Machine Learning Project

Predicting Customer Churn for a Telecommunications Company

Professor: Giacomo Fiumara

Student: Alikhan Alashybay 536353

Table of Content:

Introduction:

In a generation like ours where technology is growing ten times faster than anytime before in our history it's a big advantage to understand and use it to our benefit. The data on customer churn plays crucial role as it directly depends on the life of any business or startup. Moreover, with the technologies that we have prediction of churn is a matter of time and human resource, which means after the prediction, business holder may in advance be prepared for any type of situation. In this project we will go step by step on the predicting churn using python and machine learning models.

Understanding the Dataset:

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

import pickle
```

After the basic process of importing packets for our operations we move on to see the dataset. In general the dataset consists of the 3749 entries with 17 columns, it was found using .info() and .describe() functions. Need to mention that we are not going to consider columns Unnamed and CustomerId because in my opinion it’s not relative as it has no meaningful data for further data manipulation. To remove those columns we are going to use .drop(‘column_name’, axis=1).

```
data = data.drop(columns=['CustomerID', 'Unnamed: 0'])
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3749 entries, 0 to 3748
Data columns (total 15 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Age               3562 non-null   float64
 1   Gender            3749 non-null   object 
 2   Tenure            3749 non-null   int64  
 3   Service_Internet  3028 non-null   object 
 4   Service_Phone     3749 non-null   object 
 5   Service_TV        3749 non-null   object 
 6   Contract          3749 non-null   object 
 7   PaymentMethod     3562 non-null   object 
 8   MonthlyCharges    3749 non-null   float64
 9   TotalCharges      3749 non-null   float64
 10  StreamingMovies   3749 non-null   object 
 11  StreamingMusic    3749 non-null   object 
 12  OnlineSecurity    3749 non-null   object 
 13  TechSupport       3749 non-null   object 
 14  Churn             3749 non-null   object 
dtypes: float64(3), int64(1), object(11)
memory usage: 439.5+ KB
```

Data Preprocessing:

Now it’s time to handle missing values where .isnull() function comes to help.

```
data.isnull().sum()
```

```
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
```

```
miss_vals = data.isnull().sum()
miss_vals = miss_vals[miss_vals > 0]
miss_vals
```

```
Age                187
Service_Internet    721
PaymentMethod      187
dtype: int64
```

So in the result we get columns “Age”, “Service_Internet”, and “PaymentMethod” and from above operations where we know that “Age” is numerical values and the other two are categorical. So the next step to handle the missing values is to replace the median for numerical features(“Age”) and with random instances for the categorical features(“Service_Internet” and “PaymentMethod”).

For categorical features

```
# handle the missing values for categorical feats
for column in ['Service_Internet', 'PaymentMethod']:
  data[column] = data[column].fillna(random.choice(data[column].dropna().unique()))

print(data['Service_Internet'].isna().sum(), data['PaymentMethod'].isna().sum())
```

```
0 0
```

For numeric features

```
# handle the missing values for numerical feats
imputer = SimpleImputer(missing_values=np.nan, strategy='median')
data['Age'] = imputer.fit_transform(data['Age'].values.reshape(-1, 1))

data['Age'].isna().sum()
```

```
0
```

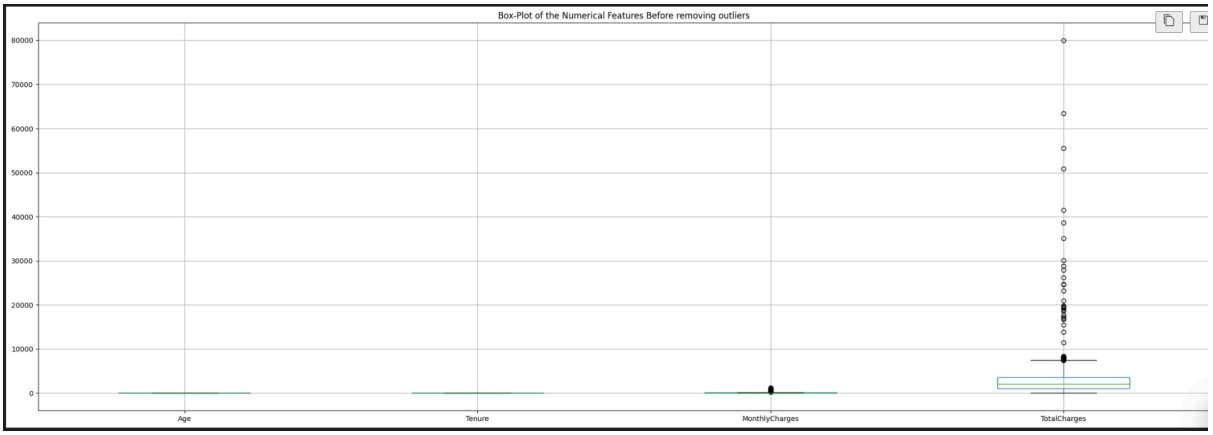
Label Encoding for the categorical features:

Need to mention that half of the dataset is type of object, some of them has meaningful values of an array consisting of values which cannot be changed as I believe it will be important to further operations. However, some of them are simple “Yes” or “No” values, which can be changed into 1 and 0 to simplify things. Now, we need to list all the categorical features using the LabelEncoder() function. We simply create an array with columns which we want to change and iterate using simple for loop.

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

label_encoder = LabelEncoder()
for feature in cat_features:
    data[feature] = label_encoder.fit_transform(data[feature])

data.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

Now let’s handle the outliers from the dataset which we prepared and see what can be done with them.



In general outliers are the data points that are different from other data which may influence the final result, that is one of the reasons why they

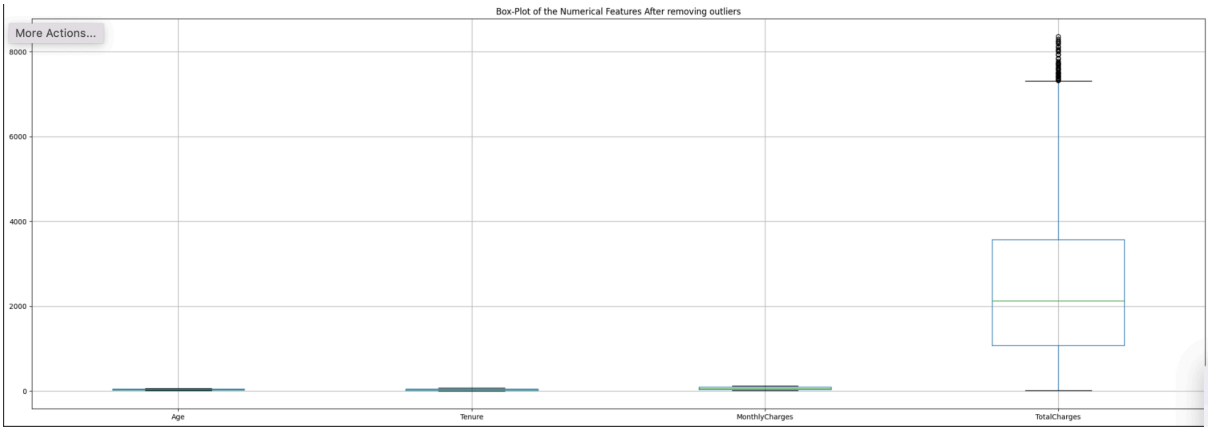
should be handled. From the box plot we are able to visually see that outliers exist in “MonthlyCharges” and “TotalCharges”. There are many ways to handle outliers from trimming to simply removing them, but we are going to use inter quartile range technique. To do so we need to create a custom function to handle outliers and then we will be able to apply it to all the numerical features and save index outliers.

```
def iqr_outliers(dataset,feature_name, multiplier=2):
    Q1 = dataset[feature_name].quantile(0.25)
    Q3 = dataset[feature_name].quantile(0.75)

    IQR = Q3 - Q1

    lwr_bound = Q1 - multiplier * IQR
    upp_bound = Q3 + multiplier * IQR

    ls = dataset.index[np.logical_or(dataset[feature_name]<lwr_bound,
                                     dataset[feature_name]>upp_bound)]
    return ls #return the indexes

outliers_detected={}
for i in numerical_features:
    outliers = iqr_outliers(data,i)
    outliers_detected[i] = outliers

    print('Variable',i)
    print(outliers)
    print(data[i].iloc[outliers])
    print('\n')
```

Click to add a breakpoint.

```
Variable Age
Index([], dtype='int64')
Series([], Name: Age, dtype: float64)


Variable Tenure
Index([], dtype='int64')
Series([], Name: Tenure, dtype: int64)


Variable MonthlyCharges
Index([  34,  106,  217,  253,  379,  590,  639,  667,  733,  787,  791,  938,
        944, 1043, 1151, 1258, 1261, 1343, 1619, 1904, 1952, 1982, 2356, 2495,
       2497, 2509, 2590, 2601, 2736, 2851, 2977, 3085, 3137, 3210, 3246, 3459,
       3514],
      dtype='int64')
34       492.8
106      773.8
217      255.1
253      862.2
379      929.8
590      933.3
639      491.0
667      329.8
733      597.3
```

After successful creation of the function we replace detected outliers with the median of the column. The reason why we didn't simply remove the outliers(imputation) is that the overall dataset is relatively small so it is better to basically replace them.



Scaling the Feature:

To go further on we need to range features as they are in different ranges right now. This may cause some issues in the training phase. In terms of range out we are going to use the MinMaxScaler() function. The reason why its not another scaler like StandardScaler() or Normalizer() is that in our case some of the features has time period which in StandardScaler() converts from -1 to 1, from where we are able to say that negative values are not usual for the time period. However, in the MinMaxScaler() we are able to range between 0 and 1.

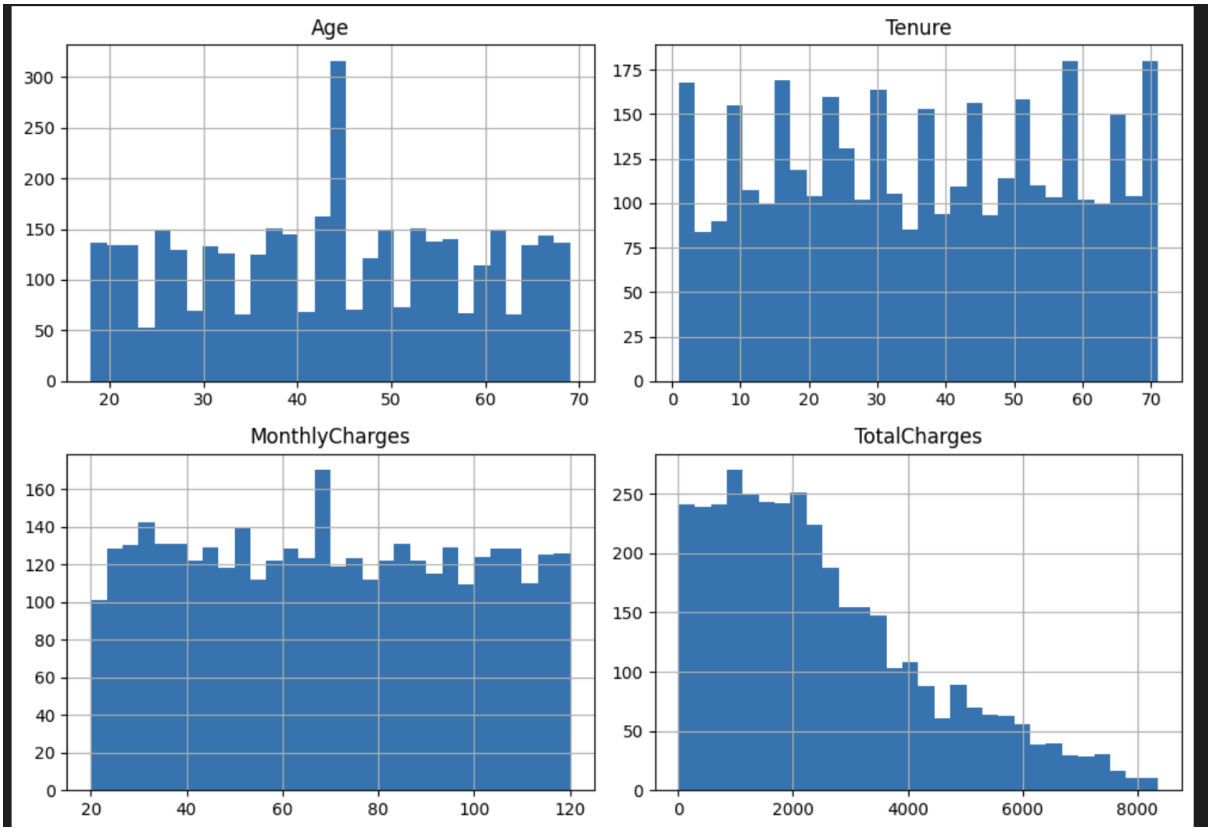
```
numerical_features += ['MonthlyCharges_per_Tenure', 'TotalCharges_per_Tenure']

scaler = MinMaxScaler()
data[numerical_features] = scaler.fit_transform(data[numerical_features])

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

X.head()
```

	Age	Gender	Tenure	Service_Internet	Service_Phone	Service_TV	Contract	PaymentMethod	MonthlyCharges	TotalCharges	StreamingMovies	StreamingMusic	OnlineSecurity
0	0.745098	1	0.171429	0	1	0	1	3	0.5188	0.110089	0	0	1
1	1.000000	1	0.171429	0	0	1	2	3	0.9099	0.172065	1	1	0
2	0.549020	1	0.842857	1	0	1	0	3	0.9674	0.837328	1	1	0
3	0.274510	0	0.800000	1	1	1	0	0	0.5816	0.532154	0	1	0
4	0.823529	1	0.728571	1	1	1	2	2	0.1033	0.186603	1	0	1

EDA:

EDA or in other words Exploratory Data Analysis is an essential step in understanding the dataset and uncovering bottom rocks that will affect our model.

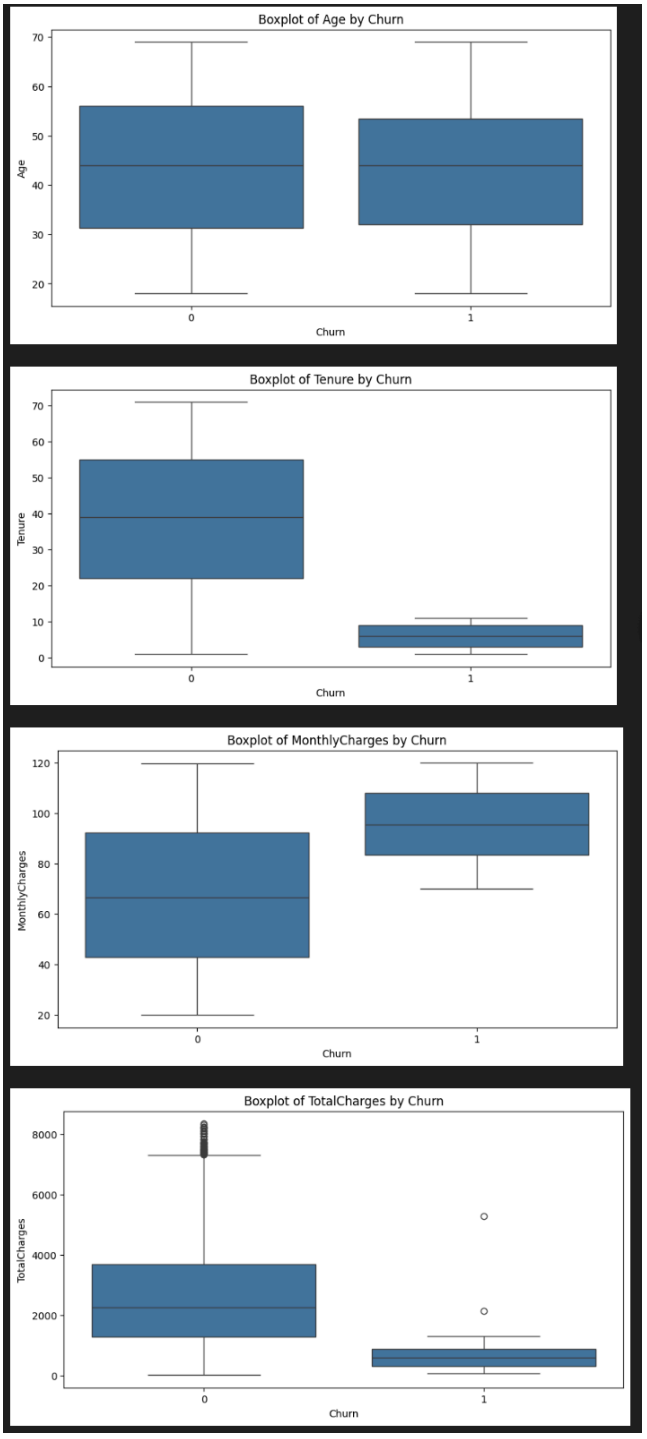
Distribution of Numerical Features:



We are going to use histogram to see distribution, frequencies and skewness. One of the examples is “TotalCharges” which reveals skewness aligned to the left indicating that most customers have rural charges less than 3000. This information reveals that a major portion of the customers base incurs relatively lower total charges.

Analysis of Categorical Features:
To analyze categorical features we used bar plots to see the frequency of each category. The bar plot indicates that a significant number of customers are involved in specific categories.

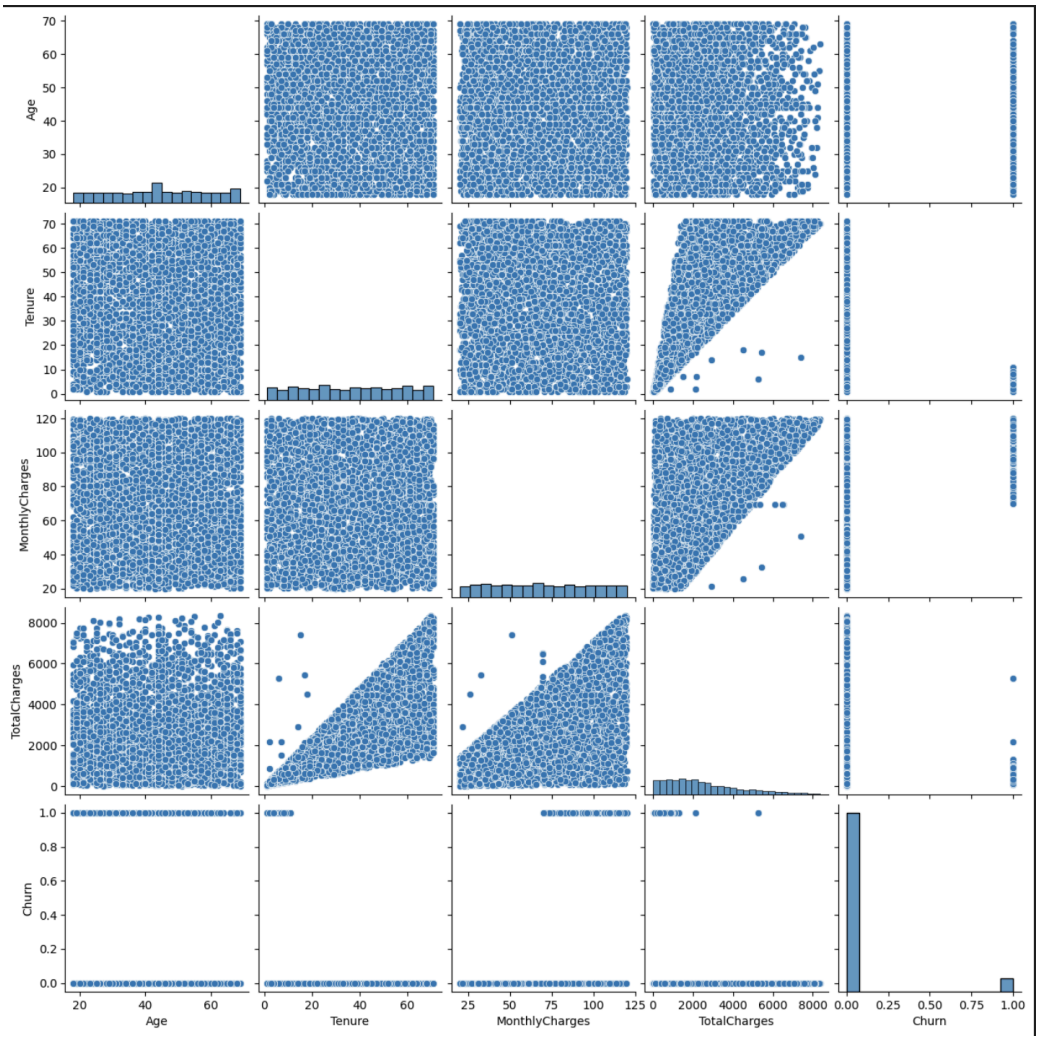
Relationship in features and target variable:



Now we want to see some correlation between a numerical feature and our target which is “Churn”. From the result we are able to visually see the relationship and trend between the feature and target. The box plot

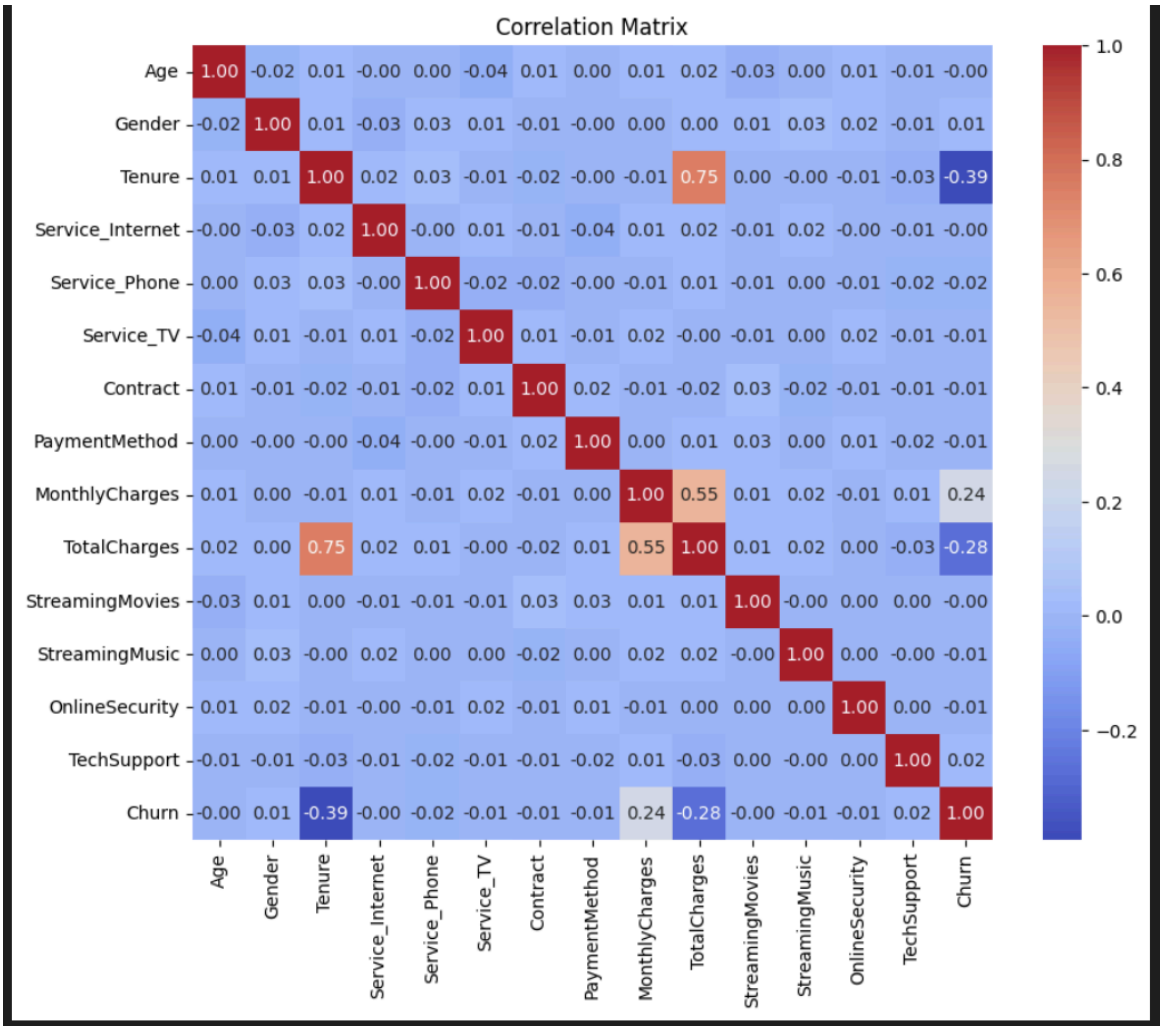
displays that customers with higher monthly charges are more likely to churn. This highlights “MonthlyCharges” as a significant influencer of the churn.

Pairwise Scatter plot:



The scatter plot will be useful to identify the relational trends, actual trends, outliers, and distribution. Next essential step in EDA will be a heat map which identifies the relation between the feature and other features. The scatter plot between the “MonthlyCharges” and “TotalCharges” and “Tenure” shows a positive linear relationship, confirming the correlation analysis. Generally this plot is able to help us to see distribution of data points and potential outliers.

Correlation Analysis:



The Heatmap of the correlation matrix displays the relationship between the features. “MonthlyCharges” and “TotalCharges” show strong positive correlation, indicating that customers with higher monthly charges tend to accumulate higher total charges over time. Need to add that coefficient ranges between 1 and -1 where first is strong correlation and second negative correlation.

Feature Eng:

This part involves creating new features from existing to improve performance of our model. This process is guided by domain knowledge and the insight from EDA.

Let's combine "MonthlyCharges_Tenure" which identifies high-cost users who might be at risk of churn, and "TotalCharges_Tenure" => highlights customers who have spent a lot over their tenure and might be more sensitive to service issues. Domain knowledge: "TotalService " => helps to identify customers having reliant on the company's services and might churn if dissatisfied. "SeniorCitizen" => senior citizens might have different churn behavior.

Modelling:

First step is to split the data into target variables and other features. Need to mention that in the first attempt the model was overfitting which may be caused because of the overall small data and three features with missing values. That's why one of the solutions was to use feature selection to select the most important features.

```
def select_features(X, y):
    model = RandomForestClassifier(random_state=42)
    rfe = RFE(model, n_features_to_select=10)
    X_final = rfe.fit_transform(X, y)
    final_features = X.columns[rfe.get_support()]
    print(final_features)
    return final_features
```

```
# Split data to train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

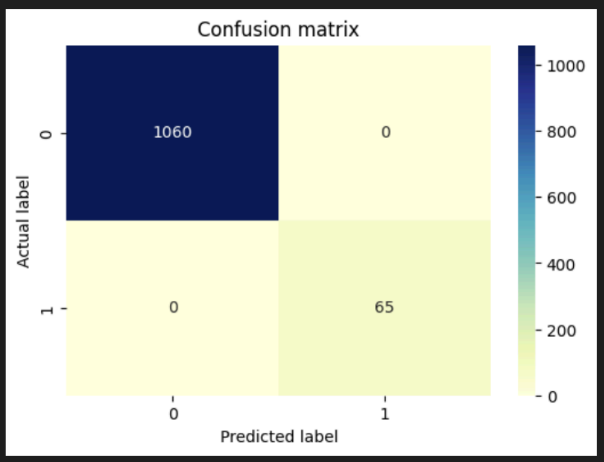
Moving on, 10 important features were selected from the dataset to train models. The next step is to actually train the machine to predict the customer churn. Furthermore, it's time to split the dataset into 70% and 30% using the train_test_split() function with a random state of 42 to ensure reproducibility.

Model testing techniques:
Few most popular tests are used on this modal which are: Random Forest, Decision Tree, Gradient Boots, and Logistic Regression. To make our work flow easier we create general function to train models. Need to mention that there is also interesting function GridSearchCV() function to split dataset in several parts and train with the best accuracy.

Evaluation Metrics:
The function feature_importance(arg: model) was created to see the importance of the features for that specific model to train the model. Other important function check_overfitting() we check the accuracy of the train and test.

```python
# Random Forest
rf_param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
rf = evaluate_model(RandomForestClassifier(random_state=42), rf_param_grid, X_train, y_train, X_test, y_test)
check_overfitting(rf, X_train, y_train, X_test, y_test)
feature_importance(rf, selected_features)
```

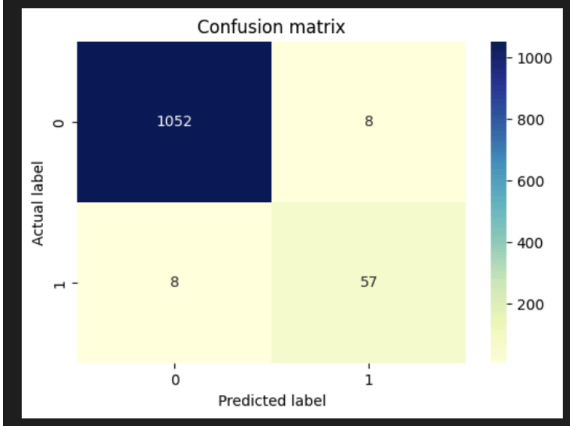
```
Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 50}
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1060
           1       1.00      1.00      1.00        65

    accuracy                           1.00      1125
   macro avg       1.00      1.00      1.00      1125
weighted avg       1.00      1.00      1.00      1125
```




```
Training Accuracy: 1.0000
Testing Accuracy: 1.0000
                    feature  importance
7  MonthlyCharges_per_Tenure    0.368184
5            MonthlyCharges    0.189950
1                    Tenure    0.189012
6              TotalCharges    0.125787
8    TotalCharges_per_Tenure    0.119861
0                       Age    0.003485
9             TotalServices    0.001470
2                Service TV    0.001091
```

```python
# Logistic Regression
lr_param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear']
}
lr = evaluate_model(LogisticRegression(random_state=42, max_iter=1000), lr_param_grid, X_train, y_train, X_test, y_test)
check_overfitting(lr, X_train, y_train, X_test, y_test)
```

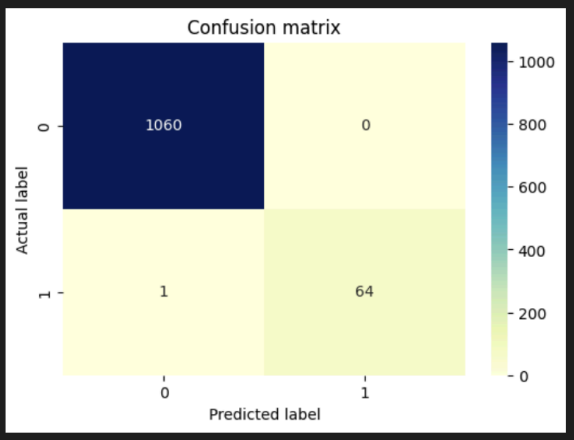
```
Best Parameters: {'C': 100, 'penalty': 'l1', 'solver': 'liblinear'}
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1060
           1       0.88      0.88      0.88        65

    accuracy                           0.99      1125
   macro avg       0.93      0.93      0.93      1125
weighted avg       0.99      0.99      0.99      1125
```




```
Training Accuracy: 0.9889
Testing Accuracy: 0.9858
```

```python
# Gradient Boosting
gb_param_grid = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
gb = evaluate_model(GradientBoostingClassifier(random_state=42), gb_param_grid, X_train, y_train, X_test, y_test)
check_overfitting(gb, X_train, y_train, X_test, y_test)
feature_importance(gb, selected_features)
```

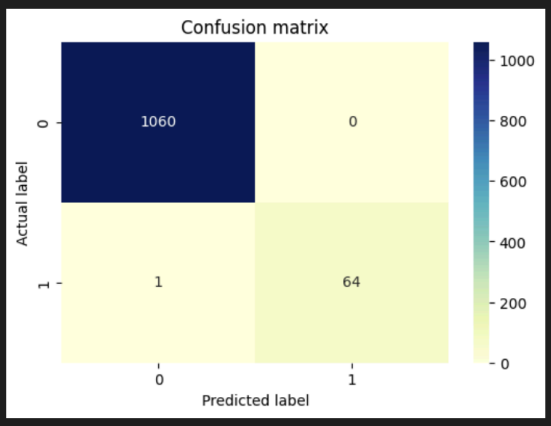
```
Best Parameters: {'learning_rate': 0.01, 'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1060
           1       1.00      0.98      0.99        65

    accuracy                           1.00      1125
   macro avg       1.00      0.99      1.00      1125
weighted avg       1.00      1.00      1.00      1125
```




```
Training Accuracy: 1.0000
Testing Accuracy: 0.9991
                     feature  importance
7  MonthlyCharges_per_Tenure    0.590972
5             MonthlyCharges    0.311060
1                     Tenure    0.087871
6                TotalCharges    0.008248
8    TotalCharges_per_Tenure    0.001849
0                        Age    0.000000
2                 Service_TV    0.000000
3                   Contract    0.000000
4              PaymentMethod    0.000000
```

```python
# Decision Tree
dt_param_grid = {
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
dt = evaluate_model(DecisionTreeClassifier(random_state=42), dt_param_grid, X_train, y_train, X_test, y_test)
check_overfitting(dt, X_train, y_train, X_test, y_test)
feature_importance(dt, selected_features)
```

```
Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2}
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1060
           1       1.00      0.98      0.99        65

    accuracy                           1.00      1125
   macro avg       1.00      0.99      1.00      1125
weighted avg       1.00      1.00      1.00      1125
```




```
Training Accuracy: 1.0000
Testing Accuracy: 0.9991
                    feature  importance
7  MonthlyCharges_per_Tenure    0.583291
5            MonthlyCharges    0.322769
1                    Tenure    0.093940
0                       Age    0.000000
2               Service_TV     0.000000
3                  Contract    0.000000
4             PaymentMethod    0.000000
6              TotalCharges    0.000000
8   TotalCharges_per_Tenure    0.000000
9             TotalServices    0.000000
```

Conclusion:

The main idea of this project was to predict churn using different techniques applied on the dataset so the results would be beneficial for a business, as it was said at the beginning, prediction of a customer flow may help in many different ways starting from creating situational plan to even closing.