Telcom_Churn_Prediction

December 6, 2024

0.1 Step 1: Initial Data Load and Inspection

Objective: To load the dataset and inspect its structure. This step ensures we know what kind of data we're dealing with, including the types of variables, number of rows and columns, and any immediate issues like missing values.Tasks:

1. Load the dataset. 2. Display the first few rows to understand its layout. 3. Check for data types, basic statistics, and any obvious missing values.

1. Load the dataset

[5]:
```python
# Load the dataset
import pandas as pd
df = pd.read_csv('./Telco_Customer_Churn.csv')
```

2. Inspect the First Few Rows This will give us a quick look at the layout of the dataset.

[7]:
```python
# Inspect the First Few Rows
df.head()
```

[7]:
```
   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]
```

3. Check Data Types and Missing Values provides the data types, which helps us see if any columns need encoding (like categorical features) and spot any missing values.

[9]:
```python
# Check Data Types and Missing Values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        7043 non-null   object
 1   gender            7043 non-null   object
 2   SeniorCitizen     7043 non-null   int64
 3   Partner           7043 non-null   object
 4   Dependents        7043 non-null   object
 5   tenure            7043 non-null   int64
 6   PhoneService      7043 non-null   object
 7   MultipleLines     7043 non-null   object
 8   InternetService   7043 non-null   object
 9   OnlineSecurity    7043 non-null   object
 10  OnlineBackup      7043 non-null   object
 11  DeviceProtection  7043 non-null   object
 12  TechSupport       7043 non-null   object
 13  StreamingTV       7043 non-null   object
 14  StreamingMovies   7043 non-null   object
 15  Contract          7043 non-null   object
 16  PaperlessBilling  7043 non-null   object
 17  PaymentMethod     7043 non-null   object
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object
 20  Churn             7043 non-null   object
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

4. Summary Statistics This gives us basic statistics for numerical columns, which helps understand data distributions

[12]:
```python
df.describe()
```

[12]:

	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000
50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000

0.2 Step 2: Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) with a focus on identifying patterns and relationships that could help predict customer churn.Plan for this EDA phase:

1. Handling TotalCharges: Since this column had an unexpected data type (object), we'll convert it to float and handle any non-numeric values. 2. Analyzing Distributions: We'll look at the distribution of key numerical and categorical variables, focusing on tenure, MonthlyCharges, Contract, and Churn. 3. Exploring Correlations: We'll create a correlation heatmap to see relationships among numerical features.

1. Convert TotalCharges to Numeric and Handle Missing Values This step ensures that TotalCharges is in a usable numeric format, with missing values filled.

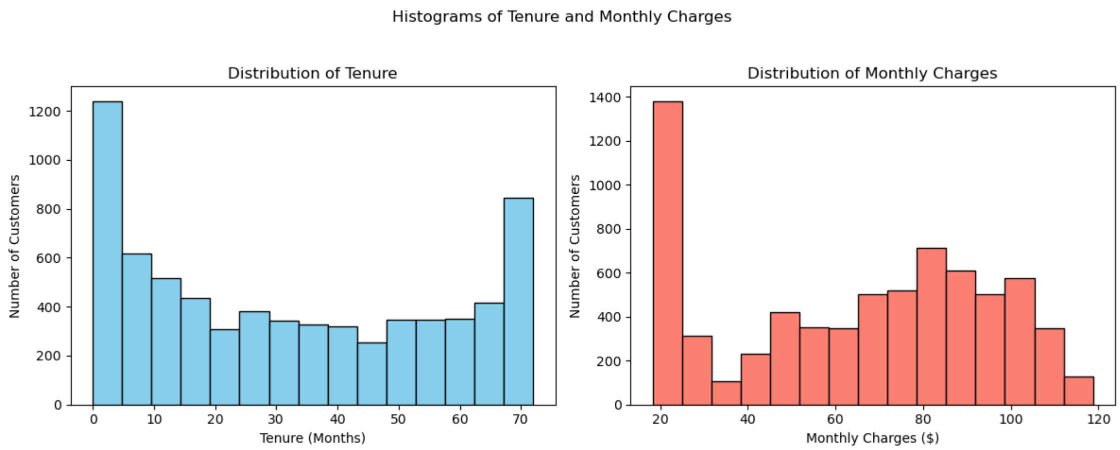
[15]:
```python
# Convert TotalCharges to numeric, and coerce any non-numeric values to NaN
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')

# Count missing values in TotalCharges (if any) and fill with median or mean
missing_total_charges = df['TotalCharges'].isna().sum()
print("Missing values in TotalCharges:", missing_total_charges)

# Fill missing values with the median of TotalCharges
df['TotalCharges'] = df['TotalCharges'].fillna(df['TotalCharges'].median())
```

```
Missing values in TotalCharges: 11
```

2. Visualize Numerical Distributions For tenure and MonthlyCharges, histograms can help us see if certain ranges are more common, potentially showing clusters of customer behaviors.

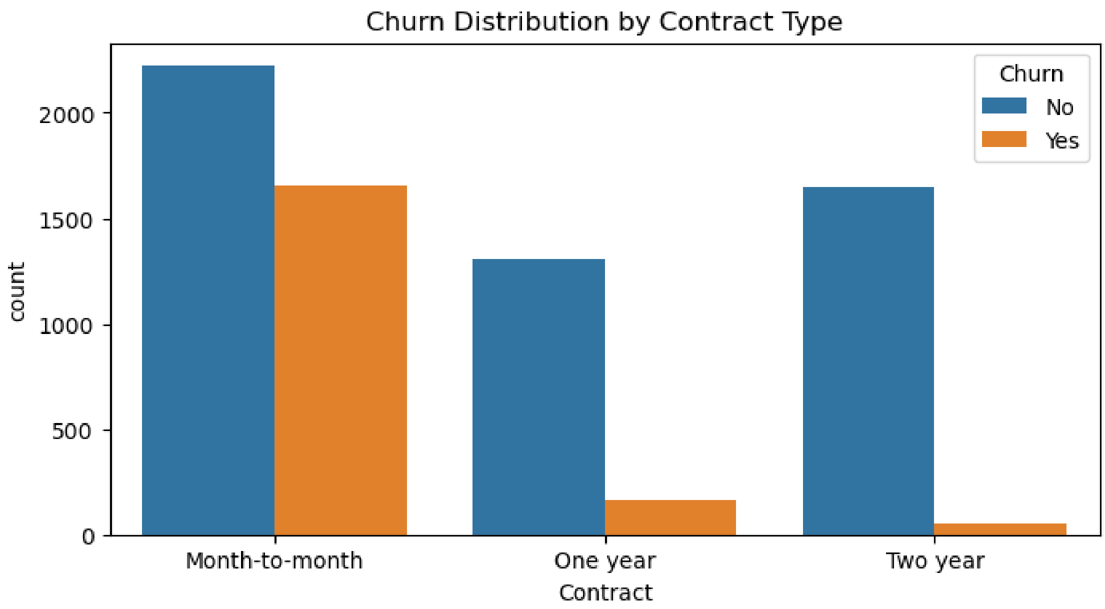
[17]:
```python
import matplotlib.pyplot as plt

# Plot histograms for tenure and MonthlyCharges with labels
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Tenure histogram
axes[0].hist(df['tenure'], bins=15, color='skyblue', edgecolor='black')
```

```
axes[0].set_title('Distribution of Tenure')
axes[0].set_xlabel('Tenure (Months)')
axes[0].set_ylabel('Number of Customers')

# MonthlyCharges histogram
axes[1].hist(df['MonthlyCharges'], bins=15, color='salmon', edgecolor='black')
axes[1].set_title('Distribution of Monthly Charges')
axes[1].set_xlabel('Monthly Charges ($)')
axes[1].set_ylabel('Number of Customers')

plt.suptitle('Histograms of Tenure and Monthly Charges')
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



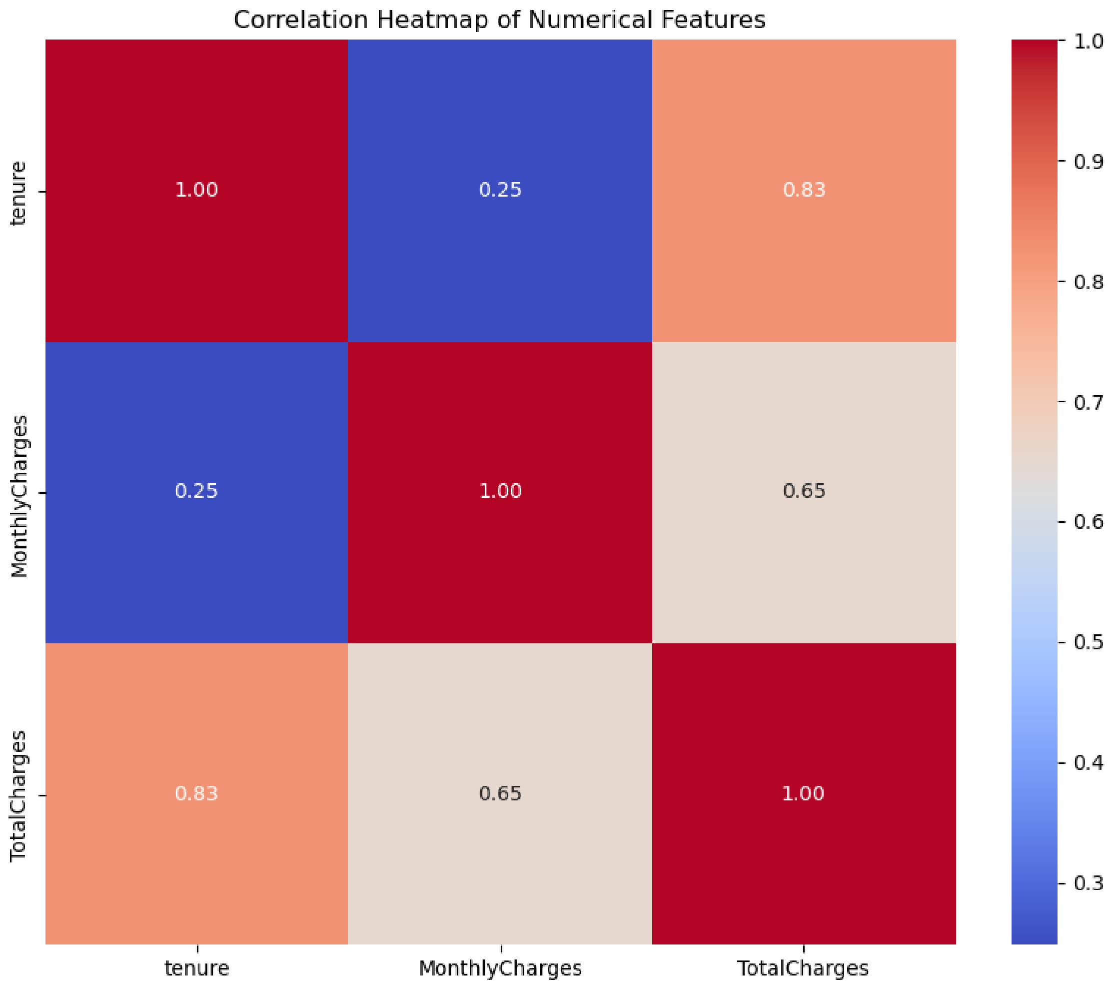
3. Plot Categorical Features Against Churn We'll look at how categorical features like Contract type are distributed across churn statuses. This can reveal if certain customer types (e.g., month-to-month contracts) have higher churn rates.

[19]:
```
import seaborn as sns

# Plot for Contract type by Churn
plt.figure(figsize=(8, 4))
sns.countplot(data=df, x='Contract', hue='Churn')
plt.title('Churn Distribution by Contract Type')
plt.show()
```



4. Correlation Heatmap for Numerical Features A correlation heatmap can help us identify if any numerical features are highly related to each other or to churn.

[21]:
```
# Plot correlation heatmap for numerical features
plt.figure(figsize=(10, 8))
sns.heatmap(df[['tenure', 'MonthlyCharges', 'TotalCharges']].corr(),␣
 ↪annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap of Numerical Features')
plt.show()
```

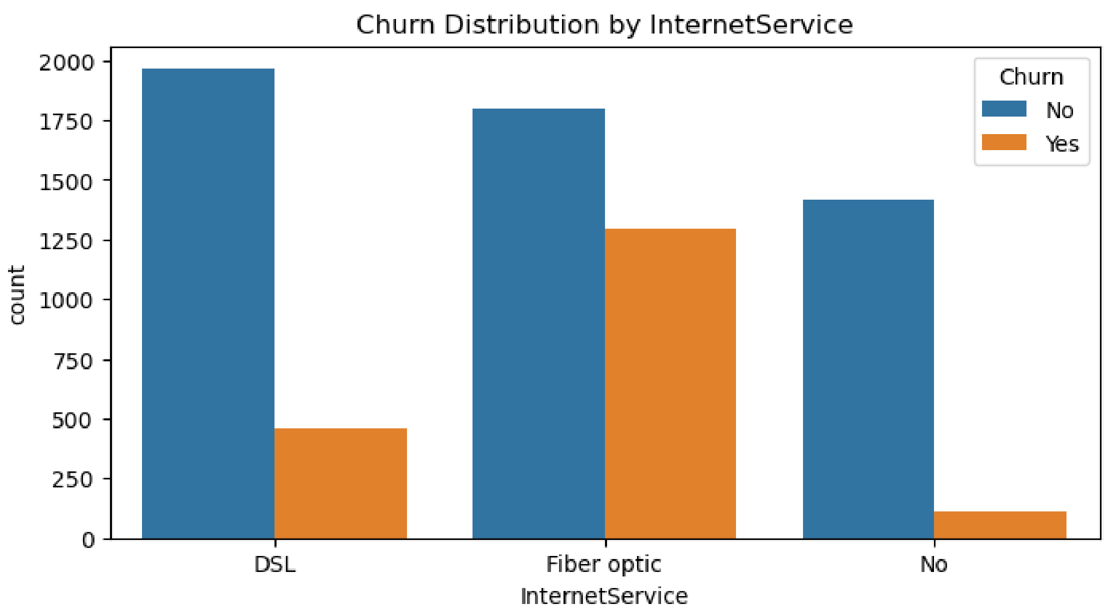


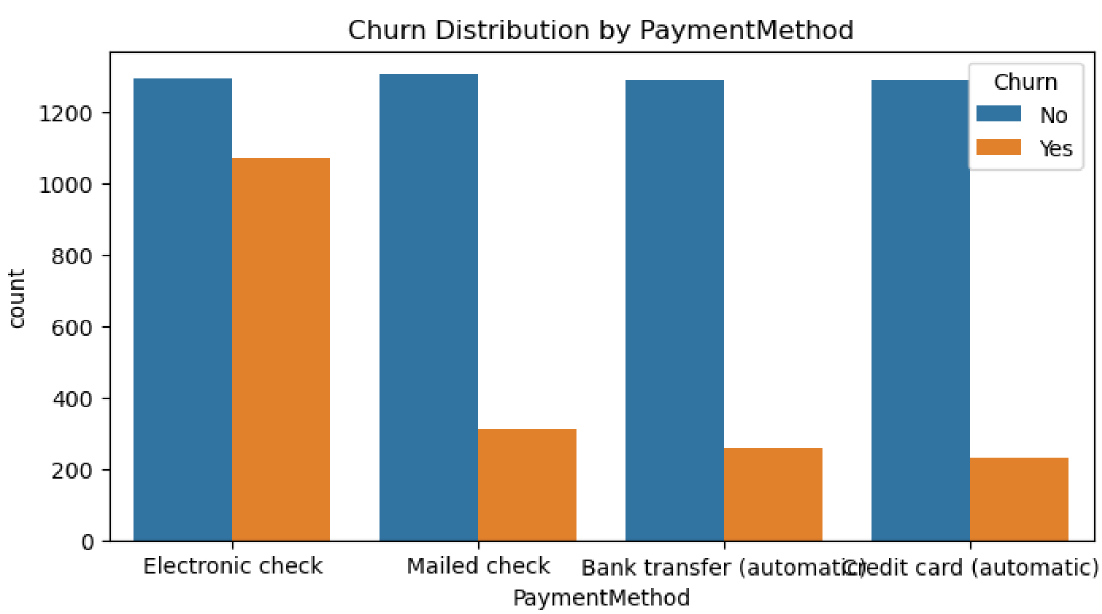
0.3 Extended EDA Plan

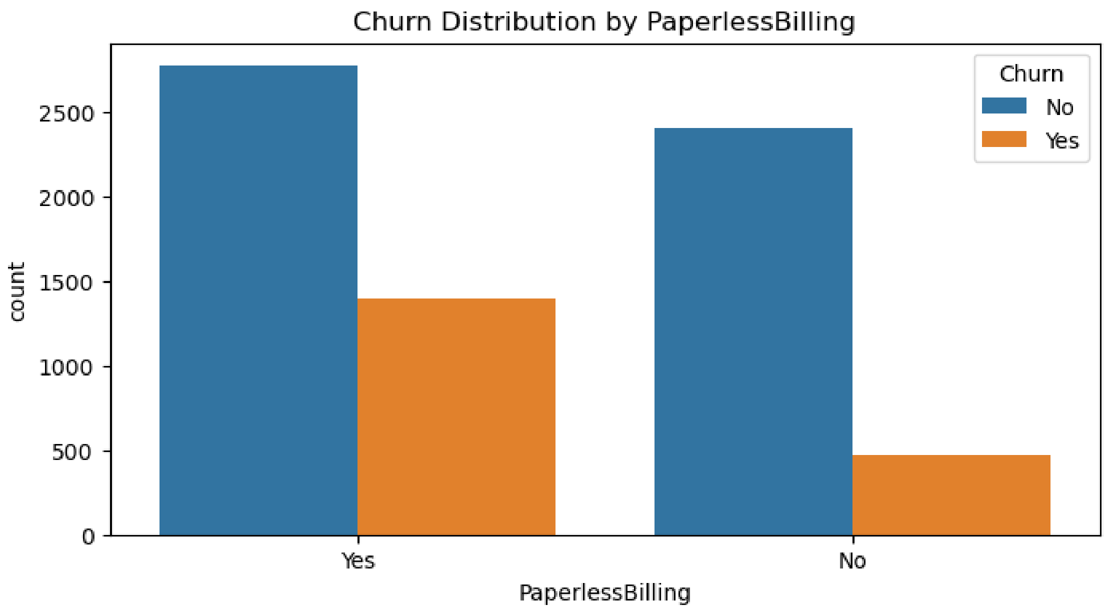
1. Churn Rate by Categorical Variables: Analyzing other categorical features (e.g., PaymentMethod, InternetService, PaperlessBilling) to see if certain categories show higher churn rates. 2. Numerical Feature Analysis by Churn: Comparing distributions of numerical features (e.g., tenure, MonthlyCharges, TotalCharges) for churned vs. non-churned customers. 3. Bivariate Analysis: Exploring interactions between features to see if certain combinations are predictive of churn. 4. Outlier Detection: Checking for any unusual values in MonthlyCharges and TotalCharges that might need handling before modeling. 5. Further Correlation Analysis: Looking specifically at correlations with the Churn variable and possibly calculating the correlation between encoded categorical features and churn.

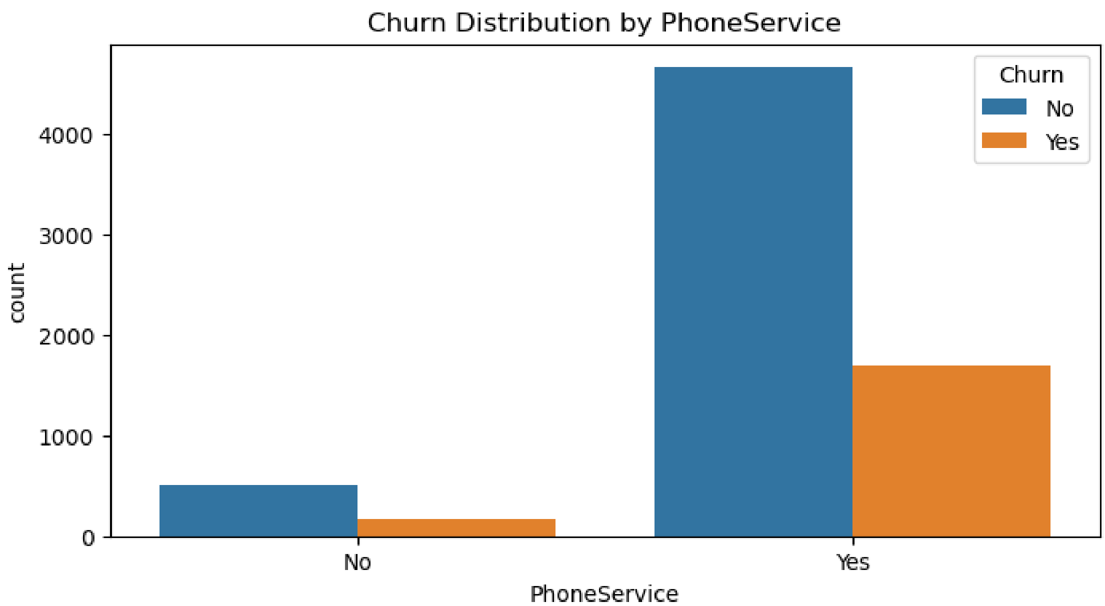
1. Churn Rate by Categorical Variables This will help identify if certain customer behaviors or characteristics are associated with higher churn. We'll create bar plots for several categorical variables against churn.

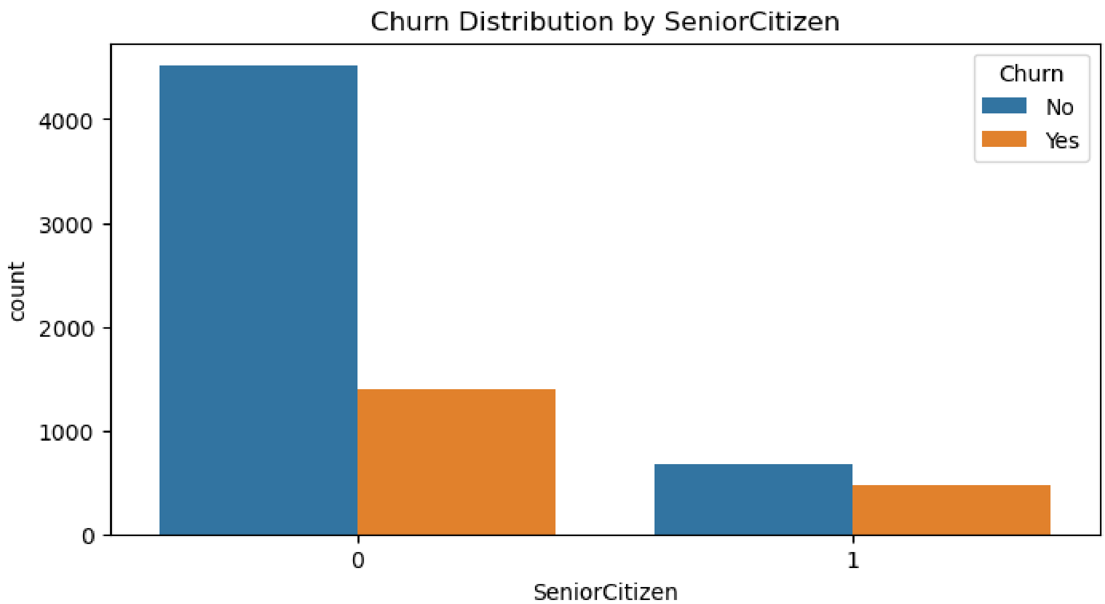
[24]:
```
categorical_features = ['InternetService', 'PaymentMethod', 'PaperlessBilling',␣
↪'PhoneService', 'SeniorCitizen']

for feature in categorical_features:
    plt.figure(figsize=(8, 4))
    sns.countplot(data=df, x=feature, hue='Churn')
    plt.title(f'Churn Distribution by {feature}')
    plt.show()
```





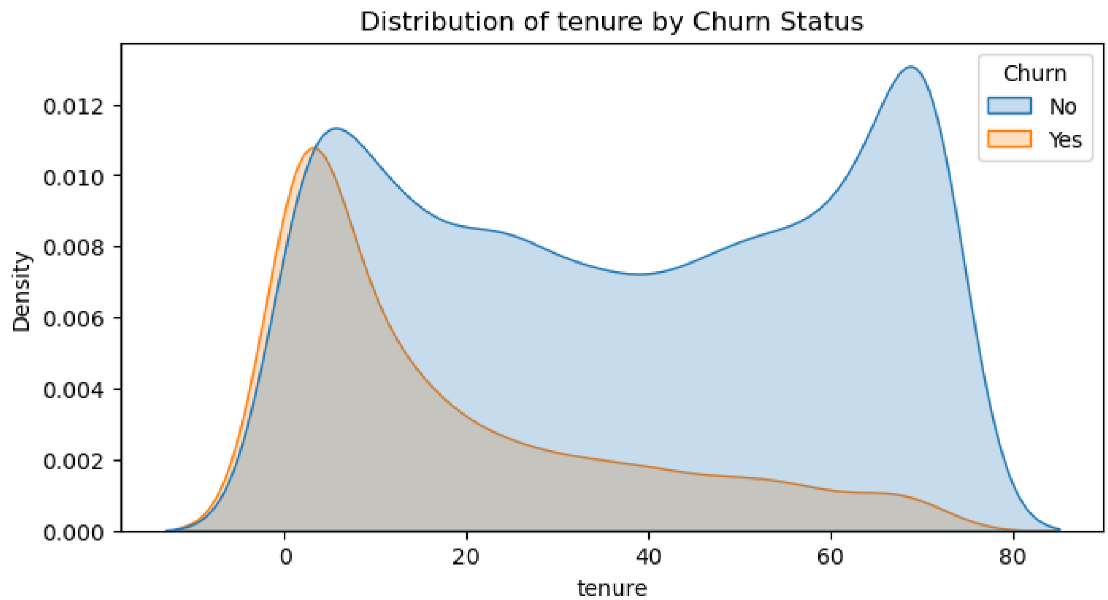

Churn Distribution by PaperlessBilling
Churn
No
Yes
2500
2000
1500
1000
500
0
count
Yes
No
PaperlessBilling

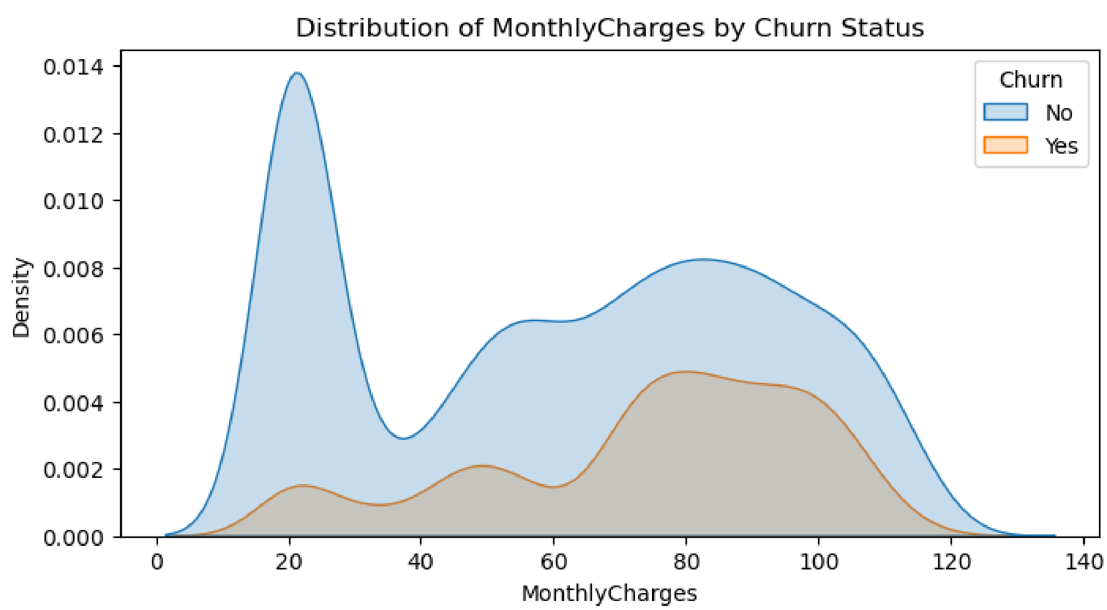

Churn Distribution by PhoneService
Churn
No
Yes
4000
3000
2000
1000
0
count
No
Yes
PhoneService

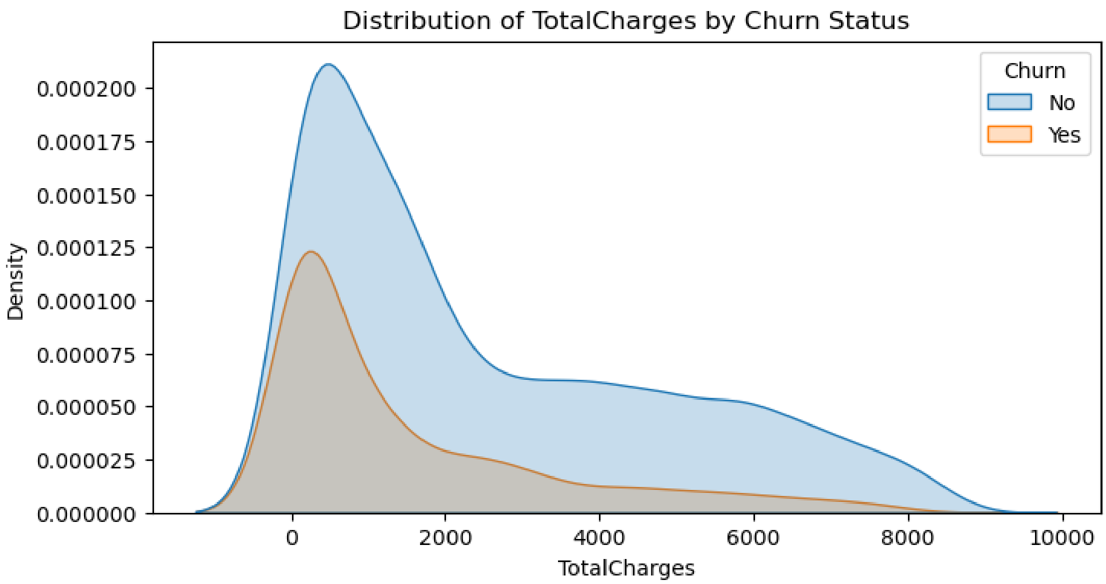


2. Numerical Feature Analysis by Churn This will allow us to see how tenure, MonthlyCharges, and TotalCharges distributions differ for customers who churned vs. those who didn't.

[26]:
```python
numerical_features = ['tenure', 'MonthlyCharges', 'TotalCharges']

for feature in numerical_features:
    plt.figure(figsize=(8, 4))
    sns.kdeplot(data=df, x=feature, hue='Churn', fill=True)
    plt.title(f'Distribution of {feature} by Churn Status')
    plt.xlabel(feature)
    plt.ylabel('Density')
    plt.show()
```


Distribution of tenure by Churn Status
Churn
No
Yes
Density
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0
20
40
60
80
tenure

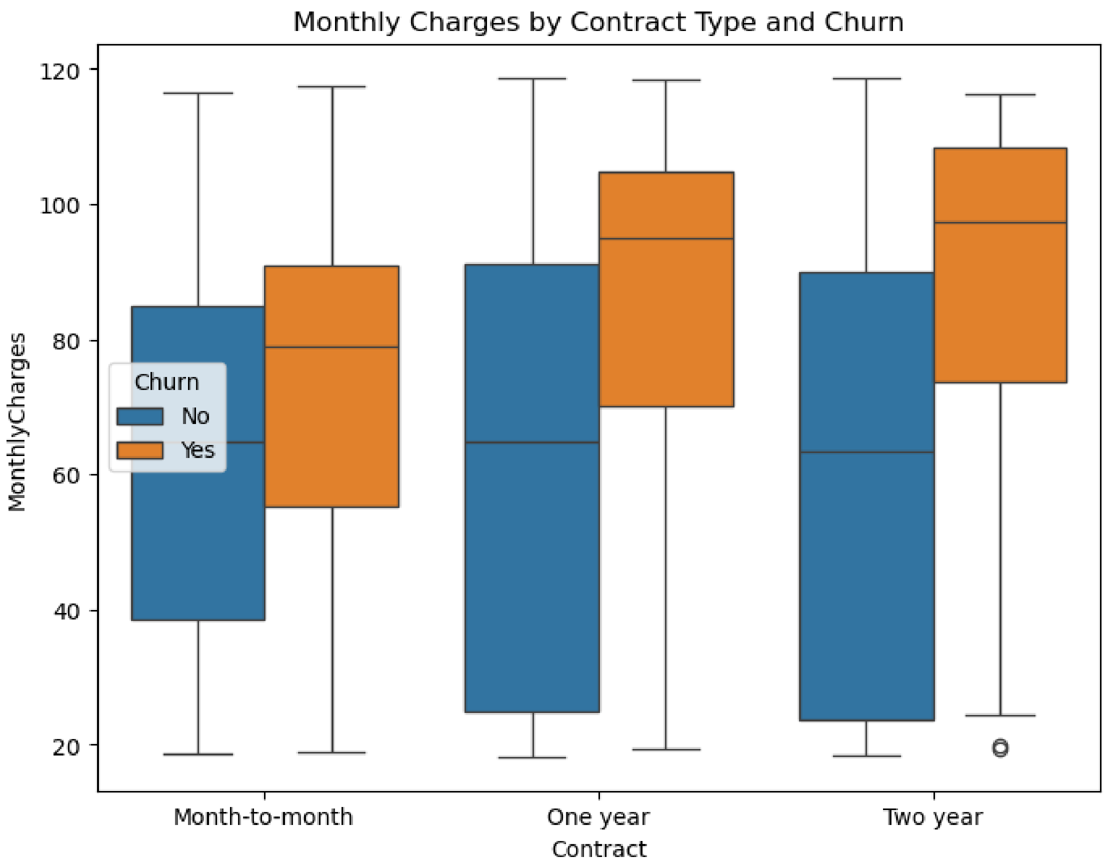

Distribution of MonthlyCharges by Churn Status
Churn
No
Yes
Density
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
0
20
40
60
80
100
120
140
MonthlyCharges

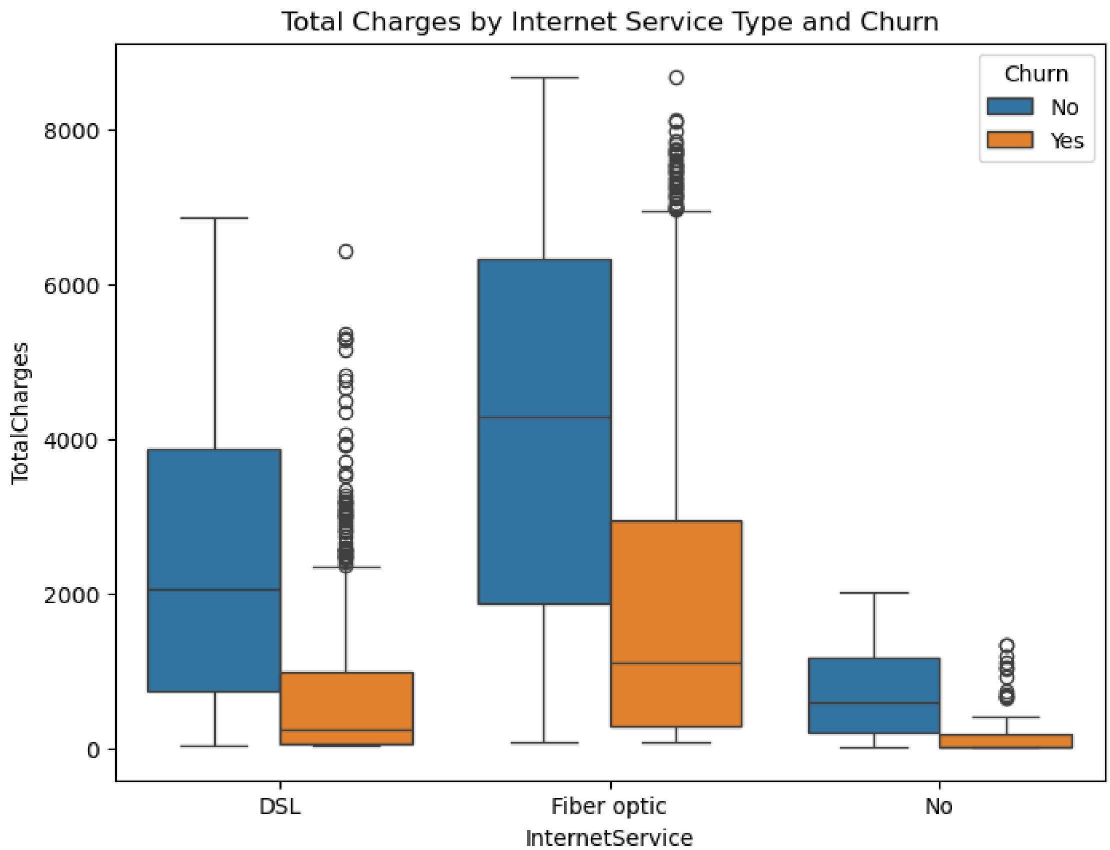


3. Bivariate Analysis of Key Features Let's explore possible interactions, particularly between:

Contract and MonthlyCharges InternetService and TotalCharges

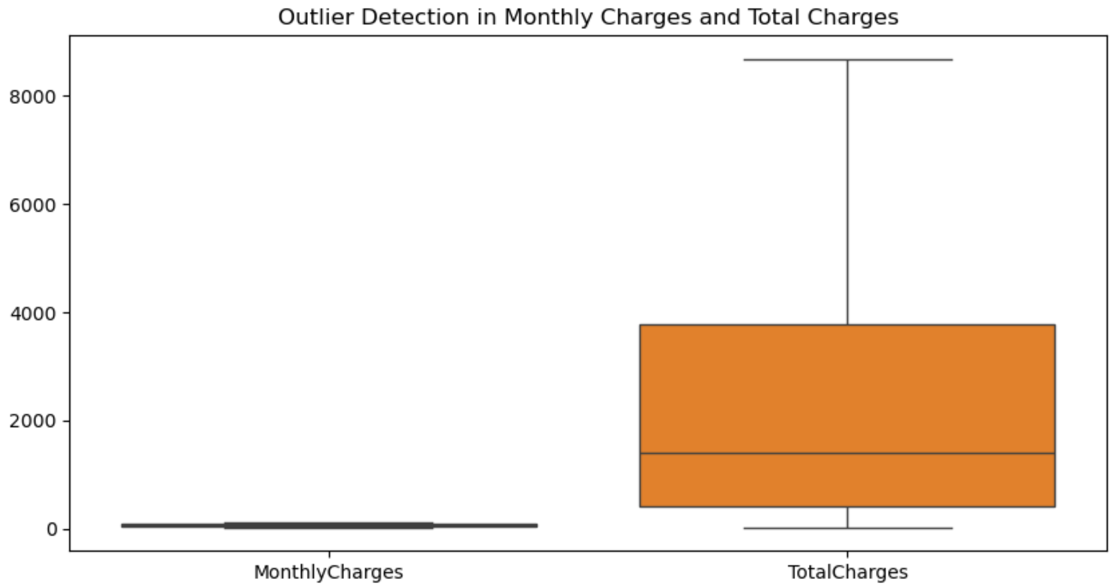
[28]:
```
# Contract type vs Monthly Charges, split by Churn
plt.figure(figsize=(8, 6))
sns.boxplot(data=df, x='Contract', y='MonthlyCharges', hue='Churn')
plt.title('Monthly Charges by Contract Type and Churn')
plt.show()

# Internet Service vs Total Charges, split by Churn
plt.figure(figsize=(8, 6))
sns.boxplot(data=df, x='InternetService', y='TotalCharges', hue='Churn')
plt.title('Total Charges by Internet Service Type and Churn')
plt.show()
```


Monthly Charges by Contract Type and Churn
MonthlyCharges
120
100
80
60
40
20
Churn
No
Yes
Month-to-month
One year
Two year
Contract



4. Outlier Detection for MonthlyCharges and TotalCharges Checking for extreme values in MonthlyCharges and TotalCharges to see if they might need special handling.

[30]:
```
# Boxplots for outlier detection
plt.figure(figsize=(10, 5))
sns.boxplot(data=df[['MonthlyCharges', 'TotalCharges']])
plt.title('Outlier Detection in Monthly Charges and Total Charges')
plt.show()
```



5. Further Correlation Analysis with Churn Let's encode the Churn column to numerical (0 for No, 1 for Yes) and check correlations with other numerical variables. This will help us quantify the relationship between each feature and churn.

[32]:
```
# Ensure only numeric columns are used in correlation calculation
numeric_df = df.select_dtypes(include=['float64', 'int64'])

# Add the encoded Churn column to the numeric-only DataFrame
numeric_df['Churn_Encoded'] = df['Churn'].apply(lambda x: 1 if x == 'Yes' else␣
↪0)

# Calculate correlations with the Churn_Encoded column
correlations_with_churn = numeric_df.corr()['Churn_Encoded'].
↪sort_values(ascending=False)
print("Correlations with Churn:\n", correlations_with_churn)
```

```
Correlations with Churn:
 Churn_Encoded     1.000000
MonthlyCharges    0.193356
SeniorCitizen     0.150889
TotalCharges     -0.199037
tenure           -0.352229
Name: Churn_Encoded, dtype: float64
```

0.4 Step 3: Data Preprocessing

With our EDA insights in hand, we're now ready to move to data preprocessing and feature engineering. This step will prepare our dataset for modeling, ensuring that it's clean, consistent, and optimized for accurate predictions.Step-by-Step Plan Post-EDA: 1. Feature Selection and Engineering: Carefully choose the features that will be used in the model. 2. Data Splitting: Split the data into training and test sets before any data transformation or scaling. 3. Preprocessing Pipelines: Apply separate pipelines for numerical and categorical features to avoid data contamination. 4. Model Training: Start with a baseline model (e.g., Logistic Regression) and ensure proper cross-validation. 5. Evaluation and Next Steps: Analyze results and identify next steps for improving the model.

[34]:
```
df.head()
```

[34]:
```
   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       1889.50     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]
```

1. Feature Selection and Engineering Choose features based on their importance and logical correlation with the target variable (Churn). Avoid using features that could directly reveal target information (e.g., unique identifiers).

[36]:
```
# Create a copy of the original dataset after EDA
df_clean = df.copy()
df_clean.head()
```

[36]:
```
     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       1889.50     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]
```

[37]:
```python
# Drop 'customerID' column only if it exists
if 'customerID' in df_clean.columns:
    df_clean = df_clean.drop(['customerID'], axis=1)


# Ensure the target column is separated
X = df_clean.drop('Churn', axis=1)
y = df_clean['Churn']
```

2. Train-Test Split Split the data before any transformations to avoid data leakage.

[39]:
```python
from sklearn.model_selection import train_test_split

# Split the data into training and test sets (e.g., 80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

3. Preprocessing Pipelines Set up separate pipelines for numerical and categorical features.Preprocessing Pipelines: Ensure numerical and categorical features are handled separately and only use training data for transformations.

[41]:
```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder

# Identify numerical and categorical columns
numerical_features = X_train.select_dtypes(include=['int64', 'float64']).columns
categorical_features = X_train.select_dtypes(include=['object', 'category']).
 ↪columns

# Create a preprocessing pipeline
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(drop='first'), categorical_features)
    ]
)

# Verify preprocessor setup by fitting on training data only
X_train_preprocessed = preprocessor.fit_transform(X_train)
X_test_preprocessed = preprocessor.transform(X_test)
```

0.5 Step 4: Model Training with Baseline Model

Train a simple model, such as Logistic Regression, as a baseline.

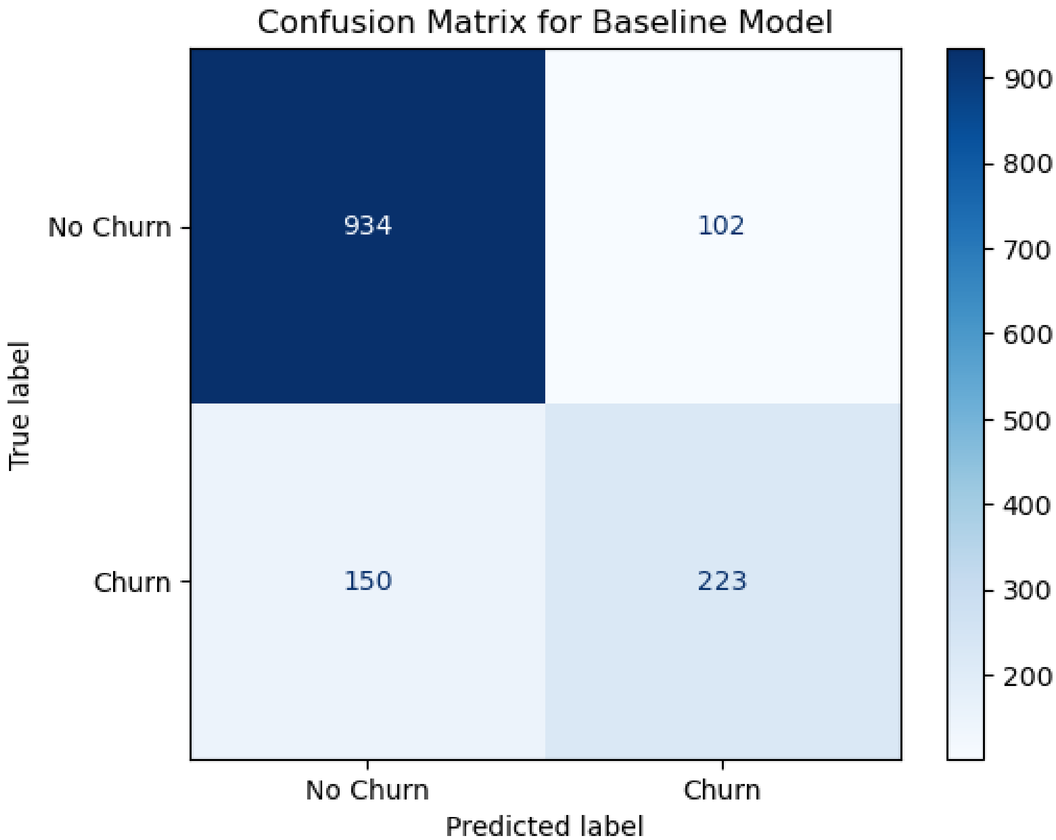
[43]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, roc_auc_score

# Create a pipeline for the model
model_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(max_iter=1000, random_state=42))
])

# Train the model
model_pipeline.fit(X_train, y_train)

# Predict and evaluate the model
y_pred = model_pipeline.predict(X_test)
y_pred_prob = model_pipeline.predict_proba(X_test)[:, 1]

# Print evaluation metrics
print("Accuracy:", accuracy_score(y_test, y_pred))
print("AUC-ROC:", roc_auc_score(y_test, y_pred_prob))
```

```
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

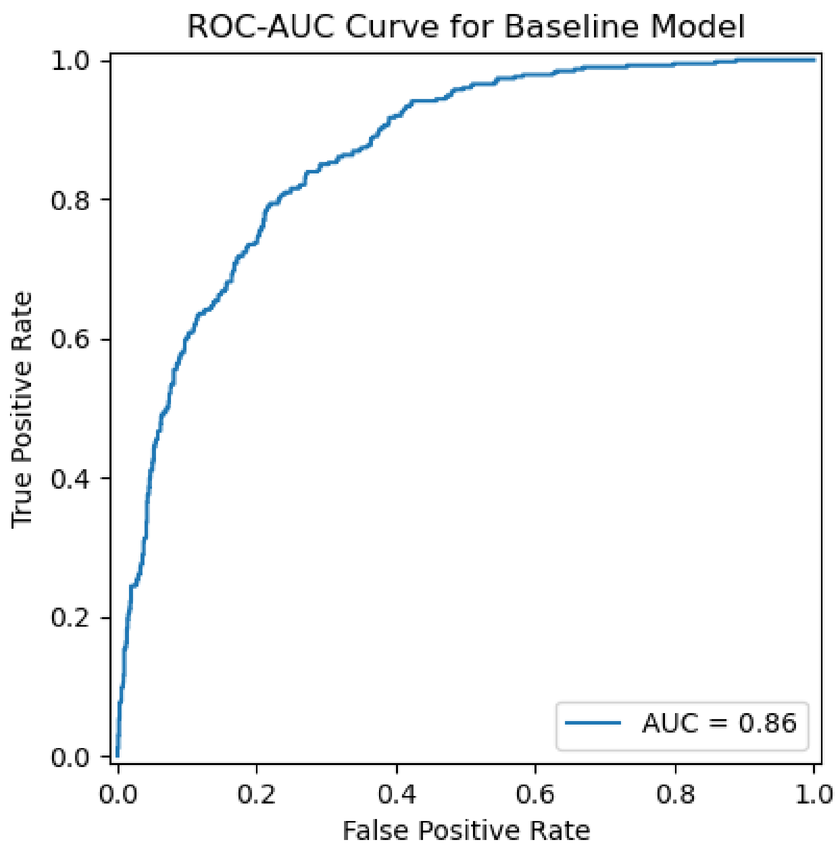
```
Accuracy: 0.8211497515968772
AUC-ROC: 0.8621334375355824

Classification Report:
               precision    recall  f1-score   support

          No       0.86      0.90      0.88      1036
         Yes       0.69      0.60      0.64       373

    accuracy                           0.82      1409
   macro avg       0.77      0.75      0.76      1409
weighted avg       0.82      0.82      0.82      1409
```

[44]:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Generate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=['No␣
↪Churn', 'Churn'])
disp.plot(cmap='Blues')
plt.title('Confusion Matrix for Baseline Model')
plt.show()
```



[45]:
```
from sklearn.metrics import roc_curve, RocCurveDisplay

# Compute the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob, pos_label='Yes')

# Display the ROC curve
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=roc_auc_score(y_test,␣
↪y_pred_prob))
roc_display.plot()
plt.title('ROC-AUC Curve for Baseline Model')
plt.show()
```



Step 5: Validation with Cross-Validation Use cross-validation to ensure that the model generalizes well. Cross-Validation: Provides a more realistic estimate of the model's performance by validating on different data splits.

[48]:
```
from sklearn.model_selection import cross_val_score

# Cross-validate the model
cv_scores = cross_val_score(model_pipeline, X_train, y_train, cv=5,␣
 ↪scoring='accuracy')
print("Cross-validation scores:", cv_scores)
print("Mean cross-validation score:", cv_scores.mean())
```

```
Cross-validation scores: [0.81011535 0.80834073 0.79769299 0.78970719
0.80195382]
Mean cross-validation score: 0.801562014874681
```

1 Part 2 of our project journey

Doing pre-processing for data again to prepare for it's second journey

[51]:
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

df_processed = df_clean.copy()
# Drop 'customerID' column only if it exists
if 'customerID' in df_processed.columns:
    df_processed = df_processed.drop(['customerID'], axis=1)

# Step 2: Handle missing or invalid values
# Convert TotalCharges to numeric, coerce errors to NaN, and fill NaN with␣
↪median
df_processed['TotalCharges'] = pd.to_numeric(df_processed['TotalCharges'],␣
↪errors='coerce')
df_processed['TotalCharges'] = df_processed['TotalCharges'].
↪fillna(df_processed['TotalCharges'].median())

# Step 3: Identify categorical and numerical features
categorical_features = df_processed.select_dtypes(include=['object']).
↪drop(['Churn'], axis=1).columns
numerical_features = df_processed.select_dtypes(include=['int64', 'float64']).
↪columns

# Step 4: Define target and features
X = df_processed.drop(['Churn'], axis=1)
y = df_processed['Churn'].map({'Yes': 1, 'No': 0})  # Encode target variable

# Step 5: Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Step 6: Create preprocessing pipeline
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(drop='first'), categorical_features)
    ]
)

# Step 7: Fit and transform training data
X_train_preprocessed = preprocessor.fit_transform(X_train)
X_test_preprocessed = preprocessor.transform(X_test)

# Verify preprocessed shapes
X_train_preprocessed.shape, X_test_preprocessed.shape, y_train.shape, y_test.
↪shape
```

[51]: ((5634, 30), (1409, 30), (5634,), (1409,))

[97]:
```
df_processed.shape
```

[97]: (7043, 20)

[99]:
```
df_processed.head()
```

[99]:
```
   gender  SeniorCitizen Partner Dependents  tenure PhoneService  \
0  Female              0     Yes         No       1           No
1    Male              0      No         No      34          Yes
2    Male              0      No         No       2          Yes
3    Male              0      No         No      45           No
4  Female              0      No         No       2          Yes

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

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

  PaperlessBilling              PaymentMethod  MonthlyCharges  TotalCharges  \
0              Yes           Electronic check           29.85         29.85
1               No               Mailed check           56.95       1889.50
2              Yes               Mailed check           53.85        108.15
3               No  Bank transfer (automatic)           42.30       1840.75
4              Yes           Electronic check           70.70        151.65

  Churn
0    No
1    No
2   Yes
3    No
4   Yes
```

Let's start by addressing class imbalance using SMOTE or class weights, as this forms the foundation for improving model recall. Here's how to proceed:

1. Using SMOTE (Synthetic Minority Oversampling Technique):

[54]:
```python
from imblearn.over_sampling import SMOTE

# Apply SMOTE to the preprocessed training data
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train_preprocessed,␣
 ↪y_train)

# Check the new class distribution
from collections import Counter
print("Class distribution after SMOTE:", Counter(y_train_balanced))
```

```
Class distribution after SMOTE: Counter({0: 4138, 1: 4138})
```

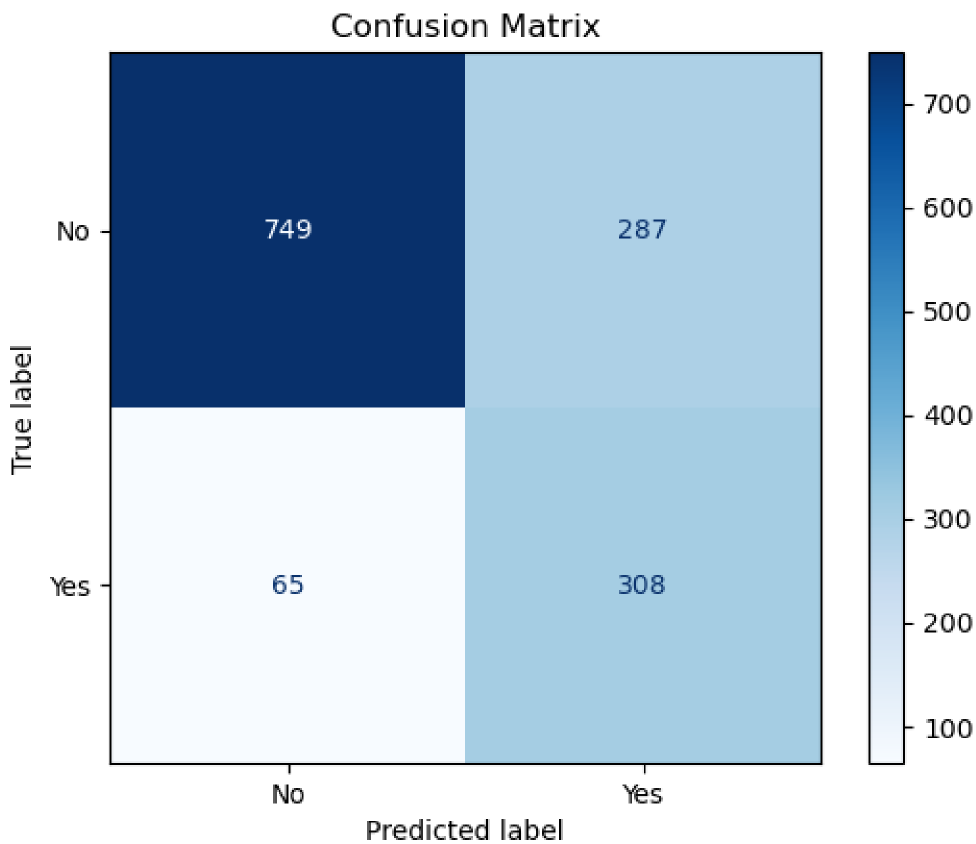
2. Using Class Weights in the Model:

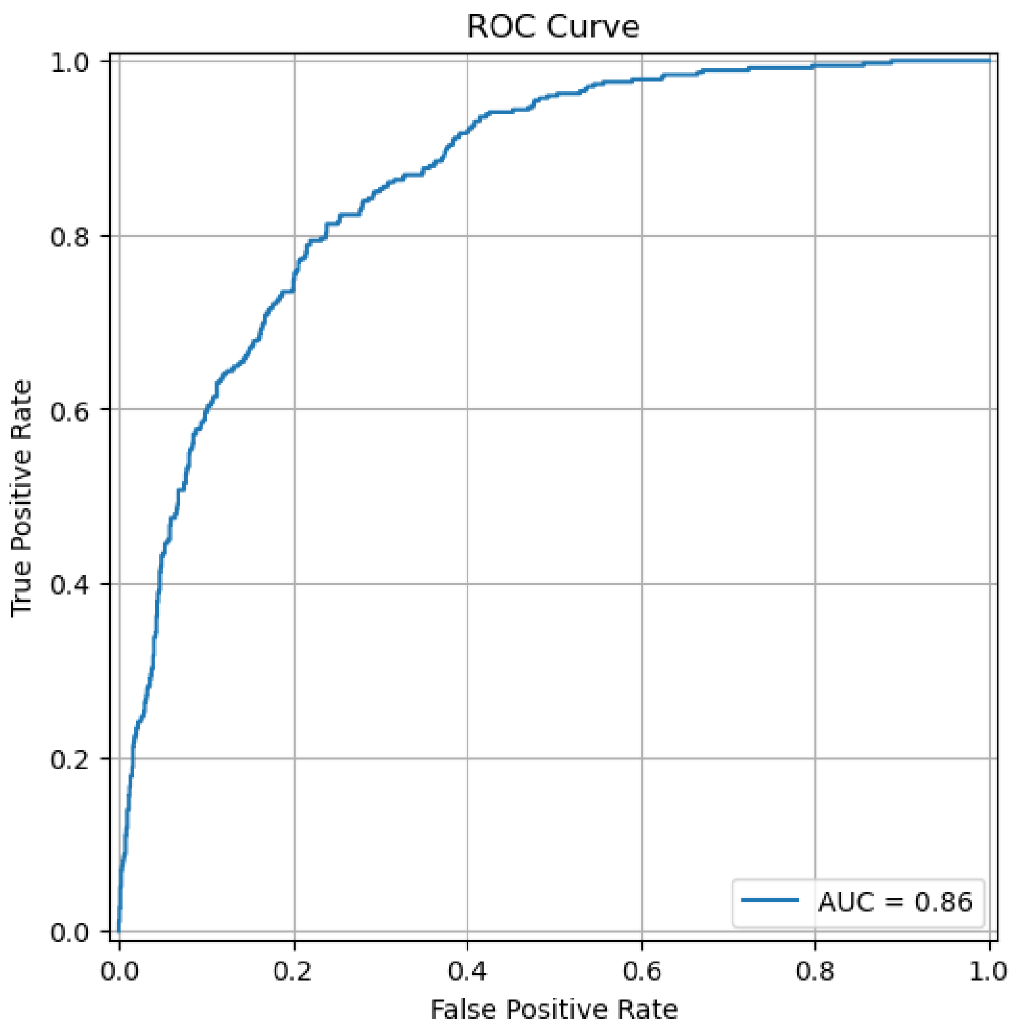
Let's retrain this model and also using class weights and we can see the improvement in Eveluation matrix is it not awesome

[57]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, roc_auc_score

# Train logistic regression with class weighting
model_with_weights = LogisticRegression(max_iter=1000, class_weight='balanced',␣
 ↪random_state=42)
model_with_weights.fit(X_train_preprocessed, y_train)

# Predict on test data
y_pred = model_with_weights.predict(X_test_preprocessed)
y_pred_prob = model_with_weights.predict_proba(X_test_preprocessed)[:, 1]

# Compute confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix,␣
 ↪display_labels=["No", "Yes"])

# Plot confusion matrix
plt.figure(figsize=(8, 6))
disp.plot(cmap="Blues")
plt.title("Confusion Matrix")
plt.show()

# Compute and plot ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
roc_auc = roc_auc_score(y_test, y_pred_prob)

plt.figure(figsize=(8, 6))
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=roc_auc)
roc_display.plot(ax=plt.gca())
```

```
plt.title("ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.grid()
plt.show()

# Display metrics
print("Accuracy:", accuracy_score(y_test, y_pred))
print("AUC-ROC:", roc_auc)
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
<Figure size 800x600 with 0 Axes>
```






```
Accuracy: 0.7501774308019872
AUC-ROC: 0.8621308497313858

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.72      0.81      1036
           1       0.52      0.83      0.64       373

    accuracy                           0.75      1409
   macro avg       0.72      0.77      0.72      1409
weighted avg       0.81      0.75      0.76      1409
```

These results suggest that the model is better at identifying non-churning customers than churningWe are successfully abale to achieve highe recall but we can observe that accuracey is decreasedkNow let's run all other model and see how they performed on given preprocessed data and compare all of them

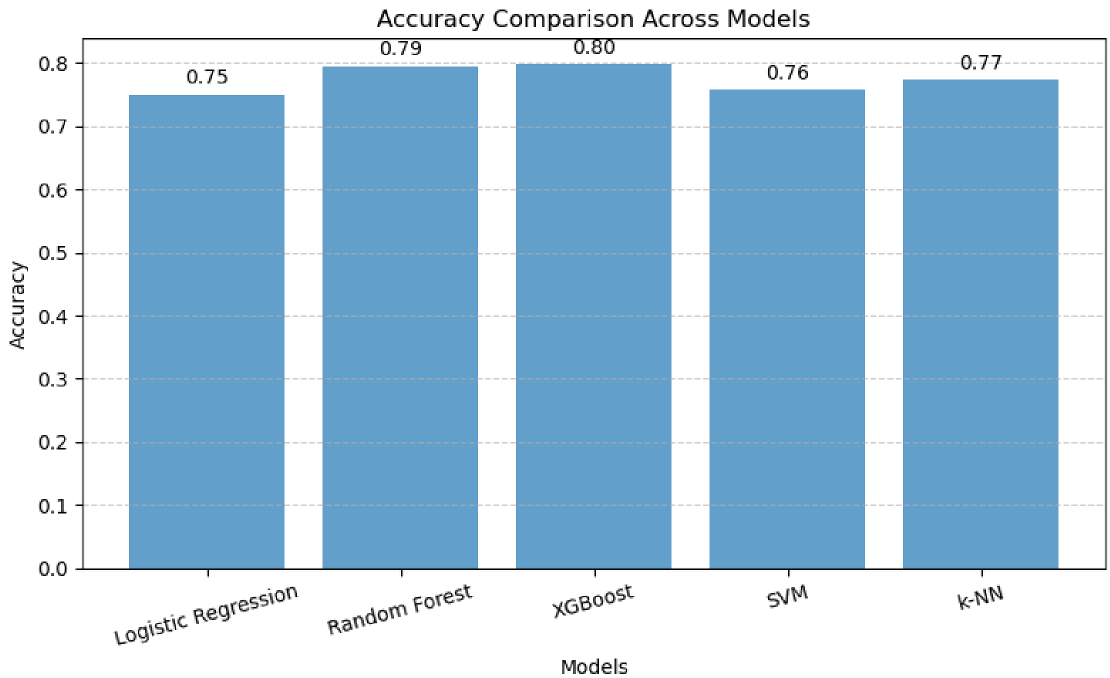
[58]:
```
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, roc_auc_score


# Redefine models
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000,␣
↪class_weight='balanced', random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=100,␣
↪class_weight='balanced', random_state=42),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='logloss',␣
↪random_state=42),
    "SVM": SVC(probability=True, random_state=42, class_weight='balanced'),
    "k-NN": KNeighborsClassifier()
}

# Function to train, predict, and evaluate a model
def evaluate_model(name, model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_pred_prob = model.predict_proba(X_test)[:, 1] if hasattr(model,␣
↪"predict_proba") else None

    metrics = {
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred),
        "F1-Score": f1_score(y_test, y_pred),
        "AUC-ROC": roc_auc_score(y_test, y_pred_prob) if y_pred_prob is not␣
↪None else None,
    }
    return metrics

# Evaluate all models
evaluation_results = {}
for model_name, model in models.items():
    evaluation_results[model_name] = evaluate_model(model_name, model,␣
↪X_train_preprocessed, X_test_preprocessed, y_train, y_test)

# Create a DataFrame for the evaluation results
```

```
evaluation_df = pd.DataFrame(evaluation_results).T
print(evaluation_df)

# tools.display_dataframe_to_user(name="Model Evaluation Results",␣
↪dataframe=evaluation_df)
```

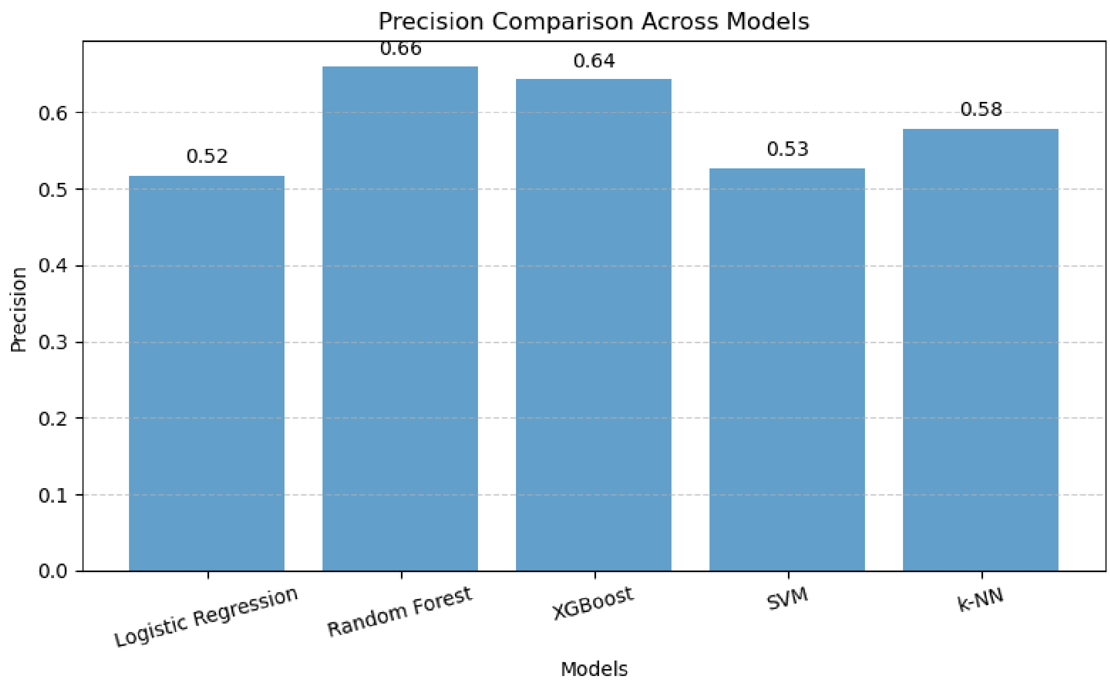
```
C:\Users\Unknown1\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[13:51:37] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)

                     Accuracy  Precision    Recall  F1-Score   AUC-ROC
Logistic Regression  0.750177   0.517647  0.825737  0.636364  0.862131
Random Forest        0.794890   0.660305  0.463807  0.544882  0.839710
XGBoost              0.798439   0.643087  0.536193  0.584795  0.839101
SVM                  0.757275   0.526863  0.815013  0.640000  0.844931
k-NN                 0.772889   0.579104  0.520107  0.548023  0.796759
```

Let's visualize this and comapre it check further

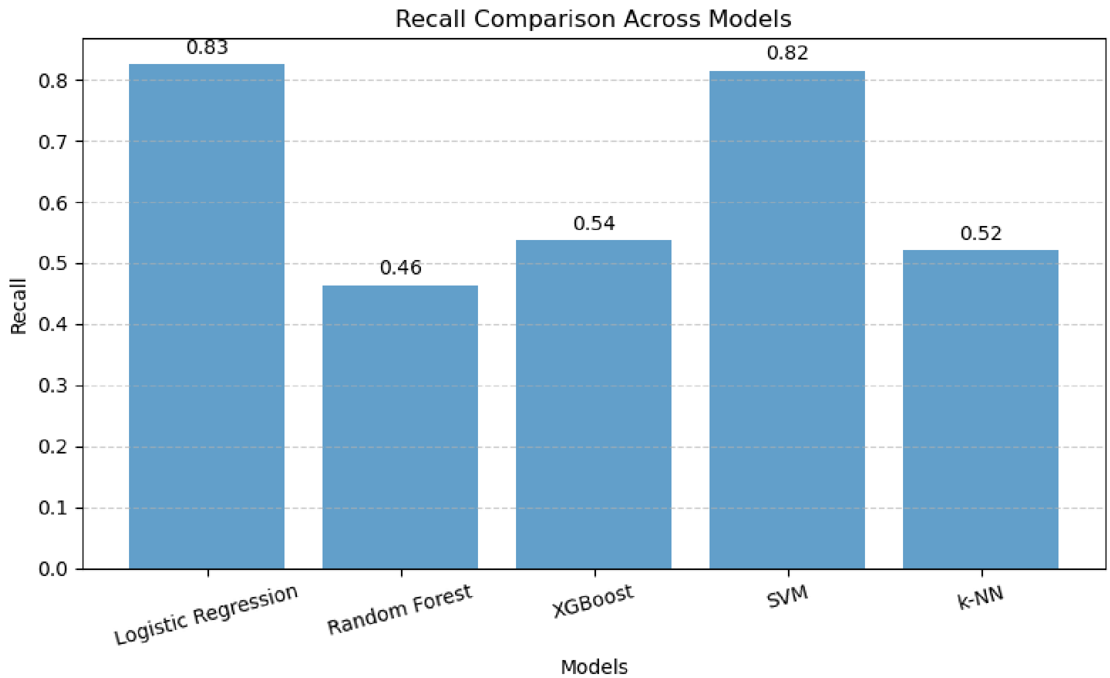
[59]:
```
# Plot each metric for all models dynamically
for metric in evaluation_df.columns:
    plt.figure(figsize=(8, 5))
    bars = plt.bar(evaluation_df.index, evaluation_df[metric], alpha=0.7)
    plt.title(f"{metric} Comparison Across Models")
    plt.ylabel(metric)
    plt.xlabel("Models")
    plt.xticks(rotation=15)
    plt.grid(axis='y', linestyle='--', alpha=0.6)

    # Add numbers on top of bars
    for bar in bars:
        plt.text(bar.get_x() + bar.get_width() / 2,
                 bar.get_height() + 0.01,
                 f"{bar.get_height():.2f}",
                 ha='center', va='bottom', fontsize=10)

    plt.tight_layout()
    plt.show()
```

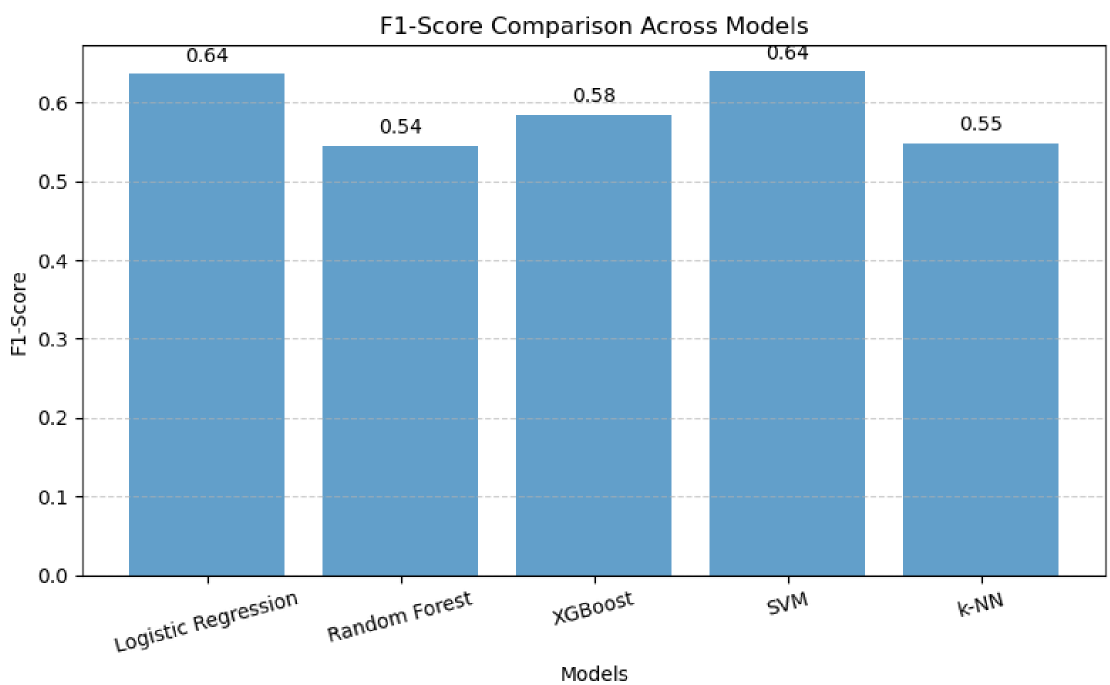


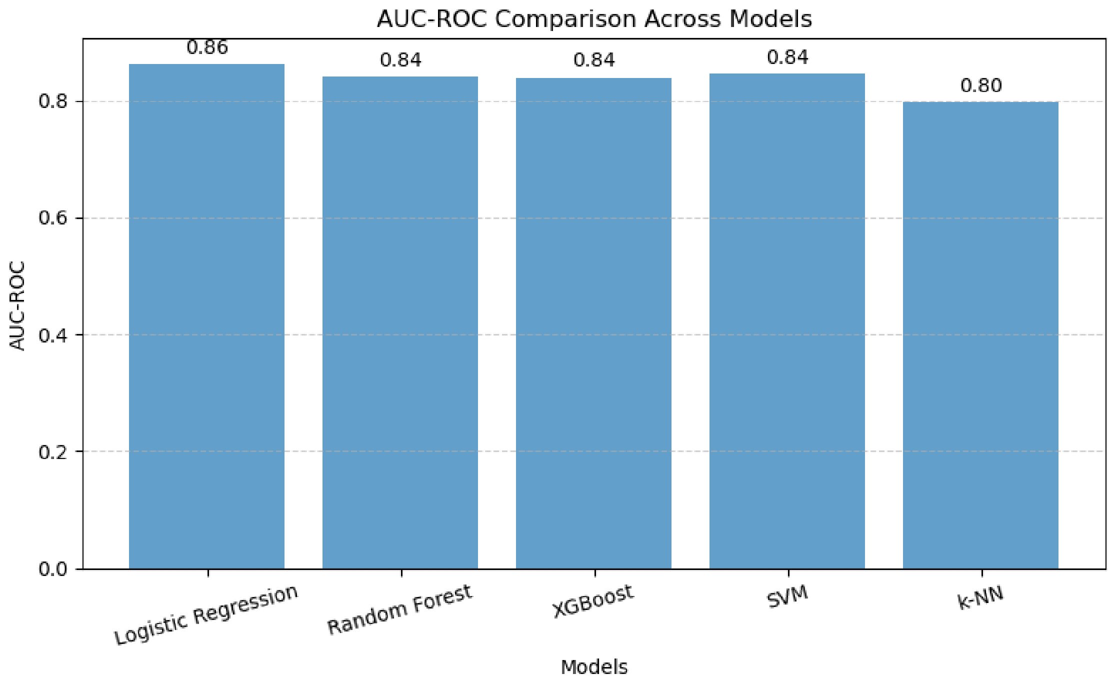
Accuracy Comparison Across Models
Accuracy
Models
0.75
0.79
0.80
0.76
0.77
Logistic Regression
Random Forest
XGBoost
SVM
k-NN



Precision Comparison Across Models
Precision
Models
0.52
0.66
0.64
0.53
0.58
Logistic Regression
Random Forest
XGBoost
SVM
k-NN


Recall Comparison Across Models
0.83
0.46
0.54
0.82
0.52
Recall
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Logistic Regression
Random Forest
XGBoost
SVM
k-NN
Models


F1-Score Comparison Across Models
0.64
0.54
0.58
0.64
0.55
F1-Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Logistic Regression
Random Forest
XGBoost
SVM
k-NN
Models


AUC-ROC Comparison Across Models
0.86
0.84
0.84
0.84
0.80
0.0
0.2
0.4
0.6
0.8
AUC-ROC
Logistic Regression
Random Forest
XGBoost
SVM
k-NN
Models

```
[ ]:
```