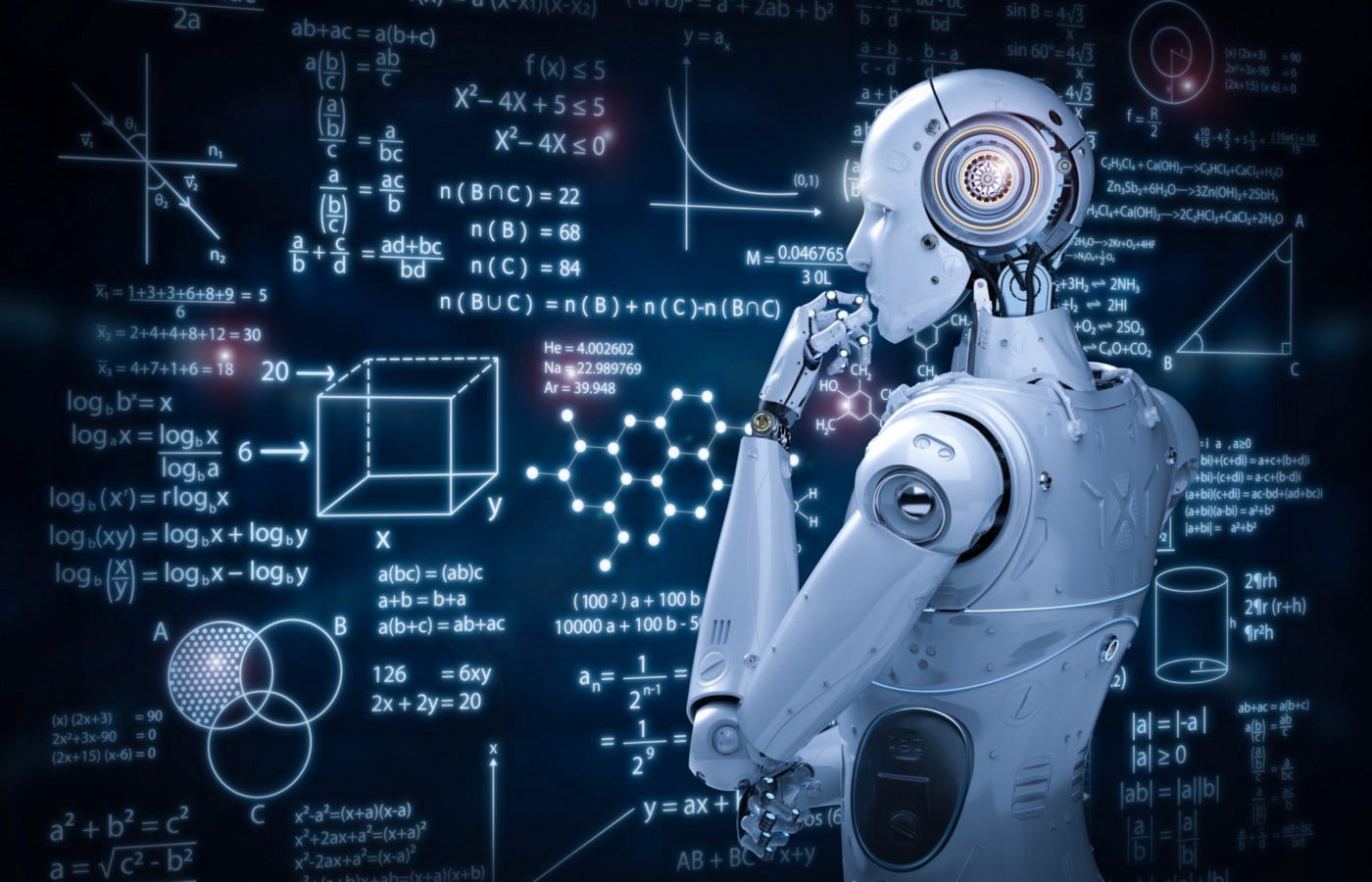
**CUSTOMER CHURN PREDICTION**



Introduction to Customer Churn Prediction

After taking some courses on Data Science, I feel a necessity for applying those skills to some projects. For this, I analyzed and made a machine learning model on a dataset that comes from an Iranian telecom company, with each row representing a customer over a year period. I took this dataset from ***Kaggle***. In this article, I am going to share my experience while working with this data. You will learn about

* Retrieving data from a database
* Handling imbalanced data
* Data Visualization
* Making a machine learning model

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 |

[view raw](https://gist.github.com/srang992/05d54d36b915dee6dd57712a205b2a20/raw/bac4c78947fdb6b414deb841154d284e5d60a733/needed_libraries.py)[needed\_libraries.py](https://gist.github.com/srang992/05d54d36b915dee6dd57712a205b2a20#file-needed_libraries-py)hosted with  by [GitHub](https://github.com/)

Retrieving Data from a Database

After taking the data from Kaggle, I push those data into a database for practicing a little bit of SQL while doing this project. You can find the database and the data as a **CSV**from my [Github repo](https://github.com/srang992/Telecom-customer-churn" \t "_blank). 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) |

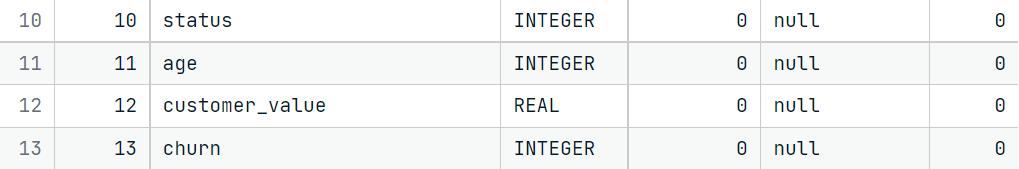
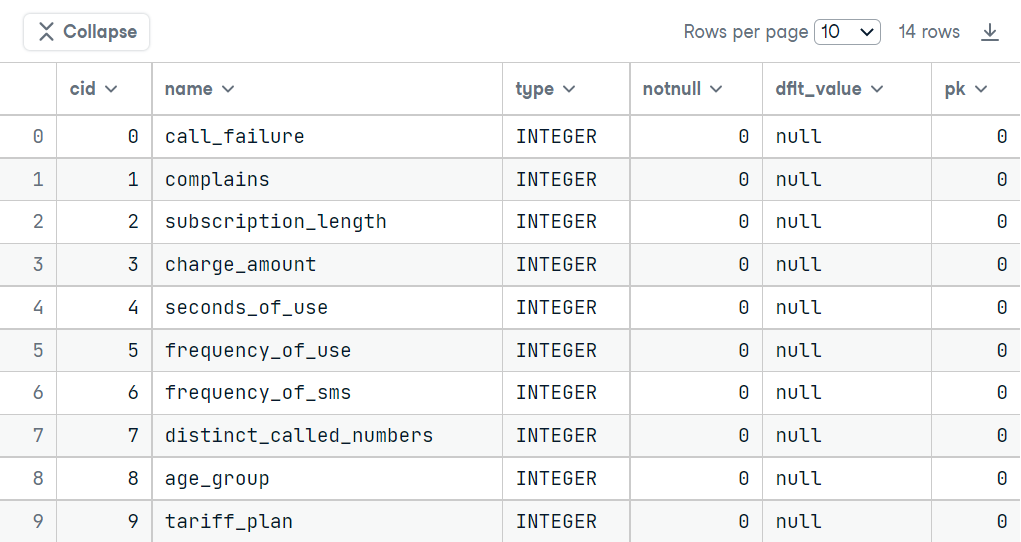
[view raw](https://gist.github.com/srang992/dc80f00dfa3adf8af80ea1cca6f7a01c/raw/8689a9faf8d875f2c1e752da05c8f9135e7f16ed/database_read.py)[database\_read.py](https://gist.github.com/srang992/dc80f00dfa3adf8af80ea1cca6f7a01c#file-database_read-py)hosted with  by [GitHub](https://github.com/)

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

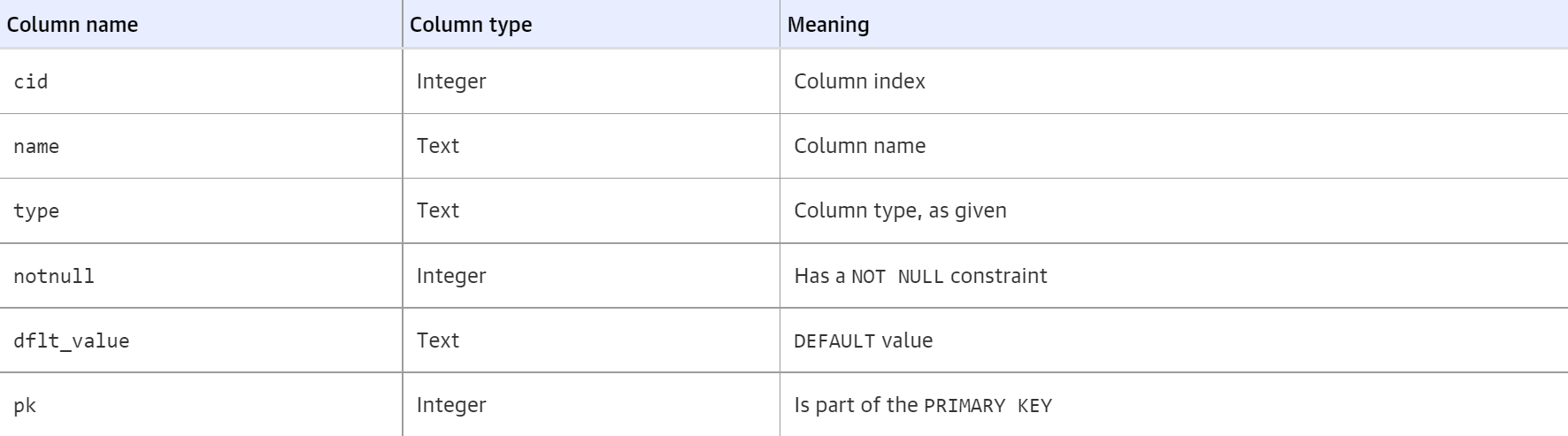


We can see from the above result that there is only one table in that database. That makes sense as I only pushed one table into the database. Now let’s see the structure of the ***customer\_churn***table. For this, we have to run the code belowRSVP!

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



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



So, from our result, we can say that:

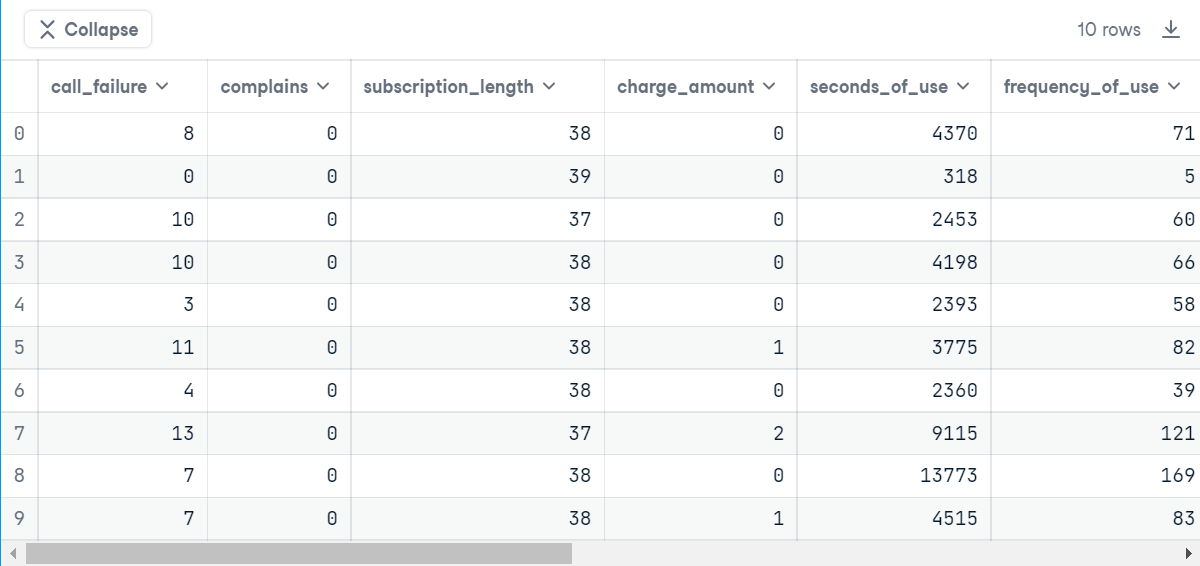
* There are 14 columns in our table (row count of the table).
* All columns consist of either INTEGER or REAL data types (***type*** column).
* There is no column in which the NOT NULL constraint is applied while making this table (***notnull*** column).
* The default value is all column is null (***dflt\_value***column).
* There is no primary key available (***pk*** column).

We came to know so many things in just 1 line of code, isn’t that awesome?

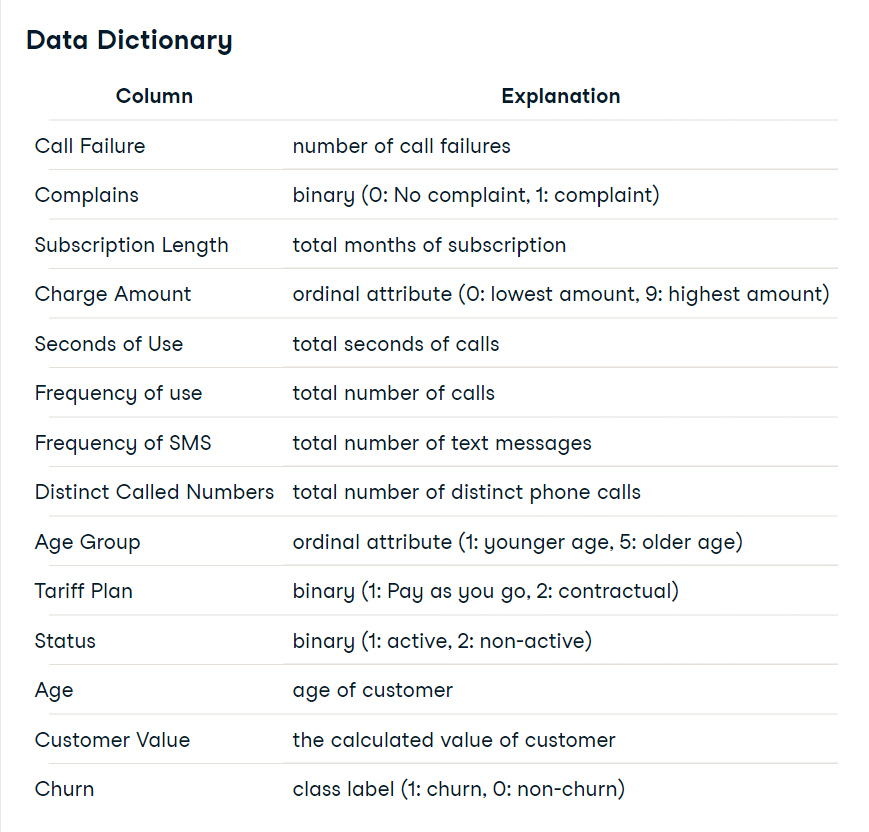
Now, let’s see some values in the table. Run the below code.

pd.read\_sql("SELECT \* FROM customer\_churn LIMIT 10", conn)

You will see the output something like this.



As there are so many columns, I am unable to display all of them. But I will give you a brief description of all the columns and their values to better understand the data. Just see the below image.



From the above image, you already guessed that though all columns are numerical, some encoded categorical columns exist. the encoded categorical columns are –  ***Complaints***, ***Charge Amount***,***Age Group***, ***Tariff Plan***, ***Status***, and ***Churn***. The ***Churn***column is the target column here.

Now, it’s time for reading this dataset as a pandas dataframe. But we don’t take all columns.

* 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.

Now, let’s write the code.

|  |  |
| --- | --- |
|  | # 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) |

[view raw](https://gist.github.com/srang992/cde573e67cf4775523d31c51d29dcd13/raw/9e2364238883ed62985274c5585a304ca79bc116/dataframe.py)[dataframe.py](https://gist.github.com/srang992/cde573e67cf4775523d31c51d29dcd13#file-dataframe-py)hosted with  by [GitHub](https://github.com/)

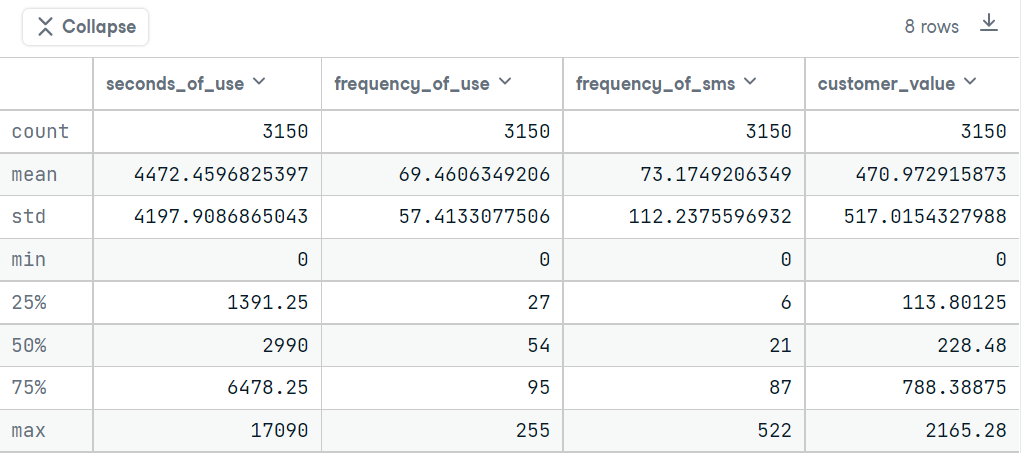
Now our data is ready to do some serious analysis.

Data Analysis for Customer Churn Prediction

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

tel\_data[["seconds\_of\_use", "frequency\_of\_use", "frequency\_of\_sms", "customer\_value"]].describe()

this code will give us the below output:



From the above statistics, we can easily notice that all numerical columns have a minimum value of 0. They may be the new customers or the missing values of these columns are present in those columns as 0. Which one is true? Let’s find out.

**assert**

tel\_data[tel\_data['seconds\_of\_use'] == 0].shape[0] ==

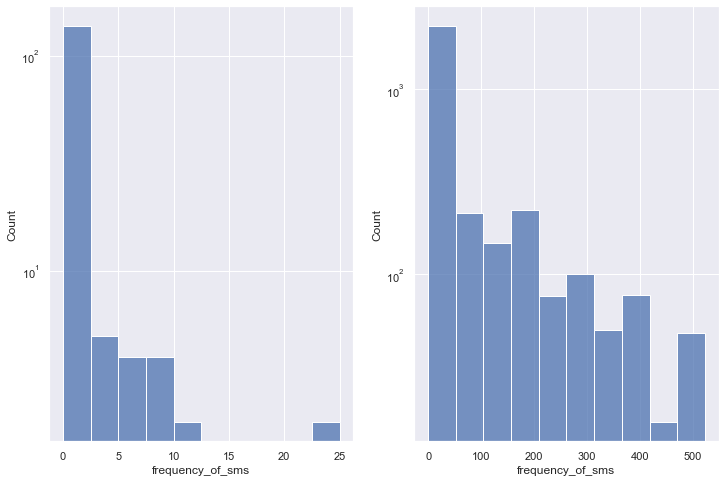
tel\_data[tel\_data['frequency\_of\_use'] == 0].shape[0]

if we run the above code, we got nothing as output. it may happen that these two columns – ***seconds\_of\_use*** and ***frequency\_of\_use***– carry the same importance in the data. Now we have to find out that the frequency\_of\_sms columns are 0 too where these two columns are 0? Let’s see.

|  |  |
| --- | --- |
|  | 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() |

[view raw](https://gist.github.com/srang992/205d0267ca120f8ca3e627f1d582b4b1/raw/3e893216024fadd1c264e88ff53a3e61fda5bfb3/first_plot.py)[first\_plot.py](https://gist.github.com/srang992/205d0267ca120f8ca3e627f1d582b4b1#file-first_plot-py)hosted with  by [GitHub](https://github.com/)

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.

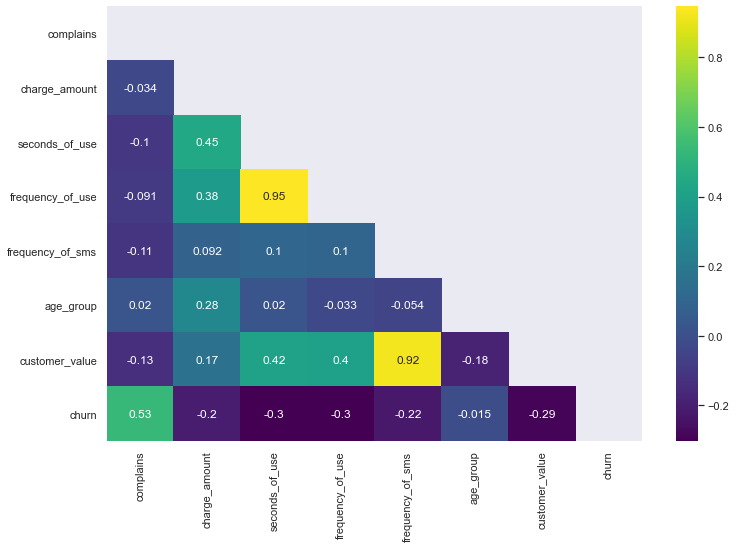
Now, you may notice that I used a log scale on the y-axis while plotting. Why? Because of the large difference between the values in the ***frequency\_of\_sms*** column. if there is a large value difference in a column and we plot those values without using a log scale, it is very hard to come into some inference from the plot. So we have to use a log scale when we face such a problem.

Now, this data contains 8 columns. Are all of them important? Let’s see by finding the correlation matrix.

|  |  |
| --- | --- |
|  | 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() |

[view raw](https://gist.github.com/srang992/d5bb3661e0262dd8296c3115eb8ef539/raw/b6189216786a3ef3343af509468e28b2c315596e/corr_plot.py)[corr\_plot.py](https://gist.github.com/srang992/d5bb3661e0262dd8296c3115eb8ef539#file-corr_plot-py)hosted with  by [GitHub](https://github.com/)

We got the image below as an output.



From the correlation matrix, we can easily notice that

* **frequency\_of\_use** and **seconds\_of\_use** have a high positive correlation between them. So we can put any one of them in the data.

I am removing the ***frequency\_of\_use*** column from the data.

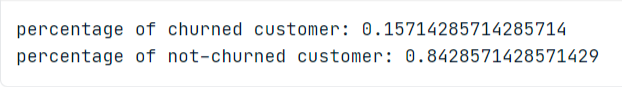
tel\_data\_selected = tel\_data.drop('frequency\_of\_use', axis=1)

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])) |

[view raw](https://gist.github.com/srang992/e4519c2964ff4a05160b12e5fe132dea/raw/04ebc2741ace931175067adfe264399e6ffccdfe/churned_percentage.py)[churned\_percentage.py](https://gist.github.com/srang992/e4519c2964ff4a05160b12e5fe132dea#file-churned_percentage-py)hosted with  by [GitHub](https://github.com/)



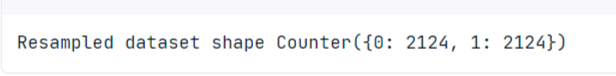
The output

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(**S**ynthetic **M**inority **O**ver-sampling **T**echniqu**E**) 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)) |

[view raw](https://gist.github.com/srang992/1b436b2f7dff380229eaf1061198f684/raw/18d96156645c3c45b9d9241cc33203ab81a3b6dc/imbalance.py)[imbalance.py](https://gist.github.com/srang992/1b436b2f7dff380229eaf1061198f684#file-imbalance-py)hosted with  by [GitHub](https://github.com/)

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.

|  |  |
| --- | --- |
|  | # making a ColumnTransformer object |
|  | ct = ColumnTransformer( |
|  | [('scaler', StandardScaler(), ['seconds\_of\_use', 'frequency\_of\_sms', 'customer\_value'])], remainder='passthrough') |
|  |  |
|  | # transforming data |
|  | X\_scaled = ct.fit\_transform(X\_resampled) |
|  | X\_test\_scaled = ct.transform(X\_test) |

[view raw](https://gist.github.com/srang992/f57192e06f04c6184fb6506b483969c6/raw/fb6793ec3463857fd5da8d4dfdcc58d9074592c5/scale.py)[scale.py](https://gist.github.com/srang992/f57192e06f04c6184fb6506b483969c6#file-scale-py)hosted with  by [GitHub](https://github.com/)

To know about the *ColumnTransformer,*see [here](https://scikit-learn.org/stable/modules/generated/sklearn.compose.ColumnTransformer.html).

Training a Model

Now it’s the time for model training. Here I am using the *LogisticRegression* for classification.

|  |  |
| --- | --- |
|  | # making an object for LogisticRegression |
|  | linear\_reg = LogisticRegression() |
|  |  |
|  | # fitting the data |
|  | linear\_reg.fit(X\_scaled, y\_resampled) |
|  |  |
|  | # predicting on x\_test |
|  | y\_pred = linear\_reg.predict(X\_test\_scaled) |

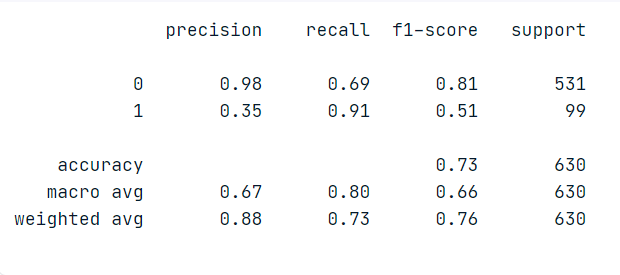
[view raw](https://gist.github.com/srang992/36f69ee43ae52cabe12fc39ab5e6f31e/raw/f67157f4fc3a557e1f6bde6200e114d82322661e/model.py)[model.py](https://gist.github.com/srang992/36f69ee43ae52cabe12fc39ab5e6f31e#file-model-py)hosted with  by [GitHub](https://github.com/)

I don’t make the model much more complex here so that a beginner can easily understand it. I am using all the default hyperparameters for Logistic Regression.

Now we have to check that this default model works well in predicting the churned customer. As we are only interested in a specific class here, so I am using recall as an evaluation metric. Recall calculates the percentage of actual positives a model correctly identified (True Positive). Here we are predicting the churned customers which are our positive class. Let’s see what we got.

from sklearn.metrics import classification\_report, ConfusionMatrixDisplay

print(classification\_report(y\_test, y\_pred))



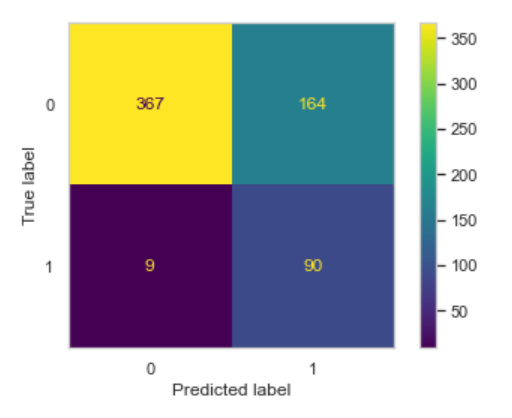
The output

From the above classification report, we can see that the model gives a decent score for predicting the customer as churned, which is great!

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() |

[view raw](https://gist.github.com/srang992/c806ad3240b4ca845fd62a1c79abf503/raw/e2aab81797d9c8af2a06be581f7def51843435bf/report.py)[report.py](https://gist.github.com/srang992/c806ad3240b4ca845fd62a1c79abf503#file-report-py)hosted with  by [GitHub](https://github.com/)

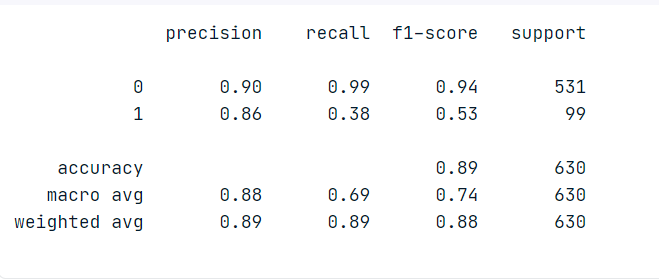


From the 99 churned customer samples, we are detecting 90 samples correctly and 9 are misclassified. This result might change in multiple reruns.

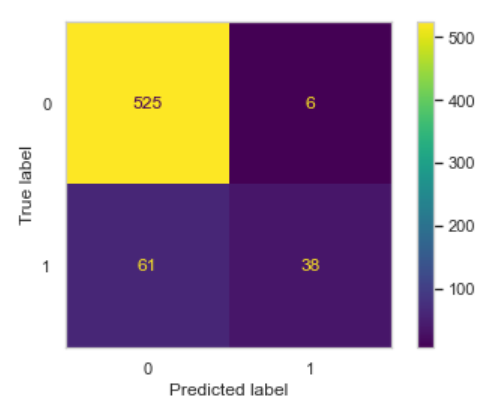
What happens if we just pass the data without resampling? Let’s see.

|  |  |
| --- | --- |
|  | # 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)) |

[view raw](https://gist.github.com/srang992/2198024b98eee113e6299d0a95a71050/raw/6ca98987f4aa8f82f8ff55c111b2f88a33b3281f/model_without_resample_data.py)[model\_without\_resample\_data.py](https://gist.github.com/srang992/2198024b98eee113e6299d0a95a71050#file-model_without_resample_data-py)hosted with  by [GitHub](https://github.com/)



Now we are getting a bad recall score. What about the Confusion Matrix? Let’s see.



From the confusion matrix, we can see that this model is not working well for detecting the churned customer.

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.