Data Preprocessing

In [1]:

```
## Importing the libraries
```

In [2]:

```
import numpy as np # it contain mathematical tools
import matplotlib.pyplot as plt # for plotting the graphs
import pandas as pd # to import dataset and manage dataset
```

In [3]:

```
# Importing the dataset
```

In [4]:

```
dataset = pd.read_csv('Data.csv')
```

In [5]:

```
X = dataset.iloc[:, :-1].values  # constructing vector or matrix of independet variable
X
```

Out[5]:

```
array([['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]], dtype=object)
```

In [6]:

```
y = dataset.iloc[:, 4].values  # constructing vector or matrix of dependet variable
```

Out[6]:

```
array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes'], d
type=object)
```

In [7]:

```
# Taking care of missing data
#(taking mean of column and pt it missed place)
```

In [8]:

```
from sklearn.preprocessing import Imputer # Imputer is to take care of missing data
imputer = Imputer(missing_values = 'NaN',strategy = 'mean',axis = 0) # NaN is placed on
 missing place,strategy can mean,median, most_frequent..,axis=0 => columnwise mean,axis
=1 => rowwise
imputer = imputer.fit(X[:, 1:3])# to fit imputer object to marix X
X[:, 1:3] = imputer.transform(X[:, 1:3]) # to fit mean in missing data by using transfo
rm method
X
```

Out[8]:

```
array([['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]], dtype=object)
```

In [9]:

```
# encoding the categorical data into (encode the text into numbers)

from sklearn.preprocessing import LabelEncoder
labelencoder_X = LabelEncoder()    # making object of LabelEncoder
labelencoder_X.fit_transform(X[:, 0]) # transforming into label
```

Out[9]:

```
array([0, 2, 1, 2, 1, 0, 2, 0, 1, 0], dtype=int64)
```

In [10]:

```
X[:, 0]=labelencoder_X.fit_transform(X[:, 0])
X
```

Out[10]:

```
array([[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]], dtype=object)
```

In [11]:

```
from sklearn.preprocessing import OneHotEncoder # dummy encoding..convert categorical f
eature int k scheme
onehotencoder = OneHotEncoder(categorical_features = [0]) # which colm to convert here
 "0"
X = onehotencoder.fit_transform(X).toarray()
X
```

Out[11]:

```
array([[  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]])
```

In [12]:

```
labelencoder_y = LabelEncoder()    # making object for y of LabelEncoder
y=labelencoder_y.fit_transform(y) # transforming into label not onehotencoder here we h
ave only two categorical featres
y
```

Out[12]:

```
array([0, 1, 0, 0, 1, 1, 0, 1, 0, 1], dtype=int64)
```

In [13]:

```
# Splitting the dataset into training and test sets
```

In [14]:

```
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) # create four variable
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41:
DeprecationWarning: This module was deprecated in version 0.18 in favor of
the model_selection module into which all the refactored classes and funct
ions are moved. Also note that the interface of the new CV iterators are d
ifferent from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [15]:

```
X_train
```

Out[15]:

```
array([[  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]])
```

In [16]:

```
X_test
```

Out[16]:

```
array([[  0.00000000e+00,   1.00000000e+00,   0.00000000e+00,
          3.00000000e+01,   5.40000000e+04],
       [  0.00000000e+00,   1.00000000e+00,   0.00000000e+00,
          5.00000000e+01,   8.30000000e+04]])
```

In [17]:

```
# Feature Scaling (Standardisation,Normalisation)
```

In [18]:

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

In [19]:

```
X_train
```

Out[19]:

```
array([[-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]])
```

In [20]:

```
X_test
```

Out[20]:

```
array([[-1.        ,  2.64575131, -0.77459667, -1.45882927, -0.90166297],
       [-1.        ,  2.64575131, -0.77459667,  1.98496442,  2.13981082]])
```

In [21]:

```
# to see current working directory
```

In [22]:

```
import os
os.getcwd()
```

Out[22]:

```
'C:\\Users\\Admin\\focus\\Assignment 8'
```

Simple Linear Regression on salary_data

In [23]:

```
dataset = pd.read_csv('Salary_Data.csv')
```

In [24]:

```
X = dataset.iloc[:, :-1].values  # constructing vector or matrix of independet variable
X
```

Out[24]:

```
array([[  1.1],
       [  1.3],
       [  1.5],
       [  2. ],
       [  2.2],
       [  2.9],
       [  3. ],
       [  3.2],
       [  3.2],
       [  3.7],
       [  3.9],
       [  4. ],
       [  4. ],
       [  4.1],
       [  4.5],
       [  4.9],
       [  5.1],
       [  5.3],
       [  5.9],
       [  6. ],
       [  6.8],
       [  7.1],
       [  7.9],
       [  8.2],
       [  8.7],
       [  9. ],
       [  9.5],
       [  9.6],
       [ 10.3],
       [ 10.5]])
```

In [25]:

```
y = dataset.iloc[:, 1].values # create vector of dependent variable
y
```

Out[25]:

```
array([  39343.,   46205.,   37731.,   43525.,   39891.,   56642.,
         60150.,   54445.,   64445.,   57189.,   63218.,   55794.,
         56957.,   57081.,   61111.,   67938.,   66029.,   83088.,
         81363.,   93940.,   91738.,   98273.,  101302.,  113812.,
        109431.,  105582.,  116969.,  112635.,  122391.,  121872.])
```

In [26]:

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state
 = 0) # create four variable
```

In [27]:

```
X_train
```

Out[27]:

```
array([[  2.9],
       [  5.1],
       [  3.2],
       [  4.5],
       [  8.2],
       [  6.8],
       [  1.3],
       [ 10.5],
       [  3. ],
       [  2.2],
       [  5.9],
       [  6. ],
       [  3.7],
       [  3.2],
       [  9. ],
       [  2. ],
       [  1.1],
       [  7.1],
       [  4.9],
       [  4. ]])
```

In [28]:

```
# fitting the lr reg on training set
```

In [29]:

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression() # created a model
regressor.fit(X_train, y_train) # here model learnt from the training data
```

Out[29]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fals
e)
```

In [30]:

```
# predicting the test set result
```

In [31]:

```
y_pred = regressor.predict(X_test)
```

In [32]:

```
y_pred
```

Out[32]:

```
array([  40835.10590871,  123079.39940819,   65134.55626083,
         63265.36777221,  115602.64545369,  108125.8914992 ,
        116537.23969801,   64199.96201652,   76349.68719258,
        100649.1375447 ])
```

In [33]:

```
y_test
```

Out[33]:

```
array([  37731.,  122391.,   57081.,   63218.,  116969.,  109431.,
        112635.,   55794.,   83088.,  101302.])
```

In [34]:

```
# Visualisation of training set result
```

In [35]:

```
plt.scatter(X_train, y_train, color = 'red') # plotting points
plt.plot(X_train, regressor.predict(X_train), color = 'blue') # plotting line of predic
tion x_tarin
plt.title('salary vs year of experience(on training set)')
plt.xlabel('Years of experience')
plt.ylabel('Salary')
plt.show()
```

In [36]:

```
plt.scatter(X_test, y_test, color = 'red') # plotting points
plt.plot(X_train, regressor.predict(X_train), color = 'blue') # plotting line of predic
tion
plt.title('salary vs year of experience(on test set)')
plt.xlabel('Years of experience')
plt.ylabel('Salary')
plt.show()
```

Multiple Linear Regression on 50_startups data

In [37]:

```
dataset = pd.read_csv('50_Startups.csv')
X = dataset.iloc[:, :-1].values  # constructing vector or matrix of independet variable
y = dataset.iloc[:, 4].values # create vector of dependent variable

# encoding the categorical data into (encode the text into numbers)

from sklearn.preprocessing import LabelEncoder
labelencoder_X = LabelEncoder()    # making object of LabelEncoder
 # transforming into label
X[:, 3]=labelencoder_X.fit_transform(X[:, 3])
from sklearn.preprocessing import OneHotEncoder # dummy encoding..convert categorical f
eature int k scheme
onehotencoder = OneHotEncoder(categorical_features = [3]) # which colm to convert here
 "0"
X = onehotencoder.fit_transform(X).toarray()
#avoid the dummy variable trap
X = X[:, 1:]


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) # create four variable
```

In [38]:

```
# fitting the multiple lr reg on training set
```

In [39]:

```
regressor_m = LinearRegression()
regressor_m.fit(X_train, y_train)
```

Out[39]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fals
e)
```

In [40]:

```
# predicting the test set results
```

In [41]:

```
y_pred = regressor_m.predict(X_test)
```

In [42]:

```
y_pred
```

Out[42]:

```
array([ 103015.20159796,  132582.27760815,  132447.73845175,
         71976.09851258,  178537.48221056,  116161.24230166,
         67851.69209676,   98791.73374687,  113969.43533013,
        167921.06569551])
```

In [43]:

```
y_test
```

Out[43]:

```
array([ 103282.38,  144259.4 ,  146121.95,   77798.83,  191050.39,
        105008.31,   81229.06,   97483.56,  110352.25,  166187.94])
```

In [44]:

```
#building the model using backward elimination
```

In [45]:

```
import statsmodels.formula.api as sm
X = np.append(arr = np.ones((50, 1)).astype(int), values = X, axis = 1) # adding b cons
tant to eqn ,here adding col in front of X
```

In [46]:

```
X
```

Out[46]:

```
array([[  1.00000000e+00,   0.00000000e+00,   1.00000000e+00,
          1.65349200e+05,   1.36897800e+05,   4.71784100e+05],
       [  1.00000000e+00,   0.00000000e+00,   0.00000000e+00,
          1.62597700e+05,   1.51377590e+05,   4.43898530e+05],
       [  1.00000000e+00,   1.00000000e+00,   0.00000000e+00,
          1.53441510e+05,   1.01145550e+05,   4.07934540e+05],
       [  1.00000000e+00,   0.00000000e+00,   1.00000000e+00,
          1.44372410e+05,   1.18671850e+05,   3.83199620e+05],
       [  1.00000000e+00,   1.00000000e+00,   0.00000000e+00,
          1.42107340e+05,   9.13917700e+04,   3.66168420e+05],
       [  1.00000000e+00,   0.00000000e+00,   1.00000000e+00,
          1.31876900e+05,   9.98147100e+04,   3.62861360e+05],
       [  1.00000000e+00,   0.00000000e+00,   0.00000000e+00,
          1.34615460e+05,   1.47198870e+05,   1.27716820e+05],
       [  1.00000000e+00,   1.00000000e+00,   0.00000000e+00,
          1.30298130e+05,   1.45530060e+05,   3.23876680e+05],
       [  1.00000000e+00,   0.00000000e+00,   1.00000000e+00,
          1.20542520e+05,   1.48718950e+05,   3.11613290e+05],
       [  1.00000000e+00,   0.00000000e+00,   0.00000000e+00,
          1.23334880e+05,   1.08679170e+05,   3.04981620e+05],
       [  1.00000000e+00,   1.00000000e+00,   0.00000000e+00,
          1.01913080e+05,   1.10594110e+05,   2.29160950e+05],
       [  1.00000000e+00,   0.00000000e+00,   0.00000000e+00,
          1.00671960e+05,   9.17906100e+04,   2.49744550e+05],
       [  1.00000000e+00,   1.00000000e+00,   0.00000000e+00,
          9.38637500e+04,   1.27320380e+05,   2.49839440e+05],
       [  1.00000000e+00,   0.00000000e+00,   0.00000000e+00,
          9.19923900e+04,   1.35495070e+05,   2.52664930e+05],
       [  1.00000000e+00,   1.00000000e+00,   0.00000000e+00,
          1.19943240e+05,   1.56547420e+05,   2.56512920e+05],
       [  1.00000000e+00,   0.00000000e+00,   1.00000000e+00,
          1.14523610e+05,   1.22616840e+05,   2.61776230e+05],
       [  1.00000000e+00,   0.00000000e+00,   0.00000000e+00,
          7.80131100e+04,   1.21597550e+05,   2.64346060e+05],
       [  1.00000000e+00,   0.00000000e+00,   1.00000000e+00,
          9.46571600e+04,   1.45077580e+05,   2.82574310e+05],
       [  1.00000000e+00,   1.00000000e+00,   0.00000000e+00,
          9.17491600e+04,   1.14175790e+05,   2.94919570e+05],
       [  1.00000000e+00,   0.00000000e+00,   1.00000000e+00,
          8.64197000e+04,   1.53514110e+05,   0.00000000e+00],
       [  1.00000000e+00,   0.00000000e+00,   0.00000000e+00,
          7.62538600e+04,   1.13867300e+05,   2.98664470e+05],
       [  1.00000000e+00,   0.00000000e+00,   1.00000000e+00,
          7.83894700e+04,   1.53773430e+05,   2.99737290e+05],
       [  1.00000000e+00,   1.00000000e+00,   0.00000000e+00,
          7.39945600e+04,   1.22782750e+05,   3.03319260e+05],
       [  1.00000000e+00,   1.00000000e+00,   0.00000000e+00,
          6.75325300e+04,   1.05751030e+05,   3.04768730e+05],
       [  1.00000000e+00,   0.00000000e+00,   1.00000000e+00,
          7.70440100e+04,   9.92813400e+04,   1.40574810e+05],
       [  1.00000000e+00,   0.00000000e+00,   0.00000000e+00,
          6.46647100e+04,   1.39553160e+05,   1.37962620e+05],
       [  1.00000000e+00,   1.00000000e+00,   0.00000000e+00,
          7.53288700e+04,   1.44135980e+05,   1.34050070e+05],
       [  1.00000000e+00,   0.00000000e+00,   1.00000000e+00,
          7.21076000e+04,   1.27864550e+05,   3.53183810e+05],
       [  1.00000000e+00,   1.00000000e+00,   0.00000000e+00,
          6.60515200e+04,   1.82645560e+05,   1.18148200e+05],
       [  1.00000000e+00,   0.00000000e+00,   1.00000000e+00,
```

```
   6.56054800e+04,   1.53032060e+05,   1.07138380e+05],
[  1.00000000e+00,   1.00000000e+00,   0.00000000e+00,
   6.19944800e+04,   1.15641280e+05,   9.11312400e+04],
[  1.00000000e+00,   0.00000000e+00,   1.00000000e+00,
   6.11363800e+04,   1.52701920e+05,   8.82182300e+04],
[  1.00000000e+00,   0.00000000e+00,   0.00000000e+00,
   6.34088600e+04,   1.29219610e+05,   4.60852500e+04],
[  1.00000000e+00,   1.00000000e+00,   0.00000000e+00,
   5.54939500e+04,   1.03057490e+05,   2.14634810e+05],
[  1.00000000e+00,   0.00000000e+00,   0.00000000e+00,
   4.64260700e+04,   1.57693920e+05,   2.10797670e+05],
[  1.00000000e+00,   0.00000000e+00,   1.00000000e+00,
   4.60140200e+04,   8.50474400e+04,   2.05517640e+05],
[  1.00000000e+00,   1.00000000e+00,   0.00000000e+00,
   2.86637600e+04,   1.27056210e+05,   2.01126820e+05],
[  1.00000000e+00,   0.00000000e+00,   0.00000000e+00,
   4.40699500e+04,   5.12831400e+04,   1.97029420e+05],
[  1.00000000e+00,   0.00000000e+00,   1.00000000e+00,
   2.02295900e+04,   6.59479300e+04,   1.85265100e+05],
[  1.00000000e+00,   0.00000000e+00,   0.00000000e+00,
   3.85585100e+04,   8.29820900e+04,   1.74999300e+05],
[  1.00000000e+00,   0.00000000e+00,   0.00000000e+00,
   2.87543300e+04,   1.18546050e+05,   1.72795670e+05],
[  1.00000000e+00,   1.00000000e+00,   0.00000000e+00,
   2.78929200e+04,   8.47107700e+04,   1.64470710e+05],
[  1.00000000e+00,   0.00000000e+00,   0.00000000e+00,
   2.36409300e+04,   9.61896300e+04,   1.48001110e+05],
[  1.00000000e+00,   0.00000000e+00,   1.00000000e+00,
   1.55057300e+04,   1.27382300e+05,   3.55341700e+04],
[  1.00000000e+00,   0.00000000e+00,   0.00000000e+00,
   2.21777400e+04,   1.54806140e+05,   2.83347200e+04],
[  1.00000000e+00,   0.00000000e+00,   1.00000000e+00,
   1.00023000e+03,   1.24153040e+05,   1.90393000e+03],
[  1.00000000e+00,   1.00000000e+00,   0.00000000e+00,
   1.31546000e+03,   1.15816210e+05,   2.97114460e+05],
[  1.00000000e+00,   0.00000000e+00,   0.00000000e+00,
   0.00000000e+00,   1.35426920e+05,   0.00000000e+00],
[  1.00000000e+00,   0.00000000e+00,   1.00000000e+00,
   5.42050000e+02,   5.17431500e+04,   0.00000000e+00],
[  1.00000000e+00,   0.00000000e+00,   0.00000000e+00,
   0.00000000e+00,   1.16983800e+05,   4.51730600e+04]])
```

In [47]:

```
X_opt = X[:, [0,1,2,3,4,5]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit() # to check dependecy of col onto
 dependent variable y
```

In [48]:

```
regressor_OLS.summary()
```

Out[48]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.951
Model:	OLS	**Adj. R-squared:**	0.945
Method:	Least Squares	**F-statistic:**	169.9
Date:	Tue, 12 Sep 2017	**Prob (F-statistic):**	1.34e-27
Time:	19:21:16	**Log-Likelihood:**	-525.38
No. Observations:	50	**AIC:**	1063.
Df Residuals:	44	**BIC:**	1074.
Df Model:	5		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	5.013e+04	6884.820	7.281	0.000	3.62e+04	6.4e+04
x1	198.7888	3371.007	0.059	0.953	-6595.030	6992.607
x2	-41.8870	3256.039	-0.013	0.990	-6604.003	6520.229
x3	0.8060	0.046	17.369	0.000	0.712	0.900
x4	-0.0270	0.052	-0.517	0.608	-0.132	0.078
x5	0.0270	0.017	1.574	0.123	-0.008	0.062

Omnibus:	14.782	**Durbin-Watson:**	1.283
Prob(Omnibus):	0.001	**Jarque-Bera (JB):**	21.266
Skew:	-0.948	**Prob(JB):**	2.41e-05
Kurtosis:	5.572	**Cond. No.**	1.45e+06

In [49]:

```
# x2 has higer p value so we need to remove this
```

In [50]:

```
X_opt = X[:, [0,1,3,4,5]]
```

In [51]:

```
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
```

In [52]:

```
regressor_OLS.summary()
```

Out[52]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.951
Model:	OLS	**Adj. R-squared:**	0.946
Method:	Least Squares	**F-statistic:**	217.2
Date:	Tue, 12 Sep 2017	**Prob (F-statistic):**	8.49e-29
Time:	19:21:16	**Log-Likelihood:**	-525.38
No. Observations:	50	**AIC:**	1061.
Df Residuals:	45	**BIC:**	1070.
Df Model:	4		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	5.011e+04	6647.870	7.537	0.000	3.67e+04	6.35e+04
x1	220.1585	2900.536	0.076	0.940	-5621.821	6062.138
x2	0.8060	0.046	17.606	0.000	0.714	0.898
x3	-0.0270	0.052	-0.523	0.604	-0.131	0.077
x4	0.0270	0.017	1.592	0.118	-0.007	0.061

Omnibus:	14.758	**Durbin-Watson:**	1.282
Prob(Omnibus):	0.001	**Jarque-Bera (JB):**	21.172
Skew:	-0.948	**Prob(JB):**	2.53e-05
Kurtosis:	5.563	**Cond. No.**	1.40e+06

In [53]:

```
X_opt = X[:, [0,3,4,5]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()
```

Out[53]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.951
Model:	OLS	**Adj. R-squared:**	0.948
Method:	Least Squares	**F-statistic:**	296.0
Date:	Tue, 12 Sep 2017	**Prob (F-statistic):**	4.53e-30
Time:	19:21:17	**Log-Likelihood:**	-525.39
No. Observations:	50	**AIC:**	1059.
Df Residuals:	46	**BIC:**	1066.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	5.012e+04	6572.353	7.626	0.000	3.69e+04	6.34e+04
x1	0.8057	0.045	17.846	0.000	0.715	0.897
x2	-0.0268	0.051	-0.526	0.602	-0.130	0.076
x3	0.0272	0.016	1.655	0.105	-0.006	0.060

Omnibus:	14.838	**Durbin-Watson:**	1.282
Prob(Omnibus):	0.001	**Jarque-Bera (JB):**	21.442
Skew:	-0.949	**Prob(JB):**	2.21e-05
Kurtosis:	5.586	**Cond. No.**	1.40e+06

In [54]:

```
X_opt = X[:, [0,3,5]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()
```

Out[54]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.950
Model:	OLS	**Adj. R-squared:**	0.948
Method:	Least Squares	**F-statistic:**	450.8
Date:	Tue, 12 Sep 2017	**Prob (F-statistic):**	2.16e-31
Time:	19:21:17	**Log-Likelihood:**	-525.54
No. Observations:	50	**AIC:**	1057.
Df Residuals:	47	**BIC:**	1063.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	4.698e+04	2689.933	17.464	0.000	4.16e+04	5.24e+04
x1	0.7966	0.041	19.266	0.000	0.713	0.880
x2	0.0299	0.016	1.927	0.060	-0.001	0.061

Omnibus:	14.677	**Durbin-Watson:**	1.257
Prob(Omnibus):	0.001	**Jarque-Bera (JB):**	21.161
Skew:	-0.939	**Prob(JB):**	2.54e-05
Kurtosis:	5.575	**Cond. No.**	5.32e+05

In [55]:

```
X_opt = X[:, [0,3]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()
```

Out[55]:

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:	Tue, 12 Sep 2017	**Prob (F-statistic):**	3.50e-32
Time:	19:21:17	**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

Polynomial regression

In [56]:

```
dataset = pd.read_csv('Position_Salaries.csv')
X = dataset.iloc[:, 1:2].values  # constructing vector or matrix of independet variable
y = dataset.iloc[:, 2].values # create vector of dependent variable

"""# encoding the categorical data into (encode the text into numbers)

from sklearn.preprocessing import LabelEncoder
labelencoder_X = LabelEncoder()    # making object of LabelEncoder
 # transforming into label
X[:, 3]=labelencoder_X.fit_transform(X[:, 3])
from sklearn.preprocessing import OneHotEncoder # dummy encoding..convert categorical f
eature int k scheme
onehotencoder = OneHotEncoder(categorical_features = [3]) # which colm to convert here
 "0"
X = onehotencoder.fit_transform(X).toarray()
#avoid the dummy variable trap
X = X[:, 1:]


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) # create four variable """
```

Out[56]:

```
'# encoding the categorical data into (encode the text into numbers)\n\nfr
om sklearn.preprocessing import LabelEncoder\nlabelencoder_X = LabelEncode
r()    # making object of LabelEncoder\n # transforming into label\nX[:,
3]=labelencoder_X.fit_transform(X[:, 3])\nfrom sklearn.preprocessing impor
t OneHotEncoder # dummy encoding..convert categorical feature int k scheme
\nonehotencoder = OneHotEncoder(categorical_features = [3]) # which colm t
o convert here "0"\nX = onehotencoder.fit_transform(X).toarray()\n#avoid t
he dummy variable trap\nX = X[:, 1:]\n\n\nfrom sklearn.cross_validation im
port train_test_split\nX_train, X_test, y_train, y_test = train_test_split
(X, y, test_size = 0.2, random_state = 0) # create four variable '
```

In [57]:

```
#fitting linear regression on dataset
```

In [58]:

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

Out[58]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fals
e)
```

In [59]:

```
#fitting the polynomial regression on dataset
```

In [60]:

```
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree = 2)
X_poly = poly_reg.fit_transform(X)
```

In [61]:

```
X_poly
```

Out[61]:

```
array([[   1.,    1.,    1.],
       [   1.,    2.,    4.],
       [   1.,    3.,    9.],
       [   1.,    4.,   16.],
       [   1.,    5.,   25.],
       [   1.,    6.,   36.],
       [   1.,    7.,   49.],
       [   1.,    8.,   64.],
       [   1.,    9.,   81.],
       [   1.,   10.,  100.]])
```

In [62]:

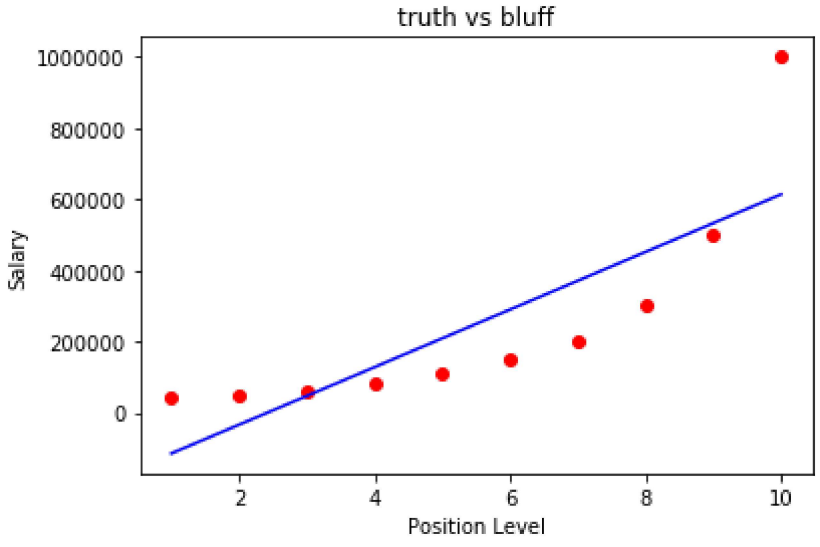
```
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly, y)
```

Out[62]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fals
e)
```

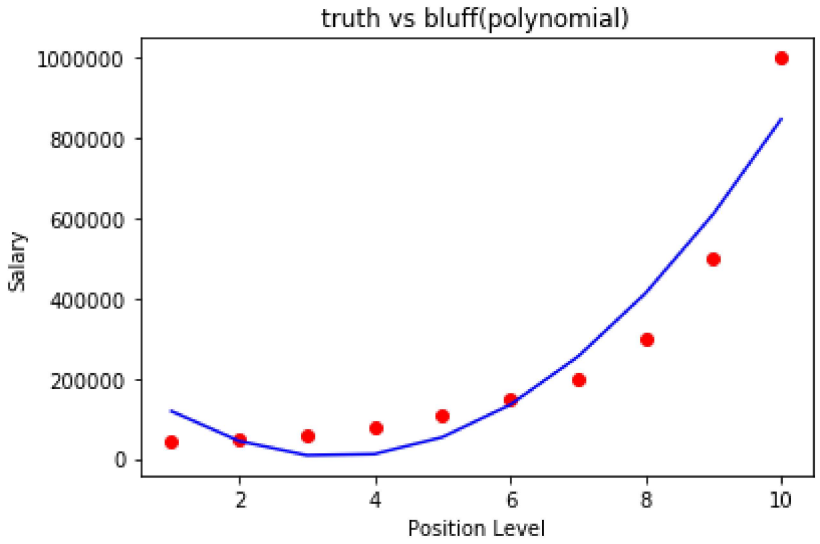
In [63]:

```
plt.scatter(X, y, color = 'red') # plotting points
plt.plot(X, lin_reg.predict(X), color = 'blue') # plotting line of prediction x_tarin
plt.title('truth vs bluff')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.show()
```



In [64]:

```
plt.scatter(X, y, color = 'red') # plotting points
plt.plot(X, lin_reg_2.predict(poly_reg.fit_transform(X)), color = 'blue') # plotting li
ne of prediction x_tarin
plt.title('truth vs bluff(polynomial)')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.show()
```



In [65]:

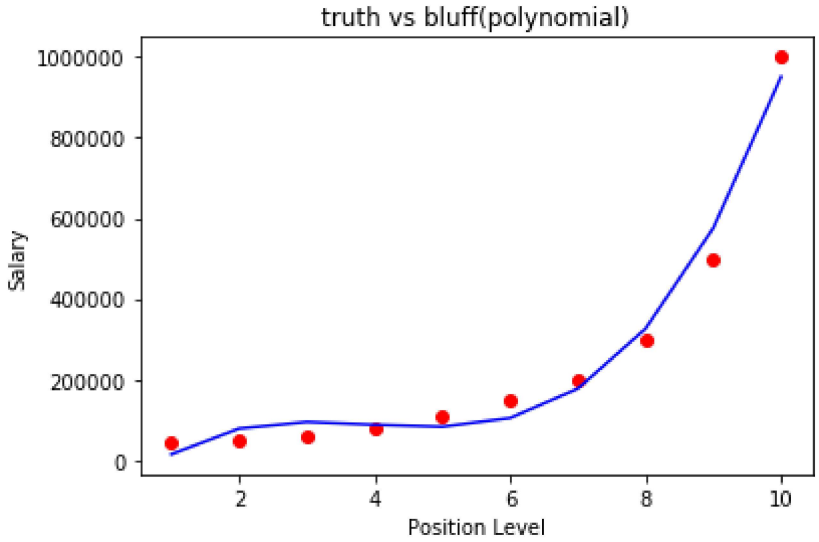
```
poly_reg = PolynomialFeatures(degree = 3)
X_poly = poly_reg.fit_transform(X)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly, y)
```

Out[65]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fals
e)
```

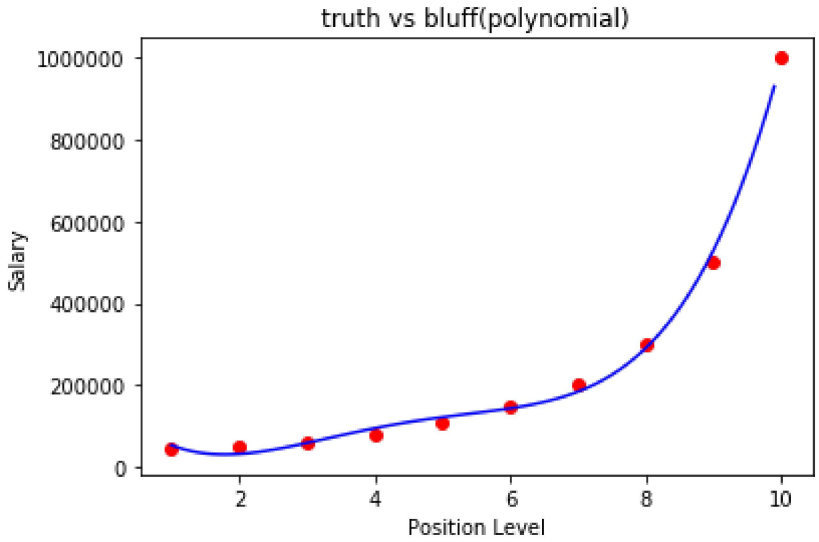
In [66]:

```
plt.scatter(X, y, color = 'red') # plotting points
plt.plot(X, lin_reg_2.predict(poly_reg.fit_transform(X)), color = 'blue') # plotting li
ne of prediction x_tarin
plt.title('truth vs bluff(polynomial)')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.show()
```



In [67]:

```
poly_reg = PolynomialFeatures(degree = 4)
X_poly = poly_reg.fit_transform(X)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly, y)
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') # plotting points
plt.plot(X_grid, lin_reg_2.predict(poly_reg.fit_transform(X_grid)), color = 'blue') # p
lotting line of prediction x_tarin
plt.title('truth vs bluff(polynomial)')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.show()
```



In [68]:

```
# predicting the new result with linear regression
lin_reg.predict(6.5)
```

Out[68]:

```
array([ 330378.78787879])
```

In [69]:

```
# predicting the new result with polynomial regression
lin_reg_2.predict(poly_reg.fit_transform(6.5))
```

Out[69]:

```
array([ 158862.45265153])
```

Support Vector Regression

In [70]:

```
dataset = pd.read_csv('Position_Salaries.csv')
X = dataset.iloc[:, 1:2].values  # constructing vector or matrix of independet variable
y = dataset.iloc[:, 2].values # create vector of dependent variable
```

In [71]:

```
#feature scaling (in svr no pre 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)))
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:44
4: DataConversionWarning: Data with input dtype int64 was converted to flo
at64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

In [72]:

```
# fitting the svr to dataset
from sklearn.svm import SVR
reg_svr = SVR(kernel = 'rbf')
reg_svr.fit(X, y)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:54
7: DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

Out[72]:

```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='auto',
  kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

In [73]:

```
#predicting the new result
y_pred = reg_svr.predict(6.5)
y_pred
```

Out[73]:

```
array([ 0.01158103])
```

In [74]:

```
y_pred = sc_y.inverse_transform(reg_svr.predict(sc_X.transform(np.array([[6.5]]))))
y_pred
```

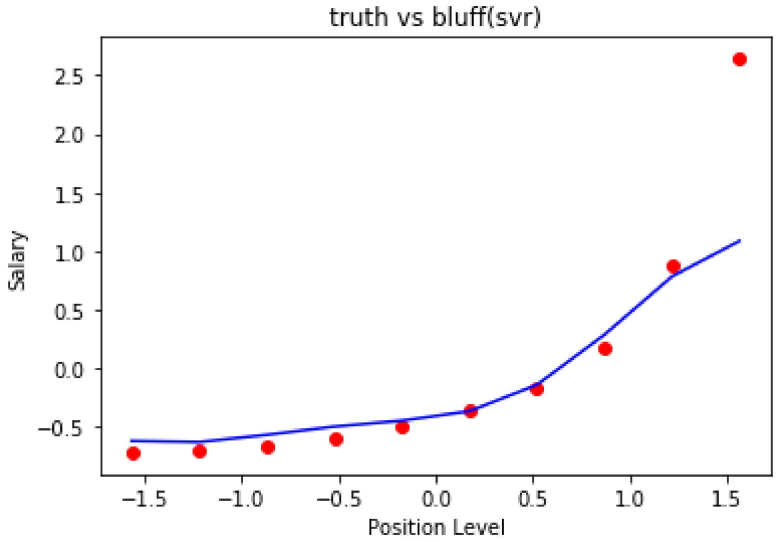
Out[74]:

```
array([ 170370.0204065])
```

In [75]:

```
#visualising the svr result

plt.scatter(X, y, color = 'red') # plotting points
plt.plot(X, reg_svr.predict((X)), color = 'blue') # plotting line of prediction
plt.title('truth vs bluff(svr)')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.show()
```



decision tree regression

In [76]:

```
dataset = pd.read_csv('Position_Salaries.csv')
X = dataset.iloc[:, 1:2].values  # constructing vector or matrix of independet variable
y = dataset.iloc[:, 2].values # create vector of dependent variable
```

In [77]:

```
# fitting DT on dataset
from sklearn.tree import DecisionTreeRegressor
reg_dt = DecisionTreeRegressor(random_state = 0)
reg_dt.fit(X, y)
```

Out[77]:

```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
           max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           presort=False, random_state=0, splitter='best')
```

In [78]:

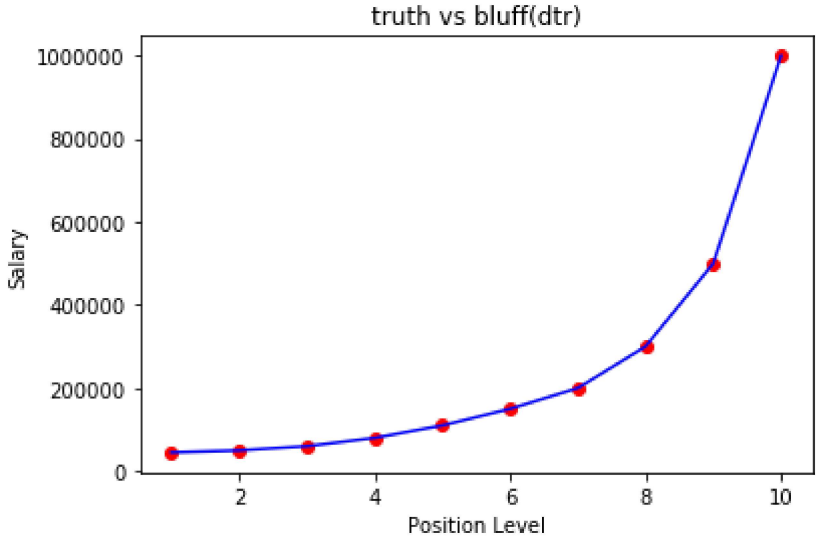
```
y_pred = reg_dt.predict(6.5)
y_pred
```

Out[78]:

```
array([ 150000.])
```

In [79]:

```
plt.scatter(X, y, color = 'red') # plotting points
plt.plot(X, reg_dt.predict((X)), color = 'blue') # plotting line of prediction
plt.title('truth vs bluff(dtr)')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.show()
```

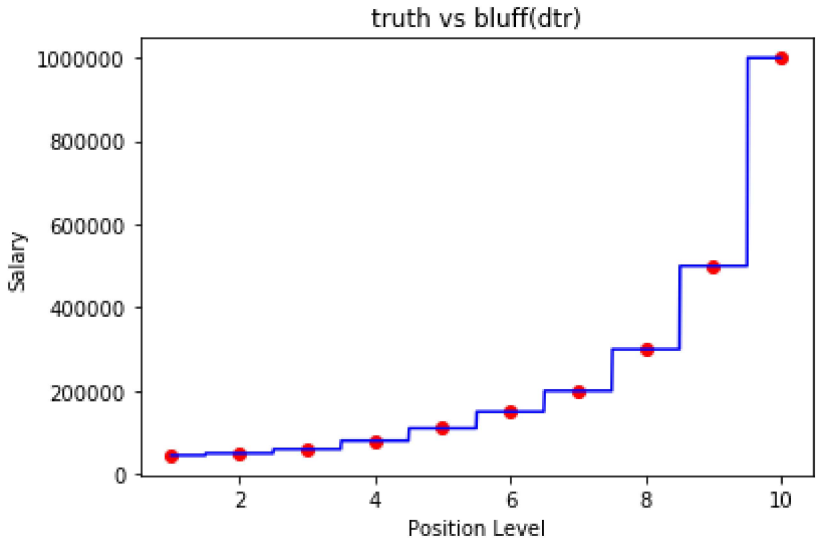


In [80]:

```
#it is non continiuos model
```

In [81]:

```
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') # plotting points
plt.plot(X_grid, reg_dt.predict(X_grid), color = 'blue') # plotting line of prediction
 x_tarin
plt.title('truth vs bluff(dtr)')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.show()
```



In [82]:

```
# here model is taking mean e.g. from 5.5 to 6.5 it will give 150K SALARY
y_pred = reg_dt.predict(5.9)
y_pred
```

Out[82]:

```
array([ 150000.])
```

Random Forest regression

In [83]:

```
from sklearn.ensemble import RandomForestRegressor
reg_rf = RandomForestRegressor(n_estimators = 400, random_state = 0)
reg_rf.fit(X, y)
```

Out[83]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=400, n_jobs=1,
           oob_score=False, random_state=0, verbose=0, warm_start=False)
```

In [84]:

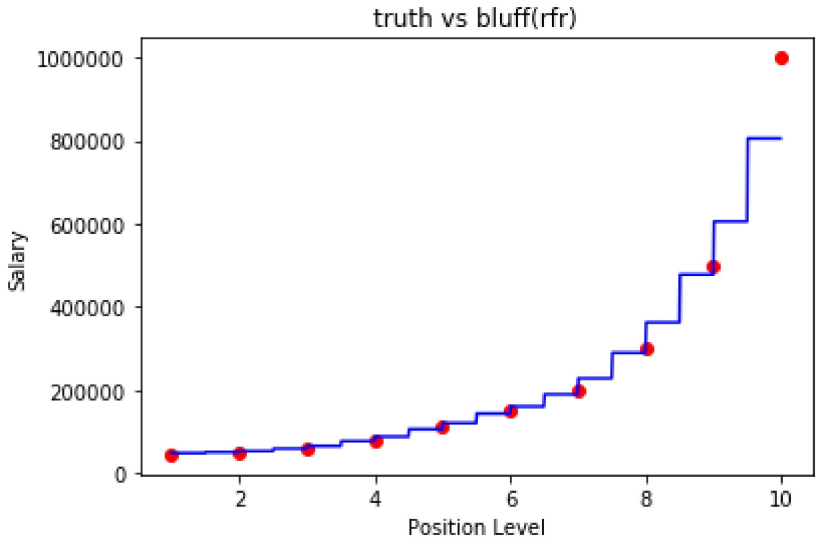
```
y_pred = reg_rf.predict(6.5)
y_pred
```

Out[84]:

```
array([ 160500.])
```

In [85]:

```
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') # plotting points
plt.plot(X_grid, reg_rf.predict(X_grid), color = 'blue') # plotting line of prediction
 x_tarin
plt.title('truth vs bluff(rfr)')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.show()
```



Logistic Regression

In [125]:

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2,3]].values  # constructing vector or matrix of independet var
iable
y = dataset.iloc[:, 4].values # create vector of dependent variable
dataset.head()
```

Out[125]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

In [87]:

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_stat
e = 0) # create four variable
```

In [88]:

```
#feature scaling (in svr no pre feature scaling)
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_y.fit_transform(X_test)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:44
4: DataConversionWarning: Data with input dtype int64 was converted to flo
at64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

In [89]:

```
# fitting the logistic regression to  training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

Out[89]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=0, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [90]:

```
# predicting the test result
y_pred = classifier.predict(X_test)
y_pred
```

Out[90]:

```
array([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, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0,
0,
       1, 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], dtype=int64)
```

In [91]:

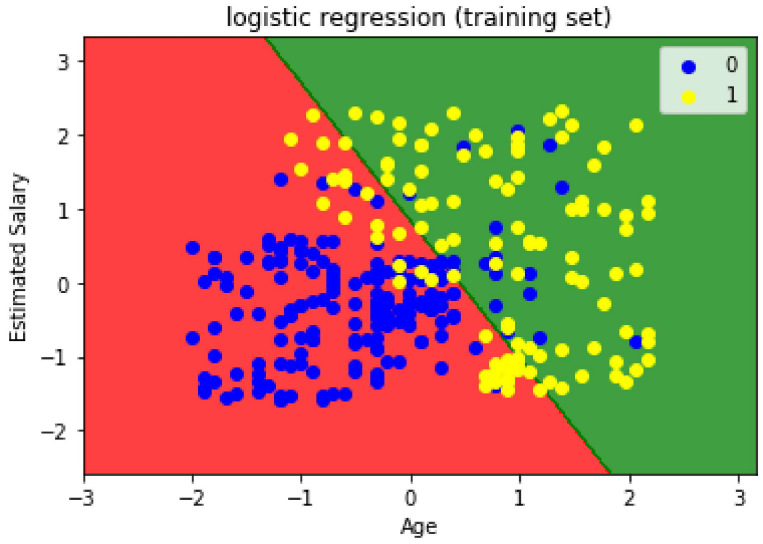
```
# making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[91]:

```
array([[63,  5],
       [ 7, 25]], dtype=int64)
```

In [92]:

```
# visualising the training set result
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(X
1.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(('blue',
'yellow'))(i),label = j)
plt.title('logistic regression (training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

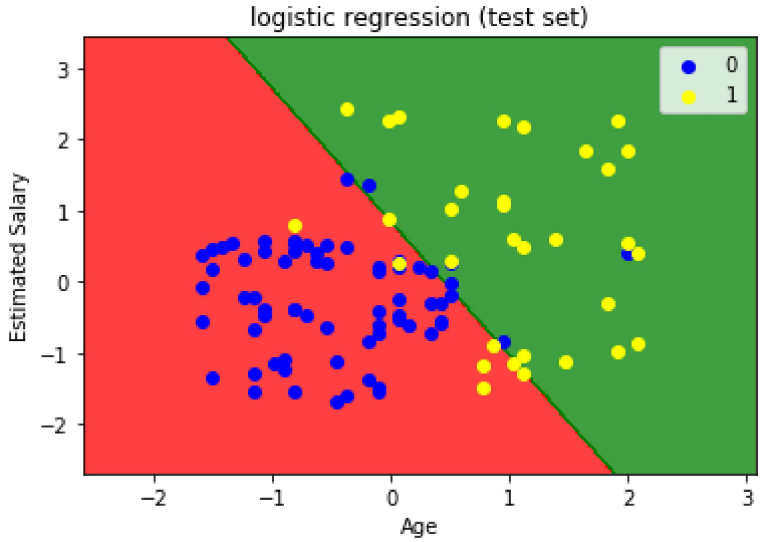


In [93]:

```
# yellow point who buy the suv,blue point who didn't buy suv
# green region where people buy suv
# all the pixel predicting 0 are in red region and 1 predicting in green region
```

In [94]:

```
# visualising the test set result
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(X
1.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(('blue',
'yellow'))(i),label = j)
plt.title('logistic regression (test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



KNN

In [95]:

```
#FITTING THE CLAASIFIER
from sklearn.neighbors import KNeighborsClassifier
classifier_knn = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier_knn.fit(X_train, y_train)
```

Out[95]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
```

In [96]:

```
# predicting the test result
y_pred = classifier_knn.predict(X_test)
y_pred
```

Out[96]:

```
array([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], dtype=int64)
```

In [97]:

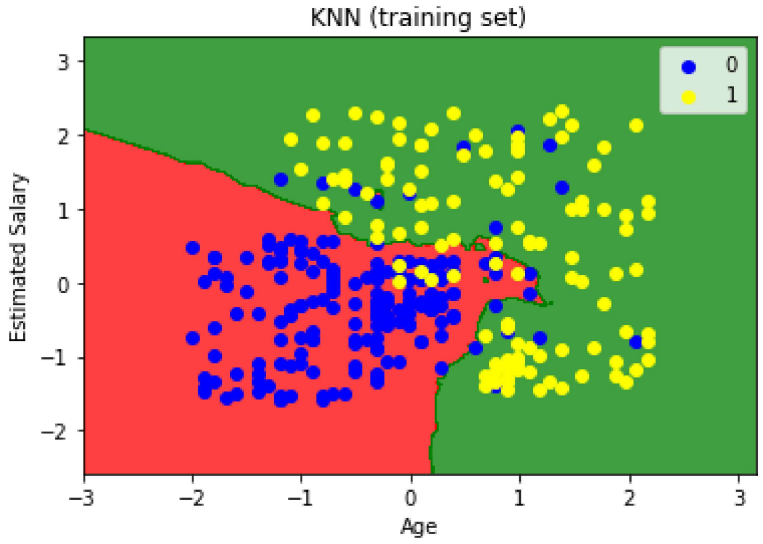
```
# making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[97]:

```
array([[64,  4],
       [ 3, 29]], dtype=int64)
```

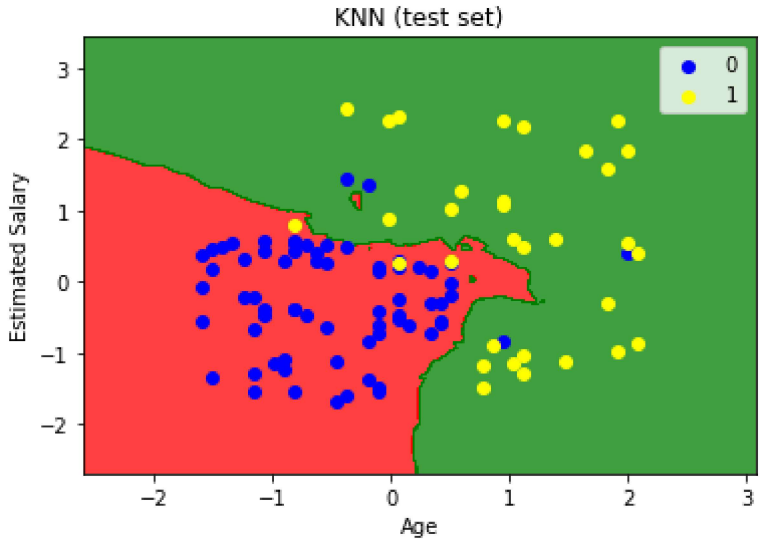
In [98]:

```
# visualising the training set result
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_knn.predict(np.array([X1.ravel(), X2.ravel()]).T).resha
pe(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(('blue',
'yellow'))(i),label = j)
plt.title('KNN (training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



In [99]:

```
# visualising the test set result
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_knn.predict(np.array([X1.ravel(), X2.ravel()]).T).re
shape(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(('blue'
, 'yellow'))(i),label = j)
plt.title('KNN (test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



SVM

In [100]:

```
from sklearn.svm import SVC
classifier_svm = SVC(kernel  = 'linear', random_state = 0)
classifier_svm.fit(X_train, y_train)
```

Out[100]:

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=0, shrinking=True,
  tol=0.001, verbose=False)
```

In [101]:

```
# predicting the test result
y_pred = classifier_knn.predict(X_test)
y_pred
```

Out[101]:

```
array([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], dtype=int64)
```

In [102]:

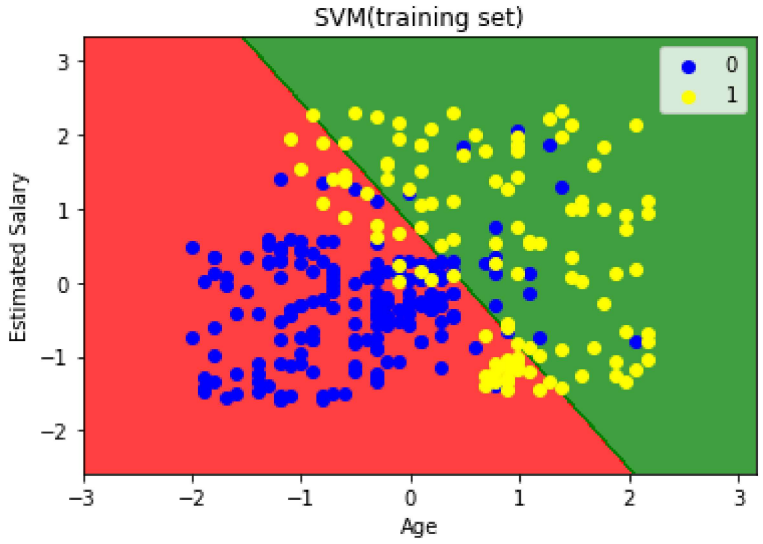
```
# making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[102]:

```
array([[64,  4],
       [ 3, 29]], dtype=int64)
```

In [103]:

```
# visualising the training set result
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_svm.predict(np.array([X1.ravel(), X2.ravel()]).T).resha
pe(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(('blue',
'yellow'))(i),label = j)
plt.title('SVM(training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



kernal svm

In [104]:

```
from sklearn.svm import SVC
classifier_rbf = SVC(kernel  = 'rbf', random_state = 0)
classifier_rbf.fit(X_train, y_train)
```

Out[104]:

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=0, shrinking=True,
  tol=0.001, verbose=False)
```

In [105]:

```
# predicting the test result
y_pred = classifier_rbf.predict(X_test)
y_pred
```

Out[105]:

```
array([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], dtype=int64)
```

In [106]:

```
# making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[106]:

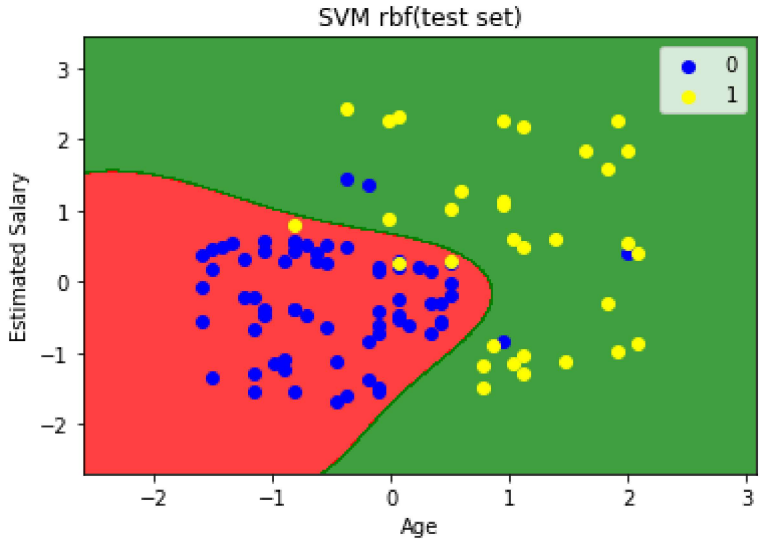
```
array([[64,  4],
       [ 3, 29]], dtype=int64)
```

In [107]:

```
# visualising the training set result
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_rbf.predict(np.array([X1.ravel(), X2.ravel()]).T).resha
pe(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(('blue',
'yellow'))(i),label = j)
plt.title('SVM rbf(training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

In [108]:

```
# visualising the test set result
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_rbf.predict(np.array([X1.ravel(), X2.ravel()]).T).resha
pe(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(('blue',
'yellow'))(i),label = j)
plt.title('SVM rbf(test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



Naive bayes

In [109]:

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

Out[109]:

```
GaussianNB(priors=None)
```

In [110]:

```
# predicting the test result
y_pred = classifier_nb.predict(X_test)
y_pred
```

Out[110]:

```
array([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, 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], dtype=int64)
```

In [111]:

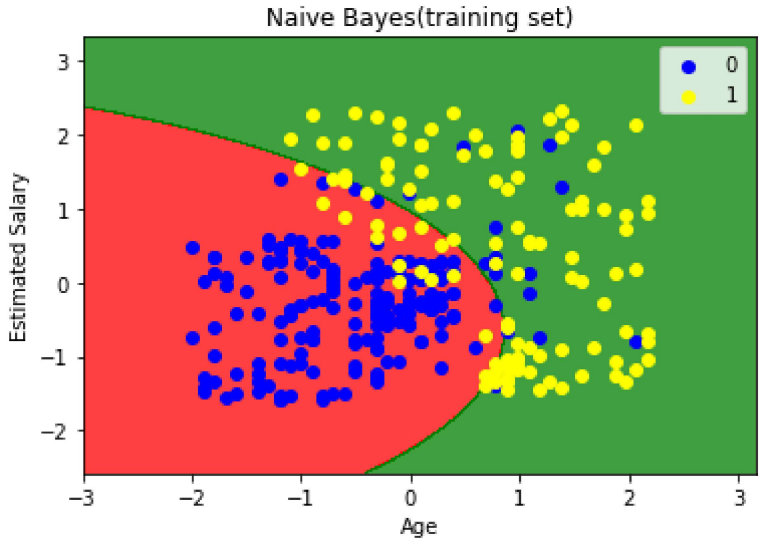
```
# making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[111]:

```
array([[64,  4],
       [ 5, 27]], dtype=int64)
```

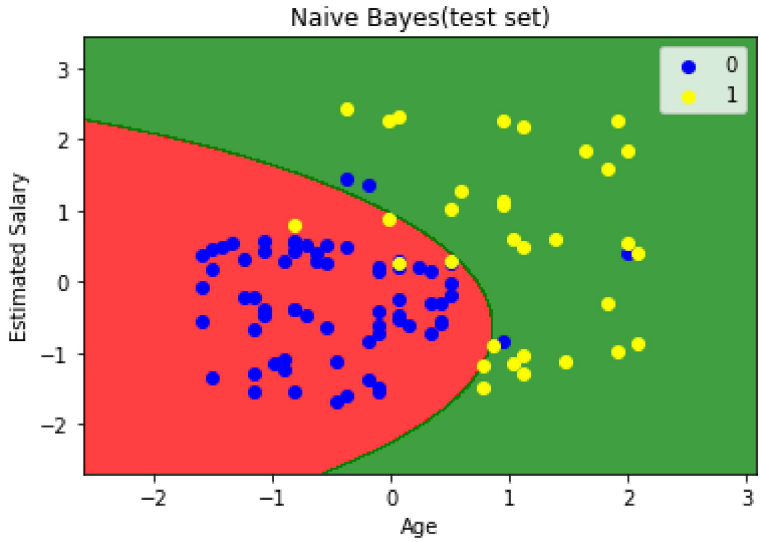
In [112]:

```
# visualising the training set result
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_nb.predict(np.array([X1.ravel(), X2.ravel()]).T).reshap
e(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(('blue',
'yellow'))(i),label = j)
plt.title('Naive Bayes(training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



In [113]:

```
# visualising the test set result
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_nb.predict(np.array([X1.ravel(), X2.ravel()]).T).res
hape(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(('blue'
, 'yellow'))(i),label = j)
plt.title('Naive Bayes(test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



Decision Tree Classifier

In [114]:

```
from sklearn.tree import DecisionTreeClassifier
classifier_dtc = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier_dtc.fit(X_train, y_train)
```

Out[114]:

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=N
one,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')
```

In [115]:

```
# predicting the test result
y_pred = classifier_dtc.predict(X_test)
y_pred
```

Out[115]:

```
array([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, 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,
       1, 1, 1, 0, 0, 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], dtype=int64)
```

In [116]:

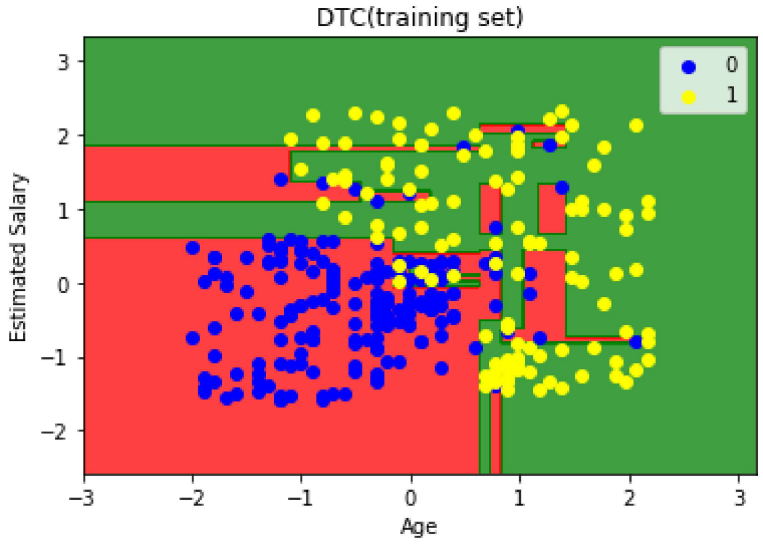
```
# making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[116]:

```
array([[61,  7],
       [ 3, 29]], dtype=int64)
```

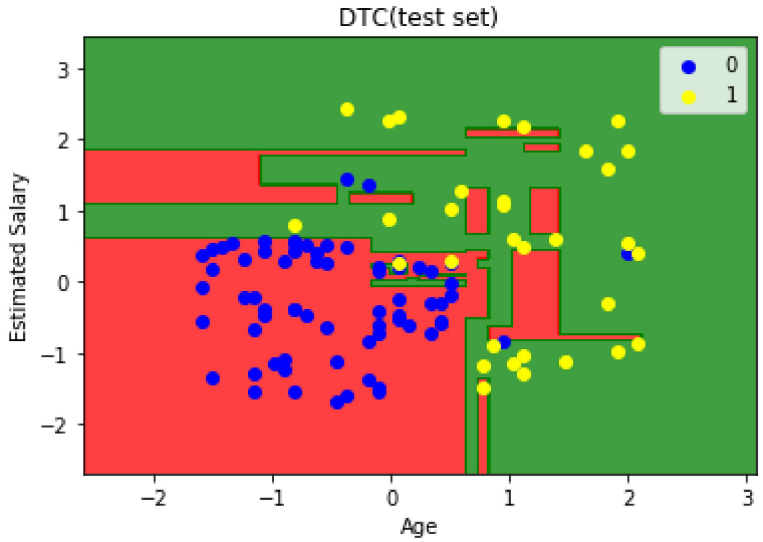
In [117]:

```
# visualising the training set result
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_dtc.predict(np.array([X1.ravel(), X2.ravel()]).T).resha
pe(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(('blue',
'yellow'))(i),label = j)
plt.title('DTC(training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



In [118]:

```
# visualising the test set result
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_dtc.predict(np.array([X1.ravel(), X2.ravel()]).T).re
shape(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(('blue'
, 'yellow'))(i),label = j)
plt.title('DTC(test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



In [119]:

```
# using "gini"
from sklearn.tree import DecisionTreeClassifier
classifier_dtc = DecisionTreeClassifier(criterion = 'gini', random_state = 0)
classifier_dtc.fit(X_train, y_train)
```

Out[119]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=Non
e,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')
```

In [120]:

```
# predicting the test result
y_pred = classifier_dtc.predict(X_test)
y_pred
```

Out[120]:

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0,
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, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0,
0,
       1, 1, 1, 0, 0, 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], dtype=int64)
```

In [121]:

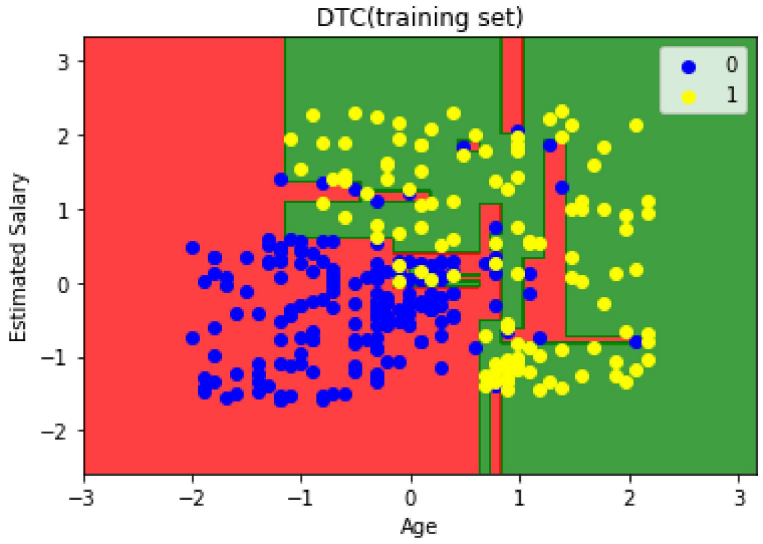
```
# making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[121]:

```
array([[61,  7],
       [ 5, 27]], dtype=int64)
```

In [122]:

```
# visualising the training set result
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_dtc.predict(np.array([X1.ravel(), X2.ravel()]).T).resha
pe(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(('blue',
'yellow'))(i),label = j)
plt.title('DTC(training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



In [123]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [124]:

```
classifier_dtc = DecisionTreeClassifier()
```