National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Shreya Merkaje Ravi
Student ID: x18190910

School of Computing
National College of Ireland

Supervisor: Dr. Muhammad Iqbal

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Shreya Merkaje Ravi
Student ID:	x18190910
Programme:	Data Analytics
Year:	2020
Module:	MSc Research Project
Supervisor:	Dr. Muhammad Iqbal
Submission Due Date:	17/08/2020
Project Title:	Configuration Manual
Word Count:	862
Page Count:	1

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:	17th August 2020

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

Shreya Merkaje Ravi
x18190910
MSc Research Project in Data Analytics

August 17, 2020

1 Introduction

This document is the configuration manual and has all details on project specifications, hardware, software requirements and the programming phases of implementation for the below research project in detail:
"Predictive Modelling to Forecast the Crop Yield and Classification of Plant Diseases"

2 System Configuration

2.1 Hardware

• Processor: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz
• RAM: 8 GB
• System Type: Windows OS, 64-bit
• Storage: 1 TB HDD

2.1.1 Software

• **Microsoft Excel 2016**: This spreadsheet tool offered by Microsoft is used for storing the downloaded datasets in flat files as CSV (comma-separated values).

• **Google colaboratory**: This is also commonly known as colab, which is a free cloud service that provides the users with free GPU services to run machine learning models in an environment like Jupyter notebook. For this research, colab extensively used for all the data analysis, visualization and modelling. In colab, there is an option to either select GPU or TPU settings depending on the volume of data used. For this research, GPU services were used, and this setting is found under the 'Runtime' tab and 'change runtime type' option. The reference is provided in Figure 1.

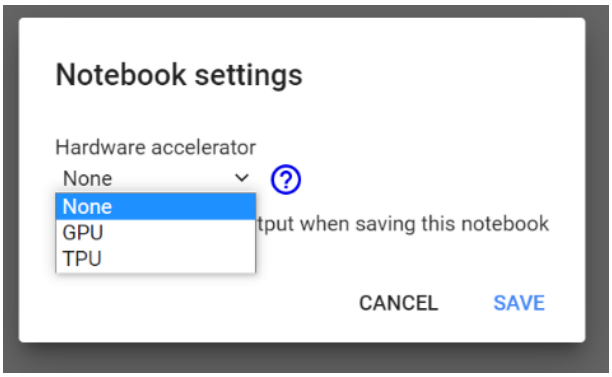

Figure 1: GPU setting in Google colab

3 Project Development

Implementation of this entire research work is done using Python programming. Initial stages of the research project development include data pre-processing, merging the datasets and label encoding. After data preparation, predictive modelling is implemented using machine learning techniques in python using the sk-learn (scikit-learn) and keras libraries.

3.1 Data Preparation

Data pre-processing is carried out using the pandas (data frame) and NumPy (arrays) libraries. In the research 3 datasets are used, among which 2 are in CSV format and another is an image dataset.

3.1.1 Agriculture Dataset

The agriculture dataset consists of data related to State name, district name, season, crop type and production.

Variable Name	Description
State_Name (String)	Name of the State in India
District_Name (String)	District names in the State
Crop_Year (Date)	Year when the crop was grown
Season (String)	Season in which the crop is grown
Crop (String)	Name of the crop grown
Area (Numeric)	Area used to cultivate the crop
Production (Numeric)	Crop produced in a year

Figure 2: Metadata of agriculture dataset

First, the check is done for missing values. Figure 2 shows that the dataset does not have any missing values.

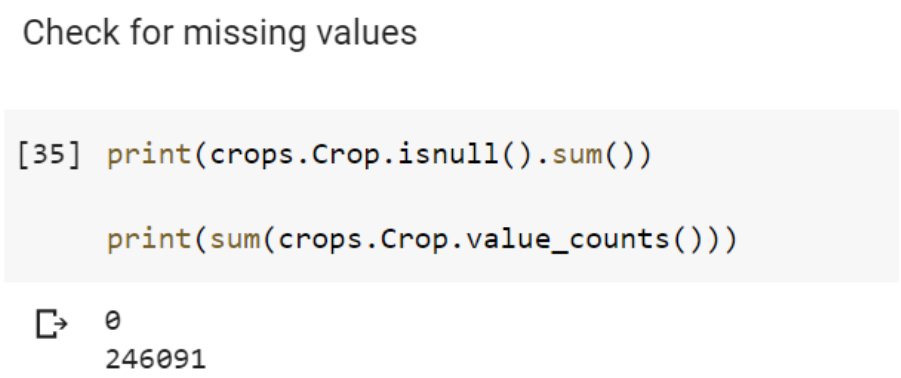

Figure 3: Missing data

Since there is no discrepancy in the dataset, no further actions were taken.

3.1.2 Rainfall Dataset

The rainfall dataset is also in csv format and has data on State name, year, monthly rainfall and annual rainfall.

Variable Name	Description
Subdivision (String)	Name of the State in India
Year (Date)	Year when the rainfall was recorded
Jan – Dec (Numeric)	Rainfall recorded in each month of the year
Annual (Numeric)	Total recorded rainfall in a year

Figure 4: Metadata of rainfall data

Few of the State names in rainfall dataset have a discrepancy in naming convention and it was handled as given in Figure 5.

```
rainfall["SUBDIVISION"] = rainfall["SUBDIVISION"].replace("ANDAMAN & NICOBAR ISLANDS","Andaman and Nicobar Islands")
rainfall["SUBDIVISION"] = rainfall["SUBDIVISION"].replace("JAMMU & KASHMIR","Jammu and Kashmir")
rainfall["SUBDIVISION"] = rainfall["SUBDIVISION"].replace("ORISSA","Odisha")


for state in found:
    state_rainfall = state.upper().strip()
    state = state.strip()
    rainfall["SUBDIVISION"] = rainfall["SUBDIVISION"].replace(state_rainfall,state)

print(rainfall.SUBDIVISION.unique())
```

Figure 5: Code to update few naming convention

The dataset also has another discrepancy in state names were a single State is updated with different names. For example, The state of Andra Pradesh is provided as both Coastal Andra Pradesh and Rayalseema. These two names have to updated to Andra Pradesh. A function provided in Figure 6 and Figure 7 is used for this purpose. The several States had to updated correspondingly.

```
1
2  def getting_values(states_information):
3      state_value = []
4      result = []
5      for state in states_information:
6          temp = rainfall.loc[rainfall["SUBDIVISION"] == state]
7          state_value.append(temp)
8      for i in range(0,len(state_value)):
9          state_value[i] = state_value[i].drop(["SUBDIVISION"],axis='columns')
10         #working fine
11     for i in range(0,len(state_value)):
12         state_value[i] = state_value[i].reset_index(drop=True)
13         #working fine
14     years = state_value[0]["YEAR"]
15
16     if(len(state_value)==2):
17         result = state_value[0].add(state_value[1],fill_value=None)
18     elif(len(state_value)==3):
19         temp1  = state_value[0].add(state_value[1],fill_value=None)
20         result = temp1.add(state_value[2],fill_value=None)
21     elif(len(state_value)==4):
22         temp1 = state_value[0].add(state_value[1],fill_value=None)
23         temp2 = temp1.add(state_value[2],fill_value=None)
24         result = temp2.add(state_value[3],fill_value=None)
25
26     result = result.drop(["YEAR"],axis='columns')
27     result.insert(0,"YEAR",years)
28     return result
29
```

Figure 6: Function to update State names

```
# KARNATAKA
states_karnataka = ['Coastal Karnataka','North Interior Karnataka','South Interior Karnataka']
karnataka  = getting_values(states_karnataka)
karnataka.insert(0,"SUBDIVISION","Karnataka")
print("KARNATAKA\n\n")
print(karnataka)
```

Figure 7: Function to update State names

The multiple States in this dataset are grouped and provided as one. These need to split and given individual values. This task was performed using the code provided in Figure 8.

```
# ASSAM
assam = rainfall.loc[rainfall["SUBDIVISION"] == "Assam & Meghalaya"]
assam["SUBDIVISION"] = assam["SUBDIVISION"].replace("Assam & Meghalaya","Assam")
print(assam)

# NAGALAND
nagaland = rainfall.loc[rainfall["SUBDIVISION"] == "Naga Mani Mizo Tripura"]
nagaland["SUBDIVISION"] = nagaland["SUBDIVISION"].replace("Naga Mani Mizo Tripura","Nagaland")
print(nagaland)

# MANIPUR
manipur = rainfall.loc[rainfall["SUBDIVISION"] == "Naga Mani Mizo Tripura"]
manipur["SUBDIVISION"] = manipur["SUBDIVISION"].replace("Naga Mani Mizo Tripura","Manipur")
print(manipur)

# MIZORAM
mizoram = rainfall.loc[rainfall["SUBDIVISION"] == "Naga Mani Mizo Tripura"]
mizoram["SUBDIVISION"] = mizoram["SUBDIVISION"].replace("Naga Mani Mizo Tripura","Mizoram")
print(mizoram)

# HARYANA
haryana = rainfall.loc[rainfall["SUBDIVISION"] == "Haryana Delhi & Chandigarh"]
haryana["SUBDIVISION"] = haryana["SUBDIVISION"].replace("Haryana Delhi & Chandigarh","Haryana")
print(haryana)

# PUDUCHERRY
puducherry = rainfall.loc[rainfall["SUBDIVISION"] == "Tamil Nadu"]
puducherry["SUBDIVISION"] = puducherry["SUBDIVISION"].replace("Tamil Nadu","Puducherry")
print(puducherry)

# DADRA and NAGAR
dadra_nagar = rainfall.loc[rainfall["SUBDIVISION"] == "Gujarat"]
dadra_nagar["SUBDIVISION"] = dadra_nagar["SUBDIVISION"].replace("Gujarat","Dadra and Nagar Haveli")
print(dadra_nagar)
```

Figure 8: Code to split the grouped States

Using the below information, new columns are added to rainfall dataset using the code provided in Figure 9 and Figure 10.

Rabi Months : October to March
Kharif Months : July to October
Summer Months : April to June
Whole year : All
Winter months: October to January

Autumn months : September to November

```
# ASSAM
assam = rainfall.loc[rainfall["SUBDIVISION"] == "Assam & Meghalaya"]
assam["SUBDIVISION"] = assam["SUBDIVISION"].replace("Assam & Meghalaya","Assam")
print(assam)

# NAGALAND
nagaland = rainfall.loc[rainfall["SUBDIVISION"] == "Naga Mani Mizo Tripura"]
nagaland["SUBDIVISION"] = nagaland["SUBDIVISION"].replace("Naga Mani Mizo Tripura","Nagaland")
print(nagaland)

# MANIPUR
manipur = rainfall.loc[rainfall["SUBDIVISION"] == "Naga Mani Mizo Tripura"]
manipur["SUBDIVISION"] = manipur["SUBDIVISION"].replace("Naga Mani Mizo Tripura","Manipur")
print(manipur)

# MIZORAM
mizoram = rainfall.loc[rainfall["SUBDIVISION"] == "Naga Mani Mizo Tripura"]
mizoram["SUBDIVISION"] = mizoram["SUBDIVISION"].replace("Naga Mani Mizo Tripura","Mizoram")
print(mizoram)

# HARYANA
haryana = rainfall.loc[rainfall["SUBDIVISION"] == "Haryana Delhi & Chandigarh"]
haryana["SUBDIVISION"] = haryana["SUBDIVISION"].replace("Haryana Delhi & Chandigarh","Haryana")
print(haryana)

# PUDUCHERRY
puducherry = rainfall.loc[rainfall["SUBDIVISION"] == "Tamil Nadu"]
puducherry["SUBDIVISION"] = puducherry["SUBDIVISION"].replace("Tamil Nadu","Puducherry")
print(puducherry)

# DADRA and NAGAR
dadra_nagar = rainfall.loc[rainfall["SUBDIVISION"] == "Gujarat"]
dadra_nagar["SUBDIVISION"] = dadra_nagar["SUBDIVISION"].replace("Gujarat","Dadra and Nagar Haveli")
print(dadra_nagar)
```

Figure 9: Code to create new columns using defined information

```
# ASSAM
assam = rainfall.loc[rainfall["SUBDIVISION"] == "Assam & Meghalaya"]
assam["SUBDIVISION"] = assam["SUBDIVISION"].replace("Assam & Meghalaya","Assam")
print(assam)

# NAGALAND
nagaland = rainfall.loc[rainfall["SUBDIVISION"] == "Naga Mani Mizo Tripura"]
nagaland["SUBDIVISION"] = nagaland["SUBDIVISION"].replace("Naga Mani Mizo Tripura","Nagaland")
print(nagaland)

# MANIPUR
manipur = rainfall.loc[rainfall["SUBDIVISION"] == "Naga Mani Mizo Tripura"]
manipur["SUBDIVISION"] = manipur["SUBDIVISION"].replace("Naga Mani Mizo Tripura","Manipur")
print(manipur)

# MIZORAM
mizoram = rainfall.loc[rainfall["SUBDIVISION"] == "Naga Mani Mizo Tripura"]
mizoram["SUBDIVISION"] = mizoram["SUBDIVISION"].replace("Naga Mani Mizo Tripura","Mizoram")
print(mizoram)

# HARYANA
haryana = rainfall.loc[rainfall["SUBDIVISION"] == "Haryana Delhi & Chandigarh"]
haryana["SUBDIVISION"] = haryana["SUBDIVISION"].replace("Haryana Delhi & Chandigarh","Haryana")
print(haryana)

# PUDUCHERRY
puducherry = rainfall.loc[rainfall["SUBDIVISION"] == "Tamil Nadu"]
puducherry["SUBDIVISION"] = puducherry["SUBDIVISION"].replace("Tamil Nadu","Puducherry")
print(puducherry)

# DADRA and NAGAR
dadra_nagar = rainfall.loc[rainfall["SUBDIVISION"] == "Gujarat"]
dadra_nagar["SUBDIVISION"] = dadra_nagar["SUBDIVISION"].replace("Gujarat","Dadra and Nagar Haveli")
print(dadra_nagar)
```

Figure 10: Code to create new columns using defined information

3.1.3 Plant Image dataset

The plant image dataset consists of 70,295 images of plant leaf which belong to 38 different disease classes. Since the dataset is huge, it was first zipped and loaded into Google drive. From colab, this data was loaded from Google drive and unzipped for further research. The code used for this process is given in Figure 11.

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

zip_ref = zipfile.ZipFile("/content/drive/My Drive/New_Plant.zip", 'r')
zip_ref.extractall("/tmp")
zip_ref.close()

DATASET_PATH = '/tmp/New Plant Diseases Dataset(Augmented)/Plant_Village/train'

```

Figure 11: Code to access data from Google drive

After loading the data, labels were created using code in Figure 12

```
labels = [x for x in range(len(disease_cls))]
images = []
labels_cls = []
name_cls = []
for i in labels:
    #temp_im_list = []
    temp_path = DATASET_PATH + '/' + disease_cls[i]
    im_list = os.listdir(temp_path)
    for j in im_list:
        temp_path1 = temp_path + '/' + j
        images.append(temp_path1)
        labels_cls.append(i)
        name_cls.append(disease_cls[i])
data = pd.DataFrame()
data['FileName']= images
data['Label']= labels_cls
data['ClassName'] = name_cls
data.shape
```

Figure 12: Label creation

3.1.4 Data Merging

The agriculture and rainfall dataset needs to merged but before merging it should be made sure to have consistent data. The code in Figure 13 is used for verifying the matching State names in both datasets.

```
count = 0
n_found = []
states_updated = rainfall.SUBDIVISION.unique()
print(f"NUMBER OF STATES IN RAINFALL {len(states_updated)}")
for state in states_data:
    if state.strip() in states_updated:
        count +=1
    else:
        n_found.append(state)

print(f"TOTAL NUMBER OF STATES MATCHING crops = {count}")
print(n_found)
```

Figure 13: Code to verify matching State names

The rainfall value from rainfall dataset was added to agriculture data on a conditional matching statement. The code for this is provided in Figure 14.

```
[31] rainfall_new = []
     for index,row in crops.iterrows():
         state = row["State_Name"]
         season = row['Season'].strip()
         year = row['Crop_Year']
         values = rainfall[season].loc[(rainfall['SUBDIVISION']==state) & (rainfall['YEAR']==year)]
         if len(values)>0:
             value = values.to_numpy()[0]
         else:
             value = np.nan
         rainfall_new.append(value)

     print(len(rainfall_new))
```

Figure 14: Code to merge data

3.2 Feature Engineering

Application of feature engineering techniques will help the models to perform better and provide good accuracy. For the combined agriculture and rainfall dataset encoding techniques were applied.

One-Hot Encoding

Using one-hot encoding it is possible to convert categorical values into numeric ones. Code to apply one-hot encoding in provided in Figure 15.

```
ohe = ce.OneHotEncoder()
d_1 = ohe.fit_transform(crops)
d_1.shape
```

Figure 15: One-Hot Encoding

Label Encoding

Using label encoding process it is possible to convert the labels into numerical form. Code to apply label encoding is provided in Figure 16.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
state = crops.State_Name
state = le.fit_transform(state)
crops = crops.assign(State_Name = state)
district = crops.District_Name
district = le.fit_transform(district)
crops = crops.assign(District_Name = district)

season = crops.Season
season = le.fit_transform(season)
crops = crops.assign(Season = season)

crop = crops.Crop
crop = le.fit_transform(crop)
crops = crops.assign(Crop = crop)
print(crops)
```

Figure 16: Label Encoding

Label encoding was found suitable for the dataset and hence one-hot encoding and binary encoding were discarded.

After encoding the missing values in the dataset was analysed and were imputed with the help of mean values. The code to update the missing values is provided in Figure 17.

```
X.Rainfall.fillna(X.Rainfall.mean(),inplace=True)
Y.Production.fillna(Y.Production.mean(),inplace=True)
Y_rep = Y.copy()
print(type(Y_rep))
print(Y_rep)
print(X.shape)
```

Figure 17: Code to update missing values

3.3 Modelling

Random Forest Regressor, Decision Tree Regressor, Gradient Boost Regressor, Convolution Neural Network were used for modelling.

3.3.1 Data Split

The data was scaled and then split into training and testing sets. Code used to split the dataset is provided in Figure 18.

```
X_train,X_test,Y_train,Y_test = train_test_split(X, Y, train_size = 0.8, test_size = 0.2,random_state = 0)


from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
Y_train = scaler.fit_transform(Y_train)
Y_test = scaler.fit_transform(Y_test)
print(type(Y_train))
print("Feature Scaling -> ready for training")
```

Figure 18: Code to split the dataset

The code to split plant image dataset is given in Figure 19.

```
from sklearn.model_selection import train_test_split
# Split the data
train, val= train_test_split(data, test_size=0.2, shuffle= True, random_state = 100)
```

Figure 19: Code to split the dataset

3.4 random Forest Regressor

Code for implenting Random Forest Regressor in given in Figure 20.

```
from sklearn.ensemble import RandomForestRegressor

# TRAINING the model
model_rr = RandomForestRegressor(n_estimators = 1000,random_state = 0)
model_rr.fit(X_train, np.ravel(Y_train))

# TESTING the model
rr_prediction = model_rr.predict(X_test)
```

Figure 20: Code for implementing Random Forest

3.5 Decision Tree Regressor

To implement Decision Tree Regressor refer Figure 21.

```
from sklearn.tree import DecisionTreeRegressor as dtr

# TRAINING the model
model_dtr = dtr(max_depth=4)
model_dtr.fit(X_train,np.ravel(Y_train))

# TESTING the model
dtr_prediction = model_dtr.predict(X_test)
```

Figure 21: Code for implementing Decision Tree Regressor

3.6 Gradient Boost Regressor

Refer the figure 22 for implementation of Gradient Boost Regressor.

```
from sklearn.tree import DecisionTreeRegressor as dtr

# TRAINING the model
model_dtr = dtr(max_depth=4)
model_dtr.fit(X_train,np.ravel(Y_train))

# TESTING the model
dtr_prediction = model_dtr.predict(X_test)
```

Figure 22: Code for implementing Gradient Boost Regressor

Code for implementing Convolution Neural Network on plant image dataset is given in Figure 23.

```
model = Sequential()

model.add(Conv2D(16, kernel_size=(3, 3), activation='relu',
                 input_shape=(resize, resize, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='Adam', metrics=['categorical_accuracy'])

print(model.summary())
```

Figure 23: Code for implementing CNN model