Python Tutorial

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 in Python given 3 different data 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" to return the dimensions of the data.

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()" function on the data.

# Notice that student is a DataFrame object  
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()" 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() 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.

# By default, the mean() function returns the mean of numeric variables of   
# the data only  
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()" 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

## 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

[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.

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

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

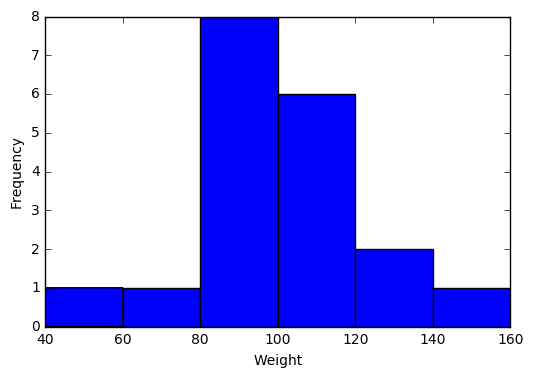
# 2 Basic Graphing and Plotting Functions

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

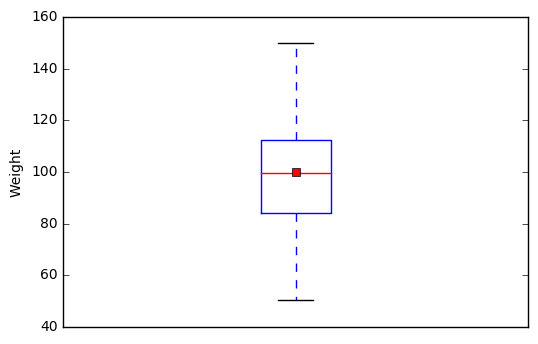
## 2.1 Visualize a single continuous variable by producing a histogram.

# Notice how the bin endpoints are set so the histogram is the same  
# as that produced by SAS and R  
# Also 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: 

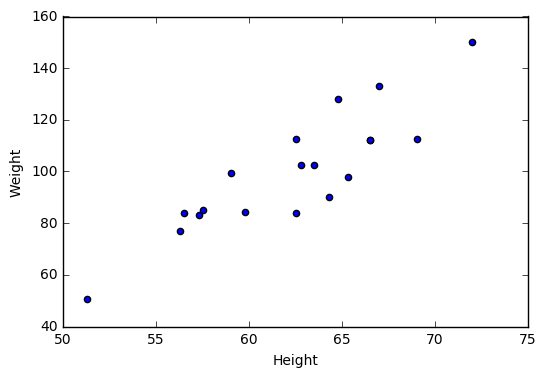
## 2.2 Visualize a single continuous variable by producing a boxplot.

# 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([]) # prevents Python from printing a "1" at the bottom of the boxplot  
plt.ylabel('Weight')  
plt.show()

Output: 

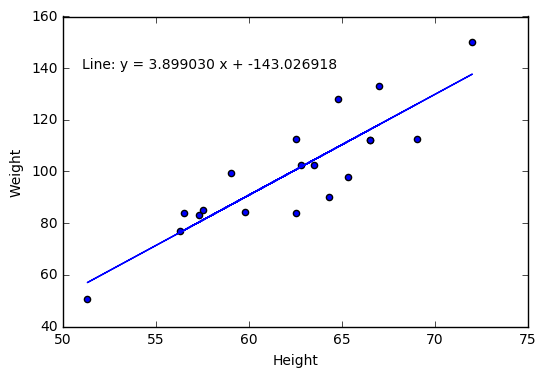
## 2.3 Visualize two continuous variables by producing a scatterplot.

# Notice here you specify the x variable first 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.

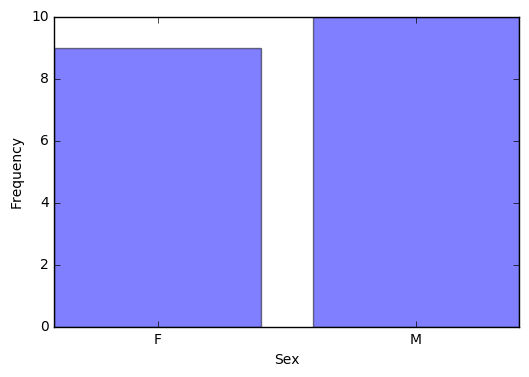
x = student["Height"]  
y = student["Weight"]  
  
# np.polyfit() models Weight as a function of Height and returns the   
# parameters  
m, b = np.polyfit(x, y, 1)  
plt.scatter(x, y)  
  
# plt.text() prints the equation of the line of best fit, with the first two   
# arguments specifying the x and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

[NumPy 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))  
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()

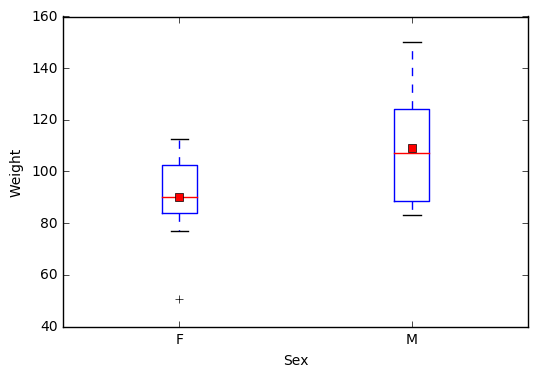
Output: 

[NumPy 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()

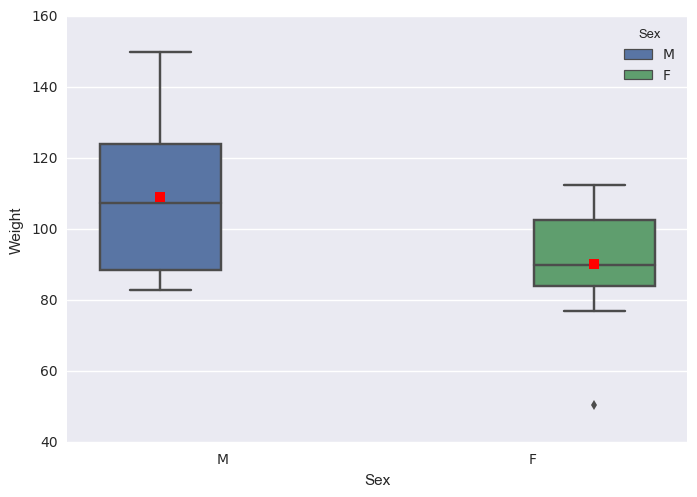
Output: 

### 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()

[seaborn boxplot](http://seaborn.pydata.org/generated/seaborn.boxplot.html)

[seaborn](#SEABORN)

Output: 

# 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())  
NA

## 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

[NumPy 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 log() function, the exp() function, the sqrt() function, and the abs() function.

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"])  
print(student.head())

## 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

## 3.5 Sort a data set by a variable.

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

# Notice the 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

## 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

## 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

## 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)

## 3.9 Caste a [Data Frame](#DataFrame) to a different object type.

student\_num = pd.concat([student["Age"], student["Height"],

## Age Height Weight  
## 0 14.0 69.0 112.5  
## 1 13.0 56.5 84.0  
## 2 13.0 65.3 98.0  
## 3 14.0 62.8 102.5  
## 4 14.0 63.5 102.5

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

# 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())  
NA

## 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

### 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

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

### 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

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

## 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())

## 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  
## Name: REVENUE, dtype: float64

[sub](#SUB)

[Decimal](#DECIMAL)

[pivot\_table()](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.pivot_table.html) for more information on the pandas pivot\_table() function

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

# 5 Regression & Modeling

The following sections focus on the Python [sklearn](#SKLEARN) package.

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

from sklearn.decomposition import PCA  
  
# 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)  
  
pca = PCA(n\_components = 4)  
pca = pca.fit(features)  
print(np.transpose(pca.components\_))

## [[ 0.36158968 0.65653988 -0.58099728 0.31725455]  
## [-0.08226889 0.72971237 0.59641809 -0.32409435]  
## [ 0.85657211 -0.1757674 0.07252408 -0.47971899]  
## [ 0.35884393 -0.07470647 0.54906091 0.75112056]]

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

## 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   
res = sm.formula.glm("greater15 ~ total\_bill", family=sm.families.Binomial(),   
 data=tips).fit()  
print(res.summary())

## 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: Wed, 07 Jun 2017 Deviance: 313.74  
## Time: 16:27:29 Pearson chi2: 247.  
## No. Iterations: 6   
## ==============================================================================  
## coef std err z P>|z| [95.0% Conf. Int.]  
## ------------------------------------------------------------------------------  
## 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 on training data and assess against testing 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')  
  
# Fit a linear regression model of tip by total\_bill on the training data   
from sklearn import linear\_model  
regr = linear\_model.LinearRegression()  
# If your data has one feature, you need to reshape the 1D array  
model = regr.fit(train["total\_bill"].reshape(-1,1), train["tip"])  
  
# Predict the tip based on the total\_bill given in the testing data   
prediction = pd.DataFrame()  
prediction["tip\_hat"] = regr.predict(test["total\_bill"].reshape(-1,1))  
  
# Compute the squared difference between predicted tip and actual tip   
prediction["diff"] = (prediction["tip\_hat"] - test["tip"])\*\*2  
  
# Compute the mean of the squared differences (mean squared error)   
# as an assessment of the model  
mean\_sq\_error = np.mean(prediction["diff"])  
print(mean\_sq\_error)

## 1.08759363430702

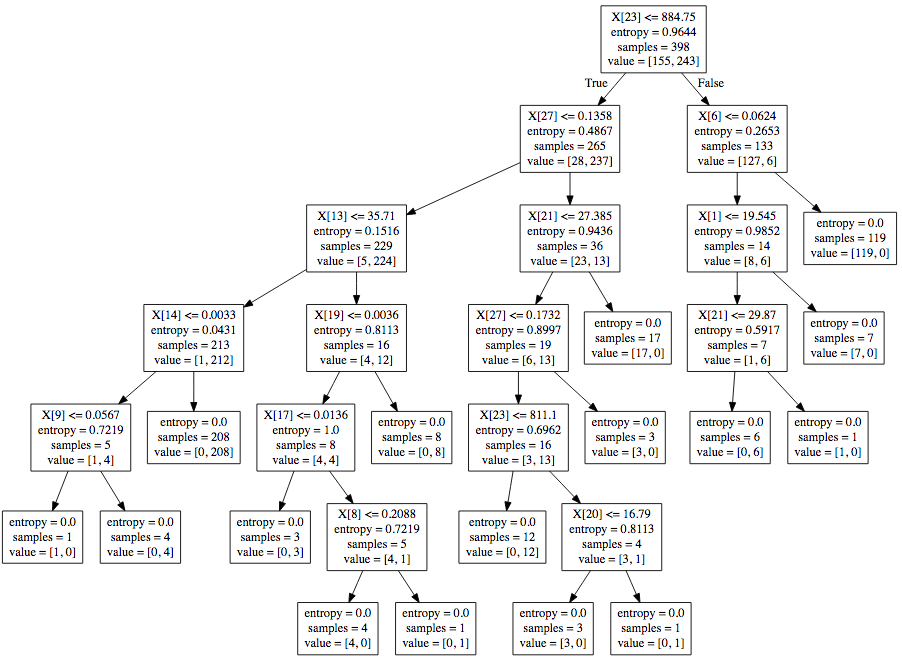
[LinearRegression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)

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

### a) Build a model, assess the model against the training data, plot the tree, 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 import tree  
  
# random\_state is used to specify a seed for a random integer so that the   
# results are reproducible  
clf = tree.DecisionTreeClassifier(criterion='entropy', random\_state=29)  
clf = clf.fit(train.drop(["Target"], axis = 1), train["Target"])  
  
# Prediction on training data  
scored = pd.DataFrame(clf.predict(train.drop(["Target"], axis = 1)))  
scored["Target"] = train["Target"]  
  
# Determine how many were correctly classified   
scored["correct"] = (scored["Target"] == scored[0])

## col\_0 count  
## correct   
## True 398

Output: 

# Determine variable importance  
var\_import = clf.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 0.592658  
## 27 0.172561  
## 6 0.055977  
## 21 0.054751  
## 13 0.032737

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

# Prediction on testing data  
scored = pd.DataFrame(clf.predict(test.drop(["Target"], axis = 1)))  
scored["Target"] = test["Target"]  
  
# Determine how many were correctly classified  
scored["correct"] = (scored["Target"] == scored[0])  
print(pd.crosstab(index=scored["correct"], columns="count"))

## col\_0 count  
## correct   
## False 9  
## True 162

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

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

### a) Build a model, assess the model against the 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/iris\_train.csv')  
test = pd.read\_csv('/Users/iris\_test.csv')  
  
from sklearn.ensemble import RandomForestClassifier  
  
clf = RandomForestClassifier(random\_state=29)  
clf = clf.fit(train.drop(["Target"], axis = 1), train["Target"])  
  
# Prediction on training data  
scored = pd.DataFrame(clf.predict(train.drop(["Target"], axis = 1)))  
scored["Target"] = train["Target"]  
  
# Determine how many were correctly classified  
scored["correct"] = (scored["Target"] == scored[0])  
print(pd.crosstab(index=scored["correct"], columns="count"))

## col\_0 count  
## correct   
## True 105

--

# Determine variable importance  
var\_import = clf.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  
## 2 0.515197  
## 3 0.388860  
## 0 0.080865  
## 1 0.015078

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

# Prediction on testing data  
scored = pd.DataFrame(clf.predict(test.drop(["Target"], axis = 1)))  
scored["Target"] = test["Target"]  
  
# Determine how many were correctly classified  
scored["correct"] = (scored["Target"] == scored[0])  
print(pd.crosstab(index=scored["correct"], columns="count"))

## col\_0 count  
## correct   
## False 3  
## True 42

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

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

### a) Build a model and assess the model against the training data.

# Notice we are re-using data sets but it is good to re-read the original   
# version back into the environment  
train = pd.read\_csv('/Users/tips\_train.csv')  
test = pd.read\_csv('/Users/tips\_test.csv')  
  
from sklearn.ensemble import RandomForestRegressor  
  
clf = RandomForestRegressor(random\_state=29)  
clf = clf.fit(train.drop(["tip"], axis = 1), train["tip"])  
  
# Prediction on training data  
scored = pd.DataFrame(clf.predict(train.drop(["tip"], axis = 1)))  
scored["Target"] = train["tip"]  
  
# Determine mean squared error  
scored["diff"] = (scored["Target"] - scored[0])\*\*2  
print(scored["diff"].mean())

## 0.2177569846153845

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

# Prediction on testing data  
scored = pd.DataFrame(clf.predict(test.drop(["tip"], axis = 1)))  
scored["Target"] = test["tip"]  
  
# Determine mean squared error  
scored["diff"] = (scored["Target"] - scored[0])\*\*2  
print(scored["diff"].mean())

## 1.182229489795918

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

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

### a) Build a model and assess the model against the training data.

# Notice we are re-using data sets but it is good to re-read the original   
# version back into the environment  
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, whereas a lower   
# learning rate requires more iterations  
# min\_samples\_leaf = minimum number of observations in the trees   
# terminal nodes  
  
clf = GradientBoostingClassifier(random\_state = 29, learning\_rate = .01, min\_samples\_leaf = 20, n\_estimators = 2500)  
clf = clf.fit(train.drop(["Target"], axis = 1), train["Target"])  
  
# Prediction on training data  
scored = pd.DataFrame(clf.predict(train.drop(["Target"], axis = 1)))  
scored["Target"] = train["Target"]  
  
# Determine how many were correctly classified  
scored["correct"] = (scored["Target"] == scored[0])  
print(pd.crosstab(index = scored["correct"], columns = "count"))

## col\_0 count  
## correct   
## True 398

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

# Prediction on testing data  
scored = pd.DataFrame(clf.predict(test.drop(["Target"], axis = 1)))  
scored["Target"] = test["Target"]  
  
# Determine how many were correctly classified  
scored["correct"] = (scored["Target"] == scored[0])

## col\_0 count  
## correct   
## False 4  
## True 167

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

## 5.9 Fit a support vector classification model.

### a) Build a model and assess the model against the training data.

# Notice we are re-using data sets but it is good to re-read the original   
# version back into the environment  
train = pd.read\_csv('/Users/breastcancer\_train.csv')  
test = pd.read\_csv('/Users/breastcancer\_test.csv')  
  
# First we need to scale the 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)  
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  
clf = SVC(random\_state = 29, kernel = 'linear')  
clf = clf.fit(train\_scaled.drop(["Target"], axis = 1), train\_scaled["Target"])  
  
# Evaluation on training data  
predictions = pd.DataFrame()  
predictions["predY"] = clf.predict(train\_scaled.drop(["Target"], axis = 1))  
  
# Determine how many were correctly classified  
predictions["actual"] = train\_scaled["Target"]  
predictions["correct"] = (predictions["actual"] == predictions["predY"])  
print(pd.crosstab(index = predictions["correct"], columns = "count"))

## col\_0 count  
## correct   
## False 6  
## True 392

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

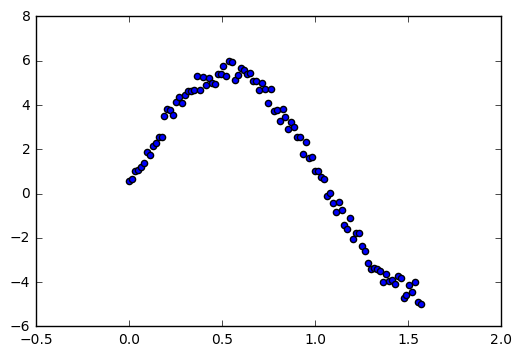
# Evaluation on testing data  
predictions = pd.DataFrame()  
predictions["predY"] = clf.predict(test\_scaled.drop(["Target"], axis = 1))  
  
# Determine how many were correctly classified  
predictions["actual"] = test\_scaled["Target"]  
predictions["correct"] = (predictions["actual"] == predictions["predY"])  
print(pd.crosstab(index = predictions["correct"], columns = "count"))

## col\_0 count  
## correct   
## False 7  
## True 164

## 5.10 Fit a support vector regression model.

### a) Generate random data based on a sine curve.

# Generate the time variable  
t = np.linspace(start = 0, stop = 0.5\*np.pi, num = 100)  
  
# Generate the sine curve with uniform noise  
y1 = 5\*np.sin(3\*t) + np.random.uniform(size=100)  
  
# Create a data frame for the generated data  
random\_data = pd.DataFrame()  
random\_data["X"] = t  
random\_data["Y"] = y1  
  
# Plot the generated data

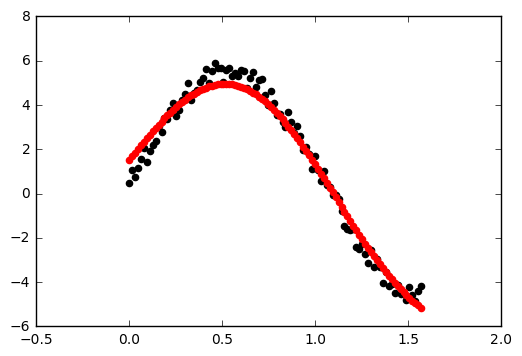
Output: 

[np.linspace()](https://docs.scipy.org/doc/numpy/reference/generated/numpy.linspace.html) [np.sin()](https://docs.scipy.org/doc/numpy-1.10.4/reference/generated/numpy.sin.html) [np.random.uniform()](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.uniform.html)

### b) Fit a support vector regression model to the data.

from sklearn.svm import SVR  
clf = SVR()  
clf = clf.fit(random\_data["X"].reshape(-1,1),random\_data["Y"])  
predictions = pd.DataFrame()  
predictions["predY"] = clf.predict(random\_data["X"].reshape(-1,1))

plt.scatter(t,y1,color="black")  
plt.scatter(t,predictions["predY"],color="r")  
plt.show()

Output: 

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

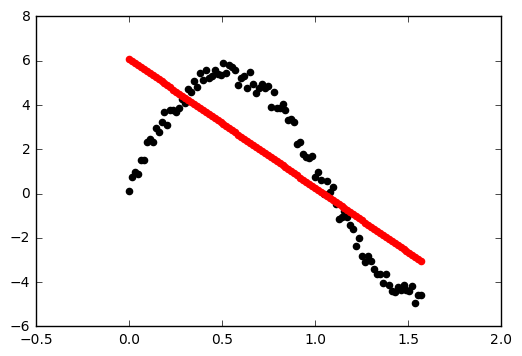
predictions["actual"] = random\_data["Y"]  
predictions["sq\_diff"] = (predictions["predY"] - predictions["actual"])\*\*2  
print(predictions["sq\_diff"].mean())

## 0.20626922373052764

### c) Fit a linear regression model to the data.

from sklearn import linear\_model  
linMod = linear\_model.LinearRegression()  
linMod = linMod.fit(random\_data["X"].reshape(-1,1), random\_data["Y"])  
predictions = pd.DataFrame()  
predictions["predY"] = linMod.predict(random\_data["X"].reshape(-1,1))

plt.scatter(t,y1,color="black")  
plt.scatter(t,predictions["predY"],color="r")  
plt.show()

Output: 

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

predictions["actual"] = random\_data["Y"]  
predictions["sq\_diff"] = (predictions["predY"] - predictions["actual"])\*\*2  
print(predictions["sq\_diff"].mean())

## 4.753313156881628

# 6 Model Evaluation & Selection

## 6.1 Evaluate the accuracy of regression models.

## a) Evaluation on training data.

# Notice we are re-using data sets but it is good to re-read the original   
# version back into the environment  
train = pd.read\_csv('/Users/tips\_train.csv')  
test = pd.read\_csv('/Users/tips\_test.csv')  
  
# 1. Linear Regression Model  
from sklearn.metrics import r2\_score  
from sklearn import linear\_model  
linMod = linear\_model.LinearRegression()  
linMod = linMod.fit(train.drop(["tip"], axis = 1), train["tip"])  
  
# Evaluation on training data  
pred\_lin = linMod.predict(train.drop(["tip"], axis = 1))  
  
# Determine coefficient of determination score  
r2\_lin = r2\_score(train["tip"], pred\_lin)  
print("Linear regression model r^2 score (coefficient of determination): %f" % r2\_lin)

## Linear regression model r^2 score (coefficient of determination): 0.496730

--

# 2. Random Forest Regression Model  
from sklearn.ensemble import RandomForestRegressor  
rfMod = RandomForestRegressor(random\_state=29)  
rfMod = rfMod.fit(train.drop(["tip"], axis = 1), train["tip"])  
  
# Evaluation on training data  
pred\_rf = rfMod.predict(train.drop(["tip"], axis = 1))  
  
# Determine coefficient of determination score  
r2\_rf = r2\_score(train["tip"], pred\_rf)  
print("Random forest regression model r^2 score (coefficient of determination): %f" % r2\_rf)

## Random forest regression model r^2 score (coefficient of determination): 0.892204

## b) Evaluation on testing data.

# 1. Linear Regression Model (linMod)  
  
# Evaluation on testing data  
pred\_lin = linMod.predict(test.drop(["tip"], axis = 1))  
  
# Determine coefficient of determination score  
r2\_lin = r2\_score(test["tip"], pred\_lin)  
print("Linear regression model r^2 score (coefficient of determination): %f" % r2\_lin)

## Linear regression model r^2 score (coefficient of determination): 0.270945

--

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

## Random forest regression model r^2 score (coefficient of determination): 0.163330

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.

## 6.2 Evaluate the accuracy of classification models.

### a) Evaluation on training data.

# Notice we are re-using data sets but it is good to re-read the original   
# version back into the environment  
train = pd.read\_csv('/Users/breastcancer\_train.csv')  
test = pd.read\_csv('/Users/breastcancer\_test.csv')  
  
# 1. Decision Tree Classification Model  
from sklearn import tree  
from sklearn.metrics import accuracy\_score  
treeMod = tree.DecisionTreeClassifier(criterion='entropy', random\_state=29)  
treeMod = treeMod.fit(train.drop(["Target"], axis = 1), train["Target"])  
  
# Evaluation on training data  
scored = pd.DataFrame(treeMod.predict(train.drop(["Target"], axis = 1)))  
scored["Target"] = train["Target"]  
  
# Determine accuracy score  
accuracy\_tree = accuracy\_score(scored["Target"], scored[0])  
print("Decision tree model accuracy: %f" % accuracy\_tree)

## Decision tree model accuracy: 1.000000

--

# 2. Random Forest Classification Model  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import accuracy\_score  
rfMod = RandomForestClassifier(random\_state=29)  
rfMod = rfMod.fit(train.drop(["Target"], axis = 1), train["Target"])  
  
# Evaluation on training data  
scored = pd.DataFrame(rfMod.predict(train.drop(["Target"], axis = 1)))  
scored["Target"] = train["Target"]  
  
# Determine accuracy score  
accuracy\_rf = accuracy\_score(scored["Target"], scored[0])  
print("Random forest model accuracy: %f" % accuracy\_rf)

## Random forest model accuracy: 0.997487

--

# 3. Gradient Boosting Classifcation Model  
from sklearn.ensemble import GradientBoostingClassifier  
from sklearn.metrics import accuracy\_score  
gbmMod = GradientBoostingClassifier(random\_state = 29, learning\_rate = .01, min\_samples\_leaf = 20, n\_estimators = 2500)  
gbmMod = gbmMod.fit(train.drop(["Target"], axis = 1), train["Target"])  
  
# Evaluation on training data  
scored = pd.DataFrame(gbmMod.predict(train.drop(["Target"], axis = 1)))  
scored["Target"] = train["Target"]  
  
# Determine accuracy score  
accuracy\_gbm = accuracy\_score(scored["Target"], scored[0])  
print("Gradient boosting model accuracy: %f" % accuracy\_gbm)

## Gradient boosting model accuracy: 1.000000

### b) Evaluation on testing data.

# 1. Decision Tree Classification Model (treeMod)  
  
# Evaluation on testing data  
scored = pd.DataFrame(treeMod.predict(test.drop(["Target"], axis = 1)))  
scored["Target"] = test["Target"]  
  
# Determine accuracy score  
accuracy\_tree = accuracy\_score(scored["Target"], scored[0])  
print("Decision tree model accuracy: %f" % accuracy\_tree)

## Decision tree model accuracy: 0.947368

--

# 2. Random Forest Classification Model (rfMod)  
  
# Evaluation on testing data  
scored = pd.DataFrame(rfMod.predict(test.drop(["Target"], axis = 1)))  
scored["Target"] = test["Target"]  
  
# Determine accuracy score  
accuracy\_rf = accuracy\_score(scored["Target"], scored[0])  
print("Random forest model accuracy: %f" % accuracy\_rf)

## Random forest model accuracy: 0.964912

--

# 3. Gradient Boosting Classifcation Model (gbmMod)  
  
# Evaluation on testing data  
scored = pd.DataFrame(gbmMod.predict(test.drop(["Target"], axis = 1)))  
scored["Target"] = test["Target"]  
  
# Determine accuracy score  
accuracy\_gbm = accuracy\_score(scored["Target"], scored[0])  
print("Gradient boosting model accuracy: %f" % accuracy\_gbm)

## Gradient boosting model accuracy: 0.976608

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.

## 6.3 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)  
model = RandomForestClassifier(random\_state = 29)  
results = model\_selection.cross\_val\_score(model, 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)  
model = RandomForestClassifier(random\_state = 29)  
results = model\_selection.cross\_val\_score(model, X, Y, cv = shuffle)  
  
print("Accuracy: %.2f%% +/- %.2f%%" % (results.mean()\*100,   
 results.std()\*100))

## Accuracy: 95.09% +/- 0.70%

# Appendix

## 1 Built-in Python Data Types

* [Boolean](#Bool)

### Numeric types:

* [int](#int)
* [long](#LONG)
* [float](#float)
* [complex](#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](#bokeh)

### [PyPlot](#PYPLOT)

### [Seaborn](#SEABORN)

# Alphabetical Index

## Array

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

For more information, please see [NumPy Arrays](https://docs.scipy.org/doc/numpy/reference/generated/numpy.array.html).

## Bokeh

[Bokeh](http://bokeh.pydata.org/en/latest/) is a Python package which is useful for interactive visualizations and is optimized for web browser presentations.

## Boolean

A [Boolean](https://docs.python.org/2/library/stdtypes.html#boolean-values) value is either True or False, and represents the truth of an expression or statement.

## 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.

## complex

A [complex number](https://docs.python.org/2/library/functions.html#complex) includes a real part and an imaginary part, both of which are floating point numbers.

## Data Frame

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

The [datetime](https://docs.python.org/2/library/datetime.html) Python module includes tools for manipulating data and time objects.

## Decimal

[Decimal](https://docs.python.org/2/library/decimal.html) is a Python package which provides tools for decimal floating point arithmetic.

## Dictionary

A [dictionary](https://docs.python.org/2/tutorial/datastructures.html#dictionaries) is 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]

## float

A [float](https://docs.python.org/2/library/functions.html#float) is a decimal point number.

## int

An [int](https://docs.python.org/3/library/functions.html#int) is a natural number. In Python, you can convert to an int from a float by using the int() function. Python stores ints with at least 32 bits of precision.

## List

A [list](https://www.tutorialspoint.com/python/python_lists.htm) is 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.

## Long

A [long](https://docs.python.org/2/library/functions.html#long) is a type of integer with unlimited precision. In Python, you can convert to a long using the long() function.

## NumPy

[NumPy](http://www.numpy.org/) is a Python package which is useful for scientific and mathematical computing.

## pandas

[pandas](http://pandas.pydata.org/) is a Python package which is useful for working with data structures and performing data analysis.

## PyPlot

[PyPlot](https://matplotlib.org/api/pyplot_api.html) is a Python package which is useful data plotting and visualization.

## Seaborn

[Seaborn](https://seaborn.pydata.org/) is another Python package which is useful for data plotting and visualization. In particular, Seaborn includes tools for drawing attractive statistical graphics.

## Series

A [Pandas Series](https://pandas.pydata.org/pandas-docs/stable/dsintro.html) is a one-dimensional data frame, which is also called an array in R. 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

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

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

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

## sklearn

scikit-learn, or more commonly known as [sklearn](http://scikit-learn.org/stable/), is a Python package which 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.

## str

[Strings](https://www.tutorialspoint.com/python/python_strings.htm) are 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

Please go [here](https://docs.python.org/3.1/library/functions.html#str) for more information on the str() function.

## sub

[sub](https://docs.python.org/2/library/re.html) is a function of the re Python package useful for replacing a pattern in a string.

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