Single Logistic Regression (Binary classification)

import required package

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

read data from source and describing

In [2]:
```
df = pd.read_csv('social_network_ads.csv')
df.head()
```

Out[2]:

	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 [3]:
```
print(df.columns)
```

```
Index(['User ID', 'Gender', 'Age', 'EstimatedSalary', 'Purchased'], dtype='object')
```

In [4]:
```
print(df.describe())
```

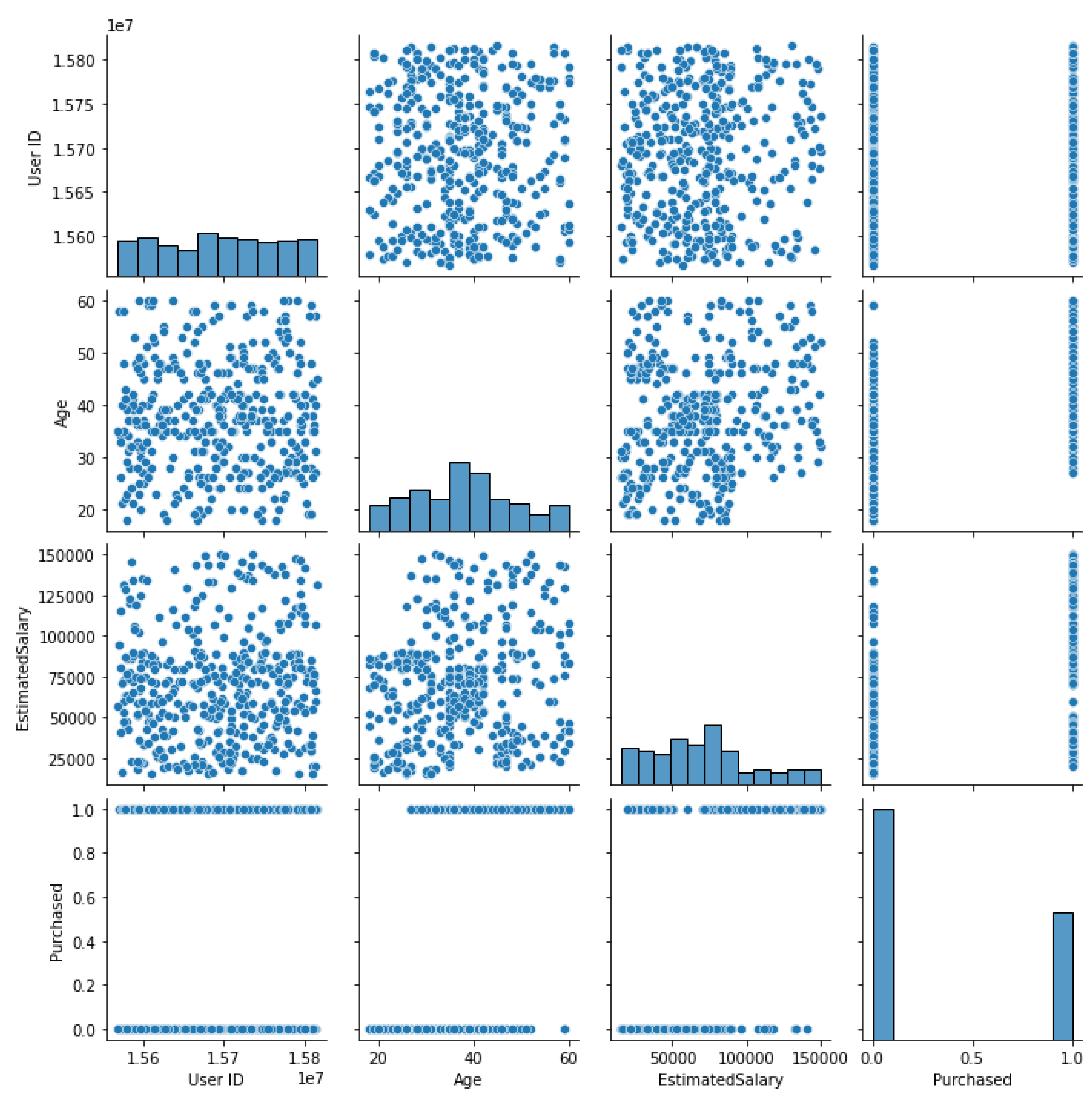
```
            User ID         Age  EstimatedSalary   Purchased
count  4.000000e+02  400.000000       400.000000  400.000000
mean   1.569154e+07   37.655000     69742.500000    0.357500
std    7.165832e+04   10.482877     34096.960282    0.479864
min    1.556669e+07   18.000000     15000.000000    0.000000
25%    1.562676e+07   29.750000     43000.000000    0.000000
50%    1.569434e+07   37.000000     70000.000000    0.000000
75%    1.575036e+07   46.000000     88000.000000    1.000000
max    1.581524e+07   60.000000    150000.000000    1.000000
```

In [5]:
```
## chec for data types
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   User ID          400 non-null    int64 
 1   Gender           400 non-null    object
 2   Age              400 non-null    int64 
 3   EstimatedSalary  400 non-null    int64 
 4   Purchased        400 non-null    int64 
dtypes: int64(4), object(1)
memory usage: 15.8+ KB
None
```

In [6]:
```
sns.pairplot(df)
```

Out[6]: <seaborn.axisgrid.PairGrid at 0x1a09d44d160>



check the relation between variables

In [7]:
```
corr = df.corr()
corr.style.background_gradient(cmap = 'Greens_r')
```

Out[7]:

	User ID	Age	EstimatedSalary	Purchased
User ID	1.000000	-0.000721	0.071097	0.007120
Age	-0.000721	1.000000	0.155238	0.622454
EstimatedSalary	0.071097	0.155238	1.000000	0.362083
Purchased	0.007120	0.622454	0.362083	1.000000

In [8]:
```
## normalization of data
#convert gender to unique values
vals = df['Gender'].unique()
print(vals)

df.replace(vals, [0, 1], inplace= True)
# print(df['Gender'])
print(df.head())
```

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

In [9]:
```
### check the relation between variables

corr = df.corr()
corr.style.background_gradient(cmap = 'Greens')
```

Out[9]:

	User ID	Gender	Age	EstimatedSalary	Purchased
User ID	1.000000	0.025249	-0.000721	0.071097	0.007120
Gender	0.025249	1.000000	0.073741	0.060435	0.042469
Age	-0.000721	0.073741	1.000000	0.155238	0.622454
EstimatedSalary	0.071097	0.060435	0.155238	1.000000	0.362083
Purchased	0.007120	0.042469	0.622454	0.362083	1.000000

using ordianal encoder

In [10]:
```
# from sklearn.preprocessing import OrdinalEncoder
# encoder = OrdinalEncoder()
# encoder.fit(df[["Gender"]])
# df[['Gender']] = encoder.transform(df[["Gender"]])
# print(df)
```

select input and op variable

In [11]:
```
x = df.drop(['User ID','Gender','Purchased'], axis=1)
y = df['Purchased']
```

split the data

In [12]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state= 123456, train_
```

creating a model

In [13]:
```
from sklearn.linear_model import LogisticRegressionCV
model = LogisticRegressionCV()
```

In [14]:
```
## fit the data
model.fit(x_train, y_train)
```

Out[14]: LogisticRegressionCV()

parameters to fine tune model

In [15]:
```
print(model.get_params())
```

```
{'Cs': 10, 'class_weight': None, 'cv': None, 'dual': False, 'fit_intercept': True, 'in
tercept_scaling': 1.0, 'l1_ratios': None, 'max_iter': 100, 'multi_class': 'auto', 'n_j
obs': None, 'penalty': 'l2', 'random_state': None, 'refit': True, 'scoring': None, 'so
lver': 'lbfgs', 'tol': 0.0001, 'verbose': 0}
```

predict the values for unseen data

In [16]:
```
y_prediction = model.predict(x_test)
#print(y_prediction)
```

evaluation of classification model

In [17]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_prediction)
print(cm)
```

```
[[55  1]
 [10 14]]
```

In [18]:
```
correct = cm[0,0] + cm[1,1]
wrong = cm[1,0] + cm[0,1]
total = correct + wrong
accuracy = correct/total
print(accuracy)
```

```
0.8625
```

In [19]:
```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_prediction))
```

```
0.8625
```

precision value

In [20]:
```
from sklearn.metrics import precision_score
print(precision_score(y_test, y_prediction))
```

```
0.9333333333333333
```

recal value

In [21]:
```
from sklearn.metrics import recall_score
print(recall_score(y_test, y_prediction))
```

```
0.5833333333333334
```

F1 score

In [22]:
```
from sklearn.metrics import f1_score
print(f1_score(y_test, y_prediction))
```

```
0.7179487179487181
```

classification report

In [23]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_prediction))
```

```
              precision    recall  f1-score   support

           0       0.85      0.98      0.91        56
           1       0.93      0.58      0.72        24

    accuracy                           0.86        80
   macro avg       0.89      0.78      0.81        80
weighted avg       0.87      0.86      0.85        80
```

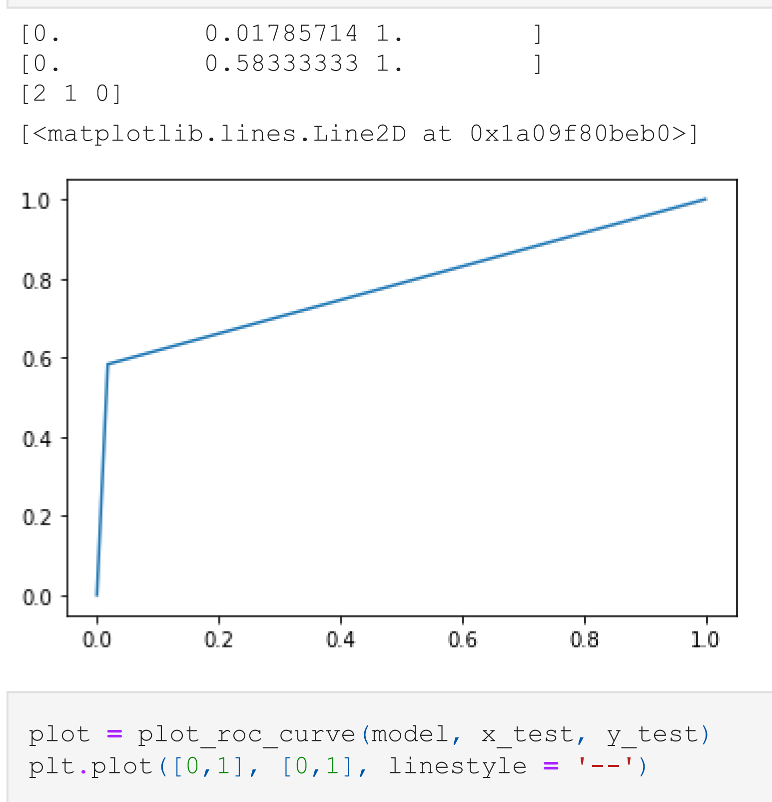
In [24]:
```
from sklearn.metrics import roc_curve, roc_auc_score, plot_roc_curve

print(roc_auc_score(y_test, y_prediction))
```

```
0.7827380952380953
```

In [25]:
```
fpr, tpr, threshold = roc_curve(y_test, y_prediction)
print(fpr)
print(tpr)
print(threshold)
plt.plot(fpr, tpr)
```

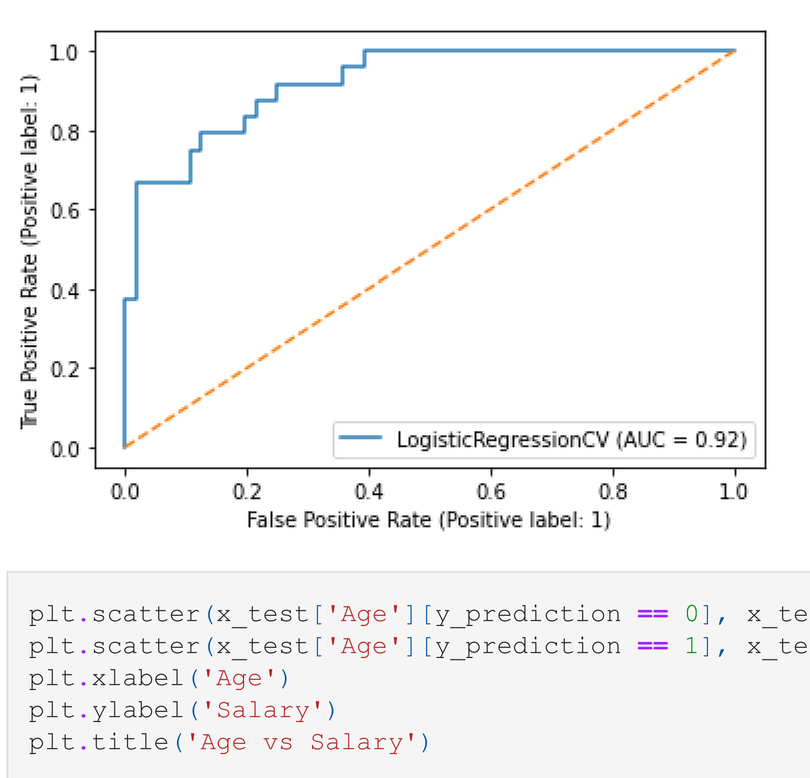
```
[0.         0.01785714 1.        ]
[0.         0.58333333 1.        ]
[2 1 0]
```

Out[25]: [<matplotlib.lines.Line2D at 0x1a09f80beb0>]



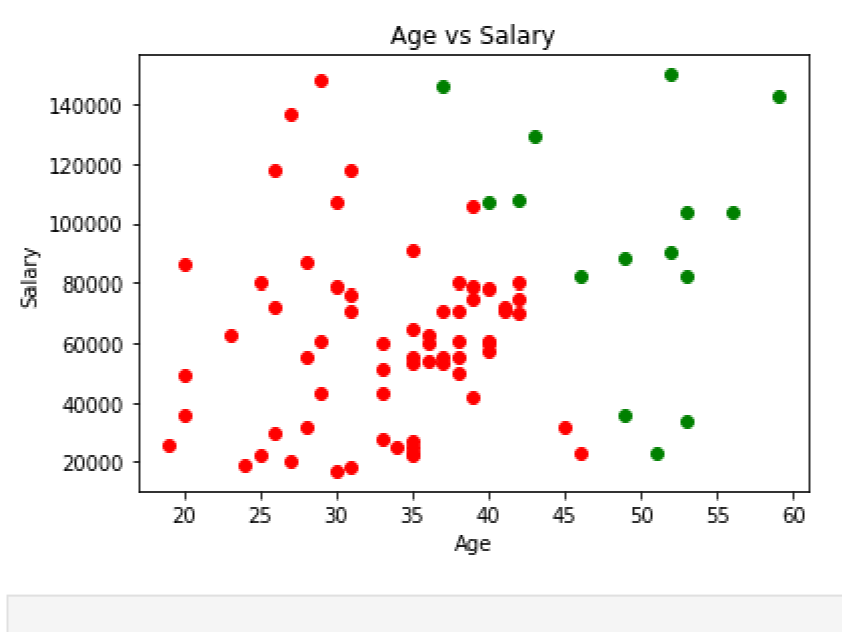
In [26]:
```
plot = plot_roc_curve(model, x_test, y_test)
plt.plot([0,1], [0,1], linestyle = '--')
```

Out[26]: [<matplotlib.lines.Line2D at 0x1a09f871e80>]



In [27]:
```
plt.scatter(x_test['Age'][y_prediction == 0], x_test['EstimatedSalary'][y_prediction =
plt.scatter(x_test['Age'][y_prediction == 1], x_test['EstimatedSalary'][y_prediction =
plt.xlabel('Age')
plt.ylabel('Salary')
plt.title('Age vs Salary')
```

Out[27]: Text(0.5, 1.0, 'Age vs Salary')



In [ ]: