In [ ]:

```
```

In [ ]:

```
```

In [1]:

```
#=======================================
# Section 2: Part 1: DATA PREPROCESSING
#=======================================
```

In [2]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd    # best for import and manage datasets
```

In [3]:

```
## 1) import dataset:
#====================

dataset = pd.read_csv('Machine Learning A-Z Template Folder/Part 1 - Data Preprocessing/Data.csv')


# matrix of features:
X = dataset.iloc[:, :-1].values    ## by comma we take all lines. Right to the comma are columns (except last)
print(X)

# dependent variable:
y = dataset.iloc[:, 3].values
print(y)
```

```
[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 nan]
 ['France' 35.0 58000.0]
 ['Spain' nan 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]
['No' 'Yes' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes']
```

In [4]:

```
## 2) missing data:
#==================




# replace by value of mean of column:

from sklearn.preprocessing import Imputer

#* class sklearn.preprocessing.Imputer(missing_values='NaN', strategy='mea
n', axis=0, verbose=0, copy=True)


# we have class, now create its object:
imputer = Imputer(missing_values = 'NaN', strategy = 'mean', axis = 0)
# fit the imputer object to data X:
imputer = imputer.fit(X[:, 1:3])
X[:, 1:3] = imputer.transform(X[:, 1:3])
print(X)
```

```
[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 63777.77777777778]
 ['France' 35.0 58000.0]
 ['Spain' 38.77777777777778 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]

/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecatio
n.py:58: DeprecationWarning: Class Imputer is deprecated; Impute
r was deprecated in version 0.20 and will be removed in 0.22. Im
port impute.SimpleImputer from sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
```

In [5]:

```
## 3) Encoding Categorical data:
#================================




from sklearn.preprocessing import LabelEncoder, OneHotEncoder

#* class sklearn.preprocessing.LabelEncoder
# Ref: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

#* class sklearn.preprocessing.OneHotEncoder(n_values=None, categorical_features=None, categories=None, drop=None, sparse=True, dtype=<class 'numpy.float64'>, handle_unknown='error')
# Ref: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html


X = X.copy()
labelencoder_X = LabelEncoder()
X[:, 0] = labelencoder_X.fit_transform(X[:, 0])
print(X)
print("-"*40)
print()



## Dummy Encoding for Categorical:
#---------------------------------


##
onehotencoder = OneHotEncoder(categorical_features = [0])
X = onehotencoder.fit_transform(X).toarray()
print(X)


# for y:
labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)
print(y)
```

```
[[0 44.0 72000.0]
 [2 27.0 48000.0]
 [1 30.0 54000.0]
 [2 38.0 61000.0]
 [1 40.0 63777.77777777778]
 [0 35.0 58000.0]
 [2 38.77777777777778 52000.0]
 [0 48.0 79000.0]
 [1 50.0 83000.0]
 [0 37.0 67000.0]]
----------------------------------------

[[1.00000000e+00 0.00000000e+00 0.00000000e+00 4.40000000e+01
  7.20000000e+04]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00 2.70000000e+01
  4.80000000e+04]
 [0.00000000e+00 1.00000000e+00 0.00000000e+00 3.00000000e+01
  5.40000000e+04]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00 3.80000000e+01
  6.10000000e+04]
 [0.00000000e+00 1.00000000e+00 0.00000000e+00 4.00000000e+01
  6.37777778e+04]
 [1.00000000e+00 0.00000000e+00 0.00000000e+00 3.50000000e+01
  5.80000000e+04]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00 3.87777778e+01
  5.20000000e+04]
 [1.00000000e+00 0.00000000e+00 0.00000000e+00 4.80000000e+01
  7.90000000e+04]
 [0.00000000e+00 1.00000000e+00 0.00000000e+00 5.00000000e+01
  8.30000000e+04]
 [1.00000000e+00 0.00000000e+00 0.00000000e+00 3.70000000e+01
  6.70000000e+04]]
[0 1 0 0 1 1 0 1 0 1]


/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/_en
coders.py:371: FutureWarning: The handling of integer data will
change in version 0.22. Currently, the categories are determined
based on the range [0, max(values)], while in the future they wi
ll be determined based on the unique values.
If you want the future behaviour and silence this warning, you c
an specify "categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to con
vert the categories to integers, then you can now use the OneHot
Encoder directly.
  warnings.warn(msg, FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/_en
coders.py:392: DeprecationWarning: The 'categorical_features' ke
yword is deprecated in version 0.20 and will be removed in 0.22.
You can use the ColumnTransformer instead.
  "use the ColumnTransformer instead.", DeprecationWarning)
```

In [6]:

```
## 4) Splitting into Training and Test set:
#==========================================



from sklearn.model_selection import train_test_split

# Ref: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
#* sklearn.model_selection.train_test_split(*arrays, **options)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0) # '*arrays=X, y'.
print(X_train, "\n", X_test, "\n", y_train, "\n", y_test)
```

```
[[0.00000000e+00 1.00000000e+00 0.00000000e+00 4.00000000e+01
  6.37777778e+04]
 [1.00000000e+00 0.00000000e+00 0.00000000e+00 3.70000000e+01
  6.70000000e+04]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00 2.70000000e+01
  4.80000000e+04]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00 3.87777778e+01
  5.20000000e+04]
 [1.00000000e+00 0.00000000e+00 0.00000000e+00 4.80000000e+01
  7.90000000e+04]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00 3.80000000e+01
  6.10000000e+04]
 [1.00000000e+00 0.00000000e+00 0.00000000e+00 4.40000000e+01
  7.20000000e+04]
 [1.00000000e+00 0.00000000e+00 0.00000000e+00 3.50000000e+01
  5.80000000e+04]]
 [[0.0e+00 1.0e+00 0.0e+00 3.0e+01 5.4e+04]
 [0.0e+00 1.0e+00 0.0e+00 5.0e+01 8.3e+04]]
 [1 1 1 0 1 0 0 1]
 [0 0]
```

In [7]:

```
## 5) Feature Scaling:
#=====================



## ** Lots of ML models are based on Euclidean Distance, so scaling is neces
sary.


from sklearn.preprocessing import StandardScaler


sc_X = StandardScaler()

#* for train, we have to 'fit' and 'transform'. # only 'transform' for test
 set.
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
print(X_train)
print(X_test)

#* Not scaling dummy variables will not break the model.

#* Here we don't need to apply feature scaling to y.

## ** Most libraries do Feature Scaling by themselves.
```

```
[[-1.          2.64575131 -0.77459667  0.26306757  0.12381479]
 [ 1.         -0.37796447 -0.77459667 -0.25350148  0.46175632]
 [-1.         -0.37796447  1.29099445 -1.97539832 -1.53093341]
 [-1.         -0.37796447  1.29099445  0.05261351 -1.11141978]
 [ 1.         -0.37796447 -0.77459667  1.64058505  1.7202972 ]
 [-1.         -0.37796447  1.29099445 -0.0813118  -0.16751412]
 [ 1.         -0.37796447 -0.77459667  0.95182631  0.98614835]
 [ 1.         -0.37796447 -0.77459667 -0.59788085 -0.48214934]]
[[-1.          2.64575131 -0.77459667 -1.45882927 -0.90166297]
 [-1.          2.64575131 -0.77459667  1.98496442  2.13981082]]
```

Standardisation	Normalisation
$x_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation }(x)}$	$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$

In [ ]:

In [ ]:

In [8]:

```
##============================
## Data Preprocessing Template:
#============================


## Data Preprocessing Template

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
#dataset = pd.read_csv('Data.csv')
dataset = pd.read_csv('Machine Learning A-Z Template Folder/Part 1 - Data Preprocessing/Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 3].values

# missing data (skipped)
# Dummy encoding Categorical data (skipped)

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

# Feature Scaling
"""from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train)"""
```

Out[8]:

```
'from sklearn.preprocessing import StandardScaler\nsc_X = StandardScaler()\nX_train = sc_X.fit_transform(X_train)\nX_test = sc_X.transform(X_test)\nsc_y = StandardScaler()\ny_train = sc_y.fit_transform(y_train)'
```

In [ ]:

In [ ]:

In [9]:

```
#==============================
# Section 3: Part 2: REGRESSION
#==============================
```

In [ ]:

```

```

In [ ]:

```

```

=> Regression technique vary from Linear Regression to SVR and Random Forests Regression.

=> following Machine Learning Regression models:

- Simple Linear Regression
- Multiple Linear Regression
- Polynomial Regression
- Support Vector for Regression (SVR)
- Decision Tree Classification
- Random Forest Classification

In [115]:

```
#=> Regression technique vary from Linear Regression to SVR and Random Fores
ts Regression.

#=> following Machine Learning Regression models:
 # Simple Linear Regression
 # Multiple Linear Regression
 # Polynomial Regression
 # Support Vector for Regression (SVR)
 # Decision Tree Classification
 # Random Forest Classification
```

In [ ]:

```

```

In [11]:

```
#==================================
# Section 4: Simple Linear Regression
#==================================
```

In [ ]:

In [14]:

```
## ** OLS: minimize 'sum of squares (of difference between y and y^)'.


### STEP 1: Preprocessing:

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
#dataset = pd.read_csv('Data.csv')
path = 'Machine Learning A-Z Template Folder/Part 2 - Regression/Section 4 - Simple Linear Regression/'
dataset = pd.read_csv(path + 'Salary_Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

## ** X is a matrix, while y is a vector.


# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 0)

# Feature Scaling    (not neaded for this ML module)
"""from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train)"""


#########

### STEP 2: Fitting simple linear regression to the training set:
from sklearn.linear_model import LinearRegression

regressor = LinearRegression()
regressor.fit(X_train, y_train)


### STEP 3: predicting the test set results:
y_pred = regressor.predict(X_test)
print(y_pred)


# Visualising the 'training set' results:
```

```
#* here we compare real to the predicted y values (salary), for the 'train'
 set.
plt.scatter(X_train, y_train, color='red')
plt.plot(X_train, regressor.predict(X_train), color='blue')
plt.title('salary vs experience (training set)')
plt.xlabel('years of experience')
plt.ylabel('salary')
plt.show()


# visualising the test set results:
#* here we compare real to the predicted y values (salary), for the 'train'
 set.
plt.scatter(X_test, y_test, color='red')
plt.plot(X_train, regressor.predict(X_train), color='blue')  #* it will be t
he same regressor for train and test.
plt.title('salary vs experience (test set)')
plt.xlabel('years of experience')
plt.ylabel('salary')
plt.show()

## ** Blue regression line is the same line for both plots.
```

```
[ 40835.10590871 123079.39940819  65134.55626083  63265.36777221
 115602.64545369 108125.8914992  116537.23969801  64199.96201652
  76349.68719258 100649.1375447 ]
```

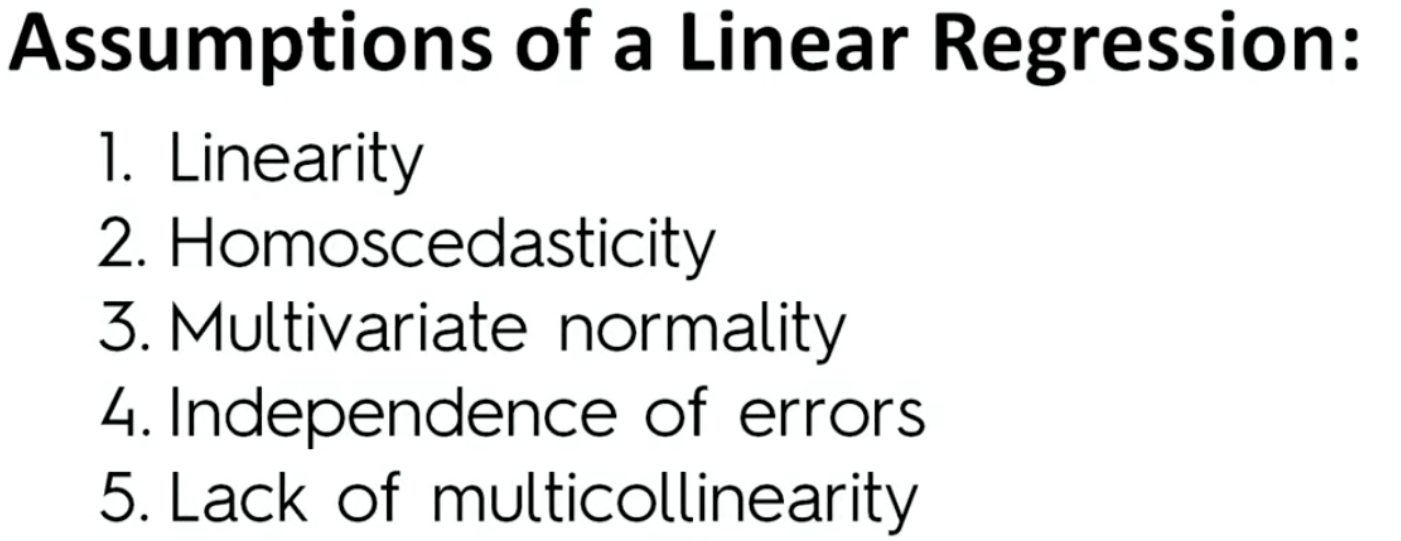
In [ ]:

In [18]:

```
#=======================================
# Section 5: Multiple Linear Regression
#=======================================
```

In [ ]:

```

```



In [22]:

```
## ** Create Dummy Variable for Categorical variables.
## ** Never to use all Dummy variables in regression equation. Always omit o
ne dummy variable.
    # thus being  Careful about 'Dummy Variable Trap'. (eg.: D_2 = 1 - D_1)
    # The 'one omit' rule apply to each set of dummy variables.



###### P VALUE:
#==============
# P values: are used to determine whether the results of their experiment ar
e within the normal range of
# values for the events being observed.
# Usually, if the P value of a data set is below a certain pre-determined am
ount (like, for instance, 0.05),
# scientists will reject the "null hypothesis" of their experiment - in othe
r words, they'll rule out the
# hypothesis that the variables of their experiment had no meaningful effect
on the results.


"""
The p-value is NOT the probability the claim is true.:
Of course, this would be an amazing thing to know! Think of it “there is 10%
chance that this medicine works”.
Unfortunately, this just isnt the case. Actually determining this probabilit
y would be really tough if not impossible!
The p-value is NOT the probability the null hypothesis is true.:
Another one that seems so logical it has to be right! This one is much close
r to the reality, but again it is
way too strong of a statement.
"""


# The p-value is actually the probability of getting a sample like ours, or
 more extreme than ours IF the
# null hypothesis is true. So, we assume the null hypothesis is true and the
n determine how “strange” our
# sample really is. If it is not that strange (a large p-value) then we do
n’t change our mind about the
# null hypothesis.
```

Out[22]:

```
'\nThe p-value is NOT the probability the claim is true.:\nOf co
urse, this would be an amazing thing to know! Think of it “there
is 10% chance that this medicine works”. \nUnfortunately, this j
ust isnt the case. Actually determining this probability would b
e really tough if not impossible!\nThe p-value is NOT the probab
ility the null hypothesis is true.:\nAnother one that seems so l
ogical it has to be right! This one is much closer to the realit
y, but again it is \nway too strong of a statement.\n'
```

In [23]:

```
# Building a model:
#==================

#  - a PDF file in the course folder.



## ** here, Backward Elimination mostly used. It is the fastest.
```

In [57]:

```
## Multiple Linear Regression implement:

### STEP 1: Preprocessing:

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
#dataset = pd.read_csv('Data.csv')
path = 'Machine Learning A-Z Template Folder/Part 2 - Regression/Section 5 - Multiple Linear Regression/'
dataset = pd.read_csv(path + '50_Startups.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 4].values
print(type(X))


# dummy encoding:
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

labelencoder_X = LabelEncoder()
X[:, 3] = labelencoder_X.fit_transform(X[:, 3])
print(X)
print()
onehotencoder = OneHotEncoder(categorical_features = [3])
X = onehotencoder.fit_transform(X).toarray()
print(type(X), "\n", X)
print()
# encoding for y (not needed)

## AVOIDING THE DUMMY VARIABLE TRAP:
X = X[:, 1:]    # removing one of the 3 dummy vars.
## ** Python lbrary for linear regression takes care of this however.
print(X)

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

# Feature Scaling    (not neaded for this ML module)
"""from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train)"""
```

```
##############

## STEP 2:
from sklearn.linear_model import LinearRegression

regressor = LinearRegression()
regressor.fit(X_train, y_train)

## STEP 3:
y_pred = regressor.predict(X_test)
print(y_pred)
```

```
<class 'numpy.ndarray'>
[[165349.2 136897.8 471784.1 2]
 [162597.7 151377.59 443898.53 0]
 [153441.51 101145.55 407934.54 1]
 [144372.41 118671.85 383199.62 2]
 [142107.34 91391.77 366168.42 1]
 [131876.9 99814.71 362861.36 2]
 [134615.46 147198.87 127716.82 0]
 [130298.13 145530.06 323876.68 1]
 [120542.52 148718.95 311613.29 2]
 [123334.88 108679.17 304981.62 0]
 [101913.08 110594.11 229160.95 1]
 [100671.96 91790.61 249744.55 0]
 [93863.75 127320.38 249839.44 1]
 [91992.39 135495.07 252664.93 0]
 [119943.24 156547.42 256512.92 1]
 [114523.61 122616.84 261776.23 2]
 [78013.11 121597.55 264346.06 0]
 [94657.16 145077.58 282574.31 2]
 [91749.16 114175.79 294919.57 1]
 [86419.7 153514.11 0.0 2]
 [76253.86 113867.3 298664.47 0]
 [78389.47 153773.43 299737.29 2]
 [73994.56 122782.75 303319.26 1]
 [67532.53 105751.03 304768.73 1]
 [77044.01 99281.34 140574.81 2]
 [64664.71 139553.16 137962.62 0]
 [75328.87 144135.98 134050.07 1]
 [72107.6 127864.55 353183.81 2]
 [66051.52 182645.56 118148.2 1]
 [65605.48 153032.06 107138.38 2]
 [61994.48 115641.28 91131.24 1]
 [61136.38 152701.92 88218.23 2]
 [63408.86 129219.61 46085.25 0]
 [55493.95 103057.49 214634.81 1]
 [46426.07 157693.92 210797.67 0]
 [46014.02 85047.44 205517.64 2]
 [28663.76 127056.21 201126.82 1]
 [44069.95 51283.14 197029.42 0]
 [20229.59 65947.93 185265.1 2]
 [38558.51 82982.09 174999.3 0]
 [28754.33 118546.05 172795.67 0]
 [27892.92 84710.77 164470.71 1]
 [23640.93 96189.63 148001.11 0]
 [15505.73 127382.3 35534.17 2]
 [22177.74 154806.14 28334.72 0]
 [1000.23 124153.04 1903.93 2]
 [1315.46 115816.21 297114.46 1]
 [0.0 135426.92 0.0 0]
 [542.05 51743.15 0.0 2]
 [0.0 116983.8 45173.06 0]]

<class 'numpy.ndarray'>
```

```
[[     0.        0.        1.   165349.2  136897.8  471784.1 ]
 [     1.        0.        0.   162597.7  151377.59 443898.53]
 [     0.        1.        0.   153441.51 101145.55 407934.54]
 [     0.        0.        1.   144372.41 118671.85 383199.62]
 [     0.        1.        0.   142107.34  91391.77 366168.42]
 [     0.        0.        1.   131876.9   99814.71 362861.36]
 [     1.        0.        0.   134615.46 147198.87 127716.82]
 [     0.        1.        0.   130298.13 145530.06 323876.68]
 [     0.        0.        1.   120542.52 148718.95 311613.29]
 [     1.        0.        0.   123334.88 108679.17 304981.62]
 [     0.        1.        0.   101913.08 110594.11 229160.95]
 [     1.        0.        0.   100671.96  91790.61 249744.55]
 [     0.        1.        0.    93863.75 127320.38 249839.44]
 [     1.        0.        0.    91992.39 135495.07 252664.93]
 [     0.        1.        0.   119943.24 156547.42 256512.92]
 [     0.        0.        1.   114523.61 122616.84 261776.23]
 [     1.        0.        0.    78013.11 121597.55 264346.06]
 [     0.        0.        1.    94657.16 145077.58 282574.31]
 [     0.        1.        0.    91749.16 114175.79 294919.57]
 [     0.        0.        1.    86419.7  153514.11      0.  ]
 [     1.        0.        0.    76253.86 113867.3  298664.47]
 [     0.        0.        1.    78389.47 153773.43 299737.29]
 [     0.        1.        0.    73994.56 122782.75 303319.26]
 [     0.        1.        0.    67532.53 105751.03 304768.73]
 [     0.        0.        1.    77044.01  99281.34 140574.81]
 [     1.        0.        0.    64664.71 139553.16 137962.62]
 [     0.        1.        0.    75328.87 144135.98 134050.07]
 [     0.        0.        1.    72107.6  127864.55 353183.81]
 [     0.        1.        0.    66051.52 182645.56 118148.2 ]
 [     0.        0.        1.    65605.48 153032.06 107138.38]
 [     0.        1.        0.    61994.48 115641.28  91131.24]
 [     0.        0.        1.    61136.38 152701.92  88218.23]
 [     1.        0.        0.    63408.86 129219.61  46085.25]
 [     0.        1.        0.    55493.95 103057.49 214634.81]
 [     1.        0.        0.    46426.07 157693.92 210797.67]
 [     0.        0.        1.    46014.02  85047.44 205517.64]
 [     0.        1.        0.    28663.76 127056.21 201126.82]
 [     1.        0.        0.    44069.95  51283.14 197029.42]
 [     0.        0.        1.    20229.59  65947.93 185265.1 ]
 [     1.        0.        0.    38558.51  82982.09 174999.3 ]
 [     1.        0.        0.    28754.33 118546.05 172795.67]
 [     0.        1.        0.    27892.92  84710.77 164470.71]
 [     1.        0.        0.    23640.93  96189.63 148001.11]
 [     0.        0.        1.    15505.73 127382.3   35534.17]
 [     1.        0.        0.    22177.74 154806.14  28334.72]
 [     0.        0.        1.     1000.23 124153.04   1903.93]
 [     0.        1.        0.     1315.46 115816.21 297114.46]
 [     1.        0.        0.        0.   135426.92      0.  ]
 [     0.        0.        1.      542.05  51743.15      0.  ]
 [     1.        0.        0.        0.   116983.8   45173.06]]

[[     0.        1.   165349.2  136897.8  471784.1 ]
 [     0.        0.   162597.7  151377.59 443898.53]
```

```
[     1.         0.    153441.51 101145.55 407934.54]
[     0.         1.    144372.41 118671.85 383199.62]
[     1.         0.    142107.34  91391.77 366168.42]
[     0.         1.    131876.9   99814.71 362861.36]
[     0.         0.    134615.46 147198.87 127716.82]
[     1.         0.    130298.13 145530.06 323876.68]
[     0.         1.    120542.52 148718.95 311613.29]
[     0.         0.    123334.88 108679.17 304981.62]
[     1.         0.    101913.08 110594.11 229160.95]
[     0.         0.    100671.96  91790.61 249744.55]
[     1.         0.     93863.75 127320.38 249839.44]
[     0.         0.     91992.39 135495.07 252664.93]
[     1.         0.    119943.24 156547.42 256512.92]
[     0.         1.    114523.61 122616.84 261776.23]
[     0.         0.     78013.11 121597.55 264346.06]
[     0.         1.     94657.16 145077.58 282574.31]
[     1.         0.     91749.16 114175.79 294919.57]
[     0.         1.     86419.7  153514.11      0.  ]
[     0.         0.     76253.86 113867.3  298664.47]
[     0.         1.     78389.47 153773.43 299737.29]
[     1.         0.     73994.56 122782.75 303319.26]
[     1.         0.     67532.53 105751.03 304768.73]
[     0.         1.     77044.01  99281.34 140574.81]
[     0.         0.     64664.71 139553.16 137962.62]
[     1.         0.     75328.87 144135.98 134050.07]
[     0.         1.     72107.6  127864.55 353183.81]
[     1.         0.     66051.52 182645.56 118148.2 ]
[     0.         1.     65605.48 153032.06 107138.38]
[     1.         0.     61994.48 115641.28  91131.24]
[     0.         1.     61136.38 152701.92  88218.23]
[     0.         0.     63408.86 129219.61  46085.25]
[     1.         0.     55493.95 103057.49 214634.81]
[     0.         0.     46426.07 157693.92 210797.67]
[     0.         1.     46014.02  85047.44 205517.64]
[     1.         0.     28663.76 127056.21 201126.82]
[     0.         0.     44069.95  51283.14 197029.42]
[     0.         1.     20229.59  65947.93 185265.1 ]
[     0.         0.     38558.51  82982.09 174999.3 ]
[     0.         0.     28754.33 118546.05 172795.67]
[     1.         0.     27892.92  84710.77 164470.71]
[     0.         0.     23640.93  96189.63 148001.11]
[     0.         1.     15505.73 127382.3   35534.17]
[     0.         0.     22177.74 154806.14  28334.72]
[     0.         1.      1000.23 124153.04   1903.93]
[     1.         0.      1315.46 115816.21 297114.46]
[     0.         0.         0.   135426.92      0.  ]
[     0.         1.       542.05  51743.15      0.  ]
[     0.         0.         0.   116983.8   45173.06]]
[103015.20159796 132582.27760815 132447.73845175  71976.09851258
 178537.48221056 116161.24230166  67851.69209676  98791.73374687
 113969.43533013 167921.06569551]
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/_en
coders.py:371: FutureWarning: The handling of integer data will
change in version 0.22. Currently, the categories are determined
based on the range [0, max(values)], while in the future they wi
ll be determined based on the unique values.
If you want the future behaviour and silence this warning, you c
an specify "categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to con
vert the categories to integers, then you can now use the OneHot
Encoder directly.
  warnings.warn(msg, FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/_en
coders.py:392: DeprecationWarning: The 'categorical_features' ke
yword is deprecated in version 0.20 and will be removed in 0.22.
You can use the ColumnTransformer instead.
  "use the ColumnTransformer instead.", DeprecationWarning)
```

In [ ]:

In [58]:

```
## Multiple Linear Regression - "Backward Elimination":
#-----------------------------------------------------


# building the optimal model using Backward Elimination:

## ** BE: Step by step remove the least significant variable (max. P Value),
untill all are below the
    # cutoff P Value.

import statsmodels.formula.api as sm

np.set_printoptions(suppress=True)

#* sm library doesn't take into account for constant term in regression equa
tion.
     # so, add a ones (1 s) column in front of X.
#print(X)
X = np.append(arr = np.ones((50, 1)).astype(int), values = X, axis = 1)  # a
xis=1: for adding column.
#print(X[:, 1])
X_opt = X[:, [0, 1, 2, 3, 4, 5]]  # specify all column indices
# regressor from OLS class:
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()

X_opt = X[:, [0, 1, 3, 4, 5]]   # removed col with index 2.
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()

X_opt = X[:, [0, 3, 4, 5]]   # removed col with index 1.
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()

X_opt = X[:, [0, 3, 5]]   # removed col with index 4.  {'R&D' and 'marketing
spent' left}
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()

X_opt = X[:, [0, 3]]   # removed col with index 5 ('marketing spent').  {'R&
D' left}
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()
```

Out[58]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.947
Model:	OLS	**Adj. R-squared:**	0.945
Method:	Least Squares	**F-statistic:**	849.8
Date:	Sat, 13 Jul 2019	**Prob (F-statistic):**	3.50e-32
Time:	18:54:48	**Log-Likelihood:**	-527.44
No. Observations:	50	**AIC:**	1059.
Df Residuals:	48	**BIC:**	1063.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	4.903e+04	2537.897	19.320	0.000	4.39e+04	5.41e+04
x1	0.8543	0.029	29.151	0.000	0.795	0.913

Omnibus:	13.727	**Durbin-Watson:**	1.116
Prob(Omnibus):	0.001	**Jarque-Bera (JB):**	18.536
Skew:	-0.911	**Prob(JB):**	9.44e-05
Kurtosis:	5.361	**Cond. No.**	1.65e+05

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.65e+05. This might indicate that there are strong multicollinearity or other numerical problems.

In [ ]:

Backward Elimination with p-values only:

import statsmodels.formula.api as sm def backwardElimination(x, sl): numVars = len(x[0]) for i in range(0, numVars): regressor_OLS = sm.OLS(y, x).fit() maxVar = max(regressor_OLS.pvalues).astype(float) if maxVar > sl: for j in range(0, numVars - i): if (regressor_OLS.pvalues[j].astype(float) == maxVar): x = np.delete(x, j, 1) regressor_OLS.summary() return x

SL = 0.05 X_opt = X[:, [0, 1, 2, 3, 4, 5]] X_Modeled = backwardElimination(X_opt, SL)

Backward Elimination with p-values and Adjusted R Squared:

import statsmodels.formula.api as sm def backwardElimination(x, SL): numVars = len(x[0]) temp = np.zeros((50,6)).astype(int) for i in range(0, numVars): regressor_OLS = sm.OLS(y, x).fit() maxVar = max(regressor_OLS.pvalues).astype(float) adjR_before = regressor_OLS.rsquared_adj.astype(float) if maxVar > SL: for j in range(0, numVars - i): if (regressor_OLS.pvalues[j].astype(float) == maxVar): temp[:,j] = x[:, j] x = np.delete(x, j, 1) tmp_regressor = sm.OLS(y, x).fit() adjR_after = tmp_regressor.rsquared_adj.astype(float) if (adjR_before >= adjR_after): x_rollback = np.hstack((x, temp[:,[0,j]])) x_rollback = np.delete(x_rollback, j, 1) print (regressor_OLS.summary()) return x_rollback else: continue regressor_OLS.summary() return x

SL = 0.05 X_opt = X[:, [0, 1, 2, 3, 4, 5]] X_Modeled = backwardElimination(X_opt, SL)

In [ ]:

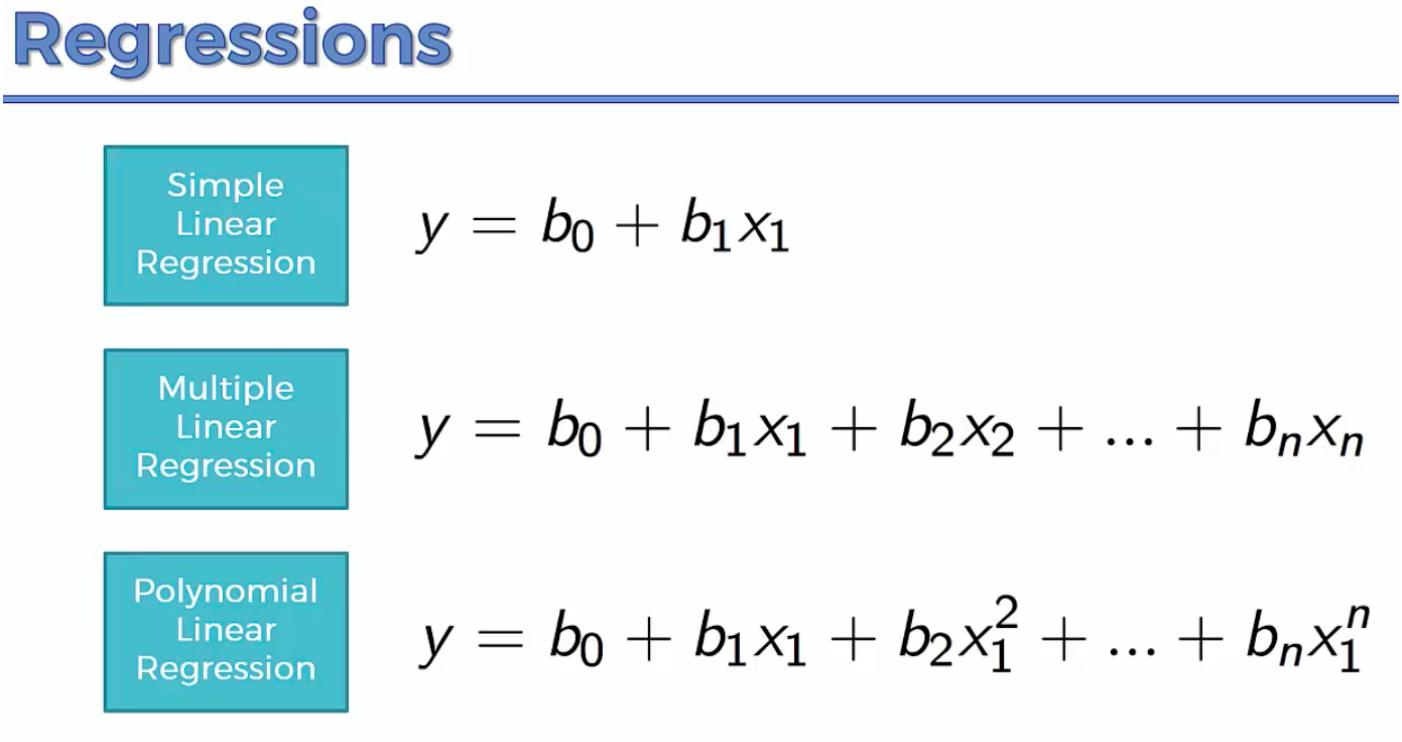
```
```

In [60]:

```
#==================================
# Section 6: Polynomial Regression
#==================================


#* Also called "Polynomial (Linear) Regression"
```

In [ ]:

```
```


Regressions
Simple Linear Regression
$y = b_0 + b_1 x_1$
Multiple Linear Regression
$y = b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_n x_n$
Polynomial Linear Regression
$y = b_0 + b_1 x_1 + b_2 x_1^2 + \ldots + b_n x_1^n$

In [ ]:

In [ ]:

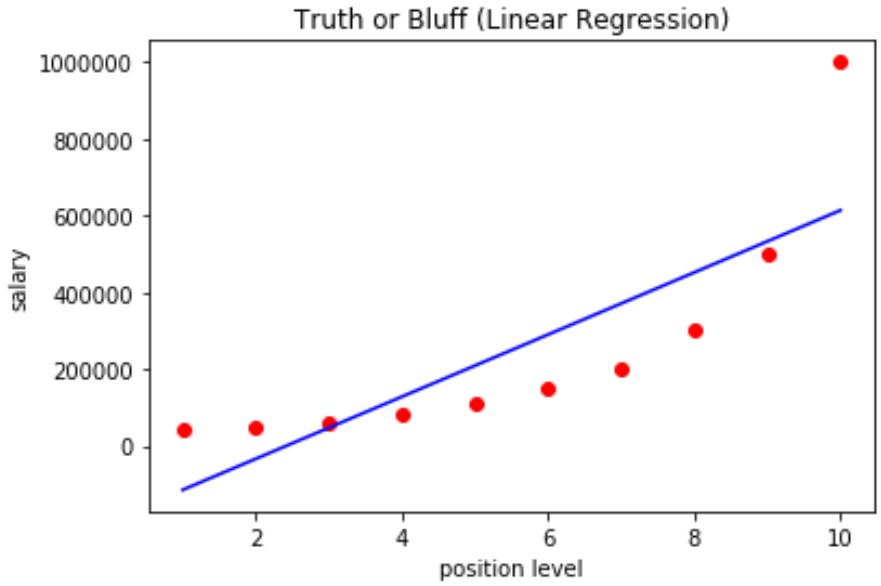
```
## ** Why "LINEAR" still:
     # Because 'y' is still expressed as 'Linear Combination' of Coefficient
s (Unknowns).


## Polynomial (Linear) Regression is a special case of Multiple Linear Regre
ssion.



## ** To create a matrix instead of vector specify a range instead of a sing
le row/column.
    # "x[:, 1:2]" instead of "x[:, 1]".
```

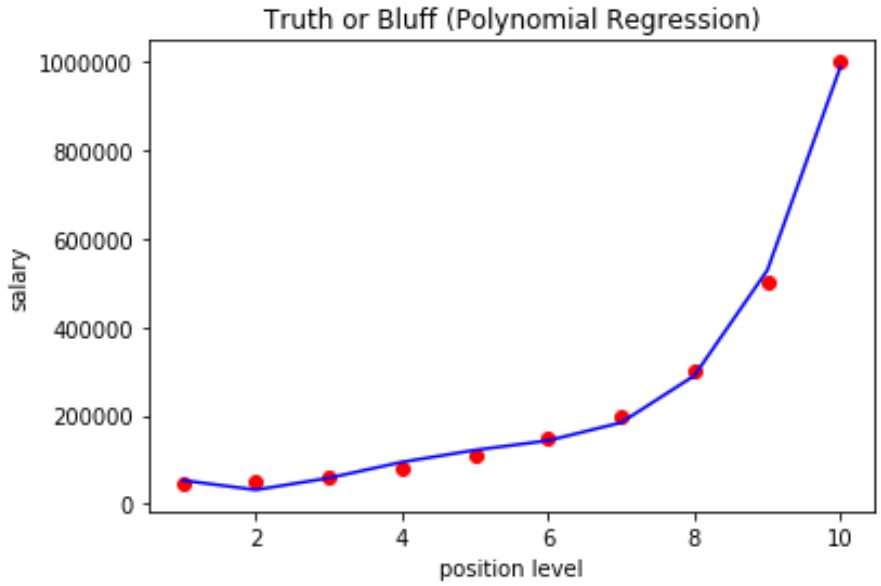
In [81]:

```
### STEP 1: Preprocessing:

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
path = 'Machine Learning A-Z Template Folder/Part 2 - Regression/Section 6 - Polynomial Regression/'
dataset = pd.read_csv(path + 'Position_Salaries.csv')
X = dataset.iloc[:, 1:2].values  #** 2nd index value specified to keep X as matrix, instead of vector.
y = dataset.iloc[:, 2].values
print(type(X))

#*Very small dataset, so Splitting into test and train: Skipped.
# # Splitting the dataset into the Training set and Test set
# from sklearn.model_selection import train_test_split
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

# Feature Scaling    (not neaded for this ML module)
"""from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train)"""


##############

# Fitting linear regression to the dataset:
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(X, y)

# Fitting Polynomial Regression to the dataset:
from sklearn.preprocessing import PolynomialFeatures

poly_reg = PolynomialFeatures(degree = 4)    #* specify the degree
X_poly = poly_reg.fit_transform(X)
#print(X_poly)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly, y)


# Visualising the Linear regression results:
```

```
plt.scatter(X, y, color = 'red')
plt.plot(X, lin_reg.predict(X), color = 'blue')
plt.title("Truth or Bluff (Linear Regression)")
plt.xlabel('position level')
plt.ylabel('salary')
plt.show()


# Visualising the Polynomial Regression results:
X_grid = np.arange(min(X), max(X), 0.1)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X, y, color = 'red')
#plt.plot(X, lin_reg_2.predict(X_poly), color = 'blue')
plt.plot(X, lin_reg_2.predict(poly_reg.fit_transform(X)), color = 'blue')
#plt.plot(X_grid, lin_reg_2.predict(poly_reg.fit_transform(X_grid)), color =
'blue')
plt.title("Truth or Bluff (Polynomial Regression)")
plt.xlabel('position level')
plt.ylabel('salary')
plt.show()


# Predicting a new result with Linear Regression:
y_pred = lin_reg.predict([[6.5]])
print(y_pred)

# Predicting a new result with Polynomial Regression:
y_pred_poly = lin_reg_2.predict(poly_reg.fit_transform([[6.5]]))
print(y_pred_poly)
```

```
<class 'numpy.ndarray'>
```






```
[330378.78787879]
[158862.45265153]
```

In [ ]:

In [ ]:

```
#============================
# Python Regression Template
#============================


# Regression Template

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
path = 'Machine Learning A-Z Template Folder/Part 2 - Regression/Section 6 - Polynomial Regression/'
dataset = pd.read_csv(path + 'Position_Salaries.csv')
X = dataset.iloc[:, 1:2].values
y = dataset.iloc[:, 2].values

# Splitting the dataset into the Training set and Test set
"""from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)"""

# Feature Scaling
"""from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train)"""

# Fitting the Regression Model to the dataset
# Create your regressor here

# Predicting a new result
y_pred = regressor.predict([[6.5]])

# Visualising the Regression results
plt.scatter(X, y, color = 'red')
plt.plot(X, regressor.predict(X), color = 'blue')
plt.title('Truth or Bluff (Regression Model)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()

# Visualising the Regression results (for 'higher resolution' and smoother curve)
X_grid = np.arange(min(X), max(X), 0.1)
X_grid = X_grid.reshape((len(X_grid), 1))
```

```
plt.scatter(X, y, color = 'red')
plt.plot(X_grid, regressor.predict(X_grid), color = 'blue')
plt.title('Truth or Bluff (Regression Model)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```

In [ ]:

```

```

In [ ]:

```
#====================================
# Section 7: Support Vector Regression
#====================================
```

In [ ]:

```

```

In [ ]:

```
# SVR can work with continuous Values instead of Classification which is SV
M.


### Intuition
```

- Support Vector Machines support linear and nonlinear regression that we can refer to as SVR
- Instead of trying to fit the largest possible street between two classes while limiting margin violations, SVR tries to fit as many instances as possible on the street while limiting margin violations.
- The width of the street is controlled by a hyper parameter Epsilon.

- SVR performs linear regression in a higher (dimensional space).
- We can think of SVR as if each data point in the training represents it's own dimension. When you evaluate your kernel between a test point and a point in the training set the resulting value gives you the coordinate of your test point in that dimension.
- The vector we get when we evaluate the test point for all points in the training set, $\vec{k}$ is the representation of the test point in the higher dimensional space.
- Once you have that vector you the use it to perform a linear regression.

It requires a training set: $\tau = \{\vec{X}, \vec{Y}\}$ which covers the domain of interest and is accompanied by solutions on that domain.

The work of the SVM is to approximate the function we used to generate the training set

$F(\vec{X}) = \vec{Y}$

In [ ]:

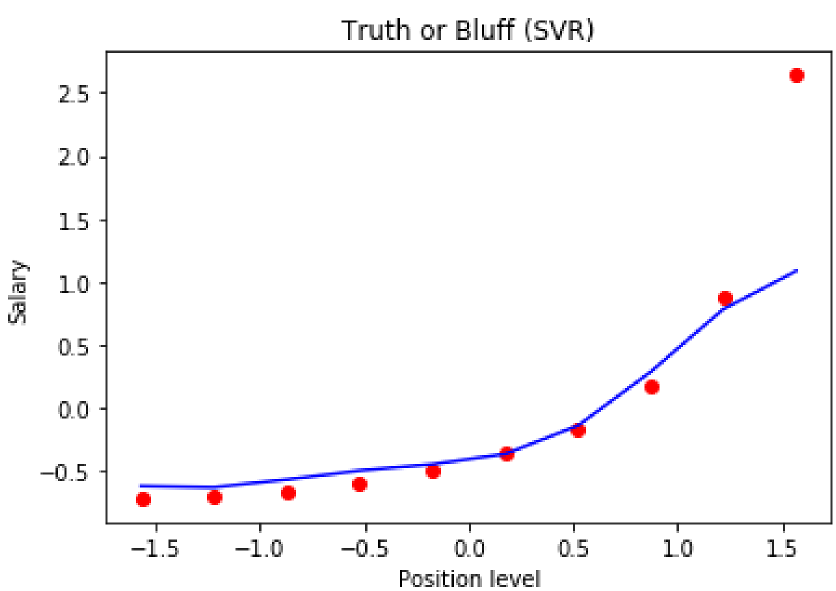
In [ ]:

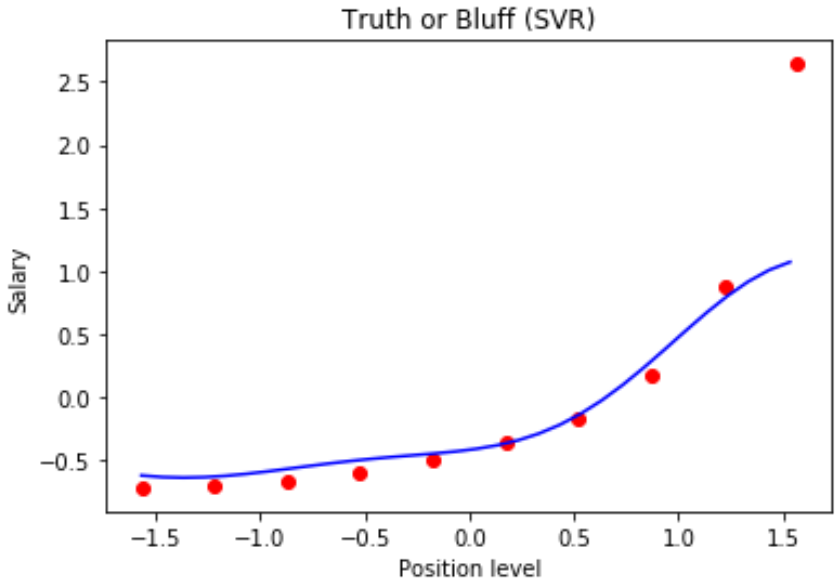
In [90]:

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
path = 'Machine Learning A-Z Template Folder/Part 2 - Regression/Section 7 - Support Vector Regression (SVR)/'
dataset = pd.read_csv(path + 'Position_Salaries.csv')
X = dataset.iloc[:, 1:2].values
y = dataset.iloc[:, 2].values

# Splitting the dataset into the Training set and Test set
"""from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)"""

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
sc_y = StandardScaler()
X = sc_X.fit_transform(X)
y = sc_y.fit_transform(y.reshape(len(y), 1))   #* needs to have a 2-D input.

# Fitting the SVR to the dataset
from sklearn.svm import SVR    #* This doesn't apply Feature Scaling automatically.
## Common "Kernels": Linear, Polynomial, Gaussian. Default: 'rbf' (Gaussian)
regressor = SVR(kernel= 'rbf')
regressor.fit(X, y)


# Predicting a new result
y_pred = sc_y.inverse_transform(regressor.predict(sc_X.transform([[6.5]])))
#* input needs transformation
#y_pred = regressor.predict(sc_X.transform(np.array([[6.5]])))
print(y_pred)

# Visualising the SVR results
plt.scatter(X, y, color = 'red')
plt.plot(X, regressor.predict(X), color = 'blue')
plt.title('Truth or Bluff (SVR)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()

# Visualising the SVR results (for 'higher resolution' and smoother curve)
X_grid = np.arange(min(X), max(X), 0.1)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X, y, color = 'red')
```

```
plt.plot(X_grid, regressor.predict(X_grid), color = 'blue')
plt.title('Truth or Bluff (SVR)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```

```
[170370.0204065]

/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:761: DataConversionWarning: A column-vector y was passed when
a 1d array was expected. Please change the shape of y to (n_samp
les, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```





In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
#====================================
# Section 8: Decision Tree Regression
#====================================


#* CART :  Classification and Regression Trees
```

In [ ]:

```
```

In [ ]:

```
## "classification" decision trees (also known as classification trees):
# classification criterion as 'entropy'. This criterion is known as the impu
rity measure.
# In classification, entropy is the most common impurity measure or splittin
g criteria.

# entropy is 0 if all samples at a node belong to the same class, and the "e
ntropy" is maximal if
    # we have a uniform class distribution.


## To use a decision tree for "regression", we need an impurity metric that
 is suitable for continuous variables,
    # so we define the impurity measure using the "weighted mean squared err
or (MSE)" of the children nodes.
# The standard ID3 algorithm can be used to construct a decision tree for re
gression by replacing
    # Information Gain with "Standard Deviation Reduction".

#* Is the split increasing the amount of information we already have.
#* Predicted (new) y value is the average of y values in that particular spl
it.


'''
In a regression tree: since the target variable does not have classes, we fi
t a regression model to the target
variable using each of the independent variables. Then for each independent
 variable, the data is split at
several split points. At each split point, the "error" between the predicted
value and the actual values is
squared to get a "Sum of Squared Errors (SSE)". The split point errors acros
s the variables are compared and
the variable/point yielding the lowest SSE is chosen as the root node/split
 point. This process is recursively
continued.
'''


# "Random Forest" is also useful as it usually has a better generalization p
erformance than an
# individual decision tree due to randomness, which helps to decrease the mo
del's variance.
# It is also less sensitive to outliers in the dataset and doesn't require m
uch parameter tuning.



#* Start at largest Information Gain (IG), repeat the splitting procedure at
each child node until the leaves are pure.
```

```
#* above process can easily lead to overfitting and very deep trees. Thus, w
e want to prune the tree (by limiting depth).



#* Regression trees are needed when the response variable is numeric or cont
inuous.
```

In [ ]:

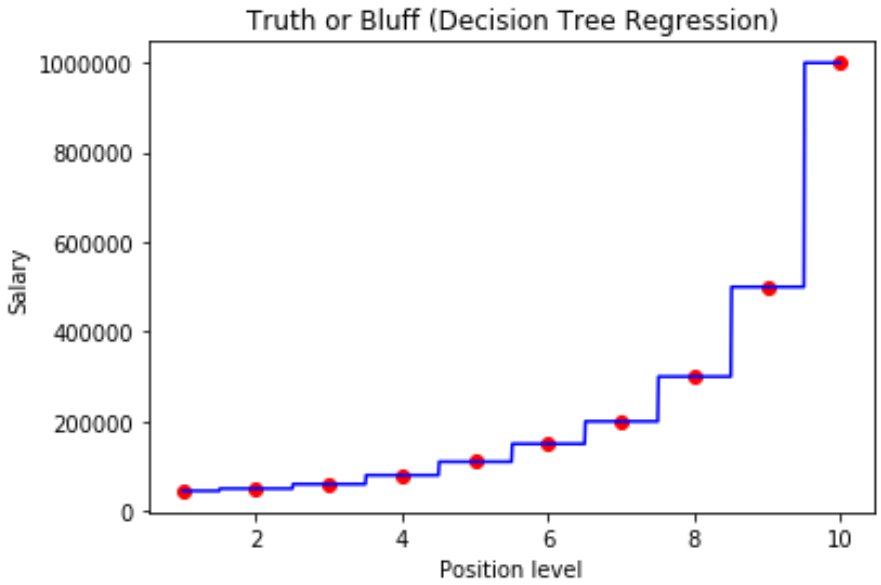
```

```

In [94]:

```
##** Graphs are interesting here.

## ** don't need Feature Scaling, as Decision Tree Regression models are bas
ed on conditions and not on euclidean distance.

# Decision Tree Regression:

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
path = 'Machine Learning A-Z Template Folder/Part 2 - Regression/Section 8 -
Decision Tree Regression/'
dataset = pd.read_csv(path + 'Position_Salaries.csv')
X = dataset.iloc[:, 1:2].values
y = dataset.iloc[:, 2].values

# Splitting the dataset into the Training set and Test set
"""from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, r
andom_state = 0)"""

# Feature Scaling
"""from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train)"""

# Fitting the Decision Tree Regression Model to the dataset
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state = 0)
regressor.fit(X, y)




# Predicting a new result
y_pred = regressor.predict([[6.5]])
print(y_pred)

# # Visualising the Decision Tree Regression results
# plt.scatter(X, y, color = 'red')
# plt.plot(X, regressor.predict(X), color = 'blue')
# plt.title('Truth or Bluff (Decision Tree Regression)')
# plt.xlabel('Position level')
```

```
# plt.ylabel('Salary')
# plt.show()
##** Above graph doesn't give the real shape of a Decision Tree Regression.
   # since Decision Tree Regression takes average of y values within a spli
t, the line in each interval should be straight.
# Its because of the way we plot, we plot for the 10 points and then it join
ts them with straight lines.
# cuz its a non linear, non continous regression model.

# Visualising the Decision Tree Regression results (for higher resolution an
d smoother curve){preffered here}
X_grid = np.arange(min(X), max(X), 0.01)  #* higher the resolution, more ver
tical the lines.
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X, y, color = 'red')
plt.plot(X_grid, regressor.predict(X_grid), color = 'blue')
plt.title('Truth or Bluff (Decision Tree Regression)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
## ** each horizontal line (separated by vertical lines) is an interval.
```

```
[150000.]
```



In [ ]:

In [ ]:

In [ ]:

```
#===================================
# Section 9: Random Forest Regression
#===================================
```

In [ ]:

```

```

In [95]:

```
#* Ensemble learning: is an ML paradigm where multiple learners are trained
 to solve the same problem. In
    # contrast to ordinary machine learning approaches which try to learn on
e hypothesis from training data,
    # ensemble methods try to construct a set of hypotheses and combine them
to use.
#* Ensemble methods use multiple learning algorithms to obtain better predic
tive performance than could be
    # obtained from any of the constituent learning algorithms alone. eg: 'R
andom Forest'
```

In [96]:

```
## Random Forest Intuition:
```

STEP 1: Pick at random K data points from the Training set.

STEP 2: Build the Decision Tree associated to these K data points.

STEP 3: Choose the number Ntree of trees you want to build and repeat STEPS 1 & 2

STEP 4: For a new data point, make each one of your Ntree trees predict the value of Y to for the data point in question, and assign the new data point the average across all of the predicted Y values.

In [ ]:

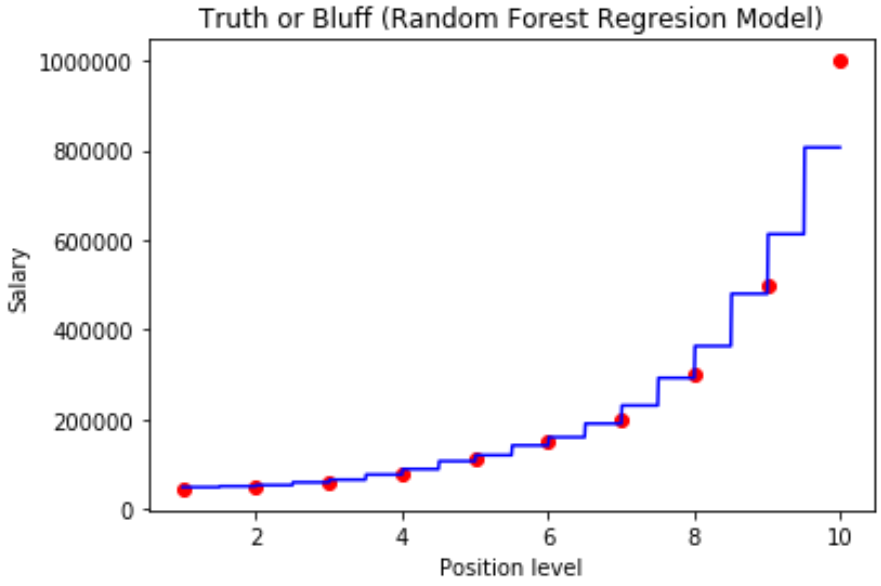
```

```

In [104]:

```
## * "Random Forest Regresion": Non Linear Non continuous Ensemble Regressio
n model.

# Random Forest Regresion:

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
path = 'Machine Learning A-Z Template Folder/Part 2 - Regression/Section 9 -
Random Forest Regression/'
dataset = pd.read_csv(path + 'Position_Salaries.csv')
X = dataset.iloc[:, 1:2].values
y = dataset.iloc[:, 2].values

# Splitting the dataset into the Training set and Test set
"""from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, r
andom_state = 0)"""

# Feature Scaling
"""from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train)"""

# Fitting the Random Forest Regresion Model to the dataset
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators = 300, random_state = 0)   #*
default is 10 trees
regressor.fit(X, y)




# Predicting a new result
y_pred = regressor.predict([[6.5]])
print(y_pred)

# Visualising the Random Forest Regresion results (for higher resolution and
smoother curve)
X_grid = np.arange(min(X), max(X), 0.01)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X, y, color = 'red')
plt.plot(X_grid, regressor.predict(X_grid), color = 'blue')
plt.title('Truth or Bluff (Random Forest Regresion Model)')
```

```
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
## ** each horizontal line (separated by vertical lines) is an interval.
```

```
[160333.33333333]
```



In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```
#============================================================
# Section 10: Evaluating Regression Models Performance
#============================================================
```

In [ ]:

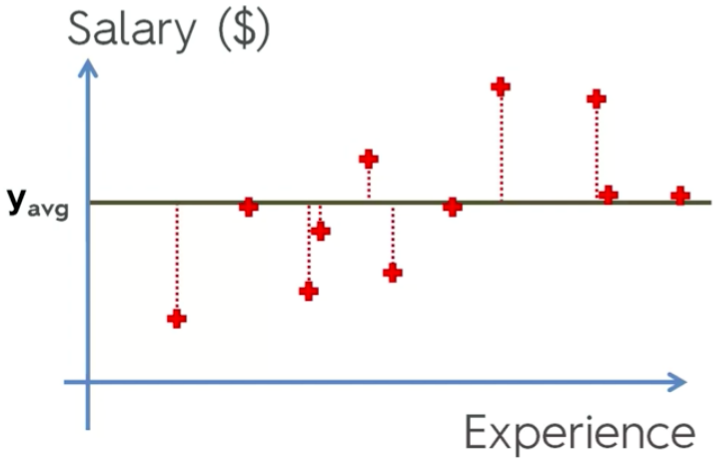
```

```

In [107]:

```
## 'Residual' sum of squares, 'Total' sum of squares
```

R Squared

Simple Linear Regression:



$$SS_{res} = SUM\ (y_i - \hat{y_i})^2$$

$$SS_{tot} = SUM\ (y_i - y_{avg})^2$$

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

Adjusted R^2

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

R^2 – Goodness of fit
(greater is better)

$y = b_0 + b_1 * x_1$

$y = b_0 + b_1 * x_1 + b_2 * x_2$

Problem:

$+ b_3 * x_3$

SS_{res} -> Min → R^2 will never decrease

In [110]:

```
### with Multiple Independent Variables:
#=======================================


## ** After adding new variable, the R-Squared (R^2) will never decrease.

## Either the added variable will help minimize 'Residual' sum of squares. T
he system will give some value to the
    # coefficient of new variable, to decrease 'Residual' sum of squares som
ehow.



### * Solution: "ADJUSTED R-SQUARED"

## Adjusted R-Squared has a ' penalizing factor'. It penalizes you for addin
g variables that don't help your model.
```

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

$$\text{Adj } R^2 = 1 - (1 - R^2)\frac{n - 1}{n - p - 1}$$

p - number of regressors

n - sample size

In [111]:

```
##** In 'Backward Eliminaton' before: we should look at the 'Adjusted R-Squa
red' value also. (along with P-Value).
```

In [112]:

```
## Interpreting Linear Regression Coefficients:
#============================================



## Always consider the units for each coefficient, while looking at their va
lues.

## For every unit change in 'independent variable',
    # the dependend variable will change by corresponding 'coefficient valu
e' amount.
```

Regression

Regression Model	Pros	Cons
Linear Regression	Works on any size of dataset, gives informations about relevance of features	The Linear Regression Assumptions
Polynomial Regression	Works on any size of dataset, works very well on non linear problems	Need to choose the right polynomial degree for a good bias/variance tradeoff
SVR	Easily adaptable, works very well on non linear problems, not biased by outliers	Compulsory to apply feature scaling, not well known, more difficult to understand
Decision Tree Regression	Interpretability, no need for feature scaling, works on both linear / nonlinear problems	Poor results on too small datasets, overfitting can easily occur
Random Forest Regression	Powerful and accurate, good performance on many problems, including non linear	No interpretability, overfitting can easily occur, need to choose the number of trees

Machine Learning A-Z © SuperDataScience

In [113]:

```
### There is a "Regularization PDF" in folder.
```

In [ ]:

```

```

In [ ]:

```
```

In [114]:

```
#####################################
# Section 11: Part 3: CLASSIFICATION
#####################################
```

In [ ]:

```
```

In [ ]:

```
```

=> Classification models include linear models like Logistic Regression, SVM, and nonlinear ones like K-NN, Kernel SVM and Random Forests.

- Logistic Regression
- K-Nearest Neighbors (K-NN)
- Support Vector Machine (SVM)
- Kernel SVM
- Naive Bayes
- Decision Tree Classification
- Random Forest Classification

In [116]:

```
### Classification models include linear models like Logistic Regression, SV
M, and nonlinear ones like K-NN,
    # Kernel SVM and Random Forests.

  # Logistic Regression
  # K-Nearest Neighbors (K-NN)
  # Support Vector Machine (SVM)
  # Kernel SVM
  # Naive Bayes
  # Decision Tree Classification
  # Random Forest Classification
```

In [ ]:

```
```

In [ ]:

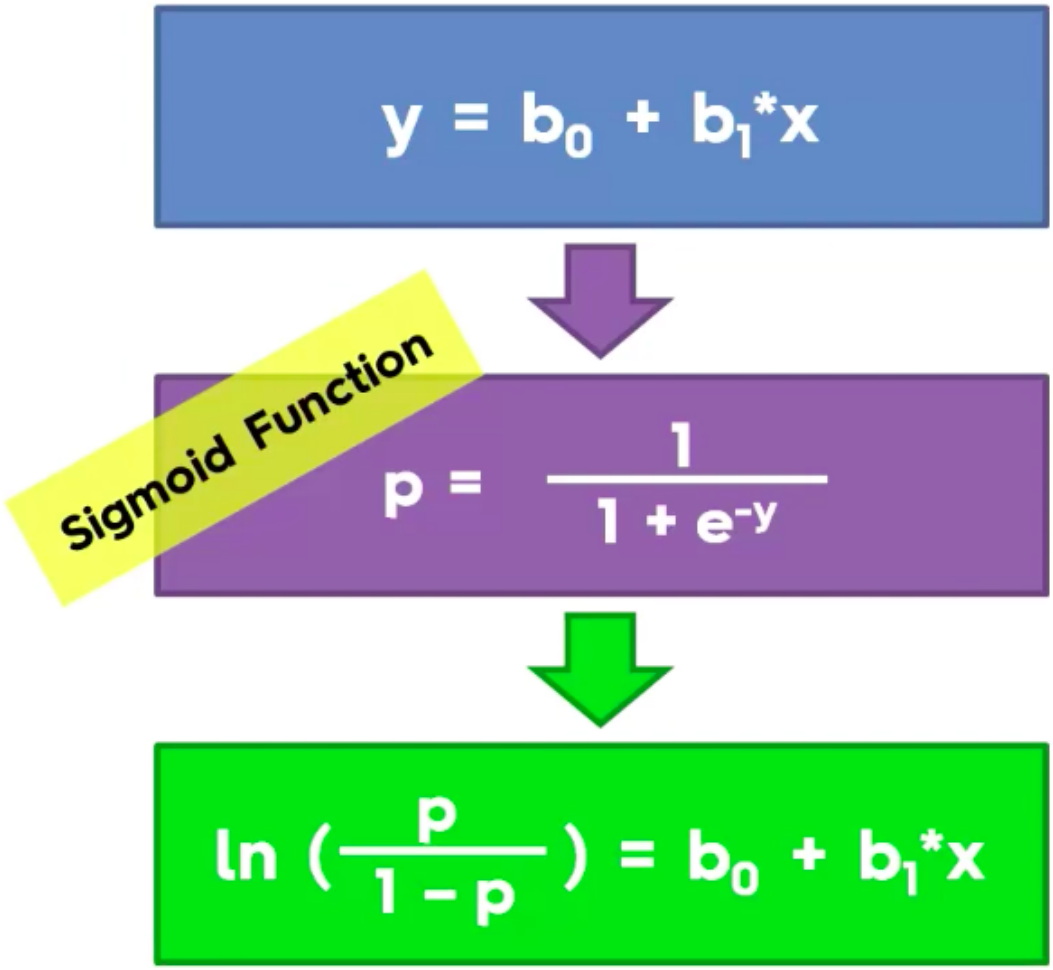
```
#================================
# Section 12: Logistic Regression
#================================
```

In [ ]:

```

```

In [ ]:

```

```



In [ ]:

```

```

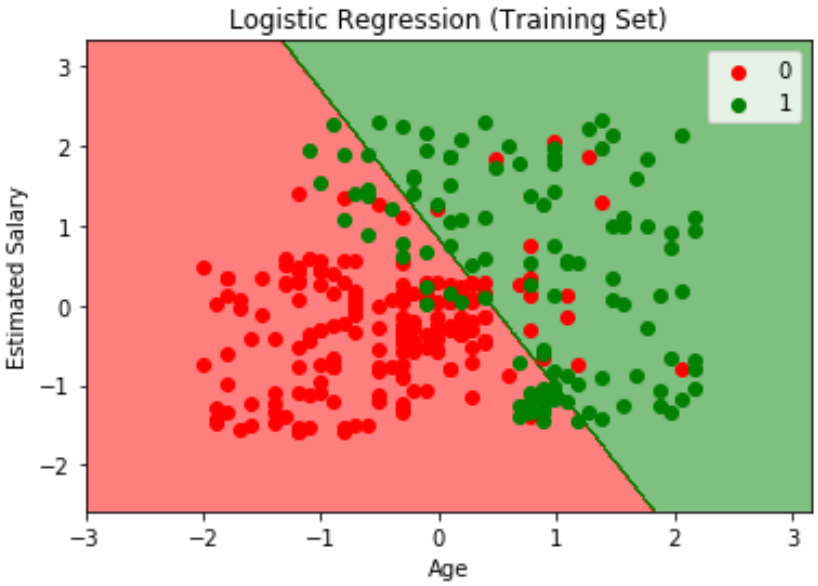
In [132]:

```
# Logistic Regresion:

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
path = 'Machine Learning A-Z Template Folder/Part 3 - Classification/Section
14 - Logistic Regression/'
dataset = pd.read_csv(path + 'Social_Network_Ads.csv')
# only using age and estimated salary:
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split  #*
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)


#######

# fitting Logistic Regression to the training set:
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state=0)
classifier.fit(X_train, y_train)

# predicting the test set results:
y_pred = classifier.predict(X_test)
print(y_pred)

# making the Confusion Matrix:
from sklearn.metrics import confusion_matrix   #* here importing a Function
 and not Class (no capitals in name)
cm = confusion_matrix(y_true=y_test, y_pred=y_pred)
print(cm)

##* visualizing the Training set results:
from matplotlib.colors import ListedColormap  # to colorize all the datapoin
ts
X_set, y_set = X_train, y_train
# prepare grid with all the pixel points:
 # 'ravel': Return a contiguous flattened array.
```

```
 # 'meshgrid': Return coordinate matrices from coordinate vectors.
X1, X2 = np.meshgrid(np.arange(start = X_set[:,0].min() - 1, stop = X_set[:,
0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:,1].min() - 1, stop = X_set[:,
1].max() + 1, step = 0.01))
# apply classifier on all the pixel observation points and make countour b/w
the two regions:
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.50, cmap = ListedColormap(('red', 'green')))    #* alp
ha is for Opaqueness.
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
# plot all the datapoints:
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],         ## * ???
                c = ListedColormap(('red', 'green'))(i), label = j)
    # c : color, sequence, or sequence of color,
plt.title('Logistic Regression (Training Set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()


##* visualizing the Test set results:
from matplotlib.colors import ListedColormap  # to colorize all the datapoin
ts
X_set, y_set = X_test, y_test
# prepare grid with all the pixel points:
 # 'ravel': Return a contiguous flattened array.
X1, X2 = np.meshgrid(np.arange(start = X_set[:,0].min() - 1, stop = X_set[:,
0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:,1].min() - 1, stop = X_set[:,
1].max() + 1, step = 0.01))
# apply classifier on all the pixel observation points and make countour b/w
the two regions:
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.80, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

# plot all the datapoints:
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Test Set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
## ** Decision boundary is a 'Straight Line' as Logistic regression classifi
er is a 'Linear Classifier'.

## ** Unlike Linear Regression, the dependent variable is categorical, which
is why it's considered a classification algorithm.


# * https://stats.stackexchange.com/questions/93569/why-is-logistic-regressi
on-a-linear-classifier
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logi
stic.py:433: FutureWarning: Default solver will be changed to 'l
bfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.

[0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0
1 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0
0 1 0 0 0
 0 0 1 0 1 1 1 1 0 0 1 1 0 1 0 0 0 1 0 0 0 0 0 0 1 1]
[[65  3]
 [ 8 24]]
```




```
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
```

Logistic regression is linear in the sense that the predictions can be written as

$$\hat{p} = \frac{1}{1 + e^{-\hat{\mu}}}, \text{ where } \hat{\mu} = \hat{\theta} \cdot x.$$

Thus, the prediction can be written in terms of $\hat{\mu}$, which is a linear function of x. (More precisely, the predicted log-odds is a linear function of x.)

Conversely, there is no way to summarize the output of a neural network in terms of a linear function of x, and that is why neural networks are called non-linear.

Also, for logistic regression, the decision boundary $\{x : \hat{p} = 0.5\}$ is linear: it's the solution to $\hat{\theta} \cdot x = 0$. The decision boundary of a neural network is in general not linear.

share cite improve this answer

answered Apr 12 '14 at 19:45

As Stefan Wagner notes, the decision boundary for a logistic classifier is linear. (The classifier needs the inputs to be linearly separable.) I wanted to expand on the math for this in case it's not obvious.

The decision boundary is the set of x such that

$$\frac{1}{1 + e^{-\theta \cdot x}} = 0.5$$

A little bit of algebra shows that this is equivalent to

$$1 = e^{-\theta \cdot x}$$

and, taking the natural log of both sides,

$$0 = -\theta \cdot x = -\sum_{i=0}^{n} \theta_i x_i$$

so the decision boundary is linear.

In [ ]:

In [ ]:

```
#=======================
# Classification Template
#=======================




# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
path = 'Machine Learning A-Z Template Folder/Part 3 - Classification/Section
14 - Logistic Regression/'
dataset = pd.read_csv(path + 'Social_Network_Ads.csv')
# only using age and estimated salary:
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split       # *
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)


#######

# fitting the Classifier to the Training set:
 # create your classifier here.


# predicting the Test set results:
y_pred = classifier.predict(X_test)
print(y_pred)

# Making the Confusion Matrix:
from sklearn.metrics import confusion_matrix   #* here importing a Function
 and not Class (no capitals in name)
cm = confusion_matrix(y_true=y_test, y_pred=y_pred)
print(cm)

##* visualizing the Training set results:
from matplotlib.colors import ListedColormap  # to colorize all the datapoin
```

```
ts
X_set, y_set = X_train, y_train
# prepare grid with all the pixel points:
 # 'ravel': Return a contiguous flattened array.
 # 'meshgrid': Return coordinate matrices from coordinate vectors.
X1, X2 = np.meshgrid(np.arange(start = X_set[:,0].min() - 1, stop = X_set[:,
0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:,1].min() - 1, stop = X_set[:,
1].max() + 1, step = 0.01))
# apply classifier on all the pixel observation points and make countour b/w
the two regions:
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.50, cmap = ListedColormap(('red', 'green')))    #* alp
ha is for Opaqueness.
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
# plot all the datapoints:
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],          ## * ???
                c = ListedColormap(('red', 'green'))(i), label = j)
    # c : color, sequence, or sequence of color,
plt.title('Logistic Regression (Training Set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()


##* visualizing the Test set results:
from matplotlib.colors import ListedColormap  # to colorize all the datapoin
ts
X_set, y_set = X_test, y_test
# prepare grid with all the pixel points:
 # 'ravel': Return a contiguous flattened array.
X1, X2 = np.meshgrid(np.arange(start = X_set[:,0].min() - 1, stop = X_set[:,
0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:,1].min() - 1, stop = X_set[:,
1].max() + 1, step = 0.01))
# apply classifier on all the pixel observation points and make countour b/w
the two regions:
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.80, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

# plot all the datapoints:
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Test Set)')
plt.xlabel('Age')
```

```
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

In [ ]:

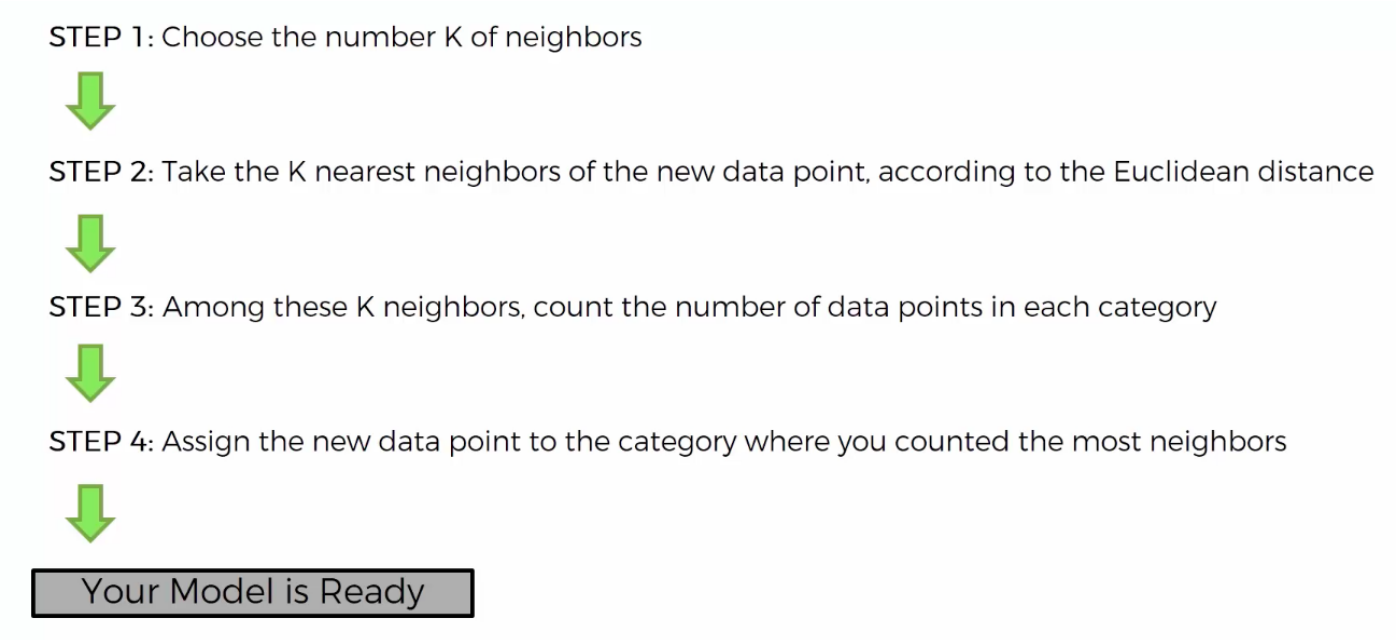
```

```

In [ ]:

```
#============================================
# Section 13 (15): K-Nearest Neighbors (K-NN)
#============================================
```

In [ ]:

```

```

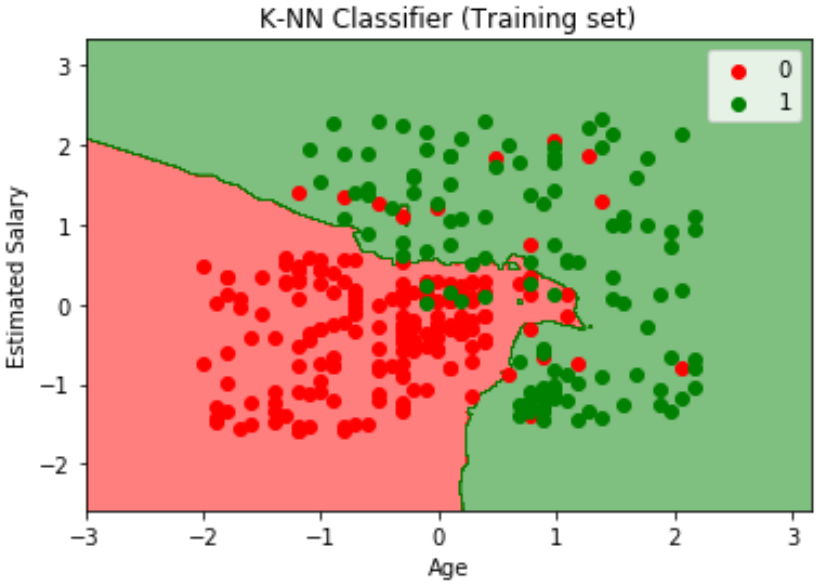


In [137]:

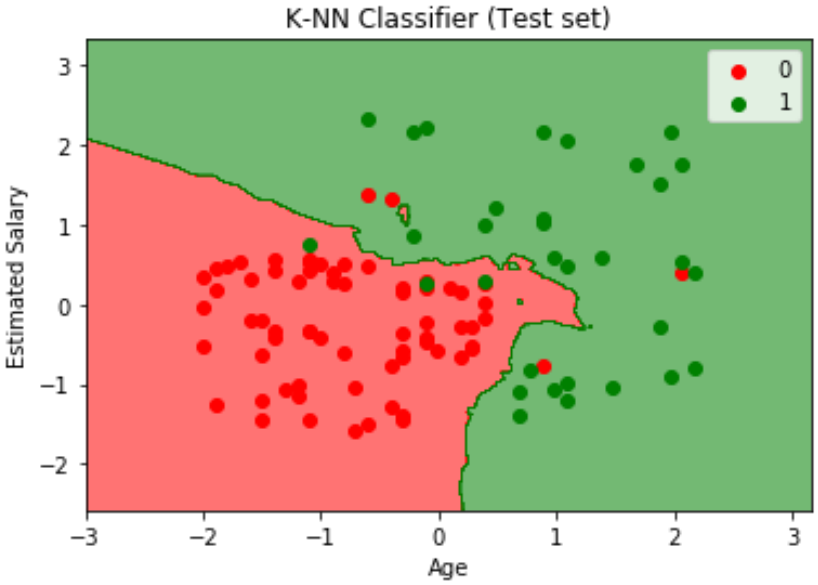
```
## K-Nearest Neighbours (K-NN):


# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
path = 'Machine Learning A-Z Template Folder/Part 3 - Classification/Section
15 - K-Nearest Neighbors (K-NN)/'
dataset = pd.read_csv(path + 'Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)


#######

# Fitting K-NN classifier to the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p =
2)
classifier.fit(X_train, y_train)



# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(y_pred)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)


# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
```

```
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set
[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set
[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.50, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('K-NN Classifier (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set
[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set
[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.55, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('K-NN Classifier (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)

[0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0
1 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 1 1 1 0 0 1 0 0 1 0 1 0 1 0 0 0
0 1 0 0 1
 0 0 0 0 1 1 1 1 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 1 1 1]
[[64  4]
 [ 3 29]]

'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
```



```
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
```

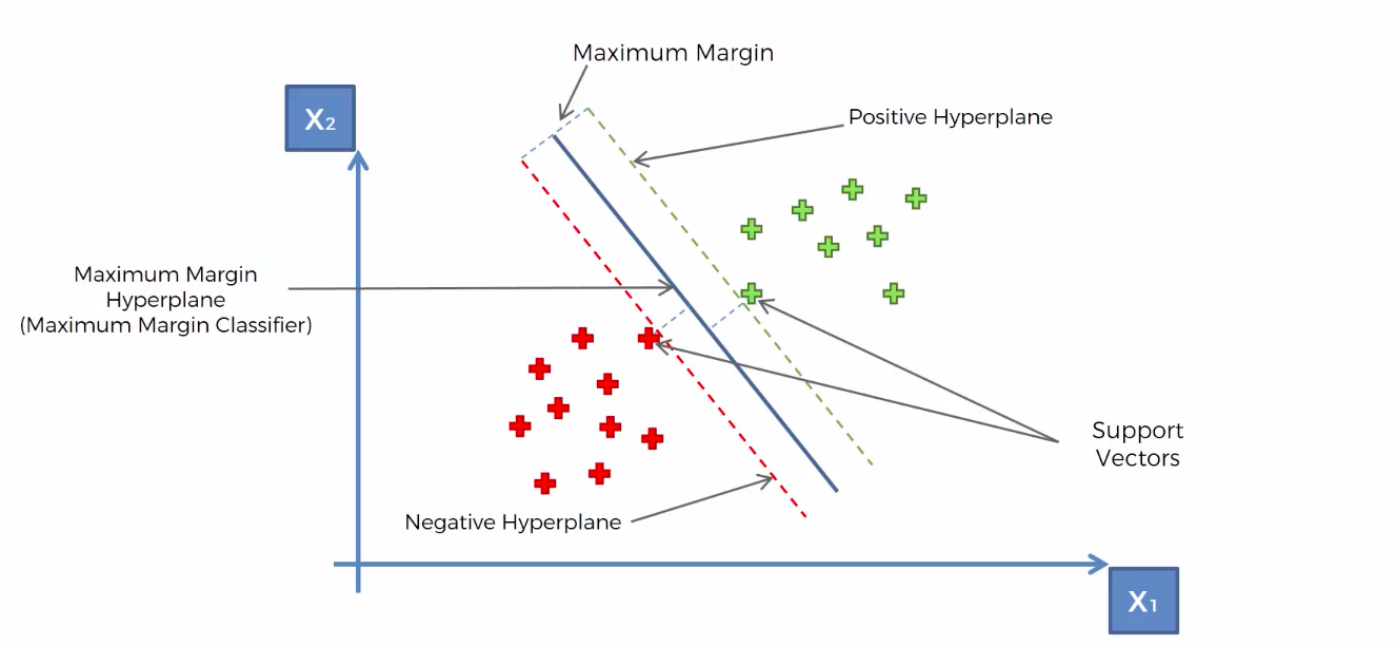


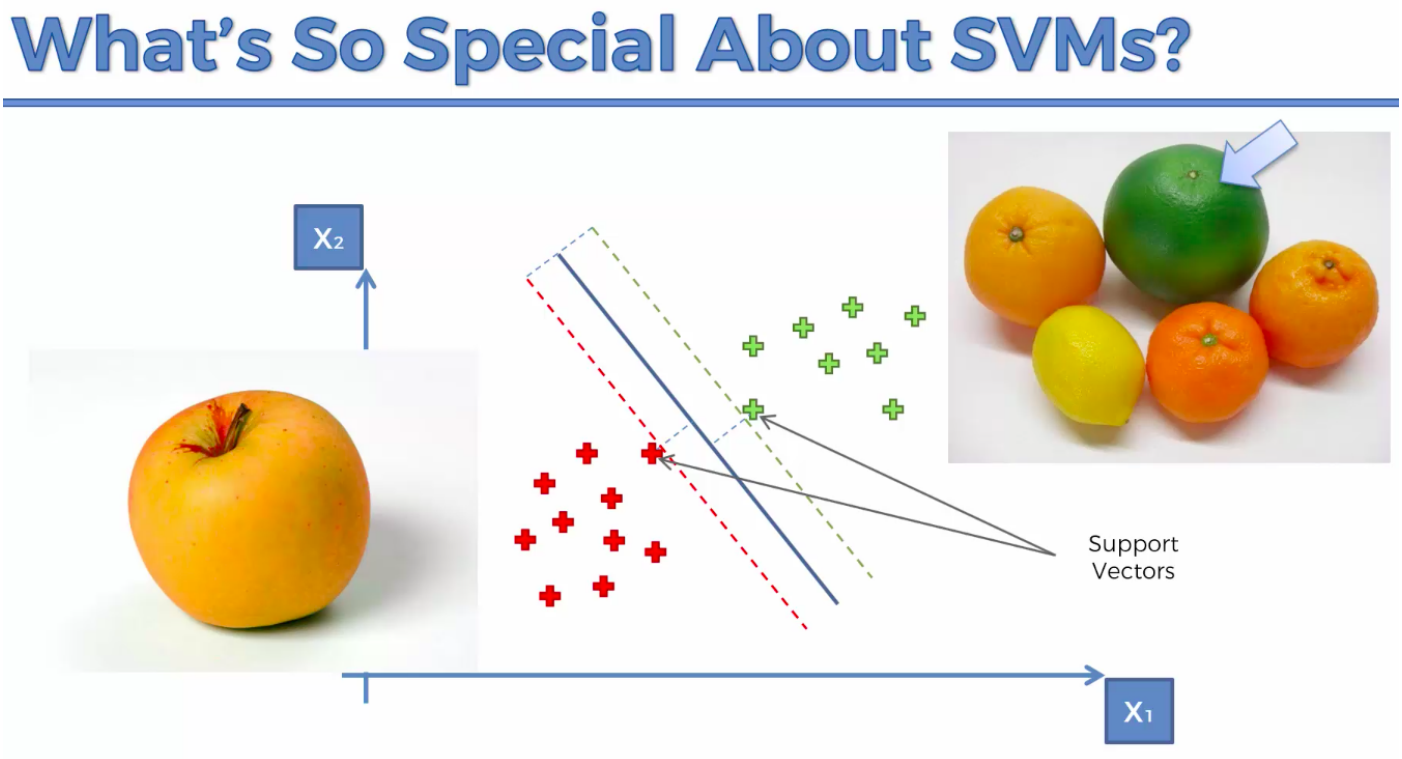
In [ ]:

In [ ]:

```
#============================================
# Section 14 (16): Support Vector Machine (SVM)
#============================================
```

In [ ]:


Maximum Margin
Positive Hyperplane
X2
Maximum Margin
Hyperplane
(Maximum Margin Classifier)
Support
Vectors
Negative Hyperplane
X1

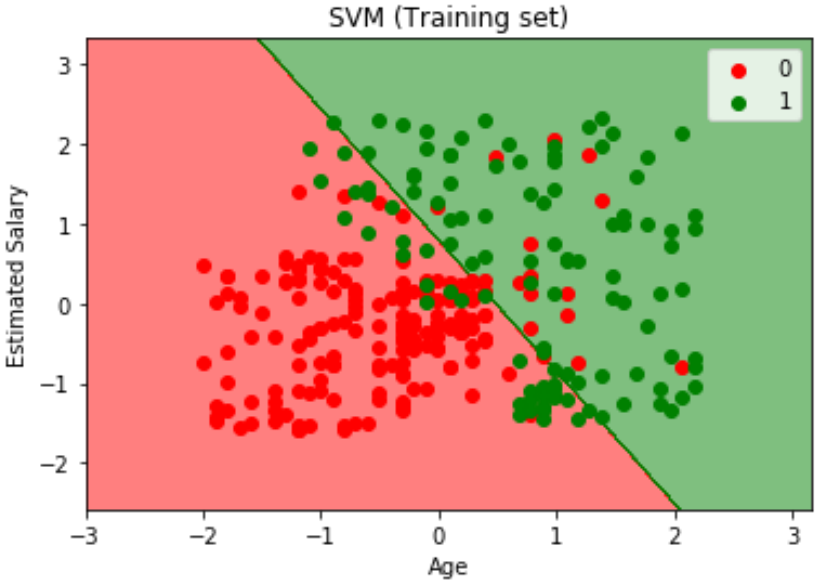

What's So Special About SVMs?
X2
Support
Vectors
X1

In [138]:

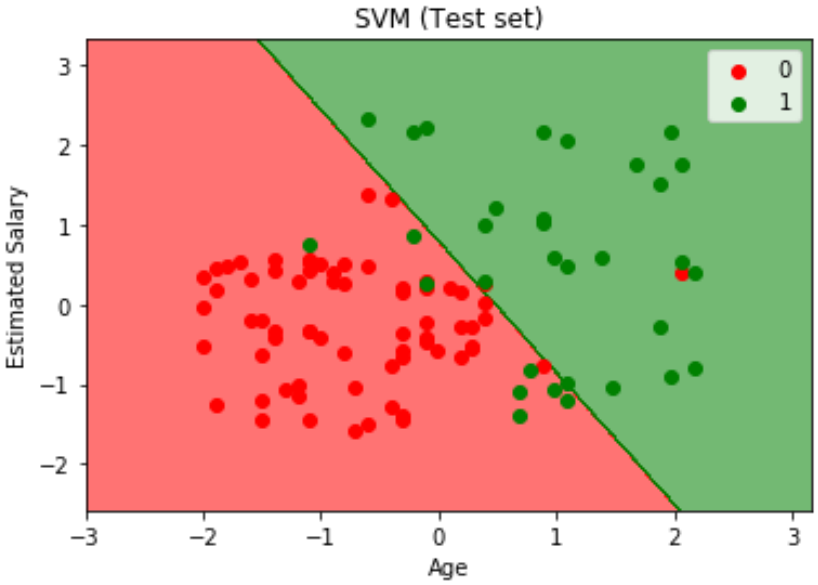
```
## Support Vector Machine (SVM):


# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
path = 'Machine Learning A-Z Template Folder/Part 3 - Classification/Section
16 - Support Vector Machine (SVM)/'
dataset = pd.read_csv(path + 'Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)


#######

# Fitting SVM to the Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_train, y_train)




# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(y_pred)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)


# Visualising the Training set results
from matplotlib.colors import ListedColormap
```

```
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set
[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set
[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.50, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('SVM (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set
[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set
[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.55, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('SVM (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)

[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0
1 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0
0 1 0 0 0
 0 0 1 0 1 1 1 1 0 0 1 1 0 1 0 0 0 1 0 0 0 0 0 0 1 1]
[[66  2]
 [ 8 24]]

'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
```



```
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
```



In [ ]:

In [ ]:

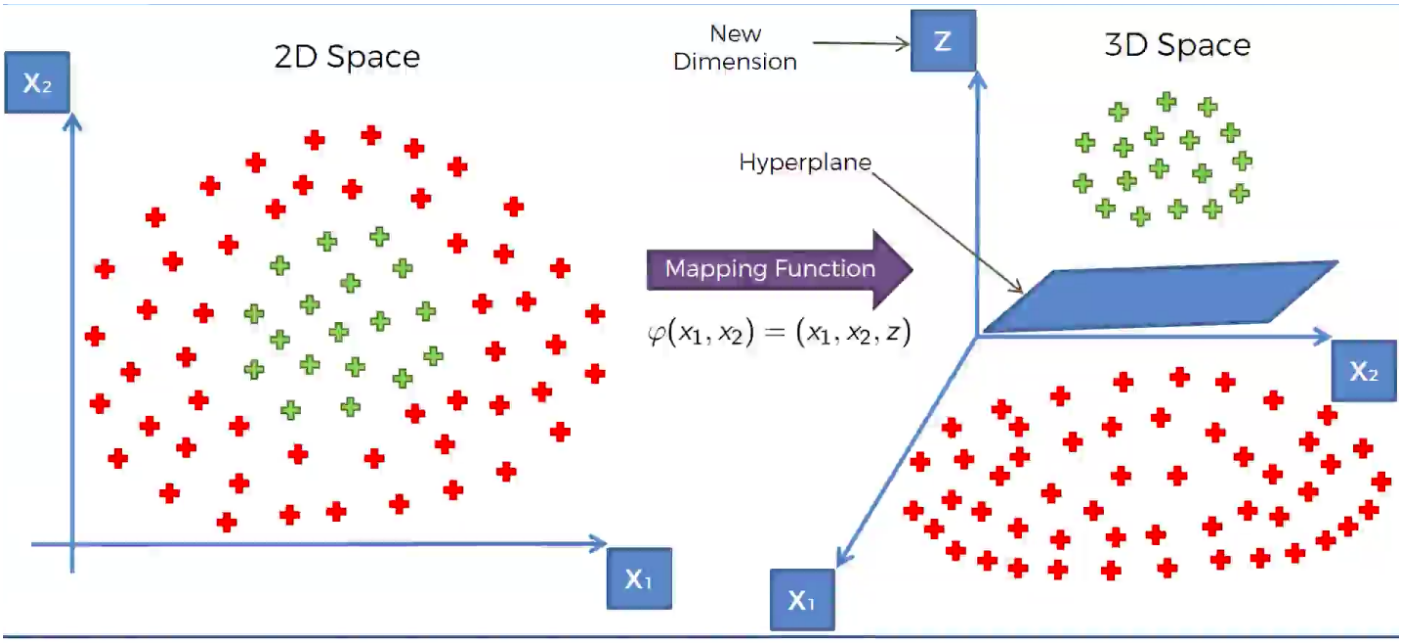
```
#============================
# Section 15 (17): Kernel SVM
#============================
```

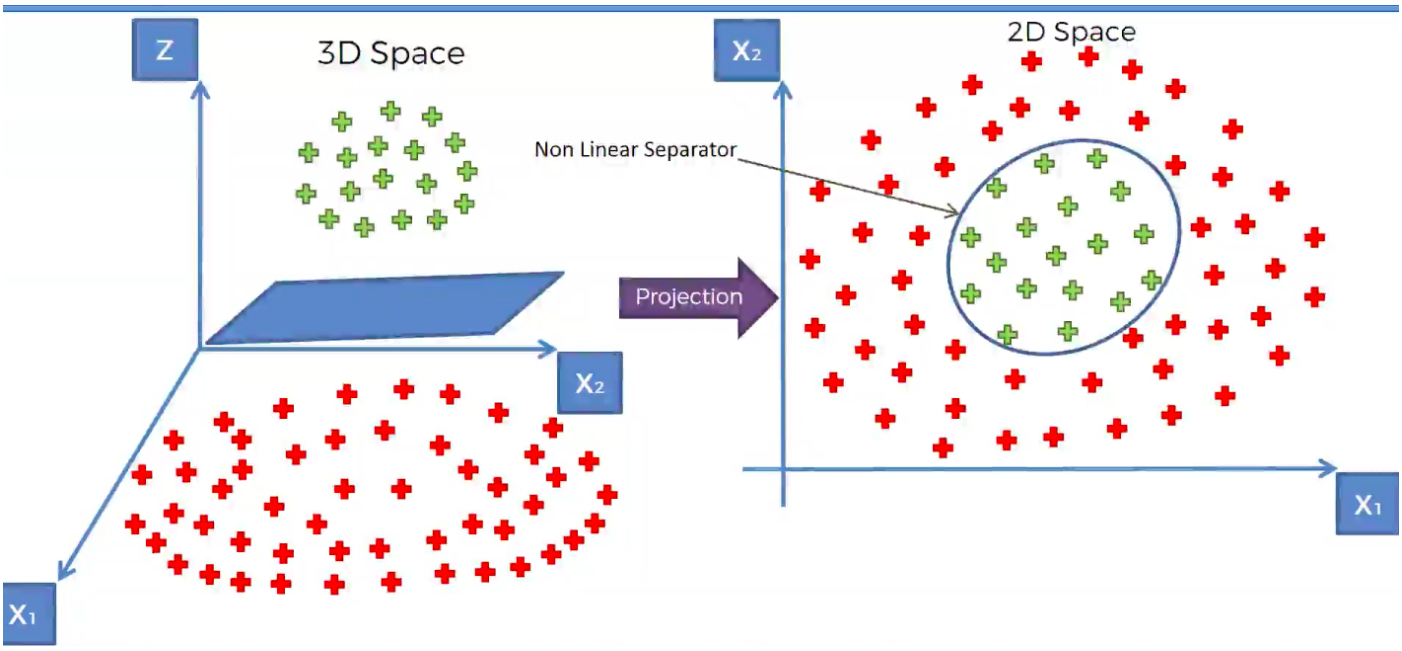
In [ ]:

```
```

In [ ]:

```
#### Mapping to a Higher Dimension:
```





In [ ]:

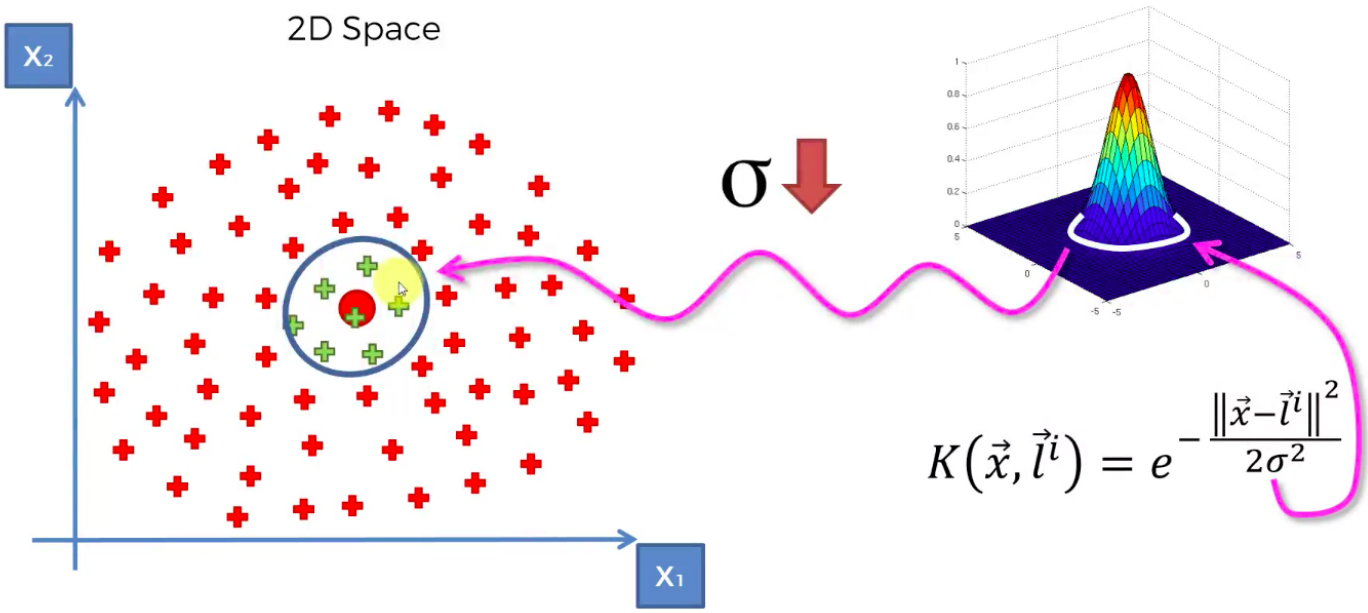
```
## ** Used when data points are not Linearly Separable

## ** Mapping to a Higher Dimensional Space can be highly compute-intensive.
```

In [ ]:

```
#### The Kernel Trick:
#====================
```

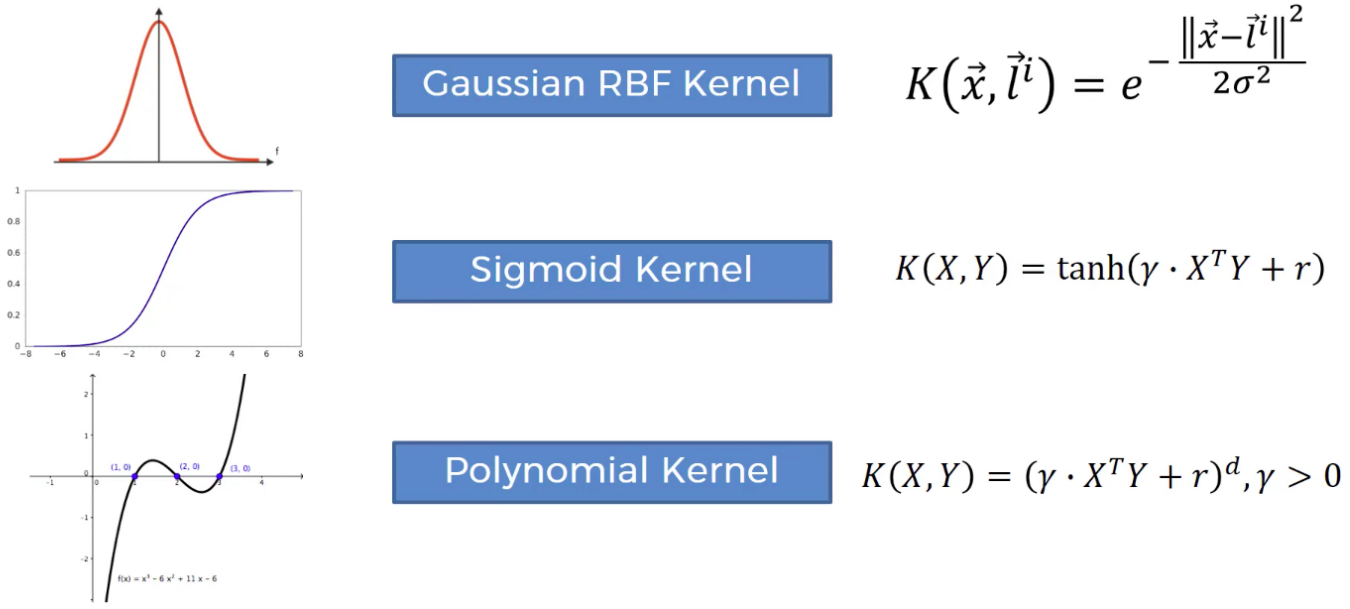
In [1]:

```
# Gaussian Radial Basis Function (RBF) kernel
```



In [2]:

```
### Types of Kernel Functions:
#============================
```



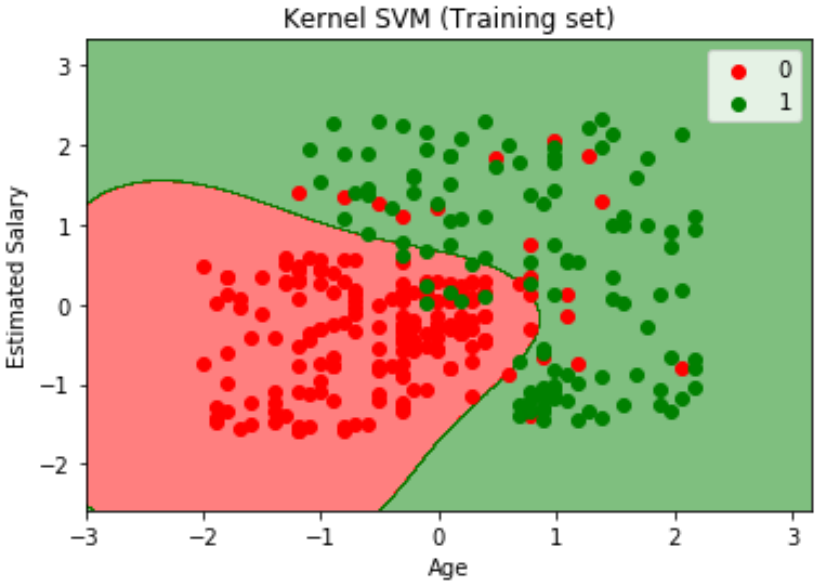
In [ ]:

In [4]:

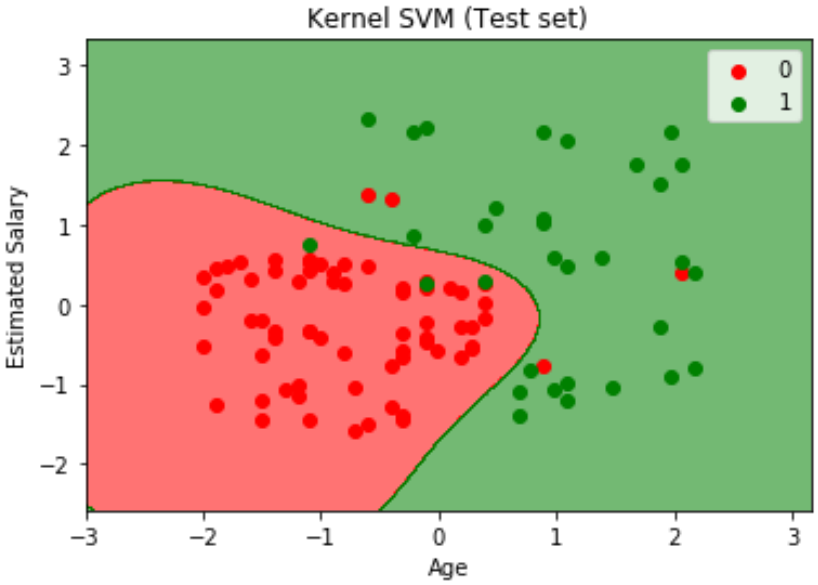
```
## Kernel SVM:

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
path = 'Machine Learning A-Z Template Folder/Part 3 - Classification/Section
17 - Kernel SVM/'
dataset = pd.read_csv(path + 'Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)


#######

# Fitting Kernel SVM to the Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', random_state = 0)
classifier.fit(X_train, y_train)




# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(y_pred)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)


# Visualising the Training set results
from matplotlib.colors import ListedColormap
```

```
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set
[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set
[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.50, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Kernel SVM (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set
[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set
[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.55, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Kernel SVM (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)


[0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0
1 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 1 1 1 0 0 1 0 0 1 0 1 0 1 0 0 0
0 1 0 0 1
 0 0 0 0 1 1 1 1 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 1 1 1]
[[64  4]
 [ 3 29]]

'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
```



```
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
```



In [ ]:

In [ ]:

```
#==============================
# Section 16 (18): Naive Bayes
#==============================
```

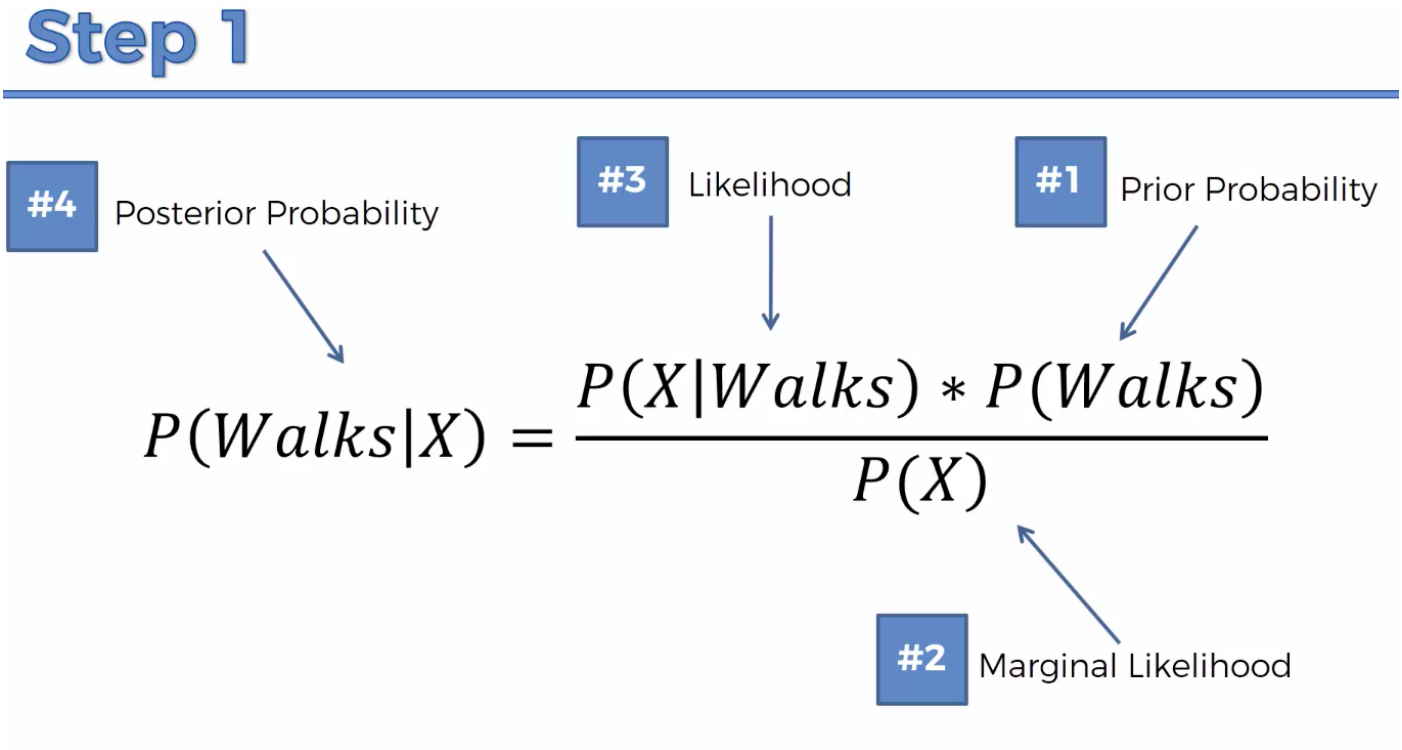
In [ ]:

In [6]:

```
# Naive Bayes is a probabilistic semi-supervised or "Supervised" "Classifica
tion" method

## ** Why Naive?: because of 'Independence assumption'. *The features might
 not be independent.

#  Another assumption made is: that all the predictors have an equal effect
 on the outcome.

## NB is a probabilistic classifier that makes classifications using the Max
imum A Posteriori decision rule in a
    # Bayesian setting.

# NB is based on Bayes Theorem. Using Bayes theorem, we can find the probabi
lity of A happening, given that B has
    # occurred. Here, B is the evidence and A is the hypothesis.



# Useful for: 'text classification', and a traditional solution for 'spam de
tection'.




## Ref: https://towardsdatascience.com/introduction-to-naive-bayes-classific
ation-4cffabb1ae54

# Ref: https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/
```

In [ ]:

```

```


Step 1
#4
Posterior Probability
#3
Likelihood
#1
Prior Probability
$P(Walks|X) = \frac{P(X|Walks) * P(Walks)}{P(X)}$
#2
Marginal Likelihood

can simply ignore that term, and instead just state that P(c_i | x_0, …, x_n) ∝ P(x_0, …, x_n | c_i) * P(c_i), where ∝ means "is proportional to". P(c_i) is simple to calculate; it is just the proportion of the data-set that falls in class i. P(x_0, …, x_n | c_i) is more difficult to compute. In order to simplify its computation, we make the assumption that x_0 through x_n are **conditionally independent** given c_i, which allows us to say that P(x_0, …, x_n | c_i) = P(x_0 | c_i) * P(x_1 | c_i) * … * P(x_n | c_i). This assumption is most likely not true — hence the name *naive* Bayes classifier, but the classifier nonetheless performs well in most situations. Therefore, our final

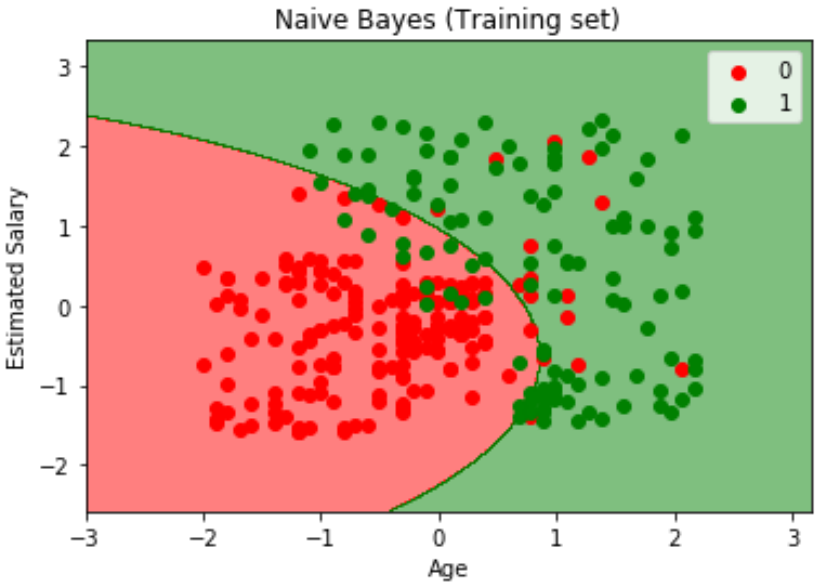
In [ ]:

In [7]:

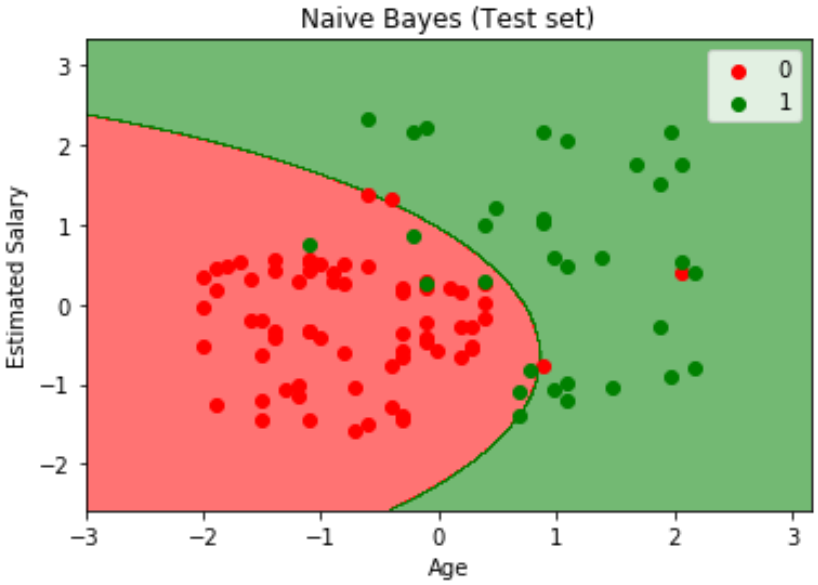
```
## Naive Bayes:


# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
path = 'Machine Learning A-Z Template Folder/Part 3 - Classification/Section
18 - Naive Bayes/'
dataset = pd.read_csv(path + 'Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)


#######

# Fitting Naive Bayes to the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()      #* no parameters/arguments
classifier.fit(X_train, y_train)







# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(y_pred)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set
[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set
[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.50, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Naive Bayes (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set
[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set
[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.55, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Naive Bayes (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.

[0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0
1 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0
0 1 0 0 0
 0 0 0 0 1 1 1 1 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 1 1 1]
[[65  3]
 [ 7 25]]
```



```
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
```



In [ ]:

In [ ]:

```
#============================================
# Section 17 (19): Decision Tree Classification
#============================================
```

In [ ]:

In [ ]:

```
## CART: Classifiction and Regression Trees


##* Basic idea: To maximize the number of (instances) some (one) category in
each region of split.

# DT: simple tool but not very powerful on their own.

# Uses (in combined form like random forest): facial recognition and some ga
mes.
```

In [ ]:

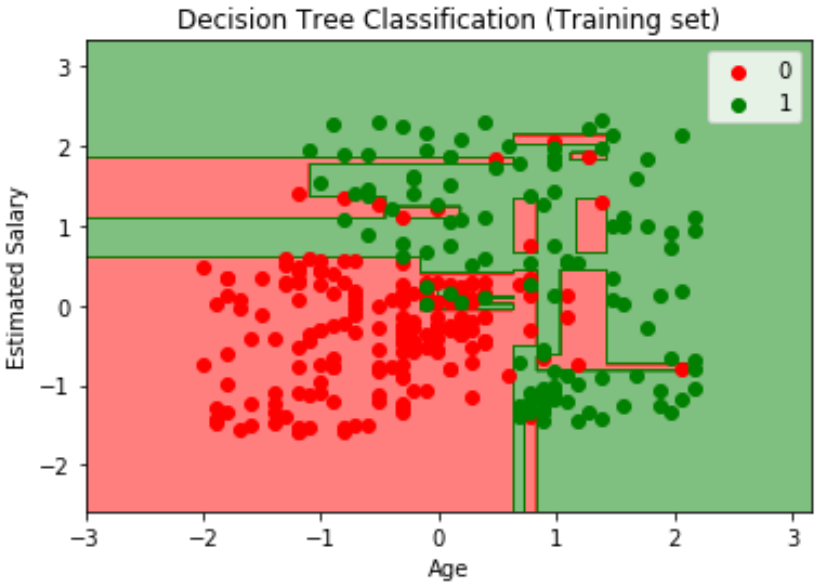
```

```

In [8]:

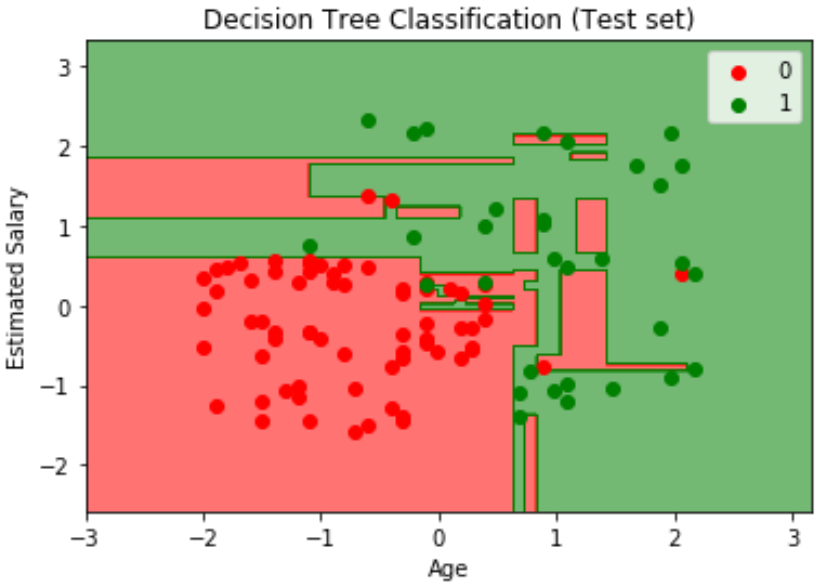
```
## Decision Tree Classification:


# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
path = 'Machine Learning A-Z Template Folder/Part 3 - Classification/Section
19 - Decision Tree Classification/'
dataset = pd.read_csv(path + 'Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 0)

# Feature Scaling    #* Not needed here as DT is not based on Euclidean Dist
ance.
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)


#######

# Fitting Decision Tree Classification to the Training set
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)




# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(y_pred)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set
[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set
[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.50, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Decision Tree Classification (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set
[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set
[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.55, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Decision Tree Classification (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.

[0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0
1 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 1 1 1 0 0 1 0 0 1 0 1 0 1 0 0 0
1 1 0 0 1
 0 0 0 0 1 1 1 1 0 0 1 0 0 1 1 0 0 1 0 0 0 1 0 1 1 1]
[[62  6]
 [ 3 29]]
```




```
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
```



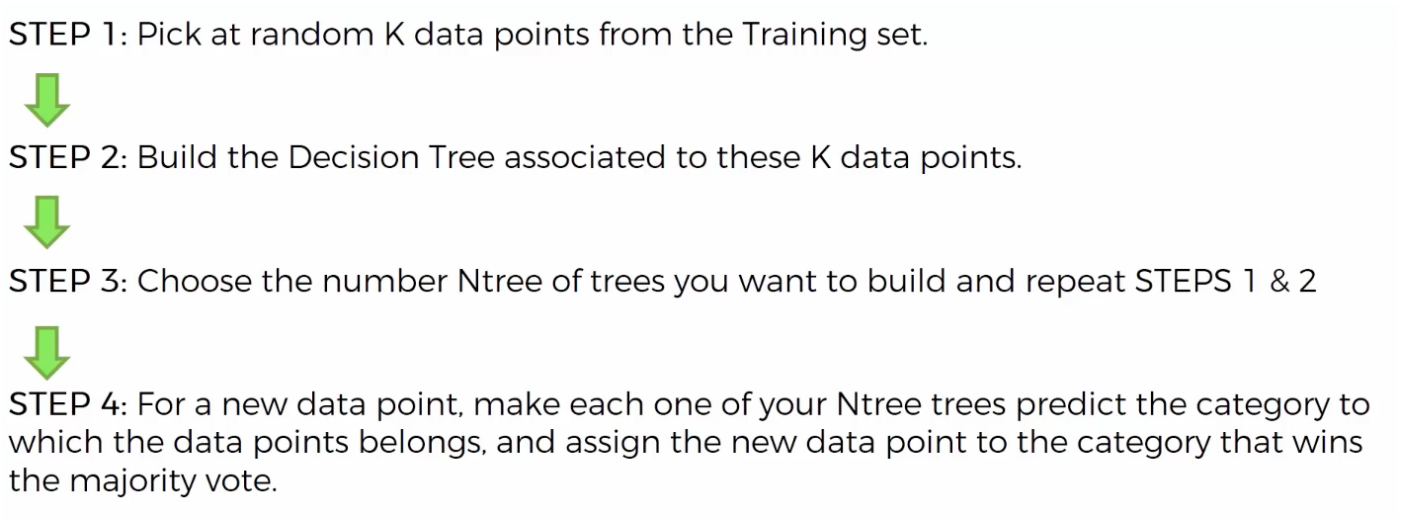
In [ ]:

In [ ]:

```
#=============================================
# Section 18 (20):  Random Forest Classification
#=============================================


# Ensemble learning
```

In [ ]:

In [ ]:


STEP 1: Pick at random K data points from the Training set.
STEP 2: Build the Decision Tree associated to these K data points.
STEP 3: Choose the number Ntree of trees you want to build and repeat STEPS 1 & 2
STEP 4: For a new data point, make each one of your Ntree trees predict the category to which the data points belongs, and assign the new data point to the category that wins the majority vote.

In [ ]:

In [11]:

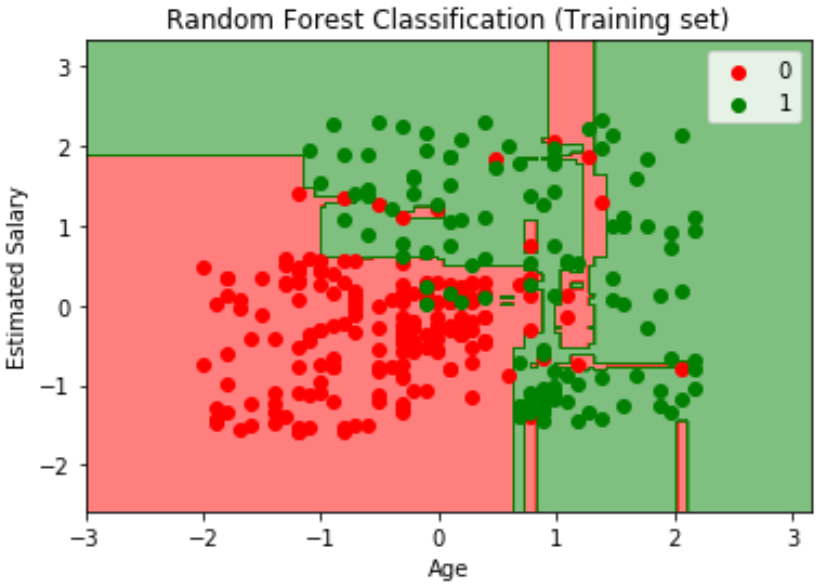
```
## Random Forest Classification:


# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
path = 'Machine Learning A-Z Template Folder/Part 3 - Classification/Section
20 - Random Forest Classification/'
dataset = pd.read_csv(path + 'Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 0)

# Feature Scaling    #* Not needed here as DT is not based on Euclidean Dist
ance. (but for plotting high resolution fast)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)


#######

# Fitting Random Forest Classification to the Training set
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy'
, random_state = 0)
classifier.fit(X_train, y_train)




# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(y_pred)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set
[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set
[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.50, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Random Forest Classification (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set
[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set
[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.55, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Random Forest Classification (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

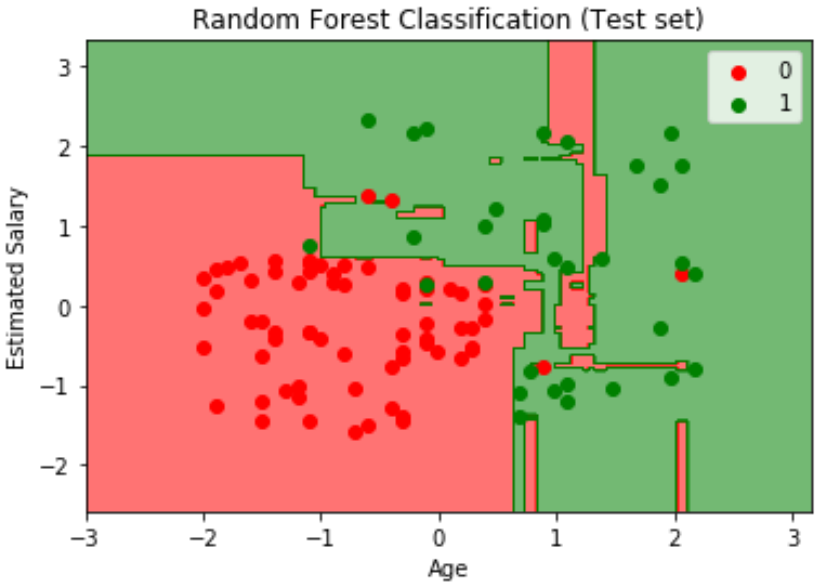
```
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

```
[0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0
1 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 1 1 1 0 0 1 0 0 1 0 1 0 1 0 0 0
0 1 0 0 1
 0 0 0 0 1 1 1 1 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 1 1 1]
[[63  5]
 [ 4 28]]
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
```




```
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
```



In [ ]:

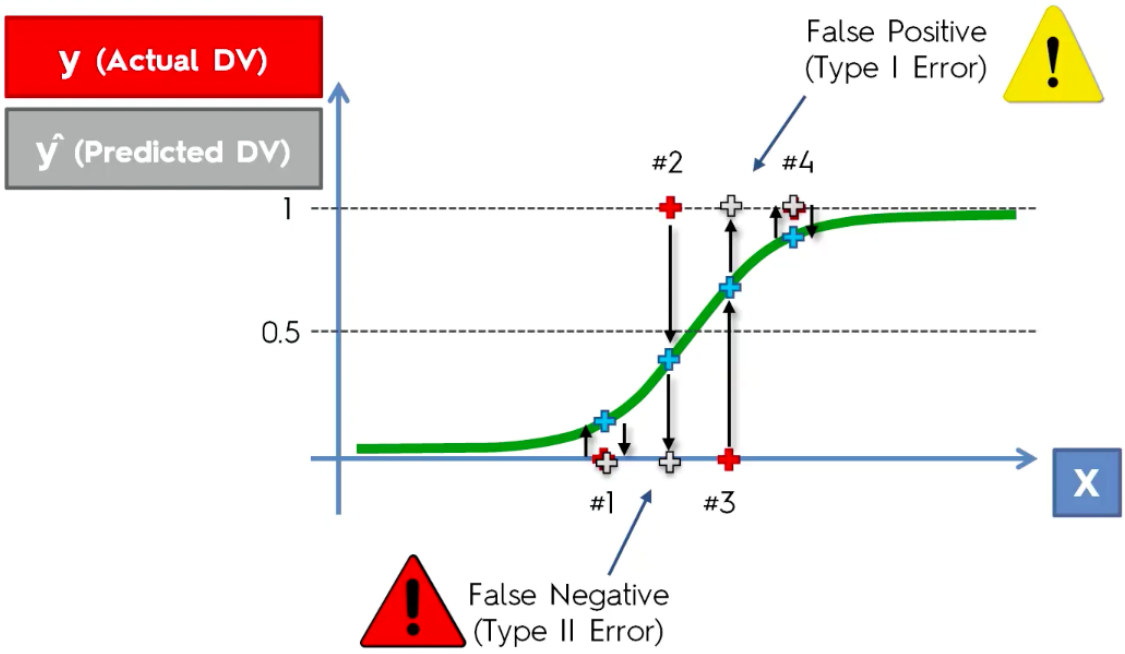
In [ ]:

```
#============================================================
# Section 19 (21): Evaluating Classification Models Performance
#============================================================
```

In [ ]:

In [ ]:

```
### False Positives & False Negatives:
```

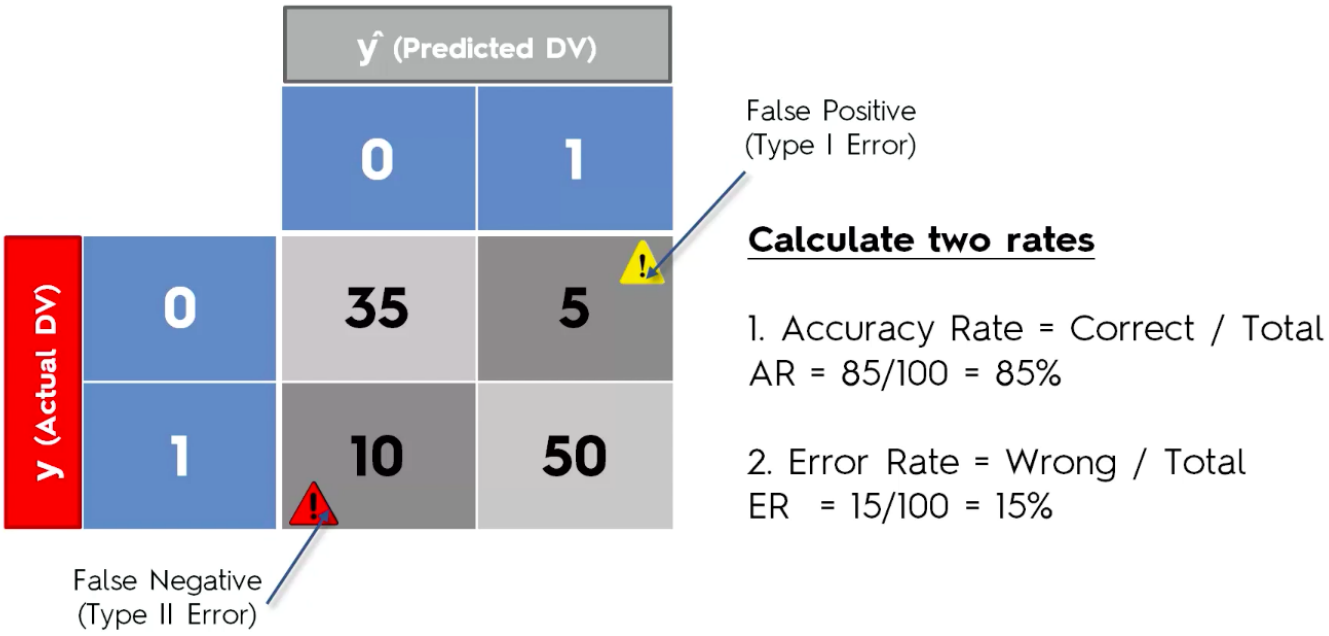


In [ ]:

```
```

In [13]:

```
### Confusion Matrix:
```

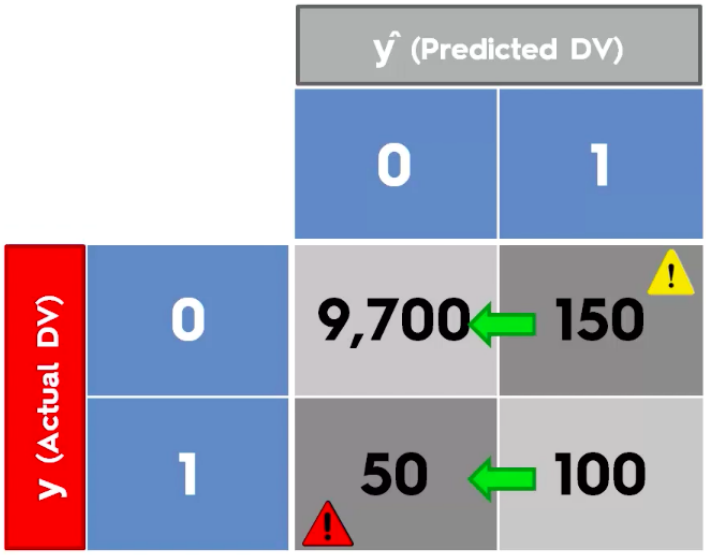


In [ ]:

```
```

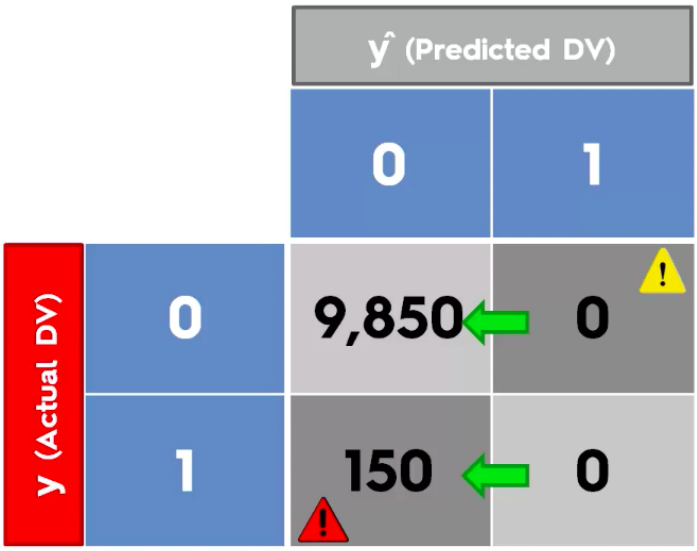
In [14]:

```
### Accuracy Paradox:

#* Should not base your judgement 'just' on the Accuracy rate.

## Here, random selection has more Accuracy than the model.
```



Scenario 1:

Accuracy Rate = Correct / Total
AR = 9,800/10,000 = 98%



Scenario 1:

Accuracy Rate = Correct / Total
AR = 9,800/10,000 = 98%

Scenario 2:

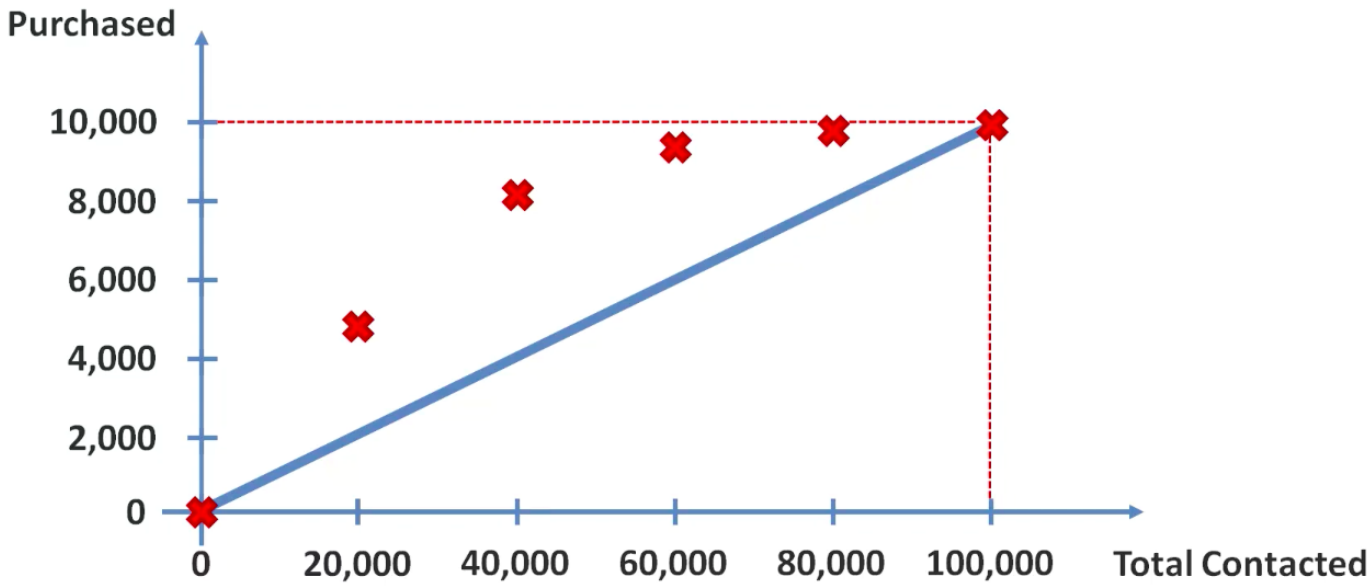
Accuracy Rate = Correct / Total
AR = 9,850/10,000 = 98.5%

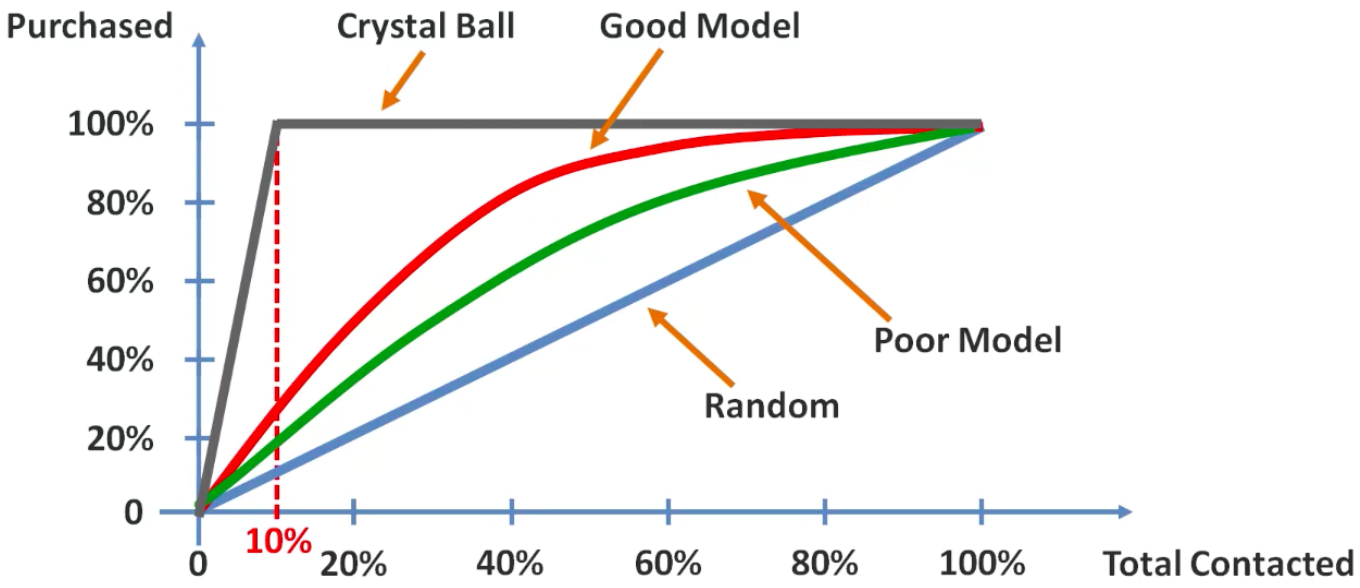
In [ ]:

```
```

In [15]:

```
### CAP (Cumulative Accuracy Profile) Curve:



# CAP: Cumulative Accuracy Profile

# Blue line is the one for random selection.

# * Area between the red and the blue line increase as the model gets bette
r.


# Gain (of the model) Chart.


## ** CAP != ROC
```





Note:

CAP = Cumulative Accuracy Profile

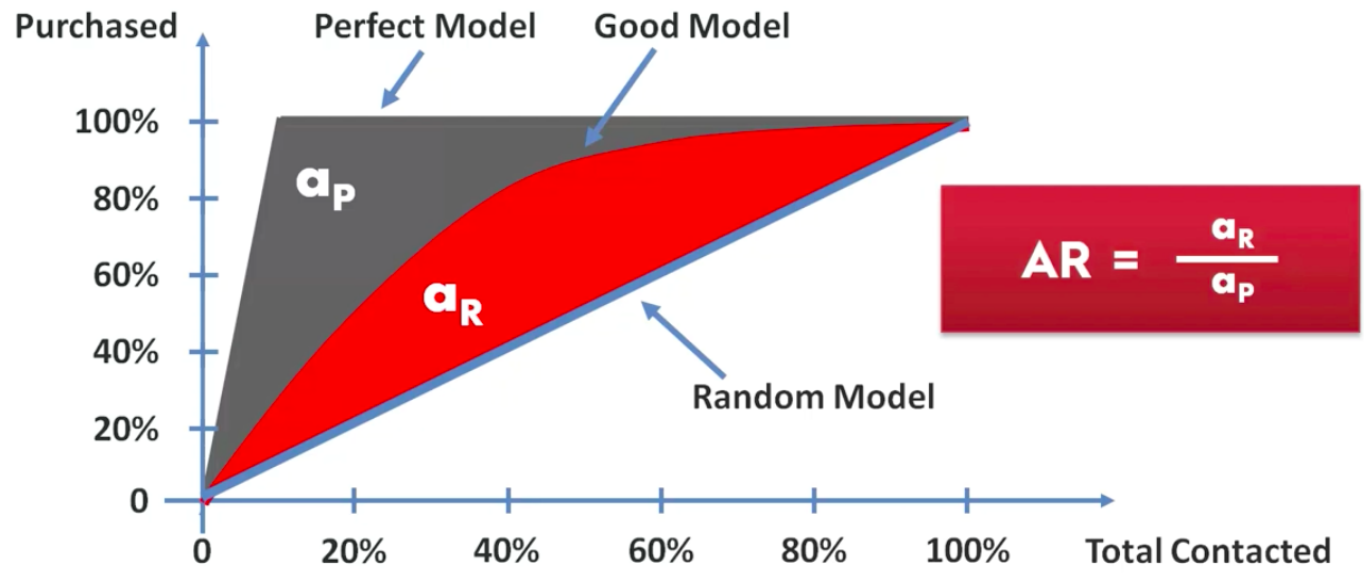
≠

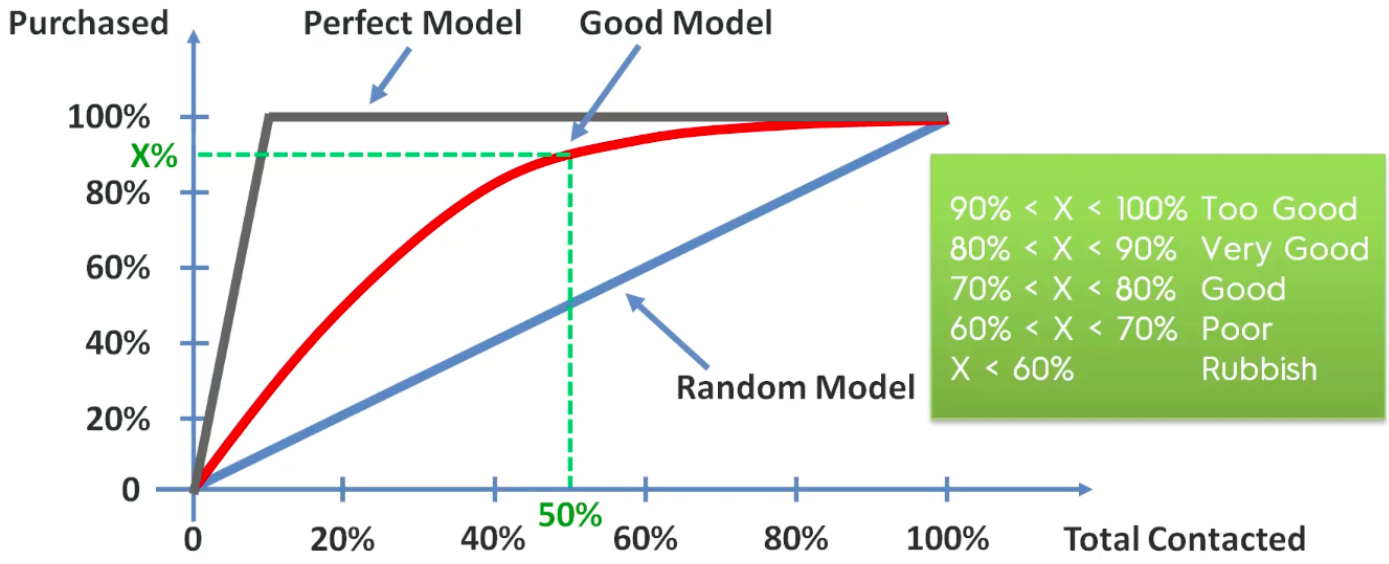
ROC = Receiver Operating Characteristic

In [ ]:

In [16]:

```
### CAP Curve Analysis:
```





In [ ]:

```

```

In [ ]:

```
### Conclusion:



## ** If your problem is linear, you should go for Logistic Regression or SV
M.

## ** If your problem is non linear, you should go for K-NN, Naive Bayes, De
cision Tree or Random Forest.



## from a business point of view, use:

# - Use Logistic Regression or Naive Bayes when you want to rank your predic
tions by their probability.

# - SVM when you want to predict to which segment your customers belong to.
 Segments can be any kind
    # of segments, for example some market segments you identified earlier w
ith clustering.

# - Decision Tree when you want to have clear interpretation of your model r
esults,

# - Random Forest when you are just looking for high performance with less n
eed for interpretation.
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```
#########################################
# Section 20 (23): Part 4: CLUSTERING
#########################################
```

In [ ]:

```

```

In [ ]:

```
```

In [ ]:

```
# In Clustering you don't know what you are looking for, and you are trying
 to identify some segments or
    # clusters in your data. When you use clustering algorithms on your data
set, unexpected things can
    # suddenly pop up like structures, clusters and groupings you would have
never thought of otherwise.



# following Machine Learning Clustering models:

    # - K-Means Clustering
    # - Hierarchical Clustering
```

In [ ]:

```
```

In [ ]:

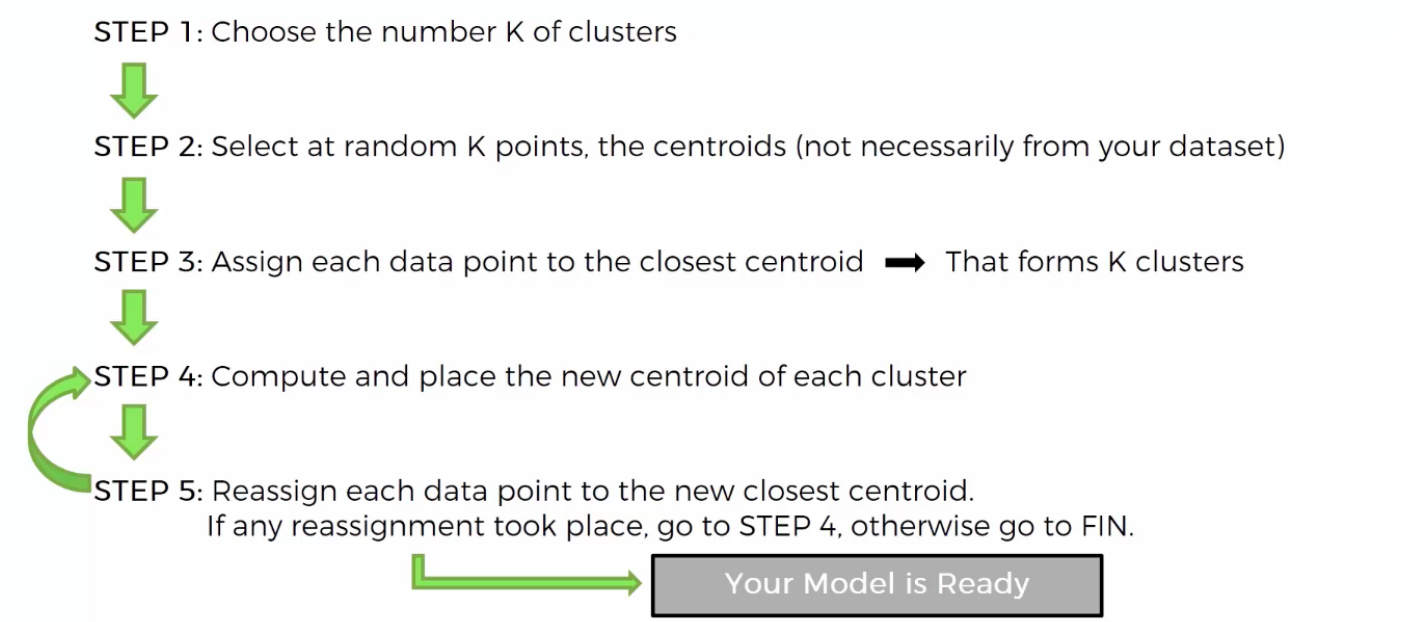
```
#==================================
# Section 21 (24): K-Means Clustering
#==================================
```

In [ ]:

```
```

In [ ]:

```
## Intuition:
```



In [ ]:

```
```

In [ ]:

```
## Intuition: Random Initialization Trap

## * initial selection of the centroids can potentially dictiate the outcome
of the algo.

## * Solution: K-Means++
```

In [ ]:

```
## K-Means: Selecting The Number Of Clusters : minimizing the WCSS.


# WCSS (Within-Cluster Sums of Squares):

# WCSS decreases from a big value to zero (when each point has its own clust
er).

# using the Elbow method.: look for drop where WCSS drop goes from substanci
al to small.
```

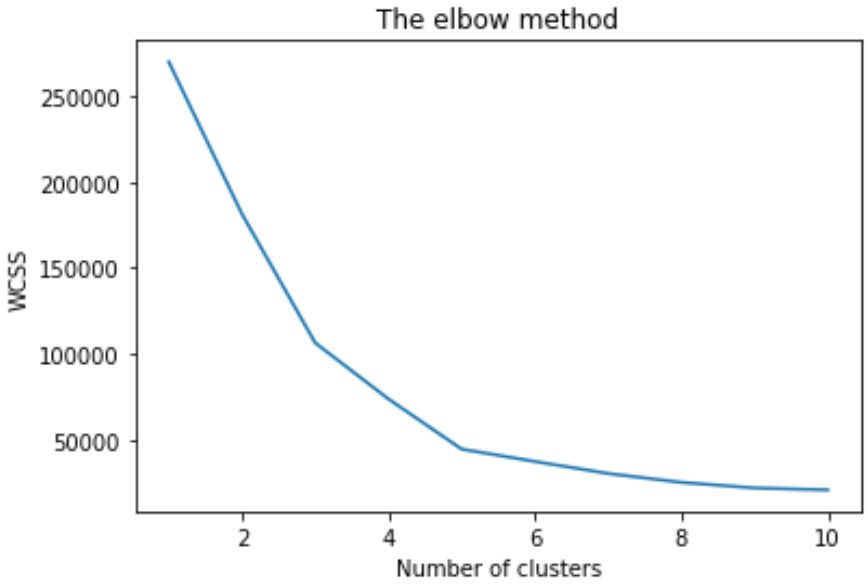
In [ ]:

```

```

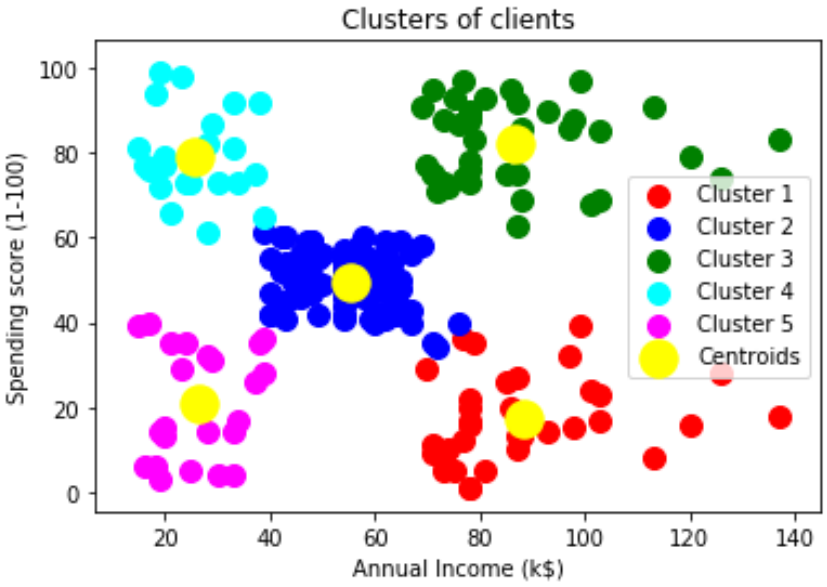
In [23]:

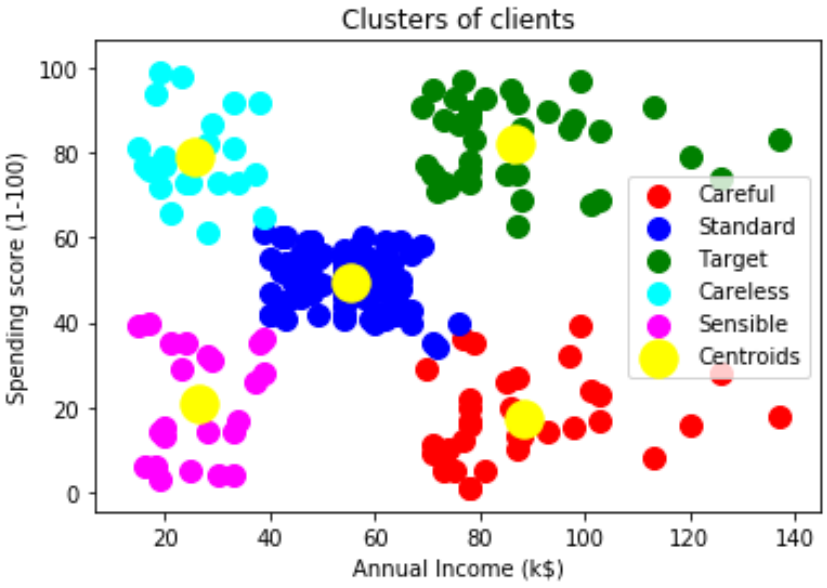
```
## K-Means Clustering:


# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the mall dataset
path = 'Machine Learning A-Z Template Folder/Part 4 - Clustering/Section 24
 - K-Means Clustering/'
dataset = pd.read_csv(path + 'Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values
#y = dataset.iloc[:, 4].values

# # Splitting the dataset into the Training set and Test set
# from sklearn.model_selection import train_test_split
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2
5, random_state = 0)

# # Feature Scaling
# from sklearn.preprocessing import StandardScaler
# sc = StandardScaler()
# X = sc.fit_transform(X)


#######


# using the elbow method to find the optimal number of clusters:

from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10,
random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()


# Applying k-means to the mall data:
kmeans = KMeans(n_clusters=5, init='k-means++', max_iter=300, n_init=10, ran
dom_state=0)
y_kmeans = kmeans.fit_predict(X) # predicts which cluster each datapoint bel
```

```
ongs to.
print(y_kmeans)

# visualizing the cluster (2-D):
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s=100, c='red', label=
'Cluster 1') # specify: only observations belonging to cluster 0.
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s=100, c='blue', label
='Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s=100, c='green', labe
l='Cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s=100, c='cyan', label
='Cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s=100, c='magenta', la
bel='Cluster 5')
# plotting the centes:
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=
300, c='yellow', label='Centroids')

plt.title('Clusters of clients')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending score (1-100)')
plt.legend()
plt.show()

# give label based on their behaviour:
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s=100, c='red', label=
'Careful') # specify: only observations belonging to cluster 0.
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s=100, c='blue', label
='Standard')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s=100, c='green', labe
l='Target')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s=100, c='cyan', label
='Careless')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s=100, c='magenta', la
bel='Sensible')
# plotting the centes:
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=
300, c='yellow', label='Centroids')

plt.title('Clusters of clients')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending score (1-100)')
plt.legend()
plt.show()
```




```
[4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4
 3 4 3 4 3 4 1 4 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 2 0 2 1 2 0 2 0 2 1 2 0 2 0 2 0 2 0 2 1 2 0 2 0 2
 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0
 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2]
```


Clusters of clients
Spending score (1-100)
Annual Income (k$)
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Centroids


Clusters of clients
Spending score (1-100)
Annual Income (k$)
Careful
Standard
Target
Careless
Sensible
Centroids

In [ ]:

```
```

In [ ]:

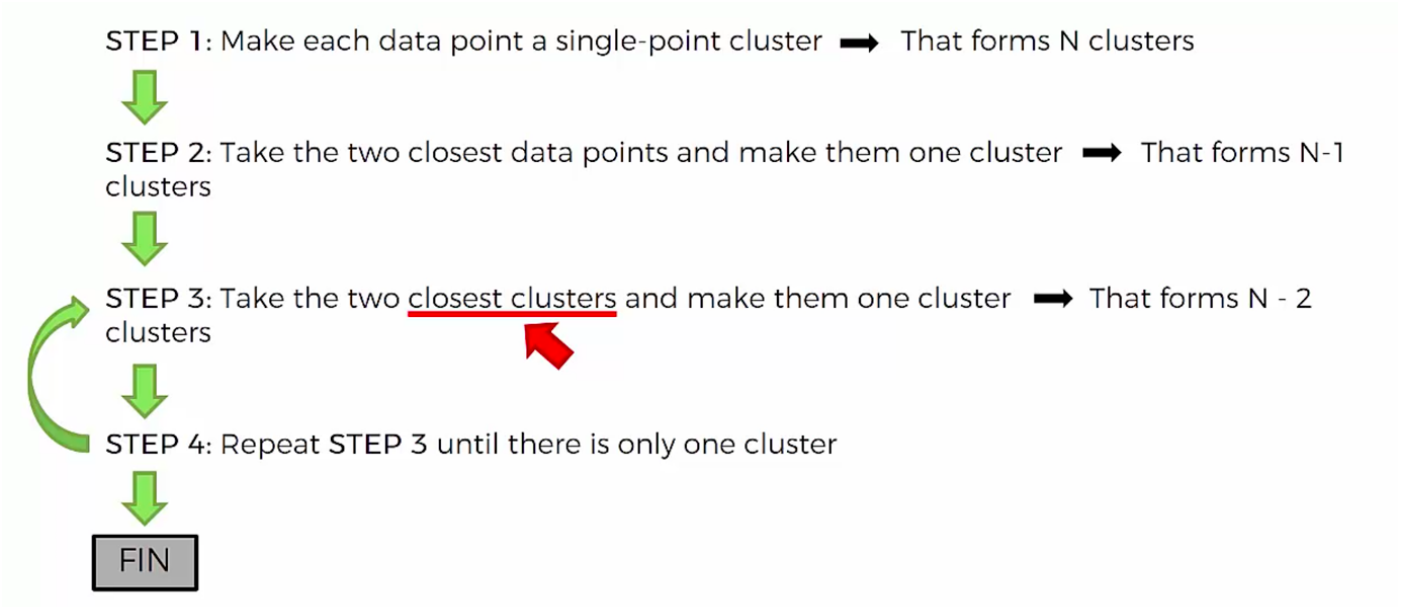
```
#==========================================
# Section 22 (25): Hierarchical Clustering
#==========================================
```

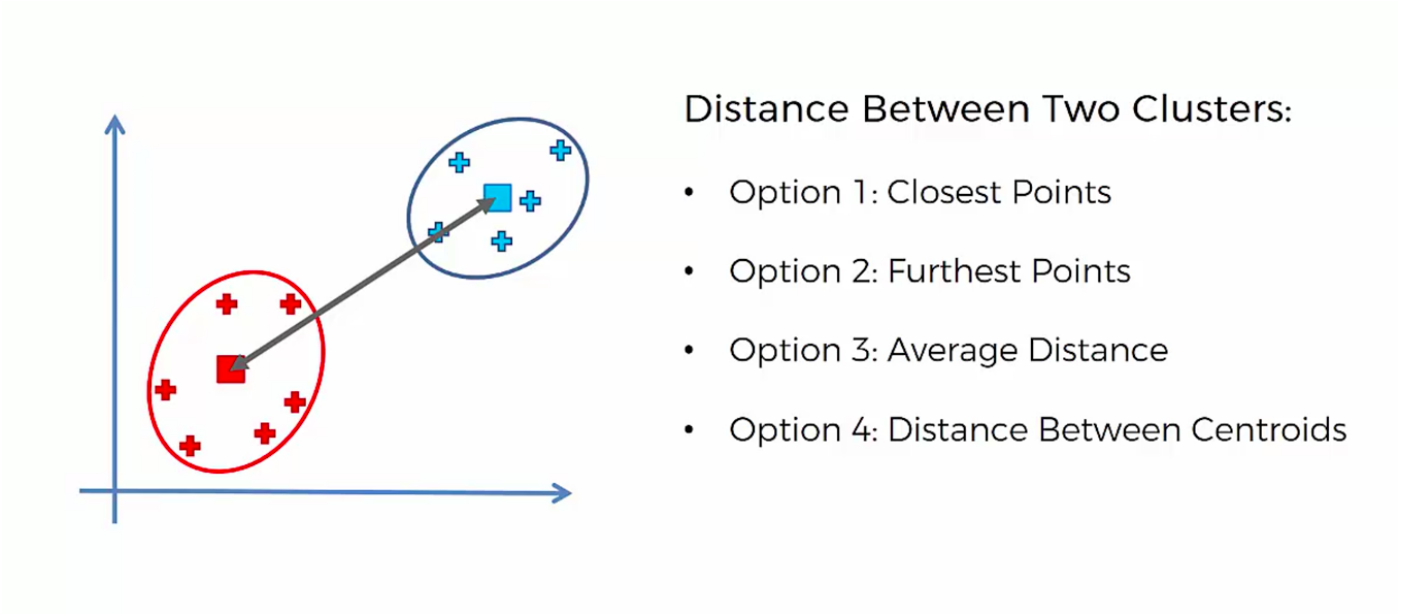
In [ ]:

```
```

In [24]:

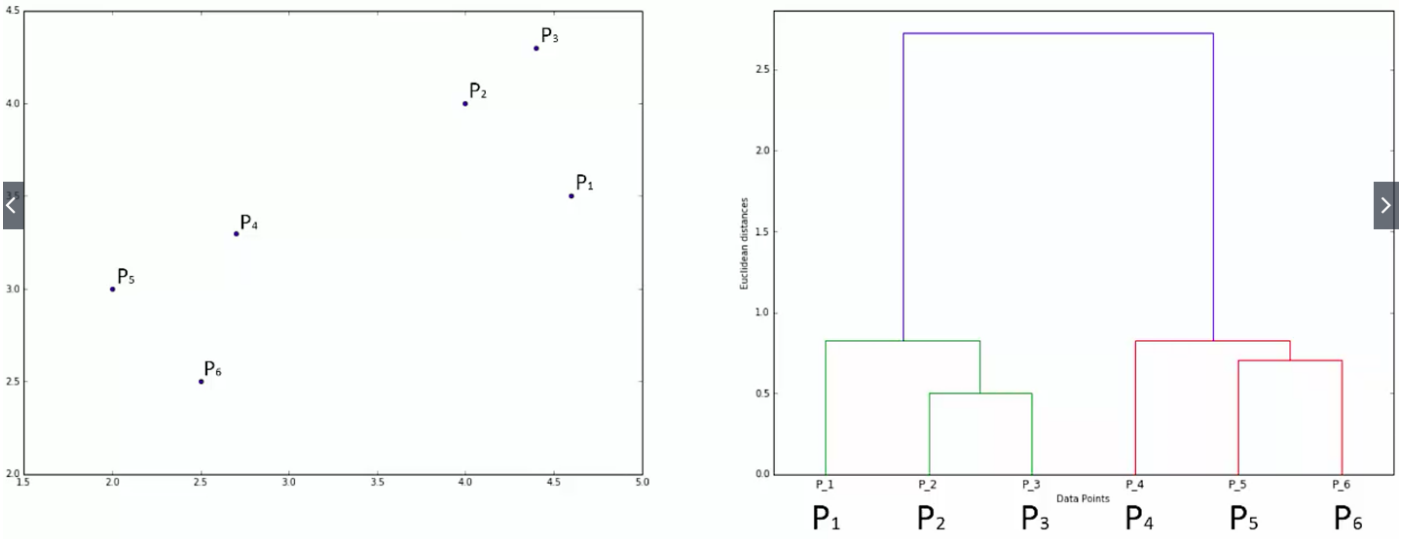
```
### Intuition:



## result same as K-Means but different process.

## 2 types: Agglomerative (bottom up approach), and Divisive (other way arou
nd)
```





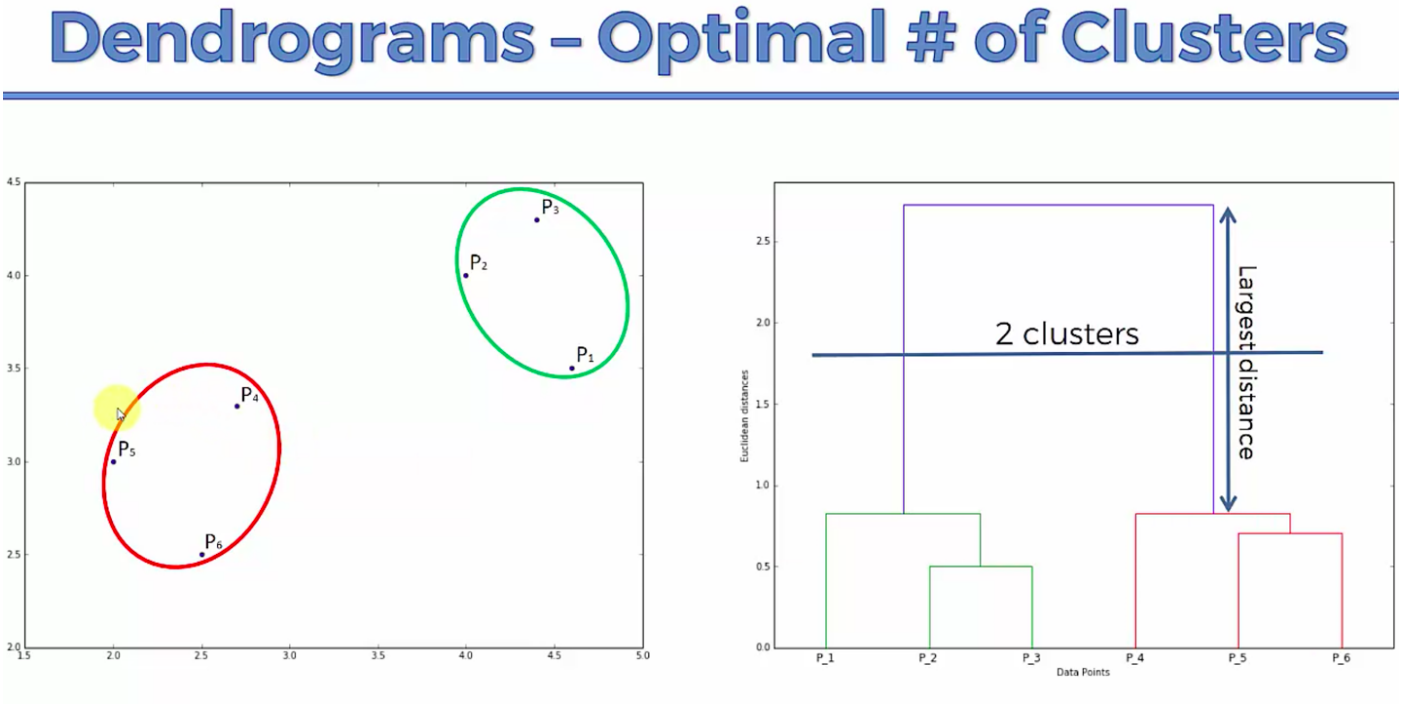
In [25]:

```
### How Dendrograms Work:
```



In [26]:

```
### HC Using Dendrograms


## Set distance (dissimilarity) thresholds on Dendrograms.

##* For optimal number of clusters: look for highest vertical distance on de
ndrogram (not crossing any horizontal lines).
```



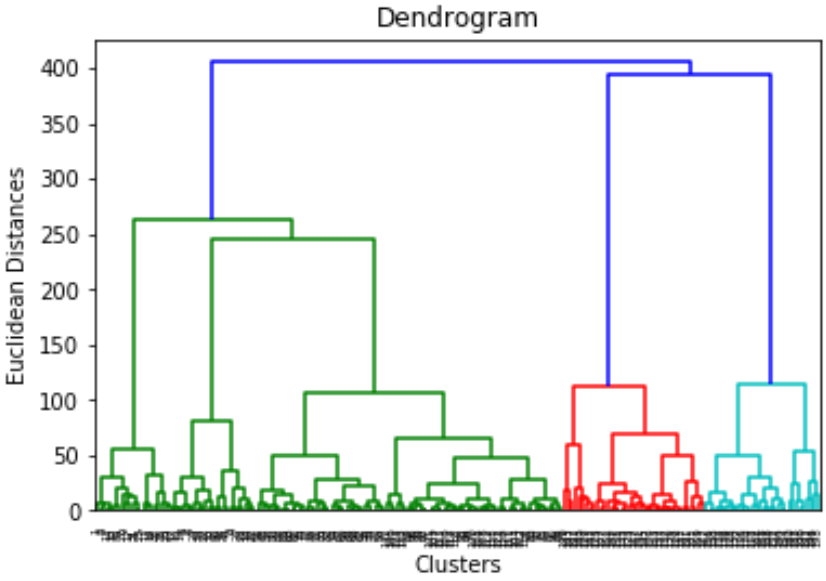
In [ ]:

```

```

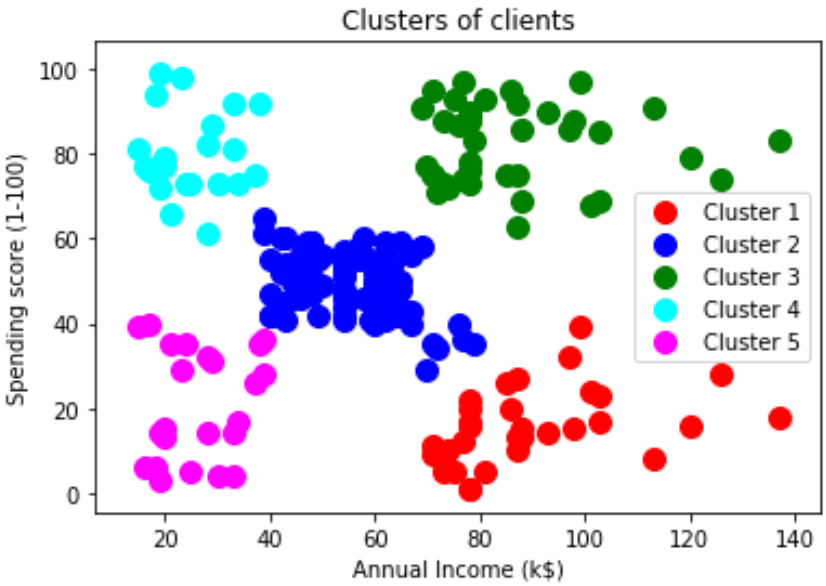
In [30]:

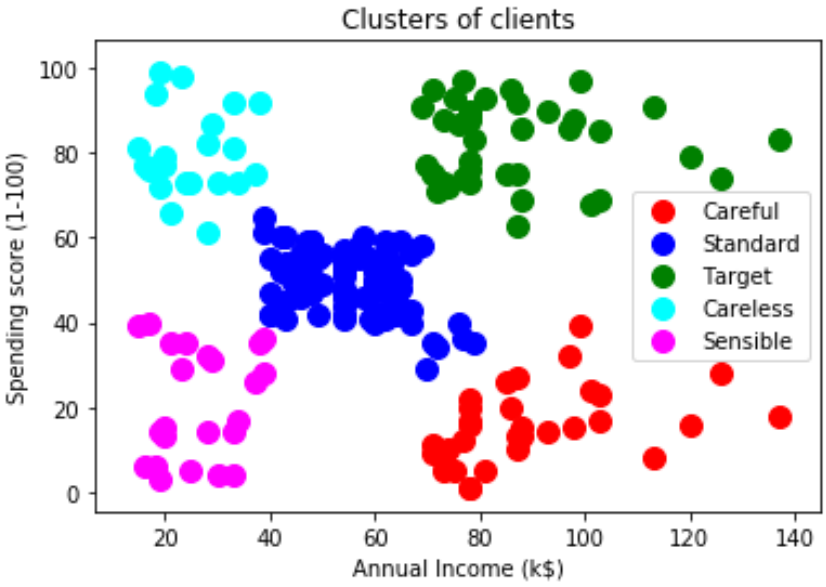
```
## Hierarchical Clustering:


# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the mall dataset
path = 'Machine Learning A-Z Template Folder/Part 4 - Clustering/Section 25
 - Hierarchical Clustering/'
dataset = pd.read_csv(path + 'Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values



#######

# using the dendrogram to find the optimal number of clusters:
import scipy.cluster.hierarchy as sch    ## **

dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))    ## **
plt.title('Dendrogram')
plt.xlabel('Clusters')
plt.ylabel('Euclidean Distances')
plt.show()


# fitting HC to the mall data:
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='wa
rd')
y_hc = hc.fit_predict(X) # predicts which cluster each datapoint belongs to.
print(y_hc)


# visualizing the cluster (2-D):
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s=100, c='red', label='Cluster
1') # specify: only observations belonging to cluster 0.
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s=100, c='blue', label='Cluste
r 2')
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s=100, c='green', label='Clust
er 3')
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s=100, c='cyan', label='Cluste
r 4')
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s=100, c='magenta', label='Clu
ster 5')
## plotting the centes:
#plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s
=300, c='yellow', label='Centroids')
```

```
plt.title('Clusters of clients')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending score (1-100)')
plt.legend()
plt.show()

# give label based on their behaviour:
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s=100, c='red', label='Carefu
l') # specify: only observations belonging to cluster 0.
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s=100, c='blue', label='Standa
rd')
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s=100, c='green', label='Targe
t')
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s=100, c='cyan', label='Carele
ss')
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s=100, c='magenta', label='Sen
sible')

plt.title('Clusters of clients')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending score (1-100)')
plt.legend()
plt.show()
```




```
[4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4
 3 4 3 4 3 4 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 2 0 2 0 2 1 2 0 2 0 2 0 2 0 2 1 2 0 2 1 2
 0 2 0 2 0 2 0 2 0 2 0 2 1 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0
 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2]
```





In [ ]:

In [ ]:

Clustering

Clustering Model	Pros	Cons
K-Means	Simple to understand, easily adaptable, works well on small or large datasets, fast, efficient and performant	Need to choose the number of clusters
Hierarchical Clustering	The optimal number of clusters can be obtained by the model itself, practical visualisation with the dendrogram	Not appropriate for large datasets

In [ ]:

In [ ]:

```
```

In [ ]:

```
```

In [31]:

```
#############################################################
# Section 23 (27): Part 5: ASSOCIATION RULE LEARNING
#############################################################
```

In [ ]:

```
```

In [ ]:

```
```

In [32]:

```
##  Association Rule Learning: helps to figure out "People who bought also b
ought ..." things/situations.


## following Association Rule Learning models:

    # - Apriori
    # - Eclat
```

In [ ]:

```
```

In [33]:

```
#=========================
# Section 24 (28): Apriori
#=========================
```

In [ ]:

```
```

In [ ]:

```
### Intuition

# 1) Support,  2) Confidence,  3) Lift (= Confidence/Support)


# It proceeds by identifying the frequent individual items in the database a
nd extending them to larger and
    # larger item sets as long as those item sets appear sufficiently often
 in the database.

# The frequent item sets determined by Apriori can be used to determine asso
ciation rules which highlight general
     #trends in the database: this has applications in domains such as 'mark
et basket analysis'.
```

1. **Support :** It is one of the measure of interestingness. This tells about usefulness and certainty of rules. **5% Support** means total 5% of transactions in database follow the rule.

   ```
   Support(A -> B) = Support_count(A U B)
   ```

2. **Confidence:** A confidence of 60% means that 60% of the customers who purchased a milk and bread also bought butter.

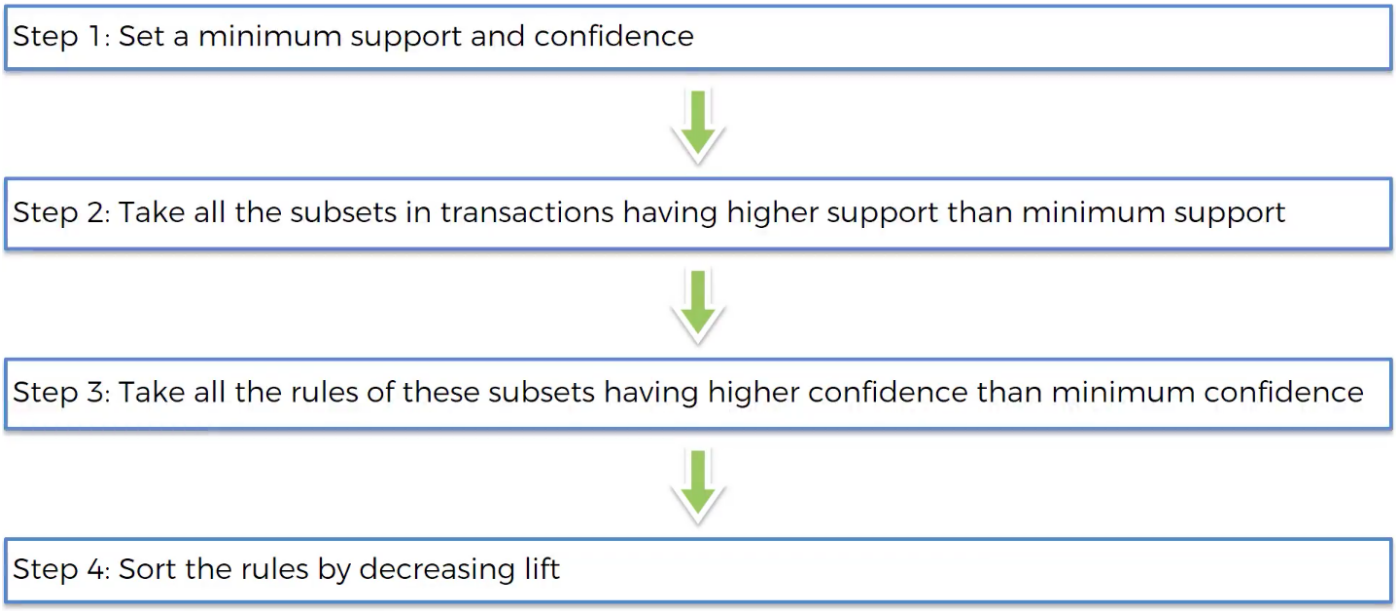
   ```
   Confidence(A -> B) = Support_count(A U B) / Support_count(A)
   ```

   If a rule satisfies both minimum support and minimum confidence, it is a strong rule.

   Support_count(X) : Number of transactions in which X appears. If X is A union B then it is the number of transactions in which A and B both are present.

In [34]:

```
## Apriori algo:
```


Step 1: Set a minimum support and confidence
Step 2: Take all the subsets in transactions having higher support than minimum support
Step 3: Take all the rules of these subsets having higher confidence than minimum confidence
Step 4: Sort the rules by decreasing lift

In [ ]:

In [42]:

```
## Apriori:


# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the mall dataset
path = 'Machine Learning A-Z Template Folder/Part 5 - Association Rule Learning/Section 28 - Apriori/'
dataset = pd.read_csv(path + 'Apriori_Python/Market_Basket_Optimisation.csv', header=None)  # first row here is not titles
transactions = []
for i in range(0, 7501):
    transactions.append([str(dataset.values[i, j]) for j in range(20)])

#print(dataset)
#print(transactions)


#######




# Training Apriori on the dataset:

import sys
sys.path.append(path)

from Apriori_Python.apyori import apriori
rules = apriori(transactions, min_support=0.003, min_confidence=0.2, min_lift=3, min_length=2)
# min support = 3*7/7500 = 0.003 # products bought 3 times a day (over 7 days).

#* Too high confidence will lead to very obvious rules.


# Visualizing the Results:
results = list(rules)
## No need to sort rules by lift in python.
print(results)
```

```
[RelationRecord(items=frozenset({'light cream', 'chicken'}), sup
port=0.004532728969470737, ordered_statistics=[OrderedStatistic
(items_base=frozenset({'light cream'}), items_add=frozenset({'ch
icken'}), confidence=0.29059829059829057, lift=4.8439506172839
5)]), RelationRecord(items=frozenset({'mushroom cream sauce', 'e
scalope'}), support=0.005732568990801226, ordered_statistics=[Or
deredStatistic(items_base=frozenset({'mushroom cream sauce'}), i
tems_add=frozenset({'escalope'}), confidence=0.3006993006993007,
lift=3.790832696715049)]), RelationRecord(items=frozenset({'past
a', 'escalope'}), support=0.005865884548726837, ordered_statisti
cs=[OrderedStatistic(items_base=frozenset({'pasta'}), items_add=
frozenset({'escalope'}), confidence=0.3728813559322034, lift=4.7
00811850163794)]), RelationRecord(items=frozenset({'honey', 'fro
mage blanc'}), support=0.003332888948140248, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'fromage blanc'}), items
_add=frozenset({'honey'}), confidence=0.2450980392156863, lift=
5.164270764485569)]), RelationRecord(items=frozenset({'herb & pe
pper', 'ground beef'}), support=0.015997866951073192, ordered_st
atistics=[OrderedStatistic(items_base=frozenset({'herb & peppe
r'}), items_add=frozenset({'ground beef'}), confidence=0.3234501
347708895, lift=3.2919938411349285)]), RelationRecord(items=froz
enset({'tomato sauce', 'ground beef'}), support=0.005332622317024
397, ordered_statistics=[OrderedStatistic(items_base=frozenset
({'tomato sauce'}), items_add=frozenset({'ground beef'}), confid
ence=0.3773584905660377, lift=3.840659481324083)]), RelationReco
rd(items=frozenset({'light cream', 'olive oil'}), support=0.0031
99573390214638, ordered_statistics=[OrderedStatistic(items_base=
frozenset({'light cream'}), items_add=frozenset({'olive oil'}),
confidence=0.20512820512820515, lift=3.1147098515519573)]), Rela
tionRecord(items=frozenset({'whole wheat pasta', 'olive oil'}),
support=0.007998933475536596, ordered_statistics=[OrderedStatist
ic(items_base=frozenset({'whole wheat pasta'}), items_add=frozen
set({'olive oil'}), confidence=0.2714932126696833, lift=4.122410
097642296)]), RelationRecord(items=frozenset({'pasta', 'shrim
p'}), support=0.005065991201173177, ordered_statistics=[OrderedS
tatistic(items_base=frozenset({'pasta'}), items_add=frozenset
({'shrimp'}), confidence=0.3220338983050847, lift=4.506672147735
896)]), RelationRecord(items=frozenset({'avocado', 'spaghetti',
'milk'}), support=0.003332888948140248, ordered_statistics=[Orde
redStatistic(items_base=frozenset({'avocado', 'spaghetti'}), ite
ms_add=frozenset({'milk'}), confidence=0.4166666666666663, lift
=3.215449245541838)]), RelationRecord(items=frozenset({'milk',
'burgers', 'cake'}), support=0.0037328356219170776, ordered_stat
istics=[OrderedStatistic(items_base=frozenset({'milk', 'cake'}),
items_add=frozenset({'burgers'}), confidence=0.2799999999999999
7, lift=3.211437308868501)]), RelationRecord(items=frozenset({'c
hocolate', 'turkey', 'burgers'}), support=0.0030662578322890282,
ordered_statistics=[OrderedStatistic(items_base=frozenset({'choc
olate', 'turkey'}), items_add=frozenset({'burgers'}), confidence
=0.27058823529411763, lift=3.1034898363014927)]), RelationRecord
(items=frozenset({'milk', 'turkey', 'burgers'}), support=0.00319
9573390214638, ordered_statistics=[OrderedStatistic(items_base=f
rozenset({'turkey', 'milk'}), items_add=frozenset({'burgers'}),
```

```
confidence=0.2823529411764706, lift=3.2384241770102533)]), Relat
ionRecord(items=frozenset({'tomatoes', 'frozen vegetables', 'cak
e'}), support=0.0030662578322890282, ordered_statistics=[Ordered
Statistic(items_base=frozenset({'frozen vegetables', 'cake'}), i
tems_add=frozenset({'tomatoes'}), confidence=0.2987012987012987,
lift=4.367560314928736), OrderedStatistic(items_base=frozenset
({'tomatoes', 'cake'}), items_add=frozenset({'frozen vegetable
s'}), confidence=0.36507936507936506, lift=3.830014430014429
6)]), RelationRecord(items=frozenset({'ground beef', 'spaghett
i', 'cereals'}), support=0.0030662578322890282, ordered_statisti
cs=[OrderedStatistic(items_base=frozenset({'ground beef', 'cerea
ls'}), items_add=frozenset({'spaghetti'}), confidence=0.67647058
82352942, lift=3.8853031258445188), OrderedStatistic(items_base=
frozenset({'spaghetti', 'cereals'}), items_add=frozenset({'groun
d beef'}), confidence=0.45999999999999996, lift=4.68176390773405
7)]), RelationRecord(items=frozenset({'ground beef', 'milk', 'ch
icken'}), support=0.0038661511798426876, ordered_statistics=[Ord
eredStatistic(items_base=frozenset({'ground beef', 'chicken'}),
items_add=frozenset({'milk'}), confidence=0.40845070422535207, l
ift=3.152046020981858)]), RelationRecord(items=frozenset({'light
cream', 'nan', 'chicken'}), support=0.004532728969470737, ordere
d_statistics=[OrderedStatistic(items_base=frozenset({'light crea
m', 'nan'}), items_add=frozenset({'chicken'}), confidence=0.2905
9829059829057, lift=4.84395061728395)]), RelationRecord(items=fr
ozenset({'olive oil', 'milk', 'chicken'}), support=0.00359952006
39914677, ordered_statistics=[OrderedStatistic(items_base=frozen
set({'milk', 'chicken'}), items_add=frozenset({'olive oil'}), co
nfidence=0.24324324324324323, lift=3.693456614509246), OrderedSt
atistic(items_base=frozenset({'olive oil', 'chicken'}), items_ad
d=frozenset({'milk'}), confidence=0.5, lift=3.858539094650206),
OrderedStatistic(items_base=frozenset({'milk', 'olive oil'}), it
ems_add=frozenset({'chicken'}), confidence=0.2109375, lift=3.516
09375)]), RelationRecord(items=frozenset({'olive oil', 'spaghett
i', 'chicken'}), support=0.0034662045060658577, ordered_statisti
cs=[OrderedStatistic(items_base=frozenset({'spaghetti', 'chicke
n'}), items_add=frozenset({'olive oil'}), confidence=0.201550387
59689922, lift=3.0603835169318647)]), RelationRecord(items=froze
nset({'chocolate', 'frozen vegetables', 'shrimp'}), support=0.00
5332622317024397, ordered_statistics=[OrderedStatistic(items_bas
e=frozenset({'chocolate', 'frozen vegetables'}), items_add=froze
nset({'shrimp'}), confidence=0.2325581953488375, lift=3.2545123
221103784), OrderedStatistic(items_base=frozenset({'chocolate',
'shrimp'}), items_add=frozenset({'frozen vegetables'}), confiden
ce=0.2962962962962634, lift=3.1084175084175087)]), RelationReco
rd(items=frozenset({'ground beef', 'herb & pepper', 'chocolat
e'}), support=0.003999466737768298, ordered_statistics=[OrderedS
tatistic(items_base=frozenset({'herb & pepper', 'chocolate'}), i
tems_add=frozenset({'ground beef'}), confidence=0.44117647058823
54, lift=4.4901827759597746)]), RelationRecord(items=frozenset
({'soup', 'chocolate', 'milk'}), support=0.003999466737768298, o
rdered_statistics=[OrderedStatistic(items_base=frozenset({'choco
late', 'soup'}), items_add=frozenset({'milk'}), confidence=0.394
7368421052632, lift=3.0462150747238472)]), RelationRecord(items=
```

```
frozenset({'ground beef', 'cooking oil', 'spaghetti'}), support=
0.004799360085321957, ordered_statistics=[OrderedStatistic(items
_base=frozenset({'ground beef', 'cooking oil'}), items_add=froze
nset({'spaghetti'}), confidence=0.5714285714285714, lift=3.28199
51870487856), OrderedStatistic(items_base=frozenset({'spaghett
i', 'cooking oil'}), items_add=frozenset({'ground beef'}), confi
dence=0.3025210084033613, lift=3.0789824749438446)]), RelationRe
cord(items=frozenset({'herb & pepper', 'eggs', 'ground beef'}),
support=0.0041327822956939075, ordered_statistics=[OrderedStatis
tic(items_base=frozenset({'eggs', 'ground beef'}), items_add=fro
zenset({'herb & pepper'}), confidence=0.2066666666666667, lift=
4.178454627133872), OrderedStatistic(items_base=frozenset({'herb
& pepper', 'eggs'}), items_add=frozenset({'ground beef'}), confi
dence=0.3297872340425532, lift=3.3564912381997174)]), RelationRe
cord(items=frozenset({'eggs', 'spaghetti', 'red wine'}), support
=0.0037328356219170776, ordered_statistics=[OrderedStatistic(ite
ms_base=frozenset({'eggs', 'red wine'}), items_add=frozenset({'s
paghetti'}), confidence=0.5283018867924528, lift=3.0342974370828
397)]), RelationRecord(items=frozenset({'nan', 'mushroom cream s
auce', 'escalope'}), support=0.005732568990801226, ordered_stati
stics=[OrderedStatistic(items_base=frozenset({'nan', 'mushroom c
ream sauce'}), items_add=frozenset({'escalope'}), confidence=0.3
006993006993007, lift=3.790832696715049)]), RelationRecord(items
=frozenset({'nan', 'pasta', 'escalope'}), support=0.005865884548
726837, ordered_statistics=[OrderedStatistic(items_base=frozense
t({'nan', 'pasta'}), items_add=frozenset({'escalope'}), confiden
ce=0.3728813559322034, lift=4.700811850163794)]), RelationRecord
(items=frozenset({'french fries', 'herb & pepper', 'ground bee
f'}), support=0.003199573390214638, ordered_statistics=[OrderedS
tatistic(items_base=frozenset({'french fries', 'ground beef'}),
items_add=frozenset({'herb & pepper'}), confidence=0.23076923076
923078, lift=4.665768194070081), OrderedStatistic(items_base=fro
zenset({'french fries', 'herb & pepper'}), items_add=frozenset
({'ground beef'}), confidence=0.46153846153846156, lift=4.697421
981004071)]), RelationRecord(items=frozenset({'nan', 'honey', 'f
romage blanc'}), support=0.003332888948140248, ordered_statistic
s=[OrderedStatistic(items_base=frozenset({'nan', 'fromage blan
c'}), items_add=frozenset({'honey'}), confidence=0.2450980392156
863, lift=5.164270764485569)]), RelationRecord(items=frozenset
({'tomatoes', 'frozen vegetables', 'green tea'}), support=0.0033
32888948140248, ordered_statistics=[OrderedStatistic(items_base=
frozenset({'frozen vegetables', 'green tea'}), items_add=frozens
et({'tomatoes'}), confidence=0.2314814814814815, lift=3.38468341
635983)]), RelationRecord(items=frozenset({'ground beef', 'froze
n vegetables', 'spaghetti'}), support=0.008665511265164644, orde
red_statistics=[OrderedStatistic(items_base=frozenset({'spaghett
i', 'frozen vegetables'}), items_add=frozenset({'ground beef'}),
confidence=0.31100478468899523, lift=3.165328208890303)]), Relat
ionRecord(items=frozenset({'frozen vegetables', 'milk', 'olive o
il'}), support=0.004799360085321957, ordered_statistics=[Ordered
Statistic(items_base=frozenset({'frozen vegetables', 'milk'}), i
tems_add=frozenset({'olive oil'}), confidence=0.2033898305084745
6, lift=3.088314005352364), OrderedStatistic(items_base=frozense
```

```
t({'frozen vegetables', 'olive oil'}), items_add=frozenset({'mil
k'}), confidence=0.4235294117647058, lift=3.2684095860566447)]),
RelationRecord(items=frozenset({'soup', 'frozen vegetables', 'mi
lk'}), support=0.003999466737768298, ordered_statistics=[Ordered
Statistic(items_base=frozenset({'frozen vegetables', 'soup'}), i
tems_add=frozenset({'milk'}), confidence=0.5, lift=3.85853909465
0206)]), RelationRecord(items=frozenset({'tomatoes', 'frozen veg
etables', 'milk'}), support=0.004132782295693907 5, ordered_stati
stics=[OrderedStatistic(items_base=frozenset({'tomatoes', 'mil
k'}), items_add=frozenset({'frozen vegetables'}), confidence=0.2
9523809523809524, lift=3.0973160173160172)]), RelationRecord(ite
ms=frozenset({'mineral water', 'frozen vegetables', 'shrimp'}),
support=0.007199040127982935, ordered_statistics=[OrderedStatist
ic(items_base=frozenset({'mineral water', 'shrimp'}), items_add=
frozenset({'frozen vegetables'}), confidence=0.3050847457627118
3, lift=3.200616332819722)]), RelationRecord(items=frozenset({'s
paghetti', 'frozen vegetables', 'olive oil'}), support=0.0057325
68990801226, ordered_statistics=[OrderedStatistic(items_base=fro
zenset({'spaghetti', 'frozen vegetables'}), items_add=frozenset
({'olive oil'}), confidence=0.20574162679425836, lift=3.12402417
52707125)]), RelationRecord(items=frozenset({'spaghetti', 'froze
n vegetables', 'shrimp'}), support=0.005999200106652446, ordered
_statistics=[OrderedStatistic(items_base=frozenset({'spaghetti',
'frozen vegetables'}), items_add=frozenset({'shrimp'}), confiden
ce=0.21531100478468898, lift=3.0131489680782684)]), RelationReco
rd(items=frozenset({'tomatoes', 'frozen vegetables', 'shrimp'}),
support=0.003999466737768298, ordered_statistics=[OrderedStatist
ic(items_base=frozenset({'frozen vegetables', 'shrimp'}), items_
add=frozenset({'tomatoes'}), confidence=0.24000000000000002, lif
t=3.5092397660818717), OrderedStatistic(items_base=frozenset({'t
omatoes', 'frozen vegetables'}), items_add=frozenset({'shrim
p'}), confidence=0.2479338842975207, lift=3.4696866905143704), O
rderedStatistic(items_base=frozenset({'tomatoes', 'shrimp'}), it
ems_add=frozenset({'frozen vegetables'}), confidence=0.357142857
14285715, lift=3.7467532467532467)]), RelationRecord(items=froze
nset({'tomatoes', 'spaghetti', 'frozen vegetables'}), support=0.
006665777896280496, ordered_statistics=[OrderedStatistic(items_b
ase=frozenset({'spaghetti', 'frozen vegetables'}), items_add=fro
zenset({'tomatoes'}), confidence=0.23923444976076558, lift=3.498
0460188216425), OrderedStatistic(items_base=frozenset({'tomatoe
s', 'spaghetti'}), items_add=frozenset({'frozen vegetables'}), c
onfidence=0.3184713375796179, lift=3.341053850607991)]), Relatio
nRecord(items=frozenset({'grated cheese', 'ground beef', 'spaghe
tti'}), support=0.005332622317024397, ordered_statistics=[Ordere
dStatistic(items_base=frozenset({'grated cheese', 'spaghetti'}),
items_add=frozenset({'ground beef'}), confidence=0.3225806451612
903, lift=3.283144395325426)]), RelationRecord(items=frozenset
({'tomatoes', 'ground beef', 'green tea'}), support=0.0030662578
322890282, ordered_statistics=[OrderedStatistic(items_base=froze
nset({'ground beef', 'green tea'}), items_add=frozenset({'tomato
es'}), confidence=0.2072072072072072, lift=3.029749047292906
7)]), RelationRecord(items=frozenset({'herb & pepper', 'ground b
eef', 'milk'}), support=0.0035995200639914677, ordered_statistic
```

```
s=[OrderedStatistic(items_base=frozenset({'herb & pepper', 'mil
k'}), items_add=frozenset({'ground beef'}), confidence=0.3913043
478260869, lift=3.9825968969382335)]), RelationRecord(items=froz
enset({'mineral water', 'herb & pepper', 'ground beef'}), suppor
t=0.006665777896280496, ordered_statistics=[OrderedStatistic(ite
ms_base=frozenset({'mineral water', 'herb & pepper'}), items_add
=frozenset({'ground beef'}), confidence=0.39062500000000006, lif
t=3.975682666214383)]), RelationRecord(items=frozenset({'herb &
pepper', 'nan', 'ground beef'}), support=0.015997866951073192, o
rdered_statistics=[OrderedStatistic(items_base=frozenset({'herb
& pepper', 'nan'}), items_add=frozenset({'ground beef'}), confid
ence=0.3234501347708895, lift=3.2919938411349285)]), RelationRec
ord(items=frozenset({'herb & pepper', 'ground beef', 'spaghett
i'}), support=0.006399146780429276, ordered_statistics=[OrderedS
tatistic(items_base=frozenset({'herb & pepper', 'spaghetti'}), i
tems_add=frozenset({'ground beef'}), confidence=0.39344262295081
97, lift=4.004359721511667)]), RelationRecord(items=frozenset
({'ground beef', 'milk', 'olive oil'}), support=0.00493267564324
7567, ordered_statistics=[OrderedStatistic(items_base=frozenset
({'ground beef', 'milk'}), items_add=frozenset({'olive oil'}), c
onfidence=0.22422422424242427, lift=3.40494417862839)]), Relatio
nRecord(items=frozenset({'soup', 'ground beef', 'milk'}), suppor
t=0.003999466737768298, ordered_statistics=[OrderedStatistic(ite
ms_base=frozenset({'ground beef', 'soup'}), items_add=frozenset
({'milk'}), confidence=0.4109589041095891, lift=3.17140199560290
94)]), RelationRecord(items=frozenset({'tomato sauce', 'nan', 'g
round beef'}), support=0.005332622317024397, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'tomato sauce', 'nan'}),
items_add=frozenset({'ground beef'}), confidence=0.3773584905660
377, lift=3.840659481324083)]), RelationRecord(items=frozenset
({'ground beef', 'spaghetti', 'pepper'}), support=0.003332888948
140248, ordered_statistics=[OrderedStatistic(items_base=frozense
t({'spaghetti', 'pepper'}), items_add=frozenset({'ground bee
f'}), confidence=0.33783783783783783, lift=3.438428251861087
6)]), RelationRecord(items=frozenset({'spaghetti', 'ground bee
f', 'shrimp'}), support=0.005999200106652446, ordered_statistics
=[OrderedStatistic(items_base=frozenset({'ground beef', 'shrim
p'}), items_add=frozenset({'spaghetti'}), confidence=0.523255813
9534884, lift=3.005315360233627)]), RelationRecord(items=frozens
et({'tomato sauce', 'ground beef', 'spaghetti'}), support=0.0030
662578322890282, ordered_statistics=[OrderedStatistic(items_base
=frozenset({'tomato sauce', 'ground beef'}), items_add=frozenset
({'spaghetti'}), confidence=0.5750000000000001, lift=3.302507656
9678413), OrderedStatistic(items_base=frozenset({'tomato sauce',
'spaghetti'}), items_add=frozenset({'ground beef'}), confidence=
0.4893617021276596, lift=4.980599901844742)]), RelationRecord(it
ems=frozenset({'light cream', 'nan', 'olive oil'}), support=0.00
3199573390214638, ordered_statistics=[OrderedStatistic(items_bas
e=frozenset({'light cream', 'nan'}), items_add=frozenset({'olive
oil'}), confidence=0.20512820512820515, lift=3.114709851551957
3)]), RelationRecord(items=frozenset({'shrimp', 'milk', 'olive o
il'}), support=0.003199573390214638, ordered_statistics=[Ordered
Statistic(items_base=frozenset({'shrimp', 'olive oil'}), items_a
```

dd=frozenset({'milk'}), confidence=0.3934426229508197, lift=3.03
62274843149164)]), RelationRecord(items=frozenset({'soup', 'mil
k', 'olive oil'}), support=0.0035995200639914677, ordered_statis
tics=[OrderedStatistic(items_base=frozenset({'milk', 'olive oi
l'}), items_add=frozenset({'soup'}), confidence=0.2109375, lift=
4.174781497361478), OrderedStatistic(items_base=frozenset({'sou
p', 'milk'}), items_add=frozenset({'olive oil'}), confidence=0.2
3684210526315788, lift=3.5962603878116344), OrderedStatistic(ite
ms_base=frozenset({'soup', 'olive oil'}), items_add=frozenset
({'milk'}), confidence=0.4029850746268656, lift=3.10986733001658
33)]), RelationRecord(items=frozenset({'spaghetti', 'milk', 'oli
ve oil'}), support=0.007199040127982935, ordered_statistics=[Ord
eredStatistic(items_base=frozenset({'spaghetti', 'milk'}), items
_add=frozenset({'olive oil'}), confidence=0.20300751879699247, l
ift=3.0825089038385434)]), RelationRecord(items=frozenset({'sou
p', 'tomatoes', 'milk'}), support=0.0030662578322890282, ordered
_statistics=[OrderedStatistic(items_base=frozenset({'tomatoes',
'milk'}), items_add=frozenset({'soup'}), confidence=0.2190476190
4761905, lift=4.335293378565146), OrderedStatistic(items_base=fr
ozenset({'tomatoes', 'soup'}), items_add=frozenset({'milk'}), co
nfidence=0.44230769230769235, lift=3.4133230452674903)]), Relati
onRecord(items=frozenset({'whole wheat pasta', 'spaghetti', 'mil
k'}), support=0.003999466737768298, ordered_statistics=[OrderedS
tatistic(items_base=frozenset({'spaghetti', 'whole wheat past
a'}), items_add=frozenset({'milk'}), confidence=0.45454545454545
46, lift=3.5077628133183696)]), RelationRecord(items=frozenset
({'mineral water', 'soup', 'olive oil'}), support=0.005199306759
098787, ordered_statistics=[OrderedStatistic(items_base=frozense
t({'mineral water', 'soup'}), items_add=frozenset({'olive oi
l'}), confidence=0.22543352601156072, lift=3.423030118649224
5)]), RelationRecord(items=frozenset({'mineral water', 'whole wh
eat pasta', 'olive oil'}), support=0.0038661511798426876, ordere
d_statistics=[OrderedStatistic(items_base=frozenset({'mineral wa
ter', 'whole wheat pasta'}), items_add=frozenset({'olive oil'}),
confidence=0.4027777777777778, lift=6.115862573099416)]), Relati
onRecord(items=frozenset({'nan', 'whole wheat pasta', 'olive oi
l'}), support=0.007998933475536596, ordered_statistics=[OrderedS
tatistic(items_base=frozenset({'nan', 'whole wheat pasta'}), ite
ms_add=frozenset({'olive oil'}), confidence=0.2714932126696833,
lift=4.122410097642296)]), RelationRecord(items=frozenset({'na
n', 'pasta', 'shrimp'}), support=0.005065991201173177, ordered_s
tatistics=[OrderedStatistic(items_base=frozenset({'nan', 'past
a'}), items_add=frozenset({'shrimp'}), confidence=0.322033898305
0847, lift=4.506672147735896)]), RelationRecord(items=frozenset
({'pancakes', 'spaghetti', 'olive oil'}), support=0.005065991201
173177, ordered_statistics=[OrderedStatistic(items_base=frozense
t({'pancakes', 'spaghetti'}), items_add=frozenset({'olive oi
l'}), confidence=0.20105820105820105, lift=3.052910052910052
6)]), RelationRecord(items=frozenset({'tomatoes', 'spaghetti',
'olive oil'}), support=0.004399413411545127, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'tomatoes', 'olive oi
l'}), items_add=frozenset({'spaghetti'}), confidence=0.611111111
1111112, lift=3.5099115194827295), OrderedStatistic(items_base=f

```
rozenset({'tomatoes', 'spaghetti'}), items_add=frozenset({'olive
oil'}), confidence=0.21019108280254778, lift=3.1915856520281
6)]), RelationRecord(items=frozenset({'tomatoes', 'whole wheat r
ice', 'spaghetti'}), support=0.0030662578322890282, ordered_stat
istics=[OrderedStatistic(items_base=frozenset({'whole wheat ric
e', 'spaghetti'}), items_add=frozenset({'tomatoes'}), confidence
=0.2169811320754717, lift=3.1726617382029496)]), RelationRecord
(items=frozenset({'avocado', 'nan', 'spaghetti', 'milk'}), suppo
rt=0.003332888948140248, ordered_statistics=[OrderedStatistic(it
ems_base=frozenset({'avocado', 'nan', 'spaghetti'}), items_add=f
rozenset({'milk'}), confidence=0.41666666666666663, lift=3.21544
9245541838)]), RelationRecord(items=frozenset({'milk', 'nan', 'b
urgers', 'cake'}), support=0.0037328356219170776, ordered_statis
tics=[OrderedStatistic(items_base=frozenset({'nan', 'milk', 'cak
e'}), items_add=frozenset({'burgers'}), confidence=0.27999999999
999997, lift=3.211437308868501)]), RelationRecord(items=frozense
t({'nan', 'chocolate', 'turkey', 'burgers'}), support=0.00306625
78322890282, ordered_statistics=[OrderedStatistic(items_base=fro
zenset({'nan', 'chocolate', 'turkey'}), items_add=frozenset({'bu
rgers'}), confidence=0.27058823529411763, lift=3.103489836301492
7)]), RelationRecord(items=frozenset({'milk', 'nan', 'turkey',
'burgers'}), support=0.003199573390214638, ordered_statistics=[O
rderedStatistic(items_base=frozenset({'nan', 'turkey', 'milk'}),
items_add=frozenset({'burgers'}), confidence=0.2823529411764706,
lift=3.2384241770102533)]), RelationRecord(items=frozenset({'tom
atoes', 'nan', 'frozen vegetables', 'cake'}), support=0.00306625
78322890282, ordered_statistics=[OrderedStatistic(items_base=fro
zenset({'nan', 'frozen vegetables', 'cake'}), items_add=frozense
t({'tomatoes'}), confidence=0.2987012987012987, lift=4.367560314
928736), OrderedStatistic(items_base=frozenset({'tomatoes', 'na
n', 'cake'}), items_add=frozenset({'frozen vegetables'}), confid
ence=0.36507936507936506, lift=3.8300144300144296)]), RelationRe
cord(items=frozenset({'nan', 'ground beef', 'spaghetti', 'cereal
s'}), support=0.0030662578322890282, ordered_statistics=[Ordered
Statistic(items_base=frozenset({'nan', 'ground beef', 'cereal
s'}), items_add=frozenset({'spaghetti'}), confidence=0.676470588
2352942, lift=3.8853031258445188), OrderedStatistic(items_base=f
rozenset({'nan', 'spaghetti', 'cereals'}), items_add=frozenset
({'ground beef'}), confidence=0.45999999999999996, lift=4.681763
907734057)]), RelationRecord(items=frozenset({'nan', 'ground bee
f', 'milk', 'chicken'}), support=0.0038661511798426876, ordered_
statistics=[OrderedStatistic(items_base=frozenset({'nan', 'groun
d beef', 'chicken'}), items_add=frozenset({'milk'}), confidence=
0.40845070422535207, lift=3.152046020981858)]), RelationRecord(i
tems=frozenset({'olive oil', 'nan', 'milk', 'chicken'}), support
=0.0035995200639914677, ordered_statistics=[OrderedStatistic(ite
ms_base=frozenset({'nan', 'milk', 'chicken'}), items_add=frozens
et({'olive oil'}), confidence=0.24324324324324323, lift=3.693456
614509246), OrderedStatistic(items_base=frozenset({'olive oil',
'nan', 'chicken'}), items_add=frozenset({'milk'}), confidence=0.
5, lift=3.858539094650206), OrderedStatistic(items_base=frozense
t({'nan', 'milk', 'olive oil'}), items_add=frozenset({'chicke
n'}), confidence=0.2109375, lift=3.51609375)]), RelationRecord(i
```

```
tems=frozenset({'olive oil', 'nan', 'spaghetti', 'chicken'}), su
pport=0.0034662045060658577, ordered_statistics=[OrderedStatisti
c(items_base=frozenset({'nan', 'spaghetti', 'chicken'}), items_a
dd=frozenset({'olive oil'}), confidence=0.20155038759689922, lif
t=3.0603835169318647)]), RelationRecord(items=frozenset({'minera
l water', 'ground beef', 'eggs', 'chocolate'}), support=0.003999
466737768298, ordered_statistics=[OrderedStatistic(items_base=fr
ozenset({'mineral water', 'eggs', 'chocolate'}), items_add=froze
nset({'ground beef'}), confidence=0.29702970297029707, lift=3.02
3093354111531)]), RelationRecord(items=frozenset({'mineral wate
r', 'ground beef', 'chocolate', 'frozen vegetables'}), support=
0.003332888948140248, ordered_statistics=[OrderedStatistic(items
_base=frozenset({'mineral water', 'chocolate', 'frozen vegetable
s'}), items_add=frozenset({'ground beef'}), confidence=0.3424657
5342465757, lift=3.4855300087358976), OrderedStatistic(items_bas
e=frozenset({'mineral water', 'ground beef', 'chocolate'}), item
s_add=frozenset({'frozen vegetables'}), confidence=0.30487804878
048785, lift=3.1984478935698455)]), RelationRecord(items=frozens
et({'ground beef', 'chocolate', 'frozen vegetables', 'spaghett
i'}), support=0.0030662578322890282, ordered_statistics=[Ordered
Statistic(items_base=frozenset({'ground beef', 'chocolate', 'fro
zen vegetables'}), items_add=frozenset({'spaghetti'}), confidenc
e=0.5348837209302326, lift=3.0721001460165964), OrderedStatistic
(items_base=frozenset({'chocolate', 'frozen vegetables', 'spaghe
tti'}), items_add=frozenset({'ground beef'}), confidence=0.38983
05084745763, lift=3.967596531978015), OrderedStatistic(items_bas
e=frozenset({'ground beef', 'chocolate', 'spaghetti'}), items_ad
d=frozenset({'frozen vegetables'}), confidence=0.333333333333333
37, lift=3.4969696969696975)]), RelationRecord(items=frozenset
({'mineral water', 'chocolate', 'frozen vegetables', 'milk'}), s
upport=0.003999466737768298, ordered_statistics=[OrderedStatisti
c(items_base=frozenset({'mineral water', 'chocolate', 'frozen ve
getables'}), items_add=frozenset({'milk'}), confidence=0.4109589
041095891, lift=3.1714019956029094)]), RelationRecord(items=froz
enset({'chocolate', 'frozen vegetables', 'spaghetti', 'milk'}),
support=0.0034662045060658577, ordered_statistics=[OrderedStatis
tic(items_base=frozenset({'chocolate', 'frozen vegetables', 'spa
ghetti'}), items_add=frozenset({'milk'}), confidence=0.440677966
10169485, lift=3.4007463207086555), OrderedStatistic(items_base=
frozenset({'chocolate', 'spaghetti', 'milk'}), items_add=frozens
et({'frozen vegetables'}), confidence=0.3170731707317073, lift=
3.3263858093126384)]), RelationRecord(items=frozenset({'mineral
water', 'chocolate', 'frozen vegetables', 'shrimp'}), support=0.
003199573390214638, ordered_statistics=[OrderedStatistic(items_b
ase=frozenset({'mineral water', 'chocolate', 'frozen vegetable
s'}), items_add=frozenset({'shrimp'}), confidence=0.328767123287
67127, lift=4.600899611531385), OrderedStatistic(items_base=froz
enset({'mineral water', 'chocolate', 'shrimp'}), items_add=froze
nset({'frozen vegetables'}), confidence=0.4210526315789474, lift
=4.417224880382776)]), RelationRecord(items=frozenset({'nan', 'c
hocolate', 'frozen vegetables', 'shrimp'}), support=0.0053326223
17024397, ordered_statistics=[OrderedStatistic(items_base=frozen
set({'nan', 'chocolate', 'frozen vegetables'}), items_add=frozen
```

set({'shrimp'}), confidence=0.23255813953488375, lift=3.2545123221103784), OrderedStatistic(items_base=frozenset({'nan', 'chocolate', 'shrimp'}), items_add=frozenset({'frozen vegetables'}), confidence=0.29629629629629634, lift=3.1084175084175087)]), RelationRecord(items=frozenset({'ground beef', 'herb & pepper', 'chocolate', 'nan'}), support=0.003999466737768298, ordered_statistics=[OrderedStatistic(items_base=frozenset({'herb & pepper', 'nan', 'chocolate'}), items_add=frozenset({'ground beef'}), confidence=0.4411764705882354, lift=4.4901827759597746)]), RelationRecord(items=frozenset({'soup', 'nan', 'chocolate', 'milk'}), support=0.003999466737768298, ordered_statistics=[OrderedStatistic(items_base=frozenset({'nan', 'chocolate', 'soup'}), items_add=frozenset({'milk'}), confidence=0.3947368421052632, lift=3.0462150747238472)]), RelationRecord(items=frozenset({'mineral water', 'chocolate', 'spaghetti', 'olive oil'}), support=0.0038661511798426876, ordered_statistics=[OrderedStatistic(items_base=frozenset({'mineral water', 'chocolate', 'spaghetti'}), items_add=frozenset({'olive oil'}), confidence=0.2436974789915966, lift=3.700353825740822)]), RelationRecord(items=frozenset({'mineral water', 'spaghetti', 'chocolate', 'shrimp'}), support=0.0034662045060658577, ordered_statistics=[OrderedStatistic(items_base=frozenset({'mineral water', 'chocolate', 'spaghetti'}), items_add=frozenset({'shrimp'}), confidence=0.21848739495798317, lift=3.057600652201178 3)]), RelationRecord(items=frozenset({'nan', 'ground beef', 'cooking oil', 'spaghetti'}), support=0.004799360085321957, ordered_statistics=[OrderedStatistic(items_base=frozenset({'nan', 'ground beef', 'cooking oil'}), items_add=frozenset({'spaghetti'}), confidence=0.5714285714285714, lift=3.2819951870487856), OrderedStatistic(items_base=frozenset({'nan', 'spaghetti', 'cooking oil'}), items_add=frozenset({'ground beef'}), confidence=0.3025210084033613, lift=3.0789824749438446)]), RelationRecord(items=frozenset({'mineral water', 'eggs', 'frozen vegetables', 'milk'}), support=0.0037328356219170776, ordered_statistics=[OrderedStatistic(items_base=frozenset({'mineral water', 'eggs', 'frozen vegetables'}), items_add=frozenset({'milk'}), confidence=0.411764705882353, lift=3.177620430888405)]), RelationRecord(items=frozenset({'herb & pepper', 'eggs', 'ground beef', 'nan'}), support=0.0041327822956939075, ordered_statistics=[OrderedStatistic(items_base=frozenset({'eggs', 'ground beef', 'nan'}), items_add=frozenset({'herb & pepper'}), confidence=0.2066666666666667, lift=4.178454627133872), OrderedStatistic(items_base=frozenset({'herb & pepper', 'eggs', 'nan'}), items_add=frozenset({'ground beef'}), confidence=0.3297872340425532, lift=3.3564912381997174)]), RelationRecord(items=frozenset({'eggs', 'nan', 'spaghetti', 'red wine'}), support=0.0037328356219170776, ordered_statistics=[OrderedStatistic(items_base=frozenset({'eggs', 'nan', 'red wine'}), items_add=frozenset({'spaghetti'}), confidence=0.5283018867924528, lift=3.0342974370828397)]), RelationRecord(items=frozenset({'french fries', 'herb & pepper', 'nan', 'ground beef'}), support=0.003199573390214638, ordered_statistics=[OrderedStatistic(items_base=frozenset({'french fries', 'nan', 'ground beef'}), items_add=frozenset({'herb & pepper'}), confidence=0.23076923076923078, lift=4.665768194070081), OrderedStatistic(items_base=frozenset({'french

```
fries', 'herb & pepper', 'nan'}), items_add=frozenset({'ground b
eef'}), confidence=0.46153846153846156, lift=4.69742198100407
1)]), RelationRecord(items=frozenset({'mineral water', 'frozen s
moothie', 'spaghetti', 'milk'}), support=0.003199573390214638, o
rdered_statistics=[OrderedStatistic(items_base=frozenset({'miner
al water', 'frozen smoothie', 'spaghetti'}), items_add=frozenset
({'milk'}), confidence=0.4705882352941177, lift=3.63156620672960
6), OrderedStatistic(items_base=frozenset({'mineral water', 'spa
ghetti', 'milk'}), items_add=frozenset({'frozen smoothie'}), con
fidence=0.2033898305084746, lift=3.211846565566459)]), RelationR
ecord(items=frozenset({'tomatoes', 'nan', 'frozen vegetables',
'green tea'}), support=0.003332888948140248, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'nan', 'frozen vegetable
s', 'green tea'}), items_add=frozenset({'tomatoes'}), confidence
=0.2314814814814815, lift=3.38468341635983)]), RelationRecord(it
ems=frozenset({'mineral water', 'ground beef', 'frozen vegetable
s', 'milk'}), support=0.0037328356219170776, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'mineral water', 'ground
beef', 'frozen vegetables'}), items_add=frozenset({'milk'}), con
fidence=0.40579710144927533, lift=3.1315679608755294), OrderedSt
atistic(items_base=frozenset({'mineral water', 'frozen vegetable
s', 'milk'}), items_add=frozenset({'ground beef'}), confidence=
0.3373493975903614, lift=3.4334570302921317), OrderedStatistic(i
tems_base=frozenset({'mineral water', 'ground beef', 'milk'}), i
tems_add=frozenset({'frozen vegetables'}), confidence=0.33734939
75903614, lift=3.539101861993428)]), RelationRecord(items=frozen
set({'ground beef', 'frozen vegetables', 'spaghetti', 'milk'}),
support=0.0030662578322890282, ordered_statistics=[OrderedStatis
tic(items_base=frozenset({'ground beef', 'frozen vegetables', 'm
ilk'}), items_add=frozenset({'spaghetti'}), confidence=0.5348837
209302326, lift=3.0721001460165964), OrderedStatistic(items_base
=frozenset({'spaghetti', 'frozen vegetables', 'milk'}), items_ad
d=frozenset({'ground beef'}), confidence=0.3709677419354839, lif
t=3.7756160546242397), OrderedStatistic(items_base=frozenset({'g
round beef', 'spaghetti', 'milk'}), items_add=frozenset({'frozen
vegetables'}), confidence=0.3150684931506849, lift=3.30535491905
3549)]), RelationRecord(items=frozenset({'mineral water', 'groun
d beef', 'frozen vegetables', 'spaghetti'}), support=0.004399413
411545127, ordered_statistics=[OrderedStatistic(items_base=froze
nset({'mineral water', 'spaghetti', 'frozen vegetables'}), items
_add=frozenset({'ground beef'}), confidence=0.3666666666666667,
lift=3.7318407960199007)]), RelationRecord(items=frozenset({'na
n', 'ground beef', 'frozen vegetables', 'spaghetti'}), support=
0.008665511265164644, ordered_statistics=[OrderedStatistic(items
_base=frozenset({'nan', 'spaghetti', 'frozen vegetables'}), item
s_add=frozenset({'ground beef'}), confidence=0.3110047846889952
3, lift=3.165328208890303)]), RelationRecord(items=frozenset({'m
ineral water', 'frozen vegetables', 'milk', 'olive oil'}), suppo
rt=0.003332888948140248, ordered_statistics=[OrderedStatistic(it
ems_base=frozenset({'mineral water', 'frozen vegetables', 'mil
k'}), items_add=frozenset({'olive oil'}), confidence=0.301204819
27710846, lift=4.573557387444516), OrderedStatistic(items_base=f
rozenset({'mineral water', 'frozen vegetables', 'olive oil'}), i
```

```
tems_add=frozenset({'milk'}), confidence=0.5102040816326531, lif
t=3.937284790459394), OrderedStatistic(items_base=frozenset({'mi
neral water', 'milk', 'olive oil'}), items_add=frozenset({'froze
n vegetables'}), confidence=0.39062500000000006, lift=4.09801136
3636364)]), RelationRecord(items=frozenset({'mineral water', 'fr
ozen vegetables', 'milk', 'soup'}), support=0.003066257832289028
2, ordered_statistics=[OrderedStatistic(items_base=frozenset({'m
ineral water', 'frozen vegetables', 'milk'}), items_add=frozense
t({'soup'}), confidence=0.27710843373493976, lift=5.484407286136
631), OrderedStatistic(items_base=frozenset({'soup', 'frozen veg
etables', 'milk'}), items_add=frozenset({'mineral water'}), conf
idence=0.7666666666666666, lift=3.21631245339299), OrderedStatis
tic(items_base=frozenset({'mineral water', 'frozen vegetables',
'soup'}), items_add=frozenset({'milk'}), confidence=0.6052631578
947368, lift=4.670863114576565), OrderedStatistic(items_base=fro
zenset({'mineral water', 'milk', 'soup'}), items_add=frozenset
({'frozen vegetables'}), confidence=0.35937500000000006, lift=3.
7701704545454553)]), RelationRecord(items=frozenset({'mineral wa
ter', 'spaghetti', 'frozen vegetables', 'milk'}), support=0.0045
32728969470737, ordered_statistics=[OrderedStatistic(items_base=
frozenset({'mineral water', 'spaghetti', 'milk'}), items_add=fro
zenset({'frozen vegetables'}), confidence=0.28813559322033894, l
ift=3.0228043143297376)]), RelationRecord(items=frozenset({'na
n', 'frozen vegetables', 'milk', 'olive oil'}), support=0.004799
360085321957, ordered_statistics=[OrderedStatistic(items_base=fr
ozenset({'nan', 'frozen vegetables', 'milk'}), items_add=frozens
et({'olive oil'}), confidence=0.20338983050847456, lift=3.088314
005352364), OrderedStatistic(items_base=frozenset({'nan', 'froze
n vegetables', 'olive oil'}), items_add=frozenset({'milk'}), con
fidence=0.4235294117647058, lift=3.2684095860566447)]), Relation
Record(items=frozenset({'soup', 'nan', 'frozen vegetables', 'mil
k'}), support=0.003999466737768298, ordered_statistics=[OrderedS
tatistic(items_base=frozenset({'nan', 'frozen vegetables', 'sou
p'}), items_add=frozenset({'milk'}), confidence=0.5, lift=3.8585
39094650206)]), RelationRecord(items=frozenset({'tomatoes', 'na
n', 'frozen vegetables', 'milk'}), support=0.004132782295693907
5, ordered_statistics=[OrderedStatistic(items_base=frozenset({'t
omatoes', 'nan', 'milk'}), items_add=frozenset({'frozen vegetabl
es'}), confidence=0.29523809523809524, lift=3.097316017316017
2)]), RelationRecord(items=frozenset({'mineral water', 'nan', 'f
rozen vegetables', 'shrimp'}), support=0.007199040127982935, ord
ered_statistics=[OrderedStatistic(items_base=frozenset({'mineral
water', 'nan', 'shrimp'}), items_add=frozenset({'frozen vegetabl
es'}), confidence=0.3068181818181818, lift=3.218801652892562)]),
RelationRecord(items=frozenset({'mineral water', 'spaghetti', 'f
rozen vegetables', 'shrimp'}), support=0.003332888948140248, ord
ered_statistics=[OrderedStatistic(items_base=frozenset({'mineral
water', 'spaghetti', 'frozen vegetables'}), items_add=frozenset
({'shrimp'}), confidence=0.2777777777777778, lift=3.887334162520
7294), OrderedStatistic(items_base=frozenset({'mineral water',
'spaghetti', 'shrimp'}), items_add=frozenset({'frozen vegetable
s'}), confidence=0.39062500000000006, lift=4.098011363636364)]),
RelationRecord(items=frozenset({'mineral water', 'spaghetti', 'f
```

```
rozen vegetables', 'tomatoes'}), support=0.0030662578322890282,
ordered_statistics=[OrderedStatistic(items_base=frozenset({'mine
ral water', 'spaghetti', 'frozen vegetables'}), items_add=frozen
set({'tomatoes'}), confidence=0.2555555555555556, lift=3.7366904
916612524), OrderedStatistic(items_base=frozenset({'mineral wate
r', 'frozen vegetables', 'tomatoes'}), items_add=frozenset({'spa
ghetti'}), confidence=0.5227272727272727, lift=3.002279688152582
6), OrderedStatistic(items_base=frozenset({'mineral water', 'spa
ghetti', 'tomatoes'}), items_add=frozenset({'frozen vegetable
s'}), confidence=0.32857142857142857, lift=3.447012987012987)]),
RelationRecord(items=frozenset({'nan', 'spaghetti', 'frozen vege
tables', 'olive oil'}), support=0.005732568990801226, ordered_st
atistics=[OrderedStatistic(items_base=frozenset({'nan', 'spaghet
ti', 'frozen vegetables'}), items_add=frozenset({'olive oil'}),
confidence=0.20574162679425836, lift=3.1240241752707125)]), Rela
tionRecord(items=frozenset({'nan', 'spaghetti', 'frozen vegetabl
es', 'shrimp'}), support=0.005999200106652446, ordered_statistic
s=[OrderedStatistic(items_base=frozenset({'nan', 'spaghetti', 'f
rozen vegetables'}), items_add=frozenset({'shrimp'}), confidence
=0.21531100478468898, lift=3.0131489680782684)]), RelationRecord
(items=frozenset({'tomatoes', 'nan', 'frozen vegetables', 'shrim
p'}), support=0.003999466737768298, ordered_statistics=[OrderedS
tatistic(items_base=frozenset({'nan', 'frozen vegetables', 'shri
mp'}), items_add=frozenset({'tomatoes'}), confidence=0.240000000
00000002, lift=3.5092397660818717), OrderedStatistic(items_base=
frozenset({'tomatoes', 'nan', 'frozen vegetables'}), items_add=f
rozenset({'shrimp'}), confidence=0.2479338842975207, lift=3.4696
866905143704), OrderedStatistic(items_base=frozenset({'tomatoe
s', 'nan', 'shrimp'}), items_add=frozenset({'frozen vegetable
s'}), confidence=0.35714285714285715, lift=3.746753246753246
7)]), RelationRecord(items=frozenset({'tomatoes', 'nan', 'spaghe
tti', 'frozen vegetables'}), support=0.006665777896280496, order
ed_statistics=[OrderedStatistic(items_base=frozenset({'nan', 'sp
aghetti', 'frozen vegetables'}), items_add=frozenset({'tomatoe
s'}), confidence=0.23923444976076558, lift=3.4980460188216425),
OrderedStatistic(items_base=frozenset({'tomatoes', 'nan', 'spagh
etti'}), items_add=frozenset({'frozen vegetables'}), confidence=
0.3184713375796179, lift=3.341053850607991)]), RelationRecord(it
ems=frozenset({'grated cheese', 'ground beef', 'spaghetti', 'na
n'}), support=0.005332622317024397, ordered_statistics=[OrderedS
tatistic(items_base=frozenset({'grated cheese', 'nan', 'spaghett
i'}), items_add=frozenset({'ground beef'}), confidence=0.3225806
451612903, lift=3.283144395325426)]), RelationRecord(items=froze
nset({'tomatoes', 'nan', 'ground beef', 'green tea'}), support=
0.0030662578322890282, ordered_statistics=[OrderedStatistic(item
s_base=frozenset({'nan', 'ground beef', 'green tea'}), items_add
=frozenset({'tomatoes'}), confidence=0.2072072072072072, lift=3.
0297490472929067)]), RelationRecord(items=frozenset({'herb & pep
per', 'nan', 'ground beef', 'milk'}), support=0.0035995200639914
677, ordered_statistics=[OrderedStatistic(items_base=frozenset
({'herb & pepper', 'nan', 'milk'}), items_add=frozenset({'ground
beef'}), confidence=0.3913043478260869, lift=3.982596896938233
5)]), RelationRecord(items=frozenset({'mineral water', 'herb & p
```

```
epper', 'nan', 'ground beef'}), support=0.006665777896280496, or
dered_statistics=[OrderedStatistic(items_base=frozenset({'minera
l water', 'herb & pepper', 'nan'}), items_add=frozenset({'ground
beef'}), confidence=0.39062500000000006, lift=3.97568266621438
3)]), RelationRecord(items=frozenset({'herb & pepper', 'nan', 'g
round beef', 'spaghetti'}), support=0.006399146780429276, ordere
d_statistics=[OrderedStatistic(items_base=frozenset({'herb & pep
per', 'nan', 'spaghetti'}), items_add=frozenset({'ground bee
f'}), confidence=0.3934426229508197, lift=4.004359721511667)]),
RelationRecord(items=frozenset({'nan', 'ground beef', 'milk', 'o
live oil'}), support=0.004932675643247567, ordered_statistics=[O
rderedStatistic(items_base=frozenset({'nan', 'ground beef', 'mil
k'}), items_add=frozenset({'olive oil'}), confidence=0.224242424
24242427, lift=3.40494417862839)]), RelationRecord(items=frozens
et({'soup', 'nan', 'ground beef', 'milk'}), support=0.0039994667
37768298, ordered_statistics=[OrderedStatistic(items_base=frozen
set({'nan', 'ground beef', 'soup'}), items_add=frozenset({'mil
k'}), confidence=0.4109589041095891, lift=3.1714019956029094)]),
RelationRecord(items=frozenset({'mineral water', 'ground beef',
'spaghetti', 'olive oil'}), support=0.0030662578322890282, order
ed_statistics=[OrderedStatistic(items_base=frozenset({'mineral w
ater', 'spaghetti', 'olive oil'}), items_add=frozenset({'ground
beef'}), confidence=0.2987012987012987, lift=3.040106433593543
5)]), RelationRecord(items=frozenset({'mineral water', 'ground b
eef', 'tomatoes', 'spaghetti'}), support=0.0030662578322890282,
ordered_statistics=[OrderedStatistic(items_base=frozenset({'mine
ral water', 'ground beef', 'tomatoes'}), items_add=frozenset({'s
paghetti'}), confidence=0.5609756097560976, lift=3.2219586897247
23), OrderedStatistic(items_base=frozenset({'mineral water', 'sp
aghetti', 'tomatoes'}), items_add=frozenset({'ground beef'}), co
nfidence=0.32857142857142857, lift=3.344117076952898)]), Relatio
nRecord(items=frozenset({'nan', 'ground beef', 'spaghetti', 'pep
per'}), support=0.003332888948140248, ordered_statistics=[Ordere
dStatistic(items_base=frozenset({'nan', 'spaghetti', 'pepper'}),
items_add=frozenset({'ground beef'}), confidence=0.3378378378378
3783, lift=3.4384282518610876)]), RelationRecord(items=frozenset
({'spaghetti', 'nan', 'ground beef', 'shrimp'}), support=0.00599
9200106652446, ordered_statistics=[OrderedStatistic(items_base=f
rozenset({'nan', 'ground beef', 'shrimp'}), items_add=frozenset
({'spaghetti'}), confidence=0.5232558139534884, lift=3.005315360
233627)]), RelationRecord(items=frozenset({'tomato sauce', 'na
n', 'ground beef', 'spaghetti'}), support=0.0030662578322890282,
ordered_statistics=[OrderedStatistic(items_base=frozenset({'toma
to sauce', 'nan', 'ground beef'}), items_add=frozenset({'spaghet
ti'}), confidence=0.5750000000000001, lift=3.3025076569678413),
OrderedStatistic(items_base=frozenset({'tomato sauce', 'nan', 's
paghetti'}), items_add=frozenset({'ground beef'}), confidence=0.
4893617021276596, lift=4.980599901844742)]), RelationRecord(item
s=frozenset({'mineral water', 'spaghetti', 'milk', 'olive oi
l'}), support=0.003332888948140248, ordered_statistics=[OrderedS
tatistic(items_base=frozenset({'mineral water', 'spaghetti', 'mi
lk'}), items_add=frozenset({'olive oil'}), confidence=0.21186440
6779661, lift=3.216993755575379)]), RelationRecord(items=frozens
```

```
et({'mineral water', 'spaghetti', 'tomatoes', 'milk'}), support=
0.003332888948140248, ordered_statistics=[OrderedStatistic(items
_base=frozenset({'mineral water', 'spaghetti', 'milk'}), items_a
dd=frozenset({'tomatoes'}), confidence=0.211864406779661, lift=
3.0978458387022165)]), RelationRecord(items=frozenset({'nan', 's
hrimp', 'milk', 'olive oil'}), support=0.003199573390214638, ord
ered_statistics=[OrderedStatistic(items_base=frozenset({'nan',
'shrimp', 'olive oil'}), items_add=frozenset({'milk'}), confiden
ce=0.39999999999999997, lift=3.0868312757201646)]), RelationReco
rd(items=frozenset({'soup', 'nan', 'milk', 'olive oil'}), suppor
t=0.0035995200639914677, ordered_statistics=[OrderedStatistic(it
ems_base=frozenset({'nan', 'milk', 'olive oil'}), items_add=froz
enset({'soup'}), confidence=0.2109375, lift=4.174781497361478),
OrderedStatistic(items_base=frozenset({'soup', 'nan', 'milk'}),
items_add=frozenset({'olive oil'}), confidence=0.236842105263157
88, lift=3.5962603878116344), OrderedStatistic(items_base=frozen
set({'nan', 'soup', 'olive oil'}), items_add=frozenset({'mil
k'}), confidence=0.4029850746268656, lift=3.1098673300165833)]),
RelationRecord(items=frozenset({'nan', 'spaghetti', 'milk', 'oli
ve oil'}), support=0.007199040127982935, ordered_statistics=[Ord
eredStatistic(items_base=frozenset({'nan', 'spaghetti', 'mil
k'}), items_add=frozenset({'olive oil'}), confidence=0.203007518
79699247, lift=3.0825089038385434)]), RelationRecord(items=froze
nset({'soup', 'nan', 'tomatoes', 'milk'}), support=0.00306625783
22890282, ordered_statistics=[OrderedStatistic(items_base=frozen
set({'tomatoes', 'nan', 'milk'}), items_add=frozenset({'soup'}),
confidence=0.21904761904761905, lift=4.335293378565146), Ordered
Statistic(items_base=frozenset({'tomatoes', 'nan', 'soup'}), ite
ms_add=frozenset({'milk'}), confidence=0.44230769230769235, lift
=3.4133230452674903)]), RelationRecord(items=frozenset({'whole w
heat pasta', 'nan', 'spaghetti', 'milk'}), support=0.00399946673
7768298, ordered_statistics=[OrderedStatistic(items_base=frozens
et({'nan', 'spaghetti', 'whole wheat pasta'}), items_add=frozens
et({'milk'}), confidence=0.4545454545454546, lift=3.507762813318
3696)]), RelationRecord(items=frozenset({'mineral water', 'nan',
'soup', 'olive oil'}), support=0.005199306759098787, ordered_sta
tistics=[OrderedStatistic(items_base=frozenset({'mineral water',
'nan', 'soup'}), items_add=frozenset({'olive oil'}), confidence=
0.22543352601156072, lift=3.4230301186492245)]), RelationRecord
(items=frozenset({'mineral water', 'nan', 'whole wheat pasta',
'olive oil'}), support=0.0038661511798426876, ordered_statistics
=[OrderedStatistic(items_base=frozenset({'mineral water', 'nan',
'whole wheat pasta'}), items_add=frozenset({'olive oil'}), confi
dence=0.4027777777777778, lift=6.115862573099416)]), RelationRec
ord(items=frozenset({'pancakes', 'nan', 'spaghetti', 'olive oi
l'}), support=0.005065991201173177, ordered_statistics=[OrderedS
tatistic(items_base=frozenset({'pancakes', 'nan', 'spaghetti'}),
items_add=frozenset({'olive oil'}), confidence=0.201058201058201
05, lift=3.0529100529100526)]), RelationRecord(items=frozenset
({'tomatoes', 'nan', 'spaghetti', 'olive oil'}), support=0.00439
9413411545127, ordered_statistics=[OrderedStatistic(items_base=f
rozenset({'tomatoes', 'nan', 'olive oil'}), items_add=frozenset
({'spaghetti'}), confidence=0.6111111111111112, lift=3.509911519
```

4827295), OrderedStatistic(items_base=frozenset({'tomatoes', 'na
n', 'spaghetti'}), items_add=frozenset({'olive oil'}), confidenc
e=0.21019108280254778, lift=3.19158565202816)]), RelationRecord
(items=frozenset({'tomatoes', 'nan', 'spaghetti', 'whole wheat r
ice'}), support=0.0030662578322890282, ordered_statistics=[Order
edStatistic(items_base=frozenset({'whole wheat rice', 'nan', 'sp
aghetti'}), items_add=frozenset({'tomatoes'}), confidence=0.2169
811320754717, lift=3.1726617382029496)]), RelationRecord(items=f
rozenset({'mineral water', 'nan', 'ground beef', 'chocolate', 'e
ggs'}), support=0.003999466737768298, ordered_statistics=[Ordere
dStatistic(items_base=frozenset({'mineral water', 'eggs', 'choco
late', 'nan'}), items_add=frozenset({'ground beef'}), confidence
=0.29702970297029707, lift=3.023093354111531)]), RelationRecord
(items=frozenset({'mineral water', 'nan', 'ground beef', 'frozen
vegetables', 'chocolate'}), support=0.003332888948140248, ordere
d_statistics=[OrderedStatistic(items_base=frozenset({'mineral wa
ter', 'nan', 'chocolate', 'frozen vegetables'}), items_add=froze
nset({'ground beef'}), confidence=0.34246575342465757, lift=3.48
55300087358976), OrderedStatistic(items_base=frozenset({'mineral
water', 'ground beef', 'nan', 'chocolate'}), items_add=frozenset
({'frozen vegetables'}), confidence=0.30487804878048785, lift=3.
1984478935698455)]), RelationRecord(items=frozenset({'nan', 'gro
und beef', 'frozen vegetables', 'chocolate', 'spaghetti'}), supp
ort=0.0030662578322890282, ordered_statistics=[OrderedStatistic
(items_base=frozenset({'ground beef', 'nan', 'chocolate', 'froze
n vegetables'}), items_add=frozenset({'spaghetti'}), confidence=
0.5348837209302326, lift=3.0721001460165964), OrderedStatistic(i
tems_base=frozenset({'nan', 'chocolate', 'frozen vegetables', 's
paghetti'}), items_add=frozenset({'ground beef'}), confidence=0.
3898305084745763, lift=3.967596531978015), OrderedStatistic(item
s_base=frozenset({'ground beef', 'nan', 'chocolate', 'spaghett
i'}), items_add=frozenset({'frozen vegetables'}), confidence=0.3
3333333333333337, lift=3.4969696969696975)]), RelationRecord(ite
ms=frozenset({'mineral water', 'nan', 'frozen vegetables', 'choc
olate', 'milk'}), support=0.003999466737768298, ordered_statisti
cs=[OrderedStatistic(items_base=frozenset({'mineral water', 'na
n', 'chocolate', 'frozen vegetables'}), items_add=frozenset({'mi
lk'}), confidence=0.4109589041095891, lift=3.171401995602909
4)]), RelationRecord(items=frozenset({'nan', 'frozen vegetable
s', 'chocolate', 'milk', 'spaghetti'}), support=0.00346620450606
58577, ordered_statistics=[OrderedStatistic(items_base=frozenset
({'nan', 'chocolate', 'frozen vegetables', 'spaghetti'}), items_
add=frozenset({'milk'}), confidence=0.44067796610169485, lift=3.
4007463207086555), OrderedStatistic(items_base=frozenset({'nan',
'chocolate', 'spaghetti', 'milk'}), items_add=frozenset({'frozen
vegetables'}), confidence=0.3170731707317073, lift=3.32638580931
26384)]), RelationRecord(items=frozenset({'mineral water', 'na
n', 'frozen vegetables', 'chocolate', 'shrimp'}), support=0.0031
99573390214638, ordered_statistics=[OrderedStatistic(items_base=
frozenset({'mineral water', 'nan', 'chocolate', 'frozen vegetabl
es'}), items_add=frozenset({'shrimp'}), confidence=0.32876712328
767127, lift=4.600899611531385), OrderedStatistic(items_base=fro
zenset({'mineral water', 'nan', 'chocolate', 'shrimp'}), items_a

```
dd=frozenset({'frozen vegetables'}), confidence=0.42105263157894
74, lift=4.417224880382776)]), RelationRecord(items=frozenset
({'mineral water', 'nan', 'chocolate', 'spaghetti', 'olive oi
l'}), support=0.0038661511798426876, ordered_statistics=[Ordered
Statistic(items_base=frozenset({'mineral water', 'nan', 'chocola
te', 'spaghetti'}), items_add=frozenset({'olive oil'}), confiden
ce=0.2436974789915966, lift=3.700353825740822)]), RelationRecord
(items=frozenset({'mineral water', 'nan', 'chocolate', 'spaghett
i', 'shrimp'}), support=0.0034662045060658577, ordered_statistic
s=[OrderedStatistic(items_base=frozenset({'mineral water', 'na
n', 'chocolate', 'spaghetti'}), items_add=frozenset({'shrimp'}),
confidence=0.21848739495798317, lift=3.0576006522011783)]), Rela
tionRecord(items=frozenset({'mineral water', 'nan', 'frozen vege
tables', 'milk', 'eggs'}), support=0.0037328356219170776, ordere
d_statistics=[OrderedStatistic(items_base=frozenset({'mineral wa
ter', 'eggs', 'nan', 'frozen vegetables'}), items_add=frozenset
({'milk'}), confidence=0.411764705882353, lift=3.17762043088840
5)]), RelationRecord(items=frozenset({'mineral water', 'nan', 'm
ilk', 'frozen smoothie', 'spaghetti'}), support=0.00319957339021
4638, ordered_statistics=[OrderedStatistic(items_base=frozenset
({'mineral water', 'frozen smoothie', 'nan', 'spaghetti'}), item
s_add=frozenset({'milk'}), confidence=0.4705882352941177, lift=
3.631566206729606), OrderedStatistic(items_base=frozenset({'mine
ral water', 'nan', 'spaghetti', 'milk'}), items_add=frozenset
({'frozen smoothie'}), confidence=0.2033898305084746, lift=3.211
846565566459)]), RelationRecord(items=frozenset({'mineral wate
r', 'nan', 'ground beef', 'frozen vegetables', 'milk'}), support
=0.0037328356219170776, ordered_statistics=[OrderedStatistic(ite
ms_base=frozenset({'mineral water', 'nan', 'ground beef', 'froze
n vegetables'}), items_add=frozenset({'milk'}), confidence=0.405
79710144927533, lift=3.1315679608755294), OrderedStatistic(items
_base=frozenset({'mineral water', 'nan', 'frozen vegetables', 'm
ilk'}), items_add=frozenset({'ground beef'}), confidence=0.33734
93975903614, lift=3.4334570302921317), OrderedStatistic(items_ba
se=frozenset({'mineral water', 'nan', 'ground beef', 'milk'}), i
tems_add=frozenset({'frozen vegetables'}), confidence=0.33734939
75903614, lift=3.539101861993428)]), RelationRecord(items=frozen
set({'nan', 'ground beef', 'frozen vegetables', 'milk', 'spaghet
ti'}), support=0.0030662578322890282, ordered_statistics=[Ordere
dStatistic(items_base=frozenset({'nan', 'ground beef', 'frozen v
egetables', 'milk'}), items_add=frozenset({'spaghetti'}), confid
ence=0.5348837209302326, lift=3.0721001460165964), OrderedStatis
tic(items_base=frozenset({'nan', 'spaghetti', 'frozen vegetable
s', 'milk'}), items_add=frozenset({'ground beef'}), confidence=
0.3709677419354839, lift=3.7756160546242397), OrderedStatistic(i
tems_base=frozenset({'nan', 'ground beef', 'spaghetti', 'mil
k'}), items_add=frozenset({'frozen vegetables'}), confidence=0.3
150684931506849, lift=3.305354919053549)]), RelationRecord(items
=frozenset({'mineral water', 'nan', 'ground beef', 'frozen veget
ables', 'spaghetti'}), support=0.004399413411545127, ordered_sta
tistics=[OrderedStatistic(items_base=frozenset({'mineral water',
'nan', 'spaghetti', 'frozen vegetables'}), items_add=frozenset
({'ground beef'}), confidence=0.3666666666666667, lift=3.7318407
```

```
960199007)]), RelationRecord(items=frozenset({'mineral water',
'nan', 'frozen vegetables', 'milk', 'olive oil'}), support=0.003
33288948140248, ordered_statistics=[OrderedStatistic(items_base
=frozenset({'mineral water', 'nan', 'frozen vegetables', 'mil
k'}), items_add=frozenset({'olive oil'}), confidence=0.301204819
27710846, lift=4.573557387444516), OrderedStatistic(items_base=f
rozenset({'mineral water', 'nan', 'frozen vegetables', 'olive oi
l'}), items_add=frozenset({'milk'}), confidence=0.51020408163265
31, lift=3.937284790459394), OrderedStatistic(items_base=frozens
et({'mineral water', 'nan', 'milk', 'olive oil'}), items_add=fro
zenset({'frozen vegetables'}), confidence=0.39062500000000006, l
ift=4.098011363636364)]), RelationRecord(items=frozenset({'miner
al water', 'nan', 'frozen vegetables', 'milk', 'soup'}), support
=0.0030662578322890282, ordered_statistics=[OrderedStatistic(ite
ms_base=frozenset({'mineral water', 'nan', 'frozen vegetables',
'milk'}), items_add=frozenset({'soup'}), confidence=0.2771084337
3493976, lift=5.484407286136631), OrderedStatistic(items_base=fr
ozenset({'soup', 'nan', 'frozen vegetables', 'milk'}), items_add
=frozenset({'mineral water'}), confidence=0.7666666666666666, li
ft=3.21631245339299), OrderedStatistic(items_base=frozenset({'mi
neral water', 'nan', 'frozen vegetables', 'soup'}), items_add=fr
ozenset({'milk'}), confidence=0.6052631578947368, lift=4.6708631
14576565), OrderedStatistic(items_base=frozenset({'mineral wate
r', 'nan', 'milk', 'soup'}), items_add=frozenset({'frozen vegeta
bles'}), confidence=0.35937500000000006, lift=3.770170454545455
3)]), RelationRecord(items=frozenset({'mineral water', 'nan', 'f
rozen vegetables', 'milk', 'spaghetti'}), support=0.004532728969
470737, ordered_statistics=[OrderedStatistic(items_base=frozense
t({'mineral water', 'nan', 'spaghetti', 'milk'}), items_add=froz
enset({'frozen vegetables'}), confidence=0.28813559322033894, li
ft=3.0228043143297376)]), RelationRecord(items=frozenset({'miner
al water', 'nan', 'frozen vegetables', 'spaghetti', 'shrimp'}),
support=0.003332888948140248, ordered_statistics=[OrderedStatist
ic(items_base=frozenset({'mineral water', 'nan', 'spaghetti', 'f
rozen vegetables'}), items_add=frozenset({'shrimp'}), confidence
=0.2777777777777778, lift=3.8873341625207294), OrderedStatistic
(items_base=frozenset({'mineral water', 'nan', 'spaghetti', 'shr
imp'}), items_add=frozenset({'frozen vegetables'}), confidence=
0.39062500000000006, lift=4.098011363636364)]), RelationRecord(i
tems=frozenset({'mineral water', 'tomatoes', 'nan', 'frozen vege
tables', 'spaghetti'}), support=0.0030662578322890282, ordered_s
tatistics=[OrderedStatistic(items_base=frozenset({'mineral wate
r', 'nan', 'spaghetti', 'frozen vegetables'}), items_add=frozens
et({'tomatoes'}), confidence=0.2555555555555556, lift=3.73669049
16612524), OrderedStatistic(items_base=frozenset({'mineral wate
r', 'nan', 'frozen vegetables', 'tomatoes'}), items_add=frozense
t({'spaghetti'}), confidence=0.5227272727272727, lift=3.00227968
81525826), OrderedStatistic(items_base=frozenset({'mineral wate
r', 'nan', 'spaghetti', 'tomatoes'}), items_add=frozenset({'froz
en vegetables'}), confidence=0.32857142857142857, lift=3.4470129
87012987)]), RelationRecord(items=frozenset({'mineral water', 'n
an', 'ground beef', 'spaghetti', 'olive oil'}), support=0.003066
2578322890282, ordered_statistics=[OrderedStatistic(items_base=f
```

```
rozenset({'mineral water', 'nan', 'spaghetti', 'olive oil'}), it
ems_add=frozenset({'ground beef'}), confidence=0.298701298701298
7, lift=3.0401064335935435)]), RelationRecord(items=frozenset
({'mineral water', 'tomatoes', 'nan', 'ground beef', 'spaghett
i'}), support=0.0030662578322890282, ordered_statistics=[Ordered
Statistic(items_base=frozenset({'mineral water', 'nan', 'ground
beef', 'tomatoes'}), items_add=frozenset({'spaghetti'}), confide
nce=0.5609756097560976, lift=3.221958689724723), OrderedStatisti
c(items_base=frozenset({'mineral water', 'nan', 'spaghetti', 'to
matoes'}), items_add=frozenset({'ground beef'}), confidence=0.32
857142857142857, lift=3.344117076952898)]), RelationRecord(items
=frozenset({'mineral water', 'nan', 'milk', 'spaghetti', 'olive
oil'}), support=0.003332888948140248, ordered_statistics=[Ordere
dStatistic(items_base=frozenset({'mineral water', 'nan', 'spaghe
tti', 'milk'}), items_add=frozenset({'olive oil'}), confidence=
0.211864406779661, lift=3.216993755575379)]), RelationRecord(ite
ms=frozenset({'mineral water', 'tomatoes', 'nan', 'milk', 'spagh
etti'}), support=0.003332888948140248, ordered_statistics=[Order
edStatistic(items_base=frozenset({'mineral water', 'nan', 'spagh
etti', 'milk'}), items_add=frozenset({'tomatoes'}), confidence=
0.211864406779661, lift=3.0978458387022165)])]
```

In [ ]:

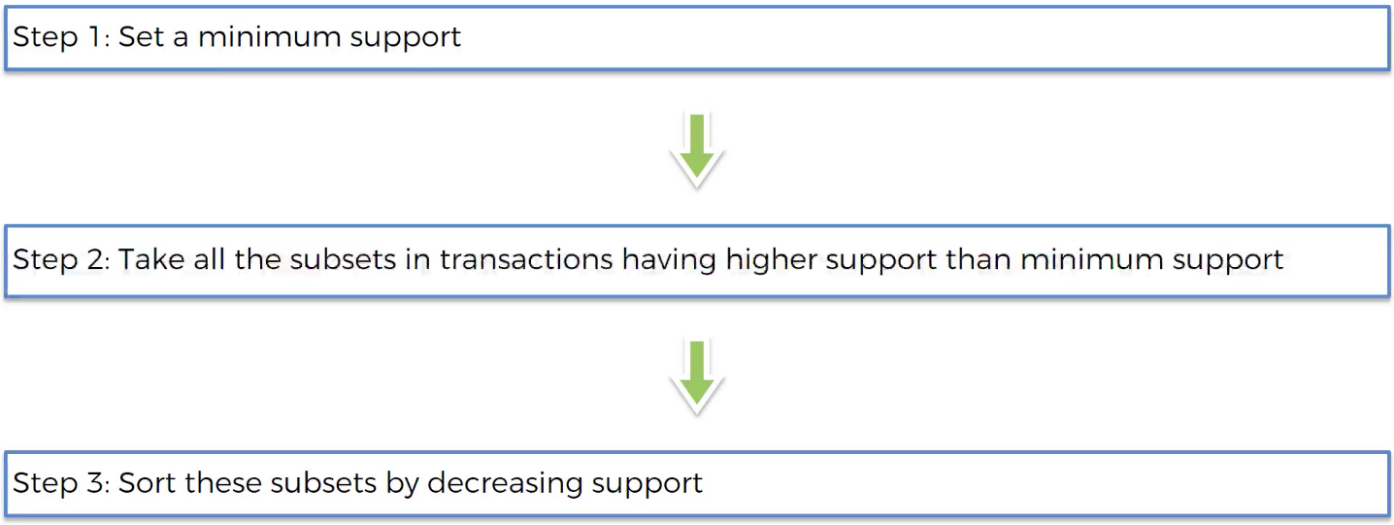
In [ ]:

```
#=======================
# Section 25 (29): Eclat
#=======================
```

In [ ]:

In [43]:

```
### Intuition:


## We look at sets instead of individual item frequency, as we are only work
ing with support.
```



In [ ]:

```
#* Eclat is a simpler version of the Apriori, and can be see as a little obs
olete, so it is not available in python.
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```
#################################################
# Section 26 (31): Part 6: REINFORCEMEMT LEARNING
#################################################
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```
# Reinforcement Learning is a branch of Machine Learning, also called Online
Learning or Interactive Learning.
# It is used to solve interacting problems where the data observed up to tim
e t is considered to decide which
    # action to take at time t + 1.
# Desired outcomes provide the AI with reward, undesired with punishment.


# We implement the following Reinforcement Learning models:

    # - Upper Confidence Bound (UCB)
    # - Thompson Sampling
```

In [ ]:

```

```

In [ ]:

```
#============================================
# Section 27 (32):  Upper Confidence Bound (UCB)
#============================================
```

In [ ]:

```

```

In [3]:

```
### The Multi-Armed Bandit Problem:



## Used when less/no funds for A/B testing. Or in replacement.
```

- We have d arms. For example, arms are ads that we display to users each time they connect to a web page.
- Each time a user connects to this web page, that makes a round.
- At each round n, we choose one ad to display to the user.
- At each round n, ad i gives reward $r_i(n) \in \{0, 1\}$: $r_i(n) = 1$ if the user clicked on the ad i, 0 if the user didn't.
- Our goal is to maximize the total reward we get over many rounds.

In [4]:

```
# Method 1:
#----------
### Upper Confidence Bound (UCB) Intuition:



# UCB is a deterministic algorithm.



## most common example is advetising.
```

Step 1. At each round n, we consider two numbers for each ad i:

- $N_i(n)$ - the number of times the ad i was selected up to round n,
- $R_i(n)$ - the sum of rewards of the ad i up to round n.

Step 2. From these two numbers we compute:

- the average reward of ad i up to round n

$$\bar{r}_i(n) = \frac{R_i(n)}{N_i(n)}$$

- the confidence interval $[\bar{r}_i(n) - \Delta_i(n), \bar{r}_i(n) + \Delta_i(n)]$ at round n with

$$\Delta_i(n) = \sqrt{\frac{3}{2}\frac{\log(n)}{N_i(n)}}$$

Step 3. We select the ad i that has the maximum UCB $\bar{r}_i(n) + \Delta_i(n)$.

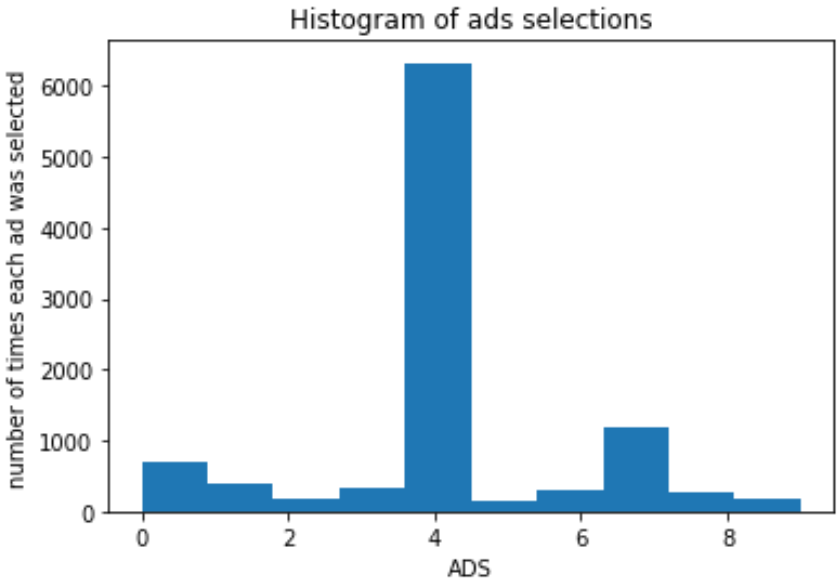
In [ ]:

In [2]:

```
## Upper Confidence Bound (UCB)

# (optimizing the adds Click Through Rate (CTR))

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
path = 'Machine Learning A-Z Template Folder/Part 6 - Reinforcement Learnin
g/Section 32 - Upper Confidence Bound (UCB)/'
dataset = pd.read_csv(path + 'Ads_CTR_Optimisation.csv')



#######


## Implementing UCB:

# first 10 rounds, just iterate (maybe randomly) through the ads one, as no
 prior info.
import math
#STEP 1:
N = 10000   # total number of rounds: N
d = 10      # total number of ads: d
ads_selected = []
# (list of) number of times ad i was selected upto round n:
numbers_of_selections = [0] * d    # we have 'd' ads
# (list of) Sum of rewards of ad i up to round n:
sums_of_rewards = [0] * d
total_reward = 0
#STEP 2:
for n in range(N):
    max_upper_bound = 0
    ad = 0
    for i in range(d):
        if numbers_of_selections[i] > 0:
            # average reward of ad i up to round n:
            average_reward = sums_of_rewards[i] / numbers_of_selections[i]
            # UB of confidence interval at round n:
            delta_i = math.sqrt(3/2 * (math.log(n+1) / numbers_of_selections
[i]))
            # Upper Bound (UB):
            upper_bound = average_reward + delta_i
        else:
            upper_bound = 1e400   # 10^400
# STEP 3: select the ad i with max. UCB:
        # get max. of the UB for ads at each round
```

```
        if upper_bound > max_upper_bound:
            max_upper_bound = upper_bound
            ad = i    # index of max UB ad.
    # append the selected ad to ads_selected:
    ads_selected.append(ad)
    # update numbers_of_selections (for the selected ad):
    numbers_of_selections[ad] += 1
    # update sums_of_rewards:
    reward = dataset.values[n, ad]   # row: round; Col: ad selected
    sums_of_rewards[ad] += reward
    total_reward += reward



print(total_reward)




####### My version:









## Visualizing the results:
plt.hist(ads_selected)
plt.title("Histogram of ads selections")
plt.xlabel('ADS')
plt.ylabel('number of times each ad was selected')
plt.show()
```

```
2178
```



In [ ]:

```
```

In [5]:

```
#====================================
# Section 28 (33):  Thompson Sampling
#====================================
```

In [ ]:

```
```

In [ ]:

```
```

Bayesian Inference

- Ad i gets rewards $\boldsymbol{y}$ from Bernoulli distribution $p(\boldsymbol{y}|\theta_i) \sim \mathcal{B}(\theta_i)$.
- θ_i is unknown but we set its uncertainty by assuming it has a uniform distribution $p(\theta_i) \sim \mathcal{U}([0,1])$, which is the prior distribution.
- Bayes Rule: we approach θ_i by the posterior distribution

$$\underbrace{p(\theta_i|\mathbf{y})}_{\text{posterior distribution}} = \frac{p(\boldsymbol{y}|\theta_i)p(\theta_i)}{\int p(\boldsymbol{y}|\theta_i)p(\theta_i)d\theta_i} \propto \underbrace{p(\boldsymbol{y}|\theta_i)}_{\text{likelihood function}} \times \underbrace{p(\theta_i)}_{\text{prior distribution}}$$

- We get $p(\theta_i|\mathbf{y}) \sim \beta(\text{number of successes} + 1, \text{number of failures} + 1)$
- At each round n we take a random draw $\theta_i(n)$ from this posterior distribution $p(\theta_i|\boldsymbol{y})$, for each ad i.
- At each round n we select the ad i that has the highest $\theta_i(n)$.

In [ ]:

In [ ]:

```
# Method 2:
#----------
### Thompson Sampling Intuition:



## Creating our own distribution (for each bandit) using samples (not the ac
tual bandit distribution).

## TS is a probalistic algorithm.
```

In [ ]:

```
## TS steps:
```

Step 1. At each round n, we consider two numbers for each ad i:

- $N_i^1(n)$ - the number of times the ad i got reward 1 up to round n,
- $N_i^0(n)$ - the number of times the ad i got reward 0 up to round n.

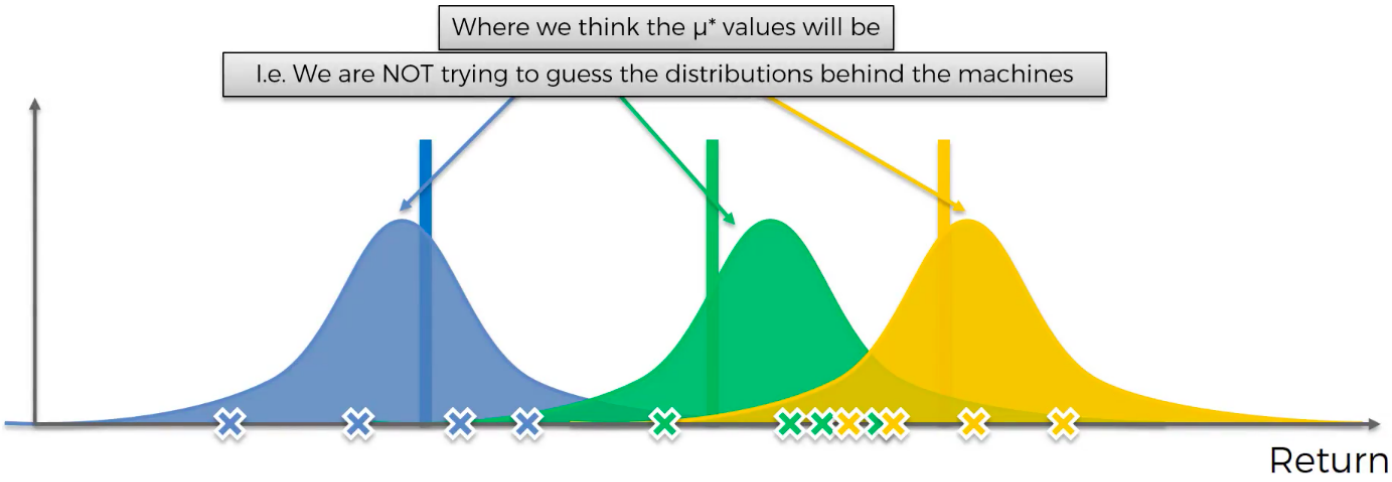
Step 2. For each ad i, we take a random draw from the distribution below:

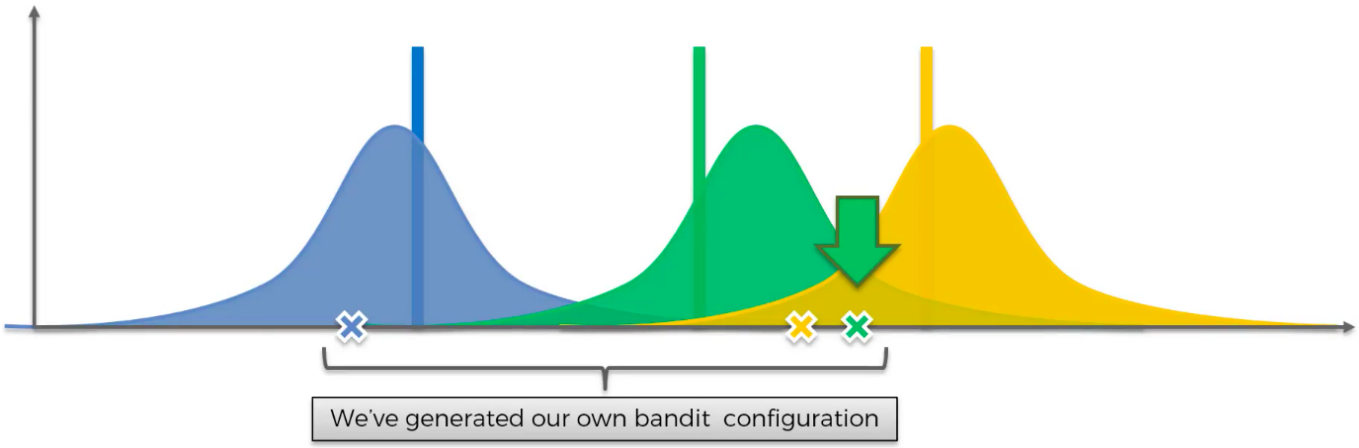
$$\theta_i(n) = \beta(N_i^1(n) + 1, N_i^0(n) + 1)$$

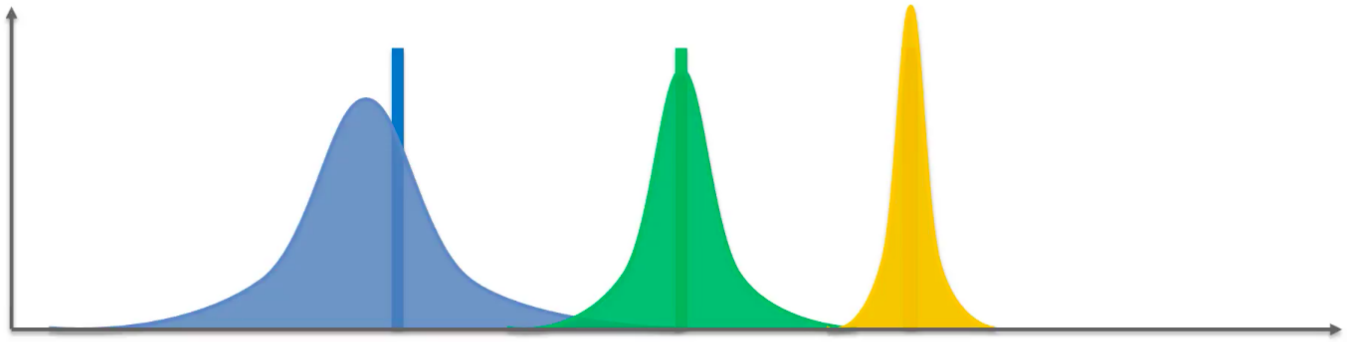
Step 3. We select the ad that has the highest $\theta_i(n)$.

In [ ]:

```

```


Where we think the μ* values will be
I.e. We are NOT trying to guess the distributions behind the machines
Return


We've generated our own bandit configuration



In [ ]:

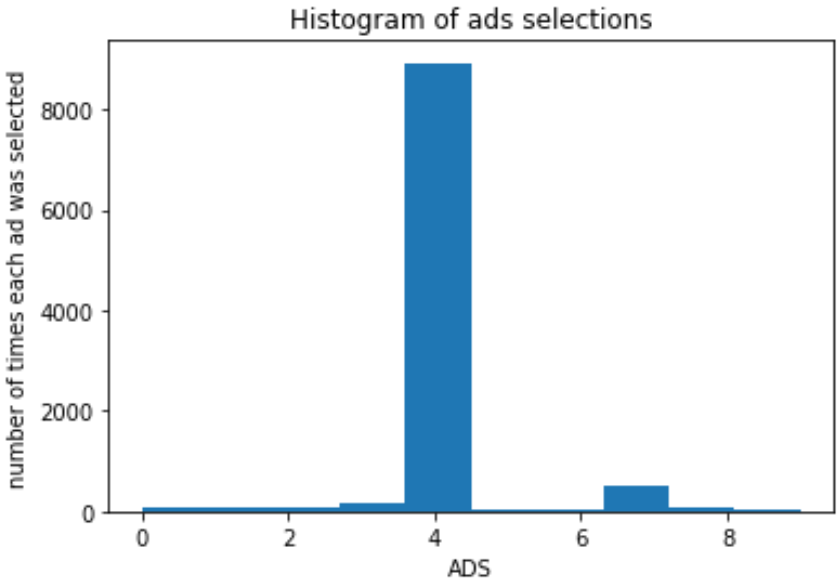
In [ ]:

In [17]:

```
## Thompson Sampling:



# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
path = 'Machine Learning A-Z Template Folder/Part 6 - Reinforcement Learnin
g/Section 33 - Thompson Sampling/'
dataset = pd.read_csv(path + 'Ads_CTR_Optimisation.csv')


#######


## Implementing Thompson Sampling:

import random
#STEP 1:
N = 10000   # total number of rounds: N
d = 10      # total number of ads: d
# (list of) number of times ad i got reward 1 upto round n:
numbers_of_rewards_1 = [0] * d    # we have 'd' ads
# (list of) number of times ad i got reward 0 upto round n:
numbers_of_rewards_0 = [0] * d

#STEP 2: for each ad, random draw from distribution:
total_reward = 0
ads_selected = []
for n in range(N):
    max_random = 0
    ad = 0
    for i in range(d):
        random_beta = random.betavariate(numbers_of_rewards_1[i] + 1, number
s_of_rewards_0[i] + 1)

# STEP 3: select the ad i with highest theta:
        if random_beta > max_random:
            max_random = random_beta
            ad = i    # index of max theta.
    # append the selected ad to ads_selected:
    ads_selected.append(ad)
    # update sums_of_rewards:
    reward = dataset.values[n, ad]   # row: round; Col: ad selected
    if reward == 1:
        numbers_of_rewards_1[ad] += 1
    else:
```

```
        numbers_of_rewards_0[ad] += 1
    total_reward += reward



print(total_reward)




## Visualizing the results:
plt.hist(ads_selected)
plt.title("Histogram of ads selections")
plt.xlabel('ADS')
plt.ylabel('number of times each ad was selected')
plt.show()
```

```
2564
```

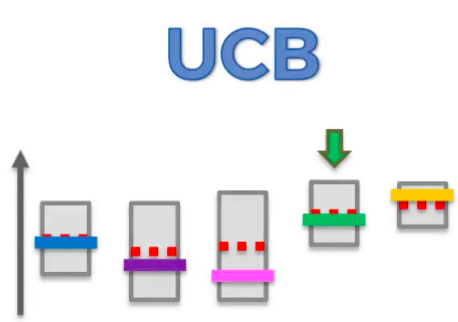


In [ ]:

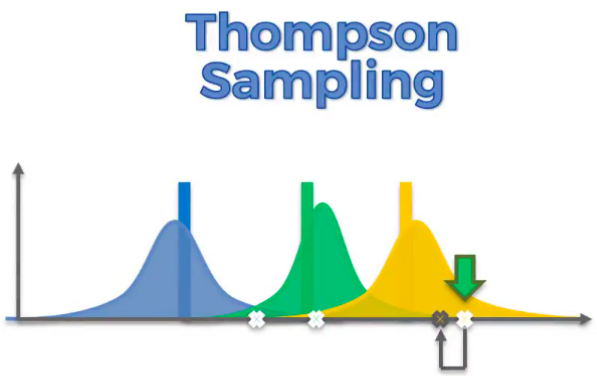
```

```

In [7]:

```
### Algorithm Comparison: UCB vs Thompson Sampling
#-----------------------------------------------
```



- Deterministic
- Requires update at every round



- Probabilistic
- Can accommodate delayed feedback
- Better empirical evidence

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
##########################################################
# Section 29 (35): Part 7: Natural Language Processing
##########################################################
```

In [ ]:

```
```

In [ ]:

```
```

In [18]:

```
## most of NLP algorithms are "Classification models", and they include:

    # - Logistic Regression,
    # - Naive Bayes,
    # - CART which is a model based on decision trees,
    # - Maximum Entropy again related to Decision Trees,
    # - Hidden Markov Models which are models based on Markov processes.


## well-known model in NLP is: the Bag of Words model.: It is a model used t
o preprocess the texts to classify
    # before fitting the classification algorithms on the observations conta
ining the texts.







# In this part:

    # - Clean texts to prepare them for the Machine Learning models,
    # - Create a Bag of Words model,
    # - Apply Machine Learning models onto this Bag of Worlds model.
```

In [ ]:

```

```

In [ ]:

```
#============================================================
# Section 29 (36): Natural Language Processing Algorithms
#============================================================
```

In [ ]:

```

```

In [19]:

```
## NLP Intuition:


## doing predictive anlysis, mostly on text.




## NLP Uses: -
```

In [20]:

```
# NLP Uses:
```

NLP Uses

- Sentiment analysis. Identifying the mood or subjective opinions within large amounts of text, including average sentiment and opinion mining.
- Use it to predict the genre of the book.
- Question Answering
- Use NLP to build a machine translator or a speech recognition system
- Document Summarization

In [ ]:

```
## NLP Libraries:

 # - Natural Language Toolkit - NKTK
 # - SpaCy
 # - Stanford NLP
 # - OpenNLP
```

In [ ]:

```

```

In [ ]:

```
## Popular NLP model: Bag of Words


# STEPS in 'Bag of Words'
-
```

In [ ]:

```
## Text Cleaning steps:

 # - Stemming, remove capitals, remove dots, articles, numbers, etc.
```

In [38]:

```
## ** Use "tsv" file as the text will (most likely) itself contain 'commas'.
(to keep things clear).



# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
path = 'Machine Learning A-Z Template Folder/Part 7 - Natural Language Proce
ssing/Section 36 - Natural Language Processing/'
dataset = pd.read_csv(path + 'Restaurant_Reviews.tsv', delimiter='\t', quoti
ng=3)  #*


## STEP 2:
## Cleaning the Texts: {walkthorugh of steps using one review}
import re
import nltk

# download stopword list:
nltk.download('stopwords')
# import:
from nltk.corpus import stopwords

from nltk.stem.porter import PorterStemmer

# Keep only aplhapets:
review = re.sub('[^a-zA-Z]', ' ', dataset['Review'][0])   # only keep letter
s (a-z).

# convert to lowercase:
review = review.lower()

# remove non significant words (articles, prepositions, etc):
review = review.split()  # making list of words from string
#review = [word for word in review if not word in set(stopwords.words('engli
sh'))] # comment after combine with next step

# stemming - only keep root of word:
ps = PorterStemmer()
review = [ps.stem(word) for word in review if not word in set(stopwords.word
s('english'))]

# join back into string (sep. by space):
review = ' '.join(review)

print(review)
```

```
#########


## STEP 2:
## Cleaning the Texts:
import re
import nltk
#nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer

corpus = []  # creating a corpus of 1000 clean review
for i in range(1000):
    review = re.sub('[^a-zA-Z]', ' ', dataset['Review'][i])   # only keep le
tters (a-z).
    review = review.lower()
    review = review.split()  # making list of words from string
    ps = PorterStemmer()
    review = [ps.stem(word) for word in review if not word in set(stopwords.
words('english'))]
    review = ' '.join(review)
    corpus.append(review)


print(review)




## STEP 3:
## Create bag of words model:
from sklearn.feature_extraction.text import CountVectorizer
CV = CountVectorizer(max_features=1500)  #* above cleaning steps are availab
le as params. of CV. (not flexible(not preffered))
X = CV.fit_transform(corpus).toarray()  #*
print(X.shape, X)  #**
y = dataset.iloc[:, 1].values


# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20,
random_state = 0)

# Fitting Naive Bayes to the Training set
```

```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(y_pred)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[nltk_data] Downloading package stopwords to /Users/nitin/nltk_d
ata...
[nltk_data]   Package stopwords is already up-to-date!

wow love place
wast enough life pour salt wound draw time took bring check
(1000, 1500) [[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
[1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 1 1 1 0 1 1
1 1 1 0 1
 0 1 1 1 1 1 0 0 0 1 1 0 0 1 1 1 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 1
1 1 1 1 1
 0 1 1 0 0 1 0 1 1 0 1 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 0 1 1 1 0 1
1 1 0 0 0
 1 0 1 1 0 1 1 1 1 1 0 1 1 0 0 1 1 0 1 1 1 0 0 1 1 1 1 1 1 0 1 1
0 1 0 1 1
 1 1 1 0 1 1 1 0 1 1 1 1 1 1 0 0 1 0 0 1 0 0 0 0 1 1 0 0 1 0 1 0 0
1 0 0 1 0
 1 0 1 0 1 1 0 1 1 1 0 1 1 1 1]
[[55 42]
 [12 91]]
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```
########################################
# Section 30 (38): Part 8: Deep Learning
########################################
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```
## Tasks employed:

 # - Artificial Neural Networks for Regression and Classification
 # - Convolutional Neural Networks for Computer Vision
 # - Recurrent Neural Networks for Time Series Analysis
 # - Self Organizing Maps for Feature Extraction
 # - Deep Boltzmann Machines for Recommendation Systems
 # - Auto Encoders for Recommendation Systems




## This part covers:

 # - Artificial Neural Networks for a Business Problem
 # - Convolutional Neural Networks for a Computer Vision task
```

In [ ]:

```

```

In [ ]:

```
#==========================================
# Section 31 (39): Artificial Neural Networks
#==========================================
```

In [ ]:

```

```

In [ ]:

```
### ACtivation functions:

 # - Threshold fn
 # - Sigmoid
 # - Rectifier
 # - Hyperbolic Tangent (tanh)
 # -



## * SGD helps in avoiding local minimas.
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
#############################################################
# Section 33 (42): Part 9: Dimensionality Reduction
#############################################################
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
## Two types of Dimensionality Reduction techniques:

    # - Feature Selection
        # eg: are Backward Elimination, Forward Selection, Bidirectional Eli
mination, Score Compariso, etc.
        # (- these covered in part 2)

    # - Feature Extraction




## In this part we will cover the following "Feature Extraction" techniques:

    # - Principal Component Analysis (PCA)
    # - Linear Discriminant Analysis (LDA)
    # - Kernel PCA
    # - Quadratic Discriminant Analysis (QDA)
```

In [ ]:

```
```

In [ ]:

```
#=============================================
# Section 34 (43): Principal Component Analysis (PCA)
#=============================================
```

In [ ]:

```
```

In [ ]:

```
## PCA Intuition:


# one of the most used 'Unsupervised algorithm'.

# Most popular Dimensionality Reduction algorithm.




## Uses:

 # - noise filtering
 # - feature extraction
 # - visualization
 # - stock market predictions
 # - gene data analysis







### PCA Goal:

 # - Identify patters in data
 # - Detect the correlation between variables




## ** To find list of Principal axis.

## ** Highly affected by the outliers in the data.

## ** The fact that the Dependent variable (DV) is not considered makes PCA
 an 'Unsupervised model'.
```

In [ ]:

```

```

Principal Component Analysis - PCA

- Standardize the data.
- Obtain the Eigenvectors and Eigenvalues from the covariance matrix or correlation matrix, or perform Singular Vector Decomposition.
- Sort eigenvalues in descending order and choose the k eigenvectors that correspond to the k largest eigenvalues where k is the number of dimensions of the new feature subspace ($k \leq d$)/.
- Construct the projection matrix $\mathbf{W}$ from the selected k eigenvectors.
- Transform the original dataset $\mathbf{X}$ via $\mathbf{W}$ to obtain a k-dimensional feature subspace $\mathbf{Y}$

https://plot.ly/ipython-notebooks/principal-component-analysis/

Machine Learning A-Z © Kirill Eremenko

In [ ]:

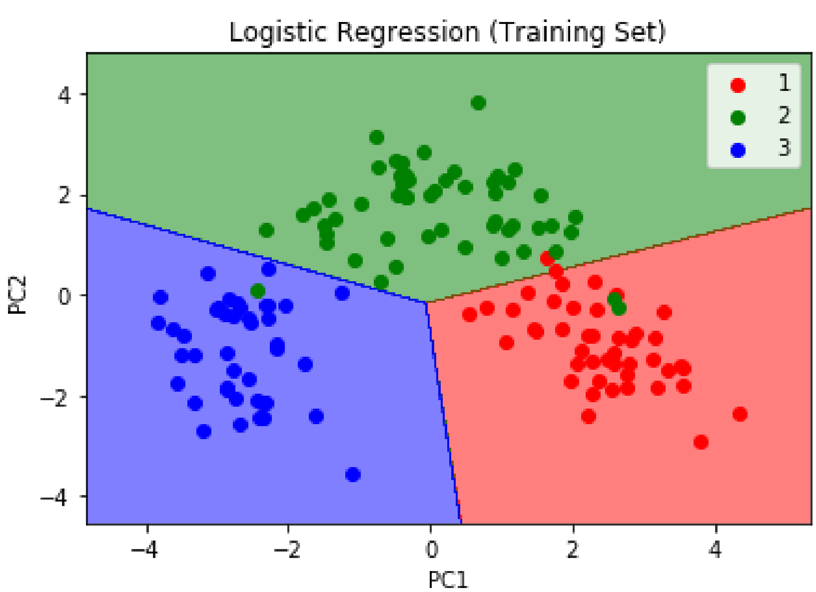
In [43]:

```
##PCA:


# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
path = 'Machine Learning A-Z Template Folder/Part 9 - Dimensionality Reducti
on/Section 43 - Principal Component Analysis (PCA)/'
dataset = pd.read_csv(path + 'Wine.csv')
X = dataset.iloc[:, 0:13].values
y = dataset.iloc[:, 13].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split  #*
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20,
random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)


##
###*  Applying PCA:
##

from sklearn.decomposition import PCA
pca = PCA(n_components=2)  #* use 'None' value to get variance for each var.
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
explained_variance = pca.explained_variance_ratio_
print(explained_variance)




#######

# fitting Logistic Regression to the training set:
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state=0)
classifier.fit(X_train, y_train)

# predicting the test set results:
y_pred = classifier.predict(X_test)
```

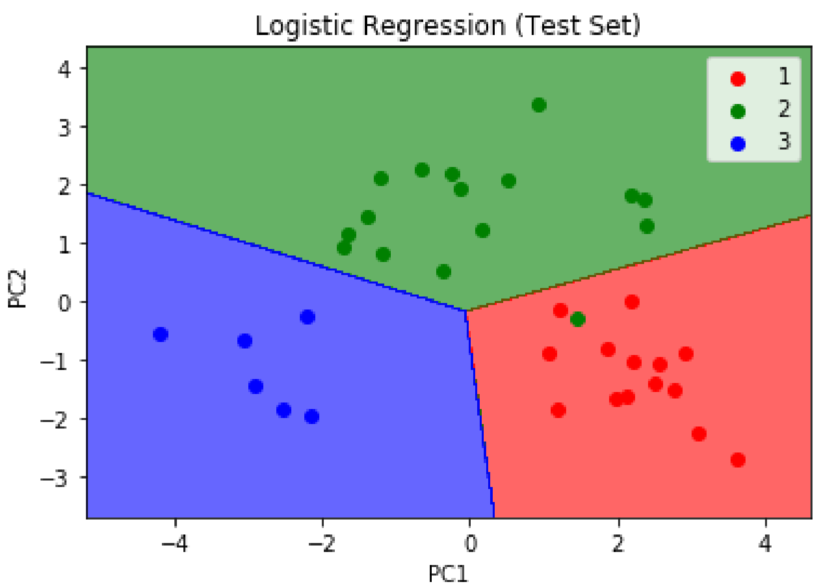
```
print(y_pred)

# making the Confusion Matrix:
from sklearn.metrics import confusion_matrix   #* here importing a Function
 and not Class (no capitals in name)
cm = confusion_matrix(y_true=y_test, y_pred=y_pred)
print(cm)

##* visualizing the Training set results:
from matplotlib.colors import ListedColormap  # to colorize all the datapoin
ts
X_set, y_set = X_train, y_train
# prepare grid with all the pixel points:
 # 'ravel': Return a contiguous flattened array.
 # 'meshgrid': Return coordinate matrices from coordinate vectors.
X1, X2 = np.meshgrid(np.arange(start = X_set[:,0].min() - 1, stop = X_set[:,
0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:,1].min() - 1, stop = X_set[:,
1].max() + 1, step = 0.01))
# apply classifier on all the pixel observation points and make countour b/w
the two regions:
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.50, cmap = ListedColormap(('red', 'green', 'blue')))
#* alpha is for Opaqueness.
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
# plot all the datapoints:
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],         ## * ???
                c = ListedColormap(('red', 'green', 'blue'))(i), label = j)
    # c : color, sequence, or sequence of color,
plt.title('Logistic Regression (Training Set)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.show()


##* visualizing the Test set results:
from matplotlib.colors import ListedColormap  # to colorize all the datapoin
ts
X_set, y_set = X_test, y_test
# prepare grid with all the pixel points:
 # 'ravel': Return a contiguous flattened array.
X1, X2 = np.meshgrid(np.arange(start = X_set[:,0].min() - 1, stop = X_set[:,
0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:,1].min() - 1, stop = X_set[:,
1].max() + 1, step = 0.01))
# apply classifier on all the pixel observation points and make countour b/w
the two regions:
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
```

```
            alpha = 0.60, cmap = ListedColormap(('red', 'green', 'blue')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

# plot all the datapoints:
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green', 'blue'))(i), label = j)
plt.title('Logistic Regression (Test Set)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.show()
```

```
[0.36884109 0.19318394]
[1 3 2 1 2 1 1 3 2 2 3 3 1 2 3 2 1 1 2 1 2 1 1 2 2 2 2 2 2 3 1 1
2 1 1 1]
[[14  0  0]
 [ 1 15  0]
 [ 0  0  6]]

/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logi
stic.py:433: FutureWarning: Default solver will be changed to 'l
bfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logi
stic.py:460: FutureWarning: Default multi_class will be changed
to 'auto' in 0.22. Specify the multi_class option to silence thi
s warning.
  "this warning.", FutureWarning)
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
```



```
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
```



In [ ]:

In [ ]:

```
#=====================================================
# Section 35 (44): Linear Discriminant Analysis (LDA)
#=====================================================
```

In [ ]:

In [ ]:

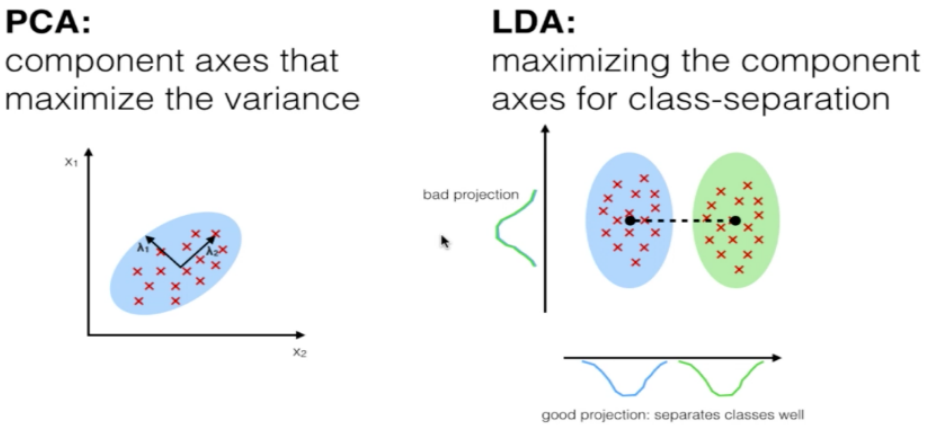
```
### LDA Intuition:


## Used in the pre-processing step for pattern classification.

## Goal is to project a dataset onto a lower-dimensional space.


## ** LDA differs from PCA because in addition to finding component axis, we
are interested in axis that
    # maximize the separation between multiple classes.


## ** Goal (LDA): to project a feature space onto a small subspace k while m
aintaining the class-discriminatory information.


## ** Both PCA and LDA are 'linear' transformation techniques.

## ** LDA is 'supervised' because of the relation to the dependent variable.

## ** LDA 'extracts' p new independent variables that separate the most the
 classes of the dependent variable.

## ** "Supervised": DV is considered in this method.
```

In [ ]:

```
### STEPS:
```

1. Compute the d-dimensional mean vectors for the different classes from the dataset.
2. Compute the scatter matrices (in-between-class and within-class scatter matrix).
3. Compute the eigenvectors ($\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_d$) and corresponding eigenvalues ($\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \ldots, \boldsymbol{\lambda}_d$) for the scatter matrices.
4. Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a $d \times k$ dimensional matrix $\boldsymbol{W}$ (where every column represents an eigenvector).
5. Use this $d \times k$ eigenvector matrix to transform the samples onto the new subspace. This can be summarized by the matrix multiplication: $\boldsymbol{Y} = \boldsymbol{X} \times \boldsymbol{W}$ (where $\boldsymbol{X}$ is a $n \times d$-dimensional matrix representing the n samples, and $\boldsymbol{y}$ are the transformed $n \times k$-dimensional samples in the new subspace).

https://sebastianraschka.com/Articles/2014_python_lda.html



LDA: https://sebastianraschka.com/Articles/2014_python_lda.html

In [ ]:

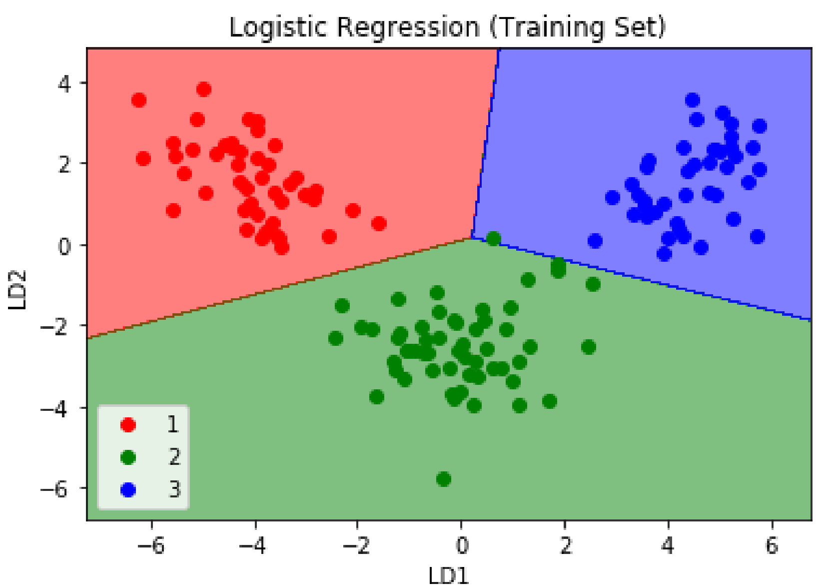
In [45]:

```
## LDA:


# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
path = 'Machine Learning A-Z Template Folder/Part 9 - Dimensionality Reducti
on/Section 44 - Linear Discriminant Analysis (LDA)/'
dataset = pd.read_csv(path + 'Wine.csv')
X = dataset.iloc[:, 0:13].values
y = dataset.iloc[:, 13].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split  #*
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20,
random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)


##
###*  Applying LDA:
##

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
lda = LDA(n_components=2)
X_train = lda.fit_transform(X_train, y_train)    #*  'Supervised'
X_test = lda.transform(X_test)




#######

# fitting Logistic Regression to the training set:
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state=0)
classifier.fit(X_train, y_train)

# predicting the test set results:
y_pred = classifier.predict(X_test)
print(y_pred)
```

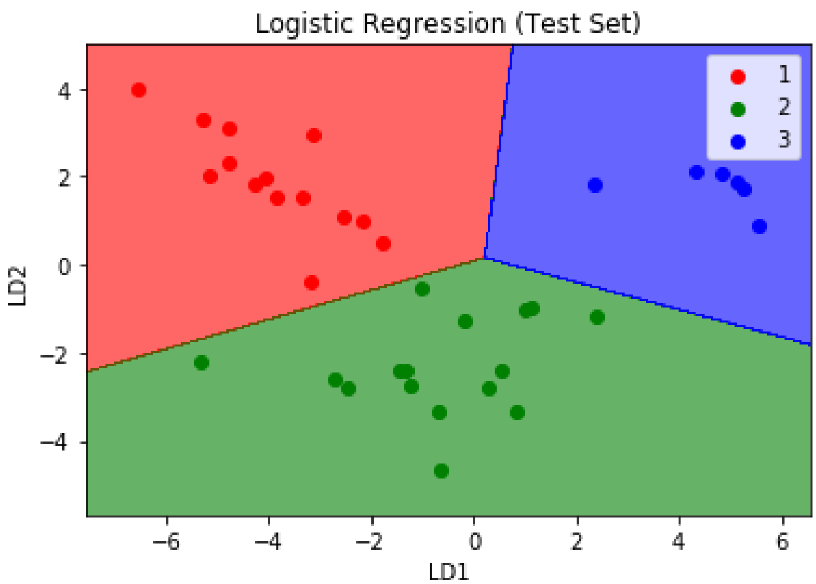
```
# making the Confusion Matrix:
from sklearn.metrics import confusion_matrix   #* here importing a Function
 and not Class (no capitals in name)
cm = confusion_matrix(y_true=y_test, y_pred=y_pred)
print(cm)

##* visualizing the Training set results:
from matplotlib.colors import ListedColormap  # to colorize all the datapoin
ts
X_set, y_set = X_train, y_train
# prepare grid with all the pixel points:
 # 'ravel': Return a contiguous flattened array.
 # 'meshgrid': Return coordinate matrices from coordinate vectors.
X1, X2 = np.meshgrid(np.arange(start = X_set[:,0].min() - 1, stop = X_set[:,
0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:,1].min() - 1, stop = X_set[:,
1].max() + 1, step = 0.01))
# apply classifier on all the pixel observation points and make countour b/w
the two regions:
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.50, cmap = ListedColormap(('red', 'green', 'blue')))
#* alpha is for Opaqueness.
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
# plot all the datapoints:
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],         ## * ???
                c = ListedColormap(('red', 'green', 'blue'))(i), label = j)
    # c : color, sequence, or sequence of color,
plt.title('Logistic Regression (Training Set)')
plt.xlabel('LD1')
plt.ylabel('LD2')
plt.legend()
plt.show()


##* visualizing the Test set results:
from matplotlib.colors import ListedColormap  # to colorize all the datapoin
ts
X_set, y_set = X_test, y_test
# prepare grid with all the pixel points:
 # 'ravel': Return a contiguous flattened array.
X1, X2 = np.meshgrid(np.arange(start = X_set[:,0].min() - 1, stop = X_set[:,
0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:,1].min() - 1, stop = X_set[:,
1].max() + 1, step = 0.01))
# apply classifier on all the pixel observation points and make countour b/w
the two regions:
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.60, cmap = ListedColormap(('red', 'green', 'blue')))
```

```
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

# plot all the datapoints:
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green', 'blue'))(i), label = j)
plt.title('Logistic Regression (Test Set)')
plt.xlabel('LD1')
plt.ylabel('LD2')
plt.legend()
plt.show()
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logi
stic.py:433: FutureWarning: Default solver will be changed to 'l
bfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logi
stic.py:460: FutureWarning: Default multi_class will be changed
to 'auto' in 0.22. Specify the multi_class option to silence thi
s warning.
  "this warning.", FutureWarning)

[1 3 2 1 2 2 1 3 2 2 3 3 1 2 3 2 1 1 2 1 2 1 1 2 2 2 2 2 2 3 1 1
2 1 1 1]
[[14  0  0]
 [ 0 16  0]
 [ 0  0  6]]

'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
```



```
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
```



In [ ]:

```

```

In [ ]:

```
#============================
# Section 36 (45): Kernel PCA
#============================





#* for non-linearly saparable data
```

In [ ]:

In [51]:

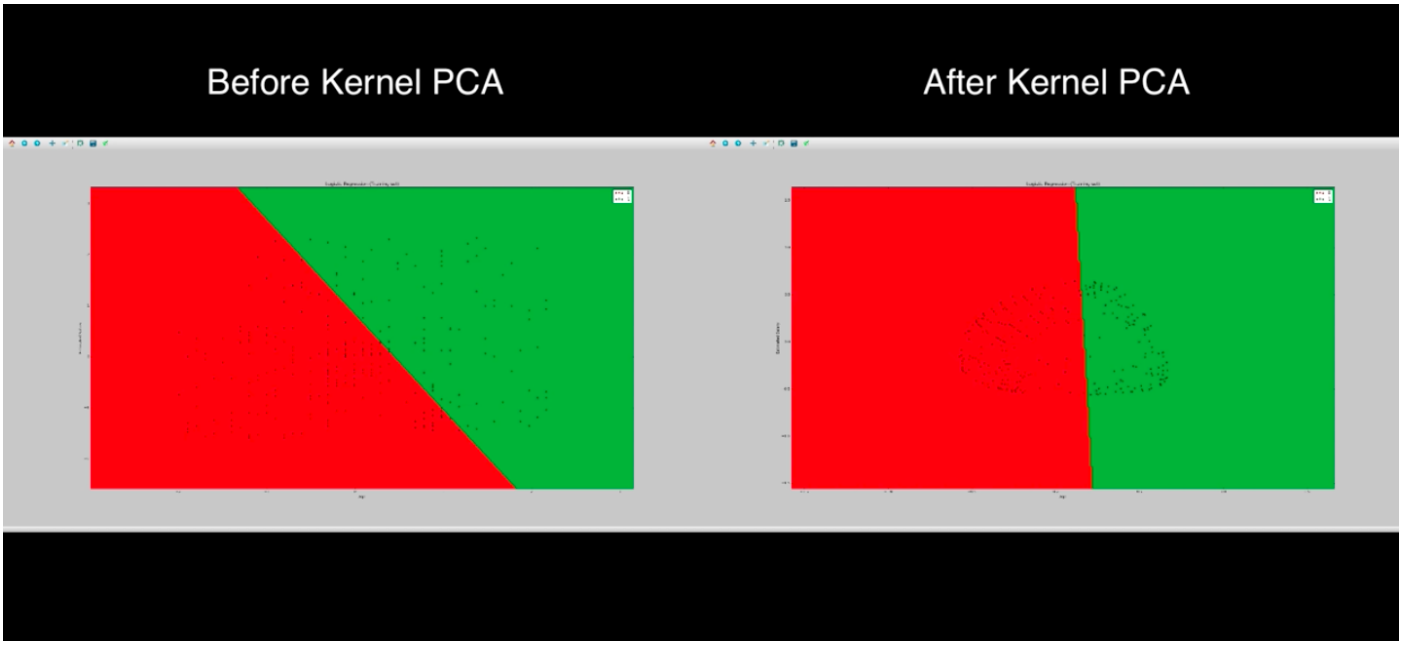
```
### Kernel PCA

# Data same as part 3.


# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
path = 'Machine Learning A-Z Template Folder/Part 9 - Dimensionality Reducti
on/Section 45 - Kernel PCA/'
dataset = pd.read_csv(path + 'Social_Network_Ads.csv')
# only using age and estimated salary:
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split  #*
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)



##
###*  Applying Kernel PCA:
##

from sklearn.decomposition import KernelPCA
kpca = KernelPCA(n_components=2, kernel='rbf')   #*
X_train = kpca.fit_transform(X_train)
X_test = kpca.transform(X_test)



#######

# fitting Logistic Regression to the training set:
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state=0)
classifier.fit(X_train, y_train)

# predicting the test set results:
```

```
y_pred = classifier.predict(X_test)
print(y_pred)

# making the Confusion Matrix:
from sklearn.metrics import confusion_matrix   #* here importing a Function
 and not Class (no capitals in name)
cm = confusion_matrix(y_true=y_test, y_pred=y_pred)
print(cm)

##* visualizing the Training set results:
from matplotlib.colors import ListedColormap  # to colorize all the datapoin
ts
X_set, y_set = X_train, y_train
# prepare grid with all the pixel points:
 # 'ravel': Return a contiguous flattened array.
 # 'meshgrid': Return coordinate matrices from coordinate vectors.
X1, X2 = np.meshgrid(np.arange(start = X_set[:,0].min() - 1, stop = X_set[:,
0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:,1].min() - 1, stop = X_set[:,
1].max() + 1, step = 0.01))
# apply classifier on all the pixel observation points and make countour b/w
the two regions:
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.50, cmap = ListedColormap(('red', 'green')))    #* alp
ha is for Opaqueness.
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
# plot all the datapoints:
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],         ## * ???
                c = ListedColormap(('red', 'green'))(i), label = j)
    # c : color, sequence, or sequence of color,
plt.title('Logistic Regression (Training Set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()


##* visualizing the Test set results:
from matplotlib.colors import ListedColormap  # to colorize all the datapoin
ts
X_set, y_set = X_test, y_test
# prepare grid with all the pixel points:
 # 'ravel': Return a contiguous flattened array.
X1, X2 = np.meshgrid(np.arange(start = X_set[:,0].min() - 1, stop = X_set[:,
0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:,1].min() - 1, stop = X_set[:,
1].max() + 1, step = 0.01))
# apply classifier on all the pixel observation points and make countour b/w
the two regions:
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
```

```
).reshape(X1.shape),
             alpha = 0.60, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

# plot all the datapoints:
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Test Set)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.show()
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logi
stic.py:433: FutureWarning: Default solver will be changed to 'l
bfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.

[0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0
1 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0
0 1 0 0 0
 0 0 0 0 1 1 1 1 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 1 1 1]
[[64  4]
 [ 6 26]]
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
```



In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
############################################################
# Section 37 (47): Part 10: Model Selection & Boosting
############################################################
```

In [ ]:

```
```

In [ ]:

```
```

In [52]:

```
## Qns to be answered:
 # - How to deal with the bias variance tradeoff when building a model and e
valuating its performance ?
 # - How to choose the optimal values for the hyperparameters (the parameter
s that are not learned) ?
 # - How to find the most appropriate Machine Learning model for my business
problem ?




## In this part answer these questions using Model Selection techniques incl
uding:

 # - k-Fold Cross Validation
 # - Grid Search



## * XGBoost
```

In [ ]:

```

```

In [ ]:

```
#==================================
# Section 38 (48): Model Selection
#==================================
```

In [ ]:

```

```

In [55]:

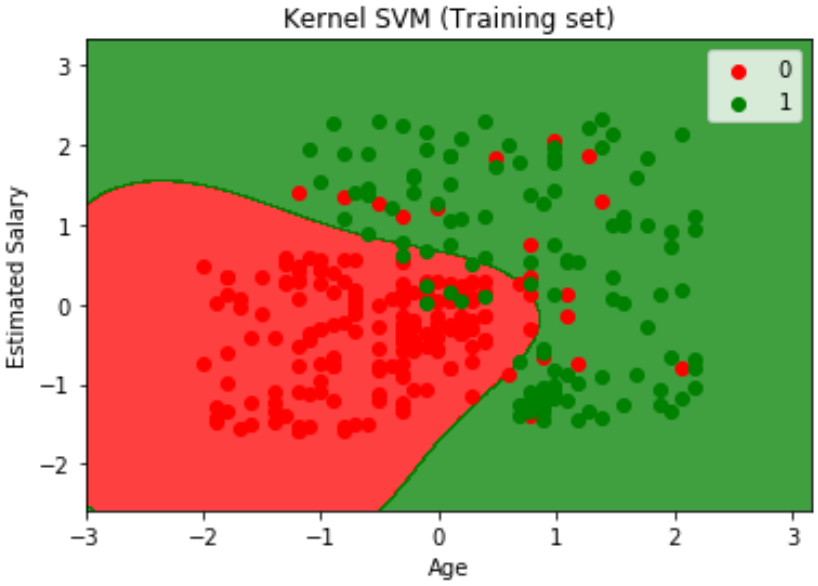
```
##* K-Fold Cross Validation:
#---------------------------



## * K-Fold Cross Validation is applied on the 'training set'.




## Kernel SVM

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
path = 'Machine Learning A-Z Template Folder/Part 10 - Model Selection & Boo
sting/Section 48 - Model Selection/'
dataset = pd.read_csv(path + 'Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)


###

# Fitting "Kernel SVM" to the Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', random_state = 0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

```
## Applying K-Fold Cross Validation:
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator=classifier, X=X_train, y=y_train, cv=
10)
print(accuracies)
print(accuracies.mean())
print(accuracies.std())



# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set
[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set
[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Kernel SVM (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set
[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set
[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Kernel SVM (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
```

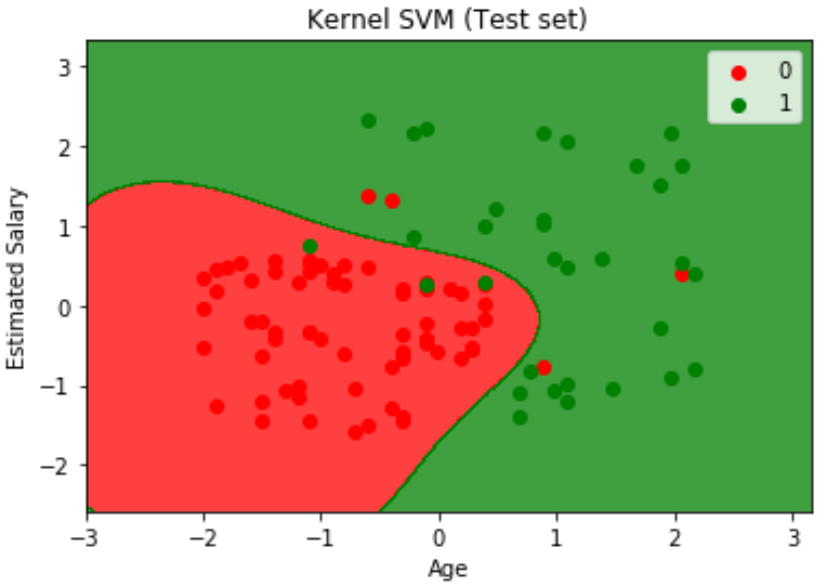
```
plt.legend()
plt.show()
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:196:
FutureWarning: The default value of gamma will change from 'aut
o' to 'scale' in version 0.22 to account better for unscaled fea
tures. Set gamma explicitly to 'auto' or 'scale' to avoid this w
arning.
  "avoid this warning.", FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:196:
FutureWarning: The default value of gamma will change from 'aut
o' to 'scale' in version 0.22 to account better for unscaled fea
tures. Set gamma explicitly to 'auto' or 'scale' to avoid this w
arning.
  "avoid this warning.", FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:196:
FutureWarning: The default value of gamma will change from 'aut
o' to 'scale' in version 0.22 to account better for unscaled fea
tures. Set gamma explicitly to 'auto' or 'scale' to avoid this w
arning.
  "avoid this warning.", FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:196:
FutureWarning: The default value of gamma will change from 'aut
o' to 'scale' in version 0.22 to account better for unscaled fea
tures. Set gamma explicitly to 'auto' or 'scale' to avoid this w
arning.
  "avoid this warning.", FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:196:
FutureWarning: The default value of gamma will change from 'aut
o' to 'scale' in version 0.22 to account better for unscaled fea
tures. Set gamma explicitly to 'auto' or 'scale' to avoid this w
arning.
  "avoid this warning.", FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:196:
FutureWarning: The default value of gamma will change from 'aut
o' to 'scale' in version 0.22 to account better for unscaled fea
tures. Set gamma explicitly to 'auto' or 'scale' to avoid this w
arning.
  "avoid this warning.", FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:196:
FutureWarning: The default value of gamma will change from 'aut
o' to 'scale' in version 0.22 to account better for unscaled fea
tures. Set gamma explicitly to 'auto' or 'scale' to avoid this w
arning.
  "avoid this warning.", FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:196:
FutureWarning: The default value of gamma will change from 'aut
o' to 'scale' in version 0.22 to account better for unscaled fea
tures. Set gamma explicitly to 'auto' or 'scale' to avoid this w
arning.
  "avoid this warning.", FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:196:
FutureWarning: The default value of gamma will change from 'aut
o' to 'scale' in version 0.22 to account better for unscaled fea
tures. Set gamma explicitly to 'auto' or 'scale' to avoid this w
arning.
```

```
  "avoid this warning.", FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:196:
FutureWarning: The default value of gamma will change from 'aut
o' to 'scale' in version 0.22 to account better for unscaled fea
tures. Set gamma explicitly to 'auto' or 'scale' to avoid this w
arning.
  "avoid this warning.", FutureWarning)


[0.80645161 0.96666667 0.8        0.93333333 0.86666667 0.833333
33
 0.93333333 0.93333333 0.96666667 0.96551724]
0.9005302187615868
0.06388957356626285


'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
```




```
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
```



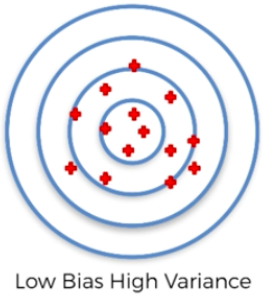
In [ ]:

The Bias-Variance Tradeoff


High Bias Low Variance


High Bias High Variance


Low Bias Low Variance


Low Bias High Variance

In [ ]:

In [57]:

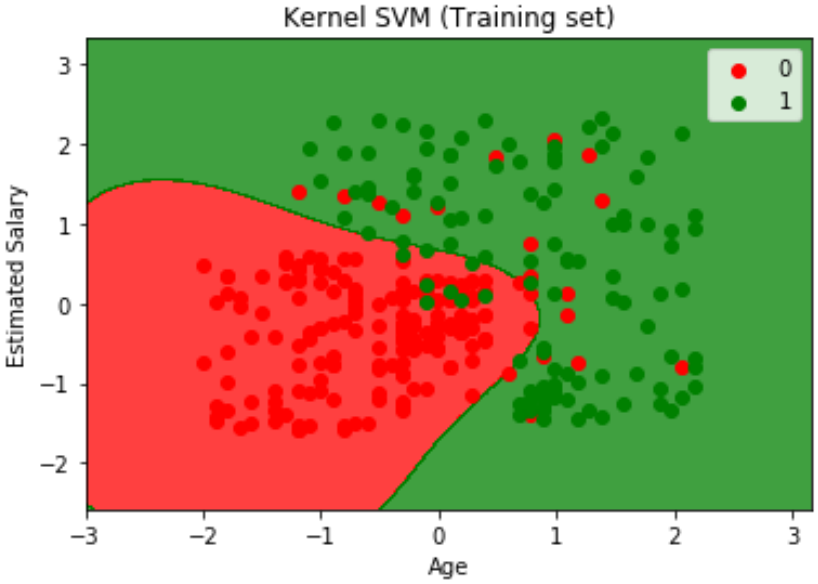
```
##* Grid Search:  finding optimal values for hyper-parameters
#---------------



# This answers: How to choose the optimal values for the hyperparameters (th
e parameters that are not learned) ?




# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
path = 'Machine Learning A-Z Template Folder/Part 10 - Model Selection & Boo
sting/Section 48 - Model Selection/'
dataset = pd.read_csv(path + 'Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)


###

# Fitting "Kernel SVM" to the Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', random_state = 0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

```
## Applying K-Fold Cross Validation:
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator=classifier, X=X_train, y=y_train, cv=
10)
print(accuracies)
print(accuracies.mean())
print(accuracies.std())



## Applying 'Grid Search' to find the best model and the best parameters:
from sklearn.model_selection import GridSearchCV
# a list of dictionaries (for parameters to be optimized):    #**
parameters = [{'C': [1, 10, 100, 1000], 'kernel': ['linear']},
              {'C': [1, 10, 100, 1000], 'kernel': ['rbf'], 'gamma': [0.1, 0.2
, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]}]# [0.5, 0.1, 0.01, 0.001, 0.0001]}]

grid_search = GridSearchCV(estimator=classifier,
                           param_grid=parameters,
                           scoring='accuracy',        # many options to pick
from
                           cv=10,         # 10 fold cross-validation
                           n_jobs=-1)      # 'n_jobs': in case of large data
set

grid_search = grid_search.fit(X_train, y_train)
best_accuracy = grid_search.best_score_   # mean of 10 accuracies of k fold.
print("best accuracy: ", best_accuracy)
best_parameters = grid_search.best_params_
print("best params.: ", '\t', best_parameters)




# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set
[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set
[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Kernel SVM (Training set)')
```

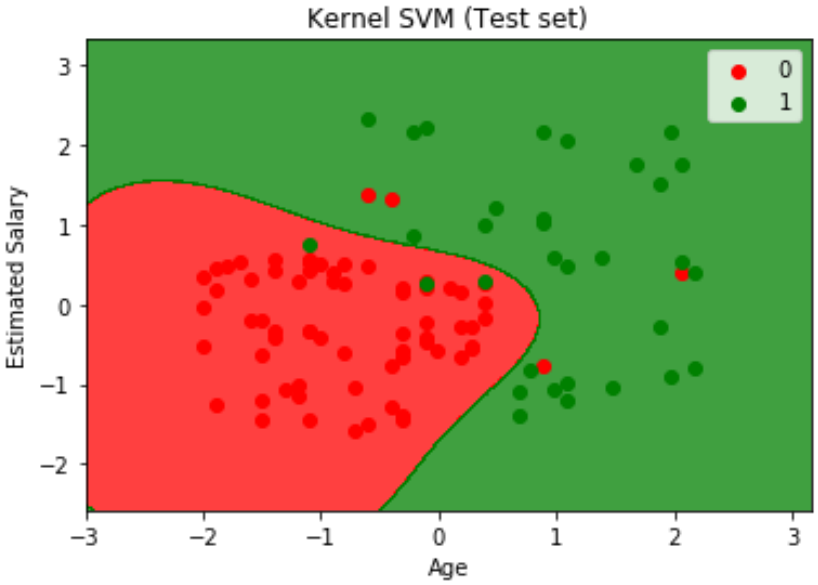
```
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set
[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set
[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Kernel SVM (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:196:
FutureWarning: The default value of gamma will change from 'aut
o' to 'scale' in version 0.22 to account better for unscaled fea
tures. Set gamma explicitly to 'auto' or 'scale' to avoid this w
arning.
  "avoid this warning.", FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:196:
FutureWarning: The default value of gamma will change from 'aut
o' to 'scale' in version 0.22 to account better for unscaled fea
tures. Set gamma explicitly to 'auto' or 'scale' to avoid this w
arning.
  "avoid this warning.", FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:196:
FutureWarning: The default value of gamma will change from 'aut
o' to 'scale' in version 0.22 to account better for unscaled fea
tures. Set gamma explicitly to 'auto' or 'scale' to avoid this w
arning.
  "avoid this warning.", FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:196:
FutureWarning: The default value of gamma will change from 'aut
o' to 'scale' in version 0.22 to account better for unscaled fea
tures. Set gamma explicitly to 'auto' or 'scale' to avoid this w
arning.
  "avoid this warning.", FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:196:
FutureWarning: The default value of gamma will change from 'aut
o' to 'scale' in version 0.22 to account better for unscaled fea
tures. Set gamma explicitly to 'auto' or 'scale' to avoid this w
arning.
  "avoid this warning.", FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:196:
FutureWarning: The default value of gamma will change from 'aut
o' to 'scale' in version 0.22 to account better for unscaled fea
tures. Set gamma explicitly to 'auto' or 'scale' to avoid this w
arning.
  "avoid this warning.", FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:196:
FutureWarning: The default value of gamma will change from 'aut
o' to 'scale' in version 0.22 to account better for unscaled fea
tures. Set gamma explicitly to 'auto' or 'scale' to avoid this w
arning.
  "avoid this warning.", FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:196:
FutureWarning: The default value of gamma will change from 'aut
o' to 'scale' in version 0.22 to account better for unscaled fea
tures. Set gamma explicitly to 'auto' or 'scale' to avoid this w
arning.
  "avoid this warning.", FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:196:
FutureWarning: The default value of gamma will change from 'aut
o' to 'scale' in version 0.22 to account better for unscaled fea
tures. Set gamma explicitly to 'auto' or 'scale' to avoid this w
arning.
```

```
  "avoid this warning.", FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:196:
FutureWarning: The default value of gamma will change from 'aut
o' to 'scale' in version 0.22 to account better for unscaled fea
tures. Set gamma explicitly to 'auto' or 'scale' to avoid this w
arning.
  "avoid this warning.", FutureWarning)


[0.80645161 0.96666667 0.8        0.93333333 0.86666667 0.833333
33
 0.93333333 0.93333333 0.96666667 0.96551724]
0.9005302187615868
0.06388957356626285

/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_
search.py:841: DeprecationWarning: The default of the `iid` para
meter will change from True to False in version 0.22 and will be
removed in 0.24. This will change numeric results when test-set
sizes are unequal.
  DeprecationWarning)

best accuracy:  0.9033333333333333
best params.:    {'C': 1, 'gamma': 0.7, 'kernel': 'rbf'}

'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
```



```
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, w
hich should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array
with a single row if you really want to specify the same RGB or
RGBA value for all points.
```



In [ ]:

```
```

In [ ]:

```
#=========================
# Section 39 (49): XGBoost
#=========================
```

In [ ]:

In [1]:

```
## XGBoost:  eXtreme Gradient Boosting.


## ** XGBoost is a gradient boosting model with decision trees




## XGBoost is a software library, can be accessed via a variety of interface
s.
## Specifically, XGBoost supports the following main interfaces:

 # - Command Line Interface (CLI).
 # - C++ (the language in which the library is written).
 # - Python interface as well as a model in scikit-learn.
 # - R interface as well as a model in the caret package.
 # - Julia.
 # - Java and JVM languages like Scala and platforms like Hadoop.




## Three main forms of gradient boosting are supported:

 # - 'Gradient Boosting' algorithm also called gradient boosting machine inc
luding the learning rate.
 # - 'Stochastic Gradient Boosting' with sub-sampling at the row, column and
column per split levels.
 # - 'Regularized Gradient Boosting' with both L1 and L2 regularization.



# Generally, XGBoost is fast. Really fast when compared to other implementat
ions of gradient boosting.

# XGBoost dominates structured or tabular datasets on classification and reg
ression predictive modeling problems.


# This algorithm goes by lots of different names such as:
 # gradient boosting, multiple additive regression trees, stochastic gradien
t boosting or gradient boosting machines.



# Boosting is an ensemble technique where new models are added to correct th
e errors made by existing models.
```

Out[1]:

7

In [ ]:

In [2]:

```
### XGBoost:

##** XGBoost is a gradient boosting model with decision trees.

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
path = 'Machine Learning A-Z Template Folder/Part 10 - Model Selection & Boo
sting/Section 49 - XGBoost/'
dataset = pd.read_csv(path + 'Churn_Modelling.csv')
X = dataset.iloc[:, 3:13].values
y = dataset.iloc[:, 13].values

# Encoding categorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X_1 = LabelEncoder()
X[:, 1] = labelencoder_X_1.fit_transform(X[:, 1])
labelencoder_X_2 = LabelEncoder()
X[:, 2] = labelencoder_X_2.fit_transform(X[:, 2])
onehotencoder = OneHotEncoder(categorical_features = [1])
X = onehotencoder.fit_transform(X).toarray()
X = X[:, 1:]

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, r
andom_state = 0)


## ** No feature scaling needed.


# Fitting "XGBoost" to the training set:
from xgboost import XGBClassifier
classifier = XGBClassifier()
classifier.fit(X_train, y_train)
print(classifier)


# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(y_pred)


# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

```
print(cm)


# Applying k-Fold Cross Validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_trai
n, cv = 10)
print(accuracies.mean())
print(accuracies.std())
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/_en
coders.py:371: FutureWarning: The handling of integer data will
change in version 0.22. Currently, the categories are determined
based on the range [0, max(values)], while in the future they wi
ll be determined based on the unique values.
If you want the future behaviour and silence this warning, you c
an specify "categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to con
vert the categories to integers, then you can now use the OneHot
Encoder directly.
  warnings.warn(msg, FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/_en
coders.py:392: DeprecationWarning: The 'categorical_features' ke
yword is deprecated in version 0.20 and will be removed in 0.22.
You can use the ColumnTransformer instead.
  "use the ColumnTransformer instead.", DeprecationWarning)

XGBClassifier(base_score=0.5, booster='gbtree', colsample_byleve
l=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0, learning
_rate=0.1,
       max_delta_step=0, max_depth=3, min_child_weight=1, missin
g=None,
       n_estimators=100, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=None,
       subsample=1, verbosity=1)
[0 0 0 ... 0 0 0]
[[1521   74]
 [ 197  208]]
0.8629994451163204
0.010677872171663988
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
# 5 NLP Libraries:

    # The Conqueror: NLTK
    # The Prince: TextBlob
    # The Mercenary: Stanford CoreNLP
    # The Usurper: spaCy
    # The Admiral: gensim


# REF: https://elitedatascience.com/python-nlp-libraries
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: