Annexure 2

Introduction to AI and ML

CSL236

Project Report

Faculty name: Dr. Meghna Sharma

Student name: Bhavay Mehta, Abhishek Singla, Ujjwal Singh

Roll No.: 22CSU434, 22CSU437, 22CSU484

Semester:5th Semester

Group: DS-B DS-4

Department of Computer Science and Engineering

The NorthCap University, Gurugram- 122001, India

Session 2024-25

Table of Contents

S.No		Page No.
1.	Project Description	1
2.	Problem Statement	1
3.	Analysis 3.1 Hardware Requirements 3.2 Software Requirements	1-2
4.	Design 4.1 Data/Input Output Description: 4.2 Algorithmic Approach / Algorithm / DFD / ER diagram/Program Steps	2-4
5.	Implementation and Testing (stage/module wise)	4-5
6.	Output (Screenshots)	5-10
7.	Conclusion and Future Scope	10-11

Databel - Customer Churn Analysis

1. Project Description

This project focuses on the analysis and prediction of customer churn using machine learning techniques. The goal is to identify whether a customer is likely to churn based on historical behavioral and transactional data. Various machine learning models, such as Logistic Regression, Support Vector Machines (SVM), Random Forest, Decision Trees, and K-Nearest Neighbors (KNN), are evaluated to determine the most accurate classifier for this task.

The analysis includes feature selection, data preprocessing, model training, and validation using metrics such as accuracy and ROC curves.

2. Problem Statement

Customer churn is a critical challenge faced by organizations, especially in subscription-based industries such as telecommunications, banking, and retail. Losing customers impacts revenue and increases acquisition costs. The objective of this project is to:

- Build a predictive model to identify churners.
- Compare the performance of different machine learning algorithms.
- Provide actionable insights that help businesses proactively retain customers.

3. Analysis

The dataset contains features such as customer usage behavior, demographic details, and subscription plans. The target variable is **Churn Label**, indicating whether the customer has churned (Yes) or not (No). Data preprocessing and exploratory data analysis were performed to handle missing values, standardize the features, and understand correlations.

Five models were implemented:

- **Random Forest** achieved the highest accuracy of **97.01%**, followed by:
    - Logistic Regression: 96.94%
    - SVM: 96.56%
    - KNN: 96.03%
    - Decision Tree: 94.25%

Performance was visualized using accuracy comparison and ROC curves.

3.1 Hardware Requirements

- **Processor**: Intel Core i5 or equivalent
- **RAM**: 16 GB (minimum)
- **Storage**: 512 GB SSD
- **Operating System**: Windows 11

3.2 Software Requirements

- **Python 3.x**: Core programming language for data analysis.
- **Libraries**:
    - Pandas for data manipulation
    - NumPy for numerical calculations
    - Matplotlib and Seaborn for data visualization
    - sklearn for machine learning models (e.g., Logistic Regression)
    - StandardScaler for feature scaling
    - statsmodels for multicollinearity analysis using VIF
    - Jupyter Notebook for coding environment
- **Development Environment**: Jupyter Notebook or any Python IDE
- **Visualization Tools**: Matplotlib, Seaborn for graphical representation of data.

4. Design

4.1 Data/Input Output Description

The dataset contains customer data, likely from a telecommunications or subscription-based service. Here is a brief breakdown of the columns:

1. Customer ID: Unique identifier for each customer.

2. Churn Label: Indicates if the customer has churned ("Yes") or not ("No").

3. Account Length (in months): Duration the customer has been with the service.

4. Local Calls & Local Mins: Number and duration of local calls.

5. Intl Calls & Intl Mins: Number and duration of international calls.

6. Intl Active: Whether international services are active ("Yes" or "No").

7. Intl Plan: Indicates if the customer has an international plan.

8. Extra International Charges: Charges for international services.

9. Senior: Indicates if the customer is a senior citizen.

10. Group: Whether the customer is part of a group plan.

11. Number of Customers in Group: Number of customers in the group plan.

12. Device Protection & Online Backup: Service-related fields indicating extra services.

13. Contract Type: Type of contract (e.g., Month-to-Month, One Year).

14. Payment Method: How the customer pays (e.g., Direct Debit, Paper Check).

15. Monthly Charge & Total Charges: Monthly and total billed amounts.

16. Churn Category & Churn Reason: Categories and reasons related to customer churn (if they have churned).

Some rows have missing values for churn category and churn reason, which likely apply to customers who have not churned.

4.2 Algorithmic Approach / Algorithm / DFD / ER Diagram / Program Steps

Algorithmic Approach

1. **Data Loading**:
    - Import essential libraries and load the dataset using pd.read_csv().
    - Inspect the dataset for structure, data types, and missing values.
2. **Data Cleaning**:
    - Handle missing values through imputation (mean, median).
    - Remove irrelevant or redundant columns.
3. **Target Variable Transformation**:

    Churn Label: Indicates if the customer has churned ("Yes") or not ("No").
4. **Exploratory Data Analysis (EDA)**:

    - Visualize the distribution of features using histograms and boxplots.
    - Generate a correlation matrix to understand feature relationships.

5. **Multicollinearity Check**:
    - Calculate Variance Inflation Factor (VIF) for each feature.
    - Remove or adjust features with high multicollinearity to ensure model stability.
6. **Feature Scaling**:
    - Standardize the data using StandardScaler to ensure all features are on a similar scale.
7. **Data Splitting**:
    - Split the dataset into training (80%) and testing (20%) sets.
8. **Model Training**:
    - Train a classification model (e.g., **SVM, Random Forest, KNN, Logistic Regression**) using the training dataset.
    - Evaluate feature importance and examine model coefficients.
9. **Model Evaluation**:
    - Evaluate the classification model using metrics such as **Accuracy** and **F1 Score**.
    - Visualize the performance using confusion matrices and ROC curves.
10. **Cross-Validation**:
    - Perform 5-fold cross-validation to assess model robustness.
    - Summarize the cross-validation results using boxplots to visualize metric consistency.

5. Implementation and Testing (Stage/Module Wise)

Stage 1: Data Preparation

- **Tasks**: Data loading, cleaning, and target variable transformation.
- **Testing**: Verified that the data was correctly loaded and cleaned, with appropriate handling of missing values. Ensured successful transformation of the target variable into categories.

Stage 2: Exploratory Data Analysis (EDA)

- **Tasks**: Visualize data distribution, check for outliers, and analyze correlations.

- **Testing**: Created visualizations to confirm data distributions and correlations. Addressed any outliers detected during this phase.

Stage 3: Multicollinearity and Feature Engineering

- **Tasks**: Calculate VIF and handle multicollinearity by removing or adjusting features.
- **Testing**: Verified that high-VIF features were identified and managed. Confirmed feature adjustments did not negatively impact the dataset.

Stage 4: Data Splitting and Feature Scaling

- **Tasks**: Split data into training and testing subsets and perform feature scaling.
- **Testing**: Ensured that the split ratio was maintained and that all features were scaled appropriately without introducing bias.

Stage 5: Model Training and Evaluation

- **Tasks**: Train classification models using the training set and evaluate their performance.
- **Testing**: Evaluated model performance using metrics and visualizations. Conducted validation tests to ensure predictions aligned with expectations.

Stage 6: Cross-Validation

- **Tasks:** Conducted 5-fold cross-validation to evaluate the stability and reliability of the machine learning models in predicting customer churn.
- **Testing:** Analyzed cross-validation results to ensure consistency in performance metrics such as accuracy, precision, recall, and AUC across all folds, confirming the robustness of the models in handling variations in the data.

6.Output (Screenshots)

Importing the libraries

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.decomposition import PCA

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score,classification_report
import warnings
plt.style.use('ggplot')
%matplotlib inline

warnings.filterwarnings('ignore')
```

Importing dataset

```python
pd.set_option("display.max_column",29)
```

```python
df = pd.read_csv(r"C:\Users\Lakshay\Downloads\Databel - Data (1).csv")
```

```python
df.head()
```

ustomer ID	Churn Label	Account Length (in months)	Local Calls	Local Mins	Intl Calls	Intl Mins	Intl Active	Intl Plan	Extra International Charges	Customer Service Calls	A Month G Downlo
4444-BZPU	No	1	3	8.0	0.0	0.0	No	no	0.0	0	
5676-PTZX	No	33	179	431.3	0.0	0.0	No	no	0.0	0	
8532-ZEKQ	No	44	82	217.6	0.0	0.0	No	yes	0.0	0	
1314-SMPJ	No	10	47	111.6	60.0	71.0	Yes	yes	0.0	0	
2956-TXCJ	No	62	184	621.2	310.0	694.4	Yes	yes	0.0	0	

```python
df.sample(5)
```

	Customer ID	Churn Label	Account Length (in months)	Local Calls	Local Mins	Intl Calls	Intl Mins	Intl Active	Intl Plan	Extra International Charges	Customer Service Calls
4524	8401-RNML	Yes	6	44	69.0	24.0	84.6	Yes	no	21.2	1
1967	3356-CDIH	No	70	402	844.0	0.0	0.0	No	yes	0.0	0
3824	2083-SPHH	Yes	41	224	493.3	212.0	689.0	Yes	no	172.3	4
3063	6857-XPYD	No	33	216	425.7	132.0	264.0	Yes	no	66.0	0
1901	4140-JQIZ	No	72	333	1012.9	0.0	0.0	No	no	0.0	0

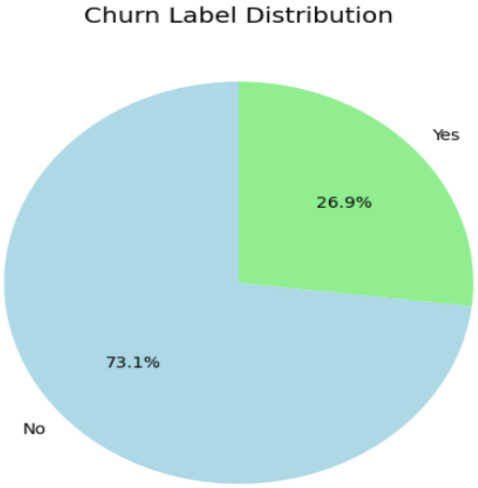
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6687 entries, 0 to 6686
Data columns (total 29 columns):
 #   Column                             Non-Null Count  Dtype  
---  ------                             --------------  -----  
 0   Customer ID                        6687 non-null   object 
 1   Churn Label                        6687 non-null   object 
 2   Account Length (in months)         6687 non-null   int64  
 3   Local Calls                        6687 non-null   int64  
 4   Local Mins                         6687 non-null   float64
 5   Intl Calls                         6687 non-null   float64
 6   Intl Mins                          6687 non-null   float64
 7   Intl Active                        6687 non-null   object 
 8   Intl Plan                          6687 non-null   object 
 9   Extra International Charges        6687 non-null   float64
 10  Customer Service Calls             6687 non-null   int64  
 11  Avg Monthly GB Download            6687 non-null   int64  
 12  Unlimited Data Plan                6687 non-null   object 
 13  Extra Data Charges                 6687 non-null   int64  
 14  State                              6687 non-null   object 
 15  Phone Number                       6687 non-null   object 
 16  Gender                             6687 non-null   object 
 17  Age                                6687 non-null   int64  
 18  Under 30                           6687 non-null   object 
 19  Senior                             6687 non-null   object 
 20  Group                              6687 non-null   object 
 21  Number of Customers in Group       6687 non-null   int64  
 22  Device Protection & Online Backup  6687 non-null   object 
 23  Contract Type                      6687 non-null   object 
 24  Payment Method                     6687 non-null   object 
 25  Monthly Charge                     6687 non-null   int64  
 26  Total Charges                      6687 non-null   int64  
 27  Churn Category                     1769 non-null   object 
 28  Churn Reason                       1769 non-null   object 
dtypes: float64(4), int64(9), object(16)
memory usage: 1.5+ MB
```

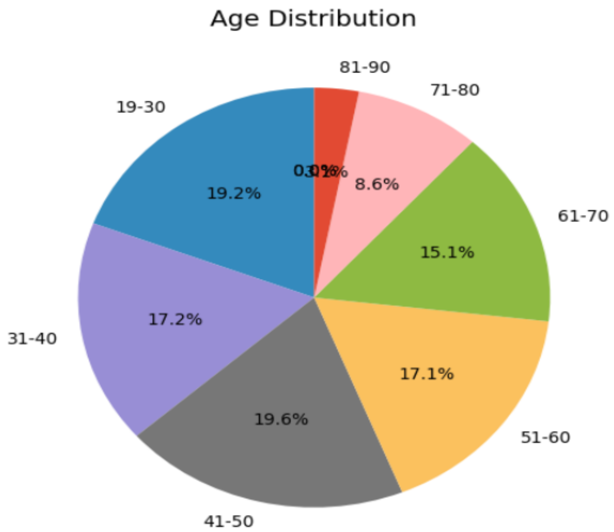
```python
df.drop("Customer ID", axis=1, inplace=True)
```

EDA

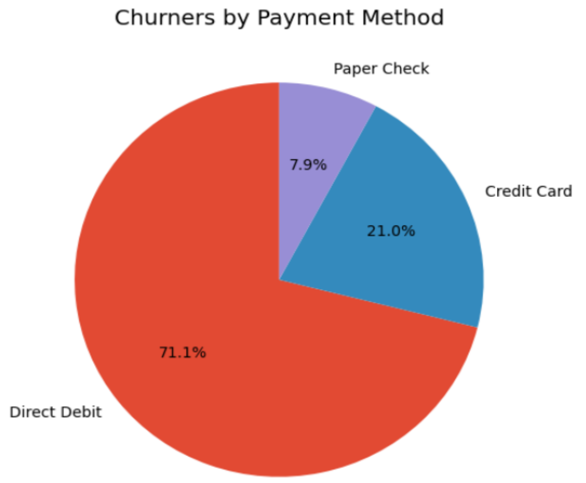
```
# Count the number of 'Yes' and 'No' in the 'Churn Label' column
churn_counts = df['Churn Label'].value_counts()

# Plot the pie chart
plt.figure(figsize=(8, 6))
churn_counts.plot.pie(autopct='%1.1f%%', startangle=90, colors=['lightblue', 'light
plt.title('Churn Label Distribution')
plt.ylabel('')
plt.show()
```



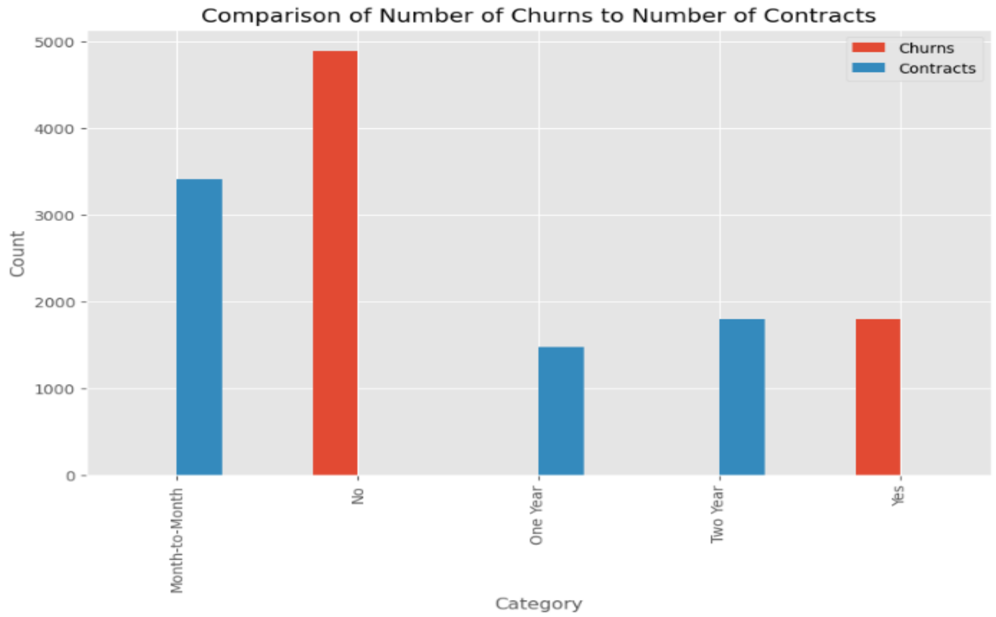

```
# Define age groups
bins = [0, 18, 30, 40, 50, 60, 70, 80, 90, 100]
labels = ['0-18', '19-30', '31-40', '41-50', '51-60', '61-70', '71-80', '81-90', '9

# Create age groups
df['Age Group'] = pd.cut(df['Age'], bins=bins, labels=labels, right=False)

# Count the number of customers in each age group
age_group_counts = df['Age Group'].value_counts().sort_index()

# Plot the pie chart for age distribution
plt.figure(figsize=(8, 6))
age_group_counts.plot.pie(labels=labels, autopct='%1.1f%%', startangle=90)
plt.title('Age Distribution')
plt.ylabel('')
plt.show()
```



```python
# Count the number of churners by payment method
churn_payment_counts = df[df['Churn Label'] == 'Yes']['Payment Method'].value_count

# Plot the pie chart
plt.figure(figsize=(8, 6))
churn_payment_counts.plot.pie(autopct='%1.1f%%', startangle=90)
plt.title('Churners by Payment Method')
plt.ylabel('')
plt.show()
```




```python
# Count the number of churns and total contracts
churn_counts = df['Churn Label'].value_counts()
contract_counts = df['Contract Type'].value_counts()

# Create a DataFrame for plotting
churn_contract_df = pd.DataFrame({
    'Churns': churn_counts,
    'Contracts': contract_counts
})


churn_contract_df.plot(kind='bar', figsize=(10, 6))
plt.title('Comparison of Number of Churns to Number of Contracts')
plt.xlabel('Category')
plt.ylabel('Count')
plt.show()
```



Data Preprocessing

```
df.shape
```

```
(6687, 29)
```

DUPLICATE DATA

```
# Check for duplicate rows
duplicate_rows = df.duplicated().sum()
print(f'Number of duplicate rows: {duplicate_rows}')
```

```
Number of duplicate rows: 0
```

Null DATA

```
# Check for null values
null_values = df.isnull().sum()
print(null_values)
```

```
Churn Label                               0
Account Length (in months)                0
Local Calls                               0
Local Mins                                0
Intl Calls                                0
Intl Mins                                 0
Intl Active                               0
Intl Plan                                 0
Extra International Charges               0
Customer Service Calls                    0
Avg Monthly GB Download                   0
Unlimited Data Plan                       0
Extra Data Charges                        0
State                                     0
Phone Number                              0
Gender                                    0
Age                                       0
Under 30                                  0
Senior                                    0
Group                                     0
Number of Customers in Group              0
Device Protection & Online Backup         0
Contract Type                             0
Payment Method                            0
Monthly Charge                            0
Total Charges                             0
Churn Category                         4918
Churn Reason                           4918
Age Group                                 0
dtype: int64
```

```
print(df['Churn Reason'].unique())
```

```
[nan 'Competitor made better offer' 'Moved'
 'Competitor had better devices'
 'Competitor offered higher download speeds' 'Attitude of support person'
 'Network reliability' "Don't know" 'Service dissatisfaction'
 'Product dissatisfaction' 'Poor expertise of online support'
 'Price too high' 'Limited range of services'
 'Lack of affordable download/upload speed' 'Long distance charges'
 'Competitor offered more data' 'Attitude of service provider'
 'Poor expertise of phone support' 'Extra data charges' 'Deceased'
 'Lack of self-service on Website']
```

```
# given it is a categorical column, with a lot of unique values, it is better to dr
df.drop('Churn Reason', axis=1, inplace=True)
```

DATA ENCODING

checking for unique values in each column

```
object_columns = df.select_dtypes(include=['object']).columns
for column in object_columns:
    print(f"{column} ({df[column].nunique()}): {df[column].unique()}")
```

```
Churn Label (2): ['No' 'Yes']
Intl Active (2): ['No' 'Yes']
Intl Plan (2): ['no' 'yes']
Unlimited Data Plan (2): ['Yes' 'No']
State (51): ['KS' 'OH' 'MO' 'WV' 'RI' 'IA' 'NY' 'ID' 'VT' 'TX' 'CO' 'SC' 'NE' 'IL'
 'NH' 'LA' 'AZ' 'OK' 'GA' 'MA' 'MD' 'AR' 'WI' 'OR' 'MI' 'WY' 'VA' 'CA'
 'MN' 'SD' 'WA' 'UT' 'NJ' 'NM' 'NV' 'DC' 'IN' 'KY' 'ME' 'MT' 'MS' 'AL'
 'FL' 'AK' 'DE' 'TN' 'NC' 'CT' 'PA' 'ND' 'HI']
Phone Number (6677): ['382-4657' '371-7191' '375-9999' ... '328-3647' '346-8275' '257
-5893']
Gender (3): ['Female' 'Male' 'Prefer not to say']
Under 30 (2): ['No' 'Yes']
Senior (2): ['No' 'Yes']
Group (2): ['No' 'Yes']
Device Protection & Online Backup (2): ['No' 'Yes']
Contract Type (3): ['Month-to-Month' 'One Year' 'Two Year']
Payment Method (3): ['Direct Debit' 'Paper Check' 'Credit Card']
Churn Category (5): [nan 'Competitor' 'Other' 'Attitude' 'Dissatisfaction' 'Price']
```

dropping Phone Number, since it is unique for each customer

```
df.drop('Phone Number', axis=1, inplace=True)
object_columns = object_columns.drop('Phone Number')
object_columns = object_columns.drop('Churn Label')
```

giving each state a number as in to not add 51 more columns during one hot encoding

```python
states = ['KS', 'OH', 'MO', 'WV', 'RI', 'IA', 'NY', 'ID', 'VT', 'TX',
          'CO', 'SC', 'NE', 'IL', 'NH', 'LA', 'AZ', 'OK', 'GA', 'MA',
          'MD', 'AR', 'WI', 'OR', 'MI', 'WY', 'VA', 'CA', 'MN', 'SD',
          'WA', 'UT', 'NJ', 'NM', 'NV', 'DC', 'IN', 'KY', 'ME', 'MT',
          'MS', 'AL', 'FL', 'AK', 'DE', 'TN', 'NC', 'CT', 'PA', 'ND', 'HI']

state_map = {state: idx + 1 for idx, state in enumerate(states)}

# Encode the State column
for i in range(len(df)):
    df.loc[i, 'State'] = state_map[df.loc[i, 'State']]

df.head()
object_columns = object_columns.drop('State')

```

```python
df['State'] = df['State'].astype('int64')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6687 entries, 0 to 6686
Data columns (total 27 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   Churn Label                         6687 non-null   object
 1   Account Length (in months)          6687 non-null   int64
 2   Local Calls                         6687 non-null   int64
 3   Local Mins                          6687 non-null   float64
 4   Intl Calls                          6687 non-null   float64
 5   Intl Mins                           6687 non-null   float64
 6   Intl Active                         6687 non-null   object
 7   Intl Plan                           6687 non-null   object
 8   Extra International Charges         6687 non-null   float64
 9   Customer Service Calls              6687 non-null   int64
 10  Avg Monthly GB Download             6687 non-null   int64
 11  Unlimited Data Plan                 6687 non-null   object
 12  Extra Data Charges                  6687 non-null   int64
 13  State                               6687 non-null   int64
 14  Gender                              6687 non-null   object
 15  Age                                 6687 non-null   int64
 16  Under 30                            6687 non-null   object
 17  Senior                              6687 non-null   object
 18  Group                               6687 non-null   object
 19  Number of Customers in Group        6687 non-null   int64
 20  Device Protection & Online Backup   6687 non-null   object
 21  Contract Type                       6687 non-null   object
 22  Payment Method                      6687 non-null   object
 23  Monthly Charge                      6687 non-null   int64
 24  Total Charges                       6687 non-null   int64
 25  Churn Category                      1769 non-null   object
 26  Age Group                           6687 non-null   category
dtypes: category(1), float64(4), int64(10), object(12)
memory usage: 1.3+ MB
```

Apply one-hot encoding to the remaining object columns

```
df = pd.get_dummies(df, columns=object_columns, drop_first=True)
```

```
pd.set_option('display.max_columns', None)
df.head(10)
```

	Churn Label	Account Length (in months)	Local Calls	Local Mins	Intl Calls	Intl Mins	Extra International Charges	Customer Service Calls	Avg Monthly GB Download	Extra Data Charges	State
0	No	1	3	8.0	0.0	0.0	0.0	0	3	0	1
1	No	33	179	431.3	0.0	0.0	0.0	0	3	0	2
2	No	44	82	217.6	0.0	0.0	0.0	0	3	0	2
3	No	10	47	111.6	60.0	71.0	0.0	0	2	0	3
4	No	62	184	621.2	310.0	694.4	0.0	0	3	0	4
5	No	17	68	120.7	0.0	0.0	0.0	0	0	0	5
6	No	57	428	849.2	0.0	0.0	0.0	0	5	0	6
7	No	25	54	203.7	0.0	0.0	0.0	0	12	0	6
8	No	70	171	627.4	0.0	0.0	0.0	0	1	0	7
9	No	50	206	445.8	0.0	0.0	0.0	0	0	0	8

Data Correlation

PCA (principal Component aanlysis)

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6687 entries, 0 to 6686
Data columns (total 33 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   Churn Label                             6687 non-null   object
 1   Account Length (in months)              6687 non-null   int64
 2   Local Calls                             6687 non-null   int64
 3   Local Mins                              6687 non-null   float64
 4   Intl Calls                              6687 non-null   float64
 5   Intl Mins                               6687 non-null   float64
 6   Extra International Charges             6687 non-null   float64
 7   Customer Service Calls                  6687 non-null   int64
 8   Avg Monthly GB Download                 6687 non-null   int64
 9   Extra Data Charges                      6687 non-null   int64
 10  State                                   6687 non-null   int64
 11  Age                                     6687 non-null   int64
 12  Number of Customers in Group            6687 non-null   int64
 13  Monthly Charge                          6687 non-null   int64
 14  Total Charges                           6687 non-null   int64
 15  Age Group                               6687 non-null   category
 16  Intl Active_Yes                         6687 non-null   bool
 17  Intl Plan_yes                           6687 non-null   bool
 18  Unlimited Data Plan_Yes                 6687 non-null   bool
 19  Gender_Male                             6687 non-null   bool
 20  Gender_Prefer not to say                6687 non-null   bool
 21  Under 30_Yes                            6687 non-null   bool
 22  Senior_Yes                              6687 non-null   bool
 23  Group_Yes                               6687 non-null   bool
 24  Device Protection & Online Backup_Yes   6687 non-null   bool
 25  Contract Type_One Year                  6687 non-null   bool
 26  Contract Type_Two Year                  6687 non-null   bool
 27  Payment Method_Direct Debit             6687 non-null   bool
 28  Payment Method_Paper Check              6687 non-null   bool
 29  Churn Category_Competitor               6687 non-null   bool
 30  Churn Category_Dissatisfaction          6687 non-null   bool
 31  Churn Category_Other                    6687 non-null   bool
 32  Churn Category_Price                    6687 non-null   bool
dtypes: bool(17), category(1), float64(4), int64(10), object(1)
memory usage: 901.7+ KB
```

```
from sklearn.decomposition import PCA

# Ensure all columns are numeric before applying StandardScaler
df_numeric = df.drop('Churn Label', axis=1).apply(pd.to_numeric, errors='coerce').f

# Standardize the data before applying PCA
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df_numeric)

# Apply PCA
pca = PCA(n_components=2)  # You can change the number of components as needed
principal_components = pca.fit_transform(df_scaled)

# Create a new dataframe with the principal components
pca_df = pd.DataFrame(data=principal_components, columns=['Principal Component 1',
pca_df['Churn Label'] = df['Churn Label'].values

print(pca_df.head())
```

```
   Principal Component 1  Principal Component 2 Churn Label
0              -2.639666               0.261894          No
1              -0.408363               0.476147          No
2              -0.227758              -0.498509          No
3              -1.936924               0.629647          No
4               2.641583              -0.451409          No
```

Multicolenearity

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Calculate VIF for each feature
vif_data = pd.DataFrame()
vif_data["feature"] = df.drop('Churn Label', axis=1).columns
vif_data["VIF"] = [variance_inflation_factor(df_scaled, i) for i in range(df_scaled

print(vif_data)
```

```
                               feature        VIF
0             Account Length (in months)   8.665392
1                            Local Calls  12.810502
2                             Local Mins  15.439205
3                             Intl Calls   3.965186
4                              Intl Mins   6.981950
5            Extra International Charges   3.371074
6                 Customer Service Calls   1.502761
7                Avg Monthly GB Download   1.733181
8                     Extra Data Charges   1.543250
9                                  State   1.005854
10                                   Age   4.287617
11          Number of Customers in Group   5.715883
12                        Monthly Charge   4.473452
13                         Total Charges   8.463103
14                             Age Group        NaN
15                       Intl Active_Yes   2.428988
16                         Intl Plan_yes   1.458218
17                Unlimited Data Plan_Yes   2.328113
18                           Gender_Male   1.002802
19                 Gender_Prefer not to say   1.004238
20                          Under 30_Yes   2.426679
21                            Senior_Yes   2.572160
22                             Group_Yes   5.923754
23  Device Protection & Online Backup_Yes   1.445963
24                     Contract Type_One Year   1.550508
25                     Contract Type_Two Year   2.230302
26             Payment Method_Direct Debit   1.228691
27              Payment Method_Paper Check   1.113909
28              Churn Category_Competitor   1.497975
29          Churn Category_Dissatisfaction   1.201076
30                   Churn Category_Other   1.131634
31                   Churn Category_Price   1.137175
```

```
# Calculate VIF for each feature
vif_data = pd.DataFrame()
vif_data["feature"] = df.drop('Churn Label', axis=1).columns
vif_data["VIF"] = [variance_inflation_factor(df_scaled, i) for i in range(df_scalec

# Identify features with high VIF
high_vif_features = vif_data[vif_data["VIF"] > 10]["feature"].tolist()

df_reduced = df.drop(columns=high_vif_features)

print("Features with high VIF removed:", high_vif_features)
print("Remaining features:", df_reduced.columns)
```

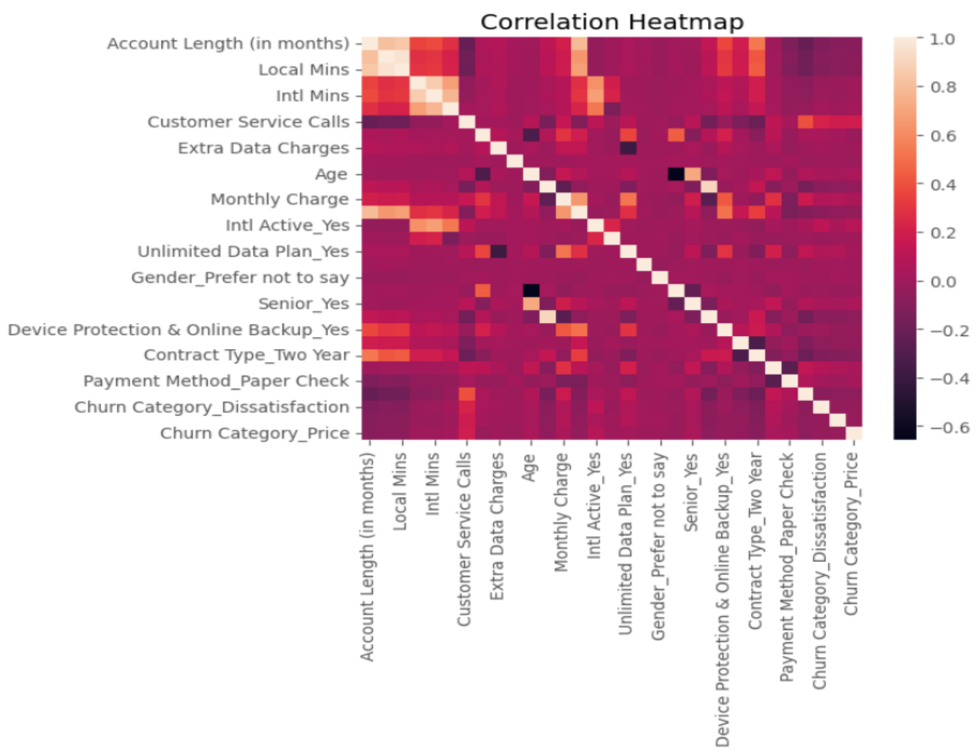
```
Features with high VIF removed: ['Local Calls', 'Local Mins']
Remaining features: Index(['Churn Label', 'Account Length (in months)', 'Intl Call
s', 'Intl Mins',
       'Extra International Charges', 'Customer Service Calls',
       'Avg Monthly GB Download', 'Extra Data Charges', 'State', 'Age',
       'Number of Customers in Group', 'Monthly Charge', 'Total Charges',
       'Age Group', 'Intl Active_Yes', 'Intl Plan_yes',
       'Unlimited Data Plan_Yes', 'Gender_Male', 'Gender_Prefer not to say',
       'Under 30_Yes', 'Senior_Yes', 'Group_Yes',
       'Device Protection & Online Backup_Yes', 'Contract Type_One Year',
       'Contract Type_Two Year', 'Payment Method_Direct Debit',
       'Payment Method_Paper Check', 'Churn Category_Competitor',
       'Churn Category_Dissatisfaction', 'Churn Category_Other',
       'Churn Category_Price'],
      dtype='object')
```

```
df.shape
```

```
(6687, 33)
```

Heatmap

```
# Drop the 'Age Group' column before calculating the correlation matrix
correlation_matrix = df.drop(['Churn Label', 'Age Group'], axis=1).corr()
sns.heatmap(correlation_matrix, fmt = '.2f')
plt.title('Correlation Heatmap')
plt.show()
```



DIVINDING DATA INTO TRAINING AND TESING

```
X = df_reduced.drop('Churn Label', axis=1)
y = df_reduced['Churn Label']

# Ensure all columns are numeric
X = X.apply(pd.to_numeric, errors='coerce').fillna(0)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Testing the model

KNN

```
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier()  # KNN classifier

knn_model.fit(X_train_scaled, y_train)
y_pred = knn_model.predict(X_test_scaled)

```

```
knn_accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", knn_accuracy)

```

```
Accuracy: 0.9603886397608371
```

DecisionTree

```
from sklearn.tree import DecisionTreeClassifier
decision_tree_model = DecisionTreeClassifier()  # Decision tree classifier

decision_tree_model.fit(X_train_scaled, y_train)
y_pred = decision_tree_model.predict(X_test_scaled)

```

```
decision_tree_accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", decision_tree_accuracy)


```

```
Accuracy: 0.9417040358744395
```

LogisticRegression

```
from sklearn.linear_model import LogisticRegression
Logistic_model = LogisticRegression()

Logistic_model.fit(X_train_scaled, y_train)
y_pred = Logistic_model.predict(X_test_scaled)
```

```
Logistic_accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", Logistic_accuracy)
```

```
Accuracy: 0.9693572496263079
```

RANDOM FORREST

```
from sklearn.ensemble import RandomForestClassifier
Random_Forest_model = RandomForestClassifier()
Random_Forest_model.fit(X_train_scaled, y_train)
y_pred = Random_Forest_model.predict(X_test_scaled)

```

```
Random_Forest_accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", Random_Forest_accuracy)

```

```
Accuracy: 0.9708520179372198
```

SVM

```
from sklearn.svm import SVC
SVM_model = SVC()  # Support Vector Classifier

SVM_model.fit(X_train_scaled, y_train)
y_pred = SVM_model.predict(X_test_scaled)

```

```
SVM_accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", SVM_accuracy)

```

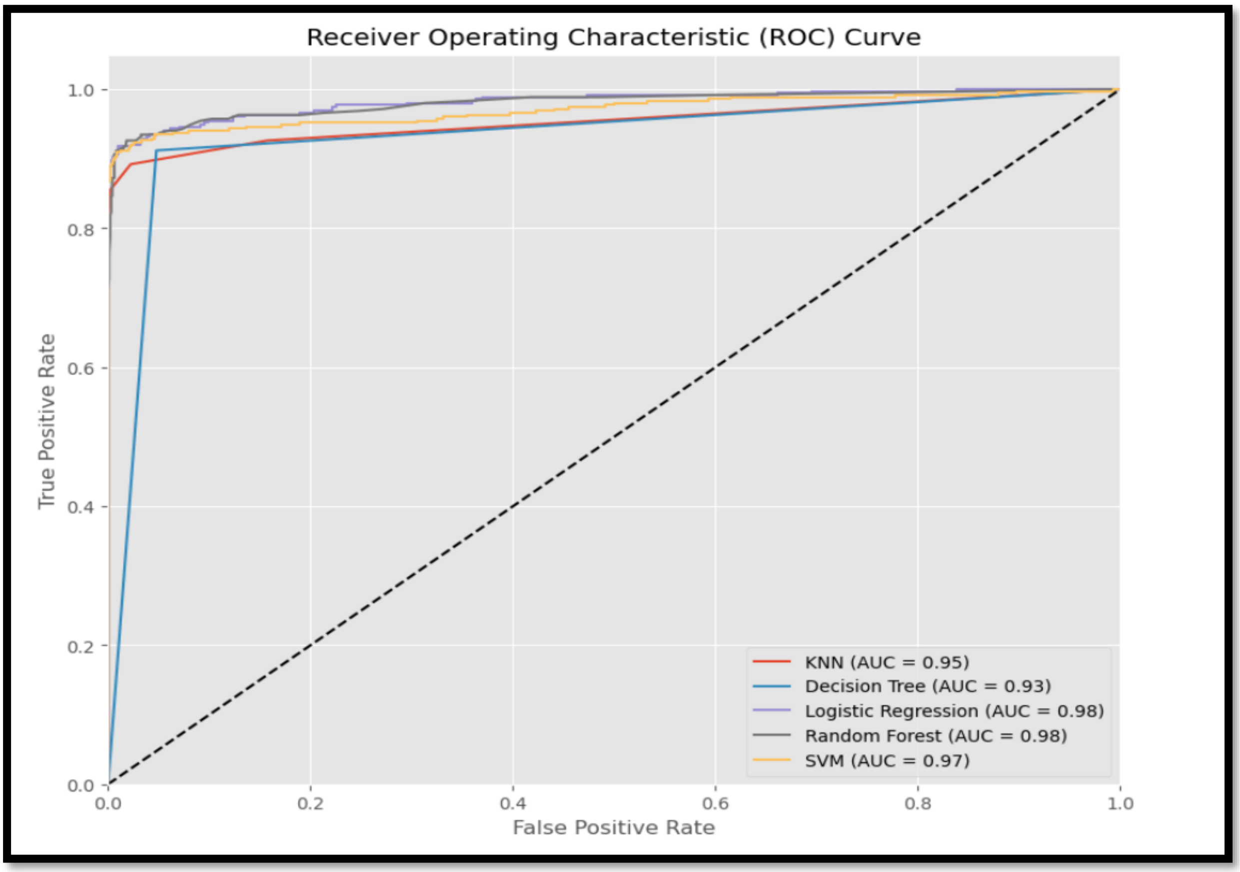
```
Accuracy: 0.9656203288490284
```

COMPARING THE MODELS

```
accuracy_dict = {
    "Random Forest": Random_Forest_accuracy,
    "Decision Tree Classifier": decision_tree_accuracy,
    "K-Nearest Neighbors (KNN)": knn_accuracy,
    "Support Vector Machine (SVM)": SVM_accuracy,
    "Logistic Regression": Logistic_accuracy,
}

```

ROC Curve ¶

```python
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

y_test_binary = y_test.map({'No': 0, 'Yes': 1}).values

models = {
    "KNN": knn_model,
    "Decision Tree": decision_tree_model,
    "Logistic Regression": Logistic_model,
    "Random Forest": Random_Forest_model,
    "SVM": SVM_model,
}

plt.figure(figsize=(10, 8))

for model_name, model in models.items():
    try:
        if hasattr(model, "predict_proba"):
            y_score = model.predict_proba(X_test_scaled)[:, 1]
        elif hasattr(model, "decision_function"):
            y_score = model.decision_function(X_test_scaled)
        else:
            y_score = model.predict(X_test_scaled)

        fpr, tpr, _ = roc_curve(y_test_binary, y_score)
        roc_auc = roc_auc_score(y_test_binary, y_score)

        plt.plot(fpr, tpr, label=f'{model_name} (AUC = {roc_auc:.2f})')
    except Exception as e:
        print(f"Error with model {model_name}: {e}")

# Plot diagonal
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

```



F1 Scores

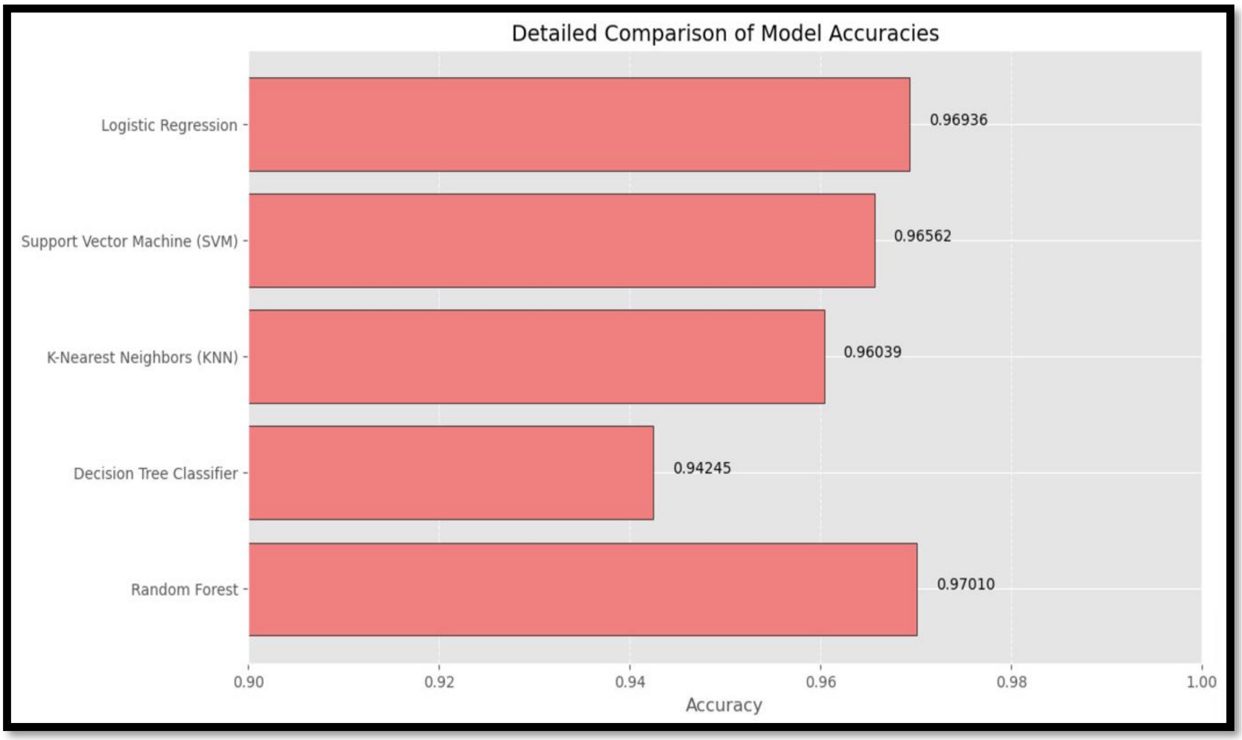
```python
from sklearn.metrics import f1_score

# Calculate F1 scores
f1_scores = {
    "Random Forest": f1_score(y_test, Random_Forest_model.predict(X_test_scaled), p
    "Decision Tree Classifier": f1_score(y_test, decision_tree_model.predict(X_test
    "K-Nearest Neighbors (KNN)": f1_score(y_test, knn_model.predict(X_test_scaled),
    "Support Vector Machine (SVM)": f1_score(y_test, SVM_model.predict(X_test_scale
    "Logistic Regression": f1_score(y_test, Logistic_model.predict(X_test_scaled),
}

# Print F1 scores
for model_name, f1 in f1_scores.items():
    print(f"{model_name}: {f1:.5f}")
```

```
Random Forest: 0.94273
Decision Tree Classifier: 0.89167
K-Nearest Neighbors (KNN): 0.91908
Support Vector Machine (SVM): 0.93051
Logistic Regression: 0.93944
```

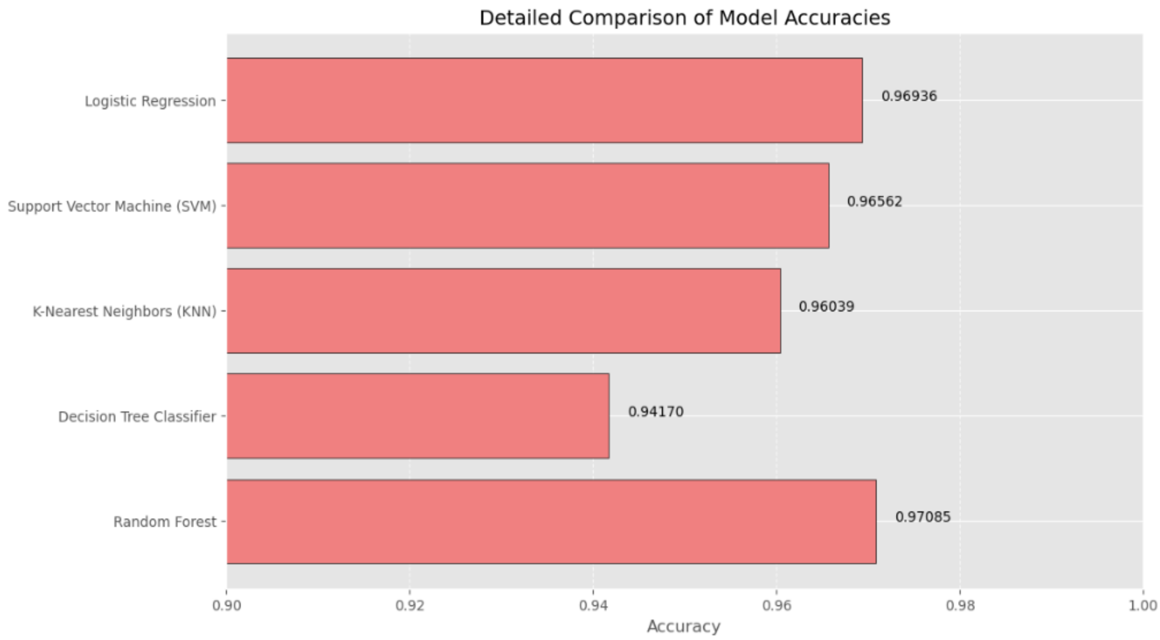
Comparng Accuracies

```python
best_model_name = max(accuracy_dict, key=accuracy_dict.get)
best_model_value = accuracy_dict[best_model_name]

print(f"The best model is {best_model_name} with an accuracy of {best_model_value}"
```

```
The best model is Random Forest with an accuracy of 0.9708520179372198
```



```python

model_names = list(accuracy_dict.keys())
accuracies = list(accuracy_dict.values())

plt.figure(figsize=(12, 7))
bars = plt.barh(model_names, accuracies, color='lightcoral', edgecolor='black')
for bar, accuracy in zip(bars, accuracies):
    plt.text(
        bar.get_width() + 0.002,
        bar.get_y() + bar.get_height() / 2,
        f"{accuracy:.5f}"
    )

plt.xlabel('Accuracy')
plt.title('Detailed Comparison of Model Accuracies')
plt.xlim(0.90, 1)  # Zoom in on the specific range
plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.show()
```



7. Conclusion and Future Scope

7.1 Conclusion

This project successfully applied machine learning techniques to predict customer churn with high accuracy. Among the five models tested, **Random Forest** emerged as the best-performing algorithm, achieving an accuracy of **97.01%**. Key insights from the analysis highlight that features such as **Monthly Charges**, **Contract Type**, and **Payment Method** play a significant role in predicting churn. These insights can help businesses design targeted retention strategies.

The results demonstrate that machine learning can provide actionable and data-driven solutions to minimize customer churn, thereby improving business profitability and customer satisfaction.

Key takeaways include:

- **Data Preprocessing and Cleaning**: Addressing missing values, irrelevant data, and outliers ensured a high-quality dataset ready for analysis.
- **Exploratory Data Analysis (EDA)**: Visualizations like pie charts, histograms, and heatmaps facilitated a clearer understanding of data distribution and feature relationships, influencing model decisions.
- **Multicollinearity Check**: By assessing the Variance Inflation Factor (VIF), features with high multicollinearity were identified and adjusted, improving the stability of the predictive model.
- **Feature Scaling and Model Training**: Standardization of features through scaling was crucial in maintaining model performance. The classification model was able to achieve good accuracy and reliability through comprehensive training and evaluation.
- **Model Validation**: Cross-validation ensured that the model maintained consistent performance across various data splits, indicating its robustness.

7.2 Future Scope

1. **Enhancing Data Quality**:
    - Incorporate additional customer data, such as sentiment analysis from customer reviews or social media interactions, for improved predictions.
    - Handle class imbalance more effectively by exploring advanced resampling techniques like SMOTE (Synthetic Minority Oversampling Technique).
2. **Deploying Predictive Models**:
    - Integrate the model into a live system for real-time churn prediction.
    - Build dashboards for decision-makers to visualize trends and take proactive actions.
3. **Exploring Advanced Algorithms**:
    - Experiment with ensemble methods like Gradient Boosting Machines (e.g., XGBoost, LightGBM) or neural networks for potential performance improvement.
    - Fine-tune hyperparameters using automated tools like GridSearchCV or Optuna.
4. **Cost-Based Analysis**:
    - Incorporate cost-sensitive learning to balance the business impact of false positives (retaining non-churners) versus false negatives (losing churners).
5. **Scalability and Automation**:
    - Automate the end-to-end workflow using tools like Apache Airflow.
    - Ensure scalability to handle larger datasets and complex features.
6. **Customer Retention Strategies**:
    - Translate model insights into actionable strategies, such as personalized offers, loyalty programs, or improved customer support, based on the reasons for churn.

By leveraging these advancements, the project can evolve into a robust system that not only predicts churn but also actively prevents it, driving long-term value for businesses.