Telco Customer Churn Prediction

Data Overview

- **Dataset**: Telco Customer Churn
- **Rows**: 7043
- **Columns**: 21
- **Objective**: Predict whether a customer will churn (**Yes/No**)
    - **Yes** →They have left the service.
    - **No** →They are still an active user of the service.

Libraries And Data Loading

In [61]:
```
# Data manipulation and analysis
import pandas as pd
import numpy as np
from scipy import stats

# Data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Preprocessing
from sklearn.preprocessing import LabelEncoder, StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split

# Handling imbalance
from imblearn.over_sampling import SMOTE

# Machine Learning Models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
from sklearn.svm import SVC

# Model evaluation
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_
from sklearn.metrics import confusion_matrix, classification_report, roc_curve

# Hyperparameter tuning
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV, cross_val_score


df = pd.read_csv('Telco-Customer-Churn.csv')
```

EDA

1) Data Observations & Cleaning

In [62]:
```
df.head()
```

Out[62]:

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

5 rows × 21 columns

In [63]:
```
df.drop('customerID', axis=1, inplace=True)
```

In [64]:
```
df.info()
```

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

Initial Observations

- **Target column**: `Churn` (Yes/No).
- **Feature types**:

- **Categorical** (e.g., `gender`, `InternetService`, `Contract`)
- **Numerical** (e.g., `tenure`, `MonthlyCharges`)
- **Note**: `TotalCharges` is of type **object**, but it should be **numeric** – needs cleaning.

In [65]:
```python
for col in df.columns:
    print(col,len(df[df[col]==" "]))
    print("_________________________________________")
```

```
gender 0
_________________________________________
SeniorCitizen 0
_________________________________________
Partner 0
_________________________________________
Dependents 0
_________________________________________
tenure 0
_________________________________________
PhoneService 0
_________________________________________
MultipleLines 0
_________________________________________
InternetService 0
_________________________________________
OnlineSecurity 0
_________________________________________
OnlineBackup 0
_________________________________________
DeviceProtection 0
_________________________________________
TechSupport 0
_________________________________________
StreamingTV 0
_________________________________________
StreamingMovies 0
_________________________________________
Contract 0
_________________________________________
PaperlessBilling 0
_________________________________________
PaymentMethod 0
_________________________________________
MonthlyCharges 0
_________________________________________
TotalCharges 11
_________________________________________
Churn 0
_________________________________________
```

In [66]:
```python
df["TotalCharges"] = pd.to_numeric(df["TotalCharges"], errors='coerce')
df["TotalCharges"] = df["TotalCharges"].fillna(0)
```

In [67]:
```python
df.isnull().sum()
```

Out[67]:
```
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

In [68]:
```
df.info()
```

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

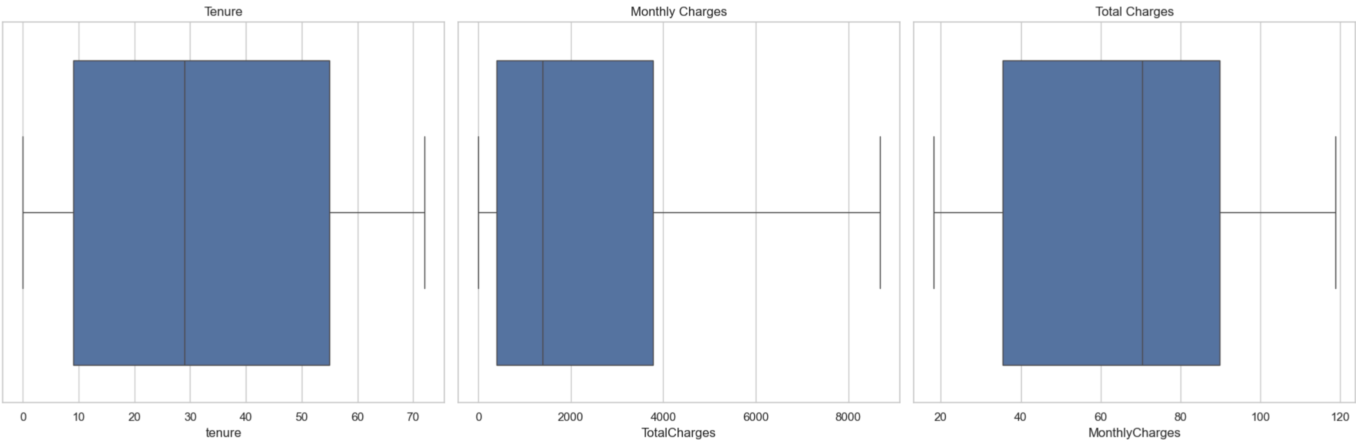
Numerical Features Handling

In [69]:
```
df.describe().T
```

Out[69]:

	count	mean	std	min	25%	50%	75%	max
SeniorCitizen	7043.0	0.162147	0.368612	0.00	0.00	0.00	0.00	1.00
tenure	7043.0	32.371149	24.559481	0.00	9.00	29.00	55.00	72.00
MonthlyCharges	7043.0	64.761692	30.090047	18.25	35.50	70.35	89.85	118.75
TotalCharges	7043.0	2279.734304	2266.794470	0.00	398.55	1394.55	3786.60	8684.80

In [70]:
```
# Set the figure size and layout
fig, axes = plt.subplots(1, 3, figsize=(18, 6))  # 1 row, 3 columns

# Plot each boxplot in a subplot
sns.boxplot(x='tenure', data=df, ax=axes[0])
axes[0].set_title('Tenure')

sns.boxplot(x='MonthlyCharges', data=df, ax=axes[2])
axes[1].set_title('Monthly Charges')

sns.boxplot(x='TotalCharges', data=df, ax=axes[1])
axes[2].set_title('Total Charges')

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



In [71]:
```
def detect_outliers_iqr(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound)]
    print(f"{column}: {len(outliers)} outliers")
    return outliers

# Apply to each numerical feature
for col in ['tenure', 'MonthlyCharges', 'TotalCharges']:
    detect_outliers_iqr(df, col)
```

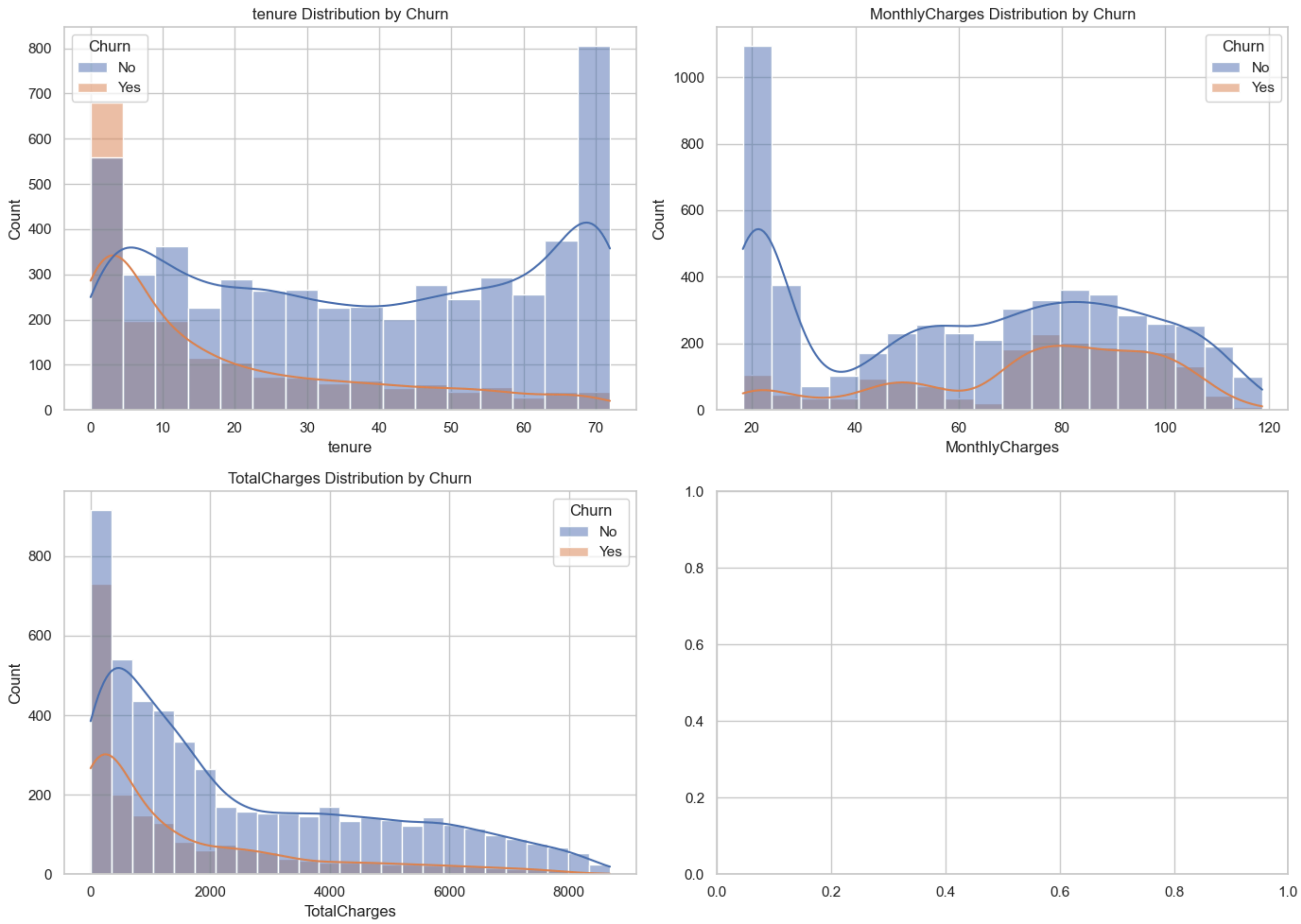
```
tenure: 0 outliers
MonthlyCharges: 0 outliers
TotalCharges: 0 outliers
```

In [72]:
```
from scipy.stats import zscore
import numpy as np

def detect_outliers_zscore(df, column, threshold=3):
    z_scores = zscore(df[column].dropna())
```

```python
    outliers = df[np.abs(z_scores) > threshold]
    print(f"{column}: {len(outliers)} outliers (Z-score > {threshold})")
    return outliers

# Apply to each numerical feature
for col in ['tenure', 'MonthlyCharges', 'TotalCharges']:
    detect_outliers_zscore(df, col)
```

```
tenure: 0 outliers (Z-score > 3)
MonthlyCharges: 0 outliers (Z-score > 3)
TotalCharges: 0 outliers (Z-score > 3)
```

In [73]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Set style
sns.set(style="whitegrid")

# Separate numerical and target
numerical_features = ["tenure", "MonthlyCharges", "TotalCharges"]

# Plot distributions of numerical features
fig, axes = plt.subplots(2, 2, figsize=(14, 10))
for i, feature in enumerate(numerical_features):
    row, col = divmod(i, 2)
    sns.histplot(data=df, x=feature, hue="Churn", kde=True, ax=axes[row][col])
    axes[row][col].set_title(f"{feature} Distribution by Churn")

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



Visual Analysis

1. Tenure vs Churn

- Customers with **low tenure** are more likely to churn.
- Longer tenure is associated with customer retention.

2. **Monthly Charges vs Churn**

- Churn is higher among customers with **high monthly charges**.
- Indicates pricing could be a churn driver.

3. **Total Charges vs Churn**

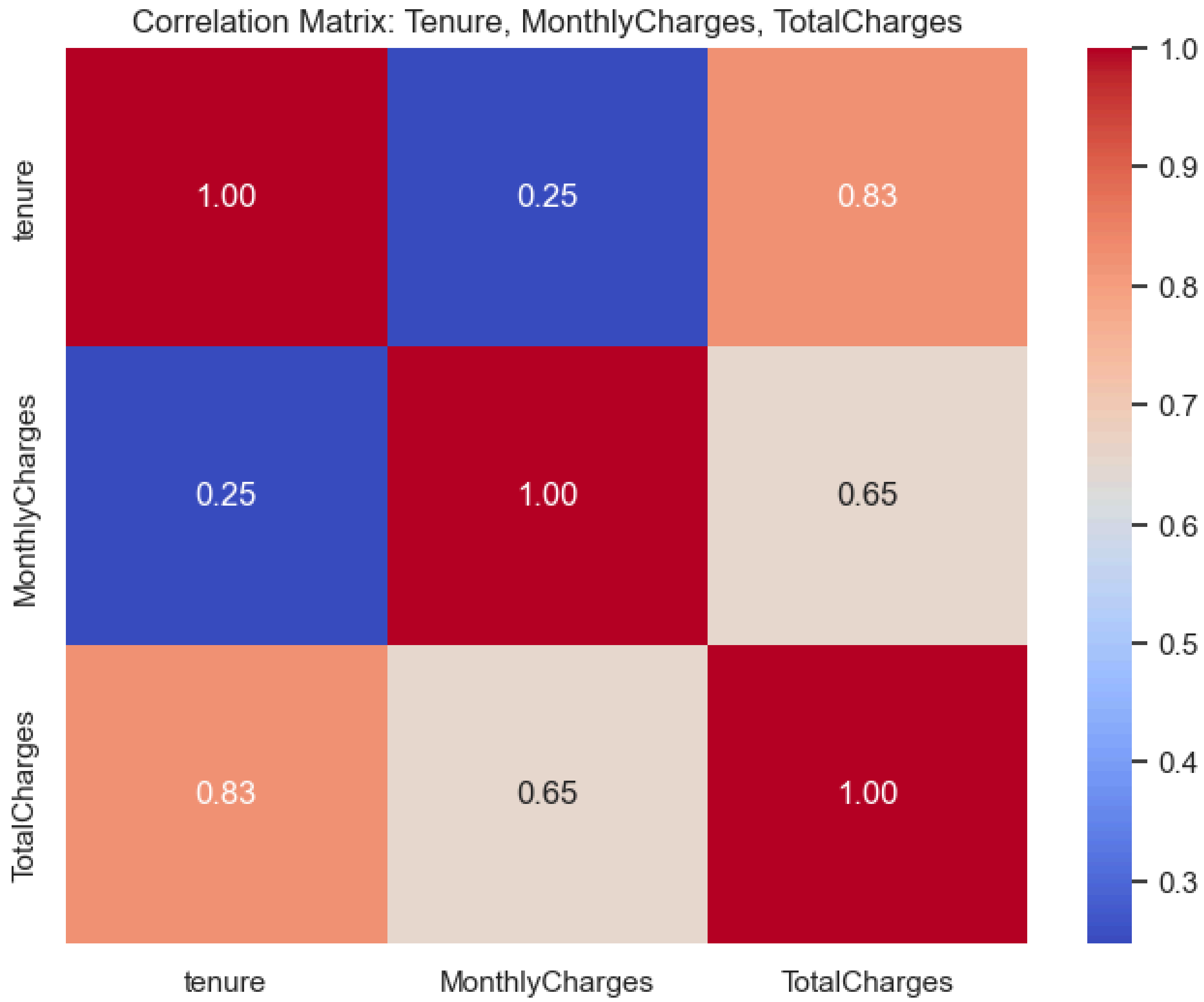
- Churners mostly have **low total charges**, possibly due to short tenure.
- Reinforces the idea that new users are more likely to leave.

Key Takeaways

- **Tenure**, **MonthlyCharges**, and **TotalCharges** show strong patterns with churn.
- Recently joined or low-spending users have a higher churn risk.
- These features may be **important predictors** in the classification model.

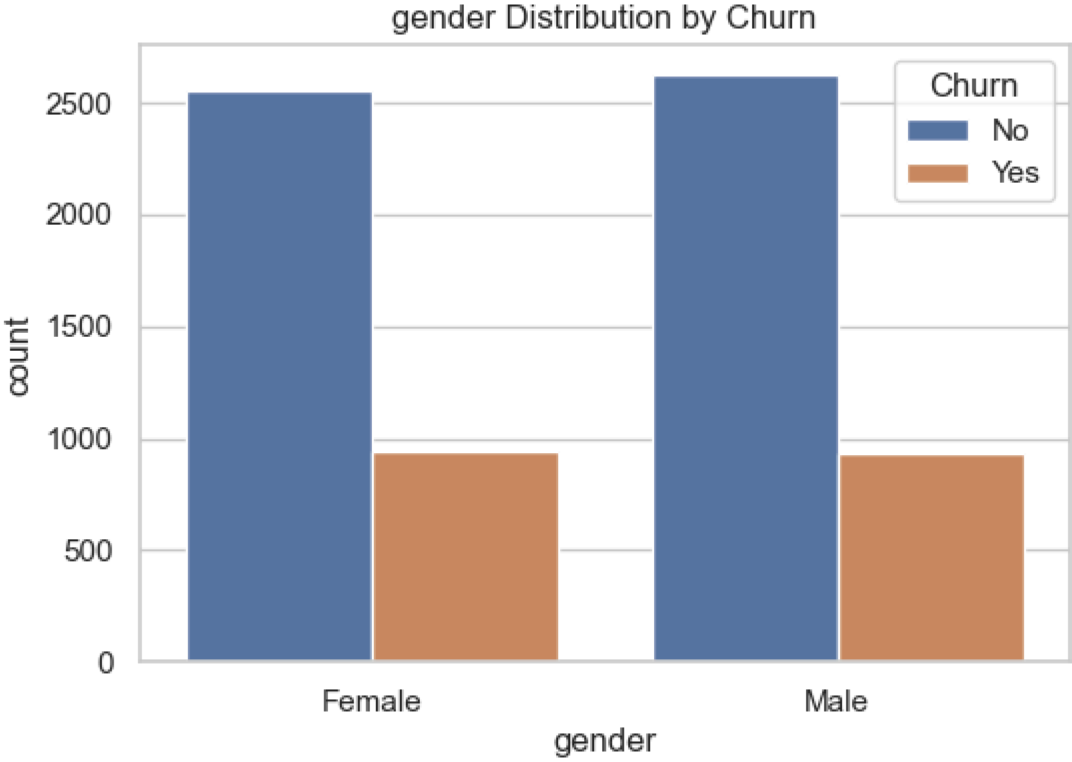
In [74]:

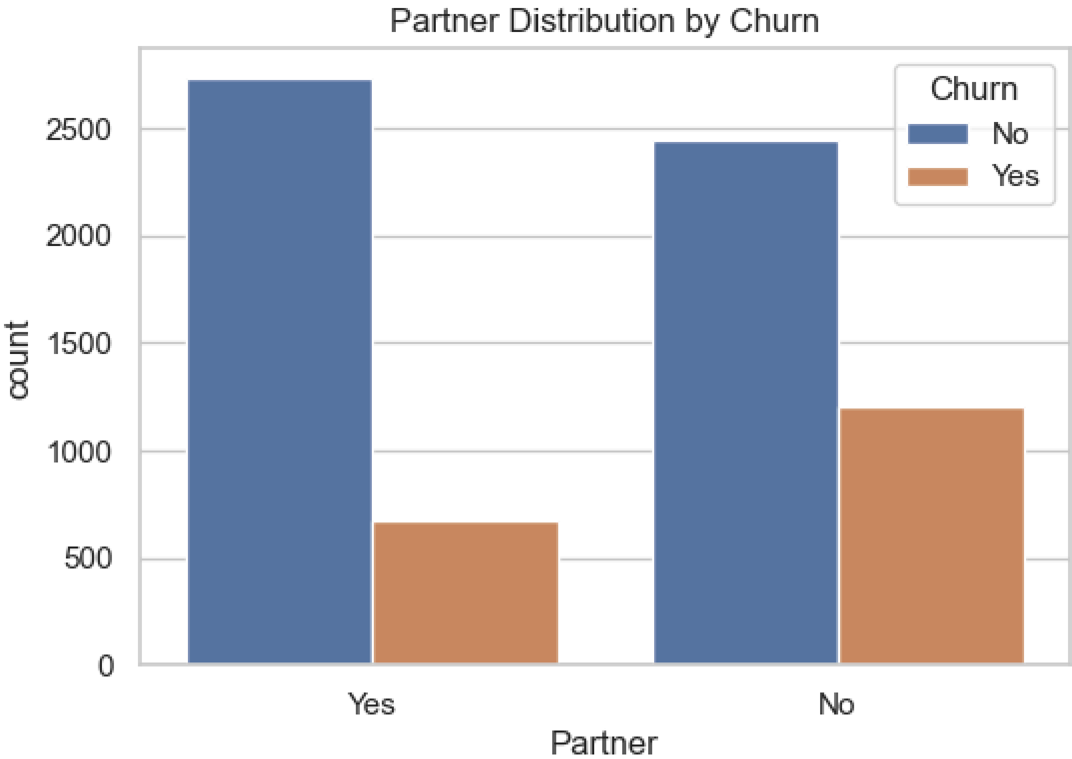
```
# Select relevant features
corr_features = df[['tenure', 'MonthlyCharges', 'TotalCharges']]

# Calculate correlation matrix
corr_matrix = corr_features.corr()

# Plot heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', cbar=True)
plt.title('Correlation Matrix: Tenure, MonthlyCharges, TotalCharges')
plt.show()
```

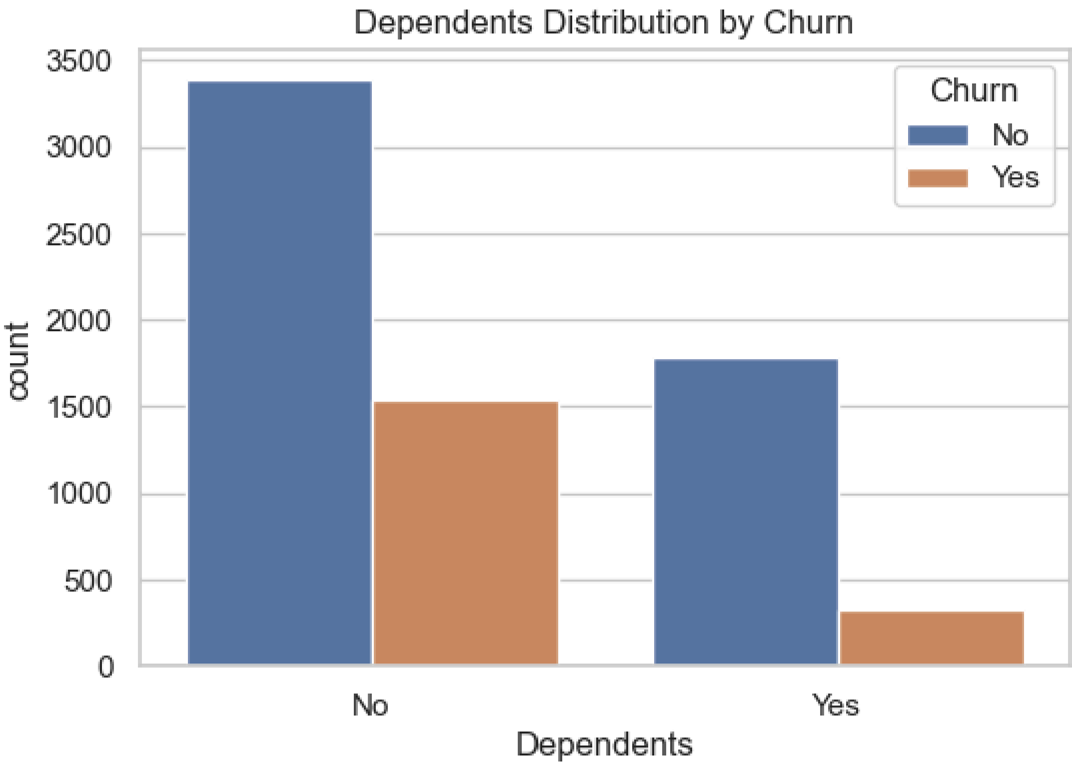


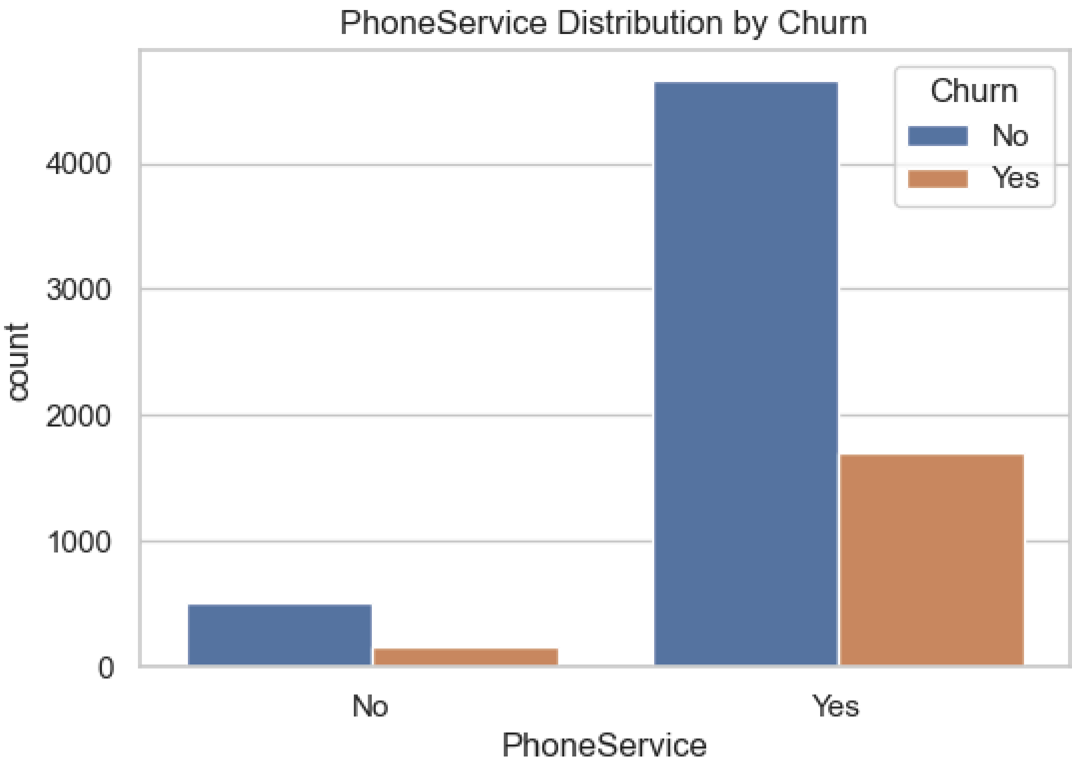
Categorical Features Handling

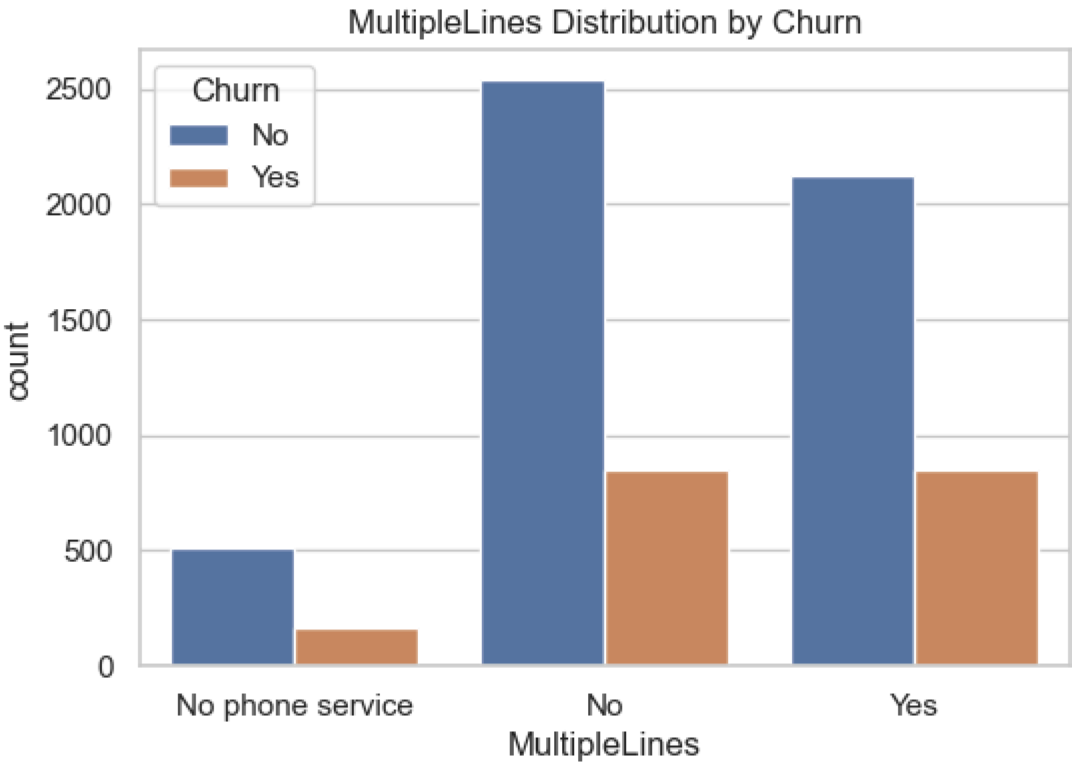
In [75]:
```
Categorical_cols=df.select_dtypes(include='object').columns.to_list()+['SeniorCitizen']

for col in Categorical_cols:
    plt.figure(figsize=(6,4))
    sns.countplot(data=df,x=col,hue='Churn')
    plt.title(f"{col} Distribution by Churn")
    plt.show()
```

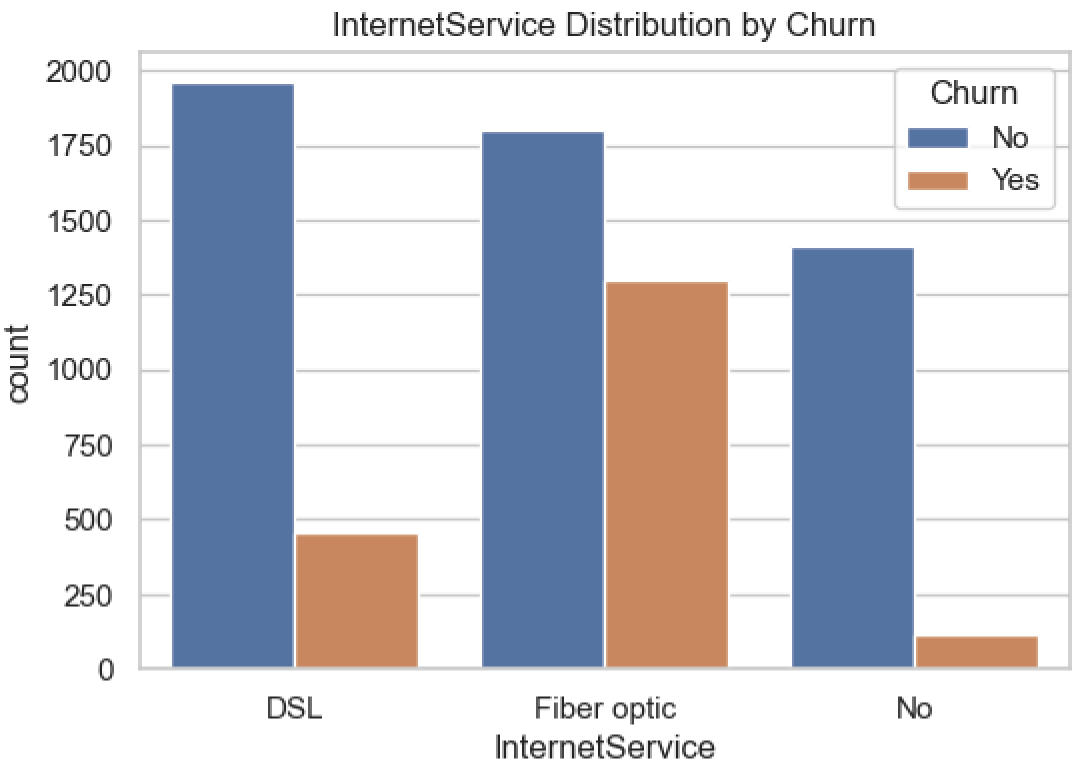

gender Distribution by Churn
count
2500
2000
1500
1000
500
0
Churn
No
Yes
Female
Male
gender

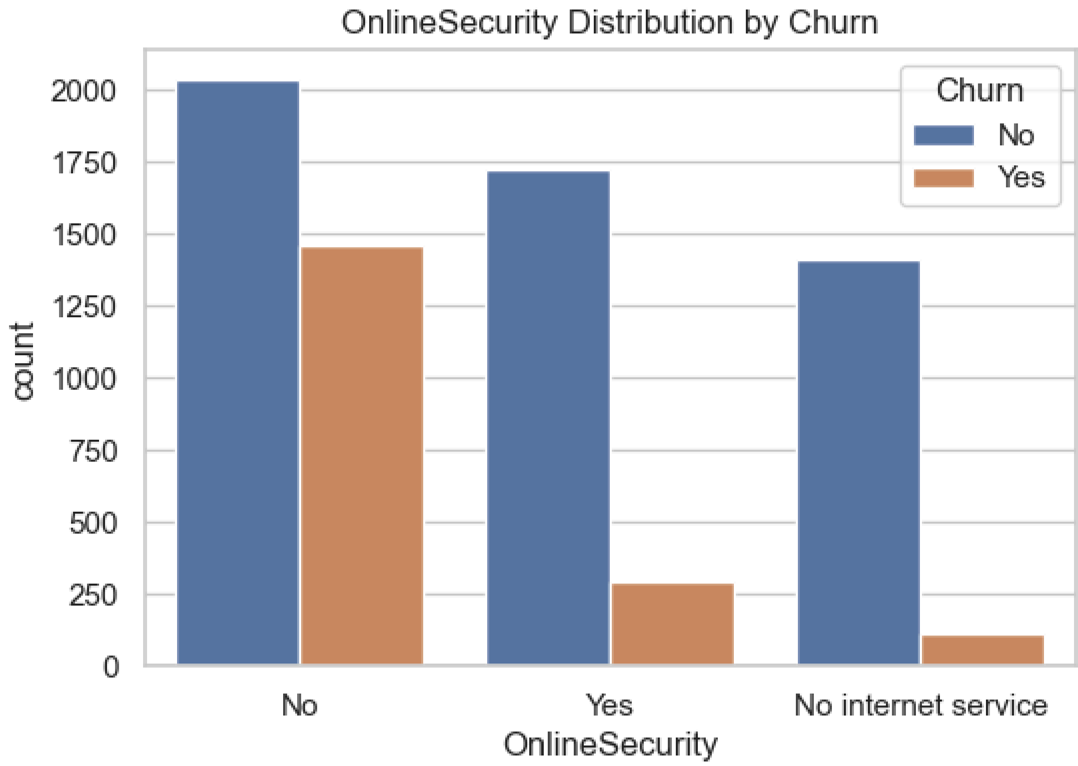

Partner Distribution by Churn
count
2500
2000
1500
1000
500
0
Churn
No
Yes
Yes
No
Partner

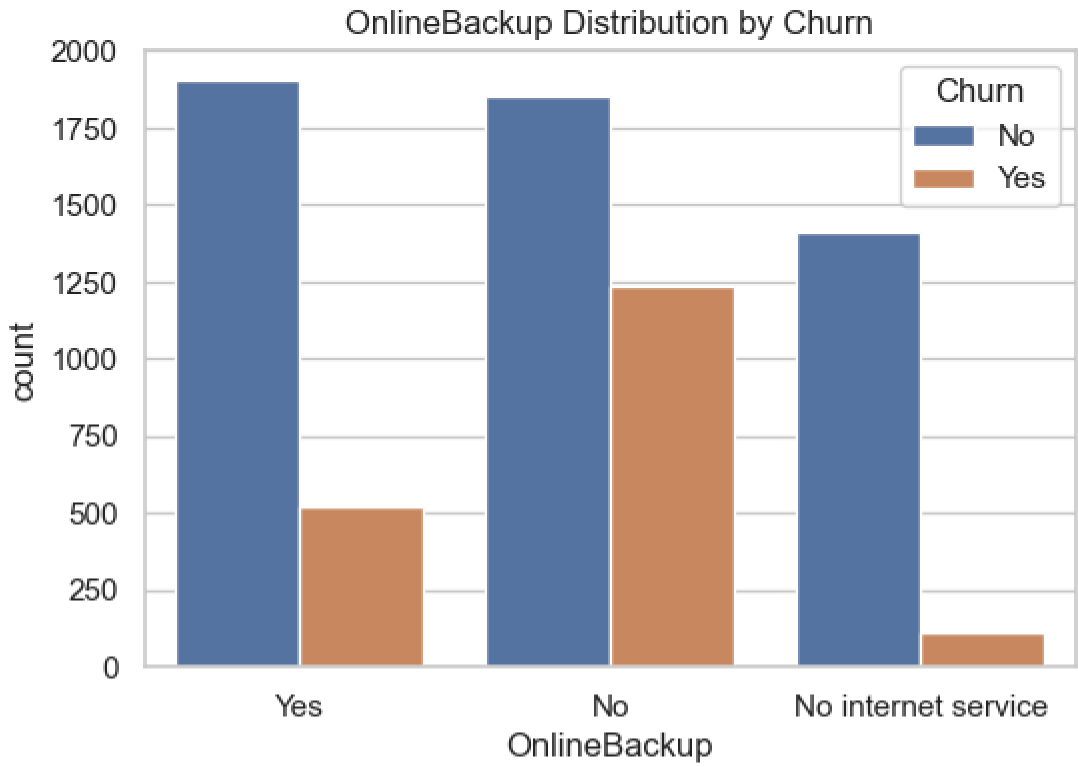

Dependents Distribution by Churn
count
0
500
1000
1500
2000
2500
3000
3500
Churn
No
Yes
No
Yes
Dependents

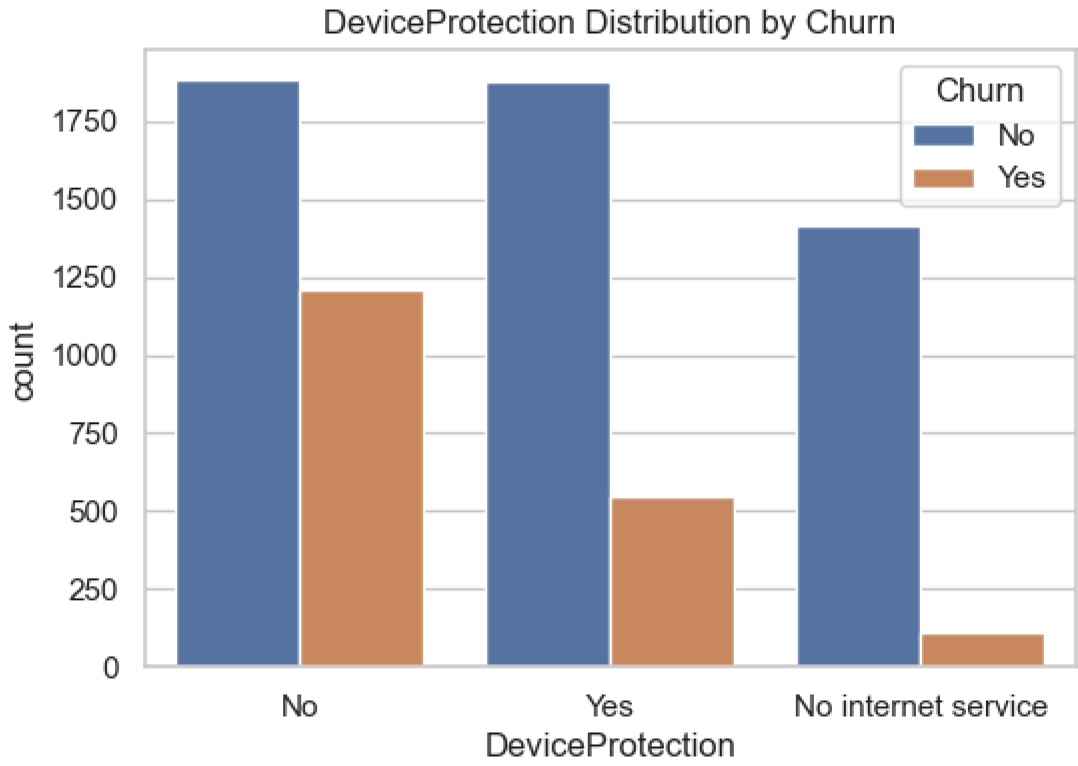

PhoneService Distribution by Churn
count
0
1000
2000
3000
4000
Churn
No
Yes
No
Yes
PhoneService

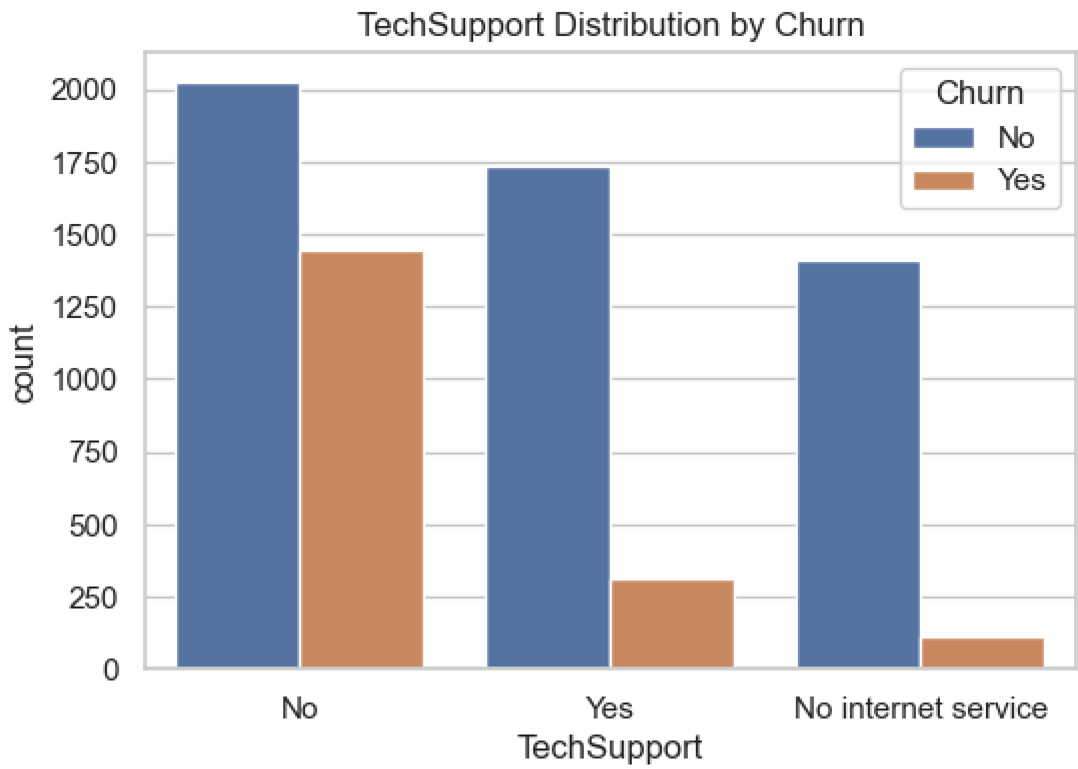

MultipleLines Distribution by Churn
Churn
No
Yes
count
2500
2000
1500
1000
500
0
No phone service
No
Yes
MultipleLines

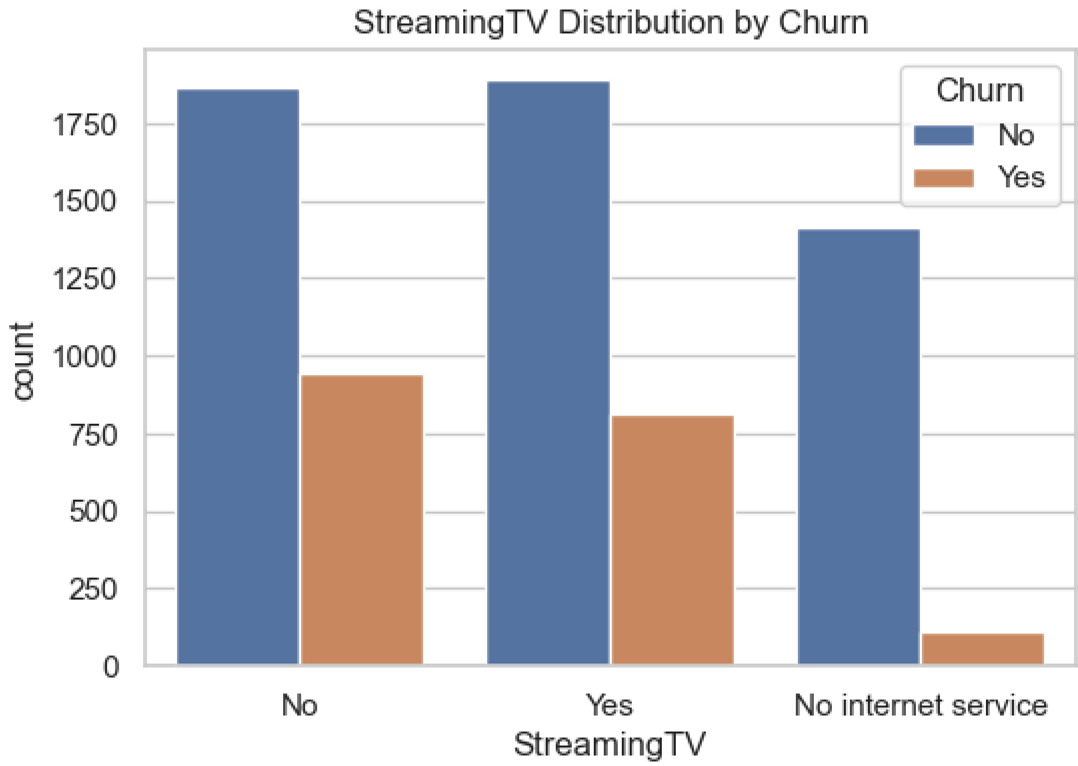

InternetService Distribution by Churn
Churn
No
Yes
count
2000
1750
1500
1250
1000
750
500
250
0
DSL
Fiber optic
No
InternetService

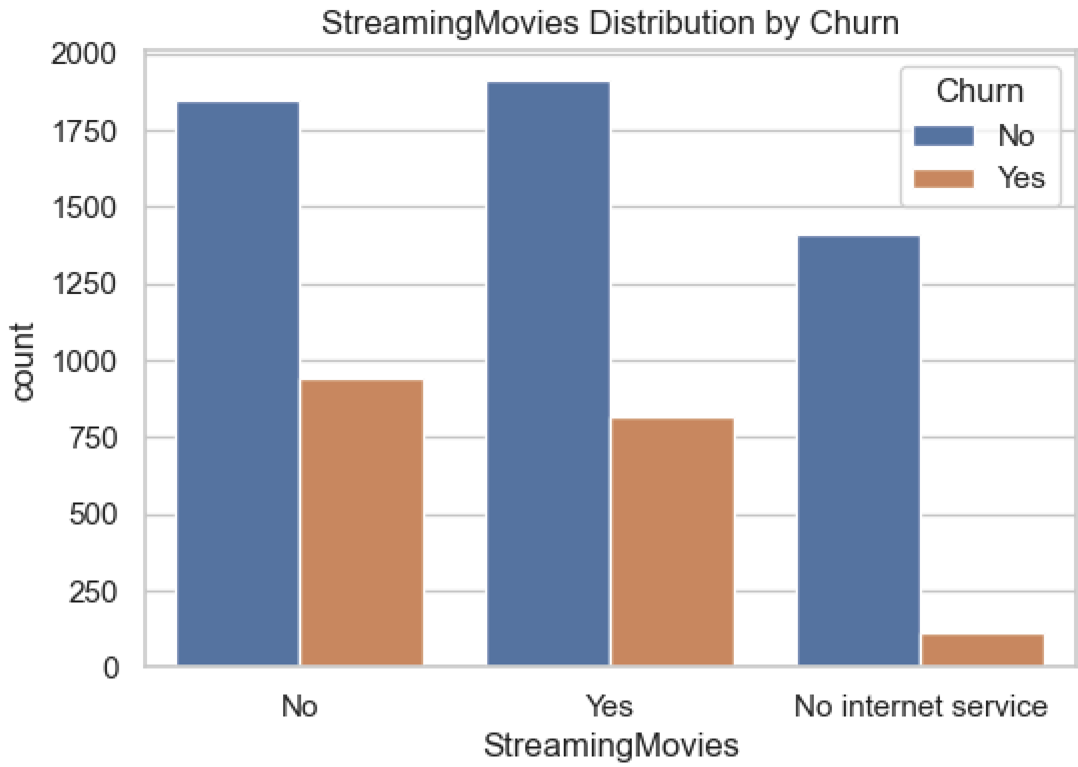

OnlineSecurity Distribution by Churn
Churn
No
Yes
count
2000
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
OnlineSecurity

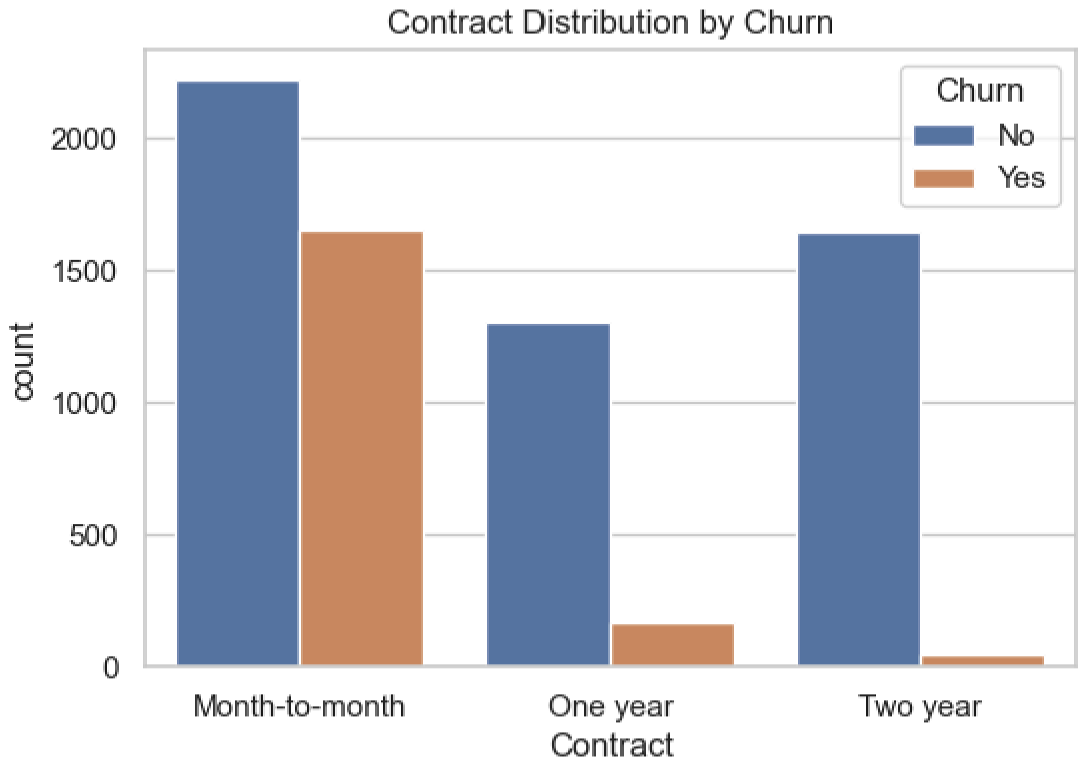

OnlineBackup Distribution by Churn
Churn
No
Yes
count
2000
1750
1500
1250
1000
750
500
250
0
Yes
No
No internet service
OnlineBackup

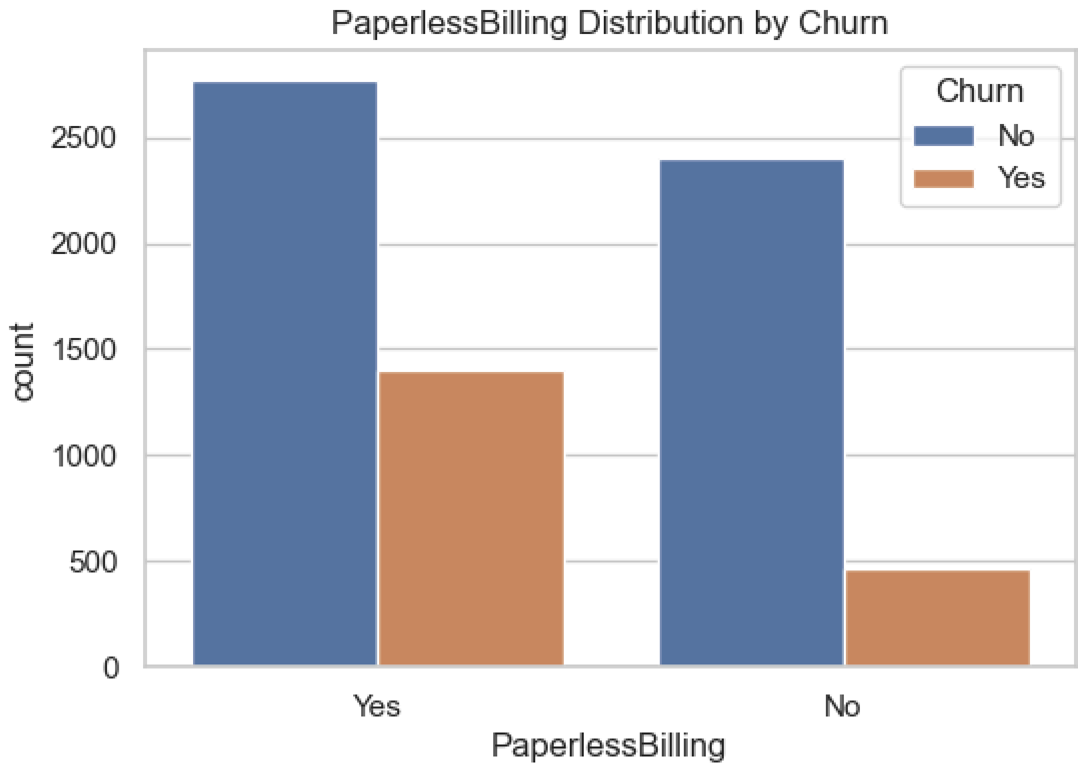

DeviceProtection Distribution by Churn
Churn
No
Yes
count
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
DeviceProtection

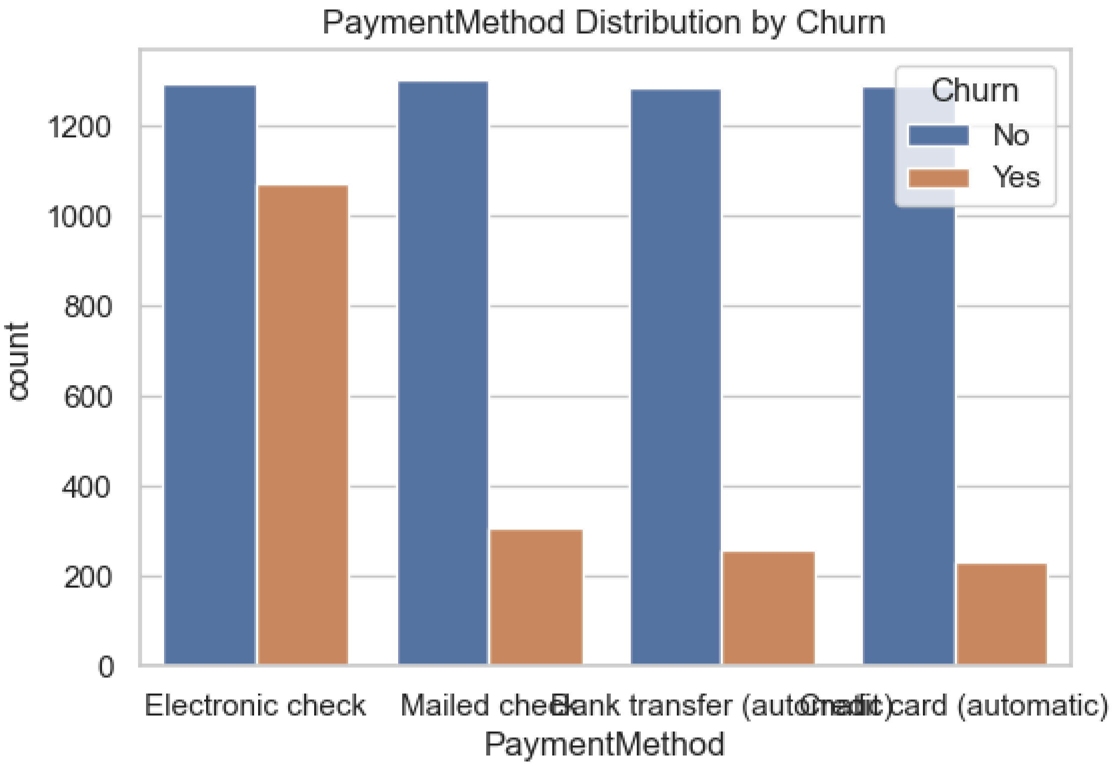

TechSupport Distribution by Churn
Churn
No
Yes
count
2000
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
TechSupport

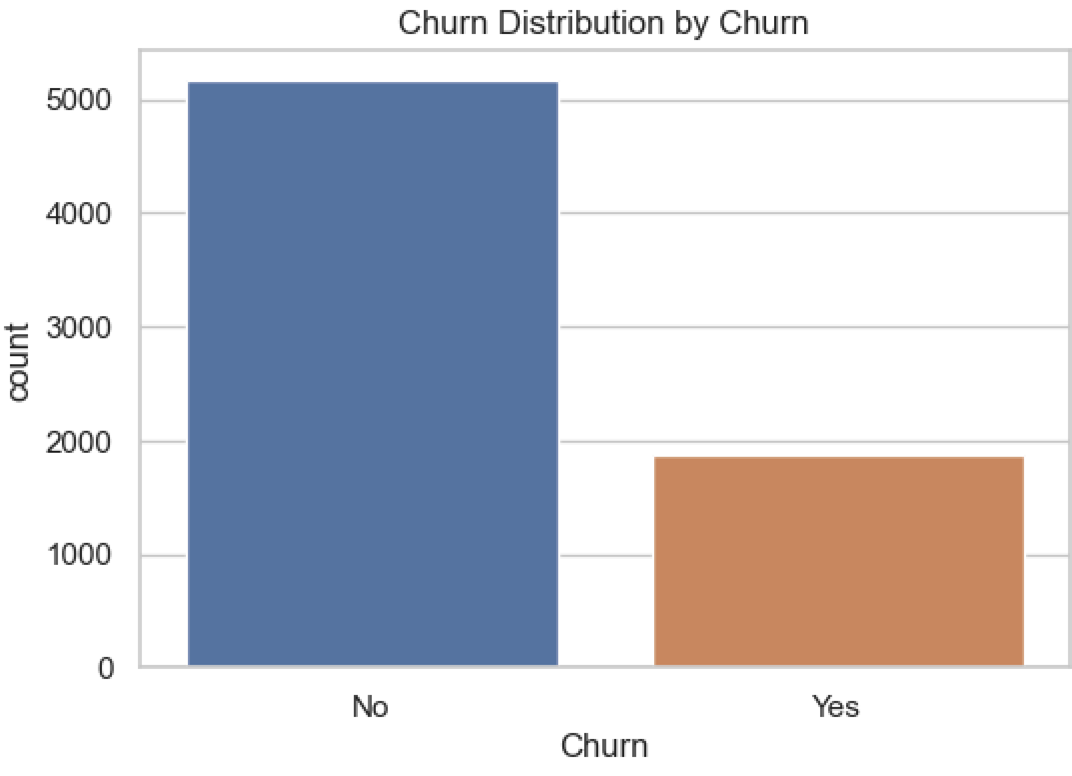

StreamingTV Distribution by Churn
count
1750
1500
1250
1000
750
500
250
0
Churn
No
Yes
No
Yes
No internet service
StreamingTV

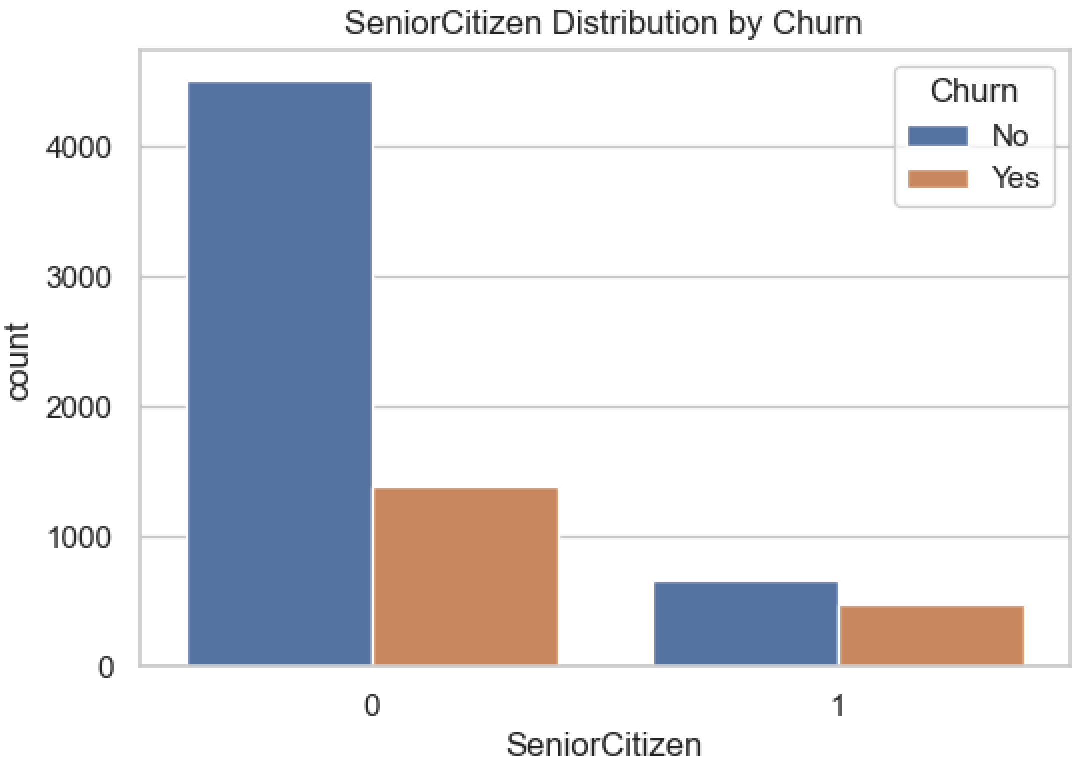

StreamingMovies Distribution by Churn
count
2000
1750
1500
1250
1000
750
500
250
0
Churn
No
Yes
No
Yes
No internet service
StreamingMovies


Contract Distribution by Churn
Churn
No
Yes
count
2000
1500
1000
500
0
Month-to-month
One year
Two year
Contract


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


PaymentMethod Distribution by Churn
Churn
No
Yes
count
0
200
400
600
800
1000
1200
Electronic check
Mailed check
Bank transfer (automatic)
Credit card (automatic)
PaymentMethod


Churn Distribution by Churn
count
0
1000
2000
3000
4000
5000
No
Yes
Churn



In [76]:
```
# Check data types and unique values in each column
unique_values = df.nunique()
unique_values
```

Out[76]:
```
gender                 2
SeniorCitizen          2
Partner                2
Dependents             2
tenure                73
PhoneService           2
MultipleLines          3
InternetService        3
OnlineSecurity         3
OnlineBackup           3
DeviceProtection       3
TechSupport            3
StreamingTV            3
StreamingMovies        3
Contract               3
PaperlessBilling       2
PaymentMethod          4
MonthlyCharges      1585
TotalCharges        6531
Churn                  2
dtype: int64
```

In [77]:
```
for col in df.columns:
    if col not in ['tenure','MonthlyCharges','TotalCharges']:
        print(col,df[col].unique())
        print("_________________________________________")
```

```
gender ['Female' 'Male']
____________________________________________
SeniorCitizen [0 1]
____________________________________________
Partner ['Yes' 'No']
____________________________________________
Dependents ['No' 'Yes']
____________________________________________
PhoneService ['No' 'Yes']
____________________________________________
MultipleLines ['No phone service' 'No' 'Yes']
____________________________________________
InternetService ['DSL' 'Fiber optic' 'No']
____________________________________________
OnlineSecurity ['No' 'Yes' 'No internet service']
____________________________________________
OnlineBackup ['Yes' 'No' 'No internet service']
____________________________________________
DeviceProtection ['No' 'Yes' 'No internet service']
____________________________________________
TechSupport ['No' 'Yes' 'No internet service']
____________________________________________
StreamingTV ['No' 'Yes' 'No internet service']
____________________________________________
StreamingMovies ['No' 'Yes' 'No internet service']
____________________________________________
Contract ['Month-to-month' 'One year' 'Two year']
____________________________________________
PaperlessBilling ['Yes' 'No']
____________________________________________
PaymentMethod ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
____________________________________________
Churn ['No' 'Yes']
____________________________________________
```

Feature Engineering

In [78]:
```
df['Churn'] = df['Churn'].map({'No': 0, 'Yes': 1})
```

In [ ]:
```

```

In [79]:
```
encoders={}

for column in Categorical_cols:
    label_encoder = LabelEncoder()
    df[column] = label_encoder.fit_transform(df[column])
    encoders[column]=label_encoder
```

In [80]:
```
encoders
```

Out[80]:
```
{'gender': LabelEncoder(),
 'Partner': LabelEncoder(),
 'Dependents': LabelEncoder(),
 'PhoneService': LabelEncoder(),
 'MultipleLines': LabelEncoder(),
 'InternetService': LabelEncoder(),
 'OnlineSecurity': LabelEncoder(),
 'OnlineBackup': LabelEncoder(),
 'DeviceProtection': LabelEncoder(),
 'TechSupport': LabelEncoder(),
 'StreamingTV': LabelEncoder(),
 'StreamingMovies': LabelEncoder(),
 'Contract': LabelEncoder(),
 'PaperlessBilling': LabelEncoder(),
 'PaymentMethod': LabelEncoder(),
 'Churn': LabelEncoder(),
 'SeniorCitizen': LabelEncoder()}
```

In [81]:
```
df
```

Out[81]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetServic
0	0	0	1	0	1	0	1	
1	1	0	0	0	34	1	0	
2	1	0	0	0	2	1	0	
3	1	0	0	0	45	0	1	
4	0	0	0	0	2	1	0	
...	...	...	...	...	...	...	...	
7038	1	0	1	1	24	1	2	
7039	0	0	1	1	72	1	2	
7040	0	0	1	1	11	0	1	
7041	1	1	1	0	4	1	2	
7042	1	0	0	0	66	1	0	

7043 rows × 20 columns

In [82]:
```
scaler=StandardScaler()
df[numerical_features]=scaler.fit_transform(df[numerical_features])
```

In [83]:
```
df
```

Out[83]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetSer
0	0	0	1	0	-1.277445	0	1	
1	1	0	0	0	0.066327	1	0	
2	1	0	0	0	-1.236724	1	0	
3	1	0	0	0	0.514251	0	1	
4	0	0	0	0	-1.236724	1	0	
...	...	...	...	...	...	...	...	
7038	1	0	1	1	-0.340876	1	2	
7039	0	0	1	1	1.613701	1	2	
7040	0	0	1	1	-0.870241	0	1	
7041	1	1	1	0	-1.155283	1	2	
7042	1	0	0	0	1.369379	1	0	

7043 rows × 20 columns

In [84]:

```
# Features and target
x = df.drop("Churn", axis=1)
y = df["Churn"]
```

In [ ]:

```
# Split the data first
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, str

# Then scale the data (no leakage)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [85]:

```
# 1. SMOTE (on training data only)
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled, y_train)

# 2. Define classifiers
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "XGBoost": XGBClassifier(eval_metric='logloss'),
    "SVM": SVC(probability=True),
    "KNN": KNeighborsClassifier(),
    "Naive Bayes": GaussianNB(),
    "AdaBoost": AdaBoostClassifier()
}

# 3. Train all models
trained_models = {}

for name, model in models.items():
    model.fit(X_train_resampled, y_train_resampled)
    trained_models[name] = model
    print(f"{name} trained successfully.")
```

```
Logistic Regression trained successfully.
Decision Tree trained successfully.
Random Forest trained successfully.
XGBoost trained successfully.
SVM trained successfully.
KNN trained successfully.
Naive Bayes trained successfully.
AdaBoost trained successfully.
```

In [86]:

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_

# Initialize dictionary to store performance metrics
model_performance_before_HT = {}

# Evaluate each model
for name, model in trained_models.items():
    y_pred = model.predict(X_test_scaled)
    y_prob = model.predict_proba(X_test_scaled)[:, 1]  # For AUC-ROC

    # Calculate performance metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    auc_roc = roc_auc_score(y_test, y_prob)

    # Store metrics
    model_performance_before_HT[name] = {
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1-Score': f1,
        'AUC-ROC': auc_roc
    }

    # Print results
    print(f"\n{name} Performance:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-Score: {f1:.4f}")
    print(f"AUC-ROC: {auc_roc:.4f}")
```

```
Logistic Regression Performance:
Accuracy: 0.7424
Precision: 0.5095
Recall: 0.7888
F1-Score: 0.6191
AUC-ROC: 0.8391

Decision Tree Performance:
Accuracy: 0.7218
Precision: 0.4799
Recall: 0.5749
F1-Score: 0.5231
AUC-ROC: 0.6747

Random Forest Performance:
Accuracy: 0.7736
Precision: 0.5700
Recall: 0.5989
F1-Score: 0.5841
AUC-ROC: 0.8204

XGBoost Performance:
Accuracy: 0.7779
Precision: 0.5796
Recall: 0.5936
F1-Score: 0.5865
AUC-ROC: 0.8146

SVM Performance:
Accuracy: 0.7466
Precision: 0.5167
Recall: 0.7032
F1-Score: 0.5957
AUC-ROC: 0.8097

KNN Performance:
Accuracy: 0.6778
Precision: 0.4340
Recall: 0.7032
F1-Score: 0.5367
AUC-ROC: 0.7473

Naive Bayes Performance:
Accuracy: 0.7268
Precision: 0.4907
Recall: 0.7754
F1-Score: 0.6010
AUC-ROC: 0.8187

AdaBoost Performance:
Accuracy: 0.7388
Precision: 0.5054
Recall: 0.7567
F1-Score: 0.6060
AUC-ROC: 0.8335
```

In [87]:

```
import pandas as pd

# Convert model performance dictionary into DataFrame
performance_df = pd.DataFrame(model_performance_before_HT).T

# Display the performance metrics for each model
performance_df
```

Out[87]:

	Accuracy	Precision	Recall	F1-Score	AUC-ROC
Logistic Regression	0.742370	0.509499	0.788770	0.619098	0.839138
Decision Tree	0.721789	0.479911	0.574866	0.523114	0.674667
Random Forest	0.773598	0.569975	0.598930	0.584094	0.820356
XGBoost	0.777857	0.579634	0.593583	0.586526	0.814600
SVM	0.746629	0.516699	0.703209	0.595696	0.809721
KNN	0.677786	0.433993	0.703209	0.536735	0.747265
Naive Bayes	0.726757	0.490694	0.775401	0.601036	0.818722
AdaBoost	0.738822	0.505357	0.756684	0.605996	0.833482

In [88]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_

# 1. Initialize base models (XGBoost removed)
base_models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Random Forest': RandomForestClassifier(),
    'Naive Bayes': GaussianNB(),
    'AdaBoost': AdaBoostClassifier()
}

# 2. Define hyperparameter grids (XGBoost removed)
param_grids = {
    'Logistic Regression': {
        'C': [0.01, 0.1, 1, 10],
        'penalty': ['l1', 'l2'],
        'solver': ['liblinear']
    },
    'Random Forest': {
        'n_estimators': [100, 200],
        'max_depth': [None, 10, 20],
        'min_samples_split': [2, 5]
    },
    'Naive Bayes': {
        'var_smoothing': [1e-09, 1e-08, 1e-07]
    },
    'AdaBoost': {
        'n_estimators': [50, 100, 200],
        'learning_rate': [0.01, 0.1, 1]
    }
}

# 3. Tune and store best models
trained_models = {}
for name, model in base_models.items():
    print(f"\nTuning {name}...")
    grid = GridSearchCV(model, param_grids[name], cv=5, scoring='f1', n_jobs=-1)
    grid.fit(X_train_scaled, y_train)
    trained_models[name] = grid.best_estimator_
    print(f"Best parameters for {name}: {grid.best_params_}")

# 4. Evaluate tuned models
```

```python
model_performance = {}

for name, model in trained_models.items():
    y_pred = model.predict(X_test_scaled)

    # Check if model supports predict_proba
    if hasattr(model, "predict_proba"):
        y_prob = model.predict_proba(X_test_scaled)[:, 1]
    else:
        y_prob = y_pred

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    auc_roc = roc_auc_score(y_test, y_prob)

    model_performance[name] = {
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1-Score': f1,
        'AUC-ROC': auc_roc
    }

    print(f"\n{name} Performance:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-Score: {f1:.4f}")
    print(f"AUC-ROC: {auc_roc:.4f}")
```

```
Tuning Logistic Regression...
Best parameters for Logistic Regression: {'C': 1, 'penalty': 'l1', 'solver': 'liblinear'}

Tuning Random Forest...
Best parameters for Random Forest: {'max_depth': 10, 'min_samples_split': 5, 'n_estimators': 2
00}

Tuning Naive Bayes...
Best parameters for Naive Bayes: {'var_smoothing': 1e-09}

Tuning AdaBoost...
Best parameters for AdaBoost: {'learning_rate': 1, 'n_estimators': 200}

Logistic Regression Performance:
Accuracy: 0.7991
Precision: 0.6426
Recall: 0.5481
F1-Score: 0.5916
AUC-ROC: 0.8404

Random Forest Performance:
Accuracy: 0.7970
Precision: 0.6507
Recall: 0.5080
F1-Score: 0.5706
AUC-ROC: 0.8394

Naive Bayes Performance:
Accuracy: 0.7466
Precision: 0.5160
Recall: 0.7326
F1-Score: 0.6055
AUC-ROC: 0.8201

AdaBoost Performance:
Accuracy: 0.7949
Precision: 0.6349
Recall: 0.5348
F1-Score: 0.5806
AUC-ROC: 0.8447
```

In [89]:

```
# Convert model performance dictionary into DataFrame
performance = pd.DataFrame(model_performance).T

# Display the performance metrics for each model
performance
```

Out[89]:

	Accuracy	Precision	Recall	F1-Score	AUC-ROC
Logistic Regression	0.799148	0.642633	0.548128	0.591631	0.840445
Random Forest	0.797019	0.650685	0.508021	0.570571	0.839440
Naive Bayes	0.746629	0.516008	0.732620	0.605525	0.820104
AdaBoost	0.794890	0.634921	0.534759	0.580552	0.844713

In [90]:

```
# Initialize dictionary to store training performance metrics
model_performance_train = {}

# Evaluate each model on the training set
for name, model in trained_models.items():
    y_pred_train = model.predict(X_train_scaled)
```

```python
    if hasattr(model, "predict_proba"):
        y_prob_train = model.predict_proba(X_train_scaled)[:, 1]
    else:
        y_prob_train = y_pred_train

    accuracy = accuracy_score(y_train, y_pred_train)
    precision = precision_score(y_train, y_pred_train)
    recall = recall_score(y_train, y_pred_train)
    f1 = f1_score(y_train, y_pred_train)
    auc_roc = roc_auc_score(y_train, y_prob_train)

    model_performance_train[name] = {
        'Accuracy': round(accuracy, 4),
        'Precision': round(precision, 4),
        'Recall': round(recall, 4),
        'F1-Score': round(f1, 4),
        'AUC-ROC': round(auc_roc, 4)
    }

# Optionally show as table
import pandas as pd
performance_train_df = pd.DataFrame(model_performance_train).T
print("\n📊 Model Training Performance Summary:\n")
print(performance_train_df)
```

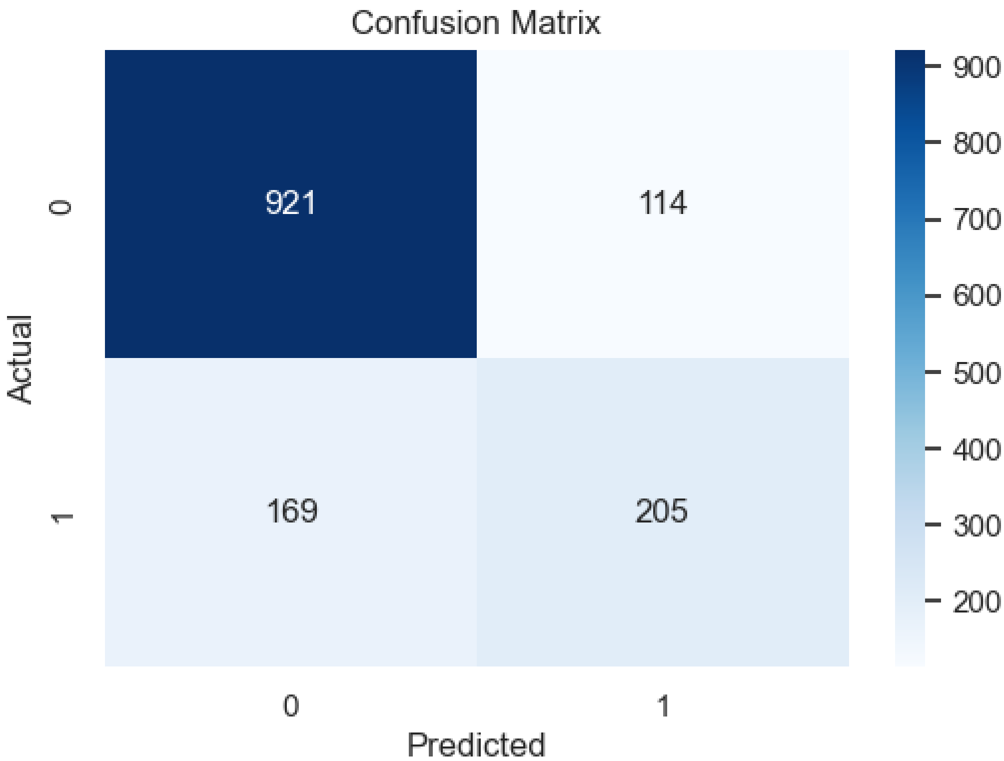
```
📊 Model Training Performance Summary:

                     Accuracy  Precision  Recall  F1-Score  AUC-ROC
Logistic Regression    0.8042     0.6571  0.5485    0.5979   0.8476
Random Forest          0.8791     0.8304  0.6843    0.7503   0.9531
Naive Bayes            0.7563     0.5293  0.7378    0.6164   0.8263
AdaBoost               0.8080     0.6664  0.5532    0.6045   0.8544
```

In [91]:
```python
import joblib
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression

# Use the best parameters found during GridSearchCV
best_lr = LogisticRegression(C=1, penalty='l1', solver='liblinear', max_iter=1000)

# Create pipeline with scaler + model
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('model', best_lr)
])

# Fit on original training data
pipeline.fit(X_train, y_train)

# Save the pipeline
joblib.dump(pipeline, 'final_model_pipeline.pkl')
print("Final model pipeline saved as 'final_model_pipeline.pkl'")
```

```
Final model pipeline saved as 'final_model_pipeline.pkl'
```

In [92]:
```python
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns

# Predict and evaluate
y_pred = pipeline.predict(X_test)
y_prob = pipeline.predict_proba(X_test)[:, 1]
```

```python
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Classification Report
print("Classification Report:\n")
print(classification_report(y_test, y_pred))

# AUC-ROC
roc = roc_auc_score(y_test, y_prob)
print(f"AUC-ROC: {roc:.4f}")
```

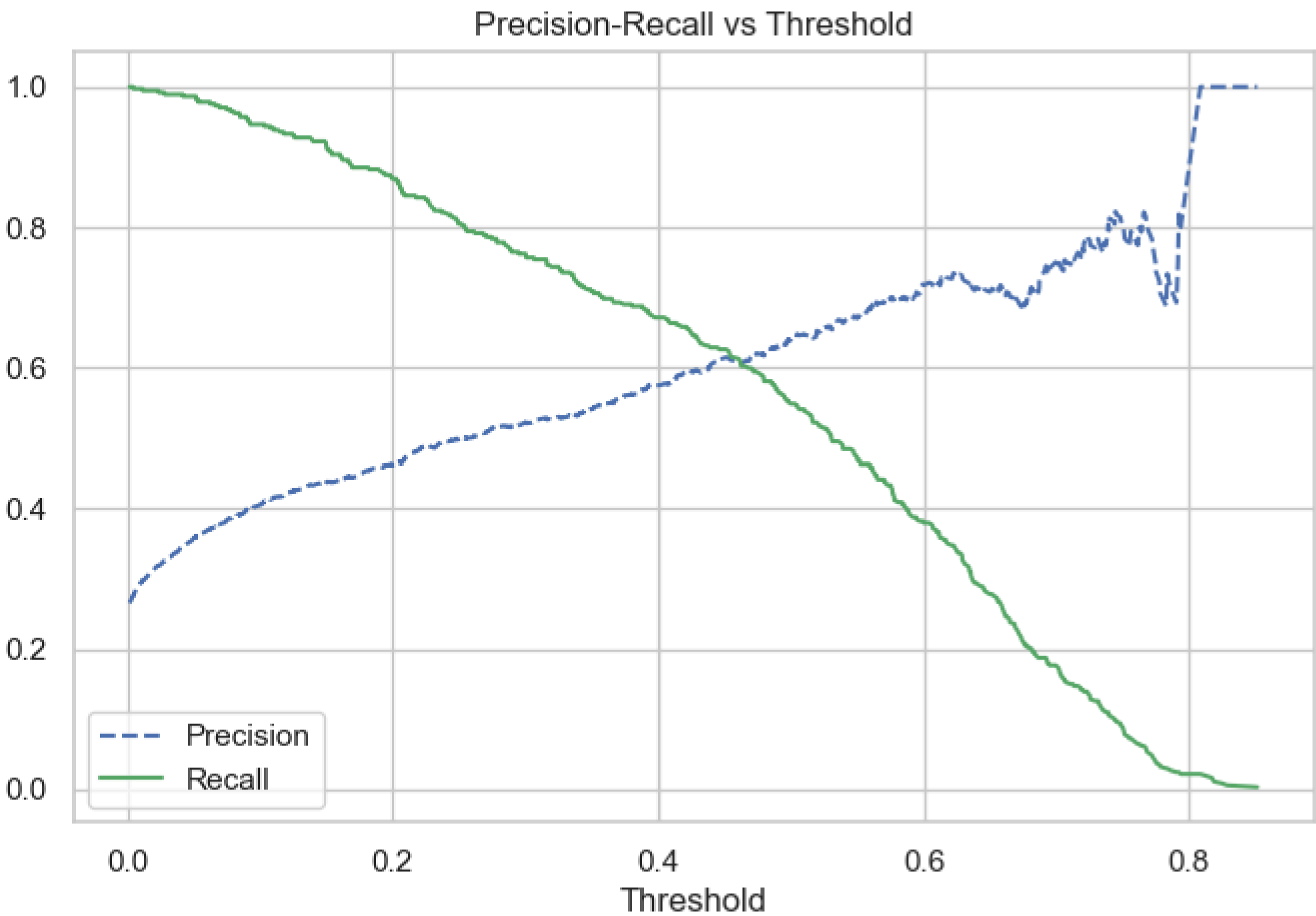



```
Classification Report:

              precision    recall  f1-score   support

           0       0.84      0.89      0.87      1035
           1       0.64      0.55      0.59       374

    accuracy                           0.80      1409
   macro avg       0.74      0.72      0.73      1409
weighted avg       0.79      0.80      0.79      1409

AUC-ROC: 0.8404
```

In [93]:

```python
from sklearn.metrics import precision_recall_curve

precision, recall, thresholds = precision_recall_curve(y_test, y_prob)

# Plot
plt.figure(figsize=(8, 5))
plt.plot(thresholds, precision[:-1], "b--", label="Precision")
plt.plot(thresholds, recall[:-1], "g-", label="Recall")
plt.xlabel("Threshold")
plt.legend()
```

```
plt.title("Precision-Recall vs Threshold")
plt.grid(True)
plt.show()
```



In [ ]:

In [ ]:

In [ ]: