

## ✓ Introduction to Logistic Regression

When data scientists may come across a new classification problem, the first algorithm that may come across their mind is Logistic Regression. It is a supervised learning classification algorithm which is used to predict observations to a discrete set of classes. Practically, it is used to classify observations into different categories. Hence, its output is discrete in nature. Logistic Regression is also called Logit Regression. It is one of the most simple, straightforward and versatile classification algorithms which is used to solve classification problems.

Logistic Regression is commonly used to estimate the probability that an instance belongs to a particular class. If the estimated probability that an instance is greater than 50%, then the model predicts that the instance belongs to that class 1, or else it predicts that it does not. This makes it a binary classifier. In this notebook we will look at the theory behind Logistic Regression and use it to indicating whether or not a particular internet user clicked on an Advertisement. We will try to create a model that will predict whether or not they will click on an ad based off the features of that user.

## ✓ Import Necessary Libraries

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

%matplotlib inline
sns.set_style('whitegrid')
plt.style.use("fivethirtyeight")
```

## ✓ Import the dataset

```
data = pd.read_csv("/content/advertising.csv")
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 10 columns):
 #   Column                                Non-Null Count  Dtype
---  -
 0   Daily Time Spent on Site              1000 non-null   float64
 1   Age                                    1000 non-null   int64
 2   Area Income                           1000 non-null   float64
 3   Daily Internet Usage                  1000 non-null   float64
 4   Ad Topic Line                         1000 non-null   object
 5   City                                  1000 non-null   object
 6   Male                                  1000 non-null   int64
 7   Country                              1000 non-null   object
 8   Timestamp                            1000 non-null   object
 9   Clicked on Ad                         1000 non-null   int64
dtypes: float64(3), int64(3), object(4)
memory usage: 78.2+ KB
```

This data set contains the following features:

'Daily Time Spent on Site': consumer time on site in minutes 'Age': customer age in years 'Area Income': Avg. Income of geographical area of consumer 'Daily Internet Usage': Avg. minutes a day consumer is on the internet 'Ad Topic Line': Headline of the advertisement 'City': City of consumer 'Male': Whether or not consumer was male 'Country': Country of consumer 'Timestamp': Time at which consumer clicked on Ad or closed window 'Clicked on Ad': 0 or 1 indicated clicking on Ad

```
data.describe()
```



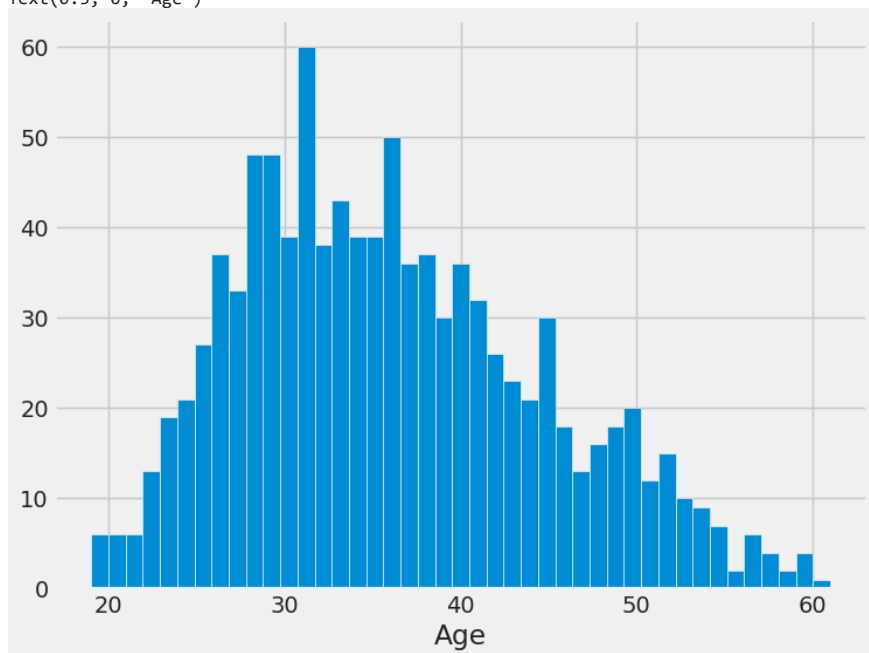
	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Male	Clicked on Ad
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	65.000200	36.009000	55000.000080	180.000100	0.481000	0.500000
std	15.853615	8.785562	13414.634022	43.902339	0.499889	0.500250
min	32.600000	19.000000	13996.500000	104.780000	0.000000	0.000000
25%	51.360000	29.000000	47031.802500	138.830000	0.000000	0.000000
50%	68.215000	35.000000	57012.300000	183.130000	0.000000	0.500000
75%	78.547500	42.000000	65470.635000	218.792500	1.000000	1.000000

## ▼ Data Visualization


```
plt.figure(figsize=(8, 6))
data.Age.hist(bins=data.Age.nunique())
plt.xlabel('Age')
```

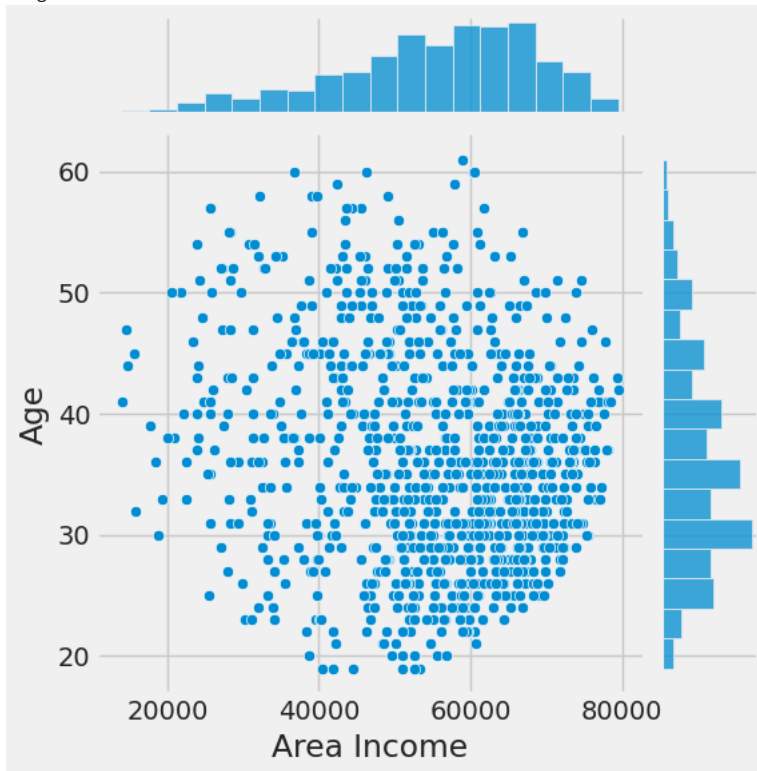


Text(0.5, 0, 'Age')




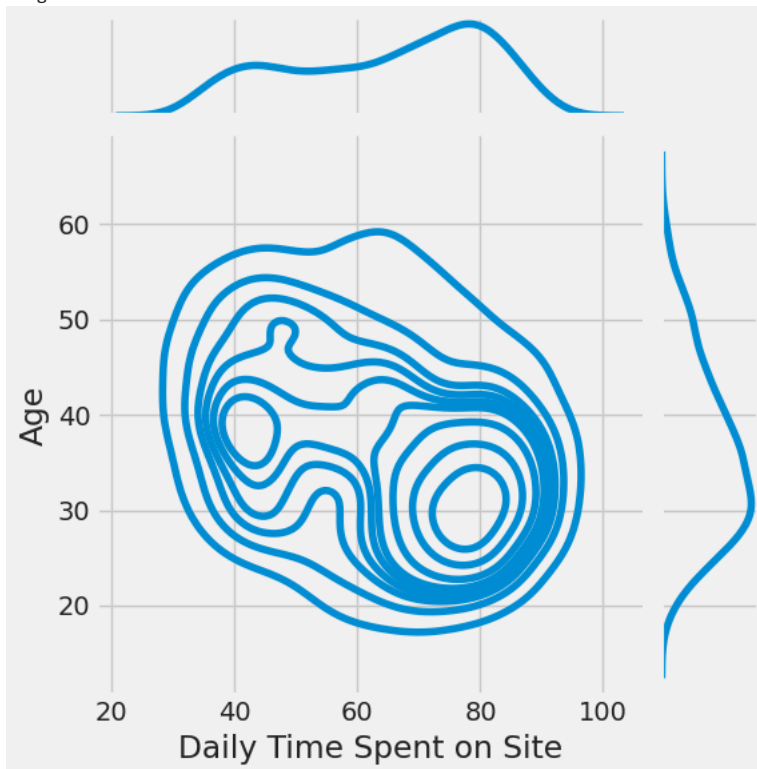
```
plt.figure(figsize=(8, 6))
sns.jointplot(x=data["Area Income"], y=data.Age)
```

 <seaborn.axisgrid.JointGrid at 0x7bb2abf7af50>  
<Figure size 800x600 with 0 Axes>



```
plt.figure(figsize=(8, 6))
sns.jointplot(x=data["Daily Time Spent on Site"], y=data.Age, kind='kde')
```

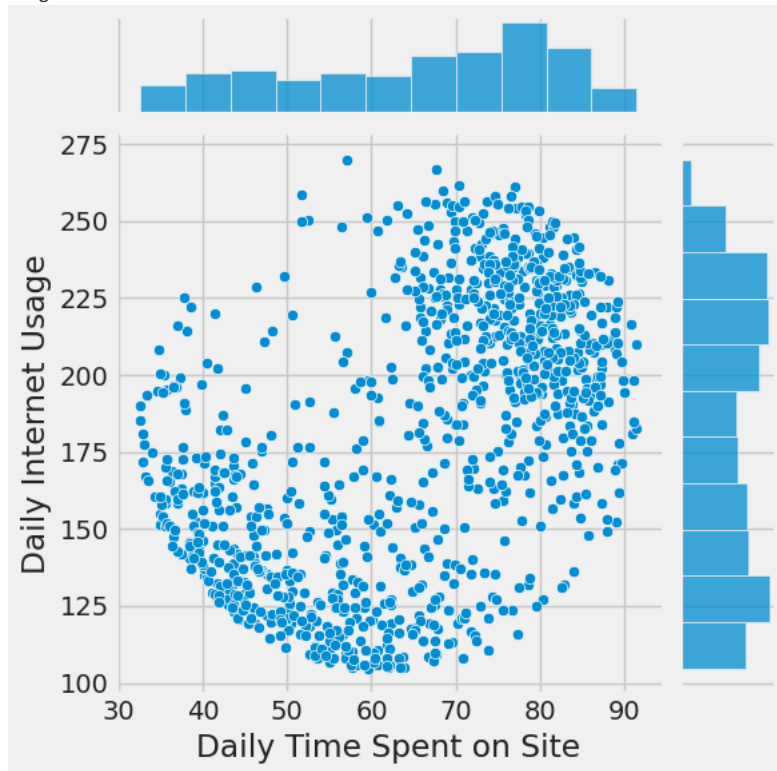
 <seaborn.axisgrid.JointGrid at 0x7bb2a9c5b2e0>  
<Figure size 800x600 with 0 Axes>



```
plt.figure(figsize=(8, 6))
sns.jointplot(x=data["Daily Time Spent on Site"], y=data["Daily Internet Usage"])
```



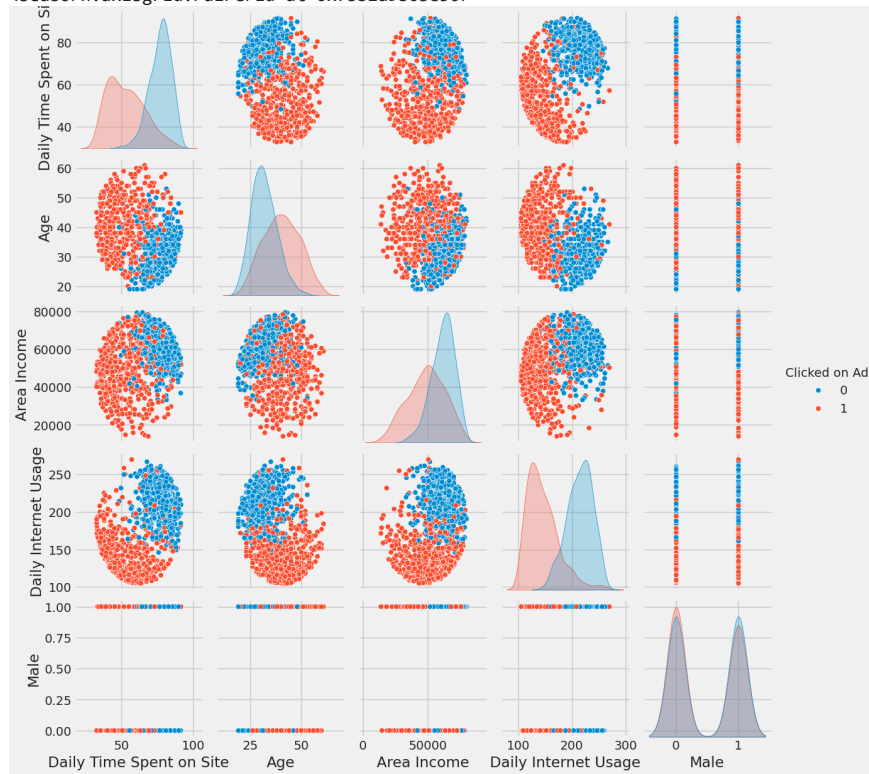
<seaborn.axisgrid.JointGrid at 0x7bb2a51ba470>  
<Figure size 800x600 with 0 Axes>



```
sns.pairplot(data, hue='Clicked on Ad')
```



<seaborn.axisgrid.PairGrid at 0x7bb2a9b03e50>



```
data['Clicked on Ad'].value_counts()
```



Clicked on Ad

0 500

1 500

Name: count, dtype: int64

```
import pandas as pd
```

```
# Check for non-numerical values in each column
```

```
for col in data.columns:
```

```
    non_numeric_values = data[col].apply(lambda x: not pd.to_numeric(x, errors='ignore'))
```

```
    if non_numeric_values.any():
```

```
        print(f"Column '{col}' contains non-numerical values:")
```

```
        print(data[col][non_numeric_values])
```

```
Column 'Male' contains non-numerical values:
```

```
0      0
```

```
2      0
```

```
4      0
```

```
6      0
```

```
10     0
```

```
..
```

```
989    0
```

```
992    0
```

```
994    0
```

```
998    0
```

```
999    0
```

```
Name: Male, Length: 519, dtype: int64
```

```
Column 'Clicked on Ad' contains non-numerical values:
```

```
0      0
```

```
1      0
```

```
2      0
```

```
3      0
```

```
4      0
```

```
..
```

```
986    0
```

```
988    0
```

```
989    0
```

```
993    0
```

```
998    0
```

```
Name: Clicked on Ad, Length: 500, dtype: int64
```

```
data = data.dropna(subset=['Daily Time Spent on Site', 'Daily Internet Usage'])
```

```
data['Daily Time Spent on Site'].fillna(data['Daily Time Spent on Site'].mean(), inplace=True)
```

```
data['Daily Internet Usage'].fillna(data['Daily Internet Usage'].median(), inplace=True)
```

```
x = np.linspace(-6, 6, num=1000)
```

```
plt.figure(figsize=(10, 6))
```

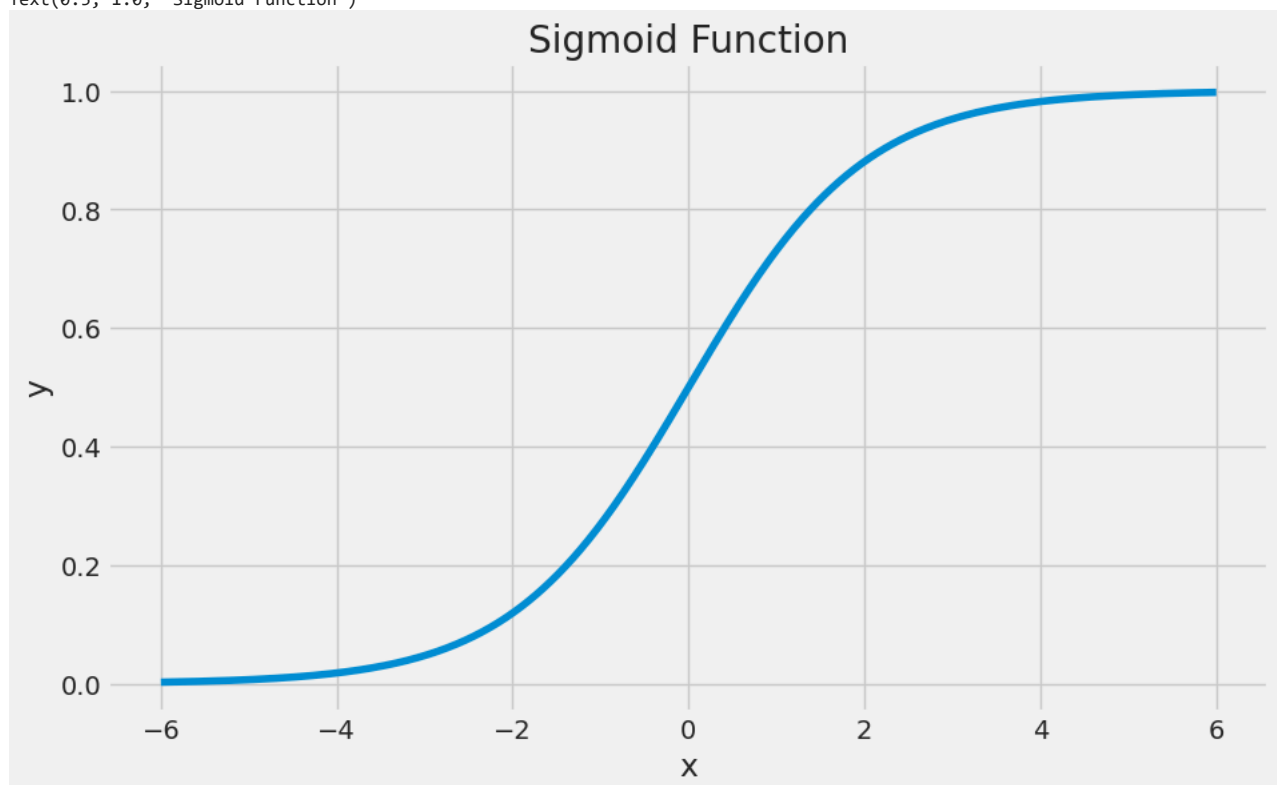
```
plt.plot(x, (1 / (1 + np.exp(-x))))
```

```
plt.xlabel("x")
```

```
plt.ylabel("y")
```

```
plt.title("Sigmoid Function")
```

```
Text(0.5, 1.0, 'Sigmoid Function')
```



## ✓ Plotting the confusion matrix for both train and test set

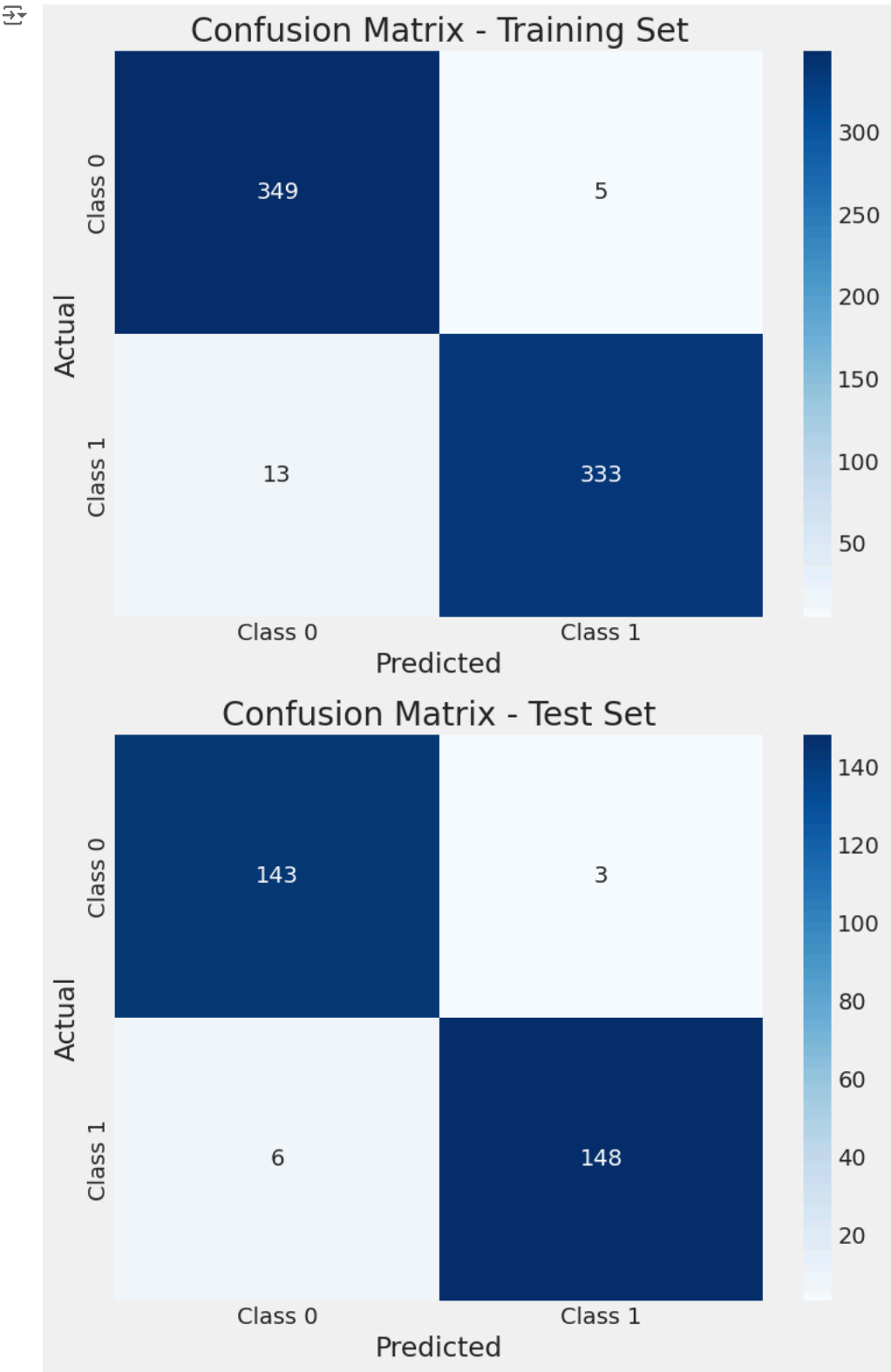
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(y_true, y_pred, title='Confusion Matrix'):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.title(title)
    plt.show()

# Assuming the LogisticRegression model is trained and stored in the variable lr_clf
y_train_pred = lr_clf.predict(X_train)
y_test_pred = lr_clf.predict(X_test)

# Plot confusion matrix for training set
plot_confusion_matrix(y_train, y_train_pred, title='Confusion Matrix - Training Set')

# Plot confusion matrix for test set
plot_confusion_matrix(y_test, y_test_pred, title='Confusion Matrix - Test Set')
```





```

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

def print_score(clf, X_train, y_train, X_test, y_test, train=True):
    if train:
        pred = clf.predict(X_train)
        clf_report = pd.DataFrame(classification_report(y_train, pred, output_dict=True))
        print("Train Result:\n=====")
        print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
        print("_____")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_____")
        print(f"Confusion Matrix: \n {confusion_matrix(y_train, pred)}\n")

    elif train==False:
        pred = clf.predict(X_test)
        clf_report = pd.DataFrame(classification_report(y_test, pred, output_dict=True))
        print("Test Result:\n=====")
        print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
        print("_____")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_____")
        print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n")

```

It defines a function `print_score` that evaluates the performance of a trained classifier (`clf`) on both the training set (`X_train, y_train`) and the test set (`X_test, y_test`). It prints detailed metrics including the accuracy score, classification report, and confusion matrix for the advertising dataset.

```

from sklearn.preprocessing import StandardScaler, MinMaxScaler, OrdinalEncoder
from sklearn.compose import make_column_transformer
from sklearn.model_selection import train_test_split

X = data.drop(['Timestamp', 'Clicked on Ad', 'Ad Topic Line', 'Country', 'City'], axis=1)
y = data['Clicked on Ad']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# cat_columns = []
num_columns = ['Daily Time Spent on Site', 'Age', 'Area Income', 'Daily Internet Usage', 'Male']

ct = make_column_transformer(
    (MinMaxScaler(), num_columns),
    (StandardScaler(), num_columns),
    remainder='passthrough'
)

X_train = ct.fit_transform(X_train)
X_test = ct.transform(X_test)

```

This prepares data for training and testing a machine learning model by performing several preprocessing steps

## ✓ Implimenting Logistic Regression in Scikit-Learn

```

from sklearn.linear_model import LogisticRegression

lr_clf = LogisticRegression(solver='liblinear')
lr_clf.fit(X_train, y_train)

print_score(lr_clf, X_train, y_train, X_test, y_test, train=True)
print_score(lr_clf, X_train, y_train, X_test, y_test, train=False)

```

```

→ Train Result:
=====
Accuracy Score: 97.43%

_____
CLASSIFICATION REPORT:
_____

```

	0	1	accuracy	macro avg	weighted avg
precision	0.964088	0.985207	0.974286	0.974648	0.974527
recall	0.985876	0.962428	0.974286	0.974152	0.974286
f1-score	0.974860	0.973684	0.974286	0.974272	0.974279
support	354.000000	346.000000	0.974286	700.000000	700.000000

```

_____
Confusion Matrix:
[[349  5]
 [ 13 333]]

```

Test Result:

=====

Accuracy Score: 97.00%

CLASSIFICATION REPORT:

	0	1	accuracy	macro avg	weighted avg
precision	0.959732	0.980132	0.97	0.969932	0.970204
recall	0.979452	0.961039	0.97	0.970246	0.970000
f1-score	0.969492	0.970492	0.97	0.969992	0.970005
support	146.000000	154.000000	0.97	300.000000	300.000000

Confusion Matrix:

```
[[143  3]
 [ 6 148]]
```

The results of the classifier on both the training and test datasets indicate strong performance with high accuracy and good balance between precision, recall, and F1-scores across both classes (0 and 1).

Training Results Accuracy: The classifier achieved an accuracy of 97.43%, meaning it correctly classified 97.43% of the training samples.

Precision, Recall, F1-Score: Class 0: Precision is 96.41%, recall is 98.59%, and F1-score is 97.49%. Class 1: Precision is 98.52%, recall is 96.24%, and F1-score is 97.37%. Confusion Matrix: Out of 354 samples of class 0, 349 were correctly classified, and 5 were misclassified. Out of 346 samples of class 1, 333 were correctly classified, and 13 were misclassified. Test Results Accuracy: The classifier achieved an accuracy of 97.00% on the test set, indicating consistent performance and suggesting that the model generalizes well to unseen data. Precision, Recall, F1-Score: Class 0: Precision is 95.97%, recall is 97.95%, and F1-score is 96.95%. Class 1: Precision is 98.01%, recall is 96.10%, and F1-score is 97.05%. Confusion Matrix: Out of 146 samples of class 0, 143 were correctly classified, and 3 were misclassified. Out of 154 samples of class 1, 148 were correctly classified, and 6 were misclassified.

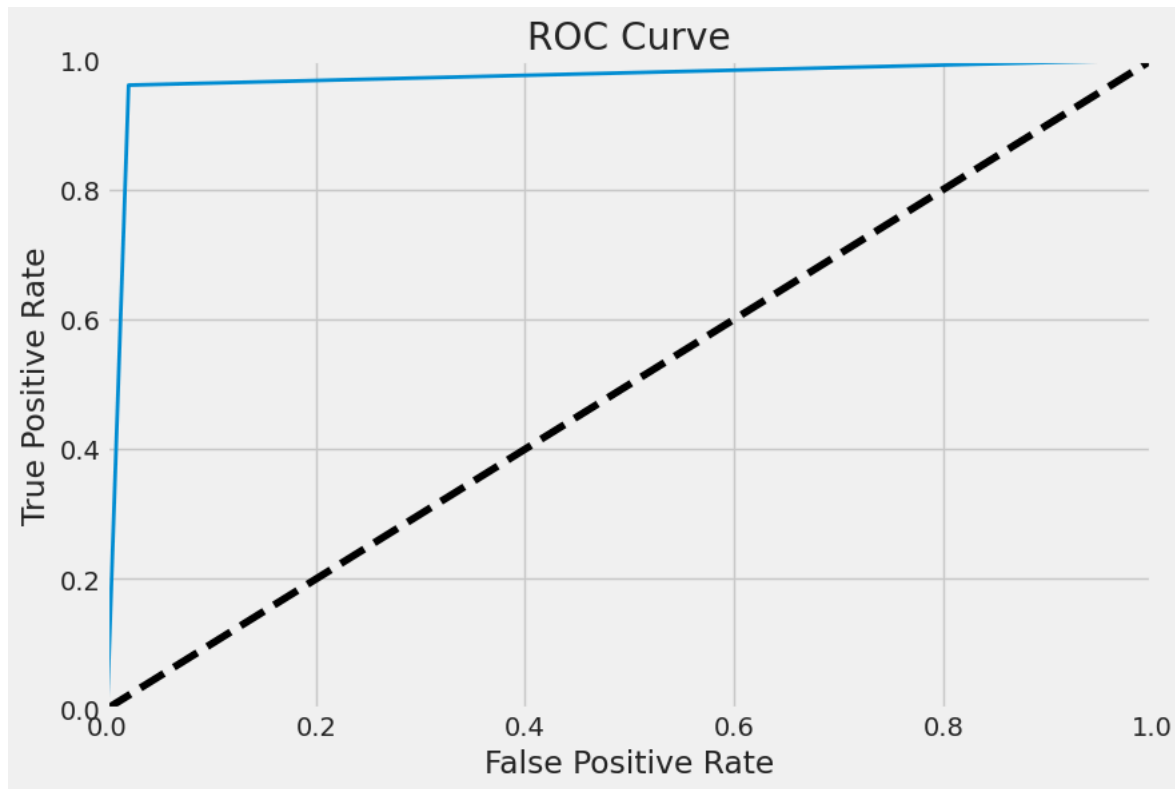
Conclusion : The classifier performs exceptionally well on both the training and test datasets, maintaining high accuracy and balanced precision, recall, and F1-scores for both classes. The minimal difference in performance between the training and test sets suggests that the model is not overfitting and is capable of generalizing effectively to new data.

## ✓ The Receiver Operating Characteristics (ROC) Curve

```
from sklearn.metrics import roc_curve
```

```
def plot_roc_curve(fpr, tpr, label=None):
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], "k--")
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
```

```
fpr, tpr, thresholds = roc_curve(y_test, lr_clf.predict(X_test))
plt.figure(figsize=(9,6));
plot_roc_curve(fpr, tpr)
plt.show();
```



```
from sklearn.metrics import roc_auc_score  
roc_auc_score(y_test, lr_clf.predict(X_test))
```



```
0.9702455079167409
```

Use ROC curve whenever the negative class is rare or when you care more about the false negatives than the false positives

In the example above, the ROC curve seemed to suggest that the classifier is good.

ROC-AUC Score: The ROC-AUC score for the test set is 0.9702.

A ROC-AUC score of 0.9702 is very high, indicating that the classifier has excellent ability to distinguish between the positive class (clicked on ad) and the negative class (not clicked on ad). This score means that there is a 97.02% chance that the classifier will correctly distinguish between a randomly chosen positive instance and a randomly chosen negative instance.

Conclusion: The ROC-AUC score of 0.9702 reinforces the conclusion that the classifier is performing exceptionally well on the test set, demonstrating its robust discriminative power and ability to generalize effectively to unseen data. This metric is particularly useful in situations where the classes are imbalanced, as it considers the performance across all possible thresholds, giving a more comprehensive evaluation of the model's performance.