Configuration Manual

MSc Research Project
Data Analytics

Abinaya Sundarapandiyan
Student ID: x21135053

School of Computing
National College of Ireland

Supervisor: Cristina Hava Muntean

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Abinaya Sundarapandiyan
Student ID:	x21135053
Programme:	Data Analytics
Year:	2022-2023
Module:	MSc Research Project
Supervisor:	Cristina Hava Muntean
Submission Due Date:	01/02/2023
Project Title:	Configuration Manual
Word Count:	378
Page Count:	7

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:	Abinaya Sundarapandiyan
Date:	1st February 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

Abinaya Sundarapandiyan
x21135053

1 Introduction

This configuration manual provides information on this research's software and hardware requirements. All the steps implemented in the research work are explained with screenshots.

2 System Requirements

Below is the system requirement. The complete project is developed in python in google colab.

- Google Colab: Intel Xeon CPU @2.20 GHz
- The GPU Instance was 250GB
- The RAM - 13 GB
- The Disk Space - 78GB
- System RAM - 16.0 GB
- Processor - Intel(R)i5 11th Gen
- OS - 64-bit Windows 11 Pro
- Software - Python

3 Import Library/Packages

It is essential to import all the necessary libraries which will be required for this project.

Packages Import

```
[ ] #Importing the necessary packages
    import pandas as pd
    import numpy as np
    import seaborn as sns

    from sklearn.preprocessing import LabelEncoder
```

Figure 1: Package Import

```
[ ] from sklearn.model_selection import train_test_split,cross_val_score
    from sklearn.preprocessing import MinMaxScaler
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.naive_bayes import GaussianNB
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.ensemble import AdaBoostClassifier
    from sklearn.ensemble import VotingClassifier
    from sklearn.feature_selection import chi2
    from sklearn.feature_selection import SelectKBest, f_classif
    import xgboost as xgb
    from sklearn import metrics
    from sklearn.model_selection import cross_val_score
    from sklearn.metrics import confusion_matrix,plot_confusion_matrix,accuracy_score,prec
    from sklearn.metrics import log_loss
    import scikitplot.plotters
```

Figure 2: Package Import

4 Data Acquisition

The dataset was downloaded from Kaggle and loaded into google drive for use. Then the dataset was imported into google colab and read.

```
[ ] #Mounting google drive
    from google.colab import drive
    drive.mount('/content/gdrive',force_remount=True)

    Mounted at /content/gdrive

[ ] data = pd.read_csv("/content/gdrive/MyDrive/data/data.csv")
```

Figure 3: Loading from Google Drive and reading the data

5 Data Preprocessing

Various preprocessing steps are carried out. The steps involve handling null value, dropping the unnecessary column, and dataset split of depression, stress, and anxiety, label encoding.

```
[ ] #replacing null values with no degree
    data_fnl=data.copy()
    data_fnl['major']=data_fnl['major'].replace(np.NaN,'No Degree')
    data_fnl['major']
```

Figure 4: Checking for Null Values

```
[ ]  #since majority of them are without degree it will not have much impact
     data_fnl=data_fnl.drop('major',axis=1)
```

```
[ ]  # QE and QI indicates the time and postion recorded while answering the qu
     time = [i for i in data_fnl.iloc[:,0:126] if  'E' in i]
     position = [i for i in data_fnl.iloc[:,0:126] if  'I' in i]
     data_fnl=data_fnl.drop(position,axis=1)
     data_fnl=data_fnl.drop(time,axis=1)
     data_fnl=data_fnl.drop(data_fnl.iloc[:,43:47],axis=1)
     data1=data_fnl.copy()
     data1=data1.drop(data_fnl.iloc[:,53:69],axis=1)
     data1=data1.replace(to replace=0,value=3)
```

Figure 5: Dropping Unnecessary Columns

```
[ ]  def sub(data2):
         return data2.subtract(1,axis=1)
     data2=sub(data2)
     DASS_keys = {'Depression': [3, 5, 10, 13, 16, 17, 21, 24, 26, 31, 34, 37, 38, 42],
                  'Anxiety': [2, 4, 7, 9, 15, 19, 20, 23, 25, 28, 30, 36, 40, 41],
                  'Stress': [1, 6, 8, 11, 12, 14, 18, 22, 27, 29, 32, 33, 35, 39]}
     Dep = []
     for i in DASS_keys["Depression"]:
         Dep.append('Q'+str(i)+'A')
     Stress = []
     for i in DASS_keys["Stress"]:
         Stress.append('Q'+str(i)+'A')
     Anx = []
     for i in DASS_keys["Anxiety"]:
         Anx.append('Q'+str(i)+'A')
     depression= data2.filter(Dep)
     stress = data2.filter(Stress)
     anxiety = data2.filter(Anx)
```

Figure 6: Dataset Split

```
#Lable encoding the column condition
Condition= LabelEncoder()
Condition.fit(Depression.Condition)
Depression['Condition'] = Condition.transform (Depression.Condition)
Stress['Condition'] = Condition.transform (Stress.Condition)
Anxiety['Condition'] = Condition.transform (Anxiety.Condition)
```

Figure 7: Lable Encoding

6 Exploratory Data Analysis

Exploratory data analysis is done to understand the data. The distribution of the severity level for different illnesses was analyzed. The distribution age and many features were analyzed. A couple of Exploratory data analysis snippets are provided below.

```
[ ] # sns.set(font_scale=2)
    plt.figure(figsize=(12,8))
    sns.countplot(Depression.sort_values('Condition').Condition)
    plt.title('People Condition of Depression Level',fontsize=15)
```

Figure 8: Distribution of Condition

```
[ ] #1=Male
    #2=Female
    #3=Other

    plt.figure(figsize=(10,6))
    sns.countplot(Anxiety1.sort_values('gender').gender,hue=Anxiety['Condition'],palette='BuGn_r')
    plt.title('Anxiety Condition of Different Gender',fontsize=15)
```

Figure 9: Severity Level Distribution for Gender

7 Feature Selection

Using Chi-Square the features were selected. 20 required features were selected from 38 features for all the three depression, stress and anxiety.

```
[ ] #Voting classifier
    from sklearn.ensemble import VotingClassifier
    clf1 = KNeighborsClassifier()
    clf2 = xgb.XGBClassifier()
    clf3 = GaussianNB()
    eclf1 = VotingClassifier(estimators=[('dtc', clf1), ('xgb', clf2), ('GNB', clf3)], voting='soft')
    eclf1.fit(X_train_scaled,y_train)
    Acc_eclf1=round(accuracy_score(y_test,eclf1.predict(X_test_scaled)),3)
    f1_eclf1=round(f1_score(y_test, eclf1.predict(X_test_scaled),average='weighted'),3)
    recall_eclf1=round(recall_score(y_test,eclf1.predict(X_test_scaled),average='weighted'),3)
    precision_eclf1=round(precision_score(y_test,eclf1.predict(X_test_scaled),average='weighted'),3)
    scikitplot.metrics.plot_confusion_matrix(y_test,eclf1.predict(X_test_scaled))
    print('Accuracy Depression:',Acc_eclf1)
    print('F1_Score Depression:',f1_eclf1)
    print('Recall_Score Depression:',recall_eclf1)
    print('Precision_Score Depression:',precision_eclf1)
```

Figure 10: Chi Square Feature Selection

8 Train and Test Split

Train and test data is split into 80/20 ratios. Scalar transformation is done before model building.

```
[ ] #train test split 80/20
    X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20,random_state=0)
```

```
[ ] X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
```

Figure 11: Train and Test Split

9 Model Building

Different traditional machine-learning models were implemented along with the ensemble model voting classifier and feed-forward neural network for model comparison and validation.

```
[ ] #Voting classifier
    from sklearn.ensemble import VotingClassifier
    clf1 = KNeighborsClassifier()
    clf2 = xgb.XGBClassifier()
    clf3 = GaussianNB()
    eclf1 = VotingClassifier(estimators=[('dtc', clf1), ('xgb', clf2), ('GNB', clf3)], voting='soft')
    eclf1.fit(X_train_scaled,y_train)
    Acc_eclf1=round(accuracy_score(y_test,eclf1.predict(X_test_scaled)),3)
    f1_eclf1=round(f1_score(y_test, eclf1.predict(X_test_scaled),average='weighted'),3)
    recall_eclf1=round(recall_score(y_test,eclf1.predict(X_test_scaled),average='weighted'),3)
    precision_eclf1=round(precision_score(y_test,eclf1.predict(X_test_scaled),average='weighted'),3)
    scikitplot.metrics.plot_confusion_matrix(y_test,eclf1.predict(X_test_scaled))
    print('Accuracy Depression:',Acc_eclf1)
    print('F1_Score Depression:',f1_eclf1)
    print('Recall_Score Depression:',recall_eclf1)
    print('Precision_Score Depression:',precision_eclf1)
```

Figure 12: Voting Classifier Model

```
[ ] # Split into train+val and test
    X7_trainval, X7_test, y7_trainval, y7_test = train_test_split(X4, y4, test_size=0.2, random_state=69)
```

```
[ ] # Split train into train-val
    X7_train, X7_val, y7_train, y7_val = train_test_split(X7_trainval, y7_trainval, test_size=0.1, random_state=21)
```

```
[ ] scaler = MinMaxScaler()
    X7_train = scaler.fit_transform(X7_train)
    X7_val = scaler.transform(X7_val)
    X7_test = scaler.transform(X7_test)
    X7_train, y7_train = np.array(X7_train), np.array(y7_train)
    X7_val, y7_val = np.array(X7_val), np.array(y7_val)
    X7_test, y7_test = np.array(X7_test), np.array(y7_test)
```

Figure 13: Train, Validation Split for Feed Forward Neural Network

```
class MulticlassClassification(nn.Module):
    def __init__(self, num_feature, num_class):
        super(MulticlassClassification, self).__init__()

        self.layer_1 = nn.Linear(num_feature, 512)
        self.layer_2 = nn.Linear(512, 128)
        self.layer_3 = nn.Linear(128, 64)
        self.layer_out = nn.Linear(64, num_class)

        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(p=0.2)
        self.batchnorm1 = nn.BatchNorm1d(512)
        self.batchnorm2 = nn.BatchNorm1d(128)
        self.batchnorm3 = nn.BatchNorm1d(64)

    def forward(self, x):
        x = self.layer_1(x)
        x = self.batchnorm1(x)
        x = self.relu(x)

        x = self.layer_2(x)
        x = self.batchnorm2(x)
        x = self.relu(x)
        x = self.dropout(x)

        x = self.layer_3(x)
        x = self.batchnorm3(x)
        x = self.relu(x)
        x = self.dropout(x)

        x = self.layer_out(x)
```

Figure 14: 3 Layes Neural Network Model

```
print("Begin training.")
for e in tqdm(range(1, EPOCHS+1)):

    # TRAINING
    train_epoch_loss = 0
    train_epoch_acc = 0
    model.train()
    for X_train_batch, y_train_batch in train_loader:
        X_train_batch, y_train_batch = X_train_batch.to(device), y_train_batch.to(device)
        optimizer.zero_grad()

        y_train_pred = model(X_train_batch)

        train_loss = criterion(y_train_pred, y_train_batch)
        train_acc = multi_acc(y_train_pred, y_train_batch)

        train_loss.backward()
        optimizer.step()

        train_epoch_loss += train_loss.item()
        train_epoch_acc += train_acc.item()
```

```
    # VALIDATION
    with torch.no_grad():

        val_epoch_loss = 0
        val_epoch_acc = 0

        model.eval()
        for X_val_batch, y_val_batch in val_loader:
            X_val_batch, y_val_batch = X_val_batch.to(device), y_val_batch.to(device)

            y_val_pred = model(X_val_batch)

            val_loss = criterion(y_val_pred, y_val_batch)
            val_acc = multi_acc(y_val_pred, y_val_batch)

            val_epoch_loss += val_loss.item()
            val_epoch_acc += val_acc.item()
            loss_stats['train'].append(train_epoch_loss/len(train_loader))
            loss_stats['val'].append(val_epoch_loss/len(val_loader))
            accuracy_stats['train'].append(train_epoch_acc/len(train_loader))
            accuracy_stats['val'].append(val_epoch_acc/len(val_loader))
```

Figure 15: Training and validation of the FNN model