```
#Data handling libraries
import numpy as np
import pandas as pd

#Data Visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px


#Data Preprocessing libraries
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

```
df = pd.read_csv("/content/drive/MyDrive/Datasets/Admission_Pre
print(df)
```

```
     Serial No.  GRE Score  TOEFL Score  ...  CGPA  Research  Chance of Admit 
0             1        337          118  ...  9.65         1              0.92
1             2        324          107  ...  8.87         1              0.76
2             3        316          104  ...  8.00         1              0.72
3             4        322          110  ...  8.67         1              0.80
4             5        314          103  ...  8.21         0              0.65
..          ...        ...          ...  ...   ...       ...               ...
395         396        324          110  ...  9.04         1              0.82
396         397        325          107  ...  9.11         1              0.84
397         398        330          116  ...  9.45         1              0.91
398         399        312          103  ...  8.78         0              0.67
399         400        333          117  ...  9.66         1              0.95

[400 rows x 9 columns]
```

```
df.describe()
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CG
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.0000

```
df.columns
```

```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

25% 100.750000 308.000000 103.000000 2.000000 2.500000 3.000000 8.1700

```
df.head()
```

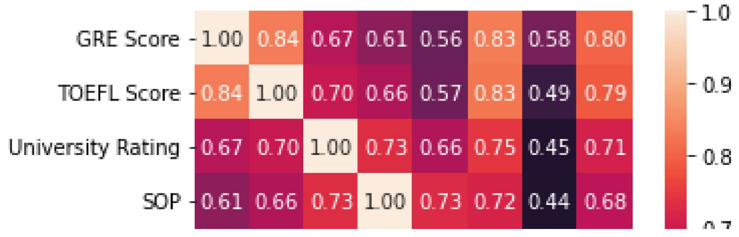
	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	1	337	118	4	4.5	4.5	9.65	1
1	2	324	107	4	4.0	4.5	8.87	1
2	3	316	104	3	3.0	3.5	8.00	1
3	4	322	110	3	3.5	2.5	8.67	1
4	5	314	103	2	2.0	3.0	8.21	0

```
#Drop the serial no.
df = df.drop(df.columns[0], axis=1)
```

Exploratory Data Analysis

Heatmap

```
heatmp = sns.heatmap(np.corrcoef(df.values.T), annot=True, cbar
plt.show()
```

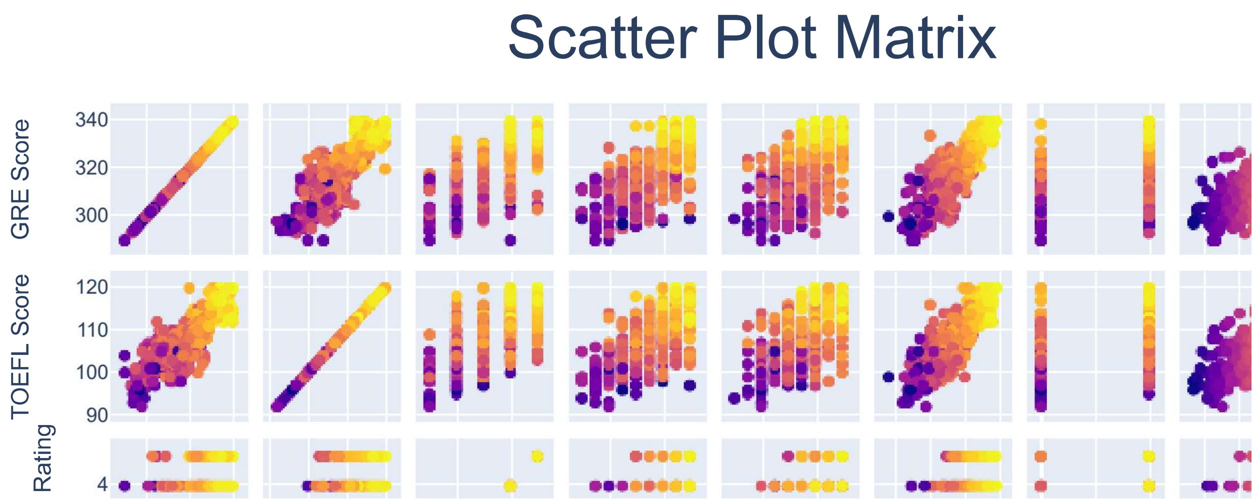


Scatter Plot Matrix

```
fig = px.scatter_matrix(df,
                         height=800, width=800,
                        color='Chance of Admit ',
                        )

fig.update_layout(font_family='Helvetica', font_size=10,
                  title=dict(text='Scatter Plot Matrix', x=0.5,
                  )

fig.show()
```



```
# s1 = df[df["Chance of Admit "]>0.80]
# s1.count()
```

```
df.loc[df["Chance of Admit "] >= 0.8, "Chance of Admit "] = 1
df.loc[df["Chance of Admit "] < 0.8, "Chance of Admit "] = 0
```

```
df.head()
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Ad
0	337	118	4	4.5	4.5	9.65	1	
1	324	107	4	4.0	4.5	8.87	1	
2	316	104	3	3.0	3.5	8.00	1	
3	322	110	3	3.5	2.5	8.67	1	
4	314	103	2	2.0	3.0	8.21	0	

```
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

```
X
```

```
array([[337.  , 118.  ,   4.  , ...,   4.5 ,   9.65,   1.  ],
       [324.  , 107.  ,   4.  , ...,   4.5 ,   8.87,   1.  ],
       [316.  , 104.  ,   3.  , ...,   3.5 ,   8.  ,   1.  ],
       ...,
       [330.  , 116.  ,   4.  , ...,   4.5 ,   9.45,   1.  ],
       [312.  , 103.  ,   3.  , ...,   4.  ,   8.78,   0.  ],
       [333.  , 117.  ,   4.  , ...,   4.  ,   9.66,   1.  ]])
```

```
y
```

```
array([1., 0., 0., 1., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 1., 1.,
       1., 1., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 0., 0., 0.,
       0., 0., 0., 0., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 1., 1., 0., 0., 0., 0., 1., 0., 1., 1., 1., 0., 0., 0., 1., 1.,
       0., 0., 1., 0., 1., 1., 1., 1., 1., 1., 0., 1., 1., 0., 1., 1., 1.,
       0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
       0., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1.,
       0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 1., 1., 1., 1.,
       0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1., 0., 0., 0., 0., 1., 1.,
       0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
       0., 0., 0., 0., 1., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1.,
       0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0.,
       0., 1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
       0., 0., 1., 0., 0., 1., 0., 1., 0., 0., 0., 0., 1., 0., 1., 1., 1.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 1., 1., 1., 1., 0., 0., 1., 0., 0., 0., 0., 0., 1., 1., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 1., 1., 0., 0., 0., 0., 0.,
       0., 1., 0., 1., 1., 1., 1., 0., 1.])
```

Splitting the dataset into the Training set and Test set

[ ] ↳ 5 cells hidden

Feature Scaling

[ ] ↳ 3 cells hidden

Training the Decision Tree Classification model on the Training set

[ ] ↳ 1 cell hidden

Predicting

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

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

Confusion Matrix

```
from sklearn.metrics import confusion_matrix
matrix = confusion_matrix(y_test, y_pred, labels=[0, 1])
print(matrix)
```

```
[[64  7]
 [ 4 25]]
```

Classification Report

```
from sklearn.metrics import classification_report
cr = classification_report(y_test, y_pred)
print(cr)
```

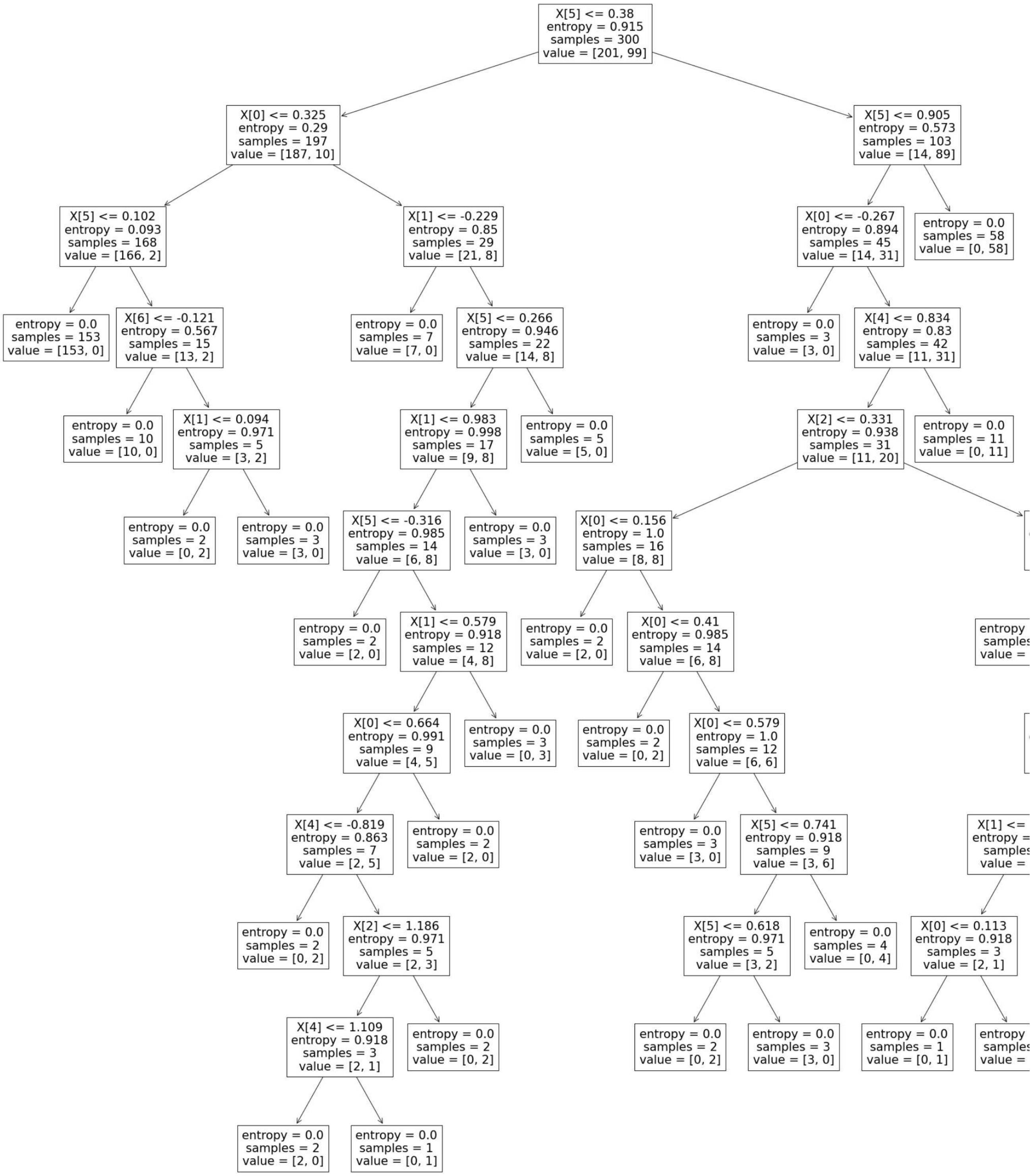
```
              precision    recall  f1-score   support

         0.0       0.94      0.90      0.92        71
         1.0       0.78      0.86      0.82        29

    accuracy                           0.89       100
   macro avg       0.86      0.88      0.87       100
weighted avg       0.89      0.89      0.89       100
```

Decision Tree

```
from sklearn.tree import plot_tree
fig = plt.figure(figsize=(40, 40))
plot_tree(classifier)
plt.show()
```


X[5] <= 0.38
entropy = 0.915
samples = 300
value = [201, 99]
X[0] <= 0.325
entropy = 0.29
samples = 197
value = [187, 10]
X[5] <= 0.905
entropy = 0.573
samples = 103
value = [14, 89]
X[5] <= 0.102
entropy = 0.093
samples = 168
value = [166, 2]
X[1] <= -0.229
entropy = 0.85
samples = 29
value = [21, 8]
X[0] <= -0.267
entropy = 0.894
samples = 45
value = [14, 31]
entropy = 0.0
samples = 58
value = [0, 58]
entropy = 0.0
samples = 153
value = [153, 0]
X[6] <= -0.121
entropy = 0.567
samples = 15
value = [13, 2]
entropy = 0.0
samples = 7
value = [7, 0]
X[5] <= 0.266
entropy = 0.946
samples = 22
value = [14, 8]
entropy = 0.0
samples = 3
value = [3, 0]
X[4] <= 0.834
entropy = 0.83
samples = 42
value = [11, 31]
entropy = 0.0
samples = 10
value = [10, 0]
X[1] <= 0.094
entropy = 0.971
samples = 5
value = [3, 2]
X[1] <= 0.983
entropy = 0.998
samples = 17
value = [9, 8]
entropy = 0.0
samples = 5
value = [5, 0]
X[2] <= 0.331
entropy = 0.938
samples = 31
value = [11, 20]
entropy = 0.0
samples = 11
value = [0, 11]
entropy = 0.0
samples = 2
value = [0, 2]
entropy = 0.0
samples = 3
value = [3, 0]
X[5] <= -0.316
entropy = 0.985
samples = 14
value = [6, 8]
entropy = 0.0
samples = 3
value = [3, 0]
X[0] <= 0.156
entropy = 1.0
samples = 16
value = [8, 8]
entropy = 0.0
samples = 2
value = [2, 0]
X[1] <= 0.579
entropy = 0.918
samples = 12
value = [4, 8]
entropy = 0.0
samples = 2
value = [2, 0]
X[0] <= 0.41
entropy = 0.985
samples = 14
value = [6, 8]
entropy
samples
value =
X[0] <= 0.664
entropy = 0.991
samples = 9
value = [4, 5]
entropy = 0.0
samples = 3
value = [0, 3]
entropy = 0.0
samples = 2
value = [0, 2]
X[0] <= 0.579
entropy = 1.0
samples = 12
value = [6, 6]
X[4] <= -0.819
entropy = 0.863
samples = 7
value = [2, 5]
entropy = 0.0
samples = 2
value = [2, 0]
entropy = 0.0
samples = 3
value = [3, 0]
X[5] <= 0.741
entropy = 0.918
samples = 9
value = [3, 6]
X[1] <=
entropy =
samples
value =
entropy = 0.0
samples = 2
value = [0, 2]
X[2] <= 1.186
entropy = 0.971
samples = 5
value = [2, 3]
X[5] <= 0.618
entropy = 0.971
samples = 5
value = [3, 2]
entropy = 0.0
samples = 4
value = [0, 4]
X[0] <= 0.113
entropy = 0.918
samples = 3
value = [2, 1]
X[4] <= 1.109
entropy = 0.918
samples = 3
value = [2, 1]
entropy = 0.0
samples = 2
value = [0, 2]
entropy = 0.0
samples = 2
value = [0, 2]
entropy = 0.0
samples = 3
value = [3, 0]
entropy = 0.0
samples = 1
value = [0, 1]
entropy
samples
value =
entropy = 0.0
samples = 2
value = [2, 0]
entropy = 0.0
samples = 1
value = [0, 1]

Results

```
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.res
```

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

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

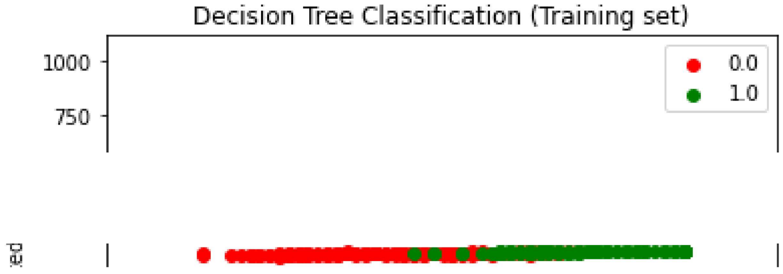
Checking Errors

[ ] ↳ 1 cell hidden

Visualizing

```
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,
                     np.arange(start = X_set[:, 1].min() - 1000
# plt.contourf(X1, X2, classifier.predict(sc.transform(np.array
            #  alpha = 0.75, cmap = ListedColormap(('red', 'gre
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 =
plt.title('Decision Tree Classification (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

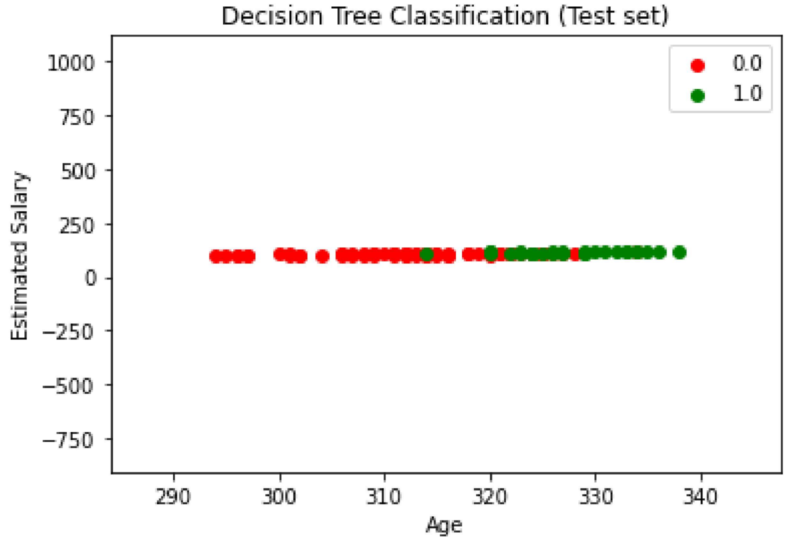
```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoide
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoide
```



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,
                     np.arange(start = X_set[:, 1].min() - 1000
# plt.contourf(X1, X2, classifier.predict(sc.transform(np.array
#              alpha = 0.75, cmap = ListedColormap(('red', 'gre
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 =
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 avoide
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoide
```


Decision Tree Classification (Test set)
● 0.0
● 1.0
Estimated Salary
Age