Logistic Regression

Logistic regression is a statistical method used for binary classification problems, where the outcome is one of two possible values. It's a type of regression analysis used to predict the probability of a certain event occurring. The goal of logistic regression is to model the probability of a binary outcome (0 or 1, yes or no, true or false) as a function of one or more predictor variables

Online Shopper Revenue Prediction,

The primary objective of this project is to predict whether a given online shopper will generate revenue or not, using a logistic regression model. The outcome is binary, with the prediction being either "yes" (the shopper generates revenue) or "no" (the shopper does not generate revenue). This project aims to utilize various features from the data to build a predictive model that can assist in understanding and predicting online shopper behavior. This data set contains the following:

Administrative: Number of pages viewed in the "Administrati"category.

Administrativeve_Duration: Time spent on administrative pages.

Informational: Number of pages viewed in the "Informational" category.

Informational_Duration: Time spent on informational pages.

ProductRelated: Number of pages viewed related to products.

ProductRelated_Duration: Time spent on product-related pages.

BounceRates: Percentage of single-page sessions.

ExitRates: Percentage of sessions where the visitor left from a specific page.

PageValues: Average value assigned to a page.

SpecialDay: Proximity to a special day (like a holiday).

Month: The month when the visit occurred.

OperatingSystems: Type of operating system used.

Browser: Browser used by the visitor.

Region: Geographic region of the visitor.

TrafficType: Type of traffic source.

VisitorType: Whether the visitor is new or returning.

Weekend: Whether the visit occurred on a weekend.

Revenue: Binary variable indicating whether the visitor made a purchasee for no purchase).

Data Preprocessing

To prepare the data for modeling, several preprocessing steps were conducted

Import Libraries

The dataset was loaded into a pandas DataFrame for analysis and processing

In [1]:
```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score
```

Load dataset

In [2]:
```
data=pd.read_excel('C:/Users/arifa/Downloads/online_shoppers_intention.xlsx')
data
```

Out[2]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	Pro
0	0	0.0	0	0.0	
1	0	0.0	0	0.0	
2	0	0.0	0	0.0	
3	0	0.0	0	0.0	
4	0	0.0	0	0.0	
...	...	...	...	...	
12325	3	145.0	0	0.0	
12326	0	0.0	0	0.0	
12327	0	0.0	0	0.0	
12328	4	75.0	0	0.0	
12329	0	0.0	0	0.0	

12330 rows × 18 columns

Define features and target variable

The target variable is 'Revenue', indicating whether a purchase was made. The rest of the columns are features used to predict the target.

In [3]:
```
X = data.drop('Revenue', axis=1)
y = data['Revenue'].astype(int)
```

Define categorical and numerical features

In [4]:
```
categorical_features = ['Month', 'VisitorType', 'Weekend']
numerical_features = ['Administrative', 'Administrative_Duration', 'Informational',
                      'Informational_Duration', 'ProductRelated', 'ProductRelated_D
                      'BounceRates', 'ExitRates', 'PageValues', 'SpecialDay']
```

Handling Categorical Variables:

The categorical variables ('Month', 'VisitorType', and 'Weekend') were encoded using OneHotEncoder. This converts the categorical data into a format that can be provided to ML algorithms to do a better job in prediction.

Encoding-

Encoding is the process of converting categorical data into numerical form so that machine learning algorithms can process it. Here are some common encoding techniques:

1. Label Encoding

• Assigns each category a unique integer value.

• Example: For categories ['Red', 'Green', 'Blue'], the encoding might be {'Red': 0, 'Green': 1, 'Blue': 2}

• Suitable when the categorical feature has an ordinal relationship (i.e., there's a meaningful order).

2. One-Hot Encoding

• Creates binary columns for each category, where each column represents one category.

• Example: For categories ['Red', 'Green', 'Blue'], the one-hot encoding would create three columns: Red, Green, Blue, where only one column is 1 and the others are 0 for each instance.

• Ideal for categorical features without an ordinal relationship

You can choose one of the encoding methods based on your data's nature and apply it consistently across all categorical features (e.g., month, visitor type, and boolean variables like weekend and revenue).n . .

In [5]:
```
from sklearn.preprocessing import OneHotEncoder

# Define the list of categorical features
categorical_features = ["Month", "VisitorType", "Weekend"]

# Initialize the OneHotEncoder
encoder = OneHotEncoder(sparse_output=False, drop='first')  # Adjust this line base

# Apply the OneHotEncoder to the categorical features
X_cat = encoder.fit_transform(X[categorical_features])

# Convert the encoded features back into a DataFrame
X_cat_df = pd.DataFrame(X_cat, columns=encoder.get_feature_names_out(categorical_fe

# Drop the original categorical columns from the DataFrame
X = X.drop(categorical_features, axis=1)

# Concatenate the encoded columns back to the original DataFrame
X = pd.concat([X, X_cat_df], axis=1)

print(X.head())
```

```
   Administrative  Administrative_Duration  Informational  \
0               0                      0.0              0
1               0                      0.0              0
2               0                      0.0              0
3               0                      0.0              0
4               0                      0.0              0

   Informational_Duration  ProductRelated  ProductRelated_Duration  \
0                     0.0               1                 0.000000
1                     0.0               2                64.000000
2                     0.0               1                 0.000000
3                     0.0               2                 2.666667
4                     0.0              10               627.500000

   BounceRates  ExitRates  PageValues  SpecialDay  ...  Month_Jul  Month_June  \
0         0.20       0.20         0.0         0.0  ...        0.0         0.0
1         0.00       0.10         0.0         0.0  ...        0.0         0.0
2         0.20       0.20         0.0         0.0  ...        0.0         0.0
3         0.05       0.14         0.0         0.0  ...        0.0         0.0
4         0.02       0.05         0.0         0.0  ...        0.0         0.0

   Month_Mar  Month_May  Month_Nov  Month_Oct  Month_Sep  VisitorType_Other  \
0        0.0        0.0        0.0        0.0        0.0                0.0
1        0.0        0.0        0.0        0.0        0.0                0.0
2        0.0        0.0        0.0        0.0        0.0                0.0
3        0.0        0.0        0.0        0.0        0.0                0.0
4        0.0        0.0        0.0        0.0        0.0                0.0

   VisitorType_Returning_Visitor  Weekend_True
0                            1.0           0.0
1                            1.0           0.0
2                            1.0           0.0
3                            1.0           0.0
4                            1.0           1.0

[5 rows x 26 columns]
```

Standardize numerical features

Numerical features were standardized using StandardScaler to ensure that each feature contributes equally to the model, preventing features with larger ranges from dominating the model.

It is particularly useful when the data follows a normal distribution or when working with algorithms that assume normally distributed data.

In [6]:
```
scaler = StandardScaler()
X_num = scaler.fit_transform(X[numerical_features])
X_num_df = pd.DataFrame(X_num, columns=numerical_features)
X = X.drop(numerical_features, axis=1)
X = pd.concat([X, X_num_df], axis=1)
```

Split data into training and testing sets

The dataset was split into training and testing sets using an 80-20 split. This allows the model to be trained on one portion of the data and tested on another to evaluate its performance.

In [8]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

Initialize and train the model

Model Initialization:A Logistic Regression model was chosen for its efficiency and interpretability in binary classification tasks.

Model Training:The model was trained on the training data using the fit method, which adjusts the model's parameters to minimize the prediction error

In [9]:
```
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
```

Out[9]:
```
LogisticRegression(max_iter=1000)
```

Make predictions

In [10]:
```
y_pred = model.predict(X_test)
```

Evaluate the model

Confusion Matrix:

A confusion matrix was generated to evaluate the model's performance. It showed the number of correct and incorrect predictions made by the model.

In [11]:
```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix:
[[2009   46]
 [ 266  145]]
```

2009 (True Negatives - TN):

The model correctly predicted that 2009 visitors did not generate revenue (actual class 0, predicted class 0).

46 (False Positives - FP):

The model incorrectly predicted that 46 visitors generated revenue when they actually did not (actual class 0, predicted class 1).

266 (False Negatives - FN):

The model incorrectly predicted that 266 visitors did not generate revenue when they actually did (actual class 1, predicted class 0).

145 (True Positives - TP):

The model correctly predicted that 145 visitors generated revenue (actual class 1, predicted class 1).

Classification Report

The classification report provided detailed metrics such as precision, recall, and F1-score for both classes (revenue generated and not generated). The F1-score, which balances precision and recall, was particularly important for understanding the trade-off between false positives and false negatives.

1.Precision

High precision means that when the model predicts a positive class, it is often correct. (true positive predictions out of all positive prediction.)

2.Recall

High recall means that the model identifies most of the positive instances.(true positive predictions out of all actual positie vinstance).

3.F1-Score

The F1-score is useful when you need a balance between precision and recall, especially when dealing with imbalce dataset.

4.Support

The number of actual occurrences of the class in the dataset.

In [12]:
```
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.98      0.93      2055
           1       0.76      0.35      0.48       411

    accuracy                           0.87      2466
   macro avg       0.82      0.67      0.70      2466
weighted avg       0.86      0.87      0.85      2466
```

The model performs well in predicting class 0 (No Revenue) with high precision and recall.

For class 1 (Revenue), the model has lower recall, indicating it misses a significant number of positive cases.

The overall accuracy of 87% indicates a good overall performance.

Retest if it works , when we add train data, then test it, if it gives the result

In [14]:
```
# Prepare new data
new_data = pd.DataFrame({
    'Administrative': [0],
    'Administrative_Duration': [0.0],
    'Informational': [0],
    'Informational_Duration': [0.0],
    'ProductRelated': [2],
    'ProductRelated_Duration': [64.0],
    'BounceRates': [0.0],
    'ExitRates': [0.1],
    'PageValues': [0.0],
    'SpecialDay': [0.0],
    'Month': ['Feb'],
    'VisitorType': ['Returning_Visitor'],
    'Weekend': [False]
})
```

In [15]:
```
# Preprocess new data
X_new_cat = encoder.transform(new_data[categorical_features])
X_new_cat_df = pd.DataFrame(X_new_cat, columns=encoder.get_feature_names_out(catego
X_new_num = scaler.transform(new_data[numerical_features])
X_new_num_df = pd.DataFrame(X_new_num, columns=numerical_features)
X_new = pd.concat([X_new_cat_df, X_new_num_df], axis=1)
```

In [16]:
```
# Ensure the new data has the same columns as training data
X_new = X_new.reindex(columns=X.columns, fill_value=0)
```

In [17]:
```
# Make predictions on new data
y_new_pred = model.predict(X_new)
```

In [18]:
```
# Print the predictions
print("Predicted values for the new data:")
print(y_new_pred)
```

```
Predicted values for the new data:
[0]
```