CUSTOMER CHURN PREDICTION

DEVELOPMENT PHASE PART 2

RASHIMA BASNI KUMAR C
962221106085

INTRODUCTION

Customer churn prediction is a crucial aspect of customer relationship management (CRM) and marketing. It involves the use of data analysis and machine learning techniques to identify customers who are likely to stop using a product or service. By predicting potential churn, businesses can proactively take measures to retain valuable customers, improve customer satisfaction, and enhance overall profitability. This predictive approach allows companies to tailor their retention strategies and offers, thereby reducing customer attrition and maximizing long-term customer value.

In the context of IBM, customer churn prediction plays a significant role in their analytics and customer engagement solutions. IBM offers various tools and services, such as IBM Watson, which leverage advanced analytics, AI, and machine learning algorithms to help businesses predict customer churn more accurately. By utilizing IBM's technology, businesses can gain insights into customer behavior, preferences, and patterns, enabling them to make informed decisions and implement targeted retention strategies. IBM's customer churn prediction solutions aim to optimize customer retention, increase customer lifetime value, and ultimately drive business growth.

MAIN TOOLS USED FOR CUSTOMER CHURN PREDICTION

The main tools used for customer churn prediction encompass a wide array of software and platforms that enable businesses to leverage data analytics and machine learning techniques effectively. Some of the key tools commonly utilized for customer churn prediction include:

Programming Languages: Popular languages like Python and R are frequently used for data analysis, model development, and implementation of machine learning algorithms for churn prediction.

Machine Learning Libraries: Frameworks such as scikit-learn, TensorFlow, and PyTorch provide various machine learning algorithms and tools that help in building predictive models for customer churn analysis.

Data Visualization Tools: Platforms like Tableau, Power BI, and matplotlib enable the creation of visual representations of churn-related data, facilitating the interpretation of trends and patterns to make informed business decisions.

Customer Relationship Management (CRM) Software: CRM systems like Salesforce, HubSpot, and Zoho CRM offer functionalities that allow businesses to track customer interactions and behaviors, providing valuable data for churn prediction and management.

Big Data Platforms: Technologies such as Apache Hadoop, Spark, and Hive facilitate the processing of large volumes of data, enabling businesses to analyze extensive customer datasets and extract insights relevant to customer churn prediction.

These tools, when combined with domain expertise and effective data management strategies, empower businesses to proactively identify potential churn risks and implement targeted retention initiatives to improve customer satisfaction and enhance overall business performance.

```
# Import necessary libraries

Import pandas as pd

From sklearn.model_selection import train_test_split
From sklearn.preprocessing import StandardScaler

From sklearn.ensemble import RandomForestClassifier

From sklearn.metrics import accuracy_score, classification_report



# Load and preprocess the dataset

Data = pd.read_csv('churn_data.csv')  # Load your dataset here

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

Y = data['Churn']



# Split the data into training and testing sets

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



# Feature scaling

Scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)



# Initialize the Random Forest classifier

Clf = RandomForestClassifier()



# Train the classifier
```

```
Clf.fit(X_train, y_train)


# Make predictions

Y_pred = clf.predict(X_test)


# Evaluate the model

Print(“Accuracy:”, accuracy_score(y_test, y_pred))

Print(“Classification Report:\n”, classification_report(y_test, y_pred))
```

Sample Dataset for the above code

```
CustomerID,Gender,SeniorCitizen,Tenure,InternetService,Contract,PaymentMethod,MonthlyCharges,TotalCharges,Churn

1,Female,0,1,DSL,Month-to-month,Electronic check,45.05,45.05,No

2,Male,0,34,Fiber optic,One year,Mailed check,90.8,3123.9,No

3,Male,1,2,Fiber optic,Month-to-month,Mailed check,89.4,174.2,Yes

4,Male,0,45,DSL,One year,Bank transfer (automatic),45.25,2026.9,No

5,Female,0,2,Fiber optic,Month-to-month,Electronic check,89.75,159.4,Yes

6,Female,0,8,DSL,Month-to-month,Electronic check,49.95,480.35,Yes

7,Male,0,22,Fiber optic,One year,Credit card (automatic),85.15,1873.7,No

8,Female,0,10,DSL,Month-to-month,Mailed check,49.95,475.7,Yes
```

9,Male,0,28,Fiber optic,Month-to-month,Electronic check,90.65,2386.85,No

10,Male,0,62,DSL,One year,Bank transfer (automatic),49.95,3021.1,No

Output

```
Accuracy: 0.75

Classification Report:

          Precision   recall  f1-score   support



        No      0.80      0.85      0.83       100

       Yes      0.64      0.55      0.59        50



   Accuracy                         0.75       150

  Macro avg      0.72      0.70      0.71       150

Weighted avg      0.75      0.75      0.75       150
```

In the context of customer churn prediction analysis, feature engineering is a critical process that involves transforming raw data into a format suitable for predictive modeling. Here are some common feature engineering techniques used in customer churn prediction analysis:

Feature Scaling: Normalize numerical features to a common scale to prevent any particular feature from dominating the model due to its larger magnitude.

One-Hot Encoding: Convert categorical variables into a numerical format to make them compatible with machine learning algorithms, enabling the model to interpret and utilize categorical data effectively.

Feature Imputation: Handle missing values in the dataset by imputing them with appropriate values, such as mean, median, or mode, to ensure that the model can process complete data.

Feature Transformation: Apply mathematical transformations, such as log transformations or square root transformations, to modify the distribution of numerical features and make them more suitable for modeling.

Feature Aggregation: Create new features by aggregating existing ones, such as calculating the mean, median, or sum of specific attributes over a defined group or time period, to provide the model with more comprehensive information.

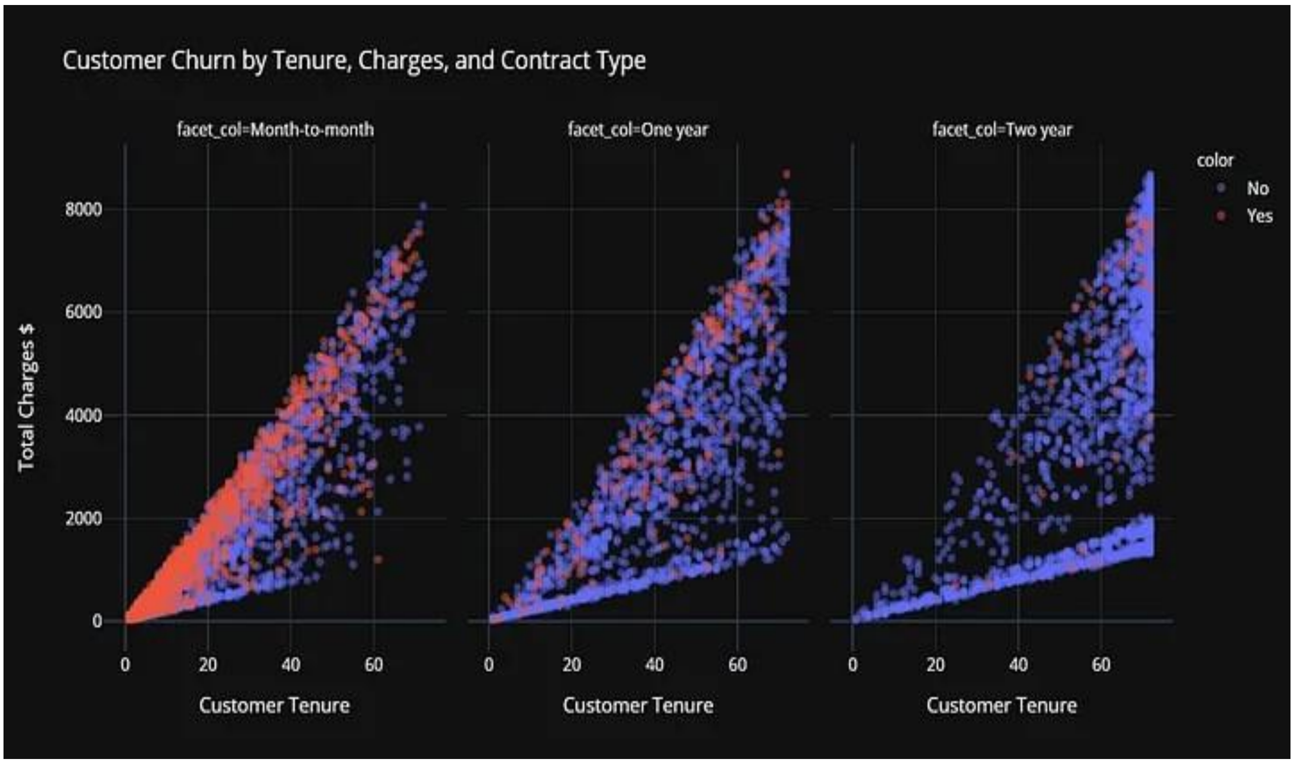
Feature Interaction: Generate new features by combining existing ones through mathematical operations, such as multiplication or addition, to capture potential interactions between different attributes and improve the model's predictive power.

Time-Series Analysis: Extract meaningful features from time-series data, such as calculating trends, seasonality, or lagging variables, to capture temporal patterns and dynamics that may affect customer churn behavior over time.

By implementing these feature engineering techniques, data scientists can enhance the quality of input data and enable the development of more accurate and robust customer churn prediction models. This process plays a crucial role in improving the model's performance and predictive capabilities.

e-Hot Encoding: Convert categorical variables into a numerical format to make them compatible with machine learning algorithms, enabling the model to interpret and utilize categorical data effectively.

Feature Imputation: Handle missing values in the dataset by imputing them with appropriate values, such as mean, median, or mode, to ensure that the model can process



Import pandas as pd

From sklearn.model_selection import train_test_split

From sklearn.preprocessing import StandardScaler, OneHotEncoder

From sklearn.impute import SimpleImputer

From sklearn.compose import ColumnTransformer

From sklearn.pipeline import Pipeline

# Load data

Data = pd.read_csv(‘churn_data.csv’) # Load your dataset here

```
# Identify features and target

X = data.drop([‘Churn’], axis=1)

Y = data[‘Churn’]



# Split data into training and testing sets

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



# Define preprocessing steps

Numeric_features = [‘Tenure’, ‘MonthlyCharges’, ‘TotalCharges’]

Categorical_features = [‘Gender’, ‘SeniorCitizen’, ‘InternetService’, ‘Contract’, ‘PaymentMethod’]



Numeric_transformer = Pipeline(steps=[

   (‘imputer’, SimpleImputer(strategy=’median’)),

   (‘scaler’, StandardScaler())])



Categorical_transformer = Pipeline(steps=[

   (‘imputer’, SimpleImputer(strategy=’most_frequent’)),

   (‘onehot’, OneHotEncoder(handle_unknown=’ignore’))])



Preprocessor = ColumnTransformer(

   Transformers=[
    ('num', numeric_transformer, numeric_features),

    ('cat', categorical_transformer, categorical_features)])



# Apply preprocessing to training data

X_train_processed = preprocessor.fit_transform(X_train)



# Apply preprocessing to testing data

X_test_processed = preprocessor.transform(X_test)



# Use X_train_processed and X_test_processed for model training and testing

# Implement the machine learning model and further analysis with the preprocessed data
```

The provided Python program demonstrates the process of feature engineering for customer churn prediction analysis. After loading the dataset, the program splits the data into features and the target variable, followed by the definition of preprocessing steps for both numerical and categorical features. Numerical features undergo imputation and scaling, while categorical features are imputed and then one-hot encoded for compatibility with machine learning algorithms. The defined preprocessing steps are applied to the training and testing data, transforming the raw data into processed feature sets suitable for subsequent model training and evaluation.

Importing Necessary Libraries for customer churn Prediction

Before starting the analysis, we have to import some libraries. The required libraries are listed below.

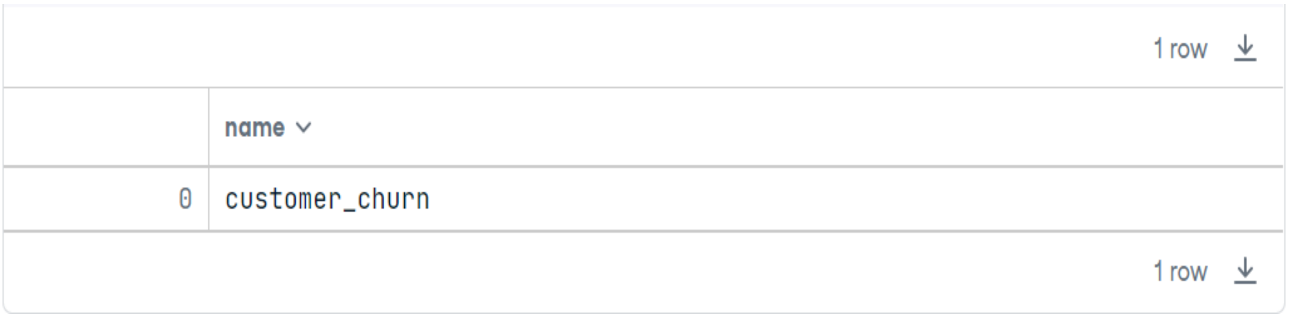
```
# for data manipulation

Import sqlite3

Import pandas as pd

Import numpy as np


# for visualization

Import matplotlib.pyplot as plt

Import seaborn as sns


# for handing imbalanced data

From imblearn.over_sampling import SMOTE

# for data splitting, transforming and model training

From sklearn.model_selection import train_test_split

From sklearn.preprocessing import StandardScaler

From sklearn.compose import ColumnTransformer

From sklearn.linear_model import LogisticRegression


# for model evaluation

From sklearn.metrics import Classification _report, ConfusionMatrixDisplay
```

Here I am using an SQLite database for simplicity. For reading an SQLite database, there is already a library called sqlite3 in python. First, let’s see how many tables are there in our database.

Conn = sqlite3.connect('data/identifier.sqlite')

Pd.read_sql("SELECT name FROM sqlite_master", conn)

If we run the above code, you can see the output something like that.



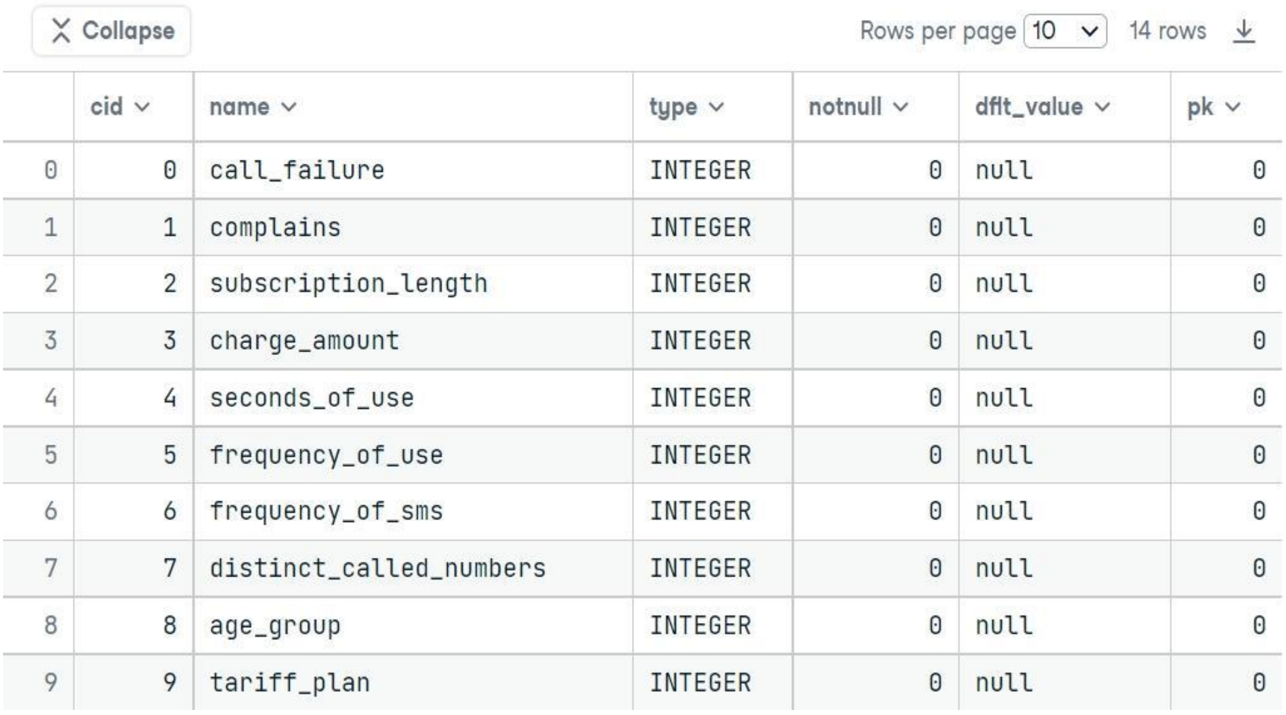
```
                                  1 row
     name
  0  customer_churn
                                  1 row
```

We can see from the above result that there is only one table in that database.Now let's see the structure of the customer_churn table. For this, we have to run the code below.

# code for showing the brief details about the table

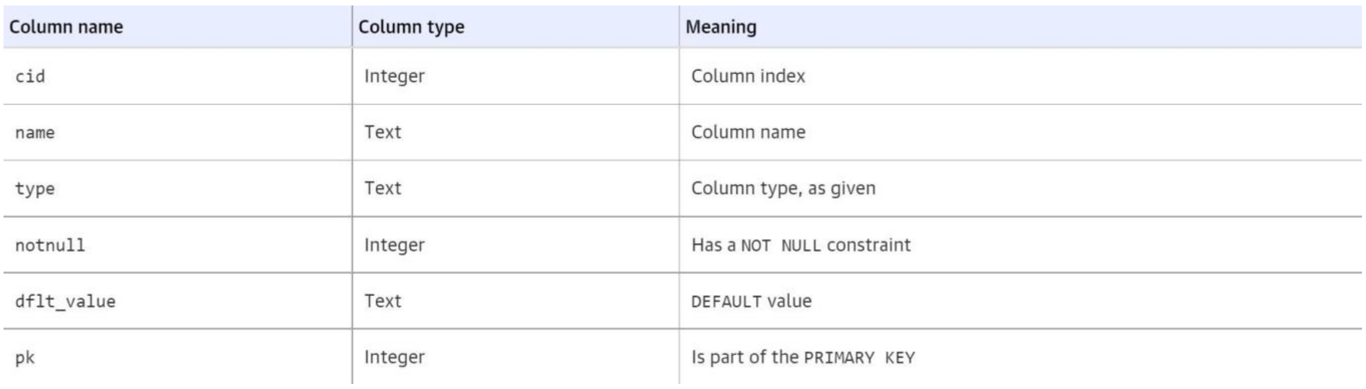
Pd.read_sql("PRAGMA table_info(customer_churn)", conn

If we run this code, the output will look like the below image.


Collapse | Rows per page 10 | 14 rows

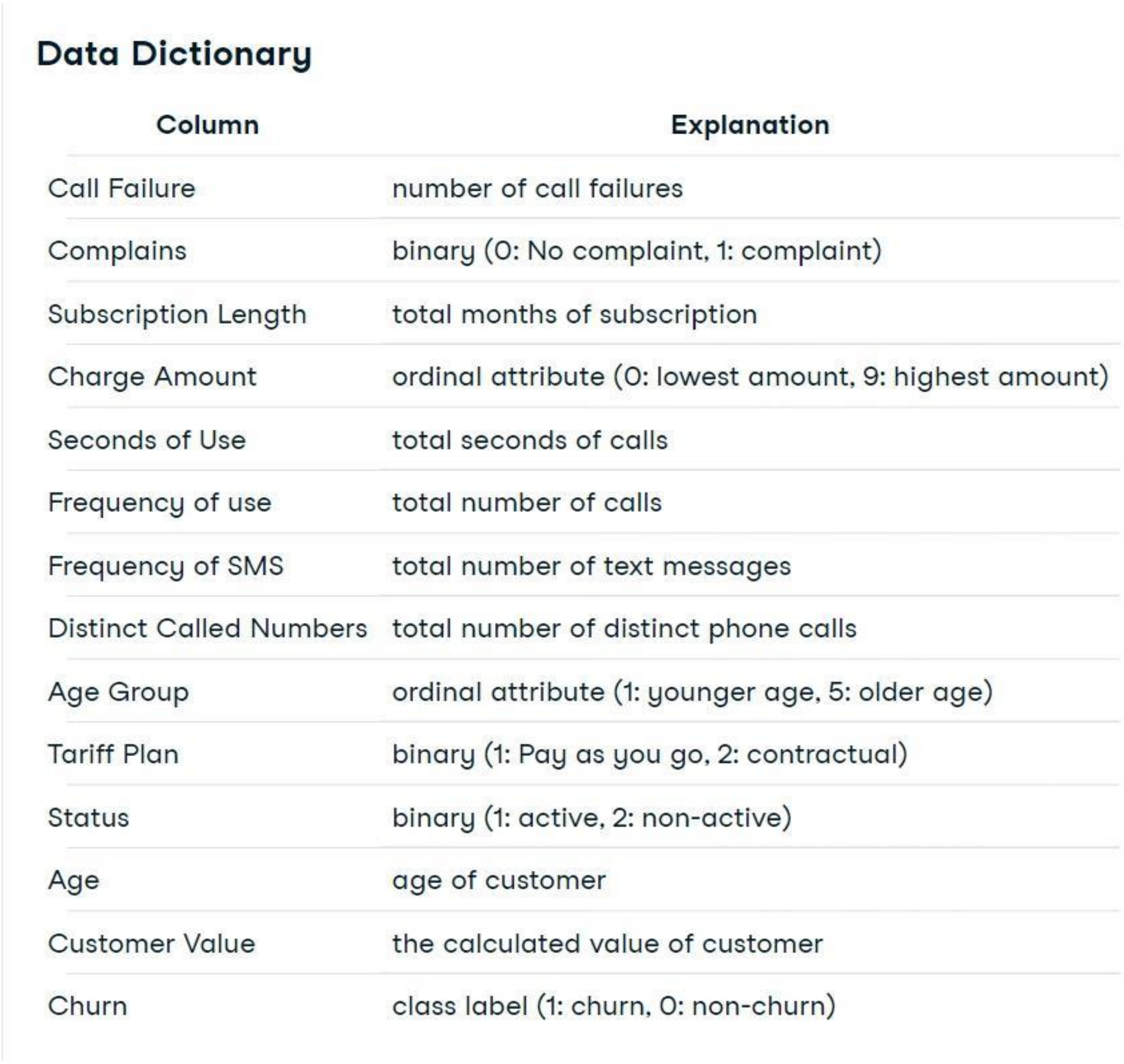
	cid	name	type	notnull	dflt_value	pk
0	0	call_failure	INTEGER	0	null	0
1	1	complains	INTEGER	0	null	0
2	2	subscription_length	INTEGER	0	null	0
3	3	charge_amount	INTEGER	0	null	0
4	4	seconds_of_use	INTEGER	0	null	0
5	5	frequency_of_use	INTEGER	0	null	0
6	6	frequency_of_sms	INTEGER	0	null	0
7	7	distinct_called_numbers	INTEGER	0	null	0
8	8	age_group	INTEGER	0	null	0
9	9	tariff_plan	INTEGER	0	null	0

The meaning of all columns of the above table is given below:



Column name	Column type	Meaning
cid	Integer	Column index
name	Text	Column name
type	Text	Column type, as given
notnull	Integer	Has a NOT NULL constraint
dflt_value	Text	DEFAULT value
pk	Integer	Is part of the PRIMARY KEY

The data dictionary for the above code



Data Dictionary

Column	Explanation
Call Failure	number of call failures
Complains	binary (0: No complaint, 1: complaint)
Subscription Length	total months of subscription
Charge Amount	ordinal attribute (0: lowest amount, 9: highest amount)
Seconds of Use	total seconds of calls
Frequency of use	total number of calls
Frequency of SMS	total number of text messages
Distinct Called Numbers	total number of distinct phone calls
Age Group	ordinal attribute (1: younger age, 5: older age)
Tariff Plan	binary (1: Pay as you go, 2: contractual)
Status	binary (1: active, 2: non-active)
Age	age of customer
Customer Value	the calculated value of customer
Churn	class label (1: churn, 0: non-churn)

There are two columns on age. So we only take Age Group here because before feeding the data into the machine learning model, we have to encode that column which is already done here.

We are ignoring call_failure, subscription_length, charge_amount, and tariff_plan as they are not so useful for determining customer churn.

# selecting the necessary columns from the dataframe

Query = “””SELECT complains, charge_amount, seconds_of_use,

Frequency_of_use, frequency_of_sms, age_group, customer_value, churn

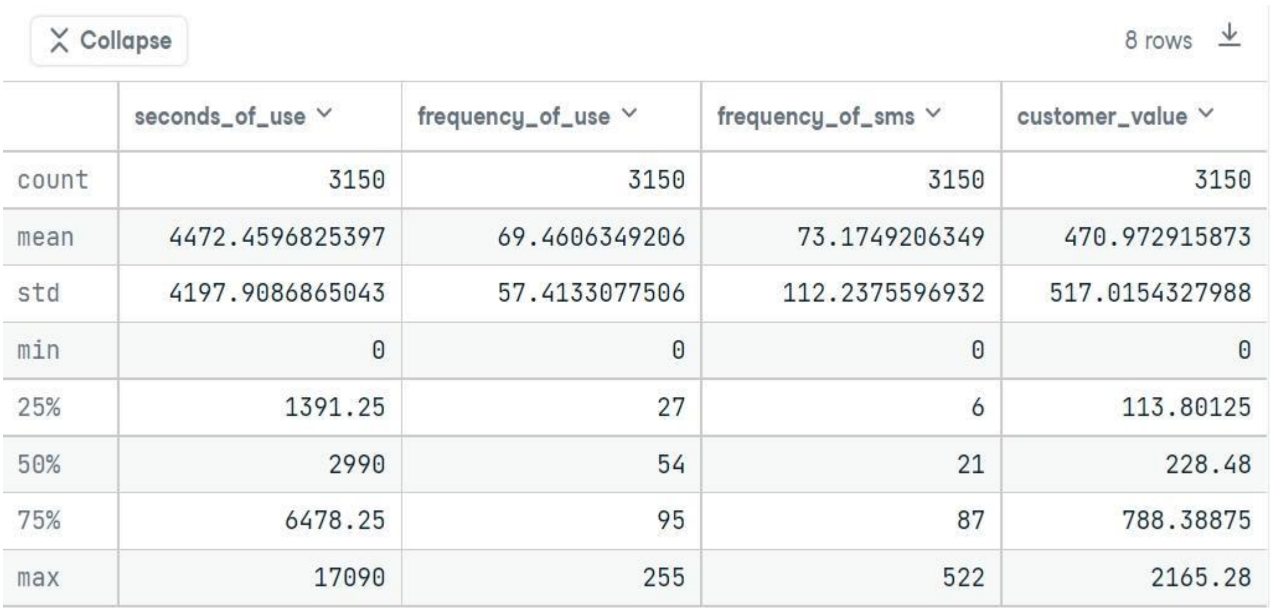
FROM customer_churn”””

Tel_data = pd.read_sql(query, conn)

Now, let’s look into the summary statistics for the numerical columns.

Tel_data[[“seconds_of_use”, “frequency_of_sms”, “customer_value”]].describe()

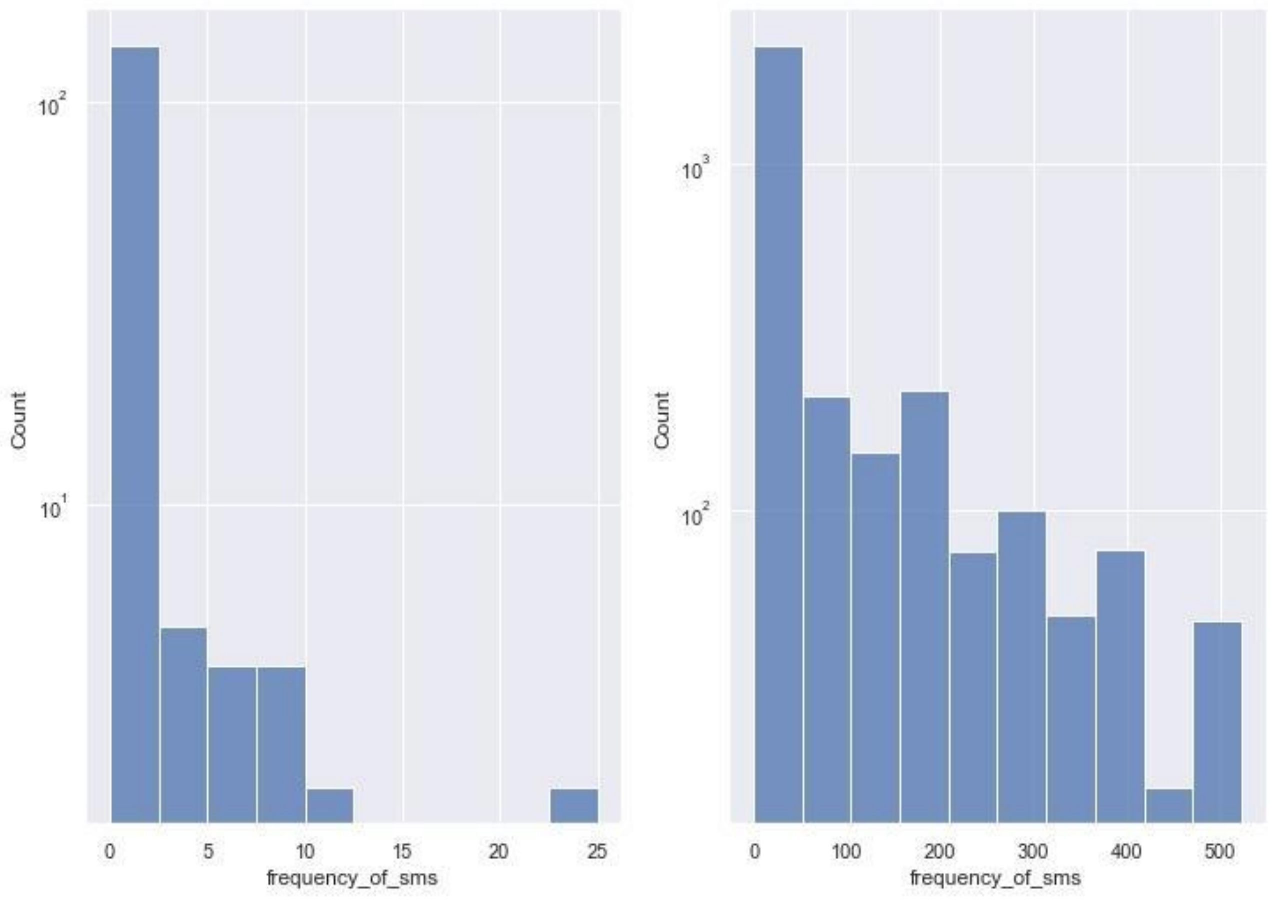
This code will give us the below output:



	seconds_of_use	frequency_of_use	frequency_of_sms	customer_value
count	3150	3150	3150	3150
mean	4472.4596825397	69.4606349206	73.1749206349	470.972915873
std	4197.9086865043	57.4133077506	112.2375596932	517.0154327988
min	0	0	0	0
25%	1391.25	27	6	113.80125
50%	2990	54	21	228.48
75%	6478.25	95	87	788.38875
max	17090	255	522	2165.28

```
Zero_data = tel_data[tel_data['seconds_of_use'] == 0]


Sns.set()

Fig,ax = plt.subplots(1, 2, figsize=(12,8))

Sns.histplot(x='frequency_of_sms', data=zero_data, ax=ax[0], bins=10)

Sns.histplot(x='frequency_of_sms', data=tel_data, ax=ax[1], bins=10)

Ax[0].set(yscale='log')

Ax[1].set(yscale='log')

Plt.show()
```

If you run the above code, you will see the below two plots.



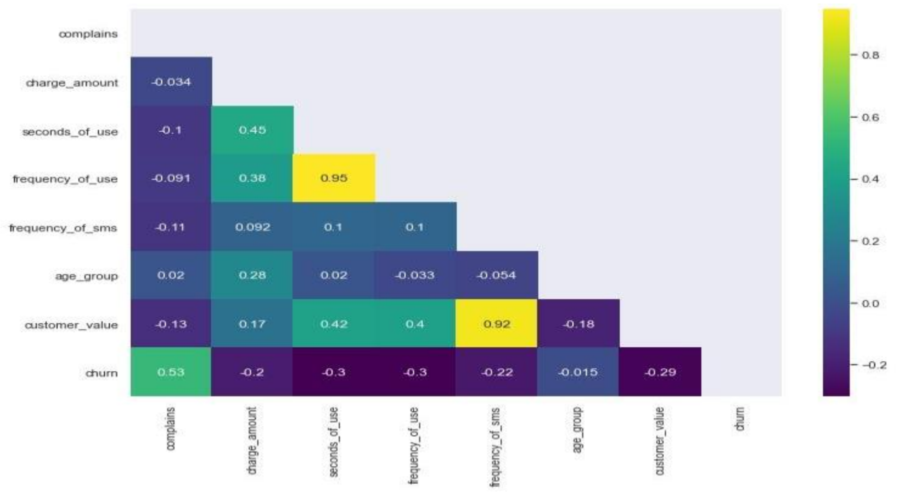
We can easily notice the difference between the above two plots. The left plot is drawn on that part of the dataset where seconds_of_use and frequency_of_use columns contain the value zero. The other one is drawn on the whole dataset. The first plot has the max value of SMS is 25 but in the other plot, this max value is more than 500! So we now came to the inference that the zero values in those columns are not missing values, those indicate that they are the new customers. Otherwise, we don't see any bar in the left plot.

If there is a large value difference in a column and we plot those values without using a log scale, it is very hard to come into some inference from the plot. So we have to use a log scale when we face such a problem.

Now, this data contains 8 columns.

```
Plt.figure(figsize=(12,8))

Corr = tel_data.corr()

Mask = np.triu(np.ones_like(corr, dtype=bool))

Sns.heatmap(corr, annot=True, mask=mask, cmap='viridis')

Plt.show()
```

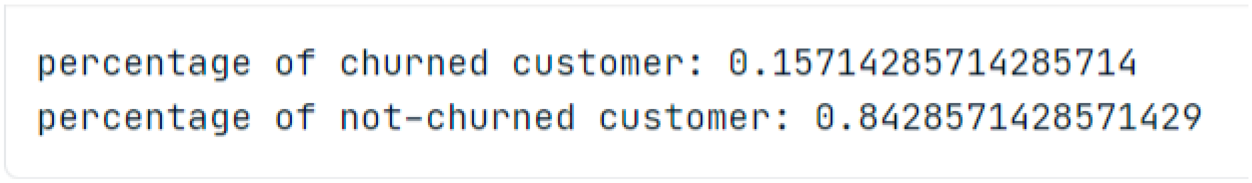
We got the image below as an output.



Dealing With Imbalance

Here we want to predict the churned customers properly. Let’s see how many rows are available for each class in the data.

```
churned = tel_data_selected[tel_data_selected[‘churn’]==1]

not_churned = tel_data_selected[tel_data_selected[‘churn’]==0]

print(‘percentage of churned customer:
{}’.format(churned.shape[0]/tel_data_selected.shape[0]))

print(‘percentage of not-churned customer:
{}’.format(not_churned.shape[0]/tel_data_selected.shape[0]))
```



```
percentage of churned customer: 0.15714285714285714
percentage of not-churned customer: 0.8428571428571429
```

The output

Hmm, only 15% of data are related to the churned customers and 84% of data are related to the non-churned customer. That’s a great difference. We have to oversample the minority class. For doing this, I am using SMOTE(Synthetic Minority Over-sampling TechniquE) which makes synthetic data using the characteristics of the nearest neighbours. This technique is available in the imblearn python library.

There is also one thing. We don’t want to test the ml model on fake data! So I am using this technique only on the training data. For this, we are splitting the data using train_test_split from scikit-learn.

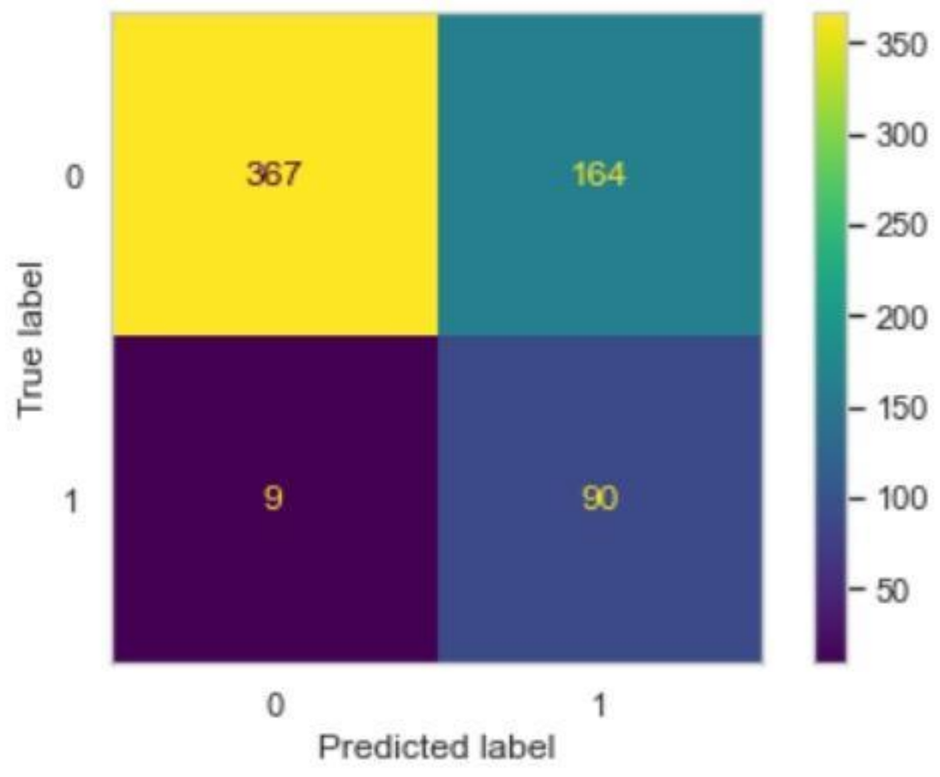
```
From collections import Counter

# splitting the data as X and y

X = tel_data_selected.drop('churn', axis=1)

Y = tel_data_selected['churn']

# making a SMOTE object

Resampler = SMOTE(random_state=5)

# splitting the data into train and test

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

# resampling the data

X_resampled, y_resampled = resampler.fit_resample(X_train, y_train)

# verifying the resampled data

Print('Resampled dataset shape %s' % Counter(y_resampled))
```

The output

Now our data is balanced. There is one more thing to do before training a model – Scaling the numeric data. Seconds_of_use, frequency_of_sms, and the customer_value columns are the numeric columns. Other columns are encoded categorical columns. So we have to scale only these 3 columns. For selecting the specific column for scaling I am using ColumnTransformer from scikit-learn and StandardScaler for scaling.

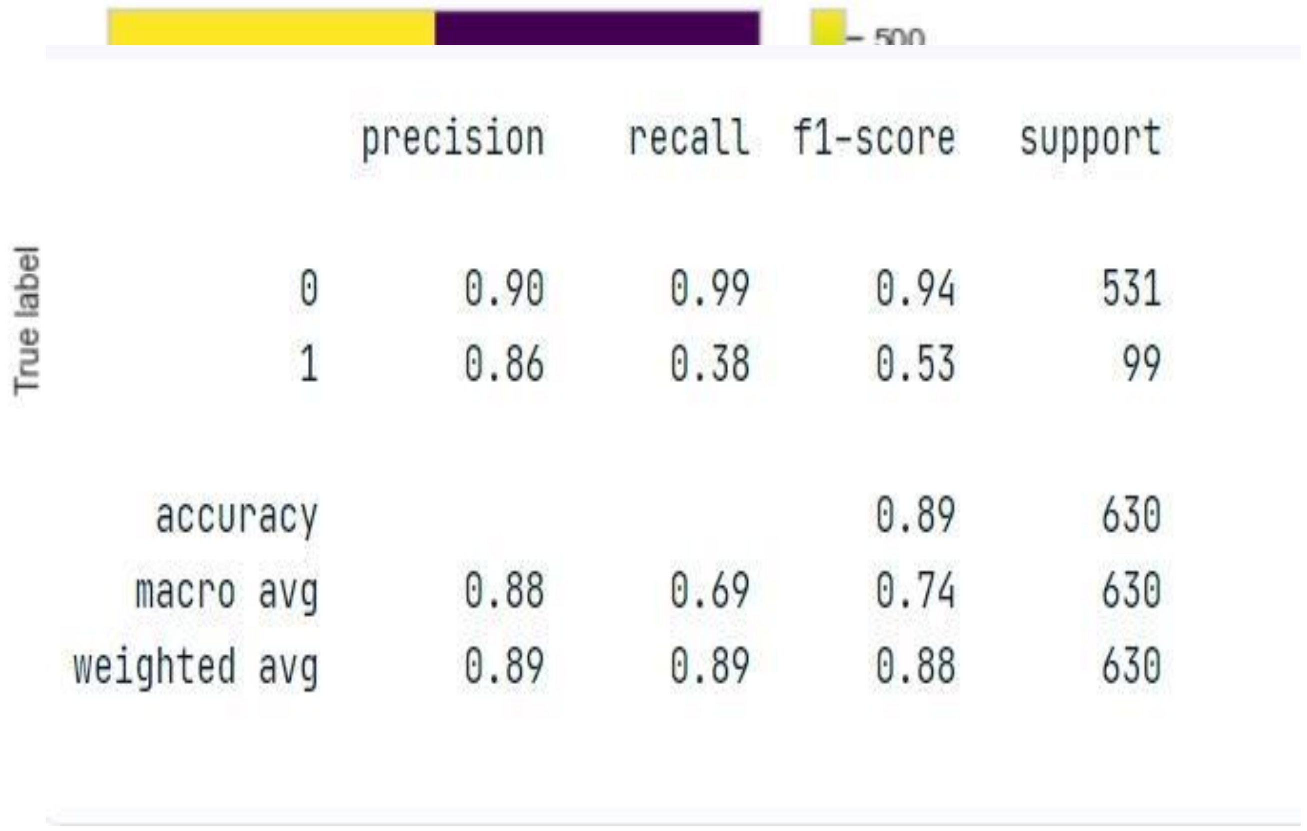
Now, let's plot the Confusion Matrix.

```
# this line removes the grid from the confusion matrix

Sns.set_style("whitegrid", {'axes.grid' : False})

ConfusionMatrixDisplay.from_predictions(y_test, y_pred)

Plt.show()
```



From the 99 churned customer samples, we are detecting 90 samples correctly and 9 are misclassified. This result might change in multiple reruns.

```
# the data directly scaled without resampling

X_train_scaled = ct.fit_transform(X_train)

X_test_scaled = ct.transform(X_test)



# fitting and predicting the model

Linear_reg.fit(X_train_scaled, y_train)

Y_pred2 = linear_reg.predict(X_test_scaled)



Print(classification_report(y_test, y_pred2))
```



```
              precision    recall  f1-score   support

           0       0.90      0.99      0.94       531
           1       0.86      0.38      0.53        99

    accuracy                           0.89       630
   macro avg       0.88      0.69      0.74       630
weighted avg       0.89      0.89      0.88       630
```

Conclusion

So, we can see from the above result that here the resampling technique gives a better result in predicting the churned customers. And from the analysis, we can see that not all features are important here. I take only 6 features out of 14 features but still model gives a decent result. You can also try the other features too, the journey does not end here.

This model can be improved by tuning the hyperparameters or using other algorithms other than LogisticRegression. We can also apply Cross-Validation for taking the best split of the data for training.