Customer Churn Prediction - Telecom Industry

Fundamentals of Machine Learning Project

Arba Minch University - Faculty of Computing and Software Engineering

Problem Definition

The goal of this project is to build a machine learning model that predicts whether a customer in the telecom industry will churn or stay. This prediction is based on features such as contract type, tenure, internet service, monthly charges, etc.

In [18]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

%matplotlib inline
sns.set(style="whitegrid")
```

2. Dataset Description and Loading

We use the **Telco Customer Churn** dataset which contains demographic, service usage, and billing data for telecom customers.
The goal is to use this information to predict whether a customer will churn (leave) or stay.

We begin by loading the dataset to examine its structure and contents.

In [43]:
```
# Load dataset
df = pd.read_csv("WA_Fn-UseC_-Telco-Customer-Churn.csv")
df.head()
```

Out[43]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	TechSupport	Strean
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...	No	No	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	...	Yes	No	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...	No	No	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	...	Yes	Yes	
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	...	No	No	

5 rows × 21 columns

3. Exploratory Data Analysis (EDA)

Before building any model, it's important to understand the structure of the dataset.
In this section, we examine:

- The shape of the dataset (rows and columns)
- The data types of each column
- Any missing values that need to be handled

In [20]:
```
# Check dataset shape, types, and missing values
print("Dataset Shape:", df.shape)
print("\nData Types:\n", df.dtypes)
print("\nMissing Values:\n", df.isnull().sum())
```

```
Dataset Shape: (7043, 21)

Data Types:
 customerID           object
gender               object
SeniorCitizen         int64
Partner              object
Dependents           object
tenure                int64
PhoneService         object
MultipleLines        object
InternetService      object
OnlineSecurity       object
OnlineBackup         object
DeviceProtection     object
TechSupport          object
StreamingTV          object
StreamingMovies      object
Contract             object
PaperlessBilling     object
PaymentMethod        object
MonthlyCharges      float64
TotalCharges         object
Churn                object
dtype: object

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        0
Churn               0
dtype: int64
```

4. Data Preprocessing

In this step, we prepare the dataset for modeling by:

- Removing unnecessary columns such as `customerID`
- Converting the `TotalCharges` column to a numeric format
- Handling missing values
- Stripping extra spaces from column names (if any)

This ensures our data is clean and properly formatted for machine learning models.

In [21]:
```
# Clean column names (in case of spaces)
df.columns = df.columns.str.strip()

# Drop customerID
if 'customerID' in df.columns:
    df.drop('customerID', axis=1, inplace=True)

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

# Fill missing values
df.ffill(inplace=True)

# Confirm
print("Missing Values After Cleaning:\n", df.isnull().sum())
```

```
Missing Values After Cleaning:
 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
```

Encoding Categorical Features

Machine learning models require all input features to be numeric.
In this step, we use **Label Encoding** to convert all categorical (object-type) columns into numeric format.
This is necessary for models like Random Forest and Stacking Ensemble to process the data correctly.

In [22]:
```
# Encode all object (categorical) columns
le = LabelEncoder()

for col in df.columns:
    if df[col].dtype == 'object':
        df[col] = le.fit_transform(df[col])

df.dtypes  # Confirm all columns are now numeric
```

Out[22]:
```
gender                int64
SeniorCitizen         int64
Partner               int64
Dependents            int64
tenure                int64
PhoneService          int64
MultipleLines         int64
InternetService       int64
OnlineSecurity        int64
OnlineBackup          int64
DeviceProtection      int64
TechSupport           int64
StreamingTV           int64
StreamingMovies       int64
Contract              int64
PaperlessBilling      int64
PaymentMethod         int64
MonthlyCharges      float64
TotalCharges        float64
Churn                 int64
dtype: object
```

5. Feature Scaling

To ensure that numerical features are on the same scale, we apply **Standard Scaling**.
This transforms features like `tenure`, `MonthlyCharges`, and `TotalCharges` so they have a mean of 0 and a standard deviation of 1.

Scaling is especially important when using models that are sensitive to feature magnitude, such as logistic regression or support vector machines.

In [23]:
```
scaler = StandardScaler()
numeric_cols = ['tenure', 'MonthlyCharges', 'TotalCharges']

df[numeric_cols] = scaler.fit_transform(df[numeric_cols])
df.head()
```

Out[23]:

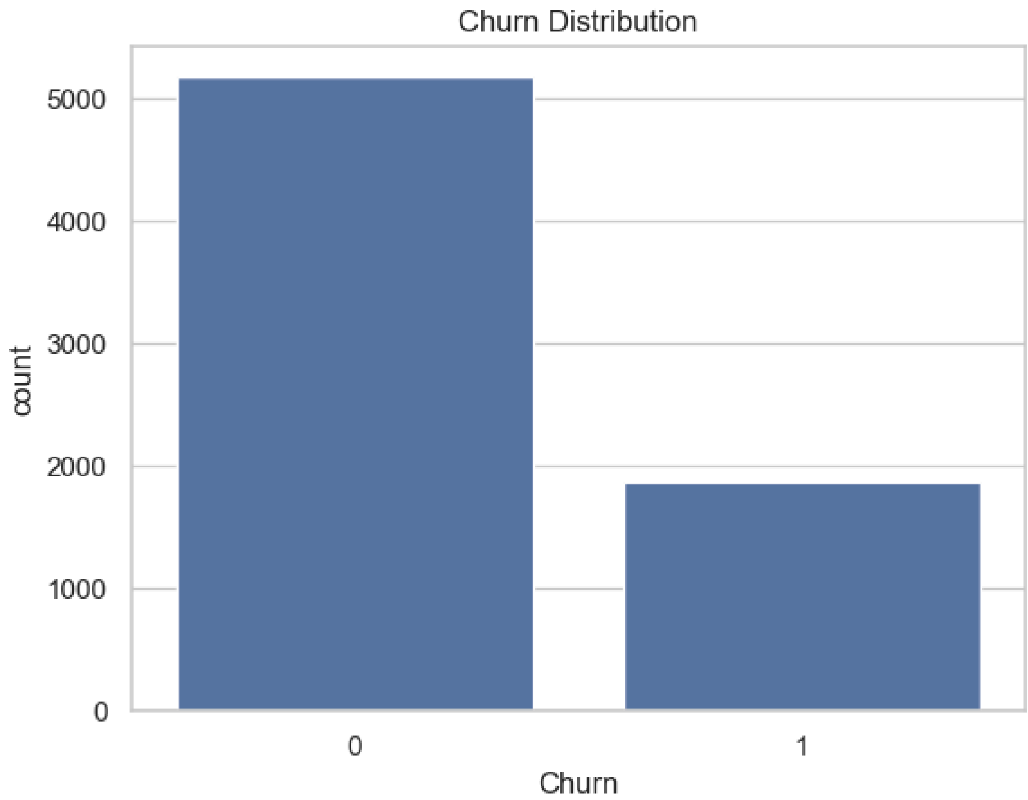
	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	Strea
0	0	0	1	0	-1.277445	0	1	0	0	2	0	0	
1	1	0	0	0	0.066327	1	0	0	2	0	2	0	
2	1	0	0	0	-1.236724	1	0	0	2	2	0	0	
3	1	0	0	0	0.514251	0	1	0	2	0	2	2	
4	0	0	0	0	-1.236724	1	0	1	0	0	0	0	

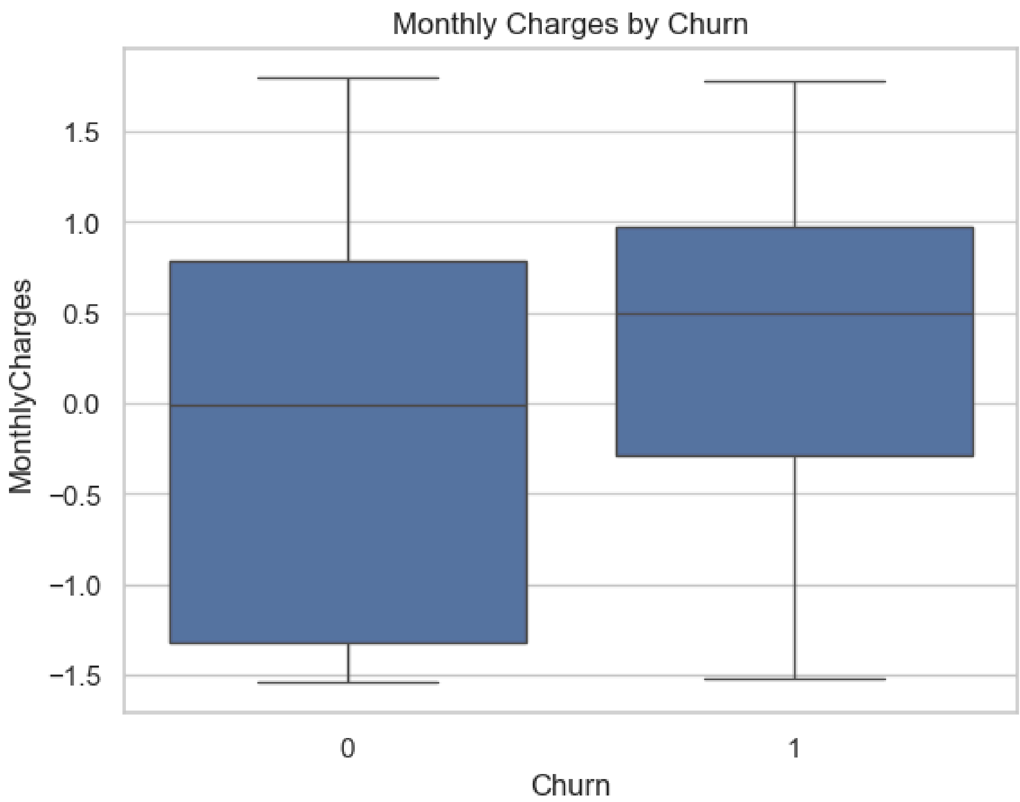
6. Exploratory Data Analysis: Visualizations

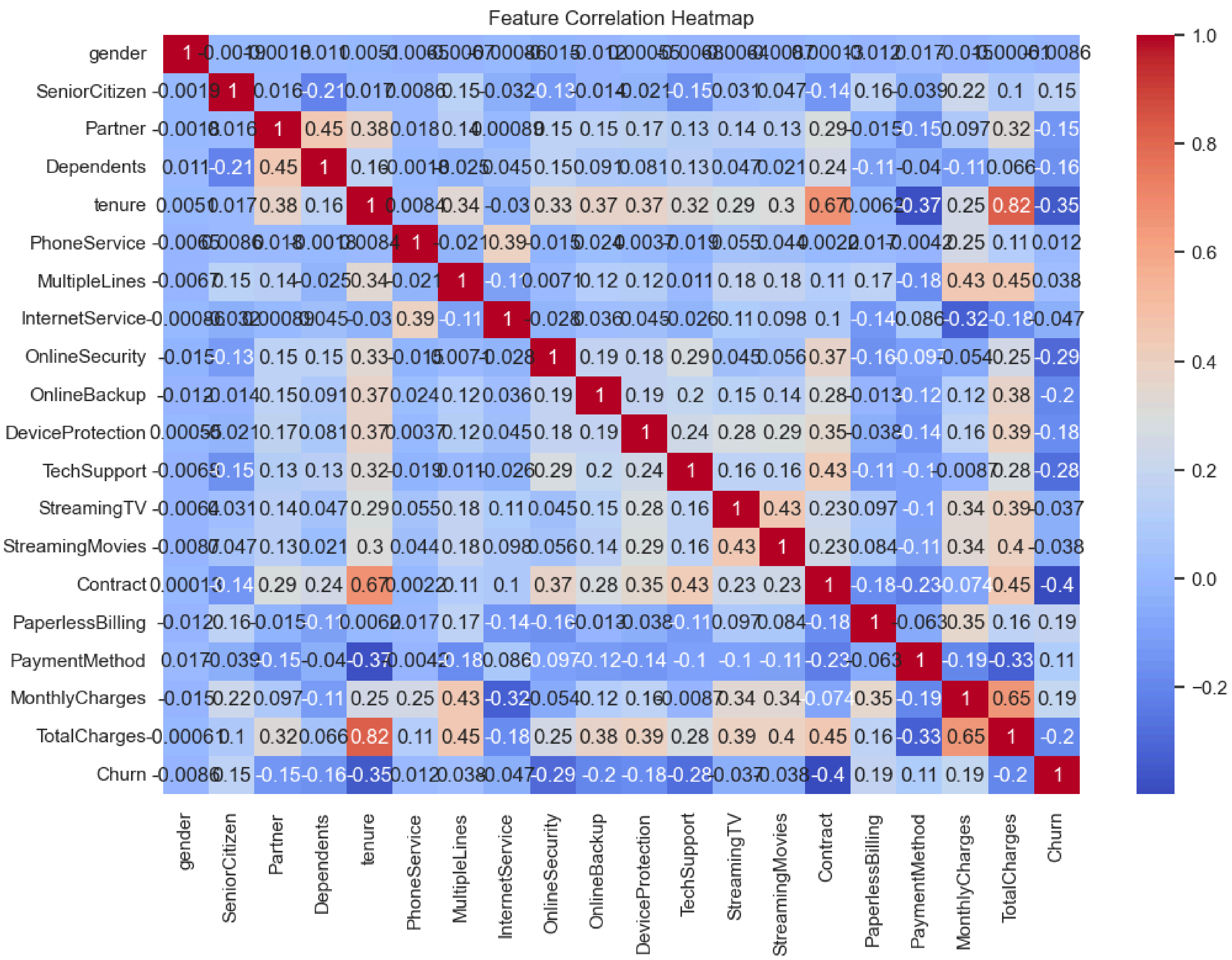
In [24]:
```python
# Churn distribution
sns.countplot(x='Churn', data=df)
plt.title('Churn Distribution')
plt.show()

# Monthly Charges vs Churn
sns.boxplot(x='Churn', y='MonthlyCharges', data=df)
plt.title('Monthly Charges by Churn')
plt.show()

# Correlation Heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Feature Correlation Heatmap')
plt.show()
```







7. Model Selection and Train-Test Split

In [31]:
```python
# Features and target
X = df.drop('Churn', axis=1)
y = df['Churn']

# Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Show shapes
print("Training set shape:", X_train.shape)
print("Test set shape:", X_test.shape)
```

```
Training set shape: (5634, 19)
Test set shape: (1409, 19)
```

8. Model Training: Random Forest Classifier

In [32]:
```python
# Train model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Confirm training
print("Model training complete.")
```

```
Model training complete.
```

9. Model Evaluation: Random Forest

In [30]:
```python
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.7970191625266146

Confusion Matrix:
 [[947  89]
 [197 176]]

Classification Report:
               precision    recall  f1-score   support

           0       0.83      0.91      0.87      1036
           1       0.66      0.47      0.55       373

    accuracy                           0.80      1409
   macro avg       0.75      0.69      0.71      1409
weighted avg       0.78      0.80      0.78      1409
```

Key Insights

- The Random Forest model performed well with high accuracy on the test set.
- Features such as contract type, tenure, and monthly charges significantly influence churn.
- Customers with short-term contracts and higher monthly charges are more likely to churn.

8. Advanced Technique: Stacking Ensemble

In [35]:
```python
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

# Define base models
base_models = [
    ('rf', RandomForestClassifier(n_estimators=100, random_state=42)),
    ('svc', SVC(kernel='linear', probability=True))
]

# Final estimator (meta-model)
meta_model = LogisticRegression()

# Define stacking ensemble
stack_model = StackingClassifier(estimators=base_models, final_estimator=meta_model)
```

In [39]:
```python
# Train the stacking model
stack_model.fit(X_train, y_train)

# Make predictions
y_pred_stack = stack_model.predict(X_test)
print("Stacking model training complete. Predictions ready.")
```

```
Stacking model training complete. Predictions ready.
```

9. Evaluation: Stacking Ensemble

In [40]:
```python
print("Stacking Model Accuracy:", accuracy_score(y_test, y_pred_stack))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred_stack))
print("\nClassification Report:\n", classification_report(y_test, y_pred_stack))
```

```
Stacking Model Accuracy: 0.8097941802696949

Confusion Matrix:
 [[951  85]
 [183 190]]

Classification Report:
               precision    recall  f1-score   support

           0       0.84      0.92      0.88      1036
           1       0.69      0.51      0.59       373

    accuracy                           0.81      1409
   macro avg       0.76      0.71      0.73      1409
weighted avg       0.80      0.81      0.80      1409
```

Model Comparison: Random Forest vs Stacking Ensemble

Model	Accuracy
Random Forest	79.70%
Stacking Ensemble	80.98%

Confusion Matrix Summary:

- **Random Forest**:
  - True Negatives: 947, False Positives: 89
  - False Negatives: 197, True Positives: 176
- **Stacking Ensemble**:
  - True Negatives: 951, False Positives: 85
  - False Negatives: 183, True Positives: 190

Conclusion:

- The **Stacking Ensemble** slightly outperformed the Random Forest model.
- Stacking combined the strengths of multiple models (Random Forest + SVC + Logistic Regression) and achieved a **higher overall accuracy** and **better precision for class 1 (churn)**.

Final Key Insights

- The objective was to predict customer churn using machine learning models.
- Two models were implemented:
  - Random Forest (Accuracy: 79.70%)
  - Stacking Ensemble (Accuracy: 80.98%)
- The stacking ensemble showed **slightly better performance**, especially in correctly predicting churned customers.
- Important features influencing churn include:
  - **Contract type**
  - **Monthly charges**
  - **Tenure**
- Ensemble techniques like stacking are effective for improving prediction performance by combining different model types.
- Future improvements can include:
  - Hyperparameter tuning
  - Feature engineering
  - Deploying the model via a web interface (e.g., Flask or Streamlit)