#Import necessary libries

In [2]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import GridSearchCV
```

#Load the dataset

In [3]:
```
data = pd.read_csv("C:\\Users\\HP\\Desktop\\car_data.csv")
data
```

Out[3]:

	User ID	Age	AnnualSalary	Purchased
0	385	35	20000	0
1	681	40	43500	0
2	353	49	74000	0
3	895	40	107500	1
4	661	25	79000	0
...	...	...	...	...
995	863	38	59000	0
996	800	47	23500	0
997	407	28	138500	1
998	299	48	134000	1
999	687	44	73500	0

#Perform data preprocessing

In [4]:
```
x = data.drop('Purchased', axis=1)
x
```

Out[4]:

	User ID	Age	AnnualSalary
0	385	35	20000
1	681	40	43500
2	353	49	74000
3	895	40	107500
4	661	25	79000
...	...	...	...
995	863	38	59000
996	800	47	23500
997	407	28	138500
998	299	48	134000
999	687	44	73500

1000 rows × 3 columns

In [5]:
```
y=data['Purchased']
y
```

Out[5]:
```
0      0
1      0
2      0
3      1
4      0
      ..
995    0
996    0
997    1
998    1
999    0
Name: Purchased, Length: 1000, dtype: int64
```

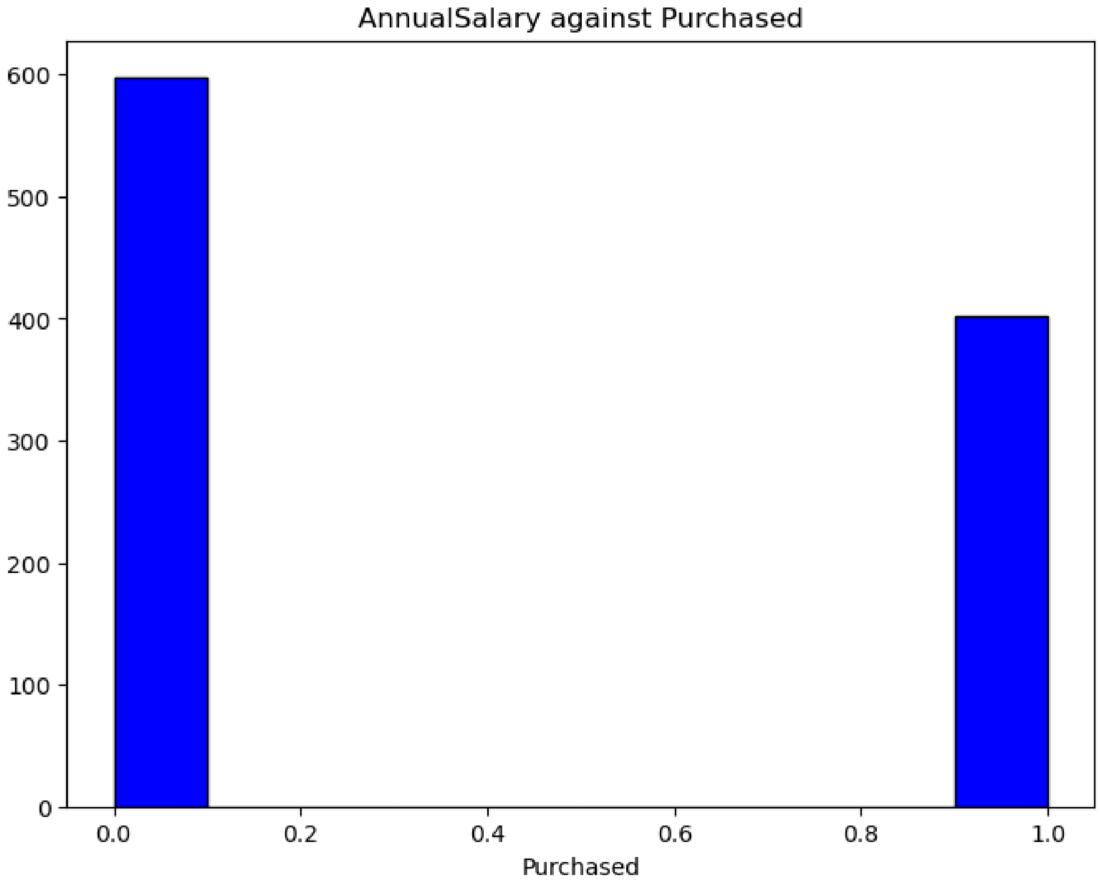
In [6]:

```python
#Check for the missing data
missing_data = data.isnull()

#Count missing values in each column
missing_count = missing_data.sum()

#Display the count of missing values
print("Missing Data")
print(missing_count)
```

```
Missing Data
User ID         0
Age             0
AnnualSalary    0
Purchased       0
dtype: int64
```

In [7]:

```python
#Visualising the graphics

import matplotlib.pyplot as plt
plt.figure(figsize=(8,6))
plt.hist(y, bins=10, color='blue', edgecolor='black')
plt.ylabel('')
plt.xlabel('AnnualSalary')
plt.xlabel('Purchased')
plt.title('AnnualSalary against Purchased')
plt.show()
```



In [8]:

```python
#Splitting the data
#Import the library
from sklearn.model_selection import train_test_split
```

In [9]:

```python
#Split the data into training and testing sets
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_st
```

In [10]:

```
#Standardising the data
#Importing library
from sklearn.preprocessing import StandardScaler

#Initialise the StandardScaler
scaler = StandardScaler()

#Fit the scaler to the training data and transform the training data
x_train_scaled = scaler.fit_transform(x_train)

#Transform the testing data using the same scaler
x_test_scaled = scaler.transform(x_test)
```

In [11]:

```
#Building the model
```

In [12]:

```
from sklearn.linear_model import LogisticRegression

#Initialise the Logistic Regression model
logistic_regression_model = LogisticRegression()
```

In [13]:

```
#Fitting the model on the training data
logistic_regression_model.fit(x_train_scaled,y_train)
```

Out[13]:

```
▾ LogisticRegression

LogisticRegression()
```

In [14]:

```
#Preddicting the target variable for the testing data
y_pred = logistic_regression_model.predict(x_test_scaled)
y_pred
```

Out[14]:

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

In [15]:

```
#Getting coefficient
coefficients = logistic_regression_model.coef_

#Getting intercept
intercept = logistic_regression_model.intercept_

#Displaying coefficient and intercept
print("Cofficients:", coefficients)
print("intercept:", intercept)
```

```
Cofficients: [[-0.01217401  2.2094714   1.17459755]]
intercept: [-0.82747317]
```

In [16]:

```
#Evaluating the model
```

In [17]:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,
```

In [18]:

```
#Checking for uniques values in y_test and y_pred
unique_y_test = np.unique(y_test)
unique_y_pred = np.unique(y_pred)

print("Unique values in y_test:", unique_y_test)
print("Unique values in y_pred:", unique_y_pred)
```

```
Unique values in y_test: [0 1]
Unique values in y_pred: [0 1]
```

In [19]:

```
# compute accuracy
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

Out[19]:

```
0.815
```

In [20]:

```
#Compute precision
precision = precision_score(y_test, y_pred)
precision
```

Out[20]:

```
0.8805970149253731
```

In [21]:

```
#Compute recall
recall = recall_score(y_test,y_pred)
recall
```

Out[21]:

```
0.6704545454545454
```

In [22]:

```
#Convert class labels to binary format
y_test_binary = np.where(y_test ,1,0)
y_pred_binary = np.where(y_pred ,1,0)

#Compute f1-score
f1 = f1_score(y_test_binary, y_pred_binary)

#Print f1-score
print("f1-score:", f1)
```

```
f1-score: 0.7612903225806451
```

In [23]:

```
#Compute predicted probabibilities
y_prob = logistic_regression_model.predict_proba(x_test_scaled)

#Extract the probabibilities for the positive class
y_prob_positive = y_prob[:,1]

#Compute ROC-AUC score
roc_auc = roc_auc_score(y_test_binary,y_prob_positive)

#Print ROC-AUC score
print("ROC-AUC score:", roc_auc)
```

```
ROC-AUC score: 0.8933644480519481
```

In [42]:

```
#Display the evaluation metrics
print("Accuracy:", accuracy)
print("Precision:",precision)
print("Recall:",recall)
print("F1-score:",f1)
print("ROC-AUC score:",roc_auc)
```

```
Accuracy: 0.815
Precision: 0.8805970149253731
Recall: 0.6704545454545454
F1-score: 0.7612903225806451
ROC-AUC score: 0.8933644480519481
```

MODEL OPTIMISATION

In [53]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
```

In [54]:

```
model = LogisticRegression()
model
```

Out[54]:

```
▾ LogisticRegression
LogisticRegression()
```

In [55]:

```
#Define the Grid of hyperparameters
param_grid = {'penalty': ['l1','l2'],
              'C': [0.1, 1],
              'solver': ['liblinear','saga']
    }
```

In [56]:

```
#Initailise GridSearchCV
grid_search = GridSearchCV(model, param_grid, cv=5,scoring='accuracy')
```

In [57]:

```
#Perform grid search to find the best hyperparameters
grid_search.fit(x_train_scaled, y_train)
```

Out[57]:

```
▸          GridSearchCV
▸ estimator: LogisticRegression
     ▸ LogisticRegression
```

In [58]:

```
#Getting the best hyperparameters
best_params = grid_search.best_params_
best_params
```

Out[58]:

```
{'C': 0.1, 'penalty': 'l1', 'solver': 'liblinear'}
```

In [59]:

```
#Initialise logistic regression model with the best hyperparameters
best_logistic_regression_model = LogisticRegression(**best_params)
```

In [60]:

```
#Fit the model on the training data
best_logistic_regression_model.fit(x_train_scaled,y_train)
```

Out[60]:

```
▾                  LogisticRegression
LogisticRegression(C=0.1, penalty='l1', solver='liblinear')
```

#Split the dataset into training and testing sets

In [61]:

```
#Predict the target variables for testing data
y_pred_best = best_logistic_regression_model.predict(x_test_scaled)
y_pred_best
```

Out[61]:

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

In [62]:

```
#Evaluate the model
accuracy_best = accuracy_score(y_test, y_pred_best)
precision_best = precision_score(y_test, y_pred_best)
recall_best = recall_score(y_test_binary, y_pred_binary)
f1_best = f1_score(y_test_binary, y_pred_binary)
y_prob_best = best_logistic_regression_model.predict_proba(x_test_scaled)
y_prob_postive_best = y_prob_best[:, 1]
roc_auc_best = roc_auc_score(y_test_binary, y_prob_postive_best)

#Display the evaluation metrics for optimised model
print("Optimized Model Evaluation Metrics:")
print("Accuracy:", accuracy_best)
print("Precision:", precision_best)
print("Recall:", recall_best)
print("F1-score:", f1_best)
print("ROC-AUC Score:", roc_auc_best)
```

```
Optimized Model Evaluation Metrics:
Accuracy: 0.825
Precision: 0.8840579710144928
Recall: 0.6704545454545454
F1-score: 0.7612903225806451
ROC-AUC Score: 0.8947849025974026
```