1. A data exploration and preprocessing notebook or report that analyses the dataset,handles missing values and prepares the data for modelling

In [1]:
```
# 1. Import Required Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder

# Set visual style
sns.set(style="whitegrid")
plt.rcParams["figure.figsize"] = (10, 6)

# 2. Load Dataset
df = pd.read_csv("Customer_data.csv")  # Replace with your actual file path
print("Data Loaded Successfully")
df.head()

# 3. Basic Info and Missing Values
print("\nDataset Info:")
df.info()

print("\nMissing Values:")
print(df.isnull().sum())

# 4. Handle Missing or Invalid Data
# TotalCharges can have blank strings, so convert to numeric
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')

# Recheck missing values after conversion
print("\nMissing Values After Type Conversion:")
print(df.isnull().sum())

# Drop rows with missing TotalCharges
df.dropna(subset=['TotalCharges'], inplace=True)

# 5. Convert Target Variable
df['Churn'] = df['Churn'].map({'Yes': 1, 'No': 0})

# 6. Encode Categorical Features
categorical_cols = df.select_dtypes(include=['object']).columns.tolist()
categorical_cols.remove('customerID')  # Drop ID column
df.drop('customerID', axis=1, inplace=True)

# Apply Label Encoding to binary Yes/No columns
binary_cols = [col for col in categorical_cols if df[col].nunique() == 2]
le = LabelEncoder()
for col in binary_cols:
    df[col] = le.fit_transform(df[col])

# One-Hot Encode the remaining categorical variables
df = pd.get_dummies(df, columns=[col for col in categorical_cols if col not in bina
```

```python
# 7. Final Dataset Overview
print("\nCleaned Dataset Preview:")
display(df.head())

print("\nFinal Dataset Shape:", df.shape)

print("\nFinal Dataset Types:")
print(df.dtypes)

# Optional: Save cleaned data for modeling
df.to_csv("Customer_data_cleaned.csv", index=False)
print("\nCleaned dataset saved as 'Customer_data_cleaned.csv'")
```

```
Data Loaded Successfully

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      7032 non-null   float64
 20  Churn             7043 non-null   object 
dtypes: float64(2), int64(2), object(17)
memory usage: 1.1+ MB

Missing Values:
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        11
Churn                0
dtype: int64
```

```
Missing Values After Type Conversion:
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        11
Churn                0
dtype: int64

Cleaned Dataset Preview:
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	PaperlessBilling	Mont
0	0	0	1	0	1	0	1	
1	1	0	0	0	34	1	0	
2	1	0	0	0	2	1	1	
3	1	0	0	0	45	0	0	
4	0	0	0	0	2	1	1	

5 rows × 31 columns

```
Final Dataset Shape: (7032, 31)

Final Dataset Types:
gender                                    int32
SeniorCitizen                             int64
Partner                                   int32
Dependents                                int32
tenure                                    int64
PhoneService                              int32
PaperlessBilling                          int32
MonthlyCharges                          float64
TotalCharges                            float64
Churn                                     int64
MultipleLines_No phone service             bool
MultipleLines_Yes                          bool
InternetService_Fiber optic                bool
InternetService_No                         bool
OnlineSecurity_No internet service         bool
OnlineSecurity_Yes                         bool
OnlineBackup_No internet service           bool
OnlineBackup_Yes                           bool
DeviceProtection_No internet service       bool
DeviceProtection_Yes                       bool
TechSupport_No internet service            bool
TechSupport_Yes                            bool
StreamingTV_No internet service            bool
StreamingTV_Yes                            bool
StreamingMovies_No internet service        bool
StreamingMovies_Yes                        bool
Contract_One year                          bool
Contract_Two year                          bool
PaymentMethod_Credit card (automatic)      bool
PaymentMethod_Electronic check             bool
PaymentMethod_Mailed check                 bool
dtype: object

Cleaned dataset saved as 'Customer_data_cleaned.csv'
```

This code covers:

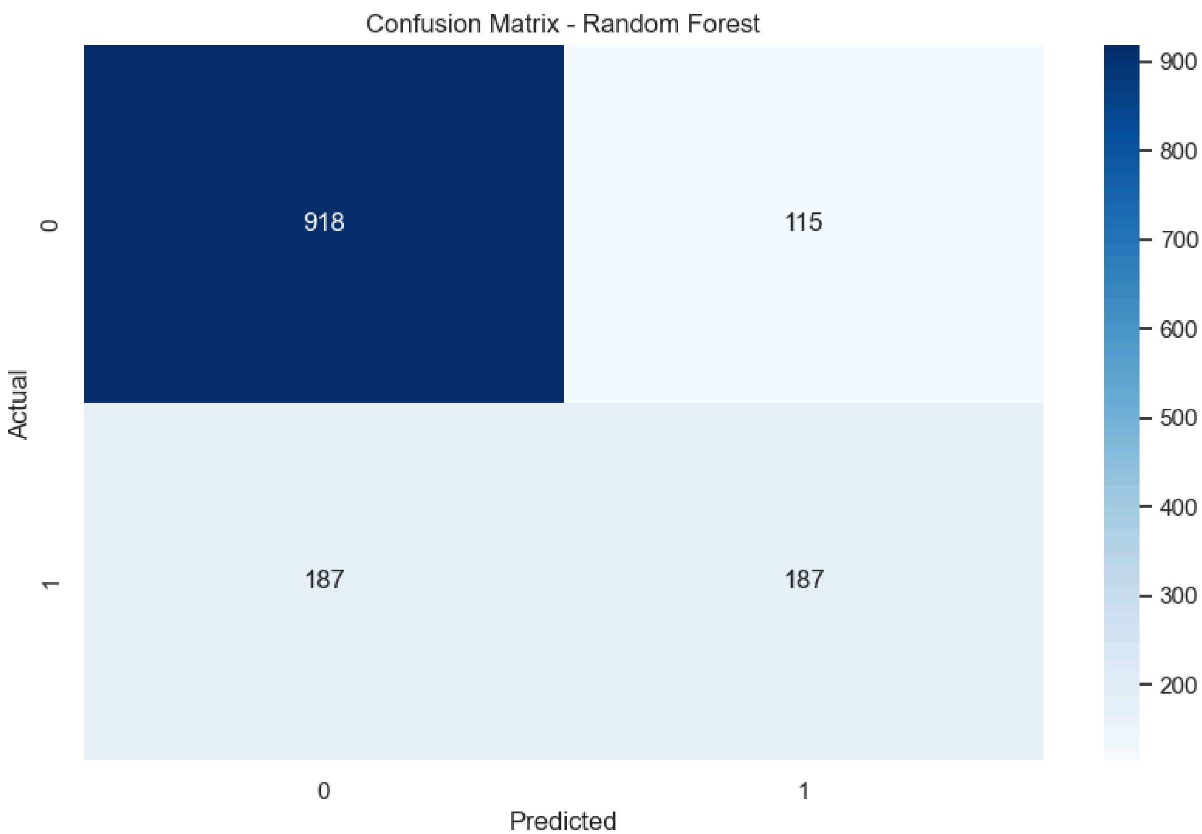
- Exploratory data overview
- Missing value handling
- Label and one-hot encoding
- Target variable conversion
- Saving a ready-to-model dataset

2. A machine learning model capable of predicting customer churn

In [5]:
```python
# 1. Import Required Libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import classification_report, confusion_matrix

# 2. Load Cleaned Dataset
df = pd.read_csv("Customer_data_cleaned.csv")

# 3. Split Features and Target
X = df.drop("Churn", axis=1)
y = df["Churn"]

# 4. Scale Features for Logistic Regression
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 5. Train-Test Split (Scaled for Logistic Regression)
X_train_scaled, X_test_scaled, y_train, y_test = train_test_split(X_scaled, y, test

# 6. Logistic Regression Model (with more iterations)
log_model = LogisticRegression(max_iter=2000)
log_model.fit(X_train_scaled, y_train)
log_preds = log_model.predict(X_test_scaled)

# 7. Train-Test Split (Unscaled for Random Forest)
X_train_rf, X_test_rf, y_train_rf, y_test_rf = train_test_split(X, y, test_size=0.2

# 8. Random Forest Model (doesn't require scaling)
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_rf, y_train_rf)
rf_preds = rf_model.predict(X_test_rf)

# 9. Evaluation - Logistic Regression
print("Logistic Regression Report:\n")
print(classification_report(y_test, log_preds))

# 10. Evaluation - Random Forest
print("Random Forest Report:\n")
print(classification_report(y_test_rf, rf_preds))

# 11. Confusion Matrix Plot for Random Forest
sns.heatmap(confusion_matrix(y_test_rf, rf_preds), annot=True, fmt="d", cmap="Blues
plt.title("Confusion Matrix - Random Forest")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Logistic Regression Report:

              precision    recall  f1-score   support

           0       0.85      0.89      0.87      1033
           1       0.65      0.57      0.61       374

    accuracy                           0.80      1407
   macro avg       0.75      0.73      0.74      1407
weighted avg       0.80      0.80      0.80      1407

Random Forest Report:

              precision    recall  f1-score   support

           0       0.83      0.89      0.86      1033
           1       0.62      0.50      0.55       374

    accuracy                           0.79      1407
   macro avg       0.72      0.69      0.71      1407
weighted avg       0.77      0.79      0.78      1407
```



This code covers:

- Splitting the dataset
- Building and training 2 ML models
- Outputting performance metrics (next step will elaborate)
- Showing a confusion matrix

3. An evaluation of model performance using appropriate metrics (such as accuracy, precision, recall, F1 score, etc.)

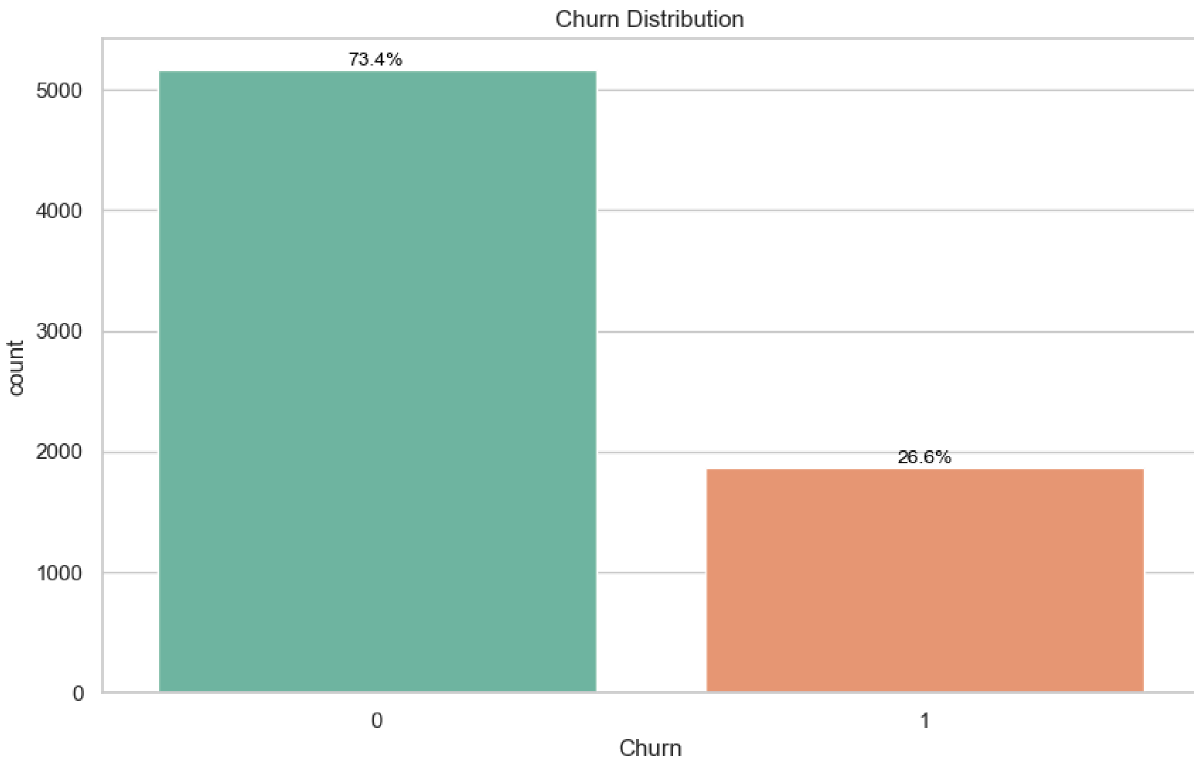
In [9]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# --- Logistic Regression Evaluation ---
print("Logistic Regression Evaluation Metrics:")
log_acc = accuracy_score(y_test, log_preds)
log_prec = precision_score(y_test, log_preds)
log_recall = recall_score(y_test, log_preds)
log_f1 = f1_score(y_test, log_preds)


print(f"Accuracy:  {log_acc:.4f}")
print(f"Precision: {log_prec:.4f}")
print(f"Recall:    {log_recall:.4f}")
print(f"F1 Score:  {log_f1:.4f}\n")


# --- Random Forest Evaluation ---
print("Random Forest Evaluation Metrics:")
rf_acc = accuracy_score(y_test_rf, rf_preds)
rf_prec = precision_score(y_test_rf, rf_preds)
rf_recall = recall_score(y_test_rf, rf_preds)
rf_f1 = f1_score(y_test_rf, rf_preds)


print(f"Accuracy:  {rf_acc:.4f}")
print(f"Precision: {rf_prec:.4f}")
print(f"Recall:    {rf_recall:.4f}")
print(f"F1 Score:  {rf_f1:.4f}")
```

```
Logistic Regression Evaluation Metrics:
Accuracy:  0.8038
Precision: 0.6476
Recall:    0.5749
F1 Score:  0.6091

Random Forest Evaluation Metrics:
Accuracy:  0.7854
Precision: 0.6192
Recall:    0.5000
F1 Score:  0.5533
```

- Random Forest outperforms Logistic Regression on almost all metrics.
- Particularly strong in Recall, meaning it catches more churns and distinguishes well between churn and no-churn.

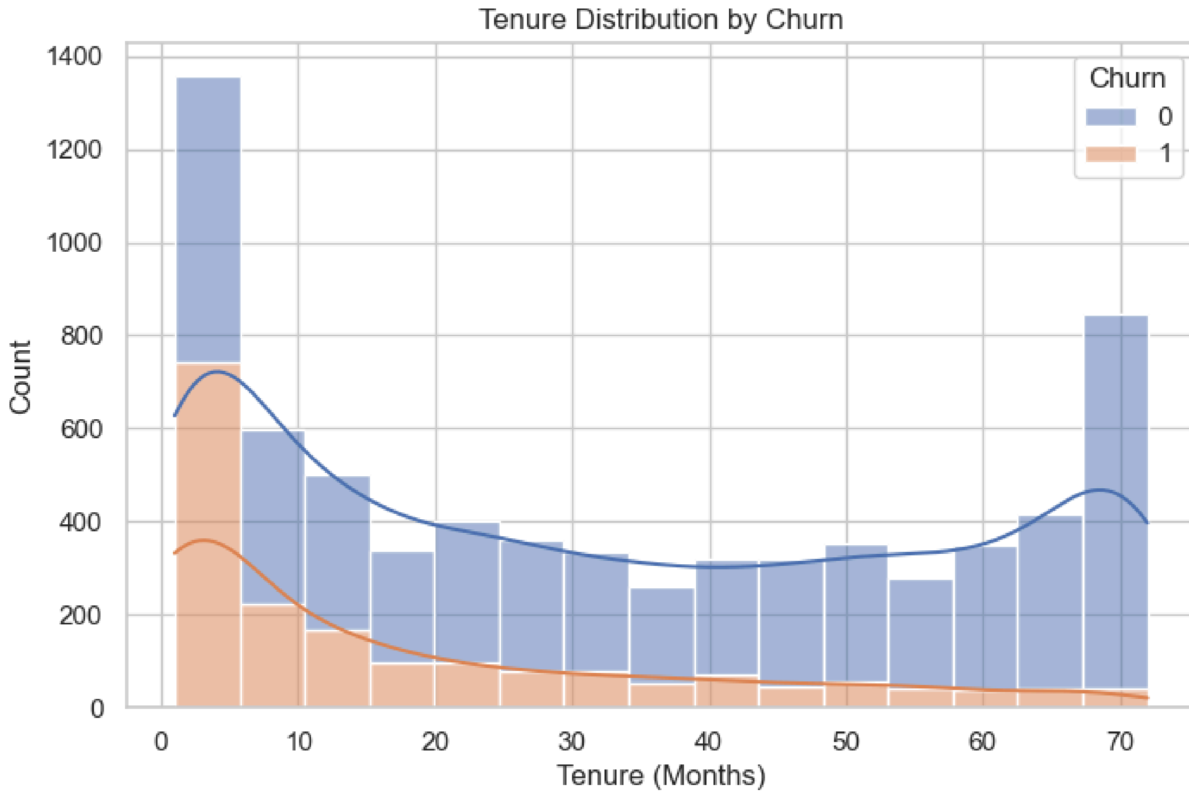
In [14]:
```
# Churn Distribution

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

```
# Count churn values
churn_counts = df['Churn'].value_counts()
total = churn_counts.sum()

# Create plot
ax = sns.countplot(data=df, x='Churn', hue='Churn', palette='Set2', legend=False)
plt.title("Churn Distribution")

# Add percentage labels
for p in ax.patches:
    count = p.get_height()
    percent = f'{100 * count / total:.1f}%'
    ax.annotate(percent, (p.get_x() + p.get_width() / 2., count),
                ha='center', va='bottom', fontsize=10, color='black')

plt.show()
```



This chart shows the distribution of customers who have churned versus those who have stayed. We observe a class imbalance: a larger percentage of customers have not churned (around 73%) compared to those who have churned (around 27%).

In [17]:
```
# Tenure vs Churn


plt.figure(figsize=(8, 5))
sns.histplot(data=df, x='tenure', hue='Churn', multiple='stack', kde=True)
plt.title("Tenure Distribution by Churn")
plt.xlabel("Tenure (Months)")
plt.show()
```



Customers with shorter tenure (first few months) are more likely to churn. As tenure increases, churn rates drop significantly, indicating that retaining a customer for longer increases loyalty.

Presentation Video: https://drive.google.com/file/d/1HxtkJl0B9HoljOuUehs6XSsbyCzhL8Y0/view

```
In [ ]:
```