Case Study Based Project
on
Classification Models in Machine Learning using dataset of a car company

Overview of the project

There is a wide variety of classification applications from medicine to marketing. Classification models include linear models like Logistic Regression, SVM, and nonlinear ones like K-NN, Kernel SVM and Random Forests.

In this project, I will going to implement the following Machine Learning Classification models:

1. Logistic Regression
2. K-Nearest Neighbors (K-NN)
3. Support Vector Machine (SVM)
4. Kernel SVM
5. Naive Bayes
6. Decision Tree Classification
7. Random Forest Classification

And after implement all these classification models on the dataset, I will check, which model is showing higher accuracy, when it predict purchased variable based on dataset.

Dataset

This is the dataset, I will be using for all classification models. In this dataset, two independent variables(Age & Estimated Salary) and one dependent variable(Purchase) with 400 observations.

Age	Estimated Salary	Purchased	Age	Estimated Salary	Purchased	Age	Estimated Salary	Purchased	Age	Estimated Salary	Purchase
19	19000	0	21	16000	0	32	120000	1	28	123000	1
35	20000	0	28	44000	0	59	83000	0	35	73000	0
26	43000	0	27	90000	0	24	58000	0	28	37000	0
27	57000	0	35	27000	0	24	19000	0	27	88000	0
19	76000	0	33	28000	0	23	82000	0	28	59000	0
27	58000	0	30	49000	0	22	63000	0	32	86000	0
27	84000	0	26	72000	0	31	68000	0	33	149000	1
32	150000	1	27	31000	0	25	80000	0	19	21000	0
25	33000	0	27	17000	0	24	27000	0	21	72000	0
35	65000	0	33	51000	0	20	23000	0	26	35000	0
26	80000	0	35	108000	0	33	113000	0	27	89000	0
26	52000	0	30	15000	0	32	18000	0	26	86000	0
20	86000	0	28	84000	0	34	112000	1	38	80000	0
32	18000	0	23	20000	0	18	52000	0	39	71000	0
18	82000	0	25	79000	0	22	27000	0	37	71000	0
29	80000	0	27	54000	0	28	87000	0	38	61000	0
47	25000	1	30	135000	1	26	17000	0	37	55000	0
45	26000	1	31	89000	0	30	80000	0	42	80000	0
46	28000	1	24	32000	0	39	42000	0	40	57000	0
48	29000	1	18	44000	0	20	49000	0	35	75000	0
45	22000	1	29	83000	0	35	88000	0	36	52000	0
47	49000	1	35	23000	0	30	62000	0	40	59000	0
48	41000	1	27	58000	0	31	118000	1	41	59000	0
45	22000	1	24	55000	0	24	55000	0	36	75000	0
46	23000	1	23	48000	0	28	85000	0	37	72000	0
47	20000	1	28	79000	0	26	81000	0	40	75000	0

Thought line of case study project

Let’s imagine dataset I am going to use is one of the favorite car company and I am a Business Analysts for this car company. The General Manager of this car company asked me to predict which customers will buy that brand new SVU car, with highest conversion rate.

The General Manager provided me dataset. In this dataset each row corresponds to different customers with their age & salary, and purchased as dependent variable telling whether or not these customers bought previously some older SVU’s of this car company. So this car company has basically launch a new model of SVU. And all zero’s and one’s that you see in the dataset for each customer is saying whether or not these customers have bought one of those previous SUV’s so that your model will be trained on this dataset.

And for new customers having different age & different estimates salary we will predict if yes or no, they will buy that new SUV. So purchased “0” means that customer didn’t buy any previous SUV and “1” means that the customer bought some previous SUV’s. and therefore, all the future preditions that will be equal to one will probably mean that customer has a high chance to buy the new SUV. If, of course, we offer a great deal. And finally, once we predict the customers that are going to buy the SUV, well, the final step of the strategy will be for advertising team to post ads will be targeted to the customers where we predict that they are going to buy that new SUV.

So finally idea is predict the model based on the dataset and then the advertising team will use the results of this predictive model to optimize the targeting of future customers.

Logistic Regression

Importing the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing the dataset

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
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 = 0.25, random_state = 0)
```

```
print(X_train)
```

```
[    35  50000]
[    42  73000]
[    47  49000]
[    59  29000]
[    49  65000]
[    45 131000]
[    31  89000]
[    46  82000]
[    47  51000]
[    26  15000]
[    60 102000]
[    38 112000]
[    40 107000]
[    42  53000]
[    35  59000]
[    48  41000]
[    48 134000]
[    38 113000]
[    29 148000]
[    26  15000]
[    60  42000]
```

```
[    60  42000]
[    24  19000]
[    42 149000]
[    46  96000]
[    28  59000]
[    39  96000]
[    28  89000]
[    41  72000]
[    45  26000]
[    33  69000]
[    20  82000]
[    31  74000]
[    42  80000]
[    35  72000]
[    33 149000]
[    40  71000]
[    51 146000]
[    46  79000]
[    35  75000]
[    38  51000]
[    36  75000]
[    37  78000]
[    38  61000]
[    60 108000]
[    20  82000]
[    57  74000]
[    42  65000]
[    26  80000]
[    46 117000]
[    35  61000]
[    21  68000]
[    28  44000]
[    41  87000]
[    37  33000]
[    27  90000]
[    39  42000]
[    28 123000]
[    31 118000]
```

```
print(y_train)
```

```
[0 1 0 1 1 1 0 0 0 0 0 0 1 1 1 0 1 0 0 1 0 1 0 1 0 0 1 1 1 1 0 1 0 1 0 0 1
 0 0 1 0 0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 1 0 1
 1 1 0 0 1 1 0 0 1 1 0 1 0 0 1 1 0 1 1 1 0 0 0 0 0 1 0 0 1 1 1 1 1 0 1 1 0
 1 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 1 0 1 1 0 1 0 0 0 0 1 0 0 0 1 1 0 0
 0 0 1 0 1 0 0 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0 0 0 0 1 0 0
 0 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0
 0 1 1 0 0 0 0 1 0 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0
 0 0 1 0 1 1 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1
 0 0 0 0]
```

```
print(X_test)
```

```
[[    30  87000]
 [    38  50000]
```

```
[    35  75000]
[    30  79000]
[    35  50000]
[    27  20000]
[    31  15000]
[    36 144000]
[    18  68000]
[    47  43000]
[    30  49000]
[    28  55000]
[    37  55000]
[    39  77000]
[    20  86000]
[    32 117000]
[    37  77000]
[    19  85000]
[    55 130000]
[    35  22000]
[    35  47000]
[    47 144000]
[    41  51000]
[    47 105000]
[    23  28000]
[    49 141000]
[    28  87000]
[    29  80000]
[    37  62000]
[    32  86000]
[    21  88000]
[    37  79000]
[    57  60000]
[    37  53000]
[    24  58000]
[    18  52000]
[    22  81000]
[    34  43000]
[    31  34000]
[    49  36000]
[    27  88000]
[    41  52000]
[    27  84000]
[    35  20000]
[    43 112000]
[    27  58000]
[    37  80000]
[    52  90000]
[    26  30000]
[    49  86000]
[    57 122000]
[    34  25000]
[    35  57000]
[    34 115000]
[    59  88000]
[    45  32000]
[    29  83000]
[    26  80000]
```

Feature Scaling

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

```
print(X_train)
```

```
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761  1.89663484]
 [ 1.37390747 -1.40858358]
 [ 1.47293972  0.99784738]
 [ 0.08648817 -0.79972756]
 [-0.01254409 -0.24885782]
 [-0.21060859 -0.5677824 ]
 [-0.21060859 -0.19087153]
 [-0.30964085 -1.29261101]
 [-0.30964085 -0.5677824 ]
 [ 0.38358493  0.09905991]
 [ 0.8787462  -0.59677555]
 [ 2.06713324 -1.17663843]
 [ 1.07681071 -0.13288524]
 [ 0.68068169  1.78066227]
 [-0.70576986  0.56295021]
 [ 0.77971394  0.35999821]
 [ 0.8787462  -0.53878926]
 [-1.20093113 -1.58254245]
 [ 2.1661655   0.93986109]
 [-0.01254409  1.22979253]
 [ 0.18552042  1.08482681]
 [ 0.38358493 -0.48080297]
 [-0.30964085 -0.30684411]
 [ 0.97777845 -0.8287207 ]
 [ 0.97777845  1.8676417 ]
 [-0.01254409  1.25878567]
 [-0.90383437  2.27354572]
 [-1.20093113 -1.58254245]
 [ 2.1661655  -0.79972756]
 [-1.39899564 -1.46656987]
 [ 0.38358493  2.30253886]
 [ 0.77971394  0.76590222]
 [-1.00286662 -0.30684411]
 [ 0.08648817  0.76590222]
 [-1.00286662  0.56295021]
 [ 0.28455268  0.07006676]
 [ 0.68068169 -1.26361786]
 [-0.50770535 -0.01691267]
 [-1.79512465  0.35999821]
```

```
 [-0.70576986  0.12805305]
 [ 0.38358493  0.30201192]
 [-0.30964085  0.07006676]
 [-0.50770535  2.30253886]
 [ 0.18552042  0.04107362]
 [ 1.27487521  2.21555943]
 [ 0.77971394  0.27301877]
 [-0.30964085  0.1570462 ]
 [-0.01254409 -0.53878926]
 [-0.21060859  0.1570462 ]
 [-0.11157634  0.24402563]
 [-0.01254409 -0.24885782]
 [ 2.1661655   1.11381995]
 [-1.79512465  0.35999821]
 [ 1.86906873  0.12805305]
```

```
print(X_test)
```

```
[[-0.80480212  0.50496393]
 [-0.01254409 -0.5677824 ]
 [-0.30964085  0.1570462 ]
 [-0.80480212  0.27301877]
 [-0.30964085 -0.5677824 ]
 [-1.10189888 -1.43757673]
 [-0.70576986 -1.58254245]
 [-0.21060859  2.15757314]
 [-1.99318916 -0.04590581]
 [ 0.8787462  -0.77073441]
 [-0.80480212 -0.59677555]
 [-1.00286662 -0.42281668]
 [-0.11157634 -0.42281668]
 [ 0.08648817  0.21503249]
 [-1.79512465  0.47597078]
 [-0.60673761  1.37475825]
 [-0.11157634  0.21503249]
 [-1.89415691  0.44697764]
 [ 1.67100423  1.75166912]
 [-0.30964085 -1.37959044]
 [-0.30964085 -0.65476184]
 [ 0.8787462   2.15757314]
 [ 0.28455268 -0.53878926]
 [ 0.8787462   1.02684052]
 [-1.49802789 -1.20563157]
 [ 1.07681071  2.07059371]
 [-1.00286662  0.50496393]
 [-0.90383437  0.30201192]
 [-0.11157634 -0.21986468]
 [-0.60673761  0.47597078]
 [-1.6960924   0.53395707]
 [-0.11157634  0.27301877]
 [ 1.86906873 -0.27785096]
 [-0.11157634 -0.48080297]
 [-1.39899564 -0.33583725]
 [-1.99318916 -0.50979612]
 [-1.59706014  0.33100506]
```

```
[-0.4086731  -0.77073441]
[-0.70576986 -1.03167271]
[ 1.07681071 -0.97368642]
[-1.10189888  0.53395707]
[ 0.28455268 -0.50979612]
[-1.10189888  0.41798449]
[-0.30964085 -1.43757673]
[ 0.48261718  1.22979253]
[-1.10189888 -0.33583725]
[-0.11157634  0.30201192]
[ 1.37390747  0.59194336]
[-1.20093113 -1.14764529]
[ 1.07681071  0.47597078]
[ 1.86906873  1.51972397]
[-0.4086731  -1.29261101]
[-0.30964085 -0.3648304 ]
[-0.4086731   1.31677196]
[ 2.06713324  0.53395707]
[ 0.68068169 -1.089659  ]
[-0.90383437  0.38899135]
```

Training the Logistic Regression model on the Training set

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

Predicting a new result

```
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

Predicting the Test set results

```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[1 1]
```

```
[0 0]
[0 0]
[1 1]
[0 0]
[1 1]
[0 0]
[1 1]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 1]
[1 1]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[1 1]
[0 0]
[1 1]
[1 1]
[0 0]
[0 0]
[0 0]
[1 1]
[0 1]
[0 0]
[0 0]
[0 1]
[0 0]
[0 0]
[1 1]
[0 0]
[0 1]

[0 0]
[1 1]
[0 0]
[0 0]
[0 0]
```

Making the Confusion Matrix

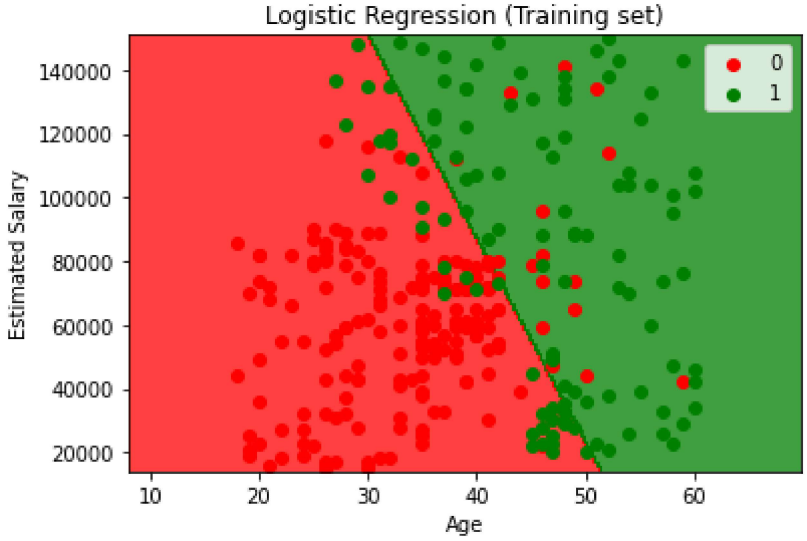
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[65  3]
 [ 8 24]]
0.89
```

Visualising the Training set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10,
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).r
             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', 'greer
plt.title('Logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

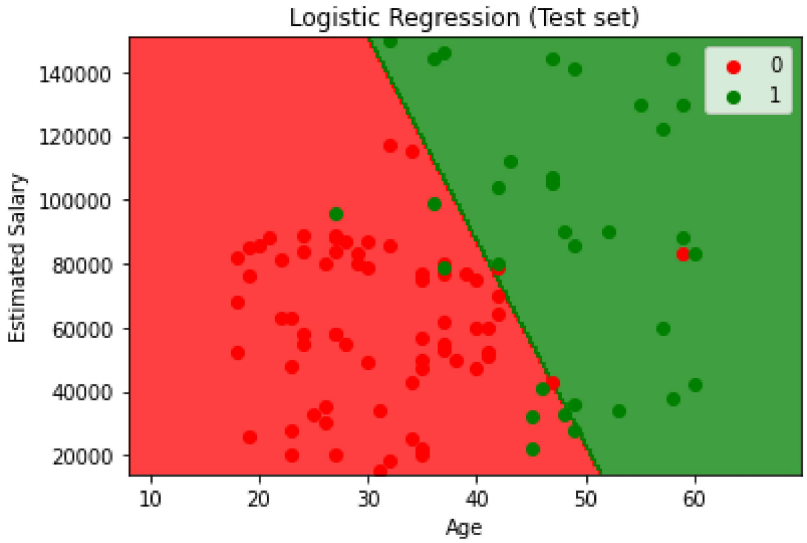
```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided
```



Visualising the Test set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10,
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).r
             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', 'greer
plt.title('Logistic Regression (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided
```



K-Nearest Neighbors (K-NN)

Importing the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing the dataset

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
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 = 0.25, random_state = 0)
```

```
print(X_train)
```

```
[[    44  39000]
 [    32 120000]
 [    38  50000]
 [    32 135000]
 [    52  21000]
 [    53 104000]
 [    39  42000]
 [    38  61000]
 [    36  50000]
 [    36  63000]
 [    35  25000]
 [    35  50000]
 [    42  73000]
 [    47  49000]
 [    59  29000]
 [    49  65000]
 [    45 131000]
 [    31  89000]
 [    46  82000]
 [    47  51000]
```

```
[    26  15000]
[    60 102000]
[    38 112000]
[    40 107000]
[    42  53000]
[    35  59000]
[    48  41000]
[    48 134000]
[    38 113000]
[    29 148000]
[    26  15000]
[    60  42000]
[    24  19000]
[    42 149000]
[    46  96000]
[    28  59000]
[    39  96000]
[    28  89000]
[    41  72000]
[    45  26000]
[    33  69000]
[    20  82000]
[    31  74000]
[    42  80000]
[    35  72000]
[    33 149000]
[    40  71000]
[    51 146000]
[    46  79000]
[    35  75000]
[    38  51000]
[    36  75000]
[    37  78000]
[    38  61000]
[    60 108000]
[    20  82000]
[    57  74000]
[    42  65000]
```

```
print(y_train)
```

```
[0 1 0 1 1 1 0 0 0 0 0 0 1 1 1 0 1 0 0 1 0 1 0 1 0 0 1 1 1 1 0 1 0 1 0 0 1
 0 0 1 0 0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 1 0 1
 1 1 0 0 1 1 0 0 1 1 0 1 0 0 1 1 0 1 1 1 0 0 0 0 0 1 0 0 1 1 1 1 1 0 1 1 0
 1 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 1 0 1 1 0 1 0 0 0 0 1 0 0 0 1 1 0 0
 0 0 1 0 1 0 0 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0 0 0 0 1 0 0
 0 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0
 0 1 1 0 0 0 0 1 0 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0
 0 0 1 0 1 1 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1
 0 0 0 0]
```

```
print(X_test)
```

```
[[    30  87000]
 [    38  50000]
```

```
[    35  75000]
[    30  79000]
[    35  50000]
[    27  20000]
[    31  15000]
[    36 144000]
[    18  68000]
[    47  43000]
[    30  49000]
[    28  55000]
[    37  55000]
[    39  77000]
[    20  86000]
[    32 117000]
[    37  77000]
[    19  85000]
[    55 130000]
[    35  22000]
[    35  47000]
[    47 144000]
[    41  51000]
[    47 105000]
[    23  28000]
[    49 141000]
[    28  87000]
[    29  80000]
[    37  62000]
[    32  86000]
[    21  88000]
[    37  79000]
[    57  60000]
[    37  53000]
[    24  58000]
[    18  52000]
[    22  81000]
[    34  43000]
[    31  34000]
[    49  36000]
[    27  88000]
[    41  52000]
[    27  84000]
[    35  20000]
[    43 112000]
[    27  58000]
[    37  80000]
[    52  90000]
[    26  30000]
[    49  86000]
[    57 122000]
[    34  25000]
[    35  57000]
[    34 115000]
[    59  88000]
[    45  32000]
[    29  83000]
[    26  80000]
```

Feature Scaling

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

```
print(X_train)
```

```
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761  1.89663484]
 [ 1.37390747 -1.40858358]
 [ 1.47293972  0.99784738]
 [ 0.08648817 -0.79972756]
 [-0.01254409 -0.24885782]
 [-0.21060859 -0.5677824 ]
 [-0.21060859 -0.19087153]
 [-0.30964085 -1.29261101]
 [-0.30964085 -0.5677824 ]
 [ 0.38358493  0.09905991]
 [ 0.8787462  -0.59677555]
 [ 2.06713324 -1.17663843]
 [ 1.07681071 -0.13288524]
 [ 0.68068169  1.78066227]
 [-0.70576986  0.56295021]
 [ 0.77971394  0.35999821]
 [ 0.8787462  -0.53878926]
 [-1.20093113 -1.58254245]
 [ 2.1661655   0.93986109]
 [-0.01254409  1.22979253]
 [ 0.18552042  1.08482681]
 [ 0.38358493 -0.48080297]
 [-0.30964085 -0.30684411]
 [ 0.97777845 -0.8287207 ]
 [ 0.97777845  1.8676417 ]
 [-0.01254409  1.25878567]
 [-0.90383437  2.27354572]
 [-1.20093113 -1.58254245]
 [ 2.1661655  -0.79972756]
 [-1.39899564 -1.46656987]
 [ 0.38358493  2.30253886]
 [ 0.77971394  0.76590222]
 [-1.00286662 -0.30684411]
 [ 0.08648817  0.76590222]
 [-1.00286662  0.56295021]
 [ 0.28455268  0.07006676]
 [ 0.68068169 -1.26361786]
 [-0.50770535 -0.01691267]
 [-1.79512465  0.35999821]
```

```
[-0.70576986  0.12805305]
[ 0.38358493  0.30201192]
[-0.30964085  0.07006676]
[-0.50770535  2.30253886]
[ 0.18552042  0.04107362]
[ 1.27487521  2.21555943]
[ 0.77971394  0.27301877]
[-0.30964085  0.1570462 ]
[-0.01254409 -0.53878926]
[-0.21060859  0.1570462 ]
[-0.11157634  0.24402563]
[-0.01254409 -0.24885782]
[ 2.1661655   1.11381995]
[-1.79512465  0.35999821]
[ 1.86906873  0.12805305]
```

```
print(X_test)
```

```
[[-0.80480212  0.50496393]
 [-0.01254409 -0.5677824 ]
 [-0.30964085  0.1570462 ]
 [-0.80480212  0.27301877]
 [-0.30964085 -0.5677824 ]
 [-1.10189888 -1.43757673]
 [-0.70576986 -1.58254245]
 [-0.21060859  2.15757314]
 [-1.99318916 -0.04590581]
 [ 0.8787462  -0.77073441]
 [-0.80480212 -0.59677555]
 [-1.00286662 -0.42281668]
 [-0.11157634 -0.42281668]
 [ 0.08648817  0.21503249]
 [-1.79512465  0.47597078]
 [-0.60673761  1.37475825]
 [-0.11157634  0.21503249]
 [-1.89415691  0.44697764]
 [ 1.67100423  1.75166912]
 [-0.30964085 -1.37959044]
 [-0.30964085 -0.65476184]
 [ 0.8787462   2.15757314]
 [ 0.28455268 -0.53878926]
 [ 0.8787462   1.02684052]
 [-1.49802789 -1.20563157]
 [ 1.07681071  2.07059371]
 [-1.00286662  0.50496393]
 [-0.90383437  0.30201192]
 [-0.11157634 -0.21986468]
 [-0.60673761  0.47597078]
 [-1.6960924   0.53395707]
 [-0.11157634  0.27301877]
 [ 1.86906873 -0.27785096]
 [-0.11157634 -0.48080297]
 [-1.39899564 -0.33583725]
 [-1.99318916 -0.50979612]
 [-1.59706014  0.33100506]
```

```
[-0.4086731  -0.77073441]
[-0.70576986 -1.03167271]
[ 1.07681071 -0.97368642]
[-1.10189888  0.53395707]
[ 0.28455268 -0.50979612]
[-1.10189888  0.41798449]
[-0.30964085 -1.43757673]
[ 0.48261718  1.22979253]
[-1.10189888 -0.33583725]
[-0.11157634  0.30201192]
[ 1.37390747  0.59194336]
[-1.20093113 -1.14764529]
[ 1.07681071  0.47597078]
[ 1.86906873  1.51972397]
[-0.4086731  -1.29261101]
[-0.30964085 -0.3648304 ]
[-0.4086731   1.31677196]
[ 2.06713324  0.53395707]
[ 0.68068169 -1.089659  ]
[-0.90383437  0.38899135]
[-1.20093113  0.30201192]
```

Training the K-NN model on the Training set

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
```

Predicting a new result

```
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

Predicting the Test set results

```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
```

```
[0 0]
[1 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[1 0]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[1 1]
[0 0]
[1 1]
[0 0]
[1 1]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 1]
[1 1]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[1 1]
[0 0]
[1 1]
[1 1]
[0 0]
[0 0]
[1 0]
[1 1]
[1 1]
[0 0]
[0 0]
```

Making the Confusion Matrix

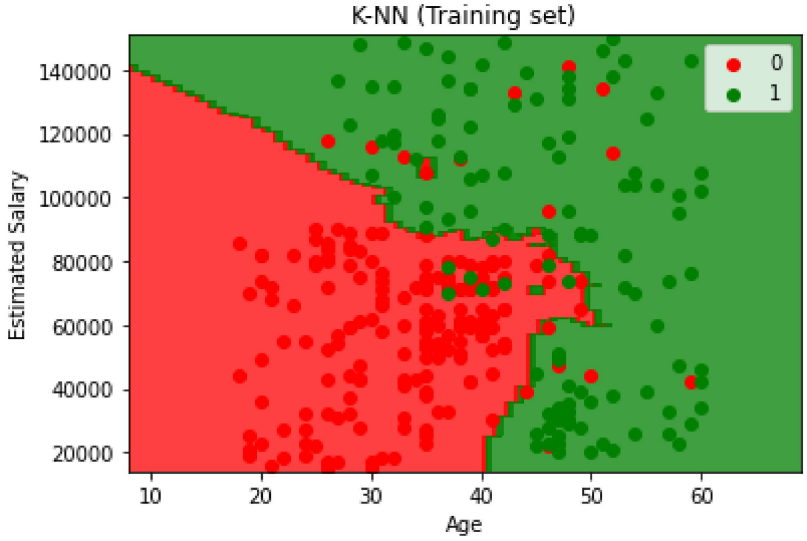
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[64  4]
 [ 3 29]]
0.93
```

Visualising the Training set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10,
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).r
             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', 'greer
plt.title('K-NN (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

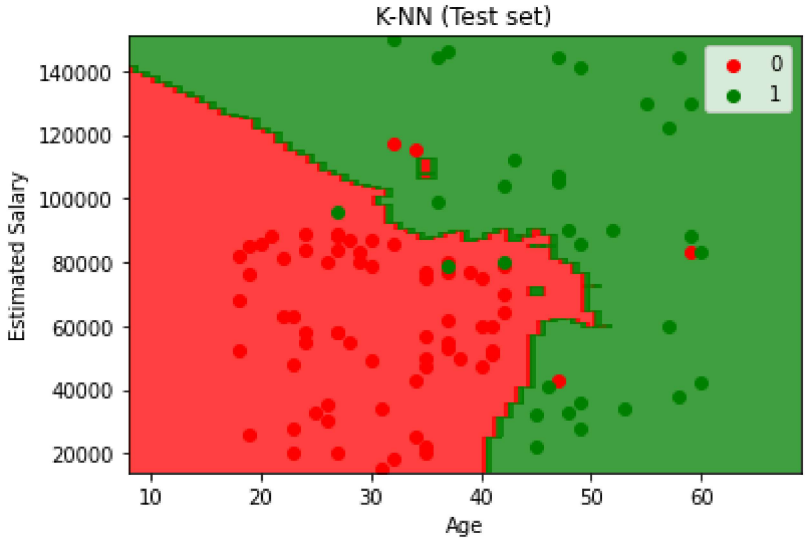
```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avc
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avc
```



Visualising the Test set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10,
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).r
             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', 'greer
plt.title('K-NN (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avc
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avc
```



Support Vector Machine (SVM)

Importing the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing the dataset

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
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 = 0.25, 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)
```

Training the SVM model on the Training set

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

Predicting a new result

```
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

Predicting the Test set results

```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [1 1]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 1]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
```

```
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[1 1]
[0 0]
[1 1]
[1 1]
[0 0]
[0 0]
[0 0]
[1 1]
[0 1]
[0 0]
[0 0]
```

Making the Confusion Matrix

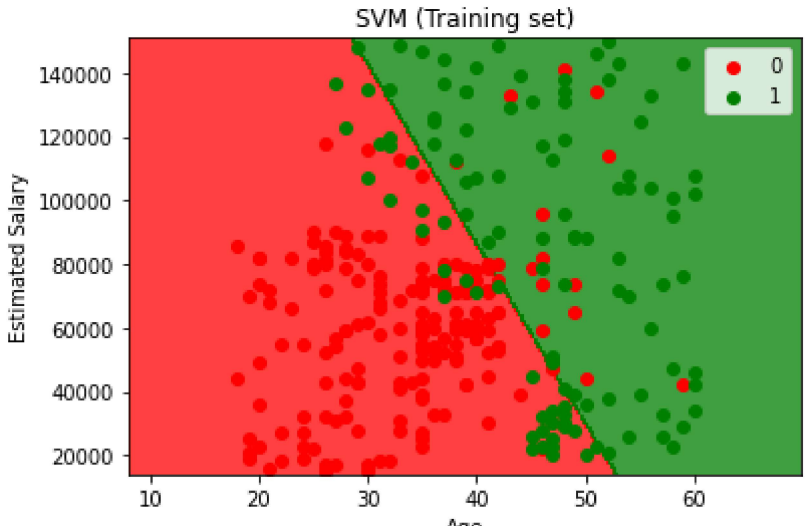
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[66  2]
 [ 8 24]]
0.9
```

Visualising the Training set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10,
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).r
             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', 'greer
plt.title('SVM (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

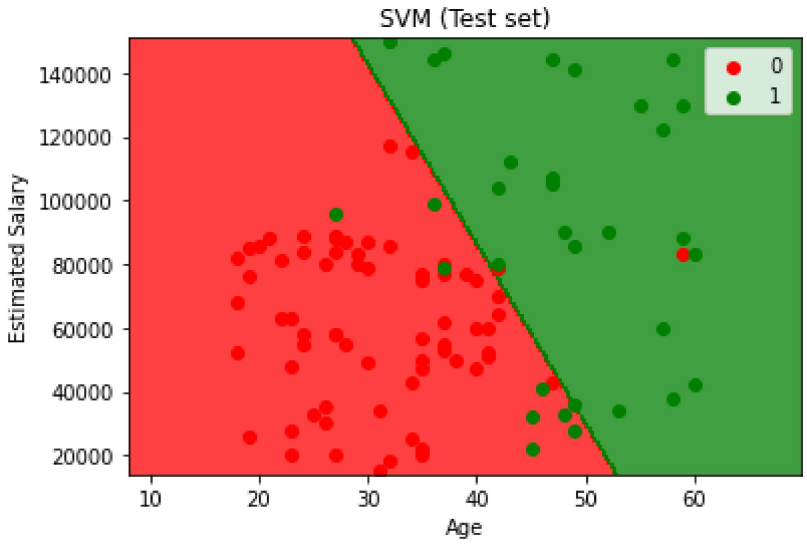
```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided
```



Visualising the Test set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10,
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).r
             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', 'greer
plt.title('SVM (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avc
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avc
```



Kernel SVM

Importing the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing the dataset

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
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 = 0.25, 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)
```

Training the Kernel SVM model on the Training set

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

Predicting a new result

```
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

Predicting the Test set results

```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[1 1]
[0 0]
[1 1]
[1 1]
[0 0]
[0 0]
[1 0]
[1 1]
[1 1]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[1 1]
[0 0]
[1 1]
[0 0]
[1 1]
[0 0]
[0 0]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[0 0]
```

```
[0 0]
[0 0]
[1 1]
[1 1]
[1 1]
[1 0]
[0 0]
[0 0]
[1 1]
[0 1]
[0 0]
[1 1]
[1 1]
[0 0]
[0 0]
[1 1]
[0 0]
```

Making the Confusion Matrix

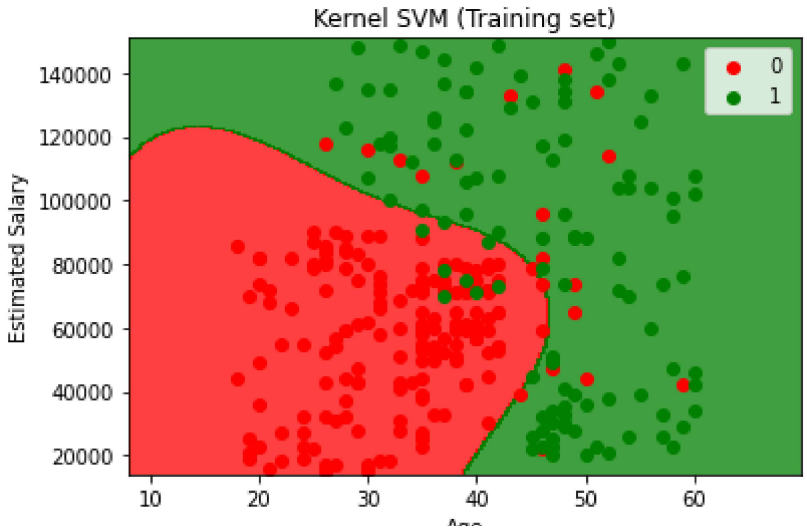
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[64  4]
 [ 3 29]]
0.93
```

Visualising the Training set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10,
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).r
             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', 'greer
plt.title('Kernel SVM (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

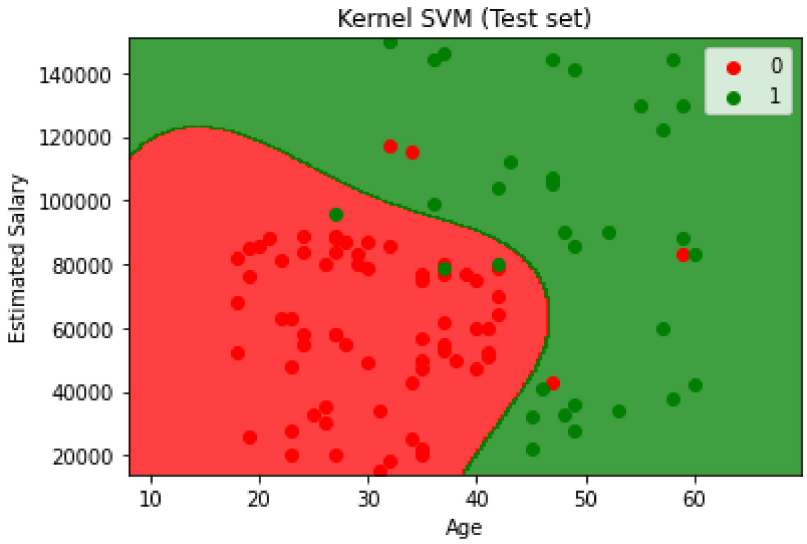
```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided
```



Visualising the Test set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10,
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).r
             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', 'greer
plt.title('Kernel SVM (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided
```



Naive Bayes

Importing the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing the dataset

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
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 = 0.25, 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)
```

Training the Naive Bayes model on the Training set

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

Predicting a new result

```
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

Predicting the Test set results

```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [1 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [1 1]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 1]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
```

```
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[1 1]
[0 0]
[1 1]
[1 1]
[0 0]
[0 0]
[1 0]
[1 1]
[0 1]
[0 0]
[0 0]
```

Making the Confusion Matrix

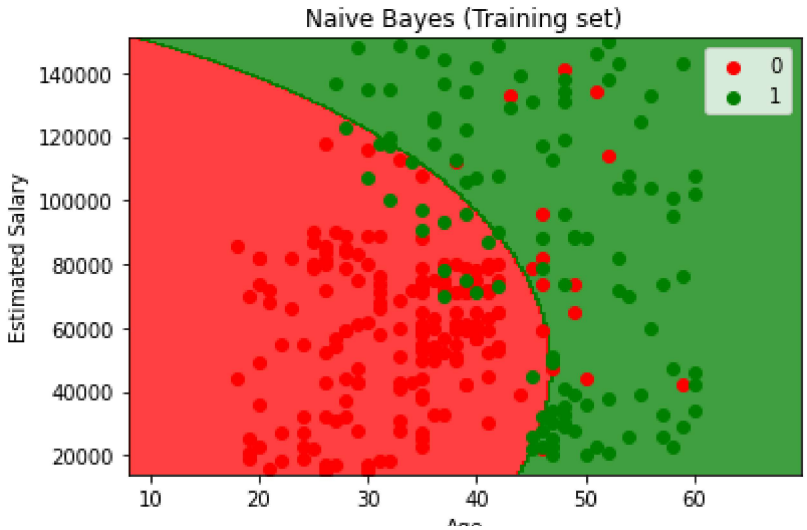
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[65  3]
 [ 7 25]]
0.9
```

Visualising the Training set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10,
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).r
             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', 'greer
plt.title('Naive Bayes (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

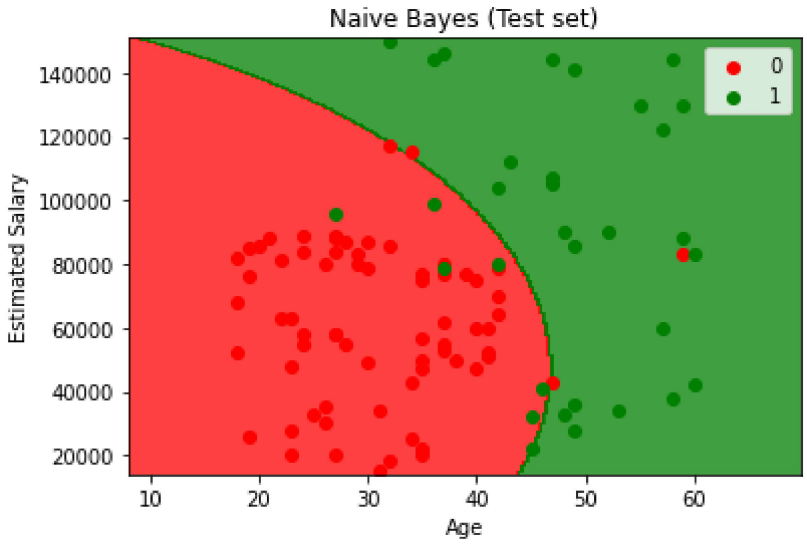
```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avc
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avc
```



Visualising the Test set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10,
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).r
             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', 'greer
plt.title('Naive Bayes (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided
```



Decision Tree Classification

Importing the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing the dataset

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
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 = 0.25, 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)
```

Training the Decision Tree Classification model on the Training set

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

Predicting a new result

```
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

Predicting the Test set results

```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 0]
 [0 0]
 [1 0]
 [1 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [1 1]
 [0 0]
 [0 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 1]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
```

```
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[1 1]
[0 0]
[1 1]
[1 1]
[0 0]
[0 0]
[1 0]
[1 1]
[1 1]
[0 0]
[0 0]
```

Making the Confusion Matrix

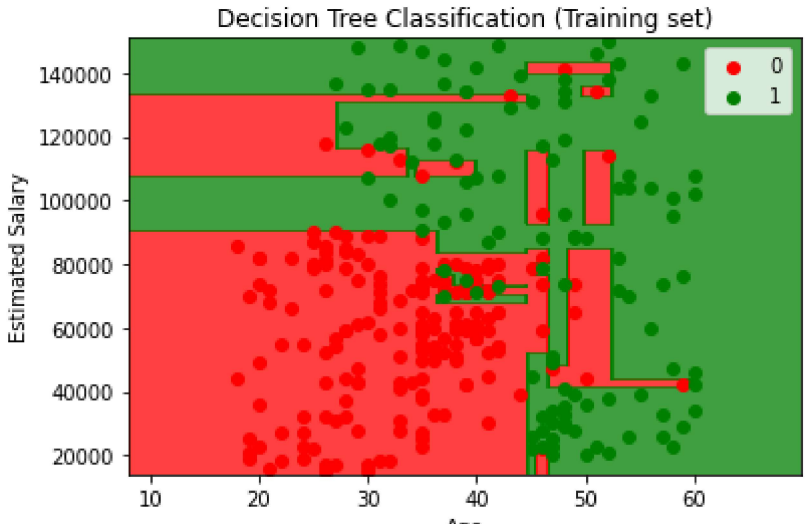
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[62  6]
 [ 3 29]]
0.91
```

Visualising the Training set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10,
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).r
             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', 'greer
plt.title('Decision Tree Classification (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avc
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avc
```



Visualising the Test set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10,
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).r
             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', 'greer
plt.title('Decision Tree Classification (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avc
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avc
```

Colab paid products - Cancel contracts here

Random Forest Classification

Importing the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing the dataset

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
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 = 0.25, 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)
```

Training the Random Forest Classification model on the Training set

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state =
classifier.fit(X_train, y_train)
```

Predicting a new result

```
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

Predicting the Test set results

```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [1 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 0]
 [1 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [1 1]
 [0 0]
 [0 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 1]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
```

```
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[1 1]
[0 0]
[1 1]
[1 1]
[0 0]
[0 0]
[1 0]
[1 1]
[1 1]
[0 0]
[0 0]
```

Making the Confusion Matrix

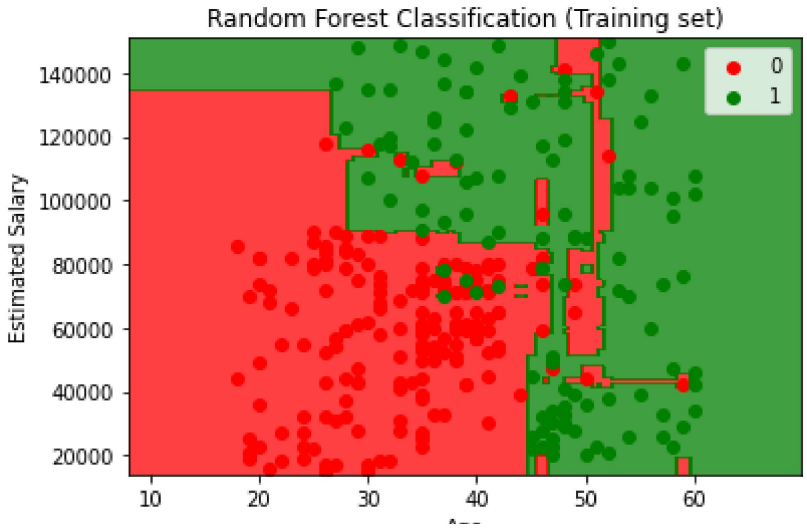
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[63  5]
 [ 4 28]]
0.91
```

Visualising the Training set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10,
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).r
             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', 'greer
plt.title('Random Forest Classification (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

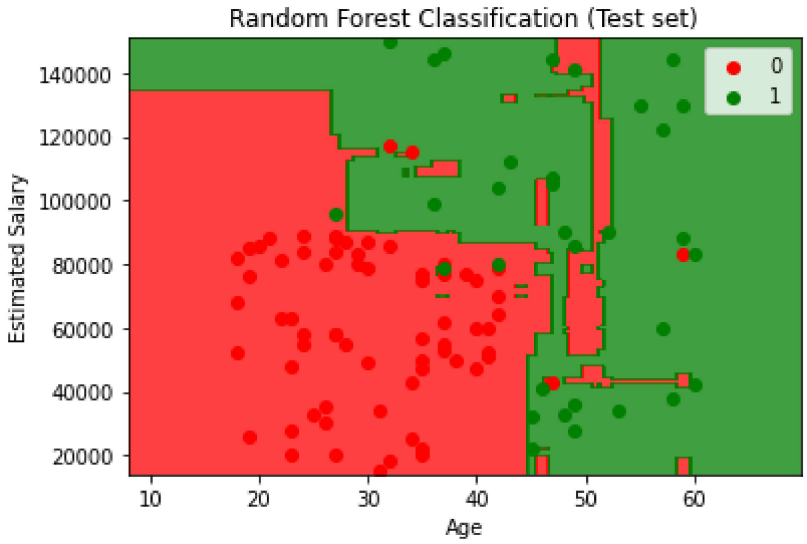
```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avc
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avc
```



Visualising the Test set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10,
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).r
             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', 'greer
plt.title('Random Forest Classification (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avc
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avc
```



Conclusion

After implement all these classification models on the dataset, here I am going to check which model is showing higher accuracy, when it predict purchased variable based on dataset.

Model	Accuracy Score(%) of the Model
Logistic Regression	89
K-Nearest Neighbors (K-NN)	93
Support Vector Machine (SVM)	90
Kernel SVM	93
Naive Bayes	90
Decision Tree Classification	91
Random Forest Classification	91

As we can see K-Nearest Neighbors and Kernel SVM having high accuracy of 93% on prediction of purchased variable on the dataset. So I can say that **K-Nearest Neighbors** and **kernel SVM** model is the best suit for this dataset for future predictions.