Telco Customer Churn Prediction

Table of Contents

Business Problem

Customer turnover is the corporate issue we want predictive analytics to help with. Churn is the result of consumers stopping to deal with a company. Retaining current consumers is usually more affordable for companies than attracting new ones. Predicting turnover so enables businesses to pinpoint which consumers could be prone to leave. This will help them to be proactive in improving retention, raising client happiness, and generating income while lowering the expenses of customer acquisition. In subscription-based businesses where continuous relationships and long-term income sources are vital, addressing turnover is extremely important.

Goal

This project aims to create a prediction model based on consumer data to find which clients most likely will leave. Using past data will enable us to forecast future turnover and support the company in acting pro-early. Improved client retention, lower turnover rates, and generally greater company results will follow from this.

How predictive analysis helps the Business

By means of analysis of the elements influencing customer behaviour and identification of the patterns linked with turnover, predictive analytics can help solve the churn issue. Predictive models like Decision Trees or K-Nearest Neighbours (KNN) can categorize consumers into those likely to churn and those who will remain loyal by analyzing data such tenure, contract type, monthly expenses, and customer support contacts.

For instance, the given analysis's Decision Tree model attained an accuracy of over 79%, meaning that it can rather fairly forecast whether a customer is going to leave.

Early identification of at-risk consumers made possible by predictive analytics allows the organization to use loyalty programs, customer service enhancements, or focused incentives to keep them, hence lowering turnover and increasing general profitability.

Data Exploration & cleaning

Data exploration begin with identifying the columns of the data and understanding the data type of the variable and exploring them, this can be done by the function “dataset.info ()” and the result is as shown in the figure 1.

Fig 1:

Info of the dataset

```
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        7043 non-null   object
 1   gender            7043 non-null   object
 2   SeniorCitizen     7043 non-null   int64
 3   Partner           7043 non-null   object
 4   Dependents        7043 non-null   object
 5   tenure            7043 non-null   int64
 6   PhoneService      7043 non-null   object
 7   MultipleLines     7043 non-null   object
 8   InternetService   7043 non-null   object
 9   OnlineSecurity    7043 non-null   object
 10  OnlineBackup      7043 non-null   object
 11  DeviceProtection  7043 non-null   object
 12  TechSupport       7043 non-null   object
 13  StreamingTV       7043 non-null   object
 14  StreamingMovies   7043 non-null   object
 15  Contract          7043 non-null   object
 16  PaperlessBilling  7043 non-null   object
 17  PaymentMethod     7043 non-null   object
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object
 20  Churn             7043 non-null   object
```

The selected dataset contains 20 columns with 7043 rows, most of the rows are objects and few of them are int and float. Let us further understand the data by using the command "dataset.head ()" as shown in fig 2.

Fig 2:

Understanding the data by looking at the first few columns

customerI	gender	SeniorCiti	Partner	Dependen	tenure	PhoneSer	MultipleLi	InternetSe	OnlineSec	...	DevicePro	TechSupp	Streaming	Streaming	Contract	Paperless	PaymentM	MonthlyCl	TotalCharges	Churn
7590-VHV	Female	0	Yes	No	1	No	No phone	DSL	No	...	No	No	No	No	Month-to-r	Yes	Electronic	29.85	29.85	No
5575-GNV	Male	0	No	No	34	Yes	No	DSL	Yes	...	Yes	No	No	No	One year	No	Mailed che	56.95	1889.5	No
3668-QPY	Male	0	No	No	2	Yes	No	DSL	Yes	...	No	No	No	No	Month-to-r	Yes	Mailed che	53.85	108.15	Yes
7795-CFO	Male	0	No	No	45	No	No phone	DSL	Yes	...	Yes	Yes	No	No	One year	No	Bank trans	42.3	1840.75	No
9237-HQIT	Female	0	No	No	2	Yes	No	Fiber optic	No	...	No	No	No	No	Month-to-r	Yes	Electronic	70.7	151.65	Yes

Looking at the data type of the features and the values of the features, we found that there's and issue with "Total charges" feature you can see in the fig3 and rest all the features are fine.

Fig 3:

Issue with Total charges feature

```
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        7043 non-null   object
 1   gender            7043 non-null   object
 2   SeniorCitizen     7043 non-null   int64
 3   Partner           7043 non-null   object
 4   Dependents        7043 non-null   object
 5   tenure            7043 non-null   int64
 6   PhoneService      7043 non-null   object
 7   MultipleLines     7043 non-null   object
 8   InternetService   7043 non-null   object
 9   OnlineSecurity    7043 non-null   object
 10  OnlineBackup      7043 non-null   object
 11  DeviceProtection  7043 non-null   object
 12  TechSupport       7043 non-null   object
 13  StreamingTV       7043 non-null   object
 14  StreamingMovies   7043 non-null   object
 15  Contract          7043 non-null   object
 16  PaperlessBilling  7043 non-null   object
 17  PaymentMethod     7043 non-null   object
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object
 20  Churn             7043 non-null   object
```

upport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges	hurn
No	No	No	Month-to-month	Yes	Electronic check	29.85	29.85	No
No	No	No	One year	No	Mailed check	56.95	1889.5	No
No	No	No	Month-to-month	Yes	Mailed check	53.85	108.15	Yes
Yes	No	No	One year	No	Bank transfer (automatic)	42.30	1840.75	No
No	No	No	Month-to-month	Yes	Electronic check	70.70	151.65	Yes

When you look at the data type of the Total Charges it says “object”, but when you look at the values in that column, all the columns are in numbers, this suggest that those numbers are stored in the form of string as you see in the fig 4, next step is to convert those string values in float, this can be done by the function as shown in the fig 5.

Fig 4:

Exploring the values in Total Charges column

```
# Total charges are in object instead of float
ds.TotalCharges.values
```

```
array(['29.85', '1889.5', '108.15', ..., '346.45', '306.6', '6844.5'],
      dtype=object)
```

Fig 5:

Converting string to float

```
#converting string to float
ds.TotalCharges = pd.to_numeric(ds.TotalCharges, errors='coerce')
```

After converting the feature to float, let us identify the data type and explore further

Fig 6:

```
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        7043 non-null   object
 1   gender            7043 non-null   object
 2   SeniorCitizen     7043 non-null   int64
 3   Partner           7043 non-null   object
 4   Dependents        7043 non-null   object
 5   tenure            7043 non-null   int64
 6   PhoneService      7043 non-null   object
 7   MultipleLines     7043 non-null   object
 8   InternetService   7043 non-null   object
 9   OnlineSecurity    7043 non-null   object
 10  OnlineBackup      7043 non-null   object
 11  DeviceProtection  7043 non-null   object
 12  TechSupport       7043 non-null   object
 13  StreamingTV       7043 non-null   object
 14  StreamingMovies   7043 non-null   object
 15  Contract          7043 non-null   object
 16  PaperlessBilling  7043 non-null   object
 17  PaymentMethod     7043 non-null   object
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7032 non-null   float64
 20  Churn             7043 non-null   object
dtypes: float64(2), int64(2), object(17)
memory usage: 1.1+ MB
```

Total charges are converted from string to float, when you look at the non-null values it says 7032 but the rows in the dataset are 7043, this suggests that there are some missing values in the dataset.

Handling Data Quality Issues

Missing Values

We can find the missing values in the dataset by using the command “isnull(). sum()” it returns all the missing values that contain in the dataset. We have 11 missing values in our dataset as shown in the figure 7.

Fig 7:

Identifying the missing values

```
#checking for null values
ds.isnull().sum()
```

	0
customerID	0
gender	0
SeniorCitizen	0
Partner	0
Dependents	0
tenure	0
PhoneService	0
MultipleLines	0
InternetService	0
OnlineSecurity	0
OnlineBackup	0
DeviceProtection	0
TechSupport	0
StreamingTV	0
StreamingMovies	0
Contract	0
PaperlessBilling	0
PaymentMethod	0
MonthlyCharges	0
TotalCharges	11
Churn	0

When comparing to the rows in dataset i.e 7043 rows, 11 missing values is negligible, we can handle these missing values by deleting the rows of the missing values as shown in figure 8.

Fig 8:

Deleting the missing values

```
#dropping rows of missing values
ds1 = ds[ds.TotalCharges.notnull()]
ds1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7032 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7032 non-null   object 
 1   gender            7032 non-null   object 
 2   SeniorCitizen     7032 non-null   int64  
 3   Partner           7032 non-null   object 
 4   Dependents        7032 non-null   object 
 5   tenure            7032 non-null   int64  
 6   PhoneService      7032 non-null   object 
 7   MultipleLines     7032 non-null   object 
 8   InternetService   7032 non-null   object 
 9   OnlineSecurity    7032 non-null   object 
 10  OnlineBackup      7032 non-null   object 
 11  DeviceProtection  7032 non-null   object 
 12  TechSupport       7032 non-null   object 
 13  StreamingTV       7032 non-null   object 
 14  StreamingMovies   7032 non-null   object 
 15  Contract          7032 non-null   object 
 16  PaperlessBilling  7032 non-null   object 
 17  PaymentMethod     7032 non-null   object 
 18  MonthlyCharges    7032 non-null   float64
 19  TotalCharges      7032 non-null   float64
 20  Churn             7032 non-null   object 
```

After dropping the missing values our dataset contains 20 columns and 7032 rows.

Identifying the unique values

Fig 9:

Shows the no.of unique values

```
customerID : 7032
gender : 2
SeniorCitizen : 2
Partner : 2
Dependents : 2
tenure : 72
PhoneService : 2
MultipleLines : 3
InternetService : 3
OnlineSecurity : 3
OnlineBackup : 3
DeviceProtection : 3
TechSupport : 3
StreamingTV : 3
StreamingMovies : 3
Contract : 3
PaperlessBilling : 2
PaymentMethod : 4
MonthlyCharges : 1584
TotalCharges : 6530
Churn : 2
```

```
customerID : ['7590-VHVEG' '5575-GNVDE' '3668-QPYBK' ... '4801-JZAZL' '8361-LTMKD'
 '3186-AJIEK']
gender : ['Female' 'Male']
SeniorCitizen : [0 1]
Partner : ['Yes' 'No']
Dependents : ['No' 'Yes']
tenure : [ 1 34  2 45  8 22 10 28 62 13 16 58 49 25 69 52 71 21 12 30 47 72 17 27
  5 46 11 70 63 43 15 60 18 66  9  3 31 50 64 56  7 42 35 48 29 65 38 68
 32 55 37 36 41  6  4 33 67 23 57 61 14 20 53 40 59 24 44 19 54 51 26 39]
PhoneService : ['No' 'Yes']
MultipleLines : ['No phone service' 'No' 'Yes']
InternetService : ['DSL' 'Fiber optic' 'No']
OnlineSecurity : ['No' 'Yes' 'No internet service']
OnlineBackup : ['Yes' 'No' 'No internet service']
DeviceProtection : ['No' 'Yes' 'No internet service']
TechSupport : ['No' 'Yes' 'No internet service']
StreamingTV : ['No' 'Yes' 'No internet service']
StreamingMovies : ['No' 'Yes' 'No internet service']
Contract : ['Month-to-month' 'One year' 'Two year']
PaperlessBilling : ['Yes' 'No']
PaymentMethod : ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
MonthlyCharges : [29.85 56.95 53.85 ... 63.1  44.2  78.7 ]
TotalCharges : [  29.85 1889.5   108.15 ...  346.45  306.6  6844.5 ]
Churn : ['No' 'Yes']
```

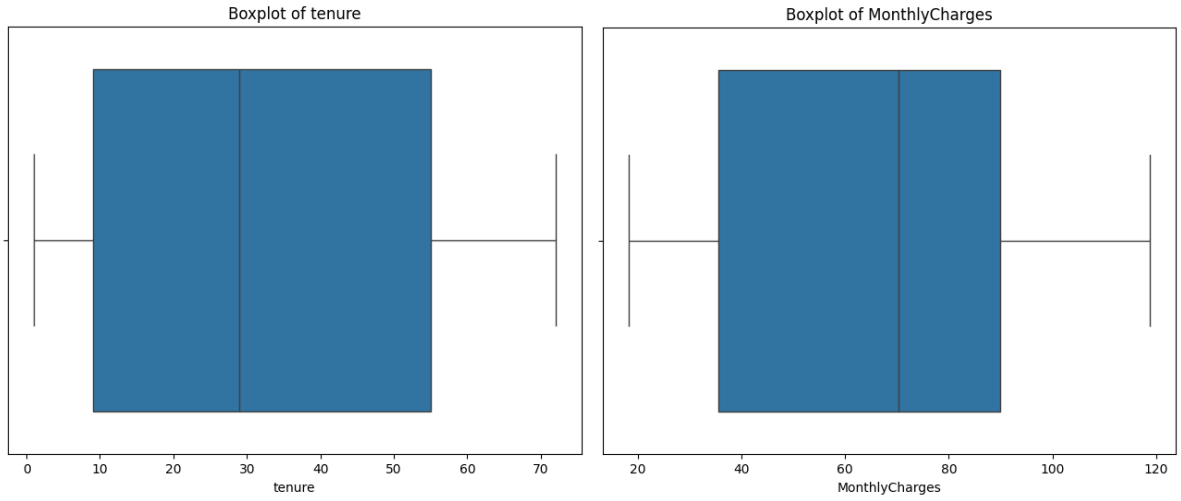
Most features in the dataset contains 2-3 unique value, there are four columns that contains multiple unique value they are Customer ID, tenure, Monthly charges and Total charges.

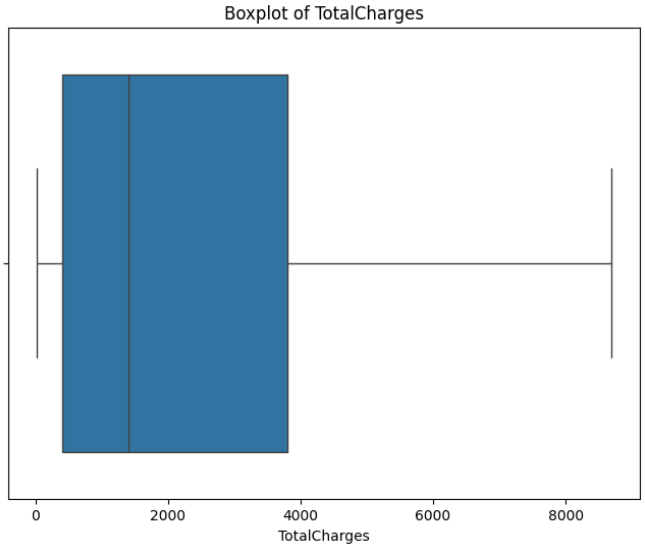
Outliers

Let us explore the dataset and identify the outliers in the dataset.

Fig 10:

Identifying the outliers





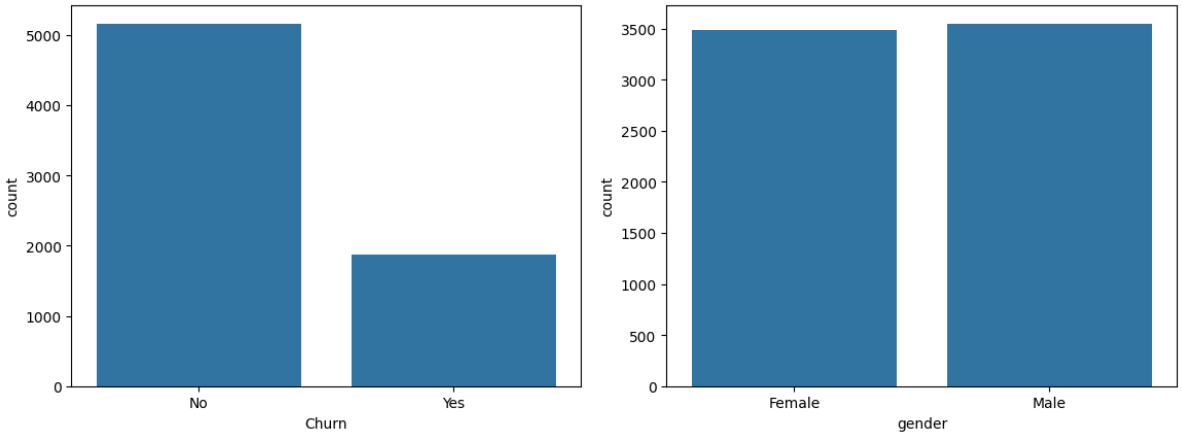
From the above plot we observe that the dataset doesn't contains with any outliers.

Understanding the distribution

Let us plot some graph and identify the distribution and understand the insights form those.

Fig 11:

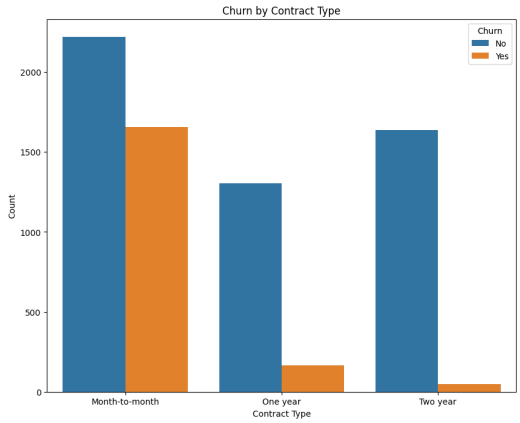
Plotting the distribution of columns



Churn is our target feature, looking at the graph we understand the dataset contains more data related to people who do not churn, this suggest that it is an imbalanced data set. Gender column gives that almost dataset has equal amount of male and female.

Fig 12:

Plotting the graphs with respect to our target feature



- Senior citizens exhibit a greater rate of churn relative to non-senior persons. Senior citizens exhibit a higher tendency for turnover.
- People who don't have a partner ("No") are more likely to leave than people who do have a partner ("Yes").
- People who don't have dependents are more likely to leave than people who do.
- There isn't a strong link between having phone service and churn
- Customers with more than one line are a little more likely to leave than customers who don't have phone service or don't have more than one line.
- People who have fibre optic service are much more likely to leave than people who have DSL or no internet service.
- Customers who do not have online security, online backup, device protection, tech support are more likely to leave.

Summary of the Most Important Factors That Cause Churn:

- Churn is strongly linked to short tenure (newer users).
- People are more likely to leave a company if they have month-to-month contracts or accept electronic checks as payment.
- Higher monthly and total charges make customers to churn.

- People who don't have online security, tech help, or other extra services are more likely to leave.
- Fibre optic users are more likely to churn than DSL users.

Data Processing

After doing all the data exploration and understand the distribution of data and handling the data quality issues, the next step is to convert the data into numerical dataset for the model building and analysis. Converting the non-numerical columns to numerical we follow the below steps

Fig 13:

Converting gender values to 1 and 0.

```
ds1['gender'].replace({'Female':1,'Male':0},inplace=True)
```

Fig 14:

Replacing No internet services &Phone services with "No" and Yes & NO values to 1 & 0

```
ds1.replace('No internet service','No',inplace=True)
ds1.replace('No phone service','No',inplace=True)
```

```
ds1.replace({'Yes':1,'No':0},inplace=True)
```

One hot encoding:

Categorical columns such as Internet service, contract and Payment Method can be converted to numerical columns by one hot encoding as shown in the fig 15.

Fig 15:

One hot encoding

```
ds2 = pd.get_dummies(data= ds1, columns=['InternetService', 'Contract', 'PaymentMethod'])
```

Feature Engineering

After the data exploration, cleaning and processing, the next step is to select, derive or delete features. From the dataset we observe that “customer id” has all the rows unique values and it does not relate to the target variable, so we can drop that feature.

Fig 16:

Checking and dropping customer id

```
#finding duplicates
ds1['customerID'].duplicated().sum()
```

```
0
```

```
#dropping the column
ds1.drop('customerID', axis=1, inplace=True)
```

Fig 17:

Features after Dropping customer id

#	Column	Non-Null Count	Dtype
0	gender	7032 non-null	int64
1	SeniorCitizen	7032 non-null	int64
2	Partner	7032 non-null	int64
3	Dependents	7032 non-null	int64
4	tenure	7032 non-null	int64
5	PhoneService	7032 non-null	int64
6	MultipleLines	7032 non-null	int64
7	OnlineSecurity	7032 non-null	int64
8	OnlineBackup	7032 non-null	int64
9	DeviceProtection	7032 non-null	int64
10	TechSupport	7032 non-null	int64
11	StreamingTV	7032 non-null	int64
12	StreamingMovies	7032 non-null	int64
13	PaperlessBilling	7032 non-null	int64
14	MonthlyCharges	7032 non-null	float64
15	TotalCharges	7032 non-null	float64
16	Churn	7032 non-null	int64
17	InternetService_0	7032 non-null	bool
18	InternetService_DSL	7032 non-null	bool
19	InternetService_Fiber optic	7032 non-null	bool
20	Contract_Month-to-month	7032 non-null	bool
21	Contract_One year	7032 non-null	bool
22	Contract_Two year	7032 non-null	bool
23	PaymentMethod_Bank transfer (automatic)	7032 non-null	bool
24	PaymentMethod_Credit card (automatic)	7032 non-null	bool
25	PaymentMethod_Electronic check	7032 non-null	bool
26	PaymentMethod_Mailed check	7032 non-null	bool

After the data processing and cleaning, our dataset ended with 26 columns and 7032 rows with no null values and categorical values.

Derived Features:

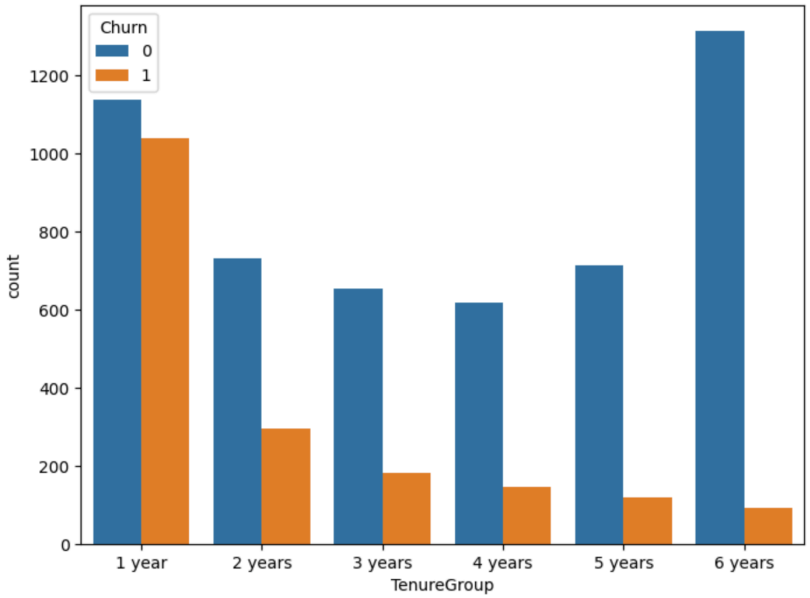
1. Tenure Groups: Customers often leave more often in the first few months of their relationship with a business. Grouping people by how long they've been with the company helps you figure out which groups are more likely to leave.

Fig 18:

Tenure groups

```
#1. Create tenure groups
ds2['TenureGroup'] = pd.cut(ds1['tenure'], bins=[0, 12, 24, 36, 48, 60, 72],
                            labels=['1 year', '2 years', '3 years', '4 years', '5 years', '6 years'])
```

```
#plotting the distribution
plt.figure(figsize=(8, 6))
sns.countplot(data=ds2, x="TenureGroup", hue= 'Churn')
plt.show()
```



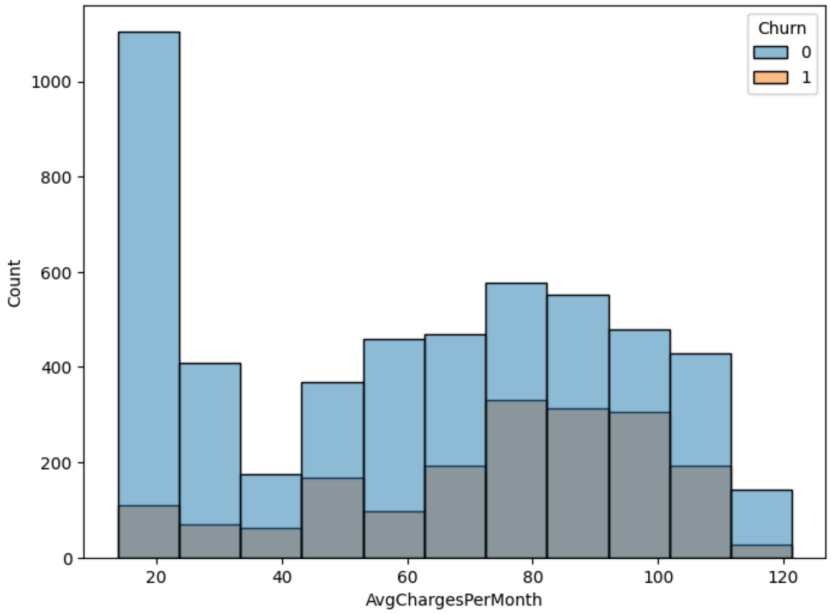
2. Average monthly charges: People whose average monthly charges are less than $40 are much less likely to leave. People whose average monthly charges are between $60 and $80 are more likely to leave. As charges go up, the churn rate goes up even more, which suggests that high charges may make people churn.

   Fig 19:

   Average monthly charges

```
# 2. AvgChargesPerMonth
ds2['AvgChargesPerMonth'] = ds2['TotalCharges'] / ds2['tenure']

#plot the distribution of this
plt.figure(figsize=(8, 6))
sns.histplot(data=ds2, x="AvgChargesPerMonth", hue="Churn", binwidth=10)
plt.show()
```



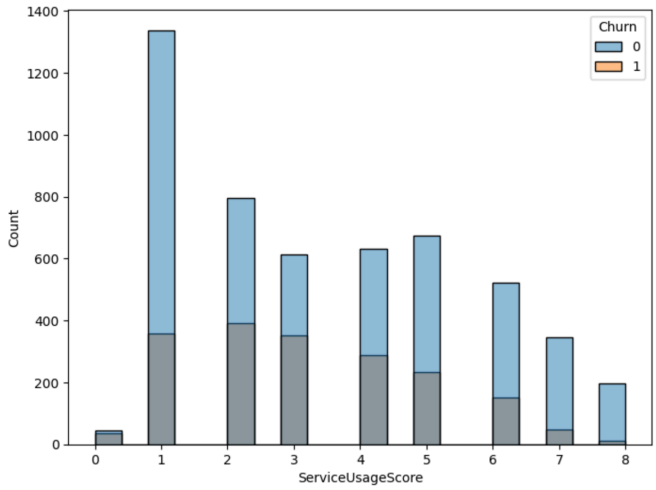
3. Service Usage score: Customers with a low score tend to be more inclined to leave, possibly due to their limited use of services and a weaker connection to the company. On the other hand, customers who have higher Service Usage Scores appear to be more engaged and show a lower likelihood of leaving, as indicated in the low churn rates within the higher score categories.

Fig 20:

Service Usage score

```
# 3. Service usage score
service_columns = ['PhoneService', 'MultipleLines', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies']
ds2['ServiceUsageScore'] = ds2[service_columns].sum(axis=1)

#plotting the distribution
plt.figure(figsize=(8, 6))
sns.histplot(data=ds2, x="ServiceUsageScore", hue= 'Churn')
plt.show()
```



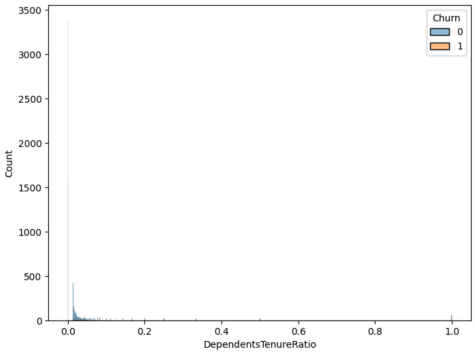
4. Dependents Tenure Ratio: The Dependents Tenure Ratio shows how long a customer has been a customer and their dependents. New customers with dependents (high ratio) are more likely to leave, while long-term customers with dependents (low ratio) are more likely to stay. This feature can help with methods for keeping customers, especially for new customers with dependents who need more interaction to keep them from leaving.

Fig 21:

Dependents Tenure Ratio

```
# 4. Dependents to tenure ratio
ds2['DependentsTenureRatio'] = ds2['Dependents'] / ds2['tenure']

#plotting the distribution
plt.figure(figsize=(8, 6))
sns.histplot(data=ds2, x="DependentsTenureRatio", hue= 'Churn')
plt.show()
```



Scaling the features

To scale the features in the dataset we use minmax scaler, because most of the values in the dataset are 0 and 1, hence minmax scaler is the best option.

Fig 22:

MinMax Scaler

```
#scaling the columns
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
ds2[['tenure','MonthlyCharges','TotalCharges','AvgChargesPerMonth','ServiceUsageScore']] = scaler.fit_transform(ds2[['tenure','MonthlyCharges','TotalCharges','AvgCh
```

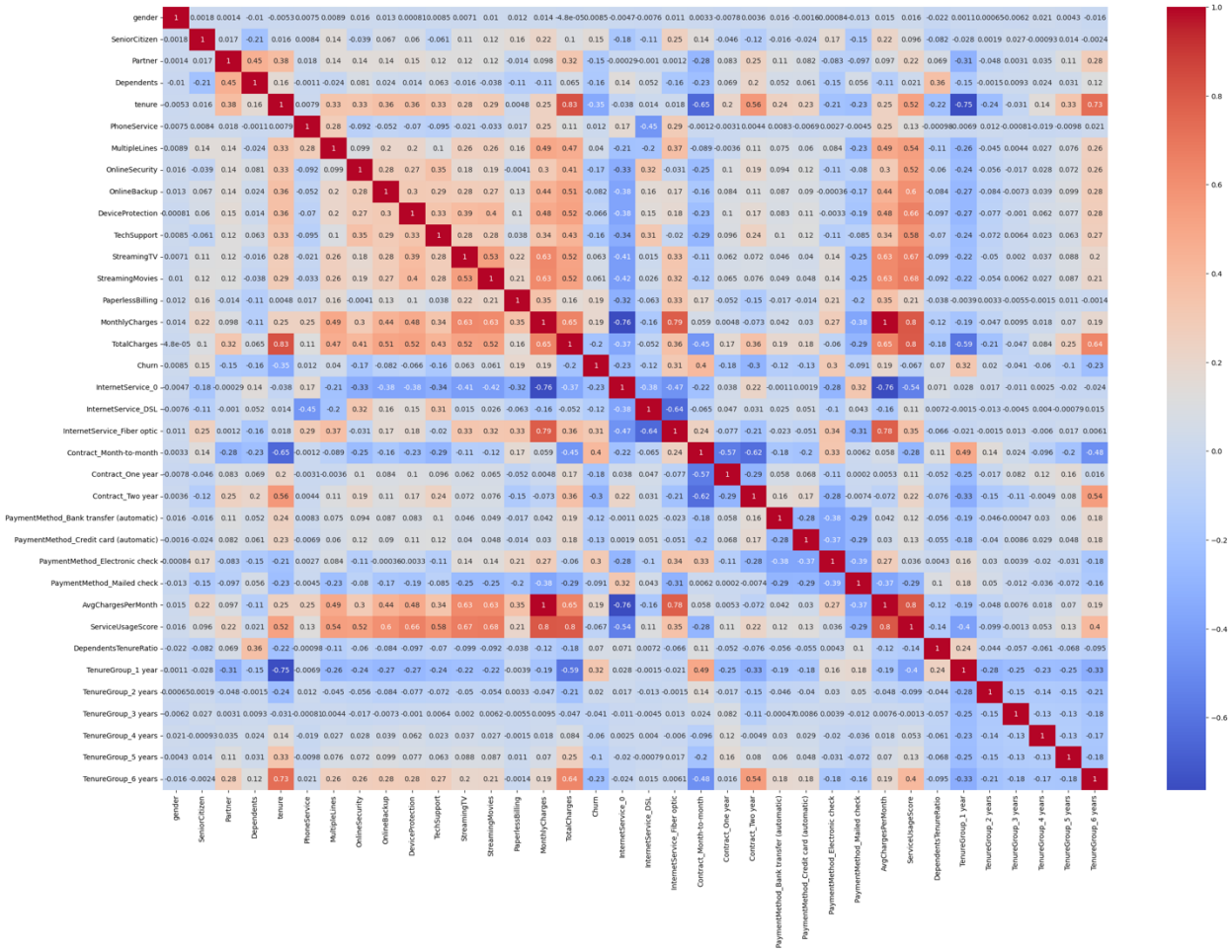
Feature Selection

To select the features that needed for the prediction, we use the following methods.

Correlation:

Fig 23:

Correlation matrix of features



This correlation matrix helps us to understand the features relation with target variable and with another variable. Notable in our dataset “gender” columns has very low correlation with “churn” and Average charges per month and monthly charges are closely related, hence we drop “gender” and “Average charges per month”

Lasso method

Fig 24

Features selection by Lasso method

```
# feature selection by Lasso method

from sklearn.linear_model import Lasso
from sklearn.feature_selection import SelectFromModel

# Create a Lasso model with a suitable alpha value
lasso = Lasso(alpha=0.01)

# Fit the Lasso model to your data
lasso.fit(X, y)

# Use SelectFromModel to identify features with non-zero coefficients
model = SelectFromModel(lasso, prefit=True)
X_new = model.transform(X)

# Get the selected feature names
selected_features = X.columns[model.get_support()]

print("Selected Features:")
print(selected_features)
```

```
Selected Features:
Index(['tenure', 'OnlineSecurity', 'TechSupport', 'PaperlessBilling',
       'InternetService_0', 'InternetService_Fiber optic',
       'Contract_Month-to-month', 'PaymentMethod_Electronic check',
       'TenureGroup_1 year'],
      dtype='object')
```

By Lasso method it selects only 9 features and it drops all other features which are correlated with target variable, we tried using different alpha values but this method couldn't give the best results.

Final ABT

After completion of Data exploration and feature engineering, fig 24 shows our final ABT with 34 features.

Fig 24:

Final ABT

```
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   SeniorCitizen                           7032 non-null   int64
 1   Partner                                 7032 non-null   int64
 2   Dependents                              7032 non-null   int64
 3   tenure                                  7032 non-null   float64
 4   PhoneService                            7032 non-null   int64
 5   MultipleLines                           7032 non-null   int64
 6   OnlineSecurity                          7032 non-null   int64
 7   OnlineBackup                            7032 non-null   int64
 8   DeviceProtection                        7032 non-null   int64
 9   TechSupport                             7032 non-null   int64
 10  StreamingTV                             7032 non-null   int64
 11  StreamingMovies                         7032 non-null   int64
 12  PaperlessBilling                        7032 non-null   int64
 13  MonthlyCharges                          7032 non-null   float64
 14  TotalCharges                            7032 non-null   float64
 15  Churn                                   7032 non-null   int64
 16  InternetService_0                       7032 non-null   bool
 17  InternetService_DSL                     7032 non-null   bool
 18  InternetService_Fiber optic             7032 non-null   bool
 19  Contract_Month-to-month                 7032 non-null   bool
 20  Contract_One year                       7032 non-null   bool
 21  Contract_Two year                       7032 non-null   bool
 22  PaymentMethod_Bank transfer (automatic) 7032 non-null   bool
 23  PaymentMethod_Credit card (automatic)   7032 non-null   bool
 24  PaymentMethod_Electronic check          7032 non-null   bool
 25  PaymentMethod_Mailed check              7032 non-null   bool
 26  ServiceUsageScore                       7032 non-null   float64
 27  DependentsTenureRatio                   7032 non-null   float64
 28  TenureGroup_1 year                      7032 non-null   bool
 29  TenureGroup_2 years                     7032 non-null   bool
 30  TenureGroup_3 years                     7032 non-null   bool
 31  TenureGroup_4 years                     7032 non-null   bool
 32  TenureGroup_5 years                     7032 non-null   bool
 33  TenureGroup_6 years                     7032 non-null   bool
```

Data Splitting

After the completing of Final ABT, the target feature is assigned to “y” and the independent variables are assigned to “X” as showing in the fig 25.

Fig 25:

Assigning the features to X and y

```
#splitting the dataset
X = ds2.drop('Churn', axis=1)
y = ds2['Churn']
```

After assigning the features to X and y, we split the features into training and testing with the help of “train_test_split” from sklearn.model_selection with train and test size of 80 and 20.

Fig 26:

Splitting the data set into train and test

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

Model Building

After the completion of splitting the data set, we train the model with two models

Decision Tree

We train our model using Decision Tree, import, fitting the model and evaluating the model as shown in the fig 27.

Fig 27:

Decision Tree model building, testing and evaluating

```
model = DecisionTreeClassifier(random_state=42)
```

```
model.fit(X_train, y_train)
```

```
DecisionTreeClassifier(random_state=42)
```

```
# Making predictions on the test set
y_pred = model.predict(X_test)
```

```
# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

```
Accuracy: 0.7313432835820896
Precision: 0.4896373056994819
Recall: 0.5108108108108108
F1 Score: 0.5
```

After evaluating the Decision Tree Model, we got the accuracy 73.134, precision 0.48, Recall 0.51 and F1 Score 0.5, indicates that the model is moderately effective although has difficulty in accurately identifying churners. A precision of 0.48 signifies that the model correctly predicts customer churn only 48% of the time. A recall of 0.51 indicates that it identifies 51% of the actual churners. An F1 score of 0.5 indicates an average equilibrium

between precision and recall, signifying that the model requires improvement to more effectively forecast churn while reducing false positives and false negatives. It is not better than the baseline accuracy as shown in the fig 28.

Fig 28

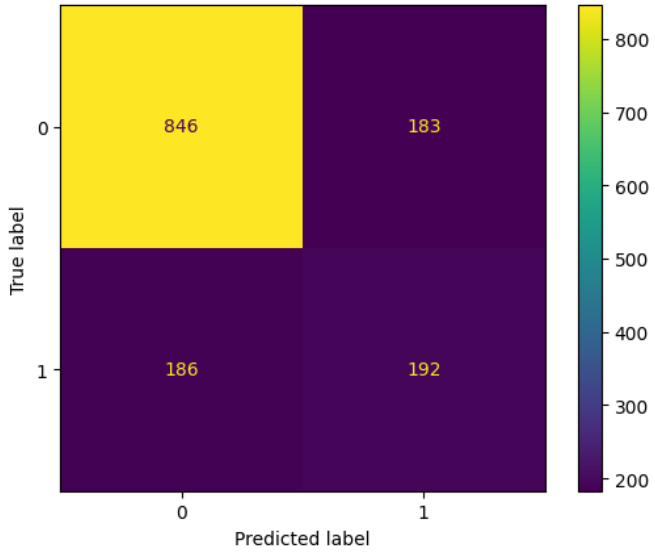
Baseline accuracy

```
## Checking our baseline accuracy
y.value_counts(normalize=True)
```

Churn	proportion
0	0.734215
1	0.265785

Fig 29

Confusion matrix



Top left (846): These are the customers who were correctly predicted as staying (no churn).

Top-right (183): These are customers who were incorrectly predicted as staying, but they actually churned.

Bottom-left (186): These are customers who were incorrectly predicted as churning, but they actually stayed.

Bottom-right (192): These are customers who were correctly predicted as churning.

KNN Model

We train our model using KNN import, fitting the model and evaluating the model as shown in the fig 30.

Fig 30

KNN model building, testing and evaluating

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=8, p=2, metric='minkowski', weights='distance')
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)

def quality_metrics(y_test, y_pred):
  accuracy = accuracy_score(y_test, y_pred)
  precision = precision_score(y_test, y_pred)
  recall = recall_score(y_test, y_pred)
  f1 = f1_score(y_test, y_pred)
  print(f"Accuracy: {accuracy}")
  print(f"Precision: {precision}")
  print(f"Recall: {recall}")
  print(f"F1 Score: {f1}")

quality_metrics(y_test,y_pred_knn)
```

```
Accuracy: 0.7590618336886994
Precision: 0.5439093484419264
Recall: 0.518918918918919
F1 Score: 0.5311203319502075
```

This model has initially worked a bit better than the baseline accuracy, an accuracy of 0.759 indicates that it accurately predicts customer churn or retention around 76% of the time. The precision of 0.54 and recall of 0.51 indicate that the model is not particularly effective at identifying consumers who churn (precision) and is failing to detect almost half of the real churners (recall). The F1 score of 0.53, which equilibrates precision and recall, indicates low performance, implying that the model requires improvements in properly identifying churners.

Hyperparameter Tuning

a. Decision Tree: Adjusting different settings for the model to get the best outcome.

Depth = 2	Accuracy: 0.7540867093105899 Precision: 0.5508274231678487 Recall: 0.5989717223650386 F1 Score: 0.5738916256157636
Depth = 3	Accuracy: 0.7803837953091685 Precision: 0.7105263157894737 Recall: 0.34704370179948585 F1 Score: 0.46632124352331605
Depth = 4	Accuracy: 0.7825159914712153 Precision: 0.6416382252559727 Recall: 0.4832904884318766 F1 Score: 0.5513196480938416
Depth = 5	Accuracy: 0.7818052594171997 Precision: 0.6990291262135923 Recall: 0.37017994858611825 F1 Score: 0.48403361344537815
Depth = 6	Accuracy: 0.7924662402274343 Precision: 0.6381766381766382 Recall: 0.5758354755784062 F1 Score: 0.6054054054054054
Depth = 7	Accuracy: 0.7796730632551528 Precision: 0.5852272727272727 Recall: 0.5567567567567567 F1 Score: 0.5706371191135735
Depth = 8	Accuracy: 0.7704335465529495 Precision: 0.5654596100278552 Recall: 0.5486486486486486 F1 Score: 0.5569272976680385
Depth = 9	Accuracy: 0.7739872068230277 Precision: 0.5751445086705202

	Recall: 0.5378378378378 F1 Score: 0.5558659217877094
Depth = 10	Accuracy: 0.7683013503909026 Precision: 0.5604395604395604 Recall: 0.5513513513513514 F1 Score: 0.5558583106267031
Depth = 11	Accuracy: 0.7611940298507462 Precision: 0.5425 Recall: 0.5864864864864865 F1 Score: 0.5636363636363636
Depth = 12	Accuracy: 0.744136460554371 Precision: 0.5126262626262627 Recall: 0.5486486486486486 F1 Score: 0.5300261096605745
Depth = 13	Accuracy: 0.7526652452025586 Precision: 0.5282051282051282 Recall: 0.5567567567567567 F1 Score: 0.5421052631578946
Depth = 14	Accuracy: 0.7412935323383084 Precision: 0.5074257425742574 Recall: 0.5540540540540541 F1 Score: 0.5297157622739017

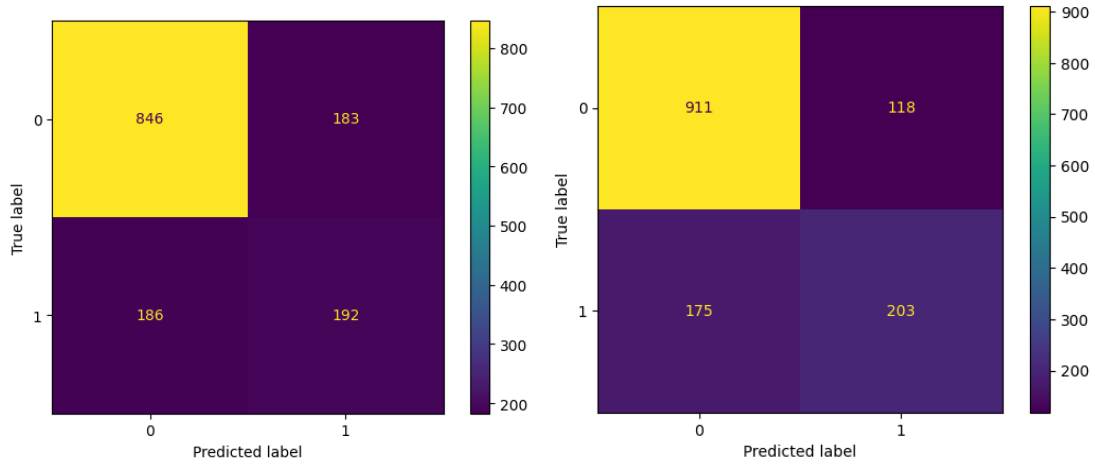
Considering the overall output from the various settings, the best configuration for the Decision tree model: Depth = 6, criterion = ‘entropy’

Confusion Matrix:

Fig 31:

Confusion matrix before and after tuning

Before After



The comparison of the confusion matrices before and after tuning indicates an improvement in the performance of the model. After adjusting the model, it became better at predicting both customers who won't churn and those who will. It correctly identified more non-churners and churners, reduced the number of false alarms (customers mistakenly predicted to churn), and slightly improved in catching actual churners that were missed before. Overall, the model's performance improved across the board.

b. KNN:

Setting 1:

Metric: Minkowski, Weights: distance

n_neighbors	Accuracy
1	0.7327647476901208
2	0.7341862117981521
3	0.7484008528784648
4	0.7469793887704336
5	0.7654584221748401
6	0.7640369580668088
7	0.7654584221748401
8	0.7590618336886994
9	0.7668798862828714
10	0.7661691542288557
11	0.7619047619047619

12	0.7647476901208244
13	0.7675906183368870
14	0.7619047619047619

Setting 2:

Metric: Manhattan, Weights: distance

n_neighbors	Accuracy
1	0.7299218194740583
2	0.7313432835820896
3	0.7533759772565742
4	0.7498223169864960
5	0.7654584221748401
6	0.7675906183368870
7	0.7611940298507462
8	0.7611940298507462
9	0.7626154939587776
10	0.7654584221748401
11	0.7619047619047619
12	0.7604832977967306
13	0.7640369580668088
14	0.7633262260127932

Setting 3:

Metric: Minkowski, Weights: uniform

n_neighbors	Accuracy
1	0.7327647476901208
2	0.7675906183368870
3	0.7547974413646056
4	0.7675906183368870
5	0.7690120824449183
6	0.7704335465529495
7	0.7711442786069652
8	0.7789623312011372
9	0.7782515991471215
10	0.7825159914712153
11	0.7725657427149965
12	0.7775408670931059
13	0.7754086709310590

14	0.7789623312011372

Setting 4:

Varying p in minkowski distance

p (minkowski power)	Accuracy
1	0.7654584221748401
2	0.7661691542288557
3	0.7668798862828714
4	0.7661691542288557
5	0.7668798862828714
6	0.7675906183368870
7	0.7675906183368870
8	0.7683013503909026
9	0.7675906183368870
10	0.7683013503909026
11	0.7690120824449183
12	0.7683013503909026
13	0.7697228144989339
14	0.7683013503909026

The best accuracy is found to be 77.6% at 10 neighbours with p=2, metric= minkowski and weights are uniform.

Best Model Settings

a. Decision Tree: The best model setting for decision tress is as at depth 6 and criterion is entropy, and the evaluation is shown in the fig 32.

Fig 32:

Best settings for Decision Tree

```
#final model
model_fin = DecisionTreeClassifier(max_depth=6, criterion='entropy')
model_fin.fit(X_train, y_train)
y_pred_fin = model_fin.predict(X_test)

# Evaluating the model
accuracy_fin = accuracy_score(y_test, y_pred_fin)
precision_fin = precision_score(y_test, y_pred_fin)
recall_fin = recall_score(y_test, y_pred_fin)
f1_fin = f1_score(y_test, y_pred_fin)

print(f"Accuracy: {accuracy_fin}")
print(f"Precision: {precision_fin}")
print(f"Recall: {recall_fin}")
print(f"F1 Score: {f1_fin}")
```

```
Accuracy: 0.7917555081734187
Precision: 0.632398753894081
Recall: 0.5370370370370371
F1 Score: 0.5808297567954221
```

b. KNN: The best accuracy is found to be 77.6% at 10 neighbours with p=2, metric= minkowski and weights are uniform. the fig 33.

Fig 22:

Best settings for KNN

```
#Final Model
knn_fin = KNeighborsClassifier(n_neighbors=10, p=2, metric='minkowski', weights='uniform')
knn_fin.fit(X_train, y_train)
y_pred_knn_fin = knn_fin.predict(X_test)

quality_metrics(y_test,y_pred_knn_fin)
```

```
Accuracy: 0.7761194029850746
Precision: 0.6265060240963856
Recall: 0.4126984126984127
F1 Score: 0.4976076555023922
```

Model Comparison

Decision Tree	**KNN**
Accuracy: 0.7924662402274343	Accuracy: 0.7761194029850746

Precision: 0.6381766381766382 Recall: 0.5758354755784062 F1 Score: 0.6054054054054054	Precision: 0.6265060240963856 Recall: 0.4126984126984127 F1 Score: 0.4976076555023922

The Decision Tree model is superior because to its greater accuracy (about 79.2%) in comparison to the KNN model (roughly 77.6%). Furthermore, the Decision Tree exceeds KNN in precision (0.638 compared to 0.626), recall (0.576 compared to 0.413), and F1 Score (0.605 compared to 0.498). This indicates that the Decision Tree not only detects a greater percentage of true positive cases but also sustains an improved balance between precision and recall, making it a more dependable model for this business problem

Dataset Link

This is the dataset that I've used for the assignment.

https://www.kaggle.com/datasets/blastchar/telco-customer-churn/data

Colab link

This is the Colab link to find the code

https://colab.research.google.com/drive/1pDARUXZXsfe9-TLGirZHK4uIfohs0M9n?usp=sharing