Configuration Manual

MSc Research Project
Data Analytics

Sana Inamdar
Student ID: x22217061

School of Computing
National College of Ireland

Supervisor: Dr Catherine Mulwa

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Sana Inamdar
Student ID:	x22217061
Programme:	Data Analytics
Year:	2024
Module:	MSc Research Project
Supervisor:	Dr Catherine Mulwa
Submission Due Date:	12/08/2024
Project Title:	Configuration Manual
Word Count:	598
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:	Sana Inamdar
Date:	12/08/2024

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

Sana Inamdar
x22217061

1 Introduction

The configuration manual covers the important parts of the project such as hardware and software used, and the steps taken to implement the codes for this research project: "Predicting energy consumption using Machine Learning and Deep Learning".

2 System Configuration

Here we will discuss about the system configurations we used to execute the machine learning and deep learning codes smoothly as also it is necessary to have all the pre-requisites handy before starting the implementation.

2.1 Hardware Requirements

1. Processor : 12th Gen Intel(R) Core(TM) i5-1235U 1.30 GHz
2. Installed RAM: 16.0 GB (15.7 GB usable)
3. System type :64-bit operating system, x64-based processor
4. Operating System: Windows 11

2.2 Software Requirements

1. Microsoft Excel: Used for saving data and data analysis.
2. Jupyter Notebook: Data cleaning, Data pre-processing, Feature Selection and Feature engineering and execution of machine learning models and deep learning models.
3. Python:Version Python 3.7 or higher is recommended.

3 Project Development

The project development here has several steps involved such as: Data Understanding, Data Preparation, Data Preprocessing, Feature Selection, Feature Engineering and Implementation of Machine learning and Deep learning algorithms. The research project has code written of various lines for the analysis. Below are all the essential steps explained of this research project:

3.1 Data Preparation and Data Preprocessing

The PJM dataset is taken from Kaggle and is also accessible from the PJM's website. The dataset contains historic data of different electrical companies spread in different parts of the United States. The first step is importing all the necessary libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import holidays

from IPython.core.display  import HTML
from IPython.display       import Image
from datetime              import date
from tabulate              import tabulate
from scipy.stats           import chi2_contingency

from boruta                import BorutaPy
from sklearn.ensemble      import RandomForestRegressor

from sklearn.metrics       import mean_absolute_error, mean_squared_error
from sklearn.linear_model  import LinearRegression, Lasso
from sklearn.ensemble      import RandomForestRegressor
import xgboost as xgb

import random
import warnings
warnings.filterwarnings( 'ignore' )
```

Figure 1: Importing Libraries

Next step is to import all the CSV files present in our dataset and storing it in different dataframes

```
aep = pd.read_csv( 'data/AEP_hourly.csv', low_memory=False )
comed = pd.read_csv( 'data/COMED_hourly.csv', low_memory=False )
dayton = pd.read_csv( 'data/DAYTON_hourly.csv', low_memory=False )
deok = pd.read_csv( 'data/DEOK_hourly.csv', low_memory=False )
dom = pd.read_csv( 'data/DOM_hourly.csv', low_memory=False )
duq = pd.read_csv( 'data/DUQ_hourly.csv', low_memory=False )
ekpc = pd.read_csv( 'data/EKPC_hourly.csv', low_memory=False )
fe = pd.read_csv( 'data/FE_hourly.csv', low_memory=False )
ni = pd.read_csv( 'data/NI_hourly.csv', low_memory=False )
pjm = pd.read_csv( 'data/PJM_Load_hourly.csv', low_memory=False )
pjme = pd.read_csv( 'data/PJME_hourly.csv', low_memory=False )
pjmw = pd.read_csv( 'data/PJMW_hourly.csv', low_memory=False )
```

Figure 2: Storing csv files in dataframes

Combining all the dataframes into one by adding new column(electric company) in each of the file making it easy to combine all the dataframes together.

```
aep['electric_company'] = 'AEP'
comed['electric_company'] = 'COMED'
dayton['electric_company'] = 'DAYTON'
deok['electric_company'] = 'DEOK'
dom['electric_company'] = 'DOM'
duq['electric_company'] = 'DUQ'
ekpc['electric_company'] = 'EKPC'
fe['electric_company'] = 'FE'
ni['electric_company'] = 'NI'
pjm['electric_company'] = 'PJM'
pjme['electric_company'] = 'PJME'
pjmw['electric_company'] = 'PJMW'
```

Figure 3: Concatenating dataframes

Checking null values

```
df1.isna().sum()

datetime                0
mw_energy_consumption   0
electric_company        0
```

Figure 4: Null values

3.2 Feature Engineering

For getting more insights from our dataset which now overall has 3 columns, we have derived more columns out of the datetime column to get a broader perspective of our data. The following are the columns derived during feature engineering:

```
datetime                 datetime64[ns]
mw_energy_consumption           float64
electric_company                 object
date                     datetime64[ns]
year                              int32
month                             int32
hour_of_day                       int32
season                           object
holidays                         object
day of week                       int32
```

Figure 5: Datatypes

3.3 Feature Selection using Boruta

By using Boruta function we selected the most relevant variables.

```
# training and test dataset for Boruta
X_train_n = X_train.drop( ['date', 'datetime', 'mw_energy_consumption'], axis=1 ).values
y_train_n = y_train.values.ravel()

# define RandomForestRegression
rf = RandomForestRegressor( n_jobs=-1 )

# define Boruta
boruta = BorutaPy( rf, n_estimators='auto', verbose=2, random_state=42 ).fit( X_train_n, y_train_n )
```

Figure 6: Boruta function

4 Codes for Machine Learning and Deep Learning Models

4.1 Machine Learning

In Machine Learning we have implemented three models which are Linear Regression, Random Forest and XG Boost, we will access these models based on their accuracy and the error score.

4.1.1 Linear Regression Model

```
# model
lr = LinearRegression().fit( x_train, y_train )

# prediction
yhat_lr = lr.predict( x_test )

# performance
lr_result = ml_error( 'Linear Regression', np.expm1( y_test ), np.expm1( yhat_lr ) )
lr_result
```

Figure 7: Linear Regression Model

Linear regression using cross validation to lessen the error

```
lr_result_cv = cross_validation( X_training, 5, 'Linear Regression', lr, verbose=False )
lr_result_cv
```

Figure 8: Linear Regressor cross validation Model

4.1.2 Random Forest Model

```
# model
rf = RandomForestRegressor( n_estimators=100, n_jobs=-1, random_state=42 ).fit( x_train, y_train )

# prediction
yhat_rf = rf.predict( x_test )

# performance
rf_result = ml_error( 'Random Forest Regressor', np.expm1( y_test ), np.expm1( yhat_rf ) )
rf_result
```

Figure 9: Random Forest Model

Random Forest using cross validation to lessen the error

```
rf_result_cv = cross_validation( X_training, 5, 'Random Forest Regressor', rf, verbose=True )
rf_result_cv
```

Figure 10: Random Forest cross validation

4.1.3 XG Boost Model

```
# model
model_xgb = xgb.XGBRegressor( objective='reg:squarederror',
                              n_estimators=100,
                              eta=0.01,
                              max_depth=10,
                              subsample=0.7,
                              colsample_bytree=0.9 ).fit( x_train, y_train )

# prediction
yhat_xgb = model_xgb.predict( x_test )

# performance
xgb_result = ml_error( 'XGBoost Regressor', np.expm1( y_test ), np.expm1( yhat_xgb ) )
xgb_result
```

Figure 11: XG Boost

XG Boost using cross validation to lessen the error.

```
xgb_result_cv = cross_validation( X_training, 5, 'XGBoost Regressor', model_xgb, verbose=True )
xgb_result_cv
```

Figure 12: XG Boost cross validation

4.2 Deep Learning Models

In Deep Learning we have implemented two models which are LSTM and RNN, we will access these models based on their accuracy and the error score.

4.2.1 Simple Recurrent Neural Network

```
simple_rnn_model = Sequential([
    SimpleRNN(50, input_shape=(x_train_reshaped.shape[1], x_train_reshaped.shape[2])),
    Dense(1)
])
simple_rnn_model.compile(optimizer='adam', loss='mse')
```

Figure 13: Simple RNN

4.2.2 Simple Long Short Term Memory

```
# Simple LSTM
simple_lstm_model = Sequential([
    LSTM(50, input_shape=(x_train_reshaped.shape[1], x_train_reshaped.shape[2])),
    Dense(1)
])
simple_lstm_model.compile(optimizer='adam', loss='mse')
```

Figure 14: Simple LSTM

5 Experiments with RNN and LSTM

5.1 RNN with 3 layers

```
complex_rnn_model = Sequential([
    SimpleRNN(50, return_sequences=True, input_shape=(x_train_reshaped.shape[1], x_train_reshaped.shape[2])),
    LayerNormalization(),
    Dropout(0.5),
    SimpleRNN(50, return_sequences=True),
    LayerNormalization(),
    Dropout(0.5),
    SimpleRNN(50),
    Dense(1)
])
complex_rnn_model.compile(optimizer='adam', loss='mse')
```

Figure 15: RNN with 3 layers

5.2 Simple LSTM with SELU activation

```
selu_lstm_model = Sequential([
    LSTM(50, activation='selu', input_shape=(x_train_reshaped.shape[1], x_train_reshaped.shape[2])),
    Dense(1)
])
selu_lstm_model.compile(optimizer='adam', loss='mse')
```

Figure 16: LSTM with SELU function

5.3 LSTM with 0.5 dropout

```
stacked_lstm_model = Sequential([
    LSTM(50, return_sequences=True, input_shape=(x_train_reshaped.shape[1], x_train_reshaped.shape[2])),
    Dropout(0.5),
    LSTM(50),
    Dense(1)
])
stacked_lstm_model.compile(optimizer='adam', loss='mse')
```

Figure 17: 0.5 dropout

5.4 LSTM with 0.5 dropout and gradient clipping

```
# Simple LSTM with 0.5 dropout and gradient clipping
dropout_lstm_model = Sequential([
    LSTM(50, input_shape=(x_train_reshaped.shape[1], x_train_reshaped.shape[2])),
    Dropout(0.5),
    Dense(1)
])
dropout_lstm_model.compile(optimizer=Adam(clipvalue=1.0), loss='mse')
```

Figure 18: 0.5 dropout and gradient clipping