01_exploration

January 28, 2026

1 Telecom Customer Churn - Data Exploration

Project: COMP 9130 - Mini Project 2: Classification Challenge
Dataset: Telecom Customer Churn
Objective: Predict which customers will leave (churn)

1.1 Table of Contents

1.2 1. Import Libraries

[1]:
```
# Data manipulation
import pandas as pd
import numpy as np

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

# Statistical analysis
from scipy import stats

# Settings
import warnings
warnings.filterwarnings('ignore')

# Display settings
pd.set_option('display.max_columns', None)
```

```
pd.set_option('display.max_rows', 100)
plt.style.use('seaborn-v0_8-darkgrid')
sns.set_palette('husl')

# Set random seed for reproducibility
np.random.seed(42)

print("Libraries imported successfully!")
```

```
Libraries imported successfully!
```

1.3 2. Load Dataset

Dataset Information: - **Source:** Telco Customer Churn (Kaggle) - **Size:** 7,043 samples - **Features:** 20 (demographics, services, charges, tenure) - **Target:** Binary (Yes/No churn) - **Expected Imbalance:** ~27% churn, 73% stay

[2]:
```
# Load the dataset
# Note: Update the path to where you've saved the dataset
df = pd.read_csv('../data/WA_Fn-UseC_-Telco-Customer-Churn.csv')

print(f"Dataset loaded successfully!")
print(f"Shape: {df.shape}")
print(f"\nRows: {df.shape[0]:,}")
print(f"Columns: {df.shape[1]}")
```

```
Dataset loaded successfully!
Shape: (7043, 21)

Rows: 7,043
Columns: 21
```

1.4 3. Initial Data Inspection

[3]:
```
# First few rows
print("First 5 rows of the dataset:")
df.head()
```

```
First 5 rows of the dataset:
```

[3]:
```
   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]:

```python
# Dataset info
print("Dataset Information:")
df.info()
```

```
Dataset Information:
<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
```

[5]:
```python
# Statistical summary
print("Statistical Summary of Numerical Features:")
df.describe()
```

```
Statistical Summary of Numerical Features:
```

[5]:

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

[6]:
```python
print(f"Dataset shape: {df.shape[0]} rows, {df.shape[1]} columns")
```

```
Dataset shape: 7043 rows, 21 columns
```

[7]:
```python
# Identify feature types
numerical_features = df.select_dtypes(include=['int64', 'float64']).columns.
 ↪tolist()
categorical_features = df.select_dtypes(include=['object']).columns.tolist()

# Remove target variable from categorical features if present
if 'Churn' in categorical_features:
    categorical_features.remove('Churn')

# Remove customerID if present
if 'customerID' in categorical_features:
    categorical_features.remove('customerID')

print(f"Numerical Features ({len(numerical_features)}): {numerical_features}")
print(f"\nCategorical Features ({len(categorical_features)}):␣
 ↪{categorical_features}")
```

```
Numerical Features (3): ['SeniorCitizen', 'tenure', 'MonthlyCharges']

Categorical Features (16): ['gender', 'Partner', 'Dependents', 'PhoneService',
'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup',
'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract',
'PaperlessBilling', 'PaymentMethod', 'TotalCharges']
```

1.5 4. Target Variable Analysis

Critical for Classification: Understanding class distribution helps us: - Detect class imbalance - Choose appropriate evaluation metrics - Decide on imbalance handling strategies

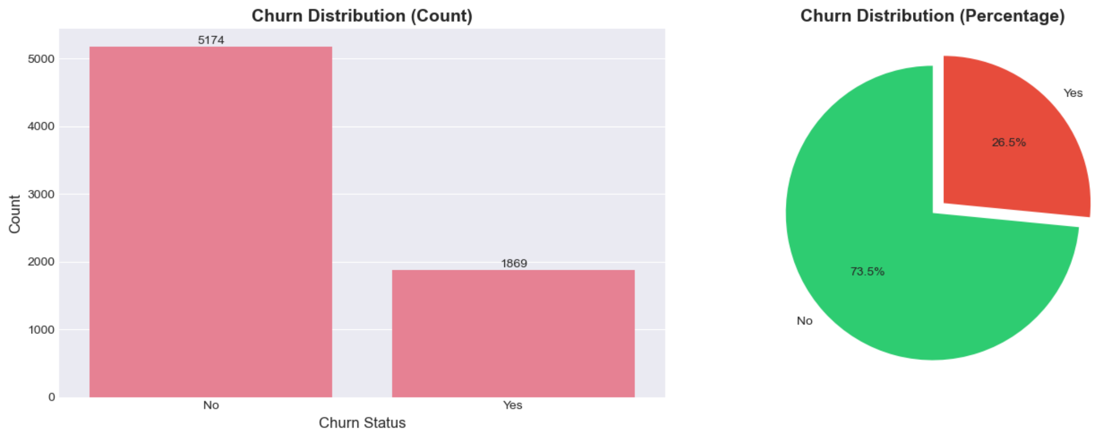
[8]:
```python
# Target variable distribution
print("Target Variable: Churn")
print("=" * 50)
churn_counts = df['Churn'].value_counts()
churn_percentages = df['Churn'].value_counts(normalize=True) * 100

target_summary = pd.DataFrame({
    'Count': churn_counts,
    'Percentage': churn_percentages
})

print(target_summary)
print("\n" + "=" * 50)

# Calculate imbalance ratio
minority_class = churn_counts.min()
majority_class = churn_counts.max()
imbalance_ratio = majority_class / minority_class

print(f"\nImbalance Ratio: {imbalance_ratio:.2f}:1")
print(f"Minority Class Size: {minority_class:,} ({churn_percentages.min():.
 ↪2f}%)")
print(f"Majority Class Size: {majority_class:,} ({churn_percentages.max():.
 ↪2f}%)")

# Determine if imbalanced
minority_percentage = churn_percentages.min()
if minority_percentage < 40 or minority_percentage > 60:
    print(f"\n   IMBALANCED DATASET DETECTED!")
    print(f"   We need to apply imbalance handling techniques.")
else:
    print(f"\n  Dataset is relatively balanced.")
```

```
Target Variable: Churn
==================================================
       Count  Percentage
```

```
Churn
No      5174   73.463013
Yes     1869   26.536987

==================================================

Imbalance Ratio: 2.77:1
Minority Class Size: 1,869 (26.54%)
Majority Class Size: 5,174 (73.46%)

   IMBALANCED DATASET DETECTED!
    We need to apply imbalance handling techniques.
```

The dataset is imbalanced (~27% churn, ~73% non-churn), therefore accuracy alone is misleading.

[9]:
```python
# Visualize target distribution
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Count plot
sns.countplot(data=df, x='Churn', ax=axes[0])
axes[0].set_title('Churn Distribution (Count)', fontsize=14, fontweight='bold')
axes[0].set_xlabel('Churn Status', fontsize=12)
axes[0].set_ylabel('Count', fontsize=12)

# Add value labels on bars
for container in axes[0].containers:
    axes[0].bar_label(container, fmt='%d')

# Pie chart
colors = ['#2ecc71', '#e74c3c']  # Green for No, Red for Yes
axes[1].pie(churn_counts, labels=churn_counts.index, autopct='%1.1f%%',
            startangle=90, colors=colors, explode=(0, 0.1))
axes[1].set_title('Churn Distribution (Percentage)', fontsize=14,␣
 ↪fontweight='bold')

plt.tight_layout()
plt.show()

print("\n  Interpretation:")
print("   - The dataset is IMBALANCED with approximately 73% non-churn and 27%␣
 ↪churn.")
print("   - We should NOT use accuracy as the primary metric.")
print("   - We should use F1-score, ROC-AUC, or precision-recall metrics␣
 ↪instead.")
print("   - We'll need to handle this imbalance in the modeling phase.")
```




```
Interpretation:
  - The dataset is IMBALANCED with approximately 73% non-churn and 27% churn.
  - We should NOT use accuracy as the primary metric.
  - We should use F1-score, ROC-AUC, or precision-recall metrics instead.
  - We'll need to handle this imbalance in the modeling phase.
```

1.6 5. Missing Values Analysis

[10]:
```
# Check for missing values
missing_values = df.isnull().sum()
missing_percentage = (df.isnull().sum() / len(df)) * 100

missing_df = pd.DataFrame({
    'Missing_Count': missing_values,
    'Percentage': missing_percentage
})

missing_df = missing_df[missing_df['Missing_Count'] > 0].
 ↪sort_values('Missing_Count', ascending=False)

if len(missing_df) > 0:
    print("Missing Values Found:")
    print("=" * 50)
    print(missing_df)

    # Visualize missing values
    plt.figure(figsize=(10, 6))
    missing_df['Percentage'].plot(kind='barh')
    plt.xlabel('Percentage of Missing Values')
    plt.title('Missing Values by Feature')
    plt.tight_layout()
```

```
    plt.show()
else:
    print("  No missing values found in the dataset!")
```

```
 No missing values found in the dataset!
```

[11]:
```
# Check for potential hidden missing values (empty strings, whitespace, etc.)
print("Checking for hidden missing values (empty strings, whitespace, etc.)...")
print("=" * 50)

for col in df.columns:
    if df[col].dtype == 'object':
        # Check for empty strings or whitespace
        empty_count = (df[col].str.strip() == '').sum()
        whitespace_count = (df[col].str.isspace()).sum()

        if empty_count > 0 or whitespace_count > 0:
            print(f"{col}: {empty_count} empty strings, {whitespace_count}␣
↪whitespace-only values")

print("\nCheck complete.")
```

```
Checking for hidden missing values (empty strings, whitespace, etc.)…
==================================================
TotalCharges: 11 empty strings, 11 whitespace-only values

Check complete.
```

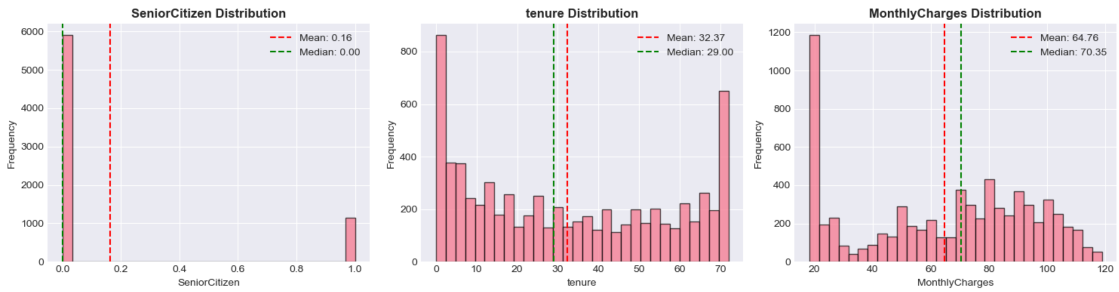
1.7 6. Feature Analysis

1.7.1 6.1 Numerical Features

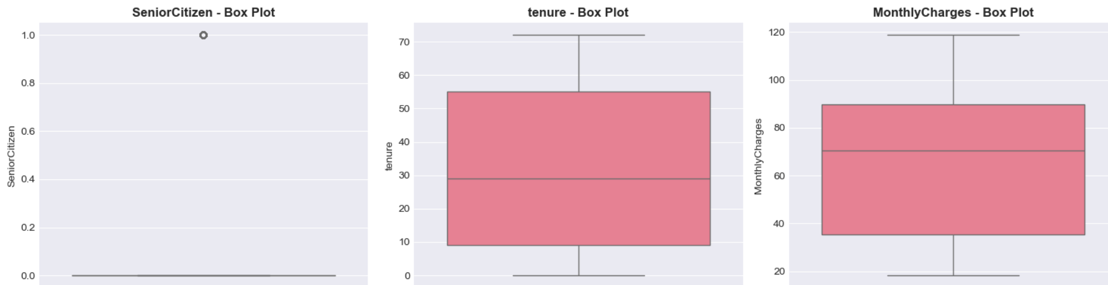
[12]:
```
# Distribution of numerical features
print(f"Analyzing {len(numerical_features)} numerical features...")

# Create subplots for all numerical features
n_features = len(numerical_features)
n_cols = 3
n_rows = (n_features + n_cols - 1) // n_cols

fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, n_rows * 4))
axes = axes.flatten() if n_features > 1 else [axes]

for idx, feature in enumerate(numerical_features):
    # Histogram with KDE
    axes[idx].hist(df[feature].dropna(), bins=30, edgecolor='black', alpha=0.7)
    axes[idx].set_title(f'{feature} Distribution', fontweight='bold')
    axes[idx].set_xlabel(feature)
    axes[idx].set_ylabel('Frequency')
```

```
    # Add mean and median lines
    mean_val = df[feature].mean()
    median_val = df[feature].median()
    axes[idx].axvline(mean_val, color='red', linestyle='--', label=f'Mean:␣
↪{mean_val:.2f}')
    axes[idx].axvline(median_val, color='green', linestyle='--', label=f'Median:
↪ {median_val:.2f}')
    axes[idx].legend()

# Hide extra subplots
for idx in range(n_features, len(axes)):
    axes[idx].set_visible(False)

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

Analyzing 3 numerical features…

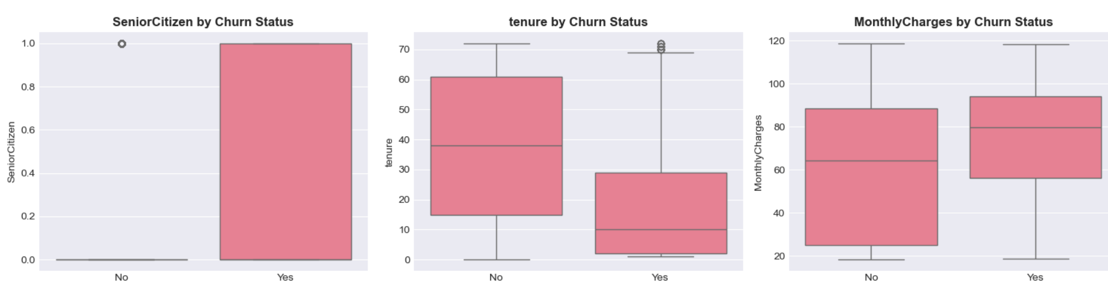


[13]:
```
# Box plots to identify outliers in numerical features
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, n_rows * 4))
axes = axes.flatten() if n_features > 1 else [axes]

for idx, feature in enumerate(numerical_features):
    sns.boxplot(data=df, y=feature, ax=axes[idx])
    axes[idx].set_title(f'{feature} - Box Plot', fontweight='bold')
    axes[idx].set_ylabel(feature)

# Hide extra subplots
for idx in range(n_features, len(axes)):
    axes[idx].set_visible(False)

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



1.7.2 6.2 Numerical Features by Churn Status

[14]:
```python
# Compare numerical features between churned and non-churned customers
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, n_rows * 4))
axes = axes.flatten() if n_features > 1 else [axes]

for idx, feature in enumerate(numerical_features):
    # Box plot by churn status
    sns.boxplot(data=df, x='Churn', y=feature, ax=axes[idx])
    axes[idx].set_title(f'{feature} by Churn Status', fontweight='bold')
    axes[idx].set_xlabel('Churn')
    axes[idx].set_ylabel(feature)

# Hide extra subplots
for idx in range(n_features, len(axes)):
    axes[idx].set_visible(False)

plt.tight_layout()
plt.show()

print("  Look for features where churned and non-churned customers show␣
↪different distributions.")
print("   These features will be important for our classification model.")
```




```
Look for features where churned and non-churned customers show different
```

distributions.

These features will be important for our classification model.

1.7.3 6.3 Categorical Features

[15]:
```
# Examine categorical features
print(f"Analyzing {len(categorical_features)} categorical features...\n")

for feature in categorical_features:
    print(f"\n{feature}:")
    print("-" * 50)
    value_counts = df[feature].value_counts()
    print(value_counts)
    print(f"Unique values: {df[feature].nunique()}")
```

```
Analyzing 16 categorical features…


gender:
--------------------------------------------------
gender
Male      3555
Female    3488
Name: count, dtype: int64
Unique values: 2

Partner:
--------------------------------------------------
Partner
No     3641
Yes    3402
Name: count, dtype: int64
Unique values: 2

Dependents:
--------------------------------------------------
Dependents
No     4933
Yes    2110
Name: count, dtype: int64
Unique values: 2

PhoneService:
--------------------------------------------------
PhoneService
Yes    6361
No      682
Name: count, dtype: int64
```

```
Unique values: 2

MultipleLines:
--------------------------------------------------
MultipleLines
No                  3390
Yes                 2971
No phone service     682
Name: count, dtype: int64
Unique values: 3

InternetService:
--------------------------------------------------
InternetService
Fiber optic    3096
DSL            2421
No             1526
Name: count, dtype: int64
Unique values: 3

OnlineSecurity:
--------------------------------------------------
OnlineSecurity
No                     3498
Yes                    2019
No internet service    1526
Name: count, dtype: int64
Unique values: 3

OnlineBackup:
--------------------------------------------------
OnlineBackup
No                     3088
Yes                    2429
No internet service    1526
Name: count, dtype: int64
Unique values: 3

DeviceProtection:
--------------------------------------------------
DeviceProtection
No                     3095
Yes                    2422
No internet service    1526
Name: count, dtype: int64
Unique values: 3

TechSupport:
```

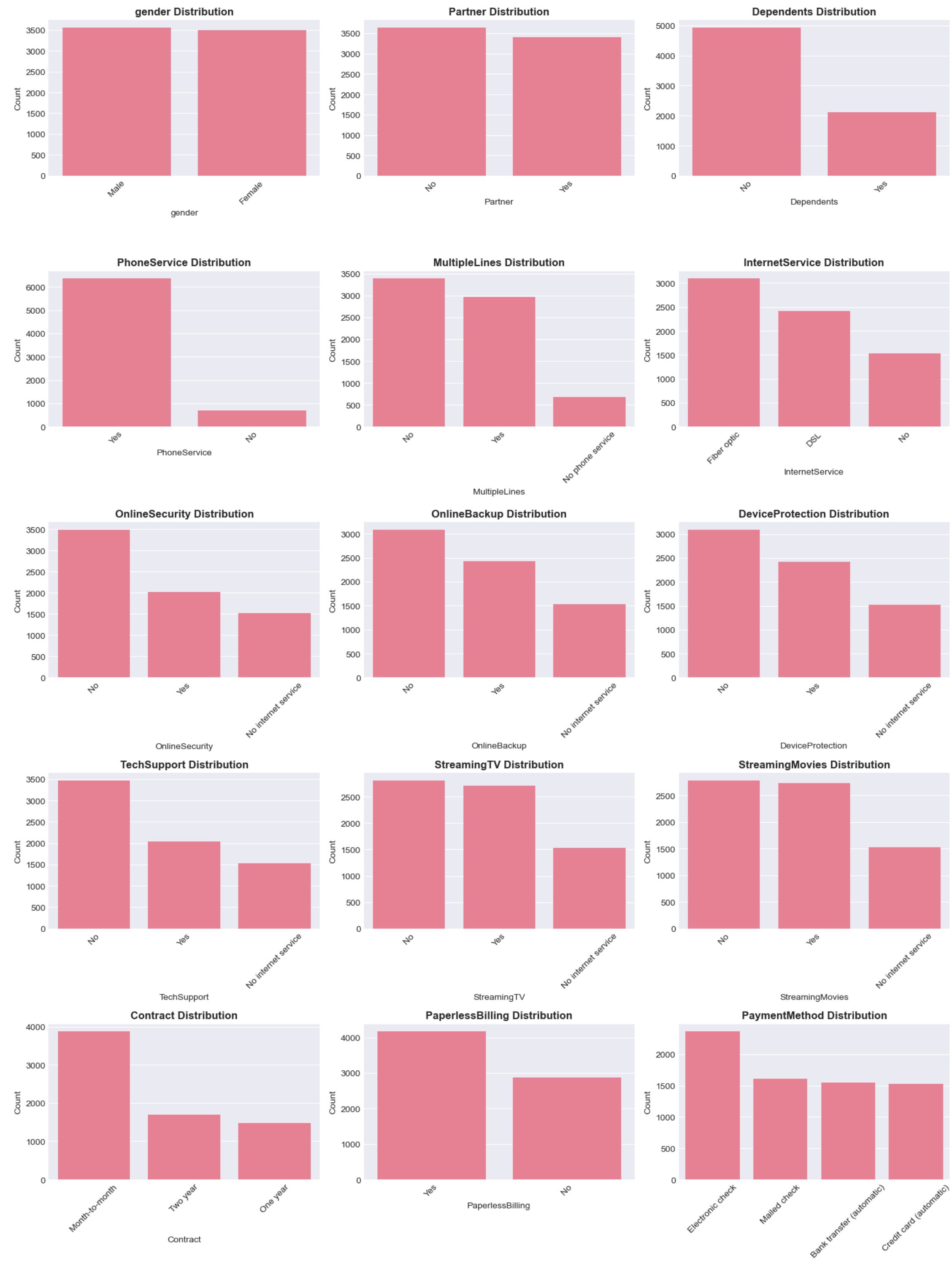
```
--------------------------------------------------
TechSupport
No                     3473
Yes                    2044
No internet service    1526
Name: count, dtype: int64
Unique values: 3

StreamingTV:
--------------------------------------------------
StreamingTV
No                     2810
Yes                    2707
No internet service    1526
Name: count, dtype: int64
Unique values: 3

StreamingMovies:
--------------------------------------------------
StreamingMovies
No                     2785
Yes                    2732
No internet service    1526
Name: count, dtype: int64
Unique values: 3

Contract:
--------------------------------------------------
Contract
Month-to-month    3875
Two year          1695
One year          1473
Name: count, dtype: int64
Unique values: 3

PaperlessBilling:
--------------------------------------------------
PaperlessBilling
Yes    4171
No     2872
Name: count, dtype: int64
Unique values: 2

PaymentMethod:
--------------------------------------------------
PaymentMethod
Electronic check             2365
Mailed check                 1612
```

```
Bank transfer (automatic)    1544
Credit card (automatic)      1522
Name: count, dtype: int64
Unique values: 4

TotalCharges:
--------------------------------------------------
TotalCharges
          11
20.2      11
19.75      9
20.05      8
19.9       8
          ..
6849.4     1
692.35     1
130.15     1
3211.9     1
6844.5     1
Name: count, Length: 6531, dtype: int64
Unique values: 6531
```

[16]:

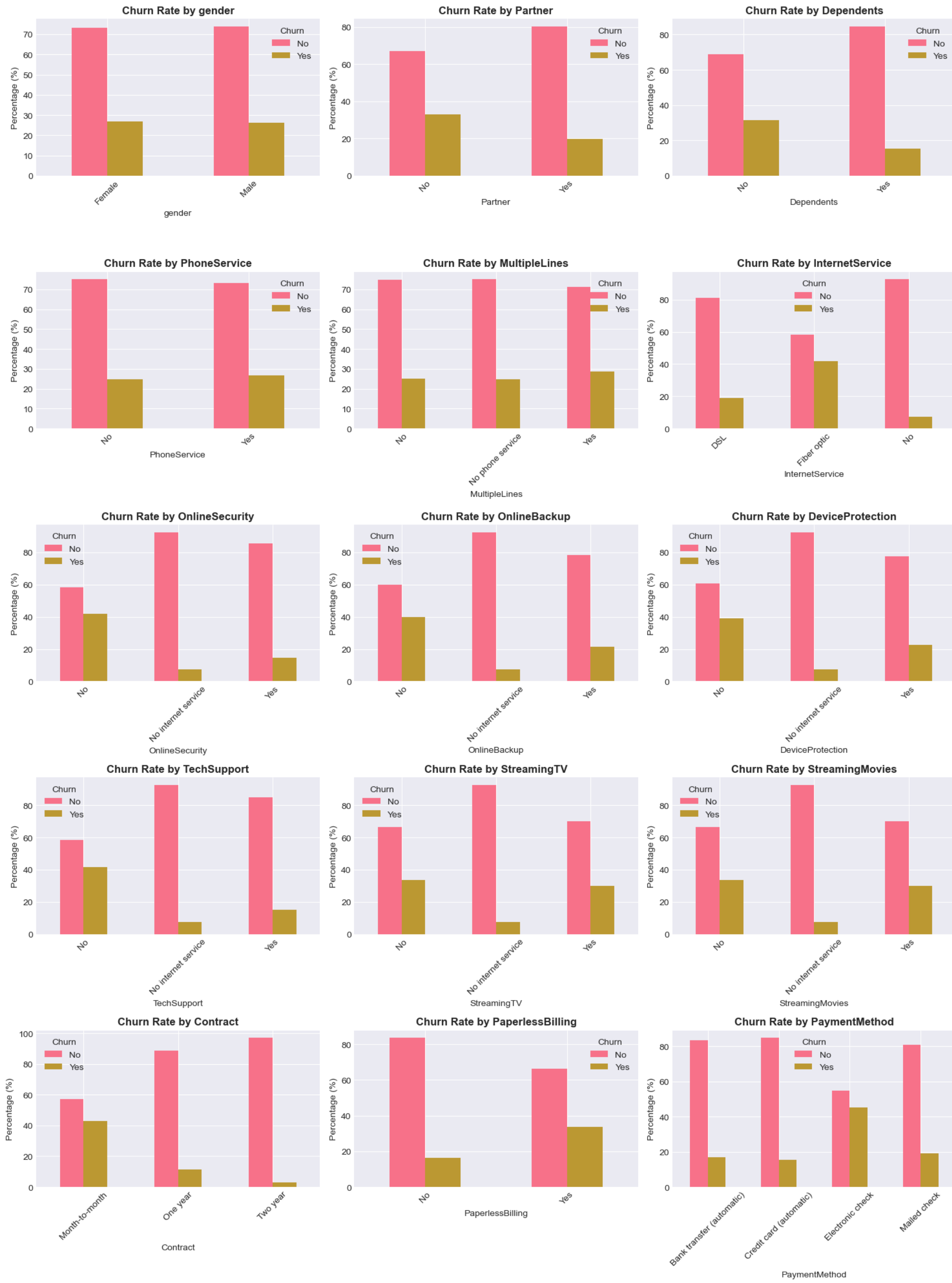
```python
# Visualize categorical features (top features with fewer unique values)
# Select features with reasonable number of categories for visualization
viz_features = [f for f in categorical_features if df[f].nunique() <= 10]

if len(viz_features) > 0:
    n_viz = len(viz_features)
    n_cols_cat = 3
    n_rows_cat = (n_viz + n_cols_cat - 1) // n_cols_cat

    fig, axes = plt.subplots(n_rows_cat, n_cols_cat, figsize=(15, n_rows_cat *␣
 ↪4))
    axes = axes.flatten() if n_viz > 1 else [axes]

    for idx, feature in enumerate(viz_features):
        sns.countplot(data=df, x=feature, ax=axes[idx], order=df[feature].
 ↪value_counts().index)
        axes[idx].set_title(f'{feature} Distribution', fontweight='bold')
        axes[idx].set_xlabel(feature)
        axes[idx].set_ylabel('Count')
        axes[idx].tick_params(axis='x', rotation=45)

    # Hide extra subplots
    for idx in range(n_viz, len(axes)):
        axes[idx].set_visible(False)
```

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



1.7.4 6.4 Categorical Features by Churn Status

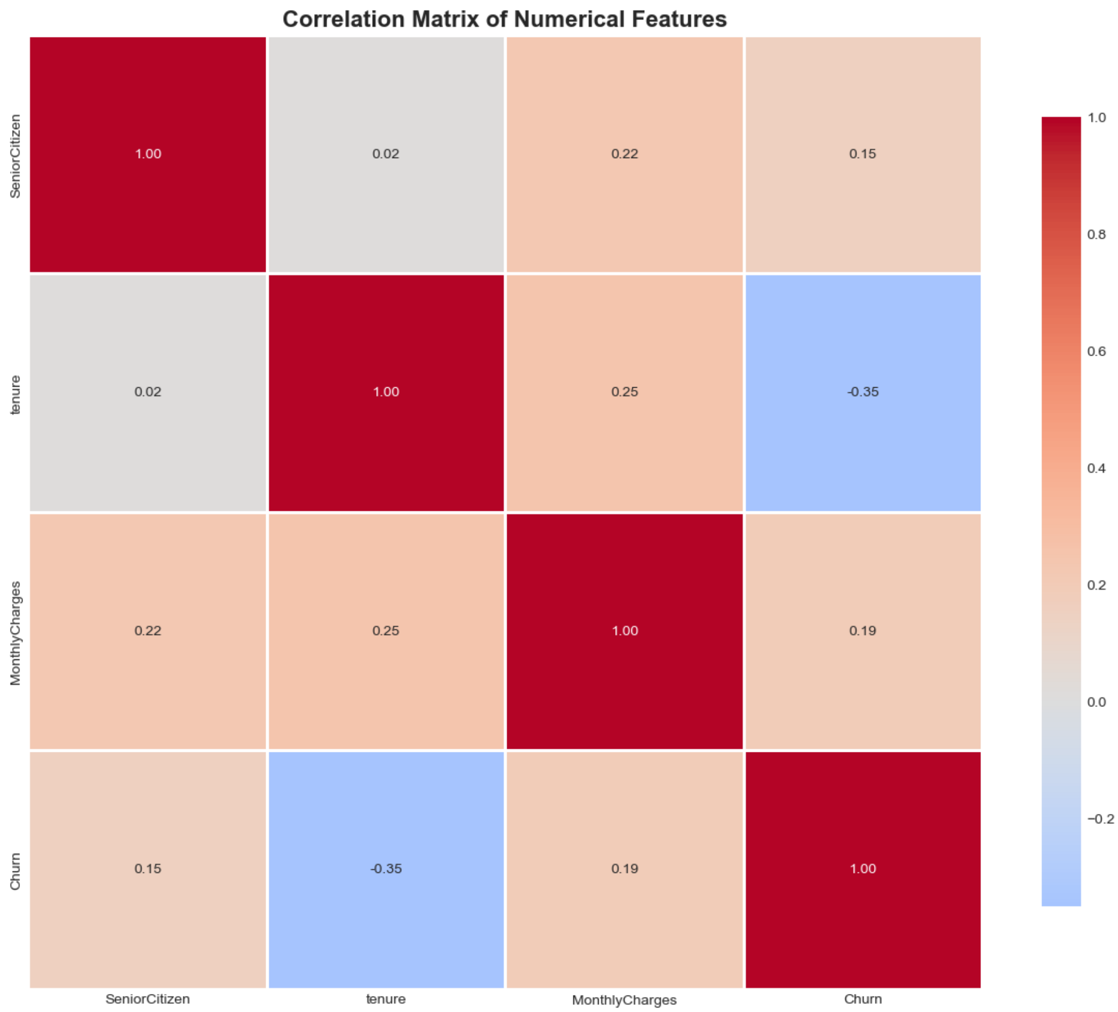
[17]:
```
# Analyze churn rate by categorical features
if len(viz_features) > 0:
    fig, axes = plt.subplots(n_rows_cat, n_cols_cat, figsize=(15, n_rows_cat *␣
 ↪4))
    axes = axes.flatten() if n_viz > 1 else [axes]

    for idx, feature in enumerate(viz_features):
        # Create a cross-tabulation
        ct = pd.crosstab(df[feature], df['Churn'], normalize='index') * 100
        ct.plot(kind='bar', ax=axes[idx], stacked=False)
        axes[idx].set_title(f'Churn Rate by {feature}', fontweight='bold')
        axes[idx].set_xlabel(feature)
        axes[idx].set_ylabel('Percentage (%)')
        axes[idx].legend(title='Churn', labels=['No', 'Yes'])
        axes[idx].tick_params(axis='x', rotation=45)

    # Hide extra subplots
    for idx in range(n_viz, len(axes)):
        axes[idx].set_visible(False)

    plt.tight_layout()
    plt.show()

print("  Look for categories with significantly higher churn rates.")
print("   These are strong predictive features for our model.")
```



Look for categories with significantly higher churn rates.
These are strong predictive features for our model.

1.8 7. Correlation Analysis

[18]:
```python
# Create a copy of the dataframe for correlation analysis
df_corr = df.copy()

# Convert binary target to numeric (if it's Yes/No)
if df_corr['Churn'].dtype == 'object':
    df_corr['Churn'] = df_corr['Churn'].map({'No': 0, 'Yes': 1})

# Select only numerical columns for correlation
numerical_df = df_corr.select_dtypes(include=['int64', 'float64'])

# Calculate correlation matrix
correlation_matrix = numerical_df.corr()

# Visualize correlation matrix
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, fmt='.2f', cmap='coolwarm',
            center=0, square=True, linewidths=1, cbar_kws={"shrink": 0.8})
plt.title('Correlation Matrix of Numerical Features', fontsize=16,␣
 ↪fontweight='bold')
plt.tight_layout()
plt.show()
```

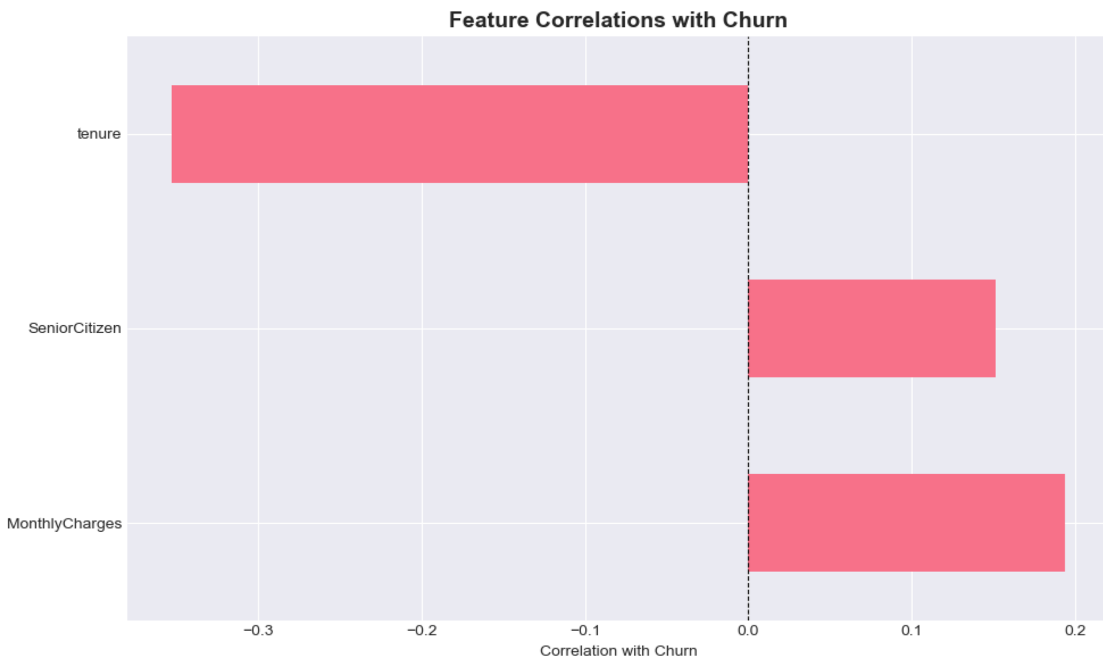


[19]:

```
# Identify features most correlated with Churn
if 'Churn' in correlation_matrix.columns:
    churn_correlations = correlation_matrix['Churn'].
 ↪sort_values(ascending=False)

    print("Features Correlation with Churn:")
    print("=" * 50)
    print(churn_correlations)

    # Visualize correlations with target
    plt.figure(figsize=(10, 6))
    churn_correlations.drop('Churn').plot(kind='barh')
    plt.xlabel('Correlation with Churn')
    plt.title('Feature Correlations with Churn', fontsize=14, fontweight='bold')
    plt.axvline(x=0, color='black', linestyle='--', linewidth=0.8)
```

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

    print("\n  Interpretation:")
    print("    - Positive correlations: Higher values → More likely to churn")
    print("    - Negative correlations: Higher values → Less likely to churn")
    print("    - Features with |correlation| > 0.1 are potentially useful␣
↪predictors")
```

```
Features Correlation with Churn:
==================================================
Churn              1.000000
MonthlyCharges     0.193356
SeniorCitizen      0.150889
tenure            -0.352229
Name: Churn, dtype: float64
```




```
  Interpretation:
    - Positive correlations: Higher values → More likely to churn
    - Negative correlations: Higher values → Less likely to churn
    - Features with |correlation| > 0.1 are potentially useful predictors
```

[20]:
```
# Identify highly correlated feature pairs (potential multicollinearity)
print("\nHighly Correlated Feature Pairs (|correlation| > 0.7):")
print("=" * 50)
```

```
# Get upper triangle of correlation matrix
upper_triangle = correlation_matrix.where(
    np.triu(np.ones(correlation_matrix.shape), k=1).astype(bool)
)

# Find features with high correlation
high_corr_pairs = []
for column in upper_triangle.columns:
    for index in upper_triangle.index:
        corr_value = upper_triangle.loc[index, column]
        if abs(corr_value) > 0.7:
            high_corr_pairs.append((index, column, corr_value))

if len(high_corr_pairs) > 0:
    for feat1, feat2, corr in high_corr_pairs:
        print(f"{feat1} <-> {feat2}: {corr:.3f}")
    print("\n  These feature pairs may cause multicollinearity.")
    print("   Consider removing one from each pair or using regularization.")
else:
    print("  No highly correlated feature pairs found.")
```

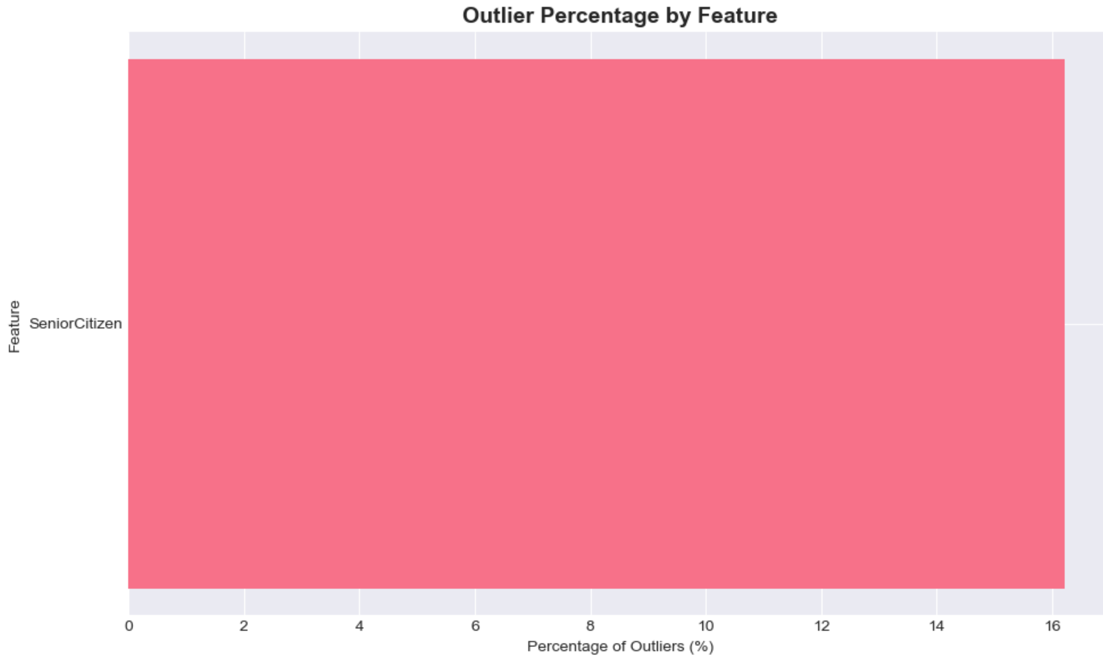
```
Highly Correlated Feature Pairs (|correlation| > 0.7):
==================================================
 No highly correlated feature pairs found.
```

1.9 8. Outlier Detection

[21]:
```
# Detect outliers using IQR method
print("Outlier Detection using IQR Method:")
print("=" * 50)

outlier_summary = []

for feature in numerical_features:
    Q1 = df[feature].quantile(0.25)
    Q3 = df[feature].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    outliers = df[(df[feature] < lower_bound) | (df[feature] > upper_bound)]
    outlier_count = len(outliers)
    outlier_percentage = (outlier_count / len(df)) * 100

    outlier_summary.append({
```

```
        'Feature': feature,
        'Outlier_Count': outlier_count,
        'Percentage': outlier_percentage,
        'Lower_Bound': lower_bound,
        'Upper_Bound': upper_bound
    })

outlier_df = pd.DataFrame(outlier_summary)
outlier_df = outlier_df[outlier_df['Outlier_Count'] > 0].
 ↪sort_values('Outlier_Count', ascending=False)

if len(outlier_df) > 0:
    print(outlier_df.to_string(index=False))

    # Visualize outliers
    plt.figure(figsize=(10, 6))
    plt.barh(outlier_df['Feature'], outlier_df['Percentage'])
    plt.xlabel('Percentage of Outliers (%)')
    plt.ylabel('Feature')
    plt.title('Outlier Percentage by Feature', fontsize=14, fontweight='bold')
    plt.tight_layout()
    plt.show()

    print("\n  Outliers detected in some features.")
    print("   Decision: Keep outliers for now (they may be legitimate values).")
    print("   Tree-based models are robust to outliers.")
else:
    print("  No significant outliers detected.")
```

```
Outlier Detection using IQR Method:
==================================================
      Feature  Outlier_Count  Percentage  Lower_Bound  Upper_Bound
SeniorCitizen           1142   16.214681          0.0          0.0
```




```
Outliers detected in some features.
 Decision: Keep outliers for now (they may be legitimate values).
 Tree-based models are robust to outliers.
```

1.10 9. Key Insights Summary

[22]:
```python
print("="*70)
print("KEY INSIGHTS FROM EXPLORATORY DATA ANALYSIS")
print("="*70)

print("\n1. DATASET OVERVIEW:")
print(f"   - Total samples: {len(df):,}")
print(f"   - Total features: {df.shape[1]}")
print(f"   - Numerical features: {len(numerical_features)}")
print(f"   - Categorical features: {len(categorical_features)}")

print("\n2. TARGET VARIABLE (CHURN):")
churn_dist = df['Churn'].value_counts(normalize=True) * 100
print(f"   - Class distribution: {churn_dist.to_dict()}")
print(f"   - Imbalance ratio: {imbalance_ratio:.2f}:1")
print(f"   -    IMBALANCED DATASET - Need special handling!")

print("\n3. DATA QUALITY:")
if len(missing_df) > 0:
    print(f"   - Missing values found in {len(missing_df)} features")
```

```
    print(f"   - Action required: Imputation or removal")
else:
    print(f"   -  No missing values")

if len(outlier_df) > 0:
    print(f"   - Outliers detected in {len(outlier_df)} features")
    print(f"   - Decision: Keep for now (tree-based models are robust)")
else:
    print(f"   -  No significant outliers")

print("\n4. FEATURE CORRELATIONS:")
if 'Churn' in correlation_matrix.columns:
    top_corr = churn_correlations.drop('Churn').abs().nlargest(3)
    print(f"   - Top 3 features correlated with churn:")
    for feat, corr in top_corr.items():
        print(f"     • {feat}: {correlation_matrix.loc[feat, 'Churn']:.3f}")

if len(high_corr_pairs) > 0:
    print(f"\n   -   {len(high_corr_pairs)} highly correlated feature pairs found")
    print(f"   - May need to address multicollinearity")

print("\n5. NEXT STEPS FOR MODELING:")
print("    Handle class imbalance (SMOTE, class weights, threshold tuning)")
print("    Encode categorical variables")
print("    Scale numerical features")
print("    Use stratified train-test split")
print("    Choose appropriate metrics (F1, ROC-AUC, NOT accuracy)")
print("    Train multiple models and compare")

print("\n" + "="*70)
print("EXPLORATION COMPLETE - Ready for Modeling Phase!")
print("="*70)
```

```
======================================================================
KEY INSIGHTS FROM EXPLORATORY DATA ANALYSIS
======================================================================

1. DATASET OVERVIEW:
   - Total samples: 7,043
   - Total features: 21
   - Numerical features: 3
   - Categorical features: 16

2. TARGET VARIABLE (CHURN):
   - Class distribution: {'No': 73.4630129206304, 'Yes': 26.536987079369588}
   - Imbalance ratio: 2.77:1
   -   IMBALANCED DATASET - Need special handling!
```

```
3. DATA QUALITY:
   -   No missing values
   - Outliers detected in 1 features
   - Decision: Keep for now (tree-based models are robust)

4. FEATURE CORRELATIONS:
   - Top 3 features correlated with churn:
     • tenure: -0.352
     • MonthlyCharges: 0.193
     • SeniorCitizen: 0.151

5. NEXT STEPS FOR MODELING:
     Handle class imbalance (SMOTE, class weights, threshold tuning)
     Encode categorical variables
     Scale numerical features
     Use stratified train-test split
     Choose appropriate metrics (F1, ROC-AUC, NOT accuracy)
     Train multiple models and compare

======================================================================
EXPLORATION COMPLETE - Ready for Modeling Phase!
======================================================================
```

1.11 Conclusion

This exploration notebook has provided comprehensive insights into the Telecom Customer Churn dataset:

Key Findings: 1. **Imbalanced Dataset**: ~73% non-churn, ~27% churn - requires special handling 2. **Data Quality**: Generally clean data with minimal issues 3. **Feature Types**: Mix of numerical and categorical features 4. **Predictive Features**: Several features show correlation with churn

Recommendations for Modeling: - Use F1-score or ROC-AUC as primary metrics (NOT accuracy) - Apply imbalance handling techniques (SMOTE, class weights) - Use stratified train-test split - Consider tree-based models (Random Forest, XGBoost) which handle mixed data types well - Implement proper preprocessing pipeline

Based on the observed class imbalance and feature distributions, special care will be taken during model training using stratified splits and imbalance-handling techniques such as class weighting and SMOTE.

Next Notebook: `02_modeling.ipynb` - Build and evaluate classification models

---