

# Customer Churn Analytics Report

## Detailed Data Analysis and Marketing Insights

Your Name

June 28, 2025

### Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Dataset Overview</b>	<b>2</b>
<b>3</b>	<b>Exploratory Data Analysis (EDA)</b>	<b>2</b>
3.1	Churn Distribution . . . . .	2
3.2	Monthly Charges and Churn . . . . .	3
3.3	Contract Type and Churn . . . . .	3
3.4	Correlation Heatmap . . . . .	4
<b>4</b>	<b>Data Preparation</b>	<b>4</b>
<b>5</b>	<b>Modeling and Evaluation</b>	<b>4</b>
<b>6</b>	<b>Tuned Logistic Regression</b>	<b>5</b>
6.1	ROC Curve . . . . .	5
<b>7</b>	<b>Feature Importance: Coefficients and Odds Ratios</b>	<b>5</b>
<b>8</b>	<b>Interpretation</b>	<b>8</b>
<b>9</b>	<b>Marketing Recommendations</b>	<b>8</b>
<b>10</b>	<b>Conclusion</b>	<b>8</b>
<b>A</b>	<b>Appendix: Full Notebook Code</b>	<b>8</b>

# 1 Introduction

This report details an end-to-end churn prediction analysis for a telecommunications company, combining exploratory data analysis (EDA), preprocessing, and modeling. A particular emphasis is placed on interpretability of the tuned Logistic Regression model, as its performance and coefficients provide actionable marketing insights.

## 2 Dataset Overview

The raw dataset has **7,043 records** and 21 features. After cleaning, we retained 7,032 records with no missing `TotalCharges`.

Table 1: Dataset Shape and Types

Shape	(7,043 rows, 21 columns) before cleaning; (7,032 rows) after cleaning
Types	2 numerical columns (int/float), 18 categorical columns

The dataset was free of duplicate rows, and categorical columns were checked for cardinality. The `customerID` column was dropped since it is a unique identifier.

## 3 Exploratory Data Analysis (EDA)

### 3.1 Churn Distribution

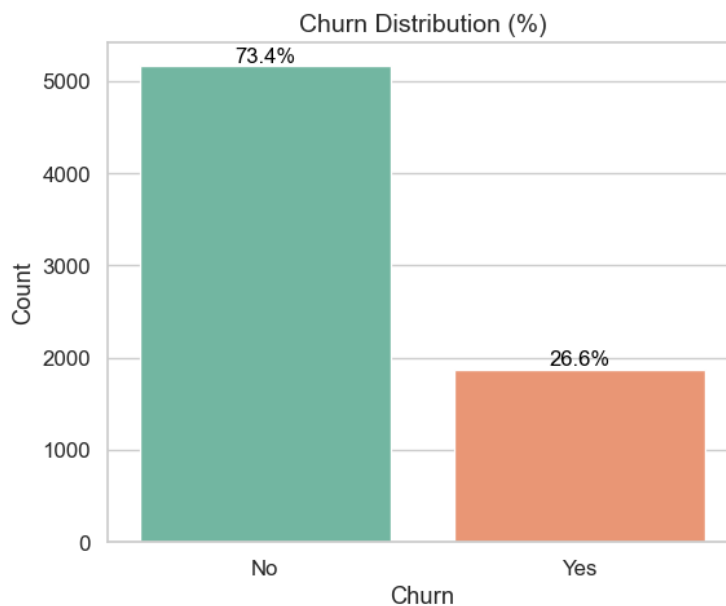


Figure 1: Churn distribution with percentage annotations, showing class imbalance in the dataset.

### 3.2 Monthly Charges and Churn

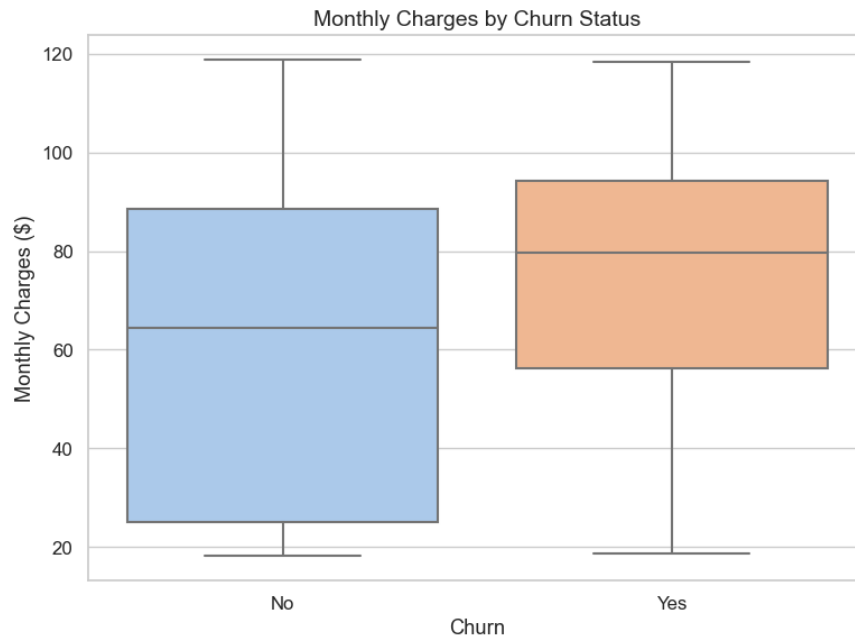


Figure 2: Boxplot of monthly charges by churn status. Customers who churn tend to have higher monthly charges.

### 3.3 Contract Type and Churn

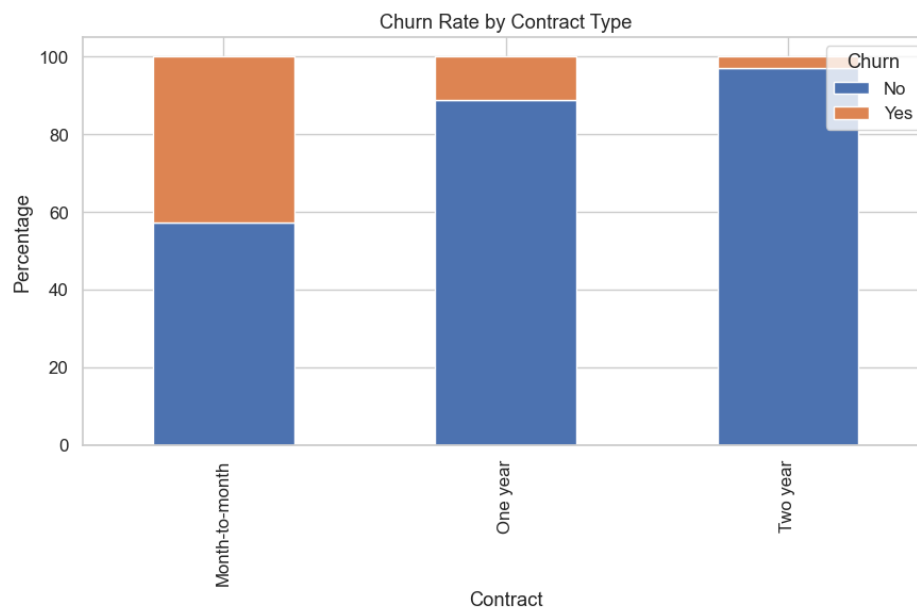


Figure 3: Stacked bar chart showing churn percentage by contract type. Month-to-month contracts have a significantly higher churn rate compared to longer-term contracts.

### 3.4 Correlation Heatmap

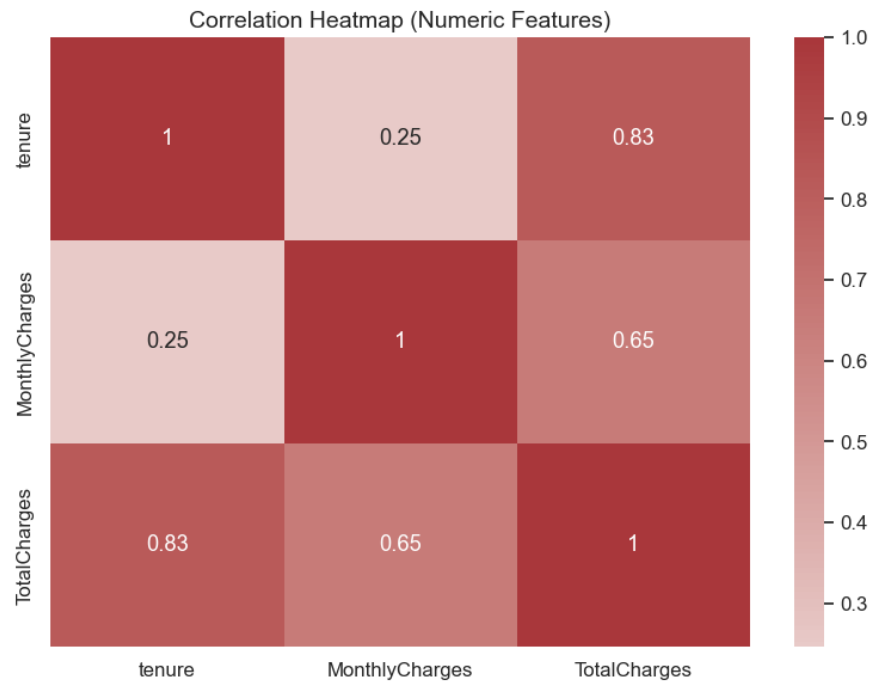


Figure 4: Correlation heatmap of numeric features including churn. Tenure and total charges are highly correlated. Monthly charges show a moderate relationship with churn.

## 4 Data Preparation

Categorical features were encoded:

- Binary variables with Label Encoding.
- Multi-class variables with One-Hot Encoding.

Numerical features were scaled with StandardScaler. Data was split into train (80%) and test (20%) sets with stratification on churn.

## 5 Modeling and Evaluation

Three classifiers were trained:

- Logistic Regression
- Decision Tree
- Random Forest

Their test performance is summarized in Table 2, highlighting Logistic Regression as the best-balanced model.

Table 2: Model Performance on Test Data

Model	Accuracy	F1 Score	ROC AUC
Logistic Regression	0.805	0.609	0.836
Decision Tree	0.711	0.454	0.629
Random Forest	0.787	0.558	0.814

## 6 Tuned Logistic Regression

GridSearchCV selected the best hyperparameters: `C=10`, `penalty=l2`, `solver=liblinear`. Cross-validation confirmed model stability with mean ROC AUC of 0.8443 (standard deviation 0.0100).

### 6.1 ROC Curve

Figure 5 shows the ROC curve, demonstrating strong model discrimination.

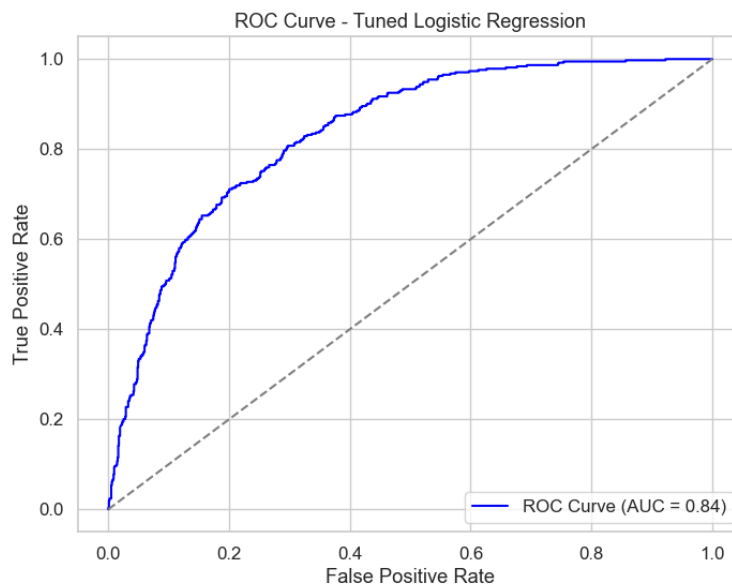


Figure 5: ROC Curve for Tuned Logistic Regression

## 7 Feature Importance: Coefficients and Odds Ratios

Figures 6 and 7 display the magnitude and direction of each feature's effect on churn probability. Features with coefficients  $> 0$  increase churn likelihood (odds ratio  $> 1$ ), while coefficients  $< 0$  reduce churn risk (odds ratio  $< 1$ ).

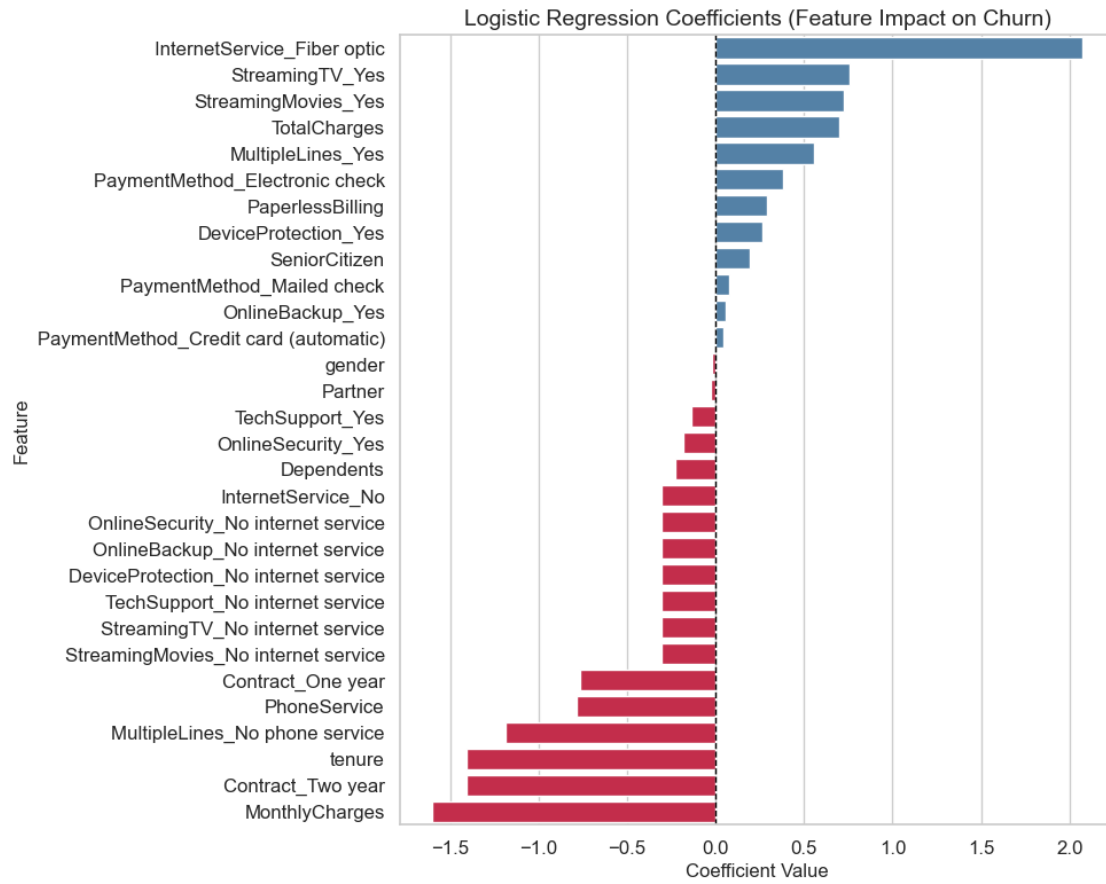


Figure 6: Logistic Regression Coefficients by Feature

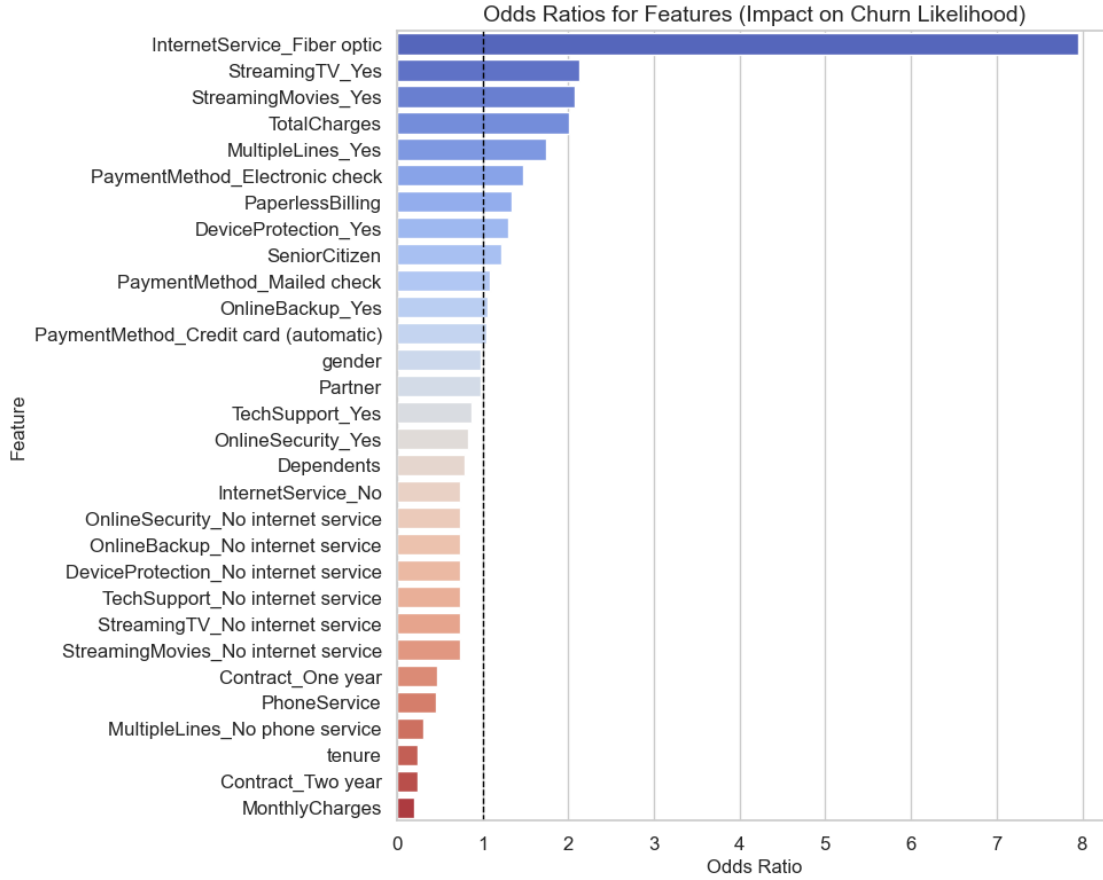


Figure 7: Odds Ratios of Features (Impact on Churn Likelihood)

Table 3: Top Features with Coefficients and Odds Ratios

Feature	Coefficient	Odds Ratio
InternetService_Fiber optic	2.07	7.95
StreamingTV_Yes	0.75	2.13
StreamingMovies_Yes	0.73	2.07
TotalCharges	0.70	2.01
MultipleLines_Yes	0.56	1.74
MonthlyCharges	-1.60	0.20
Contract_Two year	-1.41	0.24
tenure	-1.40	0.25
MultipleLines_No phone service	-1.19	0.31
Contract_One year	-0.76	0.47

## 8 Interpretation

- **Positive influencers on churn:** Customers with `Fiber optic` internet or who have enabled `StreamingTV/Movies` are significantly more likely to churn (odds ratios  $> 2$ ).
- **Negative influencers on churn:** Longer tenure, multi-year contracts, and lower monthly charges reduce churn risk considerably (odds ratios well below 1).
- Odds ratios near 1 (e.g., gender, partner status) have minimal influence on churn.

## 9 Marketing Recommendations

Based on the feature importances:

1. **Retention focus on fiber optic customers**, as they show the highest churn odds; exclusive loyalty offers or service bundling may improve retention.
2. **Encourage longer contracts**, especially two-year plans, since customers with these plans have over 75% lower odds of churning.
3. **Early intervention for high-monthly-charge customers:** proactive engagement within their first months can lower churn risk.
4. **Upsell tech support and security services:** while features like `TechSupport_Yes` reduce churn, many customers lack these services — converting them could aid retention.

## 10 Conclusion

The tuned Logistic Regression model provides a robust, interpretable approach to predict churn. Leveraging actionable insights from coefficients and odds ratios enables targeted marketing strategies that can materially reduce churn and improve customer lifetime value.

## Contact

For further details, contact: [johnolalemi90.email@gmail.com](mailto:johnolalemi90.email@gmail.com)

## A Appendix: Full Notebook Code



# ML for marketing

June 28, 2025

## 1 1 Project Overview & Objectives

- Project Goal: Analyze customer data from a telecommunications company to uncover patterns related to customer churn.
- Business Objective: Identify drivers of churn to inform marketing strategies that reduce customer attrition.
- Dataset: telco.csv with customer demographics, services subscribed, and churn labels.

## 2 2 Setup & Libraries

```
[37]: # Basic
import pandas as pd
import numpy as np

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Preprocessing & Modeling
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, \
    roc_auc_score, roc_curve, accuracy_score, precision_score, recall_score, \
    f1_score, auc
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression

# Hyperparameter Tuning
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

# Config
pd.set_option('display.max_columns', None)
sns.set(style='whitegrid', palette='muted', font_scale=1.1)
```

### 3 3 Data Loading

```
[2]: # Load dataset
telco_raw = pd.read_csv(r"C:\Users\USER\Documents\my_DS_projects\SUPERVISED\CLASSIFICATION\ML for_\Marketing\telco.csv")

# Quick check
telco_raw.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	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	1889.5	
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

## 4 4 Data Understanding

```
[3]: # Dataset shape
print(f"Shape: {telco_raw.shape}")

# Column types & non-null info
telco_raw.info()

# Descriptive stats (numerical features)
telco_raw.describe()

# Descriptive stats (categorical features)
telco_raw.describe(include='object')
```

```
Shape: (7043, 21)
<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
```

```
[3]:      customerID  gender  Partner  Dependents  PhoneService  MultipleLines  \
count          7043    7043     7043         7043          7043           7043
unique          7043         2         2          2           2            3
top      7590-VHVEG   Male      No          No           Yes            No
```

freq	1	3555	3641	4933	6361	3390
------	---	------	------	------	------	------

	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	\
count	7043	7043	7043	7043	
unique	3	3	3	3	
top	Fiber optic	No	No	No	
freq	3096	3498	3088	3095	

	TechSupport	StreamingTV	StreamingMovies	Contract	\
count	7043	7043	7043	7043	
unique	3	3	3	3	
top	No	No	No	Month-to-month	
freq	3473	2810	2785	3875	

	PaperlessBilling	PaymentMethod	TotalCharges	Churn
count	7043	7043	7043	7043
unique	2	4	6531	2
top	Yes	Electronic check		No
freq	4171	2365	11	5174

Based on the output above the following was observed - The dataset has 7,043 rows and 21 columns, with many categorical features and some numerical ones. - **TotalCharges** is stored as an object, indicating potential data quality issues. - No missing values in most columns, but we need to confirm after converting **TotalCharges**. **Action Plan:**

- Convert **TotalCharges** to numeric and handle any missing values.
- Encode categorical columns for modeling.
- Drop unnecessary columns like **customerID**.

## 5 5 Data Cleaning

Key Tasks: - Check for missing values - Convert **TotalCharges** from object to numeric - Identify and handle inconsistent values - Checkign duplicates values - Checking outliers - Checking Cardinality

```
[4]: # Missing values
telco_raw.isnull().sum()

# Convert 'TotalCharges' to numeric (coerce errors to NaN)
telco_raw['TotalCharges'] = pd.to_numeric(telco_raw['TotalCharges'],
    errors='coerce')

# Recheck missing
telco_raw.isnull().sum()

# Remove rows with missing TotalCharges (or fill)
telco_raw.dropna(subset=['TotalCharges'], inplace=True)

# Confirm cleaning
```

```
telco_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7032 entries, 0 to 7042
Data columns (total 21 columns):
#   Column                Non-Null Count  Dtype
---  -
0   customerID            7032 non-null   object
1   gender                 7032 non-null   object
2   SeniorCitizen          7032 non-null   int64
3   Partner                7032 non-null   object
4   Dependents             7032 non-null   object
5   tenure                 7032 non-null   int64
6   PhoneService           7032 non-null   object
7   MultipleLines           7032 non-null   object
8   InternetService        7032 non-null   object
9   OnlineSecurity         7032 non-null   object
10  OnlineBackup            7032 non-null   object
11  DeviceProtection        7032 non-null   object
12  TechSupport             7032 non-null   object
13  StreamingTV             7032 non-null   object
14  StreamingMovies         7032 non-null   object
15  Contract                7032 non-null   object
16  PaperlessBilling        7032 non-null   object
17  PaymentMethod           7032 non-null   object
18  MonthlyCharges          7032 non-null   float64
19  TotalCharges            7032 non-null   float64
20  Churn                   7032 non-null   object
dtypes: float64(2), int64(2), object(17)
memory usage: 1.2+ MB
```

We noticed that after cleaning the number of rows reduced from 7043 to 7032, that means about 11 rows were dropped as a result of the `TotalCharges` column. Now after cleaning we have a total of 4 numeric features and 17 objects features

## 5.1 Checking duplicates values

```
[5]: # Check for duplicate rows based on all columns
duplicate_rows = telco_raw.duplicated().sum()
print(f"Total duplicate rows: {duplicate_rows}")
```

```
Total duplicate rows: 0
```

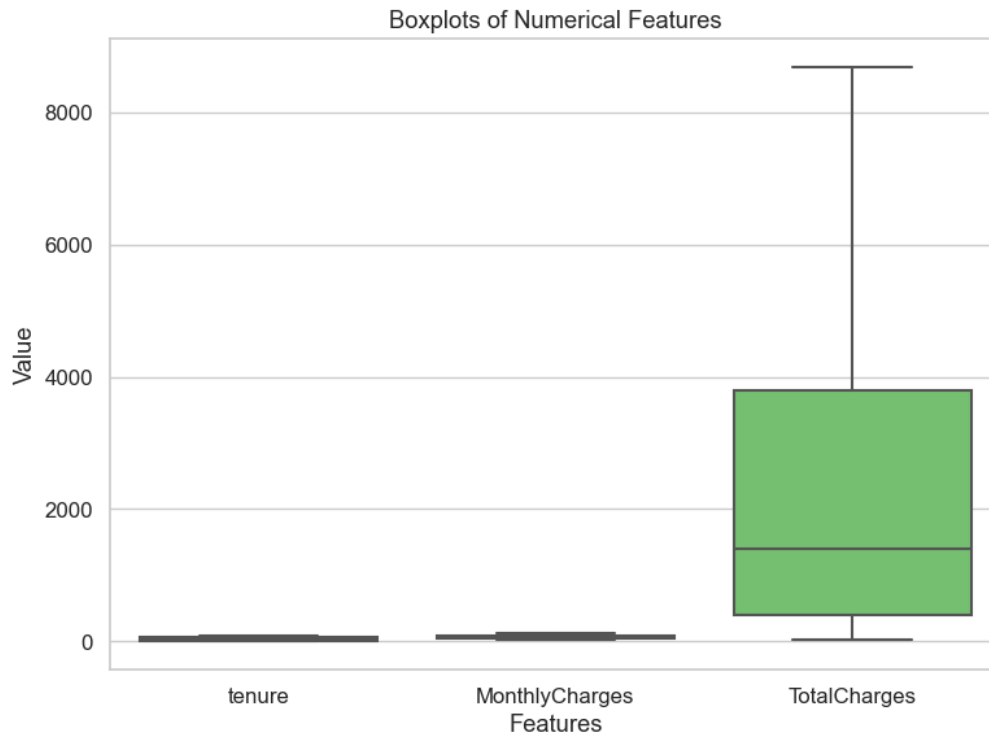
Even though each customer should have a unique `customerID`, it's wise to confirm

## 5.2 Checking for outliers

```
[6]: # Numerical columns you want to compare
num_cols = ['tenure', 'MonthlyCharges', 'TotalCharges']

# Melt the dataframe to long format
telco_melted = telco_raw[num_cols].melt(var_name='Feature', value_name='Value')

# Create a single boxplot
plt.figure(figsize=(8, 6))
sns.boxplot(data=telco_melted, x='Feature', y='Value')
plt.title('Boxplots of Numerical Features')
plt.xlabel('Features')
plt.ylabel('Value')
plt.tight_layout()
plt.show()
```



We see that there isn't an issue of outlier but we need to adjust the scales before we go ahead with modeling.

### 5.3 Checking high cardinality

```
[7]: # Select object columns
cat_cols = telco_raw.select_dtypes(include='object').columns

# Check unique values
for col in cat_cols:
    print(f"{col}: {telco_raw[col].nunique()} unique values")
```

```
customerID: 7032 unique values
gender: 2 unique values
Partner: 2 unique values
Dependents: 2 unique values
PhoneService: 2 unique values
MultipleLines: 3 unique values
InternetService: 3 unique values
OnlineSecurity: 3 unique values
OnlineBackup: 3 unique values
DeviceProtection: 3 unique values
TechSupport: 3 unique values
StreamingTV: 3 unique values
StreamingMovies: 3 unique values
Contract: 3 unique values
PaperlessBilling: 2 unique values
PaymentMethod: 4 unique values
Churn: 2 unique values
```

We do not have problematic high-cardinality categorical features, except `customerID` which we will need to drop now.

```
[8]: telco_raw.drop('customerID', axis=1, inplace=True)
telco_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7032 entries, 0 to 7042
Data columns (total 20 columns):
#   Column                Non-Null Count  Dtype
---  -
0   gender                7032 non-null  object
1   SeniorCitizen         7032 non-null  int64
2   Partner               7032 non-null  object
3   Dependents            7032 non-null  object
4   tenure                7032 non-null  int64
5   PhoneService          7032 non-null  object
6   MultipleLines         7032 non-null  object
7   InternetService       7032 non-null  object
8   OnlineSecurity        7032 non-null  object
9   OnlineBackup          7032 non-null  object
10  DeviceProtection      7032 non-null  object
```

```

11 TechSupport      7032 non-null  object
12 StreamingTV      7032 non-null  object
13 StreamingMovies  7032 non-null  object
14 Contract         7032 non-null  object
15 PaperlessBilling 7032 non-null  object
16 PaymentMethod    7032 non-null  object
17 MonthlyCharges   7032 non-null  float64
18 TotalCharges     7032 non-null  float64
19 Churn            7032 non-null  object
dtypes: float64(2), int64(2), object(16)
memory usage: 1.1+ MB

```

## 6 Exploratory Data Analysis (EDA)

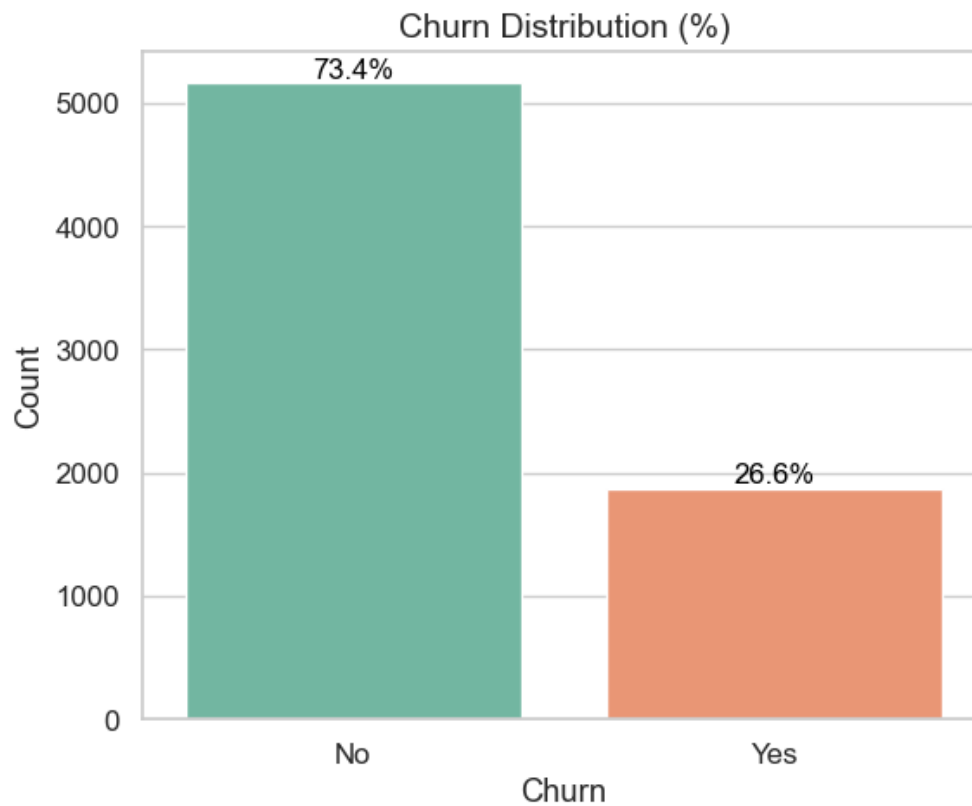
### 6.1 Target Variable (Churn)

```

[ ]: # Distribution of Churn
plt.figure(figsize=(6,5))
ax = sns.countplot(data=telco_raw, x='Churn', palette='Set2')
for p in ax.patches:
    height = p.get_height()
    ax.annotate(f'{100*height/len(telco_raw):.1f}%', (p.get_x()+p.get_width()/2.
↵, height),
                ha='center', va='bottom', fontsize=12, color='black')
plt.title('Churn Distribution (%)', fontsize=14)
plt.xlabel('Churn')
plt.ylabel('Count')
plt.tight_layout()
plt.show()

```





We noticed from the visuals above that there are more customers that did not churn.

```
[ ]: telco_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7032 entries, 0 to 7042
Data columns (total 20 columns):
#   Column                Non-Null Count  Dtype
---  -
0   gender                7032 non-null  object
1   SeniorCitizen         7032 non-null  int64
2   Partner               7032 non-null  object
3   Dependents            7032 non-null  object
4   tenure               7032 non-null  int64
5   PhoneService          7032 non-null  object
6   MultipleLines         7032 non-null  object
7   InternetService       7032 non-null  object
8   OnlineSecurity        7032 non-null  object
9   OnlineBackup          7032 non-null  object
```

```

10 DeviceProtection  7032 non-null  object
11 TechSupport      7032 non-null  object
12 StreamingTV      7032 non-null  object
13 StreamingMovies  7032 non-null  object
14 Contract         7032 non-null  object
15 PaperlessBilling 7032 non-null  object
16 PaymentMethod    7032 non-null  object
17 MonthlyCharges   7032 non-null  float64
18 TotalCharges     7032 non-null  float64
19 Churn            7032 non-null  object
dtypes: float64(2), int64(2), object(16)
memory usage: 1.1+ MB

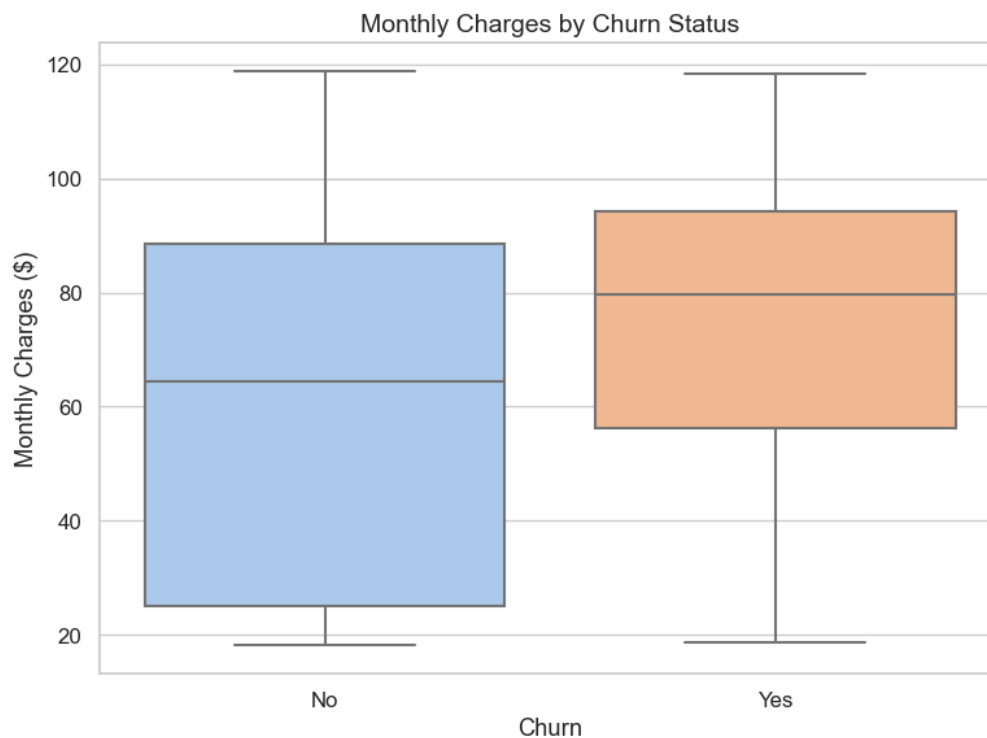
```

## 6.2 Monthly Charges vs. Churn (Boxplot)

```

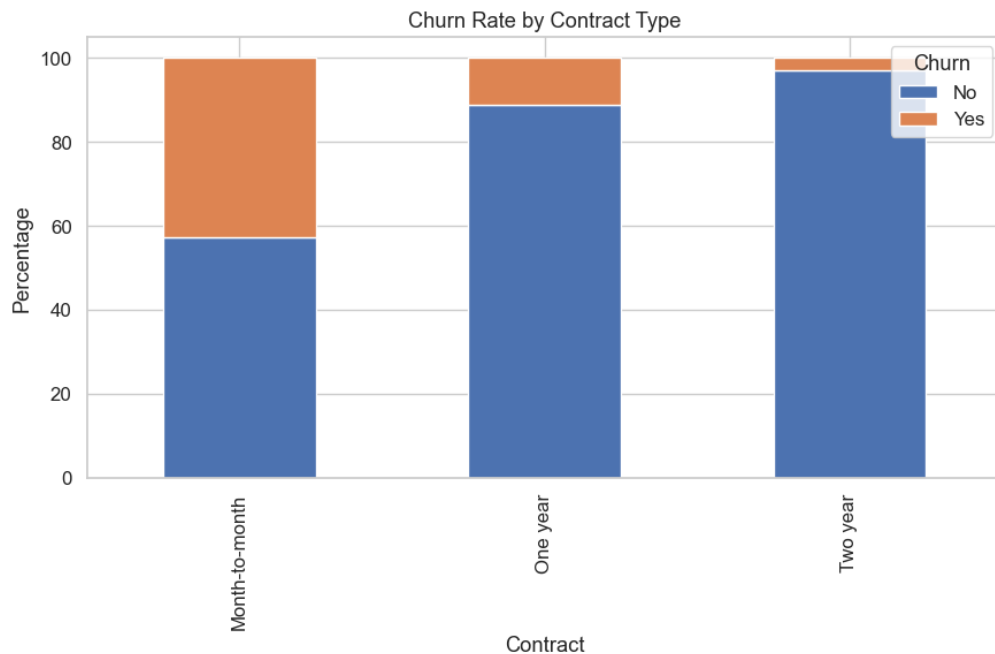
[ ]: plt.figure(figsize=(8,6))
sns.boxplot(data=telco_raw, x='Churn', y='MonthlyCharges', palette='pastel')
plt.title('Monthly Charges by Churn Status', fontsize=14)
plt.xlabel('Churn')
plt.ylabel('Monthly Charges ($)')
plt.tight_layout()
plt.show()

```



### 6.3 6.3 Contract Type vs. Churn (Stacked Bar)

```
[ ]: contract_churn = telco_raw.groupby(['Contract', 'Churn']).size().unstack().  
    fillna(0)  
contract_churn_pct = contract_churn.div(contract_churn.sum(axis=1), axis=0) *  
    100  
  
contract_churn_pct.plot(kind='bar', stacked=True, figsize=(9,6),  
                        color=['#4C72B0', '#DD8452'])  
plt.title('Churn Rate by Contract Type')  
plt.ylabel('Percentage')  
plt.legend(title='Churn', loc='upper right')  
plt.tight_layout()  
plt.show()
```



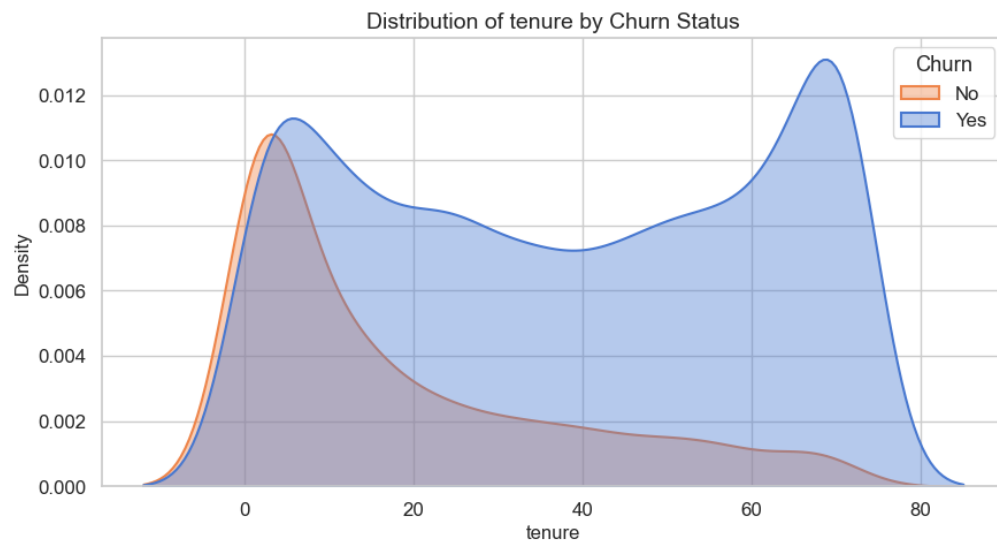
### 6.4 6.4 Numerical Features vs Churn

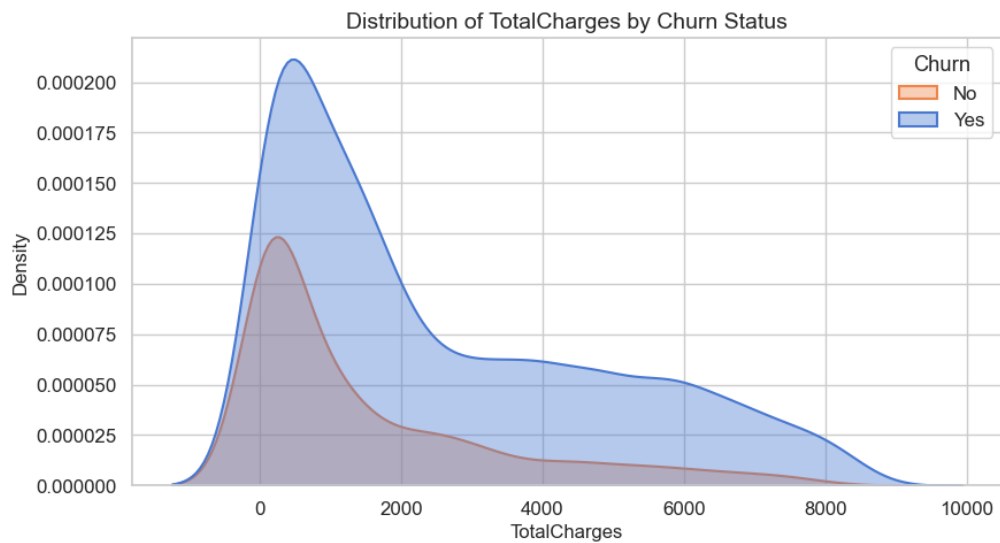
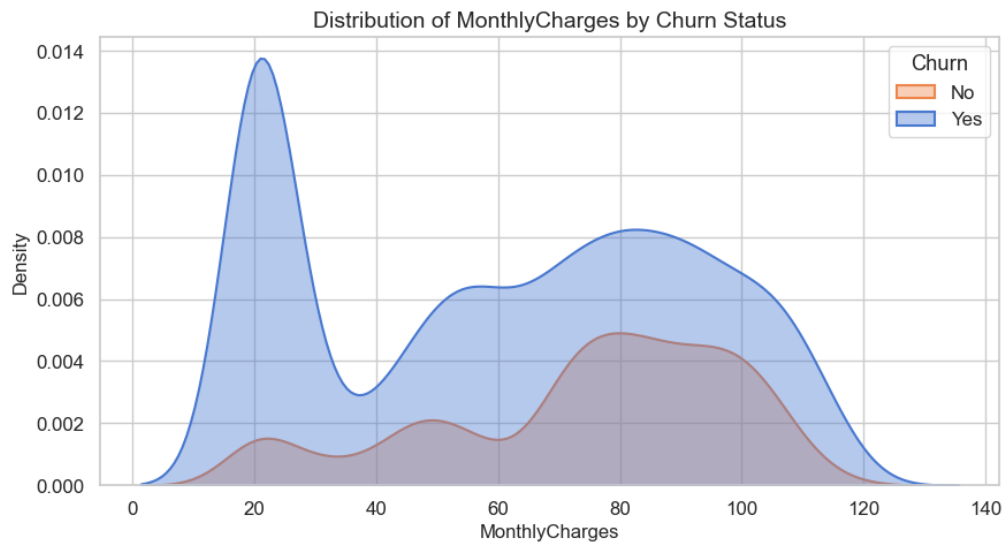
```
[ ]: num_features = ['tenure', 'MonthlyCharges', 'TotalCharges']  
  
for col in num_features:  
    plt.figure(figsize=(9, 5))
```

```

ax = sns.kdeplot(
    data=telco_raw, x=col, hue='Churn', fill=True, alpha=0.4, linewidth=1.5
)
ax.set_title(f"Distribution of {col} by Churn Status", fontsize=14)
ax.set_xlabel(col, fontsize=12)
ax.set_ylabel("Density", fontsize=12)
plt.legend(title='Churn', labels=['No', 'Yes'])
plt.tight_layout()
plt.savefig(f"{col}_distribution.png", dpi=300) # Save for LaTeX inclusion
plt.show()

```

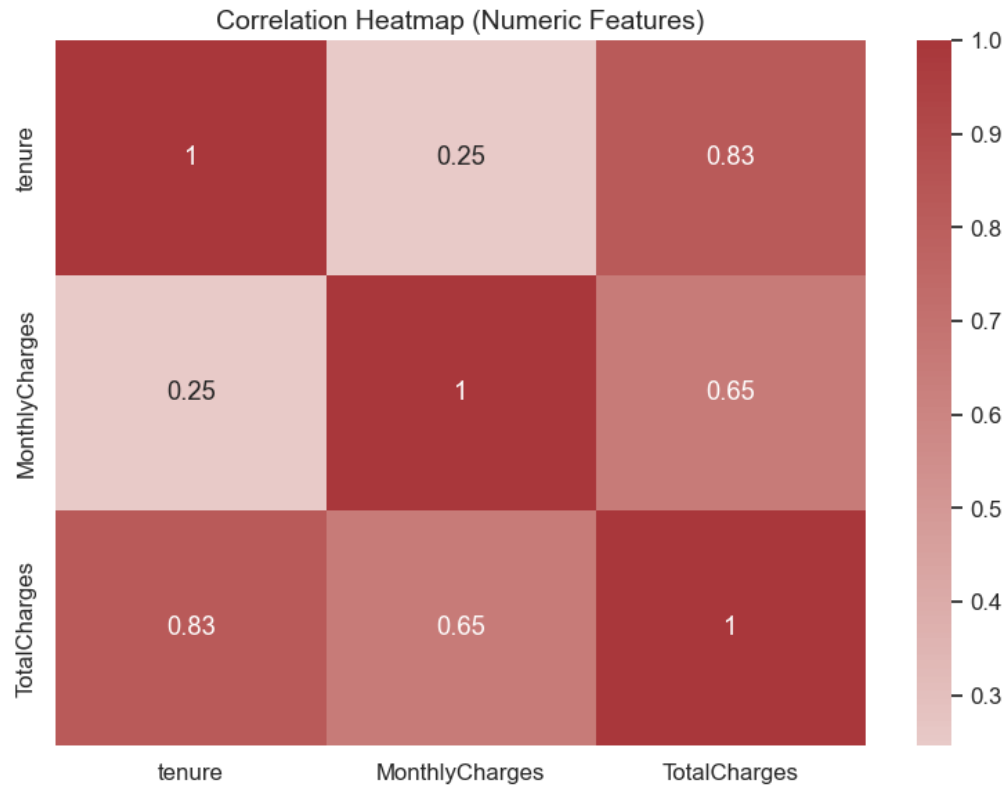




## 6.5 6.5 Correlation Heatmap of Numeric Features

```
[ ]: num_cols = ['tenure', 'MonthlyCharges', 'TotalCharges', 'Churn']
plt.figure(figsize=(8,6))
sns.heatmap(telco_raw[num_cols].corr(numeric_only=True), annot=True,
           cmap='vlag', center=0)
plt.title('Correlation Heatmap (Numeric Features)', fontsize=14)
```

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



## 7 7 Feature Engineering & Preprocessing

Steps: - Encode categorical variables - Scale numerical variables - Create training & test sets

```
[23]: # Encode target variable
telco_raw['Churn'] = telco_raw['Churn'].map({'No':0, 'Yes':1})

# Encode binary categorical features
binary_cols = ['gender', 'Partner', 'Dependents', 'PhoneService',
               ↪ 'PaperlessBilling']

le = LabelEncoder()
for col in binary_cols:
    telco_raw[col] = le.fit_transform(telco_raw[col])

# One-hot encode multi-class categorical features
```

```

multi_cols = ['MultipleLines', 'InternetService', 'OnlineSecurity',
↳ 'OnlineBackup',
               'DeviceProtection', 'TechSupport', 'StreamingTV',
↳ 'StreamingMovies',
               'Contract', 'PaymentMethod']

telco_raw = pd.get_dummies(telco_raw, columns=multi_cols, drop_first=True)

```

```

[26]: # Split features and target
X = telco_raw.drop('Churn', axis=1)
y = telco_raw['Churn']

# Train-test split (stratified to preserve churn ratio)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

# Scale numerical features
scaler = StandardScaler()
num_cols = ['tenure', 'MonthlyCharges', 'TotalCharges']

X_train[num_cols] = scaler.fit_transform(X_train[num_cols])
X_test[num_cols] = scaler.transform(X_test[num_cols])

# Check shapes to confirm
print("Train features shape:", X_train.shape)
print("Test features shape:", X_test.shape)

```

Train features shape: (5625, 30)  
Test features shape: (1407, 30)

## 8 Modeling

```

[30]: # Function for Evaluation
def evaluate_model(model, X_test, y_test, model_name="Model"):
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:,:1] if hasattr(model,
↳ "predict_proba") else None

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_prob) if y_prob is not None else
↳ roc_auc_score(y_test, y_pred)

```

```

print(f"\n {model_name} Performance:")
print(f"Accuracy: {acc:.4f}")
print(f"Precision: {prec:.4f}")
print(f"Recall: {rec:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"ROC AUC: {roc_auc:.4f}")

```

```

[38]: rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
evaluate_model(rf_model, X_train, y_train, model_name="Random Forest (Train)")
evaluate_model(rf_model, X_test, y_test, model_name="Random Forest (Test)")

```

Random Forest (Train) Performance:

Accuracy: 0.9988  
Precision: 0.9987  
Recall: 0.9967  
F1 Score: 0.9977  
ROC AUC: 1.0000

Random Forest (Test) Performance:

Accuracy: 0.7868  
Precision: 0.6217  
Recall: 0.5053  
F1 Score: 0.5575  
ROC AUC: 0.8142

```

[40]: dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)
evaluate_model(dt_model, X_train, y_train, model_name="Decision Tree(Train)")
evaluate_model(dt_model, X_test, y_test, model_name="Decision Tree (Test)")

```

Decision Tree(Train) Performance:

Accuracy: 0.9988  
Precision: 0.9993  
Recall: 0.9960  
F1 Score: 0.9977  
ROC AUC: 1.0000

Decision Tree (Test) Performance:

Accuracy: 0.7114  
Precision: 0.4568  
Recall: 0.4519  
F1 Score: 0.4543  
ROC AUC: 0.6286



```
[41]: lr_model = LogisticRegression(max_iter=1000, random_state=42)
lr_model.fit(X_train, y_train)
evaluate_model(lr_model, X_train, y_train, model_name="Logistic_
Regression(Train)")
evaluate_model(lr_model, X_test, y_test, model_name="Logistic Regression(Test)")
```

Logistic Regression(Train) Performance:  
Accuracy: 0.8048  
Precision: 0.6592  
Recall: 0.5498  
F1 Score: 0.5996  
ROC AUC: 0.8506

Logistic Regression(Test) Performance:  
Accuracy: 0.8045  
Precision: 0.6505  
Recall: 0.5722  
F1 Score: 0.6088  
ROC AUC: 0.8361

### 8.0.1 Comparing the results

```
[44]: results = []

# === RANDOM FOREST ===
# Train metrics
y_train_pred_rf = rf_model.predict(X_train)
y_train_prob_rf = rf_model.predict_proba(X_train)[: ,1]
results.append({
    'Model': 'Random Forest',
    'Set': 'Train',
    'Accuracy': accuracy_score(y_train, y_train_pred_rf),
    'Precision': precision_score(y_train, y_train_pred_rf),
    'Recall': recall_score(y_train, y_train_pred_rf),
    'F1 Score': f1_score(y_train, y_train_pred_rf),
    'ROC AUC': roc_auc_score(y_train, y_train_prob_rf)
})

# Test metrics
y_test_pred_rf = rf_model.predict(X_test)
y_test_prob_rf = rf_model.predict_proba(X_test)[: ,1]
results.append({
    'Model': 'Random Forest',
    'Set': 'Test',
    'Accuracy': accuracy_score(y_test, y_test_pred_rf),
    'Precision': precision_score(y_test, y_test_pred_rf),
```

```

        'Recall': recall_score(y_test, y_test_pred_rf),
        'F1 Score': f1_score(y_test, y_test_pred_rf),
        'ROC AUC': roc_auc_score(y_test, y_test_prob_rf)
    })

# === DECISION TREE ===
# Train metrics
y_train_pred_dt = dt_model.predict(X_train)
y_train_prob_dt = dt_model.predict_proba(X_train)[: ,1]
results.append({
    'Model': 'Decision Tree',
    'Set': 'Train',
    'Accuracy': accuracy_score(y_train, y_train_pred_dt),
    'Precision': precision_score(y_train, y_train_pred_dt),
    'Recall': recall_score(y_train, y_train_pred_dt),
    'F1 Score': f1_score(y_train, y_train_pred_dt),
    'ROC AUC': roc_auc_score(y_train, y_train_prob_dt)
})

# Test metrics
y_test_pred_dt = dt_model.predict(X_test)
y_test_prob_dt = dt_model.predict_proba(X_test)[: ,1]
results.append({
    'Model': 'Decision Tree',
    'Set': 'Test',
    'Accuracy': accuracy_score(y_test, y_test_pred_dt),
    'Precision': precision_score(y_test, y_test_pred_dt),
    'Recall': recall_score(y_test, y_test_pred_dt),
    'F1 Score': f1_score(y_test, y_test_pred_dt),
    'ROC AUC': roc_auc_score(y_test, y_test_prob_dt)
})

# === LOGISTIC REGRESSION ===
# Train metrics
y_train_pred_lr = lr_model.predict(X_train)
y_train_prob_lr = lr_model.predict_proba(X_train)[: ,1]
results.append({
    'Model': 'Logistic Regression',
    'Set': 'Train',
    'Accuracy': accuracy_score(y_train, y_train_pred_lr),
    'Precision': precision_score(y_train, y_train_pred_lr),
    'Recall': recall_score(y_train, y_train_pred_lr),
    'F1 Score': f1_score(y_train, y_train_pred_lr),
    'ROC AUC': roc_auc_score(y_train, y_train_prob_lr)
})

```

```

# Test metrics
y_test_pred_lr = lr_model.predict(X_test)
y_test_prob_lr = lr_model.predict_proba(X_test)[:,:1]
results.append({
    'Model': 'Logistic Regression',
    'Set': 'Test',
    'Accuracy': accuracy_score(y_test, y_test_pred_lr),
    'Precision': precision_score(y_test, y_test_pred_lr),
    'Recall': recall_score(y_test, y_test_pred_lr),
    'F1 Score': f1_score(y_test, y_test_pred_lr),
    'ROC AUC': roc_auc_score(y_test, y_test_prob_lr)
})

# === Create DataFrame ===
results_df = pd.DataFrame(results).set_index(['Model', 'Set'])
display(results_df)

```

			Accuracy	Precision	Recall	F1 Score	ROC AUC
Model	Set						
Random Forest	Train	0.998756	0.998660	0.996656	0.997657	0.999958	
	Test	0.786780	0.621711	0.505348	0.557522	0.814197	
Decision Tree	Train	0.998756	0.999329	0.995987	0.997655	0.999995	
	Test	0.711443	0.456757	0.451872	0.454301	0.628600	
Logistic Regression	Train	0.804800	0.659182	0.549833	0.599562	0.850569	
	Test	0.804549	0.650456	0.572193	0.608819	0.836071	

```

[45]: param_grid = {
    'C': [0.01, 0.1, 1, 10],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear']
}

grid = GridSearchCV(LogisticRegression(max_iter=1000, random_state=42),
    param_grid, cv=5, scoring='roc_auc')
grid.fit(X_train, y_train)

print("Best parameters:", grid.best_params_)
evaluate_model(grid.best_estimator_, X_test, y_test, model_name="Tuned Logistic_
    Regression")

```

Best parameters: {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}

Tuned Logistic Regression Performance:

Accuracy: 0.8010

Precision: 0.6407

Recall: 0.5722

F1 Score: 0.6045

ROC AUC: 0.8354

```
[46]: cv_scores = cross_val_score(grid.best_estimator_, X, y, cv=5, scoring='roc_auc')
print("Cross-validated ROC AUC scores:", cv_scores)
print("Mean ROC AUC: {:.4f}".format(cv_scores.mean()))
print("Standard Deviation: {:.4f}".format(cv_scores.std()))
```

Cross-validated ROC AUC scores: [0.85706447 0.85592299 0.83443288 0.83612113 0.83796843]

Mean ROC AUC: 0.8443

Standard Deviation: 0.0100

## 9 Result interpretation

```
[54]: # Create DataFrame as before
coefficients = pd.DataFrame({
    'Feature': X.columns,
    'Coefficient': grid.best_estimator_.coef_[0]
})

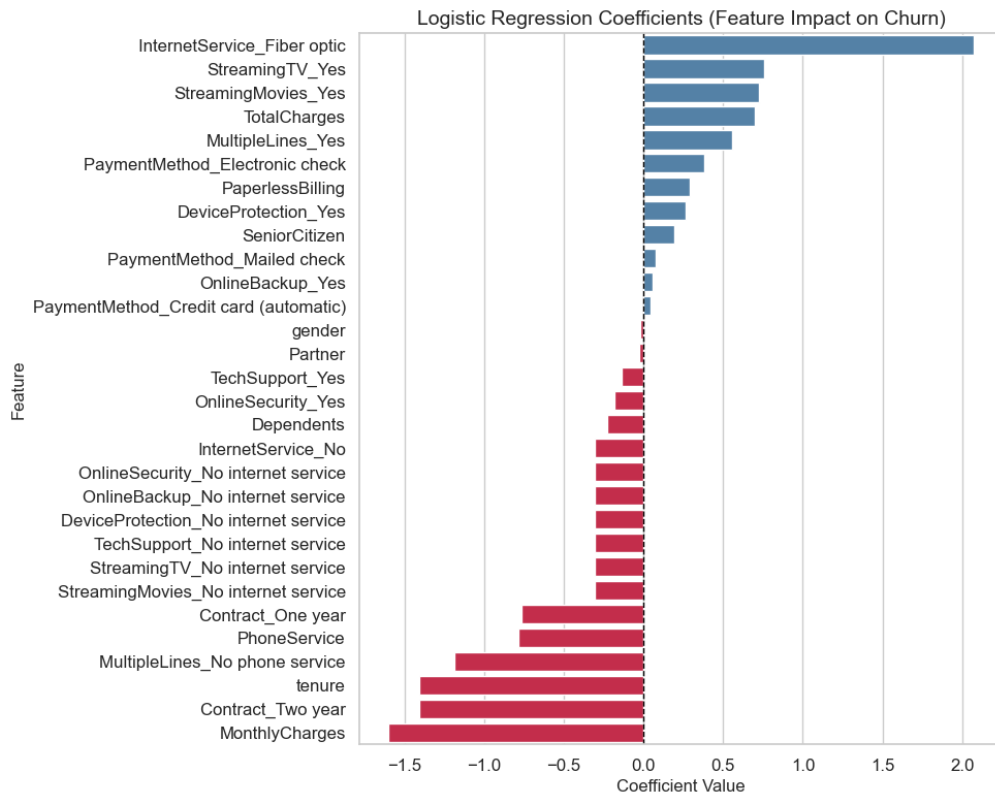
# Add absolute value for sorting convenience
coefficients['AbsCoefficient'] = coefficients['Coefficient'].abs()

# Sort: first by sign (positive first), then by absolute magnitude descending
coefficients_sorted = coefficients.sort_values(
    by=['Coefficient', 'AbsCoefficient'],
    ascending=[False, False] # positive first, largest magnitude first
).drop(columns='AbsCoefficient').reset_index(drop=True)

display(coefficients_sorted)

plt.figure(figsize=(10, 8))
sns.barplot(
    x='Coefficient',
    y='Feature',
    data=coefficients_sorted,
    palette=['crimson' if c < 0 else 'steelblue' for c in
coefficients_sorted['Coefficient']]
)
plt.axvline(0, color='black', linestyle='--', linewidth=1)
plt.title("Logistic Regression Coefficients (Feature Impact on Churn)",
fontsize=14)
plt.xlabel("Coefficient Value", fontsize=12)
plt.ylabel("Feature", fontsize=12)
plt.tight_layout()
plt.show()
```

	Feature	Coefficient
0	InternetService_Fiber optic	2.073051
1	StreamingTV_Yes	0.754252
2	StreamingMovies_Yes	0.727223
3	TotalCharges	0.698652
4	MultipleLines_Yes	0.555584
5	PaymentMethod_Electronic check	0.383086
6	PaperlessBilling	0.289443
7	DeviceProtection_Yes	0.265422
8	SeniorCitizen	0.193575
9	PaymentMethod_Mailed check	0.076093
10	OnlineBackup_Yes	0.057393
11	PaymentMethod_Credit card (automatic)	0.042106
12	gender	-0.022692
13	Partner	-0.026531
14	TechSupport_Yes	-0.138415
15	OnlineSecurity_Yes	-0.182745
16	Dependents	-0.229527
17	InternetService_No	-0.303386
18	OnlineSecurity_No internet service	-0.303386
19	OnlineBackup_No internet service	-0.303386
20	DeviceProtection_No internet service	-0.303386
21	TechSupport_No internet service	-0.303386
22	StreamingTV_No internet service	-0.303386
23	StreamingMovies_No internet service	-0.303386
24	Contract_One year	-0.761825
25	PhoneService	-0.782985
26	MultipleLines_No phone service	-1.186315
27	tenure	-1.403817
28	Contract_Two year	-1.409528
29	MonthlyCharges	-1.598564



```
[ ]: # Create a DataFrame with features, coefficients, and odds ratios
odds_df = coefficients_sorted[['Feature', 'Coefficient']].copy()
odds_df['Odds Ratio'] = np.exp(odds_df['Coefficient'])

# Reset index for a neat presentation
odds_df.reset_index(drop=True, inplace=True)

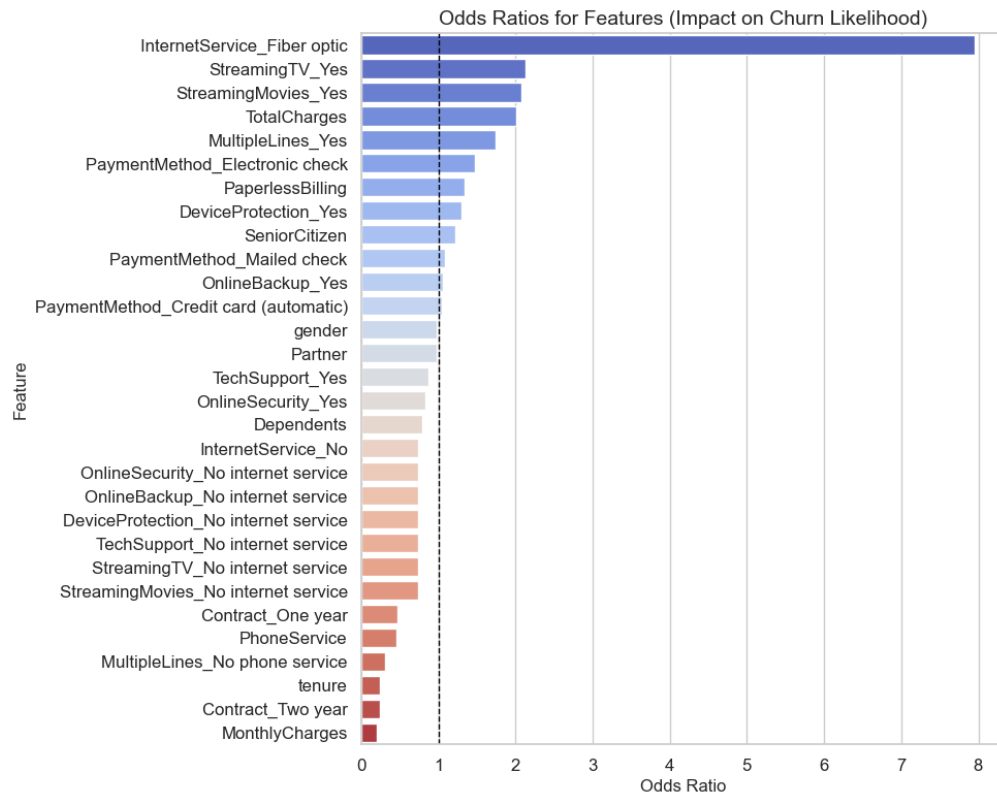
display(odds_df)

# Calculate Odds Ratios on your sorted DataFrame
coefficients_sorted['Odds Ratio'] = np.exp(coefficients_sorted['Coefficient'])

plt.figure(figsize=(10, 8))
sns.barplot(
    x='Odds Ratio',
    y='Feature',
    data=coefficients_sorted,
    palette='coolwarm'
)
```

```
plt.axvline(1, color='black', linestyle='--', linewidth=1)
plt.title("Odds Ratios for Features (Impact on Churn Likelihood)", fontsize=14)
plt.xlabel("Odds Ratio", fontsize=12)
plt.ylabel("Feature", fontsize=12)
plt.tight_layout()
plt.show();
```

	Feature	Coefficient	Odds Ratio
0	InternetService_Fiber optic	2.073051	7.949035
1	StreamingTV_Yes	0.754252	2.126021
2	StreamingMovies_Yes	0.727223	2.069327
3	TotalCharges	0.698652	2.011041
4	MultipleLines_Yes	0.555584	1.742959
5	PaymentMethod_Electronic check	0.383086	1.466805
6	PaperlessBilling	0.289443	1.335683
7	DeviceProtection_Yes	0.265422	1.303981
8	SeniorCitizen	0.193575	1.213580
9	PaymentMethod_Mailed check	0.076093	1.079062
10	OnlineBackup_Yes	0.057393	1.059072
11	PaymentMethod_Credit card (automatic)	0.042106	1.043006
12	gender	-0.022692	0.977563
13	Partner	-0.026531	0.973818
14	TechSupport_Yes	-0.138415	0.870737
15	OnlineSecurity_Yes	-0.182745	0.832980
16	Dependents	-0.229527	0.794909
17	InternetService_No	-0.303386	0.738314
18	OnlineSecurity_No internet service	-0.303386	0.738314
19	OnlineBackup_No internet service	-0.303386	0.738314
20	DeviceProtection_No internet service	-0.303386	0.738314
21	TechSupport_No internet service	-0.303386	0.738314
22	StreamingTV_No internet service	-0.303386	0.738314
23	StreamingMovies_No internet service	-0.303386	0.738314
24	Contract_One year	-0.761825	0.466814
25	PhoneService	-0.782985	0.457040
26	MultipleLines_No phone service	-1.186315	0.305344
27	tenure	-1.403817	0.245658
28	Contract_Two year	-1.409528	0.244259
29	MonthlyCharges	-1.598564	0.202187



```
[ ]: y_test_prob = grid.best_estimator_.predict_proba(X_test)[:,-1]
fpr, tpr, _ = roc_curve(y_test, y_test_prob)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {roc_auc:.2f})", color='blue')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Tuned Logistic Regression')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



