ipz4t84ic

April 14, 2025

1 Online Shopper Purchasing Intention - Milestone 6

[ ]:
```
# pip install pycaret
# !pip install -f http://h2o-release.s3.amazonaws.com/h2o/latest_stable_Py.html␣
↪h2o
```

[66]:
```
#Import relevant libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
import statsmodels.api as sm
from sklearn.model_selection import train_test_split, GridSearchCV
from imblearn.over_sampling import SMOTE, ADASYN
from imblearn.combine import SMOTETomek, SMOTEENN
from imblearn.under_sampling import TomekLinks
from sklearn.linear_model import LogisticRegression
from sklearn.dummy import DummyClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,␣
↪GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
import xgboost as xgb
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, roc_auc_score, roc_curve, auc, confusion_matrix,␣
↪classification_report
```

[4]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[5]:
```
file_path = '/content/drive/MyDrive/Colab Notebooks/datamining/
↪online_shoppers_intention.csv'
df = pd.read_csv(file_path)
```

```
#Load data into the dataframe
df = pd.read_csv(file_path)
```

1.0.1 Data Preprocessing and transformation

[6]:
```
#View the top 5 records in the dataframe
df.head()
```

[6]:
```
   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  OperatingSystems  \
0         0.20       0.20         0.0         0.0   Feb                 1
1         0.00       0.10         0.0         0.0   Feb                 2
2         0.20       0.20         0.0         0.0   Feb                 4
3         0.05       0.14         0.0         0.0   Feb                 3
4         0.02       0.05         0.0         0.0   Feb                 3

   Browser  Region  TrafficType        VisitorType  Weekend  Revenue
0        1       1            1  Returning_Visitor    False    False
1        2       1            2  Returning_Visitor    False    False
2        1       9            3  Returning_Visitor    False    False
3        2       2            4  Returning_Visitor    False    False
4        3       1            4  Returning_Visitor     True    False
```

[7]:
```
#Check for the size, number of features, feature type and not-null count of the␣
↪dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
```

```
 0   Administrative           12330 non-null  int64
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object
 11  OperatingSystems         12330 non-null  int64
 12  Browser                  12330 non-null  int64
 13  Region                   12330 non-null  int64
 14  TrafficType              12330 non-null  int64
 15  VisitorType              12330 non-null  object
 16  Weekend                  12330 non-null  bool
 17  Revenue                  12330 non-null  bool
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

From the table above, we can observe the following - There are 12330 records in this dataset with 17 different features - There are 2 categorical columns 'Month' and 'VisitorType' with 'object' data type - All the features have 12330 non-null records indicating that there are no missing values in this dataset

[8]:
```python
#Summary statistics for each feature of the dataframe
cat_col =␣
 ↪['Month','OperatingSystems','Browser','Region','TrafficType','VisitorType','Weekend','Reven
df[cat_col] = df[cat_col].astype('category')
df.describe(include='all').transpose()
```

[8]:
```
                           count unique                top    freq  \
Administrative           12330.0    NaN                NaN     NaN
Administrative_Duration  12330.0    NaN                NaN     NaN
Informational            12330.0    NaN                NaN     NaN
Informational_Duration   12330.0    NaN                NaN     NaN
ProductRelated           12330.0    NaN                NaN     NaN
ProductRelated_Duration  12330.0    NaN                NaN     NaN
BounceRates              12330.0    NaN                NaN     NaN
ExitRates                12330.0    NaN                NaN     NaN
PageValues               12330.0    NaN                NaN     NaN
SpecialDay               12330.0    NaN                NaN     NaN
Month                      12330     10                May    3364
OperatingSystems         12330.0    8.0                2.0  6601.0
Browser                  12330.0   13.0                2.0  7961.0
Region                   12330.0    9.0                1.0  4780.0
TrafficType              12330.0   20.0                2.0  3913.0
```

```
VisitorType                 12330      3  Returning_Visitor   10551
Weekend                     12330      2              False    9462
Revenue                     12330      2              False   10422

                              mean          std  min       25%         50%  \
Administrative            2.315166     3.321784  0.0       0.0         1.0
Administrative_Duration  80.818611   176.779107  0.0       0.0         7.5
Informational             0.503569     1.270156  0.0       0.0         0.0
Informational_Duration   34.472398   140.749294  0.0       0.0         0.0
ProductRelated           31.731468    44.475503  0.0       7.0        18.0
ProductRelated_Duration 1194.74622  1913.669288  0.0  184.1375  598.936905
BounceRates               0.022191     0.048488  0.0       0.0    0.003112
ExitRates                 0.043073     0.048597  0.0  0.014286    0.025156
PageValues                5.889258    18.568437  0.0       0.0         0.0
SpecialDay                0.061427     0.198917  0.0       0.0         0.0
Month                          NaN          NaN  NaN       NaN         NaN
OperatingSystems               NaN          NaN  NaN       NaN         NaN
Browser                        NaN          NaN  NaN       NaN         NaN
Region                         NaN          NaN  NaN       NaN         NaN
TrafficType                    NaN          NaN  NaN       NaN         NaN
VisitorType                    NaN          NaN  NaN       NaN         NaN
Weekend                        NaN          NaN  NaN       NaN         NaN
Revenue                        NaN          NaN  NaN       NaN         NaN

                                 75%          max
Administrative                   4.0         27.0
Administrative_Duration     93.25625      3398.75
Informational                    0.0         24.0
Informational_Duration           0.0     2549.375
ProductRelated                  38.0        705.0
ProductRelated_Duration  1464.157214  63973.52223
BounceRates                 0.016813          0.2
ExitRates                       0.05          0.2
PageValues                       0.0   361.763742
SpecialDay                       0.0          1.0
Month                            NaN          NaN
OperatingSystems                 NaN          NaN
Browser                          NaN          NaN
Region                           NaN          NaN
TrafficType                      NaN          NaN
VisitorType                      NaN          NaN
Weekend                          NaN          NaN
Revenue                          NaN          NaN
```

'Month' and 'VisitorType' are both categorical columns and should be converted into numerical features. We will be using label encoding to convert categories to number as we will be considering the seasonality of the month rather than the order of the month.

We are also converting 'Weekend' and 'Revenue' featurs from boolean to integers to have consistent numeric data when working with algorithms. It will also provide us with better interpretability and slightly better performance with certain models.

[9]:
```python
#Create a new dataframe for transformation
df_transform = df

#Label Encoding for 'Visitor_Type': 'Returning Visitor' → 1, 'New Visitor' → 0
df_transform['VisitorType'] = df['VisitorType'].map({'New_Visitor': 0,␣
 ↪'Returning_Visitor': 1, 'Other': 2})

#Label Encoding for 'Month' in 'MMM' format
month_order = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'June', 'Jul', 'Aug', 'Sep',␣
 ↪'Oct', 'Nov', 'Dec']
month_mapping = {month: idx for idx, month in enumerate(month_order, start=1)}
df_transform['Month'] = df['Month'].map(month_mapping)

#Convert 'Weekend' and 'Revenue' from boolean to integer
df_transform['Weekend'] = df_transform['Weekend'].astype(int)
df_transform['Revenue'] = df_transform['Revenue'].astype(int)
```

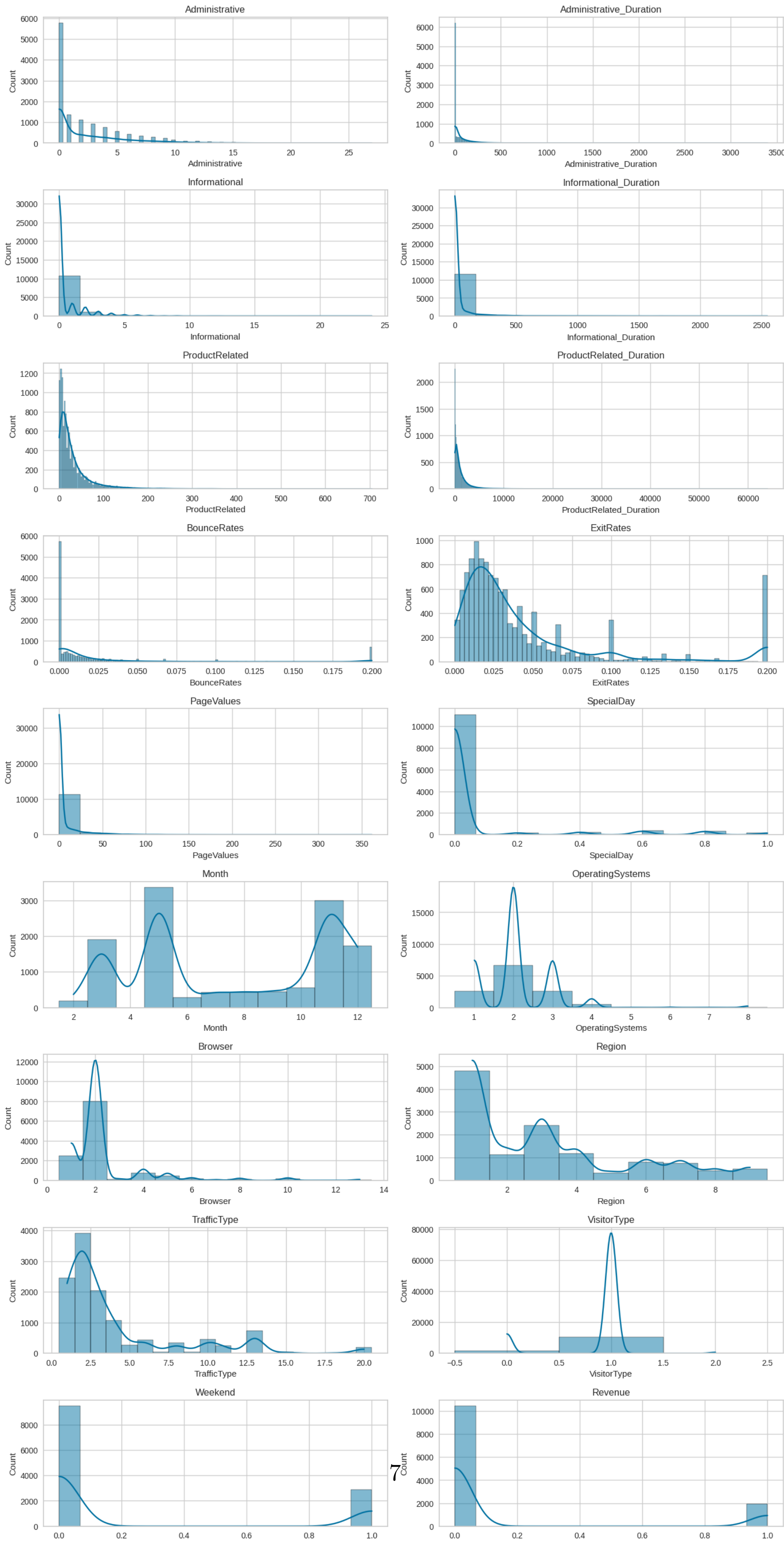
[10]:
```python
df_transform.head()
```

[10]:
```
   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 OperatingSystems  \
0         0.20       0.20         0.0         0.0     2                1
1         0.00       0.10         0.0         0.0     2                2
2         0.20       0.20         0.0         0.0     2                4
3         0.05       0.14         0.0         0.0     2                3
4         0.02       0.05         0.0         0.0     2                3

  Browser  Region  TrafficType  VisitorType  Weekend  Revenue
0       1       1            1            1        0        0
1       2       1            2            1        0        0
```

2	1	9	3	1	0	0
3	2	2	4	1	0	0
4	3	1	4	1	1	0

1.0.2 Explanatry Data Analysis

[11]:
```
plt.figure(figsize=(14, len(df_transform.columns) * 3))

#Loop through each column and create a KDE plot
for idx, feature in enumerate(df_transform.columns, 1):
    plt.subplot(len(df_transform.columns), 2, idx)
    sns.histplot(df_transform[feature], kde=True)
    plt.title(f"{feature}")
plt.tight_layout()
# Save the plot as an image file BEFORE displaying it
plt.savefig("KDE Plot.png", dpi=300, bbox_inches="tight")
plt.show()
```

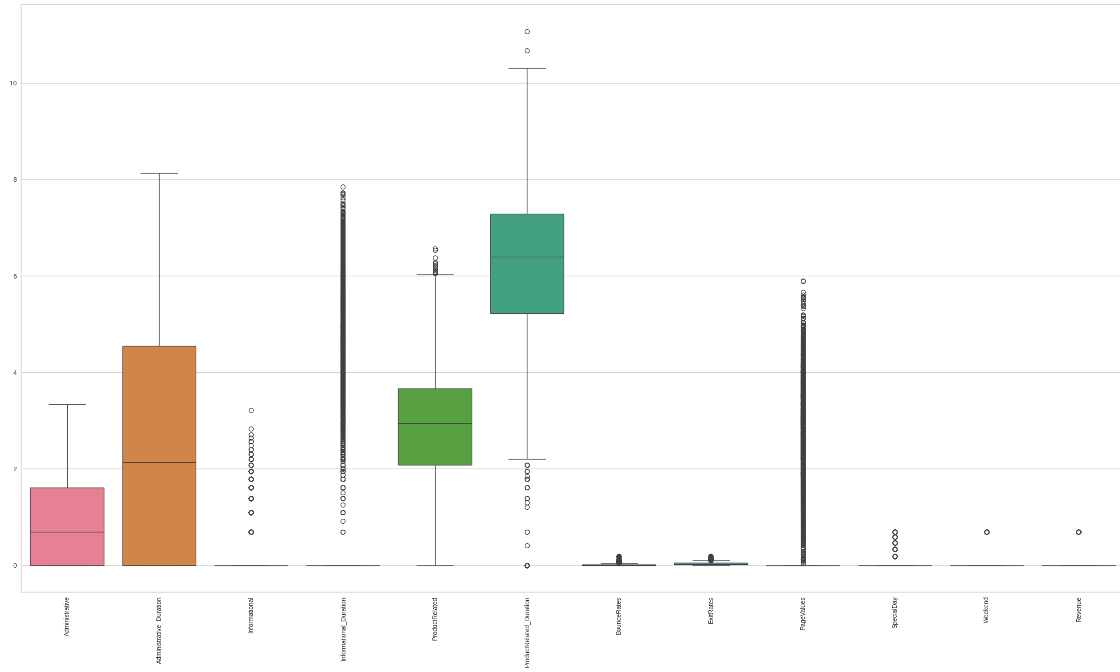

Administrative
Administrative_Duration
Informational
Informational_Duration
ProductRelated
ProductRelated_Duration
BounceRates
ExitRates
PageValues
SpecialDay
Month
OperatingSystems
Browser
Region
TrafficType
VisitorType
Weekend
Revenue
Count

[12]:
```
#Checking the skewness in dataset
num_cols = df_transform.select_dtypes(exclude=['category']).columns
df_transform[num_cols].skew()
```

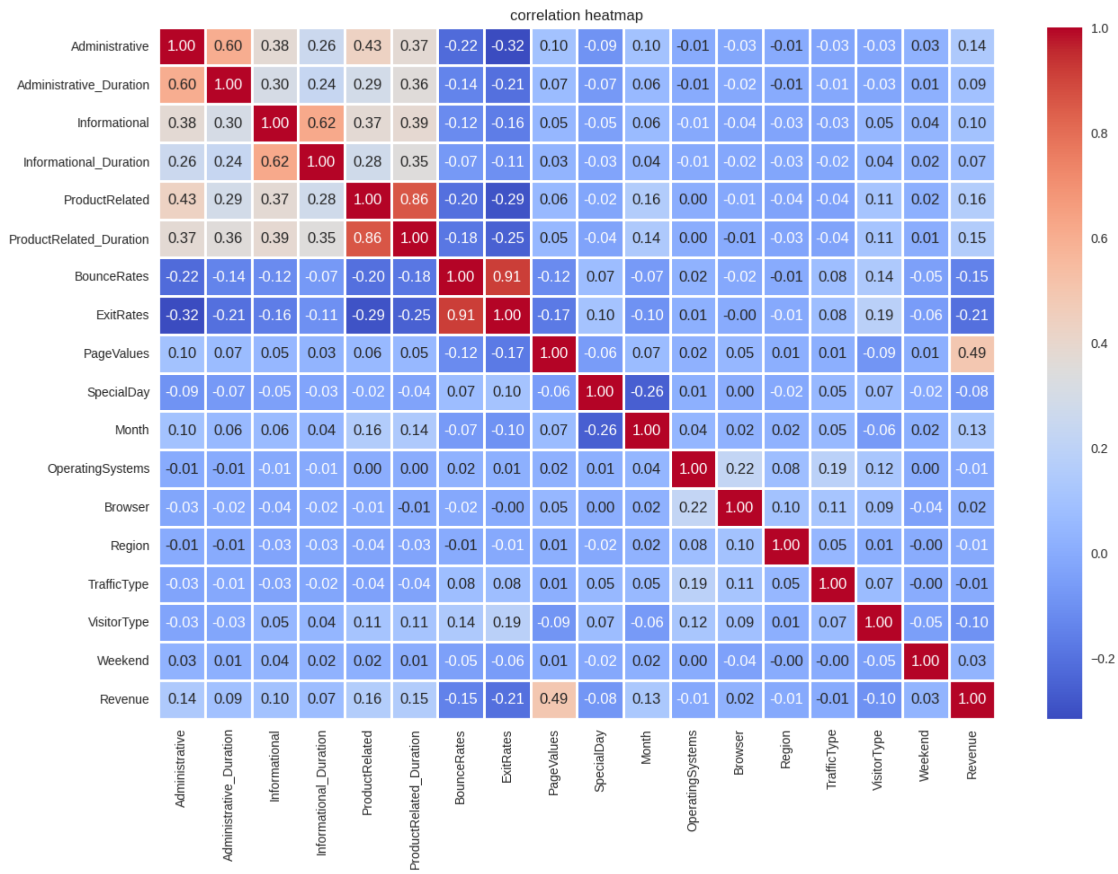
[12]:
```
Administrative              1.960357
Administrative_Duration     5.615719
Informational               4.036464
Informational_Duration      7.579185
ProductRelated              4.341516
ProductRelated_Duration     7.263228
BounceRates                 2.947855
ExitRates                   2.148789
PageValues                  6.382964
SpecialDay                  3.302667
Weekend                     1.265962
Revenue                     1.909509
dtype: float64
```

We can see from the table above, that extreme skewness is observed with multiple columns like Duration, Bounce Rate and Page value. Based on the context of the problem we are trying to solve, the dataset has number of features that are naturally skewed, which might actually improve classification accuracy. For example, Page Value or Bounce Rates may be highly right-skewed because only a small portion of users generate revenue

[13]:
```
#Scaling data to visualize outliers
symmetric_numerical_columns_df = df_transform[num_cols].apply(lambda x: np.
 ↪log1p(x))
plt.figure(figsize=(25, 15))
sns.boxplot(data=symmetric_numerical_columns_df)
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



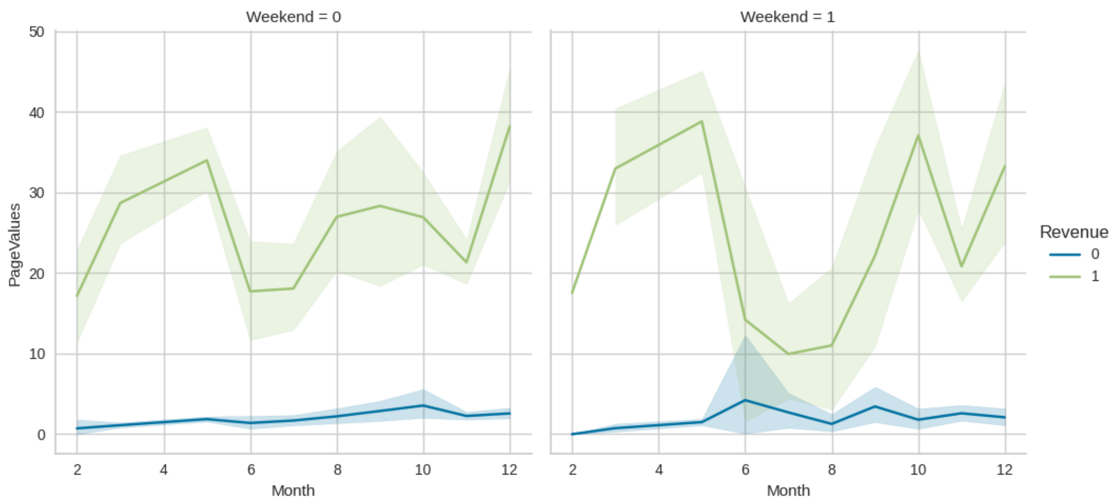
[14]:
```
#Understanding the correlation between variables
plt.figure(figsize=(15,10))
sns.heatmap(df_transform.corr(), annot = True, fmt= '.2f', cmap='coolwarm',␣
 ↪linewidths=2)
plt.title('correlation heatmap')
# Save the heatmap as an image file before displaying it
plt.savefig("correlation_heatmap.png", dpi=300, bbox_inches="tight")
plt.show()
```



[15]:
```
#Create a pair wise plot between all features that show desirable correlations
corr_columns =␣
 ↪['ProductRelated_Duration','ExitRates','BounceRates','PageValues','SpecialDay','Weekend','R
#sns.pairplot(df[corr_columns],hue ='Revenue')
#plt.show()
```

[16]:
```
#Time series relationship between weekend and pageValues
sns.relplot(x="Month", y="PageValues", data=df_transform, kind="line",␣
 ↪hue="Revenue",col="Weekend")
```

[16]: <seaborn.axisgrid.FacetGrid at 0x7a46762be910>



1.0.3 Feature Engineering and Feature Selection

From the correlation matrix, we can clearly see high correlation between the Administrative, Informational and Product related features. Hence, we will combine these features to create a new feature, Total Page Views = Administrative + Informational + Product Related.

Similarly, we will create the Total Page Duration by summing all the durations.

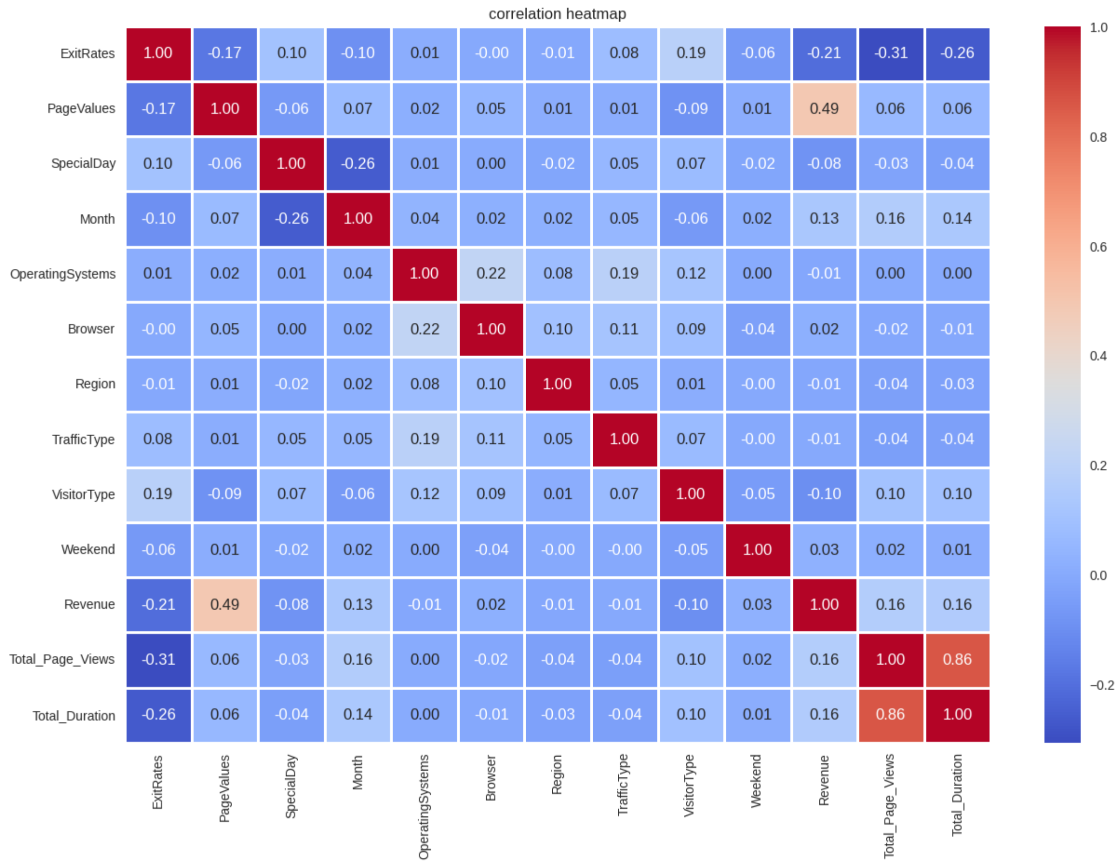
[17]:
```
#Combine different features and create new features
df_transform['Total_Page_Views'] = df_transform['Administrative'] +␣
↪df_transform['Informational'] + df_transform['ProductRelated']

df_transform['Total_Duration'] = (df_transform['Administrative_Duration'] +␣
↪df_transform['Informational_Duration'] +
                                 df_transform['ProductRelated_Duration'])
```

Bounce Rate and Exit Rates are highly correlated with a correlation of 0.91. Hence we will be removing one of these features to remove redundancy.

[18]:
```
#Drop columns with high correlation
df_transform.
↪drop(columns={'BounceRates','Administrative','Administrative_Duration','Informational',
                                       ␣
↪'Informational_Duration','ProductRelated','ProductRelated_Duration'},
                  inplace=True)
```

[19]:
```
#Understanding the correlation between variables
plt.figure(figsize=(15,10))
sns.heatmap(df_transform.corr(), annot = True, fmt= '.2f', cmap='coolwarm',␣
↪linewidths=2)
```

```
plt.title('correlation heatmap')
plt.show()
```



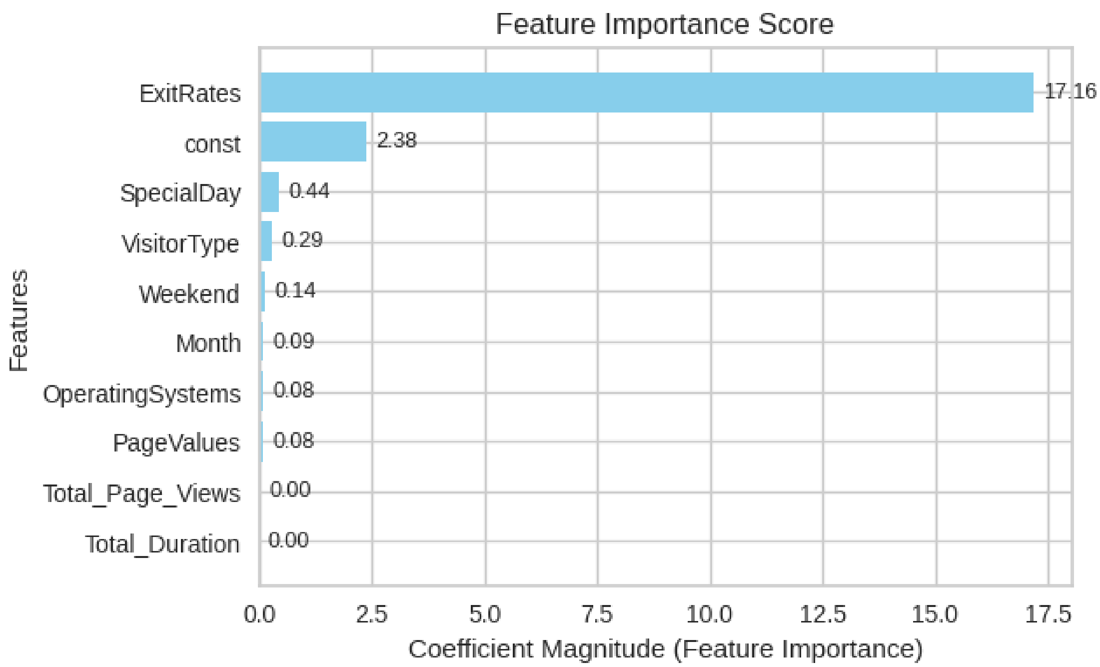
Backward elimination is a feature selection technique that starts with all the features in the model and iteratively removes the least significant ones, based on statistical tests (usually p-values), until only the most significant features remain.

[20]:
```
#'Revenue' is our target variable
X = df_transform.drop('Revenue', axis=1)
y = df_transform['Revenue']

#Fit the initial logistic regression model
X = sm.add_constant(X)
model = sm.Logit(y, X)
result = model.fit()

#Start backward elimination process
while True:
    p_values = result.pvalues
    max_p_value = p_values.max()
```

```
    if max_p_value > 0.05:
        feature_to_remove = p_values.idxmax()
        X = X.drop(feature_to_remove, axis=1)
        model = sm.Logit(y, X)
        result = model.fit()
    else:
        break

#The final features after backward elimination
selected_features = X.columns.tolist()
print("Selected features:", selected_features)
```

```
Optimization terminated successfully.
         Current function value: 0.296515
         Iterations 8
Optimization terminated successfully.
         Current function value: 0.296560
         Iterations 8
Optimization terminated successfully.
         Current function value: 0.296631
         Iterations 8
Optimization terminated successfully.
         Current function value: 0.296772
         Iterations 8
Selected features: ['const', 'ExitRates', 'PageValues', 'SpecialDay', 'Month',
'OperatingSystems', 'VisitorType', 'Weekend', 'Total_Page_Views',
'Total_Duration']
```

[21]:
```
#Get feature importance and sort them
feature_importance = result.params.abs().sort_values(ascending=False)

#Plot feature importance
plt.figure(figsize=(6, 4))
plt.barh(feature_importance.index, feature_importance.values, color='skyblue')

#Display feature importance value for each feature
for index, value in enumerate(feature_importance.values):
    plt.text(value + 0.2, index, f"{value:.2f}", va='center', fontsize=9)
plt.xlabel("Coefficient Magnitude (Feature Importance)")
plt.ylabel("Features")
plt.title("Feature Importance Score")
plt.gca().invert_yaxis()
plt.show()
```



[22]:
```
#Apply standard scalar to the numeric variables
num_cols =␣
↪['ExitRates','PageValues','SpecialDay','Weekend','Total_Page_Views','Total_Duration']
scaler = StandardScaler()
df_scaled_num = scaler.fit_transform(df_transform[num_cols])
df_scaled_num = pd.DataFrame(df_scaled_num, columns=num_cols).
↪reset_index(drop=True)

#Concatenate with the categorical variables
cat_cols =␣
↪['Revenue','Month','OperatingSystems','Browser','Region','TrafficType','VisitorType']
df_categorical = df_transform[cat_cols].reset_index(drop=True)
df_categorical[cat_cols] = df_categorical[cat_cols].astype(int)
df_scaled = pd.concat([df_scaled_num, df_categorical], axis=1)
df_scaled.head()
```

[22]:
```
   ExitRates  PageValues  SpecialDay   Weekend  Total_Page_Views  \
0   3.229316   -0.317178   -0.308821 -0.550552         -0.721321
1   1.171473   -0.317178   -0.308821 -0.550552         -0.699821
2   3.229316   -0.317178   -0.308821 -0.550552         -0.721321
3   1.994610   -0.317178   -0.308821 -0.550552         -0.699821
4   0.142551   -0.317178   -0.308821  1.816360         -0.527823

   Total_Duration  Revenue  Month  OperatingSystems  Browser  Region  \
0       -0.642894        0      2                 1        1       1
```

```
1       -0.611486      0      2                 2        2       1
2       -0.642894      0      2                 4        1       9
3       -0.641585      0      2                 3        2       2
4       -0.334952      0      2                 3        3       1

   TrafficType  VisitorType
0            1            1
1            2            1
2            3            1
3            4            1
4            4            1
```

[23]:
```python
df_scaled.shape
```

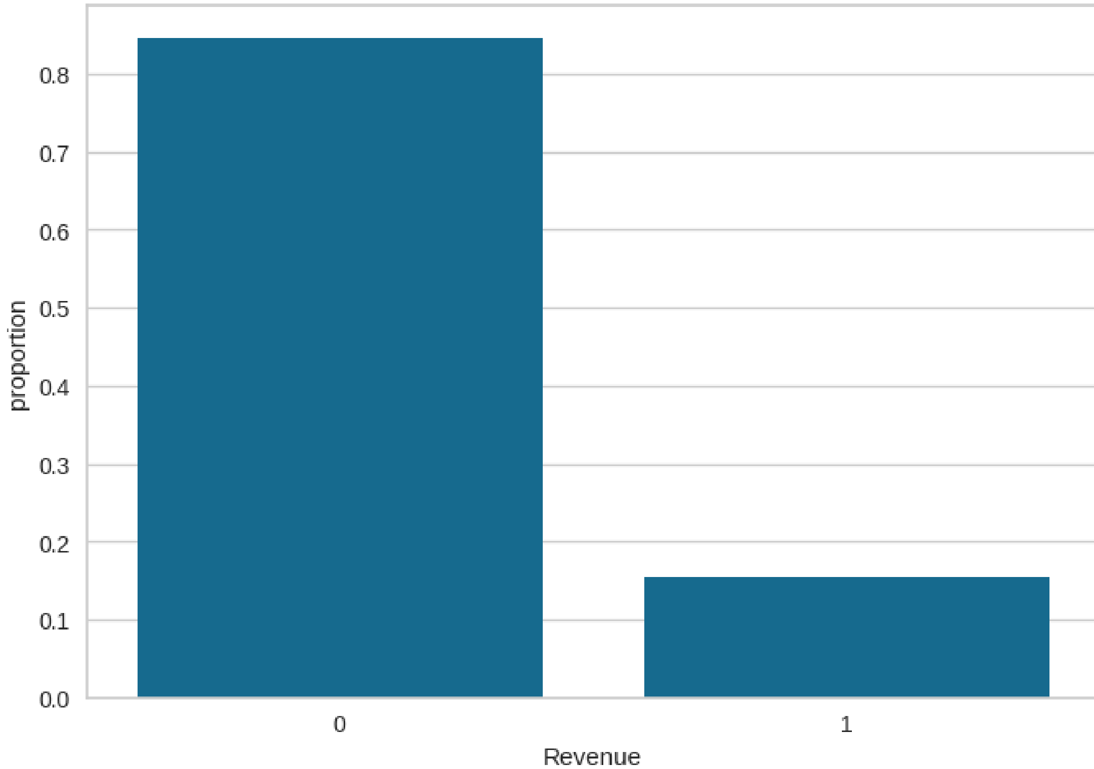
[23]:
```
(12330, 13)
```

[24]:
```python
#Define independent variables and target variable
X = df_scaled.drop(['Revenue'], axis=1)
y = df_scaled['Revenue']
```

1.0.4 Sampling

[25]:
```python
# Train-Test Split the dataset into 80:20 split and stratify it
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)
print(y_train.value_counts(normalize=True))
sns.barplot(y_train.value_counts(normalize=True))
plt.show()
```

```
Revenue
0    0.845296
1    0.154704
Name: proportion, dtype: float64
```

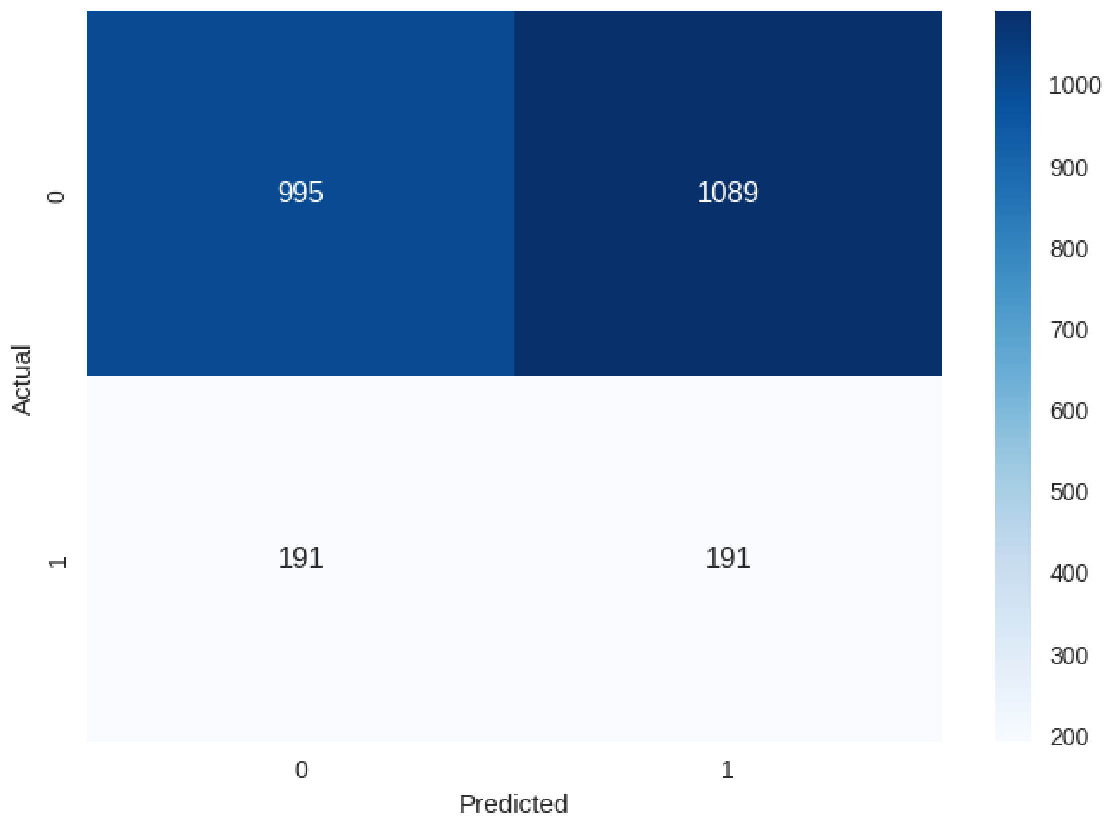


[26]:
```
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

X_train = X_resampled
y_train = y_resampled
```

1.0.5 Dummy Classifier

[27]:
```
#Train a dummy classifer
baseline_clf = DummyClassifier(strategy="stratified")
baseline_clf.fit(X_train, y_train)
y_pred_bclf = baseline_clf.predict(X_test)
```

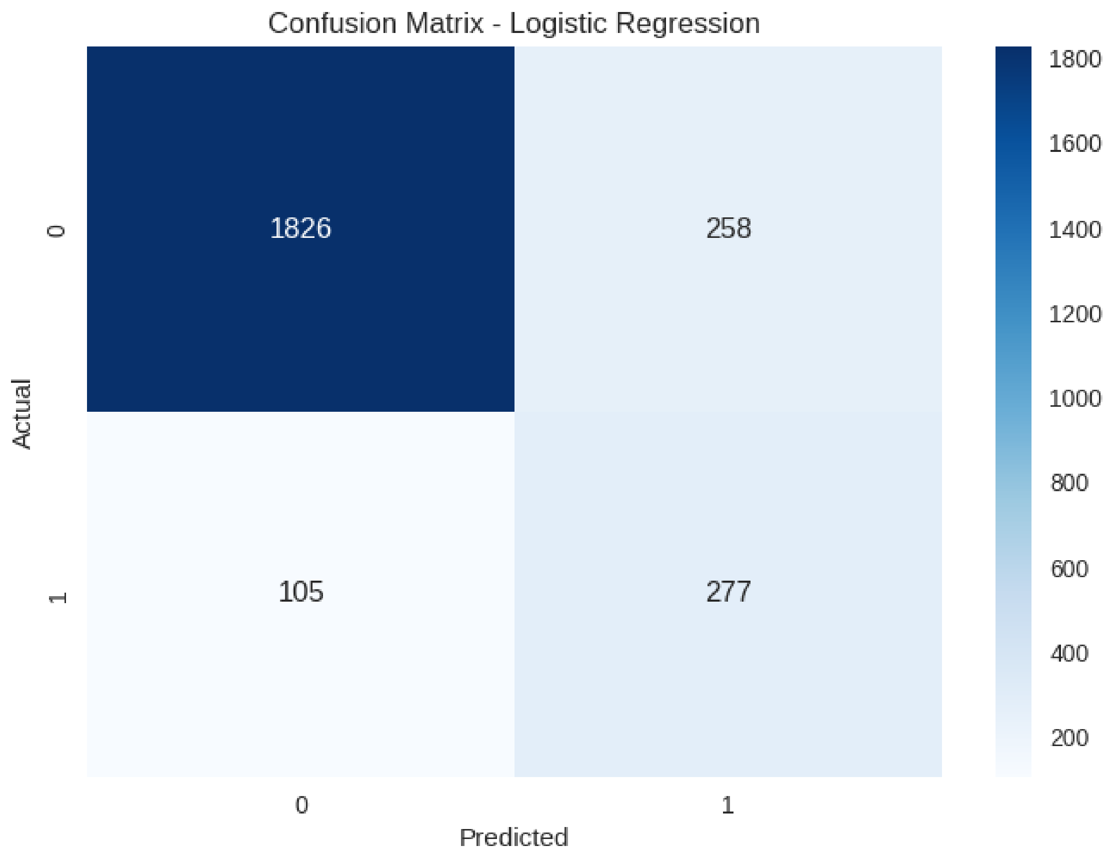
[28]:
```
#Print the confusion matrix
sns.heatmap(confusion_matrix(y_test, y_pred_bclf), annot=True, fmt="d",␣
 ↪cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



1.0.6 Logistic Regression

[29]:
```
#Train Logistic regression
model_lr = LogisticRegression(class_weight='balanced')
model_lr.fit(X_train,y_train)
y_pred_lr = model_lr.predict(X_test)

#Predict and evaluate accuracy
y_pred = model_lr.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Logistic Regression Accuracy: {accuracy:.2f}")

#Print the confusion matrix
sns.heatmap(confusion_matrix(y_test, y_pred_lr), annot=True, fmt="d",␣
 ↪cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title('Confusion Matrix - Logistic Regression')
plt.show()
```

```
Logistic Regression Accuracy: 0.85
```

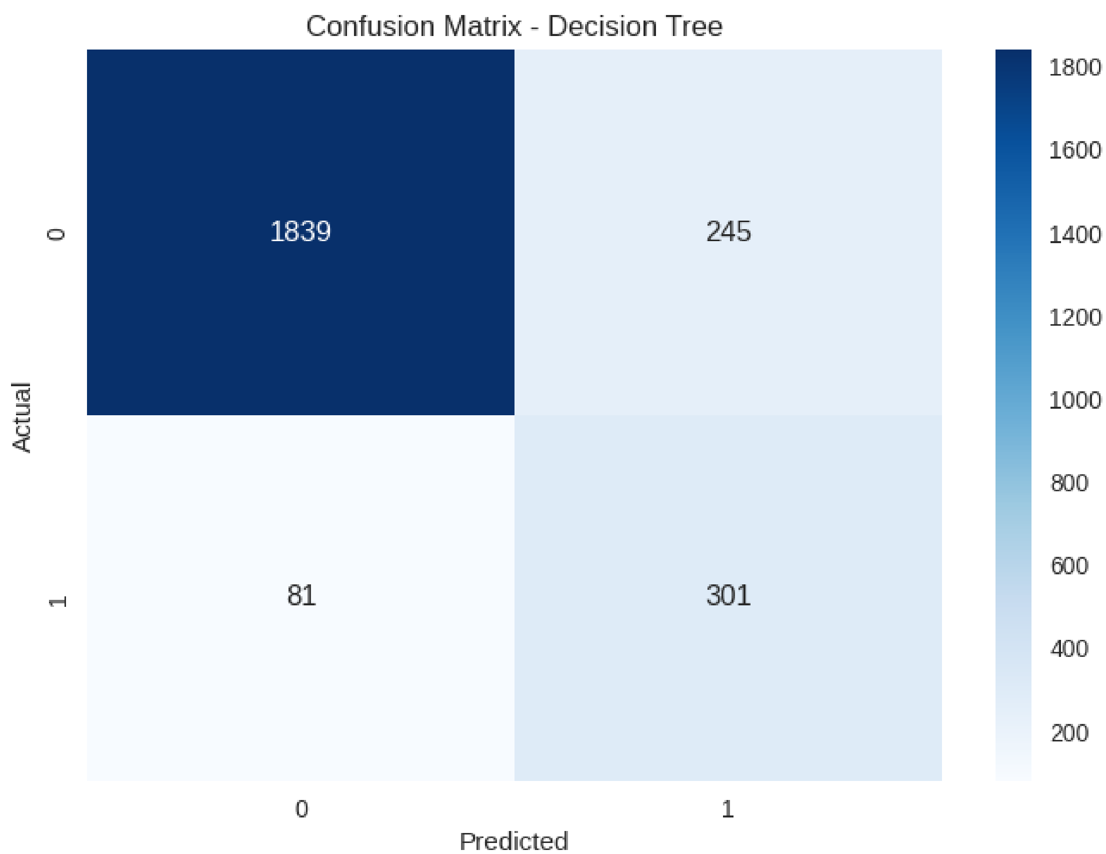


1.0.7 Decision Trees

[30]:
```
#Train a Decision Tree Classifier
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [3, 4, 5, 6, 7, 8, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [10, 15, 20, 25, 30],
    'class_weight' : ['balanced', None]
}

#Grid Search
grid_search = GridSearchCV(DecisionTreeClassifier(random_state=42), param_grid,
 ↪cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
model_dt = grid_search.best_estimator_
print(model_dt)

#Predict and evaluate accuracy
y_pred = model_dt.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Decision Tree Accuracy: {accuracy:.2f}")

#Display confusion matrix
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title('Confusion Matrix - Decision Tree')
plt.show()
```

```
DecisionTreeClassifier(class_weight='balanced', criterion='entropy',
                       min_samples_leaf=30, random_state=42)
Decision Tree Accuracy: 0.87
```

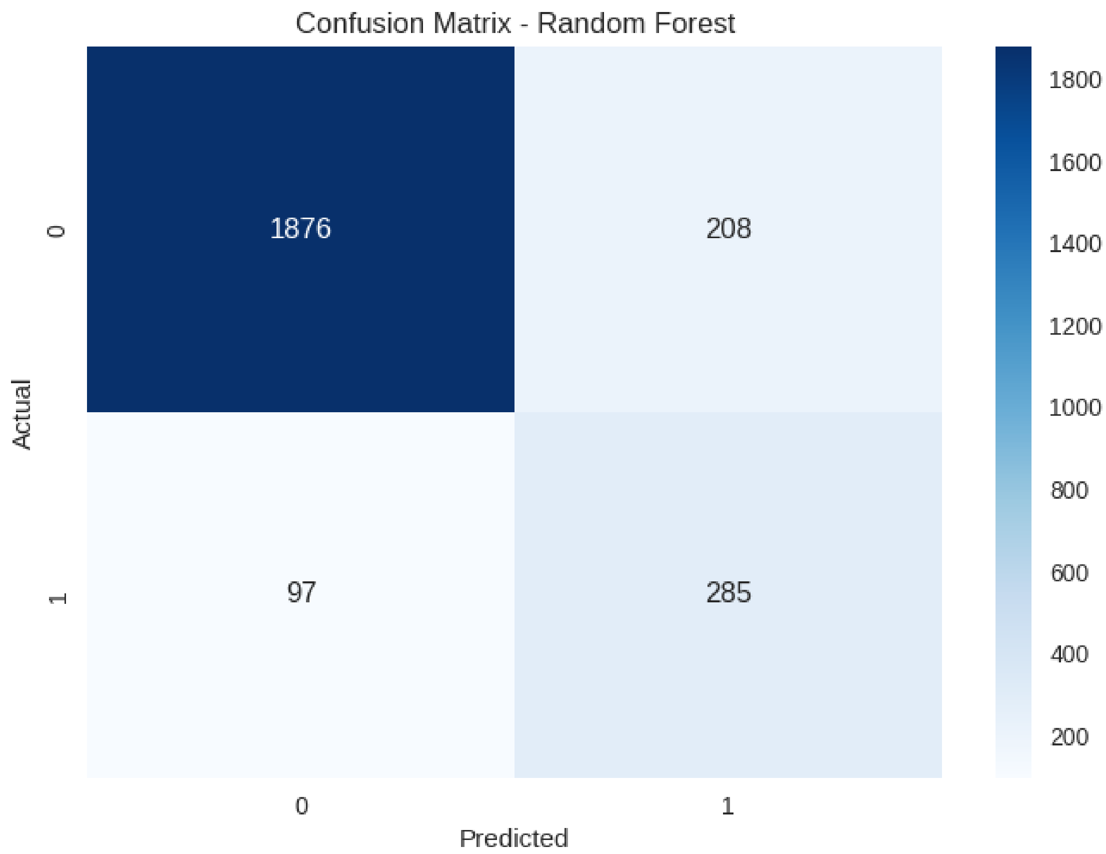


1.0.8 Random Forest

[31]:
```
#Train a Random Forest Classifier
param_grid = {
    'n_estimators': [101, 301, 501, 701],
    'max_depth': [10, 20, 25, 30],
```

```python
    'class_weight' : ['balanced', None]
}

#Grid Search
grid_search = GridSearchCV(RandomForestClassifier(random_state=42), param_grid,␣
 ↪cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
model_rf = grid_search.best_estimator_
print(model_rf)

#Predict and evaluate accuracy
y_pred = model_rf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Random Forest Accuracy: {accuracy:.2f}")

#Display confusion matrix
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title('Confusion Matrix - Random Forest')
plt.show()
```

```
RandomForestClassifier(class_weight='balanced', max_depth=25, n_estimators=501,
                       random_state=42)
Random Forest Accuracy: 0.88
```



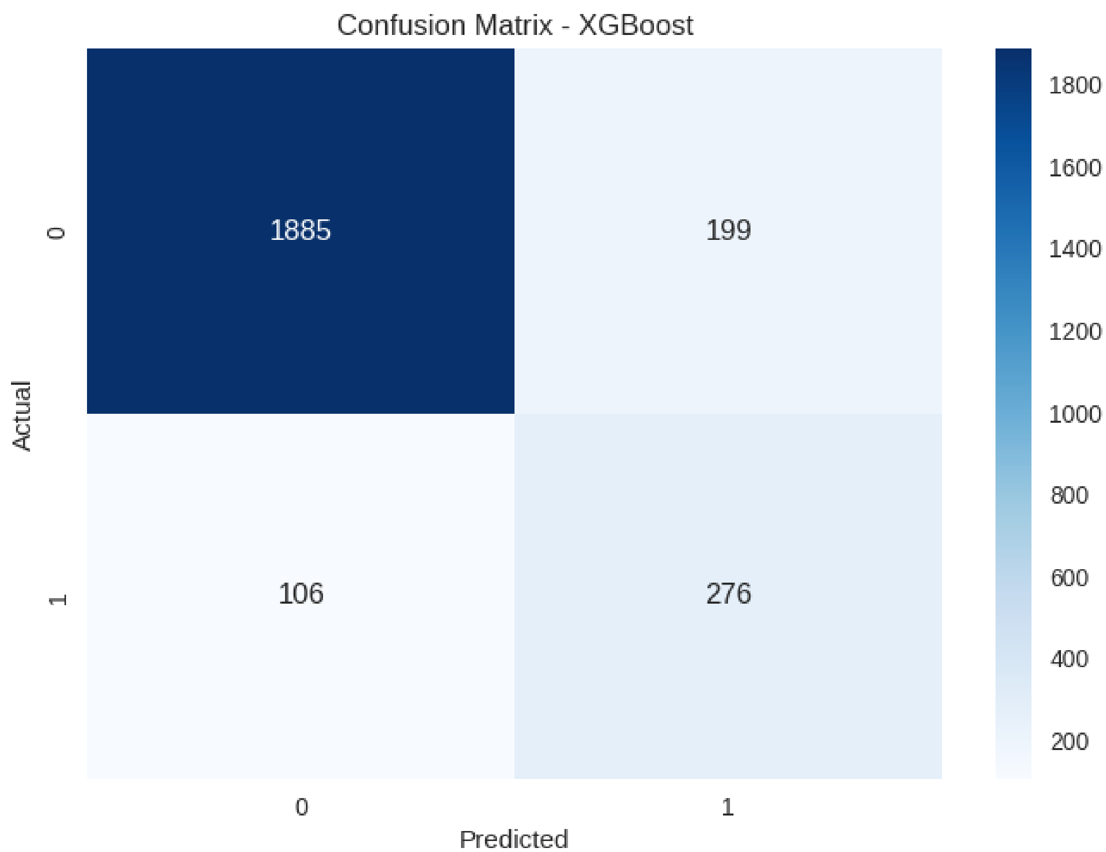
1.0.9 XGBoost

[34]:

```
#Train a XG Boost Classifier
param_grid = {
    'n_estimators': [301, 501, 701],
    'learning_rate': [0.01, 0.05],
    'max_depth': [10, 20, 25]
}

#Model and fit logistic regression
grid_search = GridSearchCV(xgb.XGBClassifier(random_state=42), param_grid,␣
 ↪cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
model_xg = grid_search.best_estimator_
print(model_xg)

#Predict and evaluate accuracy
y_pred = model_xg.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"XGBoost Accuracy: {accuracy:.2f}")
```

```python
#Display confusion matrix
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title('Confusion Matrix - XGBoost')
plt.show()
```

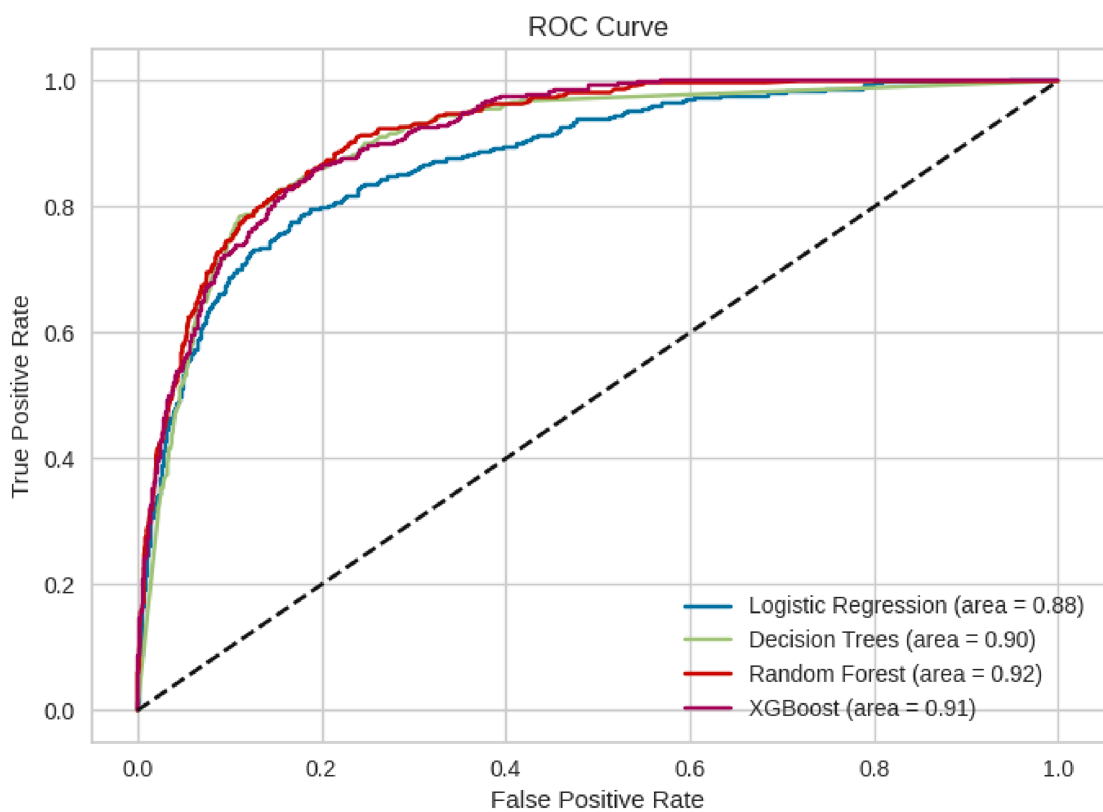
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.05, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=20, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=701, n_jobs=None,
              num_parallel_tree=None, objective='binary:logistic', …)
XGBoost Accuracy: 0.88
```



[35]:
```
#Get all the model names and models
model_name = ['Logistic Regression', 'Decision Trees', 'Random Forest',␣
↪'XGBoost']
models = [model_lr, model_dt, model_rf, model_xg]
```

[36]:
```
#Define dataframe to save results
results = []

#For each model calculate evaluation metrics and plot ROC curve
for model,name in zip(models,model_name):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred) * 100
    precision = precision_score(y_test, y_pred, average='weighted',␣
↪zero_division=0) * 100
    recall = recall_score(y_test, y_pred, average='weighted', zero_division=0)␣
↪* 100
    f1 = f1_score(y_test, y_pred, average='weighted', zero_division=0) * 100

    y_pred_proba = model.predict_proba(X_test)[:, 1]
    roc_auc = roc_auc_score(y_test, y_pred_proba) * 100
    fpr, tpr, _ = roc_curve(y_test,y_pred_proba)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'{name} (area = {roc_auc:.2f})')

    results.append([name, accuracy, precision, recall, f1, roc_auc])

results_df = pd.DataFrame(results, columns=["Model", "Accuracy", "Precision",␣
↪"Recall", "F1-score", "ROC-AUC"])
print(results_df)

#Plot the ROC curve
plt.plot([0, 1], [0, 1], 'k--')
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.show()
```

```
                 Model   Accuracy  Precision     Recall   F1-score   ROC-AUC
0  Logistic Regression  85.279805  87.934455  85.279805  86.227354  0.876277
1       Decision Trees  86.780211  89.483820  86.780211  87.677604  0.903811
2        Random Forest  87.631792  89.309589  87.631792  88.247092  0.916803
3              XGBoost  87.631792  89.010982  87.631792  88.161745  0.913320
```



AutoML using PyCaret

[37]:
```
from pycaret.classification import *

# Setup AutoML on already preprocessed + resampled training data i,e. df_scaled
automl_setup = setup(
    data=df_scaled,
    target='Revenue',
    session_id=42,
    preprocess=False  # we have already scaled and encoded everything in␣
 ↪df_scaled
)
```

<pandas.io.formats.style.Styler at 0x7a4675ca5350>

[38]:
```
# Run AutoML to compare models
best_model = compare_models()

# Evaluate model performance and analyze the model
evaluate_model(best_model)
```

<IPython.core.display.HTML object>

```
<pandas.io.formats.style.Styler at 0x7a4675d7b390>
```

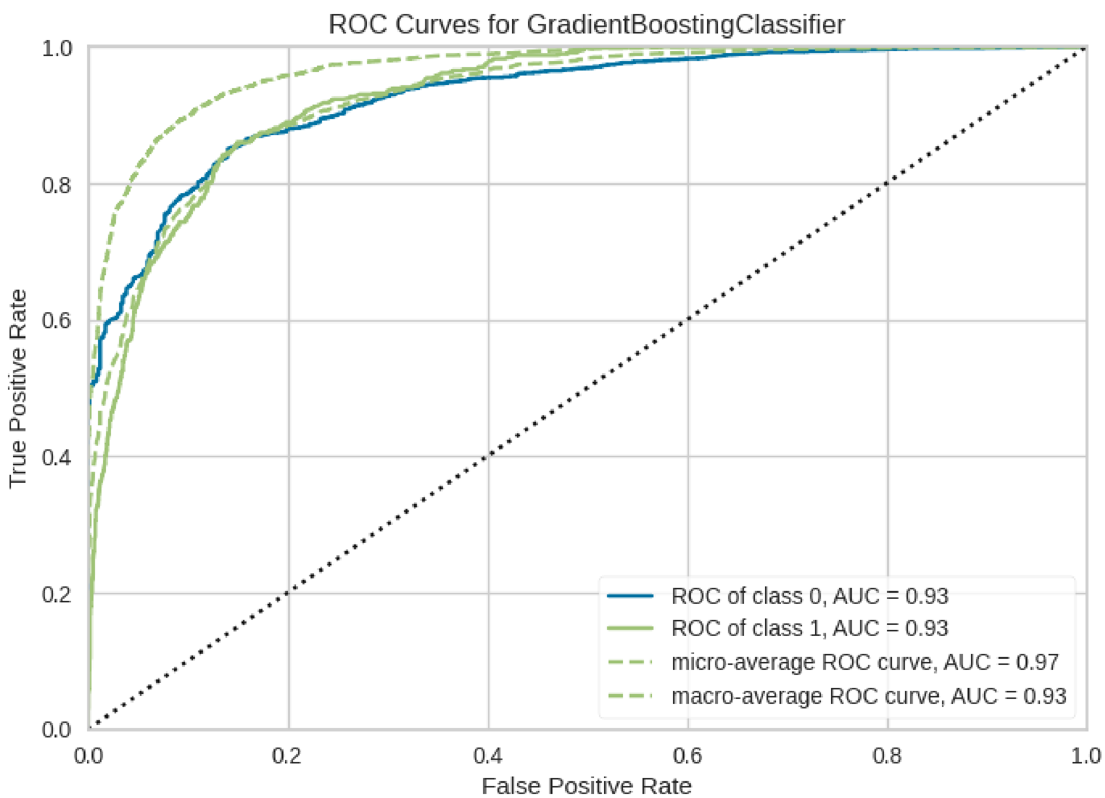
```
Processing:   0%|          | 0/65 [00:00<?, ?it/s]
```

```
<IPython.core.display.HTML object>
```

```
interactive(children=(ToggleButtons(description='Plot Type:', icons=('',),␣
↪options=(('Pipeline Plot', 'pipelin…
```

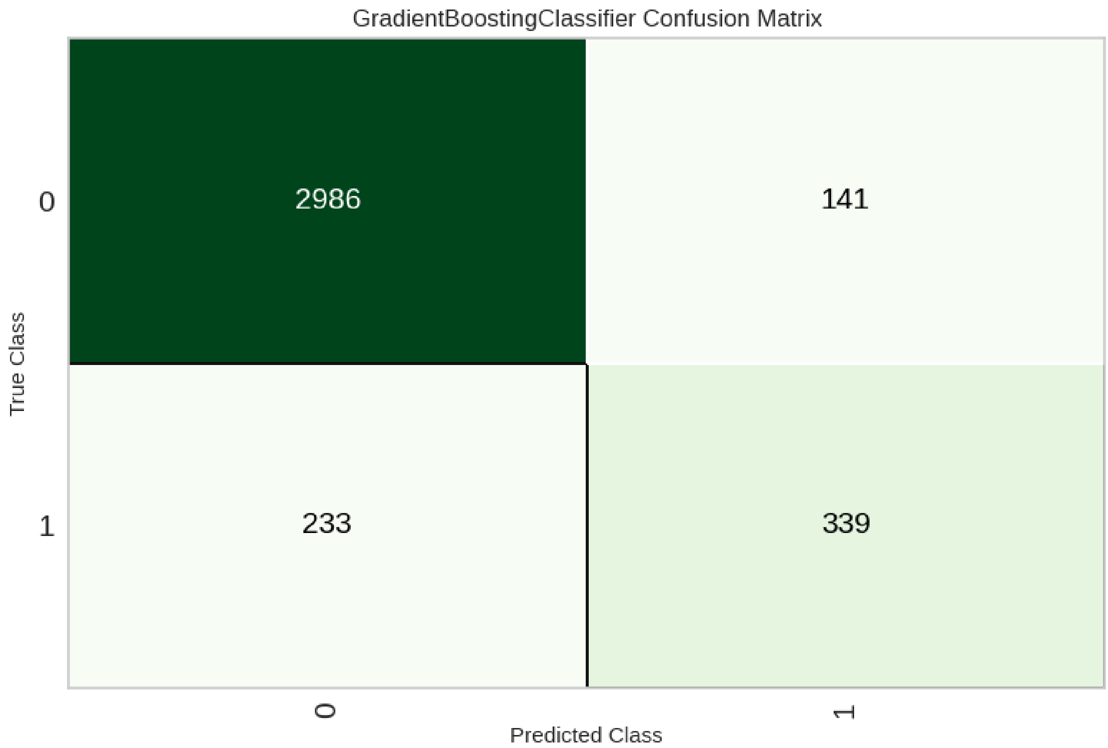
[41]:
```python
# Plot the ROC curve
plot_model(best_model, plot = 'auc')
```

```
<IPython.core.display.HTML object>
```



[43]:
```python
# Confusion matrix
plot_model(best_model, plot = 'confusion_matrix')
```

```
<IPython.core.display.HTML object>
```



[45]:
```
# Predicts the model
predict_model(best_model)
```

<pandas.io.formats.style.Styler at 0x7a4675bea010>

[45]:

	ExitRates	PageValues	SpecialDay	Weekend	Total_Page_Views	\
136	0.066335	-0.317178	-0.308821	-0.550552	-0.549323	
3454	1.583041	-0.317178	-0.308821	1.816360	-0.635322	
10365	-0.630651	-0.019683	-0.308821	-0.550552	1.192157	
11007	-0.736869	-0.317178	-0.308821	-0.550552	0.095670	
9291	-0.525081	-0.090259	-0.308821	-0.550552	3.170134	
…	…	…	…	…	…	
9369	1.171473	-0.317178	-0.308821	-0.550552	-0.699821	
12079	-0.778063	-0.317178	-0.308821	-0.550552	0.095670	
7296	-0.594102	-0.317178	-0.308821	-0.550552	0.719162	
9220	-0.371910	-0.317178	-0.308821	-0.550552	-0.613822	
3028	-0.474802	-0.317178	-0.308821	1.816360	-0.506324	

	Total_Duration	Month	OperatingSystems	Browser	Region	TrafficType	\
136	-0.493871	2	2	4	5	2	
3454	-0.631116	5	1	1	1	4	
10365	0.483425	12	2	6	1	2	
11007	-0.023254	11	3	2	2	10	

```
9291          2.661767     11                  1        1       1             1
...                ...     ...              ...      ...     ...          ...
9369         -0.635533     12                  2        2       9             1
12079        -0.336621     12                  2        2       1             2
7296          0.375365      7                  3        2       6             3
9220         -0.354418     11                  3        2       1             3
3028         -0.377646      5                  2        2       7             4

       VisitorType  Revenue  prediction_label  prediction_score
136              1        0                 0            0.9944
3454             1        0                 0            0.9957
10365            1        0                 0            0.6468
11007            1        1                 0            0.9189
9291             1        1                 1            0.6121
...            ...      ...               ...               ...
9369             1        0                 0            0.9880
12079            0        0                 0            0.9466
7296             1        0                 0            0.9498
9220             0        0                 0            0.9156
3028             0        0                 0            0.9909

[3699 rows x 15 columns]
```

[46]:
```python
#  Finalize and predict on test set
final_model = finalize_model(best_model)

predictions = predict_model(final_model, data=df_scaled)
```

```
<pandas.io.formats.style.Styler at 0x7a4859e4b610>
```

[49]:
```python
# functional API
predictions = predict_model(best_model, data=df_scaled, raw_score=True)
predictions.head()
```

```
<pandas.io.formats.style.Styler at 0x7a4675d9e2d0>
```

[49]:
```
   ExitRates  PageValues  SpecialDay   Weekend  Total_Page_Views  \
0   3.229316   -0.317178   -0.308821 -0.550552         -0.721321
1   1.171473   -0.317178   -0.308821 -0.550552         -0.699821
2   3.229316   -0.317178   -0.308821 -0.550552         -0.721321
3   1.994610   -0.317178   -0.308821 -0.550552         -0.699821
4   0.142551   -0.317178   -0.308821  1.816360         -0.527823

   Total_Duration  Month  OperatingSystems  Browser  Region  TrafficType  \
0       -0.642894      2                 1        1       1            1
1       -0.611486      2                 2        2       1            2
2       -0.642894      2                 4        1       9            3
```

```
3       -0.641585      2                 3        2       2             4
4       -0.334952      2                 3        3       1             4

   VisitorType  Revenue  prediction_label  prediction_score_0  \
0            1        0                 0              0.9961
1            1        0                 0              0.9957
2            1        0                 0              0.9960
3            1        0                 0              0.9957
4            1        0                 0              0.9941

   prediction_score_1
0              0.0039
1              0.0043
2              0.0040
3              0.0043
4              0.0059
```

Score means the probability of the predicted class (NOT the positive class). If prediction_label is 0 and prediction_score is 0.90, this means 90% probability of class 0. To see the probability of both the classes, we have kept raw_score=True.

[51]:
```
# save the model
save_model(best_model, 'my_best_pipeline')
```

```
Transformation Pipeline and Model Successfully Saved
```

[51]:
```
(Pipeline(memory=Memory(location=None),
          steps=[('placeholder', None),
                 ('trained_model',
                  GradientBoostingClassifier(ccp_alpha=0.0,
                                             criterion='friedman_mse',
init=None,
                                             learning_rate=0.1, loss='log_loss',
                                             max_depth=3, max_features=None,
                                             max_leaf_nodes=None,
                                             min_impurity_decrease=0.0,
                                             min_samples_leaf=1,
                                             min_samples_split=2,
                                             min_weight_fraction_leaf=0.0,
                                             n_estimators=100,
                                             n_iter_no_change=None,
                                             random_state=42, subsample=1.0,
                                             tol=0.0001,
validation_fraction=0.1,
                                             verbose=0, warm_start=False))],
          verbose=False),
 'my_best_pipeline.pkl')
```

[53]:
```
print(predictions.columns)
```

```
Index(['ExitRates', 'PageValues', 'SpecialDay', 'Weekend', 'Total_Page_Views',
       'Total_Duration', 'Month', 'OperatingSystems', 'Browser', 'Region',
       'TrafficType', 'VisitorType', 'Revenue', 'prediction_label',
       'prediction_score_0', 'prediction_score_1'],
      dtype='object')
```

[59]:
```
# View prediction summary

print(confusion_matrix(df_scaled['Revenue'], predictions['prediction_label']))
print(classification_report(df_scaled['Revenue'],␣
 ↪predictions['prediction_label']))
```

```
[[10025   397]
 [  682  1226]]
              precision    recall  f1-score   support

           0       0.94      0.96      0.95     10422
           1       0.76      0.64      0.69      1908

    accuracy                           0.91     12330
   macro avg       0.85      0.80      0.82     12330
weighted avg       0.91      0.91      0.91     12330
```

AutoML using H2o

[63]:
```
import h2o
from h2o.automl import H2OAutoML

# Initialize H2O cluster
h2o.init()

# Assuming your pandas dataframe is df
# Convert pandas DataFrame to H2OFrame
h2o_df = h2o.H2OFrame(df)

# Convert the target column to categorical (binary classification)
target = 'Revenue'  # Replace with your actual target column name
h2o_df[target] = h2o_df[target].asfactor()

# Split the dataset into training and test sets (80-20 split)
train, test = h2o_df.split_frame(ratios=[.8], seed=42)

# Define the predictors
predictors = [col for col in h2o_df.columns if col != target]
```

```python
# Initialize H2O AutoML and set parameters
automl = H2OAutoML(max_models=20, seed=42, max_runtime_secs=600)

# Train the AutoML model
automl.train(x=predictors, y=target, training_frame=train)

# View the leaderboard of models
leaderboard = automl.leaderboard
print(leaderboard)
```

```
Checking whether there is an H2O instance running at http://localhost:54321…
not found.
Attempting to start a local H2O server…
  Java Version: openjdk version "11.0.26" 2025-01-21; OpenJDK Runtime
Environment (build 11.0.26+4-post-Ubuntu-1ubuntu122.04); OpenJDK 64-Bit Server
VM (build 11.0.26+4-post-Ubuntu-1ubuntu122.04, mixed mode, sharing)
  Starting server from /usr/local/lib/python3.11/dist-
packages/h2o/backend/bin/h2o.jar
  Ice root: /tmp/tmpbj4wf0zl
  JVM stdout: /tmp/tmpbj4wf0zl/h2o_unknownUser_started_from_python.out
  JVM stderr: /tmp/tmpbj4wf0zl/h2o_unknownUser_started_from_python.err
  Server is running at http://127.0.0.1:54321
Connecting to H2O server at http://127.0.0.1:54321 … successful.
```

```
--------------------------
 ↪-----------------------------------------------------------------------------------------
H2O_cluster_uptime:         02 secs
H2O_cluster_timezone:       Etc/UTC
H2O_data_parsing_timezone:  UTC
H2O_cluster_version:        3.46.0.7
H2O_cluster_version_age:    17 days
H2O_cluster_name:           H2O_from_python_unknownUser_zgllxp
H2O_cluster_total_nodes:    1
H2O_cluster_free_memory:    12.75 Gb
H2O_cluster_total_cores:    8
H2O_cluster_allowed_cores:  8
H2O_cluster_status:         locked, healthy
H2O_connection_url:         http://127.0.0.1:54321
H2O_connection_proxy:       {"http": null, "https": null,
 ↪"colab_language_server": "/usr/colab/bin/language_service"}
H2O_internal_security:      False
Python_version:             3.11.12 final
--------------------------
 ↪-----------------------------------------------------------------------------------------
```

```
Parse progress:
|                                   | (done) 100%
AutoML progress:
```

```
|                                 | (done) 100%
model_id                                             auc    logloss     aucpr
mean_per_class_error      rmse        mse
GBM_3_AutoML_1_20250414_50453                   0.929466   0.232491  0.724022
0.163823  0.267608  0.0716143
GBM_grid_1_AutoML_1_20250414_50453_model_1      0.9293     0.231216  0.733876
0.178251  0.265356  0.0704137
GBM_1_AutoML_1_20250414_50453                   0.929263   0.229684  0.732321
0.163638  0.264489  0.0699546
GBM_5_AutoML_1_20250414_50453                   0.929123   0.231107  0.729452
0.16195   0.266342  0.0709379
GBM_2_AutoML_1_20250414_50453                   0.928832   0.23135   0.72526
0.162592  0.265728  0.0706113
XGBoost_3_AutoML_1_20250414_50453               0.927808   0.232923  0.72243
0.170257  0.267435  0.0715215
GBM_grid_1_AutoML_1_20250414_50453_model_2      0.925603   0.237274  0.720929
0.168568  0.268755  0.0722292
XGBoost_1_AutoML_1_20250414_50453               0.925226   0.237399  0.719107
0.173431  0.269279  0.0725109
GBM_4_AutoML_1_20250414_50453                   0.924939   0.23729   0.718676
0.170311  0.268845  0.0722776
XGBoost_grid_1_AutoML_1_20250414_50453_model_3  0.921026   0.251526  0.702938
0.183419  0.276648  0.0765338
[19 rows x 7 columns]
```

[64]:
```python
# Get the best model (leader)
leader = automl.leader

# Make predictions on the test set
predictions = leader.predict(test)
h2o_leader_params = leader.params
print(h2o_leader_params)

# Evaluate the model performance (e.g., AUC for classification)
performance = leader.model_performance(test)
print("AUC: ", performance.auc())

#Accuracy
accuracy = performance.accuracy()
print(f"Accuracy: {accuracy[0][1]}")

# Precision
precision = performance.precision()
print(f"Precision: {precision[0][0]}")

# Recall
```

```python
recall = performance.recall()
print(f"Recall: {recall[0][0]}")

# F1 Score
f1_score = performance.F1()
print(f"F1 Score: {f1_score[0][1]}")
h2o_confusion_matrix = performance.confusion_matrix()
print(h2o_confusion_matrix)
```

```
gbm prediction progress:
|                              | (done) 100%
```


```
{'model_id': {'default': None, 'actual': {'__meta': {'schema_version': 3,
'schema_name': 'ModelKeyV3', 'schema_type': 'Key<Model>'}, 'name':
'GBM_3_AutoML_1_20250414_50453', 'type': 'Key<Model>', 'URL':
'/3/Models/GBM_3_AutoML_1_20250414_50453'}, 'input': None}, 'training_frame':
{'default': None, 'actual': {'__meta': {'schema_version': 3, 'schema_name':
'FrameKeyV3', 'schema_type': 'Key<Frame>'}, 'name':
'AutoML_1_20250414_50453_training_py_3_sid_ab5a', 'type': 'Key<Frame>', 'URL':
'/3/Frames/AutoML_1_20250414_50453_training_py_3_sid_ab5a'}, 'input': {'__meta':
{'schema_version': 3, 'schema_name': 'FrameKeyV3', 'schema_type': 'Key<Frame>'},
'name': 'AutoML_1_20250414_50453_training_py_3_sid_ab5a', 'type': 'Key<Frame>',
'URL': '/3/Frames/AutoML_1_20250414_50453_training_py_3_sid_ab5a'}},
'validation_frame': {'default': None, 'actual': None, 'input': None}, 'nfolds':
{'default': 0, 'actual': 5, 'input': 5}, 'keep_cross_validation_models':
{'default': True, 'actual': False, 'input': False},
'keep_cross_validation_predictions': {'default': False, 'actual': True, 'input':
True}, 'keep_cross_validation_fold_assignment': {'default': False, 'actual':
False, 'input': False}, 'score_each_iteration': {'default': False, 'actual':
False, 'input': False}, 'score_tree_interval': {'default': 0, 'actual': 5,
'input': 5}, 'fold_assignment': {'default': 'AUTO', 'actual': 'Modulo', 'input':
'Modulo'}, 'fold_column': {'default': None, 'actual': None, 'input': None},
'response_column': {'default': None, 'actual': {'__meta': {'schema_version': 3,
'schema_name': 'ColSpecifierV3', 'schema_type': 'VecSpecifier'}, 'column_name':
'Revenue', 'is_member_of_frames': None}, 'input': {'__meta': {'schema_version':
3, 'schema_name': 'ColSpecifierV3', 'schema_type': 'VecSpecifier'},
'column_name': 'Revenue', 'is_member_of_frames': None}}, 'ignored_columns':
{'default': None, 'actual': [], 'input': []}, 'ignore_const_cols': {'default':
True, 'actual': True, 'input': True}, 'offset_column': {'default': None,
'actual': None, 'input': None}, 'weights_column': {'default': None, 'actual':
None, 'input': None}, 'balance_classes': {'default': False, 'actual': False,
'input': False}, 'class_sampling_factors': {'default': None, 'actual': None,
'input': None}, 'max_after_balance_size': {'default': 5.0, 'actual': 5.0,
'input': 5.0}, 'max_confusion_matrix_size': {'default': 20, 'actual': 20,
'input': 20}, 'ntrees': {'default': 50, 'actual': 51, 'input': 10000},
'max_depth': {'default': 5, 'actual': 8, 'input': 8}, 'min_rows': {'default':
10.0, 'actual': 10.0, 'input': 10.0}, 'nbins': {'default': 20, 'actual': 20,
'input': 20}, 'nbins_top_level': {'default': 1024, 'actual': 1024, 'input':
```

1024}, 'nbins_cats': {'default': 1024, 'actual': 1024, 'input': 1024}, 'r2_stopping': {'default': 1.7976931348623157e+308, 'actual': 1.7976931348623157e+308, 'input': 1.7976931348623157e+308}, 'stopping_rounds': {'default': 0, 'actual': 0, 'input': 3}, 'stopping_metric': {'default': 'AUTO', 'actual': 'logloss', 'input': 'logloss'}, 'stopping_tolerance': {'default': 0.001, 'actual': 0.010045812911315203, 'input': 0.010045812911315203}, 'max_runtime_secs': {'default': 0.0, 'actual': 0.0, 'input': 0.0}, 'seed': {'default': -1, 'actual': 48, 'input': 48}, 'build_tree_one_node': {'default': False, 'actual': False, 'input': False}, 'learn_rate': {'default': 0.1, 'actual': 0.1, 'input': 0.1}, 'learn_rate_annealing': {'default': 1.0, 'actual': 1.0, 'input': 1.0}, 'distribution': {'default': 'AUTO', 'actual': 'bernoulli', 'input': 'bernoulli'}, 'quantile_alpha': {'default': 0.5, 'actual': 0.5, 'input': 0.5}, 'tweedie_power': {'default': 1.5, 'actual': 1.5, 'input': 1.5}, 'huber_alpha': {'default': 0.9, 'actual': 0.9, 'input': 0.9}, 'checkpoint': {'default': None, 'actual': None, 'input': None}, 'sample_rate': {'default': 1.0, 'actual': 0.8, 'input': 0.8}, 'sample_rate_per_class': {'default': None, 'actual': None, 'input': None}, 'col_sample_rate': {'default': 1.0, 'actual': 0.8, 'input': 0.8}, 'col_sample_rate_change_per_level': {'default': 1.0, 'actual': 1.0, 'input': 1.0}, 'col_sample_rate_per_tree': {'default': 1.0, 'actual': 0.8, 'input': 0.8}, 'min_split_improvement': {'default': 1e-05, 'actual': 1e-05, 'input': 1e-05}, 'histogram_type': {'default': 'AUTO', 'actual': 'UniformAdaptive', 'input': 'AUTO'}, 'max_abs_leafnode_pred': {'default': 1.7976931348623157e+308, 'actual': 1.7976931348623157e+308, 'input': 1.7976931348623157e+308}, 'pred_noise_bandwidth': {'default': 0.0, 'actual': 0.0, 'input': 0.0}, 'categorical_encoding': {'default': 'AUTO', 'actual': 'Enum', 'input': 'AUTO'}, 'calibrate_model': {'default': False, 'actual': False, 'input': False}, 'calibration_frame': {'default': None, 'actual': None, 'input': None}, 'calibration_method': {'default': 'AUTO', 'actual': 'PlattScaling', 'input': 'AUTO'}, 'custom_metric_func': {'default': None, 'actual': None, 'input': None}, 'custom_distribution_func': {'default': None, 'actual': None, 'input': None}, 'export_checkpoints_dir': {'default': None, 'actual': None, 'input': None}, 'in_training_checkpoints_dir': {'default': None, 'actual': None, 'input': None}, 'in_training_checkpoints_tree_interval': {'default': 1, 'actual': 1, 'input': 1}, 'monotone_constraints': {'default': None, 'actual': None, 'input': None}, 'check_constant_response': {'default': True, 'actual': True, 'input': True}, 'gainslift_bins': {'default': -1, 'actual': -1, 'input': -1}, 'auc_type': {'default': 'AUTO', 'actual': 'AUTO', 'input': 'AUTO'}, 'interaction_constraints': {'default': None, 'actual': None, 'input': None}, 'auto_rebalance': {'default': True, 'actual': True, 'input': True}}

```
AUC:  0.9302554599911428
Accuracy: 0.9041718298223874
Precision: 0.9737221044041459
Recall: 0.009854520529219797
F1 Score: 0.6854082998661312
Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.4073161516023907
       0     1    Error    Rate
-----  ----  ---  -------  --------------
0      1930  113  0.0553   (113.0/2043.0)
```

1	122	256	0.3228	(122.0/378.0)
Total	2052	369	0.0971	(235.0/2421.0)