DeepSphere.AI
Enterprise AI and IIoT for Analytics

Applied Artificial Intelligence Learning Curriculum

Applied Machine Learning, Data Science, and Data Engineering

Hands-on Lab Guide

Customer Churn

Use Case Implementation

A Simple and easy follow-by-step Applied AI lab guide for 6th graders to professionals

Table of Contents

Disclaimer

We share this information for learning purposes only, and we developed this material based on our prior experience, skills, knowledge, and expertise. Our perspective on the tools, technologies, systems, applications, processes, methodologies, and others used in these materials may differ from others. We advise the users to use these materials at their own risk.

The sample programs used in these materials developed by us are based on some system and data assumptions, and these examples may or may not work for others. If there are any issues in following these materials, please feel free to contact our support services, and we will try to help you based on our support resource availability.

The respective vendors own all the hardware, software, tools, technologies, processes, methodologies, and others used in these materials. Users agree to these learning resources at their own risk, and under any circumstances, DeepSphere.AI is not liable for any of these vendor's products, services, and resources.

DeepSphere.AI and Google Cloud

DeepSphere.AI (DS.AI) is a global leader in providing an advanced and higher educational platform for schools. DS.AI provides an intelligent learning management system (iLMS) to learn applied artificial intelligence, data science, and data engineering at a personalized level. DS.AI iLMS platform hosted on Amazon web services (AWS) and the learning resources developed on Google Cloud Platform(GCP) and SAP Litmos.

To create social readiness and awareness about applied AI, DS.AI continued to develop learning resources to educate and empower schools, colleges, universities, organizations, and public entities. This article is part of a series of learning resources, and there will be several articles will be published to master applied AI on Google Colab. We use several GCP services to develop these learning resources, including storage services, compute services, network services, and other products and services.

Our goal is to go beyond concepts, ideas, visions, and strategies to provide practical problem-solving applied AI skills, knowledge, and expertise to gain the job learning experience. To achieve our goals and objectives, we use GCP products and services, including BigQuery, AutoML, AutoML Tables, Dataproc, Dataflow, Data Studio, etc.

1. Executive Summary

The purpose of this document is to provide adequate information to users to implement Customer Churn in Google Colab. In order to achieve this, we are using supervised machine learning models like Logistic Regression or XGBoost.

2. Problem Statement

Companies or Organizations often face huge customer attrition or churn. When customers leave the company, they not only lose the revenue from these customers but also loose the resources spent to acquire these customers at the first place. This is a serious concern for the companies.

In this Implementation we are trying to predict customers who are mostly likely to churn using machine learning modelling. This implementation helps the companies to know in advance the customers who are more likely to leave the business at some point in time. This helps the companies to come up with retention strategies and policies.

3. Business Challenges

Companies need to build and deploy an effective customer churn prediction models to succeed in today’s complex business scenarios. Acquiring new customers always costs heavily. Following are the challenges companies face when there is no customer prediction modelling in place

- No Sustainable and robust strategy for customer retention.
- No formula plan to reacquire the customers who have moved to other competitors.
- Issues in converting low revenue earning customers into highly profitable ones.
- Reducing customer defections and improving profits.
- Tracking customer satisfaction by product, segment and cost to serve.

All these business challenges are solved by the predictive churn models that aims at retaining customers and maximizing profits.

4. Model Selection

Model selection is the process of choosing between different machine learning approaches - e.g. Decision Tree, Logistic Regression, etc. - or choosing between different hyperparameters or sets of features for the same machine learning approach - e.g. deciding between the polynomial degrees/complexities for linear regression.

The choice of the actual machine learning algorithm (e.g. SVM or logistic regression) is less important than you'd think - there may be a "best" algorithm for a particular problem, but often its performance is not much better than other well-performing approaches for that problem.

There may be certain qualities you look for in a model:

- Interpretable - can we see or understand why the model is making the decisions it makes?
- Simple - easy to explain and understand
- Accurate
- Fast (to train and test)
- Scalable (it can be applied to a large dataset)

Our Problem here is an Supervised Classification Problem. The Problem is to predict customers who are more likely to churn. This Type of Problem can be Solved by the following Models.

1. Logistic Regression.

2. XGBoost.

(Check the Appendix section for more information about Supervised Classification models)

5. Feature Engineering:

Feature engineering is the process of using **domain knowledge** of the data to create features that make machine learning algorithms work. If feature engineering is done correctly, it increases the predictive power of machine learning algorithms by creating features from raw data that help facilitate the machine learning process. Feature Engineering is an art.

Feature engineering is the most important art in machine learning which creates the huge difference between a good model and a bad model.

✓ 5.1. Advantages of Feature Engineering

- Good features provide you with the flexibility of choosing an algorithm; even if you choose a less complex model, you get good accuracy.

- If you choose good features, then even simple ML algorithms do well.

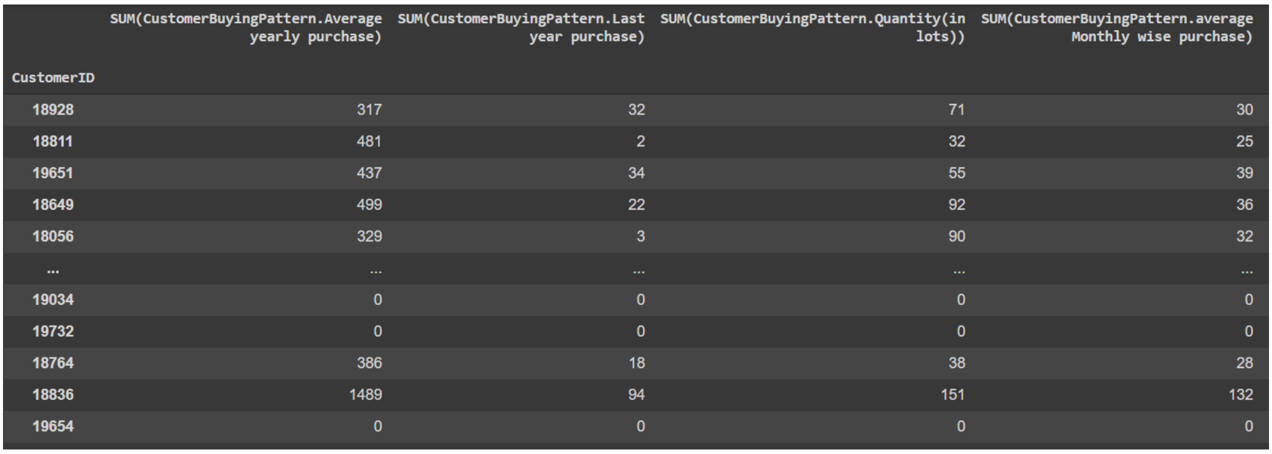
- Better features will lead you to better accuracy. You should spend more time on features engineering to generate the appropriate features for your dataset. If you derive the best and appropriate features, you have won most of the battle.

6. Data Management

There are three types of data sets Training, Test and Dev that are used at various stage of Implementation. Training dataset is the largest of three of them, while test data functions as seal of approval and you don’t need to use till the end of the development.

✓ 6.1. What is a Training Data Set?

The training data set is the actual dataset used to train the model for performing various Machine Learning Operations (Regression, Classification, Clustering etc.). This is the actual data with which the models learn with various API and algorithm to train the machine to work automatically.



	SUM(CustomerBuyingPattern.Average yearly purchase)	SUM(CustomerBuyingPattern.Last year purchase)	SUM(CustomerBuyingPattern.Quantity(in lots))	SUM(CustomerBuyingPattern.average Monthly wise purchase)
CustomerID				
18928	317	32	71	30
18811	481	2	32	25
19651	437	34	55	39
18649	499	22	92	36
18056	329	3	90	32
...	...	...	...	...
19034	0	0	0	0
19732	0	0	0	0
18764	386	18	38	28
18836	1489	94	151	132
19654	0	0	0	0

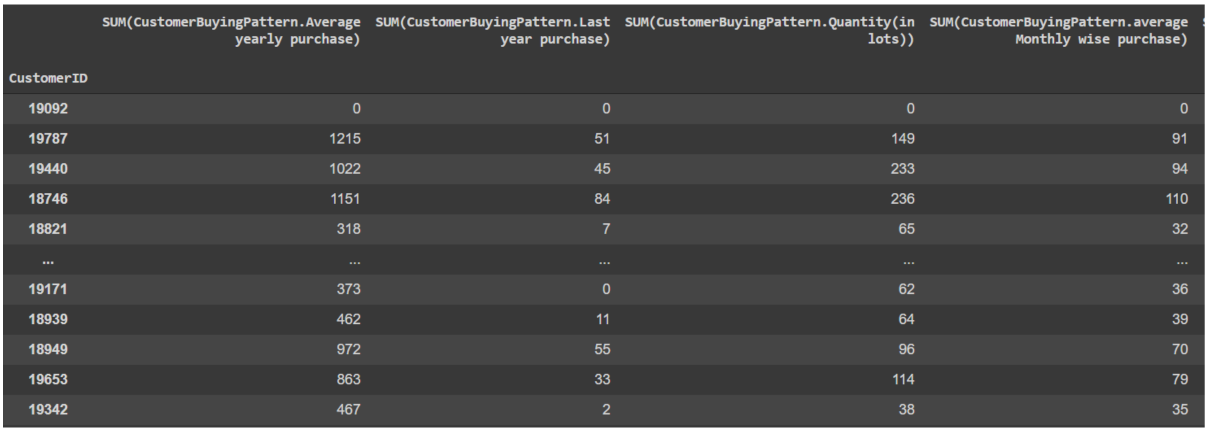
Figure 1 - Training Data

The following section describes the data training data sets and its field level characteristics

- Customer
- Average Yearly Purchase
- Last Year Purchase
- Quantity
- Customer Lifetime in Years
- Price Amount
- Last Year Unit Price
- Product Average Unit Price
- Amount Spent in Lifetime
- Service Call
- Service Failure Rate

✓ What is a Test Data Set?

Test data set helps you to validate that the training has happened efficiently in terms of either accuracy, or precision so on. Actually, such data is used for testing the model whether it is responding or working appropriately or not.



	SUM(CustomerBuyingPattern.Average yearly purchase)	SUM(CustomerBuyingPattern.Last year purchase)	SUM(CustomerBuyingPattern.Quantity(in lots))	SUM(CustomerBuyingPattern.average Monthly wise purchase)
CustomerID				
19092	0	0	0	0
19787	1215	51	149	91
19440	1022	45	233	94
18746	1151	84	236	110
18821	318	7	65	32
...	...	...	...	...
19171	373	0	62	36
18939	462	11	64	39
18949	972	55	96	70
19653	863	33	114	79
19342	467	2	38	35

Figure 2 - Test Data

The following section describes the features that’s used in the model.

- Customer
- Average Yearly Purchase
- Last Year Purchase
- Quantity
- Customer Lifetime in Years
- Price Amount
- Last Year Unit Price
- Product Average Unit Price
- Amount Spent in Lifetime
- Service Call
- Service Failure Rate

7. Learning Algorithm

- A Self Learning (not a human developed code) code, performs data analysis and extracts patterns (business characteristics) in data for business application development - A Modern approach to application/software development.
- Automatically understands and extracts data pattern when data changes (change in business circumstance) and performs data analysis based on the new/changed data set. No code change required to implement changes that took place in the data (change in business)

✓ 7.1. Machine Learning Libraries Used

- **Sklearn 0.19.0 (Scikit Learn)**
- **Pandas 0.20.3**

✓ 7.2. Classification Models Used

- **Logistic Regression**
- **XGBoost**

8. Model Building Blocks

There are several technical and functional components involved in implementing this model. Here are the key building blocks to implement the model.

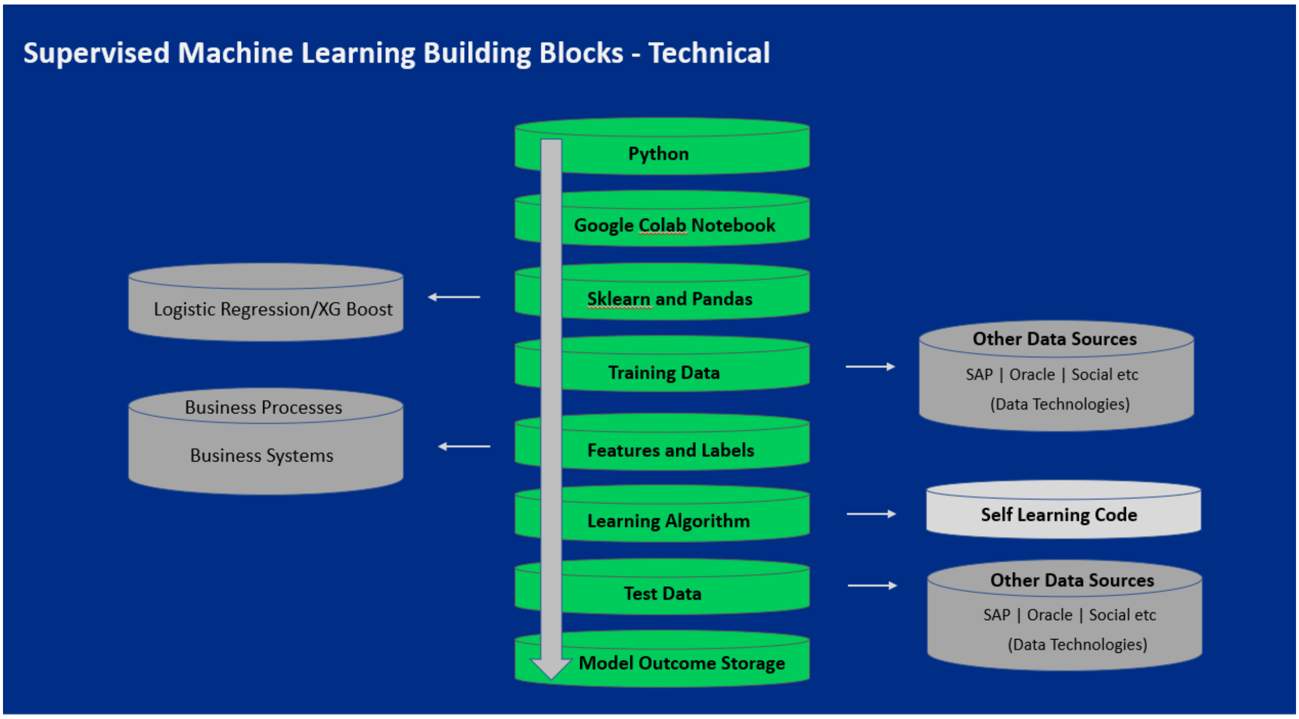
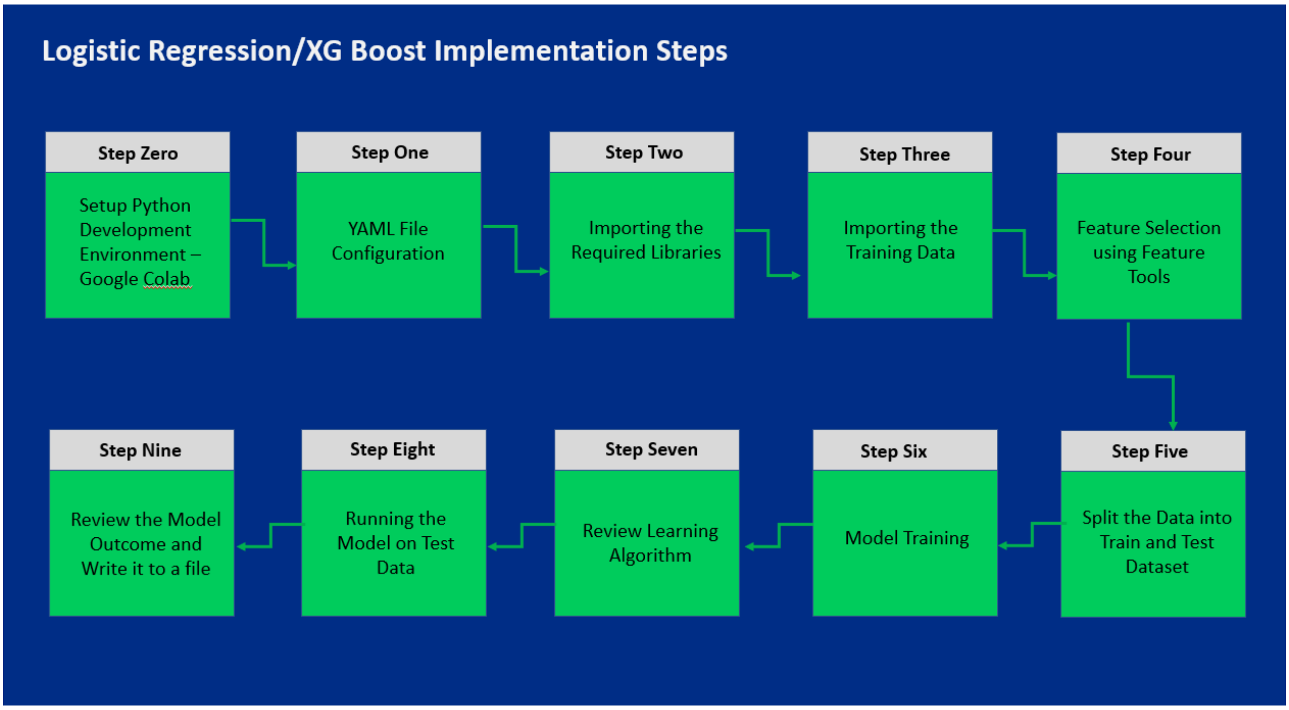


Figure 3 – Supervised Learning Building Blocks

9. Model Implementation High-level Steps

A model implementation, to address a given problem involves several steps. Here are the key steps that are involved to implement a model. You can customize these steps as needed and we developed these steps for learning purpose only.



Logistic Regression/XG Boost Implementation Steps

Step Zero: Setup Python Development Environment – Google Colab

Step One: YAML File Configuration

Step Two: Importing the Required Libraries

Step Three: Importing the Training Data

Step Four: Feature Selection using Feature Tools

Step Five: Split the Data into Train and Test Dataset

Step Six: Model Training

Step Seven: Review Learning Algorithm

Step Eight: Running the Model on Test Data

Step Nine: Review the Model Outcome and Write it to a file

Figure 4 – Model Building Implementation Steps

10. Model Building Code Block

```
# Step 1- YAML File Configuration

import configparser

import os

vAR_Config = configparser.ConfigParser(allow_no_value=True)

vAR_INI_FILE_PATH = '/content/drive/MyDrive/CUS_CHURN_FT_XG.yaml'

vAR_INI_FILE_PATH

vAR_Config.read(vAR_INI_FILE_PATH)

vAR_Data = vAR_Config.sections()

vAR_Config.sections()

vAR_Train_Data = vAR_Config['FILE PATH']['TRAINING_DATA']

vAR_Training_Data_Excel_Worsheet = vAR_Config['FILE PATH']['TRAINING_DATA_EXCEL_WORKSHEET']
print(vAR_Training_Data_Excel_Worsheet)

vAR_Train_Data_CUS = vAR_Config['FILE PATH']['TRAINING_DATA(CUS)']

vAR_Training_Data_Excel_Worsheet_CUS  = vAR_Config['FILE
PATH']['TRAINING_DATA_EXCEL_WORKSHEET(CUS)']

print(vAR_Training_Data_Excel_Worsheet_CUS)

vAR_Training_Data_CBP = vAR_Config['FILE PATH']['TRAINING_DATA(CBP)']

vAR_Training_Data_Excel_Worsheet_CBP = vAR_Config['FILE
PATH']['TRAINING_DATA_EXCEL_WORKSHEET(CBP)']

print(vAR_Training_Data_Excel_Worsheet_CBP)

vAR_Training_Data_CPP = vAR_Config['FILE PATH']['TRAINING_DATA(CPP)']

vAR_Training_Data_Excel_Worsheet_CPP = vAR_Config['FILE
PATH']['TRAINING_DATA_EXCEL_WORKSHEET(CPP)']

print(vAR_Training_Data_Excel_Worsheet_CPP)

vAR_Training_Data_CSP = vAR_Config['FILE PATH']['TRAINING_DATA(CSP)']
```

```
vAR_Training_Data_Excel_Worsheet_CSP = vAR_Config['FILE PATH']['TRAINING_DATA_EXCEL_WORKSHEET(CSP)']

print(vAR_Training_Data_Excel_Worsheet_CSP)

vAR_Training_Data_CSQ = vAR_Config['FILE PATH']['TRAINING_DATA(CSQ)']

vAR_Training_Data_Excel_Worsheet_CSQ = vAR_Config['FILE PATH']['TRAINING_DATA_EXCEL_WORKSHEET(CSQ)']

print(vAR_Training_Data_Excel_Worsheet_CSQ)

vAR_Training_Data_CS = vAR_Config['FILE PATH']['TRAINING_DATA(CS)']

vAR_Training_Data_Excel_Worsheet_CS = vAR_Config['FILE PATH']['TRAINING_DATA_EXCEL_WORKSHEET(CS)']

print(vAR_Training_Data_Excel_Worsheet_CS)
```

```
# Step 2 – Import the Required Libraries

import pandas as vAR_pd

import xgboost as vAR_xgb

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from xgboost import XGBClassifier

!pip install featuretools

import featuretools as ft
```

```
# Step 3 – Import the Training Data

import warnings

warnings.filterwarnings('ignore')
```

```
Customer_Label = vAR_pd.read_excel(vAR_Train_Data)

Customer = vAR_pd.read_excel(vAR_Train_Data_CUS)

Customer_Buying_Pattern = vAR_pd.read_excel(vAR_Training_Data_CBP)

Customer_Product_Price = vAR_pd.read_excel(vAR_Training_Data_CPP)

Customer_Spending_Power = vAR_pd.read_excel(vAR_Training_Data_CSP)

Customer_Service_Quality = vAR_pd.read_excel(vAR_Training_Data_CSQ)

Customer_Satisfaction = vAR_pd.read_excel(vAR_Training_Data_CS)
```

Check for Missing Data:

```
#Checking for null values

print(Customer_Label.isnull().sum())

print(Customer.isnull().sum())

print(Customer_Buying_Pattern.isnull().sum())

print(Customer_Product_Price.isnull().sum())

print(Customer_Spending_Power.isnull().sum())

print(Customer_Service_Quality.isnull().sum())

print(Customer_Satisfaction.isnull().sum())
```

Handling Missing Values

```
#Removing null values
Customer_Label.dropna(inplace=True)

#Re-Checking for null values
print(Customer_Label.isnull().sum())
```

____

```
# Step 4 – Feature Selection using Feature tools

#DEFINING THE ENTITIES

es = ft.EntitySet(id="CUSTOMER_CHURN")

es1 = es.entity_from_dataframe(entity_id = 'Customer', dataframe = Customer, index='CustomerID')

es2 = es.entity_from_dataframe(entity_id = 'CustomerBuyingPattern', dataframe = Customer_Buying_Pattern, index='CBPID')

es3 = es.entity_from_dataframe(entity_id = 'CustomerProductPurchase', dataframe = Customer_Product_Price, index='CPPID')

es4 = es.entity_from_dataframe(entity_id = 'CustomerSpendingPower', dataframe = Customer_Spending_Power, index = 'CSPID')

es5 = es.entity_from_dataframe(entity_id = 'CustomerServiceQuality', dataframe = Customer_Service_Quality, index = 'CSQID')

es6 = es.entity_from_dataframe(entity_id = 'CustomerSatisfaction', dataframe = Customer_Satisfaction, index = 'CSID')

es

#DEFINING THE RELATIONSHIPS

es.add_relationship(ft.Relationship(es['Customer']['CustomerID'],es['CustomerBuyingPattern']['CustomerID']))

es.add_relationship(ft.Relationship(es['Customer']['CustomerID'],es['CustomerProductPurchase']['CustomerID']))

es.add_relationship(ft.Relationship(es['Customer']['CustomerID'],es['CustomerSpendingPower']['CustomerID']))

es.add_relationship(ft.Relationship(es['Customer']['CustomerID'],es['CustomerServiceQuality']['CustomerID']))

es.add_relationship(ft.Relationship(es['Customer']['CustomerID'],es['CustomerSatisfaction']['CustomerID']))


#APPLYING DEEP SYNTHESIS

feature_matrix_Customer, feature_defs = ft.dfs(entityset=es, target_entity="Customer", agg_primitives=["SUM"], max_depth=2)

feature_matrix_Customer

print(feature_matrix_Customer.shape)
```

```
#Removing Unwanted columns

vAR_Featuresft = feature_matrix_Customer.iloc[:,4:]
print(vAR_Featuresft.shape)


#Defining the label

vAR_label = Customer_Label.iloc[:,4:]
vAR_label
```

# Step 5 – Split the Data into train and Test

```
vAR_X_TRAIN, vAR_X_TEST, vAR_Y_TRAIN, vAR_Y_TEST = train_test_split(vAR_Featuresft, vAR_label, test_size=0.20,
random_state=0)

vAR_X_TRAIN
```

____

# Step 6 – Training the model

```
#Training the logistic regression model

vAR_Model1 = LogisticRegression()

vAR_Model1.fit(vAR_X_TRAIN,vAR_Y_TRAIN)


#Training the XGBoost model

vAR_Model2 = XGBClassifier(eta=0.01,gamma=10)

vAR_Model2.fit(vAR_X_TRAIN,vAR_Y_TRAIN)
```

# Step 7 – Review the Learning Algorithm

```
vAR_Model2.predict(vAR_X_TRAIN)
```

____

# Step 8 – Running the model on test data

```
#Prediction using Logistic regression

vAR_Labels_predLG = vAR_Model1.predict(vAR_X_TEST)


#Prediction using XGBoost

vAR_Labels_predXG = vAR_Model2.predict(vAR_X_TEST)


# Checking accuracy for Logistic Regression

from sklearn.metrics import accuracy_score
accuracy_score(vAR_Y_TEST, vAR_Labels_predLG)


# Checking accuracy for XGBoost

from sklearn.metrics import accuracy_score
accuracy_score(vAR_Y_TEST, vAR_Labels_predXG)
```

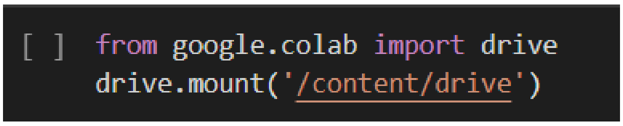
```
# Step 9 – Writing the model Outcome to a file
```

11. Model Building Steps

All the data is stored in the Google Drive, hence we need to first mount Google Drive to our Colab environment.

✓ 11.1. Mount Google Drive in Colab

To mount your google drive, run the below code, follow the link to login and enter the authorization code that you receive upon successful login. The google drive will be mounted in Colab and you'll be able to access the files.

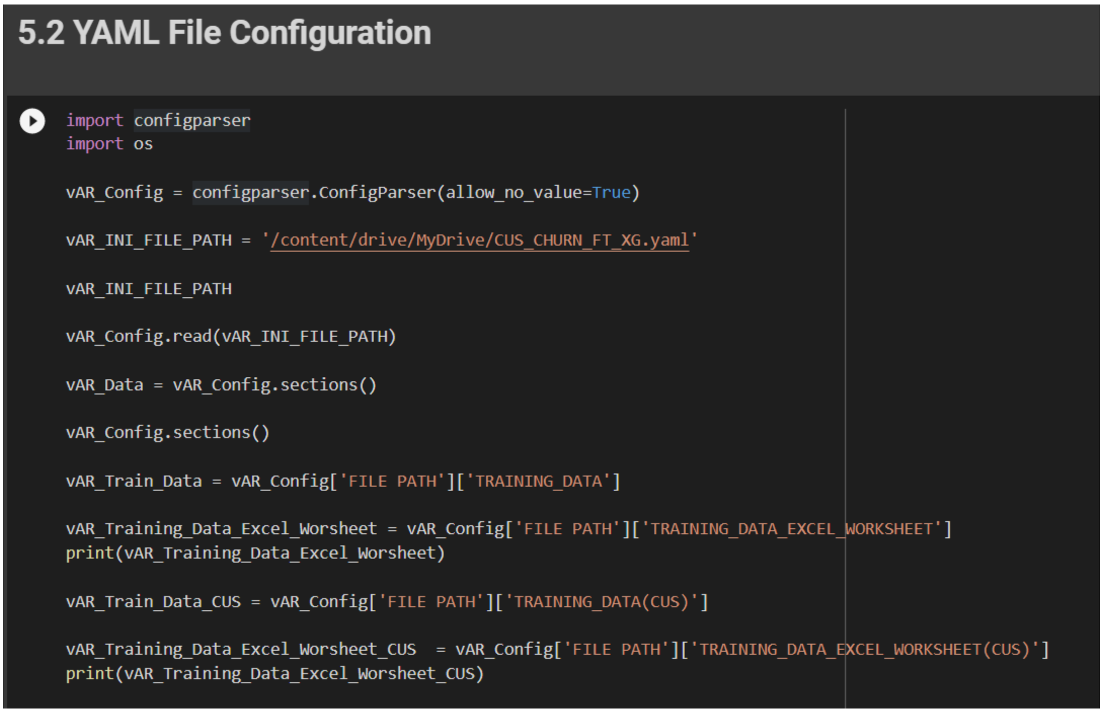


```
[ ]  from google.colab import drive
     drive.mount('/content/drive')
```

Figure 5 - Mounting Google Drive to Colab

✓ 11.2. YAML file Configuration

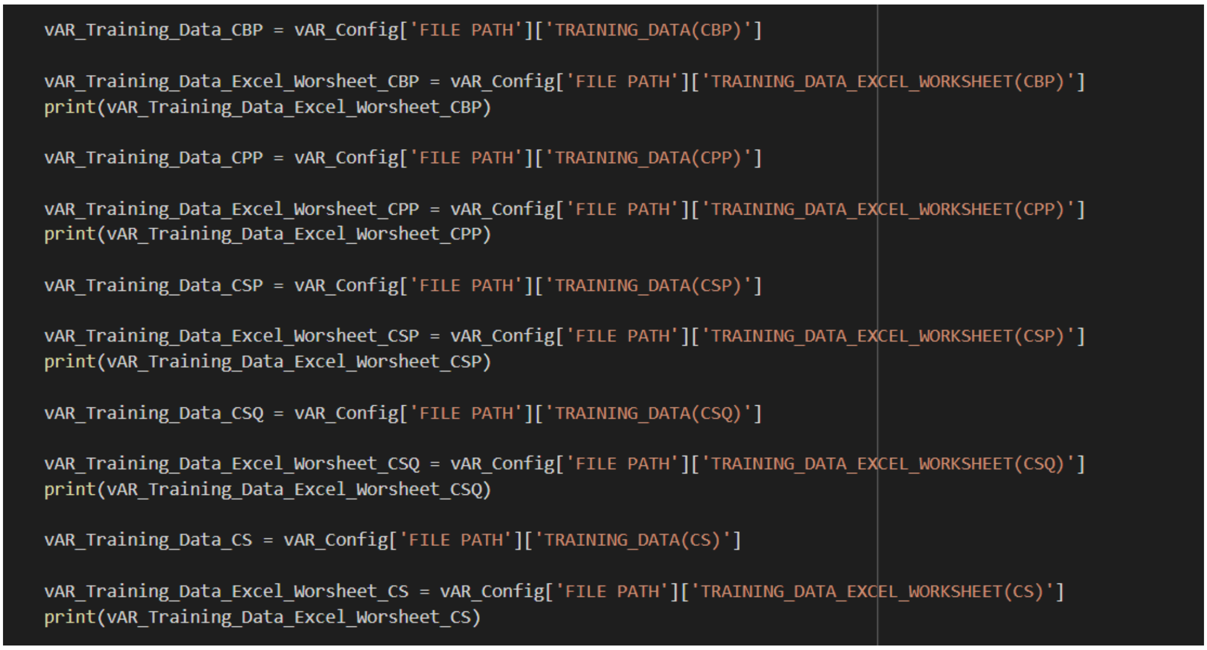
YAML file is a configuration file. This file has all the file paths for required for the model implementation. The file paths then will not be hard-coded into the implementation instead read from the YAML file dynamically.



5.2 YAML File Configuration

```
import configparser
import os

vAR_Config = configparser.ConfigParser(allow_no_value=True)

vAR_INI_FILE_PATH = '/content/drive/MyDrive/CUS_CHURN_FT_XG.yaml'

vAR_INI_FILE_PATH

vAR_Config.read(vAR_INI_FILE_PATH)

vAR_Data = vAR_Config.sections()

vAR_Config.sections()

vAR_Train_Data = vAR_Config['FILE PATH']['TRAINING_DATA']

vAR_Training_Data_Excel_Worsheet = vAR_Config['FILE PATH']['TRAINING_DATA_EXCEL_WORKSHEET']
print(vAR_Training_Data_Excel_Worsheet)

vAR_Train_Data_CUS = vAR_Config['FILE PATH']['TRAINING_DATA(CUS)']

vAR_Training_Data_Excel_Worsheet_CUS  = vAR_Config['FILE PATH']['TRAINING_DATA_EXCEL_WORKSHEET(CUS)']
print(vAR_Training_Data_Excel_Worsheet_CUS)
```

Figure 6 – YAML File Configuration



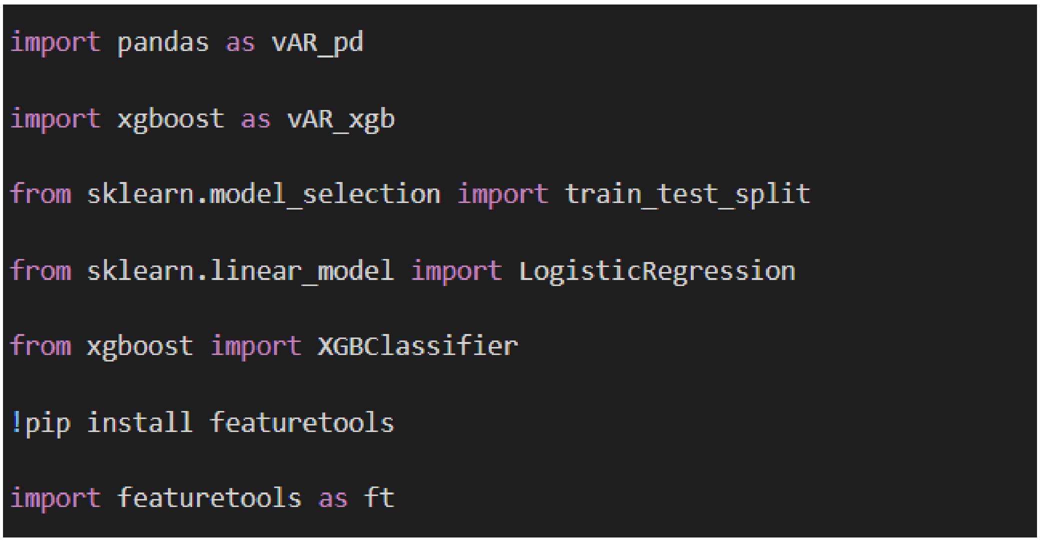
```
vAR_Training_Data_CBP = vAR_Config['FILE PATH']['TRAINING_DATA(CBP)']

vAR_Training_Data_Excel_Worsheet_CBP = vAR_Config['FILE PATH']['TRAINING_DATA_EXCEL_WORKSHEET(CBP)']
print(vAR_Training_Data_Excel_Worsheet_CBP)

vAR_Training_Data_CPP = vAR_Config['FILE PATH']['TRAINING_DATA(CPP)']

vAR_Training_Data_Excel_Worsheet_CPP = vAR_Config['FILE PATH']['TRAINING_DATA_EXCEL_WORKSHEET(CPP)']
print(vAR_Training_Data_Excel_Worsheet_CPP)

vAR_Training_Data_CSP = vAR_Config['FILE PATH']['TRAINING_DATA(CSP)']

vAR_Training_Data_Excel_Worsheet_CSP = vAR_Config['FILE PATH']['TRAINING_DATA_EXCEL_WORKSHEET(CSP)']
print(vAR_Training_Data_Excel_Worsheet_CSP)

vAR_Training_Data_CSQ = vAR_Config['FILE PATH']['TRAINING_DATA(CSQ)']

vAR_Training_Data_Excel_Worsheet_CSQ = vAR_Config['FILE PATH']['TRAINING_DATA_EXCEL_WORKSHEET(CSQ)']
print(vAR_Training_Data_Excel_Worsheet_CSQ)

vAR_Training_Data_CS = vAR_Config['FILE PATH']['TRAINING_DATA(CS)']

vAR_Training_Data_Excel_Worsheet_CS = vAR_Config['FILE PATH']['TRAINING_DATA_EXCEL_WORKSHEET(CS)']
print(vAR_Training_Data_Excel_Worsheet_CS)
```

Figure 6 - YAML File Configuration

✓ 11.3. Importing Libraries

Python libraries are specific functions containing pre-written code that can be imported into your code base by using Python's import feature. This increases your code reusability.

There are various Python libraries for Data Engineering and Machine Learning that enable the development of efficient programs to implement models. Below are the libraries which we are importing to implement our model: (Check Appendix for more information about these libraries)

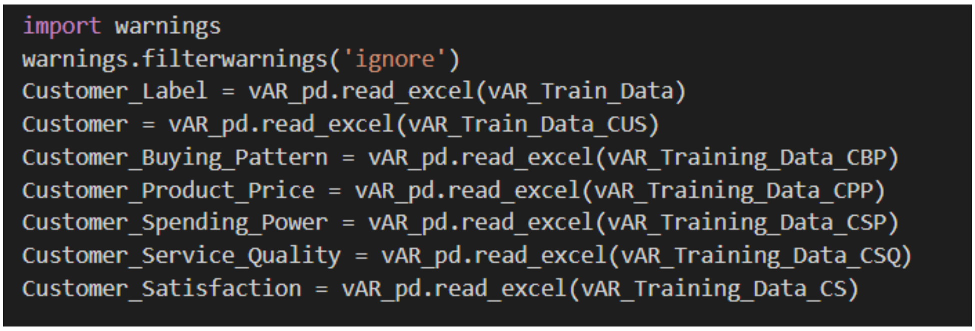


```
import pandas as vAR_pd

import xgboost as vAR_xgb

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from xgboost import XGBClassifier

!pip install featuretools

import featuretools as ft
```

Figure 7 – Importing the requiring libraries

✓ 11.4. Importing Training Data

Next immediate step after importing all libraries is getting the Training data imported. We are importing the Clustering data stored in our local system with the use of Pandas library.



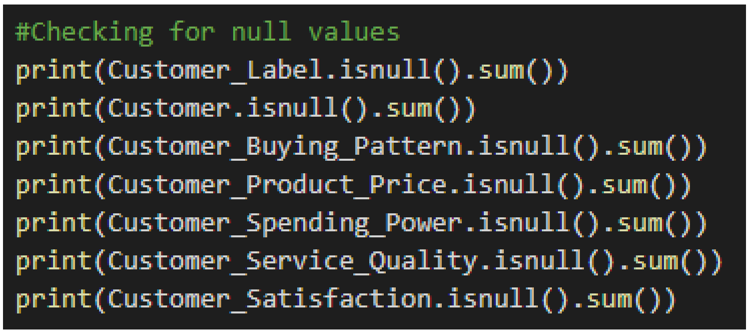
```
import warnings
warnings.filterwarnings('ignore')
Customer_Label = vAR_pd.read_excel(vAR_Train_Data)
Customer = vAR_pd.read_excel(vAR_Train_Data_CUS)
Customer_Buying_Pattern = vAR_pd.read_excel(vAR_Training_Data_CBP)
Customer_Product_Price = vAR_pd.read_excel(vAR_Training_Data_CPP)
Customer_Spending_Power = vAR_pd.read_excel(vAR_Training_Data_CSP)
Customer_Service_Quality = vAR_pd.read_excel(vAR_Training_Data_CSQ)
Customer_Satisfaction = vAR_pd.read_excel(vAR_Training_Data_CS)
```

Figure 8 – Importing the Training Data

○ 11.4.1. Check for Missing Data

The cause of missing or null values can be data corruption or failure to record data. The handling of missing data is very important during the pre-processing of the dataset as many machine learning algorithms do not support missing values.

To find missing values:

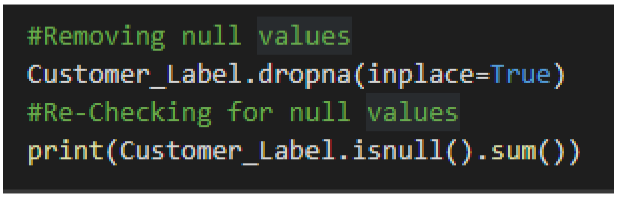


```
#Checking for null values
print(Customer_Label.isnull().sum())
print(Customer.isnull().sum())
print(Customer_Buying_Pattern.isnull().sum())
print(Customer_Product_Price.isnull().sum())
print(Customer_Spending_Power.isnull().sum())
print(Customer_Service_Quality.isnull().sum())
print(Customer_Satisfaction.isnull().sum())
```

○ 11.4.2. Handle Missing Values

As we see above, there is a null/missing value in ‘CustomerChurn’ (label) column in ‘Customer_Label’ dataset. There are many ways to handle missing value in a dataset. Columns in the dataset which are having numeric continuous values can be replaced with the mean, median, or mode of remaining values in the column. However, the missing value is in the label column so we remove the data with missing value.

To remove data with missing value:



```
#Removing null values
Customer_Label.dropna(inplace=True)
#Re-Checking for null values
print(Customer_Label.isnull().sum())
```

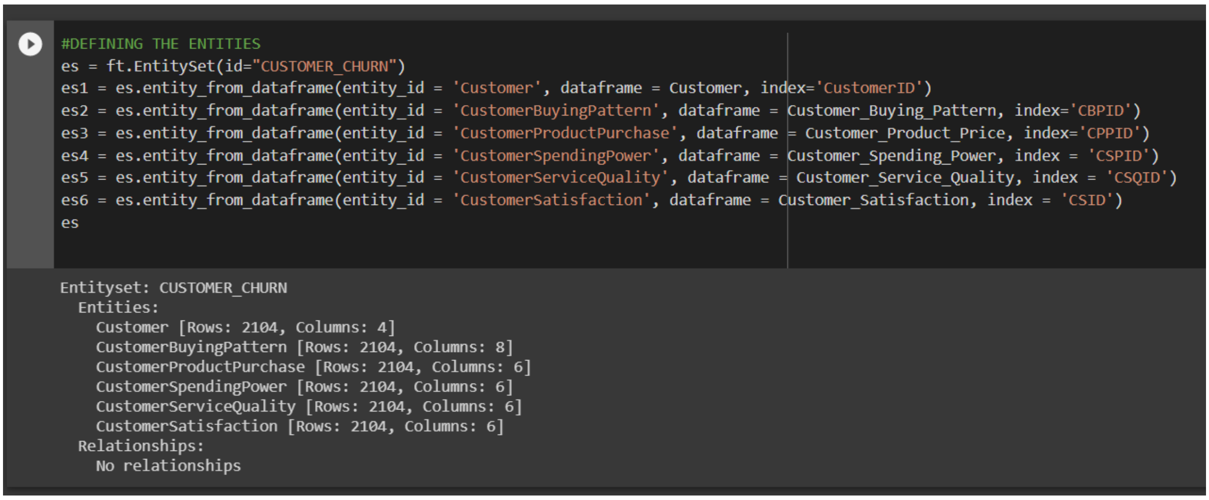
✓ 11.5. Feature Selection (Using Feature Tools)

Once we have the training data imported the next step is feature selection. We are selecting the features using the automated feature selection technique – “Feature Tools”

○ 11.5.1. Creating Entity set

- We will have to create an Entity Set. An entity is simply a table, which is represented in Pandas as a dataframe.

- An Entity Set is a structure that contains multiple dataframes and relationships between them. So, let’s create an Entity Set and add the dataframe combination to it.

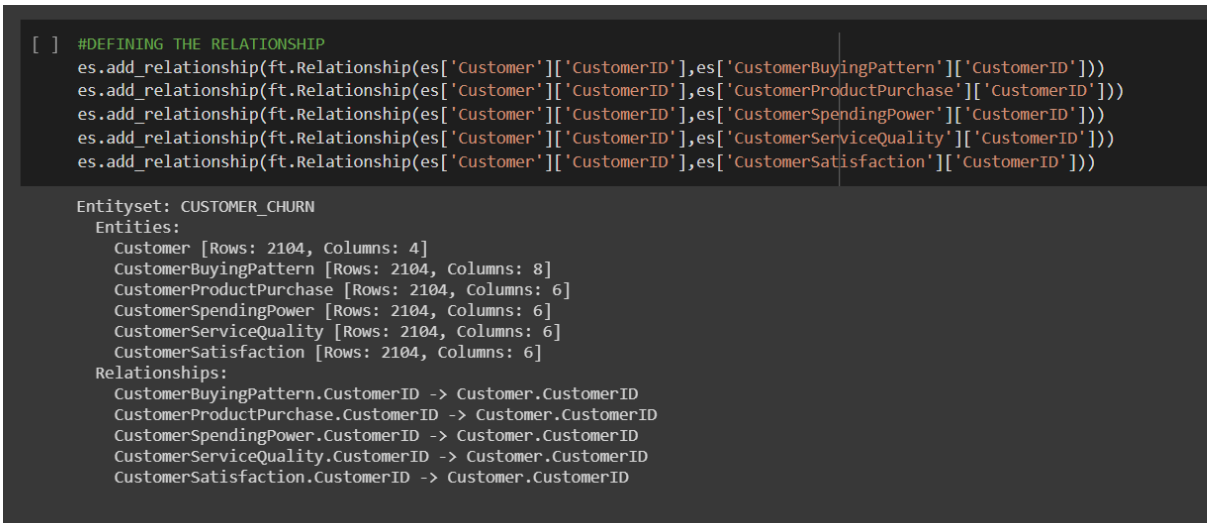


```
#DEFINING THE ENTITIES
es = ft.EntitySet(id="CUSTOMER_CHURN")
es1 = es.entity_from_dataframe(entity_id = 'Customer', dataframe = Customer, index='CustomerID')
es2 = es.entity_from_dataframe(entity_id = 'CustomerBuyingPattern', dataframe = Customer_Buying_Pattern, index='CBPID')
es3 = es.entity_from_dataframe(entity_id = 'CustomerProductPurchase', dataframe = Customer_Product_Price, index='CPPID')
es4 = es.entity_from_dataframe(entity_id = 'CustomerSpendingPower', dataframe = Customer_Spending_Power, index = 'CSPID')
es5 = es.entity_from_dataframe(entity_id = 'CustomerServiceQuality', dataframe = Customer_Service_Quality, index = 'CSQID')
es6 = es.entity_from_dataframe(entity_id = 'CustomerSatisfaction', dataframe = Customer_Satisfaction, index = 'CSID')
es
```

```
Entityset: CUSTOMER_CHURN
  Entities:
    Customer [Rows: 2104, Columns: 4]
    CustomerBuyingPattern [Rows: 2104, Columns: 8]
    CustomerProductPurchase [Rows: 2104, Columns: 6]
    CustomerSpendingPower [Rows: 2104, Columns: 6]
    CustomerServiceQuality [Rows: 2104, Columns: 6]
    CustomerSatisfaction [Rows: 2104, Columns: 6]
  Relationships:
    No relationships
```

Figure 9 – Creating an Entity Set

○ 11.5.2. Defining Relationships



```
[ ]  #DEFINING THE RELATIONSHIP
     es.add_relationship(ft.Relationship(es['Customer']['CustomerID'],es['CustomerBuyingPattern']['CustomerID']))
     es.add_relationship(ft.Relationship(es['Customer']['CustomerID'],es['CustomerProductPurchase']['CustomerID']))
     es.add_relationship(ft.Relationship(es['Customer']['CustomerID'],es['CustomerSpendingPower']['CustomerID']))
     es.add_relationship(ft.Relationship(es['Customer']['CustomerID'],es['CustomerServiceQuality']['CustomerID']))
     es.add_relationship(ft.Relationship(es['Customer']['CustomerID'],es['CustomerSatisfaction']['CustomerID']))
```

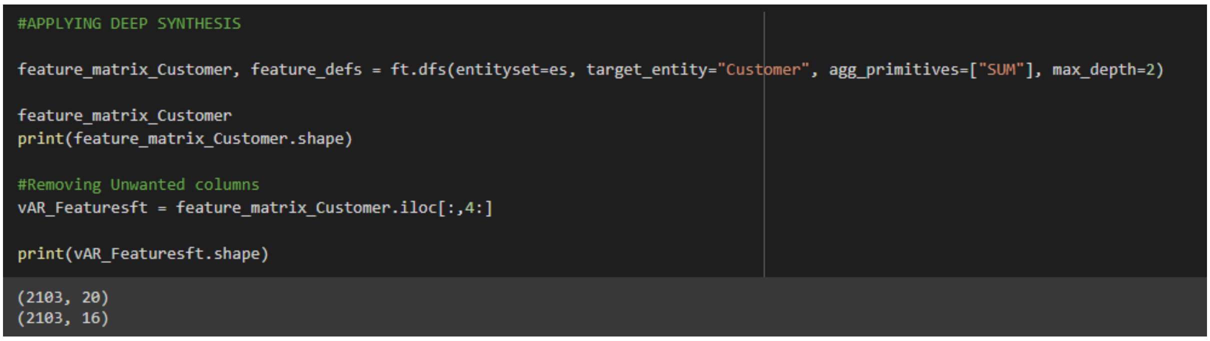
```
Entityset: CUSTOMER_CHURN
  Entities:
    Customer [Rows: 2104, Columns: 4]
    CustomerBuyingPattern [Rows: 2104, Columns: 8]
    CustomerProductPurchase [Rows: 2104, Columns: 6]
    CustomerSpendingPower [Rows: 2104, Columns: 6]
    CustomerServiceQuality [Rows: 2104, Columns: 6]
    CustomerSatisfaction [Rows: 2104, Columns: 6]
  Relationships:
    CustomerBuyingPattern.CustomerID -> Customer.CustomerID
    CustomerProductPurchase.CustomerID -> Customer.CustomerID
    CustomerSpendingPower.CustomerID -> Customer.CustomerID
    CustomerServiceQuality.CustomerID -> Customer.CustomerID
    CustomerSatisfaction.CustomerID -> Customer.CustomerID
```

Figure 10 – Defining Relationships

○ 11.5.3. Applying Deep Feature Synthesis

- Deep Feature Synthesis (DFS) is the backbone of Featuretools. It enables the creation of new features from single, as well as multiple dataframes. DFS create features by applying Feature primitives to the Entity-relationships in an EntitySet.

- A feature primitive at a very high-level is an operation applied to data to create a feature. These represent very simple calculations (sum, mean, min, max, or standard deviation) that can be stacked on top of each other to create complex features. Applying DFS with 'SUM' as a primitive.



```
#APPLYING DEEP SYNTHESIS

feature_matrix_Customer, feature_defs = ft.dfs(entityset=es, target_entity="Customer", agg_primitives=["SUM"], max_depth=2)

feature_matrix_Customer
print(feature_matrix_Customer.shape)

#Removing Unwanted columns
vAR_Featuresft = feature_matrix_Customer.iloc[:,4:]

print(vAR_Featuresft.shape)
```

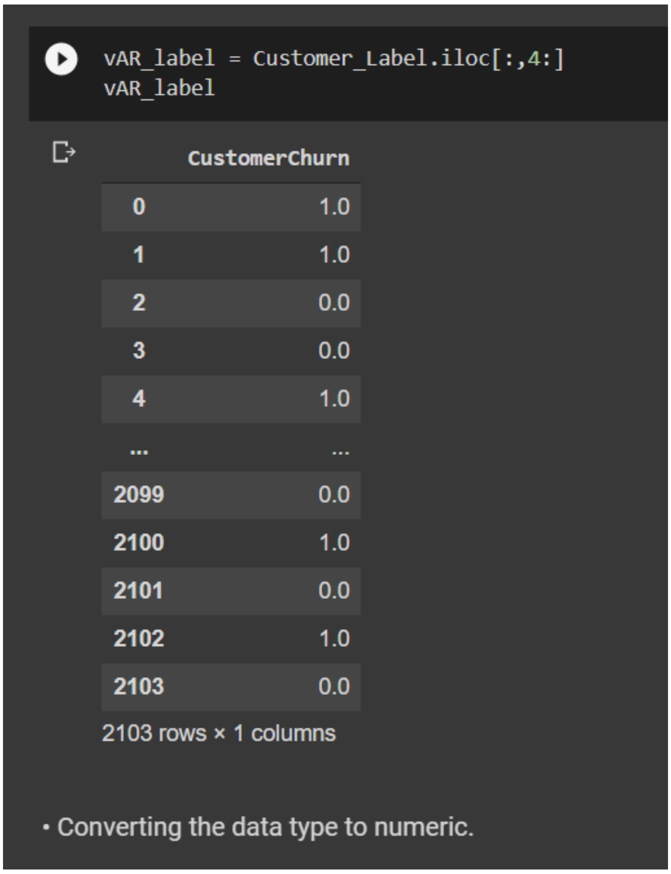
```
(2103, 20)
(2103, 16)
```

Figure 11 – Applying Deep Feature Synthesis

✓ 11.6. Defining the Labels

○ Label is the discrete attribute whose value you want to predict based on the values of other attributes.

- The term class label is usually used in the context of supervised machine learning, and in classification in particular, where one is given a set of examples of the form (attribute values, class Label) and the goal is to learn a rule that computes the label from the attribute values. The class label always takes on a finite (as opposed to infinite) number of different values.



```
vAR_label = Customer_Label.iloc[:,4:]
vAR_label
```

```
      CustomerChurn
   0            1.0
   1            1.0
   2            0.0
   3            0.0
   4            1.0
 ...            ...
2099            0.0
2100            1.0
2101            0.0
2102            1.0
2103            0.0
2103 rows × 1 columns
```

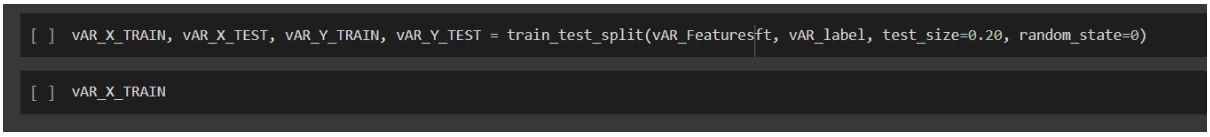
• Converting the data type to numeric.

Figure 12 – Defining the Labels

✓ 11.7. Model Training

○ 11.7.1. Splitting the data into train and test

- The train-test split procedure is used to estimate the performance of machine learning algorithms when they are used to make predictions on data not used to train the model. Training data size – 80%, Test data size – 20%



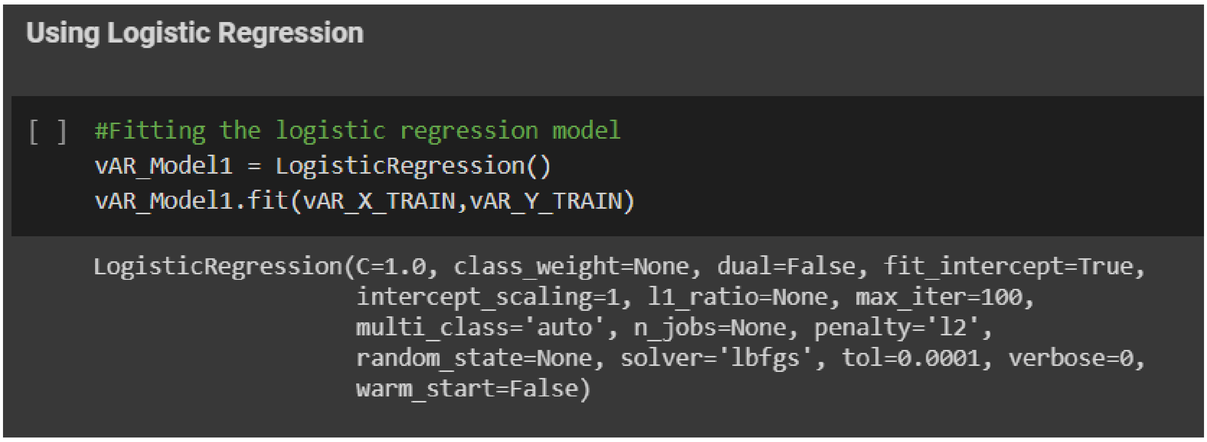
```
[ ]  vAR_X_TRAIN, vAR_X_TEST, vAR_Y_TRAIN, vAR_Y_TEST = train_test_split(vAR_Featuresft, vAR_label, test_size=0.20, random_state=0)

[ ]  vAR_X_TRAIN
```

Figure 12 – Splitting the Dataset into Train and Test

○ 11.7.2. Model fitting

Model fitting is a measure of how well a machine learning model generalizes to similar data to that on which it was trained. A model that is well-fitted produces more accurate outcomes.

▪ 11.7.2.1. Using Logistic Regression

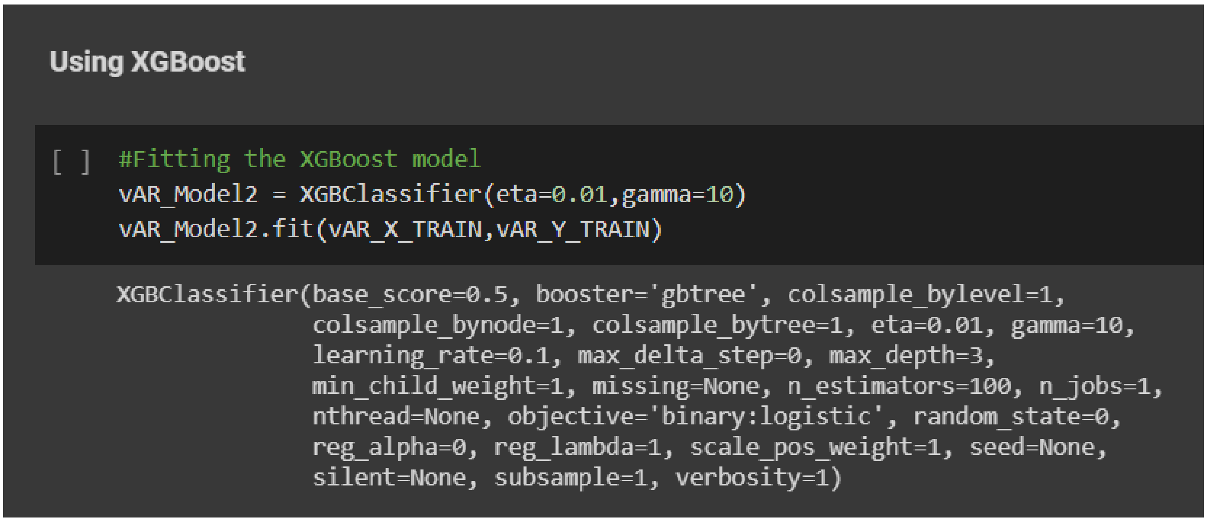


Using Logistic Regression

```
[ ]  #Fitting the logistic regression model
     vAR_Model1 = LogisticRegression()
     vAR_Model1.fit(vAR_X_TRAIN,vAR_Y_TRAIN)

     LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                        intercept_scaling=1, l1_ratio=None, max_iter=100,
                        multi_class='auto', n_jobs=None, penalty='l2',
                        random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                        warm_start=False)
```

Figure 13 – Model Training using Logistic Regression

- 11.7.2.1. Using XGBoost



Using XGBoost

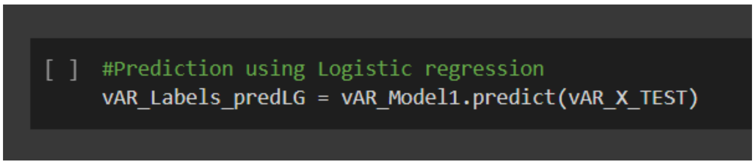
```
[ ]  #Fitting the XGBoost model
     vAR_Model2 = XGBClassifier(eta=0.01,gamma=10)
     vAR_Model2.fit(vAR_X_TRAIN,vAR_Y_TRAIN)

     XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                   colsample_bynode=1, colsample_bytree=1, eta=0.01, gamma=10,
                   learning_rate=0.1, max_delta_step=0, max_depth=3,
                   min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
                   nthread=None, objective='binary:logistic', random_state=0,
                   reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                   silent=None, subsample=1, verbosity=1)
```

Figure 14 – Model Training using XGBoost

✓ 11.8. Model Prediction

Model Prediction is running the prediction on the test data based on model training.

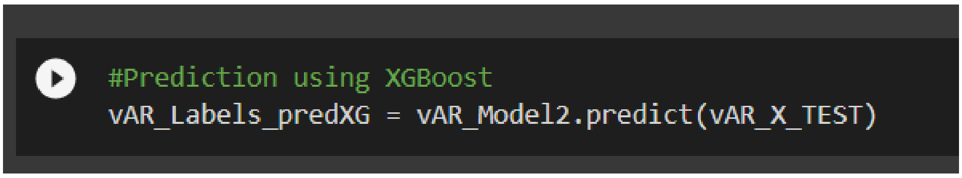
○ 11.8.1. Using Logistic Regression



```
[ ] #Prediction using Logistic regression
    vAR_Labels_predLG = vAR_Model1.predict(vAR_X_TEST)
```

Figure 14 – Model Prediction using Logistic Regression

○ 11.8.2. Using XGBoost



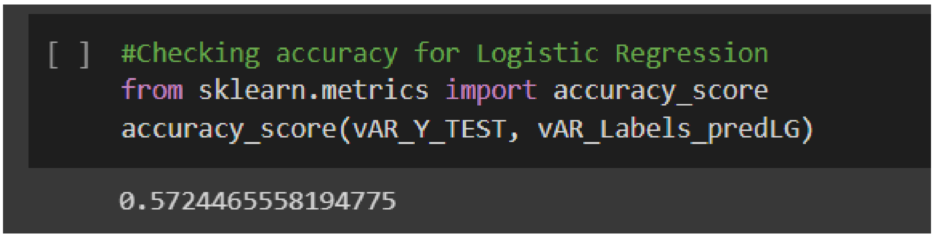
```
#Prediction using XGBoost
vAR_Labels_predXG = vAR_Model2.predict(vAR_X_TEST)
```

Figure 15 – Model Prediction using XG Boost

✓ 11.9. Model Evaluation

Model evaluation aims to estimate the generalization accuracy of a model on future (unseen/out-of-sample) data. Accuracy is defined as the percentage of correct predictions for the test data. It can be calculated easily by dividing the number of correct predictions by the number of total predictions.

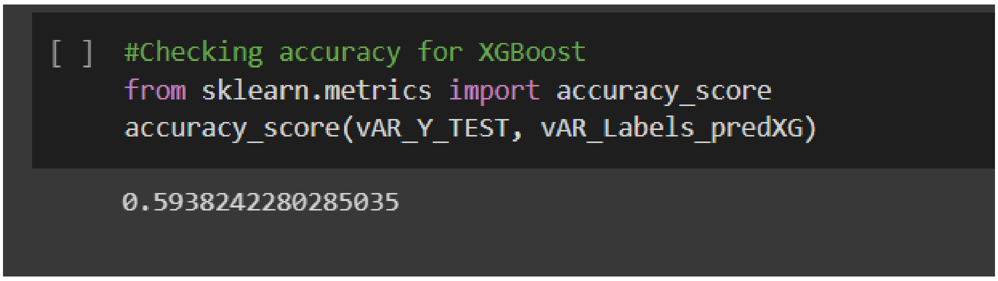
○ 11.9.1. Using Logistic Regression



```
[ ] #Checking accuracy for Logistic Regression
    from sklearn.metrics import accuracy_score
    accuracy_score(vAR_Y_TEST, vAR_Labels_predLG)

    0.5724465558194775
```

Figure 16 – Model Prediction using Logistic Regression

○ 11.9.2. Using XGBoost



```
[ ] #Checking accuracy for XGBoost
    from sklearn.metrics import accuracy_score
    accuracy_score(vAR_Y_TEST, vAR_Labels_predXG)

    0.5938242280285035
```

Figure 17 – Model Prediction using XG Boost

12. Appendix

✓ 12.1. Pandas

Pandas helps you to perform data analysis and data manipulation in Python language. Additionally, it provides us with fast and flexible data structures that make it easy to work with Relational and structured data.

✓ 12.2. Feature Tools

Featuretools is an open-source Python package for automatically creating new features from multiple tables of structured, related data. It is ideal tool for problems where there are several related tables that need to be combined into a single dataframe for training (and one for testing).

✓ 12.3. Supervised Machine Learning

Supervised learning is a technique in which the machine is thought or trained using data that is well labelled which means some data is already tagged with the correct answer. Then, the machine is provided with a new set of examples (data) so that the supervised learning algorithm analyses the training data (set of training examples) and produces a correct outcome from labelled data.

✓ 12.4. Classification

The Classification algorithm is a Supervised Learning technique that is used to identify the category of new observations on the basis of training data. In Classification, the machine learns from the given dataset or observations and then classifies new observation into a number of classes or groups. Such as, Yes or No, 0 or 1, Spam or Not Spam, cat or dog, etc. Classes can be called as targets/labels or categories.

✓ 12.5. Train/Test Split

The train-test split is a technique for evaluating the performance of a machine learning algorithm. It can be used for classification or regression problems and can be used for any supervised learning algorithm. The procedure involves taking a dataset and dividing it into two subsets. The first subset is used to fit the model and is referred to as the training dataset. The second subset is not used to train the model; instead, the input element of the dataset is provided to the model, then predictions are made and compared to the expected values. This second dataset is referred to as the test dataset.

✓ 12.6. Logistic Regression

Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.). In other words, the logistic regression model predicts P(Y=1) as a function of X.

✓ 12.7. XGBoost

XGBoost library to implement machine learning algorithms under the Gradient Boosting framework. XGBoost is portable, flexible, and efficient. It offers parallel tree boosting that helps teams to resolve many data science problems.