Sri Lanka Institute of Information Technology

Machine Learning (SE4060)

Assignment 01 - Report

Predicting Customer Churn in Telecommunication

Using

Logistic Regression

IT16170094 – Navanjani D.H.K.H.

April 2020

Table of Contents

1. Introduction

Customer churn also known as customer attrition, customer turnover, or customer defection is a financial term used when a customer eliminates to engage with a business. Even though companies usually prefer to retain as much as customers as they can, customer churn can happen because of a variety of reasons like insufficient engagement, vulnerable product-market fit, product bugginess, tough user experience, absent of proactive support, etc. The churn rate refers to the rate that consumers abandon a business within a given time limit. A churn rate beyond a particular threshold may have both tangible and intangible consequences on a company's business success.

It is very costly to win the churned customers back once lost regardless of the business domain. Therefore, customer churn is a major business problem that organizations are facing in an expeditiously varying highly competitive market place. Churned Customers means a direct loss to any business. Hence, predicting customers who might churn may help to avoid this loss.

Due to the direct effect on the profits of the businesses, especially in the telecommunication sector, customer churn is one of the most important concerns and companies are seeking to develop means to predict at-risk customers. Hence, finding factors that possibly cause to customer churn is crucial to take required actions to mitigate customer churn.

Apparently, it is very essential to predict the users' potential to churn from a business and the causes affecting the customer choices. This report depicts how Logistic Regression and Python can be utilizing to recognize customer churn with the aid of the Telco Customer Churn dataset.

2. Dataset

2.1. Description

Telco Customer Churn dataset [1], which is concentrated on programs about customer retention, will be used for the customer churn prediction. The dataset is available at https://www.kaggle.com/blastchar/telco-customer-churn . In accordance with the IBM Sample Data Sets, the context of this dataset is to retain customers by assessing related customer information, predicting their behaviors and developing dedicated customer retention programs.

2.2. Features

As expressed in Table 2.1, this dataset contains 21 columns (features) along with 7043 rows (customers).

Attribute	Description
customerID	Specifies the Customer ID
gender	Specifies whether the customer is a male or a female
SeniorCitizen	Specifies whether the customer is a senior citizen or not
Partner	Specifies whether the customer has a partner or not
Dependents	Specifies whether the customer has dependents or not
tenure	Defines the number of months the customer has stayed with the company
PhoneService	Specifies whether the customer has a phone service or not
MultipleLines	Specifies whether the customer has multiple lines or not
InternetService	Defines customer's internet service provider
OnlineSecurity	Specifies whether the customer has online security or not
OnlineBackup	Specifies whether the customer has online backup or not
DeviceProtection	Specifies whether the customer has device protection or not
TechSupport	Specifies whether the customer has tech support or not
StreamingTV	Specifies whether the customer has streaming TV or not
StreamingMovies	Specifies whether the customer has streaming movies or not
Contract	Defines the contract term of the customer
PaperlessBilling	Specifies whether the customer has paperless billing or not
PaymentMethod	Defines the customer's payment method
MonthlyCharges	Defines the amount charged to the customer monthly
TotalCharges	Defines the total amount charged to the customer
Churn	Specifies whether the customer churned or not

Table 2.1

The "Churn" column is the target which represents whether a customer churned (yes) or not (no).

3. Methodology

3.1. Logistic Regression

3.1.1. Overview

Since it is a convenient algorithm that executes flawlessly on a wide range of scenarios, Logistic Regression is one of the most well-known supervised learning classification algorithms for solving binary classification problems by predicting the probability of a target variable.

The outcome/target variable can have only two possible classes (dichotomous). By assessing probabilities and applying its underlying logistic function, Logistic Regression estimates the correlation between the target and the features. This outputs a probability ($0 \leq x \leq 1$), which can be utilized to predict the binary 0 or 1 as the output (if $x<0.5$, output= 0, else output=1).

3.1.2. Assumptions

- In binary logistic regression, the target variables must be binary and the desired outcome is represented by factor level 1
- The independent variables must be independent of each other (no multi-co linearity)
- Variables included in the model must be meaningful
- Sample size should be large

3.1.3. Functionality

The sigmoid/ logistic function is a fundamental trait in logistic regression classification. This function takes in any value and outputs it to be between 0and 1. Figure 3.1 depicts the sigmoid function graph and the logistic/sigmoid function is specified in Figure 3.2.

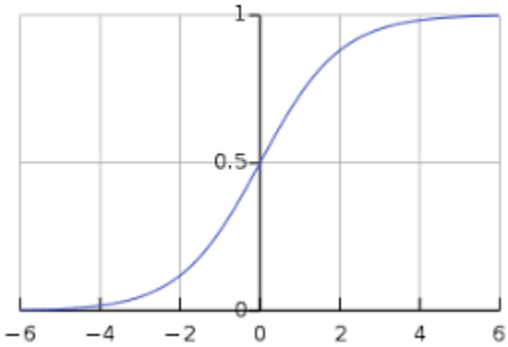

Figure 3.1

$$g(z) = \frac{1}{1+e^{-z}} where\ z = \emptyset^T x$$

Figure 3.2

Decision boundary differentiates probabilities into positive and negative classes based on a defined threshold.

$$J(\theta) = -\frac{1}{m}\sum\left[y^{(i)}\log(h\theta(x(i))) + \left(1 - y^{(i)}\right)\log(1 - h\theta(x(i)))\right]$$

Figure 3.3

Want $\min_\theta J(\theta)$:

Repeat $\{$

$$\theta_j := \theta_j - \alpha\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

(simultaneously update all θ_j)

$\}$

Figure 3.4

Minimizing the cost function (Figure 3.3) helps to develop an accurate model with minimum error. Following the logistic regression hypothesis, gradient descent (Figure 3.4), can be used to reduce the cost value and retain the cost function within 0 and 1.

3.1.4. Advantages and disadvantages

Advantages:

- Implementation is straightforward
- Model training is efficient
- Required computational resources are limited
- Interpretability is wide
- Input feature scaling is not necessary
- Tuning of hyper parameters not needed
- Regularizing is convenient
- Can be used for multiclass classifications too

Disadvantages:

- Poor performance on non-linear data (image data)
- Poor performance with irrelevant and highly correlated features
- Challenging with situations which includes non-linear classifications
- Co linearity and outliers tampers the accuracy

4. Implementation

4.1. Importing the libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pylab import rcParams
%matplotlib inline
```

Figure 4.1

As mentioned in Figure 4.1, necessary libraries and dependencies have been imported initially. *NumPy* has been used for linear algebra domain. *Pandas* has been applied for data processing, importing *CSV* data files etc. Plots has been generated with the aid of *Pyplot*. Statistical data visualization has been done with the use of *Seaborn.*

```
from sklearn.preprocessing import LabelEncoder
```

Figure 4.2

As indicated in Figure 4.2, *Sklearn LabelEncoder* has been used for transforming categorical text data into model- comprehensible numerical data.

```
from sklearn.model_selection import train_test_split
```

Figure 4.3

As denoted in Figure 4.3, *Sklearn train_test_split* function has been applied for splitting the dataset in to training and testing.

```
from sklearn.linear_model import LogisticRegression
```

Figure 4.4

As pointed out in Figure 4.4, *Logistic Regression* algorithm has been imported for making the predictions.

```
from sklearn.metrics import recall_score, confusion_matrix, precision_score, f1_score, accuracy_score, classification_report
```

Figure 4.5

As indicated in Figure 4.5, the required imports has been done for computing *precision, recall, accuracy* and *f1-score* with *Sklearn*. For the evaluation of the classification accuracy, *confusion matrix* has been computed. With the aid of *classification report* function, a text report has been generated which includes the main classification metrics.

4.2. Loading the dataset

```
churn_data = pd.read_csv("C:\\Users\\Himashi\\Desktop\\WA_Fn-UseC_-Telco-Customer-Churn.csv")
churn_data.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	TechSup
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...	No	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	...	Yes	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...	No	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	...	Yes	
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	...	No	

5 rows × 21 columns

Figure 4.6

As expressed in Figure 4.6, using the *read_csv* function in *Pandas,* the *CSV* data file has been loaded in to the *Pandas DataFrame* and with the aid of the *head()* method, the top 5 rows of the *DataFrame* has been retrieved.

4.3. Exploring the data

```
churn_data.shape
(7043, 21)
```

Figure 4.7

Pandas dataframe.shape function has been used to retrieve the tuple shape (rows, columns) of the dataset. According to the Figure 4.7, there are 7043 data objects and 21 attributes.

```
churn_data.columns.tolist()
```

```
['customerID',
 'gender',
 'SeniorCitizen',
 'Partner',
 'Dependents',
 'tenure',
 'PhoneService',
 'MultipleLines',
 'InternetService',
 'OnlineSecurity',
 'OnlineBackup',
 'DeviceProtection',
 'TechSupport',
 'StreamingTV',
 'StreamingMovies',
 'Contract',
 'PaperlessBilling',
 'PaymentMethod',
 'MonthlyCharges',
 'TotalCharges',
 'Churn']
```

Figure 4.8

Pandas dataframe.columns.tolist() function has been used to retrieve a list of column names in *the dataset as shown in the Figure* 4.8.

```
churn_data.dtypes
```

```
customerID          object
gender              object
SeniorCitizen        int64
Partner             object
Dependents          object
tenure               int64
PhoneService        object
MultipleLines       object
InternetService     object
OnlineSecurity      object
OnlineBackup        object
DeviceProtection    object
TechSupport         object
StreamingTV         object
StreamingMovies     object
Contract            object
PaperlessBilling    object
PaymentMethod       object
MonthlyCharges     float64
TotalCharges        object
Churn               object
dtype: object
```

Figure 4.9

Pandas dataframe.dtypes has been used to retrieve the data type of individual column as expressed in the Figure 4.9.

```
churn_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
customerID          7043 non-null object
gender              7043 non-null object
SeniorCitizen       7043 non-null int64
Partner             7043 non-null object
Dependents          7043 non-null object
tenure              7043 non-null int64
PhoneService        7043 non-null object
MultipleLines       7043 non-null object
InternetService     7043 non-null object
OnlineSecurity      7043 non-null object
OnlineBackup        7043 non-null object
DeviceProtection    7043 non-null object
TechSupport         7043 non-null object
StreamingTV         7043 non-null object
StreamingMovies     7043 non-null object
Contract            7043 non-null object
PaperlessBilling    7043 non-null object
PaymentMethod       7043 non-null object
MonthlyCharges      7043 non-null float64
TotalCharges        7043 non-null object
Churn               7043 non-null object
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

Figure 4.10

As indicated in Figure 4.10, *Pandas dataframe.info*() function has been used to get a concise summary of the *dataframe*.

4.4. Understanding data with visualization

4.4.1. Reviewing the distribution of the target feature: churn

```
churn_data['Churn'].value_counts(sort = False)
```

```
No     5174
Yes    1869
Name: Churn, dtype: int64
```

Figure 4.11

Pandas.Series.value_counts function has been applied to retrieve a list including counts of unique values as demonstrated in Figure 4.11. According to the above output, there are 1869 churned customers and 5174 non- churn customers within the provided dataset. According to the percentage shown in Figure 4.12, there are 26.5% of churned customers making the target slightly unbalanced.

```
labels =churn_data['Churn'].value_counts(sort = True).index
sizes = churn_data['Churn'].value_counts(sort = True)
colors = ["blue","red"]
explode = (0.1,0)
rcParams['figure.figsize'] = 8,8
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, startangle=270,)
plt.title('Percent of customer churn')
plt.show()
```

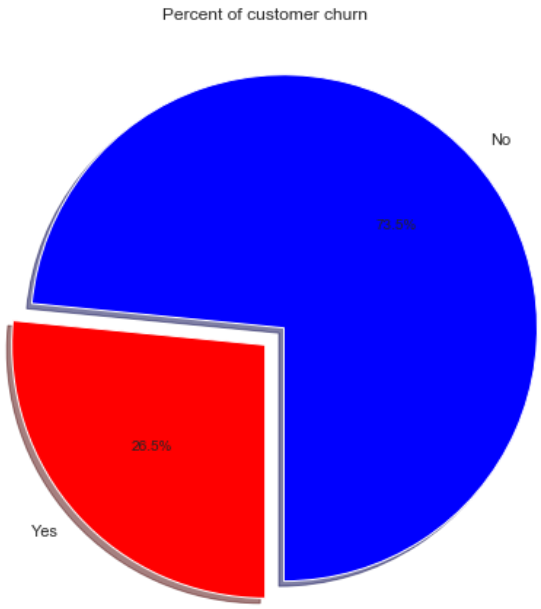

Figure 4.12

4.5. Manipulating the data

4.5.1. Converting data types

```
churn_data['TotalCharges'] = pd.to_numeric(churn_data.TotalCharges, errors='coerce')
```

Figure 4.13

```
churn_data["SeniorCitizen"] = churn_data["SeniorCitizen"].replace({1:"Yes",0:"No"})
```

Figure 4.14

As mentioned in Figure 4.13, the *TotalCharges* column has been converted from object type to numeric type. As indicated in Figure 4.14, the *SeniorCitizen* column has been converted from numeric type to object type.

4.5.2. Generating descriptive statistics

```
churn_data.describe()
```

	tenure	MonthlyCharges	TotalCharges
count	7043.000000	7043.000000	7032.000000
mean	32.371149	64.761692	2283.300441
std	24.559481	30.090047	2266.771362
min	0.000000	18.250000	18.800000
25%	9.000000	35.500000	401.450000
50%	29.000000	70.350000	1397.475000
75%	55.000000	89.850000	3794.737500
max	72.000000	118.750000	8684.800000

Figure 4.15

Pandas dataframe.describe() function has been applied to generate some basic descriptive statistics as denoted in Figure 4.15.

4.5.3. Checking for missing values

```
churn_data.isnull().sum()
```

```
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
dtype: int64
```


Figure 4.16

Pandas dataframe.isnull().sum() has been applied to count the missing values in respective columns as indicated in Figure 4.16. According to the above output, there are 11 missing values for *Total Charges*.

4.5.4. Dropping missing values

```
churn_data.dropna(inplace = True)
```

Figure 4.17

As indicated in Figure 4.17, with the use of *Pandas dropna()* method, missing values in the *Total Charges* column has been dropped from the *dataframe*.

4.5.5. Dropping unnecessary columns

```
churn_data.drop(["customerID"],axis=1,inplace = True)
```

Figure 4.18

As indicated in Figure 4.18, the *customerID* column has been dropped since it is not required for the analysis.

4.6. Data visualization and analysis

4.6.1. Reviewing the distribution of categorical features

```
sns.set(style="ticks", color_codes=True)

fig, axes = plt.subplots(nrows = 4,ncols = 4,figsize = (25,15))
sns.countplot(x = "gender", data = churn_data, hue = churn_data.Churn, ax=axes[0][0])
sns.countplot(x = "SeniorCitizen", data = churn_data, hue = churn_data.Churn, ax=axes[0][1])
sns.countplot(x = "Partner", data = churn_data, hue = churn_data.Churn, ax=axes[0][2])
sns.countplot(x = "Dependents", data = churn_data, hue = churn_data.Churn, ax=axes[0][3])
sns.countplot(x = "PhoneService", data = churn_data, hue = churn_data.Churn, ax=axes[1][0])
sns.countplot(x = "MultipleLines", data = churn_data, hue = churn_data.Churn, ax=axes[1][1])
sns.countplot(x = "InternetService", data = churn_data, hue = churn_data.Churn, ax=axes[1][2])
sns.countplot(x = "OnlineSecurity", data = churn_data, hue = churn_data.Churn, ax=axes[1][3])
sns.countplot(x = "OnlineBackup", data = churn_data, hue = churn_data.Churn, ax=axes[2][0])
sns.countplot(x = "DeviceProtection", data = churn_data, hue = churn_data.Churn, ax=axes[2][1])
sns.countplot(x = "TechSupport", data = churn_data, hue = churn_data.Churn, ax=axes[2][2])
sns.countplot(x = "StreamingTV", data = churn_data, hue = churn_data.Churn, ax=axes[2][3])
sns.countplot(x = "StreamingMovies", data = churn_data, hue = churn_data.Churn, ax=axes[3][0])
sns.countplot(x = "Contract", data = churn_data, hue = churn_data.Churn, ax=axes[3][1])
sns.countplot(x = "PaperlessBilling", data = churn_data, hue = churn_data.Churn, ax=axes[3][2])
ax=sns.countplot(x = "PaymentMethod", data = churn_data, hue = churn_data.Churn, ax=axes[3][3])
ax.set_xticklabels(ax.get_xticklabels(),rotation=90)

plt.show(fig)
```

Figure 4.19

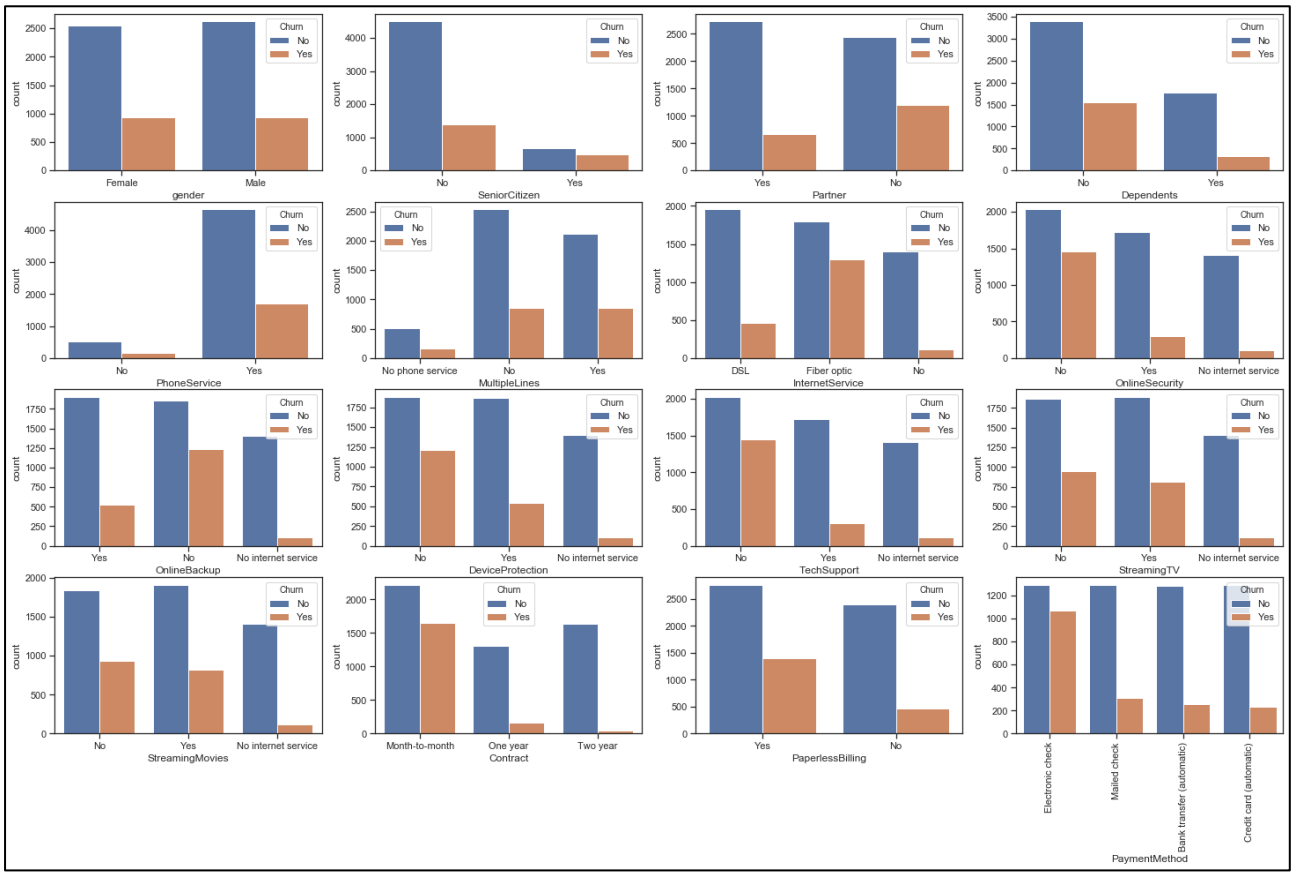

Figure 4.20

This dataset has 16 categorical features including; six binary features, nine features with three unique values each and one feature including four unique values.

As indicated in Figure 4.19, the *Python Seaborn* data visualization library has been used to generate count plots for these categorical features analysis. By reviewing the resulting *count plots* shown in Figure 4.20, some significant observations can be made as listed below.

- Gender, PhoneService, MultipleLines, and Streaming service don't have clear difference churn rates between the values in categories.
- The churn rate of non-senior customers is lesser compared to the Senior citizens.
- Customers that have partners/dependents are less likely to churn.
- Customers who do not use internet have a very low churn rate.
- Customers with Fiber optic connections are more probable to churn compared to the customers having a DSL connection.
- Customers that don’t use OnlineSecurity, OnlineBackup, DeviceProtection, and TechSupport services tend to churn more.
- Consumers that use PaperlessBilling are more prone to churn.
- Customers having short term contracts have higher churn rates.
- Customers that use the Electronic check Payment method have a high churn rate.

4.6.2. Reviewing the distribution of numerical features

```
sns.pairplot(churn_data,vars = ['tenure','MonthlyCharges','TotalCharges'], hue="Churn")
```

Figure 4.21

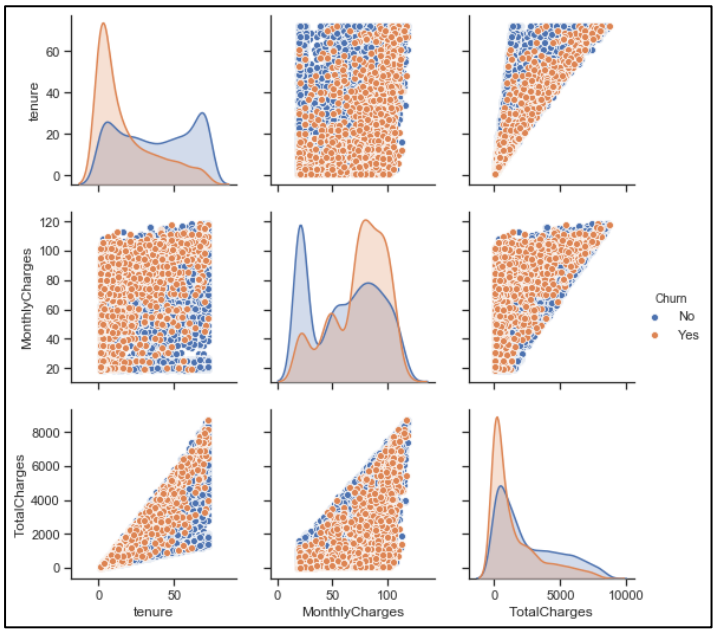

Figure 4.22

There are 3 numerical columns in this dataset:

- *Tenure*
- *MonthlyCharges*
- *TotalCharges*

As indicated in Figure 4.21, *Python Seaborn* data visualization library has been used to generate *pair plots* for these categorical features analysis. The resulting pair plots shown in Figure 4.22 depict both distribution of single variables and relationships between two variables. As per the above plots, it is obvious that customers having lower tenure and higher monthly charges tend to churn more.

4.7. Data preprocessing

```
from sklearn.preprocessing import LabelEncoder

def object_to_int(dataframe_series):
    if dataframe_series.dtype=='object':
        dataframe_series = LabelEncoder().fit_transform(dataframe_series)
    return dataframe_series
```

```
churn_data = churn_data.apply(lambda x: object_to_int(x))
```

Figure 4.23

Applying *SciKit-learn Label-Encoder* library, all the categorical variables has been converted to numerical, which can be easily understandable for the predictive models as noted in Figure 4.23.

4.8. Reviewing correlation of the features

```
churn_data.corr()["Churn"].sort_values()
```

```
Contract          -0.396150
tenure            -0.354049
OnlineSecurity    -0.289050
TechSupport       -0.282232
TotalCharges      -0.199484
OnlineBackup      -0.195290
DeviceProtection  -0.177883
Dependents        -0.163128
Partner           -0.149982
InternetService   -0.047097
StreamingMovies   -0.038802
StreamingTV       -0.036303
gender            -0.008545
PhoneService       0.011691
MultipleLines      0.038043
PaymentMethod      0.107852
SeniorCitizen      0.150541
PaperlessBilling   0.191454
MonthlyCharges     0.192858
Churn              1.000000
Name: Churn, dtype: float64
```

Figure 4.24

As shown in Figure 4.24, *Pandas dataframe.corr()* has been used to find the pair wise correlation of all columns in the dataframe.

```
f,ax = plt.subplots(figsize=(25, 25))
sns.heatmap(churn_data.corr(), annot=True, linewidths=.5, fmt= '.2f',ax=ax)
plt.show()
```

Figure 4.25

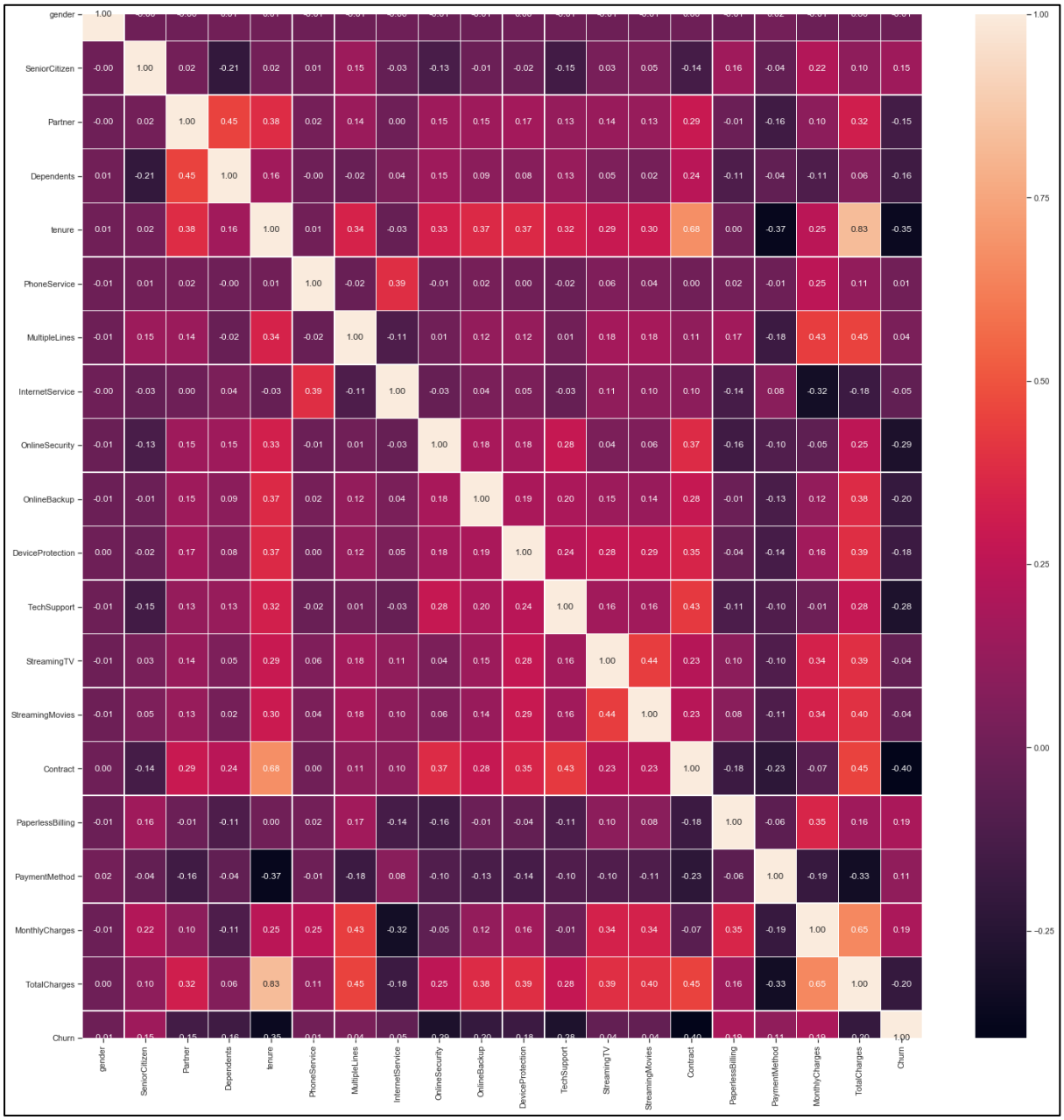

Figure 4.26

Since this dataset includes multiple variables, *Seaborn Heatmap* has been used for visualizing the *correlation matrix* of data. Figure 4.26 depicts the resulting *Seaborn Heatmap* which has been generated using the code segment specified in the Figure 4.25.

4.9. Splitting the dataset

```
#Assign the target
y = churn_data.Churn.values

#Drop the target, retaining only numerical features
new_churn_data = churn_data.drop(["Churn"],axis=1)

#Normalize values to fit between 0 and 1.
x = (new_churn_data-np.min(new_churn_data))/(np.max(new_churn_data)-np.min(new_churn_data)).values
```

Figure 4.27

```
#Split dataset into Train and Test
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,random_state =1)
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)
```

```
(5625, 19) (5625,) (1407, 19) (1407,)
```

Figure 4.28

Initially, the target has been assigned to *y* and the features have been assigned to *x* as represented in Figure 4.27. Thereafter, the dataset has been divided to *train* and *test* with 20% of the data given to the test set. Figure 4.28 depicts the shape of the separated data.

4.10. Applying Machine Learning Algorithm, creating and train the model

```
from sklearn.linear_model import LogisticRegression
lr_model = LogisticRegression()
lr_model.fit(x_train,y_train)
accuracy_lr = lr_model.score(x_test,y_test)
print("Logistic Regression accuracy is :",accuracy_lr)
```

```
Logistic Regression accuracy is : 0.7953091684434968
```

Figure 4.29

Initially, a *LogisticRegression* object has been created as shown in Figure 4.29. Then, to train the model, the *.fit()* method has been used. Upon completion of the model training, it can be observed that *Logistic Regression* provides an accuracy score of 0.7953.

4.11. Evaluating the model

Although *Logistic Regression* provides 79.53% accuracy, it is required to consider *confusion matrix, recall*, and *precision* since the data is imbalanced.

Therefore, since the data set is imbalanced, the *F1 score* will be considered more rather than accuracy.

```
from sklearn.metrics import recall_score, confusion_matrix, precision_score, f1_score, accuracy_score, classification_report

cm_lr = confusion_matrix(y_test,lr_model.predict(x_test))

f, ax = plt.subplots(figsize = (5,5))
sns.heatmap(cm_lr, annot = True, linewidths = 0.5, color = "red", fmt = ".2f", ax=ax)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.xlabel("y_predicted")
plt.ylabel("y_true")
plt.title("Confusion Matrix of Logistic Regression")
plt.show()
```

Figure 4.30

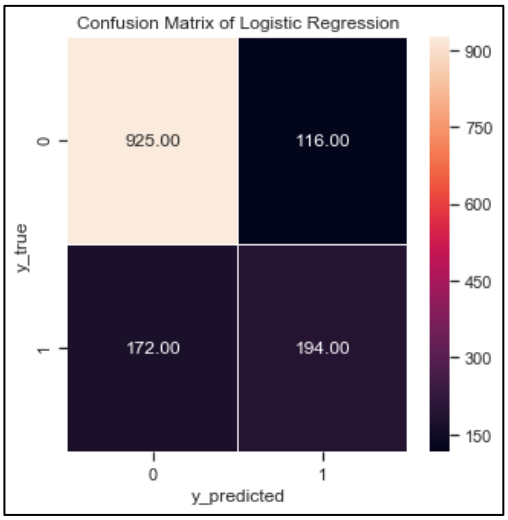

Figure 4.31

As revealed in Figure 4.30 and Figure 4.31, the generated model can be further assessed through the *confusion matrix* by analyzing misclassifications.

TN = 925, FP = 116, FN = 172, TP = 194

This means; there are total 925+116 = 1041 actual non-churn values and the algorithm predict 925 of them as non-churn and 116 of them churn. Also there are total 172 + 194 = 366 actual churn values and the algorithm predict 172 of them as non-churn and 194 of them as churn.

```
report = classification_report(y_test, lr_model.predict(x_test))
print(report)
```

```
              precision    recall  f1-score   support

           0       0.84      0.89      0.87      1041
           1       0.63      0.53      0.57       366

    accuracy                           0.80      1407
   macro avg       0.73      0.71      0.72      1407
weighted avg       0.79      0.80      0.79      1407
```

Figure 4.32

As indicated in Figure 4.32, *recall score, precision score, f1 score*, and *accuracy score* has also been used to evaluate the created model. Here, the *classification_report* function from *skleran library* has been used to obtain these metrics at once. The performance metrics predicts customers who don't churn with the *precision, recall* and *F1 score* values of 0.63, 0.53, and 0.79.

5. Results and Discussion

As stated in Harvard Business Review [2], it costs from five to twenty-five times to accomplish a new consumer than to withhold an already existing client. Therefore, preventing customer churn is an essential business purpose. This report summarizes how Logistic Regression and Python can be applied to identify unsatisfied customers beforehand, reduce churn rate, and keep the business revenue flowing.

A comprehensive description of the correlation of the features and concluded observations has mentioned in section 4.6. The generated model affords an accuracy score of 0.79 intending that 79% of the circumstances, the model has predicted a proper score for this scenario. Nevertheless, since the Churn "Yes" is almost 3 times as "No' in this dataset, this dataset class is imbalanced and that makes the accuracy measurement untenable. Since the accuracy should not be used as solely metric for imbalanced datasets, it is essential to analyze confusion matrix, recall, precision, and F1 score for assessing the model further. As per the confusion matrix, the generated model has predicted 925 out of 1041 actual non-churn customers and 194 out of 366 actual churn customers correctly. Upon completion, Logistic Regression provides 0.63 precision, 0.53 recall, and F1 score of 0.79.

Ultimately, through assessing this analysis on customer churn, customers who are most vulnerable to churn can be recognized ahead and business can improve their customer retention by taking necessary further actions. Attempting various feature engineering methods, recognizing the most significant features for the problem, and utilizing more complicated machine learning algorithms such as Neural Networks to train the model possibly assist to enhance the accuracy and will result in boosting the business strategy.

6. References

[1] BlastChar, “Telco Customer Churn,” Kaggle, 23-Feb-2018. [Online]. Available: https://www.kaggle.com/blastchar/telco-customer-churn. [Accessed: 23-Apr-2020].

[2] A. Gallo, “The Value of Keeping the Right Customers,” Harvard Business Review, 05-Nov-2014. [Online]. Available: https://hbr.org/2014/10/the-value-of-keeping-the-right-customers. [Accessed: 23-Apr-2020].

7. Appendix

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from pylab import rcParams

%matplotlib inline


churn_data = pd.read_csv("C:\\Users\\Himashi\\Desktop\\WA_Fn-UseC_-Telco-Customer-Churn.csv")

churn_data.head()


churn_data.shape


churn_data.columns.tolist()


churn_data.dtypes


churn_data.info()
```

```
churn_data['Churn'].value_counts(sort = False)


labels =churn_data['Churn'].value_counts(sort = True).index

sizes = churn_data['Churn'].value_counts(sort = True)

colors = ["blue","red"]

explode = (0.1,0)

rcParams['figure.figsize'] = 8,8

plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True,
startangle=270,)

plt.title('Percent of customer churn')

plt.show()


churn_data['TotalCharges'] = pd.to_numeric(churn_data.TotalCharges, errors='coerce')


churn_data["SeniorCitizen"] = churn_data["SeniorCitizen"].replace({1:"Yes",0:"No"})

#Re-check to verify changes

churn_data.info()


churn_data.describe()


churn_data.isnull().sum()


churn_data.dropna(inplace = True)

#Re-check for missing values

churn_data.isnull().sum()


churn_data.drop(["customerID"],axis=1,inplace = True)

#Re-check whether "customerID" column has been dropped
```

```
churn_data.head()
```

```
sns.set(style="ticks", color_codes=True)
```

```
fig, axes = plt.subplots(nrows = 4,ncols = 4,figsize = (25,15))
sns.countplot(x = "gender", data = churn_data, hue = churn_data.Churn, ax=axes[0][0])
sns.countplot(x = "SeniorCitizen", data = churn_data, hue = churn_data.Churn, ax=axes[0][1])
sns.countplot(x = "Partner", data = churn_data, hue = churn_data.Churn, ax=axes[0][2])
sns.countplot(x = "Dependents", data = churn_data, hue = churn_data.Churn, ax=axes[0][3])
sns.countplot(x = "PhoneService", data = churn_data, hue = churn_data.Churn, ax=axes[1][0])
sns.countplot(x = "MultipleLines", data = churn_data, hue = churn_data.Churn, ax=axes[1][1])
sns.countplot(x = "InternetService", data = churn_data, hue = churn_data.Churn, ax=axes[1][2])
sns.countplot(x = "OnlineSecurity", data = churn_data, hue = churn_data.Churn, ax=axes[1][3])
sns.countplot(x = "OnlineBackup", data = churn_data, hue = churn_data.Churn, ax=axes[2][0])
sns.countplot(x = "DeviceProtection", data = churn_data, hue = churn_data.Churn, ax=axes[2][1])
sns.countplot(x = "TechSupport", data = churn_data, hue = churn_data.Churn, ax=axes[2][2])
sns.countplot(x = "StreamingTV", data = churn_data, hue = churn_data.Churn, ax=axes[2][3])
sns.countplot(x = "StreamingMovies", data = churn_data, hue = churn_data.Churn, ax=axes[3][0])
sns.countplot(x = "Contract", data = churn_data, hue = churn_data.Churn, ax=axes[3][1])
sns.countplot(x = "PaperlessBilling", data = churn_data, hue = churn_data.Churn, ax=axes[3][2])
ax=sns.countplot(x = "PaymentMethod", data = churn_data, hue = churn_data.Churn, ax=axes[3][3])
ax.set_xticklabels(ax.get_xticklabels(),rotation=90)
```

```
plt.show(fig)
```

```
sns.pairplot(churn_data,vars = ['tenure','MonthlyCharges','TotalCharges'], hue="Churn")
```

```
from sklearn.preprocessing import LabelEncoder

def object_to_int(dataframe_series):
    if dataframe_series.dtype=='object':
        dataframe_series = LabelEncoder().fit_transform(dataframe_series)
    return dataframe_series

churn_data = churn_data.apply(lambda x: object_to_int(x))
churn_data.head()

plt.figure(figsize=(15,8))
churn_data.corr()['Churn'].sort_values(ascending = False).plot(kind='bar')

churn_data.corr()["Churn"].sort_values()

f,ax = plt.subplots(figsize=(25, 25))
sns.heatmap(churn_data.corr(), annot=True, linewidths=.5, fmt= '.2f',ax=ax)
plt.show()

#Assign the target
y = churn_data.Churn.values

#Drop the target, retaining only numerical features
new_churn_data = churn_data.drop(["Churn"],axis=1)

#Normalize values to fit between 0 and 1.
x = (new_churn_data-np.min(new_churn_data))/(np.max(new_churn_data)-
np.min(new_churn_data)).values
```

```
#Split dataset into Train and Test

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,random_state =1)

print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)


from sklearn.linear_model import LogisticRegression

lr_model = LogisticRegression()

lr_model.fit(x_train,y_train)

accuracy_lr = lr_model.score(x_test,y_test)

print("Logistic Regression accuracy is :",accuracy_lr)


from sklearn.metrics import recall_score, confusion_matrix, precision_score, f1_score,
accuracy_score, classification_report


cm_lr = confusion_matrix(y_test,lr_model.predict(x_test))


f, ax = plt.subplots(figsize = (5,5))

sns.heatmap(cm_lr, annot = True, linewidths = 0.5, color = "red", fmt = ".0f", ax=ax)

plt.xlabel("y_predicted")

plt.ylabel("y_true")

plt.title("Confusion Matrix of Logistic Regression")

plt.show()

report = classification_report(y_test, lr_model.predict(x_test))

print(report)
```