

University of Messina



Bachelor of Data Analysis

ACADEMIC YEAR - 2022/2023

Machine Learning

(Project report)

Supervisor:
Prof. Giacomo Fiumara

Students:
Sahil Nakrani

Understanding the Dataset

The dataset consists of 3749 entries with 17 columns. (But here we are not considering 2 columns which are 'Unnamed: 0' and 'CustomerId'. Because which are just indexes for the entries.)

In this dataset, there is two columns 'Unnamed: 0' and 'CustomerId' which are only showing the index. So, I have removed it to not have unnecessary feature.

```
# removing the index column from the dataset.  
data = data.drop('Unnamed: 0', axis=1)  
data = data.drop('CustomerId', axis=1)
```

Basic Information and Statistical Details

```
<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
```

	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

Data Preprocessing

Handling the missing values

We utilize both the functions “isnull()” to inspect null values.

```
# Check for missing values
missing_values = data.isnull().sum()

# Display columns with missing values
missing_values = missing_values[missing_values > 0]
missing_values
```

Age	187
Service_Internet	721
PaymentMethod	187
dtype:	int64

Here, feature ‘Age’ is numerical feature and [‘Service_Internet’, ‘PaymentMethod’] are categorical features.

- Replacing the numerical missing values with the median of the column.

```
imputer = SimpleImputer(missing_values=np.nan, strategy='median')
data['Age'] = imputer.fit_transform(data['Age'].values.reshape(-1, 1))
```

- Replacing the categorical missing values with the random instance of the column.

```
for column in ['Service_Internet', 'PaymentMethod']:
    data[column] = data[column].fillna(random.choice(data[column].dropna().unique()))
```

Here, we can also replace categorical feature’s missing values with mode (most frequent instance) but we have replaced with random instance. Just because already mode is having majority of distribution and as per me, it is not good to put again the mode to let the model to rely on single instance (category).

Label Encoding for the categorical features

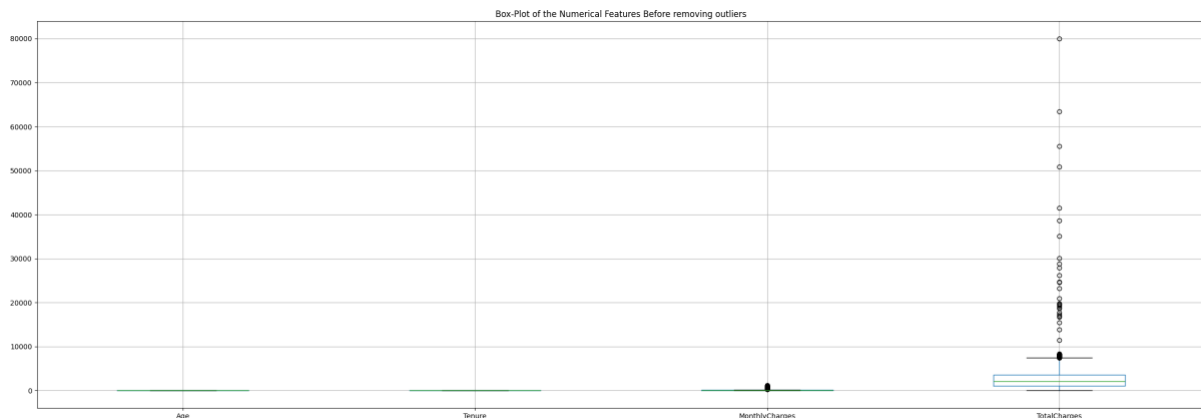
For the Exploratory Data Analysis (EDA) and for the model training part. For example terraforming the values of categorical value 'yes' and 'no' to 1 and 0.

First, I have listed all the categorical features. I'm using '**LabelEncoder()**' function to encode the categories of the features.

```
cat_features = ['Gender', 'Service_Internet', 'Service_Phone', 'Service_TV', 'Contract', 'PaymentMethod',  
'StreamingMovies', 'StreamingMusic', 'OnlineSecurity', 'TechSupport']  
  
# Label encoding for binary categorical feature  
label_encoder = LabelEncoder()  
for feature in cat_features:  
    data[feature] = label_encoder.fit_transform(data[feature])
```

Handling the Outliers

Outliers are datapoints that deviate significantly from the rest of the dataset and can have a substantial impact on analysis.



In the above box-plot, you can see there are outliers in two right sided features which are 'MonthlyCharges' and 'TotalCharges'. And we are going to handle it with the technique called inter quartile range.

So, first I have made the function to handle the outliers and then I will apply it on all the numerical features and I will save the index of the outlier.

```
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')
```

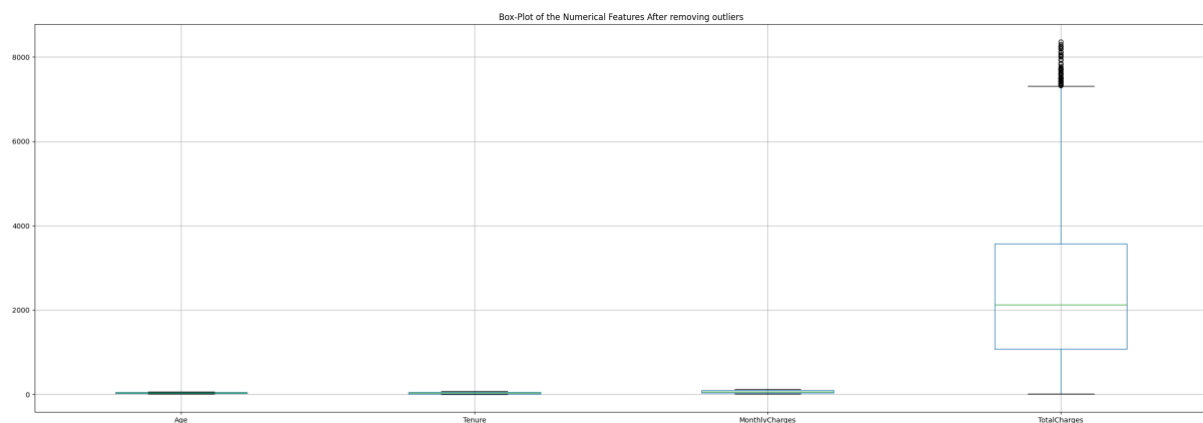
For example :

```
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],
```

Then, I have just replaced that detected outliers with the median of the column. We can also remove the outliers, but since our dataset is small so it is better to replace it with middle value not to remove.

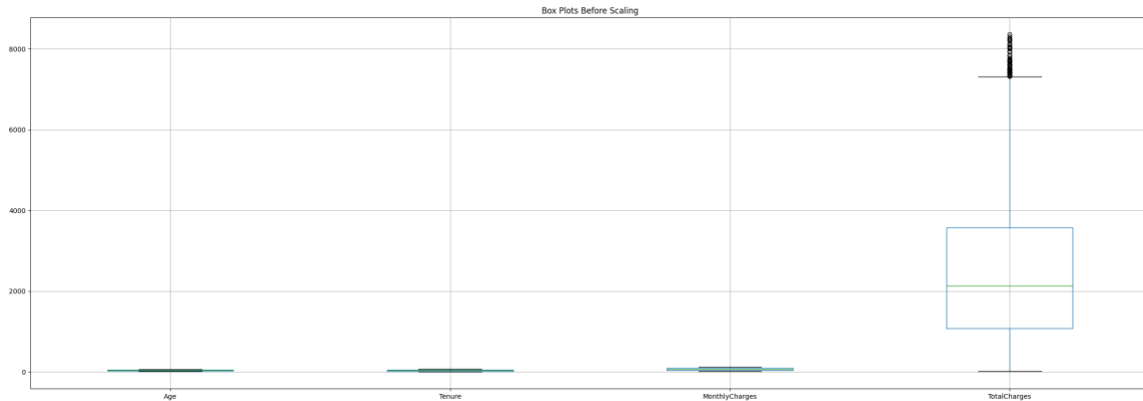
```
for i in numerical_features:
    data[i] = data[i].replace(data[i].iloc[outliers_detected[i]].values, data[i].median())
```

Let's see now the box-plot.



Scaling the Feature

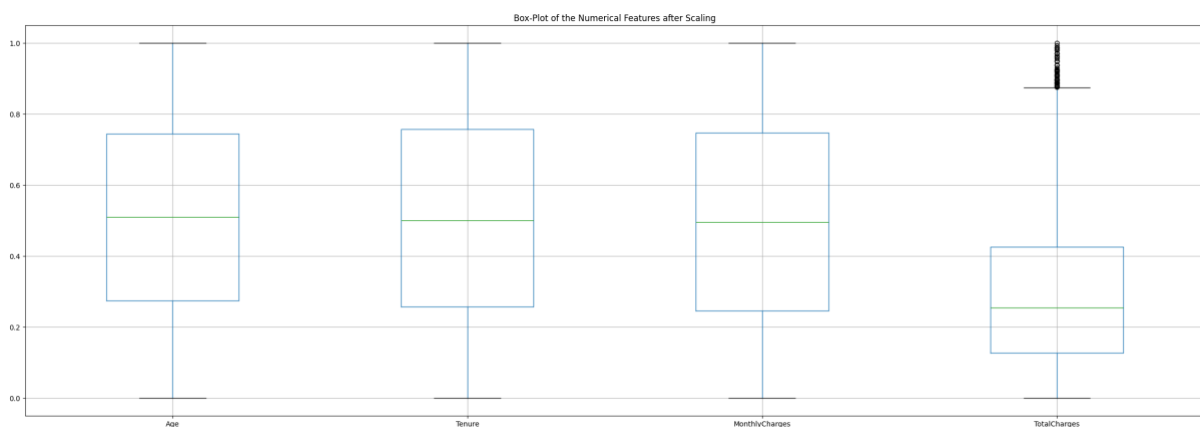
Before scaling the data, features are not in one range. They are in different range which is not good for training phase.



here, I'm using a `MinMaxScaler()` to scale the data not a `Standard Scaler`. Because we have some features which is showing time period and standard scaler is converting the range between -1 and 1. Negative values are not usual for the time period. That's why here I have used `MinMaxScaler` to have range between 0 and 1.

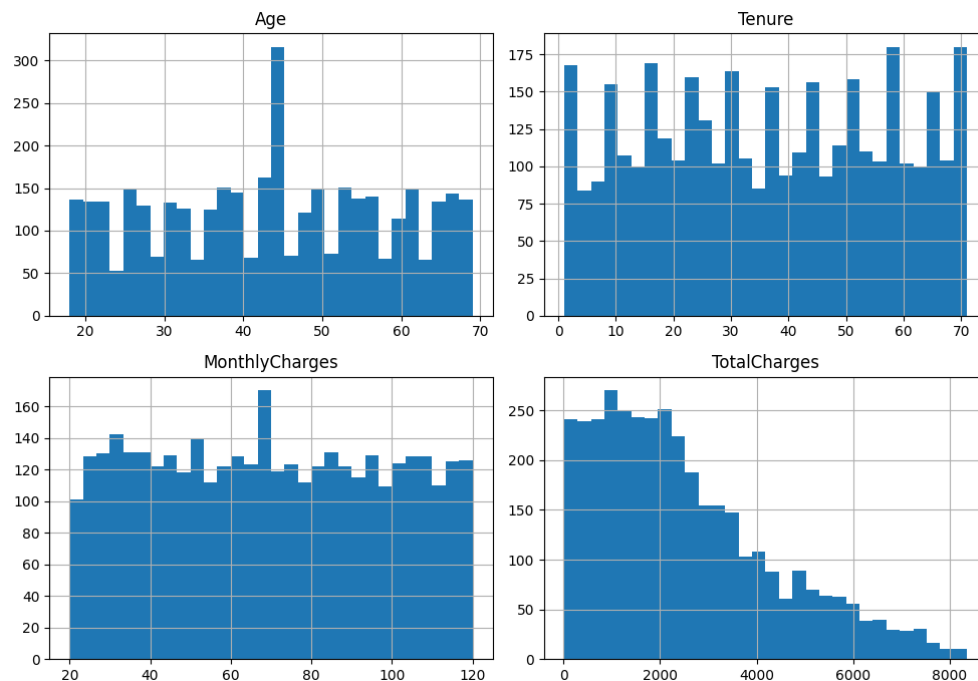
```
scaler = MinMaxScaler()
data[numerical_features] = scaler.fit_transform(data[numerical_features])
```

Here is the box-plot after scaling the numerical features.



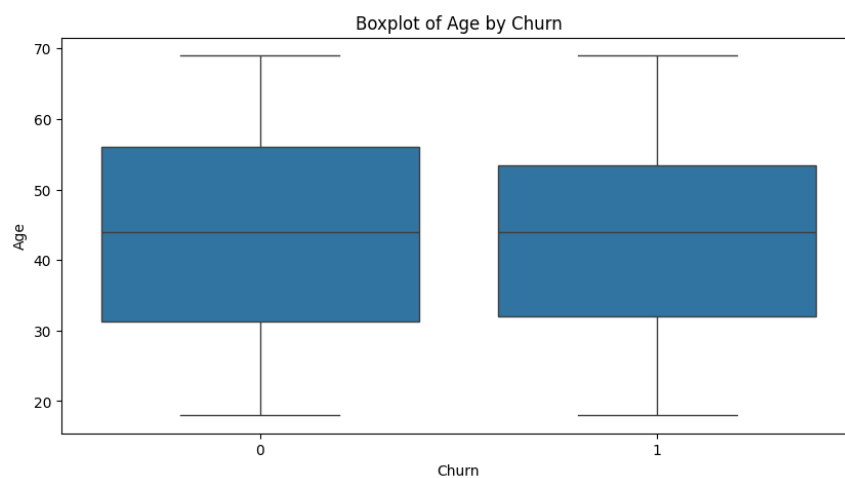
EDA (Exploratory Data Analysis)

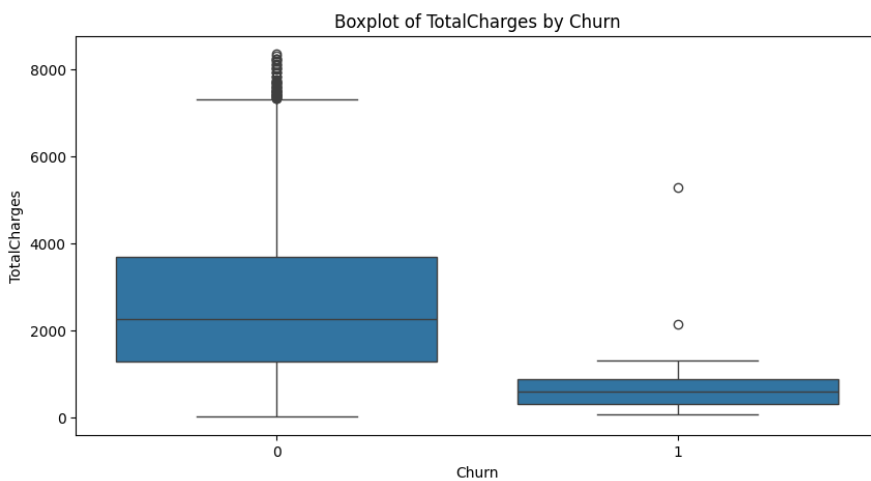
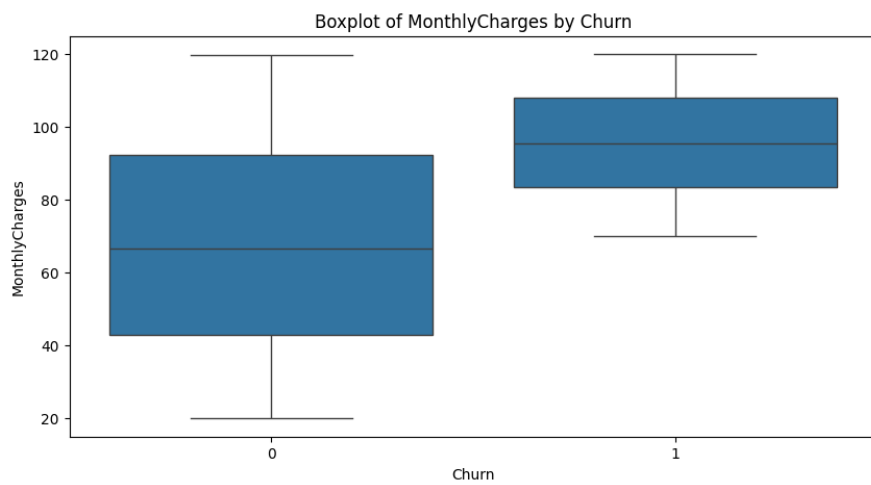
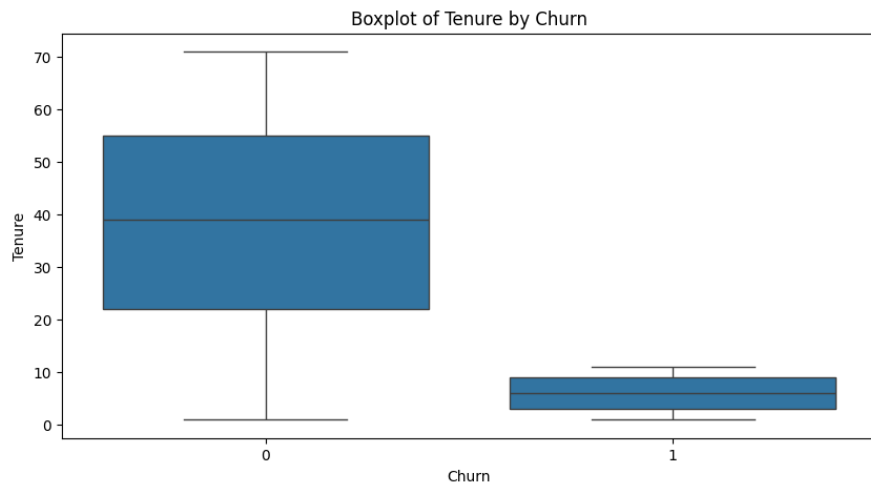
First, we are going to make a distribution graph for the numerical features using histogram.



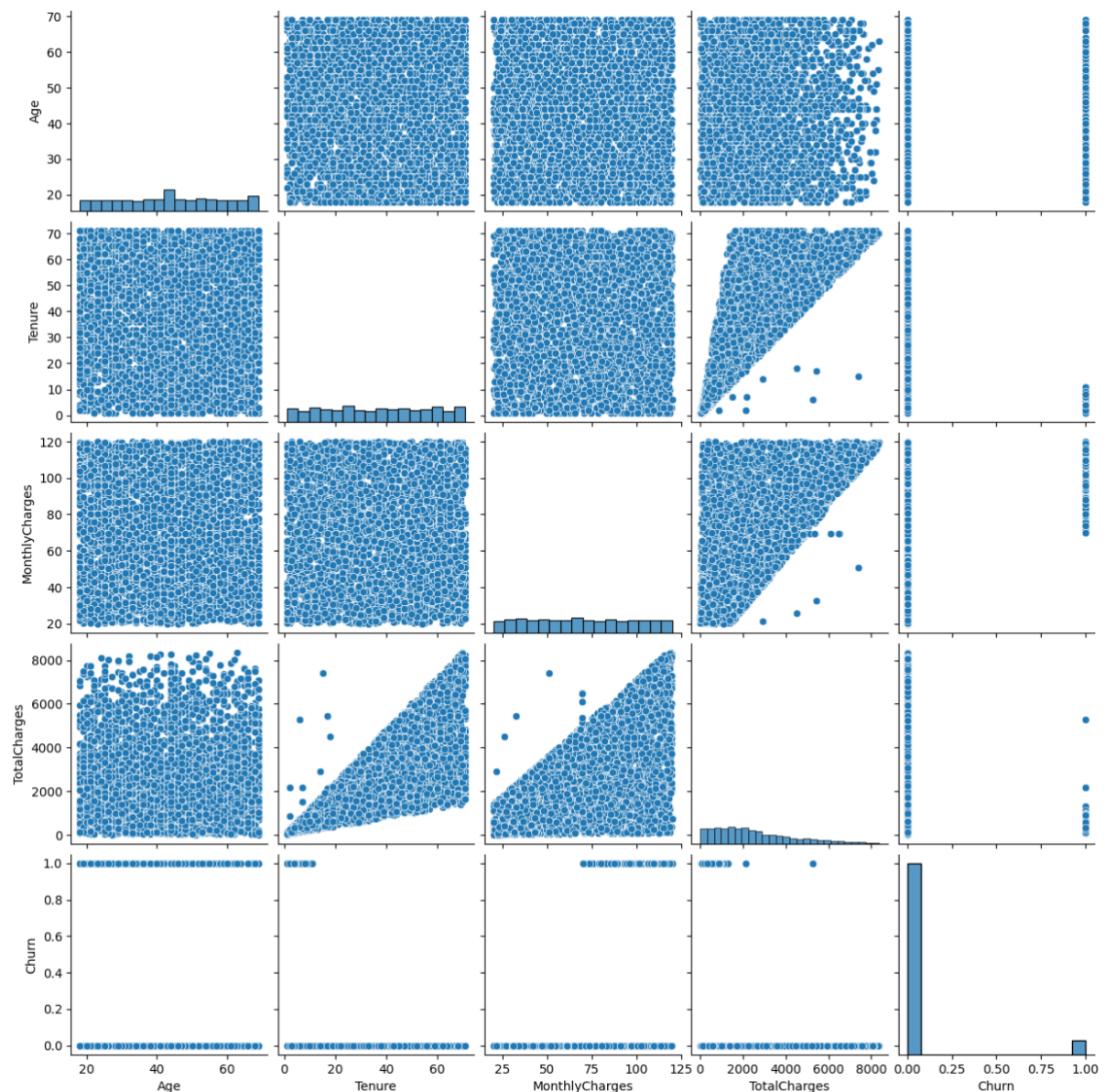
By this histogram, we can see the distribution, frequencies, and also skewness in the distribution. For example, in the distribution of the TotalCharges, we can see that it is left side skewed, so we can say that majority of people have total charges less than 3000.

Now here is the relation between a numerical features and the target feature which is Churn. This relation is visualized with the box-plot.





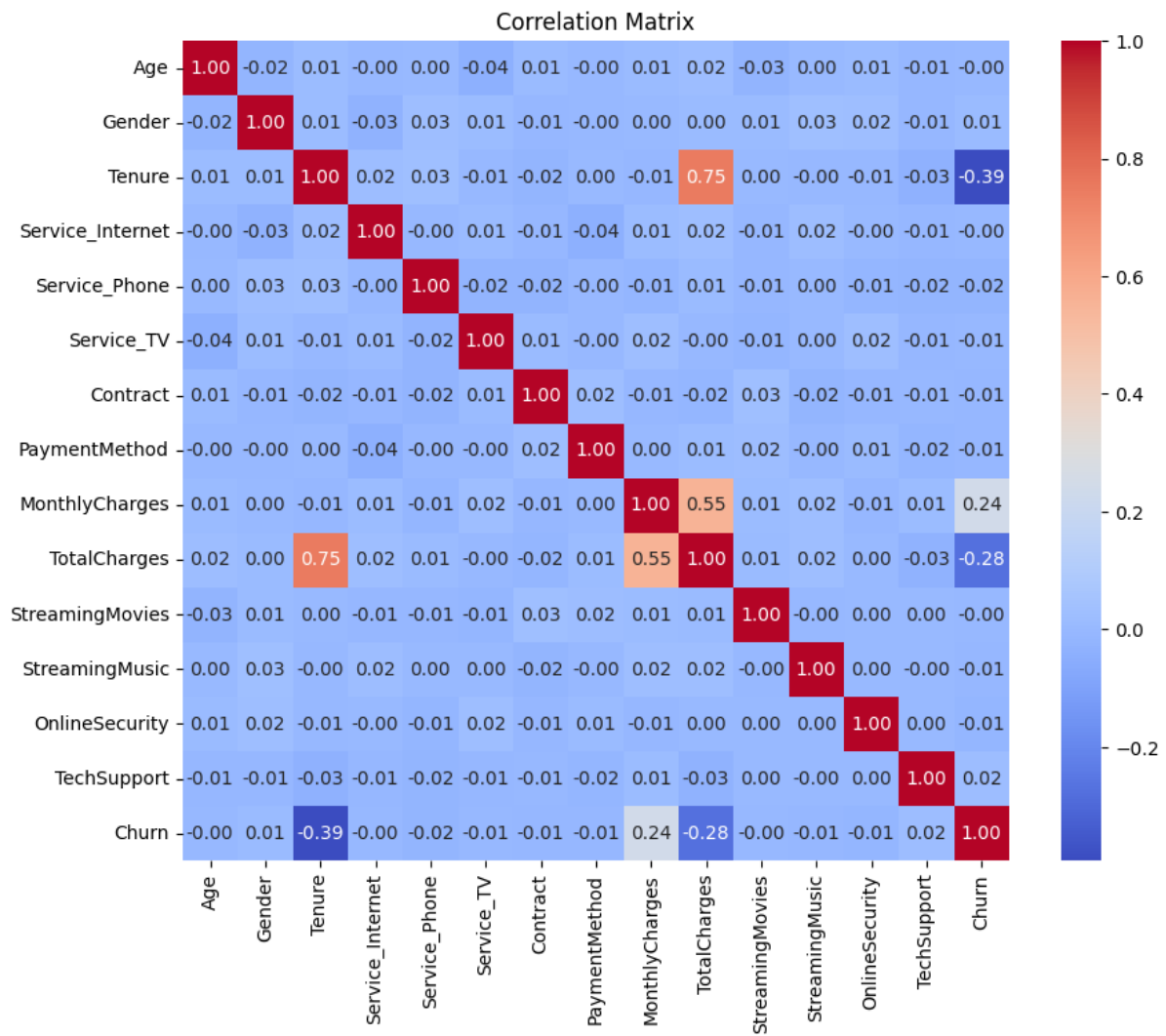
Also I have visualized the relation with the help of pairwise scatter plot. Where we can identify the relational trends, actual trends in the data, and also distribution and outliers. Here, I have mentioned the pairwise scatter plot below.



Then, let's create a heatmap of the correlation matrix. This is very important step of EDA because, by correlation matrix, we can identify the relation between the feature and other features and asl between the feature and target feature.

By correlation matrix, we can also look the percentage of dependencies of the features on other features. Because, correlation coefficient ranges between -1 and 1. Where 1 mean strongly positive correlation and -1 means strongly negative correlation.

So, let's visualize the heatmap of the correlation matrix.



Feature Engineering

I have already scaled the features, and also have put the labels for the categorical features.

But now I'm going to create some extra additional features which can effect this learning process or can affect the predictions.

Combine Existing Features :

- **MonthlyCharges_to_Tenure** : identifies high-cost users who might be at risk of churn.
- **TotalCharges_to_Tenure** : highlights customers who have spent a lot over their tenure and might be more sensitive to service issues.

Domain Knowledge :

- **TotalServices** : helps to identify customers having reliant on the company's services and might churn if dissatisfied.
- **isSeniorCitizen** : senior citizens might have different churn behaviour.

```
# Function to engineer new features
def feature_engineering(data):

    data['MonthlyCharges_per_Tenure'] = data['MonthlyCharges'] / (data['Tenure'] + 1)

    data['TotalCharges_per_Tenure'] = data['TotalCharges'] / (data['Tenure'] + 1)

    # Total number of services
    data['TotalServices'] = (data['Service_Internet'] + data['Service_Phone'] + data['Service_TV'] +
                             data['StreamingMovies'] + data['StreamingMusic'] + data['OnLineSecurity'] + data
                             ['TechSupport'])

    # Flag for senior citizen status
    data['IsSeniorCitizen'] = (data['Age'] >= 65).astype(int)

    return data

# Apply feature engineering
data = feature_engineering(data)
```

Modelling and Evaluation

Frist we are going to split the data into target variables and other features.

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

I have tried to train model now, but it is overfitting. May be because we have very less data and three features with high missing values. So, that's why I have used feature selection technique to select most important features.

```
def select_features(X, y):
    k_initial = 20
    selector = SelectKBest(score_func=f_classif, k=k_initial)
    X_new = selector.fit_transform(X, y)
    selected_features = X.columns[selector.get_support()]

    model = RandomForestClassifier(random_state=42)
    rfe = RFE(model, n_features_to_select=10)
    X_final = rfe.fit_transform(X_new, y)
    final_features = selected_features[rfe.get_support()]
    return X_final, final_features
```

So, we are selecting 10 most important features from our dataset to train the model on. Then, we are splitting the data into training and test sets using `train_test_split()` function.

```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Here, `test_size=0.3` means 30% testing set and 70% training set. `Random_state=42` means every time it is using same sample set for training and testing set.

Here, I'm testing the model by creating 4 different models :

- Random Forest Classifier
- Logistic Regression Classifier
- Gradient Boost Classifier
- Decision Tree Classifier

But first, I'm making a common function to train all the models by that function. Which is mentioned below.

```

def evaluate_model(model, param_grid, X, y):
    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

    # GridSearchCV for hyperparameter tuning
    grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='accuracy')
    grid_search.fit(X_train, y_train)

    # Training the model with the best parameters
    best_model = grid_search.best_estimator_

    # Predictions
    predictions = best_model.predict(X_test)

    # Best parameters
    best_parameters = grid_search.best_params_
    print("Best Parameters:", best_parameters)

    print("Classification Report:")
    print(classification_report(y_test, predictions))

    # Evaluation
    conf_matrix = confusion_matrix(y_test, predictions)

    # Visualization of Confusion Matrix
    plt.figure(figsize=(6, 4))
    sns.heatmap(pd.DataFrame(conf_matrix), annot=True, cmap="YlGnBu", fmt='g')
    plt.title('Confusion matrix')
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
    plt.show()

    return best_model

```

Here, I'm also using a GridSearchCV() function to split the dataset in several part and train on each split to get best model and best parameters which will suitable to improve the accuracy of the model.

And by this function as you can see, I'm printing best parameters, classification report, confusion matrix, and returning the best model.

```

def feature_importance(model):
    # Feature importances from Best Random Forest
    importances = model.feature_importances_
    feature_names = X.columns
    feature_importance_df = pd.DataFrame({'feature': feature_names, 'importance': importances})
    print(feature_importance_df.sort_values(by='importance', ascending=False))

```

By the upper function, we can see the importance of the features for that specific model to train the model. And by below mentioned function, by checking the accuracy of the train and test, it is overfitting or not.

```
# Check overfitting
def check_overfitting(model, X_train, y_train, X_test, y_test):
    train_accuracy = model.score(X_train, y_train)
    test_accuracy = model.score(X_test, y_test)
    print(f"Training Accuracy: {train_accuracy:.4f}")
    print(f"Testing Accuracy: {test_accuracy:.4f}")
```

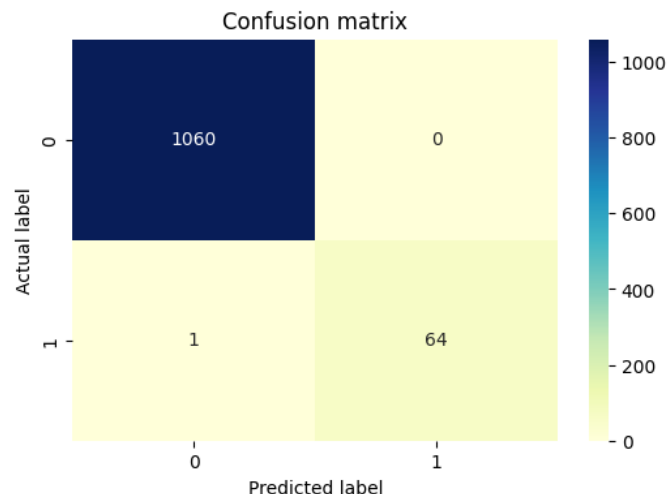
And here is the example that how I'm training the model with some range of parameters, and with the help of functions which I made before. This is the example of the random forest classifier and below this I have mentioned result of this model. Like this I have trained other models and got the results which you can see in the code file.

```
# Example usage for 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, y)
check_overfitting(rf, X_train, y_train, X_test, y_test)
feature_importance(rf)
```

```
Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 100}
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
 macro avg          1.00
weighted avg          1.00
```



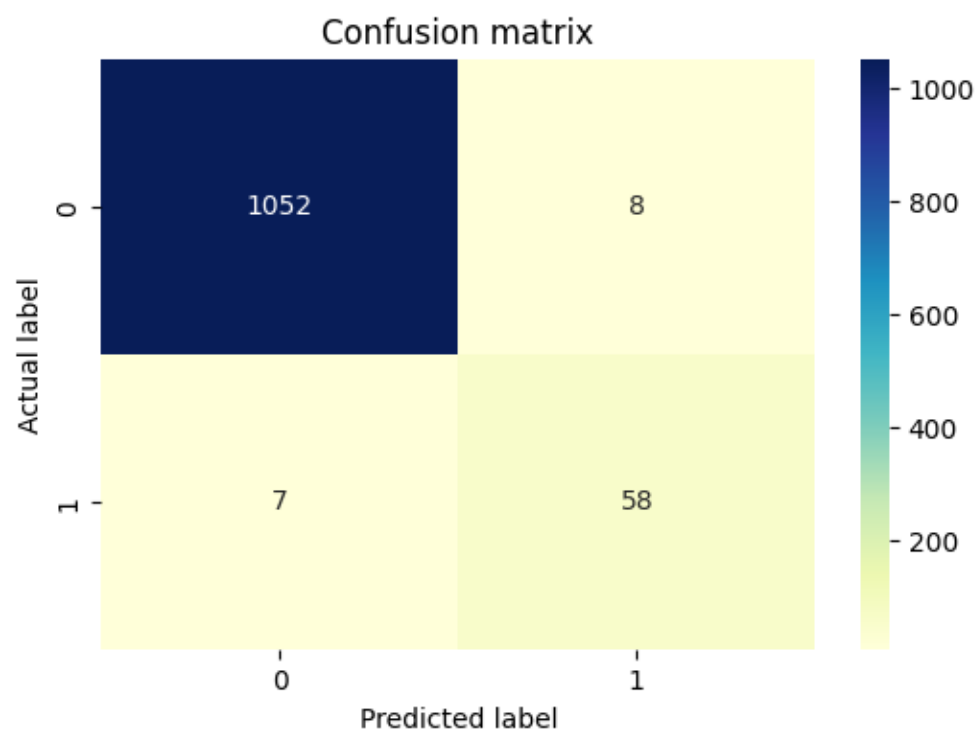
```
Training Accuracy: 1.0000
Testing Accuracy: 0.9991
```

	feature	importance
14	MonthlyCharges_per_Tenure	0.268580
2	Tenure	0.257185
8	MonthlyCharges	0.184816
9	TotalCharges	0.138522
15	TotalCharges_per_Tenure	0.132573
0	Age	0.004617
16	TotalServices	0.002740
7	PaymentMethod	0.002008
6	Contract	0.001818
5	Service_TV	0.001092
12	OnlineSecurity	0.001070
4	Service_Phone	0.000990
10	StreamingMovies	0.000866
1	Gender	0.000838
11	StreamingMusic	0.000727
13	TechSupport	0.000713
3	Service_Internet	0.000515
17	IsSeniorCitizen	0.000330

But in the Logistic Regression, I have noticed that it is giving reliable result.

```
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	0.89	0.89	65
accuracy			0.99	1125
macro avg	0.94	0.94	0.94	1125
weighted avg	0.99	0.99	0.99	1125



Training Accuracy: 0.9882
Testing Accuracy: 0.9867

Conclusion

In this project, we undertook a comprehensive approach to predict customer churn using various machine learning models. We started with data preprocessing and exploratory data analysis to understand the key characteristics and patterns within the dataset. Following this, we implemented and evaluated several machine learning models, including Logistic Regression, Decision Trees, Random Forest, and Gradient Boosting, using a range of performance metrics such as accuracy, precision, recall, F1-score.

Our initial findings revealed that while some models performed well on training data, they exhibited signs of overfitting when evaluated on the testing data. This was evident from the significant discrepancies between training and testing accuracies. To address this, we applied cross-validation techniques and observed the models' performance across multiple folds, ensuring robust and reliable performance estimates. But nothing changes that much so as per my opinion, there is less data in the dataset that's why it may overfit in this training process.

Further, we engaged in hyperparameter tuning using GridSearchCV to optimize the models' performance. This process allowed us to refine the model parameters and achieve improved evaluation metrics, reducing the overfitting issue significantly. Specifically, the Random Forest model, after tuning, demonstrated a balanced trade-off between bias and variance, making it one of the top performers in our analysis.

As per the dataset target feature, these findings offer strategic insights into customer behaviour, enabling the organization to implement targeted interventions to reduce churn and enhance customer retention.