**Shell – Data Science Excercise 2020**

**Building Predictive Model for Gas Energy Demand**

Jack Malik

June 9, 2020

Table of Content

[1. Problem Statement 3](#_Toc42557137)

[2. Implementation Approach 3](#_Toc42557138)

[2.1. Data Description 3](#_Toc42557139)

[2.2. Building a Correlation Matrix of Features and Target 4](#_Toc42557140)

[3. Building Model 5](#_Toc42557141)

[4. Computing Common Model Errors 6](#_Toc42557142)

[5. Program Output 6](#_Toc42557143)

[6. Appendix A – energy\_predictor.py Source File 7](#_Toc42557144)

# Problem Statement

A data file in a csv format contains a daily timeseries of historic weather observations (features), also called predictors and gas demand actuals (target) for gas years 2013 to 2018 inclusive for a fictional country. The objective is to use the Python toolset to derive a forecasting model that returns out of sample predictions for gas year 2018.

# Implementation Approach

We observe 4 predictors in the data file and a single target. Our strategy depends on the results of our *Exploratory Data Analysis* (EDS). We analyze all predictors and the level of correlation between them as well as between the target.

The selection of our predictive model depends on our understanding of data achieved during the EDS process we perform and the conclusions we draw which model could be most effective in predicting the target.

We first build the correlation matrix of all features and target. We hope to eliminate predictors if they are correlated to simplify the process.

## 2.1. Data Description

We use Python ***describe***() function to view data characteristics. The output of the function looks as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **station\_01\_tem.** | **station\_02\_tem.** | **station\_01\_wind** | **station\_02\_wind** | **gas\_demand\_volume** |
| **count** | 2165 | 2164 | 2164 | 2163 | 2165 |
| **mean** | 11.56079908 | 11.08987985 | 8.307989834 | 6.532556634 | 89.36482217 |
| **std** | 6.48945676 | 6.51711694 | 3.220903693 | 2.510031544 | 51.07682739 |
| **min** | -7.3 | -6.07 | 2.53 | 1.84 | 24.64 |
| **25%** | 6.24 | 5.7375 | 6.03 | 4.76 | 41.61 |
| **50%** | 11.06 | 10.93 | 7.695 | 6.09 | 76.64 |
| **75%** | 16.97 | 16.39 | 9.9125 | 7.7 | 131.08 |
| **max** | 29.21 | 28.13 | 22.19 | 18.87 | 270.69 |

## 2.2. Building a Correlation Matrix of Features and Target

We build a correlation matrix of features and target to possibly eliminate features which are correlated between themselves in hope to limit overall number of features. The program builds the following full correlation matrix:

Full correlation matrix of all features:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **station\_01\_temp.** | **station\_02\_temp.** | **station\_01\_wind** | **station\_02\_wind** | **gas\_demand\_vol.** |
| **station\_01\_temp.** | 1 | 0.98997923 | -0.205899783 | -0.155647172 | -0.940276696 |
| **station\_02\_temp.** | 0.98997923 | 1 | -0.215077823 | -0.143630781 | -0.937628986 |
| **station\_01\_wind** | -0.205899783 | -0.215077823 | 1 | 0.910181372 | 0.257729078 |
| **station\_02\_wind** | -0.155647172 | -0.143630781 | 0.910181372 | 1 | 0.217591006 |
| **gas\_demand\_vol.** | -0.940276696 | -0.937628986 | 0.257729078 | 0.217591006 | 1 |

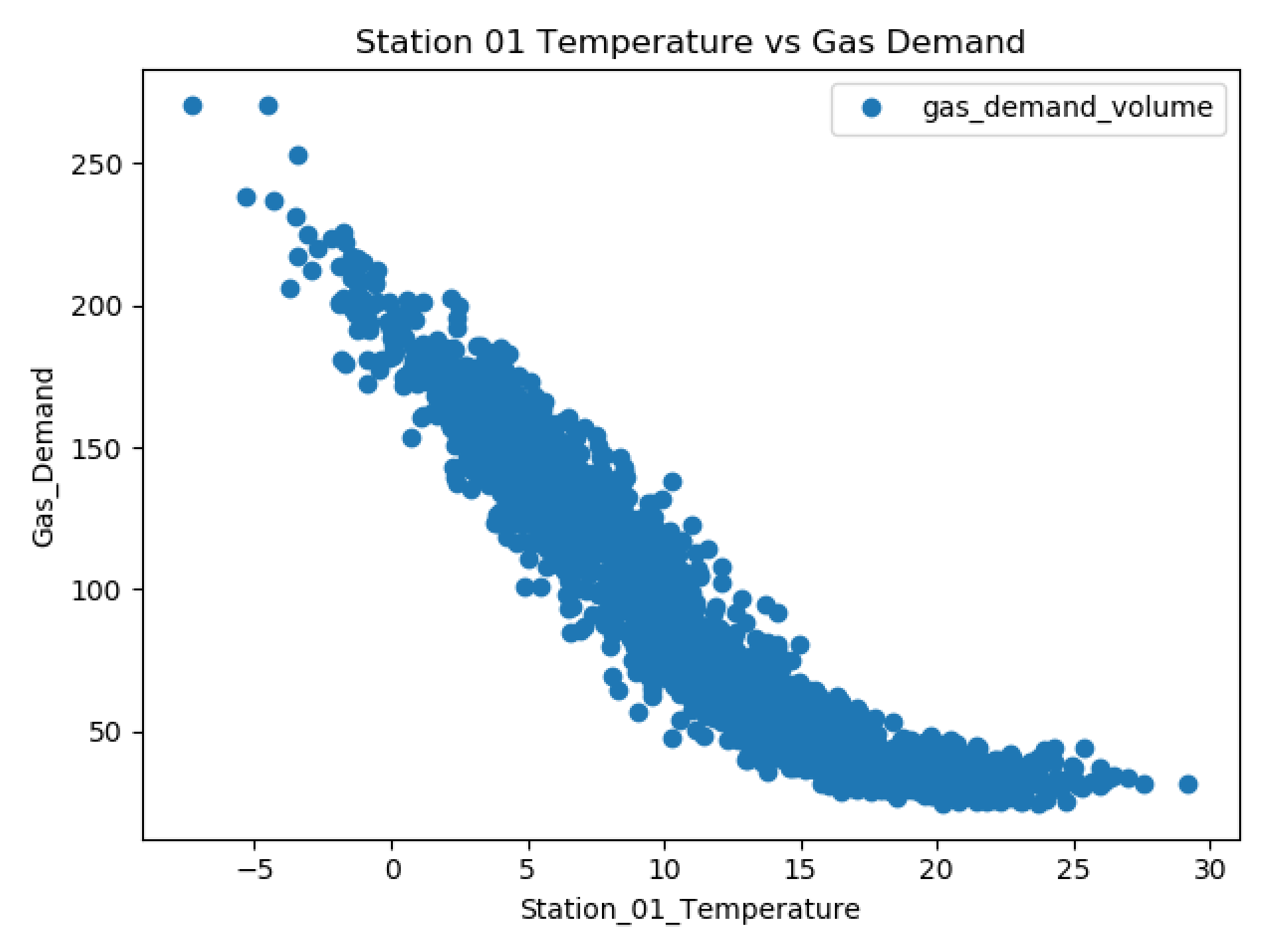
We observer strong correlation between ‘***station\_01\_temperature***’ and ‘***station\_02\_temperature***’ predictors while noticing that they are both very correlated with the volume of gas demand.

We also observe no correlation between temperature of any of the 2 stations and the levels of wind reported in those stations. Moreover, the levels of winds are not correlated with gas demand.

The above observations allow us to remove one of the station temperatures as well as winds recordings in both stations as not affecting gas demand volumes and therefore insignificant in predicting the target value of gas demand volumes.

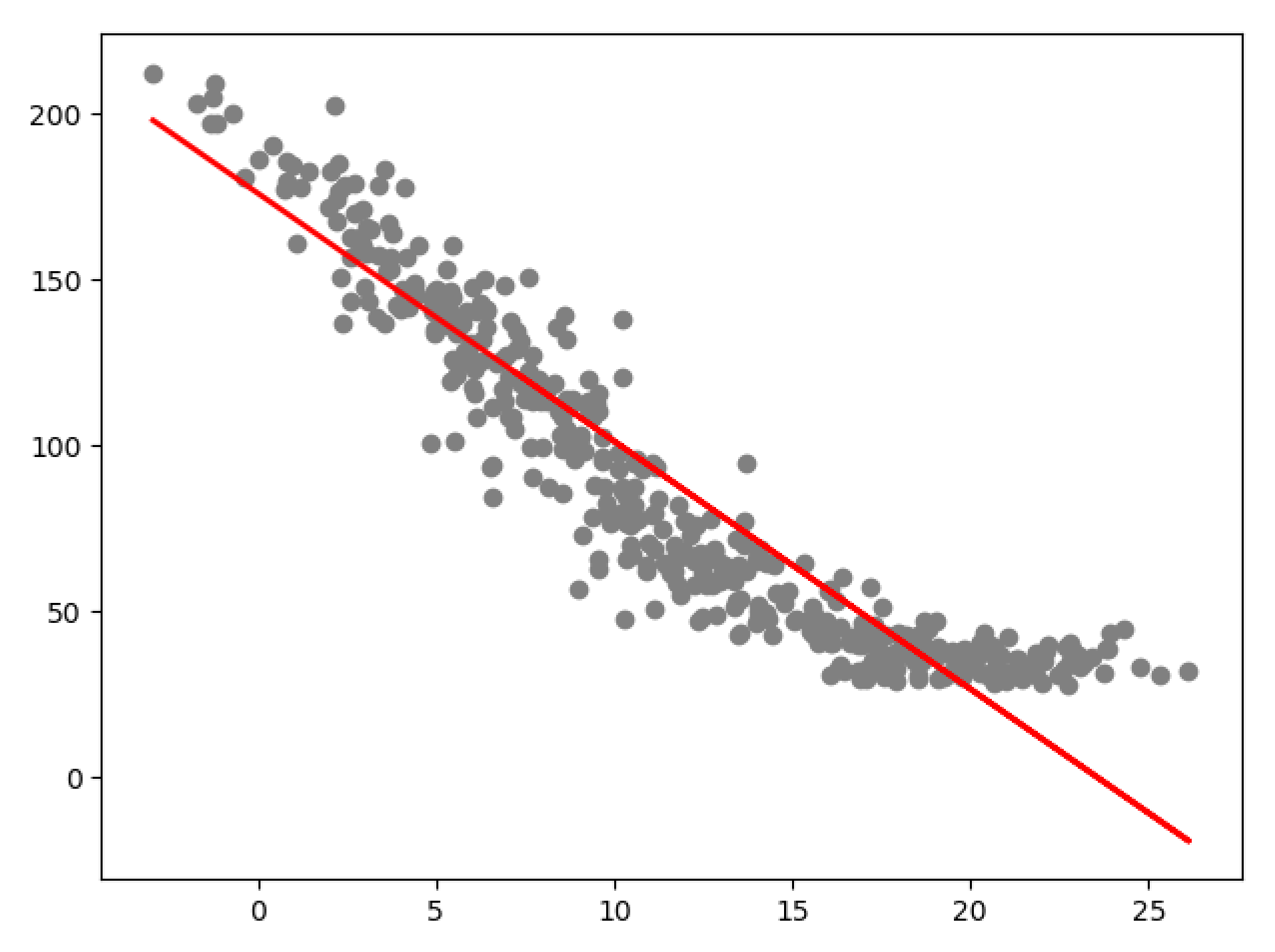
We are therefore left with a single predictor ‘***station\_01\_temperature***’ and a target data stored in ‘***gas\_demand\_vol***’. Our reduced correlation matrix looks as follows:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **station\_01\_temp.** | **station\_01\_wind** | **gas\_demand\_volume** |
| **station\_01\_temp.** | 1 | -0.205899783 | -0.940276696 |
| **station\_01\_wind** | -0.205899783 | 1 | 0.257729078 |
| **gas\_demand\_volume** | -0.940276696 | 0.257729078 | 1 |



# Building Model

We observe strong correlation between ‘station\_01\_temperature’ and ‘gas\_demand\_volume’. We therefore decide to build a simple univariate linear regression model to predict gas demand for year 2018. Our linear regression model can be visualized as follows:



# Computing Common Model Errors

Mean Absolute Error: 15.901675177380474

Mean Squared Error: 416.7833149689213

Root Mean Squared Error: 20.415271611441305

# Program Output

Below, we show a partial content of the final Excel file the program produces. The file contains 3 columns: Date, Actual Demand and Predicted Demand for gas during the month of July, 2018.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Date** | **Actual Demand** | **Predicted Demand** |
| **0** | 2018-07-01 00:00:00 | 30.34 | 28.60556262 |
| **1** | 2018-07-02 00:00:00 | 37.91 | 25.07608093 |
| **2** | 2018-07-03 00:00:00 | 38.38 | 20.3956813 |
| **3** | 2018-07-04 00:00:00 | 37.81 | 9.96069195 |
| **4** | 2018-07-05 00:00:00 | 38.35 | 12.18579997 |
| **5** | 2018-07-06 00:00:00 | 34.38 | 23.46479581 |
| **6** | 2018-07-07 00:00:00 | 29.1 | 18.78439618 |
| **7** | 2018-07-08 00:00:00 | 28.11 | 23.46479581 |
| **8** | 2018-07-09 00:00:00 | 36.58 | 24.84589734 |
| **9** | 2018-07-10 00:00:00 | 39.5 | 48.17116765 |
| **10** | 2018-07-11 00:00:00 | 38.57 | 39.27073556 |
| **11** | 2018-07-12 00:00:00 | 38.3 | 30.67721492 |
| **12** | 2018-07-13 00:00:00 | 35.87 | 26.61063819 |
| **13** | 2018-07-14 00:00:00 | 29.2 | 19.16803549 |

…… more data

When executed the program creates 4 different Excel files:

1/. Output of the describe() function.

2/. Full correlation matrix

3/. Stripped correlation matrix

4/. File containing final output of actual and predicted gas demand volumes as shown above.

# Final Remarks

The program has been developed using Anaconda 2018.12 – built py37\_0. Python version 3.7 was used. The program makes no attempt to take any user input or test code through any Unit Testing.

The program assumes that the given input data file name is ‘weather\_and\_gas\_demand\_historic\_actuals.csv’ and it is located in the same directory as the actual ‘energy\_predictor.py’ Python file. The source code is given below.

# Appendix A – energy\_predictor.py Source File

import logging

import sys

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn import metrics

import scipy as sp

from scipy.stats import shapiro

logger\_ = logging.getLogger(\_\_name\_\_)

logger\_.setLevel(logging.INFO)

handler = logging.StreamHandler(sys.stdout)

handler.setLevel(logging.INFO)

formatter = logging.Formatter('[%(asctime)s]-[%(name)s]-[%(levelname)s]: %(message)s')

handler.setFormatter(formatter)

logger\_.addHandler(handler)

class EnergyFeaturesCorrelationMatrix(object):

@staticmethod

def get\_energy\_tm\_data(full\_path\_file\_name):

data\_frame = pd.read\_csv(full\_path\_file\_name)

return data\_frame

@staticmethod

def build(src\_file\_name):

"""

Parsing energy demand file given. The content of the form:

<dy>,<station\_01\_temperature\_degrees\_celcius>,<station\_02\_temperature\_degrees\_celcius>,

<station\_01\_wind\_speed\_m\_per\_s>,<station\_02\_wind\_speed\_m\_per\_s>,<gas\_demand\_mcm>

:param src\_file\_name: source file given as part of exercise

:return: Pandas data frame containing file data

"""

try:

df = EnergyFeaturesCorrelationMatrix.get\_energy\_tm\_data(src\_file\_name)

df.dropna()

df['dy'] = pd.to\_datetime(df['dy'], format='%d/%m/%Y')

df.columns = ['date',

'station\_01\_temperature',

'station\_02\_temperature',

'station\_01\_wind',

'station\_02\_wind',

'gas\_demand\_volume']

df = df[pd.to\_numeric(df['gas\_demand\_volume'], errors='coerce').notnull()]

df = df[pd.to\_numeric(df['station\_01\_temperature'], errors='coerce').notnull()]

with pd.ExcelWriter('weather\_and\_gas\_demand\_historic\_actuals\_cleaned.xlsx') as writer:

df.to\_excel(writer, sheet\_name='Cleaned-Data')

return df

except Exception as ex:

logger\_.error("Failed. Msg: {}".format(ex))

return None

def main(argv):

logger\_.info("Starting program .. ")

# create correlation matrix of all energy demand features

mtx = EnergyFeaturesCorrelationMatrix()

data\_frame = mtx.build("weather\_and\_gas\_demand\_historic\_actuals.csv")

if data\_frame is None:

logger\_.error("Unexpected exception while building data frame.")

return 1

data\_frame.plot(x='station\_01\_temperature', y='gas\_demand\_volume', style='o')

plt.title('Station 01 Temperature vs Gas Demand')

plt.xlabel('Station\_01\_Temperature')

plt.ylabel('Gas\_Demand')

plt.show()

plt.close()

data\_frame\_no\_date = data\_frame.drop(['date'], axis=1, inplace=False)

corr\_matrix\_full = data\_frame\_no\_date.corr()

print(corr\_matrix\_full)

with pd.ExcelWriter('correlation\_matrix\_full.xlsx') as writer:

corr\_matrix\_full.to\_excel(writer, sheet\_name='CORR-VER-1')

# remove unnecessary columns

corr\_matrix\_stripped = corr\_matrix\_full.drop(['station\_02\_temperature',

'station\_02\_wind'],

axis=1, inplace=False)

# remove unnecessary rows

corr\_matrix\_stripped.drop(['station\_02\_temperature', 'station\_02\_wind'], inplace=True)

with pd.ExcelWriter('correlation\_matrix\_stripped.xlsx') as writer:

corr\_matrix\_stripped.to\_excel(writer, sheet\_name='CORR-VER-2')

# ------------------------------------------------------------------------------------

# stripped correlation matrix of the station 2 parameters as correlated

# ------------------------------------------------------------------------------------

# station\_01\_temperature station\_01\_wind gas\_demand

# station\_01\_temperature 1 -0.205899783 -0.940276696

# station\_01\_wind -0.205899783 1 0.257729078

# gas\_demand -0.940276696 0.257729078 1

# ------------------------------------------------------------------------------------

# Now remove station\_01\_wind column and row from correlation matrix

corr\_matrix = corr\_matrix\_stripped.drop(['station\_01\_wind'], inplace=False)

corr\_matrix.drop(['station\_01\_wind'], axis=1, inplace=True)

with pd.ExcelWriter('correlation\_matrix.xlsx') as writer:

corr\_matrix.to\_excel(writer, sheet\_name='CORR-MTX-FINAL')

# Perform test for normality of distribution of station\_01\_temperature values

distance, p\_value = sp.stats.normaltest(data\_frame['gas\_demand\_volume'])

stat, s\_p\_value = shapiro(data\_frame['gas\_demand\_volume'])

x = data\_frame['station\_01\_temperature'].values.reshape(-1,1)

y = data\_frame['gas\_demand\_volume'].values.reshape(-1,1)

dates = data\_frame['date'].values.reshape(-1,1)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y,

test\_size=0.2105,

shuffle=False)

x\_train\_, x\_test\_, dates\_train, dates\_test = train\_test\_split(x, dates,

test\_size=0.2105,

shuffle=False)

regressor = LinearRegression()

# training the algorithm

regressor.fit(x\_train, y\_train)

# Predicting

y\_pred = regressor.predict(x\_test)

plt.clf()

plt.scatter(x\_test, y\_test, color='gray')

plt.plot(x\_test, y\_pred, color='red', linewidth=2)

plt.show()

description = data\_frame.describe()

with pd.ExcelWriter('description.xlsx') as writer:

description.to\_excel(writer, sheet\_name='Info')

logger\_.info('Mean Absolute Error:', metrics.mean\_absolute\_error(y\_test, y\_pred))

logger\_.info('Mean Squared Error:', metrics.mean\_squared\_error(y\_test, y\_pred))

logger\_.info('Root Mean Squared Error:', np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred)))

new\_df = pd.DataFrame({'Date': dates\_test.flatten(),

'Actual Demand': y\_test.flatten(),

'Predicted Demand': y\_pred.flatten()})

with pd.ExcelWriter('new\_df.xlsx') as writer:

new\_df.to\_excel(writer, sheet\_name='Info')

logger\_.info("Completed program execution.. ")

return 0

if \_\_name\_\_ == '\_\_main\_\_':

status = main(sys.argv[1:])

sys.exit(status)