National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Sherin Parakkalayil Mathew
StudentID:X22128859

School of Computing
National College of Ireland

Supervisor: Musfira Jilani

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Sherin Parakkalayil Mathew
Student ID:	X22128859
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Musfira Jilani
Submission Due Date:	14/12/2023
Project Title:	Configuration Manual
Word Count:	1580
Page Count:	10

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	13th December 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Sherin Parakkalayil Mathew
X22128859

1 INTRODUCTION

This manual is created with stepwise implementation of how project implement worked and how model is created. In addition to it information and process of used libraries and tools required to carry out for project implementation is mentioned. Furthermore, information regarding specifications of location machine is mentioned in the manual. Each model information is mentioned in this configure manual.

2 HARDWARE CONFIGURATION

1. Operating system: Windows 10
2. Processor: Intel i5
3. System Compatibility: 64-bit
4. Hard Disk: 500GB
5. RAM: 8 GB

3 SOFTWARE CONFIGURATIONS

The software requirements for the configuration are displayed in Table 1 below. The data are displayed in Table 1.

Programming Language	Python 3.10.12
Framework	NumPy, Pandas, Matplotlib, Seaborn, Plotly, TensorFlow.
GPU	Intel(R) UHD Graphics
CPU	Intel(R) Core(TM) i5-10210u cpu @ 1.60GHz
IDE	Google Collab,Visual Studio

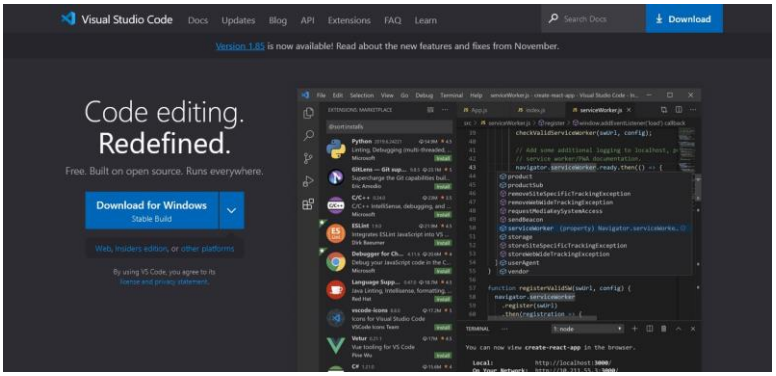
Table 1: Software Requirement Description

3.1 Python==3.10.12:-

Python is an interpreted , high-level programming language . Its high-level built in data structures , combined with dynamic typing and dynamic binding , make it very attractive for Rapid Application Development , as well as for use as a scripting or glue language to connect existing components together. Its language concept and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

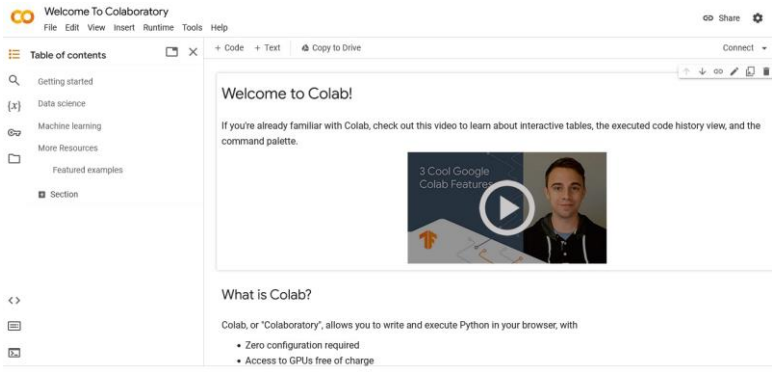
3.2 Visual Studio Code:-

Visual Studio Code (famously known as VS Code) is a free open-source text editor by Microsoft. VS Code is available for Windows, Linux, and macOS.

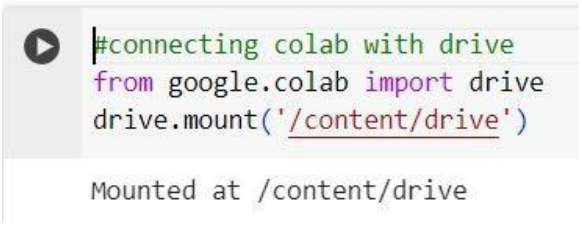


3.3 Google Colab:-

Colaboratory, or "Colab" for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing access free of charge to computing resources including GPUs.



1. Upload the whole dataset folder by opening the Google Drive tab. Given the scale of the dataset, this could take some time to complete.
2. Use the code mentioned in below image to mount the drive in the google colab environment:



4 Data Preparation and Exploration

For data preparation and Exploration, importing necessary libraries.

I. Pandas: A powerful data manipulation and analysis library used for handling structured data.

II. NumPy: Fundamental package for scientific computing with Python, mainly used for numerical operations and working with arrays.

III. Matplotlib: A popular plotting library in Python for creating static, interactive, and publication-quality visualizations.

IV. Keras: An easy-to-use high-level neural networks API, often used for fast experimentation with deep neural networks.

V. TensorFlow: An open-source machine learning framework developed by Google, widely used for building and training machine learning models, particularly neural networks.

VI. Seaborn: A data visualization library based on Matplotlib, providing a high-level interface for creating attractive and informative statistical graphics.

VII. Plotly: An interactive plotting library that allows for creating interactive web-based visualizations in Python.

VIII. Scikit-learn: A simple and efficient tool for data mining and data analysis, containing various machine learning algorithms and utilities.

IX. Pefile: A Python module to parse and work with PE (Portable Executable) files, primarily used in analyzing Windows executables.

X. Flask: A lightweight web framework for building web applications in Python, known for its simplicity and flexibility.

XI. Imbalanced-learn (Imblearn): A library used for dealing with imbalanced datasets in machine learning, providing methods for oversampling, under sampling, and generating synthetic samples.

XII. XGBoost: An efficient and scalable gradient boosting library used for super-vised learning tasks. It's known for its speed and performance in building decision tree ensembles.

```
#importing all required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.figure_factory as ff
import plotly.graph_objects as go
from sklearn import ensemble
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import mutual_info_classif,SelectPercentile
from sklearn import metrics
from sklearn.metrics import classification_report
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import confusion_matrix,accuracy_score, roc_auc_score, roc_curve
import imblearn
from imblearn.over_sampling import SMOTE
import pickle
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.metrics import confusion_matrix, classification_report , accuracy_score
from keras.layers import Input, Dense, LSTM, Conv1D, Dropout, Bidirectional, Multiply, GRU
from keras.layers import Flatten
```

Loading Dataset into pandas Dataframe

```
churndata = pd.read_csv
churndata.head()
```

During the data cleaning phase, I worked to improve the quality of the dataset by getting rid of columns that weren't needed ,figure1, making sure that data types were correct and optimized in Figure 2, and carefully fixing any cases where values were missing in figure 3. The result of their efforts was a dataframe that had been meticulously cleaned up, free of unnecessary components, and ready for insightful analysis.

```
#dropping unnecessary columns
churndata.drop(['row_number', 'invoice_id', 'customer_id'],axis='columns',inplace=True)
```

Figure 1: Dropping Unnecessary Column

```
#datatypes of dataset
churndata.info()
```

Figure 2: Dataset Datatype

```
#checking for null values
churndata.isna().sum()
```

Figure 3: Checking Null Values

In the Data Visualization part, making different plots, like boxplots, histograms, count plots, and funnel graphs, helps to see how the data is distributed, how it is related to other data, and how it patterns itself.

```
#barplot
df1 = churndata['customer_churn'].value_counts().reset_index().rename(columns={'index':'customer_churn','customer_churn':'count'})
fig = px.bar(df1, y='count', x='customer_churn', title="Countplot of Target Column(customer_churn)")
fig.show()
```

Figure 4: Barplot

```
#piechart
df1 = churndata['gender'].value_counts().reset_index().rename(columns={'index':'gender','gender':'count'})
fig = px.pie(df1, values='count', names='gender', title="Countplot of gender", hole=0.5)
fig.show()
```

Figure 5: Piechart

5 Data Preprocessing

5.1 Converting Categorical Data to Numerical

Convert categorical data to numerical using the LabelEncoder function from scikit-learn. In order to ensure machine learning compatibility, each category is given a unique numerical label in this stage. The code uses 'churndata.head()' to reflect the changes and applies LabelEncoder to a particular column (col) in the dataset ('churndata') fig 6.

5.2 Correlation Analysis

The code computes the correlation matrix for 'churndata' characteristics and creates a clustered heatmap. Using a color scale with annotations for correlation values, the heatmap shows the relationships between the features. For clarity, the figure size is adjusted to 15 by 10 shows in fig 7.

```
#converting categorical data to numerical
le = LabelEncoder()
churndata[col] = churndata[col].apply(le.fit_transform)
churndata.head()
```

Figure 6: Label Encoding

```
correlation = churndata.corr() #correlation matrix
plt.figure(figsize = (15,10))
sns.clustermap(correlation, annot = True, cmap = sns.cubehelix_palette(as_cmap=True))
```

Figure 7: Correlation Matrix

5.3 Balancing Data using SMOTE Oversampling

To address class imbalance, the Synthetic Minority Over-sampling Technique (SMOTE) is applied to oversample the minority class.

```
[ ]  1 oversample = SMOTE()
     2 X, y = oversample.fit_resample(X,y)
```

Figure 8: SMOTE Method to Balance data

5.4 Data Normalization

Data normalization is performed to scale features and ensure uniform impact in machine learning models. fig 9

5.5 Splitting Data into Train and Test Sets

The dataset is split into training and testing sets with a 90:10 ratio. fig 10

```
#data normalization
sts = StandardScaler()
X_scaled = sts.fit_transform(X)
X_scaled
```

```
array([[-1.10127809,  1.18994064, -0.59759655, ...,  0.88202423,
        -0.29692384, -0.9681003 ],
       [-1.10127809,  1.18994064, -1.79246454, ..., -0.24891571,
         0.78351067,  1.13226948],
       [ 1.45691252, -0.84037805,  0.29855444, ..., -1.55361191,
        -1.18299448,  0.14386017],
       ...,
       [-1.10127809, -0.84037805,  0.29855444, ...,  0.77141204,
         0.71259573,  0.27531334],
       [-1.10127809, -0.84037805,  0.29855444, ...,  0.12033431,
         0.69868071,  0.93080427],
       [ 0.17781722, -0.84037805,  1.19470543, ..., -0.62021922,
        -0.91365135, -1.43378489]])
```

Figure 9: Data Normalization

```
[ ]  1 #splitting the data into train and test with ratio of 90:10
     2 X_train, X_test, y_train, y_test = train_test_split(X_scaled,y,test_size=0.1,stratify=y)
```

Figure 10: Data Splitting

6 Machine Learning Model

6.1 SVM CLASSIFIER

The program makes use of a linear SVM classifier, trains it on the provided dataset, assesses accuracy, creates a heatmap of the confusion matrix, and provides a thorough classification report for a comprehensive performance evaluation fig 11.The code calculates the Receiver Operating Characteristic (ROC) curve and it computes the Area Under the Curve (AUC) score, transforms labels, and plots the ROC curve for performance evaluation. Fig 12

6.2 ADABOOST CLASSIFIER

The code utilizes an AdaBoost classifier with four weak learners, assesses accuracy, generates a heatmap of the confusion matrix, and prints a detailed classification report. This comprehensive evaluation offers insights into the classifier's performance on the test dataset. The Figure 13 and 14 shows the code and ROC Curve of ADABOOST Classifier

```
#SVM Classifier
svm_clf = SVC(probability=True, C=1.0, kernel='linear', degree=1, gamma='scale')
svmclf = svm_clf
svmclf.fit(X_train,y_train)
y_pred = svmclf.predict(X_test)
#accuracy score
print("Accuracy Score: ",accuracy_score(y_test,y_pred))
#confusion Matrix
matrix =confusion_matrix(y_test, y_pred)
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
sns.heatmap(pd.DataFrame(matrix), annot=True, cmap=sns.cubehelix_palette(as_cmap=True) ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
#Classification Report
print(classification_report(y_test, y_pred))
```

Figure 11: svm model

```
#ROC_AUC curve
probs = svmclf.predict_proba(X_test)
probs = probs[:, 1]
auc = roc_auc_score(y_test, probs)
print('AUC: %.2f' % auc)
le = preprocessing.LabelEncoder()
y_test1=le.fit_transform(y_test)
fpr, tpr, thresholds = roc_curve(y_test1, probs)
plot_roc_curve(fpr, tpr)
```

Figure 12: Roc Curve

```
#AdaBoostClassifier
adaboost_clf = AdaBoostClassifier(n_estimators=4)
adb = adaboost_clf
adb.fit(X_train,y_train)
y_pred = adb.predict(X_test)
```

13: AdaBoost Classifier

```
#ROC_AUC curve
probs = adb.predict_proba(X_test)
```

Figure 14: ROC curve Figure

```
#RandomForestClassifier
rf_clf = RandomForestClassifier(n_estimators=4, max_depth=2)
rfc = rf_clf
rfc.fit(X_train,y_train)
y_pred = rfc.predict(X_test)
```

Figure 15:Random Forest Model

```
#ROC_AUC curve
probs = rfc.predict_proba(X_test)
```

Figure 16: ROC Curve model.

6.3 RANDOM FOREST CLASSIFIER

The code implements a Random Forest classifier with four decision trees, each limited to a maximum depth of 2. The classifier is trained on the specified training data ('X train', 'y train'), and predictions are made on the test data ('X test'). Figure 15 and 16 represents the code and ROC Curve of Random Forest Model.

6.4 XGBOOST CLASSIFIER

The code utilizes an XGBoost classifier with two boosting rounds and a maximum tree depth of 1, training on 'X _train' and predicting on 'X test' fig 17 . figure 18 shows the ROC Curve.

```
#XgboostClassifier
xgb_clf = XGBClassifier(n_estimators=2, max_depth=1)
xgbc = xgb_clf
xgbc.fit(X_train,y_train)
y_pred = xgbc.predict(X_test)
```

Figure 17: XGBoost classifier

```
#ROC_AUC curve
probs = xgbc.predict_proba(X_test)
```

Figure 18: ROC Curve

7 DEEP LEARNING MODELS

Deep learning models, including CNN LSTM and GRU BILSTM displayed effective performance.

7.1 CNN LSTM

The code creates a CNN-LSTM model with one-dimensional convolutional and LSTM layers for sequence processing. It uses binary cross-entropy loss and the Adam optimizer for binary classification. Fig 19

```
#CNN LSTM
model = tf.keras.models.Sequential([
Conv1D(128, kernel_size=1, activation='relu', input_shape = (X_train.shape[1], 1)),
tf.keras.layers.LSTM(128, return_sequences=True),
tf.keras.layers.Dense(64, activation='relu'),
tf.keras.layers.LSTM(64),
tf.keras.layers.Dense(64, activation='relu'),
tf.keras.layers.Dense(32, activation='relu'),
tf.keras.layers.Dropout(0.25),
tf.keras.layers.Dense(units=2,activation='sigmoid'),
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

Figure 19: CNN LSTM Model

7.2 GRU BILSTM

The code establishes a model with GRU and Bidirectional LSTM layers, dense layers, and dropout for sequence processing. It's compiled with binary cross-entropy loss and the Adam optimizer. Fig 20

```
#GRU BILSTM
model = Sequential()
model.add(GRU(256, return_sequences=True, input_shape=(X_train.shape[1],1)))
model.add(Bidirectional(LSTM(256, return_sequences=True), input_shape=(X_train.shape[1],1)))
model.add(Dense(128, activation='relu'))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(2, activation='softmax'))
model.compile(loss='binary_crossentropy', optimizer="adam", metrics=['accuracy'])
model.summary()
```

Figure 20: GRU BILSTM Model

8 GUI USING FLASK (WEB APPLICATION SCREEN)

Built a Flask-based web app with an interactive UI for predicting customer churn. The figure 21,22,23 represent the webpage for customer churn Application.

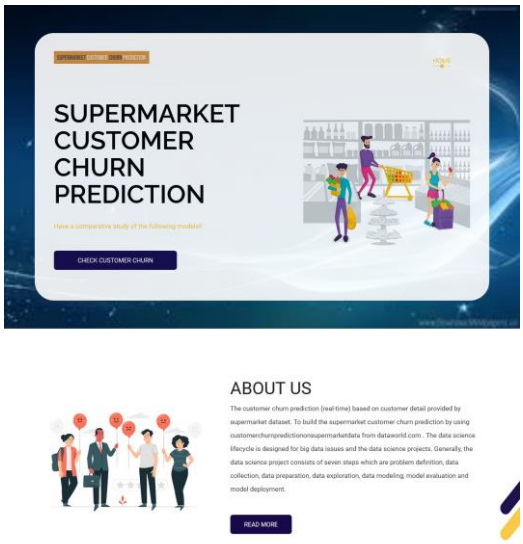


Figure 21: Web Application 1st Page

Figure 22: Web Application 2nd page

Figure 23: Customer Churn result page