Configuration Manual

MSc Research Project
Data Analytics

Karan Sachdeva
Student ID: 18185916

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Karan Sachdeva
Student ID:	18185916
Programme:	Data Analytics
Year:	2020
Module:	MSc Research Project
Supervisor:	Dr. Catherine Mulwa
Submission Due Date:	17/08/2020
Project Title:	Configuration Manual
Word Count:	XXX
Page Count:	15

Signature:	
Date:	27th September 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.	☐

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

Configuration Manual

Karan Sachdeva
18185916

1 Introduction

The configuration manual lays out hardware specification, software requirements and different stages of implementation of research project in details.
"Predict Global Horizontal Irradiation and Beam Normal Irradiation using Machine Learning and Time Series Models"
The following sections details about system requirements and implementation steps carried out in project implementation.

2 System Configuration

This section details about system requirements and softwares required for implementation.

2.1 Hardware Requirement

- **Processor**: Intel(R) Core(TM) i7-8550 CPU @ 1.80GHz
- **RAM**: 8GB
- **System Type**: Windows OS, 64 Bit
- **GPU**: Intel(R) UHD Graphics Family
- **Storage**: 512GB SSD

2.2 Software

- **Microsoft Excel 2016**: The tool is offered by Microsoft for storing dataset spreadsheet in form of CSV(Comma Separated Value).
- **Anaconda Distribution-Jupyter Notebook**: This is an open source platform that can be downloaded from anaconda distribution website [1]. The platform support various integrated design framework(IDD) for python programming, namely, Jupyter Notebook, spyder, R studio. Exploratory Data analysis, data transformation, visualizations and model application is done using Jupyter framework using Python version (3.7). For Model implementation, custom environment is set up in Anaconda platform. Libraries of Machine learning, namely, tensorflow, Keras along

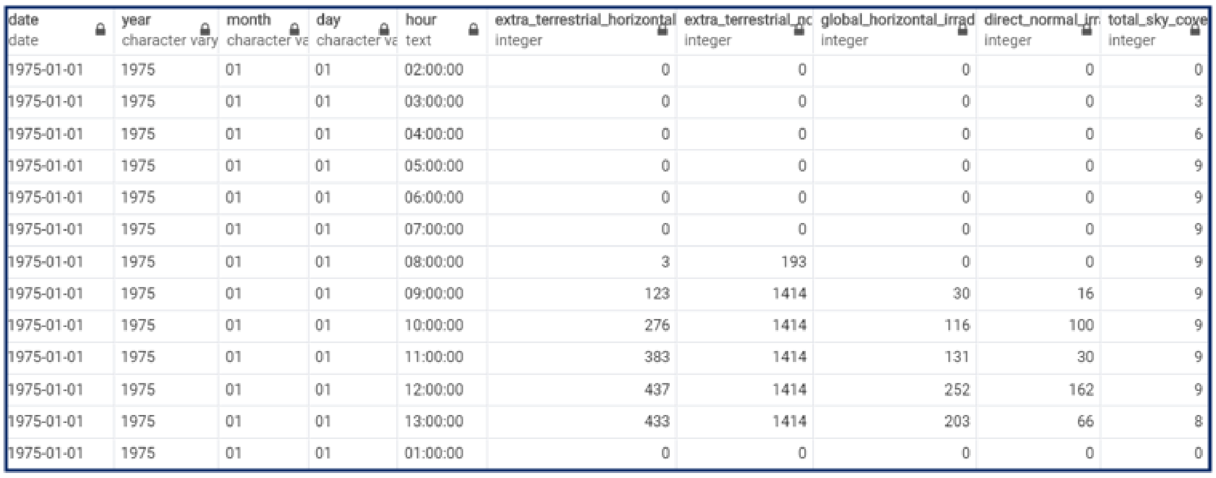
[1] https://www.anaconda.com/distribution/

with base root directories, namely, pandas, numpy are downloaded in the custom environment.

- **Postgres**: This is open source database platform that can be downloaded from Postgres website [2]. Database is used for Data cleaning, Data Merging and Data Rollup.

3 Datasource Description

Data for time series analysis is downloaded from public domain site data world `https://data.world/doe/solar-hourly-solar-dni-ghi` with weather and geographical predictor variables, released by government of China. These factors include atmospheric pressure, sky cover metrics, temperature metrics, among other factors along with GHI and BNI time series data. Data is at hourly level and it is extracted into CSV file for various locations of China for the year range (1975-2001). The screenshot below shows snapshot for raw data in imported into Postgres.



date date	year character vary	month character va	day character va	hour text	extra_terrestrial_horizontal integer	extra_terrestrial_nc integer	global_horizontal_irrad integer	direct_normal_irr integer	total_sky_cove integer
1975-01-01	1975	01	01	02:00:00	0	0	0	0	0
1975-01-01	1975	01	01	03:00:00	0	0	0	0	3
1975-01-01	1975	01	01	04:00:00	0	0	0	0	6
1975-01-01	1975	01	01	05:00:00	0	0	0	0	9
1975-01-01	1975	01	01	06:00:00	0	0	0	0	9
1975-01-01	1975	01	01	07:00:00	0	0	0	0	9
1975-01-01	1975	01	01	08:00:00	3	193	0	0	9
1975-01-01	1975	01	01	09:00:00	123	1414	30	16	9
1975-01-01	1975	01	01	10:00:00	276	1414	116	100	9
1975-01-01	1975	01	01	11:00:00	383	1414	131	30	9
1975-01-01	1975	01	01	12:00:00	437	1414	252	162	9
1975-01-01	1975	01	01	13:00:00	433	1414	203	66	8
1975-01-01	1975	01	01	01:00:00	0	0	0	0	0

Figure 1: Dataset Snapshot

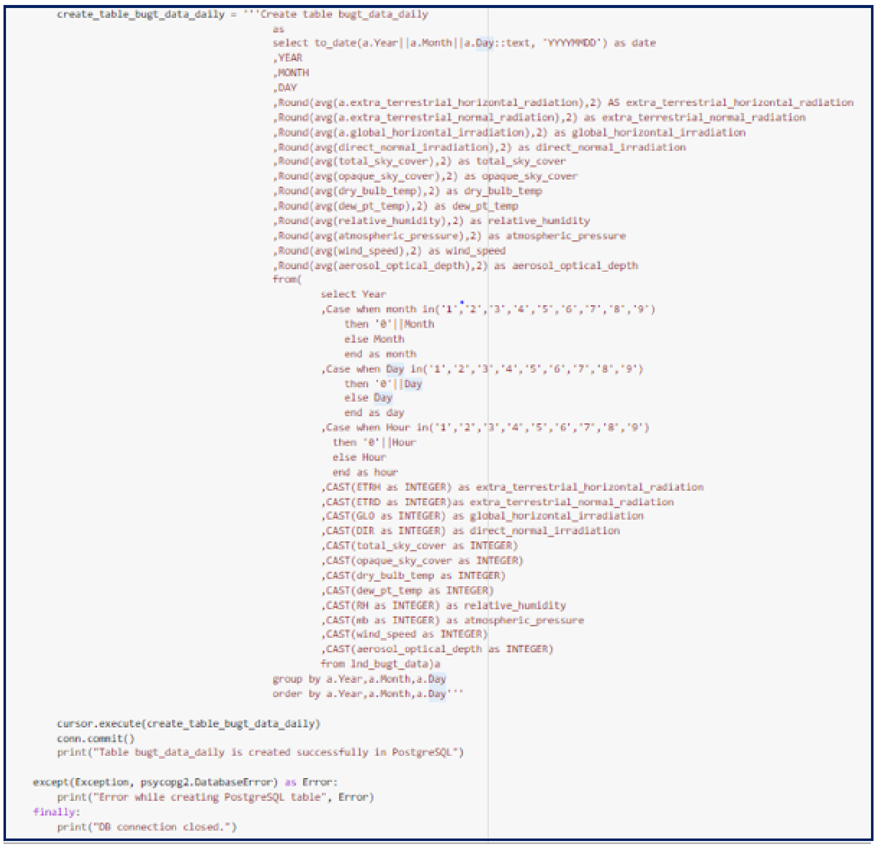
4 Data Pre-Processing and Exploratory Analysis

Data pre-Processing is undertaken in two parts. One part is undertaken in postgres database, which includes following steps

- Merging year, month, day and hour column to form date column.
- Adjustment for dates for Leap year and rolling up data from hourly grain to daily grain.

Data pre-Processing step is taken up in Jupyter using connector **psycopg** for postgres connection. Below snapshot shows code snippet for data cleaning.

[2]`https://www.postgresql.org/download/`

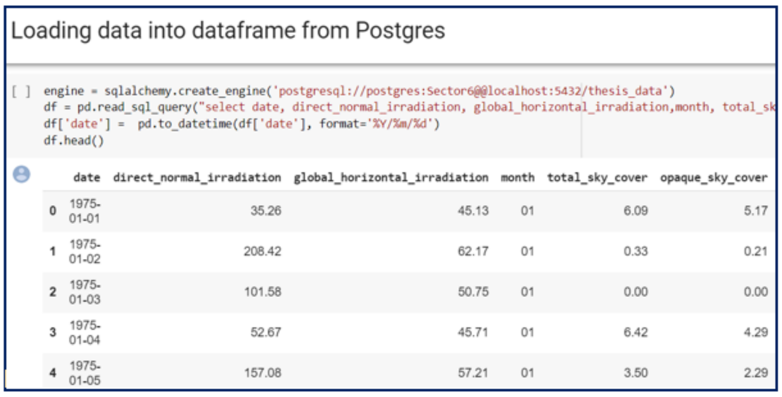


```
    create_table_bugt_data_daily = '''Create table bugt_data_daily
                                    as
                                    select to_date(a.Year||a.Month||a.Day::text, 'YYYYMMDD') as date
                                    ,YEAR
                                    ,MONTH
                                    ,DAY
                                    ,Round(avg(a.extra_terrestrial_horizontal_radiation),2) AS extra_terrestrial_horizontal_radiation
                                    ,Round(avg(a.extra_terrestrial_normal_radiation),2) as extra_terrestrial_normal_radiation
                                    ,Round(avg(a.global_horizontal_irradiation),2) as global_horizontal_irradiation
                                    ,Round(avg(direct_normal_irradiation),2) as direct_normal_irradiation
                                    ,Round(avg(total_sky_cover),2) as total_sky_cover
                                    ,Round(avg(opaque_sky_cover),2) as opaque_sky_cover
                                    ,Round(avg(dry_bulb_temp),2) as dry_bulb_temp
                                    ,Round(avg(dew_pt_temp),2) as dew_pt_temp
                                    ,Round(avg(relative_humidity),2) as relative_humidity
                                    ,Round(avg(atmospheric_pressure),2) as atmospheric_pressure
                                    ,Round(avg(wind_speed),2) as wind_speed
                                    ,Round(avg(aerosol_optical_depth),2) as aerosol_optical_depth
                                    from(
                                            select Year
                                            ,Case when month in('1','2','3','4','5','6','7','8','9')
                                                then '0'||Month
                                                else Month
                                                end as month
                                            ,Case when Day in('1','2','3','4','5','6','7','8','9')
                                                then '0'||Day
                                                else Day
                                                end as day
                                            ,Case when Hour in('1','2','3','4','5','6','7','8','9')
                                              then '0'||Hour
                                              else Hour
                                              end as hour
                                            ,CAST(ETRH as INTEGER) as extra_terrestrial_horizontal_radiation
                                            ,CAST(ETRD as INTEGER)as extra_terrestrial_normal_radiation
                                            ,CAST(GLO as INTEGER) as global_horizontal_irradiation
                                            ,CAST(DIR as INTEGER) as direct_normal_irradiation
                                            ,CAST(total_sky_cover as INTEGER)
                                            ,CAST(opaque_sky_cover as INTEGER)
                                            ,CAST(dry_bulb_temp as INTEGER)
                                            ,CAST(dew_pt_temp as INTEGER)
                                            ,CAST(RH as INTEGER) as relative_humidity
                                            ,CAST(mb as INTEGER) as atmospheric_pressure
                                            ,CAST(wind_speed as INTEGER)
                                            ,CAST(aerosol_optical_depth as INTEGER)
                                            from lnd_bugt_data)a
                                    group by a.Year,a.Month,a.Day
                                    order by a.Year,a.Month,a.Day'''

    cursor.execute(create_table_bugt_data_daily)
    conn.commit()
    print("Table bugt_data_daily is created successfully in PostgreSQL")

except(Exception, psycopg2.DatabaseError) as Error:
    print("Error while creating PostgreSQL table", Error)
finally:
    print("DB connection closed.")
```

Figure 2: Database Cleaning

5 Exploratory Data Analysis

Data is loaded into python environment for further data processing, exploratory analysis. Figure 3 below shows data loading into python using database connector **Sqlalchemy**.



Loading data into dataframe from Postgres

```
[ ] engine = sqlalchemy.create_engine('postgresql://postgres:Sector6@@localhost:5432/thesis_data')
    df = pd.read_sql_query("select date, direct_normal_irradiation, global_horizontal_irradiation,month, total_sk
    df['date'] =  pd.to_datetime(df['date'], format='%Y/%m/%d')
    df.head()
```

	date	direct_normal_irradiation	global_horizontal_irradiation	month	total_sky_cover	opaque_sky_cover
0	1975-01-01	35.26	45.13	01	6.09	5.17
1	1975-01-02	208.42	62.17	01	0.33	0.21
2	1975-01-03	101.58	50.75	01	0.00	0.00
3	1975-01-04	52.67	45.71	01	6.42	4.29
4	1975-01-05	157.08	57.21	01	3.50	2.29

Figure 3: Database Cleaning

Sections below explains further steps undertaken in exploratory data analysis and further pre-processing.

5.1 NULL value identification and Duplicate value check

Figure 4 below shows there is no NULL value present in dataset. NULL value would be represented by white lines in the figure.

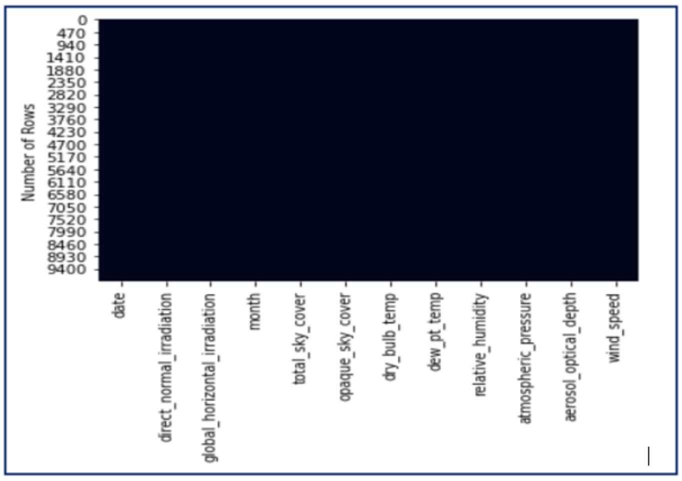
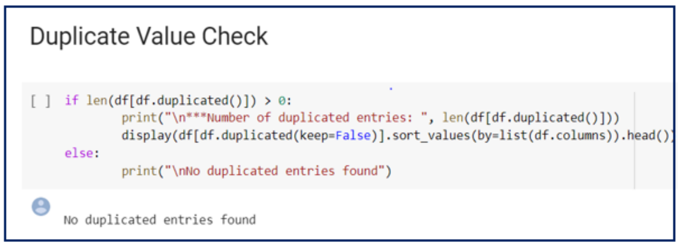


Figure 4: Database Cleaning

There are no duplicates as well in dataset as shown in code snippet in figure 5 below.



Duplicate Value Check

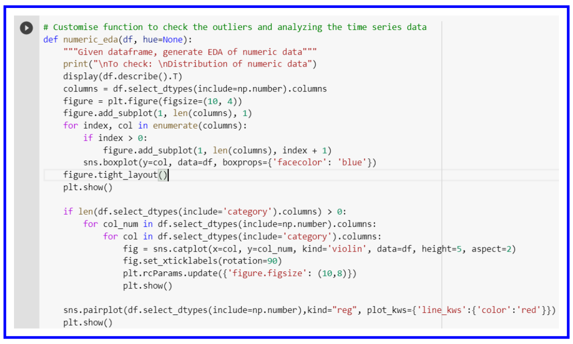
```
[ ] if len(df[df.duplicated()]) > 0:
        print("\n***Number of duplicated entries: ", len(df[df.duplicated()]))
        display(df[df.duplicated(keep=False)].sort_values(by=list(df.columns)).head()
    else:
        print("\nNo duplicated entries found")
```

```
No duplicated entries found
```

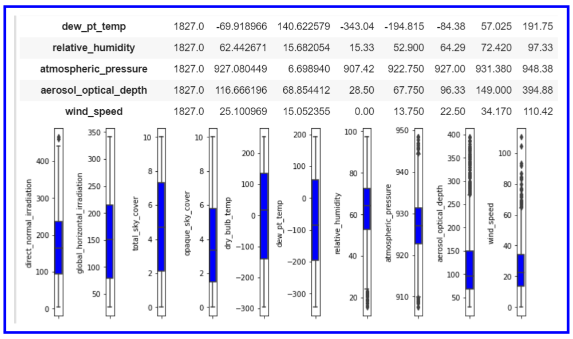
Figure 5: Database Cleaning

5.2 Numerical EDA of data

Data is checked for outliers in exogenous factors for prediction. Box plot are made for the variables for EDA. Figure 6 below shows code snippet for numerical EDA and plots for outliers.



```
# Customise function to check the outliers and analyzing the time series data
def numeric_eda(df, hue=None):
    """Given dataframe, generate EDA of numeric data"""
    print("\nTo check: \nDistribution of numeric data")
    display(df.describe().T)
    columns = df.select_dtypes(include=np.number).columns
    figure = plt.figure(figsize=(10, 4))
    figure.add_subplot(1, len(columns), 1)
    for index, col in enumerate(columns):
        if index > 0:
            figure.add_subplot(1, len(columns), index + 1)
        sns.boxplot(y=col, data=df, boxprops={'facecolor': 'blue'})
    figure.tight_layout()
    plt.show()

    if len(df.select_dtypes(include='category').columns) > 0:
        for col_num in df.select_dtypes(include=np.number).columns:
            for col in df.select_dtypes(include='category').columns:
                fig = sns.catplot(x=col, y=col_num, kind='violin', data=df, height=5, aspect=2)
                fig.set_xticklabels(rotation=90)
                plt.rcParams.update({'figure.figsize': (10,8)})
                plt.show()

    sns.pairplot(df.select_dtypes(include=np.number),kind="reg", plot_kws={'line_kws':{'color':'red'}})
    plt.show()
```



dew_pt_temp	1827.0	-69.918966	140.622579	-343.04	-194.815	-84.38	57.025	191.75
relative_humidity	1827.0	62.442671	15.682054	15.33	52.900	64.29	72.420	97.33
atmospheric_pressure	1827.0	927.080449	6.698940	907.42	922.750	927.00	931.380	948.38
aerosol_optical_depth	1827.0	116.666196	68.854412	28.50	67.750	96.33	149.000	394.88
wind_speed	1827.0	25.100969	15.052355	0.00	13.750	22.50	34.170	110.42

Figure 6: Evaluation Metrics Hybrid ARIMA for BNI Prediction

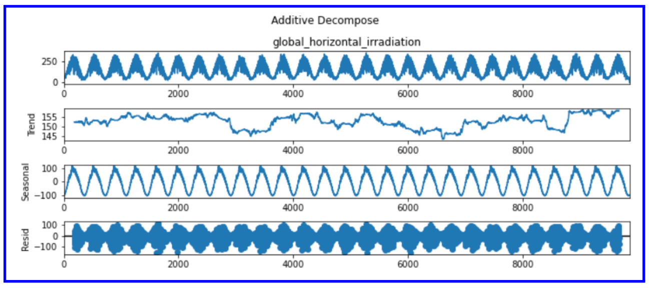
We can see from figure above, atmospheric pressure have a few outliers while aerosol optical depth and wind speed have considerable outliers in data.

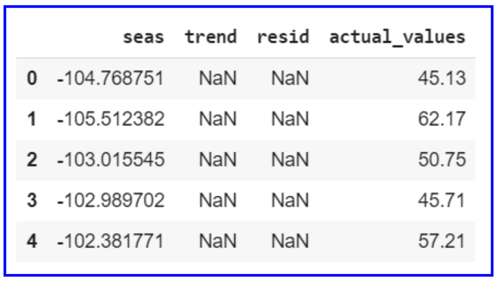
5.3 Analysis for Seasonality, Trend and Stationarity

Time series data for GHI and BNI is analyzed for Stationarity, Seasonality and Trend. Subsections below shows checks employed for Seasonality, Trend and Stationarity.

5.3.1 Check for Seasonality and Trend

Data is break down using additive decompose. Figure 7 below shows additive decompose for GHI along with decompose metrics for the GHI time series data. There is presence of Seasonality but there is no trend present in plots for trend

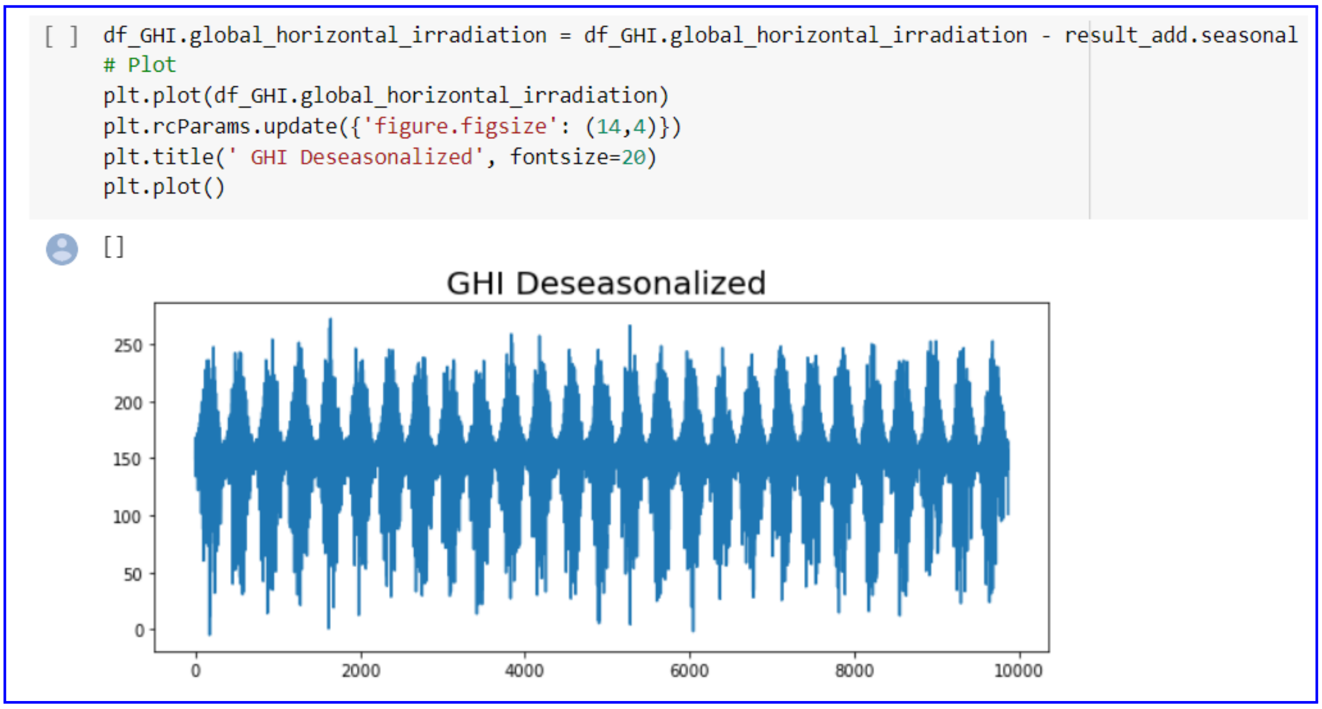




	seas	trend	resid	actual_values
0	-104.768751	NaN	NaN	45.13
1	-105.512382	NaN	NaN	62.17
2	-103.015545	NaN	NaN	50.75
3	-102.989702	NaN	NaN	45.71
4	-102.381771	NaN	NaN	57.21

Figure 7: Evaluation Metrics Hybrid ARIMA for BNI Prediction

Data for GHI is deseasonalized by subtracting the seasonal component from the data. Below snapshot in figure 8 shows deseasonalized GHI data.

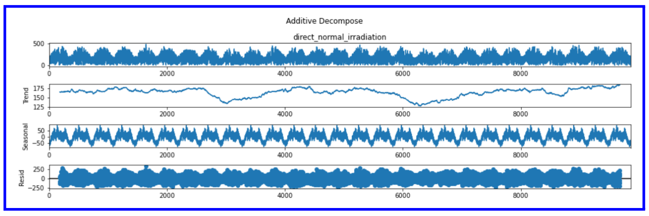


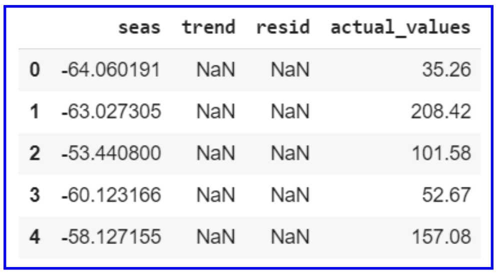
```
[ ]  df_GHI.global_horizontal_irradiation = df_GHI.global_horizontal_irradiation - result_add.seasonal
     # Plot
     plt.plot(df_GHI.global_horizontal_irradiation)
     plt.rcParams.update({'figure.figsize': (14,4)})
     plt.title(' GHI Deseasonalized', fontsize=20)
     plt.plot()
```

```
[]
```

Figure 8: Database Cleaning

Similarly, Figure 9 below shows seasonal decompose for BNI along with decompose metrics for the BNI time series data. There is seasonality in time series data but no presence of trend in dataset.

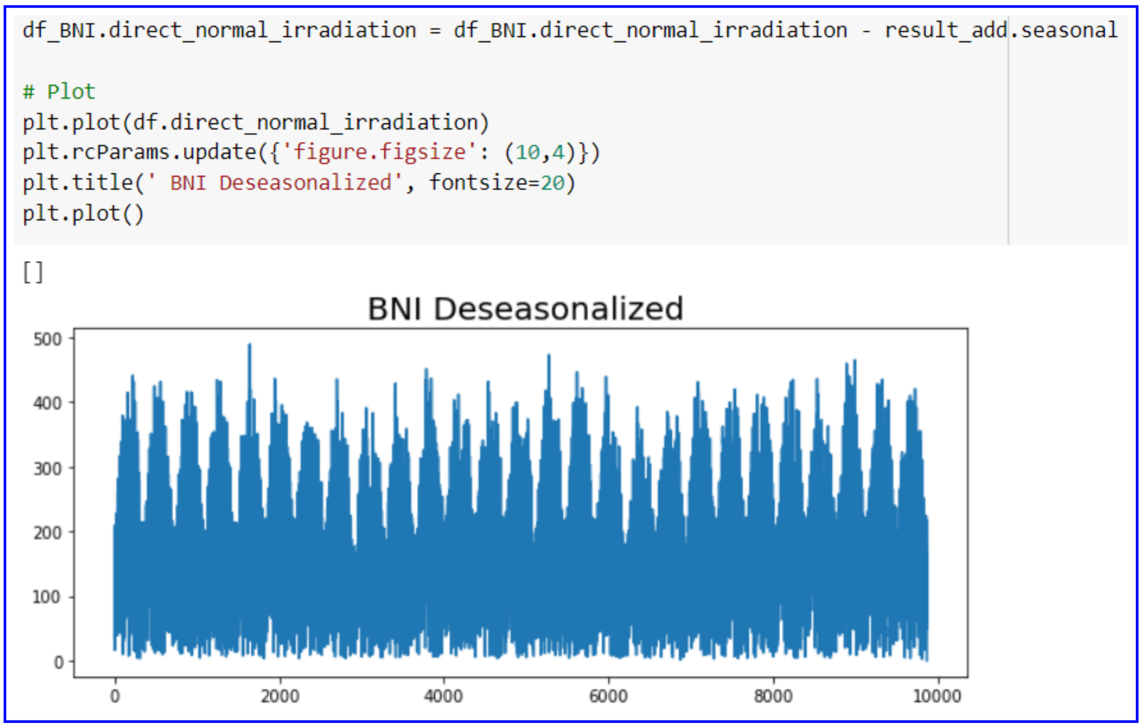




	seas	trend	resid	actual_values
0	-64.060191	NaN	NaN	35.26
1	-63.027305	NaN	NaN	208.42
2	-53.440800	NaN	NaN	101.58
3	-60.123166	NaN	NaN	52.67
4	-58.127155	NaN	NaN	157.08

Figure 9: Deseasonalized GHI

Data for BNI is deseasonalized by subtracting the seasonal component from the data. Below snapshot in figure 10 shows deseasonalized GHI data.



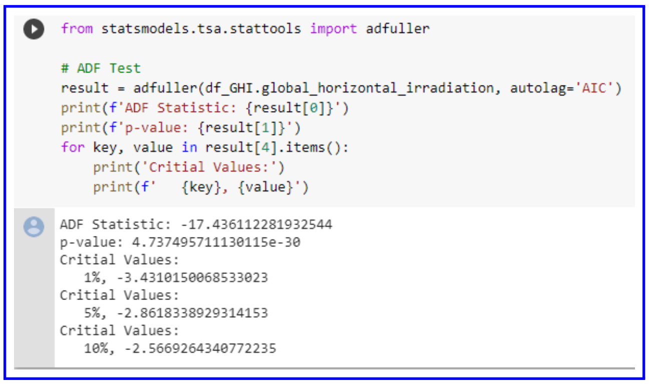
```
df_BNI.direct_normal_irradiation = df_BNI.direct_normal_irradiation - result_add.seasonal

# Plot
plt.plot(df.direct_normal_irradiation)
plt.rcParams.update({'figure.figsize': (10,4)})
plt.title(' BNI Deseasonalized', fontsize=20)
plt.plot()
```

```
[]
```

Figure 10: Deseasonalized BNI

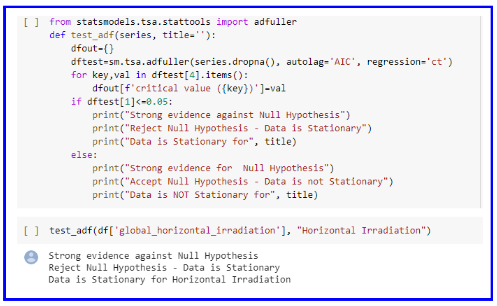
5.3.2 Check for Stationarity

Ad-fuller and variance test is conducted to check for stationarity in dataset. The data is deemed to be stationary in both the test. Below figure 11 and figure 12 adfuller test for both GHI and BNI. As can be seen from the test results that data is stationary for GHI and BNI time series data.



```
from statsmodels.tsa.stattools import adfuller

# ADF Test
result = adfuller(df_GHI.global_horizontal_irradiation, autolag='AIC')
print(f'ADF Statistic: {result[0]}')
print(f'p-value: {result[1]}')
for key, value in result[4].items():
    print('Critial Values:')
    print(f'   {key}, {value}')
```

```
ADF Statistic: -17.436112281932544
p-value: 4.737495711130115e-30
Critial Values:
   1%, -3.4310150068533023
Critial Values:
   5%, -2.8618338929314153
Critial Values:
   10%, -2.5669264340772235
```

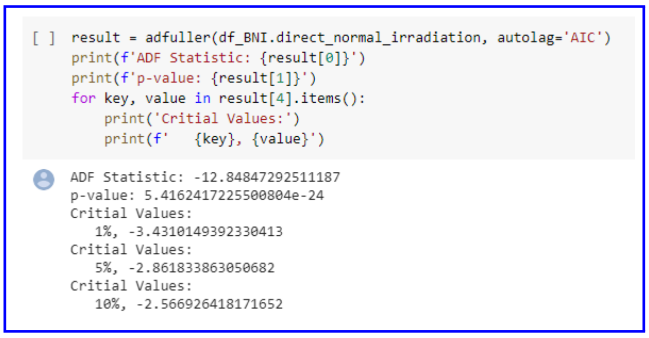


```
from statsmodels.tsa.stattools import adfuller
def test_adf(series, title=''):
    dfout={}
    dftest=sm.tsa.adfuller(series.dropna(), autolag='AIC', regression='ct')
    for key,val in dftest[4].items():
        dfout[f'critical value ({key})']=val
    if dftest[1]<=0.05:
        print("Strong evidence against Null Hypothesis")
        print("Reject Null Hypothesis - Data is Stationary")
        print("Data is Stationary for", title)
    else:
        print("Strong evidence for  Null Hypothesis")
        print("Accept Null Hypothesis - Data is not Stationary")
        print("Data is NOT Stationary for", title)
```

```
test_adf(df['global_horizontal_irradiation'], "Horizontal Irradiation")
```

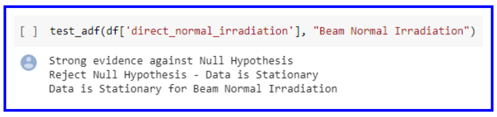
```
Strong evidence against Null Hypothesis
Reject Null Hypothesis - Data is Stationary
Data is Stationary for Horizontal Irradiation
```

Figure 11: ADF Test GHI



```
result = adfuller(df_BNI.direct_normal_irradiation, autolag='AIC')
print(f'ADF Statistic: {result[0]}')
print(f'p-value: {result[1]}')
for key, value in result[4].items():
    print('Critial Values:')
    print(f'   {key}, {value}')
```

```
ADF Statistic: -12.84847292511187
p-value: 5.4162417225500804e-24
Critial Values:
   1%, -3.4310149392330413
Critial Values:
   5%, -2.861833863050682
Critial Values:
   10%, -2.566926418171652
```

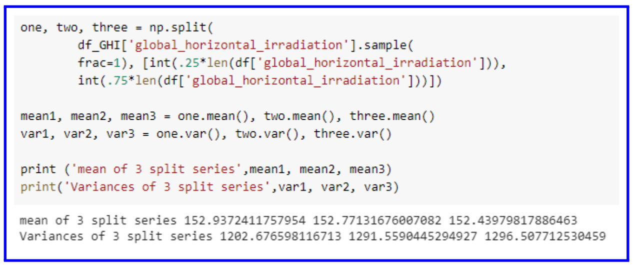


```
test_adf(df['direct_normal_irradiation'], "Beam Normal Irradiation")
```

```
Strong evidence against Null Hypothesis
Reject Null Hypothesis - Data is Stationary
Data is Stationary for Beam Normal Irradiation
```

Figure 12: ADF test BNI

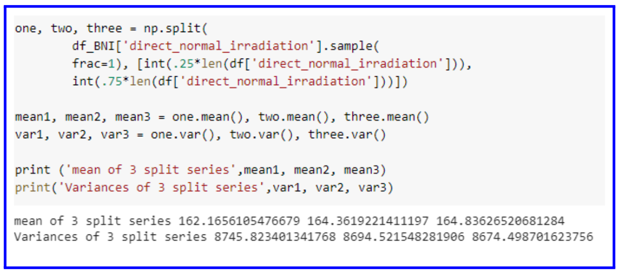
Test for variance is shown below. Data is split up into 3 equal segments and variance of each segment is approximately same. Figure 13 and figure 14 shows variance for GHI and BNI respectively. It can be seen that variance of three splits is approximately same. Hence, data is stationary.



```
one, two, three = np.split(
        df_GHI['global_horizontal_irradiation'].sample(
        frac=1), [int(.25*len(df['global_horizontal_irradiation'])),
        int(.75*len(df['global_horizontal_irradiation']))])

mean1, mean2, mean3 = one.mean(), two.mean(), three.mean()
var1, var2, var3 = one.var(), two.var(), three.var()

print ('mean of 3 split series',mean1, mean2, mean3)
print('Variances of 3 split series',var1, var2, var3)
```

```
mean of 3 split series 152.9372411757954 152.77131676007082 152.43979817886463
Variances of 3 split series 1202.676598116713 1291.5590445294927 1296.507712530459
```

Figure 13: Variance Test GHI



```
one, two, three = np.split(
        df_BNI['direct_normal_irradiation'].sample(
        frac=1), [int(.25*len(df['direct_normal_irradiation'])),
        int(.75*len(df['direct_normal_irradiation']))])

mean1, mean2, mean3 = one.mean(), two.mean(), three.mean()
var1, var2, var3 = one.var(), two.var(), three.var()

print ('mean of 3 split series',mean1, mean2, mean3)
print('Variances of 3 split series',var1, var2, var3)
```

```
mean of 3 split series 162.1656105476679 164.3619221411197 164.83626520681284
Variances of 3 split series 8745.823401341768 8694.521548281906 8674.498701623756
```

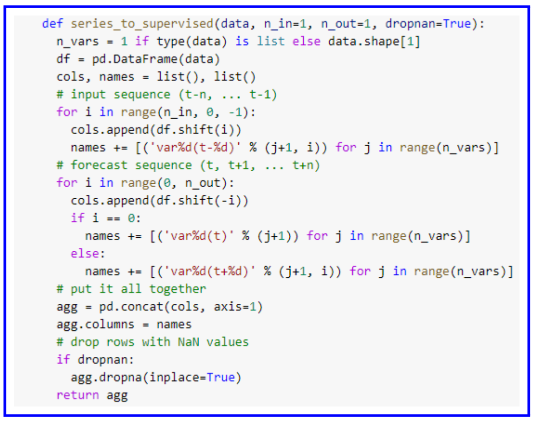
Figure 14: Variance Test BNI

6 Data Modeling for GHI

This section describes modeling techniques employed in GHI prediction. Section would be arranged in order of model implementation and evaluation metrics.

6.1 Data Transformation and Normalization

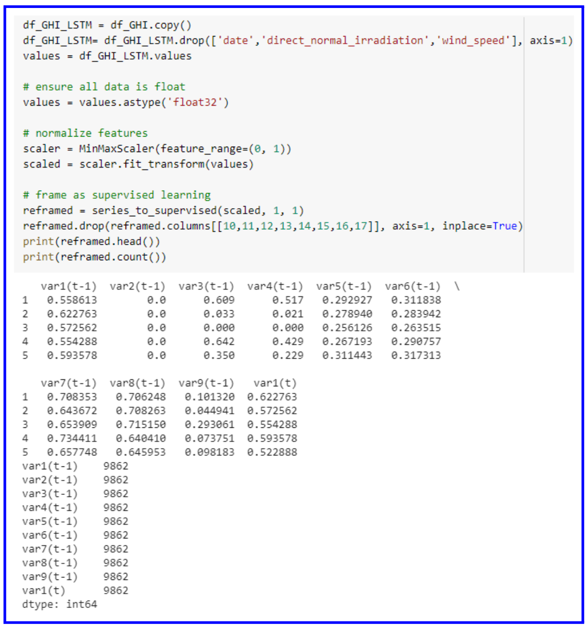
Data is converted into machine learning dataset by using pandas shift function or performing lag on dataset. Lag considered for research is 1. Figure 15 below shows function for data transformation.



```python
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
  n_vars = 1 if type(data) is list else data.shape[1]
  df = pd.DataFrame(data)
  cols, names = list(), list()
  # input sequence (t-n, ... t-1)
  for i in range(n_in, 0, -1):
    cols.append(df.shift(i))
    names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
  # forecast sequence (t, t+1, ... t+n)
  for i in range(0, n_out):
    cols.append(df.shift(-i))
    if i == 0:
      names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
    else:
      names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
  # put it all together
  agg = pd.concat(cols, axis=1)
  agg.columns = names
  # drop rows with NaN values
  if dropnan:
    agg.dropna(inplace=True)
  return agg
```

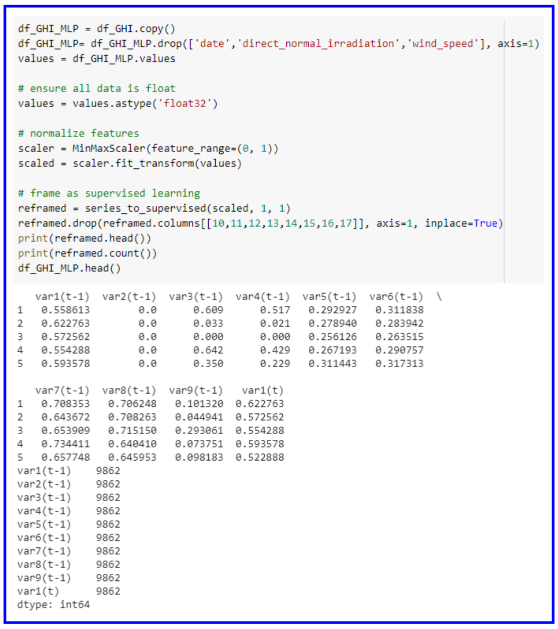
Figure 15: Pandas shift function for Data Transformation

Figure 16 shows data transformation for LSTM and MLP model using pandas shift function mentioned above.



```python
df_GHI_LSTM = df_GHI.copy()
df_GHI_LSTM= df_GHI_LSTM.drop(['date','direct_normal_irradiation','wind_speed'], axis=1)
values = df_GHI_LSTM.values

# ensure all data is float
values = values.astype('float32')

# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)

# frame as supervised learning
reframed = series_to_supervised(scaled, 1, 1)
reframed.drop(reframed.columns[[10,11,12,13,14,15,16,17]], axis=1, inplace=True)
print(reframed.head())
print(reframed.count())
```

```
   var1(t-1)  var2(t-1)  var3(t-1)  var4(t-1)  var5(t-1)  var6(t-1)  \
1   0.558613        0.0      0.609      0.517   0.292927   0.311838
2   0.622763        0.0      0.033      0.021   0.278940   0.283942
3   0.572562        0.0      0.000      0.000   0.256126   0.263515
4   0.554288        0.0      0.642      0.429   0.267193   0.290757
5   0.593578        0.0      0.350      0.229   0.311443   0.317313

   var7(t-1)  var8(t-1)  var9(t-1)   var1(t)
1   0.708353   0.706248   0.101320  0.622763
2   0.643672   0.708263   0.044941  0.572562
3   0.653909   0.715150   0.293061  0.554288
4   0.734411   0.640410   0.073751  0.593578
5   0.657748   0.645953   0.098183  0.522888
var1(t-1)    9862
var2(t-1)    9862
var3(t-1)    9862
var4(t-1)    9862
var5(t-1)    9862
var6(t-1)    9862
var7(t-1)    9862
var8(t-1)    9862
var9(t-1)    9862
var1(t)      9862
dtype: int64
```



```python
df_GHI_MLP = df_GHI.copy()
df_GHI_MLP= df_GHI_MLP.drop(['date','direct_normal_irradiation','wind_speed'], axis=1)
values = df_GHI_MLP.values

# ensure all data is float
values = values.astype('float32')

# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)

# frame as supervised learning
reframed = series_to_supervised(scaled, 1, 1)
reframed.drop(reframed.columns[[10,11,12,13,14,15,16,17]], axis=1, inplace=True)
print(reframed.head())
print(reframed.count())
df_GHI_MLP.head()
```

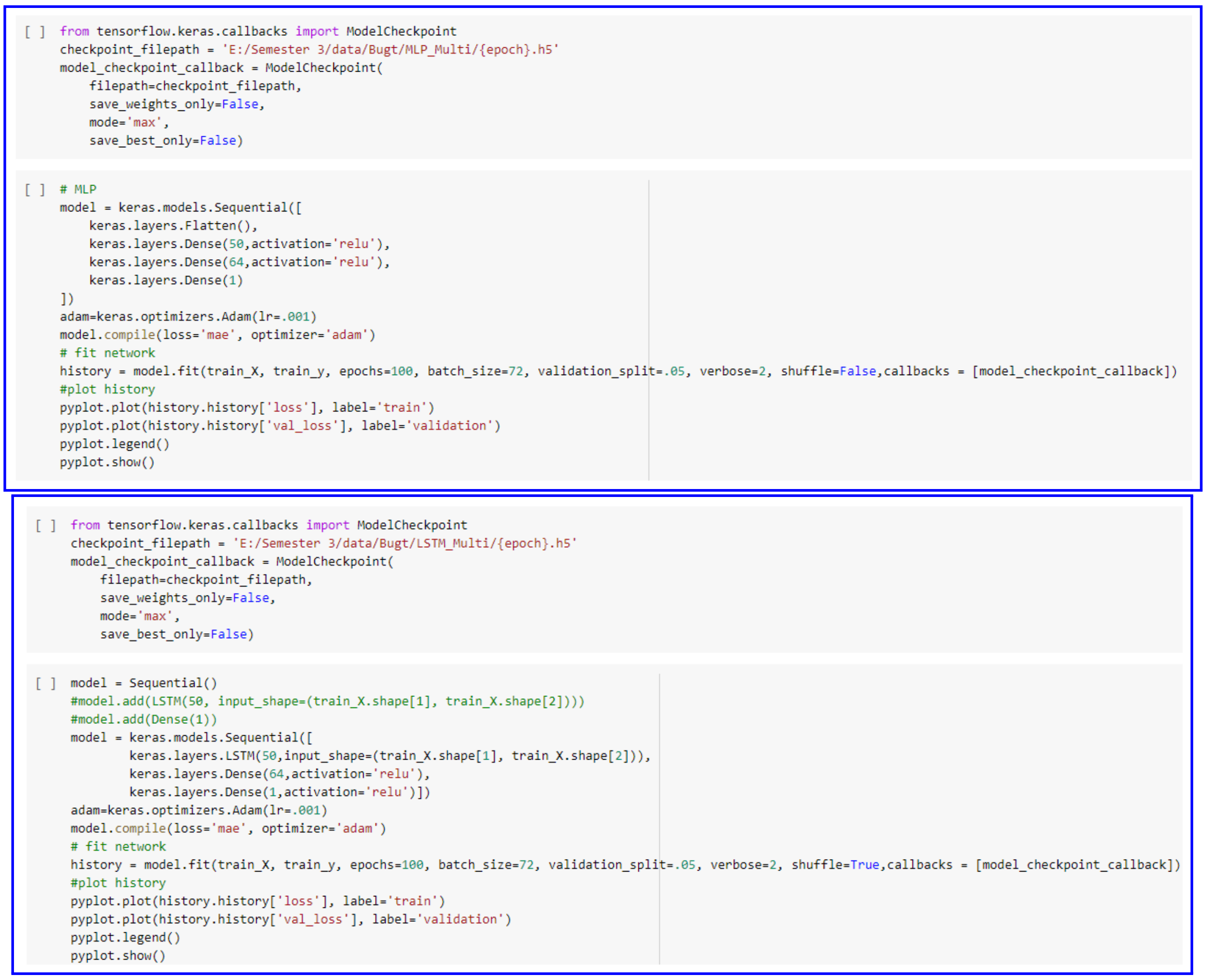
```
   var1(t-1)  var2(t-1)  var3(t-1)  var4(t-1)  var5(t-1)  var6(t-1)  \
1   0.558613        0.0      0.609      0.517   0.292927   0.311838
2   0.622763        0.0      0.033      0.021   0.278940   0.283942
3   0.572562        0.0      0.000      0.000   0.256126   0.263515
4   0.554288        0.0      0.642      0.429   0.267193   0.290757
5   0.593578        0.0      0.350      0.229   0.311443   0.317313

   var7(t-1)  var8(t-1)  var9(t-1)   var1(t)
1   0.708353   0.706248   0.101320  0.622763
2   0.643672   0.708263   0.044941  0.572562
3   0.653909   0.715150   0.293061  0.554288
4   0.734411   0.640410   0.073751  0.593578
5   0.657748   0.645953   0.098183  0.522888
var1(t-1)    9862
var2(t-1)    9862
var3(t-1)    9862
var4(t-1)    9862
var5(t-1)    9862
var6(t-1)    9862
var7(t-1)    9862
var8(t-1)    9862
var9(t-1)    9862
var1(t)      9862
dtype: int64
```

Figure 16: Data Transformation LSTM and MLP

Also, from figure 16 we can see data is normalized for time series and kept between 0 and 1 using scikit learn library MinMax Scaler function. This is mandatory step for input to neural network models.

6.2 Model Implementation for LSTM and MLP

Figure 17 shows model implementation for LSTM and MLP. For LSTM implementation, 2 dense layer is added to a layer of LSTM layer. Number of neurons is 50,64 and 1 respectively in LSTM and 2 dense layers. For MLP, flatten layer is added on top of MLP layers to make three dimensional input into two dimensional as same data input is used as that for LSTM model, which needs three dimensional input for model implementation. For hyperparameter tuning, callback function is used to select best model from 100 epochs and weights are initialized for best model to predict evaluation metrics. Also learning rate is altered to improve the model metrics.



```
[ ] from tensorflow.keras.callbacks import ModelCheckpoint
    checkpoint_filepath = 'E:/Semester 3/data/Bugt/MLP_Multi/{epoch}.h5'
    model_checkpoint_callback = ModelCheckpoint(
        filepath=checkpoint_filepath,
        save_weights_only=False,
        mode='max',
        save_best_only=False)
```

```
[ ] # MLP
    model = keras.models.Sequential([
        keras.layers.Flatten(),
        keras.layers.Dense(50,activation='relu'),
        keras.layers.Dense(64,activation='relu'),
        keras.layers.Dense(1)
    ])
    adam=keras.optimizers.Adam(lr=.001)
    model.compile(loss='mae', optimizer='adam')
    # fit network
    history = model.fit(train_X, train_y, epochs=100, batch_size=72, validation_split=.05, verbose=2, shuffle=False,callbacks = [model_checkpoint_callback])
    #plot history
    pyplot.plot(history.history['loss'], label='train')
    pyplot.plot(history.history['val_loss'], label='validation')
    pyplot.legend()
    pyplot.show()
```

```
[ ] from tensorflow.keras.callbacks import ModelCheckpoint
    checkpoint_filepath = 'E:/Semester 3/data/Bugt/LSTM_Multi/{epoch}.h5'
    model_checkpoint_callback = ModelCheckpoint(
        filepath=checkpoint_filepath,
        save_weights_only=False,
        mode='max',
        save_best_only=False)
```

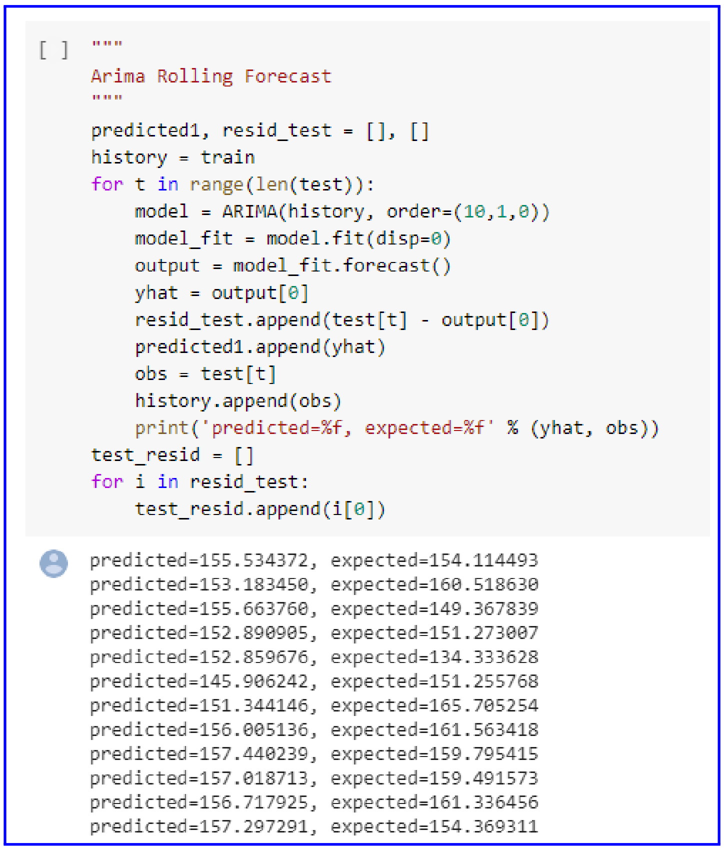
```
[ ] model = Sequential()
    #model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
    #model.add(Dense(1))
    model = keras.models.Sequential([
            keras.layers.LSTM(50,input_shape=(train_X.shape[1], train_X.shape[2])),
            keras.layers.Dense(64,activation='relu'),
            keras.layers.Dense(1,activation='relu')])
    adam=keras.optimizers.Adam(lr=.001)
    model.compile(loss='mae', optimizer='adam')
    # fit network
    history = model.fit(train_X, train_y, epochs=100, batch_size=72, validation_split=.05, verbose=2, shuffle=True,callbacks = [model_checkpoint_callback])
    #plot history
    pyplot.plot(history.history['loss'], label='train')
    pyplot.plot(history.history['val_loss'], label='validation')
    pyplot.legend()
    pyplot.show()
```

Figure 17: Model Implementation LSTM and MLP

Similarly, uni-Variate version of MLP and LSTM is implemented as well for research. The metrics of Multi-variate is better in comparison to uni-variate.

6.3 Model Implementation ARIMA

Figure 18 below shows code implementation for ARIMA. The model is executed for p,d,q of 10,0,0 as obtained by auto ARIMA function.



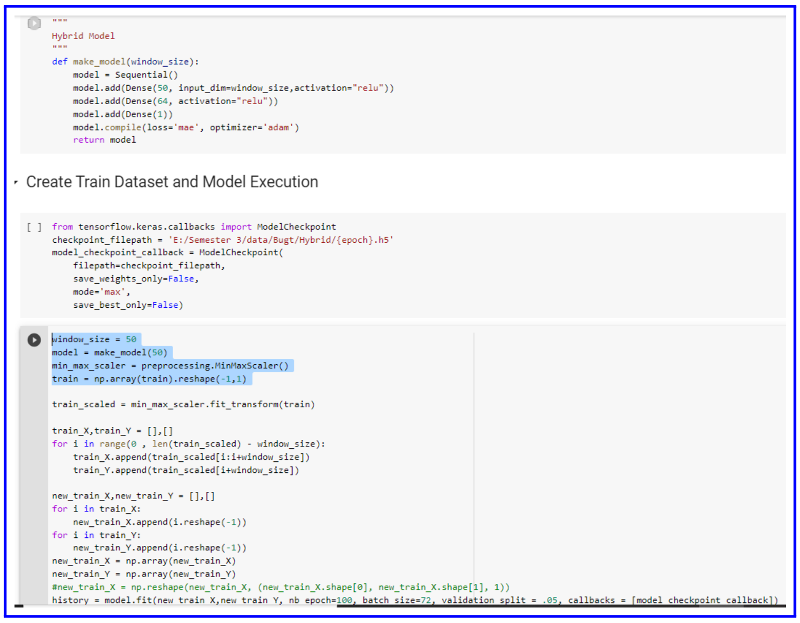
```
"""
Arima Rolling Forecast
"""
predicted1, resid_test = [], []
history = train
for t in range(len(test)):
    model = ARIMA(history, order=(10,1,0))
    model_fit = model.fit(disp=0)
    output = model_fit.forecast()
    yhat = output[0]
    resid_test.append(test[t] - output[0])
    predicted1.append(yhat)
    obs = test[t]
    history.append(obs)
    print('predicted=%f, expected=%f' % (yhat, obs))
test_resid = []
for i in resid_test:
    test_resid.append(i[0])
```

```
predicted=155.534372, expected=154.114493
predicted=153.183450, expected=160.518630
predicted=155.663760, expected=149.367839
predicted=152.890905, expected=151.273007
predicted=152.859676, expected=134.333628
predicted=145.906242, expected=151.255768
predicted=151.344146, expected=165.705254
predicted=156.005136, expected=161.563418
predicted=157.440239, expected=159.795415
predicted=157.018713, expected=159.491573
predicted=156.717925, expected=161.336456
predicted=157.297291, expected=154.369311
```

Figure 18: Model Implementation ARIMA

6.4 Model Implementation for Hybrid model

The figure 19 below clearly shows model steps for Hybrid ARIMA ANN model. Residuals obtained from ARIMA model is used as input to ANN model for prediction.



```
"""
Hybrid Model
"""
def make_model(window_size):
    model = Sequential()
    model.add(Dense(50, input_dim=window_size,activation="relu"))
    model.add(Dense(64, activation="relu"))
    model.add(Dense(1))
    model.compile(loss='mae', optimizer='adam')
    return model
```

Create Train Dataset and Model Execution

```
from tensorflow.keras.callbacks import ModelCheckpoint
checkpoint_filepath = 'E:/Semester 3/data/Bugt/Hybrid/{epoch}.h5'
model_checkpoint_callback = ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=False,
    mode='max',
    save_best_only=False)
```

```
window_size = 50
model = make_model(50)
min_max_scaler = preprocessing.MinMaxScaler()
train = np.array(train).reshape(-1,1)

train_scaled = min_max_scaler.fit_transform(train)

train_X,train_Y = [],[]
for i in range(0 , len(train_scaled) - window_size):
    train_X.append(train_scaled[i:i+window_size])
    train_Y.append(train_scaled[i+window_size])

new_train_X,new_train_Y = [],[]
for i in train_X:
    new_train_X.append(i.reshape(-1))
for i in train_Y:
    new_train_Y.append(i.reshape(-1))
new_train_X = np.array(new_train_X)
new_train_Y = np.array(new_train_Y)
#new_train_X = np.reshape(new_train_X, (new_train_X.shape[0], new_train_X.shape[1], 1))
history = model.fit(new_train_X,new_train_Y, nb_epoch=100, batch_size=72, validation_split = .05, callbacks = [model_checkpoint_callback])
```

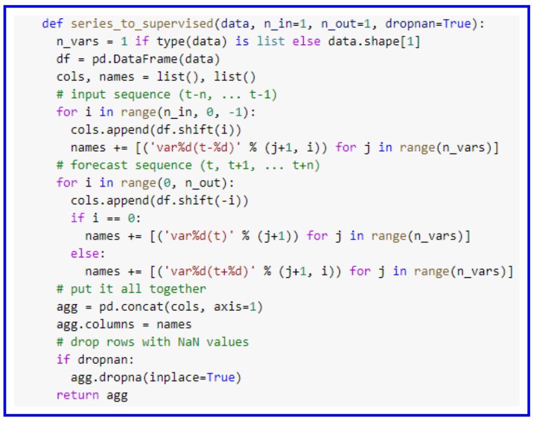
Figure 19: Model Implementation Hybrid ARIMA

7 Data Modeling for BNI

This section describes modeling techniques employed in BNI prediction. Section would be arranged in order of model implementation and evaluation metrics.

7.1 Data Transformation and Normalization

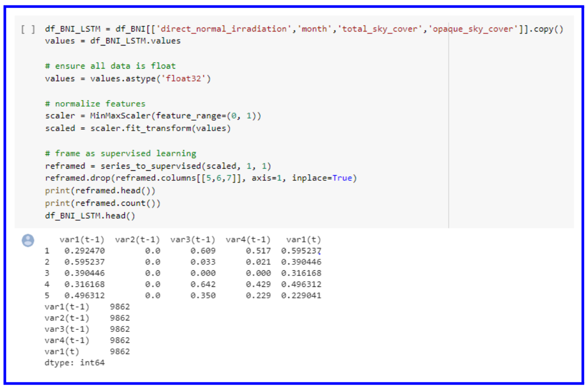
Data is converted into machine learning dataset by using pandas shift function or performing lag on dataset. Lag considered for research is 1. Figure 15 below shows function for data transformation.



```python
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
  n_vars = 1 if type(data) is list else data.shape[1]
  df = pd.DataFrame(data)
  cols, names = list(), list()
  # input sequence (t-n, ... t-1)
  for i in range(n_in, 0, -1):
    cols.append(df.shift(i))
    names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
  # forecast sequence (t, t+1, ... t+n)
  for i in range(0, n_out):
    cols.append(df.shift(-i))
    if i == 0:
      names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
    else:
      names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
  # put it all together
  agg = pd.concat(cols, axis=1)
  agg.columns = names
  # drop rows with NaN values
  if dropnan:
    agg.dropna(inplace=True)
  return agg
```

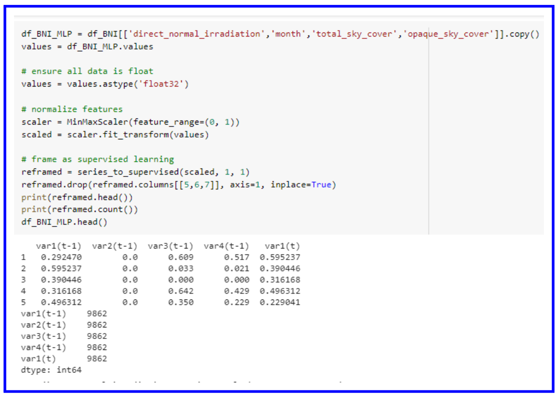
Figure 20: Pandas shift function for Data Transformation

Figure 16 shows data transformation for LSTM and MLP model using pandas shift function mentioned above.



```python
[ ] df_BNI_LSTM = df_BNI[['direct_normal_irradiation','month','total_sky_cover','opaque_sky_cover']].copy()
    values = df_BNI_LSTM.values

    # ensure all data is float
    values = values.astype('float32')

    # normalize features
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled = scaler.fit_transform(values)

    # frame as supervised learning
    reframed = series_to_supervised(scaled, 1, 1)
    reframed.drop(reframed.columns[[5,6,7]], axis=1, inplace=True)
    print(reframed.head())
    print(reframed.count())
    df_BNI_LSTM.head()
```

```
   var1(t-1)  var2(t-1)  var3(t-1)  var4(t-1)   var1(t)
1   0.292470        0.0      0.609      0.517  0.595237
2   0.595237        0.0      0.033      0.021  0.390446
3   0.390446        0.0      0.000      0.000  0.316168
4   0.316168        0.0      0.642      0.429  0.496312
5   0.496312        0.0      0.350      0.229  0.229041
var1(t-1)    9862
var2(t-1)    9862
var3(t-1)    9862
var4(t-1)    9862
var1(t)      9862
dtype: int64
```



```python
df_BNI_MLP = df_BNI[['direct_normal_irradiation','month','total_sky_cover','opaque_sky_cover']].copy()
values = df_BNI_MLP.values

# ensure all data is float
values = values.astype('float32')

# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)

# frame as supervised learning
reframed = series_to_supervised(scaled, 1, 1)
reframed.drop(reframed.columns[[5,6,7]], axis=1, inplace=True)
print(reframed.head())
print(reframed.count())
df_BNI_MLP.head()
```

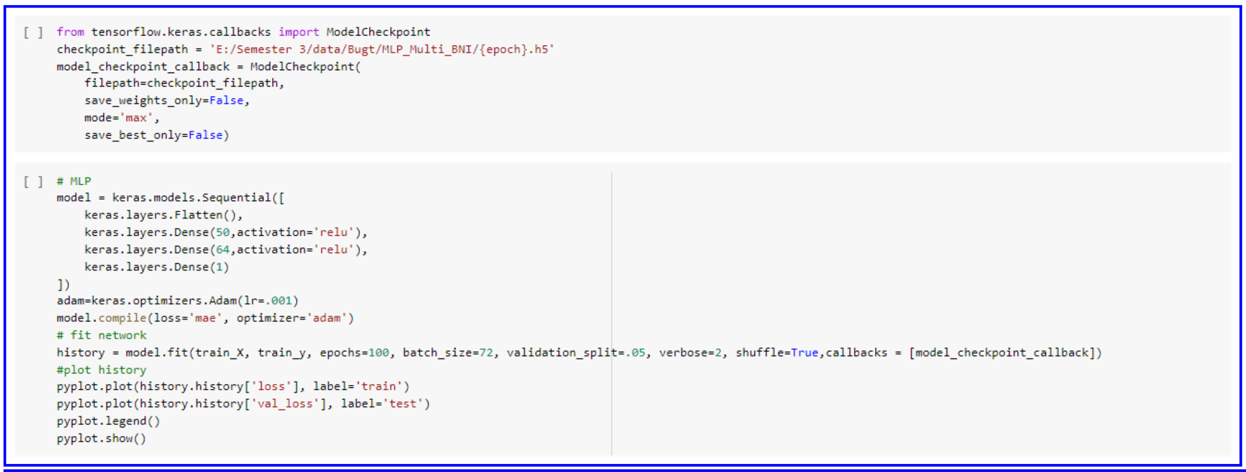
```
   var1(t-1)  var2(t-1)  var3(t-1)  var4(t-1)   var1(t)
1   0.292470        0.0      0.609      0.517  0.595237
2   0.595237        0.0      0.033      0.021  0.390446
3   0.390446        0.0      0.000      0.000  0.316168
4   0.316168        0.0      0.642      0.429  0.496312
5   0.496312        0.0      0.350      0.229  0.229041
var1(t-1)    9862
var2(t-1)    9862
var3(t-1)    9862
var4(t-1)    9862
var1(t)      9862
dtype: int64
```

Figure 21: Data Transformation LSTM and MLP

Also, from figure 16 we can see data is normalized for time series and kept between 0 and 1 using scikit learn library MinMax Scaler function. This is mandatory step for input to neural network models.

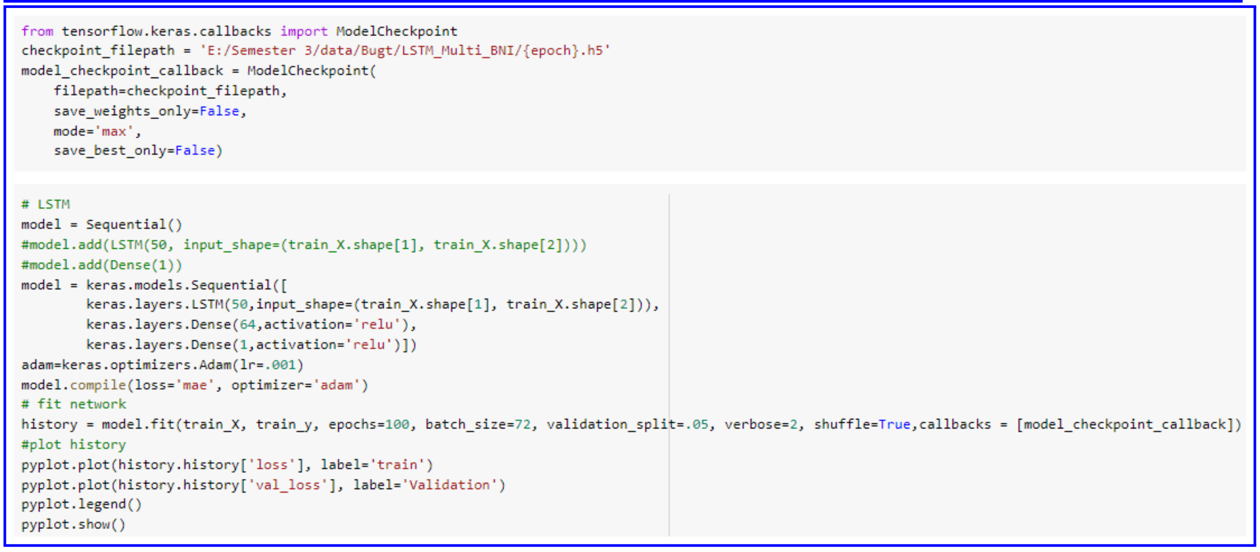
7.2 Model Implementation for LSTM and MLP

Figure 17 shows model implementation for LSTM and MLP. For LSTM implementation, 2 dense layer is added to a layer of LSTM layer. Number of neurons is 50,64 and 1 respectively in LSTM and 2 dense layers. For MLP, flatten layer is added on top of MLP layers to make three dimensional input into two dimensional as same data input is used as that for LSTM model, which needs three dimensional input for model implementation. For hyperparameter tuning, callback function is used to select best model from 100 epochs and weights are initialized for best model to predict evaluation metrics. Also learning rate is altered to improve the model metrics.



```
[ ] from tensorflow.keras.callbacks import ModelCheckpoint
    checkpoint_filepath = 'E:/Semester 3/data/Bugt/MLP_Multi_BNI/{epoch}.h5'
    model_checkpoint_callback = ModelCheckpoint(
        filepath=checkpoint_filepath,
        save_weights_only=False,
        mode='max',
        save_best_only=False)
```

```
[ ] # MLP
    model = keras.models.Sequential([
        keras.layers.Flatten(),
        keras.layers.Dense(50,activation='relu'),
        keras.layers.Dense(64,activation='relu'),
        keras.layers.Dense(1)
    ])
    adam=keras.optimizers.Adam(lr=.001)
    model.compile(loss='mae', optimizer='adam')
    # fit network
    history = model.fit(train_X, train_y, epochs=100, batch_size=72, validation_split=.05, verbose=2, shuffle=True,callbacks = [model_checkpoint_callback])
    #plot history
    pyplot.plot(history.history['loss'], label='train')
    pyplot.plot(history.history['val_loss'], label='test')
    pyplot.legend()
    pyplot.show()
```



```
from tensorflow.keras.callbacks import ModelCheckpoint
checkpoint_filepath = 'E:/Semester 3/data/Bugt/LSTM_Multi_BNI/{epoch}.h5'
model_checkpoint_callback = ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=False,
    mode='max',
    save_best_only=False)
```

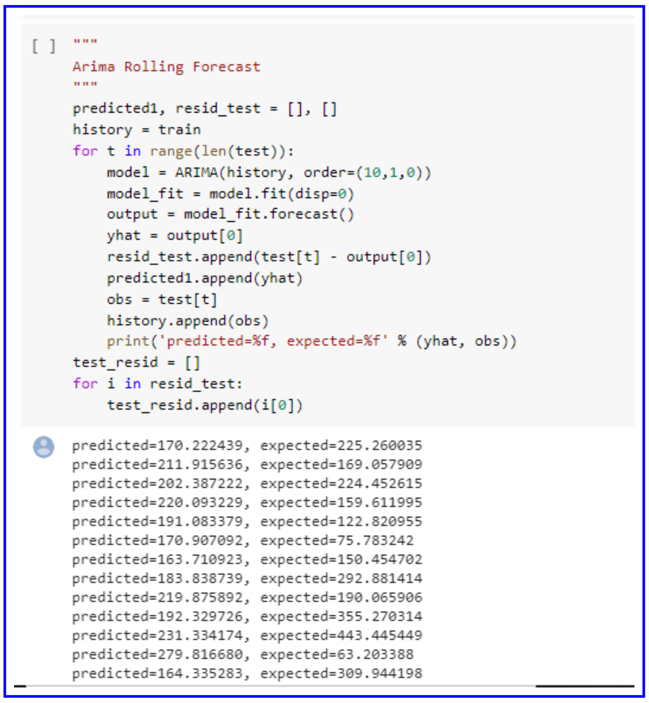
```
# LSTM
model = Sequential()
#model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
#model.add(Dense(1))
model = keras.models.Sequential([
        keras.layers.LSTM(50,input_shape=(train_X.shape[1], train_X.shape[2])),
        keras.layers.Dense(64,activation='relu'),
        keras.layers.Dense(1,activation='relu')])
adam=keras.optimizers.Adam(lr=.001)
model.compile(loss='mae', optimizer='adam')
# fit network
history = model.fit(train_X, train_y, epochs=100, batch_size=72, validation_split=.05, verbose=2, shuffle=True,callbacks = [model_checkpoint_callback])
#plot history
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='Validation')
pyplot.legend()
pyplot.show()
```

Figure 22: Model Implementation LSTM and MLP

Similarly, uni-Variate version of MLP and LSTM is implemented as well for research. The metrics of Multi-variate is better in comparison to uni-variate.

7.3 Model Implementation ARIMA

Figure 18 below shows code implementation for ARIMA. The model is executed for p,d,q of 10,0,0 as obtained by auto ARIMA function.



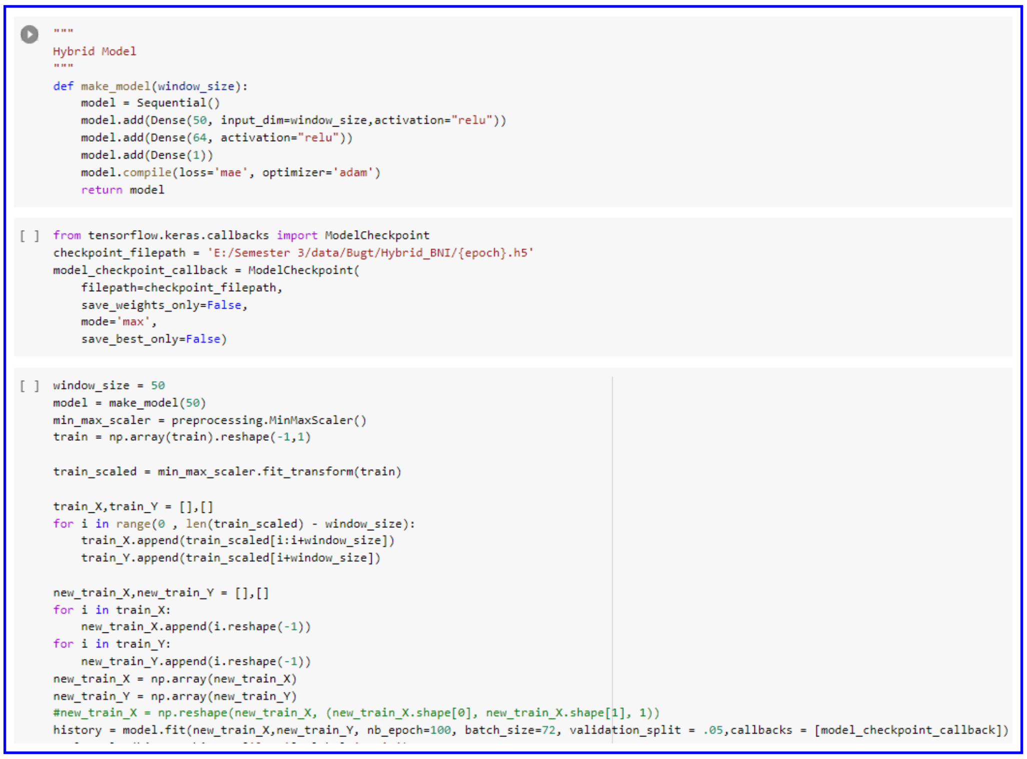
```
[ ] """
    Arima Rolling Forecast
    """
    predicted1, resid_test = [], []
    history = train
    for t in range(len(test)):
        model = ARIMA(history, order=(10,1,0))
        model_fit = model.fit(disp=0)
        output = model_fit.forecast()
        yhat = output[0]
        resid_test.append(test[t] - output[0])
        predicted1.append(yhat)
        obs = test[t]
        history.append(obs)
        print('predicted=%f, expected=%f' % (yhat, obs))
    test_resid = []
    for i in resid_test:
        test_resid.append(i[0])
```

```
predicted=170.222439, expected=225.260035
predicted=211.915636, expected=169.057909
predicted=202.387222, expected=224.452615
predicted=220.093229, expected=159.611995
predicted=191.083379, expected=122.820955
predicted=170.907092, expected=75.783242
predicted=163.710923, expected=150.454702
predicted=183.838739, expected=292.881414
predicted=219.875892, expected=190.065906
predicted=192.329726, expected=355.270314
predicted=231.334174, expected=443.445449
predicted=279.816680, expected=63.203388
predicted=164.335283, expected=309.944198
```

Figure 23: Model Implementation ARIMA

7.4 Model Implementation for Hybrid model

The figure 19 below clearly shows model steps for Hybrid ARIMA ANN model. Residuals obtained from ARIMA model is used as input to ANN model for prediction.



```
"""
Hybrid Model
"""
def make_model(window_size):
    model = Sequential()
    model.add(Dense(50, input_dim=window_size,activation="relu"))
    model.add(Dense(64, activation="relu"))
    model.add(Dense(1))
    model.compile(loss='mae', optimizer='adam')
    return model
```

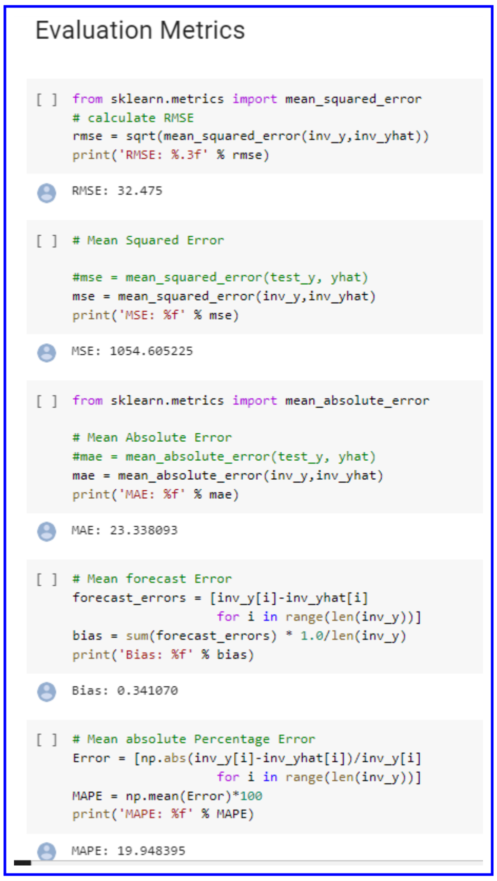
```
[ ] from tensorflow.keras.callbacks import ModelCheckpoint
    checkpoint_filepath = 'E:/Semester 3/data/Bugt/Hybrid_BNI/{epoch}.h5'
    model_checkpoint_callback = ModelCheckpoint(
        filepath=checkpoint_filepath,
        save_weights_only=False,
        mode='max',
        save_best_only=False)
```

```
[ ] window_size = 50
    model = make_model(50)
    min_max_scaler = preprocessing.MinMaxScaler()
    train = np.array(train).reshape(-1,1)

    train_scaled = min_max_scaler.fit_transform(train)

    train_X,train_Y = [],[]
    for i in range(0 , len(train_scaled) - window_size):
        train_X.append(train_scaled[i:i+window_size])
        train_Y.append(train_scaled[i+window_size])

    new_train_X,new_train_Y = [],[]
    for i in train_X:
        new_train_X.append(i.reshape(-1))
    for i in train_Y:
        new_train_Y.append(i.reshape(-1))
    new_train_X = np.array(new_train_X)
    new_train_Y = np.array(new_train_Y)
    #new_train_X = np.reshape(new_train_X, (new_train_X.shape[0], new_train_X.shape[1], 1))
    history = model.fit(new_train_X,new_train_Y, nb_epoch=100, batch_size=72, validation_split = .05,callbacks = [model_checkpoint_callback])
```

Figure 24: Model Implementation Hybrid ARIMA

8 Evaluation Metrics

Figure 20 below shows evaluation metrics for LSTM.

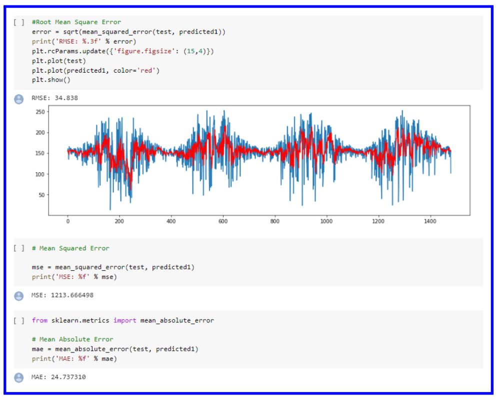


Evaluation Metrics

```
[ ] from sklearn.metrics import mean_squared_error
    # calculate RMSE
    rmse = sqrt(mean_squared_error(inv_y,inv_yhat))
    print('RMSE: %.3f' % rmse)

    RMSE: 32.475

[ ] # Mean Squared Error

    #mse = mean_squared_error(test_y, yhat)
    mse = mean_squared_error(inv_y,inv_yhat)
    print('MSE: %f' % mse)

    MSE: 1054.605225

[ ] from sklearn.metrics import mean_absolute_error

    # Mean Absolute Error
    #mae = mean_absolute_error(test_y, yhat)
    mae = mean_absolute_error(inv_y,inv_yhat)
    print('MAE: %f' % mae)

    MAE: 23.338093

[ ] # Mean forecast Error
    forecast_errors = [inv_y[i]-inv_yhat[i]
                       for i in range(len(inv_y))]
    bias = sum(forecast_errors) * 1.0/len(inv_y)
    print('Bias: %f' % bias)

    Bias: 0.341070

[ ] # Mean absolute Percentage Error
    Error = [np.abs(inv_y[i]-inv_yhat[i])/inv_y[i]
                       for i in range(len(inv_y))]
    MAPE = np.mean(Error)*100
    print('MAPE: %f' % MAPE)

    MAPE: 19.948395
```

Figure 25: Evaluation LSTM GHI

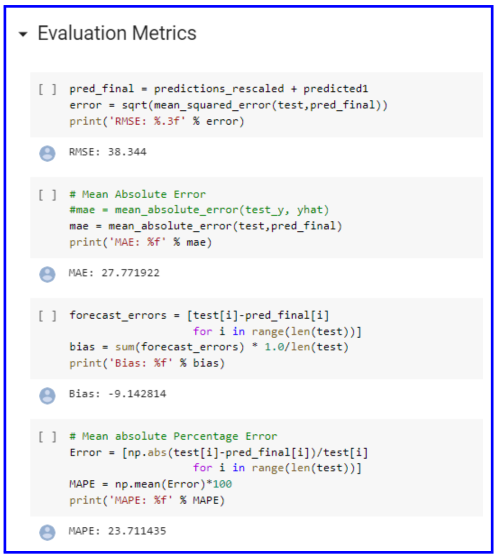
Similarly, figure 21, 22, 23 shows evaluation metrics for MLP, ARIMA and Hybrid model



```
[ ] #Root Mean Square Error
    error = sqrt(mean_squared_error(test, predicted1))
    print('RMSE: %.3f' % error)
    plt.rcParams.update({'figure.figsize': (15,4)})
    plt.plot(test)
    plt.plot(predicted1, color='red')
    plt.show()

    RMSE: 34.838
```

```
[ ] # Mean Squared Error

    mse = mean_squared_error(test, predicted1)
    print('MSE: %f' % mse)

    MSE: 1213.666498

[ ] from sklearn.metrics import mean_absolute_error

    # Mean Absolute Error
    mae = mean_absolute_error(test, predicted1)
    print('MAE: %f' % mae)

    MAE: 24.737310
```

Figure 26: Evaluation ARIMA GHI



Evaluation Metrics

```
[ ] pred_final = predictions_rescaled + predicted1
    error = sqrt(mean_squared_error(test,pred_final))
    print('RMSE: %.3f' % error)
```

```
RMSE: 38.344
```

```
[ ] # Mean Absolute Error
    #mae = mean_absolute_error(test_y, yhat)
    mae = mean_absolute_error(test,pred_final)
    print('MAE: %f' % mae)
```

```
MAE: 27.771922
```

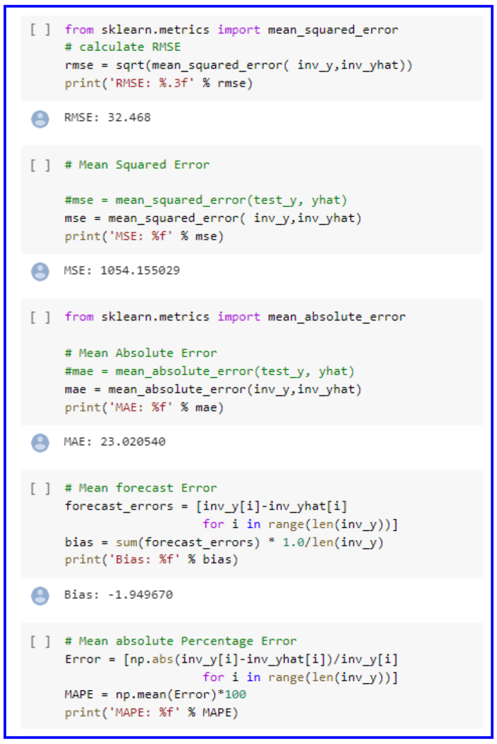
```
[ ] forecast_errors = [test[i]-pred_final[i]
                       for i in range(len(test))]
    bias = sum(forecast_errors) * 1.0/len(test)
    print('Bias: %f' % bias)
```

```
Bias: -9.142814
```

```
[ ] # Mean absolute Percentage Error
    Error = [np.abs(test[i]-pred_final[i])/test[i]
                       for i in range(len(test))]
    MAPE = np.mean(Error)*100
    print('MAPE: %f' % MAPE)
```

```
MAPE: 23.711435
```

Figure 27: Evaluation Hybrid GHI



```
[ ] from sklearn.metrics import mean_squared_error
    # calculate RMSE
    rmse = sqrt(mean_squared_error( inv_y,inv_yhat))
    print('RMSE: %.3f' % rmse)
```

```
RMSE: 32.468
```

```
[ ] # Mean Squared Error

    #mse = mean_squared_error(test_y, yhat)
    mse = mean_squared_error( inv_y,inv_yhat)
    print('MSE: %f' % mse)
```

```
MSE: 1054.155029
```

```
[ ] from sklearn.metrics import mean_absolute_error

    # Mean Absolute Error
    #mae = mean_absolute_error(test_y, yhat)
    mae = mean_absolute_error(inv_y,inv_yhat)
    print('MAE: %f' % mae)
```

```
MAE: 23.020540
```

```
[ ] # Mean forecast Error
    forecast_errors = [inv_y[i]-inv_yhat[i]
                       for i in range(len(inv_y))]
    bias = sum(forecast_errors) * 1.0/len(inv_y)
    print('Bias: %f' % bias)
```

```
Bias: -1.949670
```

```
[ ] # Mean absolute Percentage Error
    Error = [np.abs(inv_y[i]-inv_yhat[i])/inv_y[i]
                       for i in range(len(inv_y))]
    MAPE = np.mean(Error)*100
    print('MAPE: %f' % MAPE)
```

Figure 28: Evaluation MLP GHI