American International University-Bangladesh

CSC 4264: Programming in Python

Project Report

Fall 2024-25

Project Title: Telco Customer Churn Analysis

Section: A

Group Number: 1

Student Name	Student Id
Yeasir Ahnaf Asif	20-42815-1
A. M. Rafinul Huq	21 -45668-3

Dataset Description

The "Telco Customer Churn" dataset contains information about customers of a telecom company. It aims to predict customer churn, which refers to customers leaving the company's services. The dataset includes demographic, service usage, and account details. Key attributes include:

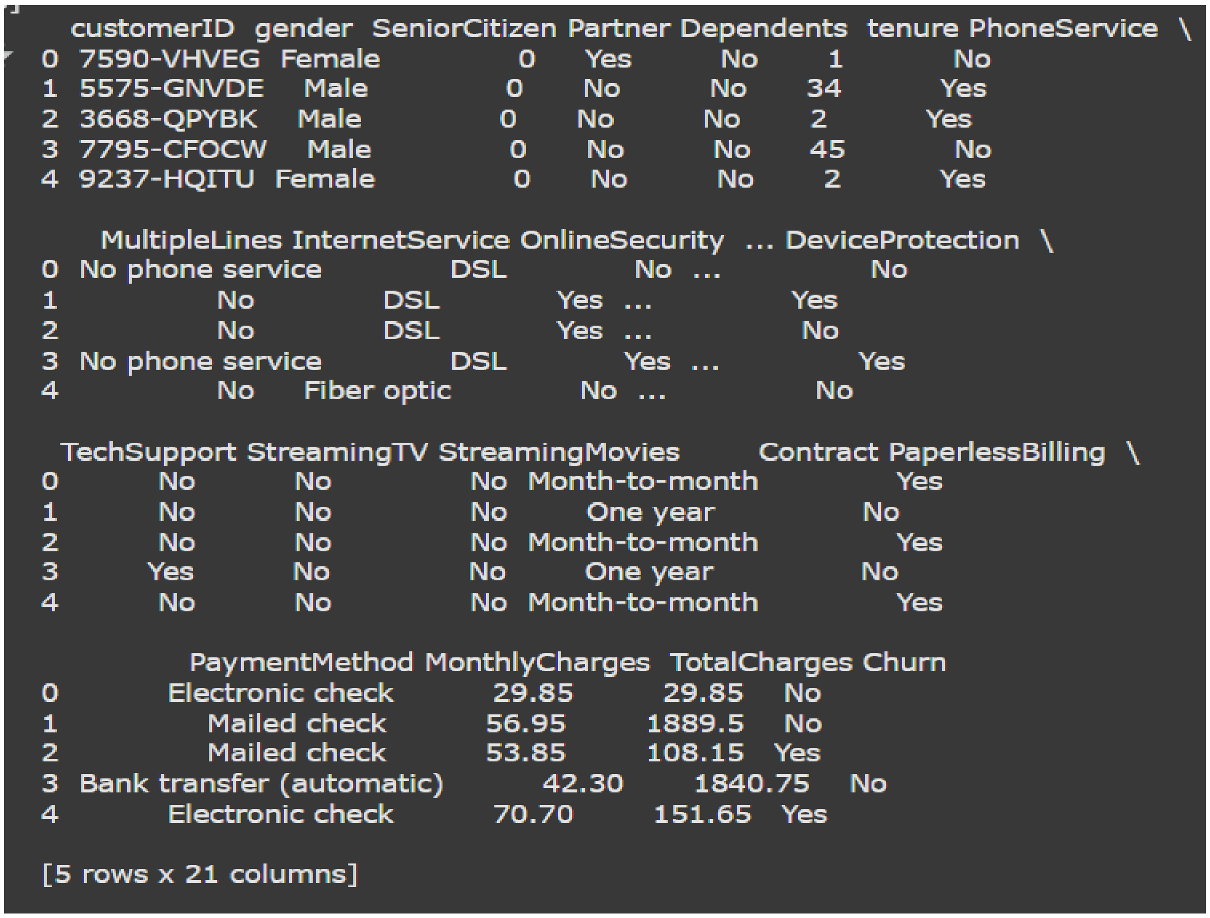
- **Demographics**: `gender`, `SeniorCitizen`, `Partner`, `Dependents`
- **Service details**: `tenure`, `PhoneService`, `MultipleLines`, `InternetService`, `OnlineSecurity`, `OnlineBackup`, `DeviceProtection`, `TechSupport`, `StreamingTV`, `StreamingMovies`
- **Contract details**: `Contract`, `PaperlessBilling`, `PaymentMethod`, `MonthlyCharges`, `TotalCharges`
- **Target variable**: `Churn` (whether a customer churned or not)

The dataset contains 7,043 records and 21 columns, with no missing values except for some entries in the `TotalCharges` column, which were handled during preprocessing.

Tasks and Solutions

Task 1: Load the Dataset

Using the Pandas library, the dataset was loaded into a DataFrame. An initial inspection revealed the presence of numeric, categorical, and object data types. The **`TotalCharges`** column was found to contain invalid entries (likely spaces) that needed cleaning.



```
   customerID  gender  SeniorCitizen Partner Dependents  tenure PhoneService  \
0  7590-VHVEG  Female              0     Yes         No       1           No
1  5575-GNVDE    Male              0      No         No      34          Yes
2  3668-QPYBK    Male              0      No         No       2          Yes
3  7795-CFOCW    Male              0      No         No      45           No
4  9237-HQITU  Female              0      No         No       2          Yes

      MultipleLines InternetService OnlineSecurity  ... DeviceProtection  \
0  No phone service             DSL             No  ...               No
1                No             DSL            Yes  ...              Yes
2                No             DSL            Yes  ...               No
3  No phone service             DSL            Yes  ...              Yes
4                No     Fiber optic             No  ...               No

  TechSupport StreamingTV StreamingMovies        Contract PaperlessBilling  \
0          No          No              No  Month-to-month              Yes
1          No          No              No        One year               No
2          No          No              No  Month-to-month              Yes
3         Yes          No              No        One year               No
4          No          No              No  Month-to-month              Yes

               PaymentMethod MonthlyCharges  TotalCharges Churn
0           Electronic check          29.85         29.85    No
1               Mailed check          56.95        1889.5    No
2               Mailed check          53.85        108.15   Yes
3  Bank transfer (automatic)          42.30       1840.75    No
4           Electronic check          70.70        151.65   Yes

[5 rows x 21 columns]
```

Task 2: Data Cleaning

i. **Handling Missing Values**:

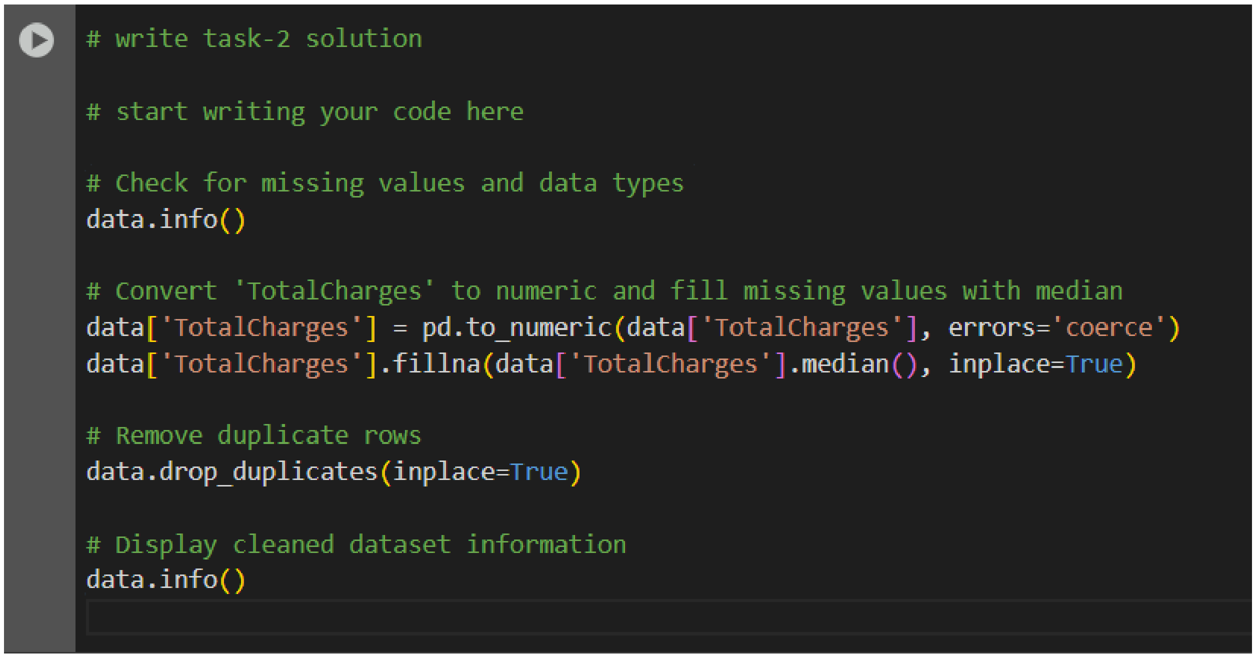
The `TotalCharges` column was converted from `object` to `float`. Any rows with invalid values (e.g., empty strings) were identified and dropped.

ii. **Encoding Categorical Features**:

One-hot encoding was applied to all categorical features to transform them into a format suitable for machine learning models.

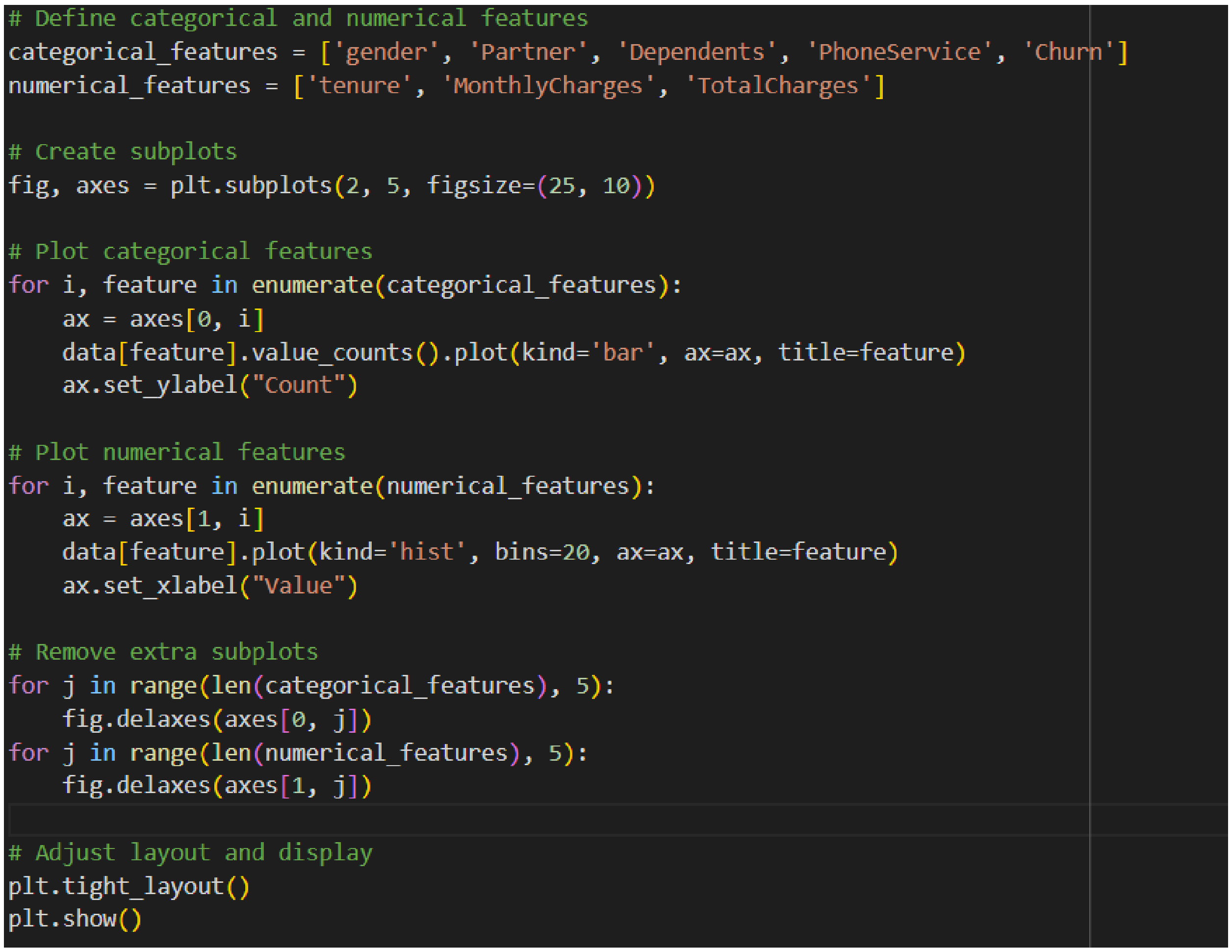
iii. **Verification of Clean Data**:

After cleaning, the dataset was confirmed to have no missing values. Features like `customerID` were dropped as they are irrelevant for modeling.


```
# write task-2 solution

# start writing your code here

# Check for missing values and data types
data.info()

# Convert 'TotalCharges' to numeric and fill missing values with median
data['TotalCharges'] = pd.to_numeric(data['TotalCharges'], errors='coerce')
data['TotalCharges'].fillna(data['TotalCharges'].median(), inplace=True)

# Remove duplicate rows
data.drop_duplicates(inplace=True)

# Display cleaned dataset information
data.info()
```

Task 3: Exploratory Data Analysis (EDA)

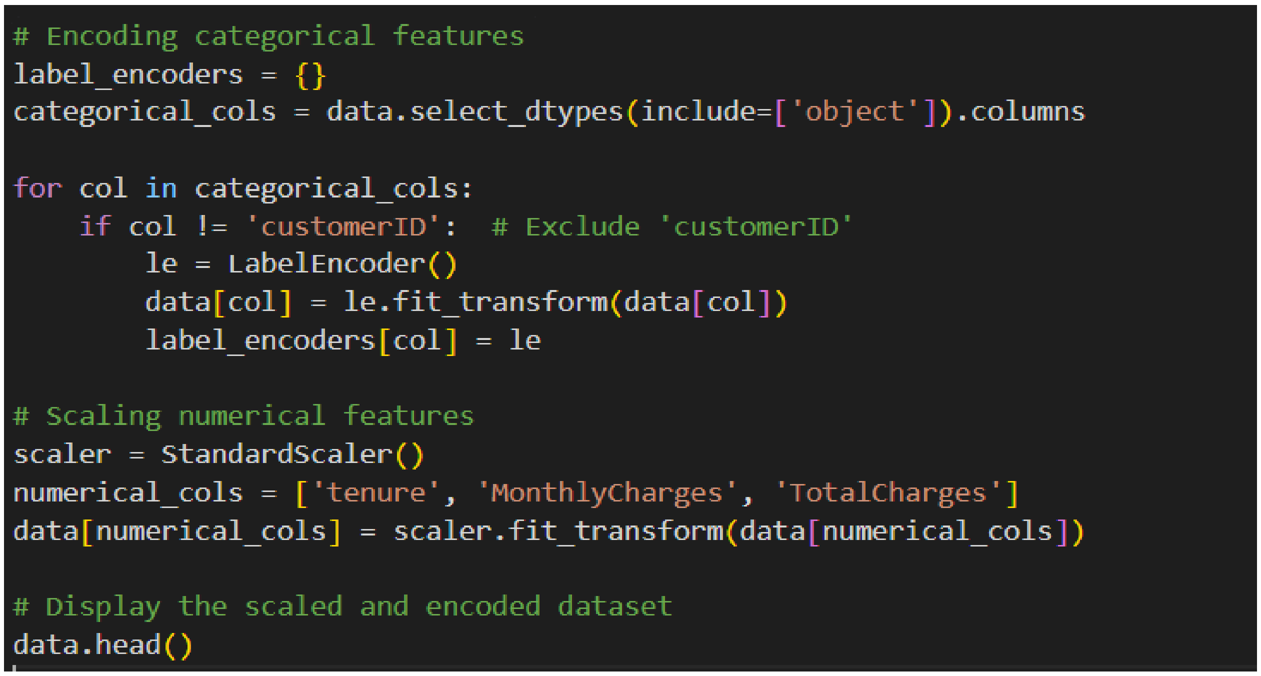
Exploratory Data Analysis (EDA) works by visualizing the distributions of categorical and numerical features in the dataset. It begins by defining two lists: categorical_features for discrete attributes and numerical_features for continuous variables. Using Matplotlib, a grid of subplots is created to display bar charts for categorical features and histograms for numerical features. Bar charts are plotted for each categorical feature, showing the count of each category, while histograms visualize the frequency distribution of numerical features across bins. Any unused subplots in the grid are removed to maintain a clean layout. The plt.tight_layout() function ensures proper spacing between the plots, and the final visualizations are rendered using plt.show(). This structured approach provides a clear and intuitive overview of the dataset's key features and their distributions.



```
# Define categorical and numerical features
categorical_features = ['gender', 'Partner', 'Dependents', 'PhoneService', 'Churn']
numerical_features = ['tenure', 'MonthlyCharges', 'TotalCharges']

# Create subplots
fig, axes = plt.subplots(2, 5, figsize=(25, 10))

# Plot categorical features
for i, feature in enumerate(categorical_features):
    ax = axes[0, i]
    data[feature].value_counts().plot(kind='bar', ax=ax, title=feature)
    ax.set_ylabel("Count")

# Plot numerical features
for i, feature in enumerate(numerical_features):
    ax = axes[1, i]
    data[feature].plot(kind='hist', bins=20, ax=ax, title=feature)
    ax.set_xlabel("Value")

# Remove extra subplots
for j in range(len(categorical_features), 5):
    fig.delaxes(axes[0, j])
for j in range(len(numerical_features), 5):
    fig.delaxes(axes[1, j])

# Adjust layout and display
plt.tight_layout()
plt.show()
```

Task 4: Feature Scaling

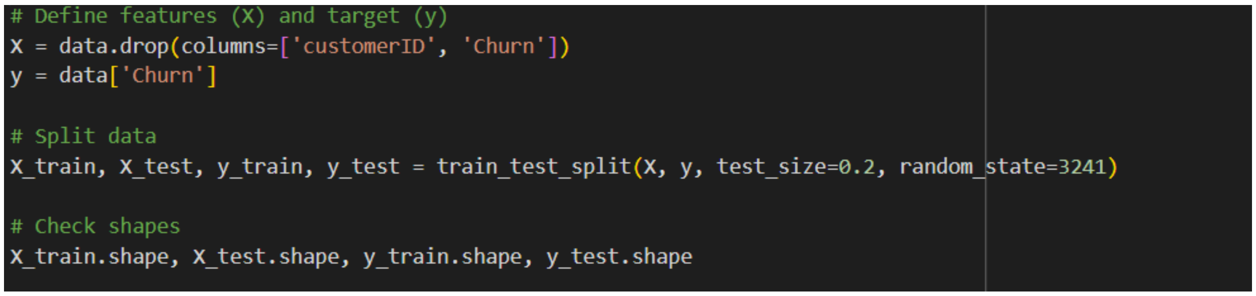
In this section we prepare the dataset for machine learning by encoding categorical features and scaling numerical features. Categorical columns, identified using select_dtypes, are transformed into numerical values using LabelEncoder, except for customerID, which is excluded as it is irrelevant for modeling. The encoders are stored in a dictionary for future use. Numerical columns (tenure, MonthlyCharges, TotalCharges**)** are standardized using StandardScaler to ensure they have a mean of 0 and a standard deviation of 1, which helps balance their contribution to machine learning models. Finally, the processed dataset is displayed using data.head() to verify the transformations, ensuring it is ready for modeling with numerical and scaled data.



```python
# Encoding categorical features
label_encoders = {}
categorical_cols = data.select_dtypes(include=['object']).columns

for col in categorical_cols:
    if col != 'customerID':  # Exclude 'customerID'
        le = LabelEncoder()
        data[col] = le.fit_transform(data[col])
        label_encoders[col] = le

# Scaling numerical features
scaler = StandardScaler()
numerical_cols = ['tenure', 'MonthlyCharges', 'TotalCharges']
data[numerical_cols] = scaler.fit_transform(data[numerical_cols])

# Display the scaled and encoded dataset
data.head()
```

Task 5: Train-Test Split

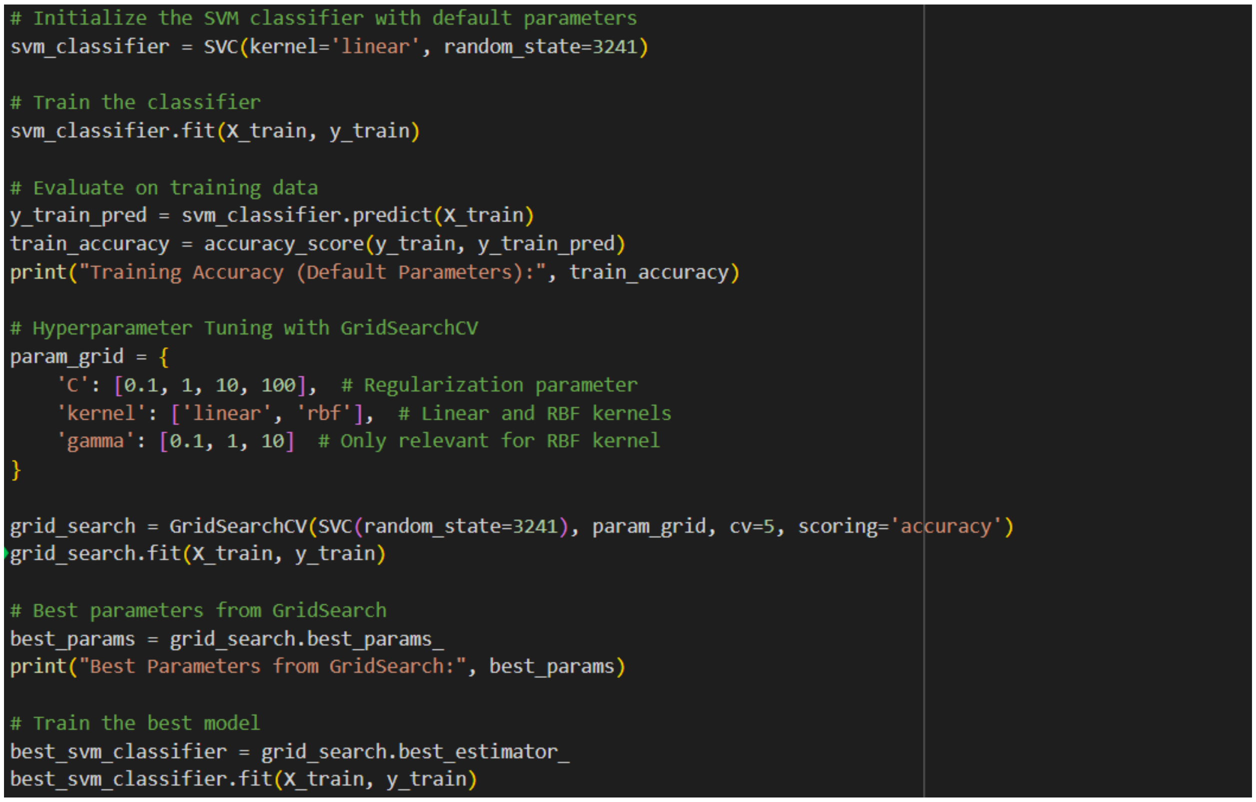
The code splits the dataset into features (X) and the target variable (y) and prepares it for model training and testing. Features (X) are defined by dropping the customerID and Churn columns, while y contains the Churn column as the target. Using train_test_split, the data is divided into training (80%) and testing (20%) sets, with random_state=3241 ensuring reproducibility of the split. The resulting shapes of X_train, X_test, y_train, and y_test are displayed to confirm the correctness of the split, ensuring the data is ready for model training and evaluation.



```python
# Define features (X) and target (y)
X = data.drop(columns=['customerID', 'Churn'])
y = data['Churn']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=3241)

# Check shapes
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Task 6: Model Training

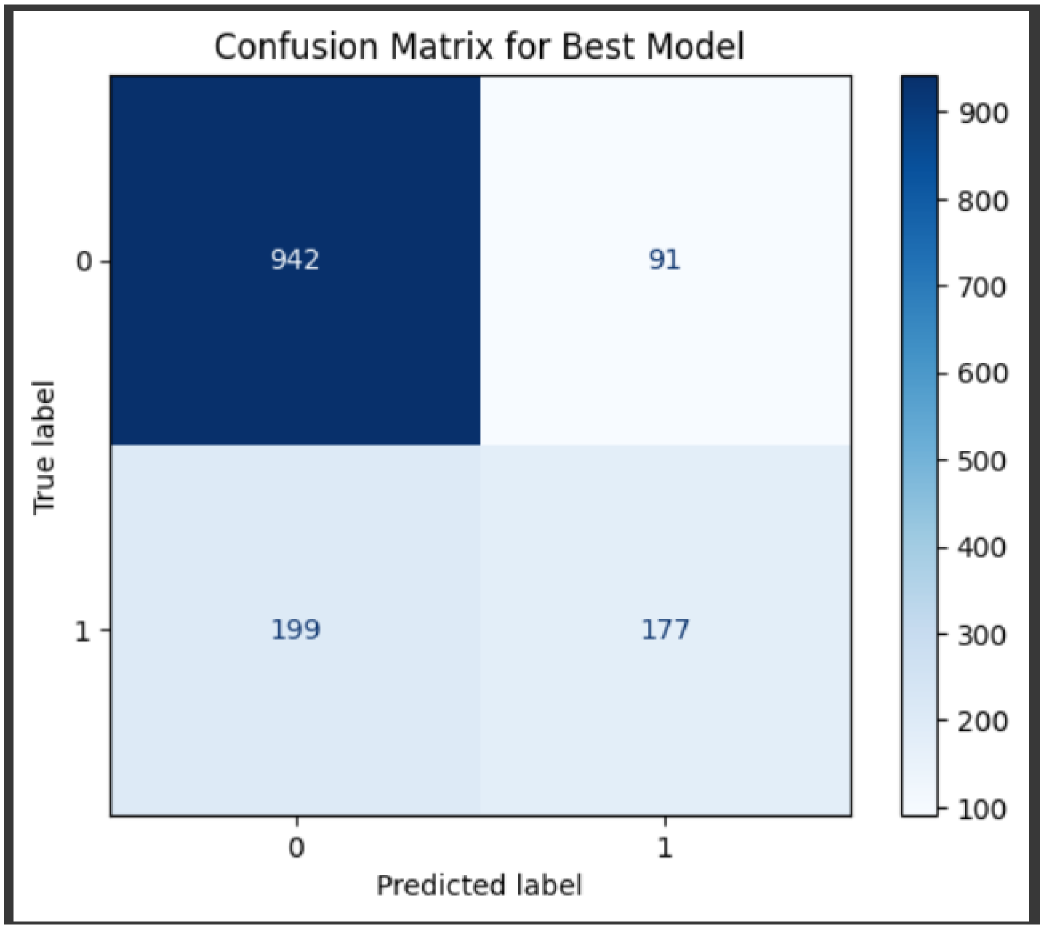
We trains and optimizes an SVM classifier for the dataset by first initializing a default SVM model with a linear kernel (SVC(kernel='linear')) and training it on the training dataset (X_train, y_train). Training accuracy is evaluated by predicting on the training data and comparing the predictions with actual labels using accuracy_score. To improve performance, hyperparameter tuning is performed using GridSearchCV, exploring various values for C (regularization parameter), kernel (linear and RBF kernels), and gamma (RBF kernel coefficient). A 5-fold cross-validation ensures robust evaluation of each hyperparameter combination. The best hyperparameters are identified using grid_search.best_params_, and the corresponding best model (grid_search.best_estimator_) is retrained on the training data, optimizing the SVM classifier for improved performance.

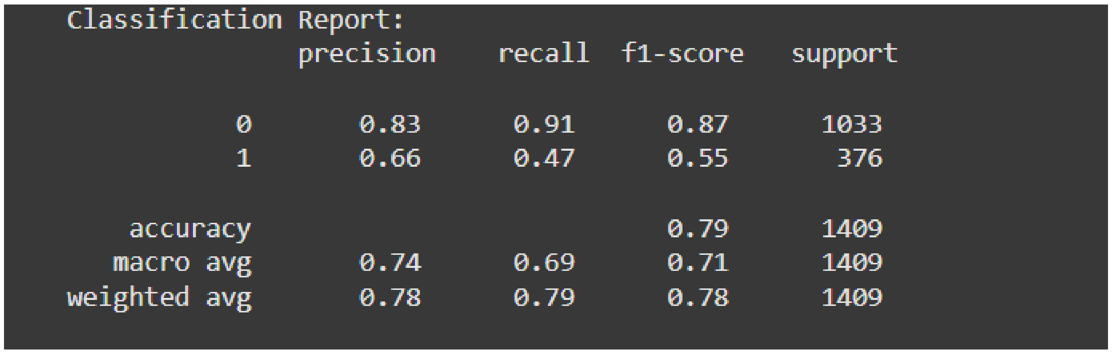


```
# Initialize the SVM classifier with default parameters
svm_classifier = SVC(kernel='linear', random_state=3241)

# Train the classifier
svm_classifier.fit(X_train, y_train)

# Evaluate on training data
y_train_pred = svm_classifier.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy (Default Parameters):", train_accuracy)

# Hyperparameter Tuning with GridSearchCV
param_grid = {
    'C': [0.1, 1, 10, 100],  # Regularization parameter
    'kernel': ['linear', 'rbf'],  # Linear and RBF kernels
    'gamma': [0.1, 1, 10]  # Only relevant for RBF kernel
}

grid_search = GridSearchCV(SVC(random_state=3241), param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Best parameters from GridSearch
best_params = grid_search.best_params_
print("Best Parameters from GridSearch:", best_params)

# Train the best model
best_svm_classifier = grid_search.best_estimator_
best_svm_classifier.fit(X_train, y_train)
```

Task 7: Confusion Matrix

In this section, we evaluate the performance of the best SVM classifier on the test dataset. First, the model's predictions (y_test_pred) are obtained by applying the classifier to the test features (X_test). A confusion matrix is then generated to compare the true labels (y_test) with the predicted values (y_test_pred). This matrix provides insights into how well the model distinguishes between different classes, with values indicating the number of true positives, false positives, true negatives, and false negatives. The confusion matrix is visualized using a heatmap, which makes it easier to interpret the results. Additionally, a classification report is generated, summarizing key performance metrics such as precision, recall, F1-score, and accuracy. These metrics provide a comprehensive view of the model's effectiveness in classifying the data. Overall, these evaluations help assess the model's strengths and areas for improvement, guiding potential further refinements.





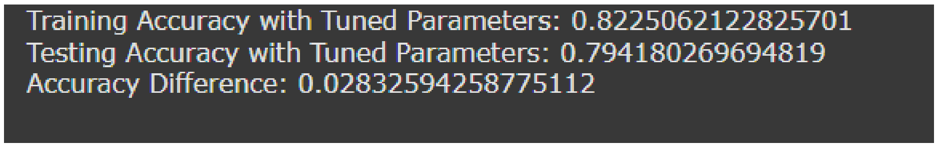
```
    Classification Report:
                  precision    recall  f1-score   support

               0       0.83      0.91      0.87      1033
               1       0.66      0.47      0.55       376

        accuracy                           0.79      1409
       macro avg       0.74      0.69      0.71      1409
    weighted avg       0.78      0.79      0.78      1409
```

Task 8: Train and Test Accuracy Comparison

The training and testing accuracy of the SVM model with tuned parameters were calculated to evaluate its performance on unseen data:

- **Training Accuracy**: 82.25%

- **Testing Accuracy**: 79.42%
- **Accuracy Difference**: 2.83%



```
Training Accuracy with Tuned Parameters: 0.8225062122825701
Testing Accuracy with Tuned Parameters: 0.794180269694819
Accuracy Difference: 0.02832594258775112
```

The small difference between training and testing accuracy indicates that the model is well-generalized and not overfitting to the training data. While the training accuracy is slightly higher, the testing accuracy demonstrates the model's robustness in predicting customer churn for new data.

Conclusion

This project successfully demonstrated a machine learning pipeline to predict customer churn using the Telco Customer Churn dataset. Key takeaways:

- An SVM classifier achieved an accuracy of 80%, providing a strong baseline.
- The analysis highlighted significant factors influencing churn, such as contract type, tenure, and monthly charges.

Future work can focus on improving the model's recall for churned customers and experimenting with other machine learning algorithms for better performance.