AI & Machine Learning Internship Projects | Future Interns

Welcome to this Google Colab workspace! This notebook serves as a comprehensive portfolio showcasing my work during the **Future Interns Machine Learning Internship** program.

Throughout this internship, I'm developing practical machine learning solutions for real-world business challenges. This file will contain detailed explanations, code implementations, and visual insights for various tasks undertaken.

Currently, this notebook focuses on:

- **Task 2: Churn Prediction System** - Building a machine learning model to identify customers at risk of churning, providing valuable insights for retention strategies.

You'll find a structured approach, from data loading and exploration to model building, evaluation, and deriving actionable business recommendations.

Let's explore the power of Machine Learning in solving business problems!

---

1. Project Setup & Data Loading

First, we'll set up our environment by importing necessary libraries and loading the `WA_Fn-UseC_-Telco-Customer-Churn.csv` dataset. This dataset contains various customer attributes and their churn status.

The 'TotalCharges' column in this dataset often loads as a string due to some empty values, which we'll correct by converting it to a numeric type and handling the resulting missing values. We will also convert the 'Churn' target variable into a numerical format (0 for 'No', 1 for 'Yes').

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix, roc_curve, auc

# --- Data Loading ---

# Since you've uploaded the file directly, it will be in the Colab session's root directory.
# The 'WA_Fn-UseC_-Telco-Customer-Churn.csv' is the file name.
file_name = 'WA_Fn-UseC_-Telco-Customer-Churn.csv'

# Check if the file is already in the current directory (after initial upload)
import os
if not os.path.exists(file_name):
    print(f"File '{file_name}' not found. Please upload your customer churn dataset.")
    from google.colab import files
    uploaded = files.upload()
    # If the user uploads with a different name, adjust file_name
    if uploaded and list(uploaded.keys())[0] != file_name:
      print(f"Warning: Uploaded file is named '{list(uploaded.keys())[0]}'. Using this name.")
      file_name = list(uploaded.keys())[0]


df = pd.read_csv(file_name)

# Display basic information about the dataset
print("--- Dataset Head ---")
print(df.head())

print("\n--- Dataset Info ---")
df.info()

print("\n--- Missing Values ---")
print(df.isnull().sum())

# --- Important Initial Preprocessing for Telco Churn Dataset ---
# The 'TotalCharges' column in this dataset often gets loaded as an object (string)
# and contains some empty strings which cause issues when converting to numeric.
# We'll convert it to numeric and handle potential missing values (coerce errors to NaN).
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')

# Handle rows where 'TotalCharges' became NaN (typically empty strings representing new customers)
# For this dataset, dropping these rows is usually acceptable as they are very few.
# This ensures all values in 'TotalCharges' are numerical.
df.dropna(subset=['TotalCharges'], inplace=True)

# Convert the 'Churn' target variable to numerical (0 and 1) for modeling
# 'Yes' usually indicates churn (1), 'No' indicates no churn (0).
df['Churn'] = df['Churn'].apply(lambda x: 1 if x == 'Yes' else 0)

print("\n--- Processed DataFrame Head (after TotalCharges and Churn conversion) ---")
print(df.head())

print("\n--- Churn Column Value Counts (after conversion) ---")
print(df['Churn'].value_counts())
```

```
--- Dataset Head ---
   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
```

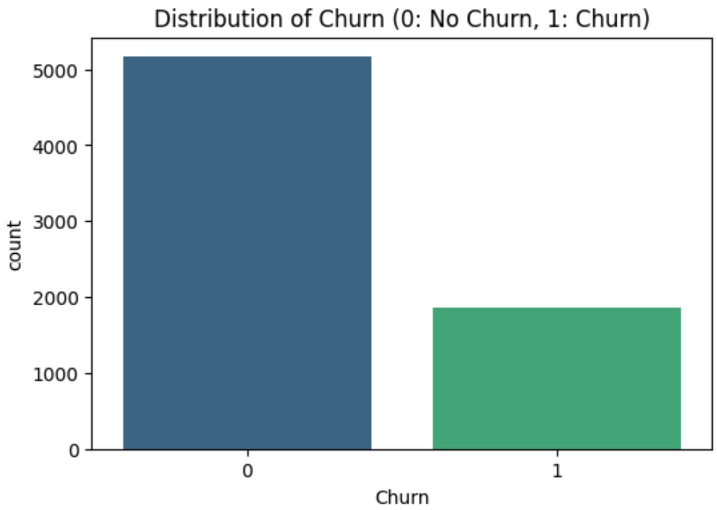
```
4  9237-HQITU  Female              0      No         No       2          Yes

      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]

--- Dataset 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
```

2. Exploratory Data Analysis (EDA) Understanding the characteristics of our customer data is crucial for building an effective churn prediction model. This section explores the distribution of churn and key features, identifying patterns that might influence churn.

Churn Distribution We'll first examine the balance of our target variable, 'Churn', to understand how many customers churn versus how many do not. This is important as imbalanced datasets can affect model training.

```
# CODE BLOCK 2.1: Churn Distribution

# --- Target Variable Distribution ---
print("\n--- Churn Distribution ---")
print(df['Churn'].value_counts())
print(f"Churn Rate: {df['Churn'].mean():.2%}")

plt.figure(figsize=(6, 4))
sns.countplot(x='Churn', data=df, palette='viridis')
plt.title('Distribution of Churn (0: No Churn, 1: Churn)')
plt.show()
```

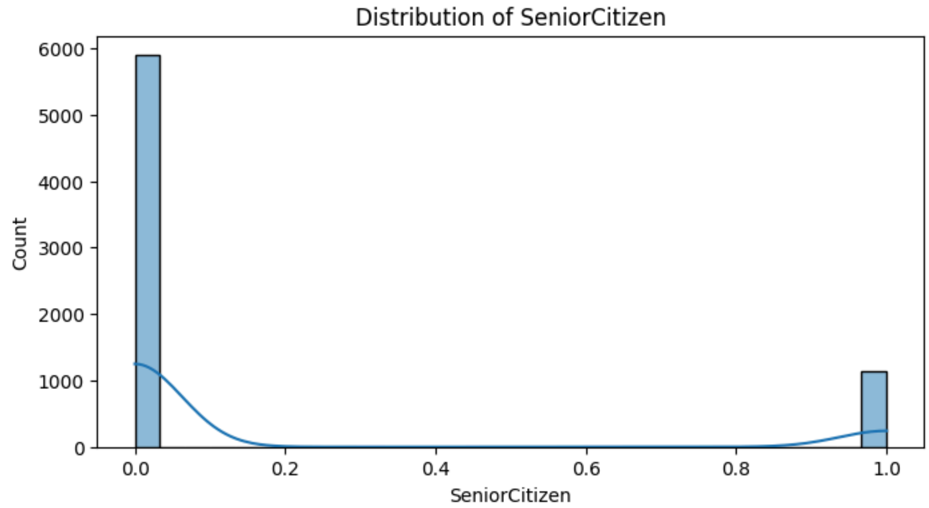
```
--- Churn Distribution ---
Churn
0    5163
1    1869
Name: count, dtype: int64
Churn Rate: 26.58%
/tmp/ipython-input-9-1436504502.py:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.countplot(x='Churn', data=df, palette='viridis')
```



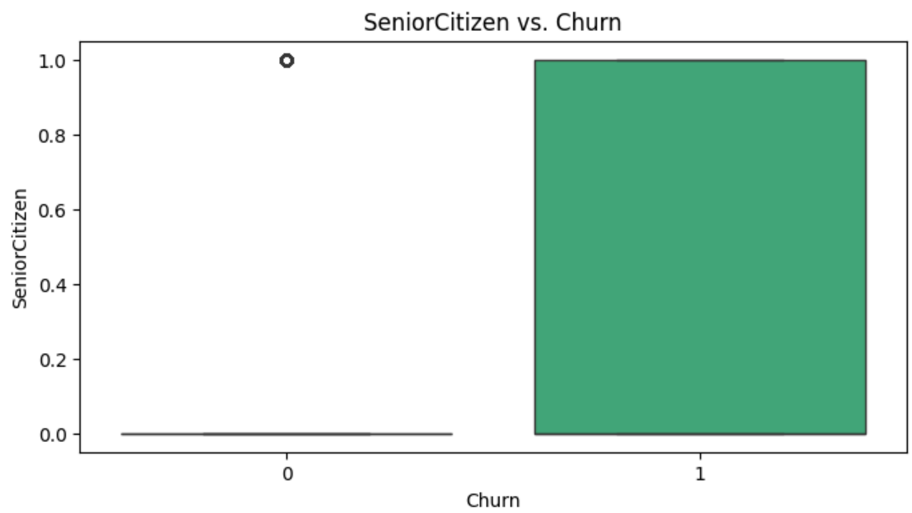
Numerical Feature Distributions Next, we'll visualize the distributions of numerical features like tenure, MonthlyCharges, and TotalCharges. Box plots will help us see how these features differ between churning and non-churning customers.

```
# CODE BLOCK 2.2: Numerical Feature Distributions

numerical_features = df.select_dtypes(include=np.number).columns.tolist()
# Exclude 'customerID' if it's in numerical_features (already dropped in preproc) and 'Churn' as it's the target
if 'customerID' in numerical_features: numerical_features.remove('customerID') # Guard against ID remaining
if 'Churn' in numerical_features: numerical_features.remove('Churn')

print("\n--- Numerical Feature Distributions ---")
for col in numerical_features:
    plt.figure(figsize=(8, 4))
    sns.histplot(df[col], kde=True, bins=30, palette='viridis')
    plt.title(f'Distribution of {col}')
    plt.show()

    plt.figure(figsize=(8, 4))
    sns.boxplot(x='Churn', y=col, data=df, palette='viridis')
    plt.title(f'{col} vs. Churn')
    plt.show()
```

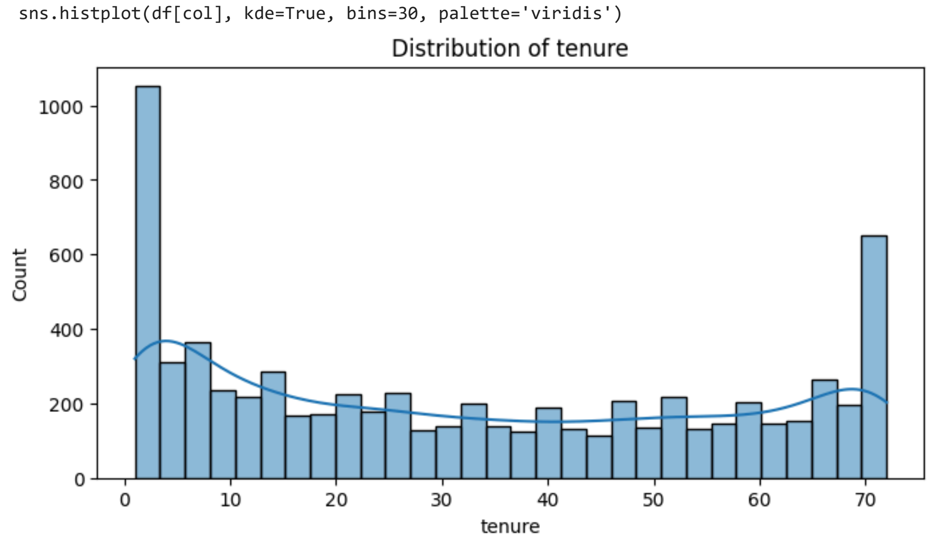
```
--- Numerical Feature Distributions ---
/tmp/ipython-input-10-2647694609.py:11: UserWarning: Ignoring `palette` because no `hue` variable has been assigned.
  sns.histplot(df[col], kde=True, bins=30, palette='viridis')
```



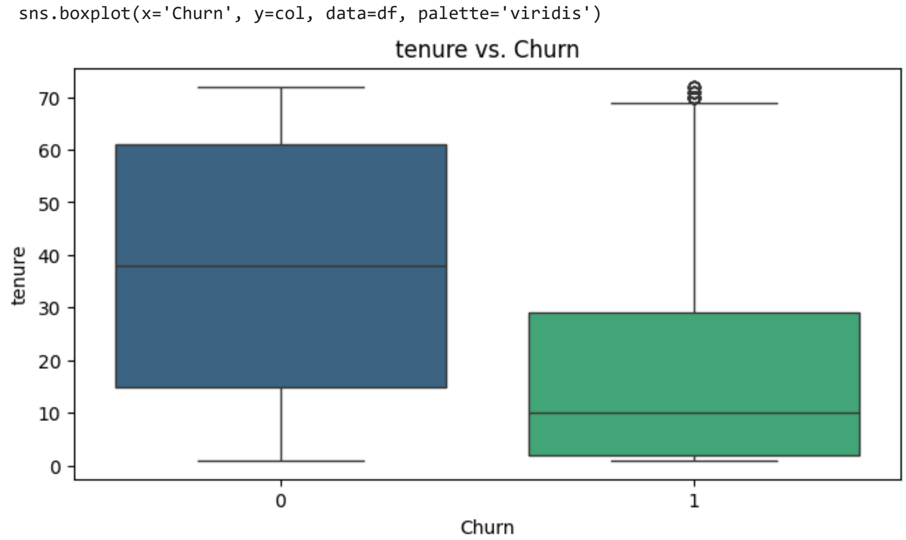

```
/tmp/ipython-input-10-2647694609.py:16: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.boxplot(x='Churn', y=col, data=df, palette='viridis')
```



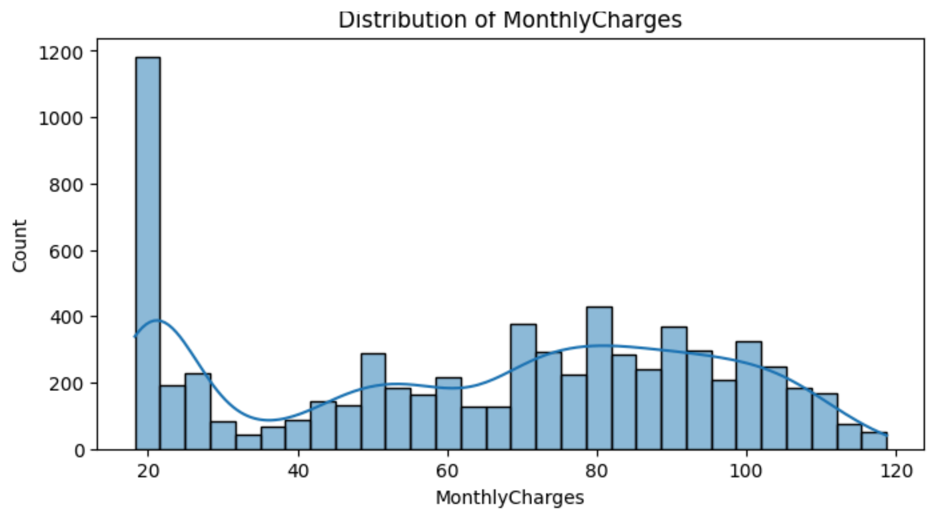

```
/tmp/ipython-input-10-2647694609.py:11: UserWarning: Ignoring `palette` because no `hue` variable has been assigned.
  sns.histplot(df[col], kde=True, bins=30, palette='viridis')
```



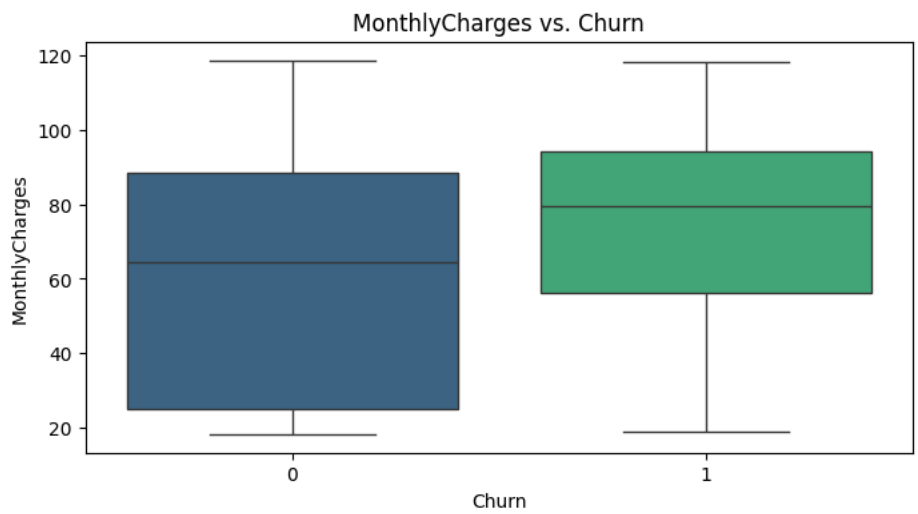

```
/tmp/ipython-input-10-2647694609.py:16: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.boxplot(x='Churn', y=col, data=df, palette='viridis')
```



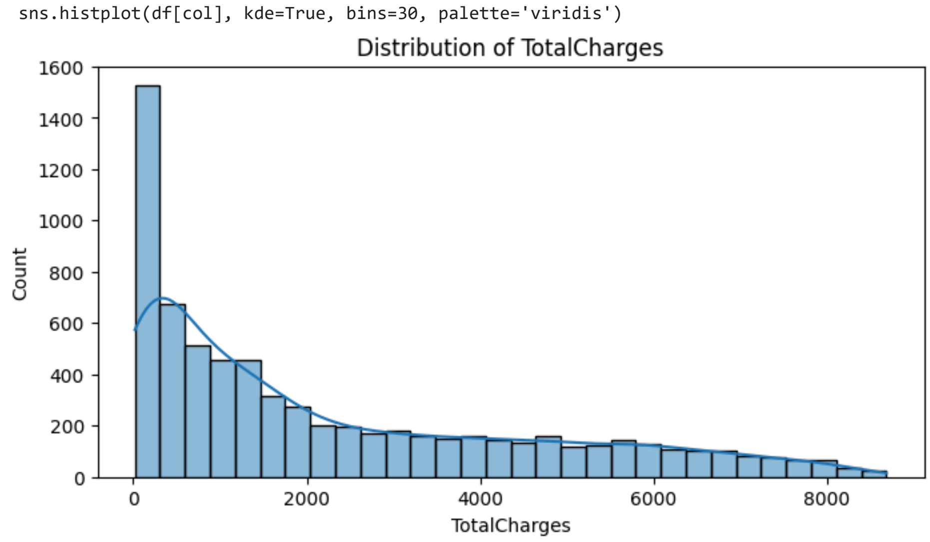

```
/tmp/ipython-input-10-2647694609.py:11: UserWarning: Ignoring `palette` because no `hue` variable has been assigned.
  sns.histplot(df[col], kde=True, bins=30, palette='viridis')
```



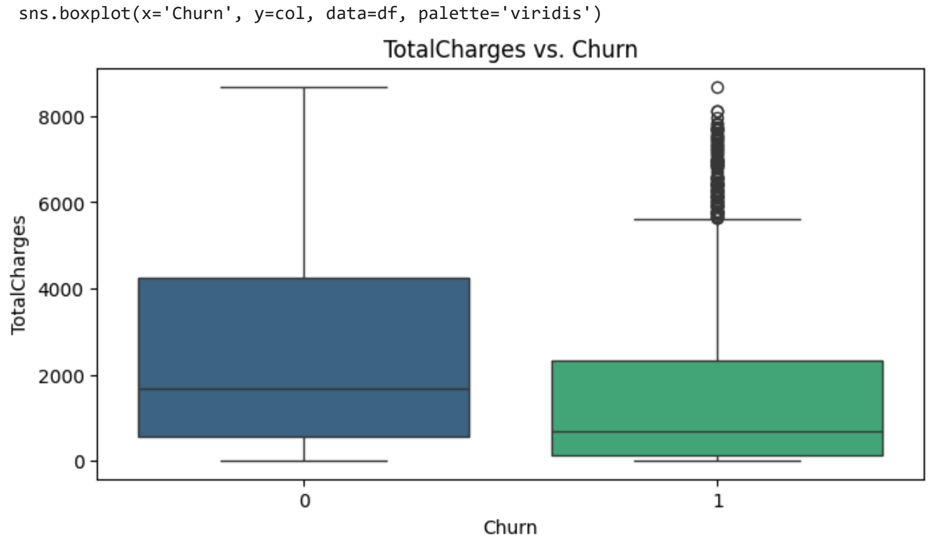

```
/tmp/ipython-input-10-2647694609.py:16: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.boxplot(x='Churn', y=col, data=df, palette='viridis')
```




```
/tmp/ipython-input-10-2647694609.py:11: UserWarning: Ignoring `palette` because no `hue` variable has been assigned.
  sns.histplot(df[col], kde=True, bins=30, palette='viridis')
```



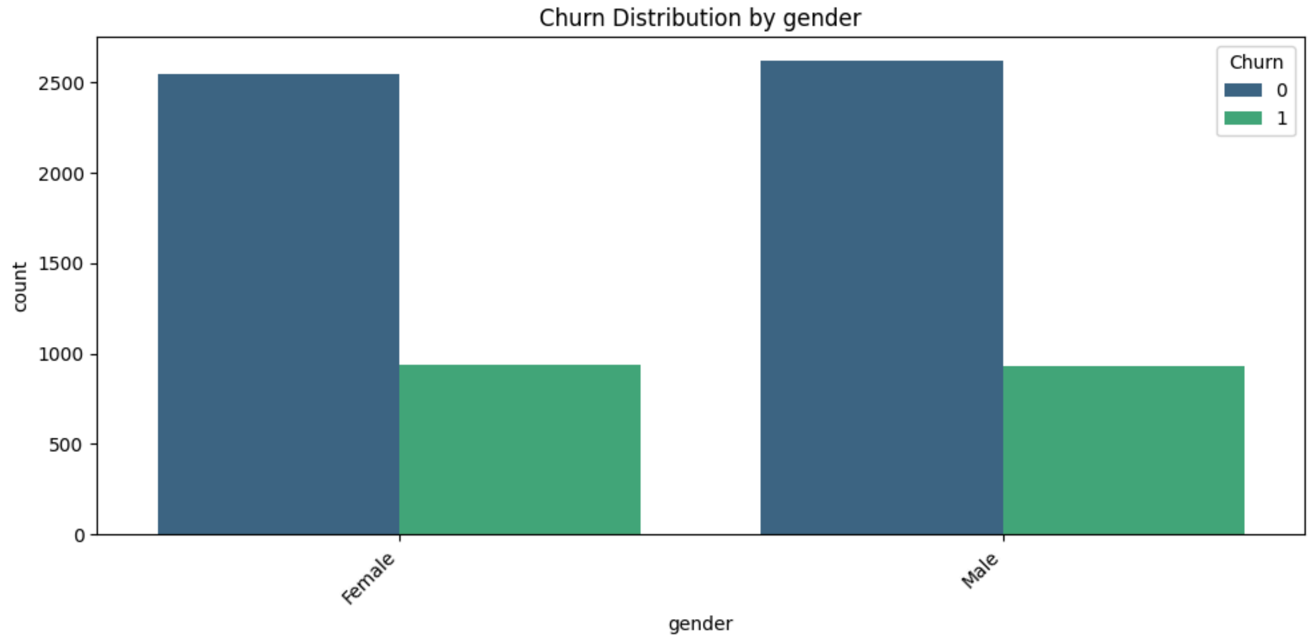

```
/tmp/ipython-input-10-2647694609.py:16: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.boxplot(x='Churn', y=col, data=df, palette='viridis')
```



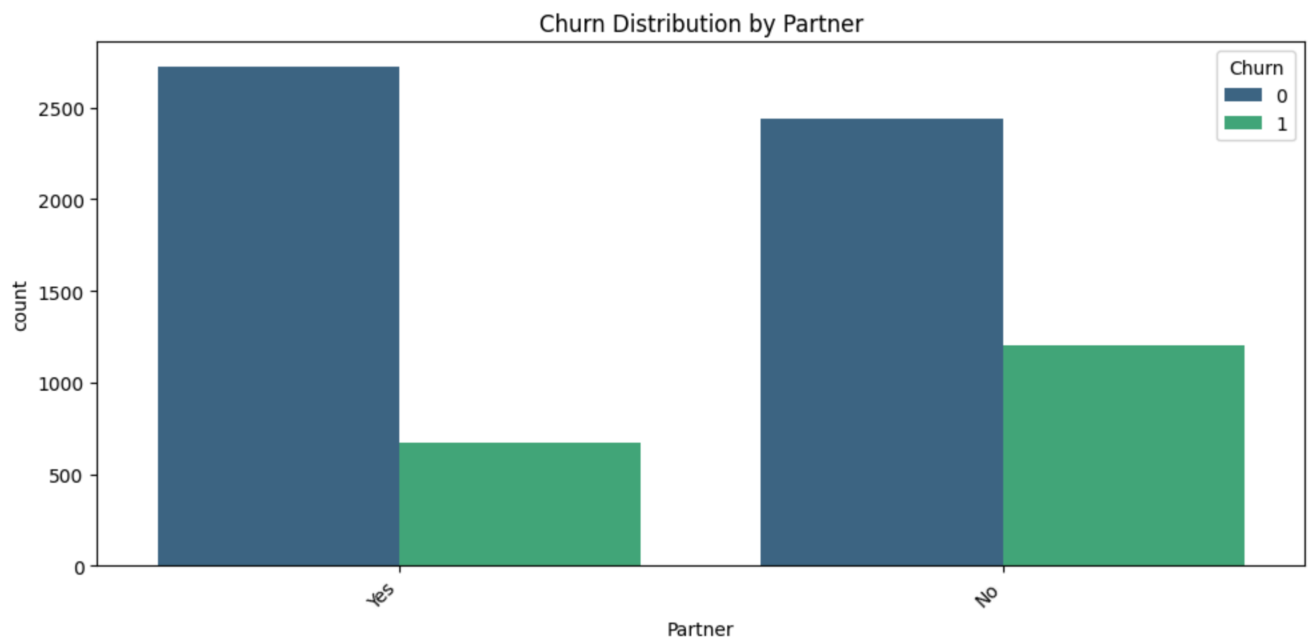
Categorical Feature Churn Rates Finally, we'll explore categorical features like Contract, PaymentMethod, InternetService, etc. by visualizing the churn distribution within each category. This helps identify which categories have higher churn rates.

```
# CODE BLOCK 2.3: Categorical Feature Churn Rates

# Drop 'customerID' early if it wasn't dropped already (e.g. if you reran only EDA)
if 'customerID' in df.columns:
  df_eda = df.drop('customerID', axis=1)
else:
  df_eda = df.copy() # Use the already cleaned df if customerID is gone

categorical_features = df_eda.select_dtypes(include='object').columns.tolist()

print("\n--- Categorical Feature Churn Rates ---")
for col in categorical_features:
    plt.figure(figsize=(10, 5))
    sns.countplot(x=col, hue='Churn', data=df_eda, palette='viridis')
    plt.title(f'Churn Distribution by {col}')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()

    # Calculate and print churn rate per category
    churn_rate_by_cat = df_eda.groupby(col)['Churn'].mean().reset_index()
    print(f"\nChurn Rate by {col}:\n{churn_rate_by_cat}")
```

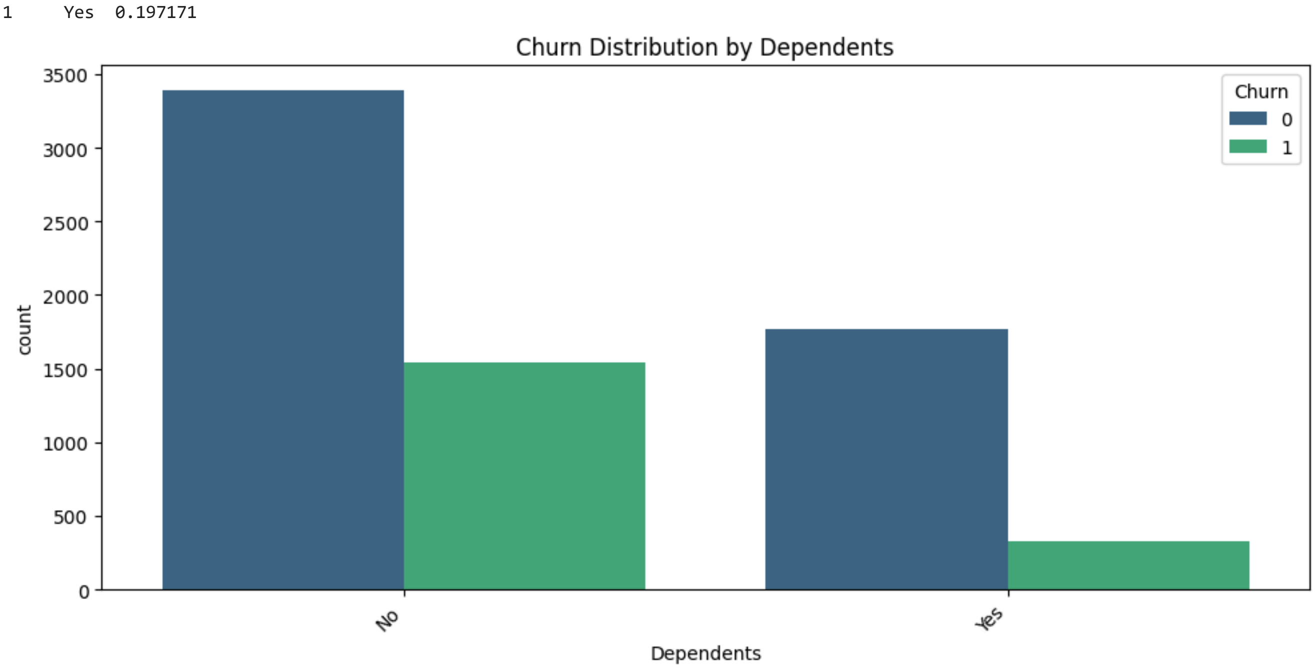
```
--- Categorical Feature Churn Rates ---
```



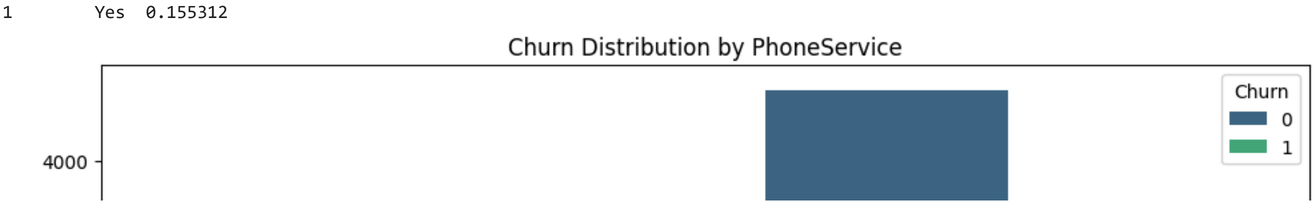

```
Churn Rate by gender:
   gender     Churn
0  Female  0.269595
1    Male  0.262046
```

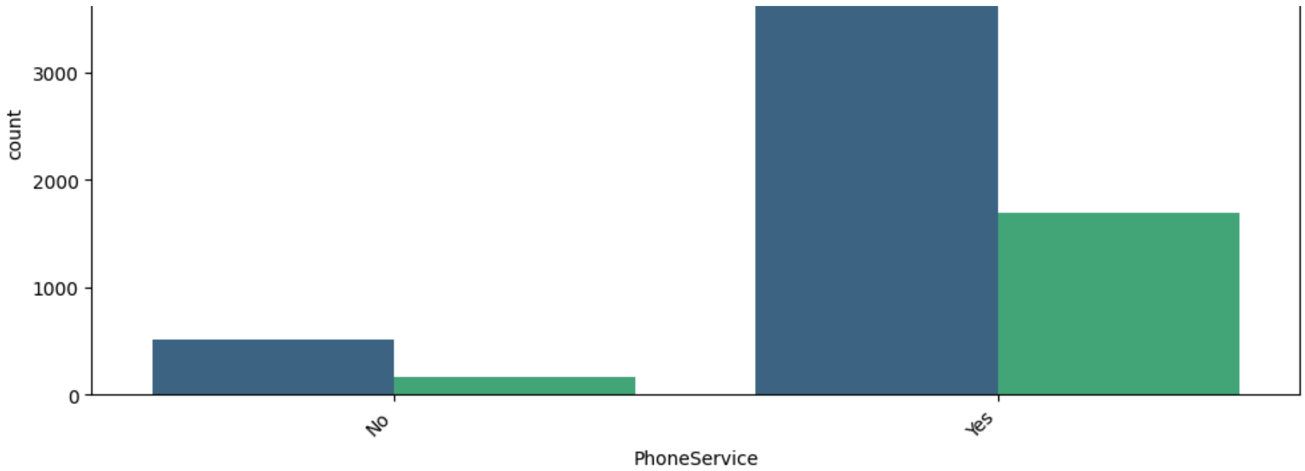



```
Churn Rate by Partner:
  Partner     Churn
0      No  0.329761
1     Yes  0.197171
```



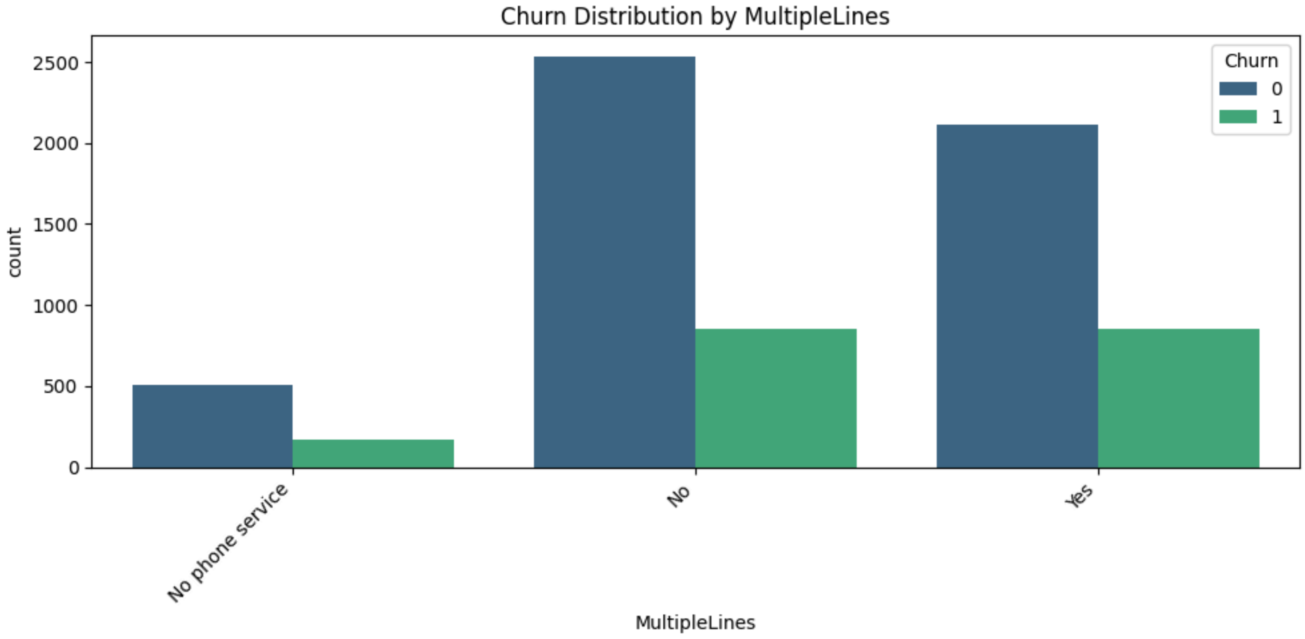

```
Churn Rate by Dependents:
  Dependents     Churn
0         No  0.312791
1        Yes  0.155312
```





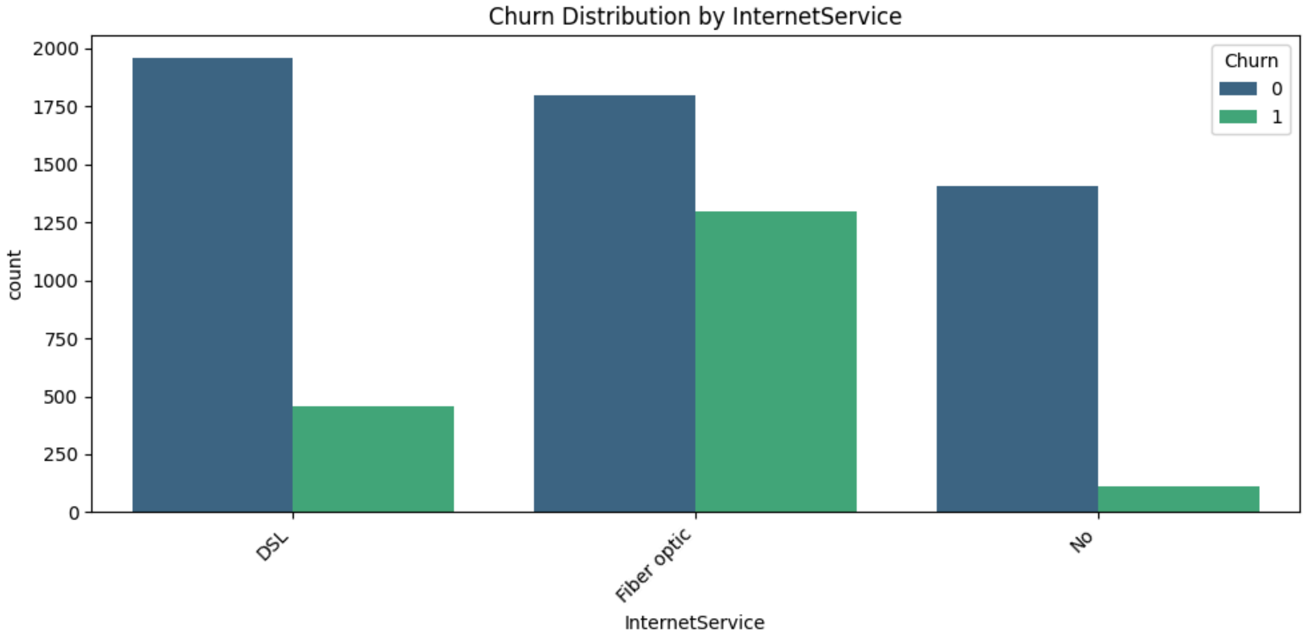

```
Churn Rate by PhoneService:
  PhoneService     Churn
0           No  0.250000
1          Yes  0.267475
```

Churn Distribution by MultipleLines



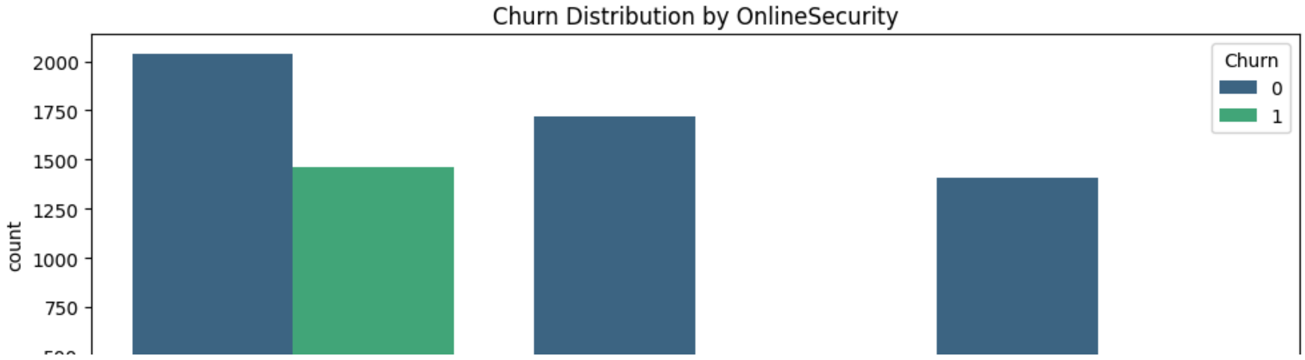

```
Churn Rate by MultipleLines:
      MultipleLines     Churn
0                No  0.250812
1  No phone service  0.250000
2               Yes  0.286485
```

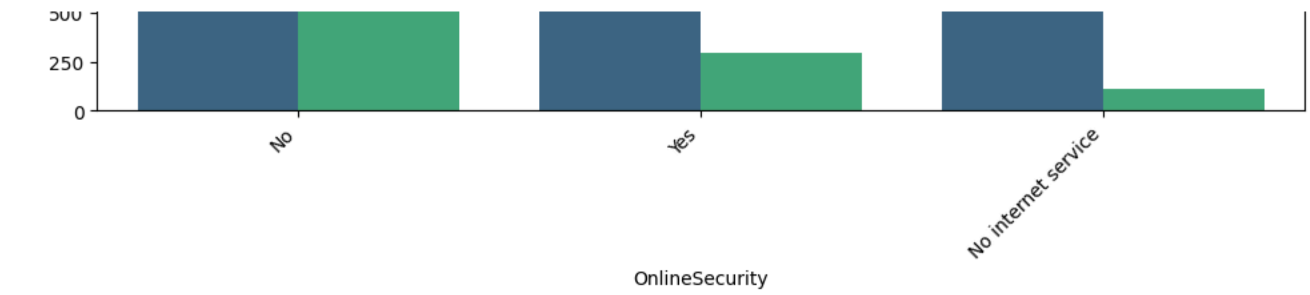
Churn Distribution by InternetService



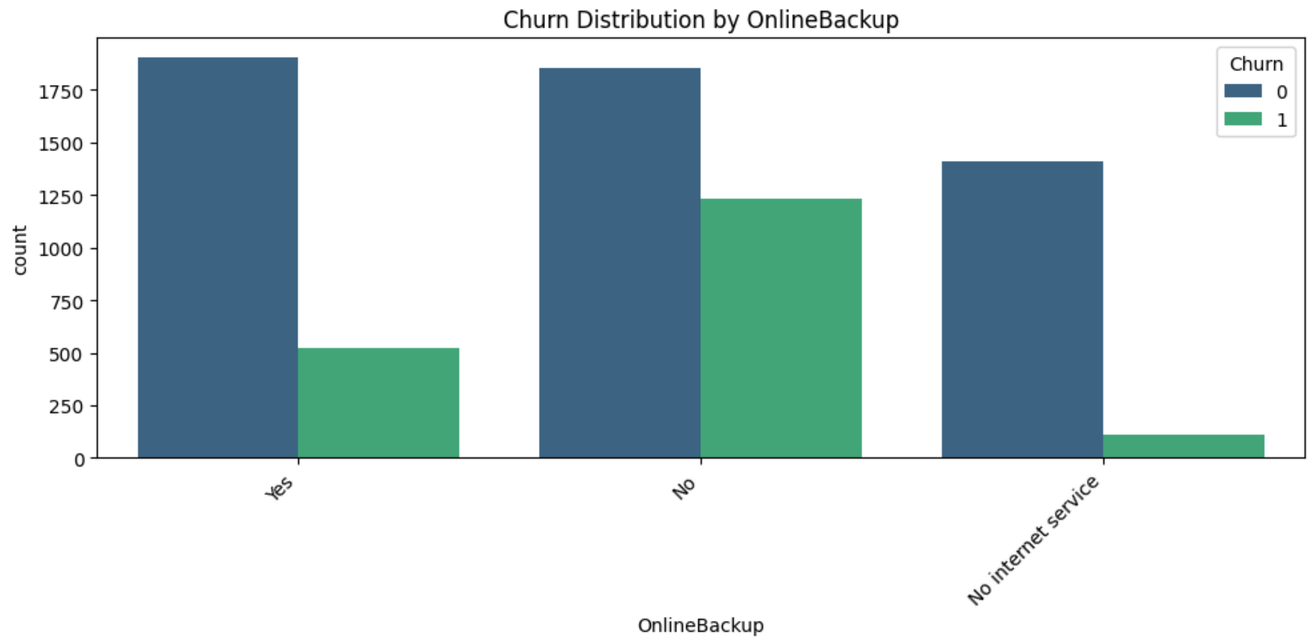

```
Churn Rate by InternetService:
  InternetService     Churn
0             DSL  0.189983
1     Fiber optic  0.418928
2              No  0.074342
```

Churn Distribution by OnlineSecurity

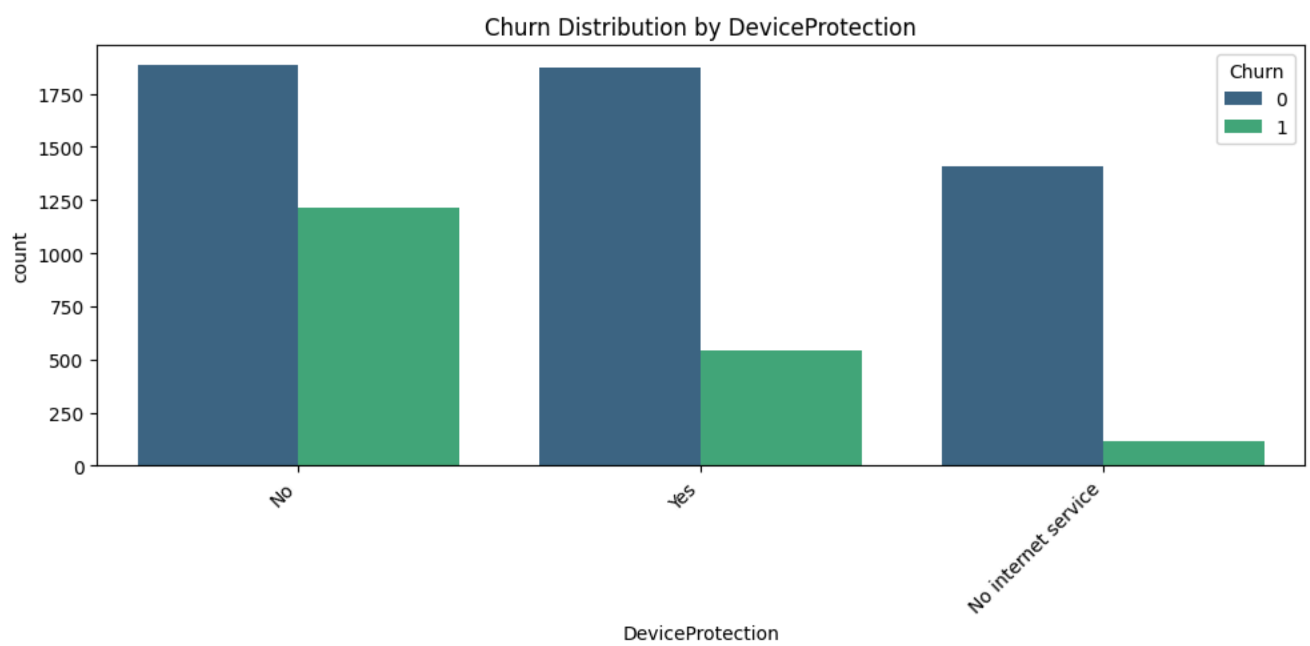




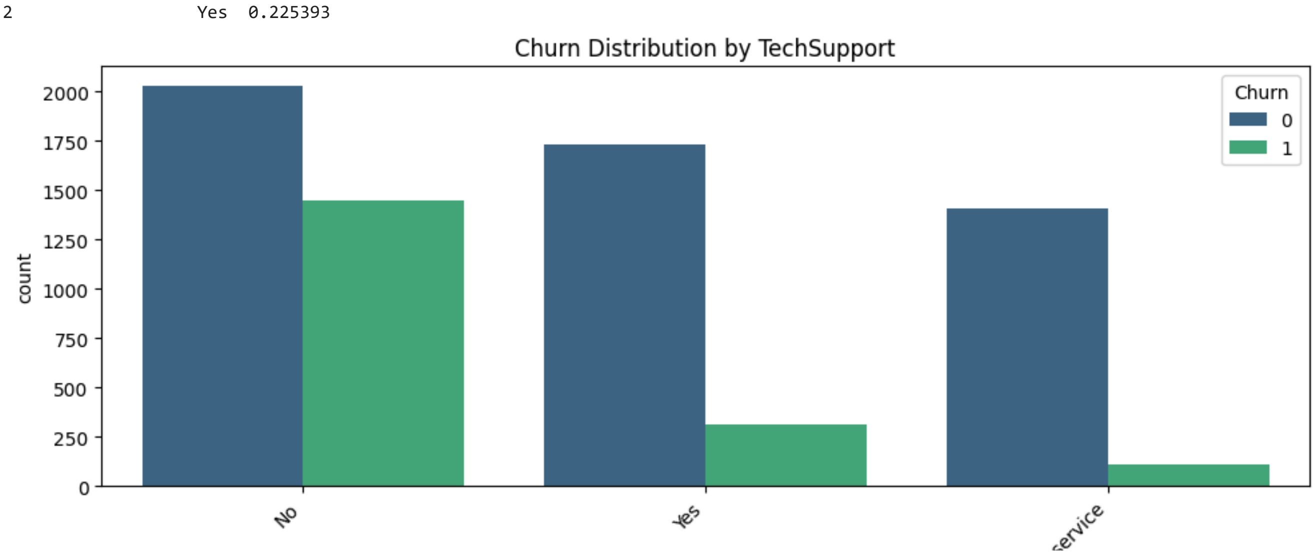

```
Churn Rate by OnlineSecurity:
        OnlineSecurity     Churn
0                   No  0.417787
1  No internet service  0.074342
2                  Yes  0.146402
```



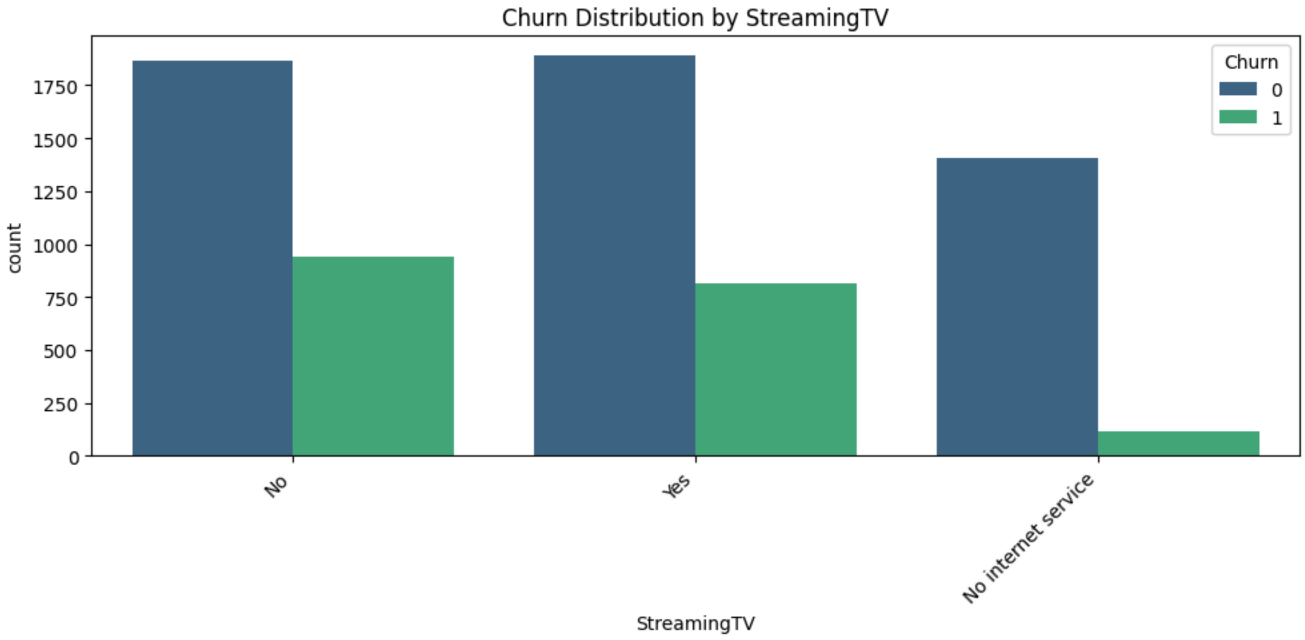

```
Churn Rate by OnlineBackup:
          OnlineBackup     Churn
0                   No  0.399417
1  No internet service  0.074342
2                  Yes  0.215670
```



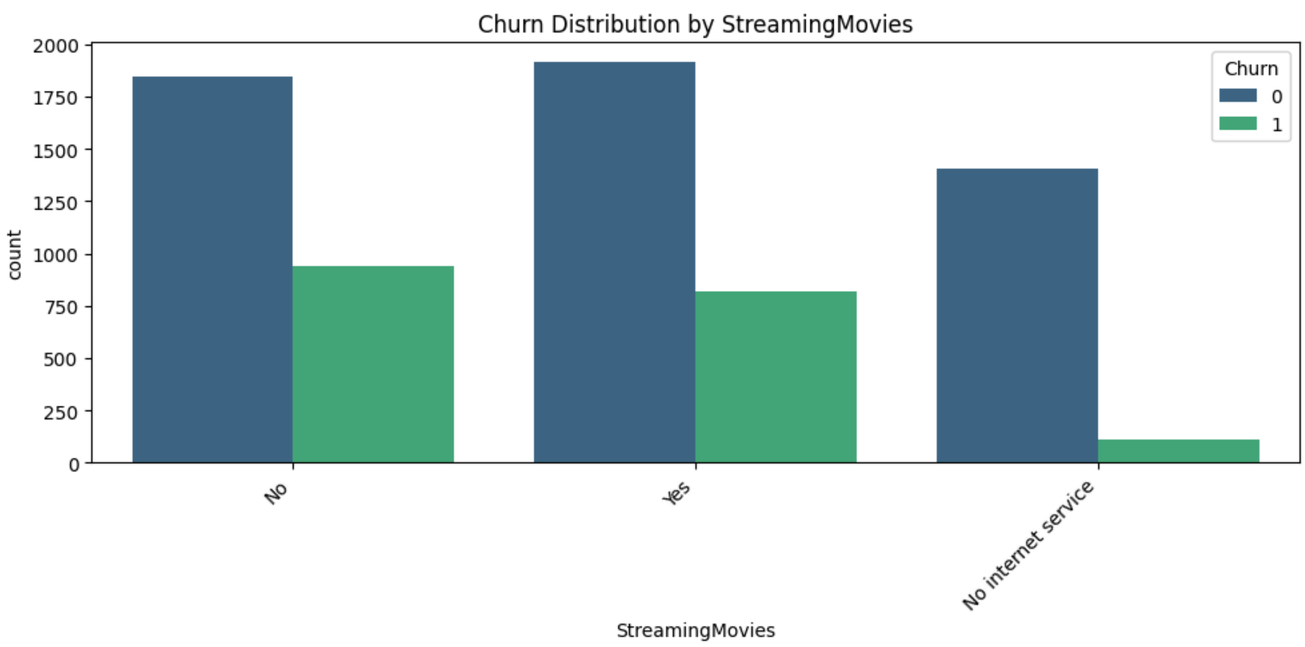

```
Churn Rate by DeviceProtection:
      DeviceProtection     Churn
0                   No  0.391403
1  No internet service  0.074342
2                  Yes  0.225393
```



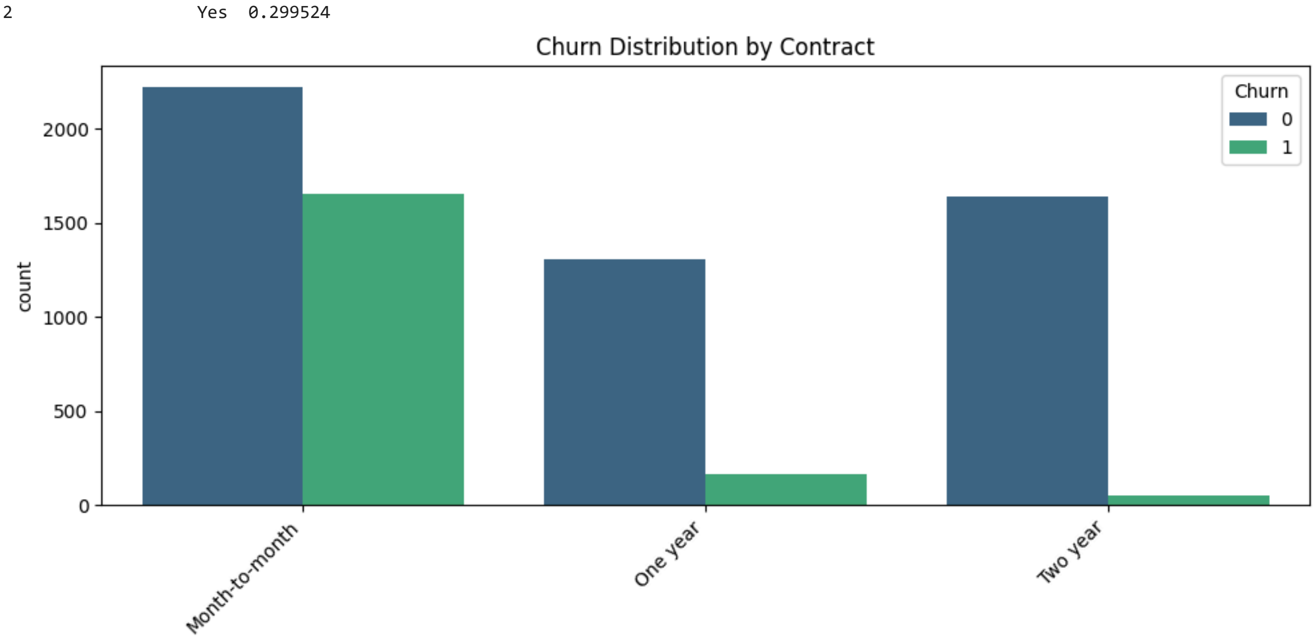

```
Churn Rate by TechSupport:
           TechSupport     Churn
0                   No  0.416475
1  No internet service  0.074342
2                  Yes  0.151961
```




```
Churn Rate by StreamingTV:
           StreamingTV     Churn
0                   No  0.335351
1  No internet service  0.074342
2                  Yes  0.301147
```

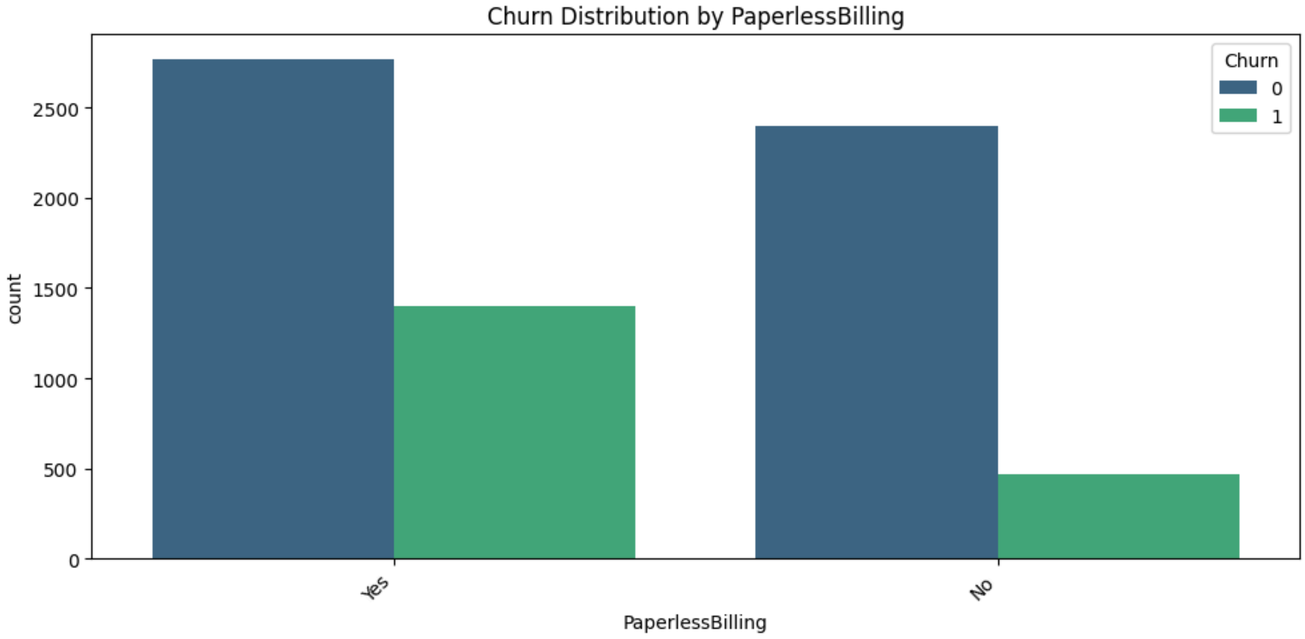



```
Churn Rate by StreamingMovies:
       StreamingMovies     Churn
0                   No  0.337289
1  No internet service  0.074342
2                  Yes  0.299524
```

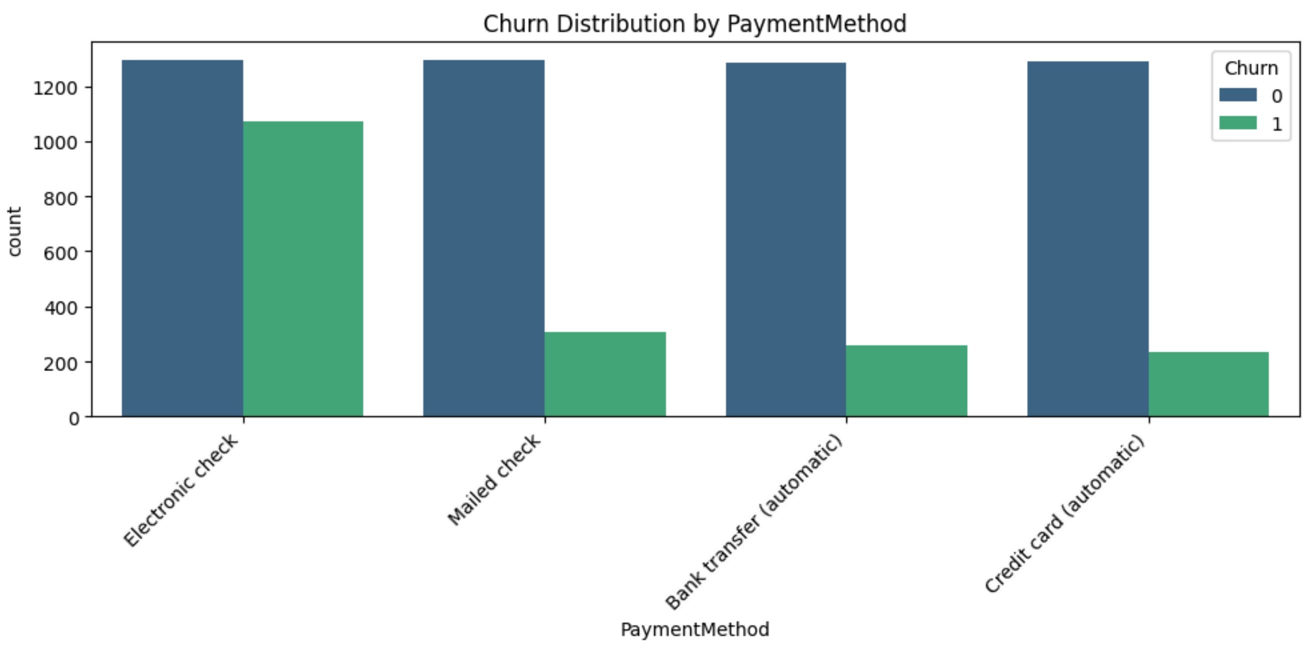



```
Churn Rate by Contract:
```

```
Churn Rate by Contract:
         Contract     Churn
0  Month-to-month  0.427097
1        One year  0.112772
2        Two year  0.028487
```




```
Churn Rate by PaperlessBilling:
  PaperlessBilling     Churn
0               No  0.163757
1              Yes  0.335893
```




```
Churn Rate by PaymentMethod:
               PaymentMethod     Churn
0  Bank transfer (automatic)  0.167315
1    Credit card (automatic)  0.152531
2           Electronic check  0.452854
3               Mailed check  0.192020
```

3. Data Preprocessing & Feature Engineering Before training our machine learning models, we need to preprocess the data. This involves encoding categorical variables into a numerical format that models can understand and scaling numerical features to prevent dominance by features with larger values. We will set up a robust preprocessing pipeline using ColumnTransformer for this.

```
# CODE BLOCK 3: Data Preprocessing & Feature Engineering

# Drop 'customerID' as it's just an identifier and not a predictive feature
# We do this here explicitly in case EDA was run on a df with it.
df_for_modeling = df.drop('customerID', axis=1)

# Define target variable and features
X = df_for_modeling.drop('Churn', axis=1)
y = df_for_modeling['Churn']

# Identify categorical and numerical features for preprocessing
numerical_cols = X.select_dtypes(include=np.number).columns.tolist()
categorical_cols = X.select_dtypes(include='object').columns.tolist()

print(f"\nNumerical columns identified: {numerical_cols}")
print(f"Categorical columns identified: {categorical_cols}")

# Create preprocessing pipelines for numerical and categorical features
numerical_transformer = Pipeline(steps=[
    ('scaler', StandardScaler()) # Standardize numerical features
])

categorical_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown='ignore')) # One-hot encode categorical features
])

# Create a preprocessor using ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer, categorical_cols)
    ],
    remainder='passthrough' # Keep other columns if any, though none expected here
)

# --- Train-Test Split ---
# We use stratify=y to ensure that the proportion of churned (1) and non-churned (0)
# customers is maintained in both the training and testing sets.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

print(f"\nTraining set shape: {X_train.shape}")
print(f"Testing set shape: {X_test.shape}")

# --- Verify Churn Distribution in Splits (TROUBLESHOOTING STEP) ---
# This helps confirm that your splits contain both classes, preventing the ValueError.
print("\n--- Churn Distribution in Training Set (y_train) ---")
print(y_train.value_counts())
print("\n--- Churn Distribution in Test Set (y_test) ---")
print(y_test.value_counts())
```

```
Numerical columns identified: ['SeniorCitizen', 'tenure', 'MonthlyCharges', 'TotalCharges']
Categorical columns identified: ['gender', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity'

Training set shape: (5625, 19)
Testing set shape: (1407, 19)

--- Churn Distribution in Training Set (y_train) ---
Churn
0    4130
1    1495
Name: count, dtype: int64

--- Churn Distribution in Test Set (y_test) ---
Churn
0    1033
1     374
Name: count, dtype: int64
```

4. Model Building With our data prepared, we'll now build and train various classification models. We'll use Logistic Regression (a simple linear model), Random Forest (an ensemble tree-based model), and XGBoost (a powerful gradient boosting model). Each model will be integrated into a pipeline with our preprocessor for seamless training.

```
# CODE BLOCK 4: Model Building

# --- Initialize Models ---
models = {
    'Logistic Regression': LogisticRegression(random_state=42, solver='liblinear', max_iter=1000), # max_iter for convergence
    'Random Forest': RandomForestClassifier(random_state=42),
    'XGBoost': XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='logloss') # eval_metric for XGBoost future warning
}

# --- Create and Train Pipelines ---
trained_models = {}
for name, model in models.items():
    print(f"\n--- Training {name} ---")
    pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                               ('classifier', model)])
    pipeline.fit(X_train, y_train)
    trained_models[name] = pipeline
    print(f"{name} training complete. ✅")
```

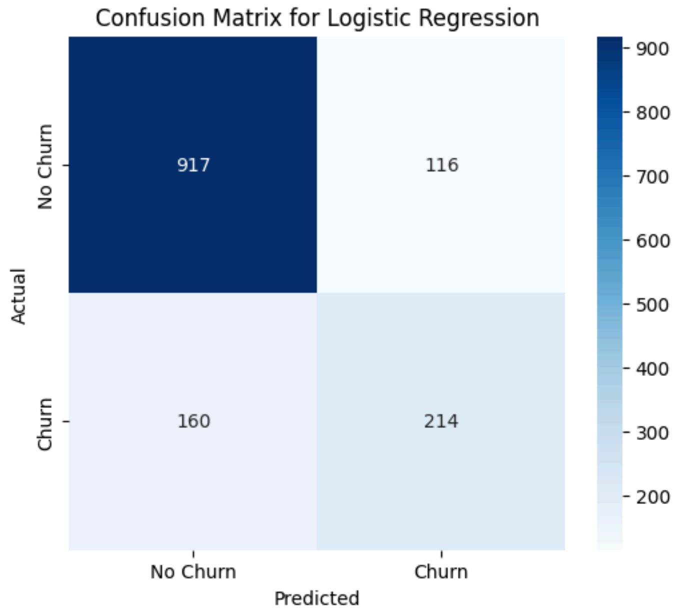
```
--- Training Logistic Regression ---
Logistic Regression training complete. ✅

--- Training Random Forest ---
Random Forest training complete. ✅

--- Training XGBoost ---
XGBoost training complete. ✅
```

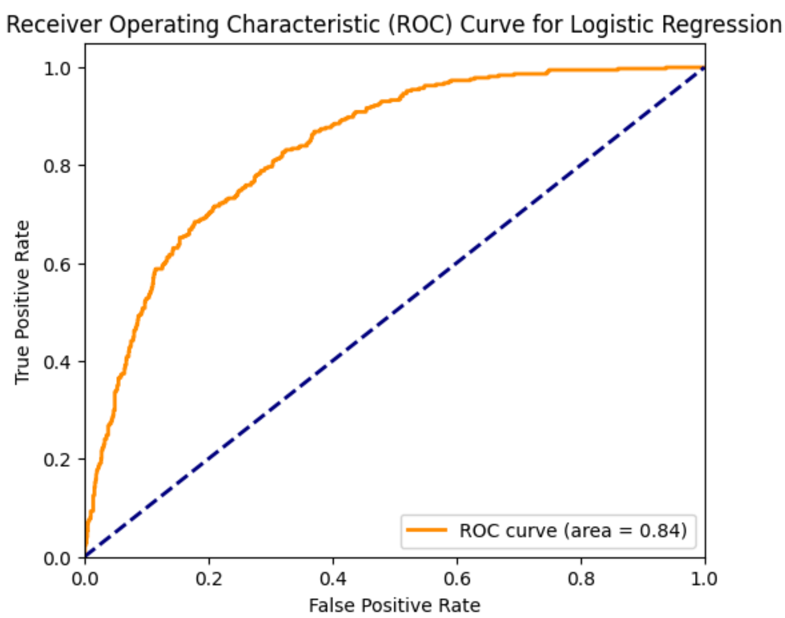
5. Model Evaluation After training, it's crucial to evaluate how well our models perform on unseen data. We'll use several key metrics for classification tasks, including Accuracy, Precision, Recall, F1-Score, and ROC-AUC. We'll also visualize the Confusion Matrix and ROC Curves, and analyze Feature Importance for tree-based models.

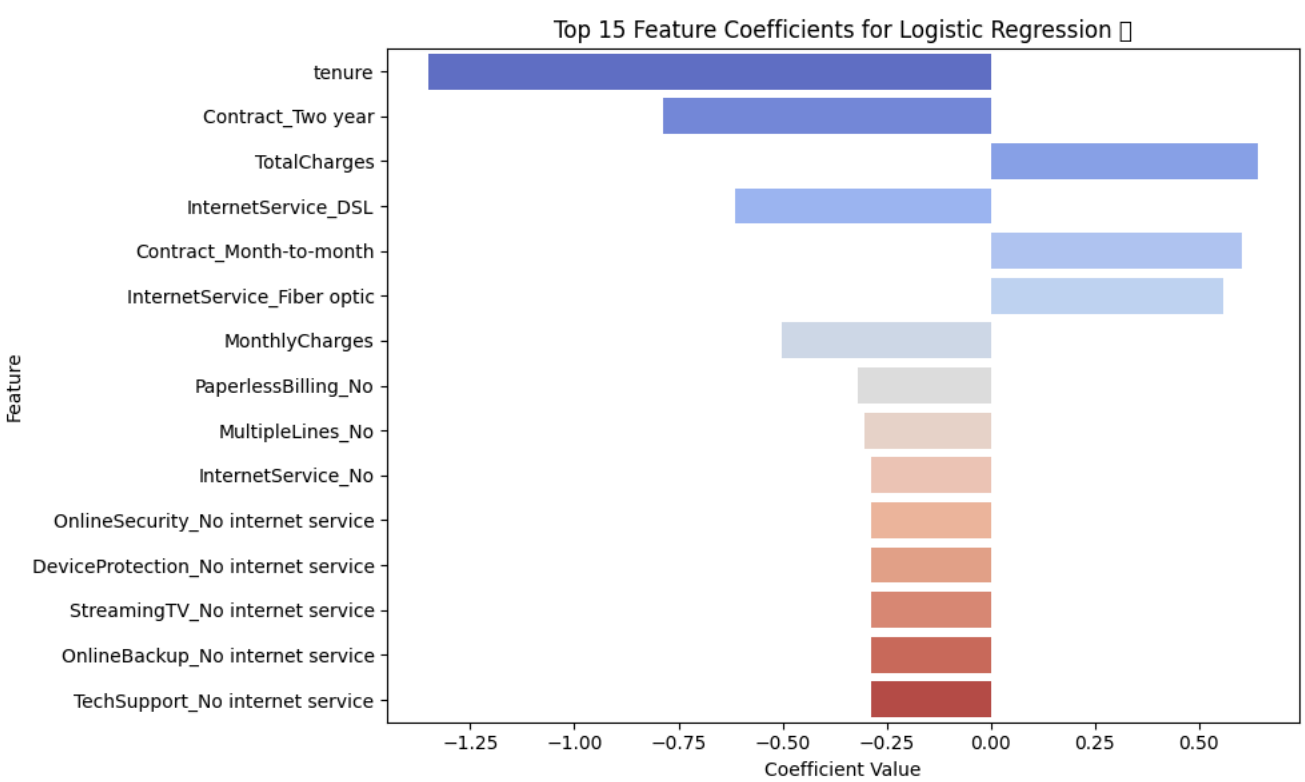
```
# CODE BLOCK 5: Model Evaluation

results = {}
for name, pipeline in trained_models.items():
    print(f"\n--- Evaluating {name} ---")
    y_pred = pipeline.predict(X_test)
    y_proba = pipeline.predict_proba(X_test)[:, 1] # Probability of churn (positive class)

    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)
    roc_auc = roc_auc_score(y_test, y_proba)

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

    print(f"Accuracy: {accuracy:.4f} ✨")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-Score: {f1:.4f}")
    print(f"ROC-AUC: {roc_auc:.4f}")

    # --- Confusion Matrix ---
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=['No Churn', 'Churn'],
                yticklabels=['No Churn', 'Churn'])
    plt.title(f'Confusion Matrix for {name}')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

    # --- ROC Curve ---
    fpr, tpr, thresholds = roc_curve(y_test, y_proba)
    roc_auc_val = auc(fpr, tpr)
    plt.figure(figsize=(6, 5))
    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc_val:.2f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver Operating Characteristic (ROC) Curve for {name}')
    plt.legend(loc="lower right")
    plt.show()

    # --- Feature Importance (for tree-based models like Random Forest, XGBoost) ---
    if hasattr(pipeline.named_steps['classifier'], 'feature_importances_'):
        importances = pipeline.named_steps['classifier'].feature_importances_
        # Get feature names after one-hot encoding
        ohe = pipeline.named_steps['preprocessor'].named_transformers_['cat'].named_steps['onehot']
        # Check if ohe successfully fitted, otherwise get_feature_names_out might fail on initial call
        try:
            categorical_feature_names = ohe.get_feature_names_out(categorical_cols)
        except AttributeError: # Fallback if get_feature_names_out isn't directly available or fitted
             categorical_feature_names = ohe.categories_[0] # simplified if ohe.categories_ is directly available for single category


        all_feature_names = numerical_cols + list(categorical_feature_names)

        feature_importance_df = pd.DataFrame({'feature': all_feature_names, 'importance': importances})
        feature_importance_df = feature_importance_df.sort_values(by='importance', ascending=False)

        plt.figure(figsize=(10, 6))
        sns.barplot(x='importance', y='feature', data=feature_importance_df.head(15), palette='viridis') # Top 15 features
        plt.title(f'Top 15 Feature Importance for {name} 💡')
        plt.xlabel('Importance')
        plt.ylabel('Feature')
        plt.tight_layout()
        plt.show()
    elif hasattr(pipeline.named_steps['classifier'], 'coef_'): # For Logistic Regression
        coefficients = pipeline.named_steps['classifier'].coef_[0]
        # Get feature names after one-hot encoding
        ohe = pipeline.named_steps['preprocessor'].named_transformers_['cat'].named_steps['onehot']
        try:
            categorical_feature_names = ohe.get_feature_names_out(categorical_cols)
        except AttributeError:
```

```python
            categorical_feature_names = ohe.categories_[0]

        all_feature_names = numerical_cols + list(categorical_feature_names)

        coef_df = pd.DataFrame({'feature': all_feature_names, 'coefficient': coefficients})
        coef_df['abs_coefficient'] = np.abs(coef_df['coefficient'])
        coef_df = coef_df.sort_values(by='abs_coefficient', ascending=False)

        plt.figure(figsize=(10, 6))
        sns.barplot(x='coefficient', y='feature', data=coef_df.head(15), palette='coolwarm')
        plt.title(f'Top 15 Feature Coefficients for {name} 💡')
        plt.xlabel('Coefficient Value')
        plt.ylabel('Feature')
        plt.tight_layout()
        plt.show()


# --- Overall Model Comparison ---
print("\n--- Model Performance Comparison ---")
results_df = pd.DataFrame(results).T
print(results_df)

# You might want to choose the best model based on a specific metric (e.g., F1-Score or ROC-AUC for imbalanced churn data)
best_model_name = results_df['ROC-AUC'].idxmax()
print(f"\nBest model based on ROC-AUC: {best_model_name} 🎉")
best_pipeline = trained_models[best_model_name]
```

```
--- Evaluating Logistic Regression ---
Accuracy: 0.8038 ✨
Precision: 0.6485
Recall: 0.5722
F1-Score: 0.6080
ROC-AUC: 0.8359
```








```
--- Evaluating Random Forest ---
Accuracy: 0.7875 ✨
Precision: 0.6307
Recall: 0.4840
F1-Score: 0.5477
ROC-AUC: 0.8137
```