```
#Data Loading
import pandas as pd

import pandas as pd

# Load the dataset
file_path = ("/content/Telco_Customer_Churn_Dataset  (3).csv")
df = pd.read_csv(file_path)

# Display first few rows
print(df.head())

# Show dataset info
print(df.info())

print(df.head())

# Show dataset info
print(df.info())
```

```
 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
None
```

```
#Data Exploration
print(df.isnull().sum())

# Show basic statistics of numerical columns
print(df.describe())

# Check unique values in categorical columns
for col in df.select_dtypes(include=['object']).columns:
    print(f"{col}: {df[col].unique()}")
```

```
customerID          0
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
       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
customerID: ['7590-VHVEG' '5575-GNVDE' '3668-QPYBK' ... '4801-JZAZL' '8361-LTMKD'
 '3186-AJIEK']
gender: ['Female' 'Male']
Partner: ['Yes' 'No']
Dependents: ['No' 'Yes']
PhoneService: ['No' 'Yes']
MultipleLines: ['No phone service' 'No' 'Yes']
InternetService: ['DSL' 'Fiber optic' 'No']
OnlineSecurity: ['No' 'Yes' 'No internet service']
OnlineBackup: ['Yes' 'No' 'No internet service']
DeviceProtection: ['No' 'Yes' 'No internet service']
TechSupport: ['No' 'Yes' 'No internet service']
StreamingTV: ['No' 'Yes' 'No internet service']
StreamingMovies: ['No' 'Yes' 'No internet service']
Contract: ['Month-to-month' 'One year' 'Two year']
PaperlessBilling: ['Yes' 'No']
PaymentMethod: ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
TotalCharges: ['29.85' '1889.5' '108.15' ... '346.45' '306.6' '6844.5']
```

```
Churn: ['No' 'Yes']
```

```
#Handling Missing Values
# Fill missing numerical values with the median
df.fillna(df.median(numeric_only=True), inplace=True)

# Fill missing categorical values with the mode
for col in df.select_dtypes(include=['object']).columns:
    df[col].fillna(df[col].mode()[0], inplace=True)

# Verify if there are any missing values left
print(df.isnull().sum().sum())  # Should return 0
```

```
0
<ipython-input-9-c503906bbcaf>:7: FutureWarning: A value is trying to be set on a copy of a DataFrame c
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate ob

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inpla


  df[col].fillna(df[col].mode()[0], inplace=True)
```

```
df.dropna(inplace=True)  # Removes rows with missing values
```

```
#Data Preprocessing
df["TotalCharges"] = pd.to_numeric(df["TotalCharges"], errors="coerce")
df["TotalCharges"].fillna(df["TotalCharges"].median(), inplace=True)  # Handle conversion errors
```

```
<ipython-input-11-ad1009e77f02>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame 
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate ob

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inpla


  df["TotalCharges"].fillna(df["TotalCharges"].median(), inplace=True)  # Handle conversion errors
```

```
df.drop(columns=["customerID"], inplace=True)
```

```
#Categorical Variable Encoding
from sklearn.preprocessing import LabelEncoder

binary_cols = ["gender", "Partner", "Dependents", "PhoneService", "PaperlessBilling", "Churn"]
le = LabelEncoder()

for col in binary_cols:
    df[col] = le.fit_transform(df[col])
```

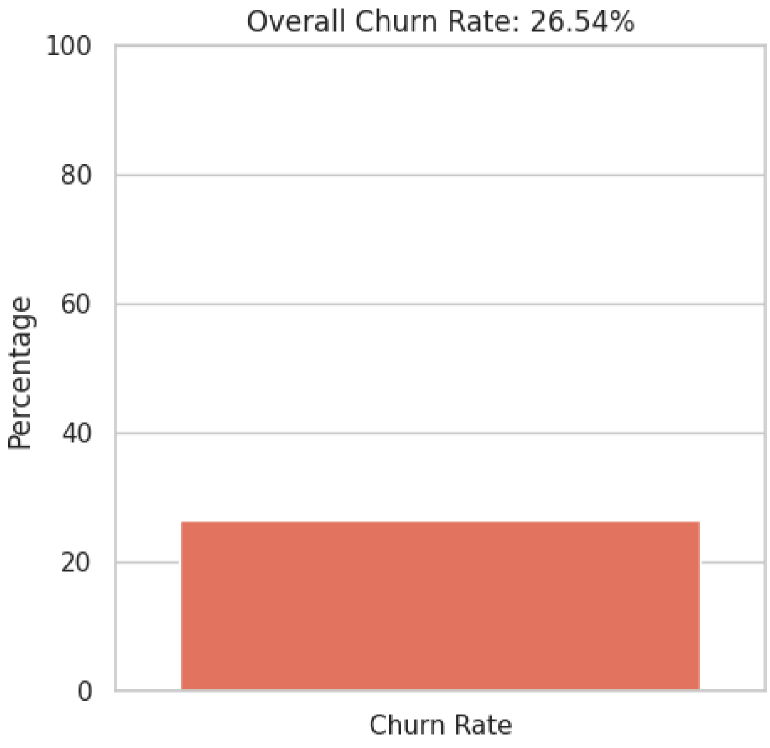
```
df = pd.get_dummies(df, columns=["Contract", "PaymentMethod", "InternetService"], drop_first=True)
```

```
from sklearn.model_selection import train_test_split
#Dataset Splitting
# Define features and target variable
X = df.drop(columns=["Churn"])  # Features
y = df["Churn"]  # Target (label)

# Split dataset (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Check shapes of resulting sets
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(5634, 23) (1409, 23) (5634,) (1409,)
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

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

# Calculate overall churn rate
churn_rate = df["Churn"].mean() * 100

# Churn Rate Visualization
plt.figure(figsize=(5, 5))
sns.barplot(x=["Churn Rate"], y=[churn_rate], palette="Reds")
plt.ylabel("Percentage")
plt.title(f"Overall Churn Rate: {churn_rate:.2f}%")
plt.ylim(0, 100)
plt.show()
```

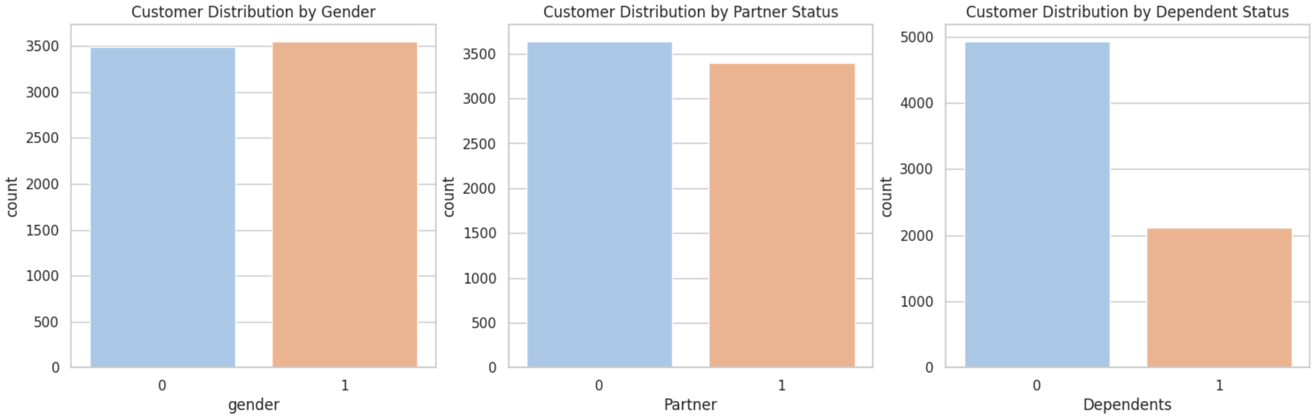
```
<ipython-input-16-addec350fba4>:13: FutureWarning:

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

  sns.barplot(x=["Churn Rate"], y=[churn_rate], palette="Reds")
```



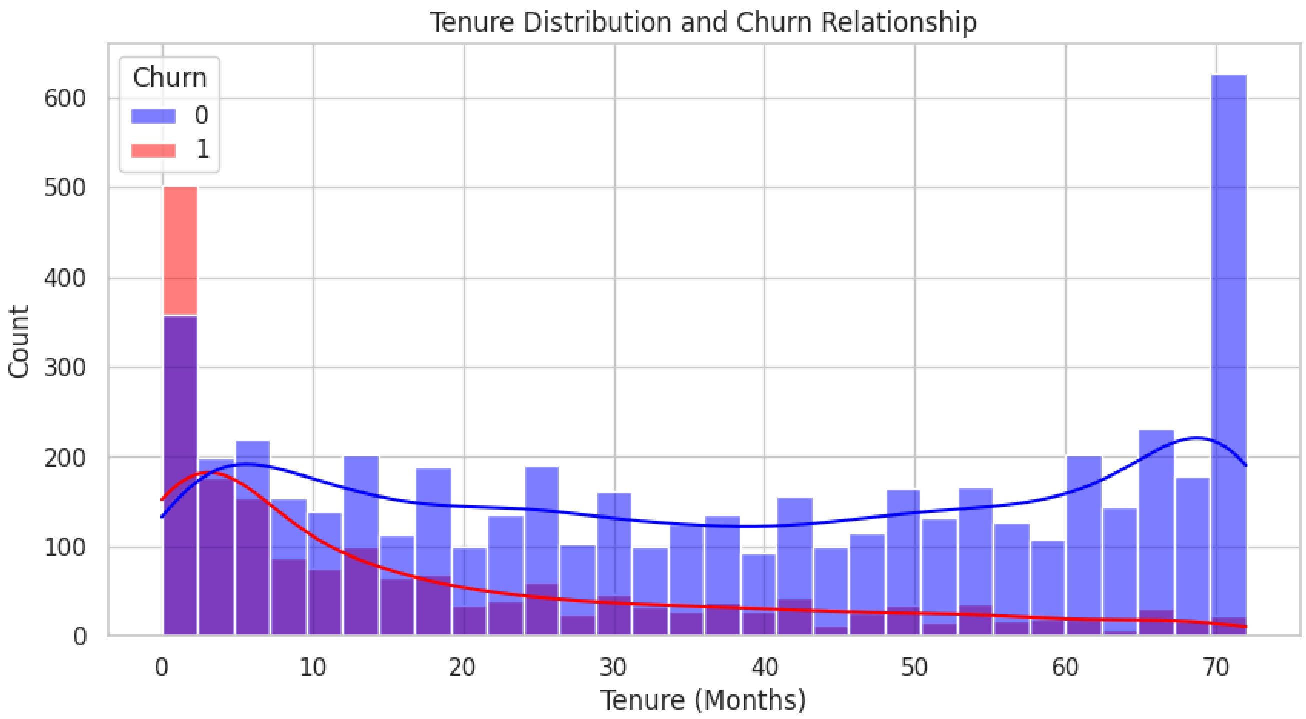
```
#Customer Distribution by Gender, Partner Status, and Dependent Status
fig, axes = plt.subplots(1, 3, figsize=(18, 5))
```

```
# Gender distribution
sns.countplot(x="gender", data=df, palette="pastel", ax=axes[0])
axes[0].set_title("Customer Distribution by Gender")

# Partner status distribution
sns.countplot(x="Partner", data=df, palette="pastel", ax=axes[1])
axes[1].set_title("Customer Distribution by Partner Status")

# Dependent status distribution
sns.countplot(x="Dependents", data=df, palette="pastel", ax=axes[2])
axes[2].set_title("Customer Distribution by Dependent Status")

plt.show()
```

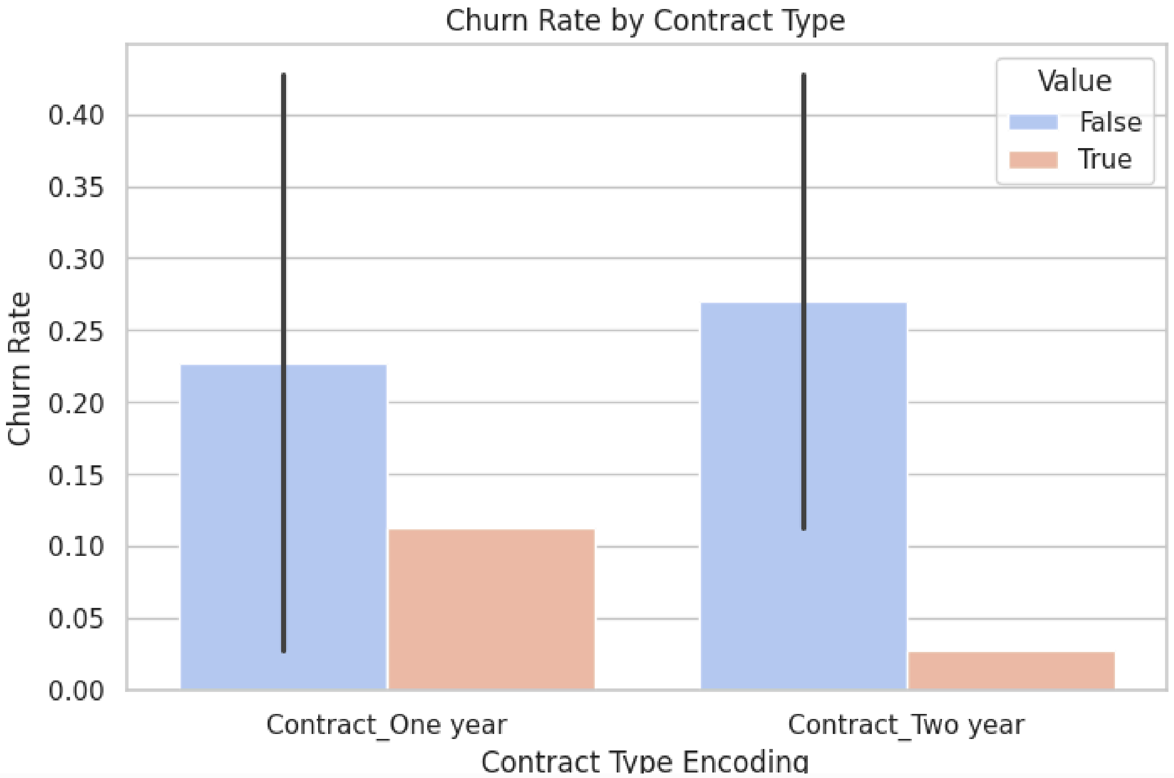
```
<ipython-input-17-d8c818680ccd>:5: FutureWarning:

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

  sns.countplot(x="gender", data=df, palette="pastel", ax=axes[0])
<ipython-input-17-d8c818680ccd>:9: FutureWarning:

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

  sns.countplot(x="Partner", data=df, palette="pastel", ax=axes[1])
<ipython-input-17-d8c818680ccd>:13: FutureWarning:

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

  sns.countplot(x="Dependents", data=df, palette="pastel", ax=axes[2])
```



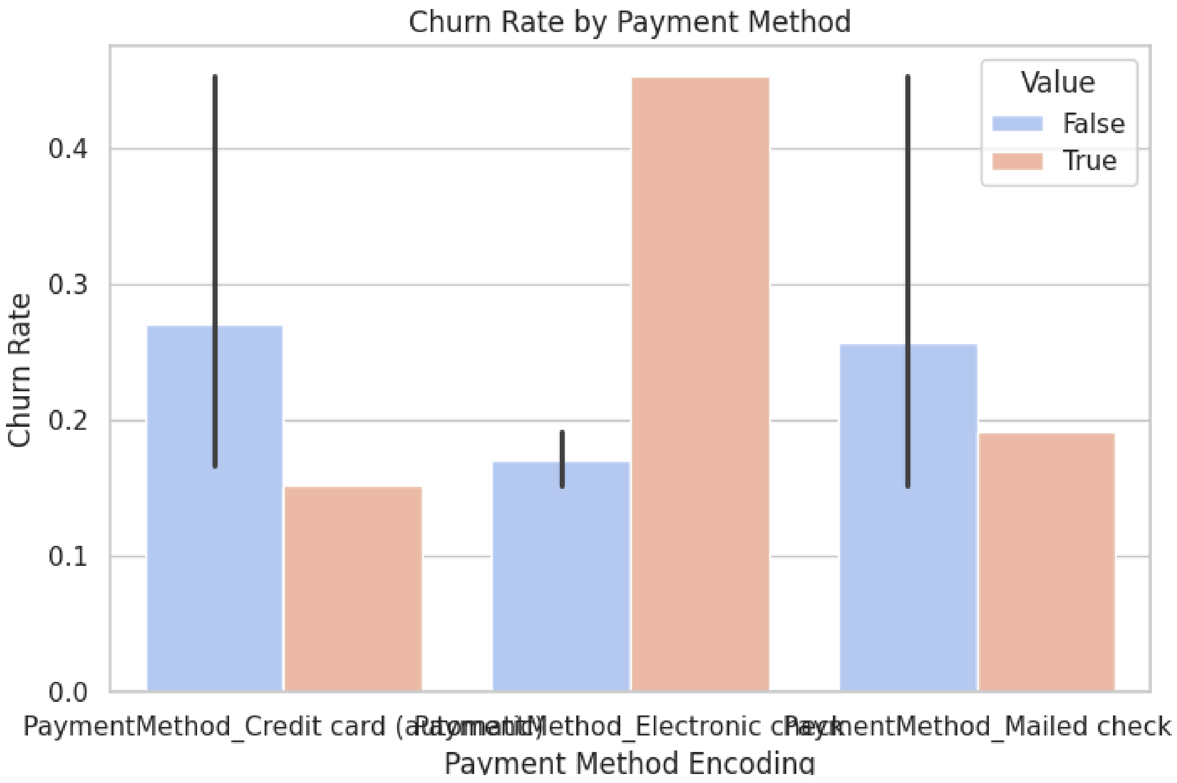
```
#Tenure Distribution and Churn Relationship
plt.figure(figsize=(10, 5))
sns.histplot(df, x="tenure", hue="Churn", kde=True, bins=30, palette=["blue", "red"])
plt.title("Tenure Distribution and Churn Relationship")
plt.xlabel("Tenure (Months)")
plt.ylabel("Count")
plt.show()
```



```
#Churn Rate by Contract Type
plt.figure(figsize=(8, 5))
contract_churn = df.groupby(["Contract_One year", "Contract_Two year"])["Churn"].mean().reset_index()
contract_churn_melted = contract_churn.melt(id_vars="Churn", var_name="Contract Type", value_name="Value")
sns.barplot(x="Contract Type", y="Churn", data=contract_churn_melted, hue="Value", palette="coolwarm")
plt.title("Churn Rate by Contract Type")
plt.xlabel("Contract Type Encoding")
plt.ylabel("Churn Rate")
plt.show()
```

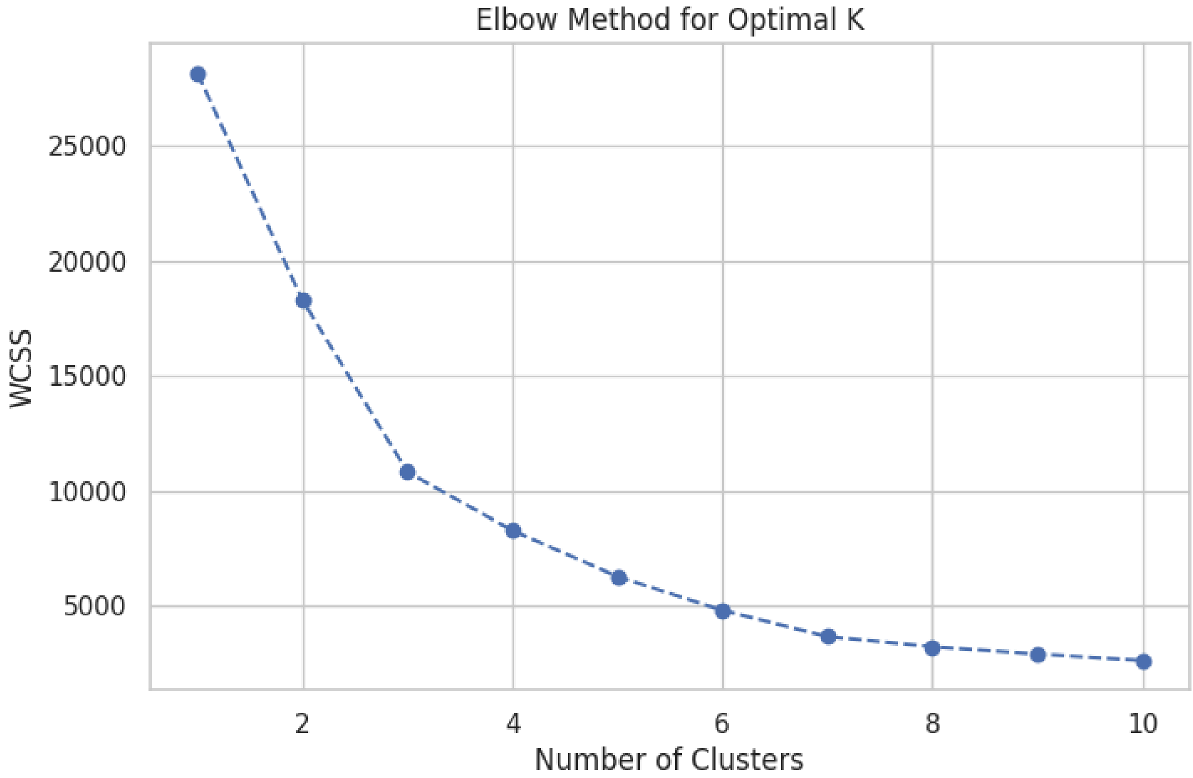


```
#Churn Rate by Payment Method
plt.figure(figsize=(8, 5))
payment_churn = df.groupby(["PaymentMethod_Credit card (automatic)",
                            "PaymentMethod_Electronic check",
                            "PaymentMethod_Mailed check"])["Churn"].mean().reset_index()
payment_churn_melted = payment_churn.melt(id_vars="Churn", var_name="Payment Method", value_name="Value")
sns.barplot(x="Payment Method", y="Churn", data=payment_churn_melted, hue="Value", palette="coolwarm")
plt.title("Churn Rate by Payment Method")
plt.xlabel("Payment Method Encoding")
plt.ylabel("Churn Rate")
plt.show()
```

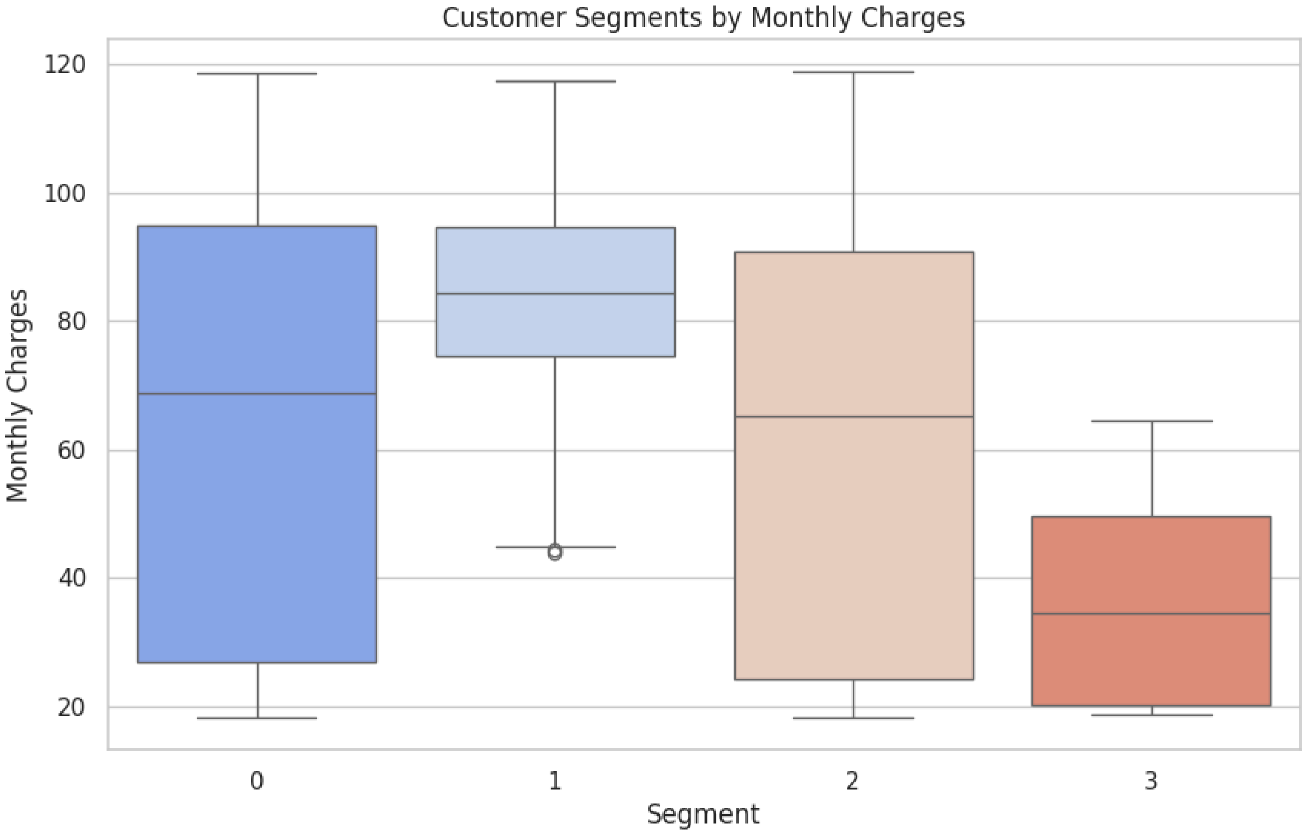


```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Select relevant features
features = ["tenure", "MonthlyCharges", "Contract_One year", "Contract_Two year"]
df_segmentation = df[features].copy()

# Normalize the features for clustering
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df_segmentation)

# Determine the optimal number of clusters using the Elbow Method
wcss = []  # Within-cluster sum of squares
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, random_state=42, n_init=10)
    kmeans.fit(df_scaled)
    wcss.append(kmeans.inertia_)

# Plot Elbow Method
plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), wcss, marker="o", linestyle="--")
plt.xlabel("Number of Clusters")
```

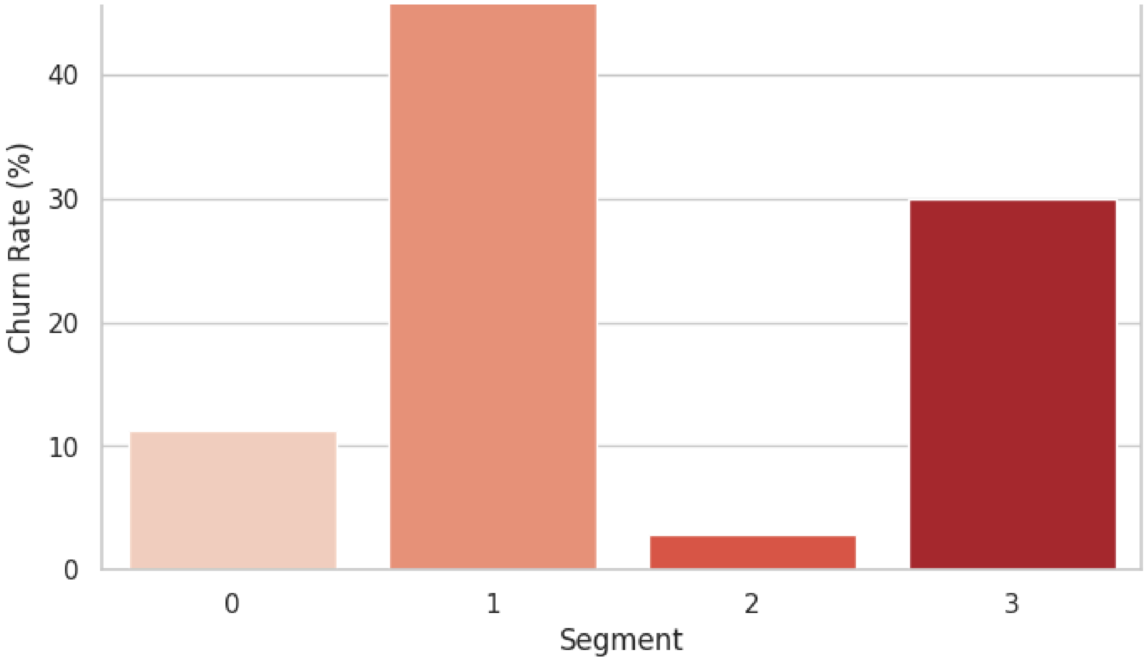
```
    plt.ylabel("WCSS")
    plt.title("Elbow Method for Optimal K")
    plt.show()

    # Choose optimal clusters (e.g., 4 based on elbow point)
    optimal_k = 4
    kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
    df["Segment"] = kmeans.fit_predict(df_scaled)

    # Analyze segments
    plt.figure(figsize=(10, 6))
    sns.boxplot(x="Segment", y="MonthlyCharges", data=df, palette="coolwarm")
    plt.title("Customer Segments by Monthly Charges")
    plt.xlabel("Segment")
    plt.ylabel("Monthly Charges")
    plt.show()

    # Analyze churn rate within each segment
    segment_churn = df.groupby("Segment")["Churn"].mean() * 100

    plt.figure(figsize=(8, 5))
    sns.barplot(x=segment_churn.index, y=segment_churn.values, palette="Reds")
    plt.title("Churn Rate by Customer Segment")
    plt.xlabel("Segment")
    plt.ylabel("Churn Rate (%)")
    plt.show()

    # Identify high-value customers at risk of churning
    high_value_customers = df[(df["MonthlyCharges"] > df["MonthlyCharges"].quantile(0.75)) & (df["Churn"] == 1)
    print("High-Value Customers at Risk of Churn:")
    print(high_value_customers[["tenure", "MonthlyCharges", "Contract_One year", "Contract_Two year", "Segment"
```



```
<ipython-input-21-bd14d74abbe2>:38: FutureWarning:

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

  sns.boxplot(x="Segment", y="MonthlyCharges", data=df, palette="coolwarm")
```



```
<ipython-input-21-bd14d74abbe2>:48: FutureWarning:

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

  sns.barplot(x=segment_churn.index, y=segment_churn.values, palette="Reds")
```



```
High-Value Customers at Risk of Churn:
      tenure  MonthlyCharges  Contract_One year  Contract_Two year  Segment
5          8           99.65              False              False        1
8         28          104.80              False              False        1
13        49          103.70              False              False        1
26        47           99.35              False              False        1
38        34          106.35              False              False        1
...      ...             ...                ...                ...      ...
6972      56          111.95               True              False        0
6986      30           94.10              False              False        1
6991       8           95.65              False              False        1
7006      40          104.50              False              False        1
7034      67          102.95              False              False        1

[578 rows x 5 columns]
```

```
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, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report

# Load dataset
df = pd.read_csv("/content/Telco_Customer_Churn_Dataset  (3).csv")

# Drop irrelevant columns (e.g., customer ID if present)
df.drop(["customerID"], axis=1, errors='ignore', inplace=True)

# Convert categorical variables into numerical values using Label Encoding
for col in df.select_dtypes(include=['object']).columns:
```