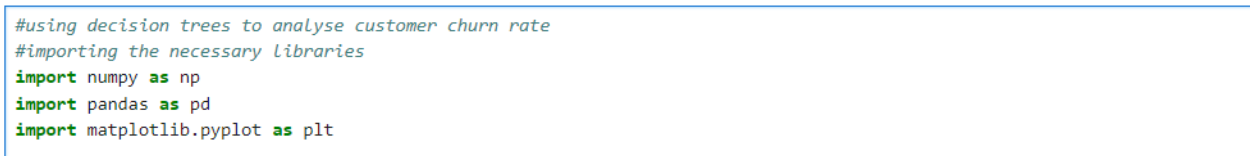
Customer churn rate analysis

Objective : To understand the churn rate of customers based on other factors

Algorithms used: Used decision tree initially and checked the accuracy and other factors , however the pictorial representation of tree was not very informative due to consideration of multiple features. Therefore proceeded to use a random forest and logistic regression to understand the important features.

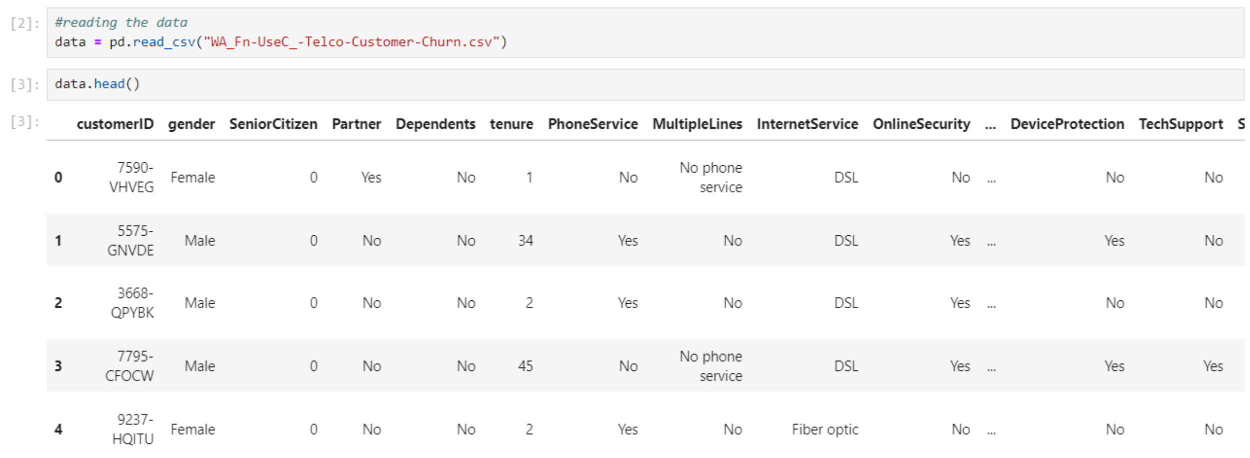
Steps involved

1)importing all the libraries



```
#using decision trees to analyse customer churn rate
#importing the necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

2)importing data and reading data for first 5 rows



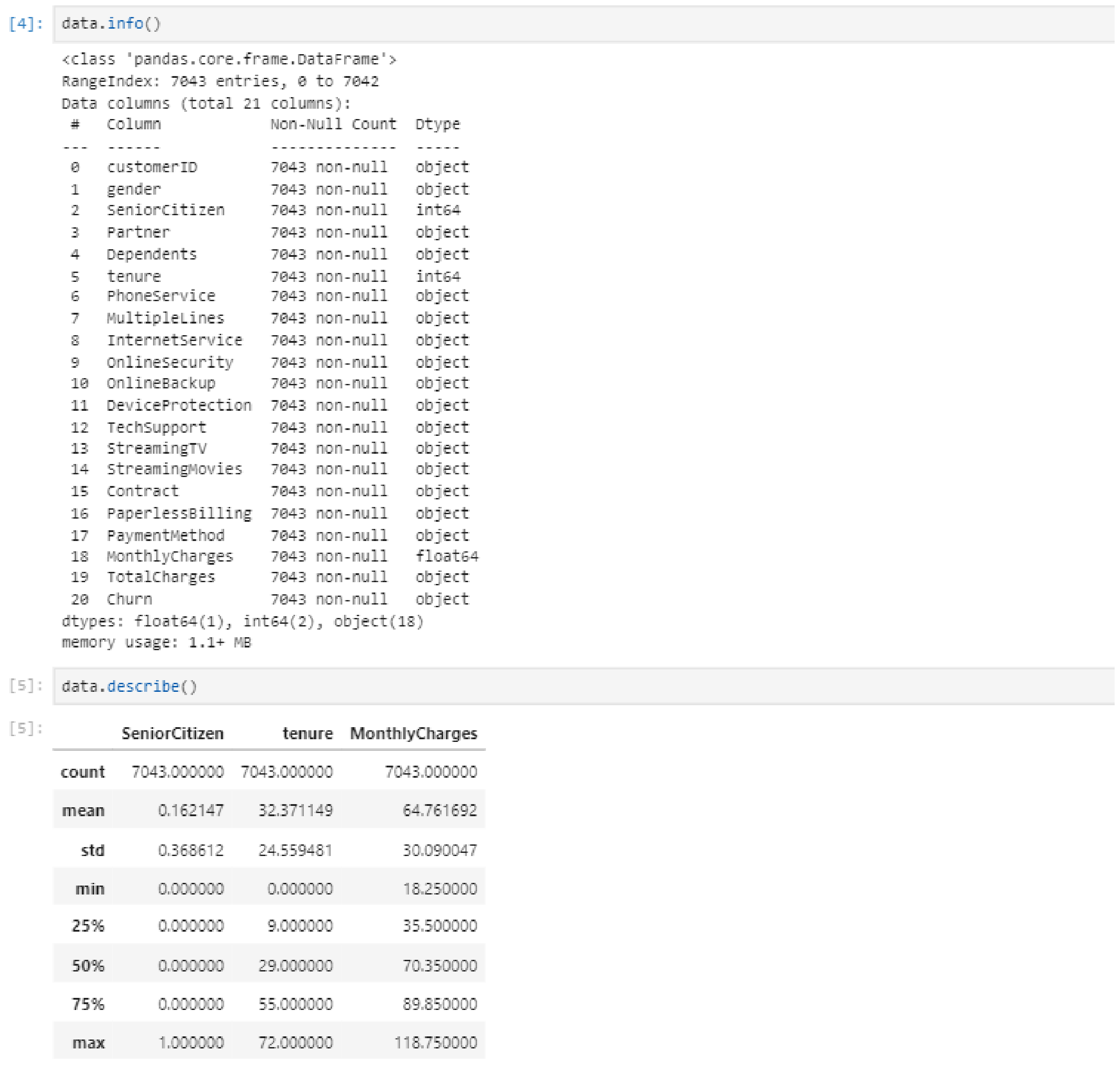
```
[2]: #reading the data
     data = pd.read_csv("WA_Fn-UseC_-Telco-Customer-Churn.csv")
```

```
[3]: data.head()
```

[3]:

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

3)checking for null values as well the statistical parameters of the data , using functions info and describe



```
[4]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        7043 non-null   object
 1   gender            7043 non-null   object
 2   SeniorCitizen     7043 non-null   int64
 3   Partner           7043 non-null   object
 4   Dependents        7043 non-null   object
 5   tenure            7043 non-null   int64
 6   PhoneService      7043 non-null   object
 7   MultipleLines     7043 non-null   object
 8   InternetService   7043 non-null   object
 9   OnlineSecurity    7043 non-null   object
 10  OnlineBackup      7043 non-null   object
 11  DeviceProtection  7043 non-null   object
 12  TechSupport       7043 non-null   object
 13  StreamingTV       7043 non-null   object
 14  StreamingMovies   7043 non-null   object
 15  Contract          7043 non-null   object
 16  PaperlessBilling  7043 non-null   object
 17  PaymentMethod     7043 non-null   object
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object
 20  Churn             7043 non-null   object
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

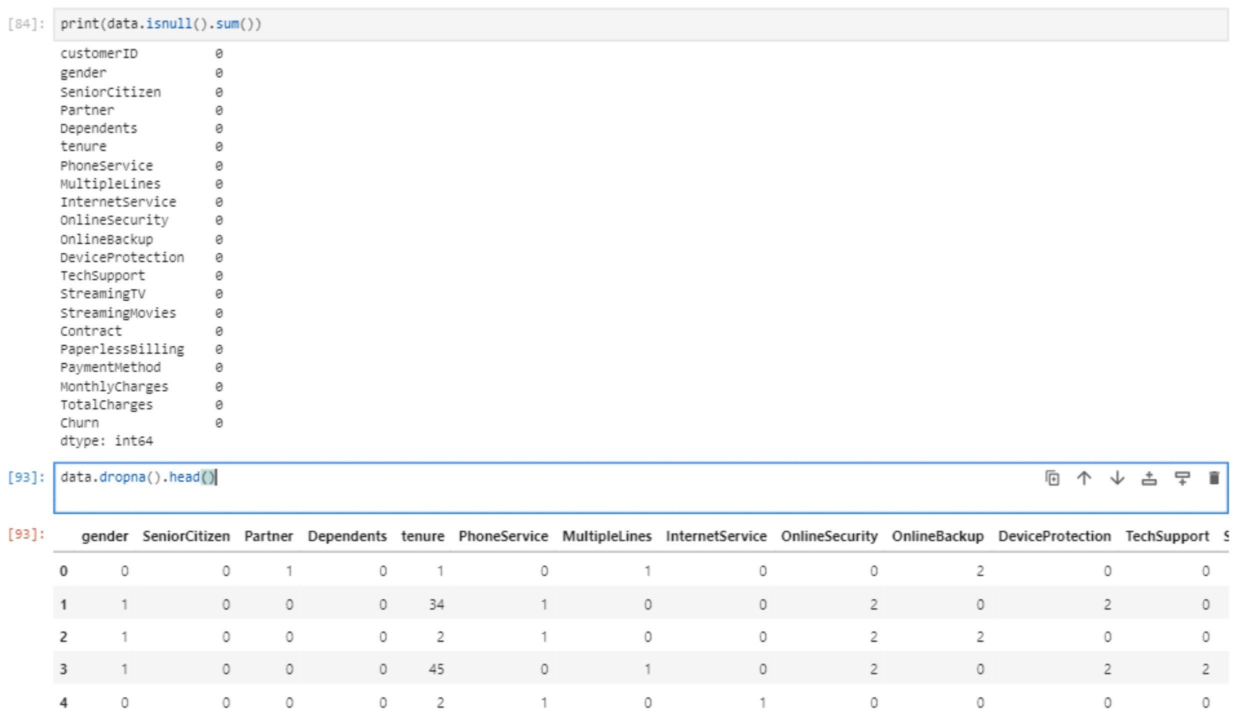
```
[5]: data.describe()
```

[5]:

	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000
50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000

statistical values like the standard deviation and inter quartile ranges can help us understand the spread of the data and hence understand the range and outliers.

4)checking for null values and removing all the NA values



```
[84]: print(data.isnull().sum())

customerID          0
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

```
[93]: data.dropna().head()
```

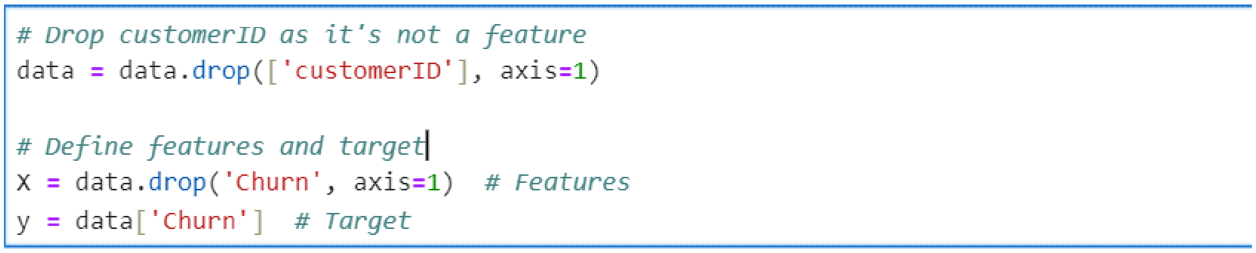
[93]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport
0	0	0	1	0	1	0	1	0	0	2	0	0
1	1	0	0	0	34	1	0	0	2	0	2	0
2	1	0	0	0	2	1	0	0	2	2	0	0
3	1	0	0	0	45	0	1	0	2	0	2	2
4	0	0	0	0	2	1	0	1	0	0	0	0

5)label encoder gives unique numerical values to different strings in the data set

```python
#Label encoder to convert non numerical data to numerical data

from sklearn.preprocessing import LabelEncoder

label_encoders = {}
#selecting the columns with string datas instead of numerical data

categorical_columns = data.select_dtypes(include =['object']).columns
for columns in categorical_columns:
    if columns != 'customerID':
        label_encoders[columns] = LabelEncoder()
        data[columns] = label_encoders[columns].fit_transform(data[columns])
```

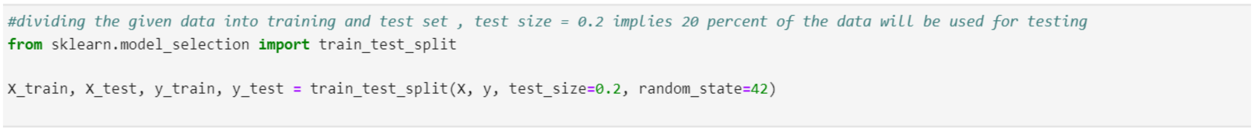
6) dropping the unnecessary columns



```python
# Drop customerID as it's not a feature
data = data.drop(['customerID'], axis=1)

# Define features and target
X = data.drop('Churn', axis=1)  # Features
y = data['Churn']  # Target
```

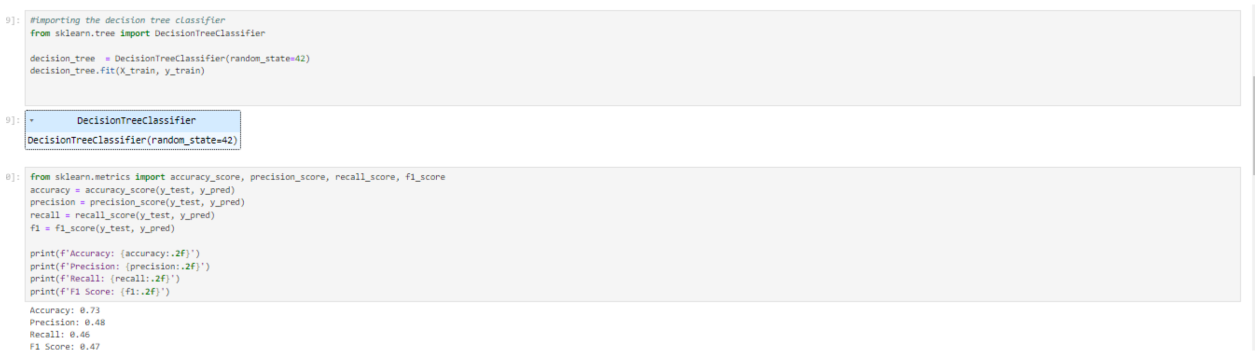
All this comes under cleaning and processing the data , the next step is to train the algorithm with data and so the data needs to be split into two sets , test set and training set

7)Importing the train_test_split function from sklearn.model_selection library



```python
#dividing the given data into training and test set , test size = 0.2 implies 20 percent of the data will be used for testing
from sklearn.model_selection import train_test_split

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

8)training the decision tree on the given data and checking the parameters to understand how effectively the algorithm works

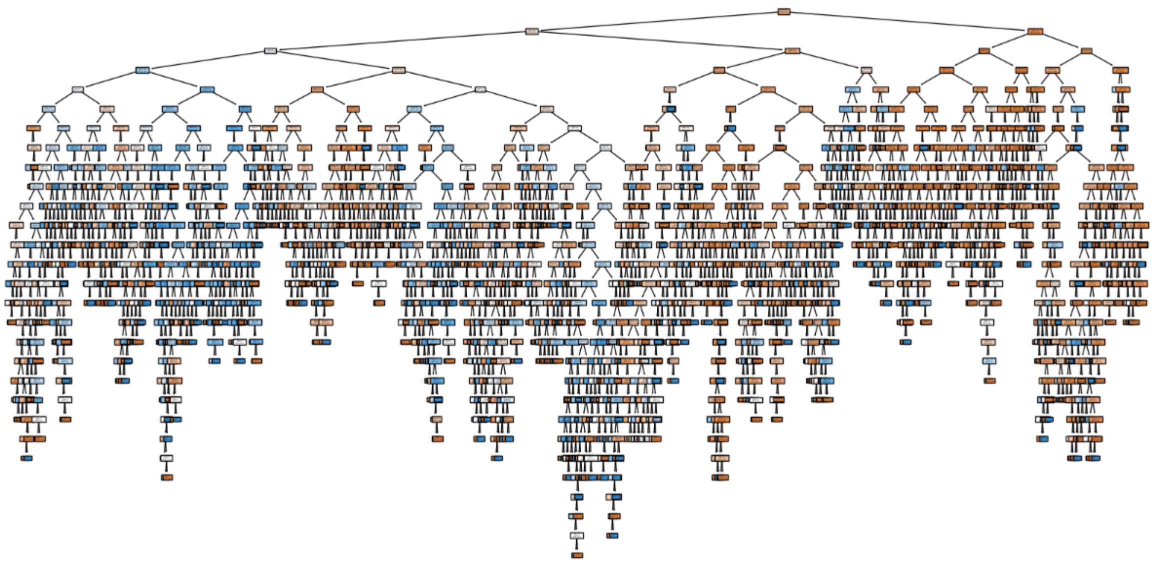


```
9]: #importing the decision tree classifier
    from sklearn.tree import DecisionTreeClassifier

    decision_tree  = DecisionTreeClassifier(random_state=42)
    decision_tree.fit(X_train, y_train)

9]: ▾     DecisionTreeClassifier
    DecisionTreeClassifier(random_state=42)

0]: from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    print(f'Accuracy: {accuracy:.2f}')
    print(f'Precision: {precision:.2f}')
    print(f'Recall: {recall:.2f}')
    print(f'F1 Score: {f1:.2f}')

    Accuracy: 0.73
    Precision: 0.48
    Recall: 0.46
    F1 Score: 0.47
```

CHECKING THE MODEL AND HOW EFFECTIVE IT IS

- **Accuracy: 0.73**: The model correctly predicts 73% of the total instances.
- **Precision: 0.48**: Of all the positive predictions, 48% are correct.
- **Recall: 0.46**: The model identifies 46% of the actual positive cases.

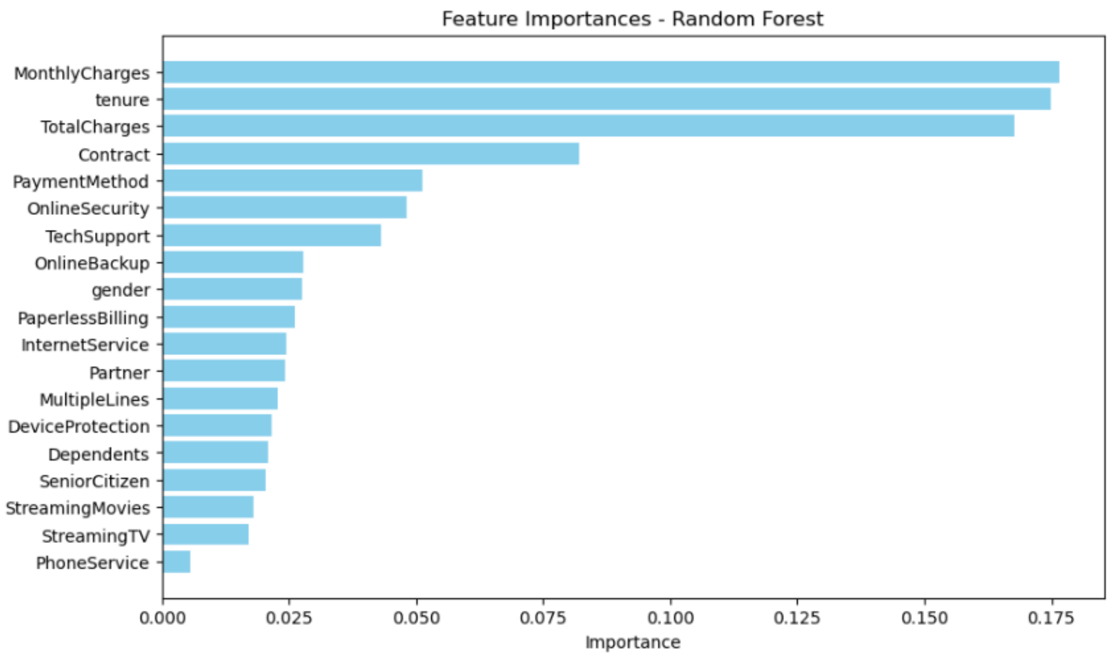
- **F1 Score: 0.47**: The model's balance between precision and recall is moderate, indicating a need for improvement in both.



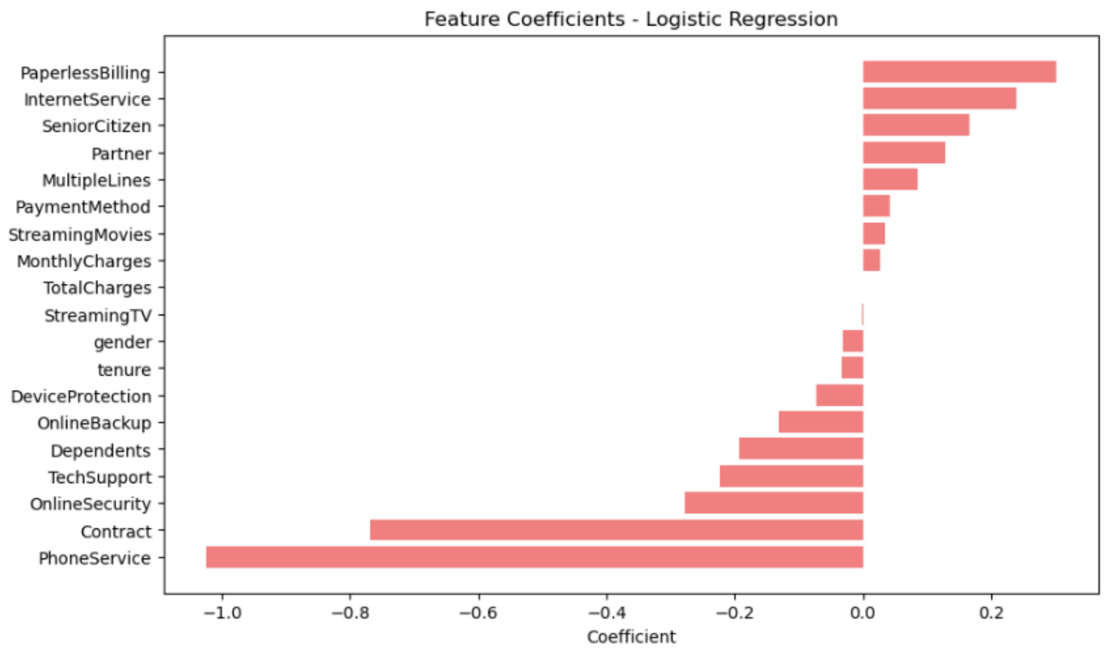
This is the visual representation of the decision tree and it is hard to draw conclusions of effects of different parameters from this. Therefore tried using random forests and logistic regression to understand the effect and importance of the other parameters.

RANDOM FOREST MODEL AND LOGISTIC REGRESSION

```
# Random Forest Model
rf_clf = RandomForestClassifier(random_state=42)
rf_clf.fit(X_train, y_train)
y_pred_rf = rf_clf.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
precision_rf = precision_score(y_test, y_pred_rf)
recall_rf = recall_score(y_test, y_pred_rf)
f1_rf = f1_score(y_test, y_pred_rf)
print(f"Random Forest- \nAccuracy: {accuracy_rf:.2f}\nPrecision: {precision_rf:.2f}\nRecall: {recall_rf:.2f}\nF1 Score: {f1_rf:.2f}")

# importances of different features in the data
importances = rf_clf.feature_importances_
feature_names = X_train.columns
importance_data = pd.DataFrame({'Feature': feature_names, 'Importance': importances}).sort_values(by='Importance', ascending=False)

# Plot feature importances
plt.figure(figsize=(10, 6))
plt.barh(importance_data['Feature'], importance_data['Importance'], color='skyblue')
plt.xlabel('Importance')
plt.title('Feature Importances - Random Forest')
plt.gca().invert_yaxis()  # Invert y-axis to have the most important feature at the top
plt.show()

# Logistic Regression Model
lr_clf = LogisticRegression(max_iter=1000, random_state=42)
lr_clf.fit(X_train, y_train)
y_pred_lr = lr_clf.predict(X_test)
accuracy_lr = accuracy_score(y_test, y_pred_lr)
precision_lr = precision_score(y_test, y_pred_lr)
recall_lr = recall_score(y_test, y_pred_lr)
f1_lr = f1_score(y_test, y_pred_lr)
print(f"Logistic Regression - \nAccuracy: {accuracy_lr:.2f} \nPrecision: {precision_lr:.2f}\nRecall: {recall_lr:.2f} \nF1 Score: {f1_lr:.2f}\n")

# Coefficients for Logistic Regression
coefficients = pd.DataFrame({'Feature': feature_names, 'Coefficient': lr_clf.coef_[0]}).sort_values(by='Coefficient', ascending=False)

# Plot coefficients
plt.figure(figsize=(10, 6))
plt.barh(coefficients['Feature'], coefficients['Coefficient'], color='lightcoral')
plt.xlabel('Coefficient')
plt.title('Feature Coefficients - Logistic Regression')
plt.gca().invert_yaxis()  # Invert y-axis to have the highest coefficient at the top
plt.show()
```



This shows that the monthly charges have the most importance in the purchase decision i.e. churn rate . The importance of features is plotted in decreasing order. This helps us understand which parameters could be dropped while training if the data set is very large.



the coefficient shows the strength of the relationship between feature and target variable.

coefficient of 0.2 indicates - odds ratio = exp(0.2) which is approximately 1.22 which means there is a slight increase in the odds of churn. if this is negative it indicates with an increase in that particular attribute the churn rate decreases.