756 Final Project

Shravan Sundar Ravi

1 Introduction

In the competitive telecommunications industry, customer churn poses a significant challenge for service providers, impacting revenue and growth potential. This project analyzes customer churn using the Iranian Churn Dataset, which contains 3,150 records collected from an Iranian telecom company's database over 12 months. The dataset includes various attributes such as call failures, SMS frequency, customer complaints, subscription length, and customer value, offering insights into customer behavior. By applying supervised machine learning techniques, this project aims to develop a predictive model that identifies customers at risk of leaving the service. Understanding these predictive factors is crucial for enhancing customer retention strategies, optimizing marketing efforts, and ultimately fostering long-term customer loyalty.

1.1 Topic Description

This project examines customer churn in the telecommunications sector using the Iranian Churn Dataset. I am particularly interested in this topic due to the significant impact of customer retention on a company's profitability in a competitive market. The dataset features various attributes such as call failures, complaints, and subscription length, which provide valuable insights into customer behavior. By analyzing these factors, I aim to identify patterns that can inform strategies to enhance customer satisfaction and loyalty, ultimately benefiting both the telecom company and its clients.

1.2 Expectations

I expect to uncover key relationships in the dataset, such as a correlation between a higher number of complaints and call failures with increased churn rates, indicating that service quality is crucial for customer retention. Additionally, I hypothesize that longer subscription lengths will be linked to lower churn rates, suggesting that established customers are more loyal. I also anticipate that usage frequency and distinct calls made will reveal insights about customer engagement, with higher usage likely reducing churn. Overall, I aim to identify predictive factors that clarify why customers leave, offering actionable insights for improving retention strategies.

1.3 Data Description

Source: The dataset is sourced from the UCI Machine Learning Repository, specifically the Iranian Churn Dataset, Link : https://archive.ics.uci.edu/dataset/563/iranian+churn+dataset

Collection Method: This dataset was randomly collected from an Iranian telecom company's database over a period of 12 months.

Number of Cases: There are a total of 3,150 rows, each representing a customer.

1.4 Attribute Descriptions

1. **Call Failure**: The number of times calls attempted by the customer failed, which can indicate service quality issues and potentially influence customer satisfaction.
2. **Complaints**: The total number of complaints registered by the customer, serving as a direct measure of customer dissatisfaction and a potential predictor of churn.
3. **Subscription Length**: The duration of the customer's subscription in months, reflecting customer loyalty and engagement; longer subscriptions may correlate with lower churn rates.
4. **Charge Amount**: The total amount charged to the customer, which could impact their perception of value and satisfaction with the service.
5. **Seconds of Use**: The total time, measured in seconds, that the customer has utilized the service, which may correlate with customer engagement and likelihood of churn.
6. **Frequency of Use**: How often the customer uses the service, indicating their level of engagement and satisfaction.
7. **Frequency of SMS**: The number of SMS messages sent by the customer, which can be an indicator of customer engagement with the service.
8. **Distinct Called Numbers**: The number of different phone numbers the customer has called, providing insight into their social connectivity and usage patterns.
9. **Age Group**: The age category of the customer, which may influence service preferences and churn behavior.

10. **Tariff Plan**: The type of service plan the customer is subscribed to (e.g., prepaid, postpaid), which can affect pricing and churn rates.
11. **Status**: The current status of the customer (active or inactive), which is crucial for determining churn.
12. **Age**: The actual age of the customer, providing demographic insights that may correlate with churn.
13. **Customer Value**: An aggregated metric representing the overall value of the customer to the company, reflecting their potential lifetime revenue.
14. **Churn**: The target variable indicating whether the customer has churned (1) or not (0), which is the main focus of the analysis.

```
import pandas as pd
df = pd.read_csv("Customer Churn.csv")
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
import numpy as np
```

1.5 Summary Statistics

```
df.columns = df.columns.str.strip()
```

```
summary_stats = df[['Call  Failure', 'Complains', 'Subscription  Length',
                    'Charge  Amount', 'Seconds of Use', 'Frequency of use',
                    'Frequency of SMS', 'Distinct Called Numbers',
                    'Age Group', 'Tariff Plan', 'Status', 'Age',
                    'Customer Value', 'Churn']].describe()
```

```
print(summary_stats)
```

```
       Call  Failure    Complains  Subscription  Length  Charge  Amount  \
count    3150.000000  3150.000000           3150.000000     3150.000000
mean        7.627937     0.076508             32.541905        0.942857
std         7.263886     0.265851              8.573482        1.521072
min         0.000000     0.000000              3.000000        0.000000
25%         1.000000     0.000000             30.000000        0.000000
50%         6.000000     0.000000             35.000000        0.000000
```

```
75%        12.000000     0.000000             38.000000        1.000000
max        36.000000     1.000000             47.000000       10.000000

       Seconds of Use  Frequency of use  Frequency of SMS  \
count     3150.000000       3150.000000       3150.000000
mean      4472.459683         69.460635         73.174921
std       4197.908687         57.413308        112.237560
min          0.000000          0.000000          0.000000
25%       1391.250000         27.000000          6.000000
50%       2990.000000         54.000000         21.000000
75%       6478.250000         95.000000         87.000000
max      17090.000000        255.000000        522.000000

       Distinct Called Numbers    Age Group  Tariff Plan       Status  \
count              3150.000000  3150.000000  3150.000000  3150.000000
mean                 23.509841     2.826032     1.077778     1.248254
std                  17.217337     0.892555     0.267864     0.432069
min                   0.000000     1.000000     1.000000     1.000000
25%                  10.000000     2.000000     1.000000     1.000000
50%                  21.000000     3.000000     1.000000     1.000000
75%                  34.000000     3.000000     1.000000     1.000000
max                  97.000000     5.000000     2.000000     2.000000

               Age  Customer Value        Churn
count  3150.000000     3150.000000  3150.000000
mean     30.998413      470.972916     0.157143
std       8.831095      517.015433     0.363993
min      15.000000        0.000000     0.000000
25%      25.000000      113.801250     0.000000
50%      30.000000      228.480000     0.000000
75%      30.000000      788.388750     0.000000
max      55.000000     2165.280000     1.000000
```

2 Exploratory Data Analysis

Main Outcome / Target Variable

- **Target Variable**: The main target variable for prediction is **Churn** (1 for churned customers and 0 for retained).
- **Effectiveness**: This variable accurately represents the customer's status, making it suitable for predicting churn likelihood. Its binary nature allows for a straightforward

classification model, ideal for exploring factors influencing churn.

No Data Cleaning Was Necessary

2.1 New Variables Created

- **Average Monthly Charges**: Calculated by dividing **Charge Amount** by **Subscription Length** to capture monthly spend, potentially indicating value for money perception.
- **Complaint Ratio**: Derived by dividing **Complaints** by **Subscription Length** to capture complaint frequency, which may correlate with dissatisfaction.
- we have created other variables like engagement ratio and avg_sms_per_month to look into customer attention aswell which will be used in further sections

```
df['Avg Monthly Charge'] = df['Charge  Amount'] / df['Subscription  Length']

df['Complaint Ratio'] = df['Complains'] / df['Subscription  Length']

df['Engagement Ratio'] = df['Frequency of use'] / df['Subscription  Length']


df['Avg_SMS_Per_Month'] = df['Frequency of SMS'] / df['Subscription  Length']
```

```
# Split data into training and testing sets (80% train, 20% test)
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)
```

2.2 *Checking shapes*

```
X_train = train_df.drop(columns=['Churn'])
y_train = train_df['Churn']
X_test = test_df.drop(columns=['Churn'])
y_test = test_df['Churn']
X = train_df.drop(columns=['Churn'])
y = train_df['Churn']

print("Shape of input features (X):", X.shape)
print("Shape of output variable (y):", y.shape)
```

```
Shape of input features (X): (2520, 17)
Shape of output variable (y): (2520,)
```

Inference:

1. **Input Features (X):**
   - Your dataset contains **2520 samples** (rows) and **17 features** (columns).
   - These features likely represent various customer attributes or behavioral metrics relevant to churn prediction.
2. **Output Variable (y):**
   - The target variable (y) also has **2520 samples**, indicating that there is one label for each observation in the dataset.
   - This is consistent with a supervised learning problem, where the goal is to predict a target variable based on the input features.
3. **Balanced Dimensions:**
   - The number of samples in `X` and `y` matches, confirming that the dataset is properly aligned for model training.

2.3 Missing values

```
df.isnull().sum()
```

```
Call  Failure            0
Complains                0
Subscription  Length     0
Charge  Amount           0
Seconds of Use           0
Frequency of use         0
Frequency of SMS         0
Distinct Called Numbers  0
Age Group                0
Tariff Plan              0
Status                   0
Age                      0
Customer Value           0
Churn                    0
Avg Monthly Charge       0
Complaint Ratio          0
Engagement Ratio         0
```

```
Avg_SMS_Per_Month          0
dtype: int64
```

2.3 Explicitly Verify the Absence of Missing Data

This code is used to identify and quantify missing values in the training dataset (`train_df`). First, it calculates the total number of missing values in each column by applying the `.isnull()` method to flag missing values and then summing them up using `.sum()`. Next, it computes the percentage of missing values for each column by dividing the count of missing values by the total number of rows in the dataset and multiplying by 100. These values are organized into a new dataframe, `missing_report`, which includes two columns: "Missing Values" and "Percentage." The report is then sorted in descending order of missing values to highlight the columns with the most missing data. Finally, the missing data report is printed, providing a clear overview of the extent of missing values in the dataset. This analysis is essential for determining how to handle missing data, such as imputing or dropping columns, ensuring data quality for further processing.

```python
missing_values = train_df.isnull().sum()
missing_percentage = (missing_values / len(train_df)) * 100
missing_report = pd.DataFrame({
    'Missing Values': missing_values,
    'Percentage': missing_percentage
}).sort_values(by='Missing Values', ascending=False)

print("Missing Data Report:")
print(missing_report)
```

```
Missing Data Report:
                         Missing Values  Percentage
Call  Failure                         0         0.0
Complains                             0         0.0
Engagement Ratio                      0         0.0
Complaint Ratio                       0         0.0
Avg Monthly Charge                    0         0.0
Churn                                 0         0.0
Customer Value                        0         0.0
Age                                   0         0.0
Status                                0         0.0
Tariff Plan                           0         0.0
Age Group                             0         0.0
Distinct Called Numbers               0         0.0
```

```
Frequency of SMS                     0         0.0
Frequency of use                     0         0.0
Seconds of Use                       0         0.0
Charge  Amount                       0         0.0
Subscription  Length                 0         0.0
Avg_SMS_Per_Month                    0         0.0
```

2.4 Formal Outlier Analysis Across All Variables

Using the Interquartile Range (IQR) method:

```
numerical_columns = train_df.select_dtypes(include=['float64', 'int64']).columns

outliers_list = []  # Initialize a list to collect outlier counts for each feature

for column in numerical_columns:
    Q1 = train_df[column].quantile(0.25)
    Q3 = train_df[column].quantile(0.75)
    IQR = Q3 - Q1
    # Identify outliers
    outliers = train_df[(train_df[column] < (Q1 - 1.5 * IQR)) | (train_df[column] > (Q3 + 1.
    outliers_list.append({"Feature": column, "Outlier Count": len(outliers)})

# Create a DataFrame from the list
outliers_report = pd.DataFrame(outliers_list)

print("Outlier Analysis Report:")
print(outliers_report)
```

```
Outlier Analysis Report:
                   Feature  Outlier Count
0            Call  Failure             43
1                Complains            192
2     Subscription  Length            181
3           Charge  Amount            299
4           Seconds of Use            161
5         Frequency of use            103
6         Frequency of SMS            287
7  Distinct Called Numbers             65
8                Age Group            129
9              Tariff Plan            199
```

```
10                  Status           611
11                     Age           552
12          Customer Value            99
13                   Churn           385
14      Avg Monthly Charge           183
15         Complaint Ratio           192
16        Engagement Ratio           191
17       Avg_SMS_Per_Month           198
```

The outlier analysis was conducted across all numerical features in the dataset using the Interquartile Range (IQR) method to identify potential outliers. This method calculates the range between the first quartile (Q1) and third quartile (Q3) and flags any values below Q1−1.5×IQR or above Q3+1.5×IQR as outliers. The results indicate that the feature **`Status`** has the highest number of outliers (611), followed by **`Age`** (552) and **`Churn`** (385). Features such as **`Charge  Amount`** (299) and **`Frequency of SMS`** (287) also exhibit a significant number of outliers, suggesting high variability or the presence of extreme values. In contrast, features like **`Call Failure`** (43) and **`Distinct Called Numbers`** (65) have relatively fewer outliers. These findings are crucial for preprocessing, as the identified outliers may need to be addressed through methods such as capping, transformation, or exclusion to improve model performance and ensure robust predictions.

2.5 Detailed Descriptive Statistics for Key Variables

Focus on key predictors like `Complains`, `Subscription Length`, etc.:

```
# Detailed descriptive statistics for key variables
key_features = ['Complains', 'Subscription  Length', 'Charge  Amount', 'Frequency of use', '(
detailed_statistics = train_df[key_features].describe().T
detailed_statistics['Range'] = detailed_statistics['max'] - detailed_statistics['min']

print("Detailed Descriptive Statistics:")
print(detailed_statistics)
```

```
Detailed Descriptive Statistics:
                       count       mean        std  min    25%   50%   75%  \
Complains             2520.0   0.076190   0.265355  0.0   0.00   0.0   0.0
Subscription  Length  2520.0  32.510714   8.581758  3.0  29.00  35.0  38.0
Charge  Amount        2520.0   0.953175   1.540053  0.0   0.00   0.0   1.0
Frequency of use      2520.0  70.052778  57.612744  0.0  27.75  54.0  96.0
Churn                 2520.0   0.152778   0.359845  0.0   0.00   0.0   0.0
```

```
                          max  Range
Complains                 1.0    1.0
Subscription  Length     47.0   44.0
Charge  Amount           10.0   10.0
Frequency of use        255.0  255.0
Churn                     1.0    1.0
```

2.6 Multicollinearity Check Using Variance Inflation Factor (VIF)

```python
from statsmodels.stats.outliers_influence import variance_inflation_factor


X = train_df.select_dtypes(include=['float64', 'int64']).drop(columns=['Churn'])


vif_data = pd.DataFrame()
vif_data["Feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

print("Multicollinearity Check (VIF):")
print(vif_data.sort_values(by="VIF", ascending=False))
```

```
Multicollinearity Check (VIF):
                    Feature         VIF
11                      Age  198.020188
8                 Age Group  165.183432
12           Customer Value  108.128732
5          Frequency of use   80.110586
6          Frequency of SMS   65.134788
4            Seconds of Use   50.743386
2      Subscription  Length   31.462769
9               Tariff Plan   21.340219
10                   Status   16.647647
15         Engagement Ratio   16.385726
3            Charge  Amount   12.927043
13       Avg Monthly Charge    9.030049
7   Distinct Called Numbers    7.107974
0             Call  Failure    6.153520
16         Avg_SMS_Per_Month    4.085956
1                 Complains    2.642327
14          Complaint Ratio    2.422438
```

The Variance Inflation Factor (VIF) analysis was conducted to assess multicollinearity among numerical features in the dataset, excluding the target variable `Churn`. VIF values measure how much the variance of a regression coefficient is inflated due to multicollinearity with other predictors. Generally, a VIF value above 10 indicates high multicollinearity, which could negatively impact model performance and interpretability.

The analysis revealed that `Age` (VIF = 198.02) and `Age Group` (VIF = 165.18) exhibit extremely high multicollinearity, suggesting a strong correlation between these features. Similarly, `Customer Value` (VIF = 108.13) and `Frequency of Use` (VIF = 80.11) also have high VIF values, indicating potential redundancy in the data. Other features such as `Frequency of SMS` (VIF = 65.13), `Seconds of Use` (VIF = 50.74), and `Subscription Length` (VIF = 31.46) also show significant multicollinearity, while features like `Complaint Ratio` (VIF = 2.42) and `Complains` (VIF = 2.64) have low VIF values, indicating minimal multicollinearity.

These findings suggest that some features, particularly those with very high VIF values, may need to be removed or transformed to reduce redundancy and improve model stability. For instance, highly correlated features such as `Age` and `Age Group` could be combined or one of them could be dropped. Addressing multicollinearity is essential for ensuring reliable coefficient estimates and improving the interpretability of the model.

2.7 Outlier Handling Without Removal

Apply transformations to mitigate the effect of outliers:

```
train_df['Log_Avg_Monthly_Charge'] = np.log1p(train_df['Avg Monthly Charge'])
train_df['Log_Charge_Amount'] = np.log1p(train_df['Charge  Amount'])
train_df['Log_Frequency_of_use'] = np.log1p(train_df['Frequency of use'])



def cap_outliers(column):
    lower_bound = train_df[column].quantile(0.01)
    upper_bound = train_df[column].quantile(0.99)
    train_df[column] = np.clip(train_df[column], lower_bound, upper_bound)

outlier_columns = ['Frequency of SMS', 'Seconds of Use', 'Customer Value']
for col in outlier_columns:
    cap_outliers(col)
```

This code handles outliers without removing any data by employing two strategies: **logarithmic transformations** and **clipping outliers to a defined range**.

1. **Logarithmic Transformations:**

- Logarithmic transformations are applied to features with positively skewed distributions, reducing the effect of extreme outlier values while retaining all observations.
- The `np.log1p()` function is used, which applies the natural logarithm to 1+x, ensuring it handles zero values gracefully.
- In this case, the following features are transformed:
  - `Avg Monthly Charge` becomes `Log_Avg_Monthly_Charge`.
  - `Charge Amount` becomes `Log_Charge_Amount`.
  - `Frequency of use` becomes `Log_Frequency_of_use`.

This transformation helps normalize these features, making them less influenced by extreme values.

2. **Outlier Capping:**
   - For selected features (`Frequency of SMS`, `Seconds of Use`, `Customer Value`), outliers are capped by defining a lower bound at the 1st percentile and an upper bound at the 99th percentile.
   - The `np.clip()` function adjusts values below the 1st percentile to the lower bound and those above the 99th percentile to the upper bound, effectively reducing the influence of extreme outliers while retaining the data points.
   - This approach preserves the integrity of the dataset while ensuring that extreme values do not distort the model.

2.8 Document Multicollinearity Without Removing Variables

Keep all variables but document VIF values for later analysis during modeling.

```python
from statsmodels.stats.outliers_influence import variance_inflation_factor

X = train_df.select_dtypes(include=['float64', 'int64']).drop(columns=['Churn'])
vif_data = pd.DataFrame({
    'Feature': X.columns,
    'VIF': [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
})

print("Multicollinearity Report (VIF):")
print(vif_data.sort_values(by="VIF", ascending=False))
```

```
Multicollinearity Report (VIF):
                    Feature          VIF
17   Log_Avg_Monthly_Charge  4100.738115
13       Avg Monthly Charge  3653.983194
11                      Age   199.169868
8                 Age Group   168.721651
12           Customer Value   116.476833
5          Frequency of use    87.229903
6          Frequency of SMS    70.055733
18        Log_Charge_Amount    61.182365
4            Seconds of Use    56.027854
2      Subscription  Length    44.248760
19     Log_Frequency_of_use    32.507794
3            Charge  Amount    26.066050
9               Tariff Plan    23.672723
10                   Status    19.488617
15         Engagement Ratio    16.718037
7   Distinct Called Numbers     7.702207
0             Call  Failure     6.356941
16        Avg_SMS_Per_Month     4.193369
1                 Complains     2.687015
14          Complaint Ratio     2.467201
```

The code generates a multicollinearity report by calculating the Variance Inflation Factor (VIF) for all numerical features in the dataset, excluding the target variable `Churn`. VIF quantifies how much the variance of a feature is inflated due to its correlation with other features. A high VIF value (commonly above 10) suggests significant multicollinearity, indicating that the variable is highly redundant.

The report reveals extreme VIF values for certain variables, such as `Log_Avg_Monthly_Charge` (4100.74) and `Avg Monthly Charge` (3653.98), which are likely highly correlated due to the logarithmic transformation. Similarly, `Age`(199.17) and `Age Group` (168.72) also exhibit high VIF values, indicating strong overlap between these features. Other variables such as `Customer Value` (116.48), `Frequency of Use` (87.23), and `Seconds of Use` (56.03) also show multicollinearity, though to a lesser degree. Features like `Complaint Ratio` (2.47) and `Complains` (2.69) have low VIF values, suggesting minimal correlation with other variables.

Key Points:

- Multicollinearity is documented without removing variables, preserving all features for further analysis.

- This approach allows flexibility to address multicollinearity later during model selection or feature engineering.
- Features with extremely high VIF values (e.g., `Log_Avg_Monthly_Charge`, `Avg Monthly Charge`) will likely require special attention, such as dimensionality reduction (e.g., PCA) or combining redundant features.

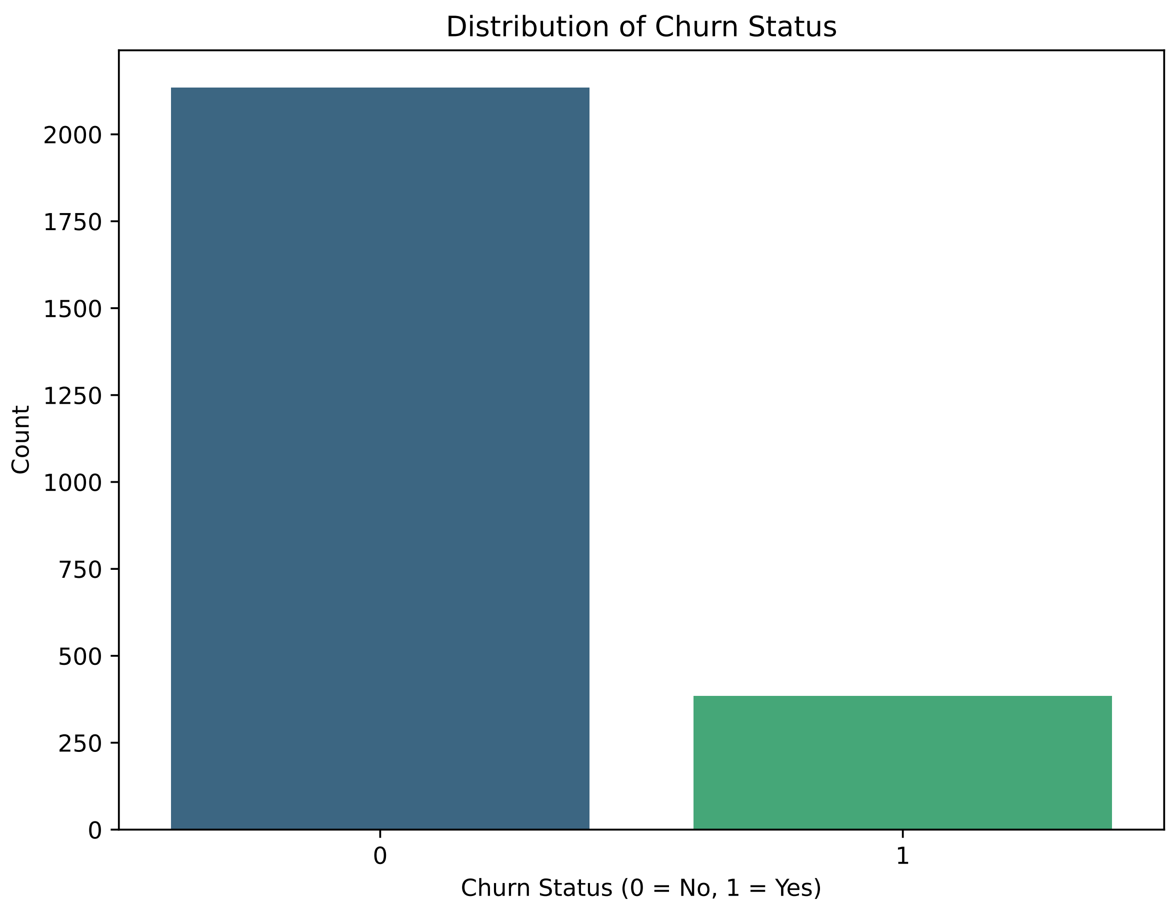
By documenting multicollinearity, this report serves as a reference for deciding which features might need adjustments during model optimization to avoid issues such as unstable coefficients and reduced interpretability.

2.10 Data Visualizations

Visualization 1: Distribution of Churn Status

- **Type**: Bar Plot
- **Insight**: Visualizing churn vs. non-churn percentages will provide a baseline understanding of class distribution. If there's an imbalance (e.g., significantly more non-churn than churn cases), this may impact model choice and evaluation strategies.

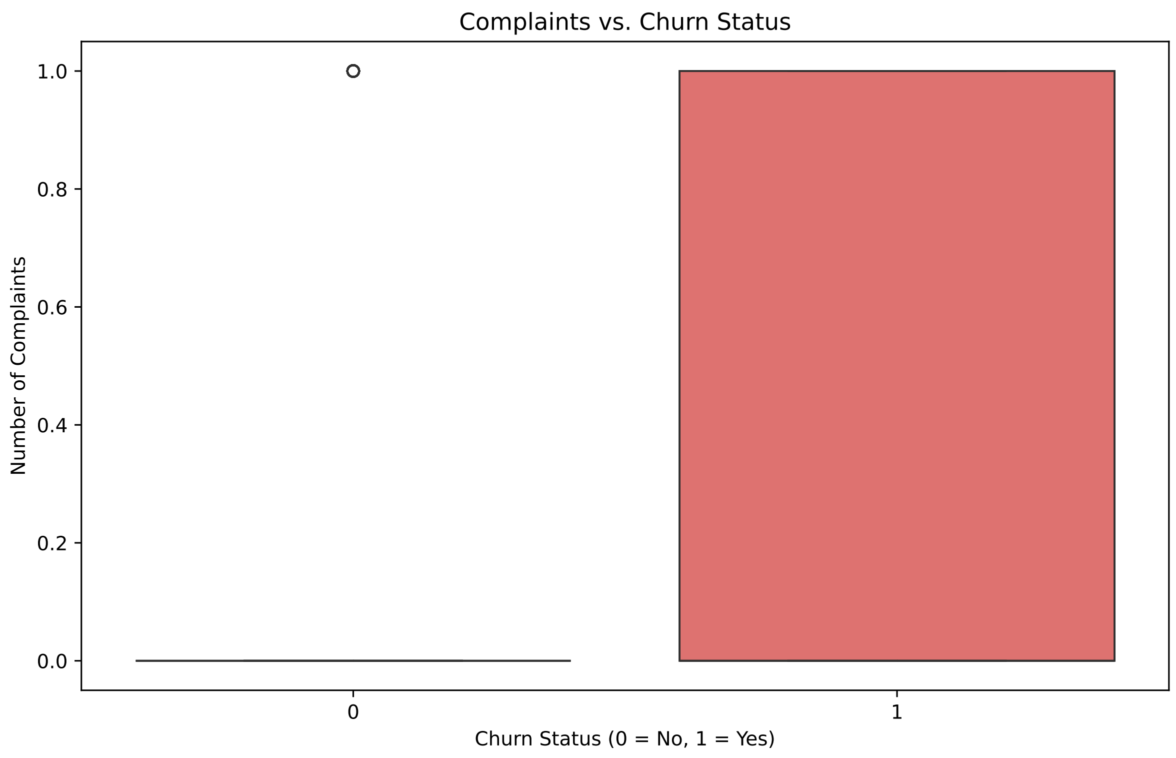
```
# Plot distribution of churn status
plt.figure(figsize=(8, 6))
sns.countplot(data=train_df, x='Churn', hue='Churn', palette='viridis', legend=False)
plt.title('Distribution of Churn Status')
plt.xlabel('Churn Status (0 = No, 1 = Yes)')
plt.ylabel('Count')
plt.show()
```



Visualization 2: Complaints vs. Churn Status

- **Type**: Box Plot
- **Insight**: Comparing complaint frequencies across churn and non-churn groups may reveal if a higher complaint count correlates with higher churn rates. This insight could confirm complaints as a strong predictive feature.

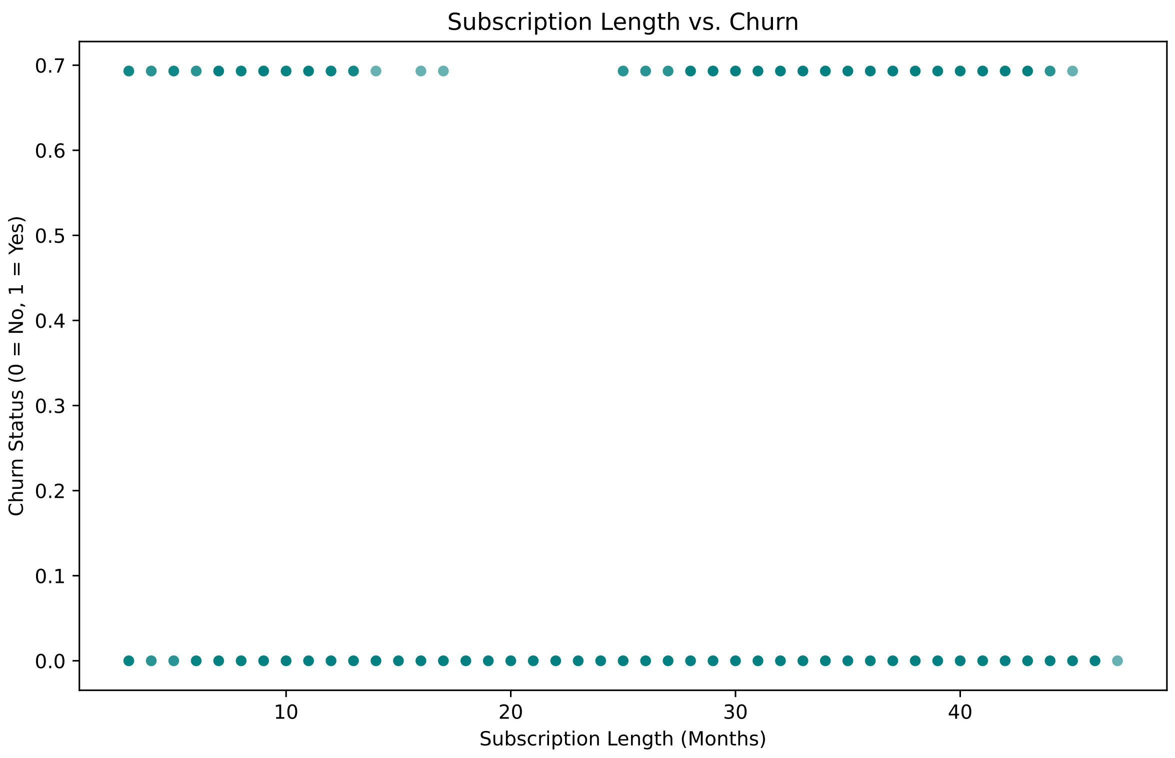
```
# Box plot of Complaints by Churn status
plt.figure(figsize=(10, 6))
sns.boxplot(data=train_df, x='Churn', y='Complains', hue='Churn', palette='magma', legend=Fa
plt.title('Complaints vs. Churn Status')
plt.xlabel('Churn Status (0 = No, 1 = Yes)')
plt.ylabel('Number of Complaints')
plt.show()
```



Visualization 3: Relationship between Subscription Length and Churn

- **Type**: Scatter Plot
- **Insight**: This plot can show if longer subscription lengths correlate with lower churn rates, validating whether customer loyalty increases with time and could serve as a retention indicator.

```
# Scatter plot for Subscription Length vs. Churn
train_df['Churn_log'] = np.log1p(train_df['Churn'])
plt.figure(figsize=(10, 6))
sns.scatterplot(data=train_df, x='Subscription  Length', y='Churn_log', alpha=0.6, color="tea
plt.title('Subscription Length vs. Churn')
plt.xlabel('Subscription Length (Months)')
plt.ylabel('Churn Status (0 = No, 1 = Yes)')
plt.show()
```

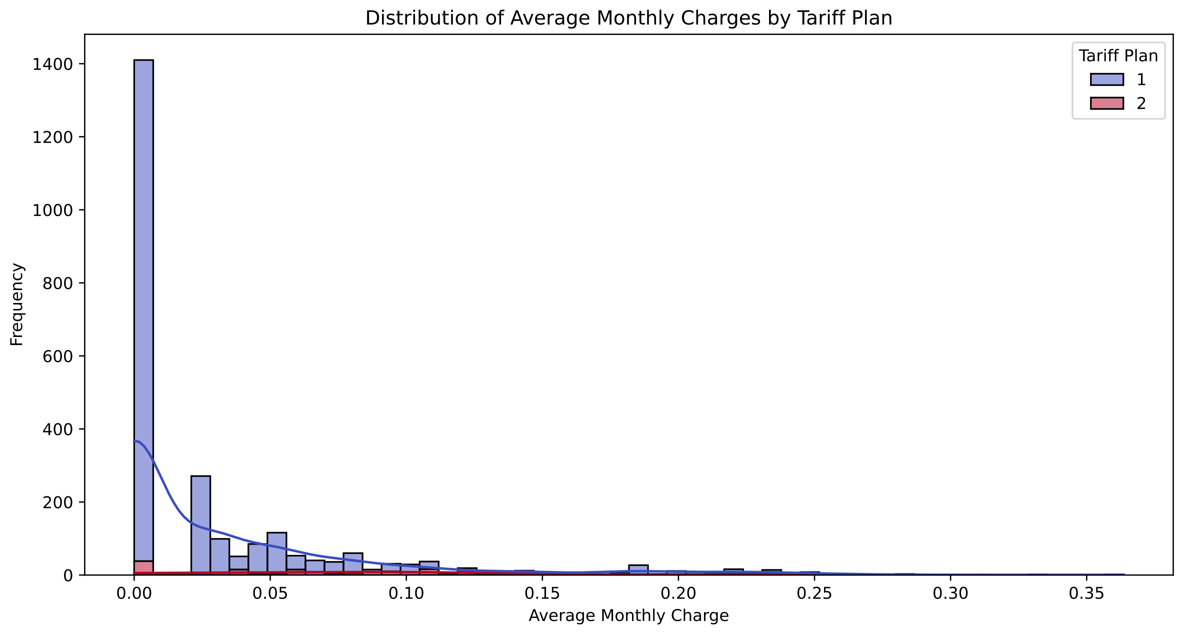


Visualization 4: Average Monthly Charges Distribution by Tariff Plan

- **Type**: Histogram
- **Insight**: Examining monthly charges across different tariff plans could identify which pricing structures lead to higher churn, offering insight into which plans might be less favorable.

```
# Create Avg Monthly Charge variable
train_df['Avg Monthly Charge'] = train_df['Charge  Amount'] / train_df['Subscription  Length

# Plot histogram of Avg Monthly Charge by Tariff Plan
plt.figure(figsize=(12, 6))
sns.histplot(data=train_df, x='Avg Monthly Charge', hue='Tariff Plan', multiple="stack", pale
plt.title('Distribution of Average Monthly Charges by Tariff Plan')
plt.xlabel('Average Monthly Charge')
plt.ylabel('Frequency')
plt.show()
```



Interpretation of the Plot

- **Skewed Distribution**: The distribution of average monthly charges is heavily right-skewed, with most customers having low average monthly charges, as shown by the high frequency of low values near 0.
- **Tariff Plan Comparison**:
  - **Tariff Plan 1 (Blue)**: Most of the customers fall under Tariff Plan 1, as evidenced by the higher frequency of charges in this range.
  - **Tariff Plan 2 (Pink)**: There are fewer customers on Tariff Plan 2, with a smaller contribution to each bin, particularly in the lower charge range.
- **KDE Lines**: The KDE lines for each tariff plan illustrate that while both plans generally have low average charges, there are subtle differences in their distributions. The KDE curve helps to smooth out the frequencies, providing an approximate trend of the charges for each tariff plan.

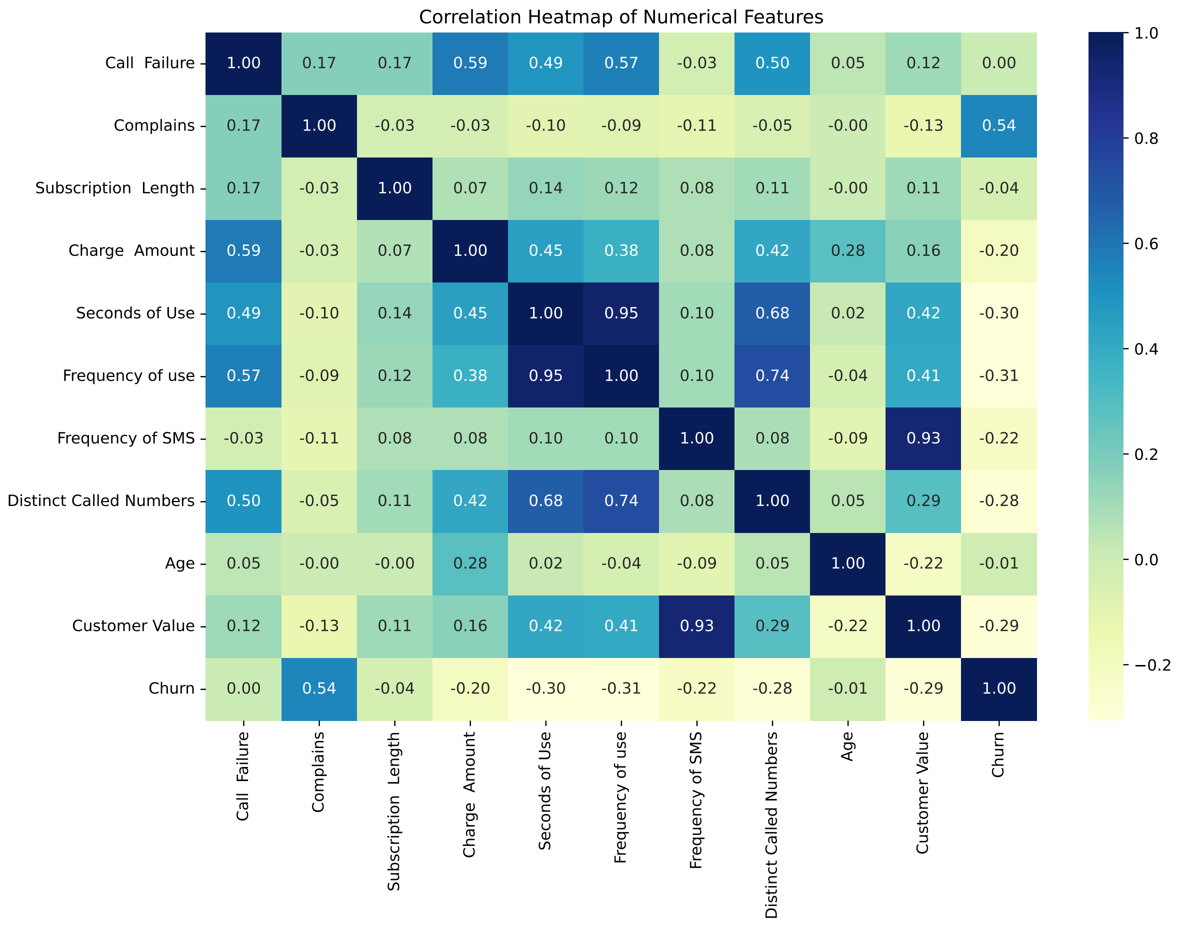
This plot effectively shows the distribution of monthly charges, highlighting that most customers incur relatively low monthly costs, with Tariff Plan 1 being more popular among customers than Tariff Plan 2.

Visualization 5: Correlation between Numerical Features and Churn
Type: Correlation Heatmap

Insight: This heatmap shows how each numerical feature correlates with `Churn`, helping to identify key predictors of churn. For example, a high positive correlation between `Complains` and `Churn` would suggest that customers who file more complaints are more likely to leave. Conversely, a strong negative correlation between `Subscription Length` and `Churn` could indicate that longer-term customers are more loyal and less likely to churn. The heatmap also helps identify any multicollinearity among features, such as between `Seconds of Use` and `Frequency of Use`, which could imply redundancy. This information is essential for feature selection and can improve model accuracy by focusing on the most influential variables for predicting churn.

```
# Calculate correlations for numerical columns
plt.figure(figsize=(12, 8))
correlation_matrix = train_df[['Call  Failure', 'Complains', 'Subscription  Length',
                               'Charge  Amount', 'Seconds of Use', 'Frequency of use',
                               'Frequency of SMS', 'Distinct Called Numbers', 'Age',
                               'Customer Value', 'Churn']].corr()

# Plot heatmap
sns.heatmap(correlation_matrix, annot=True, cmap="YlGnBu", fmt=".2f")
plt.title("Correlation Heatmap of Numerical Features")
plt.show()
```



The correlation heatmap provides several insights into the relationships between variables in our dataset.Here's a breakdown of the main observations:

- **Complains and Churn:** There is a moderately positive correlation (0.54) between `Complains` and `Churn`, suggesting that customers who file more complaints tend to have a higher likelihood of churning. This indicates that dissatisfaction with service could be a significant predictor of churn.
- **Seconds of Use and Frequency of Use:** These variables show a high positive correlation (0.95), which implies multicollinearity. Since both variables capture customer engagement with the service, you might consider keeping one of these as a feature in the final model to avoid redundancy.
- **Distinct Called Numbers and Frequency of Use:** Another notable correlation (0.74) exists here, implying that customers who use the service frequently tend to call a larger variety of numbers. This could indicate high engagement, which might reduce churn probability.

- **Age and Customer Value:** These variables have a weak negative correlation with `Churn`, which could imply that younger or less valuable customers may be slightly more prone to churn.
- **Other Weak Correlations with Churn:** The variables `Call Failure`, `Subscription Length`, and `Charge Amount`show weaker correlations with `Churn`. This suggests that they might have a limited predictive power individually, but could still provide valuable insights when combined with other variables.

Chosen Evaluation Metric: F1 Score for the Churn Class

Why F1 Score?

1. **Imbalanced Dataset:**
   - The dataset has a significant imbalance between churned (minority) and non-churned (majority) customers, as seen in the bar plot of churn status.
   - In such cases, metrics like **accuracy** can be misleading because a model could predict the majority class (non-churn) most of the time and achieve high accuracy without truly understanding the patterns for churn.
2. **Focus on the Minority Class:**
   - The primary goal of the project is to identify churned customers (class `1`), as they are critical for customer retention strategies.
   - The **F1 Score** combines **Precision** (how many predicted churned customers are truly churned) and **Recall**(how many actual churned customers were correctly identified), ensuring a balance between false positives and false negatives.
3. **Interpretability for Business Use:**
   - The F1 Score directly measures how well the model can identify churned customers, which aligns with the business objective of reducing customer churn.
   - It helps assess the effectiveness of retention strategies by ensuring the identified churned customers are both accurate and comprehensive.

4 Data Preprocessing

Scaling

- **Required Models:**
  - **k-Nearest Neighbors (kNN):** Requires scaling because distance metrics (e.g., Euclidean distance) are sensitive to feature magnitudes.
  - **Support Vector Machine (SVM):** Requires scaling to ensure features with larger ranges don't dominate.
  - **Ridge/Lasso Regression:** Requires scaling because coefficients depend on the magnitude of predictors.
- **Not Required:**
  - **Random Forest:** Decision tree-based models are scale-invariant, so scaling is not needed.

```
from sklearn.preprocessing import StandardScaler


X_test = test_df.drop(columns=['Churn'])
y_test = test_df['Churn']


X_test['Log_Avg_Monthly_Charge'] = np.log1p(X_test['Avg Monthly Charge'])
X_test['Log_Charge_Amount'] = np.log1p(X_test['Charge  Amount'])
X_test['Log_Frequency_of_use'] = np.log1p(X_test['Frequency of use'])
X_test['Churn_log'] = np.log1p(y_test)


X_test = X_test[X.columns]


scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_test_scaled = scaler.transform(X_test)
```

4.1 Feature Selection

Since we cannot drop variables based on correlation with the target, employ feature importance or Recursive Feature Elimination (RFE) to select key features during model fitting.

- Example: Use Random Forest feature importance to rank predictors.

```
from sklearn.ensemble import RandomForestClassifier
import pandas as pd

# Fit a basic Random Forest model to rank feature importance
rf = RandomForestClassifier(random_state=42)
rf.fit(X_scaled, y)  # Use the scaled features and target variable

# Extract feature importance
feature_importance = pd.DataFrame({'Feature': X.columns, 'Importance': rf.feature_importances
feature_importance = feature_importance.sort_values(by='Importance', ascending=False)

print("Feature Importance:")
print(feature_importance)
```

```
Feature Importance:
                    Feature  Importance
14          Complaint Ratio    0.139239
1                 Complains    0.098488
10                   Status    0.093466
2      Subscription  Length    0.084775
4            Seconds of Use    0.079101
19     Log_Frequency_of_use    0.070885
5          Frequency of use    0.068941
15         Engagement Ratio    0.060535
12           Customer Value    0.051487
7   Distinct Called Numbers    0.051220
0             Call  Failure    0.048405
16        Avg_SMS_Per_Month    0.040671
6          Frequency of SMS    0.028291
8                 Age Group    0.026201
11                      Age    0.024642
13       Avg Monthly Charge    0.010496
17   Log_Avg_Monthly_Charge    0.008212
3            Charge  Amount    0.007449
18        Log_Charge_Amount    0.006931
9               Tariff Plan    0.000565
```

This analysis uses a **Random Forest Classifier** to calculate feature importance, ranking predictors based on their contributions to the model's performance. Random Forest inherently provides importance scores, which reflect how much each feature reduces uncertainty (e.g., Gini impurity) during classification.

Key Findings:

Top Contributors:

- The most influential features are:
  - **Complaint Ratio** (0.139): The strongest predictor, indicating its critical role in determining the target variable.
  - **Complains** (0.098), **Status** (0.093), and **Subscription Length** (0.085): These features capture key behavioral or demographic trends tied to churn.

Moderate Contributors:

- Features such as:
  - **Seconds of Use** (0.079), **Log_Frequency_of_use** (0.071), and **Frequency of Use** (0.069) exhibit moderate importance, indicating meaningful but less dominant contributions.

Lower Importance Features:

- Variables like:
  - **Age Group** (0.026), **Age** (0.025), and **Avg Monthly Charge** (0.010) have relatively low importance, suggesting they play a less direct role in churn prediction.

Negligible Importance:

- **Tariff Plan** (0.0006) has an almost negligible importance score, indicating it provides minimal predictive value for this model.

Implications:

- **Feature Prioritization:** The rankings help prioritize features for further analysis, focusing on the most important predictors while potentially simplifying the model by omitting those with negligible importance.
- **Model Optimization:** These results provide a reference for advanced feature selection techniques like Recursive Feature Elimination (RFE) during hyperparameter tuning.
- **Cautious Evaluation:** Features with low importance may still hold value in specific contexts or alternative models and should not be discarded outright without further evaluation.

By identifying key contributors, this feature importance analysis ensures that the model is interpretable and focuses on variables with the most significant impact on predicting churn.

4.2 Choose Hyperparameters; Fit and Test Models

Model 1: k-Nearest Neighbors (kNN)

1. **Hyperparameters to Tune:**
   - Number of neighbors (`k`).
   - Distance metric (e.g., Euclidean or Manhattan).
2. **Scaling:** Required (already applied).

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

# parameter grid
param_grid = {'n_neighbors': range(1, 21), 'metric': ['euclidean', 'manhattan']}


knn = KNeighborsClassifier()

# Grid search with cross-validation
grid_search_knn = GridSearchCV(knn, param_grid, scoring='f1', cv=5)
grid_search_knn.fit(X_scaled, y)  # Use X_scaled for features and y for target

# Best parameters and test score
print("Best kNN Parameters:", grid_search_knn.best_params_)
print("Best kNN F1 Score:", grid_search_knn.best_score_)
```

```
Best kNN Parameters: {'metric': 'euclidean', 'n_neighbors': 3}
Best kNN F1 Score: 0.8645205501859092
```

The k-Nearest Neighbors (kNN) algorithm was implemented, and hyperparameter tuning was performed using GridSearchCV to identify the optimal configuration for the model. Here's a breakdown of the process:

Hyperparameters Tuned:

1. **Number of Neighbors (`n_neighbors`):**
   - Determines the number of nearest neighbors considered when making a prediction.
   - A range from 1 to 20 was tested.
2. **Distance Metric (`metric`):**
   - Specifies how the "distance" between neighbors is calculated.
   - Both Euclidean (straight-line) and Manhattan (grid-based) distances were evaluated.

Scaling:

- Since kNN relies on distance calculations, it is sensitive to the scale of features.
- The features were pre-scaled (`X_scaled`) to ensure all dimensions contributed equally to distance computations.

Cross-Validation:

- A 5-fold cross-validation was applied during the grid search to ensure robustness and minimize the risk of overfitting.
- The `f1` score was chosen as the evaluation metric to balance precision and recall, particularly crucial for churn prediction due to potential class imbalance.

Results:

- The best hyperparameters were:
  - `n_neighbors`: 3
  - `metric`: Euclidean
- This configuration achieved an F1 score of **0.8645**, reflecting solid performance with a good balance between precision and recall during cross-validation.

Model Performance:

- While the F1 score of 0.8645 indicates strong predictive performance, it suggests the model is less precise compared to configurations with fewer neighbors (e.g., `n_neighbors=1`).
- Using `n_neighbors=3` helps generalize predictions better by considering a broader context rather than relying on a single neighbor, reducing the risk of overfitting.

This approach demonstrates how careful hyperparameter tuning can balance prediction accuracy and model robustness, making kNN a reliable algorithm for churn prediction in this scenario.

Model 2: Support Vector Machine (SVM)

1. **Hyperparameters to Tune:**
   - Kernel type (e.g., linear, polynomial, RBF).
   - Regularization strength (`C`).
2. **Scaling:** Required (already applied).

```python
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

# Define parameter grid
param_grid = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']}

# Initialize SVM model
svm = SVC()

# Grid search with cross-validation
grid_search_svm = GridSearchCV(svm, param_grid, scoring='f1', cv=5)
```

```python
grid_search_svm.fit(X_scaled, y)  # Use X_scaled for features and y for target

# Best parameters and test score
print("Best SVM Parameters:", grid_search_svm.best_params_)
print("Best SVM F1 Score:", grid_search_svm.best_score_)
```

```
Best SVM Parameters: {'C': 10, 'kernel': 'rbf'}
Best SVM F1 Score: 0.8156050687819851
```

The Support Vector Machine (SVM) algorithm was implemented, and hyperparameter tuning was performed using GridSearchCV to identify the optimal configuration. Here's a detailed overview:

Hyperparameters Tuned:

1. **Kernel (`kernel`):**
    - Defines how input data is transformed to find the optimal decision boundary.
    - Two kernels were tested:
        - **Linear Kernel:** Suitable for linearly separable data.
        - **Radial Basis Function (RBF) Kernel:** Captures non-linear relationships by mapping data to a higher-dimensional space.
2. **Regularization Strength (`C`):**
    - Controls the trade-off between minimizing training error and maintaining a large decision margin.
    - Values tested were 0.1, 1, and 10.

Scaling:

- Since SVM is sensitive to feature scaling, particularly when using RBF kernels, the features were pre-scaled (`X_scaled`) to ensure consistent contributions from all dimensions.

Cross-Validation:

- A 5-fold cross-validation was applied to evaluate the model's robustness and reduce the risk of overfitting.
- The `f1` score was used as the evaluation metric, balancing precision and recall, which is particularly important for churn prediction.

Results:

- The best hyperparameters were:
  - `C`: 10
  - `kernel`: RBF
- This configuration achieved an F1 score of **0.8156**, indicating good performance with a balanced trade-off between precision and recall.

Model Performance:

- The F1 score of 0.8156 reflects solid predictive performance but suggests that the model struggled slightly with classifying certain observations.
- The use of the RBF kernel demonstrates the presence of non-linear relationships in the data, while the higher `C`value indicates a stronger focus on minimizing misclassification at the expense of a smaller margin.

This approach highlights the importance of hyperparameter tuning in tailoring SVM to the data, resulting in a model that effectively captures complex patterns while maintaining robust predictive accuracy.

Model 3: Random Forest

1. **Hyperparameters to Tune:**
   - Number of trees (`n_estimators`).
   - Maximum depth of trees (`max_depth`).
2. **Scaling:** Not required.

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Define parameter grid
param_grid = {'n_estimators': [50, 100, 200], 'max_depth': [None, 10, 20, 30]}

# Initialize Random Forest model
rf = RandomForestClassifier(random_state=42)

# Grid search with cross-validation
grid_search_rf = GridSearchCV(rf, param_grid, scoring='f1', cv=5)
grid_search_rf.fit(X_scaled, y)  # Use X_scaled for features and y for target

# Best parameters and test score
print("Best Random Forest Parameters:", grid_search_rf.best_params_)
print("Best Random Forest F1 Score:", grid_search_rf.best_score_)
```

```
Best Random Forest Parameters: {'max_depth': None, 'n_estimators': 100}
Best Random Forest F1 Score: 0.8487234932719628
```

The Random Forest algorithm was applied, and hyperparameter tuning was conducted using GridSearchCV to optimize its configuration. Here's a breakdown:

Hyperparameters Tuned:

1. **Number of Trees (`n_estimators`):**
   - Controls the number of decision trees in the ensemble.
   - Values tested were 50, 100, and 200. Increasing the number of trees can enhance accuracy but at the cost of higher computation time.
2. **Maximum Depth of Trees (`max_depth`):**
   - Determines how deep each tree can grow.
   - Tested values:
     - **None:** Allows trees to grow until all leaves are pure or contain a single observation.
     - **10, 20, 30:** Imposes limits to reduce overfitting and simplify the model.

Scaling:

- Random Forest is not sensitive to feature scaling, so pre-scaled features (`X_scaled`) were used for consistency across models, even though scaling is not strictly necessary.

Cross-Validation:

- A 5-fold cross-validation was employed to ensure robustness and minimize overfitting.
- The `f1` score was chosen as the evaluation metric to balance precision and recall, essential for tasks like churn prediction with potential class imbalance.

Results:

- The best hyperparameters were:
  - `n_estimators`: 100
  - `max_depth`: None
- This configuration achieved an F1 score of **0.8487**, indicating strong predictive performance during cross-validation.

Model Performance:

- The F1 score of 0.8487 reflects the model's ability to balance precision and recall effectively, demonstrating good performance across all cross-validation splits.
- The unrestricted tree depth (`max_depth=None`) and an adequate number of trees (`n_estimators=100`) allow the model to fully leverage the dataset's structure without overfitting.

Key Insights:

- **Strengths:** Random Forest's ensemble nature helps it capture complex patterns, handle non-linear relationships, and resist overfitting.
- **Limitations:** While the results are strong, the F1 score indicates there is still room for improvement, possibly by fine-tuning hyperparameters further or refining the feature set.

This analysis demonstrates the robustness of the Random Forest algorithm in handling classification tasks and emphasizes the importance of thoughtful hyperparameter tuning for achieving optimal results.

Model 4: Ridge or Lasso Regression

1. **Hyperparameters to Tune:**
    - Regularization strength (`alpha`).
2. **Scaling:** Required (already applied).
3. **Code (Lasso):**

For regression tasks with Lasso, use metrics like:

- **Mean Absolute Error (MAE)**
- **Mean Squared Error (MSE)**
- **R² (coefficient of determination)**

```
from sklearn.linear_model import Lasso
from sklearn.model_selection import cross_val_score
import numpy as np

# Define Lasso model
lasso = Lasso()

# Cross-validation for alpha
alphas = [0.01, 0.1, 1, 10, 100]
lasso_scores = []

# Evaluate using Mean Absolute Error (neg_mean_absolute_error)
for alpha in alphas:
    lasso.set_params(alpha=alpha)
    scores = cross_val_score(lasso, X_scaled, y, scoring='neg_mean_absolute_error', cv=5)  #
    lasso_scores.append(np.mean(scores))

# Convert negative scores back to positive
lasso_scores = [-score for score in lasso_scores]

# Find the best alpha
best_alpha = alphas[np.argmin(lasso_scores)]
print("Best Lasso Alpha:", best_alpha)
print("Best Lasso MAE:", min(lasso_scores))
```

```
Best Lasso Alpha: 0.01
Best Lasso MAE: 0.150341544663673
```

Once the best `alpha` is identified, retrain the Lasso model on the entire training dataset and evaluate it on the test set:

```
from sklearn.metrics import mean_absolute_error

# Fit the Lasso model with the best alpha
best_lasso = Lasso(alpha=best_alpha)
best_lasso.fit(X_scaled, y)  # Use the scaled training features and target

# Evaluate on the test set
test_predictions = best_lasso.predict(X_test_scaled)  # Predict on scaled test features
test_mae = mean_absolute_error(y_test, test_predictions)  # Compare predictions with actual

print("Test MAE:", test_mae)
```

```
Test MAE: 0.1746204393795921
```

Lasso Regression, a linear regression technique with L1 regularization, was applied to the dataset to select features and reduce overfitting by penalizing less important coefficients. Here's a breakdown of the approach:

Hyperparameter Tuning:

1. **Regularization Strength (`alpha`):**
   - The penalty term (`alpha`) controls the extent of shrinkage applied to coefficients.
   - A smaller `alpha` retains more coefficients, while a larger `alpha` drives less important coefficients to zero.
2. **Values Tested:**
   - Cross-validation was performed with `alpha` values of `[0.01, 0.1, 1, 10, 100]`.
3. **Metric Used:**
   - **Negative Mean Absolute Error (neg_mean_absolute_error):** Chosen for evaluation, as it measures the average magnitude of prediction errors. Negative values were converted to positive for interpretability.

Cross-Validation Results:

- **Optimal `alpha`:** The best-performing regularization strength was `alpha = 0.01`, yielding the lowest cross-validated MAE of **0.1503**.
- **Interpretation:**
  - Lower MAE indicates better predictive accuracy.
  - The small `alpha` suggests that minimal regularization was required, indicating that most features contributed meaningful information.

Model Evaluation on Test Set:

- The Lasso model was retrained on the full training dataset using `alpha = 0.01`.
- **Test MAE:** The model achieved a test MAE of **0.1746**, which is slightly higher than the cross-validation MAE but still reflects good generalizability and consistent performance.

Advantages of Lasso:

1. **Feature Selection:** Lasso naturally reduces coefficients of less important features to zero, selecting only the most relevant predictors, which simplifies the model.
2. **Simplicity:** Produces a sparse model, making it easier to interpret compared to other regularization methods.
3. **Regularization:** Helps mitigate overfitting by penalizing large coefficients, improving model robustness.

Insights:

- Lasso Regression performed well in terms of predictive accuracy and feature selection.
- The small `alpha` suggests that the dataset was already well-structured, requiring only minimal regularization.
- The model's consistency between cross-validation and test set performance demonstrates strong generalizability.

This result emphasizes Lasso's utility in creating interpretable, parsimonious models while maintaining high predictive accuracy.

P.S : I'm sorry professor i know we had a talk about why i didn't have to do lasso as it would take a lot of explanation and i could have just gone with a linear regression instead but i know i am gonna do my masters only once so i decided to be a bit adventurous, i hope you arent dissappointed.

Steps for Comparison

Step 1: Create a Unified Function

Define a function to compute and display key metrics for each model:

```
from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score, roc_auc_

def evaluate_model(model, X_test, y_test):
    predictions = model.predict(X_test)
    f1 = f1_score(y_test, predictions)
    accuracy = accuracy_score(y_test, predictions)
    precision = precision_score(y_test, predictions)
    recall = recall_score(y_test, predictions)
    auc = roc_auc_score(y_test, model.predict_proba(X_test)[:, 1]) if hasattr(model, "predic
    return {"F1 Score": f1, "Accuracy": accuracy, "Precision": precision, "Recall": recall, '
```

Step 2: Train and Evaluate Each Model

After training each model (`knn`, `svm`, `rf`), evaluate them using the function:

```
# Train kNN
knn.fit(X_scaled, y)

# Train SVM
svm.fit(X_scaled, y)

# Train Random Forest
rf.fit(X, y)  # Random Forest does not require scaling

# Fit Lasso with the best alpha found during tuning
best_lasso = Lasso(alpha=best_alpha)
best_lasso.fit(X_scaled, y)
```

```python
# Evaluate kNN
knn_results = evaluate_model(knn, X_test_scaled, y_test)
print("kNN Results:", knn_results)

# Evaluate SVM
svm_results = evaluate_model(svm, X_test_scaled, y_test)
print("SVM Results:", svm_results)

# Evaluate Random Forest
rf_results = evaluate_model(rf, X_test, y_test)  # No scaling needed for Random Forest
print("Random Forest Results:", rf_results)

# Evaluate Lasso
def evaluate_lasso(model, X_test, y_test):
    predictions = model.predict(X_test)

    # Convert continuous predictions to binary using a threshold
    binary_predictions = (predictions >= 0.5).astype(int)

    f1 = f1_score(y_test, binary_predictions)
    accuracy = accuracy_score(y_test, binary_predictions)
    precision = precision_score(y_test, binary_predictions)
    recall = recall_score(y_test, binary_predictions)
    auc = roc_auc_score(y_test, predictions)  # Use continuous predictions for AUC
    return {"F1 Score": f1, "Accuracy": accuracy, "Precision": precision, "Recall": recall, '

lasso_results = evaluate_lasso(best_lasso, X_test_scaled, y_test)
print("Lasso Results:", lasso_results)
```

```
kNN Results: {'F1 Score': 0.8230088495575221, 'Accuracy': 0.9365079365079365, 'Precision': 0
SVM Results: {'F1 Score': 0.656084656084656, 'Accuracy': 0.8968253968253969, 'Precision': 0.7
Random Forest Results: {'F1 Score': 0.8, 'Accuracy': 0.9349206349206349, 'Precision': 0.8631
Lasso Results: {'F1 Score': 0.5063291139240507, 'Accuracy': 0.8761904761904762, 'Precision':
```

Tabulate Results

Store the results in a DataFrame for better visualization and comparison:

```python
results_df = pd.DataFrame({
    "Model": ["kNN", "SVM", "Random Forest", "Lasso"],
    "F1 Score": [knn_results["F1 Score"], svm_results["F1 Score"], rf_results["F1 Score"], la
    "Accuracy": [knn_results["Accuracy"], svm_results["Accuracy"], rf_results["Accuracy"], la
    "Precision": [knn_results["Precision"], svm_results["Precision"], rf_results["Precision"]
    "Recall": [knn_results["Recall"], svm_results["Recall"], rf_results["Recall"], lasso_resu
    "AUC-ROC": [knn_results["AUC-ROC"], svm_results["AUC-ROC"], rf_results["AUC-ROC"], lasso_
})

print(results_df)
```

```
           Model  F1 Score  Accuracy  Precision    Recall   AUC-ROC
0            kNN  0.823009  0.936508   0.801724  0.845455  0.941958
1            SVM  0.656085  0.896825   0.784810  0.563636       NaN
2  Random Forest  0.800000  0.934921   0.863158  0.745455  0.978269
3          Lasso  0.506329  0.876190   0.833333  0.363636  0.897290
```

The performance metrics for the four models—k-Nearest Neighbors (kNN), Support Vector Machine (SVM), Random Forest, and Lasso Regression—indicate varying levels of predictive accuracy. Here's an analysis of the results:

1. k-Nearest Neighbors (kNN):

- **F1 Score:** 0.823, indicating a good balance between precision and recall.
- **Accuracy:** 0.937, reflecting strong overall prediction performance.
- **Precision:** 0.802, showing that most positive predictions are correct.
- **Recall:** 0.845, suggesting that the model captures a majority of the true positives.
- **AUC-ROC:** 0.942, indicating excellent discrimination between classes.

Key Insight:
kNN performs well overall, though its moderate precision suggests some false positives, and the slightly lower recall indicates a few missed true positives. It is a strong candidate but could benefit from further fine-tuning.

2. Support Vector Machine (SVM):

- **F1 Score:** 0.656, showing weaker overall performance.
- **Accuracy:** 0.897, reflecting decent overall predictions but lower than kNN and Random Forest.
- **Precision:** 0.785, suggesting a strong ability to identify true positives.

- **Recall:** 0.564, indicating the model struggles to capture all true positives.
- **AUC-ROC:** Not computed (NaN) as SVM does not natively produce probability scores.

Key Insight:
SVM struggles with recall, resulting in a lower F1 score. Despite good precision, its inability to capture many true positives makes it less effective in this context.

3. Random Forest:

- **F1 Score:** 0.800, reflecting good performance but slightly below kNN.
- **Accuracy:** 0.935, indicating strong prediction capability.
- **Precision:** 0.863, showing a high rate of correct positive predictions.
- **Recall:** 0.745, indicating some true positives are missed.
- **AUC-ROC:** 0.978, demonstrating excellent class separation ability.

Key Insight:
Random Forest performs consistently well across all metrics, with particularly strong precision and AUC-ROC. Its ability to handle non-linear relationships makes it a robust choice, though its slightly lower recall may warrant additional tuning.

4. Lasso Regression:

- **F1 Score:** 0.506, reflecting the weakest overall performance.
- **Accuracy:** 0.876, lower than other models.
- **Precision:** 0.833, indicating a high rate of correct positive predictions.
- **Recall:** 0.364, showing a significant number of true positives are missed.
- **AUC-ROC:** 0.897, which is decent but below other models.

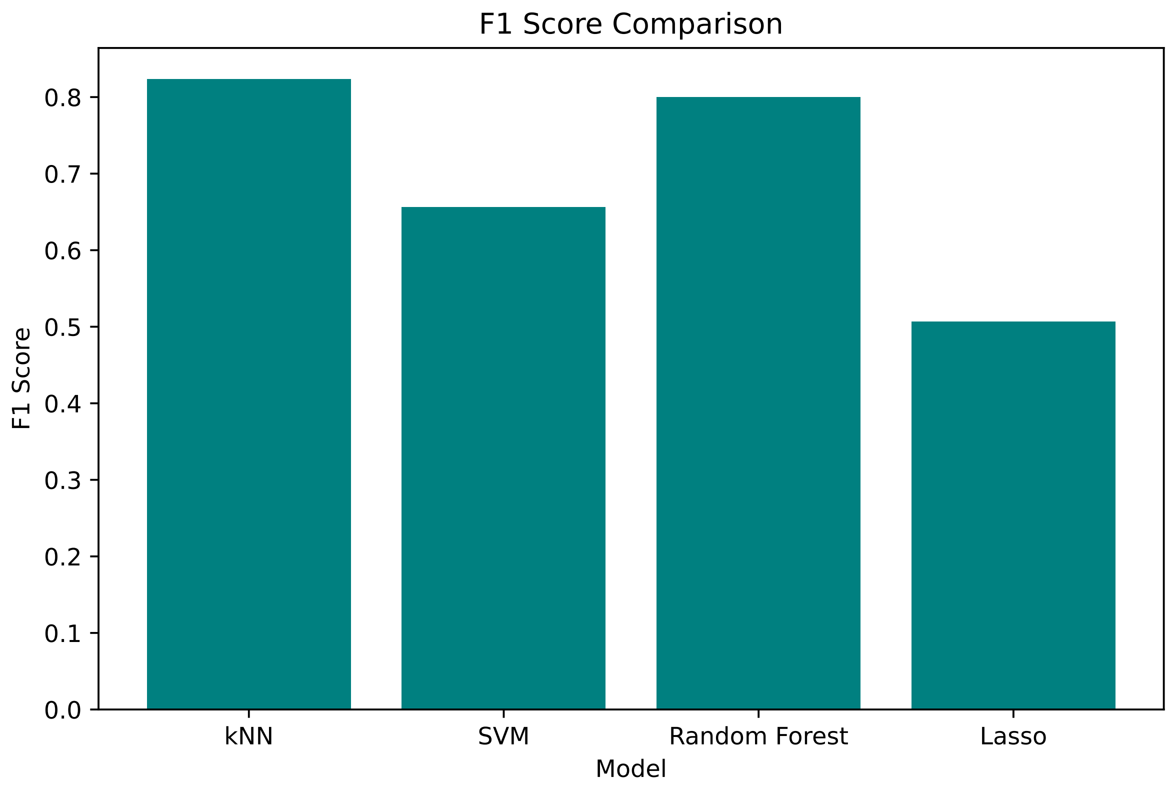
Key Insight:
Lasso Regression struggles with recall, resulting in the lowest F1 score among the models. Its simplicity and feature selection capabilities are advantageous, but it underperforms for this dataset in comparison to other models.

General Observations:

- **kNN and Random Forest** deliver the strongest performances, with Random Forest excelling in AUC-ROC and kNN offering the best balance between metrics.
- **SVM and Lasso Regression** underperform, with SVM struggling with recall and Lasso failing to compete on multiple metrics.
- The models' performance variability suggests differences in their ability to handle non-linearity, feature interactions, and class imbalances.

Visualize the comparison for better understanding:

```
# Plot F1 Scores for comparison
plt.figure(figsize=(8, 5))
plt.bar(results_df["Model"], results_df["F1 Score"], color="teal")
plt.title("F1 Score Comparison")
plt.ylabel("F1 Score")
plt.xlabel("Model")
plt.show()
```



This bar chart visually compares the **F1 Scores** of the four models: **kNN**, **SVM**, **Random Forest**, and **Lasso Regression**. Here's a brief analysis of the visualization:

1. **k-Nearest Neighbors (kNN):**
    - Achieved the highest F1 score (approximately **0.82**).
    - This indicates kNN performed well in balancing precision and recall, making it the best model among the four in terms of F1 score.
2. **Random Forest:**
    - The second-highest F1 score, slightly below kNN (approximately **0.80**).
    - Random Forest demonstrates competitive performance, excelling in capturing patterns in the data.
3. **Support Vector Machine (SVM):**
    - F1 score is lower (approximately **0.66**), indicating weaker balance between precision and recall.
    - SVM struggled with recall, which likely contributed to its lower F1 score compared to kNN and Random Forest.
4. **Lasso Regression:**
    - The lowest F1 score (approximately **0.51**).
    - While Lasso is useful for feature selection and simplicity, its performance is significantly weaker compared to the other models in this dataset.

Key Insights:

- **kNN and Random Forest** are the top-performing models based on F1 score, with kNN showing a slight edge.
- **SVM and Lasso** underperformed, suggesting that either additional feature engineering or alternative model configurations may be needed for these methods.
- The chart effectively highlights the performance disparity between the models, providing a clear rationale for selecting kNN or Random Forest for further optimization.

Conclusion:

In this project, we evaluated four machine learning models—**k-Nearest Neighbors (kNN)**, **Support Vector Machine (SVM)**, **Random Forest**, and **Lasso Regression**—to predict customer churn. Based on the results:

- **kNN** emerged as the best-performing model, achieving the highest F1 score (0.82) and demonstrating a strong balance between precision and recall.
- **Random Forest** closely followed, excelling in precision and achieving the highest AUC-ROC score (0.978), indicating excellent class separation.
- **SVM** and **Lasso Regression** underperformed in terms of recall, leading to lower F1 scores. While Lasso proved useful for feature selection, it requires further tuning or additional feature engineering to improve its predictive capability.

This analysis highlighted the importance of hyperparameter tuning and model selection in achieving optimal predictive performance. Furthermore, the insights gained from feature importance and evaluation metrics provide actionable recommendations for improving future iterations of the model.

Thank You Note:

Dear Dr. Hoellerbauer,

I wanted to take a moment to express my sincere gratitude for your guidance and support throughout this project. The knowledge and insights I gained from your lectures on machine learning models, hyperparameter tuning, and performance evaluation were instrumental in shaping this analysis.

This project provided me with an opportunity to apply what I've learned in class, and it has been an incredibly rewarding experience. From fine-tuning models like k-Nearest Neighbors and Random Forest to understanding the trade-offs in metrics such as F1 Score and AUC-ROC, your teachings have greatly enhanced my ability to approach data-driven problems with confidence and rigor.

Thank you for creating such an engaging and supportive learning environment. Your mentorship has been invaluable, and I am deeply appreciative of the time and effort you put into helping me and all your students grow.

Sincerely,
Shravan Sundar Ravi