Professor: Giacomo Fiumara

Student: Arya Khosravirad

Matricola: 534 170

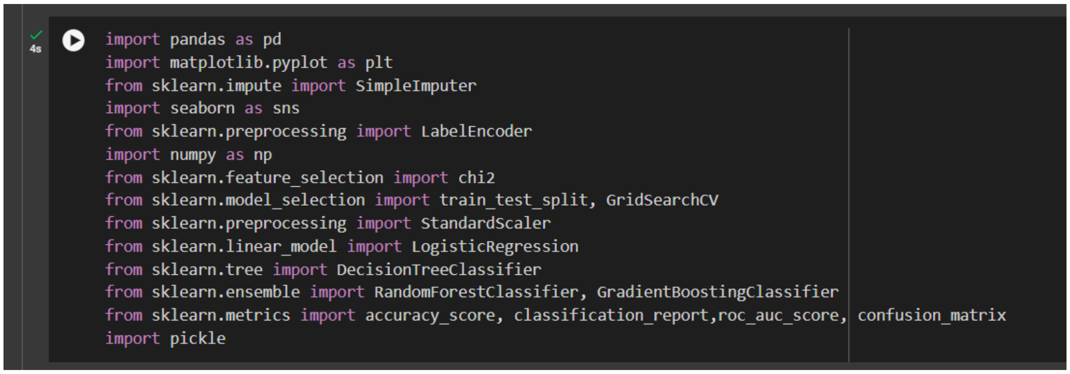
Università degli Studi di Messina

Table of content

1. Introduction

Customer churn is when people stop using a company's service. This is a big problem for businesses because it means losing customers. If a company can predict which customers might churn they can try to keep them before they go. In this project we use machine learning to look at customer data like how much they pay each month and the services they use to predict who might leave. This way, businesses can act early to keep more customers and reduce losses.

We used the following python libraries to do this:

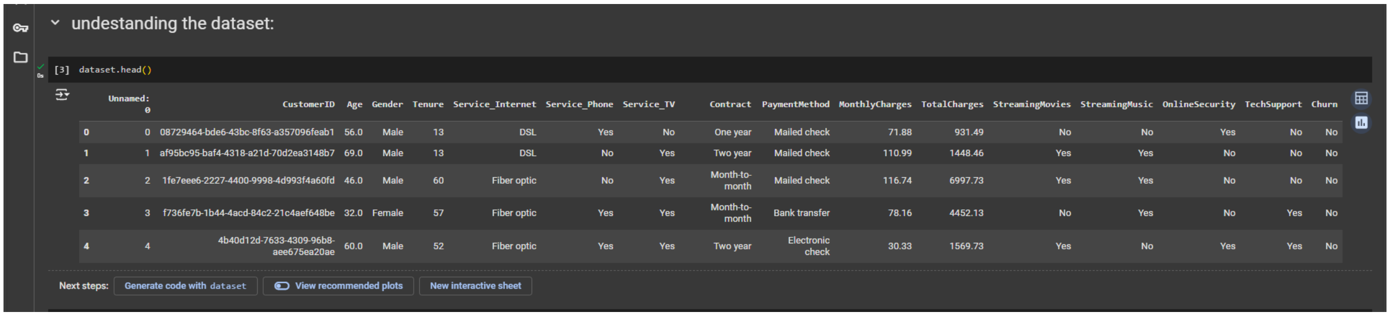


```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
import numpy as np
from sklearn.feature_selection import chi2
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report,roc_auc_score, confusion_matrix
import pickle
```

2.Understanding the Dataset

1. **Viewing the First Few Rows:**

   I started by using dataset.head() to display the first few rows of the data to get a quick look at the structure of the dataset.



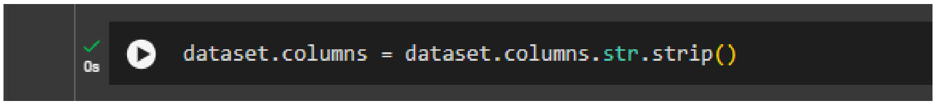
undestanding the dataset:

```
[3] dataset.head()
```

	Unnamed: 0	CustomerID	Age	Gender	Tenure	Service_Internet	Service_Phone	Service_TV	Contract	PaymentMethod	MonthlyCharges	TotalCharges	StreamingMovies	StreamingMusic	OnlineSecurity	TechSupport	Churn
0	0	08729464-bde6-43bc-8f63-a357096feab1	56.0	Male	13	DSL	Yes	No	One year	Mailed check	71.88	931.49	No	No	Yes	No	No
1	1	af95bc95-baf4-4318-a21d-70d2ea3148b7	69.0	Male	13	DSL	No	Yes	Two year	Mailed check	110.99	1448.46	Yes	Yes	No	No	No
2	2	1fe7eee6-2227-4400-9998-4d993f4a60fd	46.0	Male	60	Fiber optic	No	Yes	Month-to-month	Mailed check	116.74	6997.73	Yes	Yes	No	No	No
3	3	f736fe7b-1b44-4acd-84c2-21c4aef648be	32.0	Female	57	Fiber optic	Yes	Yes	Month-to-month	Bank transfer	78.16	4452.13	No	Yes	No	Yes	No
4	4	4b40d12d-7633-4309-96b8-aee675ea20ae	60.0	Male	52	Fiber optic	Yes	Yes	Two year	Electronic check	30.33	1569.73	Yes	No	Yes	Yes	No

2. **Cleaning the Column Names:**

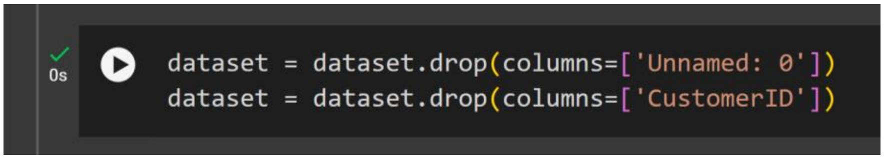
   then I removed any leading or trailing spaces. to avoid any potential issues for the next step.



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

3. **Removing Unnecessary Columns:**

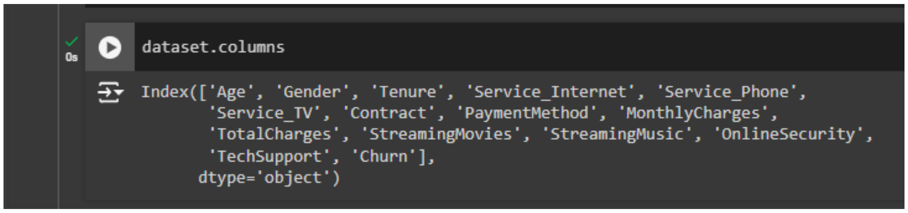
   I removed two columns Unnamed: 0 and CustomerID. The Unnamed: 0 column was just an index and the CustomerID column was a unique identifier for each customer so they did not have any useful information so I removed them.



```
dataset = dataset.drop(columns=['Unnamed: 0'])
dataset = dataset.drop(columns=['CustomerID'])
```

4. **Verifying the Column Names:**

   After cleaning and dropping columns I used dataset.columns to check the list of columns that remained and verify that columns are removed successfully.

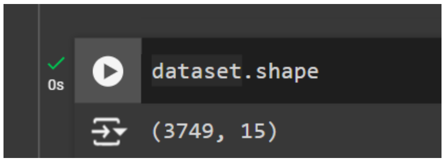


```
dataset.columns
```

```
Index(['Age', 'Gender', 'Tenure', 'Service_Internet', 'Service_Phone',
       'Service_TV', 'Contract', 'PaymentMethod', 'MonthlyCharges',
       'TotalCharges', 'StreamingMovies', 'StreamingMusic', 'OnlineSecurity',
       'TechSupport', 'Churn'],
      dtype='object')
```

5. **Checking the Dataset's Dimensions:**

   Then I checked the shape of the dataset to see how many rows and columns were left after dropping the columns.



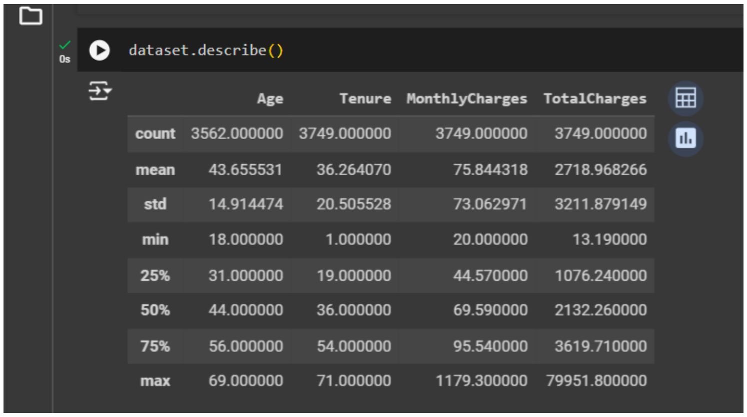
```
dataset.shape
```

```
(3749, 15)
```

6. **Summarizing the Data:**

   I used dataset.describe() to generate summary statistics for the numerical columns. These statistics gave me a better understanding of the overall distribution and variability of the data.

   The number of counts in Age column is different from others this indicate the possibility of missing values



```
dataset.describe()
```

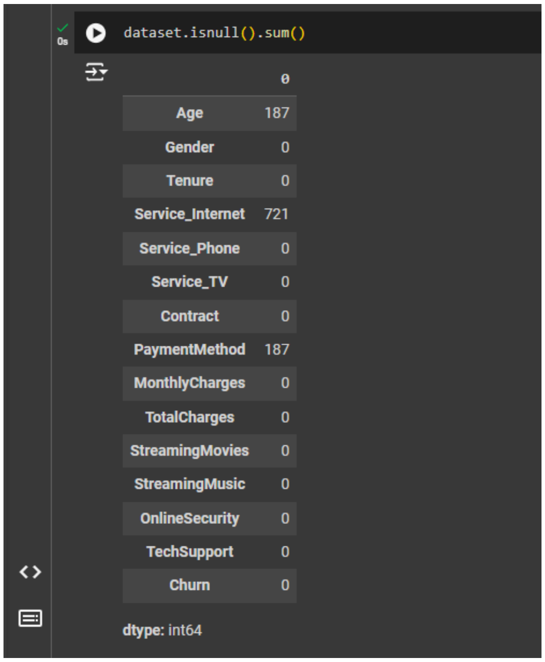
	Age	Tenure	MonthlyCharges	TotalCharges
count	3562.000000	3749.000000	3749.000000	3749.000000
mean	43.655531	36.264070	75.844318	2718.968266
std	14.914474	20.505528	73.062971	3211.879149
min	18.000000	1.000000	20.000000	13.190000
25%	31.000000	19.000000	44.570000	1076.240000
50%	44.000000	36.000000	69.590000	2132.260000
75%	56.000000	54.000000	95.540000	3619.710000
max	69.000000	71.000000	1179.300000	79951.800000

3.Data Processing

First we are going to check missing values in the dataset handling missing values are important because If the missing values are not random they can introduce bias and some machine learning algorithms can not handle missing values.

1. **Identifying Missing Values:**

   First i checked for any missing values in the dataset.



```
dataset.isnull().sum()
```

```
                     0
             Age   187
          Gender     0
          Tenure     0
Service_Internet   721
   Service_Phone     0
      Service_TV     0
        Contract     0
   PaymentMethod   187
  MonthlyCharges     0
    TotalCharges     0
 StreamingMovies     0
  StreamingMusic     0
  OnlineSecurity     0
     TechSupport     0
           Churn     0

dtype: int64
```

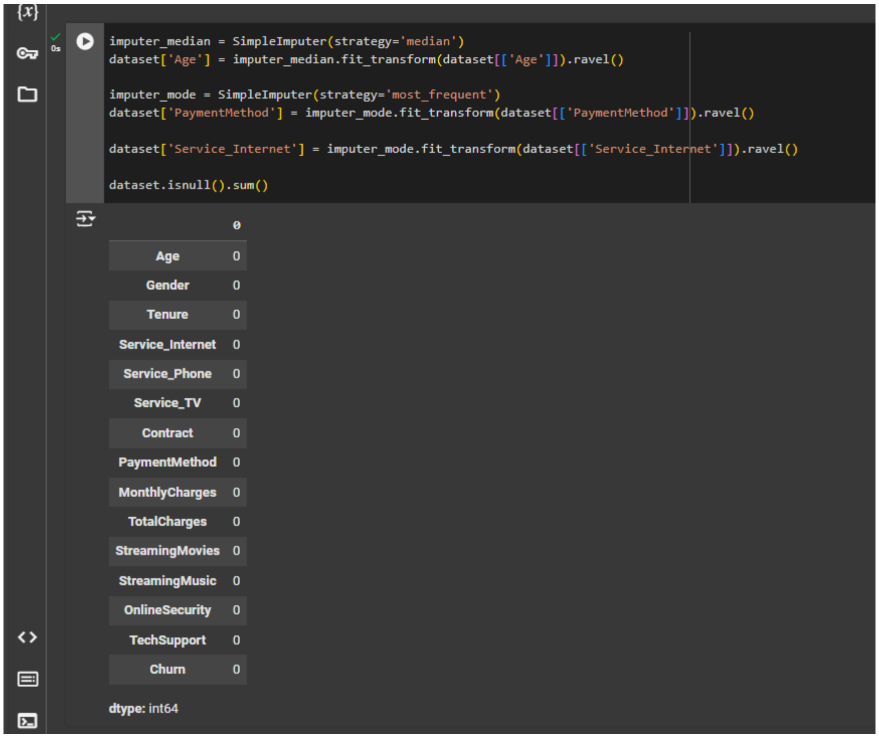
2. Handling Missing Values:

Generally for categorical features we use the mode and for numerical features we use the median or average depending on whether there are outliers.

I applied SimpleImputer with the median strategy to the age column The median is a good choice because it’s less affected by outliers than the average.

For categorical columns like PaymentMethod and Service_Internet, I used the most_frequent strategy in SimpleImputer to replace missing values with the mode because The mode represents the value that the majority of customers use. By filling in the missing values with the mode. so I kept it as close as possible to the original data.

After filling in the missing values, I checked the dataset again to make sure there were no more missing values left.

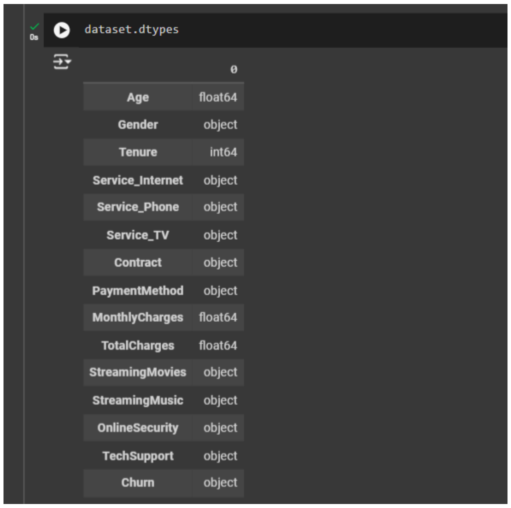


```
imputer_median = SimpleImputer(strategy='median')
dataset['Age'] = imputer_median.fit_transform(dataset[['Age']]).ravel()

imputer_mode = SimpleImputer(strategy='most_frequent')
dataset['PaymentMethod'] = imputer_mode.fit_transform(dataset[['PaymentMethod']]).ravel()

dataset['Service_Internet'] = imputer_mode.fit_transform(dataset[['Service_Internet']]).ravel()

dataset.isnull().sum()
```

```
                 0
           Age   0
        Gender   0
        Tenure   0
Service_Internet 0
 Service_Phone   0
    Service_TV   0
      Contract   0
 PaymentMethod   0
MonthlyCharges   0
  TotalCharges   0
StreamingMovies  0
StreamingMusic   0
OnlineSecurity   0
   TechSupport   0
         Churn   0

dtype: int64
```

3. Encoding Categorical Data:

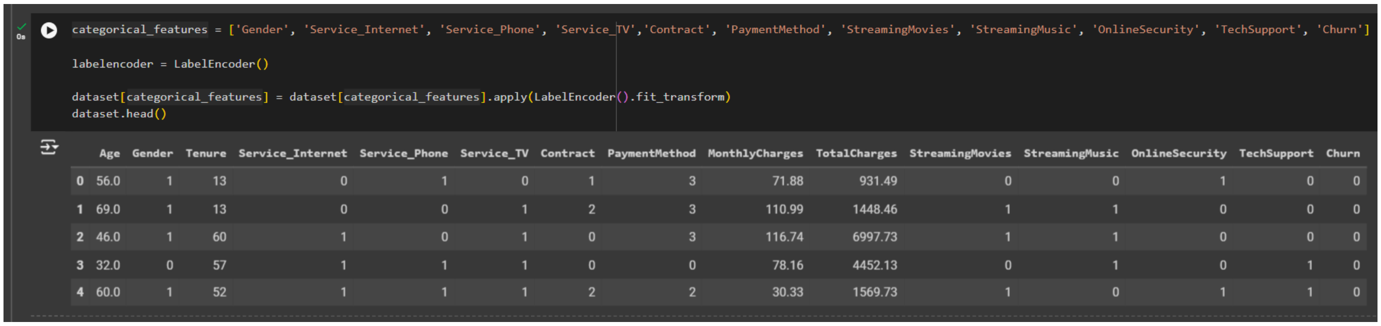
To prepare the dataset for machine learning algorithms it was necessary to convert the categorical variables into numerical form so first i identified the data types of each column to determine which columns are categorical.



```
dataset.dtypes
```

	0
Age	float64
Gender	object
Tenure	int64
Service_Internet	object
Service_Phone	object
Service_TV	object
Contract	object
PaymentMethod	object
MonthlyCharges	float64
TotalCharges	float64
StreamingMovies	object
StreamingMusic	object
OnlineSecurity	object
TechSupport	object
Churn	object

Then I used lable encoder to convert categorical variables into numerical values



```
categorical_features = ['Gender', 'Service_Internet', 'Service_Phone', 'Service_TV','Contract', 'PaymentMethod', 'StreamingMovies', 'StreamingMusic', 'OnlineSecurity', 'TechSupport', 'Churn']

labelencoder = LabelEncoder()

dataset[categorical_features] = dataset[categorical_features].apply(LabelEncoder().fit_transform)
dataset.head()
```

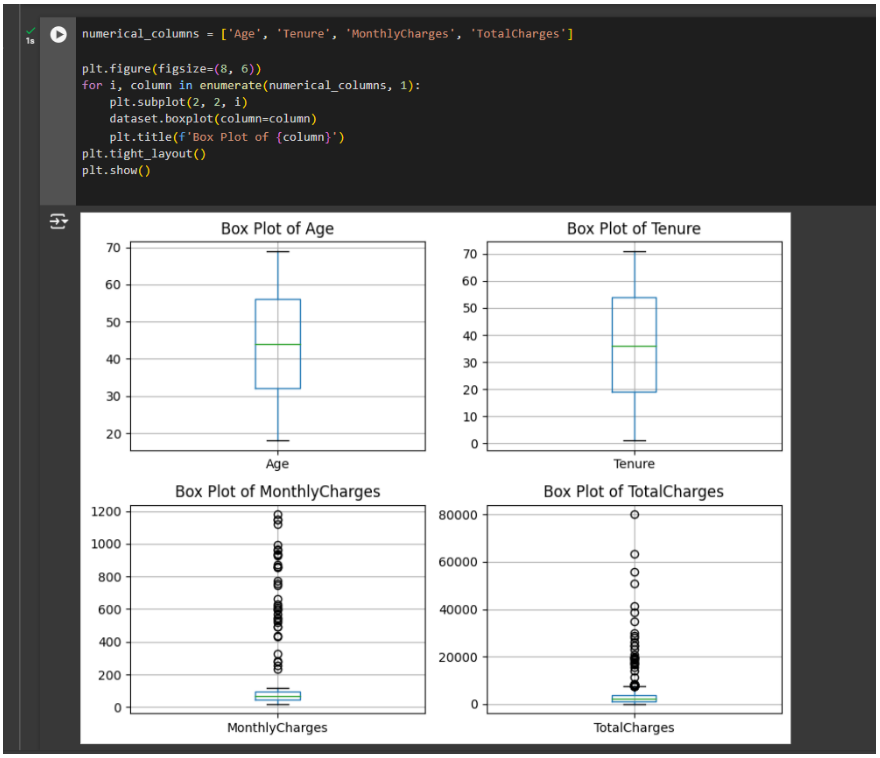
	Age	Gender	Tenure	Service_Internet	Service_Phone	Service_TV	Contract	PaymentMethod	MonthlyCharges	TotalCharges	StreamingMovies	StreamingMusic	OnlineSecurity	TechSupport	Churn
0	56.0	1	13	0	1	0	1	3	71.88	931.49	0	0	1	0	0
1	69.0	1	13	0	0	1	2	3	110.99	1448.46	1	1	0	0	0
2	46.0	1	60	1	0	1	0	3	116.74	6997.73	1	1	0	0	0
3	32.0	0	57	1	1	1	0	0	78.16	4452.13	0	1	0	1	0
4	60.0	1	52	1	1	1	2	2	30.33	1569.73	1	0	1	1	0

4. Outlier Detection and Handling:

Box Plot Analysis:

I created box plots for numerical columns to detect any outliers.

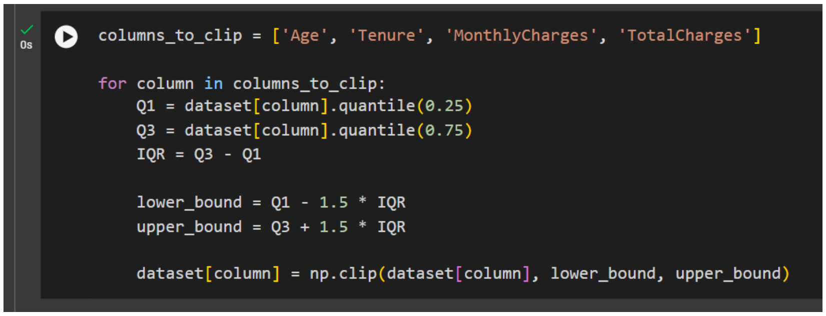
I realized that monthly charges and total charges have extreme outliers



```
numerical_columns = ['Age', 'Tenure', 'MonthlyCharges', 'TotalCharges']

plt.figure(figsize=(8, 6))
for i, column in enumerate(numerical_columns, 1):
    plt.subplot(2, 2, i)
    dataset.boxplot(column=column)
    plt.title(f'Box Plot of {column}')
plt.tight_layout()
plt.show()
```

Clipping the Outliers:

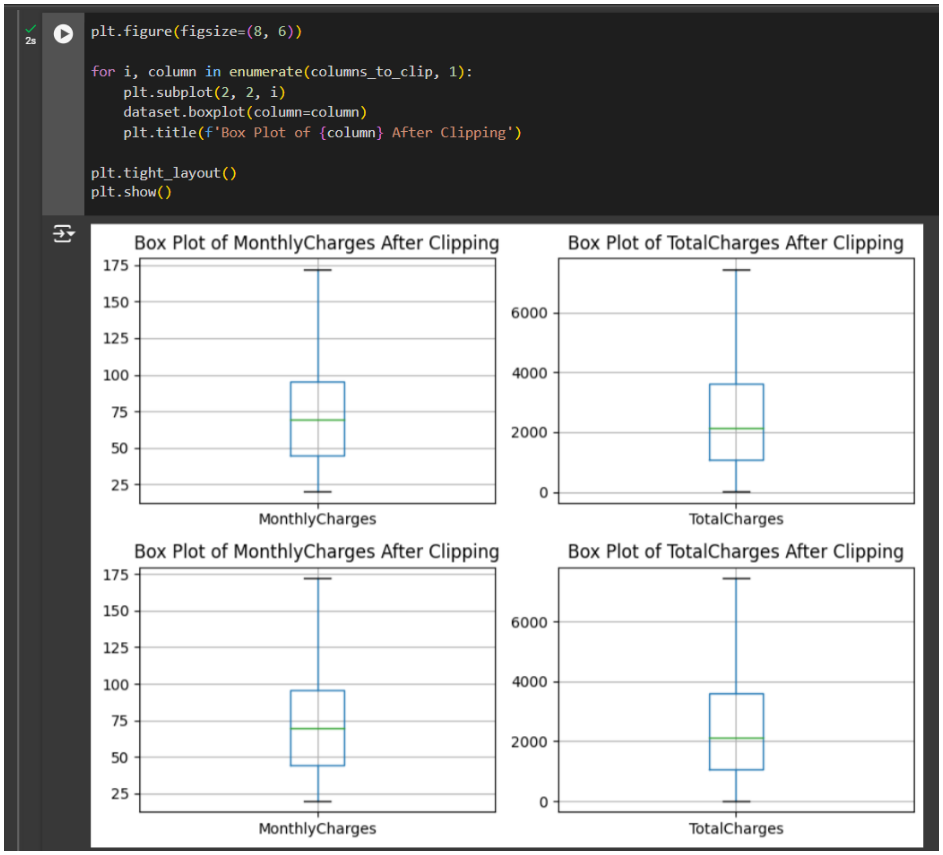
I used IQR based clipping to handle outliers I clipped outliers that were more than 1.5 times the interquartile range above or below the normal range



```
columns_to_clip = ['Age', 'Tenure', 'MonthlyCharges', 'TotalCharges']

for column in columns_to_clip:
    Q1 = dataset[column].quantile(0.25)
    Q3 = dataset[column].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    dataset[column] = np.clip(dataset[column], lower_bound, upper_bound)
```

Re Plotting to Verify:

After clipping I re plotted the box plots to verify that the outliers were handled correctly.



```
plt.figure(figsize=(8, 6))

for i, column in enumerate(columns_to_clip, 1):
    plt.subplot(2, 2, i)
    dataset.boxplot(column=column)
    plt.title(f'Box Plot of {column} After Clipping')

plt.tight_layout()
plt.show()
```

2s
Box Plot of MonthlyCharges After Clipping
175
150
125
100
75
50
25
MonthlyCharges
Box Plot of TotalCharges After Clipping
6000
4000
2000
0
TotalCharges
Box Plot of MonthlyCharges After Clipping
175
150
125
100
75
50
25
MonthlyCharges
Box Plot of TotalCharges After Clipping
6000
4000
2000
0
TotalCharges

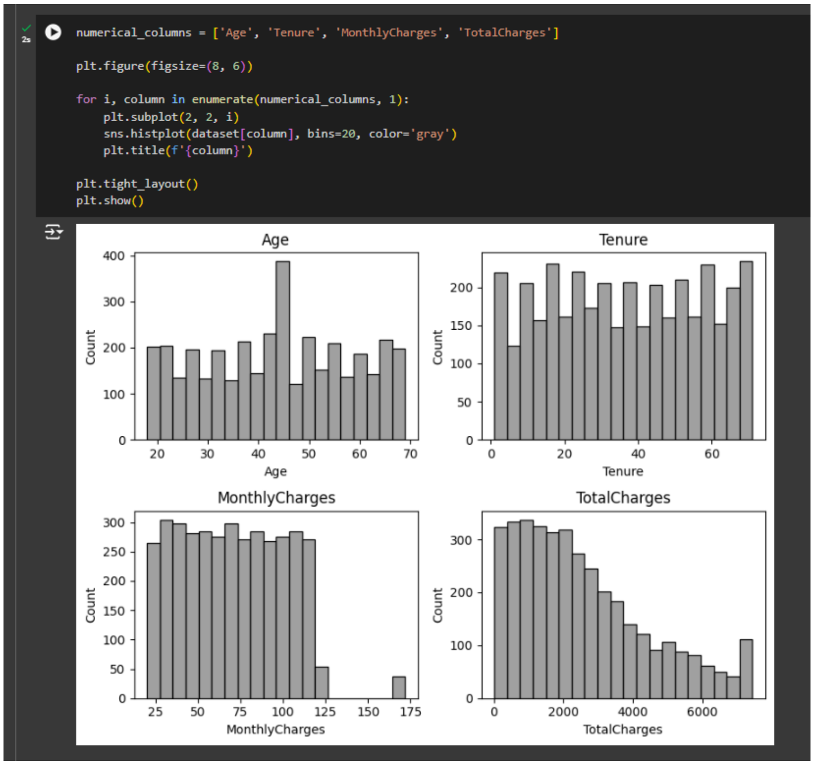
4. Exploratory Data Analysis (EDA):

In this part I looked closely at the data to understand it better and to see how the different features relate to churn. EDA is important because it helps find patterns, trends, and any unusual data. This helps in choosing the right features and models later. By making charts and doing some basic analysis I found key insights that will help in building the model.

1. Distribution of Numerical Features:

Plotting Distributions:

I made charts to show how the numerical features are distributed.



```
numerical_columns = ['Age', 'Tenure', 'MonthlyCharges', 'TotalCharges']

plt.figure(figsize=(8, 6))

for i, column in enumerate(numerical_columns, 1):
    plt.subplot(2, 2, i)
    sns.histplot(dataset[column], bins=20, color='gray')
    plt.title(f'{column}')

plt.tight_layout()
plt.show()
```

Age

The Age chart shows that customers are spread evenly across different age groups, with more customers around 40-50 years old. There is no age group with too many or too few customers.

Tenure

The Tenure chart shows that customers have stayed with the company for different lengths of time. No single group of customers stands out, meaning customers have a variety of tenure periods.

MonthlyCharges

The MonthlyCharges chart shows that most customers pay between $50 and $120 each month. Fewer customers pay more than $120, so higher monthly charges are less common.

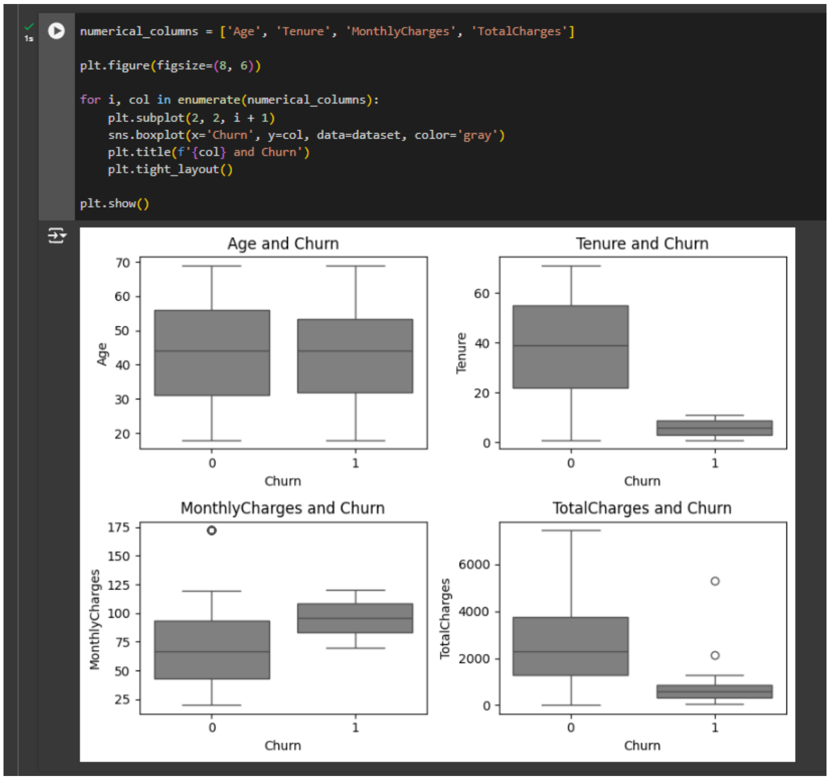
TotalCharges

The TotalCharges histogram is right-skewed meaning that most customers have lower total charges (under $3000), but a small number of customers have very high total charges, probably because they've been with the company longer or pay more each month.

2. Relationship with Churn:

Box Plots for Numerical Features:

I made box plots to compare how numerical features are related for customers who churned and who did not.



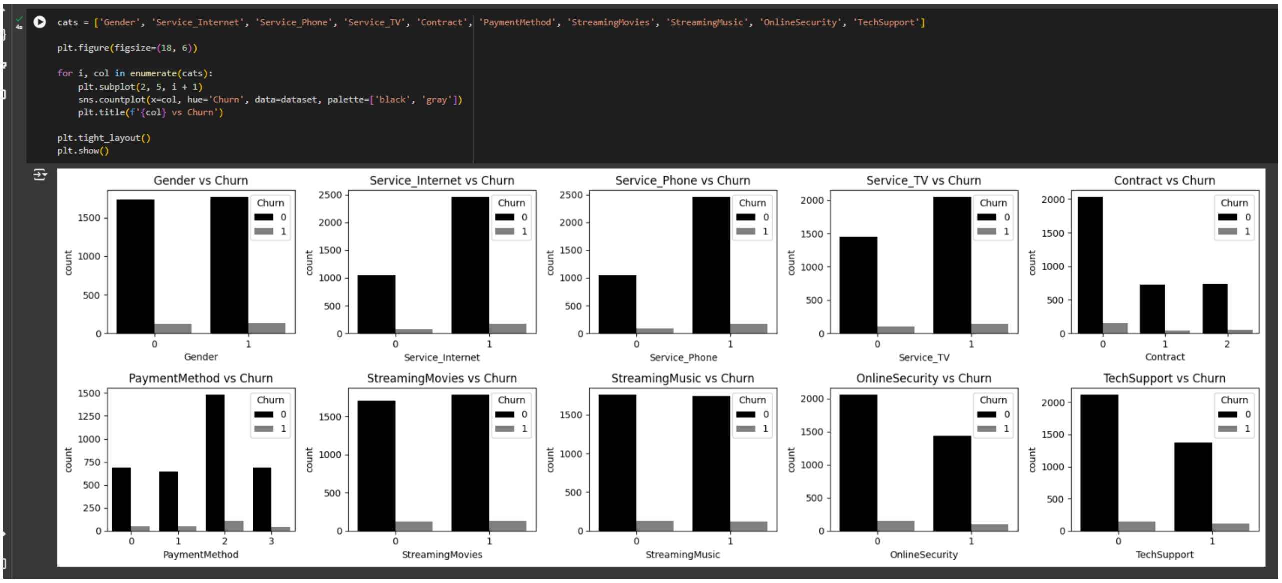
```
numerical_columns = ['Age', 'Tenure', 'MonthlyCharges', 'TotalCharges']

plt.figure(figsize=(8, 6))

for i, col in enumerate(numerical_columns):
    plt.subplot(2, 2, i + 1)
    sns.boxplot(x='Churn', y=col, data=dataset, color='gray')
    plt.title(f'{col} and Churn')
    plt.tight_layout()

plt.show()
```

From this visualization we realize that Tenure, MonthlyCharges, and TotalCharges as potentially important factors in predicting churn. And can be used in feature engineering.

Bar Plots for Categorical Features:

I also created bar charts to see how different categories relate to Churn.

These charts helped me to understand if certain services are linked to higher churn.



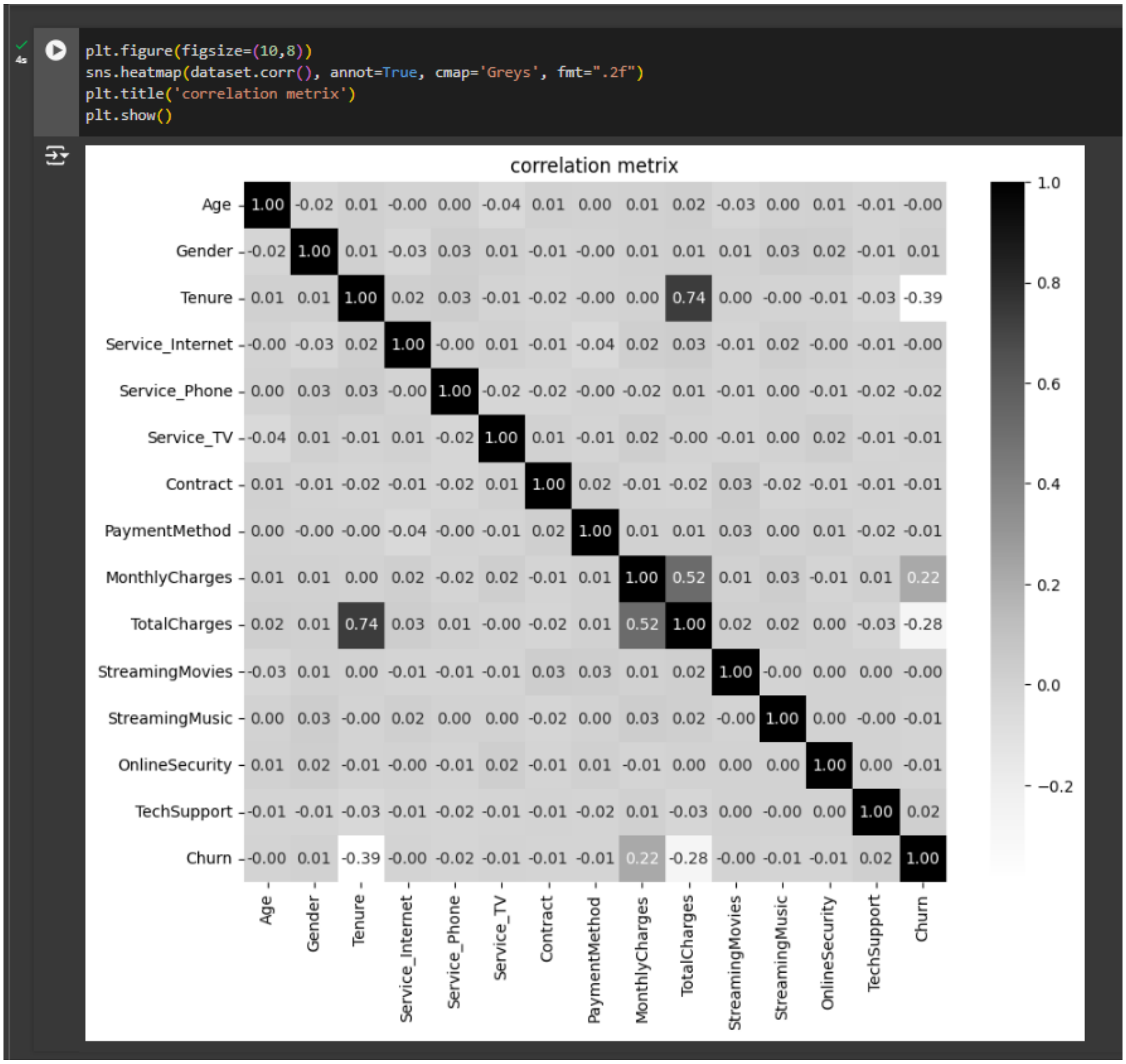
```
cats = ['Gender', 'Service_Internet', 'Service_Phone', 'Service_TV', 'Contract', 'PaymentMethod', 'StreamingMovies', 'StreamingMusic', 'OnlineSecurity', 'TechSupport']

plt.figure(figsize=(18, 6))

for i, col in enumerate(cats):
    plt.subplot(2, 5, i + 1)
    sns.countplot(x=col, hue='Churn', data=dataset, palette=['black', 'gray'])
    plt.title(f'{col} vs Churn')

plt.tight_layout()
plt.show()
```

Features like Contract, Service_Phone, OnlineSecurity, and TechSupport show clear relationships with churn and will likely be important in predicting customer churn.

3. Correlation Analysis:

Correlation Heatmap:

A correlation matrix shows the relationships between variables, helping identify important features for prediction. Values range from -1 (negative correlation) to +1 (positive correlation).



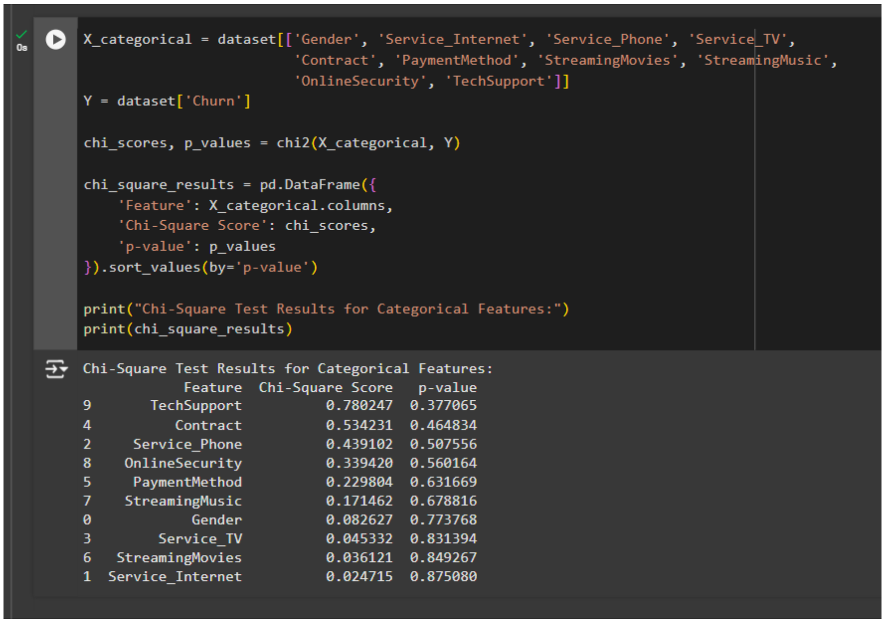
```
plt.figure(figsize=(10,8))
sns.heatmap(dataset.corr(), annot=True, cmap='Greys', fmt=".2f")
plt.title('correlation metrix')
plt.show()
```

	Age	Gender	Tenure	Service_Internet	Service_Phone	Service_TV	Contract	PaymentMethod	MonthlyCharges	TotalCharges	StreamingMovies	StreamingMusic	OnlineSecurity	TechSupport	Churn
Age	1.00	-0.02	0.01	-0.00	0.00	-0.04	0.01	0.00	0.01	0.02	-0.03	0.00	0.01	-0.01	-0.00
Gender	-0.02	1.00	0.01	-0.03	0.03	0.01	-0.01	-0.00	0.01	0.01	0.01	0.03	0.02	-0.01	0.01
Tenure	0.01	0.01	1.00	0.02	0.03	-0.01	-0.02	-0.00	0.00	0.74	0.00	-0.00	-0.01	-0.03	-0.39
Service_Internet	-0.00	-0.03	0.02	1.00	-0.00	0.01	-0.01	-0.04	0.02	0.03	-0.01	0.02	-0.00	-0.01	-0.00
Service_Phone	0.00	0.03	0.03	-0.00	1.00	-0.02	-0.02	-0.00	-0.02	0.01	-0.01	0.00	-0.01	-0.02	-0.02
Service_TV	-0.04	0.01	-0.01	0.01	-0.02	1.00	0.01	-0.01	0.02	-0.00	-0.01	0.00	0.02	-0.01	-0.01
Contract	0.01	-0.01	-0.02	-0.01	-0.02	0.01	1.00	0.02	-0.01	-0.02	0.03	-0.02	-0.01	-0.01	-0.01
PaymentMethod	0.00	-0.00	-0.00	-0.04	-0.00	-0.01	0.02	1.00	0.01	0.01	0.03	0.00	0.01	-0.02	-0.01
MonthlyCharges	0.01	0.01	0.00	0.02	-0.02	0.02	-0.01	0.01	1.00	0.52	0.01	0.03	-0.01	0.01	0.22
TotalCharges	0.02	0.01	0.74	0.03	0.01	-0.00	-0.02	0.01	0.52	1.00	0.02	0.02	0.00	-0.03	-0.28
StreamingMovies	-0.03	0.01	0.00	-0.01	-0.01	-0.01	0.03	0.03	0.01	0.02	1.00	-0.00	0.00	0.00	-0.00
StreamingMusic	0.00	0.03	-0.00	0.02	0.00	0.00	-0.02	0.00	0.03	0.02	-0.00	1.00	0.00	-0.00	-0.01
OnlineSecurity	0.01	0.02	-0.01	-0.00	-0.01	0.02	-0.01	0.01	-0.01	0.00	0.00	0.00	1.00	0.00	-0.01
TechSupport	-0.01	-0.01	-0.03	-0.01	-0.02	-0.01	-0.01	-0.02	0.01	-0.03	0.00	-0.00	0.00	1.00	0.02
Churn	-0.00	0.01	-0.39	-0.00	-0.02	-0.01	-0.01	-0.01	0.22	-0.28	-0.00	-0.01	-0.01	0.02	1.00

The matrix shows that Tenure, Contract, MonthlyCharges, and TotalCharges have strong correlations with churn and can be used for feature engineering.

4. Chi-Square Test for Categorical Features:

I ran a Chi-square test on the categorical features to see if any of them are linked to a churns. This test helps me figure out which features might be useful for predicting churn.



```
X_categorical = dataset[['Gender', 'Service_Internet', 'Service_Phone', 'Service_TV',
                         'Contract', 'PaymentMethod', 'StreamingMovies', 'StreamingMusic',
                         'OnlineSecurity', 'TechSupport']]
Y = dataset['Churn']

chi_scores, p_values = chi2(X_categorical, Y)

chi_square_results = pd.DataFrame({
    'Feature': X_categorical.columns,
    'Chi-Square Score': chi_scores,
    'p-value': p_values
}).sort_values(by='p-value')

print("Chi-Square Test Results for Categorical Features:")
print(chi_square_results)
```

```
Chi-Square Test Results for Categorical Features:
            Feature  Chi-Square Score   p-value
9       TechSupport          0.780247  0.377065
4          Contract          0.534231  0.464834
2     Service_Phone          0.439102  0.507556
8    OnlineSecurity          0.339420  0.560164
5     PaymentMethod          0.229804  0.631669
7    StreamingMusic          0.171462  0.678816
0            Gender          0.082627  0.773768
3        Service_TV          0.045332  0.831394
6   StreamingMovies          0.036121  0.849267
1  Service_Internet          0.024715  0.875080
```

Outcome:
The test showed that none of these features had a strong relationship with churn because all the p-values were high. but features like TechSupport and Contract had slightly lower p-values than others which means they might still have a small role in predicting churn.

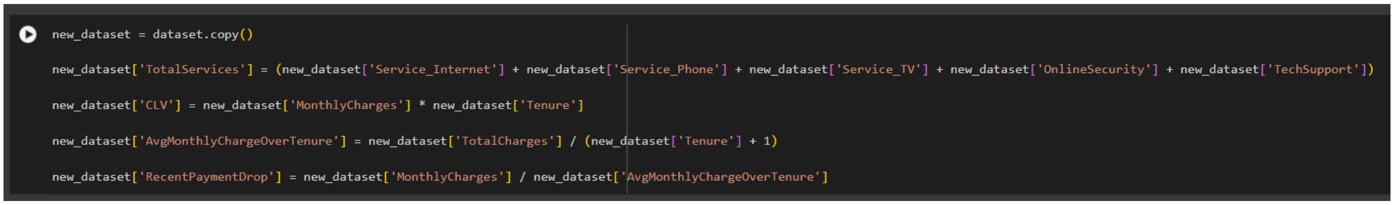
In short, I didn't find any strong connections, but a few features could still be worth looking into further.

5.Feature Engineering

1. Creating new features

   Feature engineering helps improve model accuracy by creating new features from existing data. In this step I created features that provide deeper insights into customer behavior

   We have selected these features based on previous analysis focusing on the most important attributes related to churn.



```
new_dataset = dataset.copy()

new_dataset['TotalServices'] = (new_dataset['Service_Internet'] + new_dataset['Service_Phone'] + new_dataset['Service_TV'] + new_dataset['OnlineSecurity'] + new_dataset['TechSupport'])

new_dataset['CLV'] = new_dataset['MonthlyCharges'] * new_dataset['Tenure']

new_dataset['AvgMonthlyChargeOverTenure'] = new_dataset['TotalCharges'] / (new_dataset['Tenure'] + 1)

new_dataset['RecentPaymentDrop'] = new_dataset['MonthlyCharges'] / new_dataset['AvgMonthlyChargeOverTenure']
```

TotalServices: This feature counts how many services a customer is using. Customers with more services tend to be more involved with the company so they are less likely to churn.

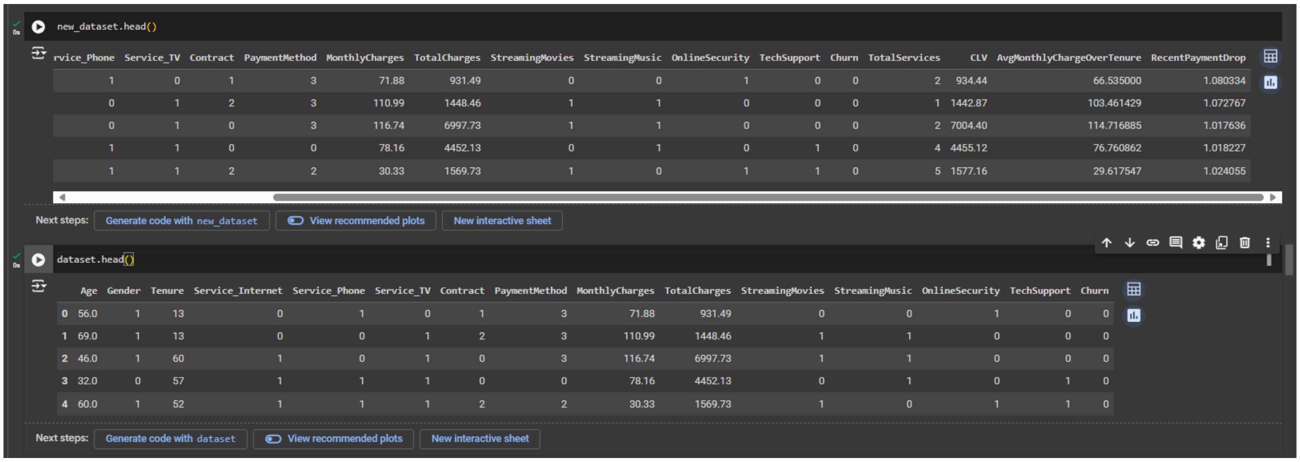
CLV (Customer Lifetime Value): This calculates how much money a customer has spent so far, by multiplying their monthly charges by Tenure. Customers with a higher CLV are more valuable and less likely to churn.

AvgMonthlyChargeOverTenure: This tells us the average amount the customer has paid every month over their entire tenure. Higher average means the person is more involved in the company and less likely to churn.

RecentPaymentDrop: This feature compares the current monthly charges to the customer’s average charges over time. If a customer is paying less recently, it might mean they’ve reduced their services or are less satisfied, which could make them more likely to leave.

2. **Verifying the process**

Then I visualized the first row of the original and new dataset to verify that the features were created an show the difference with original dataset



```
new_dataset.head()
```

```
rvice_Phone  Service_TV  Contract  PaymentMethod  MonthlyCharges  TotalCharges  StreamingMovies  StreamingMusic  OnlineSecurity  TechSupport  Churn  TotalServices      CLV  AvgMonthlyChargeOverTenure  RecentPaymentDrop
          1           0         1              3           71.88        931.49                0               0               1            0      0              2   934.44                   66.535000           1.080334
          0           1         2              3          110.99       1448.46                1               1               0            0      0              1  1442.87                  103.461429           1.072767
          0           1         0              3          116.74       6997.73                1               1               0            0      0              2  7004.40                  114.716885           1.017636
          1           1         0              0           78.16       4452.13                0               1               0            1      0              4  4455.12                   76.760862           1.018227
          1           1         2              2           30.33       1569.73                1               0               1            1      0              5  1577.16                   29.617547           1.024055
```

```
dataset.head()
```

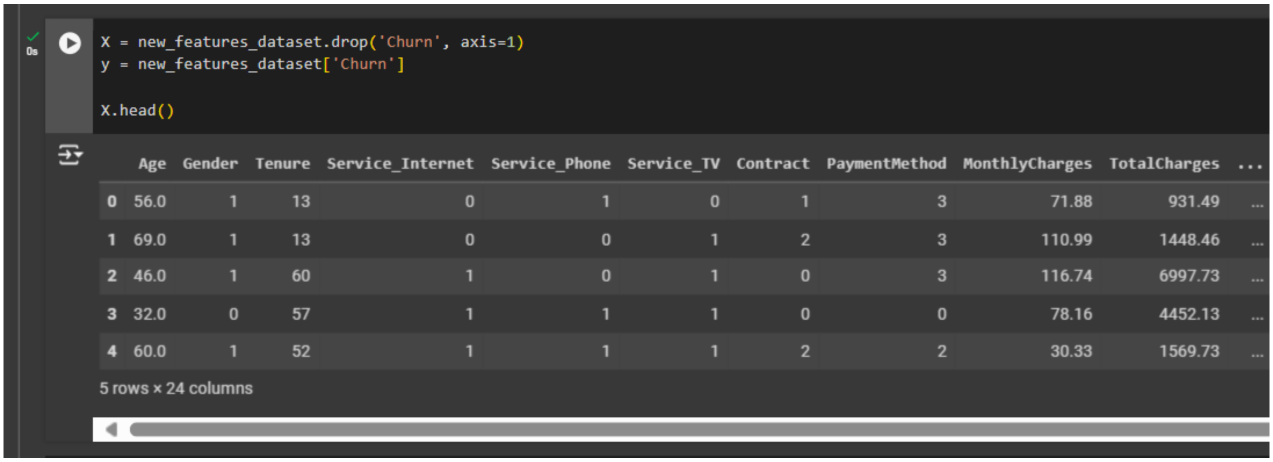
```
    Age  Gender  Tenure  Service_Internet  Service_Phone  Service_TV  Contract  PaymentMethod  MonthlyCharges  TotalCharges  StreamingMovies  StreamingMusic  OnlineSecurity  TechSupport  Churn
0  56.0       1      13                 0              1           0         1              3           71.88        931.49                0               0               1            0      0
1  69.0       1      13                 0              0           1         2              3          110.99       1448.46                1               1               0            0      0
2  46.0       1      60                 1              0           1         0              3          116.74       6997.73                1               1               0            0      0
3  32.0       0      57                 1              1           1         0              0           78.16       4452.13                0               1               0            1      0
4  60.0       1      52                 1              1           1         2              2           30.33       1569.73                1               0               1            1      0
```

6.Modeling

In the modeling and evaluation part, I focused on applying different machine learning models to predict customer churn. The process involved several steps:

1. **Feature and Target Separation:**

   The first step in the modeling process was to prepare the dataset for training and testing by splitting it into features (X) and the target variable (y).

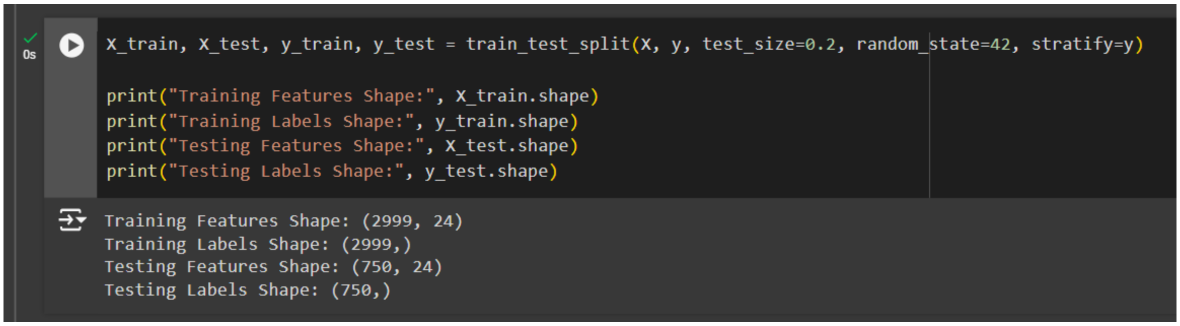


```
X = new_features_dataset.drop('Churn', axis=1)
y = new_features_dataset['Churn']

X.head()
```

```
    Age  Gender  Tenure  Service_Internet  Service_Phone  Service_TV  Contract  PaymentMethod  MonthlyCharges  TotalCharges  ...
0  56.0       1      13                 0              1           0         1              3           71.88        931.49  ...
1  69.0       1      13                 0              0           1         2              3          110.99       1448.46  ...
2  46.0       1      60                 1              0           1         0              3          116.74       6997.73  ...
3  32.0       0      57                 1              1           1         0              0           78.16       4452.13  ...
4  60.0       1      52                 1              1           1         2              2           30.33       1569.73  ...

5 rows × 24 columns
```

2. **Splitting the Data:**

Then I split the dataset into training and testing sets using a 20% test size. This allowed me to evaluate the models on data they had not seen during training then I printed the shape of the splits to verify the process.

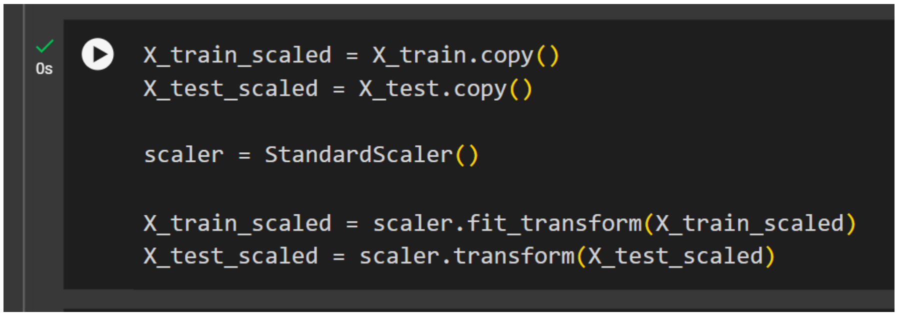


```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

print("Training Features Shape:", X_train.shape)
print("Training Labels Shape:", y_train.shape)
print("Testing Features Shape:", X_test.shape)
print("Testing Labels Shape:", y_test.shape)
```

```
Training Features Shape: (2999, 24)
Training Labels Shape: (2999,)
Testing Features Shape: (750, 24)
Testing Labels Shape: (750,)
```

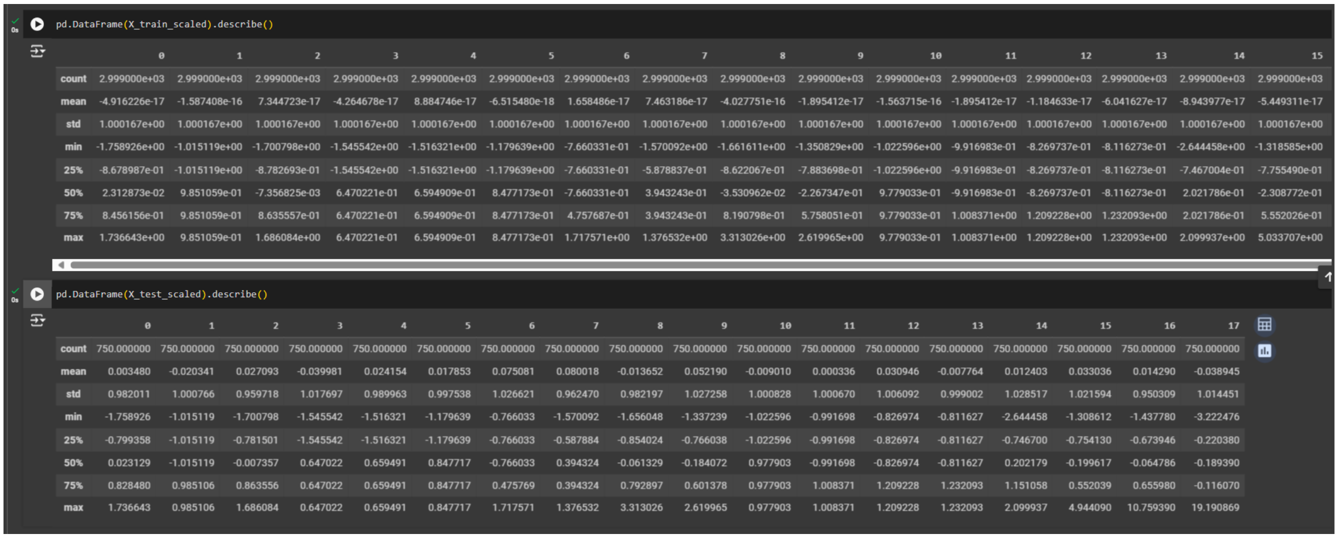
3. **Feature Scaling:**

We are scaling our data using StandardScaler to ensure all features contribute equally to the model by normalizing them to have a mean close to 0 and a standard deviation close to 1. First, the training data is scaled by calculating the mean and standard deviation of each feature. Then, the same scaling is applied to the test data using the mean and standard deviation from the training data.



```
X_train_scaled = X_train.copy()
X_test_scaled = X_test.copy()

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train_scaled)
X_test_scaled = scaler.transform(X_test_scaled)
```

I used describe() function for both train and test features to verify that the scaling worked correctly.



```
pd.DataFrame(X_train_scaled).describe()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
count	2.999000e+03	2.999000e+03	2.999000e+03	2.999000e+03	2.999000e+03	2.999000e+03	2.999000e+03	2.999000e+03	2.999000e+03	2.999000e+03	2.999000e+03	2.999000e+03	2.999000e+03	2.999000e+03	2.999000e+03	2.999000e+03
mean	-4.916226e-17	-1.587408e-16	7.344723e-17	-4.264678e-17	8.884746e-17	-6.515480e-18	1.658486e-17	7.463186e-17	-4.027751e-16	-1.895412e-17	-1.563715e-16	-1.895412e-17	-1.184633e-17	-6.041627e-17	-8.943977e-17	-5.449311e-17
std	1.000167e+00	1.000167e+00	1.000167e+00	1.000167e+00	1.000167e+00	1.000167e+00	1.000167e+00	1.000167e+00	1.000167e+00	1.000167e+00	1.000167e+00	1.000167e+00	1.000167e+00	1.000167e+00	1.000167e+00	1.000167e+00
min	-1.758926e+00	-1.015119e+00	-1.700798e+00	-1.545542e+00	-1.516321e+00	-1.179639e+00	-7.660331e-01	-1.570092e+00	-1.661611e+00	-1.350829e+00	-1.022596e+00	-9.916983e-01	-8.269737e-01	-8.116273e-01	-2.644458e+00	-1.318585e+00
25%	-8.678987e-01	-1.015119e+00	-8.782693e-01	-1.545542e+00	-1.516321e+00	-1.179639e+00	-7.660331e-01	-5.878837e-01	-8.622067e-01	-7.883698e-01	-1.022596e+00	-9.916983e-01	-8.269737e-01	-8.116273e-01	-7.467004e-01	-7.755490e-01
50%	2.312873e-02	9.851059e-01	-7.356825e-03	6.470221e-01	6.594909e-01	8.477173e-01	-7.660331e-01	3.943243e-01	-3.530962e-02	-2.267347e-01	9.779033e-01	-9.916983e-01	-8.269737e-01	-8.116273e-01	2.021786e-01	-2.308772e-01
75%	8.456156e-01	9.851059e-01	8.635557e-01	6.470221e-01	6.594909e-01	8.477173e-01	4.757687e-01	3.943243e-01	8.190798e-01	5.758051e-01	9.779033e-01	1.008371e+00	1.209228e+00	1.232093e+00	2.021786e-01	5.552026e-01
max	1.736643e+00	9.851059e-01	1.686084e+00	6.470221e-01	6.594909e-01	8.477173e-01	1.717571e+00	1.376532e+00	3.313026e+00	2.619965e+00	9.779033e-01	1.008371e+00	1.209228e+00	1.232093e+00	2.099937e+00	5.033707e+00

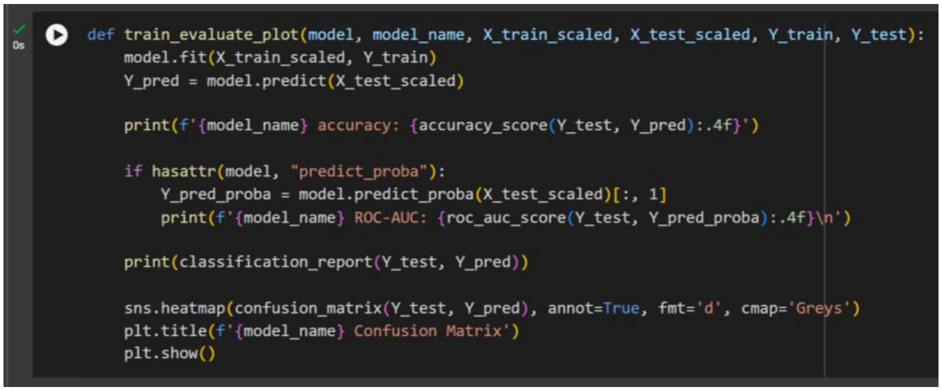
```
pd.DataFrame(X_test_scaled).describe()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
count	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000
mean	0.003480	-0.020341	0.027093	-0.039981	0.024154	0.017853	0.075081	0.080018	-0.013652	0.052190	-0.009010	0.000336	0.030946	-0.007764	0.012403	0.033036	0.014290	-0.038945
std	0.982011	1.000766	0.959718	1.017697	0.989963	0.997538	1.026621	0.962470	0.982197	1.027258	1.000828	1.000670	1.006092	0.999002	1.028517	1.021594	0.950309	1.014451
min	-1.758926	-1.015119	-1.700798	-1.545542	-1.516321	-1.179639	-0.766033	-1.570092	-1.656048	-1.337239	-1.022596	-0.991698	-0.826974	-0.811627	-2.644458	-1.308612	-1.437780	-3.222476
25%	-0.799358	-1.015119	-0.781501	-1.545542	-1.516321	-1.179639	-0.766033	-0.587884	-0.854024	-0.766038	-1.022596	-0.991698	-0.826974	-0.811627	-0.746700	-0.754130	-0.673946	-0.220380
50%	0.023129	-1.015119	-0.007357	0.647022	0.659491	0.847717	-0.766033	0.394324	-0.061329	-0.184072	0.977903	-0.991698	-0.826974	-0.811627	0.202179	-0.199617	-0.064786	-0.189390
75%	0.828480	0.985106	0.863556	0.647022	0.659491	0.847717	0.475769	0.394324	0.792897	0.601378	0.977903	1.008371	1.209228	1.232093	1.151058	0.552039	0.655980	-0.116070
max	1.736643	0.985106	1.686084	0.647022	0.659491	0.847717	1.717571	1.376532	3.313026	2.619965	0.977903	1.008371	1.209228	1.232093	2.099937	4.944090	10.759390	19.190869

4. Defining evaluation function

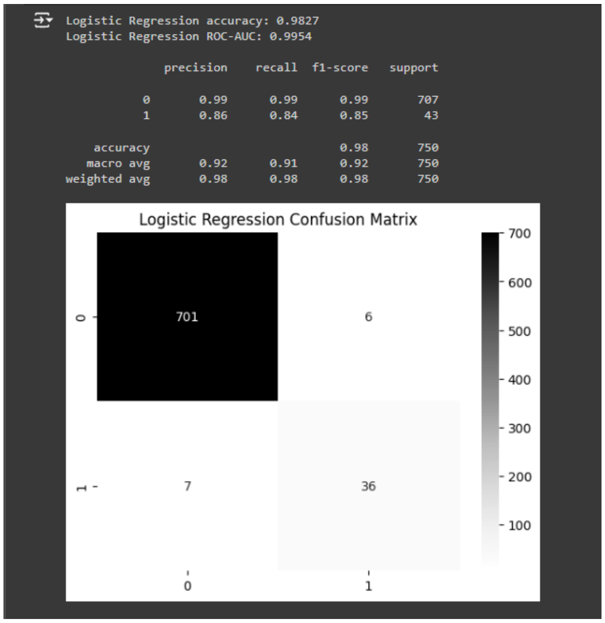
In the Model Evaluation section i used a function to test different models, like Logistic Regression, Decision Tree, Random Forest, and Gradient Boosting, for predicting churn. The function calculates important scores like accuracy,roc-auc, precision, recall, and F1-score, and shows the confusion matrix to display errors in predictions.

i trained the models using the scaled training data (X_train_scaled, Y_train) and checked how well they worked on the scaled test data (X_test_scaled, Y_test).



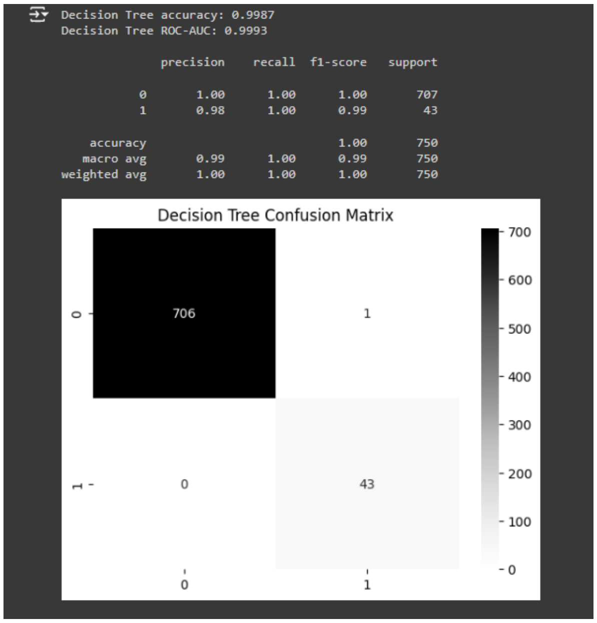
```
def train_evaluate_plot(model, model_name, X_train_scaled, X_test_scaled, Y_train, Y_test):
    model.fit(X_train_scaled, Y_train)
    Y_pred = model.predict(X_test_scaled)

    print(f'{model_name} accuracy: {accuracy_score(Y_test, Y_pred):.4f}')

    if hasattr(model, "predict_proba"):
        Y_pred_proba = model.predict_proba(X_test_scaled)[:, 1]
        print(f'{model_name} ROC-AUC: {roc_auc_score(Y_test, Y_pred_proba):.4f}\n')

    print(classification_report(Y_test, Y_pred))

    sns.heatmap(confusion_matrix(Y_test, Y_pred), annot=True, fmt='d', cmap='Greys')
    plt.title(f'{model_name} Confusion Matrix')
    plt.show()
```

Logistic regression:



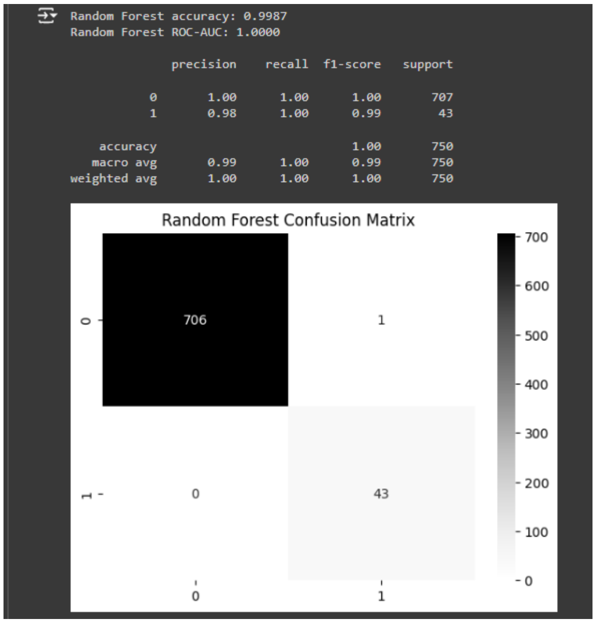
```
Logistic Regression accuracy: 0.9827
Logistic Regression ROC-AUC: 0.9954

              precision    recall  f1-score   support

           0       0.99      0.99      0.99       707
           1       0.86      0.84      0.85        43

    accuracy                           0.98       750
   macro avg       0.92      0.91      0.92       750
weighted avg       0.98      0.98      0.98       750
```

Decision Tree:



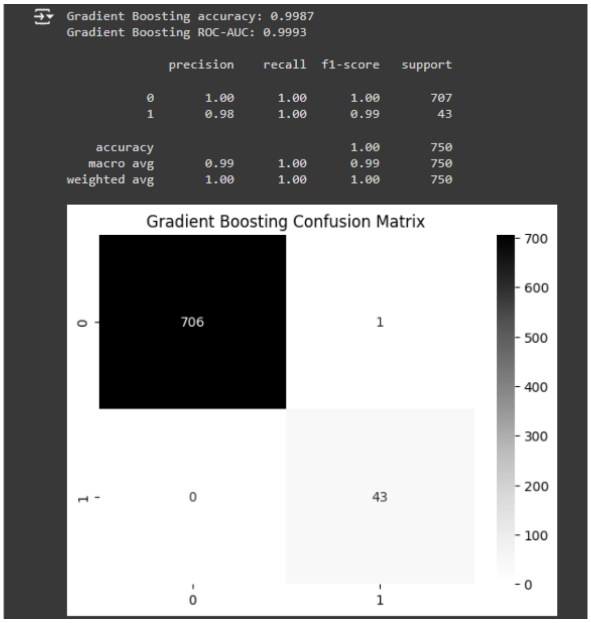
```
Decision Tree accuracy: 0.9987
Decision Tree ROC-AUC: 0.9993

              precision    recall  f1-score   support

           0       1.00      1.00      1.00       707
           1       0.98      1.00      0.99        43

    accuracy                           1.00       750
   macro avg       0.99      1.00      0.99       750
weighted avg       1.00      1.00      1.00       750
```

Random Forest:



```
Random Forest accuracy: 0.9987
Random Forest ROC-AUC: 1.0000

              precision    recall  f1-score   support

           0       1.00      1.00      1.00       707
           1       0.98      1.00      0.99        43

    accuracy                           1.00       750
   macro avg       0.99      1.00      0.99       750
weighted avg       1.00      1.00      1.00       750
```

Gradient Boosting:



```
Gradient Boosting accuracy: 0.9987
Gradient Boosting ROC-AUC: 0.9993

              precision    recall  f1-score   support

           0       1.00      1.00      1.00       707
           1       0.98      1.00      0.99        43

    accuracy                           1.00       750
   macro avg       0.99      1.00      0.99       750
weighted avg       1.00      1.00      1.00       750
```

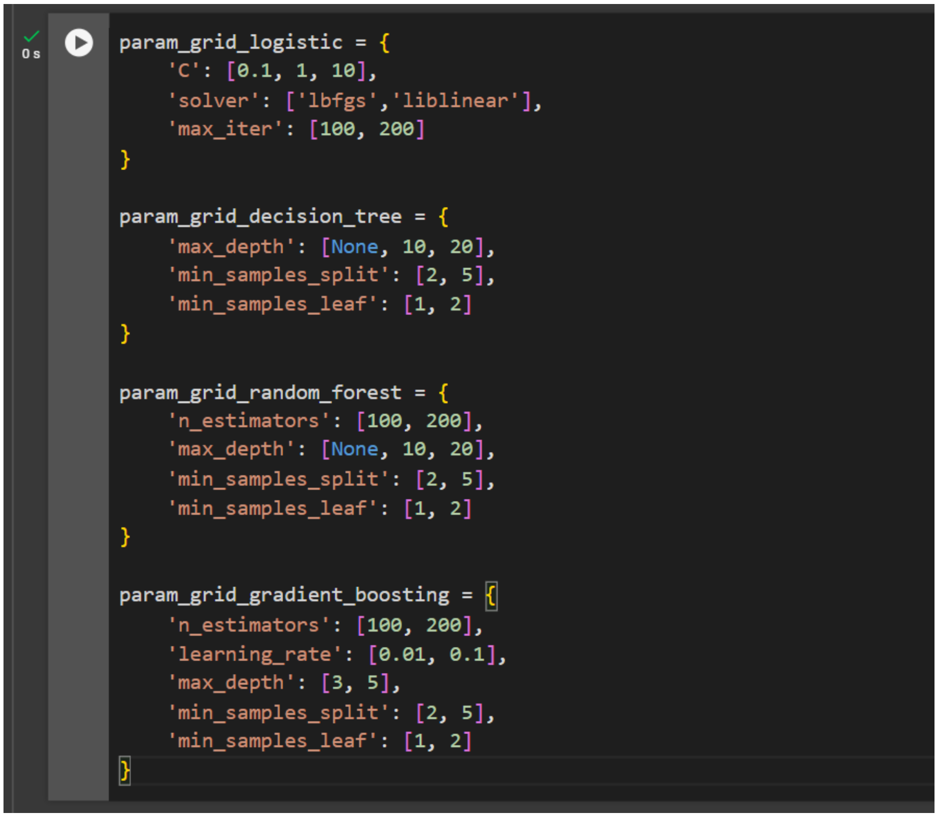
Random Forest, Decision Tree, and Gradient Boosting all performed almost perfectly. Random Forest had the best result with a perfect ROC-AUC of 1.0000, making it the top choice. Logistic Regression had lower recall and ROC-AUC.

7.Model Tuning:

In hyperparameter tuning, I use GridSearchCV to find the best parameters for my models. This helps improve accuracy and ensures the model works well on new data. By fine-tuning the model, I can avoid overfitting and make better predictions.

1. Hyperparameter Setup

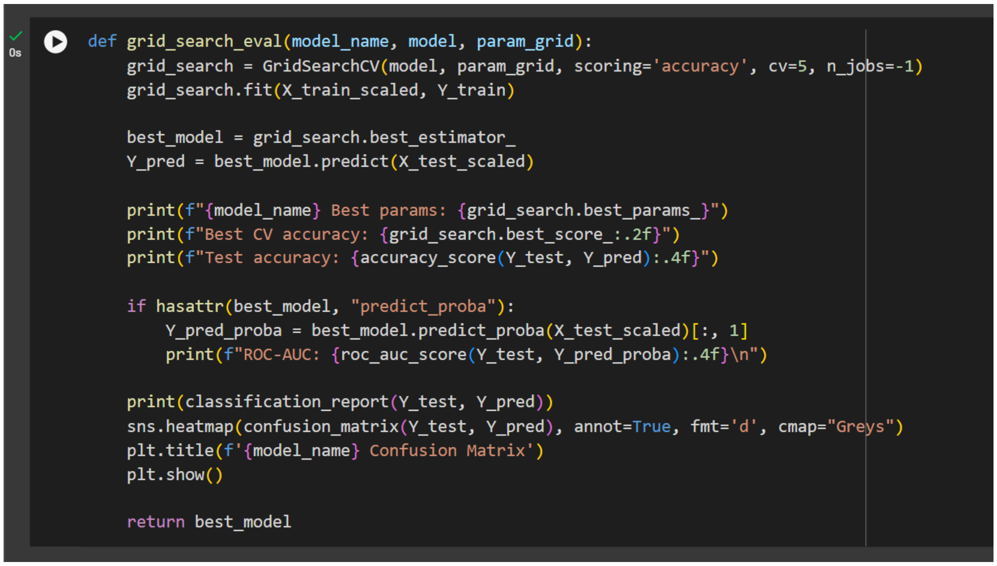
Here, I defined a grid of possible hyperparameters for each model: Logistic Regression, Decision Tree, Random Forest, and Gradient Boosting. These grids will be used for searching the best combination of hyperparameters during model tuning.



```
param_grid_logistic = {
    'C': [0.1, 1, 10],
    'solver': ['lbfgs','liblinear'],
    'max_iter': [100, 200]
}

param_grid_decision_tree = {
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

param_grid_random_forest = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

param_grid_gradient_boosting = {
    'n_estimators': [100, 200],
    'learning_rate': [0.01, 0.1],
    'max_depth': [3, 5],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}
```

2. Hyperparameter Tuning

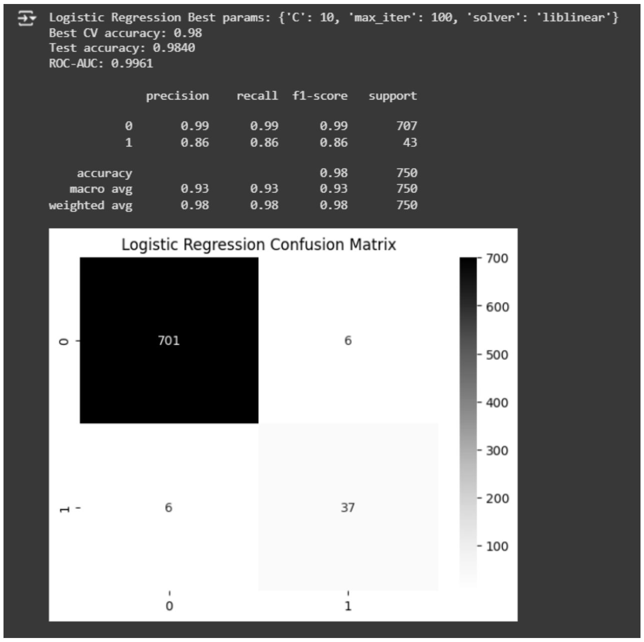
This part of the code uses GridSearchCV to test different settings for the model to find the best one. It checks each setting using the training data and chooses the best model. Then, it makes predictions on the test data, shows the accuracy, classification report, and confusion matrix to see how well the model did.



```
def grid_search_eval(model_name, model, param_grid):
    grid_search = GridSearchCV(model, param_grid, scoring='accuracy', cv=5, n_jobs=-1)
    grid_search.fit(X_train_scaled, Y_train)

    best_model = grid_search.best_estimator_
    Y_pred = best_model.predict(X_test_scaled)

    print(f"{model_name} Best params: {grid_search.best_params_}")
    print(f"Best CV accuracy: {grid_search.best_score_:.2f}")
    print(f"Test accuracy: {accuracy_score(Y_test, Y_pred):.4f}")

    if hasattr(best_model, "predict_proba"):
        Y_pred_proba = best_model.predict_proba(X_test_scaled)[:, 1]
        print(f"ROC-AUC: {roc_auc_score(Y_test, Y_pred_proba):.4f}\n")

    print(classification_report(Y_test, Y_pred))
    sns.heatmap(confusion_matrix(Y_test, Y_pred), annot=True, fmt='d', cmap="Greys")
    plt.title(f'{model_name} Confusion Matrix')
    plt.show()

    return best_model
```

These are the results of models after hyper tunning:

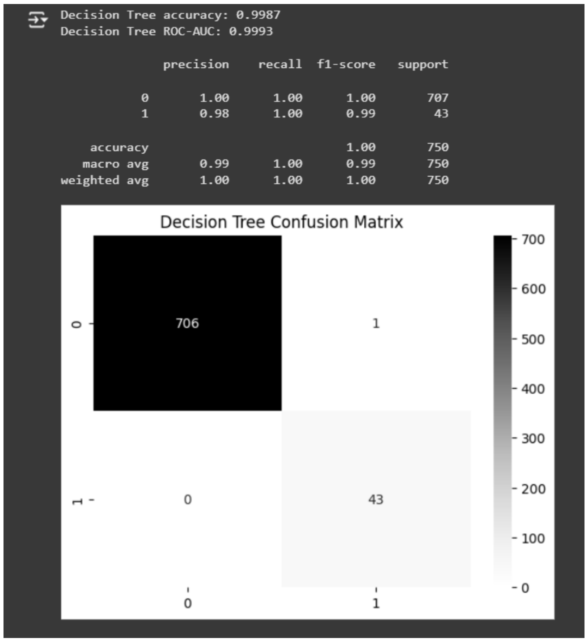
Logistic Regression



```
Logistic Regression Best params: {'C': 10, 'max_iter': 100, 'solver': 'liblinear'}
Best CV accuracy: 0.98
Test accuracy: 0.9840
ROC-AUC: 0.9961

              precision    recall  f1-score   support

           0       0.99      0.99      0.99       707
           1       0.86      0.86      0.86        43

    accuracy                           0.98       750
   macro avg       0.93      0.93      0.93       750
weighted avg       0.98      0.98      0.98       750
```

Logistic Regression Confusion Matrix

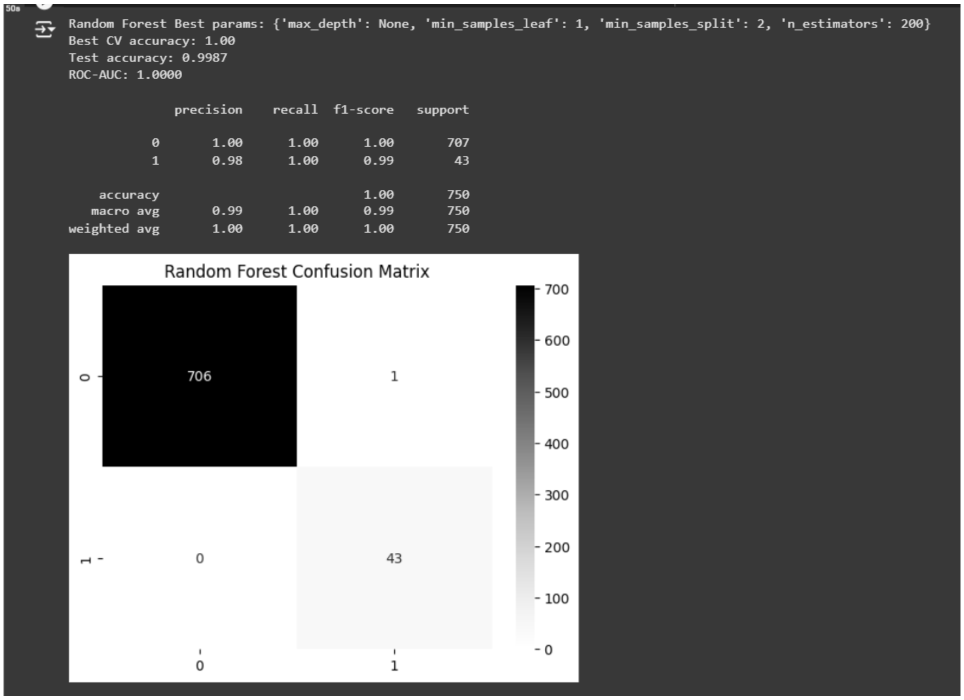
Decision Tree



```
Decision Tree accuracy: 0.9987
Decision Tree ROC-AUC: 0.9993

              precision    recall  f1-score   support

           0       1.00      1.00      1.00       707
           1       0.98      1.00      0.99        43

    accuracy                           1.00       750
   macro avg       0.99      1.00      0.99       750
weighted avg       1.00      1.00      1.00       750
```

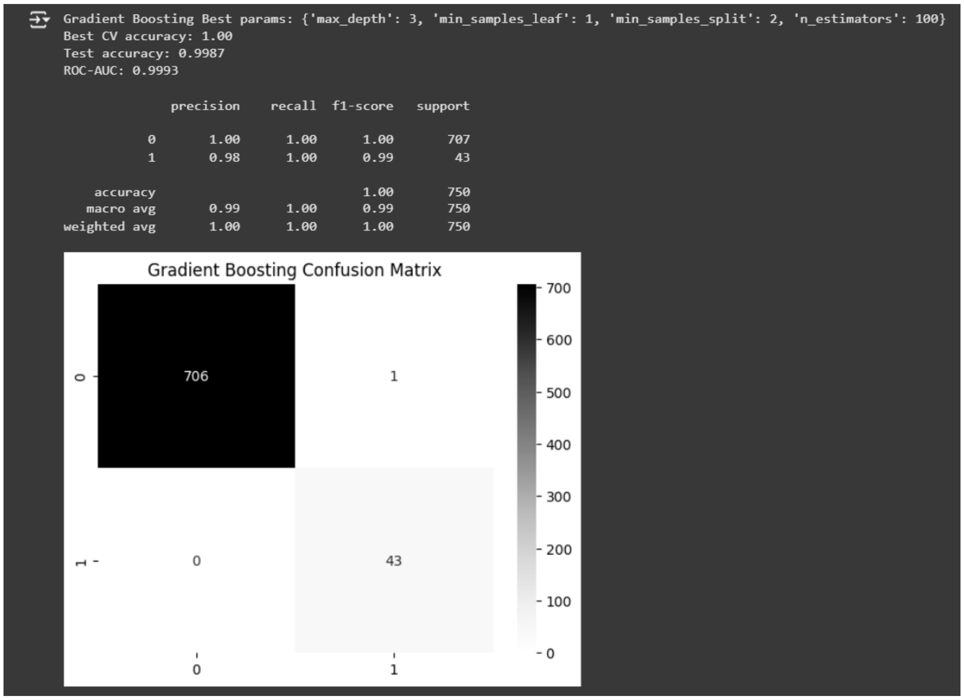
Decision Tree Confusion Matrix

Random Forest



```
Random Forest Best params: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Best CV accuracy: 1.00
Test accuracy: 0.9987
ROC-AUC: 1.0000

              precision    recall  f1-score   support

           0       1.00      1.00      1.00       707
           1       0.98      1.00      0.99        43

    accuracy                           1.00       750
   macro avg       0.99      1.00      0.99       750
weighted avg       1.00      1.00      1.00       750
```

Gradient Boosting



```
Gradient Boosting Best params: {'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
Best CV accuracy: 1.00
Test accuracy: 0.9987
ROC-AUC: 0.9993

              precision    recall  f1-score   support

           0       1.00      1.00      1.00       707
           1       0.98      1.00      0.99        43

    accuracy                           1.00       750
   macro avg       0.99      1.00      0.99       750
weighted avg       1.00      1.00      1.00       750
```

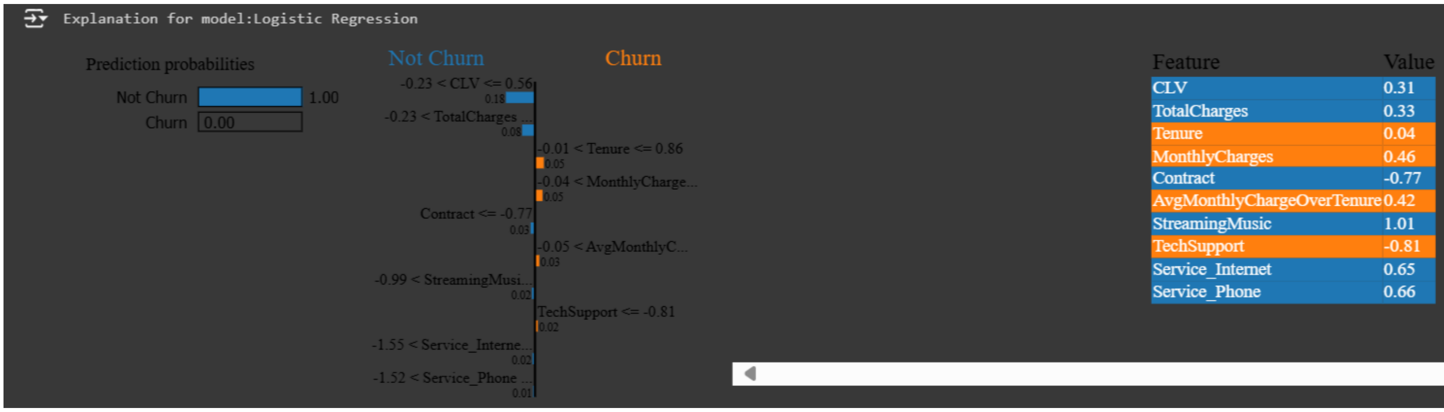
The Random Forest model performs best with a perfect ROC-AUC of 1.0 and 99.87% accuracy, similar to Decision Tree and Gradient Boosting. All three handle class 1 well, while Logistic Regression is weaker with 98.40% accuracy and lower performance for class 1. Random Forest is the top choice, but the other two are close behind, with Logistic Regression being simpler but less effective.

8. Model Interpretation

1. Model agnostic Explanation

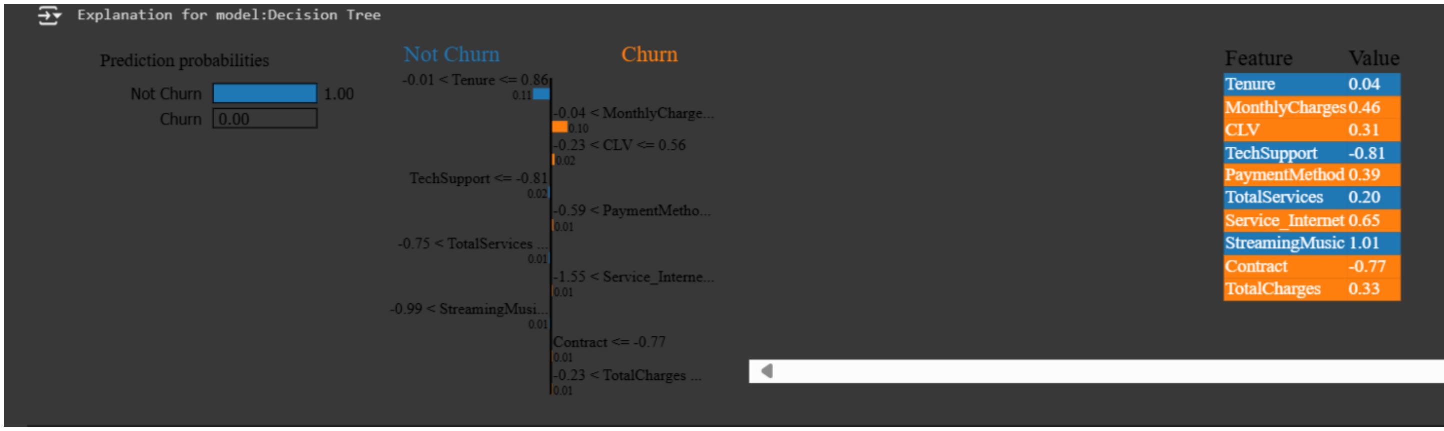
I used LIME to explain how the models make predictions. All models mostly predict "Not Churn" because of features like CLV, TotalCharges, and Service_Internet.

MonthlyCharges and Tenure push a bit towards "Churn," but don't have a big impact. The Gradient Boosting model is a bit less sure, but overall, the predictions are quite similar.



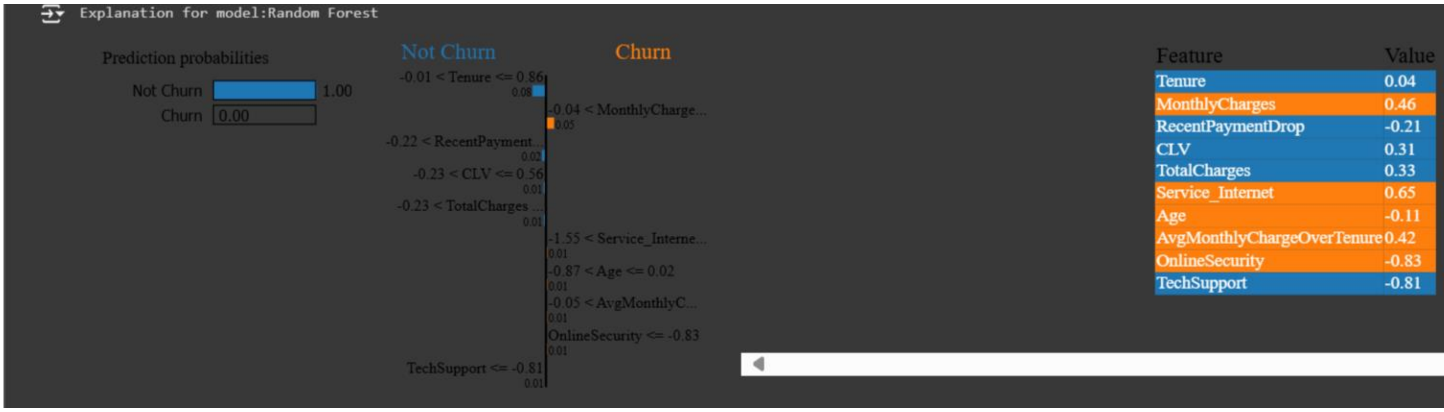
```
Explanation for model:Logistic Regression
```

Feature	Value
CLV	0.31
TotalCharges	0.33
Tenure	0.04
MonthlyCharges	0.46
Contract	-0.77
AvgMonthlyChargeOverTenure	0.42
StreamingMusic	1.01
TechSupport	-0.81
Service_Internet	0.65
Service_Phone	0.66



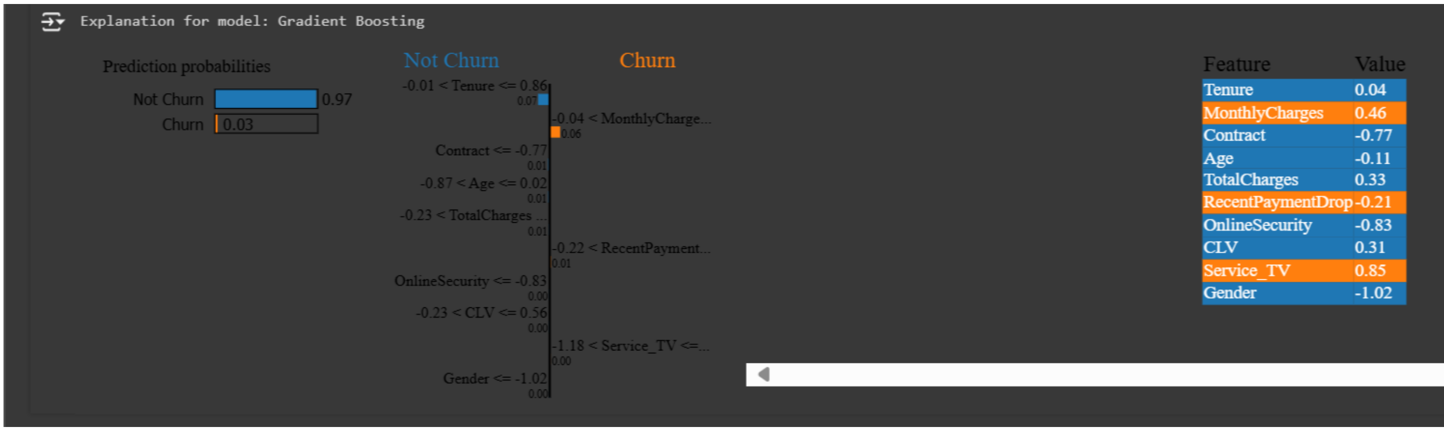
```
Explanation for model:Decision Tree
```

Feature	Value
Tenure	0.04
MonthlyCharges	0.46
CLV	0.31
TechSupport	-0.81
PaymentMethod	0.39
TotalServices	0.20
Service_Internet	0.65
StreamingMusic	1.01
Contract	-0.77
TotalCharges	0.33



```
Explanation for model:Random Forest
```

Feature	Value
Tenure	0.04
MonthlyCharges	0.46
RecentPaymentDrop	-0.21
CLV	0.31
TotalCharges	0.33
Service_Internet	0.65
Age	-0.11
AvgMonthlyChargeOverTenure	0.42
OnlineSecurity	-0.83
TechSupport	-0.81

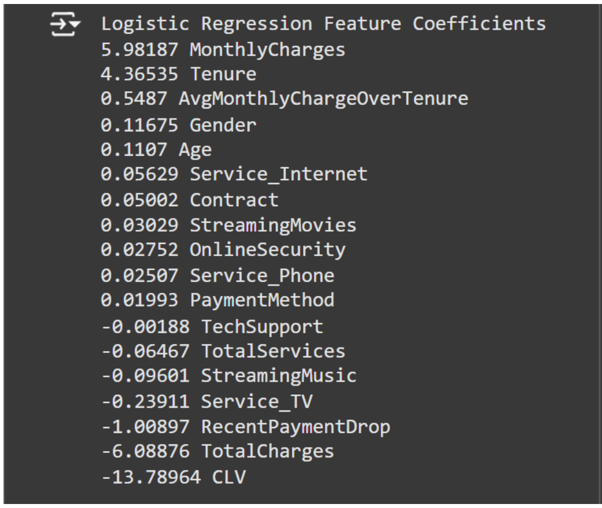


```
Explanation for model: Gradient Boosting
```

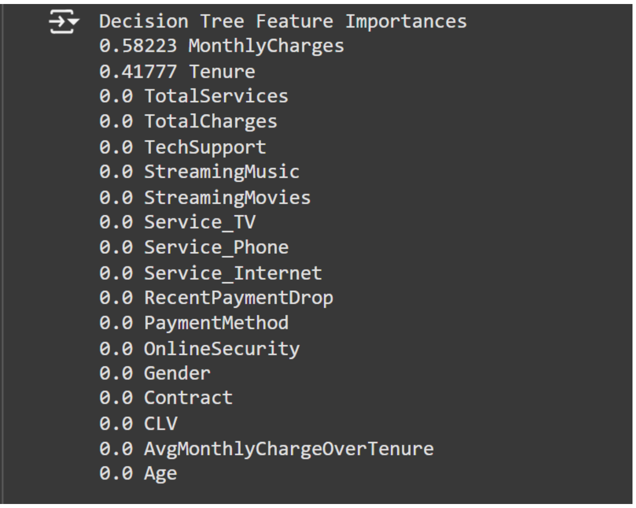
Feature	Value
Tenure	0.04
MonthlyCharges	0.46
Contract	-0.77
Age	-0.11
TotalCharges	0.33
RecentPaymentDrop	-0.21
OnlineSecurity	-0.83
CLV	0.31
Service_TV	0.85
Gender	-1.02

2. feature importance

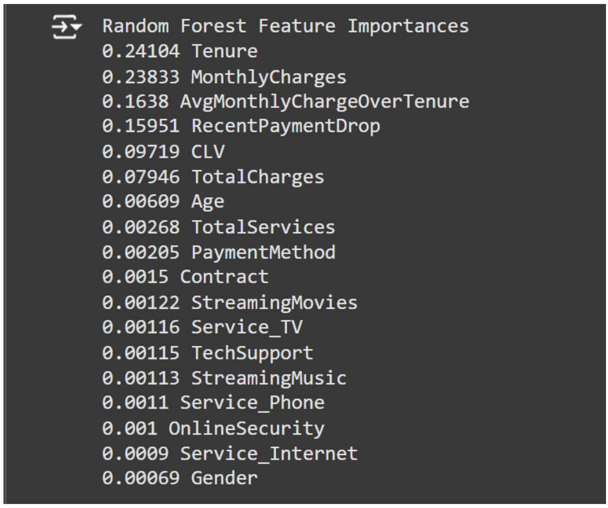
Then i used the models understand which features are most important for predicting churn. I understood that MonthlyCharges and Tenure are the most important features across all models. Logistic Regression and Random Forest spread the importance across more features, while Decision Tree and Gradient Boosting mainly focus on just a few features like MonthlyCharges and Tenure.



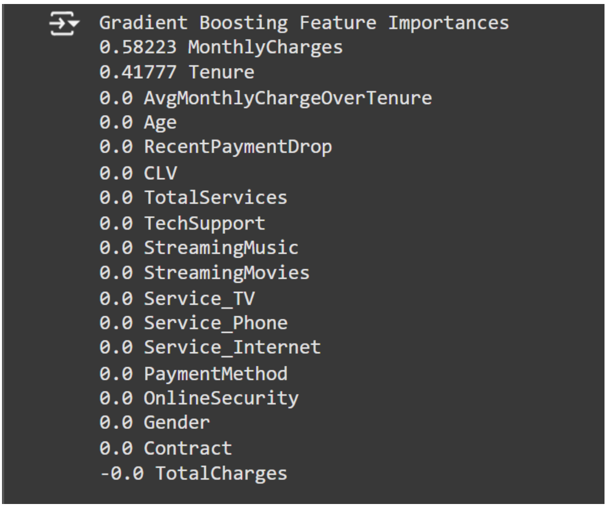
```
Logistic Regression Feature Coefficients
5.98187 MonthlyCharges
4.36535 Tenure
0.5487 AvgMonthlyChargeOverTenure
0.11675 Gender
0.1107 Age
0.05629 Service_Internet
0.05002 Contract
0.03029 StreamingMovies
0.02752 OnlineSecurity
0.02507 Service_Phone
0.01993 PaymentMethod
-0.00188 TechSupport
-0.06467 TotalServices
-0.09601 StreamingMusic
-0.23911 Service_TV
-1.00897 RecentPaymentDrop
-6.08876 TotalCharges
-13.78964 CLV
```



```
Decision Tree Feature Importances
0.58223 MonthlyCharges
0.41777 Tenure
0.0 TotalServices
0.0 TotalCharges
0.0 TechSupport
0.0 StreamingMusic
0.0 StreamingMovies
0.0 Service_TV
0.0 Service_Phone
0.0 Service_Internet
0.0 RecentPaymentDrop
0.0 PaymentMethod
0.0 OnlineSecurity
0.0 Gender
0.0 Contract
0.0 CLV
0.0 AvgMonthlyChargeOverTenure
0.0 Age
```



```
Random Forest Feature Importances
0.24104 Tenure
0.23833 MonthlyCharges
0.1638 AvgMonthlyChargeOverTenure
0.15951 RecentPaymentDrop
0.09719 CLV
0.07946 TotalCharges
0.00609 Age
0.00268 TotalServices
0.00205 PaymentMethod
0.0015 Contract
0.00122 StreamingMovies
0.00116 Service_TV
0.00115 TechSupport
0.00113 StreamingMusic
0.0011 Service_Phone
0.001 OnlineSecurity
0.0009 Service_Internet
0.00069 Gender
```



```
Gradient Boosting Feature Importances
0.58223 MonthlyCharges
0.41777 Tenure
0.0 AvgMonthlyChargeOverTenure
0.0 Age
0.0 RecentPaymentDrop
0.0 CLV
0.0 TotalServices
0.0 TechSupport
0.0 StreamingMusic
0.0 StreamingMovies
0.0 Service_TV
0.0 Service_Phone
0.0 Service_Internet
0.0 PaymentMethod
0.0 OnlineSecurity
0.0 Gender
0.0 Contract
-0.0 TotalCharges
```

9. DISCUSSION

The goal of this project was to predict which customers might churn. By knowing this in advance, the company can try to keep these customers and make them happier. In this project I looked at customer data, built models to predict churn, and improved the models to make better predictions.

Data Preprocessing: Cleaning the data by handling missing values, removing outliers, and converting categorical variables helped to improve the model's accuracy. These steps ensured that the models worked well with data, leading to better performance on both the training and testing datasets.

Exploratory Data Analysis (EDA): Through EDA, I gained valuable insights into the dataset, identifying key patterns and relationships that contributed to customer churn. Visualizations of numerical and categorical features helped shape the feature engineering process.

Feature Engineering: Creating new features helped the models make better predictions. By generating additional insights about customer behavior, the models could more accurately predict which customers might churn.

Modeling : I trained multiple models, including Logistic Regression, Decision Trees, and Random Forests. Each model was evaluated based on accuracy, precision, recall, and other metrics, demonstrating solid performance across multiple evaluation methods.

Model Tuning: Using Grid Search for hyperparameter tuning significantly improved the accuracy of the models. This shows the importance of tuning machine learning models to get better results and make more reliable predictions.

Model Interpretation: To understand how the models make their predictions, I used simple tools like feature importance and LIME. These tools helped me see which features were most important in the predictions. This made the models easier to understand and showed which factors, like MonthlyCharges and Tenure, were key in predicting customer churn.

9. Conclusion

The Random Forest model was the best overall. It had the highest accuracy and the best ROC-AUC score, meaning it was the most effective at telling the difference between customers who would leave and those who wouldn't. The model also showed that features like MonthlyCharges and Tenure were important in predicting churn. Overall, this model is the best choice to help the company find and keep customers who might churn.

10. References

1. How to Tune Hyperparameters with Grid Search
2. Scikit-learn: Machine learning in Python
3. Google Colab
4. Python Language Reference, version 3.8
5. Pandas
6. Matplotlib: A 2D Graphics Environment