Analyzing User Behavior On E-Commerce Sites to Predict Revenue Generation

This notebook demonstrates the use of an SVM classifier with a polynomial kernel to classify website visitor data, with hyperparameter tuning using GridSearchCV. The goal is to predict whether a session resulted in a purchase or not based on various features. The Task is Binary Classification.

Data Background

The **Online Shoppers Purchasing Intention Dataset** comprises **12,330 sessions**, each representing a unique user visit to an e-commerce website over a **one-year period**. The dataset includes **17 features** and a response variable, **Revenue**, which indicates whether a purchase was made during the session.

Chosen Dataset: Hugging Face - jlh/uci-shopper
Official Dataset Source: UCI Machine Learning Repository

📌 Features

User Activity

- **Administrative**: Number of pages visited related to account management.
- **Administrative_Duration**: Total time (in seconds) spent on account management pages.
- **Informational**: Number of pages visited containing website, communication, and address information.
- **Informational_Duration**: Total time (in seconds) spent on informational pages.
- **ProductRelated**: Number of pages visited related to product information.
- **ProductRelated_Duration**: Total time (in seconds) spent on product-related pages.

Engagement Metrics

- **BounceRate**: Average bounce rate of the pages visited.
- **ExitRates**: Average exit rate of the pages visited.
- **PageValues**: Average value of the pages visited.
- **SpecialDay**: Closeness of the visit to a special day (e.g., Mother's Day, Valentine's Day).

Session Details

- **Month**: Month of the visit.

- **OperatingSystems**: Operating system used by the visitor.
- **Browser**: Browser used by the visitor.
- **Region**: Geographic region from which the session originated.
- **TrafficType**: Traffic source (e.g., banner, SMS, direct).
- **VisitorType**: Type of visitor (e.g., new, returning).
- **Weekend**: Boolean indicating whether the visit occurred on a weekend.

🎯 Response Variable

- **Revenue**: A binary variable (`True` or `False`) indicating whether the session resulted in a purchase.

📊 Importance of Analysis

One of the most important patterns to explore in the **Online Shoppers Purchasing Intention Dataset** is how various user session features are associated with the **likelihood of a purchase**.

🔹 Understanding **time spent on product pages**, **traffic source category**, and **user region** can provide valuable insights.
🔹 These insights can help businesses **increase sales** by identifying key **conversion factors**.
🔹 By analyzing trends, companies can **enhance marketing strategies** and focus on elements that **maximize revenue**.

📚 Step 1: Importing Necessary Libraries

Before we begin data processing and model training, we **import all necessary libraries** that will be used throughout the project.

In [218…

```
#importing libraries

from datasets import load_dataset
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, train_test_split, learning_curve
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
```

🛠️ Step 2: Loading and Splitting the Data

In this step, we **load the dataset** and **split it** into training and test sets.

1️⃣ Loading the Dataset

- We use the `load_dataset()` function from the **datasets** package to load the `"jlh/uci-shopper"` dataset from the **Hugging Face dataset repository**.

2️⃣ Splitting the Data

- We use `train_test_split()` from the **sklearn.model_selection** package to create an **80/20 train-test split**, ensuring that **80%** of the data is used for **training** and **20%** for **testing**.

In [219…

```
# Load dataset
ds = load_dataset("jlh/uci-shopper")
df = ds["train"].to_pandas()

# Display the first 5 rows of the dataset with a clear label
print("----------------------------------------------------")
print("Displaying the first 5 rows of the loaded dataset:")
print("----------------------------------------------------")
print(df.head())
print("----------------------------------------------------\n")

# Separate features and target variable
X = df.drop(columns=["Revenue"])
y = df["Revenue"]

# # Limit data for local testing
# X = X.sample(n=1000)
# y = y.loc[X.index]

# Train-test split (80% for training, 20% for testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Display the sizes of training and test sets
print("----------------------------------------------------")
print("Dataset Split Information:")
print(f"Training set size: {X_train.shape[0]} samples")
print(f"Test set size: {X_test.shape[0]} samples")
print("----------------------------------------------------\n")
```

```
Using the latest cached version of the dataset since jlh/uci-shopper couldn't be fou
nd on the Hugging Face Hub
Found the latest cached dataset configuration 'default' at /home/mastrmatt/.cache/hu
ggingface/datasets/jlh___uci-shopper/default/0.0.0/43b926e68ec6e91d3d5d10b4469370201
17667e3 (last modified on Thu Feb 27 17:43:04 2025).
```

```
---------------------------------------------------
Displaying the first 5 rows of the loaded dataset:
---------------------------------------------------
   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        0
1        2       1            2  Returning_Visitor    False        0
2        1       9            3  Returning_Visitor    False        0
3        2       2            4  Returning_Visitor    False        0
4        3       1            4  Returning_Visitor     True        0
---------------------------------------------------

---------------------------------------------------
Dataset Split Information:
Training set size: 9864 samples
Test set size: 2466 samples
---------------------------------------------------
```

📊 Visualizing the Data

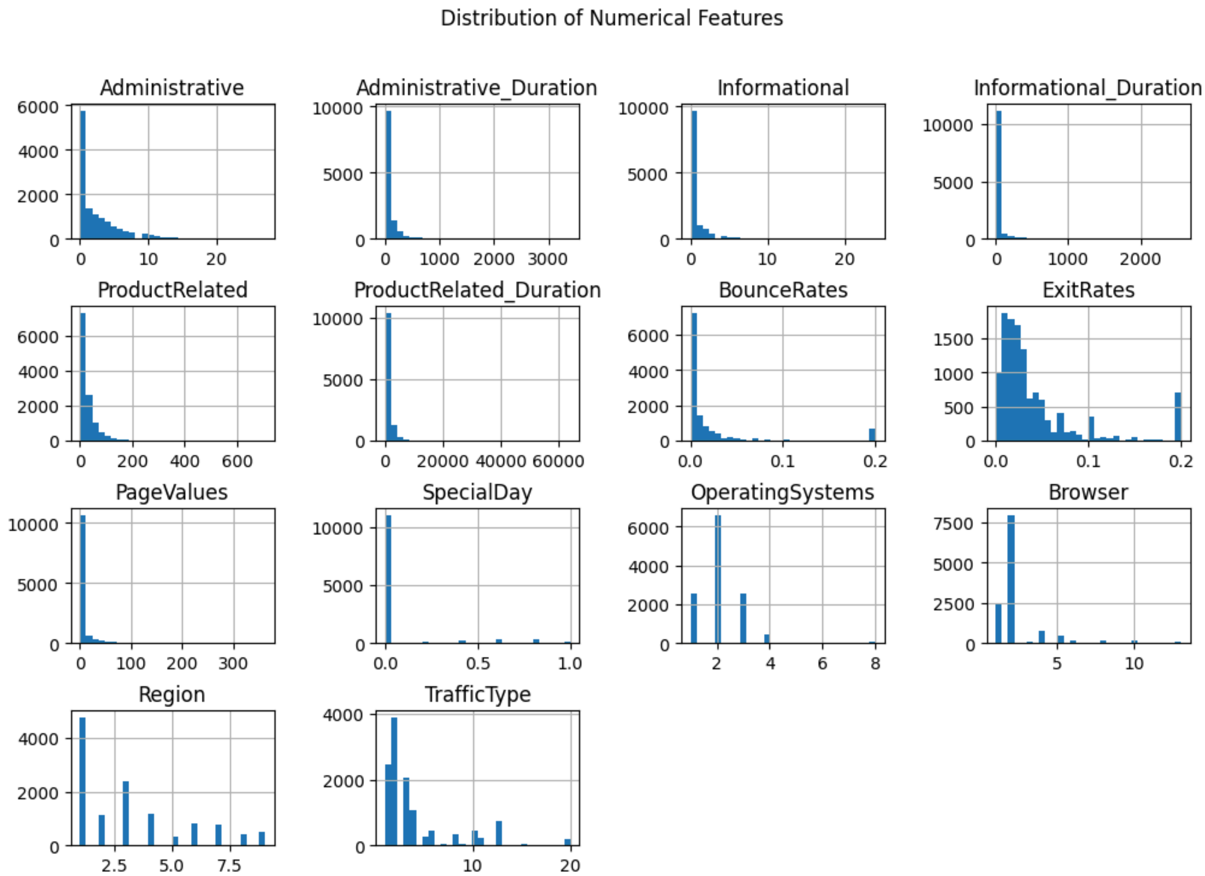
In this step, we'll focus on visualizing the data to better understand its structure and distribution. Visualizations can help identify patterns, trends, or anomalies that might not be obvious from the raw data alone.

Bar Plot for the numerical features

In [220…

```
# Plot distribution for numerical features
numerical_features = X.select_dtypes(include=["float64", "int64"]).columns
X[numerical_features].hist(figsize=(12, 8), bins=30)
plt.suptitle('Distribution of Numerical Features')
```

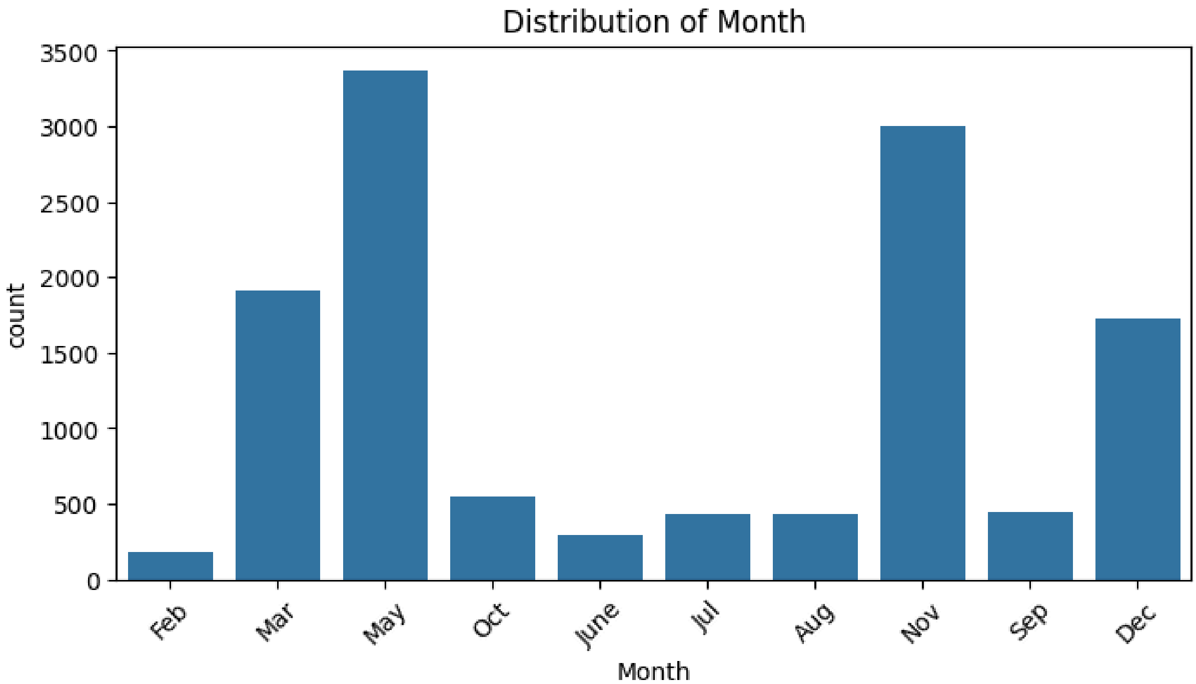
```
plt.subplots_adjust(hspace=0.5, wspace=0.5)  # Adjust the space between plots
plt.show()
```

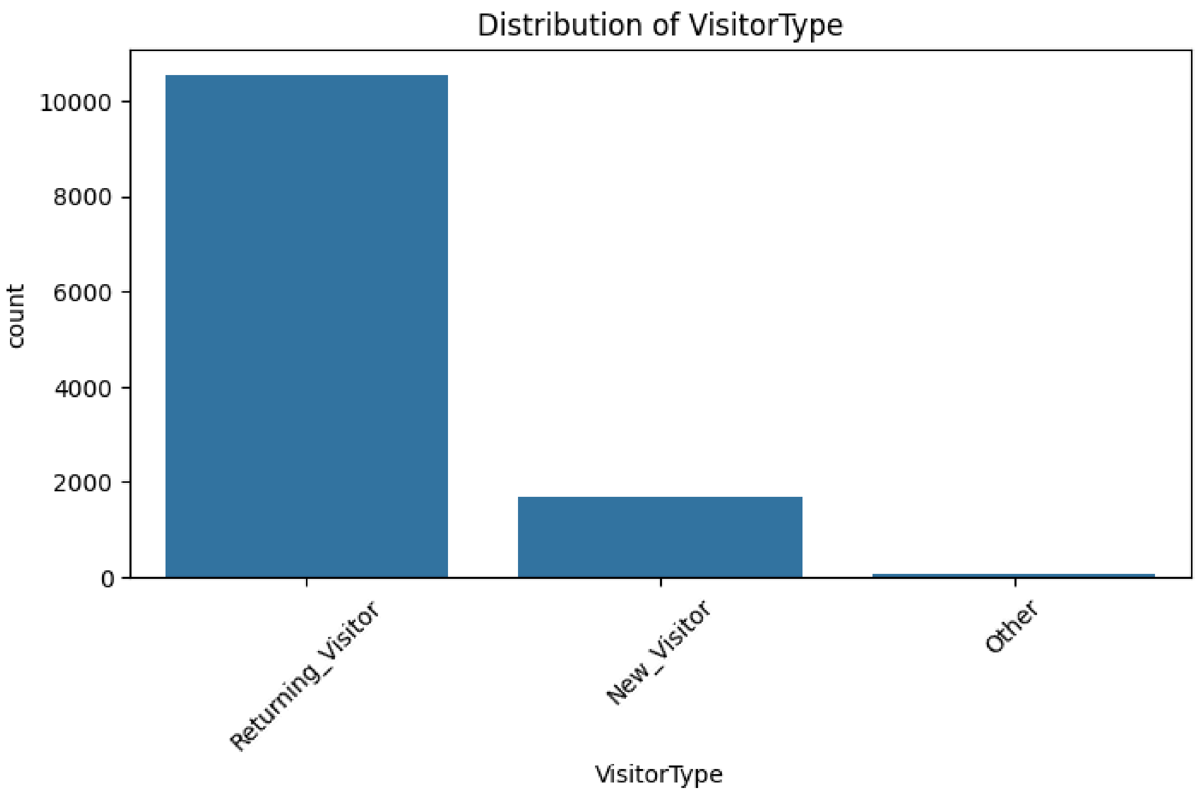


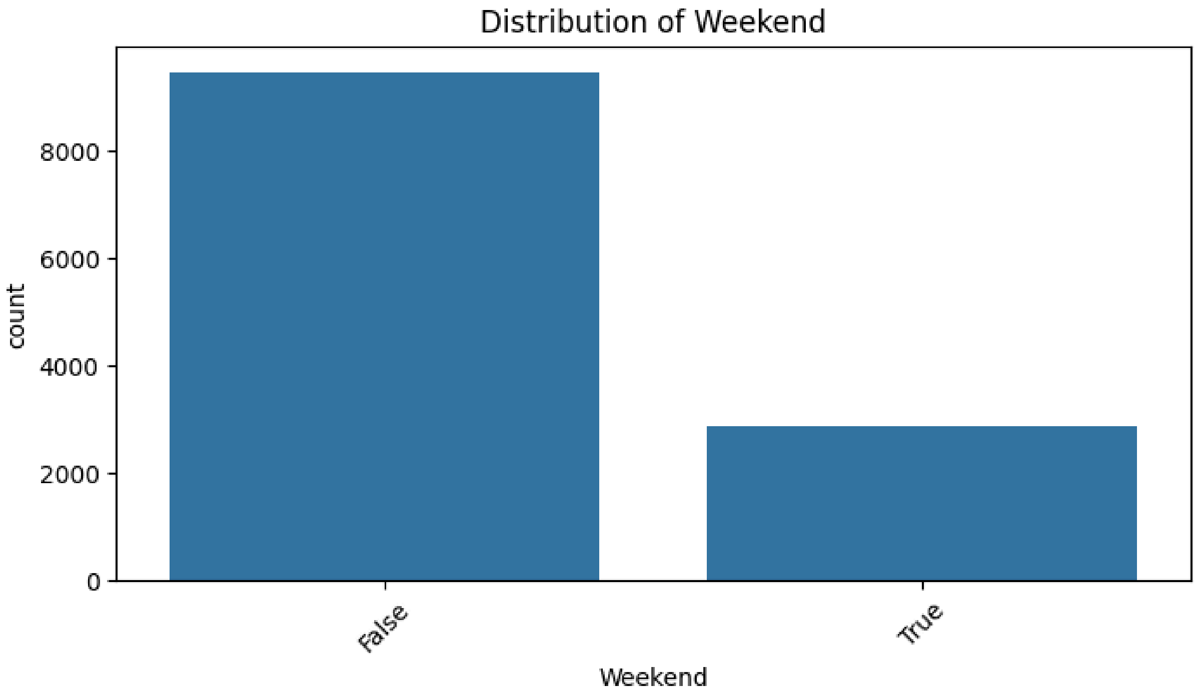
Bar Plot for the categorical features

In [221…

```
# Plot bar plot for categorical features
categorical_features = X.select_dtypes(include=["object", "bool"]).columns
for feature in categorical_features:
    plt.figure(figsize=(8, 4))
    sns.countplot(x=feature, data=X)
    plt.title(f'Distribution of {feature}')
    plt.xticks(rotation=45)
    plt.show()
```


Distribution of Month
count
3500
3000
2500
2000
1500
1000
500
0
Feb
Mar
May
Oct
June
Jul
Aug
Nov
Sep
Dec
Month

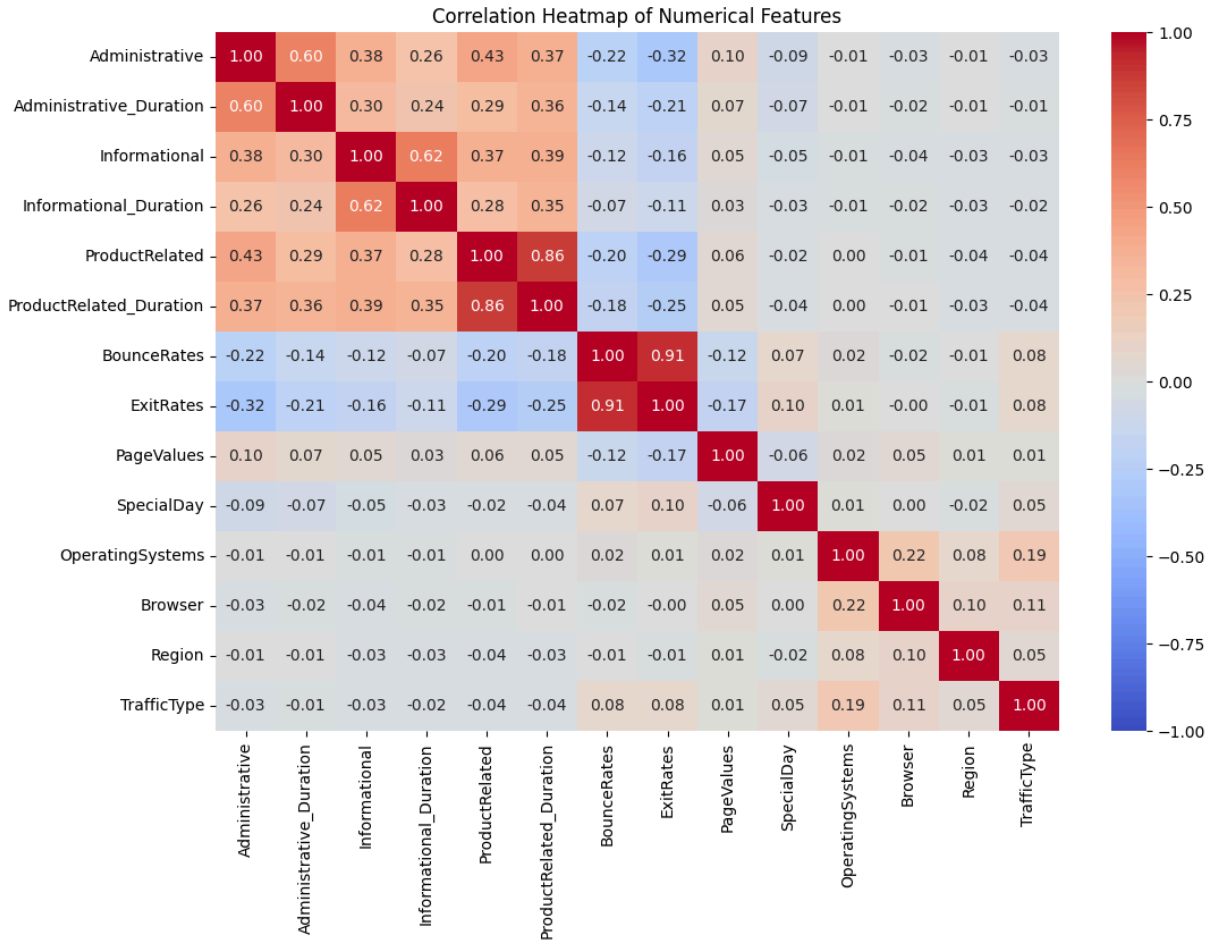

Distribution of VisitorType
count
10000
8000
6000
4000
2000
0
Returning_Visitor
New_Visitor
Other
VisitorType



Correlation Heatmap for the featutes

In [222…

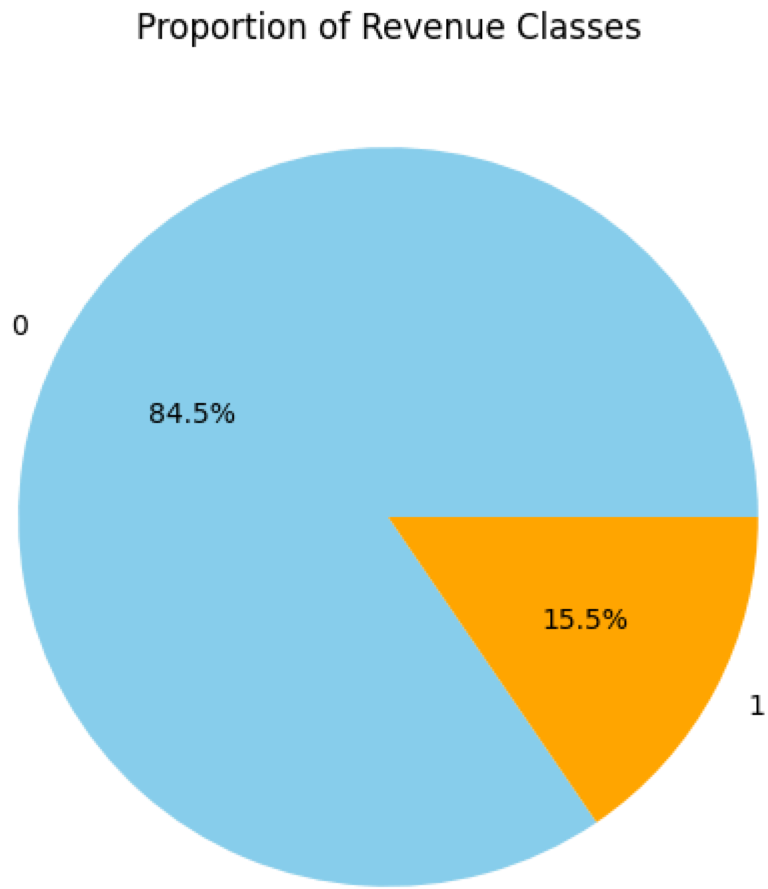
```
corr_matrix = X[numerical_features].corr()
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', vmin=-1, vmax=1)
plt.title('Correlation Heatmap of Numerical Features')
plt.show()
```



Distribution of the revenue class

In [223…

```
y.value_counts().plot(kind='pie', autopct='%1.1f%%', colors=["skyblue", "orange"],
plt.title("Proportion of Revenue Classes")
plt.ylabel("")  # Hide y-label
plt.show()
```



🛠️ Step 3: Data Preprocessing

We need to preprocess both numerical and categorical features before passing them to the model.

- The `SimpleImputer()` function from the `sklearn.impute` package is used to fill in missing values.
  - `SimpleImputer(strategy="mean")` fills missing numerical features with the **mean**.
  - `SimpleImputer(strategy='constant', fill_value='missing')` fills missing categorical features with the value **"missing"**.
- The numerical features are then **scaled** using `StandardScaler()` from the `sklearn.preprocessing` package.
  - This standardizes numerical features by subtracting the mean and dividing by the standard deviation (**z-score normalization**).
  - Standardization improves the performance of models like **Support Vector Machines (SVM)** that are sensitive to feature scale.

- The `OneHotEncoder()` function from `sklearn.preprocessing` is used to encode categorical features.
  - It converts categorical features into **binary (0 or 1) columns**, creating **n new features** per categorical variable, where **n** is the number of unique categories.

At the end of preprocessing, the dataset will now contain **29 features**.

In [224…

```
# Preprocess numerical features
numerical_features = X.select_dtypes(include=["float64", "int64"]).columns

#Impute and scale numerical features
numerical_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="mean")),
    ("scaler", StandardScaler())
])

# Preprocess categorical features
categorical_features = X.select_dtypes(include=["object", "bool"]).columns

#Impute and one-hot encode categorical features
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),  # Handl
    ('onehot', OneHotEncoder(handle_unknown='ignore'))  # One-hot encode categorica
])

# Combine numerical and categorical transformers
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ]
)

# Fit the preprocessor and transform the data
X_transformed = preprocessor.fit_transform(X)

# Extract transformed column names
# 1. Numerical feature names remain the same
num_feature_names = list(numerical_features)

# 2. Get new feature names from OneHotEncoder
cat_feature_names = preprocessor.named_transformers_['cat'].named_steps['onehot'].g

# 3. Combine numerical and categorical feature names
new_column_names = num_feature_names + list(cat_feature_names)

# Convert transformed array back into DataFrame
X_transformed_df = pd.DataFrame(X_transformed, columns=new_column_names)

# Display first 5 rows of the transformed dataset
print("----------------------------------------------------")
print("First 5 rows of the preprocessed dataset:")
print("----------------------------------------------------")
```

```
print(X_transformed_df.head())
print("--------------------------------------------------\n")
```

```
--------------------------------------------------
First 5 rows of the preprocessed dataset:
--------------------------------------------------
   Administrative  Administrative_Duration  Informational  \
0       -0.696993                -0.457191      -0.396478   
1       -0.696993                -0.457191      -0.396478   
2       -0.696993                -0.457191      -0.396478   
3       -0.696993                -0.457191      -0.396478   
4       -0.696993                -0.457191      -0.396478   

   Informational_Duration  ProductRelated  ProductRelated_Duration  \
0               -0.244931       -0.691003                -0.624348   
1               -0.244931       -0.668518                -0.590903   
2               -0.244931       -0.691003                -0.624348   
3               -0.244931       -0.668518                -0.622954   
4               -0.244931       -0.488636                -0.296430   

   BounceRates  ExitRates  PageValues  SpecialDay  ...  Month_Mar  Month_May  \
0     3.667189   3.229316   -0.317178   -0.308821  ...        0.0        0.0   
1    -0.457683   1.171473   -0.317178   -0.308821  ...        0.0        0.0   
2     3.667189   3.229316   -0.317178   -0.308821  ...        0.0        0.0   
3     0.573535   1.994610   -0.317178   -0.308821  ...        0.0        0.0   
4    -0.045196   0.142551   -0.317178   -0.308821  ...        0.0        0.0   

   Month_Nov  Month_Oct  Month_Sep  VisitorType_New_Visitor  \
0        0.0        0.0        0.0                      0.0   
1        0.0        0.0        0.0                      0.0   
2        0.0        0.0        0.0                      0.0   
3        0.0        0.0        0.0                      0.0   
4        0.0        0.0        0.0                      0.0   

   VisitorType_Other  VisitorType_Returning_Visitor  Weekend_False  \
0                0.0                            1.0            1.0   
1                0.0                            1.0            1.0   
2                0.0                            1.0            1.0   
3                0.0                            1.0            1.0   
4                0.0                            1.0            0.0   

   Weekend_True  
0           0.0  
1           0.0  
2           0.0  
3           0.0  
4           1.0  

[5 rows x 29 columns]
--------------------------------------------------
```

🏗 Step 4: Model Construction

An SVM classifier with a polynomial kernel was used. 5-fold internal cross-validation was performed to determine the best (cost, degree) pair for the SVM classifier.

- The metric during internal cross-validation was the accuracy of the classifier.
- The degrees used were: `[2, 3, 4, 7]`.
- The costs used were: `[0.1, 10, 100, 1000]`.

In [225…

```
# Model construction
model = Pipeline(steps=[
    ("preprocessor", preprocessor),
    ("classifier", SVC(kernel="poly"))
])

# hyperparameter grid, used during cross-validation
param_grid = {
    "classifier__degree": [2, 3, 4, 7 ],
    "classifier__C": [0.1, 10, 100, 1000],
}
```

🚀 Step 5: Model Training

Next, we'll use the `GridSearchCV().fit()` function from the `sklearn.model_selection` package to find the best hyperparameters for the model and train it. This function takes the previously defined `param_grid` (the hyperparameter grid) as an argument and fits a SVM with a polynomial kernel corresponding to the respective hyperparameters.

The best parameters obtained are saved, and the best accuracy achieved is also saved

In [226…

```
# Set up GridSearchCV with the pipeline, parameter grid, and cross-validation strat
# n_jobs = -2 to use all available CPU cores except one
grid_search = GridSearchCV(model, param_grid, cv=5 , scoring='accuracy', n_jobs=-2)

# Fit the grid search to the data, which will automatically run cross-validation an
grid_search.fit(X_train, y_train)

# Retrieve the best parameters and the corresponding score
best_params = grid_search.best_params_
best_score = grid_search.best_score_


print("----------------------------------------------------")
print(f"Best SVM Cost: {best_params['classifier__C']}")
print(f"Best Polynomial Kernel Degree: {best_params['classifier__degree']}")
print("----------------------------------------------------")
print(f'Training Accuracy: {best_score:.4f}')
print("----------------------------------------------------\n")
```

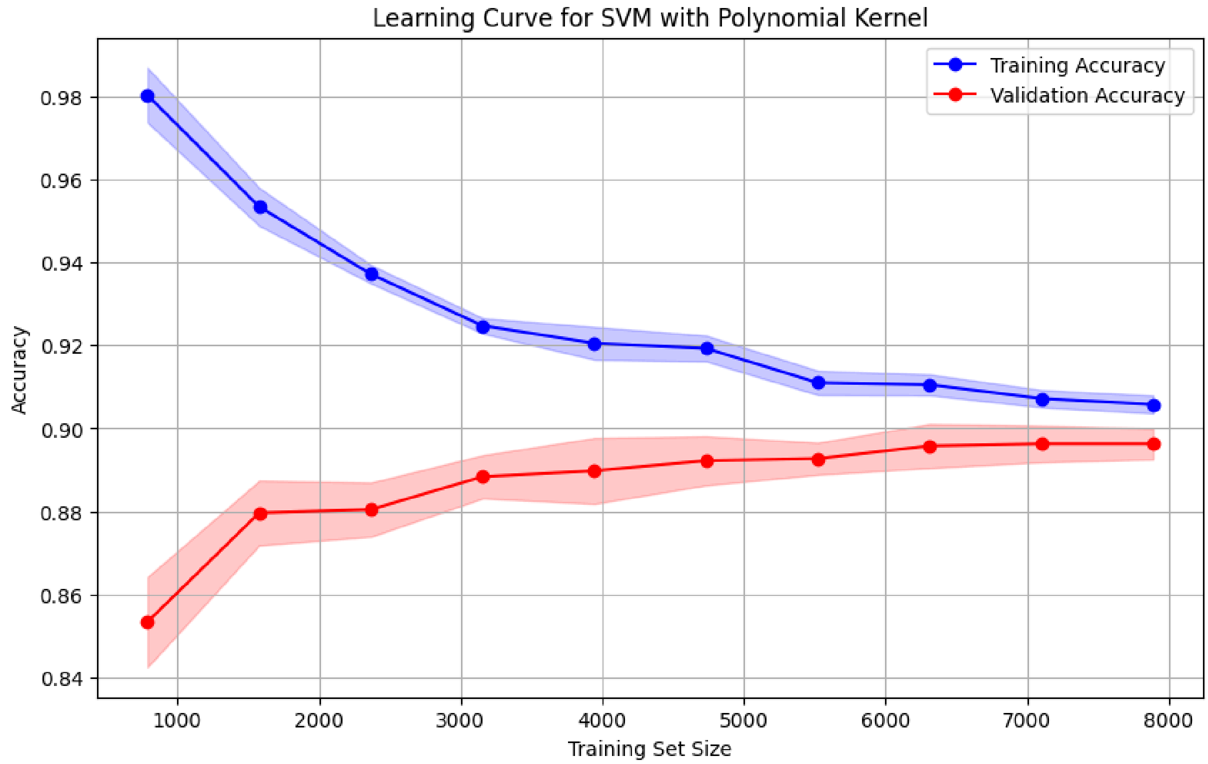
```
--------------------------------------------------
Best SVM Cost: 100
Best Polynomial Kernel Degree: 2
--------------------------------------------------
Training Accuracy: 0.8963
--------------------------------------------------
```

Learning Curve for different sample sizes

The shaded area in the learning curve represents the standard deviation of accuracy scores across different cross-validation folds from the mean accuracy score

In [227…

```
# Define range of training sizes to test
train_sizes = np.linspace(0.1, 1.0, 10)  # 10 different sizes from 10% to 100% of t

# Compute learning curve
train_sizes_abs, train_scores, val_scores = learning_curve(
    grid_search.best_estimator_,  # Use the best model from GridSearchCV
    X_train, y_train,
    train_sizes=train_sizes,
    cv=5,  # 5-fold cross-validation
    scoring="accuracy",
    n_jobs=-2  # Use all available CPU cores except one
)

# Compute mean and standard deviation for smooth curves
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
val_mean = np.mean(val_scores, axis=1)
val_std = np.std(val_scores, axis=1)

# Plot the learning curve
plt.figure(figsize=(10, 6))
plt.plot(train_sizes_abs, train_mean, "o-", color="blue", label="Training Accuracy"
plt.plot(train_sizes_abs, val_mean, "o-", color="red", label="Validation Accuracy")

# Add shaded area for standard deviation
plt.fill_between(train_sizes_abs, train_mean - train_std, train_mean + train_std, a
plt.fill_between(train_sizes_abs, val_mean - val_std, val_mean + val_std, alpha=0.2

# Labels and title
plt.xlabel("Training Set Size")
plt.ylabel("Accuracy")
plt.title("Learning Curve for SVM with Polynomial Kernel")
plt.legend()
plt.grid()
plt.show()
```



Step 6: Evaluate Model on Test Set

The predict() function from the sklearn.model_selection package was used to generate the predictions for the test data samples based on the trained model. It takes the test data set features as input and return thier corresponding predictions as output. accuracy_score() from the sklearn.metrics package was then used to compute the overall accuary of the model on the testing data set

In [228…

```
# Evaluate the model on the test set
y_pred = grid_search.predict(X_test)
test_score = accuracy_score(y_test, y_pred)

print("----------------------------------------------------")
print(f'Test Accuracy: {test_score:.4f}')
print("----------------------------------------------------\n")

# Print the first 5 test samples with their corresponding predicted values, includi
import pandas as pd

# Assuming X_test is a DataFrame and y_test is the true labels
test_samples_with_preds = pd.DataFrame(X_test[:5])  # Get the first 5 test samples
test_samples_with_preds['Predicted'] = y_pred[:5]  # Add predicted values

# Reset the index and include the index as a column
test_samples_with_preds.reset_index(drop=True, inplace=True)

print("----------------------------------------------------")
print("Displaying the first 5 test samples with their corresponding predicted value
print("----------------------------------------------------\n")
```

```
print(test_samples_with_preds)
print("--------------------------------------------------\n")
```

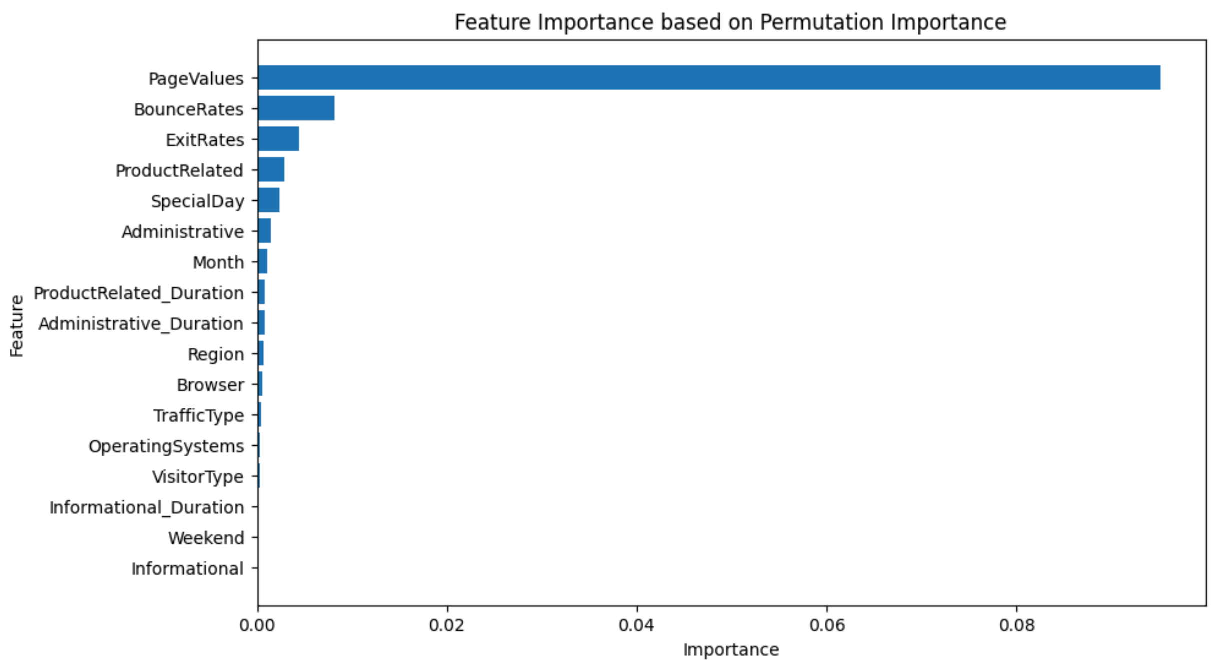
```
--------------------------------------------------
Test Accuracy: 0.8820
--------------------------------------------------

--------------------------------------------------
Displaying the first 5 test samples with their corresponding predicted values:
--------------------------------------------------

   Administrative  Administrative_Duration  Informational  \
0               3               142.500000              0
1               6               437.391304              2
2               1                41.125000              0
3               2               141.000000              0
4              18               608.140000              6

   Informational_Duration  ProductRelated  ProductRelated_Duration  \
0                    0.00              48              1052.255952
1                  235.55              83              2503.881781
2                    0.00             126              4310.004668
3                    0.00              10               606.666667
4                  733.80             168              4948.398759

   BounceRates  ExitRates  PageValues  SpecialDay Month  OperatingSystems  \
0     0.004348   0.013043    0.000000         0.0   Nov                 1
1     0.002198   0.004916    2.086218         0.0   Mar                 2
2     0.000688   0.012823    3.451072         0.0   Nov                 2
3     0.008333   0.026389   36.672294         0.0   Aug                 2
4     0.006632   0.013528   10.150644         0.0   Aug                 2

   Browser  Region  TrafficType        VisitorType  Weekend  Predicted
0        8       6           11  Returning_Visitor    False          0
1        2       3            2  Returning_Visitor    False          0
2        2       2            2  Returning_Visitor    False          0
3        5       7            4  Returning_Visitor    False          1
4        2       3            1  Returning_Visitor     True          0
--------------------------------------------------
```

Now, randomly shuffling every feature and calculating how the accuracy of the model changes based on each feature. This is called permutation importance and is used to see which features are the most impactful. This is done with the `permutation_importance()` function from the `sklearn.inspection` package. This function takes in the testing data and performs permutation importance on it. A figure ranking thier importance is then displayed

In [229…

```
from sklearn.inspection import permutation_importance
import pandas as pd

# Compute feature importance
result = permutation_importance(grid_search, X_test, y_test, scoring="accuracy", n_
```

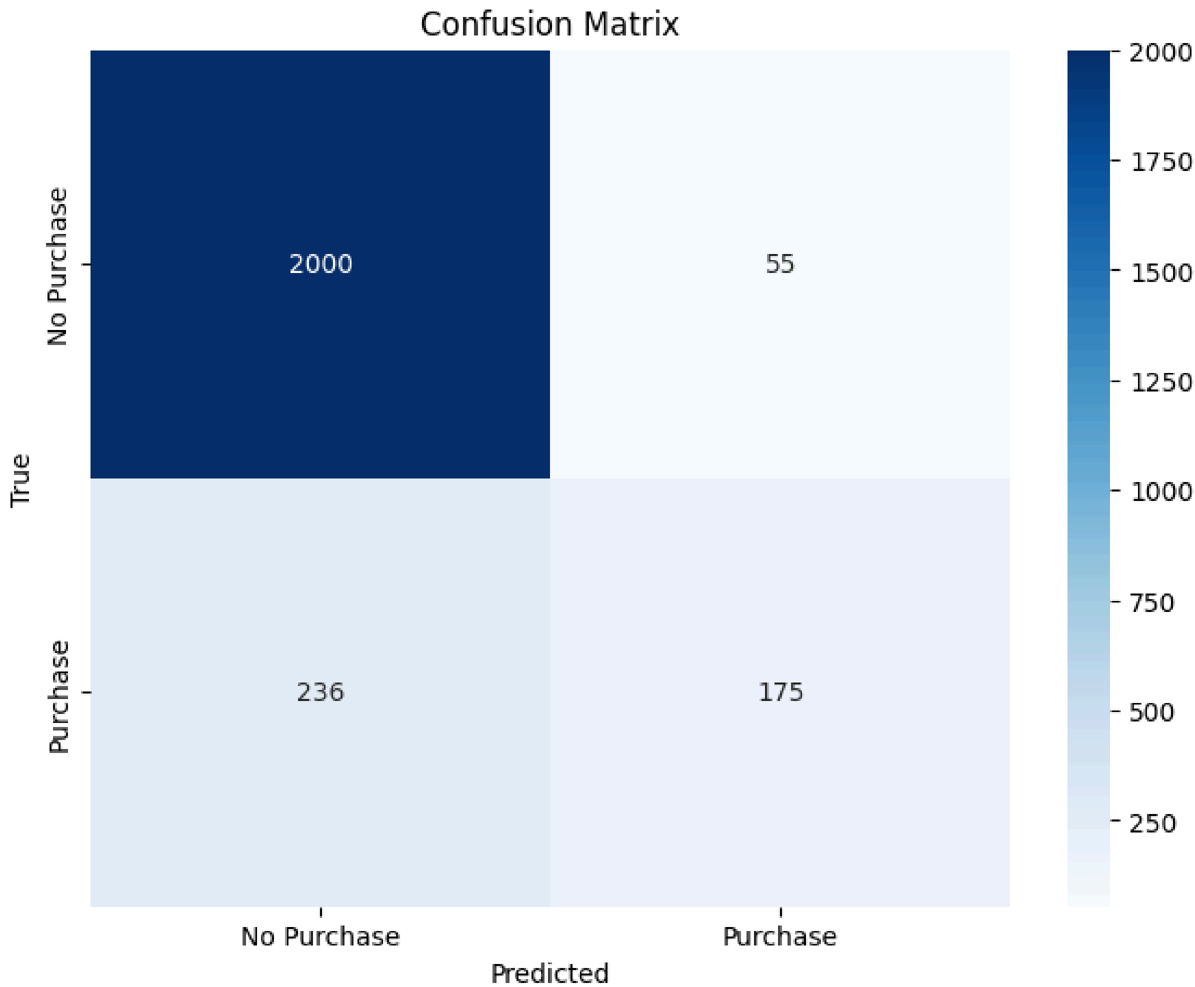
```
# Sort and display the feature importance
feature_importance = pd.DataFrame({
    "Feature": X.columns,
    "Importance": abs(result.importances_mean)
}).sort_values(by="Importance", ascending=False)

# Assuming `feature_importance` is already a DataFrame with 'Feature' and 'Importan
plt.figure(figsize=(10, 6))
plt.barh(feature_importance['Feature'], feature_importance['Importance'])
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importance based on Permutation Importance')
plt.gca().invert_yaxis()  # Invert y-axis to have the most important feature at the
plt.show()
```



Confusion Matrix

The `confusion_matrix()` function from the `sklearn.metrics` is then used to compute the confusion matrix for the test dataset. It takes as input the true class labels and the predicted class labels from the testing dataset. The results are then displayed in a figure

In [230…
```
from sklearn.metrics import confusion_matrix

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Purchase', 'Pur
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```



Data Analysis and Conclusion

After hyperparameter tuning with 5-fold cross-validation, the model achieved:

- Best SVM Cost parameter (C): 100
- Best Polynomial Kernel Degree: 2
- Training Accuracy: 89.63%
- Testing Accuracy: 88.20%

Key insights from the analysis:

1. **PageValues** emerged as the strongest predictor of purchasing behavior. This suggests that e-commerce websites should focus on increasing the value of content on each page to drive conversions.

2. Session quality metrics (bounce and exit rates) proved more valuable in predicting revenue than visitor demographics.

3. Technical attributes (Browser, Operating System) showed little correlation with purchasing decisions.

The confusion matrix indicates that the model performs well at identifying both purchase and non-purchase sessions, though there is room for improvement in reducing false negatives.

Recommendations for e-commerce businesses:

1. Implement methods to increase customer engagement with every page
2. Remove or redesign web pages with low customer engagement
3. Focus optimization efforts on improving page values rather than targeting specific browsers or operating systems
4. Monitor and improve metrics related to session quality (reducing bounce rates, exit rates)

This analysis provides actionable insights for e-commerce website optimization that can lead to in