grouped by a categorical variable, by producing side-by-side boxplots. 12](#_Toc486249336)

[a) Simple side-by-side boxplot without color. 12](#_Toc486249337)

[b) More advanced side-by-side boxplot with color. 13](#_Toc486249338)

[3 Basic Data Wrangling and Manipulation 14](#_Toc486249339)

[3.1 Create a new variable in a data set as a function of existing variables in the data set. 14](#_Toc486249340)

[3.2 Create a new variable in a data set using if/else logic of existing variables in the data set. 14](#_Toc486249341)

[3.3 Create new variables in a data set using mathematical functions applied to existing variables in the data set. 15](#_Toc486249342)

[3.4 Drop variables from a data set. 15](#_Toc486249343)

[3.5 Sort a data set by a variable. 16](#_Toc486249344)

[a) Sort data set by a continuous variable. 16](#_Toc486249345)

[b) Sort data set by a categorical variable. 16](#_Toc486249346)

[3.6 Compute descriptive statistics of continuous variables, grouped by a categorical variable. 16](#_Toc486249347)

[3.7 Add a new row to the bottom of a data set. 16](#_Toc486249348)

[3.8 Create a user-defined function and apply it to a variable in the data set to create a new variable in the data set. 17](#_Toc486249349)

[4 More Advanced Data Wrangling 18](#_Toc486249350)

[4.1 Drop observations with missing information. 18](#_Toc486249351)

[4.2 Merge two data sets together on a common variable. 18](#_Toc486249352)

[a) First, select specific columns of a data set to create two smaller data sets. 18](#_Toc486249353)

[b) Second, we want to merge the two smaller data sets on the common variable. 19](#_Toc486249354)

[c) Finally, we want to check to see if the merged data set is the same as the original data set. 19](#_Toc486249355)

[4.3 Merge two data sets together by index number only. 19](#_Toc486249356)

[a) First, select specific columns of a data set to create two smaller data sets. 19](#_Toc486249357)

[b) Second, we want to join the two smaller data sets. 20](#_Toc486249358)

[c) Finally, we want to check to see if the joined data set is the same as the original data set. 20](#_Toc486249359)

[4.4 Create a pivot table to summarize information about a data set. 20](#_Toc486249360)

[4.5 Return all unique values from a text variable. 21](#_Toc486249361)

[5 Preparation & Basic Regression 21](#_Toc486249362)

[5.1 Pre-process a data set using principal component analysis. 21](#_Toc486249363)

[5.2 Split data into training and testing data and export as a .csv file. 22](#_Toc486249364)

[5.3 Fit a logistic regression model. 22](#_Toc486249365)

[5.4 Fit a linear regression model. 23](#_Toc486249366)

[6 Supervised Machine Learning 24](#_Toc486249367)

[6.1 Fit a logistic regression model on training data and assess against testing data. 24](#_Toc486249368)

[a) Fit a logistic regression model on training data. 24](#_Toc486249369)

[b) Assess the model against the testing data. 25](#_Toc486249370)

[6.2 Fit a linear regression model on training data and assess against testing data. 25](#_Toc486249371)

[a) Fit a linear regression model on training data. 25](#_Toc486249372)

[b) Assess the model against the testing data. 26](#_Toc486249373)

[6.3 Fit a decision tree model on training data and assess against testing data. 26](#_Toc486249374)

[a) Fit a decision tree classification model. 26](#_Toc486249375)

[b) Fit a decision tree regression model. 27](#_Toc486249376)

[6.4 Fit a random forest model on training data and assess against testing data. 28](#_Toc486249377)

[a) Fit a random forest classification model. 28](#_Toc486249378)

[b) Fit a random forest regression model. 29](#_Toc486249379)

[6.5 Fit a gradient boosting model on training data and assess against testing data. 30](#_Toc486249380)

[a) Fit a gradient boosting classification model. 30](#_Toc486249381)

[b) Fit a gradient boosting regression model. 31](#_Toc486249382)

[6.6 Fit an extreme gradient boosting model on training data and assess against testing data. 31](#_Toc486249383)

[a) Fit an extreme gradient boosting classification model on training data and assess against testing data. 31](#_Toc486249384)

[b) Fit an extreme gradient boosting regression model on training data and assess against testing data. 32](#_Toc486249385)

[6.7 Fit a support vector model on training data and assess against testing data. 33](#_Toc486249386)

[a) Fit a support vector classification model. 33](#_Toc486249387)

[b) Fit a support vector regression model. 34](#_Toc486249388)

[6.8 Fit a neural network model on training data and assess against testing data. 35](#_Toc486249389)

[a) Fit a neural network classification model. 35](#_Toc486249390)

[b) Fit a neural network regression model. 35](#_Toc486249391)

[7 Unsupervised Machine Learning 36](#_Toc486249392)

[7.1 KMeans Clustering 36](#_Toc486249393)

[7.2 Spectral Clustering 37](#_Toc486249394)

[7.3 Ward Hierarchical Clustering 37](#_Toc486249395)

[7.4 DBSCAN 37](#_Toc486249396)

[7.5 Self-organizing map 38](#_Toc486249397)

[8 Forecasting 38](#_Toc486249398)

[8.1 Fit an ARIMA model to a timeseries. 38](#_Toc486249399)

[a) Plot the timeseries. 38](#_Toc486249400)

[b) Fit an ARIMA model, predict 2 years (24 months) out and plot predictions. 39](#_Toc486249401)

[8.2 Fit a Simple Exponential Smoothing model to a timeseries. 40](#_Toc486249402)

[a) Plot the timeseries. 40](#_Toc486249403)

[b) Fit a Simple Exponential Smoothing model, predict 2 years (24 months) out and plot predictions. 41](#_Toc486249404)

[8.3 Fit a Holt-Winters model to a timeseries. 42](#_Toc486249405)

[a) Plot the timeseries. 42](#_Toc486249406)

[b) Fit a Holt-Winters additive model, predict 2 years (24 months) out and plot predictions. 43](#_Toc486249407)

[9 Model Evaluation & Selection 45](#_Toc486249408)

[9.1 Evaluate the accuracy of regression models. 45](#_Toc486249409)

[a) Evaluation on training data. 45](#_Toc486249410)

[b) Evaluation on testing data. 45](#_Toc486249411)

[9.2 Evaluate the accuracy of classification models. 46](#_Toc486249412)

[a) Evaluation on training data. 46](#_Toc486249413)

[b) Evaluation on testing data. 46](#_Toc486249414)

[9.3 Evaluation with cross validation. 46](#_Toc486249415)

[a) KFold 46](#_Toc486249416)

[b) ShuffleSplit 47](#_Toc486249417)

[10 Text Analytics 47](#_Toc486249418)

[11 Deep Learning 47](#_Toc486249419)

[Appendix 47](#_Toc486249420)

[1 Built-in Python Data Types 47](#_Toc486249421)

[2 Python Plotting Packages 48](#_Toc486249422)

[3 Python packages used in this tutorial 48](#_Toc486249423)

[Alphabetical Index 49](#_Toc486249424)

[Array 49](#_Toc486249425)

[Bytes & Byte arrays 49](#_Toc486249426)

[Data Frame 49](#_Toc486249427)

[datetime 49](#_Toc486249428)

[Dictionary 50](#_Toc486249429)

[List 50](#_Toc486249430)

[Series 50](#_Toc486249431)

[Sets & Frozen Sets 51](#_Toc486249432)

[str 51](#_Toc486249433)

First, you need to import several important Python packages for data manipulation and scientific computing. The [pandas](#PANDAS) package is helpful for data manipulation and the [NumPy](#NUMPY) package is helpful for scientific computing.

import pandas as pd  
import numpy as np

In Python, comments are indicated in code with a "#" character, and arrays and matrices are zero-indexed.

# 1 Reading in Data and Basic Statistical Functions

## 1.1 Read in the data.

The following demonstrate importing data into Python given 3 different file formats. The pandas package is able to read all 3 formats, as well as many others, using [Python IO tools](http://pandas.pydata.org/pandas-docs/version/0.20/io.html).

### a) Read the data in as a .csv file.

student = pd.read\_csv('/Users/class.csv')

### b) Read the data in as a .xls file.

# Notice you must specify the file location, as well as the name of the sheet   
# of the .xls file you want to import  
student\_xls = pd.read\_excel(open('/Users/class.xls', 'rb'),   
 sheetname='class')

### c) Read the data in as a .json file.

student\_json = pd.read\_json('/Users/class.json')

## 1.2 Find the dimensions of the data set.

The dimensions of a [DataFrame](#DataFrame) in Python are known as an attribute of the object. Therefore, you can state the data name followed by [.shape](http://pandas.pydata.org/pandas-docs/version/0.17.0/generated/pandas.DataFrame.shape.html) to return the dimensions of the data, with the first integer indicating the number of rows and the second indicating the number of columns.

print(student.shape)

## (19, 5)

## 1.3 Find basic information about the data set.

Information about a [DataFrame](#DataFrame) is available by calling the [info()](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.info.html) function on the data.

print(student.info())

## <class 'pandas.core.frame.DataFrame'>  
## RangeIndex: 19 entries, 0 to 18  
## Data columns (total 5 columns):  
## Name 19 non-null object  
## Sex 19 non-null object  
## Age 19 non-null int64  
## Height 19 non-null float64  
## Weight 19 non-null float64  
## dtypes: float64(2), int64(1), object(2)  
## memory usage: 840.0+ bytes  
## None

## 1.4 Look at the first 5 observations.

The first 5 observations of a [DataFrame](#DataFrame) are available by calling the [head()](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.head.html) function on the data. By default, head() returns 5 observations. To return the first *n* observations, pass the integer *n* into the function. The [tail()](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.tail.html) function is analogous and returns the last observations.

print(student.head())

## Name Sex Age Height Weight  
## 0 Alfred M 14 69.0 112.5  
## 1 Alice F 13 56.5 84.0  
## 2 Barbara F 13 65.3 98.0  
## 3 Carol F 14 62.8 102.5  
## 4 Henry M 14 63.5 102.5

## 1.5 Calculate mean of numeric variables.

The mean of numeric variables of a [DataFrame](#DataFrame) are available by calling the [mean()](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.mean.html) function on the data.

print(student.mean())

## Age 13.315789  
## Height 62.336842  
## Weight 100.026316  
## dtype: float64

## 1.6 Compute summary statistics of the data set.

Summary statistics of a [DataFrame](#DataFrame) are available by calling the [describe()](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.describe.html) function on the data.

print(student.describe())

## Age Height Weight  
## count 19.000000 19.000000 19.000000  
## mean 13.315789 62.336842 100.026316  
## std 1.492672 5.127075 22.773933  
## min 11.000000 51.300000 50.500000  
## 25% 12.000000 58.250000 84.250000  
## 50% 13.000000 62.800000 99.500000  
## 75% 14.500000 65.900000 112.250000  
## max 16.000000 72.000000 150.000000

## 1.7 Descriptive statistics functions applied to variables of the data set.

# Notice the subsetting of student with [] and the name of the variable in   
# quotes   
print(student["Weight"].std())

## 22.773933493879046

print(student["Weight"].sum())

## 1900.5

print(student["Weight"].count())

## 19

print(student["Weight"].max())

## 150.0

print(student["Weight"].min())

## 50.5

print(student["Weight"].median())

## 99.5

## 1.8 Produce a one-way table to describe the frequency of a variable.

### a) Produce a one-way table of a discrete variable.

# columns = "count" indicates to make the descriptive portion of the table   
# the counts of each level of the index variable  
print(pd.crosstab(index=student["Age"], columns="count"))

## col\_0 count  
## Age   
## 11 2  
## 12 5  
## 13 3  
## 14 4  
## 15 4  
## 16 1

### b) Produce a one-way table of a categorical variable.

print(pd.crosstab(index=student["Sex"], columns="count"))

## col\_0 count  
## Sex   
## F 9  
## M 10

[pd.crosstab()](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.crosstab.html)

## 1.9 Produce a two-way table to describe the frequency of two categorical or discrete variables.

# Notice the specification of a variable for the columns argument, instead   
# of "count"

## Sex F M  
## Age   
## 11 1 1  
## 12 2 3  
## 13 2 1  
## 14 2 2  
## 15 2 2  
## 16 0 1

[pd.crosstab()](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.crosstab.html)

## 1.10 Select a subset of the data that meets a certain criterion.

females = student.query('Sex == "F"')  
print(females.head())

## Name Sex Age Height Weight  
## 1 Alice F 13 56.5 84.0  
## 2 Barbara F 13 65.3 98.0  
## 3 Carol F 14 62.8 102.5  
## 6 Jane F 12 59.8 84.5  
## 7 Janet F 15 62.5 112.5

[query()](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.query.html)

## 1.11 Determine the correlation between two continuous variables.

# axis = 1 option indicates to concatenate column-wise  
height\_weight = pd.concat([student["Height"], student["Weight"]], axis = 1)  
print(height\_weight.corr(method = "pearson"))

## Height Weight  
## Height 1.000000 0.877785  
## Weight 0.877785 1.000000

[pd.concat()](http://pandas.pydata.org/pandas-docs/version/0.20/generated/pandas.concat.html) | [corr()](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.corr.html)

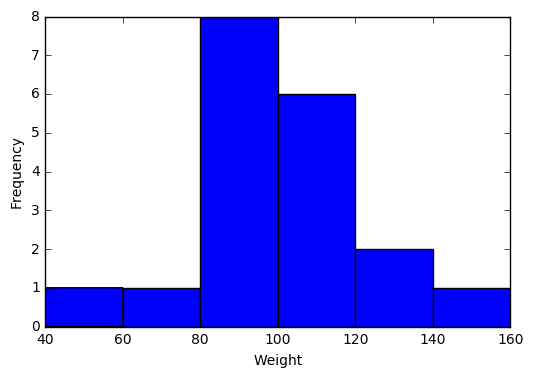
# 2 Basic Graphing and Plotting Functions

The Matplotlib [PyPlot](#PYPLOT) package is a standard Python package to use for plotting. For more information on other Python plotting packages, please see the Appendix Section 2.

import matplotlib.pyplot as plt

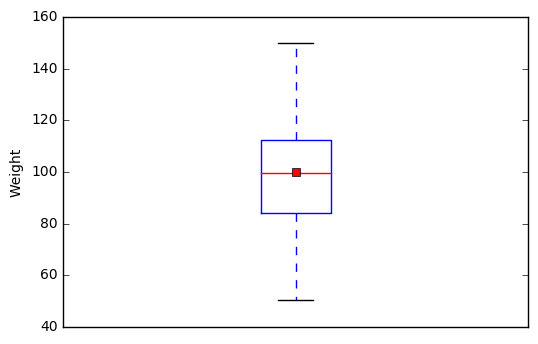
## 2.1 Visualize a single continuous variable by producing a histogram.

# Notice the labeling of the axes  
plt.hist(student["Weight"], bins=[40,60,80,100,120,140,160])  
plt.xlabel('Weight')  
plt.ylabel('Frequency')  
plt.show()

Output: 

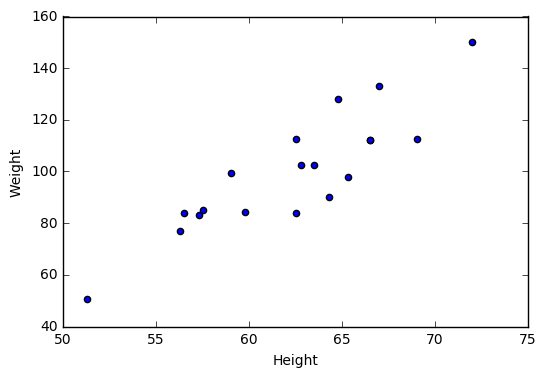
## 2.2 Visualize a single continuous variable by producing a boxplot.

# showmeans=True tells Python to plot the mean of the variable on the boxplot   
plt.boxplot(student["Weight"], showmeans=True)  
  
# prevents Python from printing a "1" at the bottom of the boxplot  
plt.xticks([])  
  
plt.ylabel('Weight')  
plt.show()

Output: 

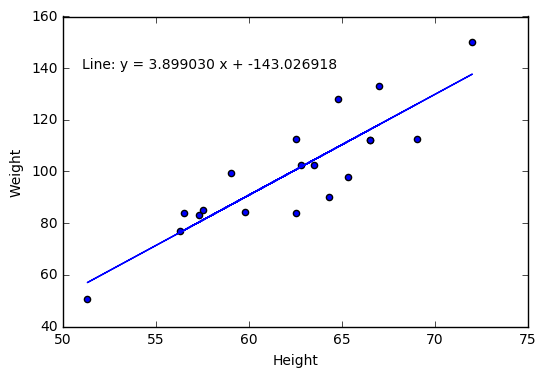
## 2.3 Visualize two continuous variables by producing a scatterplot.

# Notice here you specify the x variable, followed by the y variable   
plt.scatter(student["Height"], student["Weight"])  
plt.xlabel("Height")  
plt.ylabel("Weight")  
plt.show()

Output: 

## 2.4 Visualize a relationship between two continuous variables by producing a scatterplot and a plotted line of best fit.

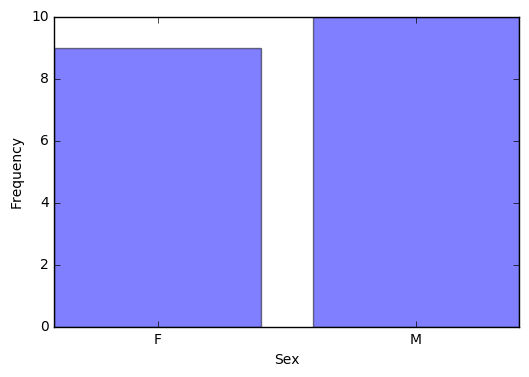
x = student["Height"]  
y = student["Weight"]  
  
# np.polyfit() models Weight as a function of Height and returns the   
# parameters  
m, b = np.polyfit(x, y, 1)  
plt.scatter(x, y)  
  
# plt.text() prints the equation of the line of best fit, with the first two   
# arguments specifying the x and y locations of the text, respectively   
# %f indicates to print a floating point number, that is specified following  
# the string and a % character  
plt.text(51, 140, "Line: y = %f x + %f"% (m,b))  
plt.plot(x, m\*x + b)  
plt.xlabel("Height")  
plt.ylabel("Weight")  
plt.show()

Output: 

[np.polyfit()](https://docs.scipy.org/doc/numpy/reference/generated/numpy.polyfit.html)

## 2.5 Visualize a categorical variable by producing a bar chart.

# Get the counts of Sex   
counts = pd.crosstab(index=student["Sex"], columns="count")  
  
# len() returns the number of categories of Sex (2)  
# np.arange() creates a vector of the specified length  
num = np.arange(len(counts))  
  
# alpha = 0.5 changes the transparency of the bars  
plt.bar(num, counts["count"], align='center', alpha=0.5)  
  
# Set the xticks to be the indices of counts  
plt.xticks(num, counts.index)  
plt.xlabel("Sex")  
plt.ylabel("Frequency")  
plt.show()

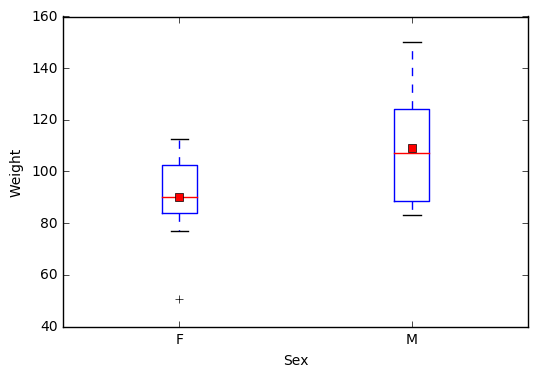
Output: 

[np.arange()](https://docs.scipy.org/doc/numpy/reference/generated/numpy.arange.html)

## 2.6 Visualize a continuous variable, grouped by a categorical variable, by producing side-by-side boxplots.

### a) Simple side-by-side boxplot without color.

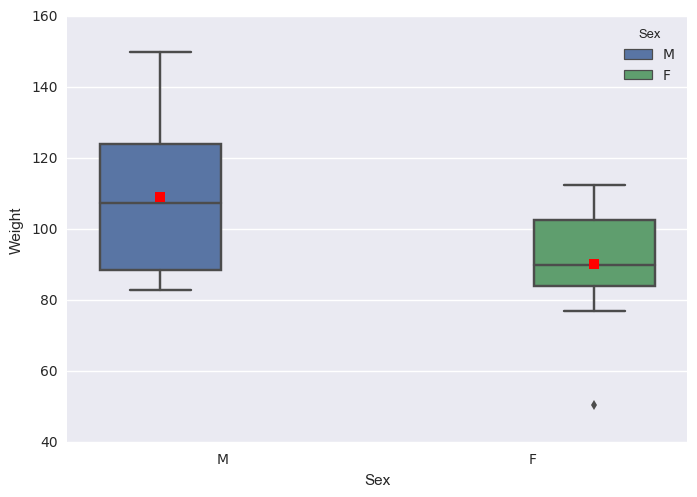
# Subset data set to return only female weights, and then only male weights   
Weight\_F = np.array(student.query('Sex == "F"')["Weight"])  
Weight\_M = np.array(student.query('Sex == "M"')["Weight"])  
Weights = [Weight\_F, Weight\_M]  
  
# PyPlot automatically plots the two weights side-by-side since Weights   
# is a 2D array  
plt.boxplot(Weights, showmeans=True, labels=('F', 'M'))  
plt.xlabel('Sex')  
plt.ylabel('Weight')  
plt.show()

Output: 

[np.array()](https://docs.scipy.org/doc/numpy/reference/generated/numpy.array.html)

### b) More advanced side-by-side boxplot with color.

import seaborn as sns  
sns.boxplot(x="Sex", y="Weight", hue="Sex", data = student, showmeans=True)  
plt.show()

Output:  [seaborn](#SEABORN)

# 3 Basic Data Wrangling and Manipulation

## 3.1 Create a new variable in a data set as a function of existing variables in the data set.

# Notice here how you can create the BMI column in the data set   
# just by naming it   
student["BMI"] = student["Weight"] / student["Height"]\*\*2 \* 703  
print(student.head())

## Name Sex Age Height Weight BMI  
## 0 Alfred M 14 69.0 112.5 16.611531  
## 1 Alice F 13 56.5 84.0 18.498551  
## 2 Barbara F 13 65.3 98.0 16.156788  
## 3 Carol F 14 62.8 102.5 18.270898  
## 4 Henry M 14 63.5 102.5 17.870296

## 3.2 Create a new variable in a data set using if/else logic of existing variables in the data set.

# Notice the use of the np.where() function for a single condition   
student["BMI Class"] = np.where(student["BMI"] < 19.0, "Underweight",   
 "Healthy")  
print(student.head())

## Name Sex Age Height Weight BMI BMI Class  
## 0 Alfred M 14 69.0 112.5 16.611531 Underweight  
## 1 Alice F 13 56.5 84.0 18.498551 Underweight  
## 2 Barbara F 13 65.3 98.0 16.156788 Underweight  
## 3 Carol F 14 62.8 102.5 18.270898 Underweight  
## 4 Henry M 14 63.5 102.5 17.870296 Underweight

[np.where()](https://docs.scipy.org/doc/numpy/reference/generated/numpy.where.html)

## 3.3 Create new variables in a data set using mathematical functions applied to existing variables in the data set.

Using the [np.log()](https://docs.scipy.org/doc/numpy/reference/generated/numpy.log.html), [np.exp()](https://docs.scipy.org/doc/numpy/reference/generated/numpy.exp.html), [np.sqrt()](https://docs.scipy.org/doc/numpy/reference/generated/numpy.sqrt.html), [np.where()](https://docs.scipy.org/doc/numpy/reference/generated/numpy.where.html), and [np.abs()](https://docs.scipy.org/doc/numpy/reference/generated/numpy.absolute.html) functions.

student["LogWeight"] = np.log(student["Weight"])  
student["ExpAge"] = np.exp(student["Age"])  
student["SqrtHeight"] = np.sqrt(student["Height"])  
student["BMI Neg"] = np.where(student["BMI"] < 19.0, -student["BMI"],   
 student["BMI"])  
student["BMI Pos"] = np.abs(student["BMI Neg"])  
  
# Create a boolean variable  
student["BMI Check"] = (student["BMI Pos"] == student["BMI"])

## Name Sex Age Height Weight BMI BMI Class LogWeight \  
## 0 Alfred M 14 69.0 112.5 16.611531 Underweight 4.722953   
## 1 Alice F 13 56.5 84.0 18.498551 Underweight 4.430817   
## 2 Barbara F 13 65.3 98.0 16.156788 Underweight 4.584967   
## 3 Carol F 14 62.8 102.5 18.270898 Underweight 4.629863   
## 4 Henry M 14 63.5 102.5 17.870296 Underweight 4.629863   
##   
## ExpAge SqrtHeight BMI Neg BMI Pos BMI Check   
## 0 1.202604e+06 8.306624 -16.611531 16.611531 True   
## 1 4.424134e+05 7.516648 -18.498551 18.498551 True   
## 2 4.424134e+05 8.080842 -16.156788 16.156788 True   
## 3 1.202604e+06 7.924645 -18.270898 18.270898 True   
## 4 1.202604e+06 7.968689 -17.870296 17.870296 True

## 3.4 Drop variables from a data set.

# axis = 1 indicates to drop columns instead of rows  
student = student.drop(["LogWeight", "ExpAge", "SqrtHeight", "BMI Neg",   
 "BMI Pos", "BMI Check"], axis = 1)  
print(student.head())

## Name Sex Age Height Weight BMI BMI Class  
## 0 Alfred M 14 69.0 112.5 16.611531 Underweight  
## 1 Alice F 13 56.5 84.0 18.498551 Underweight  
## 2 Barbara F 13 65.3 98.0 16.156788 Underweight  
## 3 Carol F 14 62.8 102.5 18.270898 Underweight  
## 4 Henry M 14 63.5 102.5 17.870296 Underweight

[drop()](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.drop.html)

## 3.5 Sort a data set by a variable.

### a) Sort data set by a continuous variable.

# Notice kind="mergesort" which indicates to use a stable sorting   
# algorithm   
student = student.sort\_values(by="Age", kind="mergesort")  
print(student.head())

## Name Sex Age Height Weight BMI BMI Class  
## 10 Joyce F 11 51.3 50.5 13.490001 Underweight  
## 17 Thomas M 11 57.5 85.0 18.073346 Underweight  
## 5 James M 12 57.3 83.0 17.771504 Underweight  
## 6 Jane F 12 59.8 84.5 16.611531 Underweight  
## 9 John M 12 59.0 99.5 20.094369 Healthy

### b) Sort data set by a categorical variable.

student = student.sort\_values(by="Sex", kind="mergesort")  
# Notice that the data is now sorted first by Sex and then within Sex by Age   
print(student.head())

## Name Sex Age Height Weight BMI BMI Class  
## 10 Joyce F 11 51.3 50.5 13.490001 Underweight  
## 6 Jane F 12 59.8 84.5 16.611531 Underweight  
## 12 Louise F 12 56.3 77.0 17.077695 Underweight  
## 1 Alice F 13 56.5 84.0 18.498551 Underweight  
## 2 Barbara F 13 65.3 98.0 16.156788 Underweight

[sort\_values()](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.sort_values.html)

## 3.6 Compute descriptive statistics of continuous variables, grouped by a categorical variable.

print(student.groupby(by="Sex").mean())

## Age Height Weight BMI  
## Sex   
## F 13.222222 60.588889 90.111111 17.051039  
## M 13.400000 63.910000 108.950000 18.594243

[groupby()](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.groupby.html)

## 3.7 Add a new row to the bottom of a data set.

# Look at the tail of the data currently  
print(student.tail())

## Name Sex Age Height Weight BMI BMI Class  
## 0 Alfred M 14 69.0 112.5 16.611531 Underweight  
## 4 Henry M 14 63.5 102.5 17.870296 Underweight  
## 16 Ronald M 15 67.0 133.0 20.828470 Healthy  
## 18 William M 15 66.5 112.0 17.804511 Underweight  
## 14 Philip M 16 72.0 150.0 20.341435 Healthy

student = student.append({'Name':'Jane', 'Sex':'F', 'Age':14, 'Height':56.3,   
 'Weight':77.0, 'BMI':17.077695,   
 'BMI Class': 'Underweight'},   
 ignore\_index=True)  
  
# Notice the change in the indices because of the ignore\_index=True option   
# which allows for a Series, or one-dimensional DataFrame, to be appended   
# to an existing DataFrame

## Name Sex Age Height Weight BMI BMI Class  
## 15 Henry M 14 63.5 102.5 17.870296 Underweight  
## 16 Ronald M 15 67.0 133.0 20.828470 Healthy  
## 17 William M 15 66.5 112.0 17.804511 Underweight  
## 18 Philip M 16 72.0 150.0 20.341435 Healthy  
## 19 Jane F 14 56.3 77.0 17.077695 Underweight

[append()](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.append.html)

## 3.8 Create a user-defined function and apply it to a variable in the data set to create a new variable in the data set.

def toKG(lb):  
 return (0.45359237 \* lb)  
  
student["Weight KG"] = student["Weight"].apply(toKG)  
print(student.head())

## Name Sex Age Height Weight BMI BMI Class Weight KG  
## 0 Joyce F 11 51.3 50.5 13.490001 Underweight 22.906415  
## 1 Jane F 12 59.8 84.5 16.611531 Underweight 38.328555  
## 2 Louise F 12 56.3 77.0 17.077695 Underweight 34.926612  
## 3 Alice F 13 56.5 84.0 18.498551 Underweight 38.101759  
## 4 Barbara F 13 65.3 98.0 16.156788 Underweight 44.452052

[apply()](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.apply.html) | [user-defined functions](https://www.tutorialspoint.com/python/python_functions.htm)

# 4 More Advanced Data Wrangling

## 4.1 Drop observations with missing information.

# Notice the use of the fish data set because it has some missing   
# observations   
fish = pd.read\_csv('/Users/fish.csv')  
  
# First sort by Weight, requesting those with NA for Weight first   
fish = fish.sort\_values(by='Weight', kind='mergesort', na\_position='first')  
print(fish.head())

## Species Weight Length1 Length2 Length3 Height Width  
## 13 Bream NaN 29.5 32.0 37.3 13.9129 5.0728  
## 40 Roach 0.0 19.0 20.5 22.8 6.4752 3.3516  
## 72 Perch 5.9 7.5 8.4 8.8 2.1120 1.4080  
## 145 Smelt 6.7 9.3 9.8 10.8 1.7388 1.0476  
## 147 Smelt 7.0 10.1 10.6 11.6 1.7284 1.1484

--

new\_fish = fish.dropna()  
print(new\_fish.head())

## Species Weight Length1 Length2 Length3 Height Width  
## 40 Roach 0.0 19.0 20.5 22.8 6.4752 3.3516  
## 72 Perch 5.9 7.5 8.4 8.8 2.1120 1.4080  
## 145 Smelt 6.7 9.3 9.8 10.8 1.7388 1.0476  
## 147 Smelt 7.0 10.1 10.6 11.6 1.7284 1.1484  
## 146 Smelt 7.5 10.0 10.5 11.6 1.9720 1.1600

[dropna()](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.dropna.html)

## 4.2 Merge two data sets together on a common variable.

### a) First, select specific columns of a data set to create two smaller data sets.

# Notice the use of the student data set again, however we want to reload it  
# without the changes we've made previously   
student = pd.read\_csv('/Users/class.csv')  
student1 = pd.concat([student["Name"], student["Sex"], student["Age"]],   
 axis = 1)  
print(student1.head())

## Name Sex Age  
## 0 Alfred M 14  
## 1 Alice F 13  
## 2 Barbara F 13  
## 3 Carol F 14  
## 4 Henry M 14

--

student2 = pd.concat([student["Name"], student["Height"], student["Weight"]],   
 axis = 1)  
print(student2.head())

## Name Height Weight  
## 0 Alfred 69.0 112.5  
## 1 Alice 56.5 84.0  
## 2 Barbara 65.3 98.0  
## 3 Carol 62.8 102.5  
## 4 Henry 63.5 102.5

### b) Second, we want to merge the two smaller data sets on the common variable.

new = pd.merge(student1, student2, on="Name")  
print(new.head())

## Name Sex Age Height Weight  
## 0 Alfred M 14 69.0 112.5  
## 1 Alice F 13 56.5 84.0  
## 2 Barbara F 13 65.3 98.0  
## 3 Carol F 14 62.8 102.5  
## 4 Henry M 14 63.5 102.5

[pd.merge()](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.merge.html)

### c) Finally, we want to check to see if the merged data set is the same as the original data set.

print(student.equals(new))

## True

[equals()](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.equals.html)

## 4.3 Merge two data sets together by index number only.

### a) First, select specific columns of a data set to create two smaller data sets.

newstudent1 = pd.concat([student["Name"], student["Sex"], student["Age"]],   
 axis = 1)  
print(newstudent1.head())

## Name Sex Age  
## 0 Alfred M 14  
## 1 Alice F 13  
## 2 Barbara F 13  
## 3 Carol F 14  
## 4 Henry M 14

--

newstudent2 = pd.concat([student["Height"], student["Weight"]], axis = 1)  
print(newstudent2.head())

## Height Weight  
## 0 69.0 112.5  
## 1 56.5 84.0  
## 2 65.3 98.0  
## 3 62.8 102.5  
## 4 63.5 102.5

### b) Second, we want to join the two smaller data sets.

new2 = newstudent1.join(newstudent2)  
print(new2.head())

## Name Sex Age Height Weight  
## 0 Alfred M 14 69.0 112.5  
## 1 Alice F 13 56.5 84.0  
## 2 Barbara F 13 65.3 98.0  
## 3 Carol F 14 62.8 102.5  
## 4 Henry M 14 63.5 102.5

[join()](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.join.html)

### c) Finally, we want to check to see if the joined data set is the same as the original data set.

print(student.equals(new2))

## True

## 4.4 Create a pivot table to summarize information about a data set.

# Notice we are using a new data set that needs to be read into the   
# environment   
price = pd.read\_csv('/Users/price.csv')  
  
# The following code is used to remove the ',' and '$' characters from   
# the ACTUAL colum so that the values can be summed   
from re import sub  
from decimal import Decimal  
def trim\_money(money):  
 return(float(Decimal(sub(r'[^\d.]', '', money))))  
  
price["REVENUE"] = price["ACTUAL"].apply(trim\_money)  
table = pd.pivot\_table(price, index=["COUNTRY", "STATE", "PRODTYPE",   
 "PRODUCT"], values="REVENUE",   
 aggfunc=np.sum)  
print(table.head())

## REVENUE  
## COUNTRY STATE PRODTYPE PRODUCT   
## Canada British Columbia FURNITURE BED 197706.6  
## SOFA 216282.6  
## OFFICE CHAIR 200905.2  
## DESK 186262.2  
## Ontario FURNITURE BED 194493.6

Note: [pd.pivot\_table()](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.pivot_table.html) is similar to the [pd.pivot()](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.pivot.html) function

[re](#re) | [Decimal](#DECIMAL)

## 4.5 Return all unique values from a text variable.

print(np.unique(price["STATE"]))

## ['Baja California Norte' 'British Columbia' 'California' 'Campeche'  
## 'Colorado' 'Florida' 'Illinois' 'Michoacan' 'New York' 'North Carolina'  
## 'Nuevo Leon' 'Ontario' 'Quebec' 'Saskatchewan' 'Texas' 'Washington']

[np.unique()](https://docs.scipy.org/doc/numpy/reference/generated/numpy.unique.html)

The following sections focus on the Python [sklearn](#SKLEARN) package. Also, in the following sections several data set will be used more than once for prediction and modeling. Often, they will be re-read into the environment so we are always going back to the original, raw data.

# 5 Preparation & Basic Regression

## 5.1 Pre-process a data set using principal component analysis.

# Notice we are using a new data set that needs to be read into the   
# environment   
iris = pd.read\_csv('/Users/iris.csv')  
features = iris.drop(["Target"], axis = 1)  
  
from sklearn import preprocessing  
features\_scaled = preprocessing.scale(features.as\_matrix())  
  
from sklearn.decomposition import PCA  
  
pca = PCA(n\_components = 4)  
pca = pca.fit(features\_scaled)  
print(np.transpose(pca.components\_))

## [[ 0.52237162 0.37231836 -0.72101681 -0.26199559]  
## [-0.26335492 0.92555649 0.24203288 0.12413481]  
## [ 0.58125401 0.02109478 0.14089226 0.80115427]  
## [ 0.56561105 0.06541577 0.6338014 -0.52354627]]

[PCA](http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html#sklearn.decomposition.PCA) | [np.transpose()](https://docs.scipy.org/doc/numpy-1.12.0/reference/generated/numpy.transpose.html)

## 5.2 Split data into training and testing data and export as a .csv file.

from sklearn.model\_selection import train\_test\_split  
  
target = iris["Target"]  
  
# The following code splits the iris data set into 70% train and 30% test  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(features, target,   
 test\_size = 0.3,   
 random\_state = 29)  
train\_x = pd.DataFrame(X\_train)  
train\_y = pd.DataFrame(Y\_train)  
test\_x = pd.DataFrame(X\_test)  
test\_y = pd.DataFrame(Y\_test)  
  
train = pd.concat([train\_x, train\_y], axis = 1)  
test = pd.concat([test\_x, test\_y], axis = 1)  
  
train.to\_csv('/Users/iris\_train.csv', index = False)  
test.to\_csv('/Users/iris\_test.csv', index = False)

[train\_test\_split()](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)

## 5.3 Fit a logistic regression model.

# Notice we are using a new data set that needs to be read into the   
# environment   
tips = pd.read\_csv('/Users/tips.csv')  
  
# The following code is used to determine if the individual left more   
# than a 15% tip   
tips["fifteen"] = 0.15 \* tips["total\_bill"]  
tips["greater15"] = np.where(tips["tip"] > tips["fifteen"], 1, 0)  
  
import statsmodels.api as sm  
  
# Notice the syntax of greater15 as a function of total\_bill   
logreg = sm.formula.glm("greater15 ~ total\_bill",   
 family=sm.families.Binomial(),   
 data=tips).fit()  
print(logreg.summary())

## /Users/elaineek/anaconda/lib/python3.5/site-packages/statsmodels/compat/pandas.py:56: FutureWarning: The pandas.core.datetools module is deprecated and will be removed in a future version. Please use the pandas.tseries module instead.  
## from pandas.core import datetools  
## Generalized Linear Model Regression Results   
## ==============================================================================  
## Dep. Variable: greater15 No. Observations: 244  
## Model: GLM Df Residuals: 242  
## Model Family: Binomial Df Model: 1  
## Link Function: logit Scale: 1.0  
## Method: IRLS Log-Likelihood: -156.87  
## Date: Mon, 26 Jun 2017 Deviance: 313.74  
## Time: 14:05:20 Pearson chi2: 247.  
## No. Iterations: 4   
## ==============================================================================  
## coef std err z P>|z| [0.025 0.975]  
## ------------------------------------------------------------------------------  
## Intercept 1.6477 0.355 4.646 0.000 0.953 2.343  
## total\_bill -0.0725 0.017 -4.319 0.000 -0.105 -0.040  
## ==============================================================================

A logistic regression model can be implemented using [sklearn](#SKLEARN), however [statsmodels.api](http://www.statsmodels.org/stable/glm.html#technical-documentation) provides a helpful summary about the model, so it is preferable for this example.

## 5.4 Fit a linear regression model.

# Fit a linear regression model of tip by total\_bill on the training data   
from sklearn.linear\_model import LinearRegression  
  
# If your data has one feature, you need to reshape the 1D array  
linreg = LinearRegression()  
linreg.fit(tips["total\_bill"].values.reshape(-1,1), tips["tip"])  
print(linreg.coef\_)  
print(linreg.intercept\_)

## [ 0.10502452]  
## 0.920269613555

[LinearRegression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)

# 6 Supervised Machine Learning

## 6.1 Fit a logistic regression model on training data and assess against testing data.

### a) Fit a logistic regression model on training data.

# Notice we are using new data sets that need to be read into the environment   
train = pd.read\_csv('/Users/tips\_train.csv')  
test = pd.read\_csv('/Users/tips\_test.csv')  
  
train["fifteen"] = 0.15 \* train["total\_bill"]  
train["greater15"] = np.where(train["tip"] > train["fifteen"], 1, 0)  
test["fifteen"] = 0.15 \* test["total\_bill"]  
test["greater15"] = np.where(test["tip"] > test["fifteen"], 1, 0)  
  
logreg = sm.formula.glm("greater15 ~ total\_bill",   
 family=sm.families.Binomial(),   
 data=train).fit()  
print(logreg.summary())

## /Users/elaineek/anaconda/lib/python3.5/site-packages/statsmodels/compat/pandas.py:56: FutureWarning: The pandas.core.datetools module is deprecated and will be removed in a future version. Please use the pandas.tseries module instead.  
## from pandas.core import datetools  
## Generalized Linear Model Regression Results   
## ==============================================================================  
## Dep. Variable: greater15 No. Observations: 195  
## Model: GLM Df Residuals: 193  
## Model Family: Binomial Df Model: 1  
## Link Function: logit Scale: 1.0  
## Method: IRLS Log-Likelihood: -125.29  
## Date: Mon, 26 Jun 2017 Deviance: 250.58  
## Time: 14:05:22 Pearson chi2: 197.  
## No. Iterations: 4   
## ==============================================================================  
## coef std err z P>|z| [0.025 0.975]  
## ------------------------------------------------------------------------------  
## Intercept 1.6461 0.395 4.172 0.000 0.873 2.420  
## total\_bill -0.0706 0.018 -3.820 0.000 -0.107 -0.034  
## ==============================================================================

### b) Assess the model against the testing data.

# Predict on testing data  
predictions = logreg.predict(test["total\_bill"])  
predY = np.where(predictions < 0.5, 0, 1)  
  
# If the prediction probability is less than 0.5, classify this as a 0  
# and otherwise classify as a 1. This isn't the best method -- a better   
# method would be randomly assigning a 0 or 1 when a probability of 0.5   
# occurrs, but this insures that results are consistent   
  
# Determine how many were correctly classified   
Results = np.where(predY == test["greater15"], "Correct", "Wrong")  
print(pd.crosstab(index=Results, columns="count"))

## /Users/elaineek/anaconda/lib/python3.5/site-packages/statsmodels/compat/pandas.py:56: FutureWarning: The pandas.core.datetools module is deprecated and will be removed in a future version. Please use the pandas.tseries module instead.  
## from pandas.core import datetools  
## col\_0 count  
## row\_0   
## Correct 34  
## Wrong 15

A logistic regression model can be implemented using [sklearn](#SKLEARN), however [statsmodels.api](http://www.statsmodels.org/stable/glm.html#technical-documentation) provides a helpful summary about the model, so it is preferable for this example.

## 6.2 Fit a linear regression model on training data and assess against testing data.

### a) Fit a linear regression model on training data.

# Notice we are using new data sets that need to be read into the environment   
train = pd.read\_csv('/Users/boston\_train.csv')  
test = pd.read\_csv('/Users/boston\_test.csv')  
  
# Fit a linear regression model  
linreg = LinearRegression()  
linreg.fit(train.drop(["Target"], axis = 1), train["Target"])  
print(linreg.coef\_)  
print(linreg.intercept\_)

## [ -8.56336900e-02 4.60343577e-02 3.64131905e-02 3.24796064e+00  
## -1.48729382e+01 3.57686873e+00 -8.70316831e-03 -1.36890461e+00  
## 3.13120107e-01 -1.28815611e-02 -9.76900124e-01 1.13257346e-02  
## -5.26715028e-01]  
## 36.1081957809

### b) Assess the model against the testing data.

# Predict on testing data  
prediction = pd.DataFrame()  
prediction["predY"] = linreg.predict(test.drop(["Target"], axis = 1))  
  
# Determine mean squared error  
prediction["sq\_diff"] = (prediction["predY"] - test["Target"])\*\*2  
print(np.mean(prediction["sq\_diff"]))

## 17.771307958891672

[LinearRegression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)

## 6.3 Fit a decision tree model on training data and assess against testing data.

### a) Fit a decision tree classification model.

#### i) Fit a decision tree classification model on training data and determine variable importance.

# Notice we are using new data sets that need to be read into the environment   
train = pd.read\_csv('/Users/breastcancer\_train.csv')  
test = pd.read\_csv('/Users/breastcancer\_test.csv')  
  
from sklearn.tree import DecisionTreeClassifier  
  
# random\_state is used to specify a seed for a random integer so that the   
# results are reproducible  
treeMod = DecisionTreeClassifier(random\_state=29)  
treeMod.fit(train.drop(["Target"], axis = 1), train["Target"])  
  
# Determine variable importance  
var\_import = treeMod.feature\_importances\_  
var\_import = pd.DataFrame(var\_import)  
var\_import = var\_import.rename(columns = {0:'Importance'})  
var\_import = var\_import.sort\_values(by="Importance", kind = "mergesort",   
 ascending = False)  
print(var\_import.head())

## Importance  
## 23 0.692681  
## 27 0.158395  
## 21 0.044384  
## 11 0.029572  
## 24 0.020485

#### ii) Assess the model against the testing data.

# Prediction on testing data  
predY = treeMod.predict(test.drop(["Target"], axis = 1))  
  
# Determine how many were correctly classified  
Results = np.where(test["Target"] == predY, "Correct", "Wrong")  
print(pd.crosstab(index=Results, columns="count"))

## col\_0 count  
## row\_0   
## Correct 161  
## Wrong 10

[DecisionTreeClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)

### b) Fit a decision tree regression model.

#### i) Fit a decision tree regression model on training data and determine variable importance.

train = pd.read\_csv('/Users/boston\_train.csv')  
test = pd.read\_csv('/Users/boston\_test.csv')  
  
from sklearn.tree import DecisionTreeRegressor  
  
treeMod = DecisionTreeRegressor(random\_state=29)  
treeMod.fit(train.drop(["Target"], axis = 1), train["Target"])  
  
# Determine variable importance  
var\_import = treeMod.feature\_importances\_  
var\_import = pd.DataFrame(var\_import)  
var\_import = var\_import.rename(columns = {0:'Importance'})  
var\_import = var\_import.sort\_values(by="Importance", kind = "mergesort",   
 ascending = False)  
print(var\_import.head())

## Importance  
## 5 0.573257  
## 12 0.203677  
## 7 0.103939  
## 4 0.041467  
## 0 0.033798

#### ii) Assess the model against the testing data.

# Prediction on testing data  
prediction = pd.DataFrame()  
prediction["predY"] = treeMod.predict(test.drop(["Target"], axis = 1))  
  
# Determine mean squared error  
prediction["sq\_diff"] = (prediction["predY"] - test["Target"])\*\*2  
print(np.mean(prediction["sq\_diff"]))

## 23.866842105263157

## 6.4 Fit a random forest model on training data and assess against testing data.

### a) Fit a random forest classification model.

#### i) Fit a random forest classification model on training data and determine variable importance.

train = pd.read\_csv('/Users/breastcancer\_train.csv')  
test = pd.read\_csv('/Users/breastcancer\_test.csv')  
  
from sklearn.ensemble import RandomForestClassifier  
  
rfMod = RandomForestClassifier(random\_state=29)  
rfMod.fit(train.drop(["Target"], axis = 1), train["Target"])  
  
# Determine variable importance  
var\_import = rfMod.feature\_importances\_  
var\_import = pd.DataFrame(var\_import)  
var\_import = var\_import.rename(columns = {0:'Importance'})  
var\_import = var\_import.sort\_values(by="Importance", kind = "mergesort",   
 ascending = False)  
print(var\_import.head())

## Importance  
## 27 0.271730  
## 13 0.120096  
## 23 0.101971  
## 20 0.076891  
## 6 0.066836

#### ii) Assess the model against the testing data.

# Prediction on testing data  
predY = rfMod.predict(test.drop(["Target"], axis = 1))  
  
# Determine how many were correctly classified  
Results = np.where(test["Target"] == predY, "Correct", "Wrong")  
print(pd.crosstab(index=Results, columns="count"))

## col\_0 count  
## row\_0   
## Correct 165  
## Wrong 6

[RandomForestClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)

### b) Fit a random forest regression model.

#### i) Fit a random forest regression model on training data and determine variable importance.

train = pd.read\_csv('/Users/boston\_train.csv')  
test = pd.read\_csv('/Users/boston\_test.csv')  
  
from sklearn.ensemble import RandomForestRegressor  
  
rfMod = RandomForestRegressor(random\_state=29)  
rfMod.fit(train.drop(["Target"], axis = 1), train["Target"])  
  
# Determine variable importance  
var\_import = rfMod.feature\_importances\_  
var\_import = pd.DataFrame(var\_import)  
var\_import = var\_import.rename(columns = {0:'Importance'})  
var\_import = var\_import.sort\_values(by="Importance", kind = "mergesort",   
 ascending = False)  
print(var\_import.head())

## Importance  
## 5 0.412012  
## 12 0.392795  
## 7 0.079462  
## 0 0.041911  
## 9 0.016374

#### ii) Assess the model against the testing data.

# Prediction on testing data  
prediction = pd.DataFrame()  
prediction["predY"] = rfMod.predict(test.drop(["Target"], axis = 1))  
  
# Determine mean squared error  
prediction["sq\_diff"] = (test["Target"] - prediction["predY"])\*\*2  
print(prediction["sq\_diff"].mean())

## 13.25032631578948

[RandomForestRegressor](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html)

## 6.5 Fit a gradient boosting model on training data and assess against testing data.

### a) Fit a gradient boosting classification model.

#### i) Fit a gradient boosting classification model on training data and determine variable importance.

train = pd.read\_csv('/Users/breastcancer\_train.csv')  
test = pd.read\_csv('/Users/breastcancer\_test.csv')  
  
from sklearn.ensemble import GradientBoostingClassifier  
  
# n\_estimators = total number of trees to fit which is analogous to the   
# number of iterations  
# learning\_rate = shrinkage or step-size reduction, where a lower   
# learning rate requires more iterations  
gbMod = GradientBoostingClassifier(random\_state = 29, learning\_rate = .01,   
 n\_estimators = 2500)  
gbMod.fit(train.drop(["Target"], axis = 1), train["Target"])  
  
# Determine variable importance  
var\_import = gbMod.feature\_importances\_  
var\_import = pd.DataFrame(var\_import)  
var\_import = var\_import.rename(columns = {0:'Importance'})  
var\_import = var\_import.sort\_values(by="Importance", kind = "mergesort",   
 ascending = False)  
print(var\_import.head())

## Importance  
## 23 0.099054  
## 27 0.088744  
## 7 0.062735  
## 21 0.043547  
## 14 0.042328

#### ii) Assess the model against the testing data.

# Prediction on testing data  
predY = gbMod.predict(test.drop(["Target"], axis = 1))  
  
# Determine how many were correctly classified  
Results = np.where(test["Target"] == predY, "Correct", "Wrong")  
print(pd.crosstab(index=Results, columns="count"))

## col\_0 count  
## row\_0   
## Correct 164  
## Wrong 7

[GradientBoostingClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html)

### b) Fit a gradient boosting regression model.

#### i) Fit a gradient boosting regression model on training data and determine variable importance.

train = pd.read\_csv('/Users/boston\_train.csv')  
test = pd.read\_csv('/Users/boston\_test.csv')  
  
from sklearn.ensemble import GradientBoostingRegressor  
  
gbMod = GradientBoostingRegressor(random\_state = 29, learning\_rate = .01,   
 n\_estimators = 2500)  
gbMod.fit(train.drop(["Target"], axis = 1), train["Target"])  
  
# Determine variable importance  
var\_import = gbMod.feature\_importances\_  
var\_import = pd.DataFrame(var\_import)  
var\_import = var\_import.rename(columns = {0:'Importance'})  
var\_import = var\_import.sort\_values(by="Importance", kind = "mergesort",   
 ascending = False)  
print(var\_import.head())

## Importance  
## 5 0.166179  
## 12 0.154570  
## 0 0.127526  
## 11 0.124045  
## 6 0.115200

#### ii) Assess the model against the testing data.

# Prediction on testing data  
prediction = pd.DataFrame()  
prediction["predY"] = gbMod.predict(test.drop(["Target"], axis = 1))  
  
# Determine mean squared error  
prediction["sq\_diff"] = (test["Target"] - prediction["predY"])\*\*2

## 9.416022842108923

[GradientBoostingRegressor](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html)

## 6.6 Fit an extreme gradient boosting model on training data and assess against testing data.

### a) Fit an extreme gradient boosting classification model on training data and assess against testing data.

#### i) Fit an extreme gradient boosting classification model on training data.

train = pd.read\_csv('/Users/breastcancer\_train.csv')  
test = pd.read\_csv('/Users/breastcancer\_test.csv')  
  
from xgboost import XGBClassifier  
  
# Fit XGBClassifier model on training data  
xgbMod = XGBClassifier(seed = 29, learning\_rate = 0.01,   
 n\_estimators = 2500)  
xgbMod.fit(train.drop(["Target"], axis = 1), train["Target"])

## /Users/elaineek/anaconda/lib/python3.5/site-packages/sklearn/cross\_validation.py:44: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model\_selection module into which all the refactored classes and functions are moved. Also note that the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.  
## "This module will be removed in 0.20.", DeprecationWarning)

#### ii) Assess the model against the testing data.

# Prediction on testing data  
predY = xgbMod.predict(test.drop(["Target"], axis = 1))  
  
# Determine how many were correctly classified  
Results = np.where(test["Target"] == predY, "Correct", "Wrong")  
print(pd.crosstab(index=Results, columns="count"))

## /Users/elaineek/anaconda/lib/python3.5/site-packages/sklearn/cross\_validation.py:44: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model\_selection module into which all the refactored classes and functions are moved. Also note that the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.  
## "This module will be removed in 0.20.", DeprecationWarning)  
## col\_0 count  
## row\_0   
## Correct 165  
## Wrong 6

### b) Fit an extreme gradient boosting regression model on training data and assess against testing data.

#### i) Fit an extreme gradient boosting regression model on training data.

train = pd.read\_csv('/Users/boston\_train.csv')  
test = pd.read\_csv('/Users/boston\_test.csv')  
  
from xgboost import XGBRegressor  
  
# Fit XGBRegressor model on training data  
xgbMod = XGBRegressor(seed = 29, learning\_rate = 0.01,   
 n\_estimators = 2500)  
xgbMod.fit(train.drop(["Target"], axis = 1), train["Target"])

## /Users/elaineek/anaconda/lib/python3.5/site-packages/sklearn/cross\_validation.py:44: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model\_selection module into which all the refactored classes and functions are moved. Also note that the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.  
## "This module will be removed in 0.20.", DeprecationWarning)

#### ii) Assess the model against the testing data.

# Prediction on testing data  
prediction = pd.DataFrame()  
prediction["predY"] = xgbMod.predict(test.drop(["Target"], axis = 1))  
  
# Determine mean squared error  
prediction["sq\_diff"] = (test["Target"] - prediction["predY"])\*\*2  
print(prediction["sq\_diff"].mean())

## /Users/elaineek/anaconda/lib/python3.5/site-packages/sklearn/cross\_validation.py:44: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model\_selection module into which all the refactored classes and functions are moved. Also note that the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.  
## "This module will be removed in 0.20.", DeprecationWarning)  
## 9.658108024646909

## 6.7 Fit a support vector model on training data and assess against testing data.

### a) Fit a support vector classification model.

#### i) Fit a support vector classification model on training data.

train = pd.read\_csv('/Users/breastcancer\_train.csv')  
test = pd.read\_csv('/Users/breastcancer\_test.csv')  
  
# First we should scale the data, since R does  
from sklearn.preprocessing import StandardScaler  
  
train\_features = train.drop(["Target"], axis = 1)  
scaler = StandardScaler().fit(np.array(train\_features))  
train\_scaled = scaler.transform(np.array(train\_features))  
train\_scaled = pd.DataFrame(train\_scaled)  
train\_scaled["Target"] = train["Target"]  
  
test\_features = test.drop(["Target"], axis = 1)  
scaler = StandardScaler().fit(np.array(test\_features))  
test\_scaled = scaler.transform(np.array(test\_features))  
test\_scaled = pd.DataFrame(test\_scaled)  
test\_scaled["Target"] = test["Target"]  
  
# Fit a support vector classification model  
from sklearn.svm import SVC  
svMod = SVC(random\_state = 29, kernel = 'linear')  
svMod.fit(train\_scaled.drop(["Target"], axis = 1),

#### ii) Assess the model against the testing data.

# Prediction on testing data  
prediction = svMod.predict(test\_scaled.drop(["Target"], axis = 1))  
  
# Determine how many were correctly classified  
Results = np.where(test\_scaled["Target"] == prediction, "Correct", "Wrong")  
print(pd.crosstab(index=Results, columns="count"))

## col\_0 count  
## row\_0   
## Correct 164  
## Wrong 7

[SVC](http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html)

### b) Fit a support vector regression model.

#### i) Fit a support vector regression model on training data.

train = pd.read\_csv('/Users/boston\_train.csv')  
test = pd.read\_csv('/Users/boston\_test.csv')  
  
from sklearn.preprocessing import StandardScaler   
  
scaler = StandardScaler().fit(np.array(train))   
train\_scaled = scaler.transform(np.array(train))   
train\_scaled = pd.DataFrame(train\_scaled)   
  
scaler = StandardScaler().fit(np.array(test))   
test\_scaled = scaler.transform(np.array(test))   
test\_scaled = pd.DataFrame(test\_scaled)   
  
# Fit a support vector regression model  
from sklearn.svm import SVR  
svMod = SVR()  
svMod.fit(train\_scaled.drop([13], axis = 1), train\_scaled[13])

#### ii) Assess the model against the testing data.

# Prediction on testing data  
prediction = pd.DataFrame()  
prediction["predY"] = svMod.predict(test\_scaled.drop([13], axis = 1))  
  
# Determine mean squared error  
prediction["sq\_diff"] = (test\_scaled[13] - prediction["predY"])\*\*2  
print(prediction["sq\_diff"].mean())

## 0.15397129809389054

[SVR](http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html)

## 6.8 Fit a neural network model on training data and assess against testing data.

### a) Fit a neural network classification model.

#### i) Fit a neural network classification model on training data.

# Notice we are using new data sets that need to be read into the environment   
train = pd.read\_csv('/Users/digits\_train.csv')  
test = pd.read\_csv('/Users/digits\_test.csv')  
  
# Fit a neural network classification model on training data  
from sklearn.neural\_network import MLPClassifier  
nnMod = MLPClassifier(max\_iter = 200, hidden\_layer\_sizes=(100,),   
 random\_state = 29)

#### ii) Assess the model against the testing data.

# Prediction on testing data  
predY = nnMod.predict(test.drop(["Target"], axis = 1))  
  
# Determine how many were correctly classified  
  
from sklearn.metrics import confusion\_matrix  
  
print(confusion\_matrix(test["Target"], predY))

## [[57 0 0 0 1 0 0 0 0 0]  
## [ 0 57 0 0 0 0 0 0 1 0]  
## [ 0 0 58 0 0 0 0 0 0 0]  
## [ 0 0 0 58 0 1 0 0 0 0]  
## [ 0 0 0 0 52 0 1 0 1 0]  
## [ 0 0 0 0 1 56 0 1 1 0]  
## [ 0 0 0 0 0 0 41 0 0 0]  
## [ 0 0 0 0 1 0 0 49 0 1]  
## [ 0 1 0 1 0 0 0 0 43 0]  
## [ 0 1 0 0 0 1 0 0 2 53]]

[MLPClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier) | [confusion\_matrix()](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html)

### b) Fit a neural network regression model.

#### i) Fit a neural network regression model on training data.

train = pd.read\_csv('/Users/boston\_train.csv')  
test = pd.read\_csv('/Users/boston\_test.csv')  
  
# Scale input data  
from sklearn.preprocessing import StandardScaler  
  
train\_features = train.drop(["Target"], axis = 1)  
scaler = StandardScaler().fit(np.array(train\_features))  
train\_scaled = scaler.transform(np.array(train\_features))  
train\_scaled = pd.DataFrame(train\_scaled)  
  
test\_features = test.drop(["Target"], axis = 1)  
scaler = StandardScaler().fit(np.array(test\_features))  
test\_scaled = scaler.transform(np.array(test\_features))  
test\_scaled = pd.DataFrame(test\_scaled)  
  
# Fit neural network regression model, dividing target by 50 for scaling  
from sklearn.neural\_network import MLPRegressor  
nnMod = MLPRegressor(max\_iter = 250, random\_state = 29, solver = 'lbfgs')  
nnMod = nnMod.fit(train\_scaled, train["Target"] / 50)

#### ii) Assess the model against testing data.

# Prediction on testing data, remembering to multiply by 50  
prediction = pd.DataFrame()  
prediction["predY"] = nnMod.predict(test\_scaled)\*50  
  
# Determine mean squared error  
prediction["sq\_diff"] = (test["Target"] - prediction["predY"])\*\*2  
print(prediction["sq\_diff"].mean())

## 17.532969200412914

[MLPRegressor](http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html#sklearn.neural_network.MLPRegressor)

# 7 Unsupervised Machine Learning

## 7.1 KMeans Clustering

iris = pd.read\_csv('/Users/iris.csv')  
iris["Species"] = np.where(iris["Target"] == 0, "Setosa",   
 np.where(iris["Target"] == 1, "Versicolor",   
 "Virginica"))  
features = pd.concat([iris["PetalLength"], iris["PetalWidth"],   
 iris["SepalLength"], iris["SepalWidth"]], axis = 1)  
  
from sklearn.cluster import KMeans  
kmeans = KMeans(n\_clusters = 3, random\_state = 29).fit(features)  
  
print(pd.crosstab(index = iris["Species"], columns = kmeans.labels\_))

## col\_0 0 1 2  
## Species   
## Setosa 0 50 0  
## Versicolor 2 0 48  
## Virginica 36 0 14

[KMeans](http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans)

## 7.2 Spectral Clustering

from sklearn.cluster import SpectralClustering  
  
spectral = SpectralClustering(n\_clusters = 3,   
 random\_state = 29).fit(features)  
  
print(pd.crosstab(index = iris["Species"], columns = spectral.labels\_))

## col\_0 0 1 2  
## Species   
## Setosa 0 50 0  
## Versicolor 48 0 2  
## Virginica 13 0 37

[SpectralClustering](http://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html#sklearn.cluster.SpectralClustering)

## 7.3 Ward Hierarchical Clustering

from sklearn.cluster import AgglomerativeClustering  
  
aggl = AgglomerativeClustering(n\_clusters = 3).fit(features)  
  
print(pd.crosstab(index = iris["Species"], columns = aggl.labels\_))

## col\_0 0 1 2  
## Species   
## Setosa 0 50 0  
## Versicolor 49 0 1  
## Virginica 15 0 35

[AgglomerativeClustering](http://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html#sklearn.cluster.AgglomerativeClustering)

## 7.4 DBSCAN

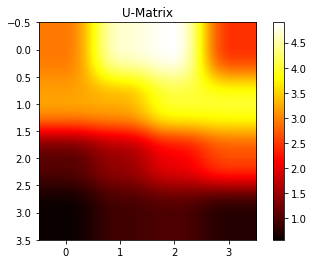
from sklearn.cluster import DBSCAN  
  
dbscan = DBSCAN().fit(features)  
  
print(pd.crosstab(index = iris["Species"], columns = dbscan.labels\_))

## col\_0 -1 0 1  
## Species   
## Setosa 1 49 0  
## Versicolor 6 0 44  
## Virginica 10 0 40

[DBCAN](http://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html#sklearn.cluster.DBSCAN)

## 7.5 Self-organizing map

from pyclustering.nnet import som  
  
sm = som.som(4,4)  
  
sm.train(features.as\_matrix(), 100)  
  
sm.show\_distance\_matrix()

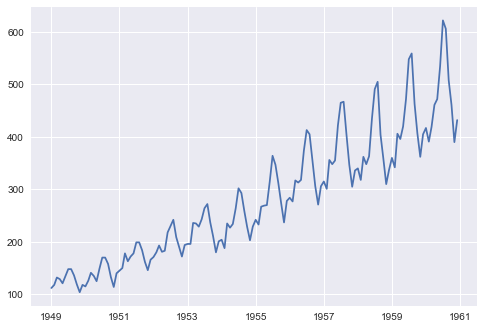
Output: 

# 8 Forecasting

## 8.1 Fit an ARIMA model to a timeseries.

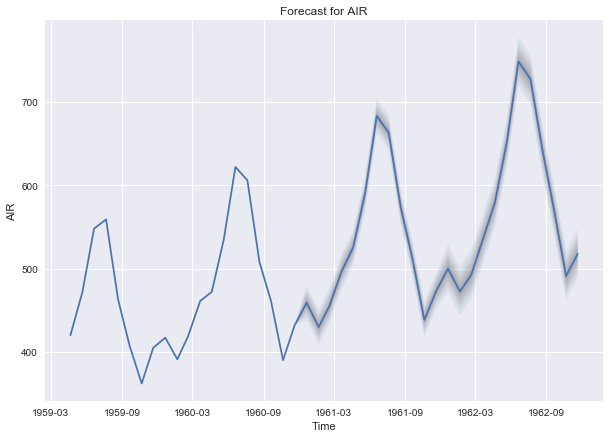
### a) Plot the timeseries.

air = pd.read\_csv('/Users/air.csv')  
air["DATE"] = pd.to\_datetime(air["DATE"], infer\_datetime\_format = True)  
air.index = air["DATE"].values  
plt.plot(air.index, air["AIR"])  
plt.show()

Output: 

### b) Fit an ARIMA model, predict 2 years (24 months) out and plot predictions.

import pyflux as pf  
  
# ar = 12 is necessary to indicate the seasonality of data  
model = pf.ARIMA(data = air, ar = 12, ma = 1, integ = 0, target = 'AIR', family = pf.Normal())  
x = model.fit("MLE")  
model.plot\_predict(h = 24)

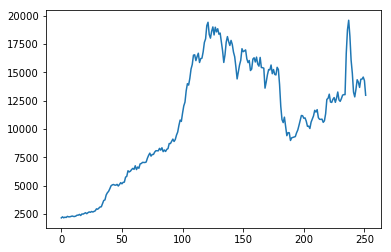
Output: 

[PyFlux ARIMA models](http://www.pyflux.com/docs/arima.html#class-description)

## 8.2 Fit a Simple Exponential Smoothing model to a timeseries.

### a) Plot the timeseries.

usecon = pd.read\_csv('/Users/usecon.csv')  
  
petrol = usecon["PETROL"]  
  
plt.plot(petrol)  
plt.show()

Output: 

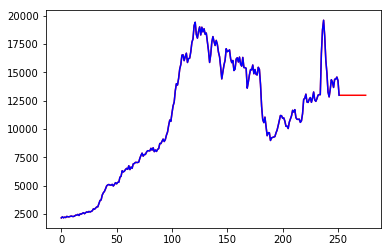
### b) Fit a Simple Exponential Smoothing model, predict 2 years (24 months) out and plot predictions.

Currently, there is not a good package in Python to fit a simple exponential smoothing model. The formula for fitting an exponential smoothing model is not difficult, so we can do it by creating our own functions in Python.

The simplest form of exponential smoothing is given by, where :

Therefore, we can implement a simple exponential smoothing model as follows:

def simple\_exp\_smoothing(data, alpha, n\_preds):  
 # Eq1:  
 output = [data[0]]  
 # Smooth given data plus we want to predict 24 units   
 # past the end  
 for i in range(1, len(data) + n\_preds):  
 # Eq2:  
 if (i < len(data)):  
 output.append(alpha \* data[i] + (1 - alpha) \* data[i-1])  
 else:  
 output.append(alpha \* output[i-1] + (1 - alpha) \* output[i-2])  
 return output  
   
pred = simple\_exp\_smoothing(petrol, 0.9999, 24)  
  
plt.plot(pd.DataFrame(pred), color = "red")  
plt.plot(petrol, color = "blue")  
plt.show()Petro

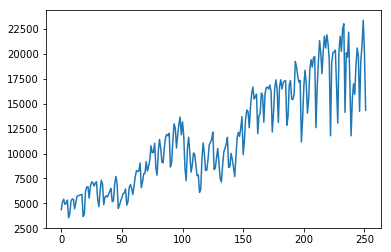
Output: 

[Basis for code](https://grisha.org/blog/2016/01/29/triple-exponential-smoothing-forecasting/)

## 8.3 Fit a Holt-Winters model to a timeseries.

### a) Plot the timeseries.

vehicle = usecon["VEHICLE"]  
  
plt.plot(vehicle)  
plt.show()

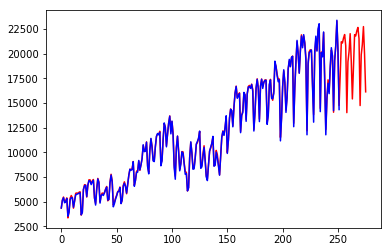
Output: 

### b) Fit a Holt-Winters additive model, predict 2 years (24 months) out and plot predictions.

Currently, there is not a good package in Python to fit a Holt-Winters additive model. The formula for fitting a Holt-Winters additive model is not difficult, so we can do it by creating our own functions in Python.

The following is an implementation of the Holt-Winters additive model given at [triple exponential smoothing code](https://grisha.org/blog/2016/01/29/triple-exponential-smoothing-forecasting/).

def initial\_trend(series, slen):  
 sum = 0.0  
 for i in range(slen):  
 sum += float(series[i+slen] - series[i]) / slen  
 return sum / slen  
def initial\_seasonal\_components(series, slen):  
 seasonals = {}  
 season\_averages = []  
 n\_seasons = int(len(series)/slen)  
 # compute season averages  
 for j in range(n\_seasons):  
 season\_averages.append(sum(series[slen\*j:slen\*j+slen])/float(slen))  
 # compute initial values  
 for i in range(slen):  
 sum\_of\_vals\_over\_avg = 0.0  
 for j in range(n\_seasons):  
 sum\_of\_vals\_over\_avg += series[slen\*j+i]-season\_averages[j]  
 seasonals[i] = sum\_of\_vals\_over\_avg/n\_seasons  
 return seasonals  
def triple\_exponential\_smoothing\_add(series, slen, alpha, beta, gamma, n\_preds):  
 result = []  
 seasonals = initial\_seasonal\_components(series, slen)  
 for i in range(len(series)+n\_preds):  
 if i == 0: # initial values  
 smooth = series[0]  
 trend = initial\_trend(series, slen)  
 result.append(series[0])  
 continue  
 if i >= len(series): # we are forecasting  
 m = i - len(series) + 1  
 result.append((smooth + m\*trend) + seasonals[i%slen])  
 else:  
 val = series[i]  
 last\_smooth, smooth = smooth, alpha\*(val-seasonals[i%slen]) + (1-alpha)\*(smooth+trend)  
 trend = beta \* (smooth-last\_smooth) + (1-beta)\*trend  
 seasonals[i%slen] = gamma\*(val-smooth) + (1-gamma)\*seasonals[i%slen]  
 result.append(smooth+trend+seasonals[i%slen])  
 return result  
   
add\_preds = triple\_exponential\_smoothing\_add(vehicles, 12, 0.5731265, 0, 0.7230956, 24)  
  
plt.plot(pd.DataFrame(add\_preds), color = "red")  
plt.plot(vehicles, color = "blue")  
plt.show()

Output: 

# 9 Model Evaluation & Selection

## 9.1 Evaluate the accuracy of regression models.

### a) Evaluation on training data.

train = pd.read\_csv('/Users/boston\_train.csv')  
test = pd.read\_csv('/Users/boston\_test.csv')  
  
# Random Forest Regression Model  
from sklearn.ensemble import RandomForestRegressor  
rfMod = RandomForestRegressor(random\_state=29)  
rfMod.fit(train.drop(["Target"], axis = 1), train["Target"])  
  
# Evaluation on training data  
predY = rfMod.predict(train.drop(["Target"], axis = 1))  
  
# Determine coefficient of determination score  
from sklearn.metrics import r2\_score  
r2\_rf = r2\_score(train["Target"], predY)  
print("Random forest regression model r^2 score (coefficient of determination): %f" % r2\_rf)

## Random forest regression model r^2 score (coefficient of determination): 0.975233

### b) Evaluation on testing data.

# Random Forest Regression Model (rfMod)  
  
# Evaluation on testing data  
predY = rfMod.predict(test.drop(["Target"], axis = 1))  
  
# Determine coefficient of determination score  
r2\_rf = r2\_score(test["Target"], predY)  
print("Random forest regression model r^2 score (coefficient of determination): %f" % r2\_rf)

## Random forest regression model r^2 score (coefficient of determination): 0.833687

The sklearn metric [r2\_score](http://scikit-learn.org/stable/modules/model_evaluation.html#r2-score-the-coefficient-of-determination) is only one option for assessing a regression model. Please go [here](http://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics) for more information about other sklearn regression metrics.

## 9.2 Evaluate the accuracy of classification models.

### a) Evaluation on training data.

train = pd.read\_csv('/Users/digits\_train.csv')  
test = pd.read\_csv('/Users/digits\_test.csv')  
  
# Random Forest Classification Model  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import accuracy\_score  
rfMod = RandomForestClassifier(random\_state=29)  
rfMod.fit(train.drop(["Target"], axis = 1), train["Target"])  
  
# Evaluation on training data  
predY = rfMod.predict(train.drop(["Target"], axis = 1))  
  
# Determine accuracy score  
accuracy\_rf = accuracy\_score(train["Target"], predY)  
print("Random forest model accuracy: %f" % accuracy\_rf)

## Random forest model accuracy: 1.000000

### b) Evaluation on testing data.

# Random Forest Classification Model (rfMod)  
  
# Evaluation on testing data  
predY = rfMod.predict(test.drop(["Target"], axis = 1))  
  
# Determine accuracy score  
accuracy\_rf = accuracy\_score(test["Target"], predY)  
print("Random forest model accuracy: %f" % accuracy\_rf)

## Random forest model accuracy: 0.940741

Note: The sklearn metric [accuracy\_score](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html) is only one option for assessing a classification model. Please go [here](http://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics) for more information about other sklearn classification metrics.

## 9.3 Evaluation with cross validation.

### a) KFold

# Notice we are using a new data set that need to be read into the   
# environment   
breastcancer = pd.read\_csv('/Users/breastcancer.csv')  
  
from sklearn import model\_selection  
from sklearn.ensemble import RandomForestClassifier  
  
X = breastcancer.drop(["Target"], axis = 1)  
Y = breastcancer["Target"]  
  
kfold = model\_selection.KFold(n\_splits = 5, random\_state = 29)  
rfMod = RandomForestClassifier(random\_state = 29)  
results = model\_selection.cross\_val\_score(rfMod, X, Y, cv = kfold)  
  
print("Accuracy: %.2f%% +/- %.2f%%" % (results.mean()\*100,   
 results.std()\*100))

## Accuracy: 94.38% +/- 2.39%

### b) ShuffleSplit

shuffle = model\_selection.ShuffleSplit(n\_splits = 5, random\_state = 29)  
rfMod = RandomForestClassifier(random\_state = 29)  
results = model\_selection.cross\_val\_score(rfMod, X, Y, cv = shuffle)  
  
print("Accuracy: %.2f%% +/- %.2f%%" % (results.mean()\*100,   
 results.std()\*100))

## Accuracy: 95.09% +/- 0.70%

# 10 Text Analytics

# 11 Deep Learning

# Appendix

## 1 Built-in Python Data Types

* [Boolean](https://docs.python.org/2/library/stdtypes.html#boolean-values)

#### Numeric types

* [int](https://docs.python.org/3/library/functions.html#int)
* [long](https://docs.python.org/2/library/functions.html#long)
* [float](https://docs.python.org/2/library/functions.html#float)
* [complex](https://docs.python.org/2/library/functions.html#complex)

#### Sequences

* [str](#str)
* [bytes](#BYTE)
* [byte array](#BYTE)
* [list](#LIST)
* [tuple](#LIST)

#### Sets

* [set](#SET)
* [frozen set](#SET)

#### Mapping:

* [dictionary](#dict)

## 2 Python Plotting Packages

#### [Bokeh](http://bokeh.pydata.org/en/latest/)

A Python package which is useful for interactive visualizations and is optimized for web browser presentations.

#### [PyPlot](https://matplotlib.org/api/pyplot_api.html)

A Python package which is useful data plotting and visualization.

#### [Seaborn](https://seaborn.pydata.org/)

A Python package which is useful for data plotting and visualization. In particular, Seaborn includes tools for drawing attractive statistical graphics.

## 3 Python packages used in this tutorial

#### [pandas](http://pandas.pydata.org/)

Working with data structures and performing data analysis

#### [NumPy](http://www.numpy.org/)

Scientific and mathematical computing

#### [re](https://docs.python.org/2/library/re.html" \l "module-re)

Regular expressions

#### [Decimal](https://docs.python.org/2/library/decimal.html)

Tools for decimal [floating point](#float) arithmetic

#### [sklearn](http://scikit-learn.org/stable/)

scikit-learn, or more commonly known as sklearn, is useful for basic and advanced data mining, machine learning, and data analysis. sklearn includes tools for classification, regression, clustering, dimensionality reduction, model selection, and data pre-processing.

#### [statsmodels.api](http://www.statsmodels.org/stable/index.html)

Tools for the estimation of many different statistical models

#### [xgboost](http://xgboost.readthedocs.io/en/latest/python/python_intro.html)

Extreme gradient boosting models

#### [pyclustering](http://pythonhosted.org/pyclustering/)

Tools for clustering input data

#### [pyflux](http://www.pyflux.com/docs/)

Tools for time series analysis and prediction

# Alphabetical Index

## [Array](https://docs.scipy.org/doc/numpy/reference/generated/numpy.array.html)

A NumPy array is a data type implemented by the [NumPy](#NUMPY) package in which the elements of the array are all of the same type. Please see the following example of array creation and access:

import numpy as np  
my\_array = np.array([1, 2, 3, 4])  
print(my\_array)

## [1 2 3 4]

print(my\_array[3])

## 4

## Bytes & Byte arrays

A [byte](https://docs.python.org/3.1/library/functions.html#bytes) is a sequence of integers which is immutable, whereas a [byte array](https://docs.python.org/3.1/library/functions.html#bytearray) is its mutable counterpart.

## [Data Frame](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html)

A two-dimensional tabular structure with labeled axes (rows and columns), where data observations are represented by rows and data variables are represented by columns.

## [datetime](https://docs.python.org/2/library/datetime.html)

A Python module which includes tools for manipulating data and time objects.

## [Dictionary](https://docs.python.org/2/tutorial/datastructures.html" \l "dictionaries)

An associative array which is indexed by keys which map to values. Therefore, a dictionary is an unordered set of key:value pairs where each key is unique. Please see the following example of dictionary creation and access:

import pandas as pd  
student = pd.read\_csv('/Users/class.csv')  
for\_dict = pd.concat([student["Name"], student["Age"]], axis = 1)  
class\_dict = for\_dict.set\_index('Name').T.to\_dict('list')  
print(class\_dict.get('James'))

## [12]

## [List](https://www.tutorialspoint.com/python/python_lists.htm)

A sequence of comma-separated objects that need not be of the same type. Please see the following example of list creation and access:

list1 = ['item1', 102]  
print(list1)

## ['item1', 102]

print(list1[1])

## 102

Python also has what are known as ["Tuples"](https://www.tutorialspoint.com/python/python_tuples.htm), which are immutable lists created in the same way as lists, except with paranthesis instead of brackets.

## [Series](https://pandas.pydata.org/pandas-docs/stable/dsintro.html)

A one-dimensional data frame. Please see the following example of Series creation and access:

import pandas as pd  
my\_array = pd.Series([1, 3, 5, 9])  
print(my\_array)

## 0 1  
## 1 3  
## 2 5  
## 3 9  
## dtype: int64

print(my\_array[1])

## 3

## Sets & Frozen Sets

A set is a unordered collection of immutable objects. The difference between a [set and a frozen set](http://www.python-course.eu/sets_frozensets.php) is that the former is mutable, while the latter is immutable. Please see the following example of set and frozen set creation and access:

s = set(["1", "2", "3"])  
print(s)  
# s is a set, which means you can add or delete elements from s

## {'1', '2', '3'}

fs = frozenset(["1", "2", "3"])  
print(fs)  
# fs is a frozenset, which means you cannot add or delete elements from fs

## frozenset({'1', '2', '3'})

## [str](https://www.tutorialspoint.com/python/python_strings.htm)

A list of characters, though characters are not a type in Python, but rather a string of length 1. Strings are indexable like arrays. Please see the following example of String creation and access:

s = 'My first string!'  
print(s)

## My first string!

print(s[5])

## r

For more information on Python packages and functions, along with helpful examples, please see [Python](https://www.python.org/).