

Time series Analysis and Modeling Final Term Project

Hsueh-Yi Lu

Master of Data Science, The George Washington University

DATS 6313: Time series Analysis and Modeling

Dr. Reza Jafari

April 26, 2022

Content

Abstract.....	3
Introduction.....	9
Description of Dataset.....	10
Stationarity.....	17
Time series Decomposition.....	19
Holt-Winters Method.....	20
Feature Selection.....	21
Base-Models	25
Multiple Linear Regression.....	26
ARMA, ARIMA, and SARIMA Model	30
Levenberg Marquardt Algorithm.....	33
Diagnostic Analysis	35
Deep Learning Model	39
Final Model Selection.....	40
Forecast Function.....	40
h-step Ahead Predictions	41
Summary and Conclusion.....	42
Appendix.....	43

datetime	0	
StnPres	26	
SeaPres	26	
Temperature	3	
Td dew point	960	
RH	960	
WS	14	
WD	368	
WSGust	15	
WDGust	15	
Precp	5778	Cloud Amount 78475
PrecpHour	8	station 0
SunShine	61636	area 0
GloblRad	430	load 0
Visb	93607	
UVI	555	

Figure 1

11

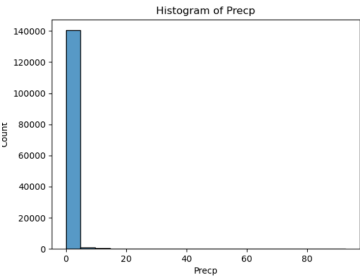


Figure 2

11



Figure 3

12

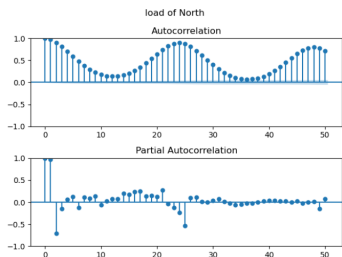


Figure 4

13

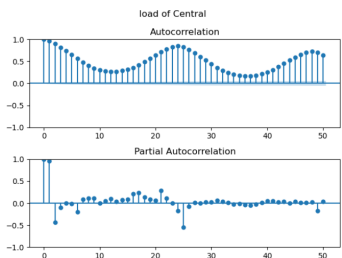


Figure 5

13

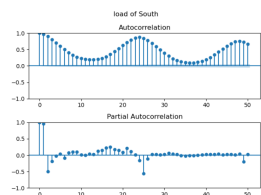


Figure 6

14

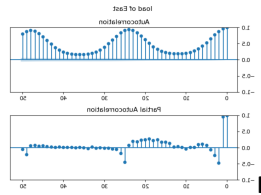


Figure 7

14

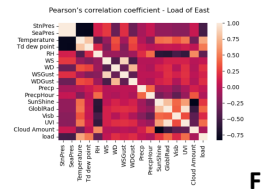


Figure 8

16

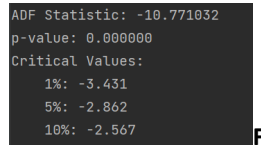


Figure 9

17

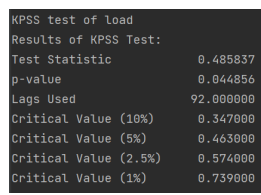


Figure 10

18

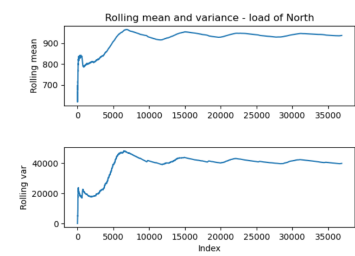


Figure 11

18

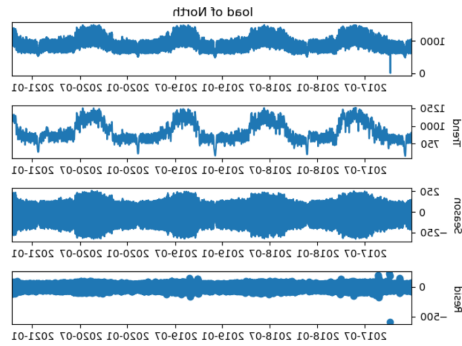


Figure 12

19

The strength of trend for this data set is 0.96
 The strength of seasonality for this data set is 0.96
 The dependent variable of area of North is highly trend and seasonal.

Figure 13

20

```
# Holt-Winter Method
import statsmodels.tsa.holtwinters as ets

y_HW = df_test.copy()
y_HW['date'] = df_train['date']
y_HW['load'] = df_train['load']
y_HW['hw_fit'] = fitted_model.forecast(len(df_test))
```

Figure 14

20

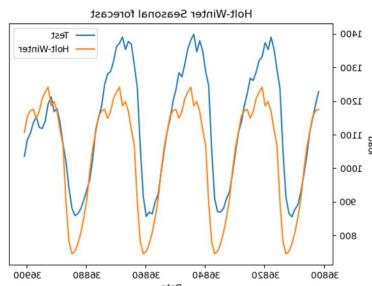


Figure 15

21

Figure 16

22

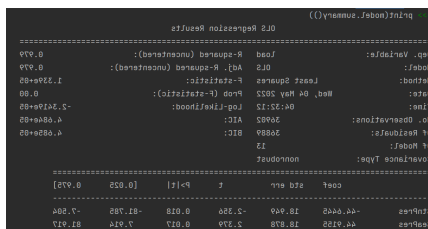


Figure 17

22

Figure 18

24

Figure 19

24

```
Variable: WS      Importance: 0.04
Variable: GlobRad Importance: 0.04
Variable: StnPres Importance: 0.03
Variable: SeaPres Importance: 0.03
Variable: RH      Importance: 0.03
Variable: WD      Importance: 0.03
Variable: WDbust  Importance: 0.03
Variable: UVI     Importance: 0.03
Variable: PrecpHour Importance: 0.02
Variable: Precp   Importance: 0.01
```

Figure 20

25

```
MSE of SES forecast is 47488.06
```

Figure 21

26

```

x1c.v  x1c.f  x1c.r  x1c.s  x1c.t  x1c.u  x1c.v  x1c.w  x1c.x  x1c.y  x1c.z
x2c.v  x2c.f  x2c.r  x2c.s  x2c.t  x2c.u  x2c.v  x2c.w  x2c.x  x2c.y  x2c.z
x3c.v  x3c.f  x3c.r  x3c.s  x3c.t  x3c.u  x3c.v  x3c.w  x3c.x  x3c.y  x3c.z
x4c.v  x4c.f  x4c.r  x4c.s  x4c.t  x4c.u  x4c.v  x4c.w  x4c.x  x4c.y  x4c.z
x5c.v  x5c.f  x5c.r  x5c.s  x5c.t  x5c.u  x5c.v  x5c.w  x5c.x  x5c.y  x5c.z
x6c.v  x6c.f  x6c.r  x6c.s  x6c.t  x6c.u  x6c.v  x6c.w  x6c.x  x6c.y  x6c.z
x7c.v  x7c.f  x7c.r  x7c.s  x7c.t  x7c.u  x7c.v  x7c.w  x7c.x  x7c.y  x7c.z
x8c.v  x8c.f  x8c.r  x8c.s  x8c.t  x8c.u  x8c.v  x8c.w  x8c.x  x8c.y  x8c.z
x9c.v  x9c.f  x9c.r  x9c.s  x9c.t  x9c.u  x9c.v  x9c.w  x9c.x  x9c.y  x9c.z
x10c.v x10c.f x10c.r x10c.s x10c.t x10c.u x10c.v x10c.w x10c.x x10c.y x10c.z
x11c.v x11c.f x11c.r x11c.s x11c.t x11c.u x11c.v x11c.w x11c.x x11c.y x11c.z
x12c.v x12c.f x12c.r x12c.s x12c.t x12c.u x12c.v x12c.w x12c.x x12c.y x12c.z
x13c.v x13c.f x13c.r x13c.s x13c.t x13c.u x13c.v x13c.w x13c.x x13c.y x13c.z
x14c.v x14c.f x14c.r x14c.s x14c.t x14c.u x14c.v x14c.w x14c.x x14c.y x14c.z
x15c.v x15c.f x15c.r x15c.s x15c.t x15c.u x15c.v x15c.w x15c.x x15c.y x15c.z
x16c.v x16c.f x16c.r x16c.s x16c.t x16c.u x16c.v x16c.w x16c.x x16c.y x16c.z
x17c.v x17c.f x17c.r x17c.s x17c.t x17c.u x17c.v x17c.w x17c.x x17c.y x17c.z
x18c.v x18c.f x18c.r x18c.s x18c.t x18c.u x18c.v x18c.w x18c.x x18c.y x18c.z
x19c.v x19c.f x19c.r x19c.s x19c.t x19c.u x19c.v x19c.w x19c.x x19c.y x19c.z
x20c.v x20c.f x20c.r x20c.s x20c.t x20c.u x20c.v x20c.w x20c.x x20c.y x20c.z
x21c.v x21c.f x21c.r x21c.s x21c.t x21c.u x21c.v x21c.w x21c.x x21c.y x21c.z
x22c.v x22c.f x22c.r x22c.s x22c.t x22c.u x22c.v x22c.w x22c.x x22c.y x22c.z
x23c.v x23c.f x23c.r x23c.s x23c.t x23c.u x23c.v x23c.w x23c.x x23c.y x23c.z
x24c.v x24c.f x24c.r x24c.s x24c.t x24c.u x24c.v x24c.w x24c.x x24c.y x24c.z
x25c.v x25c.f x25c.r x25c.s x25c.t x25c.u x25c.v x25c.w x25c.x x25c.y x25c.z
x26c.v x26c.f x26c.r x26c.s x26c.t x26c.u x26c.v x26c.w x26c.x x26c.y x26c.z
x27c.v x27c.f x27c.r x27c.s x27c.t x27c.u x27c.v x27c.w x27c.x x27c.y x27c.z
x28c.v x28c.f x28c.r x28c.s x28c.t x28c.u x28c.v x28c.w x28c.x x28c.y x28c.z
x29c.v x29c.f x29c.r x29c.s x29c.t x29c.u x29c.v x29c.w x29c.x x29c.y x29c.z
x30c.v x30c.f x30c.r x30c.s x30c.t x30c.u x30c.v x30c.w x30c.x x30c.y x30c.z
x31c.v x31c.f x31c.r x31c.s x31c.t x31c.u x31c.v x31c.w x31c.x x31c.y x31c.z
x32c.v x32c.f x32c.r x32c.s x32c.t x32c.u x32c.v x32c.w x32c.x x32c.y x32c.z
x33c.v x33c.f x33c.r x33c.s x33c.t x33c.u x33c.v x33c.w x33c.x x33c.y x33c.z
x34c.v x34c.f x34c.r x34c.s x34c.t x34c.u x34c.v x34c.w x34c.x x34c.y x34c.z
x35c.v x35c.f x35c.r x35c.s x35c.t x35c.u x35c.v x35c.w x35c.x x35c.y x35c.z
x36c.v x36c.f x36c.r x36c.s x36c.t x36c.u x36c.v x36c.w x36c.x x36c.y x36c.z
x37c.v x37c.f x37c.r x37c.s x37c.t x37c.u x37c.v x37c.w x37c.x x37c.y x37c.z
x38c.v x38c.f x38c.r x38c.s x38c.t x38c.u x38c.v x38c.w x38c.x x38c.y x38c.z
x39c.v x39c.f x39c.r x39c.s x39c.t x39c.u x39c.v x39c.w x39c.x x39c.y x39c.z
x40c.v x40c.f x40c.r x40c.s x40c.t x40c.u x40c.v x40c.w x40c.x x40c.y x40c.z
x41c.v x41c.f x41c.r x41c.s x41c.t x41c.u x41c.v x41c.w x41c.x x41c.y x41c.z
x42c.v x42c.f x42c.r x42c.s x42c.t x42c.u x42c.v x42c.w x42c.x x42c.y x42c.z
x43c.v x43c.f x43c.r x43c.s x43c.t x43c.u x43c.v x43c.w x43c.x x43c.y x43c.z
x44c.v x44c.f x44c.r x44c.s x44c.t x44c.u x44c.v x44c.w x44c.x x44c.y x44c.z
x45c.v x45c.f x45c.r x45c.s x45c.t x45c.u x45c.v x45c.w x45c.x x45c.y x45c.z
x46c.v x46c.f x46c.r x46c.s x46c.t x46c.u x46c.v x46c.w x46c.x x46c.y x46c.z
x47c.v x47c.f x47c.r x47c.s x47c.t x47c.u x47c.v x47c.w x47c.x x47c.y x47c.z
x48c.v x48c.f x48c.r x48c.s x48c.t x48c.u x48c.v x48c.w x48c.x x48c.y x48c.z
x49c.v x49c.f x49c.r x49c.s x49c.t x49c.u x49c.v x49c.w x49c.x x49c.y x49c.z
x50c.v x50c.f x50c.r x50c.s x50c.t x50c.u x50c.v x50c.w x50c.x x50c.y x50c.z
x51c.v x51c.f x51c.r x51c.s x51c.t x51c.u x51c.v x51c.w x51c.x x51c.y x51c.z
x52c.v x52c.f x52c.r x52c.s x52c.t x52c.u x52c.v x52c.w x52c.x x52c.y x52c.z
x53c.v x53c.f x53c.r x53c.s x53c.t x53c.u x53c.v x53c.w x53c.x x53c.y x53c.z
x54c.v x54c.f x54c.r x54c.s x54c.t x54c.u x54c.v x54c.w x54c.x x54c.y x54c.z
x55c.v x55c.f x55c.r x55c.s x55c.t x55c.u x55c.v x55c.w x55c.x x55c.y x55c.z
x56c.v x56c.f x56c.r x56c.s x56c.t x56c.u x56c.v x56c.w x56c.x x56c.y x56c.z
x57c.v x57c.f x57c.r x57c.s x57c.t x57c.u x57c.v x57c.w x57c.x x57c.y x57c.z
x58c.v x58c.f x58c.r x58c.s x58c.t x58c.u x58c.v x58c.w x58c.x x58c.y x58c.z
x59c.v x59c.f x59c.r x59c.s x59c.t x59c.u x59c.v x59c.w x59c.x x59c.y x59c.z
x60c.v x60c.f x60c.r x60c.s x60c.t x60c.u x60c.v x60c.w x60c.x x60c.y x60c.z
x61c.v x61c.f x61c.r x61c.s x61c.t x61c.u x61c.v x61c.w x61c.x x61c.y x61c.z
x62c.v x62c.f x62c.r x62c.s x62c.t x62c.u x62c.v x62c.w x62c.x x62c.y x62c.z
x63c.v x63c.f x63c.r x63c.s x63c.t x63c.u x63c.v x63c.w x63c.x x63c.y x63c.z
x64c.v x64c.f x64c.r x64c.s x64c.t x64c.u x64c.v x64c.w x64c.x x64c.y x64c.z
x65c.v x65c.f x65c.r x65c.s x65c.t x65c.u x65c.v x65c.w x65c.x x65c.y x65c.z
x66c.v x66c.f x66c.r x66c.s x66c.t x66c.u x66c.v x66c.w x66c.x x66c.y x66c.z
x67c.v x67c.f x67c.r x67c.s x67c.t x67c.u x67c.v x67c.w x67c.x x67c.y x67c.z
x68c.v x68c.f x68c.r x68c.s x68c.t x68c.u x68c.v x68c.w x68c.x x68c.y x68c.z
x69c.v x69c.f x69c.r x69c.s x69c.t x69c.u x69c.v x69c.w x69c.x x69c.y x69c.z
x70c.v x70c.f x70c.r x70c.s x70c.t x70c.u x70c.v x70c.w x70c.x x70c.y x70c.z
x71c.v x71c.f x71c.r x71c.s x71c.t x71c.u x71c.v x71c.w x71c.x x71c.y x71c.z
x72c.v x72c.f x72c.r x72c.s x72c.t x72c.u x72c.v x72c.w x72c.x x72c.y x72c.z
x73c.v x73c.f x73c.r x73c.s x73c.t x73c.u x73c.v x73c.w x73c.x x73c.y x73c.z
x74c.v x74c.f x74c.r x74c.s x74c.t x74c.u x74c.v x74c.w x74c.x x74c.y x74c.z
x75c.v x75c.f x75c.r x75c.s x75c.t x75c.u x75c.v x75c.w x75c.x x75c.y x75c.z
x76c.v x76c.f x76c.r x76c.s x76c.t x76c.u x76c.v x76c.w x76c.x x76c.y x76c.z
x77c.v x77c.f x77c.r x77c.s x77c.t x77c.u x77c.v x77c.w x77c.x x77c.y x77c.z
x78c.v x78c.f x78c.r x78c.s x78c.t x78c.u x78c.v x78c.w x78c.x x78c.y x78c.z
x79c.v x79c.f x79c.r x79c.s x79c.t x79c.u x79c.v x79c.w x79c.x x79c.y x79c.z
x80c.v x80c.f x80c.r x80c.s x80c.t x80c.u x80c.v x80c.w x80c.x x80c.y x80c.z
x81c.v x81c.f x81c.r x81c.s x81c.t x81c.u x81c.v x81c.w x81c.x x81c.y x81c.z
x82c.v x82c.f x82c.r x82c.s x82c.t x82c.u x82c.v x82c.w x82c.x x82c.y x82c.z
x83c.v x83c.f x83c.r x83c.s x83c.t x83c.u x83c.v x83c.w x83c.x x83c.y x83c.z
x84c.v x84c.f x84c.r x84c.s x84c.t x84c.u x84c.v x84c.w x84c.x x84c.y x84c.z
x85c.v x85c.f x85c.r x85c.s x85c.t x85c.u x85c.v x85c.w x85c.x x85c.y x85c.z
x86c.v x86c.f x86c.r x86c.s x86c.t x86c.u x86c.v x86c.w x86c.x x86c.y x86c.z
x87c.v x87c.f x87c.r x87c.s x87c.t x87c.u x87c.v x87c.w x87c.x x87c.y x87c.z
x88c.v x88c.f x88c.r x88c.s x88c.t x88c.u x88c.v x88c.w x88c.x x88c.y x88c.z
x89c.v x89c.f x89c.r x89c.s x89c.t x89c.u x89c.v x89c.w x89c.x x89c.y x89c.z
x90c.v x90c.f x90c.r x90c.s x90c.t x90c.u x90c.v x90c.w x90c.x x90c.y x90c.z
x91c.v x91c.f x91c.r x91c.s x91c.t x91c.u x91c.v x91c.w x91c.x x91c.y x91c.z
x92c.v x92c.f x92c.r x92c.s x92c.t x92c.u x92c.v x92c.w x92c.x x92c.y x92c.z
x93c.v x93c.f x93c.r x93c.s x93c.t x93c.u x93c.v x93c.w x93c.x x93c.y x93c.z
x94c.v x94c.f x94c.r x94c.s x94c.t x94c.u x94c.v x94c.w x94c.x x94c.y x94c.z
x95c.v x95c.f x95c.r x95c.s x95c.t x95c.u x95c.v x95c.w x95c.x x95c.y x95c.z
x96c.v x96c.f x96c.r x96c.s x96c.t x96c.u x96c.v x96c.w x96c.x x96c.y x96c.z
x97c.v x97c.f x97c.r x97c.s x97c.t x97c.u x97c.v x97c.w x97c.x x97c.y x97c.z
x98c.v x98c.f x98c.r x98c.s x98c.t x98c.u x98c.v x98c.w x98c.x x98c.y x98c.z
x99c.v x99c.f x99c.r x99c.s x99c.t x99c.u x99c.v x99c.w x99c.x x99c.y x99c.z
x100c.v x100c.f x100c.r x100c.s x100c.t x100c.u x100c.v x100c.w x100c.x x100c.y x100c.z

```

Figure 22

27

```

AIC of model2 468482.41
BIC of model2 468496.09
Adjusted R-square of model2 0.98
Adjusted R-square of model2 0.98

```

Figure 23

27

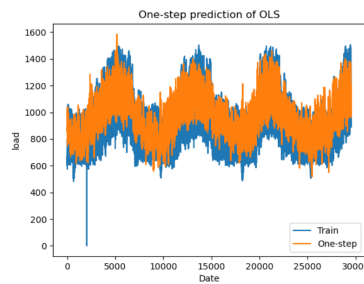


Figure 24

28

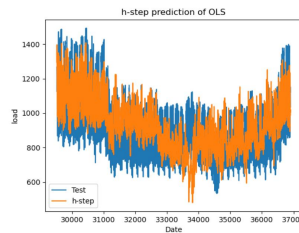


Figure 25

28

Figure 26

29

```

print(summary(f_test(a1))
print(summary(t_test(a1))
<-F Test: Farray([[4292.4971777]]), prob=0, df_denom=3, df_num=10)
=====
Test for Constraints
=====
      coef      std err      t      Pr>|t|      [0.025      0.975]
-----
c0      44.7404      18.871      2.372      0.018      7.772      81.749
c1      22.2618      8.269      26.922      0.000      22.733      23.789
c2      -1.6795      0.870      -22.833      0.000      -1.029      -1.530
c3      -9.8489      1.112      -8.858      0.000      -11.289      -8.409
c4      13.8904      0.581      23.792      0.000      12.788      14.873
c5      0.0548      0.010      5.734      0.000      0.036      0.074
c6      3.3868      0.485      6.982      0.000      2.456      4.258
c7      91.0454      2.885      31.683      0.000      85.168      97.163
c8      53.0215      2.741      19.343      0.000      47.594      58.449

```

Figure 27

30

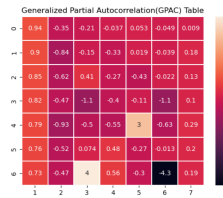


Figure 28

31

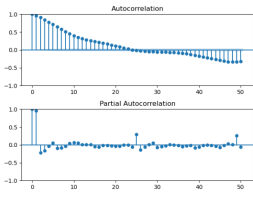


Figure 29

32

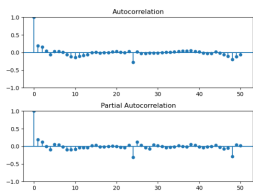


Figure 30

32

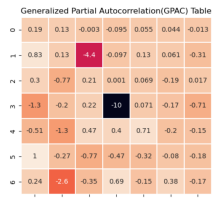


Figure 31

33

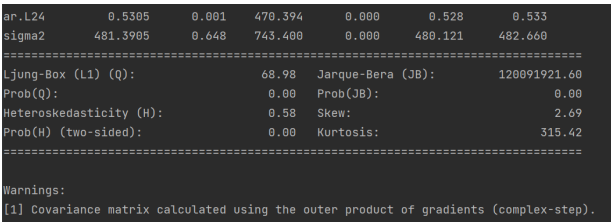


Figure 32

35

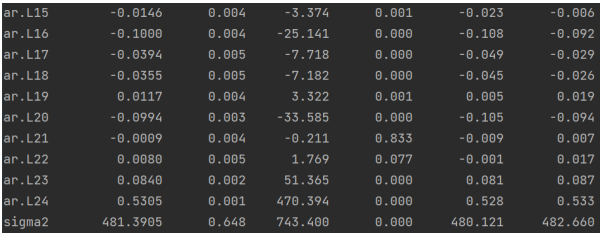


Figure 33

36

```
N = len(erf)
lb_stat, pvalue = sm.stats.acorr_ljungbox(errortrain1[1:], lags=[20], return_df=True).iloc[0]
na = 24; na1 = 24
nb = 0; nb1 = 0
alpha = 0.01; alpha1 = 0.01
DOF = N - na - nb; DOF1 = 36878
chi_critical = chi2.ppf(1 - alpha, DOF); chi_critical1 = 37812.73089274517
if lb_stat < chi_critical:
    print('The residual is white.')
if lb_stat > chi_critical:
    print('The residual is not white.')
```

Figure 34

36

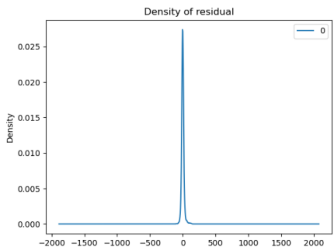


Figure 35

37

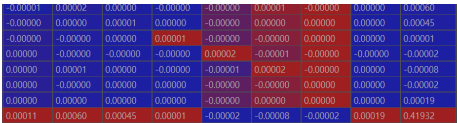


Figure 36

38

variance of forecast error of ARIMA is 361.58

Figure 37

38

```
>>> trainScore
46.89987581733659
>>> testScore
45.09088116733351
```

Figure 38

39

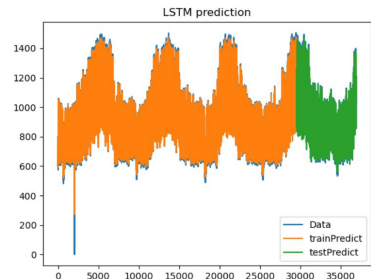


Figure 39

40

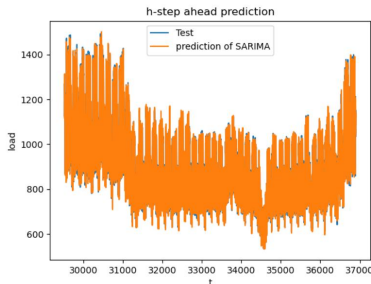


Figure 40

41

Abstract

Electricity and energy are essential for the development and survival of a country. It matters whether the government can sustain the livelihood of its people and whether the country is strong enough. This report will investigate my country of birth, Taiwan, by tracing the electricity generated in Taiwan under different seasons, time of day, and weather and climate factors through two different data sets to predict the future development of electricity in Taiwan under various aspects. Especially recent days, no matter energy policies of Government or issues about green energy are hot topics of news and discussions. So, for this reason I decided to dive into the Electricity energy of my country, and to find out whether there is a relationship between electricity load and weather. We will use the shallow learning model as our prediction model and the most suitable normal distribution of the original least square to determine the proportion of different features to the predicted coefficients. Therefore, the government of Taiwan can use this prediction for energy and electricity allocation. And it can know more about how to use other energy sources when the electricity cannot meet the needs of the people at certain times.

Introduction

For this final term project, we are going to go through data preprocessing, testing of data stationary, time series decomposition, Holt-Winter method, feature selection, base-models, developing multiple linear regression models, ARMA model determination, LM algorithm,

diagnostic analysis, deep learning model, final model selection, forecast function, h-step prediction, and conclusion of the project.

Description of Dataset

For the dataset used in this project is downloaded from Kaggle, [Energy data in Taiwan \(Taipower\)](#), which contains the electricity load data and the weather data of Taiwan from 2017 to 2021. The dataset includes one time series column, two categorical variables, and eighteen numerical variables. After merging the load dataset and weather dataset, there are 14763 observations for the raw dataset and for this project the target is the load of electricity. The other independent features are ‘datetime’ (Recorded in format of Year-Month-Date Hour:Minute:Second) , ‘StnPres’ (station pressure), ‘SeaPres’ (sea-level pressure), ‘Temperature’, ‘Td dew point’ (dew point), ‘RH’ (relative humidity), ‘WS’ (wind speed), ‘WD’ (wind direction), ‘WSGust’ (wind speed gust), ‘WDGust’ (wind direction gust), ‘Precp’ (precipitation), ‘PrecpHour’ (precipitation per hour), ‘Sunshine’ (sunshine per hour), ‘GloblRad’ (global solar irradiation on horizontal plane), ‘Visb’ (visibility), ‘UVI’ (Ultra-violet index), ‘Cloud amount’, ‘station’, and ‘area’.

a. Preprocessing

First, I checked on the percentage of the NA value of each variables and decided to drop the ‘Visb’, ‘Sunshine’, and ‘Cloud amount’ columns because the percentage of missing value in these variables is too large.

datetime	0
StnPres	26
SeaPres	26
Temperature	3
Td dew point	960
RH	960
WS	14
WD	368
WSGust	15
WDGust	15
Precp	5778
PrecpHour	8
SunShine	61636
GloblRad	430
Visb	93607
UVI	555
Cloud Amount	78475
station	0
area	0
load	0

Figure 1

Figure 1: The amount of NA values in each variable

Next, I check on the distribution of each variables which contains NA value. For example, most values of the 'Precp' column are zero so I decide to fill the missing values with zero.

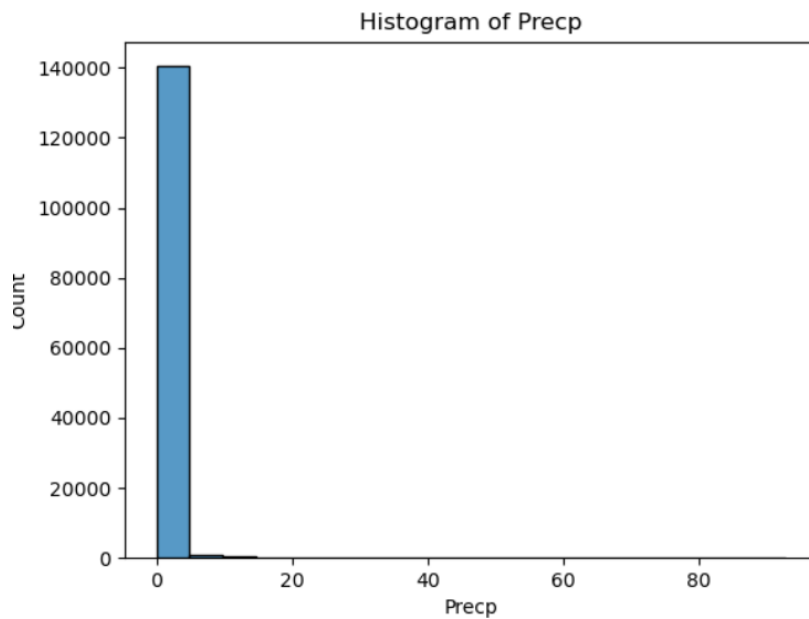


Figure 2

Figure 2: Histogram of Precp column

Using the same method to check on other variables, I decided to fill the rest of them with the most frequent value of their column.

b. Dependent variable versus time

Due to the reason that the dataset included four areas data using the same time index, so I separate them and reindex the datasets with time.

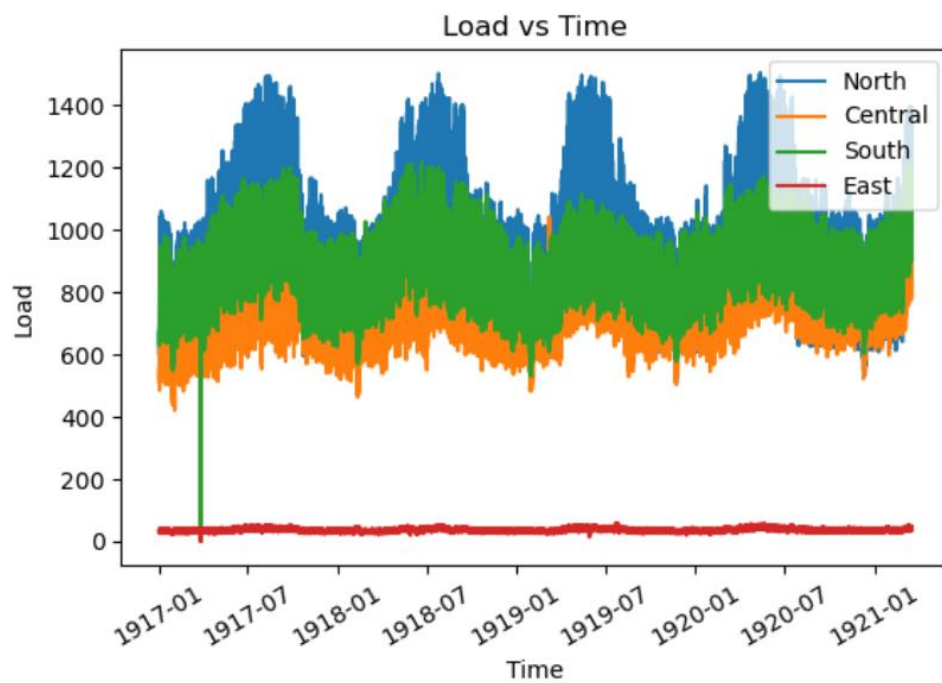


Figure 3

Figure 3: Time versus load

c. ACF/PACF of the dependent variable

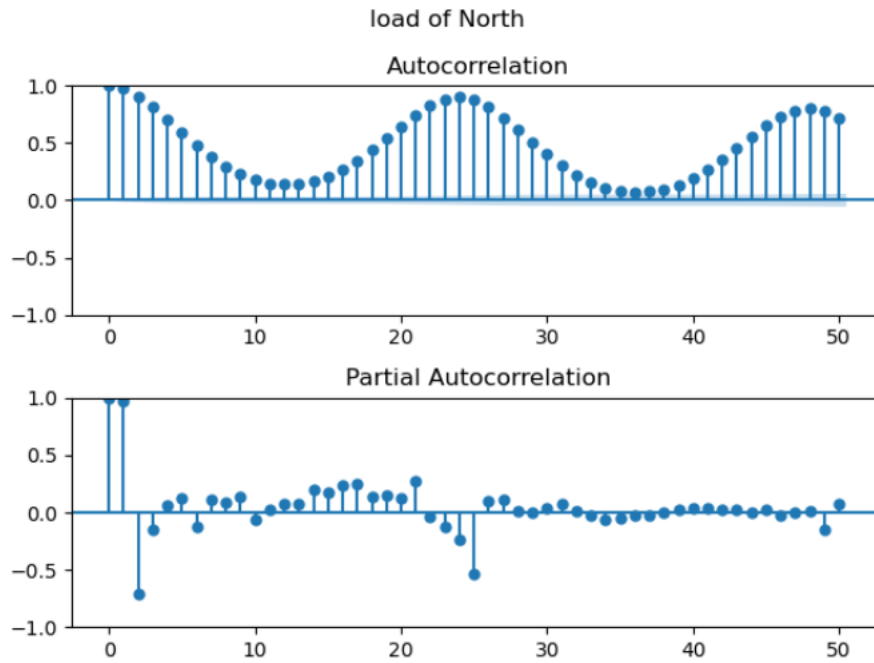


Figure 4

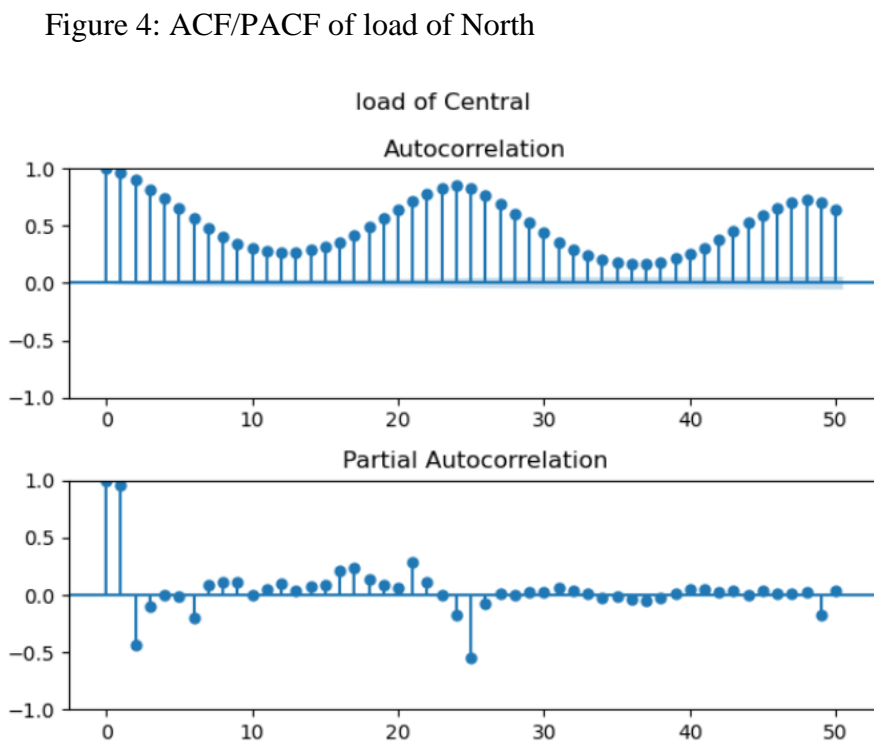


Figure 5

Figure 5: ACF/PACF of load of Central

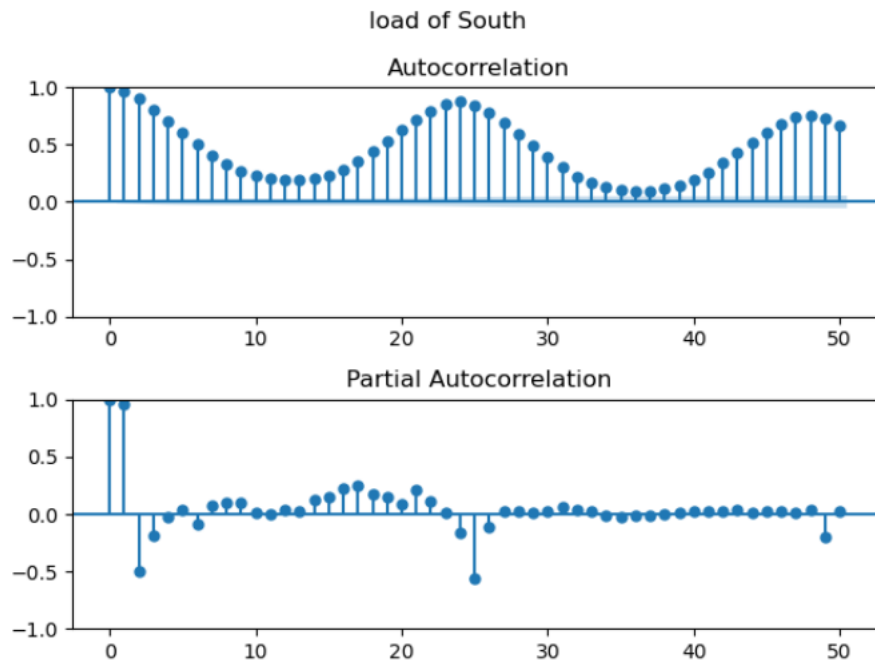


Figure 6

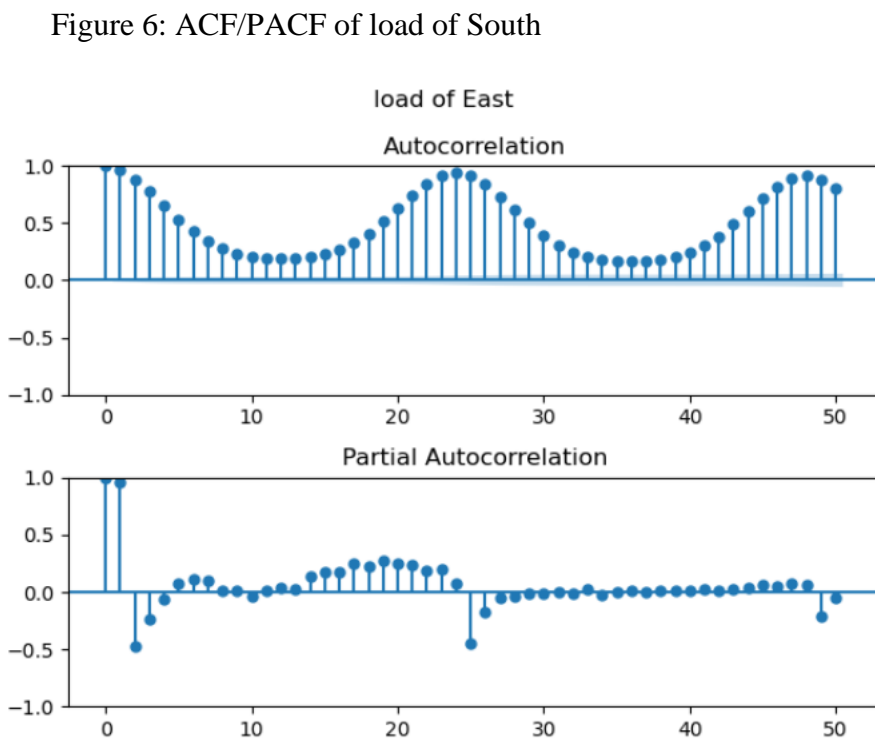
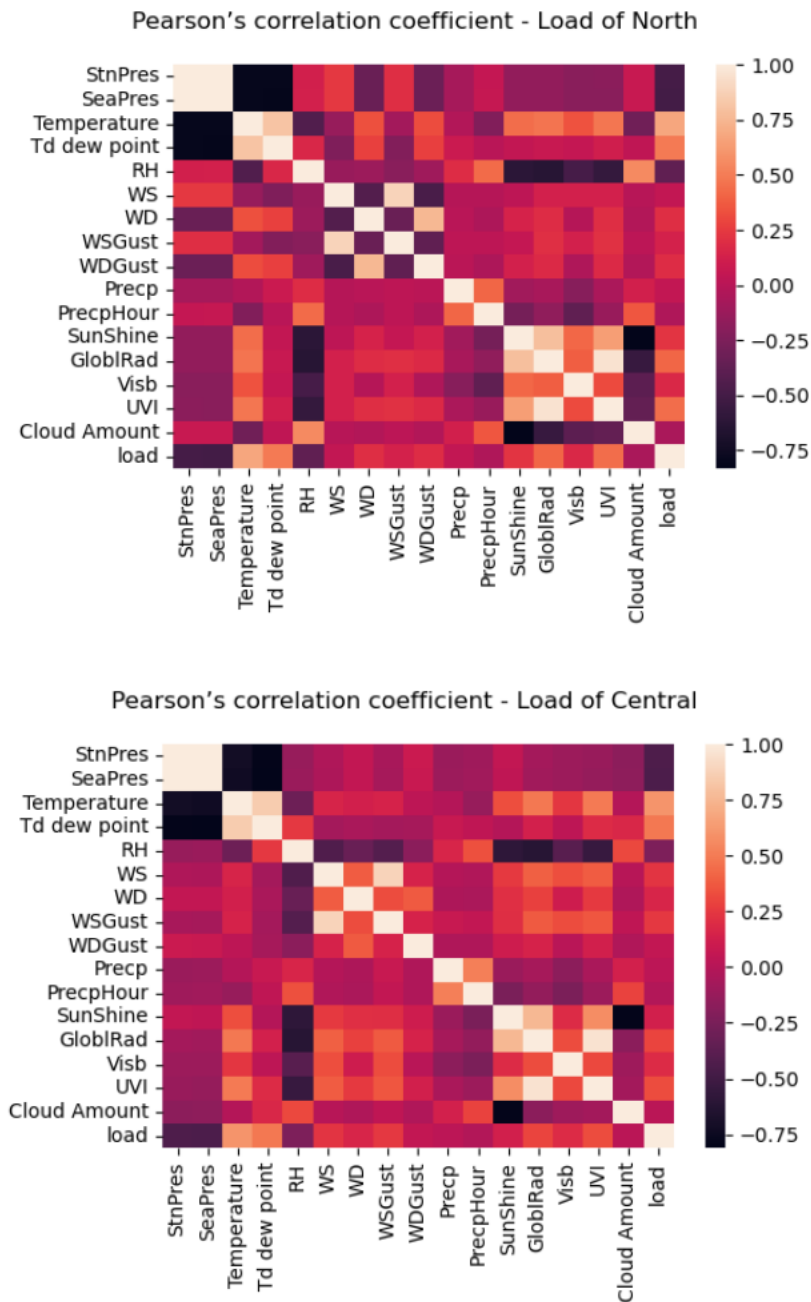


Figure 7

Figure 7: ACF/PACF of load of East

We can see that these four areas have similar ACF/PACF plots.

d. Correlation matrix



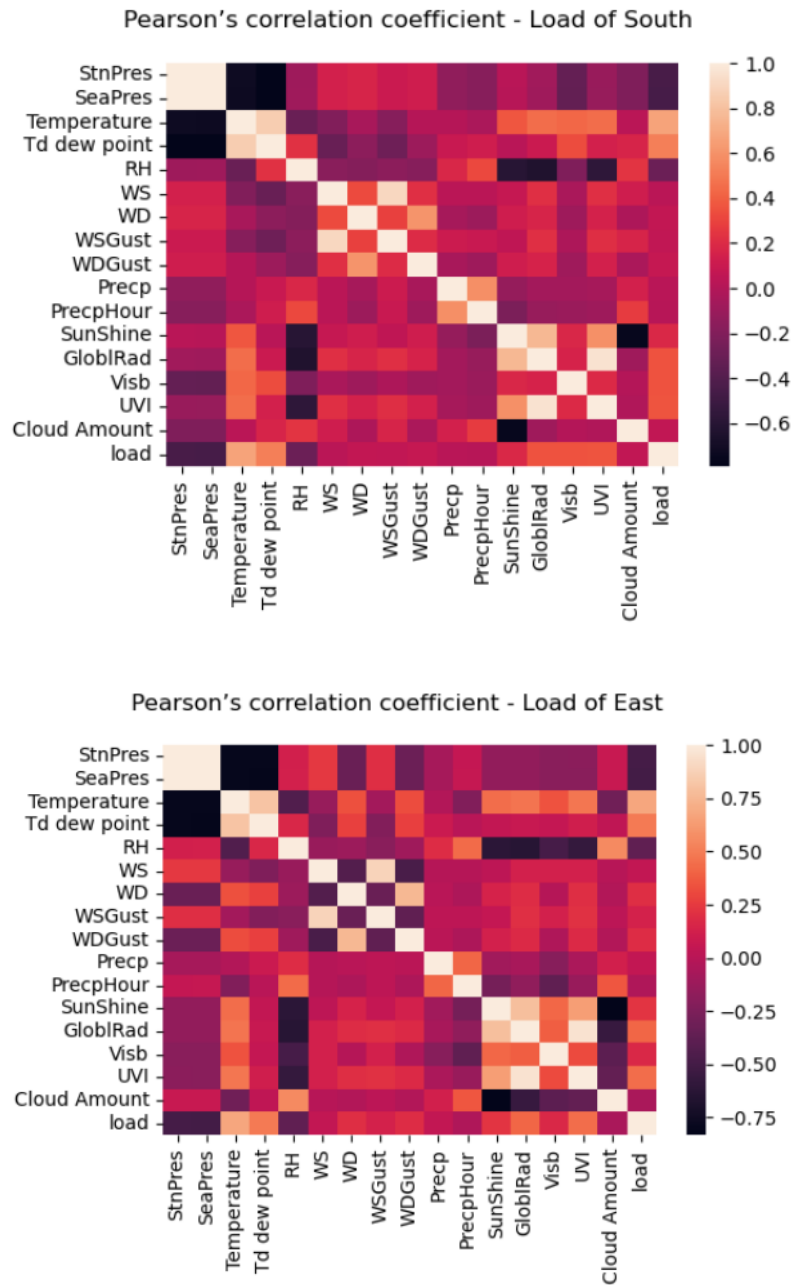


Figure 8

Figure 8: The Pearson's correlation coefficient plot of load of each area

We can see that there is still no big difference between each area so I decided to use the data of North area as the main dataset for the rest of the project research.

- e. Splitting dataset into train (80%) and test (20%)

```
train = df[:int(len(df)*80/100)]  
test = df[int(len(df)*80/100):]
```

Stationarity

ADF-test:

For the ADF-test, the p-value of it is close to zero and which means that the data is possible stationary.

```
ADF Statistic: -10.771032  
p-value: 0.000000  
Critical Values:  
  1%: -3.431  
  5%: -2.862  
 10%: -2.567
```

Figure 9

Figure 9: The result of ADF-test

KPSS-test:

For the KPSS-test, the p-value of it is 0.044856 and which is a little bit lower than 0.05. If the data is stationary the p-value of the KPSS-test should be higher than 0.05 but for now the gap of it is still acceptable. We will be moving on to check on the rolling mean and variance to get the final answer.

```
KPSS test of load
Results of KPSS Test:
Test Statistic      0.485837
p-value             0.044856
Lags Used           92.000000
Critical Value (10%) 0.347000
Critical Value (5%)  0.463000
Critical Value (2.5%) 0.574000
Critical Value (1%)  0.739000
```

Figure 10

Figure 10: The result of KPSS-test

Rolling mean and variance:

The rolling mean and variance shows that the mean and variance become stationary at the end which means that the data is probably stationary.

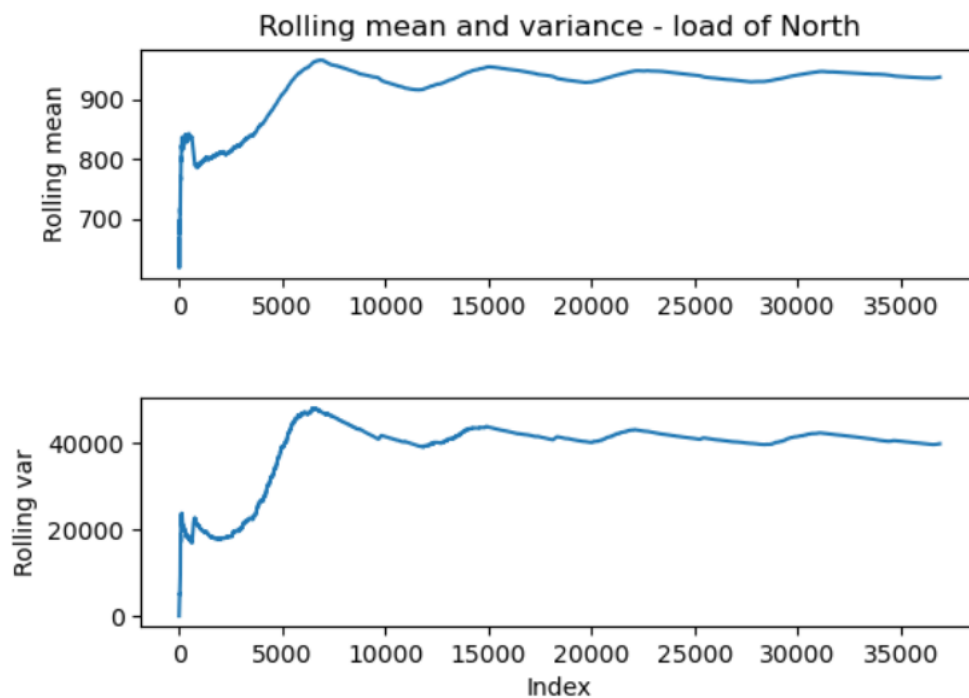


Figure 11

Figure 11: Rolling mean and variance of load of North

Time series Decomposition

For checking the strength of trending and seasonality, I fit the data with STL model (Seasonal-Trend decomposition using LoSes) of the statsmodels.tsa. By looking at the original data plot, we can see that there are trending and seasonality exist but there is another seasonality exist in the trending which means there are at least two layers of seasonality.

Next, checking on the trending part, we can see that except the original seasonality, the trending is also seasonal with a period of year so if we want to detrend the dataset we should detrend it by years to make the result fit our expectation.

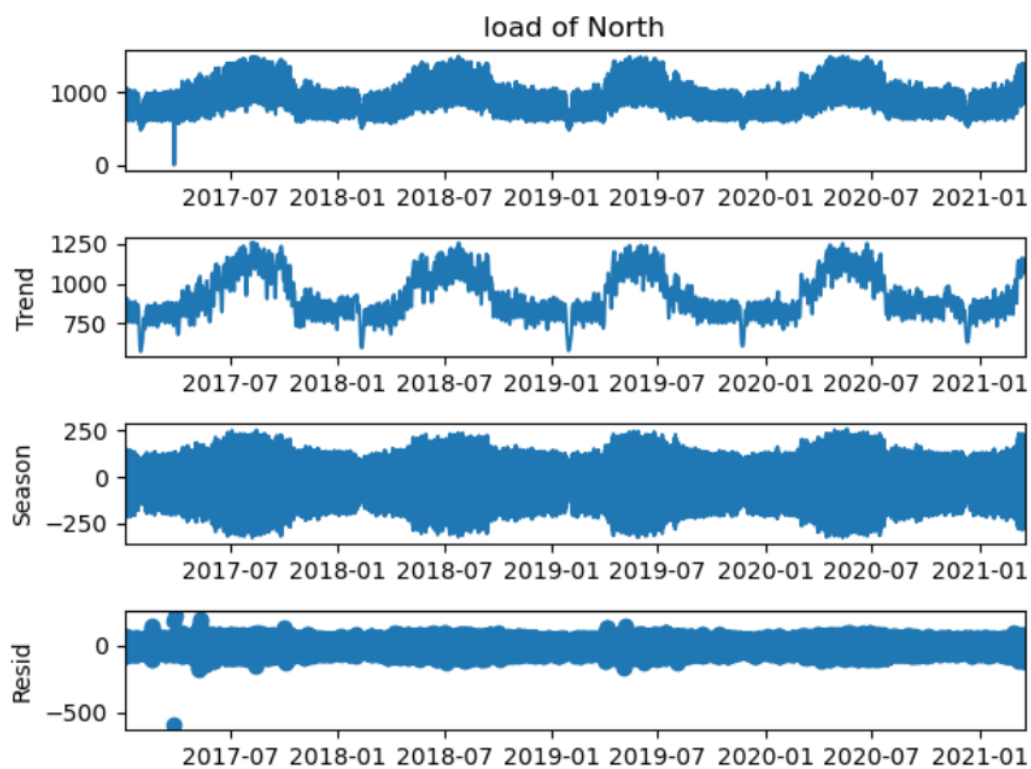


Figure 12

Figure 12: The result of trending and seasonality of the original data

```
The strength of trend for this data set is 0.96  
The strength of seasonality for this data set is 0.96  
The dependent variable of area of North is highly trend and seasonal.
```

Figure 13

Figure 13: Result of fitting data into STL model

Holt-Winters Method

By fitting the train data into the Holt-Winter model, I used the ets function from the statsmodels.tsa.holtwinters, and make a prediction using the test dataset index.

```
# Holt-Winter Method  
import statsmodels.tsa.holtwinters as ets  
  
y_HW = df_test.copy() y_HW: datetime StnPres SeaPres ... area Load HW_fit [29521 2020-07-18 10:00:00 1004.4 1007.8  
fitted_model = ets.ExponentialSmoothing(df_train['load'], trend='_add', seasonal='_add', seasonal_periods=24, damped_trend=True).fit()  
y_HW['HW_fit'] = fitted_model.forecast(len(df_test))
```

Figure 14

Figure 14: parameters of fitting Holt-Winter model

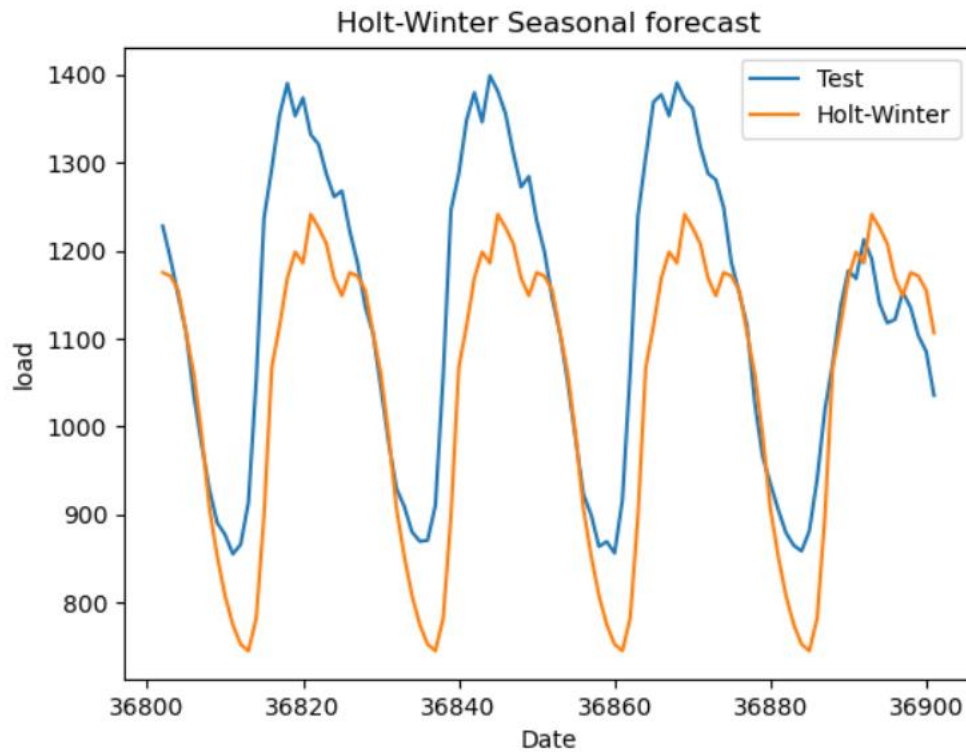


Figure 15

Figure 15: Last 100 data versus prediction of Holt-Winter method

Feature Selection

For the feature selection section, first we get the singular values and the condition number of the data to check if there is collinearity exist. As the figure followed, we can notice that the smallest number of SVD is $2.66e + 01$ which is not too close to zero, and probably can infer that as the sign of there is no significant collinearity exist. However, the condition number of data is 64299.29 and is way higher than expectation.

```
singular values of Data is [1.10085108e+11 1.07393352e+09 4.57606427e+08 7.04233201e+07
6.34703171e+06 1.11131338e+06 3.24755246e+05 1.27244112e+05
8.85091626e+04 1.28975600e+04 4.61518720e+03 2.24142505e+03
2.14527097e+03 2.66266244e+01]
The condition number of Data is 64299.29
```

Figure 16

Figure 16: SVD and condition number result of the original data

Next, I fit the data into the OLS model (ordinary least square) to perform a backward stepwise regression. I'm going to drop the variables with the highest p-value one by one until there is no variable with a p-value higher than 0.05.

```
>>> print(model.summary())
```

OLS Regression Results						
Dep. Variable:	load	R-squared (uncentered):	0.979			
Model:	OLS	Adj. R-squared (uncentered):	0.979			
Method:	Least Squares	F-statistic:	1.339e+05			
Date:	Wed, 04 May 2022	Prob (F-statistic):	0.00			
Time:	04:32:12	Log-Likelihood:	-2.3419e+05			
No. Observations:	36902	AIC:	4.684e+05			
Df Residuals:	36889	BIC:	4.685e+05			
Df Model:	13					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
StnPres	-44.6445	18.949	-2.356	0.018	-81.785	-7.504
SeaPres	44.9155	18.878	2.379	0.017	7.914	81.917

Figure 17

Temperature	22.7141	1.460	15.560	0.000	19.853	25.575
Td dew point	0.5829	1.532	0.380	0.704	-2.420	3.586
RH	-1.8094	0.350	-5.170	0.000	-2.495	-1.123
WS	-9.0741	1.136	-7.984	0.000	-11.302	-6.847
WD	0.0011	0.013	0.083	0.934	-0.025	0.027
WSGust	13.8920	0.502	27.701	0.000	12.909	14.875
WDGust	0.0544	0.013	4.351	0.000	0.030	0.079
Precp	3.3120	0.485	6.831	0.000	2.362	4.262
PrecpHour	91.8878	2.865	32.071	0.000	86.272	97.504
GloblRad	-52.8786	2.767	-19.109	0.000	-58.302	-47.455
UVI	21.2364	0.943	22.523	0.000	19.388	23.084
=====						
Omnibus:	812.279	Durbin-Watson:	0.117			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	425.472			
Skew:	0.014	Prob(JB):	4.07e-93			
Kurtosis:	2.475	Cond. No.	5.39e+04			

=====

Notes:

[1] R² is computed without centering (uncentered) since the model does not contain a constant

[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[3] The condition number is large, 5.39e+04. This might indicate that there are strong multicollinearity or other numerical problems.

Figure 17: model summary of the first OLS model

After dropping ‘WD’ and ‘Td dew point’, there is no variable has a p-value larger than 0.05 so this is the final model for the feature selection.

	coef	std err	t	P> t	[0.025	0.975]
StnPres	-44.5004	18.943	-2.349	0.019	-81.628	-7.373
SeaPres	44.7604	18.871	2.372	0.018	7.772	81.749
Temperature	23.2610	0.269	86.402	0.000	22.733	23.789
RH	-1.6795	0.076	-22.033	0.000	-1.829	-1.530
WS	-9.0689	1.133	-8.008	0.000	-11.289	-6.849
WSGust	13.8904	0.501	27.702	0.000	12.908	14.873
WDGust	0.0548	0.010	5.736	0.000	0.036	0.074
Precp	3.3060	0.485	6.822	0.000	2.356	4.256
PrecpHour	91.6654	2.805	32.683	0.000	86.168	97.163
GloblRad	-53.0215	2.741	-19.343	0.000	-58.394	-47.649
UVI	21.2252	0.942	22.528	0.000	19.379	23.072

Figure 18

Figure 18: information of the final model summary

After dropping these features, we perform the SVD and condition number again and we can see that the condition drops, which means the data will perform better for the future prediction.

```
singular values of Data is [1.09380125e+11 1.06837683e+09 2.87918824e+08 6.19952009e+06
6.18792655e+05 3.18331618e+05 1.25258170e+05 8.84896518e+04
1.29570013e+04 2.39158464e+03 2.24117683e+03 2.66433322e+01]
The condition number of Data is 64072.99
```

Figure 19

Figure 19: SVD and condition number result of the data after dropping features

However, after dropping two features there is no more features will be consider not significant for the model but the condition number still shows that there might be some features

should be dropped. So I tried to fit the data into random forest model and perform a feature importance of the dataset. As we can see that except temperature has a significant importance for the random forest model, other features have similar low importance. If we decided to remove more features in the future, we can also take a consider of the result of the random forest model too.

Variable: Temperature	Importance: 0.59
Variable: WSGust	Importance: 0.06
Variable: Td dew point	Importance: 0.05
Variable: WS	Importance: 0.04
Variable: GloblRad	Importance: 0.04
Variable: StnPres	Importance: 0.03
Variable: SeaPres	Importance: 0.03
Variable: RH	Importance: 0.03
Variable: WD	Importance: 0.03
Variable: WDGust	Importance: 0.03
Variable: UVI	Importance: 0.03
Variable: PrecpHour	Importance: 0.02
Variable: Precp	Importance: 0.01
Variable: Temperature	Importance: 0.59
Variable: WSGust	Importance: 0.06
Variable: Td dew point	Importance: 0.05

Variable: WS	Importance: 0.04
Variable: GloblRad	Importance: 0.04
Variable: StnPres	Importance: 0.03
Variable: SeaPres	Importance: 0.03
Variable: RH	Importance: 0.03
Variable: WD	Importance: 0.03
Variable: WDGust	Importance: 0.03
Variable: UVI	Importance: 0.03
Variable: PrecpHour	Importance: 0.02
Variable: Precp	Importance: 0.01

Figure 20

Figure 20: feature importance generate through random forest model

Base-Models

AFM (Average forecast method):

The MSE of AFM prediction is 35258.

Naïve :

The MSE of Naïve prediction is 47488.

Drift:

The MSE of Drift prediction is 64305.

SES ($\alpha = 0.5$):

The MSE of SES prediction (with $\alpha = 0.5$) is 47488.

```
MSE of average forecast is 35258.26
```

```
MSE of Naive forecast is 47488.06
```

```
MSE of drift forecast is 64305.89
```

```
MSE of SES forecast is 47488.06
```

Figure 21

Figure 21: MSE of h-step prediction of each model versus test data

Multiple Linear Regression

For the multiple linear regression section, I used the OLS model fitted in the feature selection section. The R-square and the adjusted R-square of this model is 0.979, and the AIC is $4.684e+05$ and the BIC is $4.685e+05$.

OLS Regression Results						
=====						
Dep. Variable:	load	R-squared (uncentered):	0.979			
Model:	OLS	Adj. R-squared (uncentered):	0.979			
Method:	Least Squares	F-statistic:	1.583e+05			
Date:	Wed, 04 May 2022	Prob (F-statistic):	0.00			
Time:	06:26:13	Log-Likelihood:	-2.3419e+05			
No. Observations:	36902	AIC:	4.684e+05			
Df Residuals:	36891	BIC:	4.685e+05			
Df Model:	11					
Covariance Type:	nonrobust					
=====						
	coef	std err	t	P> t	[0.025	0.975]

StnPres	-44.5004	18.943	-2.349	0.019	-81.628	-7.373
SeaPres	44.7604	18.871	2.372	0.018	7.772	81.749
Temperature	23.2610	0.269	86.402	0.000	22.733	23.789
RH	-1.6795	0.076	-22.033	0.000	-1.829	-1.530
WS	-9.0689	1.133	-8.008	0.000	-11.289	-6.849
WSGust	13.8904	0.501	27.702	0.000	12.908	14.873
WDGust	0.0548	0.010	5.736	0.000	0.036	0.074
Precp	3.3060	0.485	6.822	0.000	2.356	4.256
PrecpHour	91.6654	2.805	32.683	0.000	86.168	97.163
GloblRad	-53.0215	2.741	-19.343	0.000	-58.394	-47.649
UVI	21.2252	0.942	22.528	0.000	19.379	23.072
=====						
Omnibus:	812.130	Durbin-Watson:	0.117			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	425.384			
Skew:	0.013	Prob(JB):	4.26e-93			
Kurtosis:	2.475	Cond. No.	5.36e+04			

Figure 22

Figure 22: linear model summary

```

AIC of model2 468402.41
BIC of model2 468496.09
Adjusted R-square of model2 0.98
Adjusted R-square of model2 0.98

```

Figure 23

Figure 23: statistic criteria result of the model

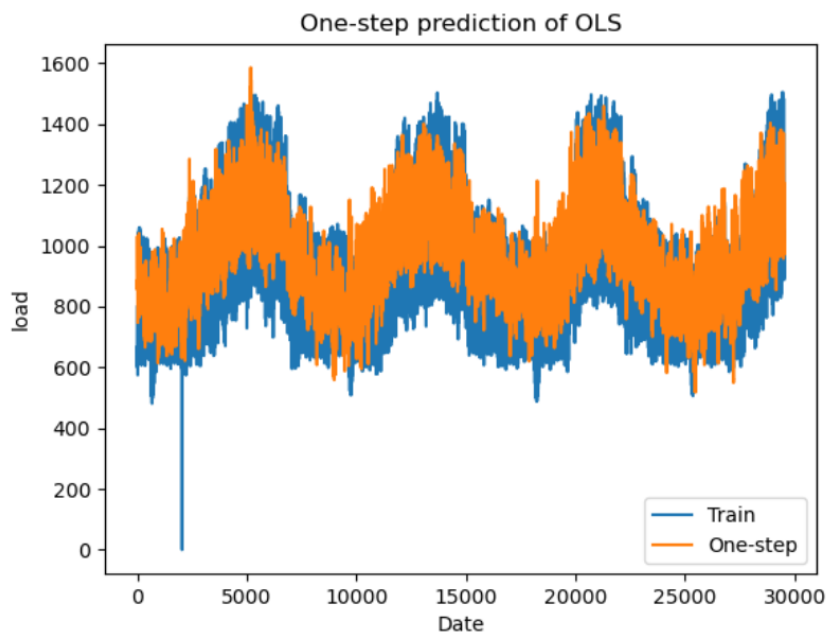


Figure 24

Figure 24: the one-step prediction of model

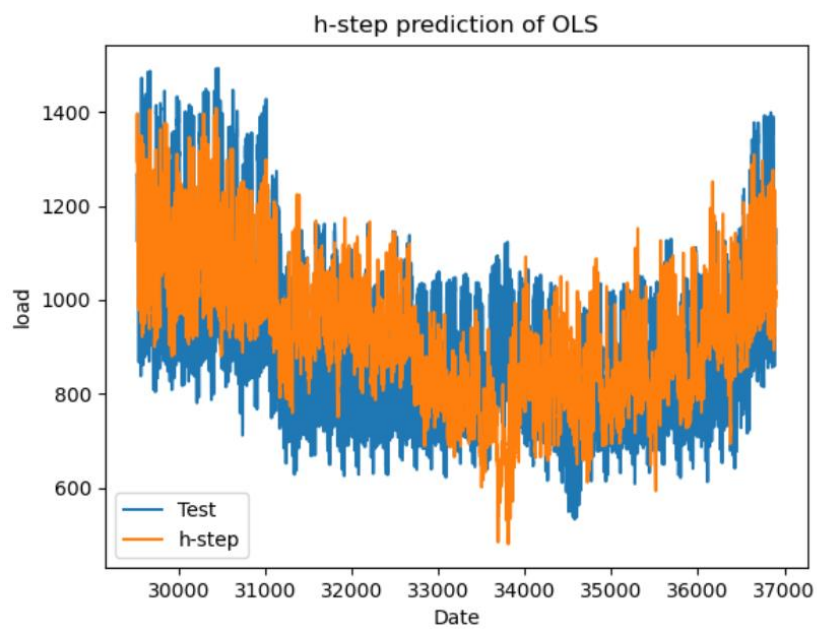


Figure 25

Figure 25: the h-step prediction of model

The MSE of OLS model prediction is 20216.

Mean of residual and variance of residual:

The mean of the residual is -4.99 and the variance of it is 18729.

ACF of residual:

```
>>> acf[:20]
array([1.          , 0.93931564, 0.84119466, 0.71667633, 0.58525414,
       0.46391368, 0.35170343, 0.25611442, 0.17994846, 0.12726441,
       0.09486763, 0.07623483, 0.06638405, 0.05922439, 0.06002367,
       0.07208657, 0.10177334, 0.15474577, 0.22537861, 0.31257999])
```

Figure 26

Figure 26: first 20 ACF values of the residual

Q-value:

The Q-value with lags 20 will be 6662521.

F-test & T-test:

We can notice that the p-value of the F-test is close to zero, which means that there is a significant difference between our linear regression model and a random model.

```

... print(model2.f_test(A))
... print(model2.t_test(A))
<F test: F=array([[4292.49971797]]), p=0.0, df_denom=3.69e+04, df_num=10>
      Test for Constraints
=====
      coef      std err          t      P>|t|      [0.025      0.975]
-----
c0         44.7604      18.871         2.372      0.018         7.772      81.749
c1         23.2610       0.269      86.402      0.000        22.733      23.789
c2         -1.6795       0.076     -22.033      0.000        -1.829     -1.530
c3         -9.0689       1.133      -8.008      0.000       -11.289     -6.849
c4         13.8904       0.501      27.702      0.000        12.908      14.873
c5          0.0548       0.010       5.736      0.000         0.036       0.074
c6          3.3060       0.485       6.822      0.000         2.356       4.256
c7         91.6654       2.805      32.683      0.000        86.168      97.163
c8        -53.0215       2.741     -19.343      0.000       -58.394     -47.649

```

Figure 27

Figure 27: results of F-test & t-test

ARMA, ARIMA, and SARIMA Model

Although we have test the data stationarity in the previous section but we can't one hundred percent sure that the data is stationary, and if the data is not stationary the testing of ARMA model and LM model wouldn't fit. To make sure the data is stationary we should process it again.

For the first sight of the GPAC table of the original data, there is no significant sign for ARMA order, there might be one for the AR (the value for first column is close to 0.8) but not too significant.

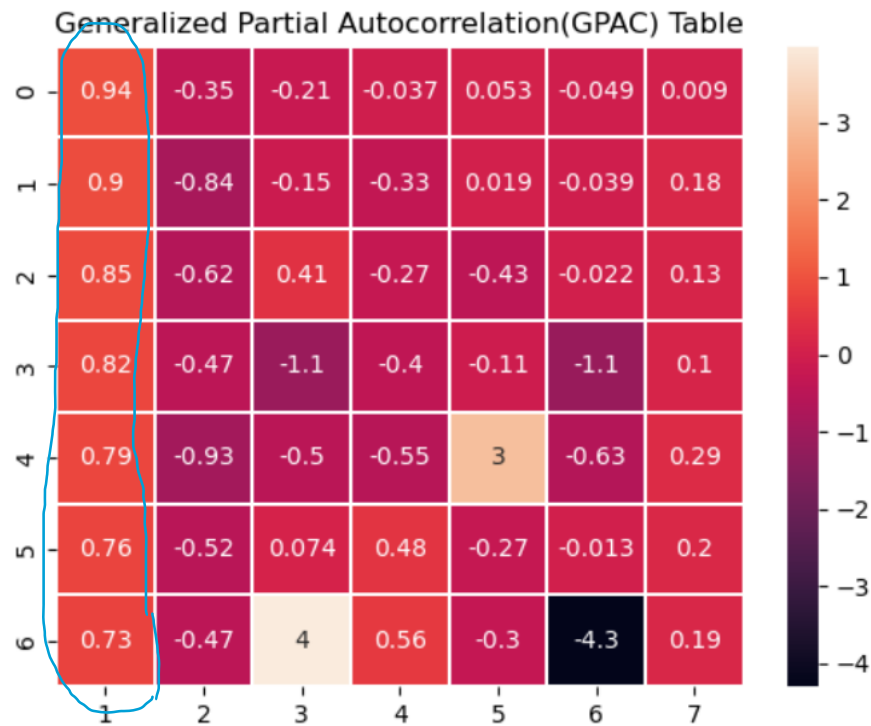


Figure 28

Figure 28: GPAC table of original data

First, by observing the ACF/PACF plot that generated in the previous section, through ACF I'll guess that there might be a 24 lags period of the data, and by PACF I'll guess there might be an order of AR. So I tried to make a 24 period difference of the data.

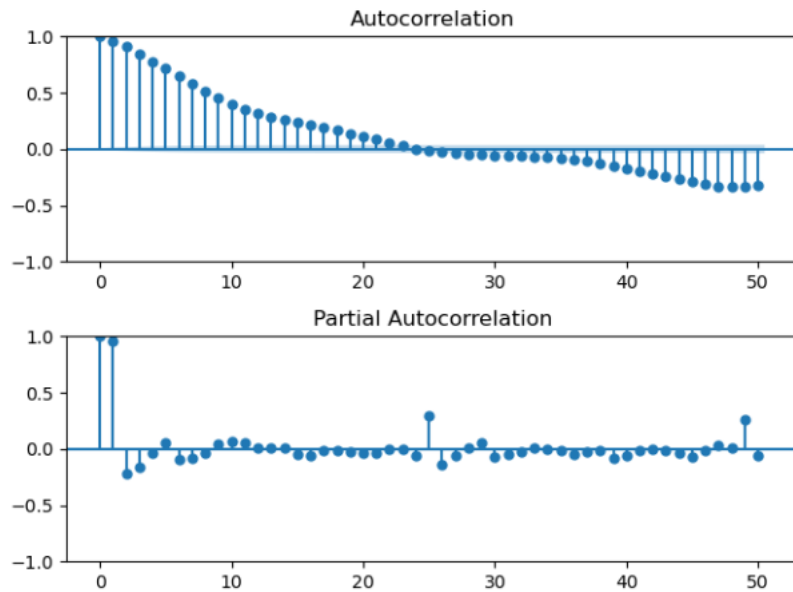


Figure 29

Figure 29: ACF/PACF after 24 period difference

As we can see, the dataset is highly similar to a model with AR order, then I tried to make a 1 period difference.

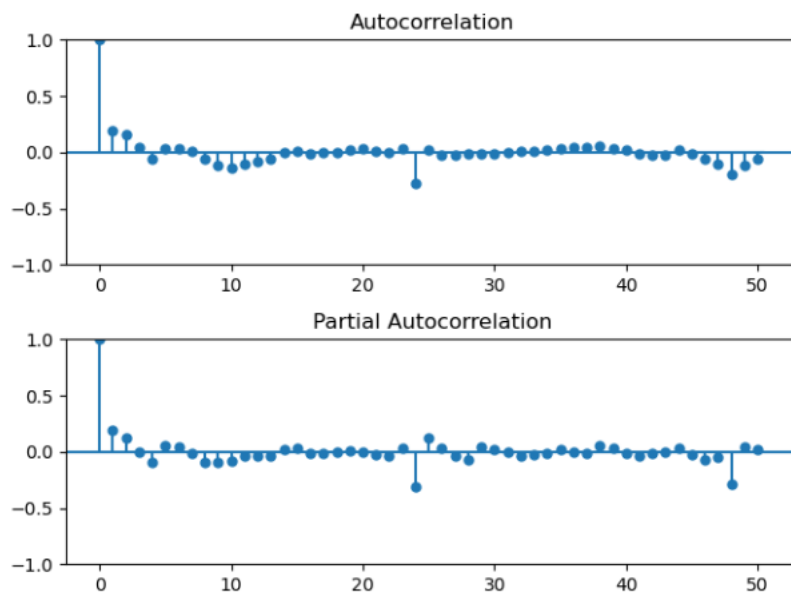


Figure 30

Figure 30: second time difference, ACF/PACF after 1 period difference

From the new GPAC table, we can't find any ARMA order.

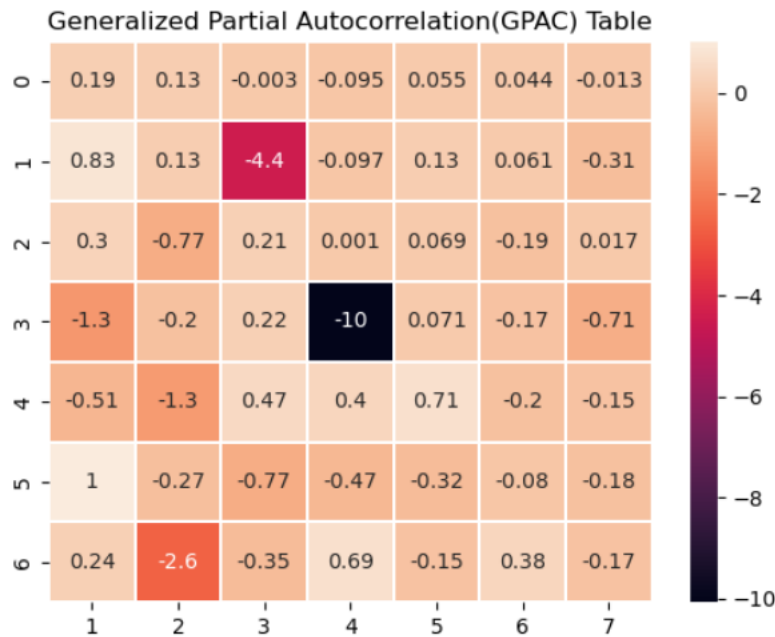


Figure 31

Figure 31: GPAC table after processing

Hence, I'll guess that the original data might be a SARIMA(0,1,0)24 or SARIMA(1,0,0)24 data.

Levenberg Marquardt Algorithm

As I mentioned in the previous section, the model might be SARIMA(0,1,0)24 or SARIMA(1,0,0) 24. By fitting the data into the ARIMA model with the order of (24,1,0) will have a similar result. We can see that except AR L1 and L24, the other coefficient is nearly 0, which pretty fit our expectation.

SARIMAX Results						
=====						
Dep. Variable:	load	No. Observations:	29521			
Model:	ARIMA(24, 1, 0)	Log Likelihood	-133058.323			
Date:	Wed, 04 May 2022	AIC	266166.647			
Time:	09:41:34	BIC	266373.967			
Sample:	0	HQIC	266233.219			
	- 29521					
Covariance Type:	opg					
=====						
	coef	std err	z	P> z	[0.025	0.975]

ar.L1	0.1592	0.001	148.580	0.000	0.157	0.161
ar.L2	0.0440	0.003	14.652	0.000	0.038	0.050
ar.L3	-0.0293	0.003	-9.886	0.000	-0.035	-0.023
ar.L4	-0.1626	0.003	-64.726	0.000	-0.167	-0.158
ar.L5	0.0460	0.003	16.802	0.000	0.041	0.051
ar.L6	-0.0209	0.004	-5.048	0.000	-0.029	-0.013
ar.L7	-0.0298	0.004	-7.548	0.000	-0.038	-0.022
ar.L8	-0.1086	0.004	-29.570	0.000	-0.116	-0.101
ar.L9	-0.0322	0.004	-7.310	0.000	-0.041	-0.024
ar.L10	-0.0322	0.005	-6.054	0.000	-0.043	-0.022
ar.L11	-0.0347	0.006	-6.094	0.000	-0.046	-0.024
ar.L12	-0.0265	0.006	-4.273	0.000	-0.039	-0.014
ar.L13	-0.0553	0.006	-9.869	0.000	-0.066	-0.044
ar.L14	-0.0083	0.005	-1.533	0.125	-0.019	0.002
ar.L15	-0.0146	0.004	-3.374	0.001	-0.023	-0.006
ar.L16	-0.1000	0.004	-25.141	0.000	-0.108	-0.092
ar.L17	-0.0394	0.005	-7.718	0.000	-0.049	-0.029
ar.L18	-0.0355	0.005	-7.182	0.000	-0.045	-0.026
ar.L19	0.0117	0.004	3.322	0.001	0.005	0.019
ar.L20	-0.0994	0.003	-33.585	0.000	-0.105	-0.094
ar.L21	-0.0009	0.004	-0.211	0.833	-0.009	0.007
ar.L22	0.0080	0.005	1.769	0.077	-0.001	0.017
ar.L23	0.0840	0.002	51.365	0.000	0.081	0.087

```

ar.L24      0.5305      0.001      470.394      0.000      0.528      0.533
sigma2      481.3905      0.648      743.400      0.000      480.121      482.660
=====
Ljung-Box (L1) (Q):          68.98      Jarque-Bera (JB):          120091921.60
Prob(Q):          0.00      Prob(JB):          0.00
Heteroskedasticity (H):          0.58      Skew:          2.69
Prob(H) (two-sided):          0.00      Kurtosis:          315.42
=====

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

```

Figure 32

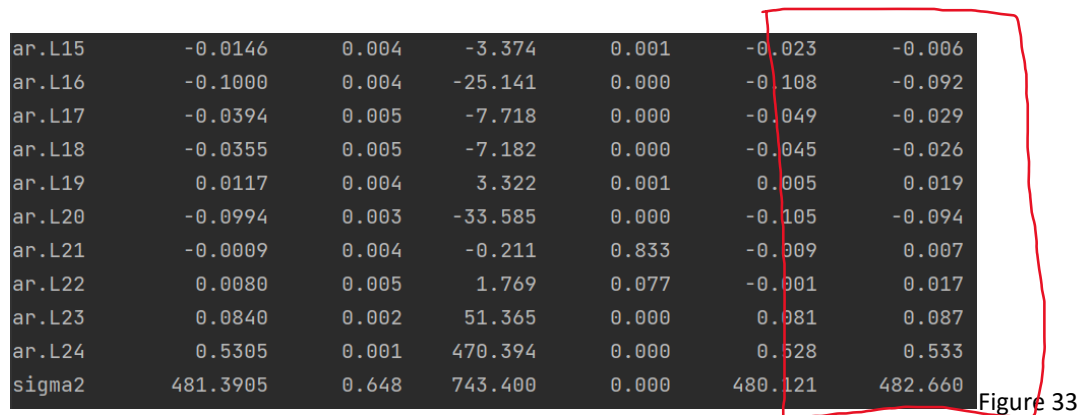
Figure 32: result of fitting data with ARIMA(24,1,0) model

Diagnostic Analysis

- a. Diagnostic test: confidence interval, zero/pole cancellation, and chi-square test

Confidence interval:

	coef	std err	z	P> z	[0.025	0.975]
ar.L1	0.1592	0.001	148.580	0.000	0.157	0.161
ar.L2	0.0440	0.003	14.652	0.000	0.038	0.050
ar.L3	-0.0293	0.003	-9.886	0.000	-0.035	-0.023
ar.L4	-0.1626	0.003	-64.726	0.000	-0.167	-0.158
ar.L5	0.0460	0.003	16.802	0.000	0.041	0.051
ar.L6	-0.0209	0.004	-5.048	0.000	-0.029	-0.013
ar.L7	-0.0298	0.004	-7.548	0.000	-0.038	-0.022
ar.L8	-0.1086	0.004	-29.570	0.000	-0.116	-0.101
ar.L9	-0.0322	0.004	-7.310	0.000	-0.041	-0.024
ar.L10	-0.0322	0.005	-6.054	0.000	-0.043	-0.022
ar.L11	-0.0347	0.006	-6.094	0.000	-0.046	-0.024
ar.L12	-0.0265	0.006	-4.273	0.000	-0.039	-0.014
ar.L13	-0.0553	0.006	-9.869	0.000	-0.066	-0.044
ar.L14	-0.0083	0.005	-1.533	0.125	-0.019	0.002



ar.L15	-0.0146	0.004	-3.374	0.001	-0.023	-0.006
ar.L16	-0.1000	0.004	-25.141	0.000	-0.108	-0.092
ar.L17	-0.0394	0.005	-7.718	0.000	-0.049	-0.029
ar.L18	-0.0355	0.005	-7.182	0.000	-0.045	-0.026
ar.L19	0.0117	0.004	3.322	0.001	0.005	0.019
ar.L20	-0.0994	0.003	-33.585	0.000	-0.105	-0.094
ar.L21	-0.0009	0.004	-0.211	0.833	-0.009	0.007
ar.L22	0.0080	0.005	1.769	0.077	-0.001	0.017
ar.L23	0.0840	0.002	51.365	0.000	0.081	0.087
ar.L24	0.5305	0.001	470.394	0.000	0.528	0.533
sigma2	481.3905	0.648	743.400	0.000	480.121	482.660

Figure33: Confidence interval of the parameters

We can notice that from ar.L2 to ar.L23 are all close to zero which means that they might be insignificant at all.

zero/pole cancellation:

There is no zero/pole cancellation because there is only AR order exist.

Chi-square test:

```

N = len(df)
lb_stat, pvalue = sm.stats.acorr_ljungbox(errortrain1[1:], lags=[20], return_df=True).iloc[0]
na = 24    na: 24
nb = 0    nb: 0
alfa = 0.01    alfa: 0.01
DOF = N - na - nb    DOF: 36878
chi_critical = chi2.ppf(1 - alfa, DOF)    chi_critical: 37512.73059274517
if lb_stat < chi_critical:
    print('The residual is white.')
if lb_stat > chi_critical:
    print('The residual is not white.')

```

Figure 34

Figure 34: process of chi-square test

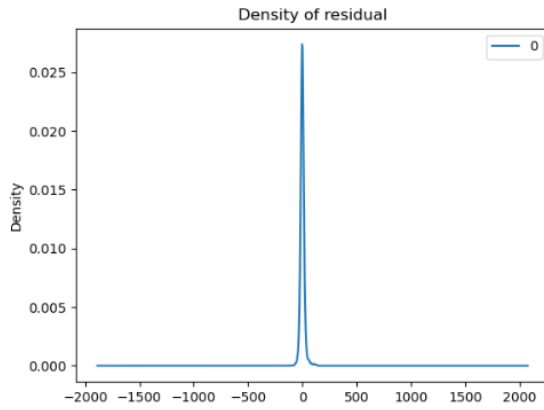


Figure 35

Figure 35: Density of residual

b. Estimated variance of the error and the estimated covariance matrix

Covariance matrix:

	ar.L1	ar.L2	ar.L3	ar.L4	ar.L5	ar.L6	ar.L7	ar.L8	ar.L9	ar.L10	ar.L11	ar.L12	ar.L13	ar.L14	ar.L15	ar.L16
ar.L1	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	-0.00000	0.00000	-0.00000	-0.00000	0.00000
ar.L2	0.00000	0.00001	0.00000	-0.00000	-0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
ar.L3	0.00000	0.00000	0.00001	0.00000	0.00000	-0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	-0.00000	0.00000
ar.L4	0.00000	-0.00000	0.00000	0.00001	0.00000	-0.00000	-0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	-0.00000
ar.L5	0.00000	-0.00000	0.00000	0.00000	0.00001	-0.00000	-0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
ar.L6	0.00000	0.00000	-0.00000	-0.00000	-0.00000	0.00002	-0.00000	-0.00000	-0.00000	0.00000	-0.00000	-0.00000	0.00000	0.00000	0.00000	0.00000
ar.L7	0.00000	0.00000	0.00000	-0.00000	-0.00000	-0.00000	0.00002	0.00000	-0.00000	-0.00000	0.00000	-0.00000	0.00000	0.00000	0.00000	0.00000
ar.L8	0.00000	0.00000	0.00000	0.00000	0.00000	-0.00000	0.00000	0.00001	-0.00000	-0.00000	-0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
ar.L9	0.00000	0.00000	0.00000	0.00000	0.00000	-0.00000	-0.00000	-0.00000	0.00002	-0.00001	-0.00000	-0.00000	0.00000	0.00000	0.00000	0.00000
ar.L10	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	-0.00000	-0.00000	-0.00001	0.00003	-0.00001	-0.00000	0.00000	0.00000	-0.00000	0.00000
ar.L11	0.00000	0.00000	0.00000	0.00000	0.00000	-0.00000	0.00000	-0.00000	-0.00000	-0.00001	0.00003	-0.00001	-0.00000	0.00000	0.00000	0.00000
ar.L12	-0.00000	0.00000	0.00000	0.00000	0.00000	-0.00000	-0.00000	0.00000	-0.00000	-0.00000	-0.00001	0.00004	-0.00001	-0.00001	-0.00000	0.00000
ar.L13	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	-0.00000	-0.00001	0.00003	-0.00000	-0.00000	-0.00000
ar.L14	-0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	-0.00001	-0.00000	0.00003	-0.00001	-0.00000
ar.L15	-0.00000	0.00000	-0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	-0.00000	0.00000	-0.00000	-0.00000	-0.00001	0.00002	-0.00000
ar.L16	0.00000	0.00000	0.00000	-0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	-0.00000	-0.00000	-0.00000	0.00002
ar.L17	0.00000	0.00000	0.00000	0.00000	0.00000	-0.00000	0.00000	0.00000	0.00000	-0.00000	0.00000	-0.00000	0.00000	-0.00000	-0.00000	-0.00000
ar.L18	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00001	0.00000	0.00000	-0.00000	0.00000	0.00000	-0.00000
ar.L19	0.00000	0.00000	0.00000	0.00000	0.00000	-0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	-0.00000
ar.L20	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
ar.L21	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	-0.00000	0.00000	-0.00000	0.00000	0.00000	-0.00000	0.00000	-0.00000
ar.L22	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
ar.L23	-0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	-0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
ar.L24	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	-0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
sigma2	0.00054	0.00013	0.00004	0.00035	0.00026	0.00045	-0.00016	-0.00015	0.00046	0.00099	-0.00027	-0.00006	0.00017	-0.00040	0.00000	0.00050

ar.L17	ar.L18	ar.L19	ar.L20	ar.L21	ar.L22	ar.L23	ar.L24	sigma2
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	-0.00000	0.00000	0.00054
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00013
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00004
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00035
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00026
-0.00000	0.00000	-0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00045
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	-0.00016
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	-0.00015
0.00000	0.00000	0.00000	0.00000	-0.00000	0.00000	0.00000	0.00000	0.00046
-0.00000	0.00001	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00099
0.00000	0.00000	0.00000	0.00000	-0.00000	0.00000	-0.00000	0.00000	-0.00027
-0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	-0.00000	-0.00006
0.00000	-0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00017
-0.00000	0.00000	0.00000	0.00000	-0.00000	0.00000	0.00000	0.00000	-0.00040
-0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
-0.00000	-0.00000	-0.00000	0.00000	-0.00000	0.00000	0.00000	0.00000	0.00050
0.00003	-0.00001	-0.00000	-0.00000	0.00000	0.00000	0.00000	0.00000	0.00011

-0.00001	0.00002	0.00000	-0.00000	-0.00000	0.00001	-0.00000	0.00000	0.00060
-0.00000	0.00000	0.00001	0.00000	-0.00000	0.00000	0.00000	0.00000	0.00045
-0.00000	-0.00000	0.00000	0.00001	-0.00000	-0.00000	0.00000	0.00000	0.00001
0.00000	-0.00000	-0.00000	-0.00000	0.00002	-0.00001	-0.00000	-0.00000	-0.00002
0.00000	0.00001	0.00000	-0.00000	-0.00001	0.00002	-0.00000	0.00000	-0.00008
0.00000	-0.00000	0.00000	0.00000	-0.00000	-0.00000	0.00000	0.00000	-0.00002
0.00000	0.00000	0.00000	0.00000	-0.00000	0.00000	0.00000	0.00000	0.00019
0.00011	0.00060	0.00045	0.00001	-0.00002	-0.00008	-0.00002	0.00019	0.41932

Figure 36

Figure 36: covariance matrix of model

Estimated variance:

The variance of error is 39989.

- c. It's a bias model.
- d. The variance of residual is 481.53 and the variance of forecast error is 361.58, which means the model performed better at the test set.

Variance of residual is 481.53

Variance of forecast error of ARIMA is 361.58

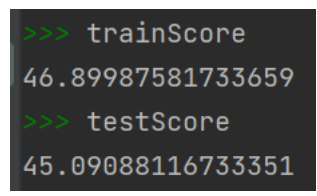
Figure 37

Figure 37: result of variance of residual and forecast error

Deep Learning Model

For the deep learning section, I fit the data with LSTM model, which is a common deep learning model for time series data. I set the model with epochs = 50, batch size = 1, and verbose = 2.

After fitting the data and model training, the MSE of training data is 46.899 and the MSE of testing data is 45.09.



```
>>> trainScore
46.89987581733659
>>> testScore
45.09088116733351
```

Figure 38

Figure 38: results of train score and test score

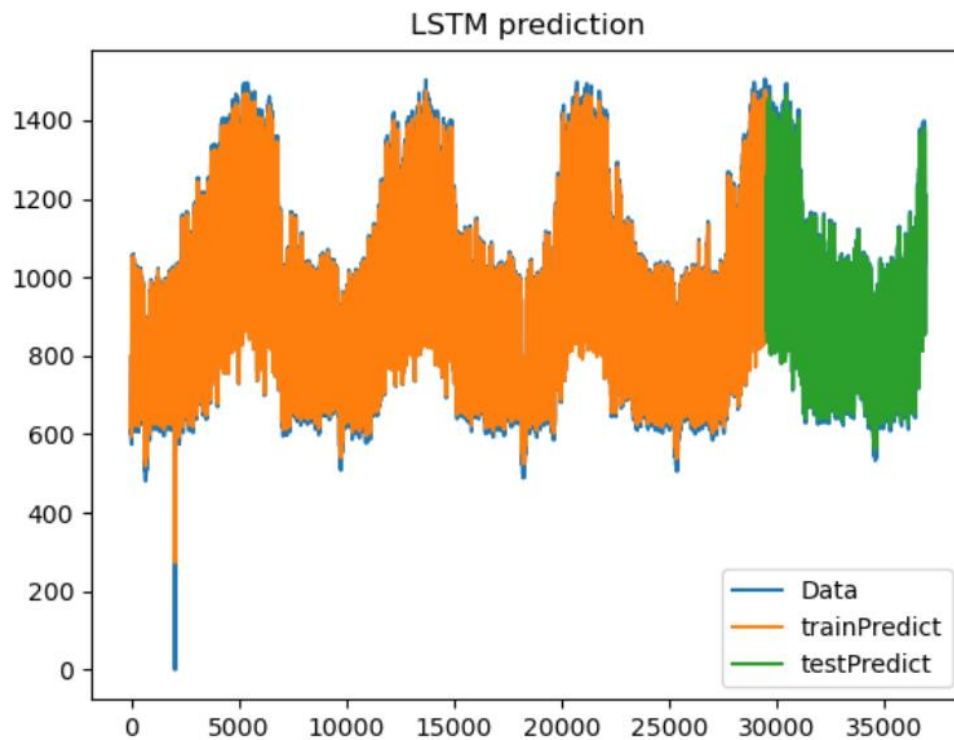


Figure 39

Figure 39: result of LSTM model

Final Model Selection

By comparing the MSE of the testing dataset, we can select the model with the lowest error. The LSTM model has the best performance, and next will be the SARIMA model. The MSE of LSTM is about 45 and the MSE of ARIMA is 361.

Forecast Function

For the forecast function, I'm using the ARIMA(24,1,0) model, which is the best model with a presentable forecast function.

Function:

$$y(t) - 0.16*y(t-1) - 0.5*y(t-24) = e(t)$$

h-step Ahead Predictions

Finally, the SARIMA(1,0,0)24 or ARIMA(24,1,0) model will be the best model for our final term project. Comparing the prediction with the original data, we can see that due to the strongly seasonality and trending makes the prediction highly fit the data.

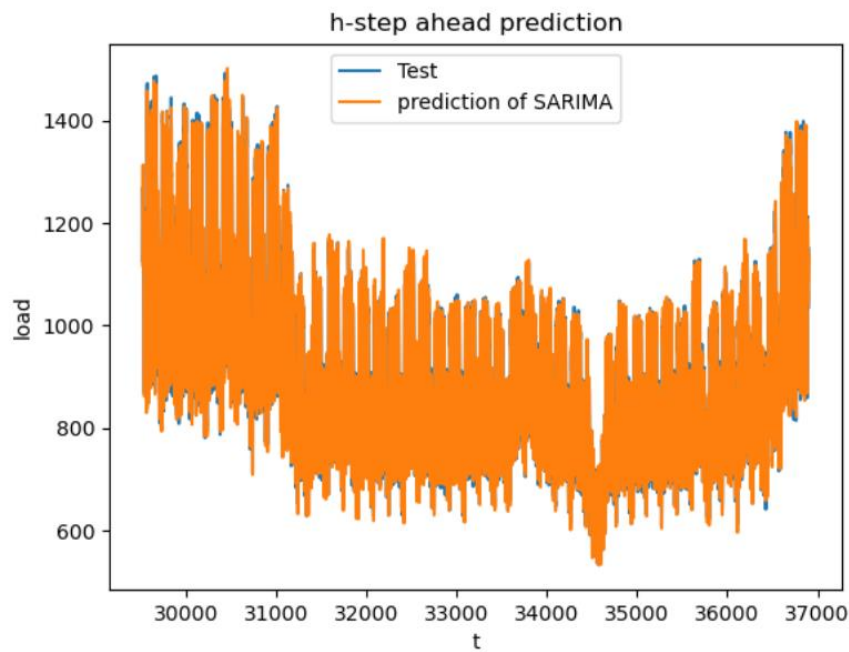


Figure 40

Figure 40: prediction of best model selected versus test data

Summary and Conclusion

For the final model selected in this project is a SARIMA model, which only fit with time series data but can't predict using other features. Although the original goal of this project is to find out if the weather data is connected to the electricity load, the performance of pure time series prediction using SARIMA model and LSMT is better.

On the other hand, the LSMT deep learning model performed better but the training time takes much longer than traditional time series model like SARIMA. Although the error is smaller than traditional model but if we discuss about efficiency, I will say that traditional model is better for dataset that is too large. However, if you get enough time and ram for training model, deep learning model like LSMT will be a better choice.

Appendix

```

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import kpss
import seaborn as sns
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from scipy.stats import chi2

#read data.....

df_raw_train_W = pd.read_csv('data/weatherData.csv')
df_raw_train_P = pd.read_csv('data/loadarea.csv')

df_train_P = df_raw_train_P.copy(deep=True)
df_train_W = df_raw_train_W.copy(deep=True)

df_train_W = df_train_W.rename(columns={'ObsTime': 'datetime'})
N = df_train_W[df_train_W['station'] == '466920_臺北']
C = df_train_W[df_train_W['station'] == '467490_臺中']
S = df_train_W[df_train_W['station'] == '467410_臺南']
E = df_train_W[df_train_W['station'] == '466990_花蓮']

n = df_train_P[df_train_P['area'] == 'north']
c = df_train_P[df_train_P['area'] == 'central']
s = df_train_P[df_train_P['area'] == 'south']
e = df_train_P[df_train_P['area'] == 'east']

North = pd.merge(N,n, on = 'datetime', how = 'inner')
Central = pd.merge(C,c, on = 'datetime', how = 'inner')
South = pd.merge(S,s, on = 'datetime', how = 'inner')
East = pd.merge(E,e, on = 'datetime', how = 'inner')

df = pd.concat([North, Central, South, East], sort=False)
df_save = df.copy()
#Check na values

print(df.isnull().sum())
# Drop Visb, Cloud Amount, and Sunshine, the percentage of missing value is
too large
df = df.drop(columns=['Visb', 'SunShine', 'Cloud Amount'])

# check if I should replace missing value with mean or most frequent value
check_v = df['Precp'].dropna()
sns.histplot(data = check_v)
plt.title('Histogram of Precp')
plt.show()

```

```
# Most value of Precp is 0 so I decided to replace missing with zero
df['Precp'] = df['Precp'].fillna(0)
# Same way to check other features, I decided to fill na with most frequent
value
df = df.apply(lambda x:x.fillna(x.value_counts().index[0]))
# Check NA
print(df.isnull().sum())

# Because the dataset included four areas data using the same time index so I
decided to separate them
# Reindex dataframe with time
df_n = pd.Series(np.array(North['load']).reshape(len(North)),
index=pd.date_range("1917-01-01 01:00:00", periods=len(North), freq="H"),
name="load of North")
df_c = pd.Series(np.array(Central['load']).reshape(len(Central)),
index=pd.date_range("1917-01-01 01:00:00", periods=len(Central), freq="H"),
name="load of Central")
df_s = pd.Series(np.array(South['load']).reshape(len(South)),
index=pd.date_range("1917-01-01 01:00:00", periods=len(South), freq="H"),
name="load of South")
df_e = pd.Series(np.array(East['load']).reshape(len(East)),
index=pd.date_range("1917-01-01 01:00:00", periods=len(East), freq="H"),
name="load of East")
print(df_n.head())

# Plot dependent variable versus time
fig, ax = plt.subplots(1,1)
plt.plot(df_n, label = 'North')
plt.plot(df_c, label = 'Central')
plt.plot(df_s, label = 'South')
plt.plot(df_e, label = 'East')
plt.legend()
plt.xticks(rotation=30)
plt.xlabel('Time')
plt.ylabel('Load')
plt.title('Load vs Time')
fig.tight_layout(pad=3)
plt.show()

# ACF/PACF
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf , plot_pacf
def ACF_PACF_Plot(y,lags):
    name = y.name
    acf = sm.tsa.stattools.acf(y, nlags=lags)
    pacf = sm.tsa.stattools.pacf(y, nlags=lags)
    fig = plt.figure()
    fig.suptitle(name)
    plt.subplot(211)
    plt.title('ACF/PACF of the raw data')
    plot_acf(y, ax=plt.gca(), lags=lags)
    plt.subplot(212)
    plot_pacf(y, ax=plt.gca(), lags=lags)
```

```
fig.tight_layout(pad=1)
plt.show()

lags = 50
ACF_PACF_Plot(df_n, lags)
ACF_PACF_Plot(df_c, lags)
ACF_PACF_Plot(df_s, lags)
ACF_PACF_Plot(df_e, lags)

# From PACF, there might be a cut off at lag one or two : AR order

# Correlation Matrix with seaborn heatmap with the Pearson's correlation
# coefficient.
# North
fig = plt.figure()
sns.heatmap(North.corr())
fig.tight_layout(pad=3)
fig.suptitle('Pearson's correlation coefficient - Load of North')
plt.show()
#Central
fig = plt.figure()
sns.heatmap(Central.corr())
fig.tight_layout(pad=3)
fig.suptitle('Pearson's correlation coefficient - Load of Central')
plt.show()
#South
fig = plt.figure()
sns.heatmap(South.corr())
fig.tight_layout(pad=3)
fig.suptitle('Pearson's correlation coefficient - Load of South')
plt.show()
#East
fig = plt.figure()
sns.heatmap(North.corr())
fig.tight_layout(pad=3)
fig.suptitle('Pearson's correlation coefficient - Load of East')
plt.show()

# We can see there is no big difference between each area
# Then I decided to use the North dataset as the main dataset
df_main = df.copy()
df = North.drop(columns=['Visb', 'SunShine', 'Cloud Amount'])
df['Precp'] = df['Precp'].fillna(0)
df = df.apply(lambda x: x.fillna(x.value_counts().index[0]))

# Split the data
# from sklearn.model_selection import train_test_split (split function from
# ML1)
train = df[:int(len(df)*80/100)]
test = df[int(len(df)*80/100):]
```

```

# Stationary check
# ADF/KPSS test for original dependent variable
# KPSS-test
from statsmodels.tsa.stattools import kpss
def kpss_test(timeseries):
    print('KPSS test of ' + timeseries.name)
    print('Results of KPSS Test:')
    kpsstest = kpss(timeseries, regression='c', nlags="auto")
    kpss_output = pd.Series(kpsstest[0:3], index=['Test Statistic', 'p-
value', 'Lags Used'])
    for key, value in kpsstest[3].items():
        kpss_output['Critical Value (%)' % key] = value
    print(kpss_output)    #----- pvalue high station

# ADF-test
from statsmodels.tsa.stattools import adfuller
def ADF_Cal(x):
    result = adfuller(x)

    print("ADF Statistic: %f" %result[0])
    print('p-value: %f' % result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value)) # ----- p-value low station

# difference
def difference(dataset, interval=1):
    diff = []
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return diff

# Rolling mean/variance
def Cal_rolling_mean_var(y):
    Rmean = np.zeros(len(y))
    for i in range(0, len(y)):
        Rmean[i] = y[0:i+1].mean()

    Rvar = np.zeros(len(y))
    for i in range(0, len(y)):
        Rvar[i] = np.var((y[0:i+1]))
    Rvar[0] = 0
    return Rmean, Rvar

# Main dataset
ADF_Cal(df['load'])
kpss_test(df['load'])
N_mean, N_var = Cal_rolling_mean_var(df['load'])
fig = plt.figure()
plt.subplot(211)
plt.title('Rolling mean and variance - load of North')
plt.plot(N_mean)
plt.ylabel('Rolling mean')

```

```
plt.subplot(212)
plt.plot(N_var)
plt.ylabel('Rolling var')
plt.xlabel('Index')
fig.tight_layout(pad=3)
plt.show()

# North
ADF_Cal(North['load'])
kpss_test(North['load'])
N_mean, N_var = Cal_rolling_mean_var(North['load'])
fig = plt.figure()
plt.subplot(211)
plt.title('Rolling mean and variance - load of North')
plt.plot(N_mean)
plt.ylabel('Rolling mean')
plt.subplot(212)
plt.plot(N_var)
plt.ylabel('Rolling var')
plt.xlabel('Index')
fig.tight_layout(pad=3)
plt.show()

# Central
ADF_Cal(Central['load'])
kpss_test(Central['load'])
C_mean, C_var = Cal_rolling_mean_var(Central['load'])
fig = plt.figure()
plt.subplot(211)
plt.title('Rolling mean and variance - load of Central')
plt.plot(C_mean)
plt.ylabel('Rolling mean')
plt.subplot(212)
plt.plot(C_var)
plt.ylabel('Rolling var')
plt.xlabel('Index')
fig.tight_layout(pad=3)
plt.show()

# South
ADF_Cal(South['load'])
kpss_test(South['load'])
S_mean, S_var = Cal_rolling_mean_var(South['load'])
fig = plt.figure()
plt.subplot(211)
plt.title('Rolling mean and variance - load of South')
plt.plot(S_mean)
plt.ylabel('Rolling mean')
plt.subplot(212)
plt.plot(S_var)
plt.ylabel('Rolling var')
plt.xlabel('Index')
fig.tight_layout(pad=3)
plt.show()
```

```
# Central
# South
ADF_Cal(Central['load'])
kpss_test(Central['load'])
C_mean, C_var = Cal_rolling_mean_var(Central['load'])
fig = plt.figure()
plt.subplot(211)
plt.title('Rolling mean and variance - load of Central')
plt.plot(C_mean)
plt.ylabel('Rolling mean')
plt.subplot(212)
plt.plot(C_var)
plt.ylabel('Rolling var')
plt.xlabel('Index')
fig.tight_layout(pad=3)
plt.show()

# East
ADF_Cal(East['load'])
kpss_test(East['load'])
E_mean, E_var = Cal_rolling_mean_var(East['load'])
fig = plt.figure()
plt.subplot(211)
plt.title('Rolling mean and variance - load of East')
plt.plot(E_mean)
plt.ylabel('Rolling mean')
plt.subplot(212)
plt.plot(E_var)
plt.ylabel('Rolling var')
plt.xlabel('Index')
fig.tight_layout(pad=3)
plt.show()

# Time Seires Decomposition
from statsmodels.tsa.seasonal import STL
df_n = pd.Series(np.array(df['load']).reshape(len(df)),
index=pd.date_range("2017-01-01 01:00:00", periods=len(df), freq="H"),
name="load of North")

STL_n = STL(df_n)
res_n = STL_n.fit()

fig = res_n.plot()
plt.show()

S = res_n.seasonal
S_adj = df_n.values - S.values

T = res_n.trend
R = res_n.resid

st = np.maximum(0, 1 - np.var(np.array(R)) / np.var(np.array(T) + np.array(R)))
```



```
print(f'The strength of trend for this data set is {st:.2f}')
```

```
ss = np.maximum(0,1-np.var(np.array(R))/np.var(np.array(S)+np.array(R)))
print(f'The strength of seasonality for this data set is {ss:.2f}')
```

```
print('The dependent variable of area of North is highly trend and
seasonal.')
```

```
# Select data by year
df_de = df_n[len(df_n) - 8760:]
STL_de = STL(df_de, period=24)
res_de = STL_de.fit()

fig = res_de.plot()
plt.show()

#Central
STL_c = STL(df_c)
res_c = STL_c.fit()

S = res_c.seasonal
S_adj = df_c.values - S.values

T = res_c.trend
R = res_c.resid

st = np.maximum(0,1-np.var(np.array(R))/np.var(np.array(T)+np.array(R)))
print(f'The strength of trend for this data set is {st:.2f}')
```

```
ss = np.maximum(0,1-np.var(np.array(R))/np.var(np.array(S)+np.array(R)))
print(f'The strength of seasonality for this data set is {ss:.2f}')
```

```
print('The dependent variable of area of Central is highly trend and
seasonal.')
```

```
#South
STL_s = STL(df_s)
res_s = STL_s.fit()

S = res_s.seasonal
S_adj = df_s.values - S.values

T = res_s.trend
R = res_s.resid

st = np.maximum(0,1-np.var(np.array(R))/np.var(np.array(T)+np.array(R)))
print(f'The strength of trend for this data set is {st:.2f}')
```

```
ss = np.maximum(0,1-np.var(np.array(R))/np.var(np.array(S)+np.array(R)))
print(f'The strength of seasonality for this data set is {ss:.2f}')
```

```

print('The dependent variable of area of South is highly trend and
seasonal.')

#East
STL_e = STL(df_e)
res_e = STL_e.fit()

S = res_e.seasonal
S_adj = df_e.values - S.values

T = res_e.trend
R = res_e.resid

st = np.maximum(0,1-np.var(np.array(R))/np.var(np.array(T)+np.array(R)))
print(f'The strength of trend for this data set is {st:.2f}')

ss = np.maximum(0,1-np.var(np.array(R))/np.var(np.array(S)+np.array(R)))
print(f'The strength of seasonality for this data set is {ss:.2f}')

print('The dependent variable of area of East is highly trend and seasonal.')

# We can see there is no big difference between each area
# Then I decided to use the North dataset as the main dataset
df_main = df.copy()

df = North.drop(columns=['Visb', 'SunShine', 'Cloud Amount'])
df = df.apply(lambda x:x.fillna(x.value_counts().index[0]))
df_train = df[:int(len(df)*80/100)]
df_test = df[int(len(df)*80/100):]

# Holt-Winter Method
import statsmodels.tsa.holtwinters as ets

y_HW = df_test.copy()
fitted_model = ets.ExponentialSmoothing(df_train['load'], trend = 'add',
seasonal= 'add', seasonal_periods=24, damped_trend=True).fit()
y_HW['HW_fit'] = fitted_model.forecast(len(df_test))
#
#
# model = ets.ExponentialSmoothing(df_train['load'], seasonal="add",
seasonal_periods=8760)
# model2 = ets.ExponentialSmoothing(df_train['load'], trend="add",
seasonal="add", seasonal_periods=12, damped_trend=True)
# fit = model.fit()
# y_HW['HW_fit'] = fit.forecast(len(df_test))
#
plt.figure()
plt.plot(df_train['load'], label='Train')
plt.plot(df_test['load'], label='Test')
plt.plot(y_HW['HW_fit'], label = 'Holt-Winter')
plt.legend(loc='best')
plt.title('Holt-Winter Seasonal forecast')
plt.xlabel('Date')

```

```
plt.ylabel('load')
plt.show()
#Last 100 data
plt.figure()
# plt.plot(df_train['load'], label='Train')
plt.plot(df_test['load'].iloc[-100:], label='Test')
plt.plot(y_HW['HW_fit'].iloc[-100:], label = 'Holt-Winter')
plt.legend(loc='best')
plt.title('Holt-Winter Seasonal forecast')
plt.xlabel('Date')
plt.ylabel('load')
plt.show()
# fig = res_n.plot()
# fig.suptitle("Trend&Seasonality&reminder")
# plt.xlabel('Time')
# plt.tight_layout()
# plt.show()

# Feature selection
from numpy import linalg as LA
df1 = df.drop(columns= ['datetime', 'station', 'area'])
X = np.matmul(df1.T, df1)
_,d,_ = np.linalg.svd(X)
print(f'singular values of Data is {d}')
print(f'The condition number of Data is {LA.cond(df1):.2f}')

# The lowest number of SVD is 2.66, although it's not close to 0 but the
condition number of Data 64299 is too high.
# OLS model to decide which feature should be eliminated
y_train = df1['load']
df1 = df1.drop(columns = ['load'])
model = sm.OLS(y_train, df1).fit()
# prediction = model.predict(X_test)
print(model.summary())
print(f'AIC of model1 {model.aic:.2f}')
print(f'BIC of model1 {model.bic:.2f}')
print(f'Adjusted R-square of model1 {model.rsquared_adj:.2f}')

R1 = df1.copy()
R1 = R1.drop(columns = ['WD'])

model1 = sm.OLS(y_train, R1).fit()
model1.summary()
print(f'AIC of model1 {model1.aic:.2f}')
print(f'BIC of model1 {model1.bic:.2f}')
print(f'Adjusted R-square of model1 {model1.rsquared_adj:.2f}')

R2 = R1.copy()
R2 = R2.drop(columns = ['Td dew point'])

model2 = sm.OLS(y_train, R2).fit()
model2.summary()
```

```

print(f'AIC of model2 {model2.aic:.2f}')
print(f'BIC of model2 {model2.bic:.2f}')
print(f'Adjusted R-square of model2 {model2.rsquared_adj:.2f}')

df_drop = df.drop(columns= ['datetime', 'station', 'area', 'Td dew point',
'WD'])
X = np.matmul(df_drop.T, df_drop)
_,d,_ = np.linalg.svd(X)
print(f'singular values of Data is {d}')
print(f'The condition number of Data is {LA.cond(df_drop):.2f}')

# The condition number dropped so the direction of dropping features are
correct

# try Random foreset feature selection
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
df1 = df.drop(columns= ['datetime', 'station', 'area'])
features = pd.get_dummies(df1)
labels = np.array(features['load'])
features= features.drop('load', axis = 1)
feature_list = list(features.columns)
features = np.array(features)
train_features, test_features, train_labels, test_labels =
train_test_split(features, labels, test_size = 0.2, random_state = 42)
# baseline_preds = test_features[:, feature_list.index('average')]
# baseline_errors = abs(baseline_preds - test_labels)
rf = RandomForestRegressor(n_estimators = 500, random_state = 42)
rf.fit(train_features, train_labels)

importances = list(rf.feature_importances_)
feature_importances = [(feature, round(importance, 2)) for feature,
importance in zip(feature_list, importances)]
feature_importances = sorted(feature_importances, key = lambda x: x[1],
reverse = True)
[print('Variable: {0:20} Importance: {}'.format(*pair)) for pair in
feature_importances]
# We can see that the most important feature is Temperature

df_test = test.copy()
df_train = train.copy()
# Base model
# AFM
y_avg = df_test.copy()
y_avg['avg_forecast'] = df_train['load'].mean()
sum = 0
for i in np.arange(len(test)):
    sum += (test['load'].iloc[i]- y_avg['avg_forecast'].iloc[i]) **2
MSEtest = sum/len(test)
print(f'MSE of average forecast is {MSEtest:.2f}')

```

```

# Naive
y_nai = df_test.copy()
y_nai['NAi_forecast'] = df_train['load'].iloc[-1]

sum = 0
for i in np.arange(len(test)):
    sum += (test['load'].iloc[i] - y_nai['NAi_forecast'].iloc[i]) ** 2
MSEtest1 = sum / len(test)
print(f'MSE of Naive forecast is {MSEtest1:.2f}')

# Drift
y_D = df_test.copy()
frac = (df_train['load'].iloc[-1]) - (df_train['load'].iloc[0])
deno = 1 / (len(df_train) - 1)
y_D['D_forecast'] = df_train['load'].iloc[-1]
a = []
for i in np.arange(len(y_D)):
    a.append((df_train['load'].iloc[-1]) + (frac * deno * (i + 1)))
y_D['D_forecast'] = a

sum = 0
for i in np.arange(len(test)):
    sum += (test['load'].iloc[i] - y_D['D_forecast'].iloc[i]) ** 2
MSEtest2 = sum / len(test)
print(f'MSE of drift forecast is {MSEtest2:.2f}')

# SES a = 0.5
y_SES = df_train.copy()
y_SES['a_050'] = df_train['load'].iloc[0]
a_050 = [np.nan, df_train['load'].iloc[0]]
for i in np.arange(len(df_train) - 2):
    a_050.append((0.5 * (df_train['load'].iloc[i + 1])) + (1 - 0.5) *
                 (df_train['load'].iloc[i + 1]))
y_SES['a_050'] = a_050
y_hat_sesf05 = df_test.copy()
y_hat_sesf05['ses_050_forecast'] = (0.5 * (df_train['load'].iloc[-1])) + (1 -
0.5) * (df_train['load'].iloc[-1])

sum = 0
for i in np.arange(len(test)):
    sum += (test['load'].iloc[i] - y_hat_sesf05['ses_050_forecast'].iloc[i])
**2
MSEtest3 = sum / len(test)
print(f'MSE of SES forecast is {MSEtest3:.2f}')

# Multiple linear model
R2 = R2.iloc[:29521, :]
predict = model2.predict(R2)
plt.figure()
plt.plot(df_train['load'], label='Train')

```

```

plt.plot(predict, label='One-step')
plt.legend(loc='best')
plt.title('One-step prediction of OLS')
plt.xlabel('Date')
plt.ylabel('load')
plt.show()

df_test = test.drop(columns = ['datetime', 'station', 'area', 'Td dew
point', 'WD', 'load'])
predictions = model2.predict(df_test)
plt.figure()
plt.plot(test['load'], label='Test')
plt.plot(predictions, label='h-step')
plt.legend(loc='best')
plt.title('h-step prediction of OLS')
plt.xlabel('Date')
plt.ylabel('load')
plt.show()

sum = 0
for i in np.arange(len(test)):
    sum += (test['load'].iloc[i] - predictions.iloc[i]) ** 2
MSEtest4 = sum / len(test)
print(f'MSE of OLS model forecast is {MSEtest4:.2f}')

print(f'AIC of model2 {model2.aic:.2f}')
print(f'BIC of model2 {model2.bic:.2f}')
print(f'Adjusted R-square of model2 {model2.rsquared_adj:.2f}')
print(f'Adjusted R-square of model2 {model2.rsquared:.2f}')

# Residual
res = np.zeros(len(df_train))
for i in np.arange(len(df_train)):
    res[i] = (df_train['load'][i] - predict[i])

res_mean = np.mean(res)
res_var = np.var(res)

# ACF
acf = sm.tsa.stattools.acf(res, nlags=len(res))

# Q-value (with lag 20)
Q = len(res) * np.sum(np.square(acf[20:]))

A = np.identity(len(model2.params))
A = A[1:,:]
print(model2.f_test(A))
print(model2.t_test(A))

# ARIMA, SARIMA, and ARMA
# GPAC

```

```

y_var = np.var(df_train['load'])

def get_GPAC(ry2, col, row, y_var = y_var):
    Va = ry2 * y_var
    Va = np.asarray(Va)
    mid = len(Va)//2
    place = np.array([0.0000] * (col * row))
    place = place.reshape(col, int(len(place) / col))
    place2 = place.copy()
    t = []

    for j in np.arange(row):
        for k in np.arange(1, 1 + col):
            if k == 1:
                t.append((Va[mid + j + 1]) / Va[mid + j])
            else:
                P = []
                D = []
                for i in np.arange(k):
                    P = np.append(P, Va[mid - j - i: mid - j - i + k])
                    P = P.reshape(k, int(len(P) / k))
                    D = P
                P = np.asarray(P)
                D = np.asarray(D)
                place[k - 1][j] = np.linalg.det(P)
                for e in np.arange(k):
                    D[e][-1] = Va[mid + j + e + 1]
                place2[k - 1][j] = np.linalg.det(D)
    T = place2 / place
    T[0] = t
    T = T.T
    T = T.round(3)
    x_axis_labels = np.arange(1, col + 1)
    ax = sns.heatmap(T, linewidth=0.3, xticklabels=x_axis_labels, annot=True)
    ax.set_title('Generalized Partial Autocorrelation (GPAC) Table')
    plt.show()
    print(T)

#.....
.....

ry1 = acf[:14][::-1]
ry2 = np.concatenate((np.reshape(ry1,14), acf[:14][1:]))
get_GPAC(ry2, 7, 7)

ACF_PACF_Plot(df['load'],lags)
# Try difference 1
def difference(dataset,interval=1):
    diff = []
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return diff

```

```

d24 = difference(df['load'], interval=24)
def ACF_PACF_Plot(y, lags):
    # name = y.name
    acf = sm.tsa.stattools.acf(y, nlags=lags)
    pacf = sm.tsa.stattools.pacf(y, nlags=lags)
    fig = plt.figure()
    # fig.suptitle(name)
    plt.subplot(211)
    plt.title('ACF/PACF of the raw data')
    plot_acf(y, ax=plt.gca(), lags=lags)
    plt.subplot(212)
    plot_pacf(y, ax=plt.gca(), lags=lags)
    fig.tight_layout(pad=1)
    plt.show()

ACF_PACF_Plot(d24, lags)
# Try difference with 24
d24_1 = difference(d24, interval=1)
ACF_PACF_Plot(d24_1, lags)

# d1_24 seems more stationary, then try GPAC
acf1_24 = sm.tsa.stattools.acf(d24_1, nlags=len(d24_1))
ry11_24 = acf1_24[:14][::-1]
ry21_24 = np.concatenate((np.reshape(ry11_24, 14), acf1_24[:14][1:]))
get_GPAC(ry21_24, 7, 7)
# From the GPAC table, I'm guessing this dataset is with no AR and MA any
more
# So the original dataset might be a SARIMA(0,1,0)24

from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(df['load'], order=(24,1,0))
model_fit = model.fit()
print(model_fit.summary())
residuals = pd.DataFrame(model_fit.resid)
residuals.plot()
residuals.plot(kind='kde')
plt.title('Density of residual')
plt.show()

model_hat = model_fit.predict(start = 0, end = len(df)-1)
VAR = np.var(model_hat)
errortrain1 = [np.nan]
for i in np.arange(len(train)-1):
    errortrain1.append(df_train['load'].iloc[i+1]- model_hat.iloc[i+1])
print(f'Variance of residual is {np.var(errortrain1[1:]):.2f}')

test_error = test.copy()
sum = 0
for i in np.arange(len(test)):
    sum += (test_error['load'].iloc[i]- model_hat.iloc[i + len(df_train)])

```



```

**2
    MSEtest6 = sum/len(test)
error = []
for i in np.arange(len(test)):
    error.append(test['load'].iloc[i]- model_hat.iloc[i + len(df_train)])
print(f'Variance of forecast error of ARIMA is {np.var(error):.2f}')

plt.figure()
plt.plot(df_train['load'], label='Train')
plt.plot(test['load'], label='Test')
plt.plot(model_hat, label='prediction of ARIMA')
plt.legend(loc='best')
plt.title('One-step ahead prediction')
plt.xlabel('t')
plt.ylabel('yt')
plt.show()

# First 100 prediction
plt.figure()
plt.plot(df_train['load'][:100], label='Train')
# plt.plot(y_test, label='Test')
plt.plot(model_hat[:100], label='one-step ahead prediction')
plt.legend(loc='best')
plt.title('One-step ahead prediction')
plt.xlabel('t')
plt.ylabel('yt')
plt.show()

# Most prediction fit the dataset
from scipy.stats import chi2
N = len(df)
lb_stat, pvalue = sm.stats.acorr_ljungbox(errortrain1[1:], lags=[20],
return_df=True).iloc[0]
na = 24
nb = 0
alfa = 0.01
DOF = N - na - nb
chi_critical = chi2.ppf(1 - alfa, DOF)
if lb_stat < chi_critical:
    print('The residual is white.')
if lb_stat > chi_critical:
    print('The residual is not white.')

# lm method
from scipy import signal
na = 1
nb = 0
y = df_train['load']
def cal_e(seta, y = y, na = na, nb = nb):
    max_order = max(na, nb)
    num = [0.00] * (max_order)
    den = [0.00] * (max_order)
    num = np.asarray(num)

```

```

den = np.asarray(den)
for i in np.arange(len(seta)-nb):
    if len(seta)-nb ==0:
        den = den
    else:
        den[i] = seta[i]
for i in np.arange(len(seta)-na):
    if len(seta)-na == 0:
        num = num
    else:
        num[i] = seta[na+i]
ar = np.r_[1, den]
ma = np.r_[1, num]
sys = (ar,ma,1)
_,e = signal.dlsim(sys,y)
return e

def x_dev(seta, tel =10 ** (-5), y =y, na =na, nb = nb, cal_e = cal_e):
    X = []
    e = cal_e(seta,y = y,na = na, nb = nb)
    for i in np.arange(na+nb):
        seta_add = seta.copy()
        seta_add[i] = seta[i] + tel
        d = cal_e(seta_add, y, na, nb) - e
        X = np.append(X, (d/tel).T)
    # X = X.reshape(len(y), na+nb)
    X = X.reshape(na + nb, len(y))
    # A = np.dot(X.T, X)
    A = np.dot(X, X.T)
    # g = np.dot(X.T, e)
    g = np.dot(X, e)
    return A,g

def LM(y, na,nb, max = 100, mu = 10 ** (-3), x_dev= x_dev, cal_e = cal_e):
    k = 0
    seta = np.zeros(na+nb)
    mu = mu
    I = np.identity(na + nb)
    SSE_record = []
    while k<max:
        k+= 1
        e = cal_e(seta, y=y, na=na, nb = nb)
        SSE = np.dot(e.T,e)
        SSE = np.asscalar(SSE)
        A,g = x_dev(seta, tel=10 ** (-6), y=y, na=na, nb=nb)
        del_seg = np.dot(np.linalg.inv((A + mu*I)),g)
        del_seg = del_seg.T
        new_seta = seta - del_seg
        new_seta = new_seta.reshape(na+nb,)
        new_e = cal_e(new_seta, y=y, na=na, nb =nb)
        SSE_new = np.dot(new_e.T,new_e)
        SSE_new = np.asscalar(SSE_new)

```

```

SSE_record.append(SSE)
if SSE_new < SSE:
    if np.linalg.norm(del_seg) < 10 ** (-3):
        seta_end = new_seta
        z = cal_e(seta_end, y=y, na=na)
        ez = np.dot(z.T, z)
        ez = np.asscalar(ez)
        cov = ez / (len(y) - (na + nb))
        Cov = np.dot(cov, np.linalg.inv(A))
        reason = 'delta theta < 10 ** (-3)'
        return seta_end, reason, Cov, new_e, SSE_record
    else:
        seta = new_seta
        mu = mu / 10
elif SSE_new > SSE:
    mu = mu * 10
    if mu > 10 ** 9:
        reason = 'Mu is too large'
        return seta, reason
if k >= max:
    reason = 'k larger than max'
    return seta, reason

# teta, _, Cov, new_e, SSE_record = LM(df_train['load'], na, nb, max = 100, mu =
10 ** (-3), x_dev= x_dev, cal_e = cal_e)

Cov = model_fit.cov_params()

# LSTM

import numpy
import matplotlib.pyplot as plt
import pandas
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

scaler = MinMaxScaler(feature_range=(0, 1))
df_LS = df['load']
dataset = df_LS.values
dataset = dataset.astype('float32')
dataset = scaler.fit_transform(dataset.reshape(-1, 1))

train_size = int(len(dataset) * 0.8)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size, :], dataset[train_size:len(dataset), :]

```

```

print(len(train), len(test))

def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return numpy.array(dataX), numpy.array(dataY)

# reshape into X=t and Y=t+1
look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)

# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

# create and fit the LSTM network
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=50, batch_size=1, verbose=2)

# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))

# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] =
testPredict

```

```
# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset), label = 'Data')
plt.plot(trainPredictPlot, label = 'trainPredict')
plt.plot(testPredictPlot, label = 'testPredict')
plt.title('LSTM prediction')
plt.legend()
plt.show()

error = []
for i in np.arange(len(testPredict)):
    error.append(test['load'].iloc[i]- testPredict[i])
print(f'Variance of prediction error of LSTM is {np.var(error):.2f}')

# Forecast Function
# model_fit.seasonalarparams

# h-step prediction of SARIMA

plt.figure()
plt.plot(test['load'], label='Test')
plt.plot(model_hat[len(df_train):], label='prediction of SARIMA')
plt.legend(loc='best')
plt.title('h-step ahead prediction')
plt.xlabel('t')
plt.ylabel('load')
plt.show()
```