Data Science 6450, Section 15: Time Series Modeling & Analysis

Professor Reza Jafari

Final Project

Benjamin Lee

BML 12/16/2020

Table of Contents

Abstract

The purpose of the final project is to develop a linear model representation of real-world time series data and apply the various time series modeling and analysis methods learned in this course. The dataset used in this project is concerned with hourly particulate matter (PM) 2.5 data from the US Embassy in Beijing. The purposes of applying modeling and analysis techniques to this dataset include understanding what features possess the most profound impact on PM 2.5 concentration and forecasting PM 2.5 data to estimate concentrations of particulate matter in Beijing in the future. Although most of the topics discussed in the course are utilized in this project, some of the more contextually important topics include feature selection, base model forecasts and SARIMA modelling.

Introduction

The outline of this project follows the basic time series modelling and analysis steps outlined in the graphic below in Figure 1.

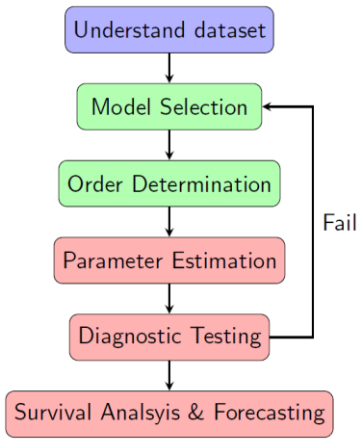


Figure 1. Time series modelling steps

Understanding the dataset involves analyzing the features of the data and applying context to understand why they are relevant to the problem at hand. By understanding which features possess the strongest effects on PM 2.5 concentration and how PM 2.5 data could change in the future, plans could be put in place to reduce the number of harmful particles in the air and make Beijing a safer place to live. Upon initial inspection of the dataset, I suspected wind speed and direction to be the most impactful features; wind can assist in distributing particulate matter across wide spaces. Other weather variables like temperature and dew point are also known to have a profound impact on the formation and distribution of particulate matter.

Model selection was led by construction and development of several different models in this project. The base models (average, naïve, drift and simple exponential smoothing forecasting methods), Holt-Winters method, multiple linear regression, and a SARIMA model were developed and compared in this project. Order determination for the SARIMA model was led by analysis of autocorrelation and partial autocorrelation plots of the stationary form of the raw data. Once orders for the model were determined, the parameters were estimated by running the models through application of the statsmodels library. Upon estimation of parameters, diagnostic testing was performed to analyze confidence intervals, check for zero/pole cancellations, and run chi-squared test for whiteness of the residuals; these actions effectively validate the model and determine whether it will perform well. After diagnostic testing was completed, forecasts were constructed for all models. The results of all models were compared, and the best model was chosen for h-step ahead predictions. A deeper dive into the details of the time series modeling and analysis process can be found below. I want to note that although the instructions request the minimum application of an ARMA model, analysis of the ACF and PACF plots of the stationary data during office hours on December 15, 2020 concluded that this data is AR and MA seasonal and that between ARMA, ARIMA and SARIMA, the best option is SARIMA modelling. Thus, use of the GPAC table and Levenberg Marquardt algorithm for ARMA parameter estimation are not needed for this project.

Description of The Dataset

As mentioned in the abstract and introduction, the dataset used in this project focuses on hourly PM 2.5 data from the US Embassy in Beijing. Meteorological data from the Beijing Capital International Airport are also included. The dataset was sourced from the UCI website; credit can be found in the References section of this document. I want to note that a subsample of the original data was used for this project.

Upon visiting office hours on December 8, 2020, I realized another student in the course happened to be working on the same dataset. I did not feel comfortable working with the same data as another student and felt that working with multiple years of data made it difficult to apply the different models effectively. The sub-sample of the original dataset encompasses hourly data for the year 2010 and consists of 8,760 samples and the same seven features as the original data. The dependent variable of this dataset is the PM 2.5 concentration, which is measured in micrograms per cubic meter. In the context of air quality, PM 2.5 refers to small airborne particles that are 2.5 microns in width; for reference, a human hair is roughly 70 microns wide. Certain PM 2.5 particles can cause diseases and make people severely ill; sulfur dioxide from power plants and nitrogen dioxide from vehicle exhausts are two prominent examples in and around Beijing. The independent variables of this dataset include the date and time, dew point, temperature, pressure, combined wind direction, combined wind speed, cumulated hours of snow and cumulated hours of rain.

Part A: Plot of the dependent variable versus time.

A plot of the dependent variable versus time can be found below.

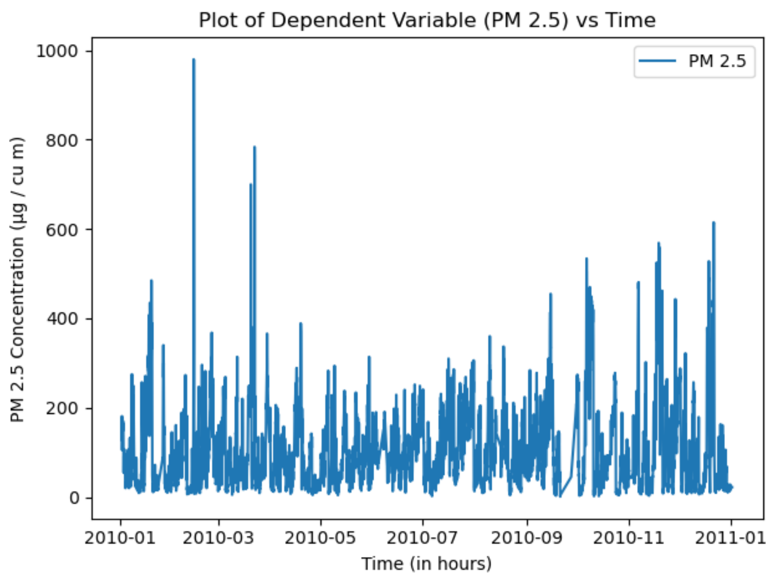


Fig 2. Plot of dependent variable versus time

The appearance of the plot in Figure 2 suggests that the dependent variable possesses moderately strong seasonal behavior with peaks at the beginning of each month. However, it is a bit difficult to discern distinct patterns in the dataset. Thus, the first month of data was plotted for clarity and can be found below in Figure 3.

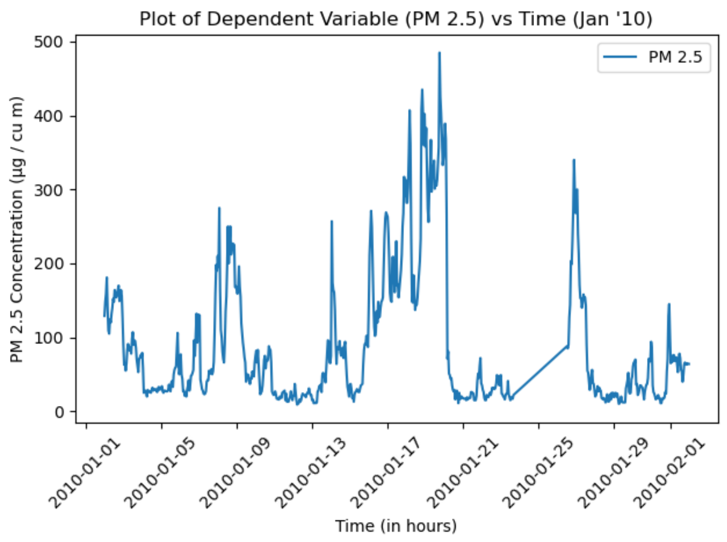


Fig 3. Plot of dependent variable versus time, January 2010

At the beginning of each week, there exists a large spike in PM 2.5 concentration followed by a decline in successive days. This is likely indicative of a weekly seasonality, but the ACF and PACF plots will explain better.

Part B: ACF of the dependent variable.

An assessment of correlation between the dependent variable and lagged versions of itself was conducted through the construction of a one-sided autocorrelation (ACF) plot, which can be found on the next page. The ACF values were calculated by implementing the equation in Figure 4 in Python.

$$r_k = \frac{\sum\limits_{t=k+1}^{T} (y_t - \bar{y})(y_{t-k} - \bar{y})}{\sum\limits_{t=1}^{T} (y_t - \bar{y})^2}$$

Fig 4. Autocorrelation function (Source: Hyndman & Athanasopoulos)

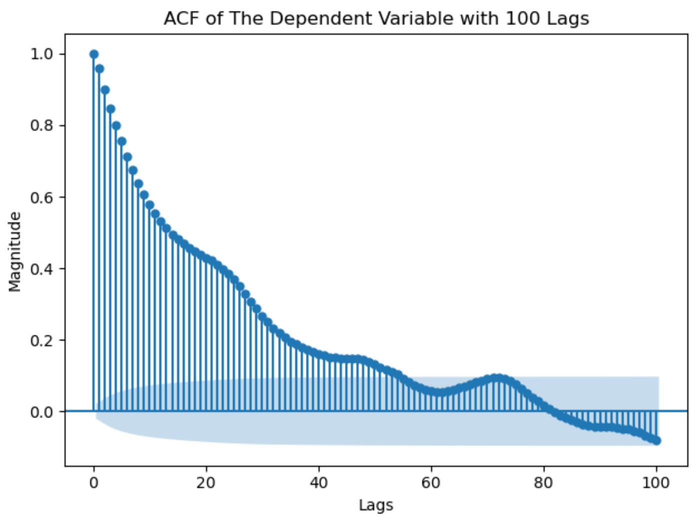


Fig 5. One-sided ACF plot of dependent variable (lags = 100)

For simplicity, the statsmodels library was used for this section. The appearance of the one-sided ACF plot in Figure 5 suggests that the dependent variable possesses a strong correlation with lagged versions of itself and is not stationary. Additionally, the PACF plot was constructed and can be found below in Figure 6.

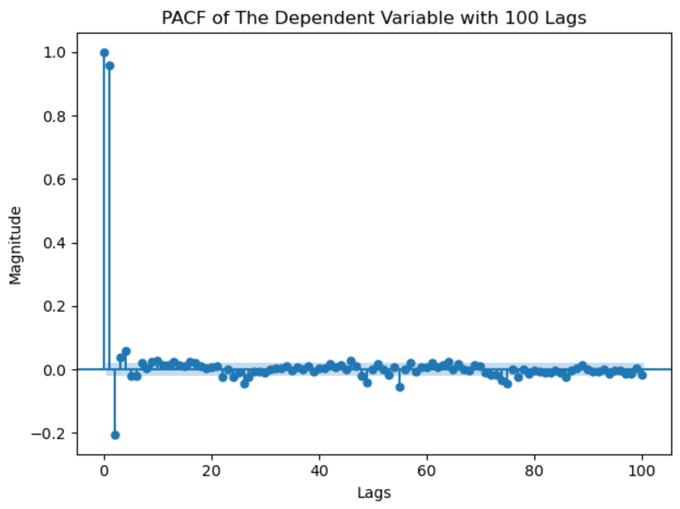


Fig 6. One-sided PACF plot of dependent variable (lags = 100)

The appearance of the PACF plot also suggests that the data is non-stationary. The reasoning behind why the ACF and PACF display non-stationary data is because both plots do not decay rapidly to values contained within the insignificance region. However, the spikes in values at roughly every 24 lags in the ACF seem to denote a seasonality period 24. More discussion on this issue can be found in the stationarity section of this document.

Part C: Correlation Matrix with seaborn heatmap and Pearson's correlation coefficient.

The correlation matrix of the dataset can be found below in Figure 7.

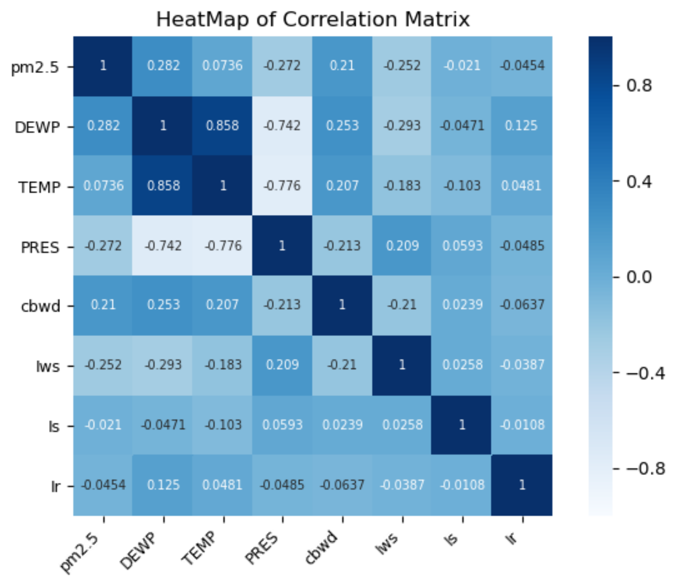


Fig 7. Correlation matrix using Pearson's correlation coefficient

The main takeaway from the correlation matrix is that the dependent variable possesses no strong correlation with any of the independent variables. The strongest correlation PM 2.5 has with any other variable is 0.282, which is with DEWP (dew point). However, correlation coefficient values below 0.4 are generally considered very weak. Notable strong correlation coefficients include the strong positive correlation between dew point and temperature (0.858) and the strong negative correlations between dew point and pressure (-0.742) and temperature and pressure (-0.776). These correlations are likely due to the climate of Beijing, which experiences hot, humid summers and cold, dry winters. However, the lack of strong correlations between the dependent variable and any of the independent variables serves as foreshadowing to the poor performance of multiple linear regression.

Part D: Preprocessing procedures (if applicable): Clean the dataset (no missing data or NAN)

Several preprocessing procedures were taken to clean the dataset. First, the date-time index was constructed using `Pandas.to_datetime()` with the columns for year, month, day and hour as inputs. Next, the irrelevant columns were dropped; this includes 'No' (an index column) and 'date_' (a column created in the process of constructing the date-time index; hours are handled as time deltas and cannot be formatted directly with `Pandas.to_datetime()`, so an extra step must be taken). Then, the missing values were filled. The only variable that contained missing values was the dependent variable (PM 2.5). The missing values were interpolated using `Pandas.DataFrame.interpolate()` with 'time' as the method argument, which is used when working with date-time indices. The combined wind direction ('cbwd') column was encoded due to its categorical nature. This includes setting NE to 0, NW to 1, SW to 2 and cv (calm and variable) to 3. The last preprocessing step taken was to split the dataset into training (80%) and testing (20%). This step was performed by using `sklearn.model_selection. train_test_split()`.

Stationarity

I want to note that this was perhaps the most difficult part of this project and where I spent most of my time working. No matter what differencing combinations I tried, the ACF and PACF plots did not resolve to values within the insignificance region after a few lags. However, after meeting with professor Jafari during office hours on December 15, 2020, he and I concluded that the data is seasonal and would require both seasonal and non-seasonal differencing. As mentioned in the description section, the ACF plot in Figure 5 displays spikes in values roughly every 24 lags; this means the seasonality period is likely 24. Thus, first-order seasonal differencing was performed on the dataset. The results can be found on the next page in Figure 8.

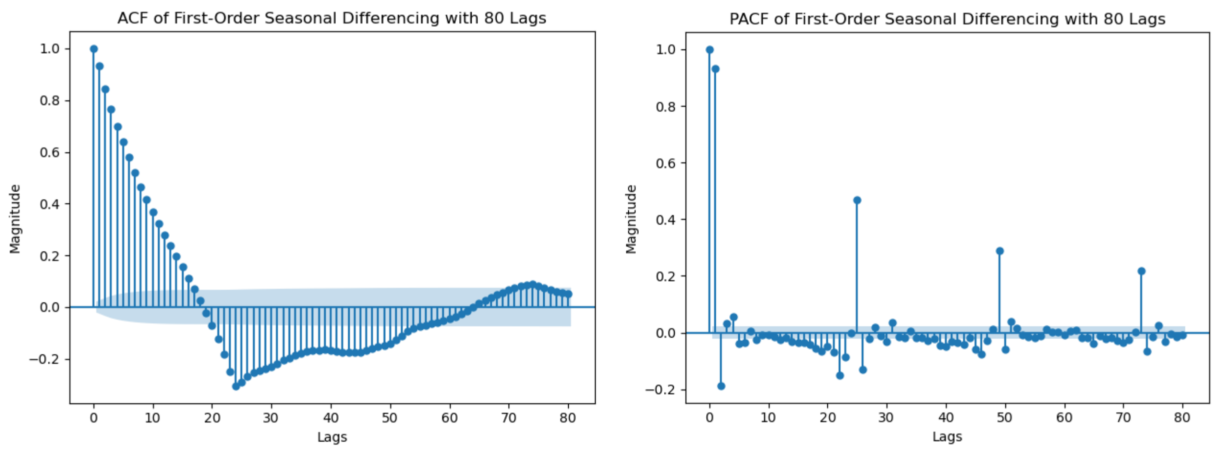


Fig 8. ACF and PACF plots of first-order seasonal differencing

The pattern of the ACF displays a rough cut-off at 24 lags, while the PACF displays tailing-off roughly every 24 lags; this is indicative of a seasonal MA pattern. However, during office hours it was suggested to try first-order differencing to test the change in ACF/PACF values. The results of first-order seasonal differencing followed by non-seasonal differencing can be found below.

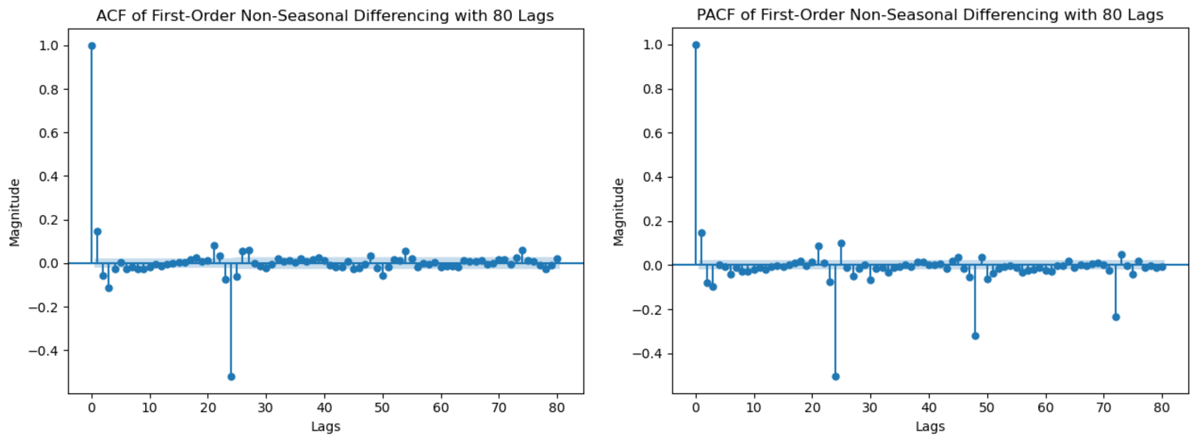


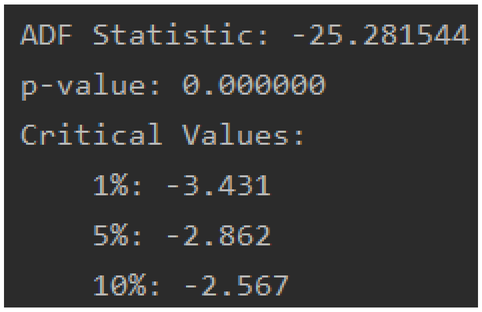
Fig 9. ACF and PACF plots of first-order seasonal differencing followed by first-order non-seasonal differencing

The results of first-order non-seasonal differencing display the ACF and PACF tailing off roughly every 24 lags; this is indicative of seasonal ARMA. Thus, for SARIMA I decided to try the following:

- Non-seasonal AR (2), seasonal AR (1) - ARMA (2, 0, 0) x ARMA (1, 0, 0, 24)

- Non-seasonal AR (2), seasonal AR (2) - ARMA (2, 0, 0) x ARMA (2, 0, 0, 24)
- Non-seasonal AR (1) , seasonal MA (1) with differencing - ARMA (1, 1, 0) x ARMA (0, 1, 1, 24)

The Augmented Dickey-Fuller test results of the first-order seasonally-differenced/first-order non-seasonally differenced data (i.e. the dataset whose ACF/PACF were plotted in Figure 9) can be found below in Figure 10.



```
ADF Statistic: -25.281544
p-value: 0.000000
Critical Values:
    1%: -3.431
    5%: -2.862
    10%: -2.567
```

Fig 10. Augmented Dickey-Fuller test results

The low value of the ADF statistic (-25.281544), combined with the near-zero p-value (0.000000) that's below the chosen threshold of 0.05 are enough to reject the null hypothesis and conclude that the dataset is stationary with 95% confidence.

Time Series Decomposition

In this section, the strength of the trend and seasonality components were computed using the equations provided on the next page in Figure 11.

$$\boxed{F_T = \max\{0, 1 - \frac{Var(R_t)}{Var(T_t + R_t)}\}}$$

$$\boxed{F_S = \max\{0, 1 - \frac{Var(R_t)}{Var(S_t + R_t)}\}}$$

Fig 11. Equations for strength of trend and seasonality

The strength of the trend was found to be 0.84717, while the strength of the seasonality is 0.41989. This means the raw dataset is strongly trended and moderately strongly seasonal. Because there exist prominent trend and seasonality components within the dataset, the data was de-trended and seasonally adjusted. Plots of the de-trended, seasonally adjusted, and original datasets can be found below in Figure 12.

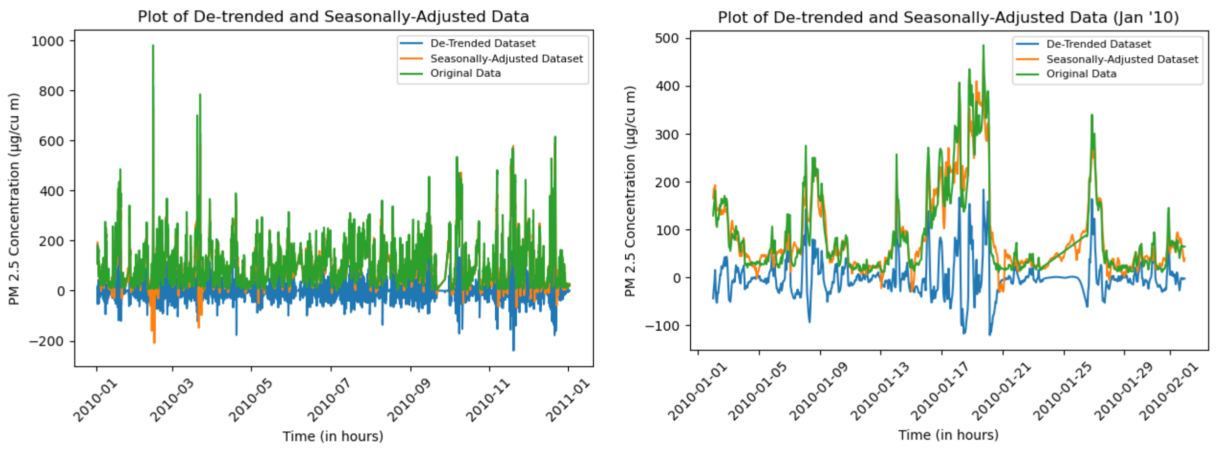
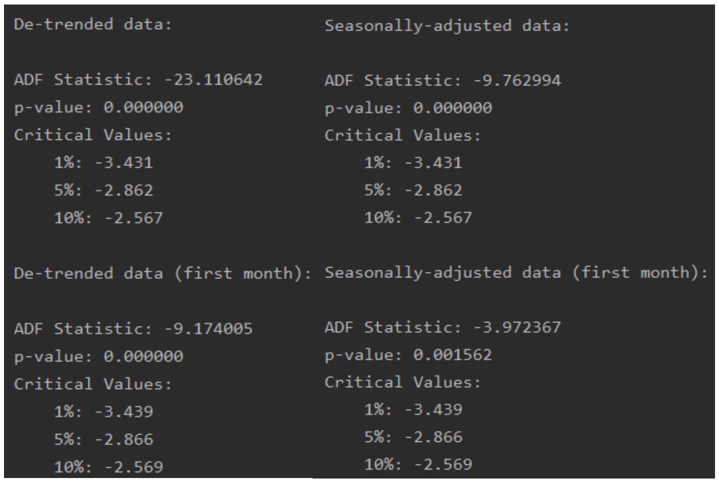


Fig 12. Plots of de-trended, seasonally adjusted, and original datasets

Because the plot of the full dataset (leftmost plot) is practically useless, the first month of data (rightmost plot) was plotted for analysis. The plot of de-trended and seasonally adjusted data displays a near-perfect overlay of seasonally adjusted data and the original dataset. The de-trended data lies below both the seasonally adjusted and original dataset curves. These results suggest that the seasonal adjustments made to the dependent variable do not incite significant changes; this aligns with the results of the strengths of trend and seasonality. The results of the ADF tests for the de-trended and seasonally adjusted data can be found below in Figure 13.



```
De-trended data:                Seasonally-adjusted data:

ADF Statistic: -23.110642       ADF Statistic: -9.762994
p-value: 0.000000               p-value: 0.000000
Critical Values:                Critical Values:
    1%: -3.431                      1%: -3.431
    5%: -2.862                      5%: -2.862
    10%: -2.567                     10%: -2.567

De-trended data (first month): Seasonally-adjusted data (first month):

ADF Statistic: -9.174005        ADF Statistic: -3.972367
p-value: 0.000000               p-value: 0.001562
Critical Values:                Critical Values:
    1%: -3.439                      1%: -3.439
    5%: -2.866                      5%: -2.866
    10%: -2.569                     10%: -2.569
```

Fig 13. Augmented Dickey-Fuller test results for de-trended and seasonally adjusted data

The results of the ADF tests are enough to reject the null hypothesis in both cases and conclude that due to the large negative value of the ADF statistic and the near-zero p-value, the de-trended and seasonally adjusted data are both stationary.

Holt-Winters Method

The Holt-Winters method was applied the raw dataset using multiplicative time series decomposition for the trend and additive time series decomposition for the seasonal. I had quite a bit of issues on this section. Although the seasonality period from Figure 3 appears to be 7, in the ACF plot in Figure 5 it appears to be 24. I tried both seasonality periods using Holt-Winters method; the results can be found below in Figure 14.

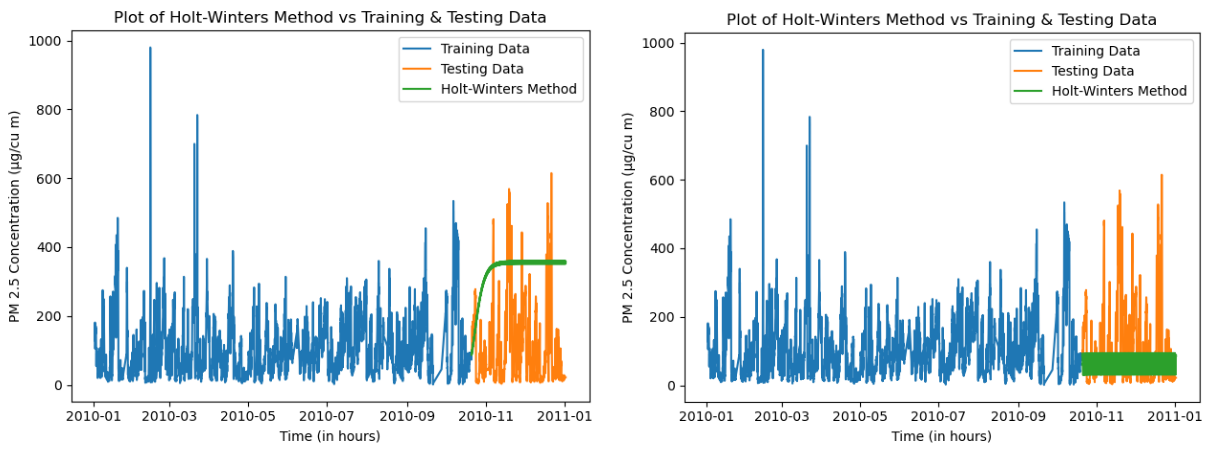


Fig 14. Holt-Winters method using seasonality periods of 7 (leftmost plot) and 24 (rightmost plot)

Personally, I believe the leftmost plot with a seasonality period of 7 better matches the test set by accounting for the spikes in PM 2.5 concentrations. Although the rightmost plot with a seasonality period of 24 covers the bulk of the test set values, it does not make very much sense to me; the predictions are a practical block instead of a line. I also tried applying Holt’s linear trend to the dataset. The results can be found on the next page in Figure 15.

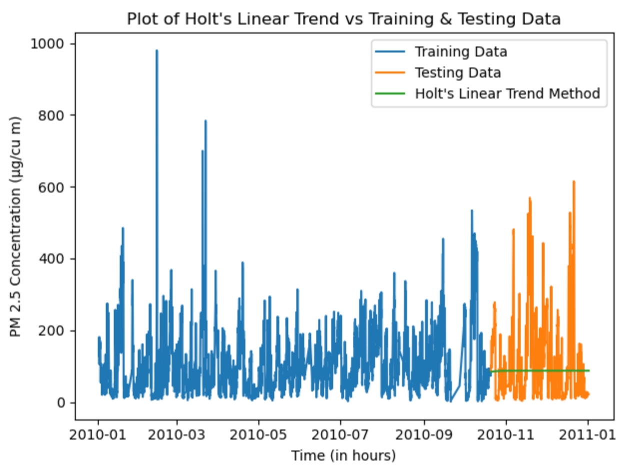


Fig 15. Plot of Holt's linear trend vs the training and testing sets

I tried changing the time series decomposition type for trend from multiplicative to additive, but it barely changed anything. This model performs very poorly in comparison to Holt-Winter method and is certainly not the best model for this dataset. Holt-Winters is a possible candidate, but likely doesn't perform as well as SARIMA.

Feature Selection

Feature selection was performed using both forward and backward stepwise regression. Upon discussing preliminary results of feature selection on the dataset during office hours on December 8, 2020, professor Jafari recommended removing the encoded categorical variable 'cbwd' (combined wind direction) from the dataset for forward and backward stepwise regression. First, forward stepwise regression was performed on the training set. The results can be found below.

```
Using only the constant ('const'):

    -   DEWP:     t-val = 27.135  |   p-val = 0.000
    -   TEMP:     t-val = 9.494   |   p-val = 0.000
    -   PRES:     t-val = -21.489 |   p-val = 0.000
    -   Iws:      t-val = -20.924 |   p-val = 0.000
    -   Is:       t-val = -2.033  |   p-val = 0.042
    -   Ir:       t-val = -4.569  |   p-val = 0.000

Between the variables with the smallest p-val, dew point (DEWP) has the largest t-val and is put
into the basket.

    -   TEMP:     t-val = -26.454 |   p-val = 0.000
    -   PRES:     t-val = -1.570  |   p-val = 0.117
    -   Iws:      t-val = -14.649 |   p-val = 0.000
    -   Is:       t-val = -0.281  |   p-val = 0.779
    -   Ir:       t-val = -7.863  |   p-val = 0.000
```

```
Between the variables with the smallest p-val, cumulative hours of rain has the largest t-val and
is put into the basket.
```

```
    -   TEMP:     t-val = -28.078 |   p-val = 0.000
    -   PRES:     t-val = -0.983  |   p-val = 0.325
    -   Iws:      t-val = -14.779 |   p-val = 0.000
    -   Is:       t-val = -0.327  |   p-val = 0.744
```

```
Between the variables with the smallest p-val, cumulative wind speed has the largest t-val and is
put into the basket.
```

```
    -   TEMP:     t-val = -25.512 |   p-val = 0.000
    -   PRES:     t-val = -2.814  |   p-val = 0.005
    -   Is:       t-val = 0.103   |   p-val = 0.918
```

```
Temperature possessed the smallest p-value below the chosen threshold (0.005) and is put into the
basket.
```

```
    -   PRES:     t-val = -17.240 |   p-val = 0.000
    -   Is:       t-val = -9.540  |   p-val = 0.000
```

```
Between the variables with the smallest p-val, cumulative hours of snow has the largest t-val and
is put into the basket.
```

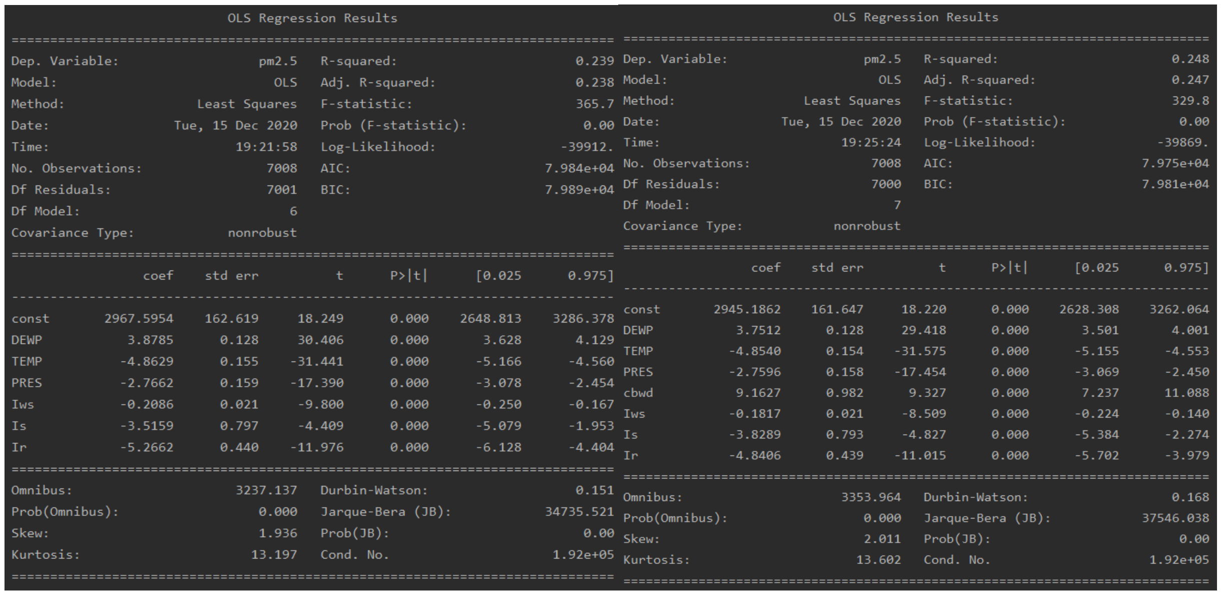
```
    -   PRES:     t-val = -17.390 |   p-val = 0.000
```

```
The p-value for pressure is below the chosen threshold (0.05) and is put into the basket.
```

In summary, at each stage of forward stepwise regression there existed a feature whose p-value was below the chosen threshold (0.05). Thus, no features were eliminated using forward stepwise regression. For backward stepwise regression, all features were considered. However, the summary of considering all features revealed the same results as the last step of forward stepwise regression, since no features were eliminated. In the first step of backward stepwise regression, there existed no features whose p-value was above the chosen threshold (0.05). Thus, no features were eliminated from the basket using backward stepwise regression. This means the results of feature selection conclude that all variables should be considered. Discussion on R-squared values and the implications of keeping all features is discussed in the next section.

Multiple Linear Regression

The results of feature selection resolve to using every feature in the dataset. I decided to exclude the categorical variable (combined wind direction) to maintain consistency from feature selection, and then put it back into the dataset and re-run multiple linear regression to compare results. The results of running multiple linear regression without the categorical variable can be found below in Figure 16.



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  pm2.5   R-squared:                       0.239
Model:                            OLS   Adj. R-squared:                  0.238
Method:                 Least Squares   F-statistic:                     365.7
Date:                Tue, 15 Dec 2020   Prob (F-statistic):               0.00
Time:                        19:21:58   Log-Likelihood:                -39912.
No. Observations:                7008   AIC:                         7.984e+04
Df Residuals:                    7001   BIC:                         7.989e+04
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       2967.5954    162.619     18.249      0.000    2648.813    3286.378
DEWP           3.8785      0.128     30.406      0.000       3.628       4.129
TEMP          -4.8629      0.155    -31.441      0.000      -5.166      -4.560
PRES          -2.7662      0.159    -17.390      0.000      -3.078      -2.454
Iws           -0.2086      0.021     -9.800      0.000      -0.250      -0.167
Is            -3.5159      0.797     -4.409      0.000      -5.079      -1.953
Ir            -5.2662      0.440    -11.976      0.000      -6.128      -4.404
==============================================================================
Omnibus:                     3237.137   Durbin-Watson:                   0.151
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            34735.521
Skew:                           1.936   Prob(JB):                         0.00
Kurtosis:                      13.197   Cond. No.                     1.92e+05
==============================================================================
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  pm2.5   R-squared:                       0.248
Model:                            OLS   Adj. R-squared:                  0.247
Method:                 Least Squares   F-statistic:                     329.8
Date:                Tue, 15 Dec 2020   Prob (F-statistic):               0.00
Time:                        19:25:24   Log-Likelihood:                -39869.
No. Observations:                7008   AIC:                         7.975e+04
Df Residuals:                    7000   BIC:                         7.981e+04
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       2945.1862    161.647     18.220      0.000    2628.308    3262.064
DEWP           3.7512      0.128     29.418      0.000       3.501       4.001
TEMP          -4.8540      0.154    -31.575      0.000      -5.155      -4.553
PRES          -2.7596      0.158    -17.454      0.000      -3.069      -2.450
cbwd           9.1627      0.982      9.327      0.000       7.237      11.088
Iws           -0.1817      0.021     -8.509      0.000      -0.224      -0.140
Is            -3.8289      0.793     -4.827      0.000      -5.384      -2.274
Ir            -4.8406      0.439    -11.015      0.000      -5.702      -3.979
==============================================================================
Omnibus:                     3353.964   Durbin-Watson:                   0.168
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            37546.038
Skew:                           2.011   Prob(JB):                         0.00
Kurtosis:                      13.602   Cond. No.                     1.92e+05
==============================================================================
```

Fig 16. Results of multiple linear regression without cbwd (leftmost graphic) and with cbwd (rightmost graphic)

The relevant statistics of multiple linear regression are very similar with and without combined wind direction included. For the sake of simplicity, I am excluding cbwd since it wasn't included in feature selection.

Part A: You need to include the complete regression analysis into your report. Perform one-step ahead prediction and compare the performance versus the test set.

The wording of this question prompt was moderately confusing; I am under the impression that one-step ahead predictions are compared against the training set, while multi-step ahead predictions are compared against the testing set. Thus, plots of the one-step ahead predictions and their comparisons to the training set can be found on the next page in Figure 17. The first month of data was also plotted to get a better sense of how well the predictions compare to the training set.

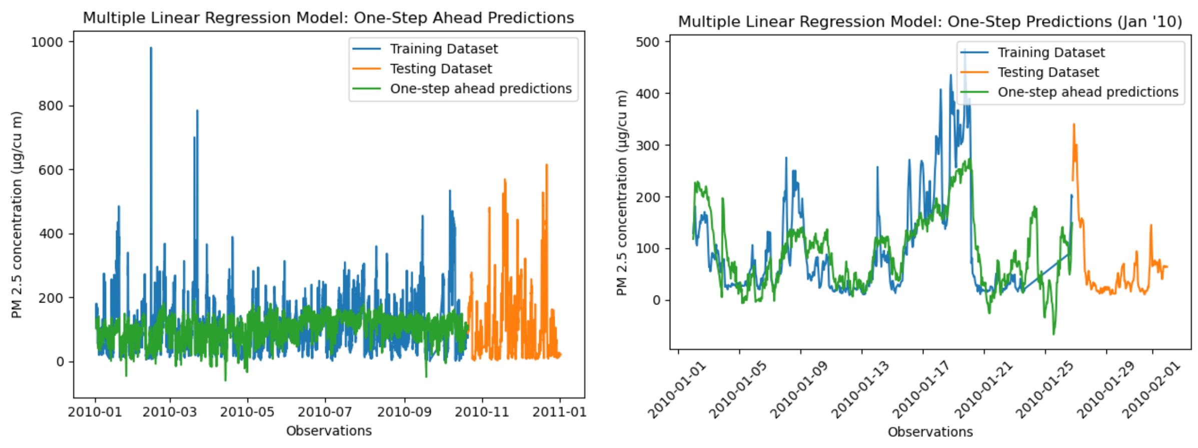


Fig 17. One-step ahead predictions

The rightmost plot shows that the predictions match the overall shape of the training data well, but individual data points are not extremely close. The spikes in the training data are not well-represented by the one-step ahead predictions, which is reinforced by the leftmost plot. Plots of the multi-step ahead predictions and their comparisons to the test set can be found below in Figure 18.

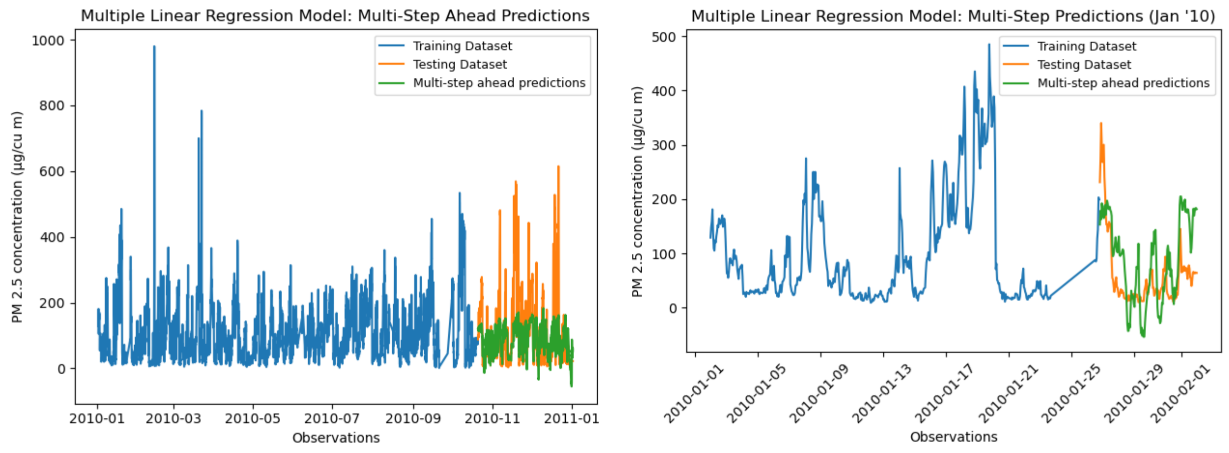


Fig 18. Multi-step ahead predictions

The rightmost plot displays forecasts that do not match up well with the testing set. The multi-step ahead forecasts show oscillations of spikes in value, while the testing set shows a steep decline and leveling-out of values before slowly increasing again. Due to the poor matchups in forecasts and testing set values, this is likely not a good model. This notion is supported by analysis of relevant test statistics in the remaining parts of this section.

Part B: Hypothesis tests: F-test, t-test.

Evidence from the test results in Figure 16 (without cbwd) display the p-value of the F-statistic as 0.00. Because the p-value of the F-test is below the chosen threshold of 0.05, we reject the null hypothesis and conclude that the model provides a better fit than the intercept-only model. The results of the t-tests for the coefficients of the predictors all display a p-value of 0.000. Because the p-value of the t-tests are below the chosen threshold of 0.05, we reject the null hypothesis for all cases and conclude that all the coefficients are not equal to zero.

Part C: AIC, BIC, RMSE, R-squared and Adjusted R-squared

The results in Figure 16 display AIC and BIC scores of 79,840 and 79,890, respectively, which are not very good. As shown in Figure 19, the AIC and BIC values are directly proportional to the SSE; this means the SSE is also very high.

$$\boxed{AIC = T\log(\frac{SSE}{T}) + 2(k+2)}$$

$$\boxed{BIC = T\log(\frac{SSE}{T}) + (k+2)\log(T)}$$

Fig 19. Equations for AIC and BIC

If the SSE is very high, then there is a large degree of variation in this multiple linear regression model. Similarly, the R-squared and adjusted R-squared values are very small (0.239 and 0.238, respectively). This is perhaps the most glaring and disappointing statistic of this model. Because the R-squared and adjusted R-squared values are so small, only a small proportion (23.8% for R-squared and 23.9% for adjusted R-squared) of the dependent variable (PM 2.5 concentration) can be reasonably explained by an independent variable of the model. I want to note that while discussing feature selection during office hours on December 8, 2020, the student that is using the same dataset mentioned to the professor and I that his R-squared value was around 0.6. I have no idea how he was able to get the R-squared value that high and would appreciate feedback on any recommendations to accurately improve that statistic. The root-mean squared error for the one-step ahead predictions was found to be 71.9691, while the RSME for multi-step ahead predictions was found to be 96.7449. The large values of the RSME suggest that there exists a large deviation of the residuals and forecast errors. This means that the data is not very concentrated around the line of best fit for the residuals and forecast errors.

Part D: ACF of residuals.

The ACF values of the residuals were computed by applying the autocorrelation function in Figure 20 to a list containing the difference of the values of the training set and the one-step ahead predictions.

$$\boxed{\hat{\tau}_k = \frac{\sum_{t=k+1}^{T}(y_t - \overline{y})(y_{t-k} - \overline{y})}{\sum_{t=1}^{T}(y_t - \overline{y})^2}}$$

Fig 20. Equation for autocorrelation function

A plot of the ACF of the residuals can be found below in Figure 21.

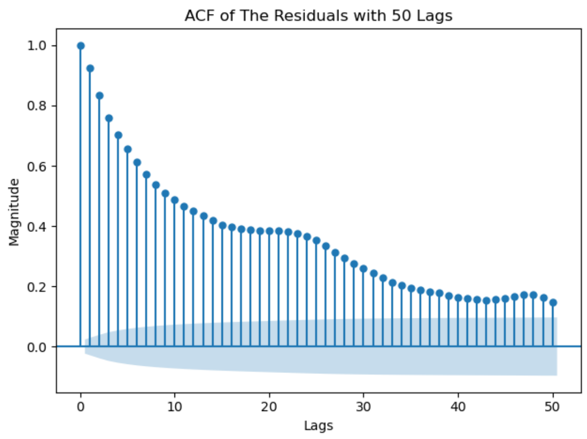


Fig 21. ACF plot of the residuals

The ACF plot above follows a similar pattern to the ACF plot of the raw data in Figure 5; there exists spikes in values roughly every 24 values, and the ACF values themselves are high and outside of the insignificance region.

Part E: Q-value

The Box-Pierce test Q-value was computed using the equation provided below in Figure 22.

$$Q = T\sum_{k=1}^{h} r_k^2$$

Fig 22. Q-value of the residuals

The resulting Q-value was 56276.18381704555. The large nature of the Q-value can be attributed to the large ACF values present in Figure 21. For seasonal data, it is recommended to use h = 2m for seasonal data, where m is the period of seasonality. Because the period of seasonality is roughly 24 or 25, h was set to 50 for convenience.

Part F: Variance and mean of the residuals.

The mean and variance of the residuals was computed using Numpy's `.mean()` and `.var()` methods. The mean of the residuals was found to be -3.7480e^{-12}, while the variance was found to be 5179.55916. The near-zero mean of the residuals suggests that this model is an unbiased estimator, but the variance is concerning. The variance follows the pattern of the RSME, whose large value suggests a proportionately large deviation from the line of best fit for the residuals.

ARMA (ARIMA or SARIMA) Model

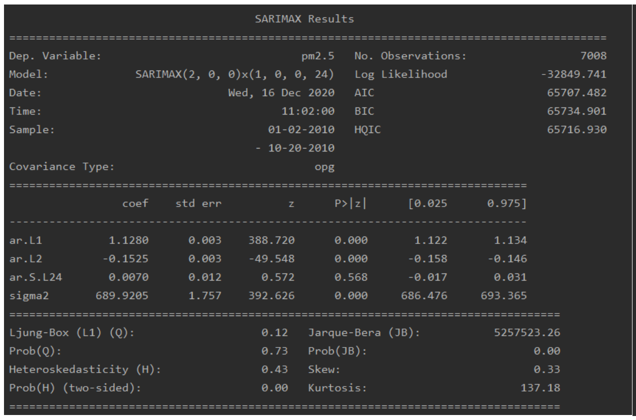
After speaking with professor Jafari during office hours on December 15, 2020, we came to an agreement that the stationary ACF and PACF plots constructed in Figures 8 and 9 follow AR seasonal, MA seasonal or ARMA seasonal patterns and that SARIMA would be the only relevant model of the three ARMA variations for this project; I am discarding ARMA and ARIMA due to the seasonal component present in my data. There is no need for order estimation using the GPAC table because the order or AR and MA can be deduced from looking at the PACF and ACF plots, respectively. The PACF plot in Figure 8 hints at a seasonal AR order of 1 or two and a non-seasonal order of 2, while the ACF hints at a MA order of 0 (seasonal and non-seasonal) due to the cutoff around lag 24. The PACF in Figure 9 hints at a seasonal AR order of 0 and non-seasonal order of 1, while the ACF hints at a seasonal MA order of 1 and non-seasonal MA order of 0. Thus, three model order estimates were constructed and tested.

- SARIMA (2, 0, 0) x (1, 0, 0, 24)
- SARIMA (2, 0, 0) x (2, 0, 0, 24)
- SARIMA (1, 1, 0) x (0, 1, 1, 24)

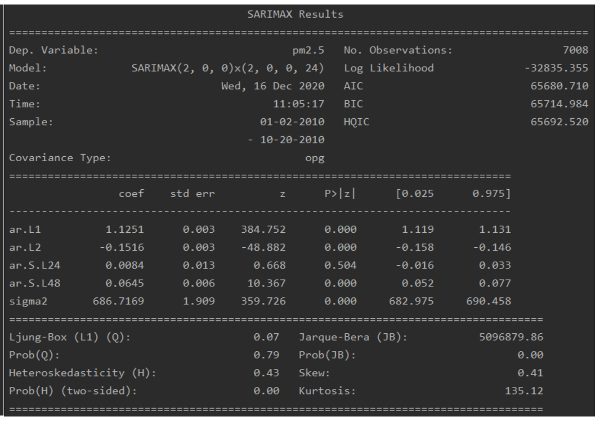
These models are tested and compared against each other in the diagnostic analysis and final model sections below.

Levenberg Marquardt Algorithm

Because there exists a seasonal component in the dataset, SARIMA will be used for this section in place of ARMA. Thus, applying the Levenberg Marquardt algorithm is automated through the SARIMAX call. The results of the three SARIMAX order estimates provided in the previous section can be found below in Figures 23 and 24.

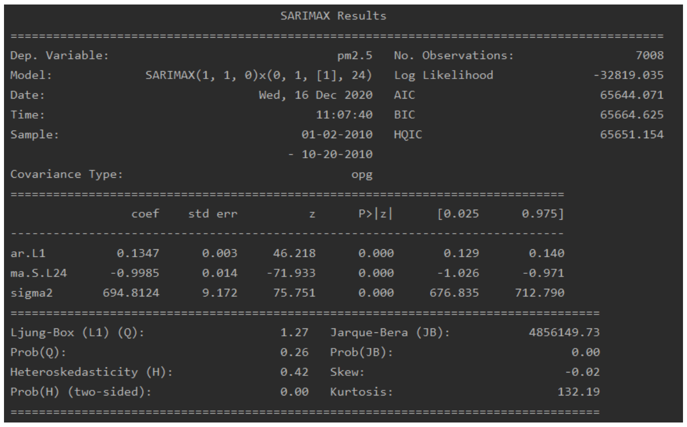


```
                                    SARIMAX Results
========================================================================================
Dep. Variable:                            pm2.5   No. Observations:                 7008
Model:             SARIMAX(2, 0, 0)x(1, 0, 0, 24)   Log Likelihood              -32849.741
Date:                          Wed, 16 Dec 2020   AIC                          65707.482
Time:                                  11:02:00   BIC                          65734.901
Sample:                              01-02-2010   HQIC                         65716.930
                                   - 10-20-2010
Covariance Type:                            opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.1280      0.003    388.720      0.000       1.122       1.134
ar.L2         -0.1525      0.003    -49.548      0.000      -0.158      -0.146
ar.S.L24       0.0070      0.012      0.572      0.568      -0.017       0.031
sigma2       689.9205      1.757    392.626      0.000     686.476     693.365
===================================================================================
Ljung-Box (L1) (Q):                   0.12   Jarque-Bera (JB):          5257523.26
Prob(Q):                              0.73   Prob(JB):                        0.00
Heteroskedasticity (H):               0.43   Skew:                            0.33
Prob(H) (two-sided):                  0.00   Kurtosis:                      137.18
===================================================================================
```



```
                                    SARIMAX Results
========================================================================================
Dep. Variable:                            pm2.5   No. Observations:                 7008
Model:             SARIMAX(2, 0, 0)x(2, 0, 0, 24)   Log Likelihood              -32835.355
Date:                          Wed, 16 Dec 2020   AIC                          65680.710
Time:                                  11:05:17   BIC                          65714.984
Sample:                              01-02-2010   HQIC                         65692.520
                                   - 10-20-2010
Covariance Type:                            opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.1251      0.003    384.752      0.000       1.119       1.131
ar.L2         -0.1516      0.003    -48.882      0.000      -0.158      -0.146
ar.S.L24       0.0084      0.013      0.668      0.504      -0.016       0.033
ar.S.L48       0.0645      0.006     10.367      0.000       0.052       0.077
sigma2       686.7169      1.909    359.726      0.000     682.975     690.458
===================================================================================
Ljung-Box (L1) (Q):                   0.07   Jarque-Bera (JB):          5096879.86
Prob(Q):                              0.79   Prob(JB):                        0.00
Heteroskedasticity (H):               0.43   Skew:                            0.41
Prob(H) (two-sided):                  0.00   Kurtosis:                      135.12
===================================================================================
```

Fig 23. Results of SARIMA (2, 0, 0) x (1, 0, 0, 24) (leftmost graphic) and SARIMA (2, 0, 0) x (2, 0, 0, 24) (rightmost graphic)



```
                                    SARIMAX Results
==========================================================================================
Dep. Variable:                              pm2.5   No. Observations:                 7008
Model:             SARIMAX(1, 1, 0)x(0, 1, [1], 24)   Log Likelihood              -32819.035
Date:                            Wed, 16 Dec 2020   AIC                          65644.071
Time:                                    11:07:40   BIC                          65664.625
Sample:                                01-02-2010   HQIC                         65651.154
                                     - 10-20-2010
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.1347      0.003     46.218      0.000       0.129       0.140
ma.S.L24      -0.9985      0.014    -71.933      0.000      -1.026      -0.971
sigma2       694.8124      9.172     75.751      0.000     676.835     712.790
===================================================================================
Ljung-Box (L1) (Q):                   1.27   Jarque-Bera (JB):          4856149.73
Prob(Q):                              0.26   Prob(JB):                        0.00
Heteroskedasticity (H):               0.42   Skew:                           -0.02
Prob(H) (two-sided):                  0.00   Kurtosis:                      132.19
===================================================================================
```

Fig 24. Results of SARIMA (1, 1, 0) x (0, 1, 1, 24)

The SARIMA parameter estimates, standard deviation of the parameter estimates, and confidence intervals can be found below. The standard deviations were calculated by applying the equation for the confidence interval. After emailing professor Jafari, he stated that the entire square root is the value of the standard deviation of the parameter estimates. The equation referenced can be found below in Figure 25.

$$\hat{\theta}_i \pm 2\sqrt{[\Sigma\hat{\theta}]_{ii}}$$

Fig 25. Equation for confidence interval of parameter estimates

- SARIMA (2, 0, 0) x (1, 0, 0, 24)

```
Coefficient a1: 1.1222654355065422 <= 1.1279526764749037 <= 1.1336399174432652
Standard deviation of a1: 0.003
Coefficient a2: -0.15848448670810195 <= -0.15245390973782344 <= -0.14642333276754493
Standard deviation of a2: 0.00325
Coefficient a24: -0.01711322118304881 <= 0.007047591869033003 <= 0.03120840492111482
Standard deviation of a24: 0.0012
```

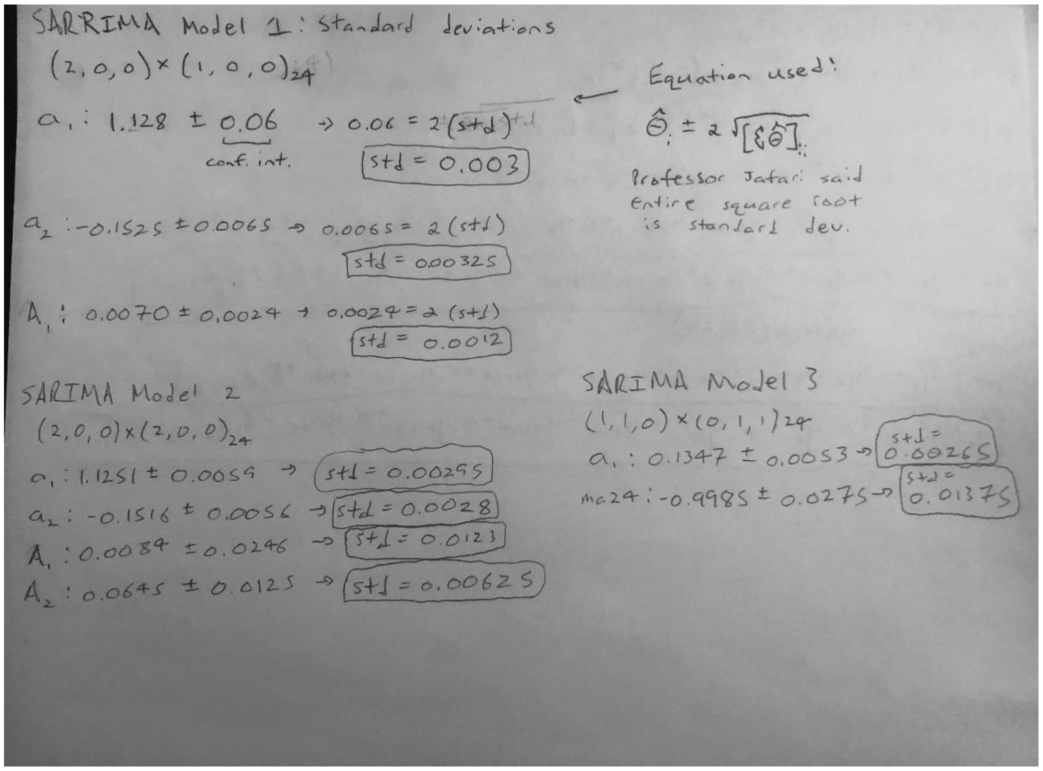
- SARIMA (2, 0, 0) x (2, 0, 0, 24)

```
Coefficient a1: 1.1194146680343355 <= 1.1251462649378206 <= 1.1308778618413056
Standard deviation of a1: 0.00295
Coefficient a2: -0.1576694897423183 <= -0.15159133150759355 <= -0.1455131732728688
Standard deviation of a2: 0.0028
Coefficient a24: -0.0162791166139701 <= 0.00841766059027725 <= 0.0331144377945246
Standard deviation of a24: 0.0123
Coefficient a48: 0.05234432799481155 <= 0.06454730013698495 <= 0.07675027227915836
Standard deviation of a48: 0.00625
```

- SARIMA (1, 1, 0) x (0, 1, 1, 24)

```
Coefficient a1: 0.128957501410126 <= 0.1346683757311466 <= 0.1403792500521672
Standard deviation of a1: 0.0028
Coefficient ma24: -1.0257474859992932 <= -0.9985400423389551 <= -0.9713325986786169
Standard deviation of ma24: 0.0028
```

The handwritten work done to compute the standard deviation values can be found in Figure 26 in the graphic below.



SARRIMA Model $\underline{1}$: Standard deviations

$(2, 0, 0) \times (1, 0, 0)_{24}$

a_1: $1.128 \pm \underbrace{0.06}_{\text{conf. int.}} \rightarrow 0.06 = 2(\text{std})$

$\boxed{\text{std} = 0.003}$

a_2: $-0.1525 \pm 0.0065 \rightarrow 0.0065 = 2(\text{std})$

$\boxed{\text{std} = 0.00325}$

A_1: $0.0070 \pm 0.0024 \rightarrow 0.0024 = 2(\text{std})$

$\boxed{\text{std} = 0.0012}$

Equation used:

$\hat{\Theta}_i \pm 2\sqrt{[\xi\hat{\Theta}]_{ii}}$

Professor Jafari said Entire square root is standard dev.

SARIMA Model 2

$(2, 0, 0) \times (2, 0, 0)_{24}$

a_1: $1.1251 \pm 0.0059 \rightarrow \boxed{\text{std} = 0.00295}$

a_2: $-0.1516 \pm 0.0056 \rightarrow \boxed{\text{std} = 0.0028}$

A_1: $0.0084 \pm 0.0246 \rightarrow \boxed{\text{std} = 0.0123}$

A_2: $0.0645 \pm 0.0125 \rightarrow \boxed{\text{std} = 0.00625}$

SARIMA Model 3

$(1, 1, 0) \times (0, 1, 1)24$

a_1: $0.1347 \pm 0.0053 \rightarrow \boxed{\text{std} = 0.00265}$

ma24: $-0.9985 \pm 0.0275 \rightarrow \boxed{\text{std} = 0.01375}$

Fig 26. Handwritten computation of standard deviation values

Diagnostic Analysis

Upon speaking with professor Jafari during office hours on December 15, 2020, I will be deriving the three SARIMA models discussed earlier in this report. I used a lag value of 55 for this section.

Part A: Diagnostic tests (confidence intervals, zero/pole cancellation, chi-square test).

SARIMA (2, 0, 0) x (1, 0, 0, 24)

- The confidence intervals for the coefficients are as follows:

```
Coefficient a1: 1.1222654355065422 <= 1.1279526764749037 <= 1.1336399174432652
Coefficient a2: -0.15848448670810195 <= -0.15245390973782344 <= -0.14642333276754493
Coefficient a24: -0.01711322118304881 <= 0.007047591869033003 <= 0.03120840492111482
```

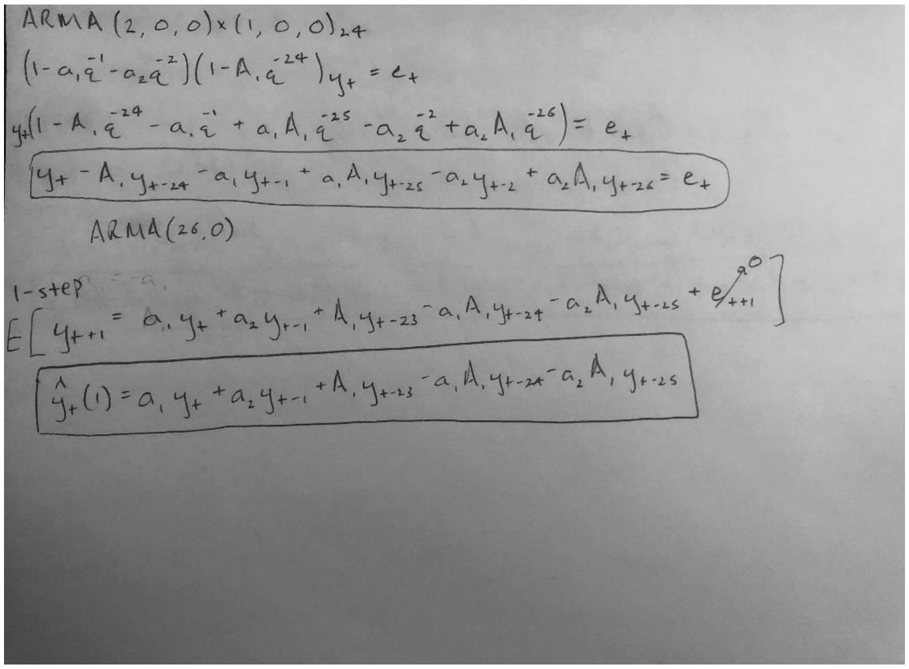
- The zeros and poles can be found below and on the next page:

```
Zeros of SARIMA (2, 0, 0) x (1, 0, 0, 24): [0.]
```

```
Poles of SARIMA (2, 0, 0) x (1, 0, 0, 24): [-1.2540077 +0.j,

0.06302751+0.04059049j,  0.06302751-0.04059049j]
```

Thankfully, none of the zeros and poles cancel out. Thus, the order of the SARIMA model doesn't need to be reduced.

- A chi-squared test for whiteness of the residuals resulted in computation of the one-step ahead predictions; this was done by hand instead of using `model.predict()`, due to concerns surrounding the accuracy of the prebuilt function. The handwritten work to find the one-step ahead predictions can be found below in Figure 27.



ARMA $(2,0,0)\times(1,0,0)_{24}$

$$(1-a_1q^{-1}-a_2q^{-2})(1-A_1q^{-24})y_t = e_t$$

$$y_t(1-A_1q^{-24}-a_1q^{-1}+a_1A_1q^{-25}-a_2q^{-2}+a_2A_1q^{-26}) = e_t$$

$$\boxed{y_t - A_1y_{t-24} - a_1y_{t-1} + a_1A_1y_{t-25} - a_2y_{t-2} + a_2A_1y_{t-26} = e_t}$$

ARMA$(26,0)$

1-step

$$E\left[y_{t+1} = a_1y_t + a_2y_{t-1} + A_1y_{t-23} - a_1A_1y_{t-24} - a_2A_1y_{t-25} + \overset{0}{\cancel{e_{t+1}}}\right]$$

$$\boxed{\hat{y}_t(1) = a_1y_t + a_2y_{t-1} + A_1y_{t-23} - a_1A_1y_{t-24} - a_2A_1y_{t-25}}$$

Fig 27. Handwritten computation of one-step ahead predictions for SARIMA (2, 0, 0) x (1, 0, 0, 24)

- The results of the chi-squared test display a Q value of 7261.5485 and a critical Q value of 42.5569 (alpha = 0.05, 29 degrees of freedom). Because Q > critical Q, we reject the null hypothesis of the test and conclude that the residuals are not white.

SARIMA (2, 0, 0) x (2, 0, 0, 24)

- The confidence intervals for the coefficients are as follows:

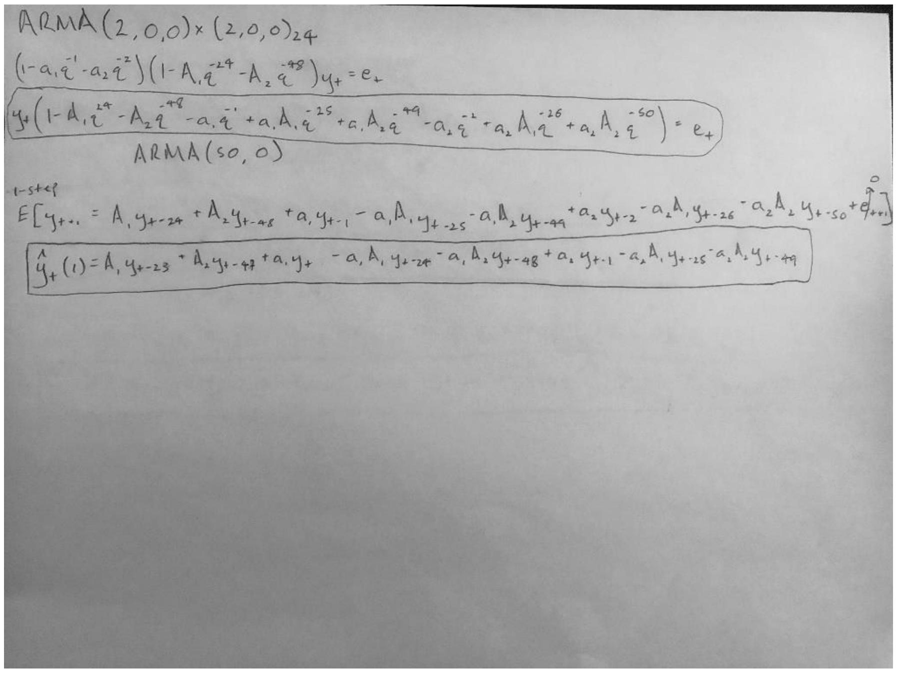
```
Coefficient a1: 1.1194146680343355 <= 1.1251462649378206 <= 1.1308778618413056

Coefficient a2: -0.1576694897423183 <= -0.15159133150759355 <= -0.1455131732728688

Coefficient a24: -0.0162791166139701 <= 0.00841766059027725 <= 0.0331144377945246

Coefficient a48: 0.0523443279948115‍5 <= 0.06454730013698495 <= 0.07675027227915836
```

- The zeros and poles can be found below and on the next page:

```
Zeros of SARIMA (2, 0, 0) x (2, 0, 0, 24): [0.]

Poles of SARIMA (2, 0, 0) x (2, 0, 0, 24): [-1.21952165+0.j, -0.3763312 +0.j,

0.23535329+0.29197928j, 0.23535329-0.29197928j]
```

Thankfully, none of the zeros and poles cancel out. Thus, the order of the SARIMA model does not need to be reduced.

- A chi-squared test for whiteness of the residuals resulted in computation of the one-step ahead predictions; the handwritten work to find the one-step ahead predictions can be found below in Figure 28.



ARMA$(2,0,0)\times(2,0,0)_{24}$

$(1-a_1q^{-1}-a_2q^{-2})(1-A_1q^{-24}-A_2q^{-48})y_t=e_t$

$$\boxed{y_t(1-A_1q^{24}-A_2q^{-48}-a_1q^{-1}+a_1A_1q^{-25}+a_1A_2q^{-49}-a_2q^{-2}+a_2A_1q^{-26}+a_2A_2q^{-50})=e_t}$$

ARMA$(50,0)$

1-step

$E[y_{t+1}=A_1y_{t-24}+A_2y_{t-48}+a_1y_{t-1}-a_1A_1y_{t-25}-a_1A_2y_{t-49}+a_2y_{t-2}-a_2A_1y_{t-26}-a_2A_2y_{t-50}+\overset{0}{\hat{e}}_{t+1}]$

$$\boxed{\hat{y}_t(1)=A_1y_{t-23}+A_2y_{t-47}+a_1y_t-a_1A_1y_{t-24}-a_1A_2y_{t-48}+a_2y_{t-1}-a_2A_1y_{t-25}-a_2A_2y_{t-49}}$$

Fig 28. Handwritten computation of one-step ahead predictions for SARIMA (2, 0, 0) x (2, 0, 0, 24)

- The results of the chi-squared test display a Q value of 7227.7257 and a critical Q value of 11.0704 (alpha = 0.05, 5 degrees of freedom). Because Q > critical Q, we reject the null hypothesis of the test and conclude that the residuals are not white.

SARIMA (1, 1, 0) x (0, 1, 1, 24)

- The confidence intervals for the coefficients are as follows:

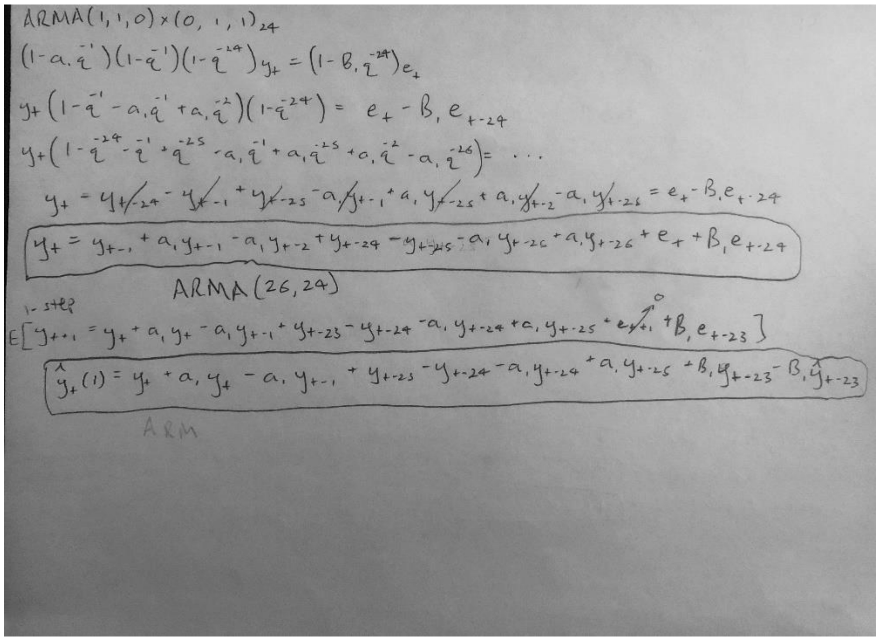
```
Coefficient a1: 0.128957501410126 <= 0.1346683757311466 <= 0.1403792500521672
Coefficient ma24: -1.0257474859992932 <= -0.9985400423389551 <= -0.9713325986786169
```

- The zeros and poles are as follows:

```
Zeros of SARIMA (1, 1, 0) x (0, 1, 1, 24): [0.99854004]
Poles of SARIMA (1, 1, 0) x (0, 1, 1, 24): [-0.13466838]
```

Thankfully, none of the zeros and poles cancel out. Thus, the order of the SARIMA model does not need to be reduced.

- A chi-squared test for whiteness of the residuals resulted in computation of the one-step ahead predictions; the handwritten work to find the one-step ahead predictions can be found below in Figure 29.



ARMA$(1,1,0)\times(0,1,1)_{24}$

$(1-a_1 q^{-1})(1-q^{-1})(1-q^{-24})y_t = (1-B_1 q^{-24})e_t$

$y_t(1-q^{-1}-a_1 q^{-1}+a_1 q^{-2})(1-q^{-24}) = e_t - B_1 e_{t-24}$

$y_t(1-q^{-24}-q^{-1}+q^{-25}-a_1 q^{-1}+a_1 q^{-25}+a_1 q^{-2}-a_1 q^{-26}) = \cdots$

$y_t - \cancel{y}_{t-24} - \cancel{y}_{t-1} + \cancel{y}_{t-25} - a_1\cancel{y}_{t-1} + a_1\cancel{y}_{t-25} + a_1\cancel{y}_{t-2} - a_1\cancel{y}_{t-26} = e_t - B_1 e_{t-24}$

$\boxed{y_t = y_{t-1} + a_1 y_{t-1} - a_1 y_{t-2} + y_{t-24} - y_{t-25} - a_1 y_{t-26} + a_1 y_{t-26} + e_t + B_1 e_{t-24}}$

ARMA(26, 24)

1-step

$E[y_{t+1} = y_t + a_1 y_t - a_1 y_{t-1} + y_{t-23} - y_{t-24} - a_1 y_{t-24} + a_1 y_{t-25} + \overset{0}{\cancel{e_{t+1}}} + B_1 e_{t-23}]$

$\boxed{\hat{y}_t(1) = y_t + a_1 y_t - a_1 y_{t-1} + y_{t-23} - y_{t-24} - a_1 y_{t-24} + a_1 y_{t-25} + B_1 y_{t-23} - B_1 \hat{y}_{t-23}}$

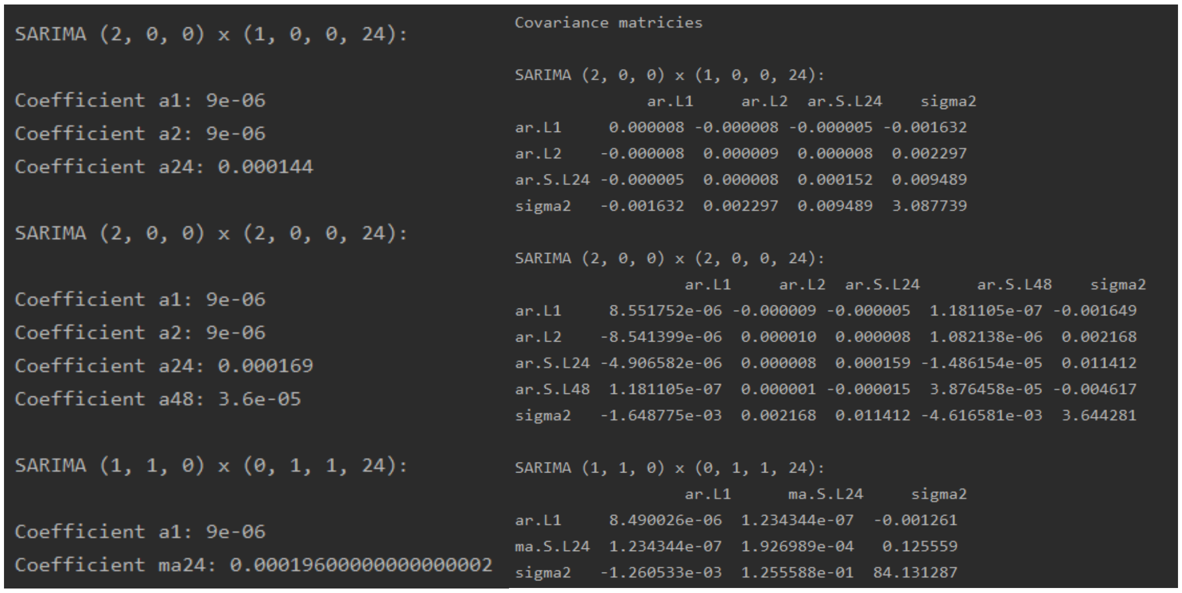
ARM

Fig 29. Handwritten computation of one-step ahead predictions for SARIMA (1, 1, 0) x (0, 1, 1, 24)

- The results of the chi-squared test display a Q value of 30683.1841 and a critical Q value of 11.0704 (alpha = 0.05, 5 degrees of freedom). Because Q > critical Q, we reject the null hypothesis of the test and conclude that the residuals are not white.

Part B: Display the estimated variance of the error and the estimated covariance of the estimated parameters.

The variance of the error of the estimated parameters was calculated by squaring the standard error printed using `model.summary()`. The covariances were found by printing the covariance matrices of the given models. The results can be found below in Figure 30.



```
SARIMA (2, 0, 0) x (1, 0, 0, 24):

Coefficient a1: 9e-06
Coefficient a2: 9e-06
Coefficient a24: 0.000144

SARIMA (2, 0, 0) x (2, 0, 0, 24):

Coefficient a1: 9e-06
Coefficient a2: 9e-06
Coefficient a24: 0.000169
Coefficient a48: 3.6e-05

SARIMA (1, 1, 0) x (0, 1, 1, 24):

Coefficient a1: 9e-06
Coefficient ma24: 0.00019600000000000002
```

```
Covariance matricies

SARIMA (2, 0, 0) x (1, 0, 0, 24):
              ar.L1     ar.L2  ar.S.L24    sigma2
ar.L1     0.000008 -0.000008 -0.000005 -0.001632
ar.L2    -0.000008  0.000009  0.000008  0.002297
ar.S.L24 -0.000005  0.000008  0.000152  0.009489
sigma2   -0.001632  0.002297  0.009489  3.087739

SARIMA (2, 0, 0) x (2, 0, 0, 24):
                  ar.L1     ar.L2  ar.S.L24      ar.S.L48    sigma2
ar.L1     8.551752e-06 -0.000009 -0.000005  1.181105e-07 -0.001649
ar.L2    -8.541399e-06  0.000010  0.000008  1.082138e-06  0.002168
ar.S.L24 -4.906582e-06  0.000008  0.000159 -1.486154e-05  0.011412
ar.S.L48  1.181105e-07  0.000001 -0.000015  3.876458e-05 -0.004617
sigma2   -1.648775e-03  0.002168  0.011412 -4.616581e-03  3.644281

SARIMA (1, 1, 0) x (0, 1, 1, 24):
                  ar.L1      ma.S.L24     sigma2
ar.L1     8.490026e-06  1.234344e-07  -0.001261
ma.S.L24  1.234344e-07  1.926989e-04   0.125559
sigma2   -1.260533e-03  1.255588e-01  84.131287
```

Fig 30. Variance of error (leftmost graphic) and covariance matrices (rightmost graphic) of the parameter estimates

Part C: Is the derived model biased or this is an unbiased estimator?

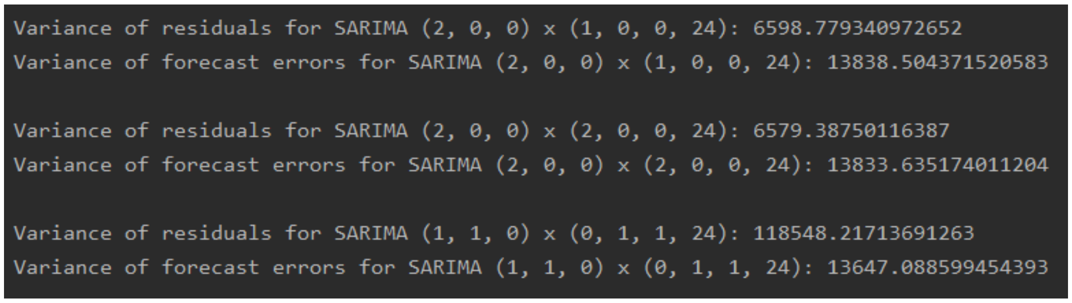
From my understanding, a model is unbiased if the mean of the residuals is non-zero or not very close to zero. Thus, I computed the mean of the residuals for each model. The results are as follows:

- SARIMA (2, 0, 0) x (1, 0, 0, 24): 98.5331
- SARIMA (2, 0, 0) x (2, 0, 0, 24): 98.5136
- SARIMA (1, 1, 0) x (0, 1, 1, 24): 202.1768

The mean of the residuals for all three SARIMA models are large, which suggest that the models are biased estimators. However, I will still keep these models under consideration after plotting the base models and comparing all models created in this project.

Part D: Check the variance of the residual errors versus the variance of the forecast errors.

Because at this step of the project I have not yet chosen the best model and hand-calculated the h-step ahead forecast equation, I am using the .forecast() method to perform h-step ahead forecasts to find forecast errors for the three models. The variance of the residuals and forecast errors can be found below in Figure 31.



```
Variance of residuals for SARIMA (2, 0, 0) x (1, 0, 0, 24): 6598.779340972652
Variance of forecast errors for SARIMA (2, 0, 0) x (1, 0, 0, 24): 13838.504371520583

Variance of residuals for SARIMA (2, 0, 0) x (2, 0, 0, 24): 6579.38750116387
Variance of forecast errors for SARIMA (2, 0, 0) x (2, 0, 0, 24): 13833.635174011204

Variance of residuals for SARIMA (1, 1, 0) x (0, 1, 1, 24): 118548.21713691263
Variance of forecast errors for SARIMA (1, 1, 0) x (0, 1, 1, 24): 13647.088599454393
```

Fig 31. Variance of the residuals and forecast errors

I am taking these values with a grain of salt, since using the forecast method here is likely not as accurate as developing an h-step ahead forecast equation on paper.

Part E: If you find out that the ARIMA or SARIMA model may better represents the dataset, then you can find the model accordingly. You are not constraint only to use of ARMA model. Finding an ARMA model is a minimum requirement and making the model better is always welcomed.

The results of the ACF and PACF plot transformations of my dataset conclude that my data is seasonal ARMA (i.e., SARIMA). I decided to test different SARIMA models for this project. Applying non-seasonal ARMA models would make little sense, given the seasonal component present. Plots of the one-step ahead predictions and h-step ahead forecasts (using the `.forecast()` method) for the three SARIMA models can be found below in Figures 32 - 34.

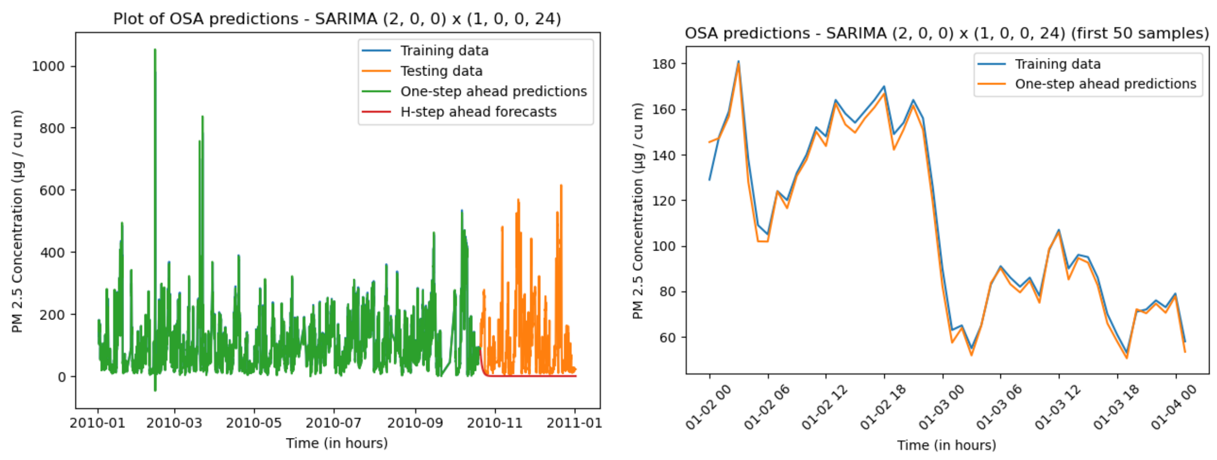


Fig 32. Prediction and forecast plots for SARIMA (2, 0, 0) x (1, 0, 0, 24)

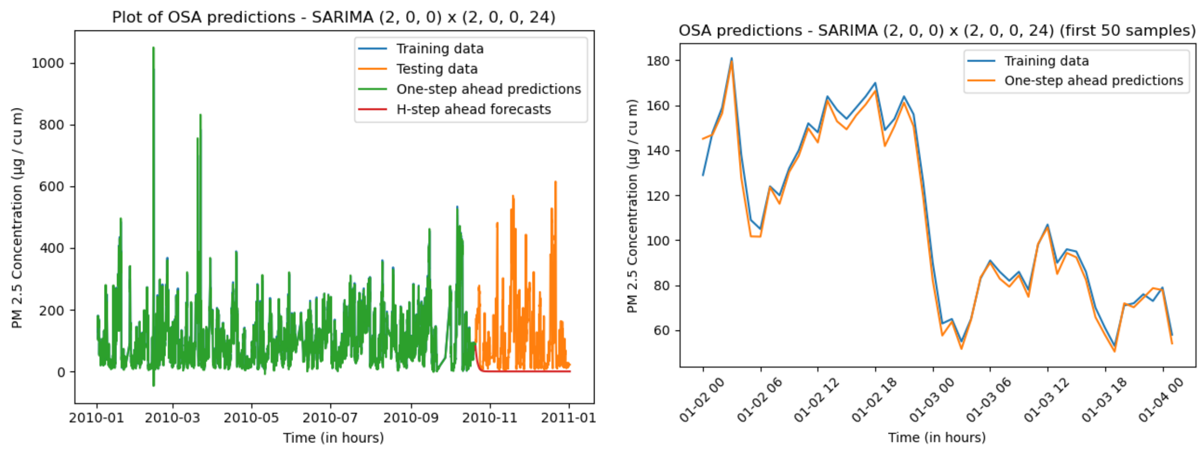


Fig 33. Prediction and forecast plots for SARIMA (2, 0, 0) x (2, 0, 0, 24)

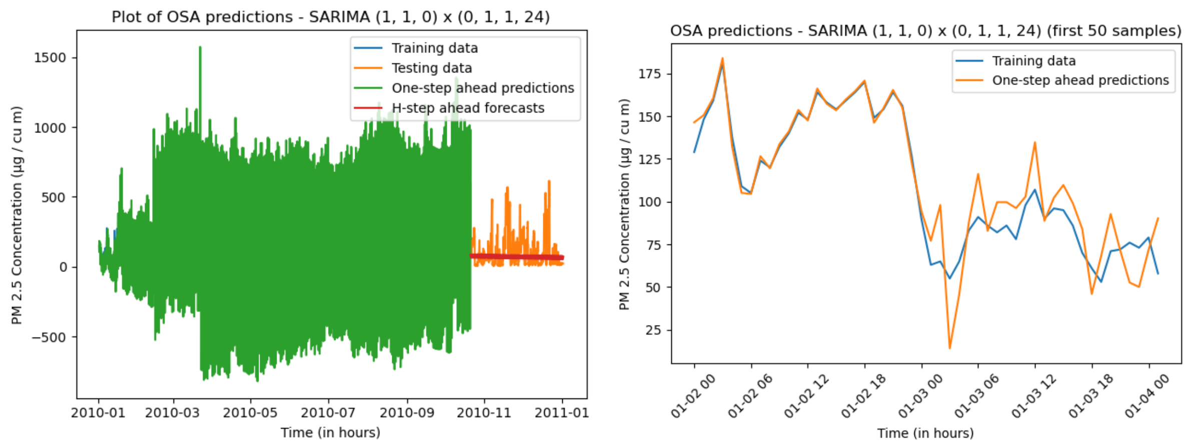
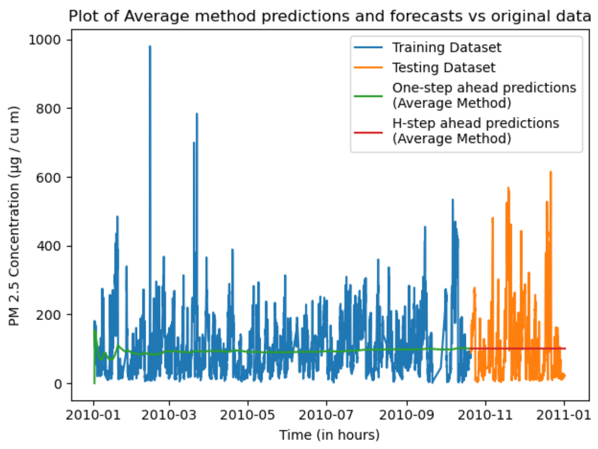


Fig 34. Prediction and forecast plots for SARIMA (1, 1, 0) x (0, 1, 1, 24)

Base Models

The average, naïve, drift, and simple exponential smoothing methods were applied to my dataset. I performed one-step ahead and h-step ahead predictions and plotted the results against the training set, I also calculated the mean of the residuals and compared the variance of the residuals with the variance of the forecast errors for all four models. The results can be found below in Figures 35 - 38.



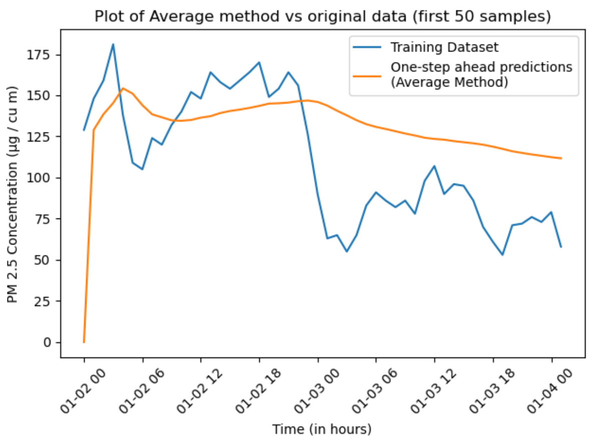
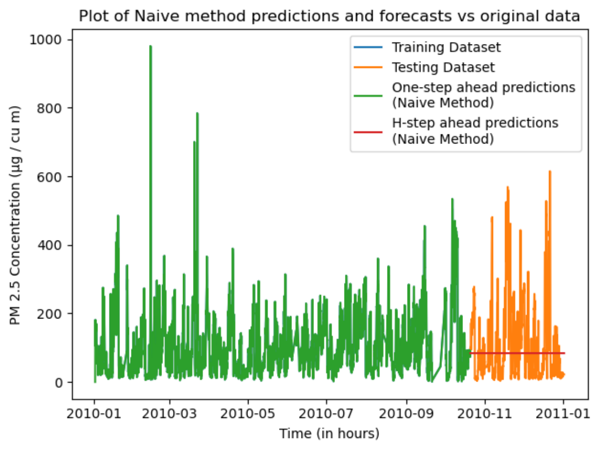


Fig 35. Average method plots

```
The mean of the residuals for the average method are 7.627246147260275
The variance of the residuals for the average method is 6752.329923563669
The variance of the forecast errors for the average method is 13865.27539673175
The Q-value of the residuals is 75420.63493741855
```



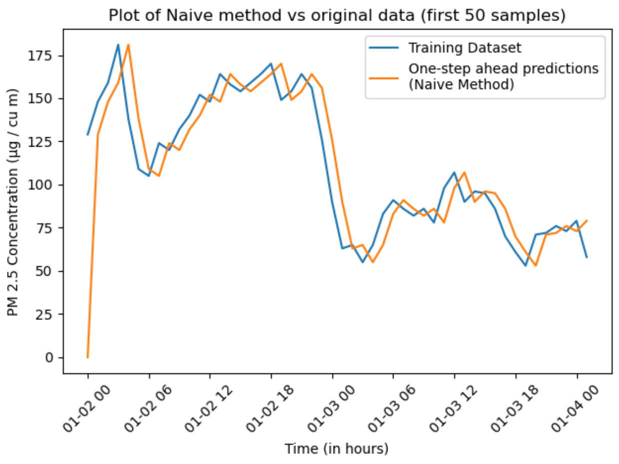


Fig 36. Naïve method plots

```
The mean of the residuals for the naive method are 0.011986301369862997

The variance of the residuals for the naive method is 716.2264165026724

The variance of the forecast errors for the naive method is 13865.27539673175

The Q-value of the residuals is 7463.81999734412
```

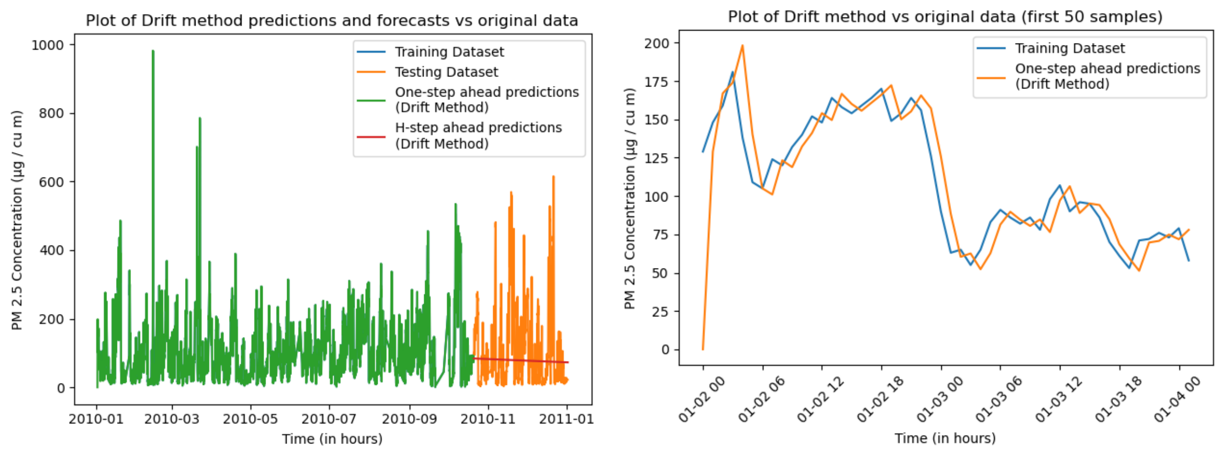


Fig 37. Drift method plots

```
The mean of the residuals for the Drift method are 0.03874486301369907

The variance of the residuals for the Drift method is 717.0149556606644

The variance of the forecast errors for the Drift method is 13846.857565597844

The Q-value of the residuals is 7465.310735143137
```

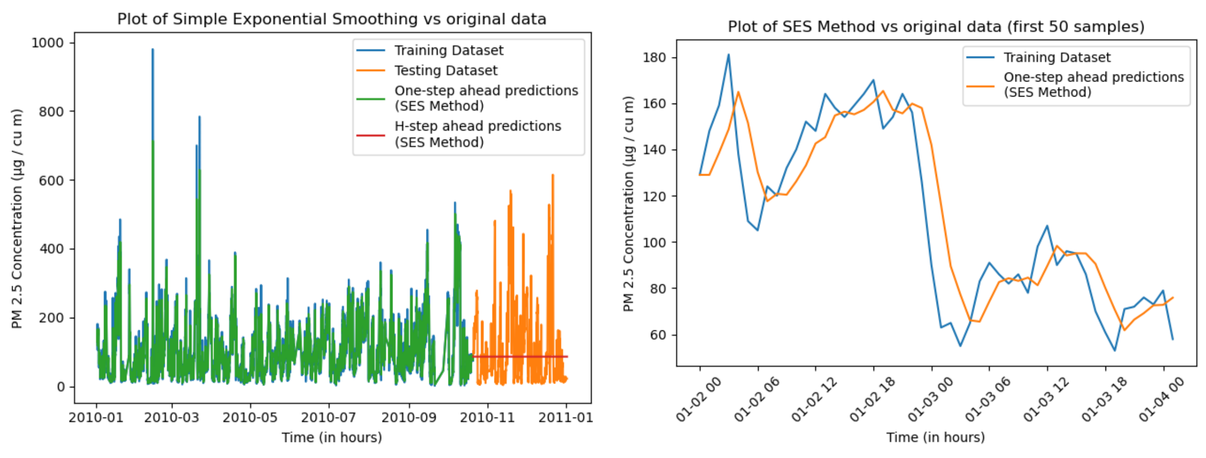


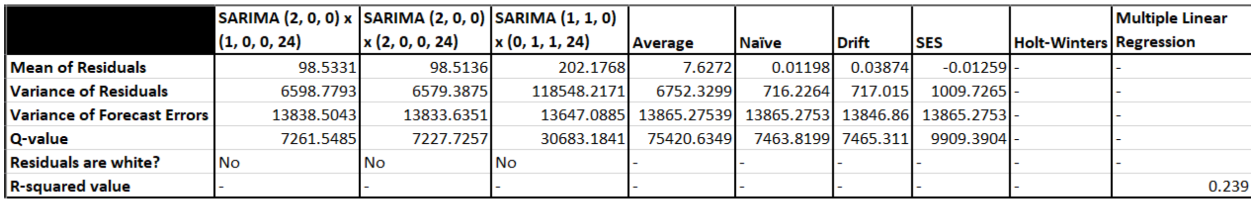
Fig 38. Simple exponential smoothing method plots

```
The mean of the residuals for the SES method are -0.012593892694063955

The variance of the residuals for the SES method is 1009.7265359435473

The variance of the forecast errors for the SES method is 13865.27539673175

The Q-value of the residuals is 9909.39043940882
```

Initial analysis of these plots displays good one-step ahead prediction matchups against the original training data (except for the average method) and terrible h-step ahead forecast matchups against the testing data. The Q-values are all very high, and the variance of the forecast errors are all much higher than the variance of the residuals. These models are likely not the best for this data, but this will be reviewed in the next section.

Final Model Selection

Upon assessing the results of all models, I am confident in moving forward with SARIMA (2, 0, 0) x (1, 0, 0, 24) model as the best model. A table of valuable statistics can be found below in Figure 39.



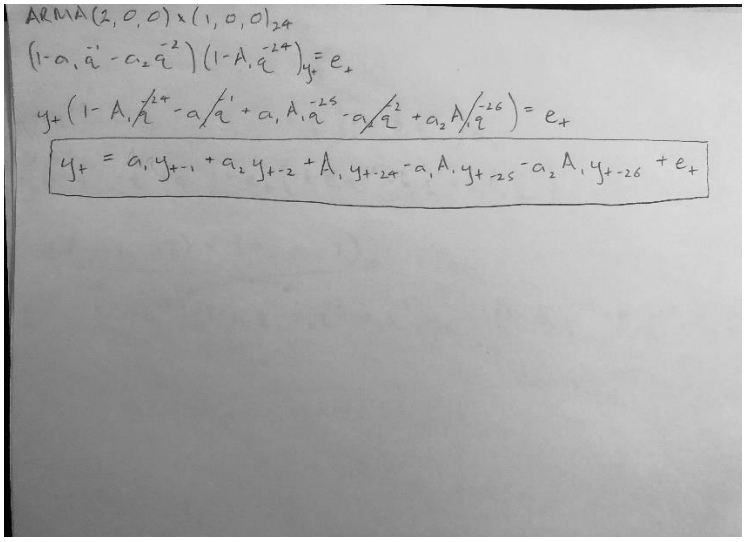
	SARIMA (2, 0, 0) x (1, 0, 0, 24)	SARIMA (2, 0, 0) x (2, 0, 0, 24)	SARIMA (1, 1, 0) x (0, 1, 1, 24)	Average	Naïve	Drift	SES	Holt-Winters	Multiple Linear Regression
Mean of Residuals	98.5331	98.5136	202.1768	7.6272	0.01198	0.03874	-0.01259	-	-
Variance of Residuals	6598.7793	6579.3875	118548.2171	6752.3299	716.2264	717.015	1009.7265	-	-
Variance of Forecast Errors	13838.5043	13833.6351	13647.0885	13865.27539	13865.2753	13846.86	13865.2753	-	-
Q-value	7261.5485	7227.7257	30683.1841	75420.6349	7463.8199	7465.311	9909.3904	-	-
Residuals are white?	No	No	No	-	-	-	-	-	-
R-squared value	-	-	-	-	-	-	-	-	0.239

Fig 39. Table of relevant statistics

Out of all the SARIMA and base models, SARIMA (2, 0, 0) x (1, 0, 0, 24) possessed the lowest Q-value. The variance of the forecasts for all models are very similar, and while the mean of the residuals is not the closest to zero, the plot of one-step ahead predictions displays a very close match to the training set; this is very promising for h-step ahead forecasts. Although the h-step ahead forecasts for SARIMA (2, 0, 0) x (1, 0, 0, 24) possessed a high variance, I am skeptical of the results using the `.forecast()` method; all of the above models performed poorly using the forecast method. For every other model, there existed a defining characteristic that resulted in discarding it. The low r-squared value of multiple linear regression, combined with the poor representation of the training and testing sets in Figures 18 and 19, make me very unconfident in the model to predict future PM 2.5 concentrations. This notion is backed by the correlation matrix in Figure 7; there existed no strong correlations between the dependent variable and any of the features. Additionally, the high values of the AIC and BIC suggest a high SSE value, which creates uncertainty in how accurate the forecasts will be. The plots of Holt-Winters using seasonality periods of 7 and 24 in Figure 14 did not match the testing set well at all. The base models were much harder to rule out. The average model a much smaller mean of residuals than the SARIMA models, but the variance of the forecast errors and the Q-value were way to high to consider viable. The naïve, drift and simple exponential smoothing models were very similar in values; the means of the residuals were very close to zero, which indicates that these models are unbiased estimators. The variance of the forecast errors are similar for naïve, drift and SES, but SES possesses a higher variance in residuals, as well as Q-value. I am tempted to pick between the naïve and drift methods, but the matchup of the one-step ahead predictions and the training set using SARIMA (2, 0, 0) x (1, 0, 0, 24) are nearly perfect. I am also a bit reluctant about my Q-values; I have had issues computing the Q-value since lab 10. I suspect I may be misunderstanding something in regards to computation of the Q-value, since nearly every time I've computed a Q-value in Python the value is extremely large. The fact that professor Jafari said my data is AR/MA seasonal points to the strong possibility of a SARIMA process, yet all of the chi-square tests for my potential SARIMA models failed and concluded that the residuals are not white. By disregarding the likely incorrect Q-value, I am confident that SARIMA (2, 0, 0) x (1, 0, 0, 24) is the best model.

Forecast Function

The hand-computed forecast function can be found below in Figure 40.



ARMA$(2,0,0)\times(1,0,0)_{24}$

$$(1-a_1 q^{-1} - a_2 q^{-2})(1-A_1 q^{-24})y_t = e_t$$

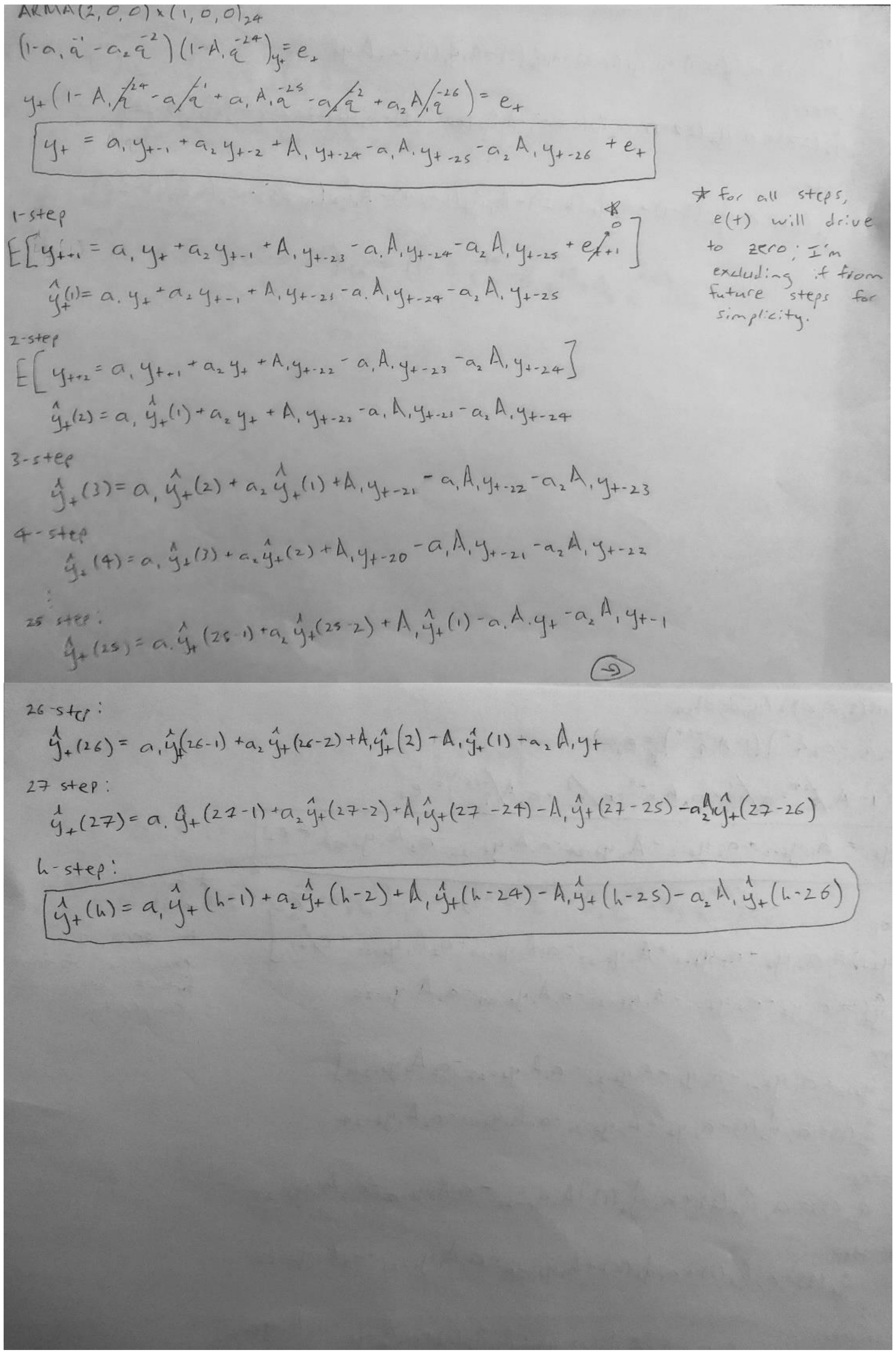
$$y_t(1 - A_1 \cancel{q}^{-24} - a_1 \cancel{q}^{-1} + a_1 A_1 q^{-25} - a_2 \cancel{q}^{-2} + a_2 A_1 \cancel{q}^{-26}) = e_t$$

$$\boxed{y_t = a_1 y_{t-1} + a_2 y_{t-2} + A_1 y_{t-24} - a_1 A_1 y_{t-25} - a_2 A_1 y_{t-26} + e_t}$$

Fig 40. Forecast function for SARIMA (2, 0, 0) x (1, 0, 0, 24)

H-Step Ahead Predictions

The h-step ahead prediction was derived from the forecast function and can be found below in Figure 41.



ARMA(2, 0, 0) x (1, 0, 0)$_{24}$

$$(1-a_1 q^{-1} - a_2 q^{-2})(1-A_1 q^{-24})y_t = e_t$$

$$y_t(1 - A_1 \cancel{q}^{-24} - a_1\cancel{q}^{-1} + a_1 A_1 q^{-25} - a_2\cancel{q}^{-2} + a_2 A_1 \cancel{q}^{-26}) = e_t$$

$$\boxed{y_t = a_1 y_{t-1} + a_2 y_{t-2} + A_1 y_{t-24} - a_1 A_1 y_{t-25} - a_2 A_1 y_{t-26} + e_t}$$

1-step

$$E[y_{t+1} = a_1 y_t + a_2 y_{t-1} + A_1 y_{t-23} - a_1 A_1 y_{t-24} - a_2 A_1 y_{t-25} + \overset{\overset{*}{0}}{\cancel{e_{t+1}}}]$$

$$\hat{y}_t(1) = a_1 y_t + a_2 y_{t-1} + A_1 y_{t-23} - a_1 A_1 y_{t-24} - a_2 A_1 y_{t-25}$$

* for all steps, e(t) will drive to zero; I'm excluding it from future steps for simplicity.

2-step

$$E[y_{t+2} = a_1 y_{t+1} + a_2 y_t + A_1 y_{t-22} - a_1 A_1 y_{t-23} - a_2 A_1 y_{t-24}]$$

$$\hat{y}_t(2) = a_1 \hat{y}_t(1) + a_2 y_t + A_1 y_{t-22} - a_1 A_1 y_{t-23} - a_2 A_1 y_{t-24}$$

3-step

$$\hat{y}_t(3) = a_1 \hat{y}_t(2) + a_2 \hat{y}_t(1) + A_1 y_{t-21} - a_1 A_1 y_{t-22} - a_2 A_1 y_{t-23}$$

4-step

$$\hat{y}_t(4) = a_1 \hat{y}_t(3) + a_2 \hat{y}_t(2) + A_1 y_{t-20} - a_1 A_1 y_{t-21} - a_2 A_1 y_{t-22}$$

$\vdots$

25 step:

$$\hat{y}_t(25) = a_1 \hat{y}_t(25-1) + a_2 \hat{y}_t(25-2) + A_1 \hat{y}_t(1) - a_1 A_1 y_t - a_2 A_1 y_{t-1}$$

26-step:

$$\hat{y}_t(26) = a_1 \hat{y}_t(26-1) + a_2 \hat{y}_t(26-2) + A_1 \hat{y}_t(2) - A_1 \hat{y}_t(1) - a_2 A_1 y_t$$

27 step:

$$\hat{y}_t(27) = a_1 \hat{y}_t(27-1) + a_2 \hat{y}_t(27-2) + A_1 \hat{y}_t(27-24) - A_1 \hat{y}_t(27-25) - a_2 A_1 \hat{y}_t(27-26)$$

h-step:

$$\boxed{\hat{y}_t(h) = a_1 \hat{y}_t(h-1) + a_2 \hat{y}_t(h-2) + A_1 \hat{y}_t(h-24) - A_1 \hat{y}_t(h-25) - a_2 A_1 \hat{y}_t(h-26)}$$

Fig 41. Derivation of the h-step ahead prediction function for SARIMA (2, 0, 0) x (1, 0, 0, 24)

The reason for doing specific steps in the handwritten work is to get a handle on what conditionals I will have to account for when I implement this function into my code. There will be specific checks for 1-step, 2-step, 3-step, 4-step, 5 to 26-step, 27-step, and h-steps after. The result of the h-step ahead predictions against the testing set can be found below in Figure 42.

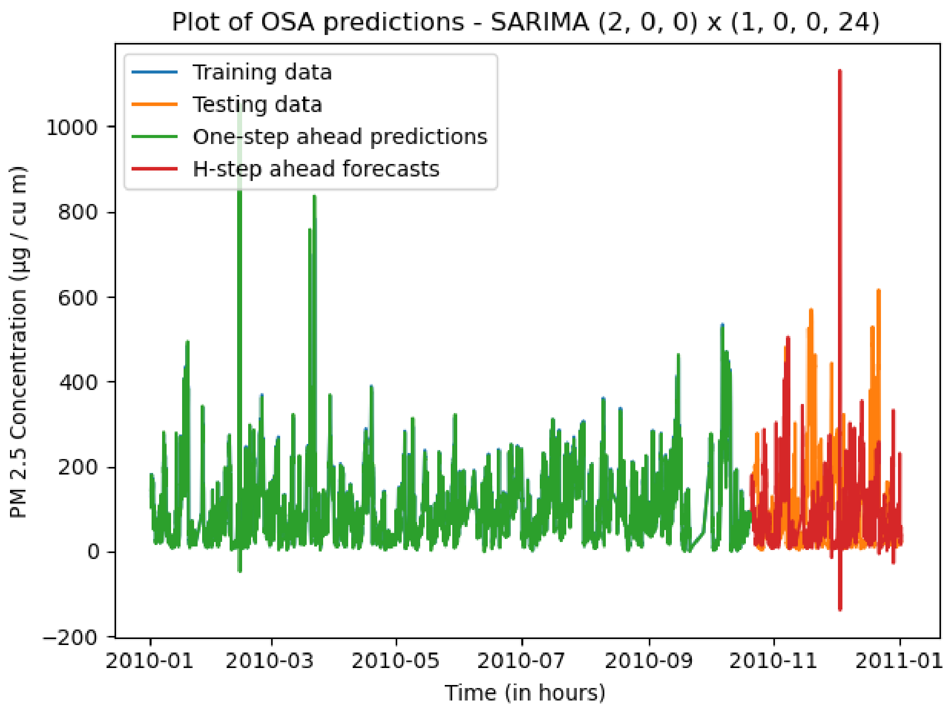


Fig 42. Plots of h-step ahead predictions vs the training set for SARIMA (2, 0, 0) x (1, 0, 0, 24)

As expected, the handwritten h-step ahead forecasts perform much better than the .forecast() method. The h-step ahead forecasts in Figure 42 match the testing set extremely well, and much better than any other model used in this project.

Summary and Conclusions

The final model has several limitations. This model assumes that the data is AR seasonal and possesses an unusually high Q-value that fails the chi-squared test for whiteness of the residuals. The residuals are not white, yet the one-step ahead predictions practically completely cover the training set (you can see only dots of blue in Figure 42). Additionally, the forecast errors of the h-step ahead forecasts went up to

20701.8117, which is much higher than any other model. Although the forecasts seem to match the testing set quite well, I am worried that implementing a h-step ahead forecast for steps after the testing set (i.e. 2011 and 2012 data) will quickly spiral out of control. In the future, I want to try implementing a seasonal naïve model to see if it would improve upon the moderately good results of the naïve model in this project.

References

Hyndman, R.J., & Athanasopoulos, G. (2018) Forecasting: principles and practice, 2nd edition, OTexts: Melbourne, Australia. OTexts.com/fpp2. Accessed on 16 December 2020

Appendix

The below contains all code relevant to this project. The code is split up by questions 5 – 14 and question 17. What you see below are the direct copies of each file.

q_5_description.py

```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
import seaborn as sns
from sklearn.model_selection import train_test_split

# ========================================================================================================================
# Final Project: Description of the dataset
# Benjamin Lee
# Data Science 6450-15: Time Series Modeling & Analysis
# 16 December 2020
# ========================================================================================================================
# Part D: Pre-processing procedures

# Reading in the dataset.
df = pd.read_csv('PRSA_data_2010.1.1-2014.12.31.csv', header=0)

# Combining the time-related columns to a single datetime column.
df['_date_'] = pd.to_datetime(df['year']*10000 + df['month']*100 + df['day'], format='%Y%m%d')
```

```
# Converting the hours to time-deltas and combining with the datetime column.
df['date'] = df['_date_'] + df['hour'].astype('timedelta64[h]')

# Dropping all irrelevant columns; this includes the index column 'No' and the old time-related columns.
df = df.drop(columns=['No', 'year', 'month', 'day', 'hour', '_date_'])

# Setting the index as the newly-created datetime column.
df = df.set_index('date')

# Excluding the first day (24 samples) of data. As per section 12.9 of the textbook:
#
# When missing values cause errors, there are at least two ways to handle the problem. First, we could just take the
# section of data after the last missing value, assuming there is a long enough series of observations to produce
# meaningful forecasts. Alternatively, we could replace the missing values with estimates. The na.interp() function
# is designed for this purpose.
#
# Source: https://otexts.com/fpp2/missing-outliers.html

# The equivalent function in Python, pd.interpolate(), cannot interpolate the first x samples of a dataset because there
# are no entries before them to use for interpolation.
#
# Source: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.interpolate.html
df = df.loc['2010-01-02':'2011-01-01']

# Interpolating using the 'time' method, since this data is based over time.
df = df.interpolate(method='time')

# Label encoding the combined wind direction variable; NE = 0, NW = 1, SW = 2 and cv = 3.
df['cbwd'] = df['cbwd'].astype('category')
df['cbwd'] = df['cbwd'].cat.codes
# 'cv' is a reference to 'calm and variable.'
#
# Source: https://royalsocietypublishing.org/doi/10.1098/rspa.2015.0257

# Initializing the dependent variable.
pm = df.loc[:, 'pm2.5']

# ====================================================================================================================
# Part A: Plotting the dependent variable versus time

plt.figure()

plt.plot(pm, label='PM 2.5')
plt.title('Plot of Dependent Variable (PM 2.5) vs Time')
plt.xlabel('Time (in hours)')
plt.ylabel('PM 2.5 Concentration (µg / cu m)')
plt.legend(loc='upper right')

plt.show()

# ========================================================================================================================
# Part B: Plotting the ACF and PACF of the dependent variable

lags = 100

plt.figure()

plot_acf(pm, lags=lags)

plt.xlabel('Lags')
plt.ylabel('Magnitude')
plt.title('ACF of The Dependent Variable with {} Lags'.format(lags))

plt.show()

plt.figure()

plot_pacf(pm, lags=lags)

plt.xlabel('Lags')
plt.ylabel('Magnitude')
plt.title('PACF of The Dependent Variable with {} Lags'.format(lags))

plt.show()

# ========================================================================================================================
# Part C: Plotting the correlation matrix of the dataset using Pearson's correlation coefficient

plt.figure()

ax = sns.heatmap(df.corr(method='pearson'), annot=True, fmt='.3g', vmin=-1, vmax=1, center=0, cmap='Blues',
                 linecolor='black', square=True, annot_kws={'size': 7})

ax.set_xticklabels(ax.get_xticklabels(), rotation=45, horizontalalignment='right', fontsize=9)
ax.set_yticklabels(ax.get_yticklabels(), rotation=0, horizontalalignment='right', fontsize=9)
```

```
plt.title('HeatMap of Correlation Matrix')

plt.show()

# ========================================================================================================================
# Part E: Splitting the dataset into training and testing

predictors = df.drop(columns='pm2.5')
response = pm.copy(deep=True)

X_train, x_test, Y_train, y_test = train_test_split(predictors, response, test_size=0.2, shuffle=False)

# ========================================================================================================================
# Additional analysis: plotting first month of dependent variable vs time

# NOTE: It is quite difficult to visualize seasonality in the dataset using all of the raw data. The first month of data
# is plotted below to get a better sense of what's going on.

pm_first_month = pm.loc[:'2010-02-01']

plt.figure()

plt.plot(pm_first_month, label='PM 2.5')

plt.title('Plot of Dependent Variable (PM 2.5) vs Time (Jan \'10)')
plt.xlabel('Time (in hours)')
plt.ylabel('PM 2.5 Concentration (µg / cu m)')
plt.legend(loc='upper right')

plt.xticks(rotation=45)

plt.show()

# The seasonal period appears to be roughly 7; the large spikes occur on Jan 2, Jan 8/9, Jan 14, Jan 20, Jan 27 and
# Jan 31.
```

q6_stationarity.py

```
import pandas as pd

import matplotlib.pyplot as plt

from statsmodels.graphics.tsaplots import plot_acf

from statsmodels.graphics.tsaplots import plot_pacf

from statsmodels.tsa.stattools import adfuller

import numpy as np

import statsmodels.api as sm



# ========================================================================================================================

# Final Project: Checking for and/or making the dependent variable stationary

# Benjamin Lee

# Data Science 6450-15: Time Series Modeling & Analysis

# 16 December 2020

# ========================================================================================================================

# Applicable functions




def difference(dataset, interval):

    diff = []

    for i in range(interval, len(dataset)):

        value = dataset[i] - dataset[i - interval]

        diff.append(value)

    return diff




def adf_cal(x):

    """

    :param x: The dataset in question.

    :return: The results of the Augmented Dickey-Fuller Test.
    """
    result = adfuller(x)


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



# =======================================================================================================================
# Pre-processing procedures; see 'q5_description.py' for additional comments

df = pd.read_csv('PRSA_data_2010.1.1-2014.12.31.csv', header=0)

df['_date_'] = pd.to_datetime(df['year']*10000 + df['month']*100 + df['day'], format='%Y%m%d')

df['date'] = df['_date_'] + df['hour'].astype('timedelta64[h]')

df = df.drop(columns=['No', 'year', 'month', 'day', 'hour', '_date_'])

df = df.set_index('date')

df = df.loc['2010-01-02':'2011-01-01']

df = df.interpolate(method='time')

df['cbwd'] = df['cbwd'].astype('category')
df['cbwd'] = df['cbwd'].cat.codes
```

```
pm = df.loc[:, 'pm2.5']



# ======================================================================================================================

# Plot of the dependent variable over time



plt.figure()



plt.plot(pm, label='PM 2.5')



plt.title('Plot of Dependent Variable (PM 2.5) vs Time')

plt.xlabel('Time (in hours)')

plt.ylabel('PM 2.5 Concentration (µg / cu m)')

plt.legend(loc='upper right')



plt.show()



# ======================================================================================================================

# Plot of the dependent variable over time (first month)



pm_first_month = pm.loc[:'2010-02-01']



plt.figure()



plt.plot(pm_first_month, label='PM 2.5')



plt.title('Plot of Dependent Variable (PM 2.5) vs Time (Jan \'10)')

plt.xlabel('Time (in hours)')

plt.ylabel('PM 2.5 Concentration (µg / cu m)')

plt.legend(loc='upper right')
plt.xticks(rotation=45)

plt.show()

print('Results of ADF test for raw data:')

adf_cal(pm)

# Analysis: Fails test; reject the null and conclude that data is stationary, but ACF and PACF say otherwise.

# ========================================================================================================================
# ACF plot of raw data

lags = 80

plt.figure()

plot_acf(pm, lags=lags)

plt.xlabel('Lags')
plt.ylabel('Magnitude')
plt.title('ACF of The Dependent Variable with {} Lags'.format(lags))

plt.show()

# ========================================================================================================================
# PACF plot of raw data

plt.figure()
```

```
plot_pacf(pm, lags=lags)


plt.xlabel('Lags')

plt.ylabel('Magnitude')

plt.title('PACF of The Dependent Variable with {} Lags'.format(lags))


plt.show()


# ========================================================================================================================

# First-order seasonal differencing


pm_seasonal = difference(dataset=pm, interval=24)


# pm_first_difference = difference(dataset=pm_seasonal, interval=1)

#

# acf = sm.tsa.stattools.acf(pm_first_difference, nlags = lags)

# pacf = sm.tsa.stattools.pacf(pm_first_difference, nlags = lags)


# ========================================================================================================================

# ACF plot of first-order seasonal differencing


plt.figure()


plot_acf(pm_seasonal, lags=lags)


plt.xlabel('Lags')

plt.ylabel('Magnitude')

plt.title('ACF of First-Order Seasonal Differencing with {} Lags'.format(lags))


plt.show()
```

```
# ======================================================================================================================

# PACF plot of first-order seasonal differencing


plt.figure()


plot_pacf(pm_seasonal, lags=lags)


plt.xlabel('Lags')

plt.ylabel('Magnitude')

plt.title('PACF of First-Order Seasonal Differencing with {} Lags'.format(lags))


plt.show()


# ======================================================================================================================

# First-order non-seasonal differencing


pm_first_difference = difference(dataset=pm_seasonal, interval=1)


# ======================================================================================================================

# ACF plot of first-order non-seasonal differencing


plt.figure()


plot_acf(pm_first_difference, lags=lags)


plt.xlabel('Lags')

plt.ylabel('Magnitude')

plt.title('ACF of First-Order Non-Seasonal Differencing with {} Lags'.format(lags))
plt.show()



# ========================================================================================================================

# PACF of first-order non-seasonal differencing



plt.figure()



plot_pacf(pm_first_difference, lags=lags)



plt.xlabel('Lags')

plt.ylabel('Magnitude')

plt.title('PACF of First-Order Non-Seasonal Differencing with {} Lags'.format(lags))



plt.show()



# ========================================================================================================================

# ADF testing



print('ADF test of first-order seasonal differencing, followed by first-order non-seasonal differencing:')



adf_cal(pm_first_difference)



# Analysis: still reject the null and conclude data is stationary.
```

q7_time_series_decomposition.py

```
from statsmodels.tsa.stattools import adfuller

import pandas as pd

from statsmodels.tsa.seasonal import STL

import numpy as np

import matplotlib.pyplot as plt



# ========================================================================================================================

# Final Project: Time series decomposition

# Benjamin Lee

# Data Science 6450-15: Time Series Modeling & Analysis

# 16 December 2020

# ========================================================================================================================

# Applicable functions




def adf_cal(x):

    """

    :param x: The dataset in question.

    :return: The results of the Augmented Dickey-Fuller Test.

    """

    result = adfuller(x)



    print('ADF Statistic: %f' % result[0])

    print('p-value: %f' % result[1])

    print('Critical Values:')

    for key, val in result[4].items():

        print('\t%s: %.3f' % (key, val))
```

```
# ========================================================================================================================

# Pre-processing procedures; see 'q5_description.py' for additional comments



df = pd.read_csv('PRSA_data_2010.1.1-2014.12.31.csv', header=0)



df['_date_'] = pd.to_datetime(df['year']*10000 + df['month']*100 + df['day'], format='%Y%m%d')



df['date'] = df['_date_'] + df['hour'].astype('timedelta64[h]')



df = df.drop(columns=['No', 'year', 'month', 'day', 'hour', '_date_'])



df = df.set_index('date')



df = df.loc['2010-01-02':'2011-01-01']



df = df.interpolate(method='time')



df['cbwd'] = df['cbwd'].astype('category')

df['cbwd'] = df['cbwd'].cat.codes



pm = df.loc[:, 'pm2.5']



# ========================================================================================================================

# Approximating the strength of the trend and seasonality



STL = STL(pm)

res = STL.fit()



trend = res.trend

seasonal = res.seasonal
```

```
residuals = res.resid



F_t = np.around(np.maximum(0, 1 - (np.var(np.array(residuals)) / np.var(np.array(trend + residuals)))), 5)

F_s = np.around(np.maximum(0, 1 - (np.var(np.array(residuals)) / np.var(np.array(seasonal + residuals)))), 5)



print('The strength of the trend is', F_t)

print('The strength of the seasonality is', F_s, '\n')



# =======================================================================================================================

# Initial assessment



# The results of the code above prove that the dependent variable is strongly trended and moderately seasonal.



# =======================================================================================================================

# Plotting the de-trended and seasonally adjusted data.



de_trended = pm - trend



seasonally_adjusted = pm - seasonal



plt.figure()



plt.plot(de_trended, label='De-Trended Dataset')

plt.plot(seasonally_adjusted, label='Seasonally-Adjusted Dataset')

plt.plot(pm, label='Original Data')



plt.xlabel('Time (in hours)')

plt.ylabel('PM 2.5 Concentration (µg/cu m)')

plt.title('Plot of De-trended and Seasonally-Adjusted Data')
plt.legend(loc='upper right', prop={'size': 8})


plt.xticks(rotation=45)


plt.show()


# ========================================================================================================================

# Assessment of plot


# The plot above does not provide significant insight into how the de-trended and seasonally-adjusted data compares to

# the raw data. Thus, the first month of data will be plotted to assess a smaller sample size.


# ========================================================================================================================

# Plotting the first month of de-trended and seasonally adjusted data.


pm_first_month = pm.loc[:'2010-02-01']


de_trended_first_month = de_trended.loc[:'2010-02-01']


seasonally_adjusted_first_month = seasonally_adjusted.loc[:'2010-02-01']


plt.figure()


plt.plot(de_trended_first_month, label='De-Trended Dataset')

plt.plot(seasonally_adjusted_first_month, label='Seasonally-Adjusted Dataset')

plt.plot(pm_first_month, label='Original Data')


plt.xlabel('Time (in hours)')

plt.ylabel('PM 2.5 Concentration (µg/cu m)')

plt.title('Plot of De-trended and Seasonally-Adjusted Data (Jan \'10)')
plt.legend(loc='upper right', prop={'size': 8})

plt.xticks(rotation=45)

plt.show()

# ========================================================================================================================
# Assessment of plot of first month

# The plot of de-trended and seasonally-adjusted data displays a near-perfect overlay of seasonally-adjusted data and
# the original dataset. The de-trended data lies below both the seasonally-adjusted and original dataset curves. These
# results suggest that the seasonal adjustments made to the dependent variable do not incite significant changes; this
# aligns with the results of the strengths of trend and seasonality. ADF tests for the de-trended data are run below.

# ========================================================================================================================
# Augmented Dickey-Fuller test

print('\nDe-trended data:\n')

adf_cal(de_trended)

print('\nDe-trended data (first month):\n')

adf_cal(de_trended_first_month)

print('\nSeasonally-adjusted data:\n')

adf_cal(seasonally_adjusted)
```

```
print('\nSeasonally-adjusted data (first month):\n')


adf_cal(seasonally_adjusted_first_month)


# ========================================================================================================================

# Assessment of ADF tests


# The ADF test for the raw de-trended data displays a large negative ADF statistic of -23.110642 and a p-value of

# 0.000000, while the ADF test for the first month of de-trended data displays a slightly smaller ADF statistic of

# -9.174005 and a p-value of 0.000000. The values of the ADF statistics and p-values suggest that the null hypothesis

# should be rejected and that in both cases, the dataset is stationary.
```

q8_holt_winters.py

```
import pandas as pd

from sklearn.model_selection import train_test_split

import statsmodels.tsa.holtwinters as ets

import matplotlib.pyplot as plt



# ========================================================================================================================

# Final Project: Holt-Winters method

# Benjamin Lee

# Data Science 6450-15: Time Series Modeling & Analysis

# 16 December 2020

# ========================================================================================================================

# Pre-processing procedures; see 'q5_description.py' for additional comments



df = pd.read_csv('PRSA_data_2010.1.1-2014.12.31.csv', header=0)



df['_date_'] = pd.to_datetime(df['year']*10000 + df['month']*100 + df['day'], format='%Y%m%d')



df['date'] = df['_date_'] + df['hour'].astype('timedelta64[h]')



df = df.drop(columns=['No', 'year', 'month', 'day', 'hour', '_date_'])



df = df.set_index('date')



df = df.loc['2010-01-02':'2011-01-01']



df = df.interpolate(method='time')



df['cbwd'] = df['cbwd'].astype('category')

df['cbwd'] = df['cbwd'].cat.codes
```

```
pm = df.loc[:, 'pm2.5']


predictors = df.drop(columns='pm2.5')

response = pm.copy(deep=True)


X_train, x_test, Y_train, y_test = train_test_split(predictors, response, test_size=0.2, shuffle=False)


# ========================================================================================================================

# Holt-Winters Method


holt_winters_training = ets.ExponentialSmoothing(Y_train, trend='mul', damped_trend=True, seasonal='add',

                                                 seasonal_periods=7, freq='H').fit()


holt_winters_forecast = holt_winters_training.forecast(steps=len(y_test))

holt_winters_forecast = pd.DataFrame(holt_winters_forecast).set_index(y_test.index)


fig, ax = plt.subplots()


ax.plot(Y_train, label='Training Data')

ax.plot(y_test, label='Testing Data')

ax.plot(holt_winters_forecast, label='Holt-Winters Method')


plt.title('Plot of Holt-Winters Method vs Training & Testing Data')

plt.xlabel('Time (in hours)')

plt.ylabel('PM 2.5 Concentration (µg/cu m)')

plt.legend(loc='upper right')


plt.show()
```

```
# ========================================================================================================================

# Analysis of plot



# After testing Holt-Winters method with a variety of combinations for the trend and seasonal values, the best forecast

# resulted in setting trend to multiplicative and seasonal to additive with a seasonal period of 7 (see

# 'q5_description.py' for reasoning behind seasonal period). I also tried plotting Holt's Linear Trend, but the

# forecasts for both options of trend were very weak.



# ========================================================================================================================

# Holt's Linear Trend using multiplicative trend



holt_linear_training = ets.ExponentialSmoothing(Y_train, trend='mul', damped_trend=True, seasonal=None, freq='H').fit()



holt_linear_forecast = holt_linear_training.forecast(steps=len(y_test))

holt_linear_forecast = pd.DataFrame(holt_linear_forecast).set_index(y_test.index)



fig, ax = plt.subplots()



ax.plot(Y_train, label='Training Data')

ax.plot(y_test, label='Testing Data')

ax.plot(holt_linear_forecast, label='Holt\'s Linear Trend Method')



plt.title('Plot of Holt\'s Linear Trend vs Training & Testing Data')

plt.xlabel('Time (in hours)')

plt.ylabel('PM 2.5 Concentration (µg/cu m)')

plt.legend(loc='upper right')



plt.show()



# ========================================================================================================================
```

q9_feature_selection.py

```
import pandas as pd

import statsmodels.api as sm

from sklearn.model_selection import train_test_split



# ========================================================================================================================

# Final Project: Feature selection

# Benjamin Lee

# Data Science 6450-15: Time Series Modeling & Analysis

# 16 December 2020

# ========================================================================================================================

# Pre-processing procedures; see 'q5_description.py' for additional comments



df = pd.read_csv('PRSA_data_2010.1.1-2014.12.31.csv', header=0)



df['_date_'] = pd.to_datetime(df['year']*10000 + df['month']*100 + df['day'], format='%Y%m%d')



df['date'] = df['_date_'] + df['hour'].astype('timedelta64[h]')



df = df.drop(columns=['No', 'year', 'month', 'day', 'hour', '_date_'])



df = df.set_index('date')



df = df.loc['2010-01-02':'2011-01-01']



df = df.interpolate(method='time')



df['cbwd'] = df['cbwd'].astype('category')

df['cbwd'] = df['cbwd'].cat.codes
```

```
pm = df.loc[:, 'pm2.5']


# Separating the data into predictor and response; during office hours, it was recommended to remove the encoded

# categorical variable from feature selection.

predictors = df.drop(columns=['pm2.5', 'cbwd'])


predictors = sm.add_constant(predictors)


response = pm.copy(deep=True)


# Splitting the predictor and response into training and testing sets.

X_train, x_test, Y_train, y_test = train_test_split(predictors, response, shuffle=False, test_size=0.2)


# ========================================================================================================================

# Forward stepwise regression


X_train_forward = X_train.copy(deep=True)


for val in range(len(X_train_forward.columns)):

    forward_model = sm.OLS(Y_train, X_train_forward[['const', X_train_forward.columns[val]]]).fit()

    # print(forward_model.summary())


# DEWP:     t-val = 27.135  |   p-val = 0.000

# TEMP:     t-val = 9.494   |   p-val = 0.000

# PRES:     t-val = -21.489 |   p-val = 0.000

# Iws:      t-val = -20.924 |   p-val = 0.000

# Is:       t-val = -2.033  |   p-val = 0.042

# Ir:       t-val = -4.569  |   p-val = 0.000


# Between the variables with the smallest p-val, dew point has the largest t-val and is put into the basket.
```

```
for val in range(len(X_train_forward.columns)):

    forward_model = sm.OLS(Y_train, X_train_forward[['const', 'DEWP', X_train_forward.columns[val]]]).fit()

    # print(forward_model.summary())


# TEMP:     t-val = -26.454 |   p-val = 0.000

# PRES:     t-val = -1.570  |   p-val = 0.117

# Iws:      t-val = -14.649 |   p-val = 0.000

# Is:       t-val = -0.281  |   p-val = 0.779

# Ir:       t-val = -7.863  |   p-val = 0.000


# Between the variables with the smallest p-val, cumulative hours of rain has the largest t-val and is put into the

# basket.


for val in range(len(X_train_forward.columns)):

    forward_model = sm.OLS(Y_train, X_train_forward[['const', 'DEWP', 'Ir', X_train_forward.columns[val]]]).fit()

    # print(forward_model.summary())


# TEMP:     t-val = -28.078 |   p-val = 0.000

# PRES:     t-val = -0.983  |   p-val = 0.325

# Iws:      t-val = -14.779 |   p-val = 0.000

# Is:       t-val = -0.327  |   p-val = 0.744


# Between the variables with the smallest p-val, cumulative wind speed has the largest t-val and is put into the basket.


for val in range(len(X_train_forward.columns)):

    forward_model = sm.OLS(Y_train, X_train_forward[['const', 'DEWP', 'Ir', 'Iws', X_train_forward.columns[val]]]).fit()

    # print(forward_model.summary())


# TEMP:     t-val = -25.512 |   p-val = 0.000
# PRES:     t-val = -2.814  |   p-val = 0.005

# Is:       t-val = 0.103   |   p-val = 0.918



# Temperature possessed the smallest p-value below the chosen threshold (0.005) and is put into the basket.



for val in range(len(X_train_forward.columns)):

    forward_model = sm.OLS(Y_train, X_train_forward[['const', 'DEWP', 'Ir', 'Iws', 'TEMP', X_train_forward.columns[val]]]).fit()

    # print(forward_model.summary())



# PRES:     t-val = -17.240 |   p-val = 0.000

# Is:       t-val = -9.540  |   p-val = 0.000



# Between the variables with the smallest p-val, cumulative hours of snow has the largest t-val and is put into the

# basket.



for val in range(len(X_train_forward.columns)):

    forward_model = sm.OLS(Y_train, X_train_forward[['const', 'DEWP', 'Ir', 'Iws', 'TEMP', 'Is',
X_train_forward.columns[val]]]).fit()

    # print(forward_model.summary())



# PRES:     t-val = -17.390 |   p-val = 0.000



# The p-value for pressure is below the chosen threshold (0.05) and is put into the basket.



forward_model = sm.OLS(Y_train, X_train_forward[['const', 'DEWP', 'Ir', 'Iws', 'TEMP', 'Is', 'PRES']]).fit()

print(forward_model.summary())



# ========================================================================================================================

# Analysis of forward stepwise regression



# The results of forward stepwise regression conclude that all features of the dataset are significant. At each step of
# the process, the feature included in the model possessed a p-value below the chosen threshold (0.05). The OLS

# regression results display a R-squared value of 0.239 and an adjusted R-squared value of 0.238, which are very low.

# The results of the r-squared values suggest that only 23.8%/23.9% of the dependent variable (PM 2.5 concentration) can

# be reasonably explained by an independent variable (the features included in forward stepwise regression). This aligns

# with the results of the correlation matrix produced in 'q5_description.py,' in which none of the features possessed a

# strong correlation with the dependent variable (none above ~ 0.3).



# ========================================================================================================================

# Backward stepwise regression



X_train_backward = X_train.copy(deep=True)



backward_model = sm.OLS(Y_train, X_train_backward).fit()

print(backward_model.summary())



# The p-values of all features are below the chosen threshold (0.05); none of the features should be removed.



# ========================================================================================================================

# Analysis of backward stepwise regression



# Because none of the features ended up being removed from forward stepwise regression, the results of backward stepwise

# regression are the same, save for the order of the features in the printout.
```

q10_multiple_linear_regression.py

```
import pandas as pd

import statsmodels.api as sm

from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt

from sklearn.metrics import mean_squared_error

import numpy as np

from statsmodels.graphics.tsaplots import plot_acf



# =====================================================================================================================

# Final Project: Multiple linear regression

# Benjamin Lee

# Data Science 6450-15: Time Series Modeling & Analysis

# 16 December 2020

# =====================================================================================================================

# Applicable functions




def acf_function(dataset, _lags_):

    """

    :param dataset: The data used in calculating the ACF values.

    :param _lags_: The number of lags specified for calculating the ACF values.

    :return: The ACF values associated with the given dataset at a specified lag value.

    # Values for t are adjusted to account for the indexing differences between Python and the handwritten ACF.

    """

    t = _lags_ + 1



    max_t = len(dataset)



    dataset_mean = np.mean(dataset)
    # Numerator

    numerator = 0

    while t < max_t + 1:

        numerator += (dataset[t - 1] - dataset_mean) * (dataset[t - _lags_ - 1] - dataset_mean)

        t += 1



    t = 1



    # Denominator

    denominator = 0

    while t < max_t + 1:

        denominator += (dataset[t - 1] - dataset_mean) ** 2

        t += 1



    acf_value = numerator / denominator



    return acf_value





def one_sided_acf_compiler(data, _lags_):

    """

    :param data: The data used in calculating the ACF values.

    :param _lags_: The number of lags specified for calculating the ACF values.

    :return: A Numpy array containing the two-sided experimental ACF values.

    """

    acf_values = []



    for k in range(_lags_ + 1):

        acf_values.append(acf_function(data, k))
    acf_values = np.asarray(acf_values)

    return acf_values


# ======================================================================================================================
# Pre-processing procedures; see 'q5_description.py' for additional comments

df = pd.read_csv('PRSA_data_2010.1.1-2014.12.31.csv', header=0)

df['_date_'] = pd.to_datetime(df['year']*10000 + df['month']*100 + df['day'], format='%Y%m%d')

df['date'] = df['_date_'] + df['hour'].astype('timedelta64[h]')

df = df.drop(columns=['No', 'year', 'month', 'day', 'hour', '_date_'])

df = df.set_index('date')

df = df.loc['2010-01-02':'2011-01-01']

df = df.interpolate(method='time')

df['cbwd'] = df['cbwd'].astype('category')
df['cbwd'] = df['cbwd'].cat.codes

pm = df.loc[:, 'pm2.5']

# Separating the data into predictor and response; during office hours, it was recommended to remove the encoded
# categorical variable from feature selection.
```

```
predictors = df.drop(columns=['pm2.5', 'cbwd'])


predictors = sm.add_constant(predictors)


response = pm.copy(deep=True)


# Splitting the predictor and response into training and testing sets.

X_train, x_test, Y_train, y_test = train_test_split(predictors, response, shuffle=False, test_size=0.2)


# ========================================================================================================================

# Analysis of 'q9_feature_selection.py'


# In 'q9_feature_selection.py,' forward and backward stepwise regression resulted in selecting all features for the

# multiple linear regression model.


# ========================================================================================================================

# Part A: Creating the model and performing one-step ahead prediction with comparison against the test set


# NOTE: I am a bit confused by the prompt of this question. As I've come to understand from previous assignments,

# one-step ahead predictions are performed on and compared with the training set. Additionally, in part D the ACF

# values computed are for the residuals, which are based on the difference between one-step ahead predictions and the

# original training set values. Thus, I believe the prompt should say "compare the performance versus the test set."


model = sm.OLS(Y_train, X_train).fit()


osa_predictions = model.predict(X_train)


figure, axes = plt.subplots()


axes.plot(Y_train, label='Training Dataset')
axes.plot(y_test, label='Testing Dataset')

axes.plot(osa_predictions, label='One-step ahead predictions')


plt.xlabel('Observations')

plt.ylabel('PM 2.5 concentration (µg/cu m)')

plt.title('Multiple Linear Regression Model: One-Step Ahead Predictions')

plt.legend(loc='upper right')


plt.show()


# Analysis: The one-step ahead predictions cover the bulk of the training set, but values above a PM 2.5 concentration

# of ~ 200 are left uncovered. These predictions are okay, but do not cover cases where the PM 2.5 concentration spikes.

# Out of curiosity, multi-step ahead forecasts were constructed and plotted below. The multi-step ahead forecasts were

# compared with the testing set.


model = sm.OLS(Y_train, X_train).fit()


msa_predictions = model.predict(x_test)


figure, axes = plt.subplots()


axes.plot(Y_train, label='Training Dataset')

axes.plot(y_test, label='Testing Dataset')

axes.plot(msa_predictions, label='Multi-step ahead predictions')


plt.xlabel('Observations')

plt.ylabel('PM 2.5 concentration (µg/cu m)')

plt.title('Multiple Linear Regression Model: Multi-Step Ahead Predictions')

plt.legend(loc='upper right', prop={'size': 9})
plt.show()


# Analysis: The results of multi-step ahead predictions are similar to the one-step ahead predictions. The multi-step

# ahead predictions cover the bulk of the testing set, but original data spikes above ~ 200 are left uncovered. The same

# process is repeated with the first and last months of data to gain a better handle on how the one-step ahead

# predictions and multi-step ahead forecasts compare to the training and testing sets, respectively.


df_first_month = df.loc[:'2010-02-01']

pm_first_month = pm.loc[:'2010-02-01']


predictors_first_month = df_first_month.drop(columns='pm2.5')


predictors_first_month = sm.add_constant(predictors_first_month)


response_first_month = pm_first_month.copy(deep=True)


X_train_first_month, x_test_first_month, Y_train_first_month, y_test_first_month = train_test_split(

    predictors_first_month, response_first_month, shuffle=False, test_size=0.2)


# Model building and plotting: one-step ahead predictions


model_first_month = sm.OLS(Y_train_first_month, X_train_first_month).fit()


osa_predictions_first_month = model_first_month.predict(X_train_first_month)


figure, axes = plt.subplots()


axes.plot(Y_train_first_month, label='Training Dataset')

axes.plot(y_test_first_month, label='Testing Dataset')

axes.plot(osa_predictions_first_month, label='One-step ahead predictions')
```

```
plt.xlabel('Observations')

plt.ylabel('PM 2.5 concentration (µg/cu m)')

plt.title('Multiple Linear Regression Model: One-Step Predictions (Jan \'10)')

plt.legend(loc='upper right')


plt.xticks(rotation=45)


plt.show()


# Model building and plotting: multi-step ahead predictions


model_first_month_multi = sm.OLS(Y_train_first_month, X_train_first_month).fit()


msa_predictions_first_month = model_first_month_multi.predict(x_test_first_month)


figure, axes = plt.subplots()


axes.plot(Y_train_first_month, label='Training Dataset')

axes.plot(y_test_first_month, label='Testing Dataset')

axes.plot(msa_predictions_first_month, label='Multi-step ahead predictions')


plt.xlabel('Observations')

plt.ylabel('PM 2.5 concentration (µg/cu m)')

plt.title('Multiple Linear Regression Model: Multi-Step Predictions (Jan \'10)')

plt.legend(loc='upper right', prop={'size': 9})


plt.xticks(rotation=45)


plt.show()
```

```
# =======================================================================================================================

# Part B: Hypothesis tests


print(model.summary())


# The p-value of the F-statistic is 0.00. Because the p-value of the F-test is below the chosen threshold (0.05), we

# reject the null hypothesis and conclude that the model provides a better fit than the intercept-only model. The

# results of the t-tests for the coefficients of the predictors all display a p-value of 0.000. Because the p-value of

# the t-tests are below the chosen threshold of 5% (0.05), we reject the null hypothesis for all cases and conclude that

# all of the coefficients are not equal to zero.


# =======================================================================================================================

# Part C: AIC, BIC, RMSE, R-squared and adjusted R-squared values


# The summary printed in part B displays all relevant statistics, save for the RSME. The AIC and BIC are very large

# (79,840 and 79,890, respectively), which are not very good; because the AIC and BIC values are directly proportional

# to the SSE, this means the SSE is also very high. If the SSE is very high, then there is a large degree of error in

# this multiple linear regression model. Similarly, the R-squared and adjusted R-squared values are very small (0.239

# and 0.238, respectively). Because the R-squared and adjusted R-squared values are so small, only a small proportion

# (23.8% for R-squared and 23.9% for adjusted R-squared) of the dependent variable (PM 2.5 concentration) can be

# reasonably explained by an independent variable of the model.


rs_me_osa = np.sqrt(mean_squared_error(Y_train, osa_predictions))


print('\nThe root mean squared error for one-step ahead predictions is', rs_me_osa)


rs_me_msa = np.sqrt(mean_squared_error(y_test, msa_predictions))


print('The root mean squared error for multi-step ahead predictions is', rs_me_msa)
```

```
# The root mean squared errors of the one-step and multi-step ahead predictions are relatively large (71.96915 and

# 96.74490, respectively). The large values of the root mean squared error suggest that there exists a large deviation

# of the residuals and forecast errors. This means that the data is not very concentrated around the line of best fit

# for the residuals and forecast errors.


# ========================================================================================================================

# Part D: ACF of the residuals


residuals = []


for val in range(len(osa_predictions)):

    residuals.append(Y_train[val] - osa_predictions[val])


lags = 50


plt.figure()


plot_acf(residuals, lags=lags)


plt.xlabel('Lags')

plt.ylabel('Magnitude')

plt.title('ACF of The Residuals with {} Lags'.format(lags))


plt.show()


# Analysis: The patterns in the ACF plot of the residuals follows the pattern of the ACF plot for the raw dataset. There

# exist slight spikes or humps in values roughly every 24 lags. This is indicative of a seasonal pattern of 24, which

# will be implemented in SARIMA modelling later on.
# ========================================================================================================================

# Part E: Q-value of residuals


residuals = np.asarray(residuals)


acf_values = one_sided_acf_compiler(residuals, lags)


summation = 0


# After receiving my grade for Homework 7, the TA noted that "Q-value have to calculate start from 1." I interpreted

# this to mean the Q-value should be calculated from lag 1 onwards; this should not include lag 0. Therefore,

# acf_values[1:] accounts for this issue.

for val in acf_values[1:]:

    summation += val ** 2


Q = len(Y_train) * np.sum(np.square(acf_values[1:]))


print('\nThe Q value of the residuals is', Q)


# The ACF plot of the residuals denotes a lack of swift decay and displays all ACF values above the insignificance

# region (lags = 50). Because the ACF values are not close to zero, the Q value is proportionately large.


# ========================================================================================================================

# Part F: Variance and mean of the residuals


res_mean = np.mean(residuals)


print('\nThe mean of the residuals is', res_mean)


res_var = np.var(residuals)
```

```
print('The variance of the residuals is', res_var)


# The mean of the residuals is near-zero, and the variance of the residuals is extremely large. The variance follows the

# pattern of the RSME, whose large value suggested a proportionately large deviation from the line of best fit for the

# residuals.


# ========================================================================================================================

# Overall analysis of multiple linear regression


# Overall, multiple linear regression is a very bad model for this dataset. The R-squared value and adjusted R-squared

# values are very low, and the AIC/BIC values are very large. This means that the change in PM 2.5 concentration cannot

# be accurately explained by changes in the other features of the dataset, and the SSE of this data is very large.
```

q11_sarima.py

```
import pandas as pd

import matplotlib.pyplot as plt

from statsmodels.graphics.tsaplots import plot_acf

from statsmodels.graphics.tsaplots import plot_pacf

from statsmodels.tsa.stattools import adfuller

import numpy as np

import statsmodels.api as sm

from sklearn.model_selection import train_test_split

from statsmodels.tsa.statespace.sarimax import SARIMAX


# ========================================================================================================================

# Final Project: Order estimation of SARIMA

# Benjamin Lee

# Data Science 6450-15: Time Series Modeling & Analysis

# 16 December 2020

# ========================================================================================================================

# Applicable functions



def difference(dataset, interval):

    diff = []

    for i in range(interval, len(dataset)):

        value = dataset[i] - dataset[i - interval]

        diff.append(value)

    return diff



def adf_cal(x):

    """
    :param x: The dataset in question.

    :return: The results of the Augmented Dickey-Fuller Test.

    """

    result = adfuller(x)



    print('ADF Statistic: %f' % result[0])

    print('p-value: %f' % result[1])

    print('Critical Values:')

    for key, val in result[4].items():

        print('\t%s: %.3f' % (key, val))




# =======================================================================================================================

# Pre-processing procedures; see 'q5_description.py' for additional comments



df = pd.read_csv('PRSA_data_2010.1.1-2014.12.31.csv', header=0)



df['_date_'] = pd.to_datetime(df['year']*10000 + df['month']*100 + df['day'], format='%Y%m%d')



df['date'] = df['_date_'] + df['hour'].astype('timedelta64[h]')



df = df.drop(columns=['No', 'year', 'month', 'day', 'hour', '_date_'])



df = df.set_index('date')



df = df.loc['2010-01-02':'2011-01-01']



df = df.interpolate(method='time')



df['cbwd'] = df['cbwd'].astype('category')
```

```
df['cbwd'] = df['cbwd'].cat.codes


pm = df.loc[:, 'pm2.5']


predictors = df.drop(columns=['pm2.5', 'cbwd'])


predictors = sm.add_constant(predictors)


response = pm.copy(deep=True)


X_train, x_test, Y_train, y_test = train_test_split(predictors, response, shuffle=False, test_size=0.2)


# ========================================================================================================================

# First-order seasonal differencing


lags = 80


pm_seasonal = difference(dataset=pm, interval=24)


# pm_first_difference = difference(dataset=pm_seasonal, interval=1)

#

# acf = sm.tsa.stattools.acf(pm_first_difference, nlags = lags)

# pacf = sm.tsa.stattools.pacf(pm_first_difference, nlags = lags)


# ========================================================================================================================

# ACF plot of first-order seasonal differencing


plt.figure()


plot_acf(pm_seasonal, lags=lags)
```

```
plt.xlabel('Lags')

plt.ylabel('Magnitude')

plt.title('ACF of First-Order Seasonal Differencing with {} Lags'.format(lags))


plt.show()


# ========================================================================================================================

# PACF plot of first-order seasonal differencing


plt.figure()


plot_pacf(pm_seasonal, lags=lags)


plt.xlabel('Lags')

plt.ylabel('Magnitude')

plt.title('PACF of First-Order Seasonal Differencing with {} Lags'.format(lags))


plt.show()


# ========================================================================================================================

# First-order non-seasonal differencing


pm_first_difference = difference(dataset=pm_seasonal, interval=1)


# ========================================================================================================================

# ACF plot of first-order non-seasonal differencing


plt.figure()
```

```
plot_acf(pm_first_difference, lags=lags)



plt.xlabel('Lags')

plt.ylabel('Magnitude')

plt.title('ACF of First-Order Non-Seasonal Differencing with {} Lags'.format(lags))



plt.show()



# ========================================================================================================================

# PACF of first-order non-seasonal differencing



plt.figure()



plot_pacf(pm_first_difference, lags=lags)



plt.xlabel('Lags')

plt.ylabel('Magnitude')

plt.title('PACF of First-Order Non-Seasonal Differencing with {} Lags'.format(lags))



plt.show()



# ========================================================================================================================

# ADF testing



print('ADF test of first-order seasonal differencing, followed by first-order non-seasonal differencing:')



adf_cal(pm_first_difference)



# Analysis: still reject the null and conclude data is stationary.
# ========================================================================================================================

# The plots of the ACF and PACF denote the following:



# First-order seasonal differencing: ACF cuts off at lag 24, PACF tailing off every 24 lags

# First-order non-seasonal differencing: ACF and PACF cut off every 24 lags.



# Conclusion: Order estimates are a combination of AR/MA seasonal and non-seasonal; after office hours,

# concluded to try the following:



# SARIMA (2, 0, 0) x (1, 0, 0, 24)

# SARIMA (2, 0, 0) x (2, 0, 0, 24)

# SARIMA (1, 1, 0) x (0, 1, 1, 24)



# ========================================================================================================================

# Creating the SARIMA models



model_1 = SARIMAX(endog=Y_train, order=(2, 0, 0), seasonal_order=(1, 0, 0, 24), freq='H').fit()



print('Summary of SARIMA (2, 0, 0) x (1, 0, 0, 24):\n')



print(model_1.summary())



model_2 = SARIMAX(endog=Y_train, order=(2, 0, 0), seasonal_order=(2, 0, 0, 24), freq='H').fit()



print('Summary of SARIMA (2, 0, 0) x (2, 0, 0, 24):\n')



print(model_2.summary())



model_3 = SARIMAX(endog=Y_train, order=(1, 1, 0), seasonal_order=(0, 1, 1, 24), freq='H').fit()
```

```
print('Summary of SARIMA (1, 1, 0) x (0, 1, 1, 24):\n')


print(model_3.summary())
```

q12_lm_algorithm.py

```
import pandas as pd

import statsmodels.api as sm

from sklearn.model_selection import train_test_split

from statsmodels.tsa.statespace.sarimax import SARIMAX

import numpy as np

from scipy.stats import chi2

import matplotlib.pyplot as plt




# ========================================================================================================================

# Final Project: Levenberg Marquardt algorithm

# Benjamin Lee

# Data Science 6450-15: Time Series Modeling & Analysis

# 16 December 2020

# ========================================================================================================================

# Pre-processing procedures; see 'q5_description.py' for additional comments



df = pd.read_csv('PRSA_data_2010.1.1-2014.12.31.csv', header=0)



df['_date_'] = pd.to_datetime(df['year']*10000 + df['month']*100 + df['day'], format='%Y%m%d')



df['date'] = df['_date_'] + df['hour'].astype('timedelta64[h]')



df = df.drop(columns=['No', 'year', 'month', 'day', 'hour', '_date_'])



df = df.set_index('date')



df = df.loc['2010-01-02':'2011-01-01']
df = df.interpolate(method='time')


df['cbwd'] = df['cbwd'].astype('category')

df['cbwd'] = df['cbwd'].cat.codes


pm = df.loc[:, 'pm2.5']


predictors = df.drop(columns=['pm2.5', 'cbwd'])


predictors = sm.add_constant(predictors)


response = pm.copy(deep=True)


X_train, x_test, Y_train, y_test = train_test_split(predictors, response, shuffle=False, test_size=0.2)


# ========================================================================================================================

# Creating the SARIMA models


model_1 = SARIMAX(endog=Y_train, order=(2, 0, 0), seasonal_order=(1, 0, 0, 24), freq='H').fit()


print('Summary of SARIMA (2, 0, 0) x (1, 0, 0, 24):\n')


print(model_1.summary())


model_2 = SARIMAX(endog=Y_train, order=(2, 0, 0), seasonal_order=(2, 0, 0, 24), freq='H').fit()


print('Summary of SARIMA (2, 0, 0) x (2, 0, 0, 24):\n')


print(model_2.summary())
```

```
model_3 = SARIMAX(endog=Y_train, order=(1, 1, 0), seasonal_order=(0, 1, 1, 24), freq='H').fit()


print('Summary of SARIMA (1, 1, 0) x (0, 1, 1, 24):\n')


print(model_3.summary())


# ========================================================================================================================

# Confidence intervals and parameter estimates


print(100*'-')


sarima_model_1_conf_int = model_1.conf_int()

sarima_model_1_conf_int = sarima_model_1_conf_int.values.tolist()

sarima_model_1_conf_int = [val for sublist in sarima_model_1_conf_int for val in sublist]


print('The confidence intervals of SARIMA (2, 0, 0) x (1, 0, 0, 24) are as follows:\n')

print('Coefficient a1:', sarima_model_1_conf_int[0], '<=', model_1.params[0], '<=', sarima_model_1_conf_int[1])

print('Coefficient a2:', sarima_model_1_conf_int[2], '<=', model_1.params[1], '<=', sarima_model_1_conf_int[3])

print('Coefficient a24:', sarima_model_1_conf_int[4], '<=', model_1.params[2], '<=', sarima_model_1_conf_int[5])


sarima_model_2_conf_int = model_2.conf_int()

sarima_model_2_conf_int = sarima_model_2_conf_int.values.tolist()

sarima_model_2_conf_int = [val for sublist in sarima_model_2_conf_int for val in sublist]


print('The confidence intervals of SARIMA (2, 0, 0) x (2, 0, 0, 24) are as follows:\n')

print('Coefficient a1:', sarima_model_2_conf_int[0], '<=', model_2.params[0], '<=', sarima_model_2_conf_int[1])

print('Coefficient a2:', sarima_model_2_conf_int[2], '<=', model_2.params[1], '<=', sarima_model_2_conf_int[3])

print('Coefficient a24:', sarima_model_2_conf_int[4], '<=', model_2.params[2], '<=', sarima_model_2_conf_int[5])

print('Coefficient a48:', sarima_model_2_conf_int[6], '<=', model_2.params[3], '<=', sarima_model_2_conf_int[7])
```

```
sarima_model_3_conf_int = model_3.conf_int()

sarima_model_3_conf_int = sarima_model_3_conf_int.values.tolist()

sarima_model_3_conf_int = [val for sublist in sarima_model_3_conf_int for val in sublist]


print('The confidence intervals of SARIMA (1, 1, 0) x (0, 1, 1, 24) are as follows:\n')

print('Coefficient a1:', sarima_model_3_conf_int[0], '<=', model_3.params[0], '<=', sarima_model_3_conf_int[1])

print('Coefficient ma24:', sarima_model_3_conf_int[2], '<=', model_3.params[1], '<=', sarima_model_3_conf_int[3])


# ========================================================================================================================

# Standard deviation of the estimates


# This section is calculated by taking the difference of the coefficient and the upper confidence interval (to find the

# value of the confidence interval) and divide the result by two; this follows the equation for confidence interval in

# the lecture 10 slides. The entire square root of the equation is the standard deviation of the coefficient.


print('\nStandard deviations:')


# SARIMA (2, 0, 0) x (1, 0, 0, 24)


print('\nSARIMA (2, 0, 0) x (1, 0, 0, 24):\n')


print('Coefficient a1:', (1.134 - 1.1280) / 2)

print('Coefficient a2:', (-0.146 + 0.1525) / 2)

print('Coefficient a24:', (0.031 - 0.0070) / 2)


# SARIMA (2, 0, 0) x (2, 0, 0, 24)


print('\nSARIMA (2, 0, 0) x (2, 0, 0, 24):\n')


print('Coefficient a1:', (1.131 - 1.1251) / 2)
print('Coefficient a2:', (-0.146 + 0.1516) / 2)

print('Coefficient a24:', (0.033 - 0.0084) / 2)

print('Coefficient a48:', (0.077 - 0.0645) / 2)


# SARIMA (1, 1, 0) x (0, 1, 1, 24)


print('\nSARIMA (1, 1, 0) x (0, 1, 1, 24):\n')


print('Coefficient a1:', (0.140 - 0.1347) / 2)

print('Coefficient ma24:', (-0.971 + 0.9985))
```

q13_diagnostic_analysis_sarima.py

```
import pandas as pd

import statsmodels.api as sm

from sklearn.model_selection import train_test_split

from statsmodels.tsa.statespace.sarimax import SARIMAX

import numpy as np

from scipy.stats import chi2

import matplotlib.pyplot as plt



# =======================================================================================================================

# Final Project: Diagnostic analysis of SARIMA processes

# Benjamin Lee

# Data Science 6450-15: Time Series Modeling & Analysis

# 16 December 2020

# =======================================================================================================================

# Applicable functions




def difference(dataset, interval):

    """

    :param dataset: The dataset in question.

    :param interval: The interval for differencing. If seasonal differencing, enter the seasonality period here. For

    # simple differencing (non-seasonal), enter the order of differencing preferred.

    :return: The differenced data. Credit is given to professor Jafari, who provided this function in email

    # correspondence.

    """

    diff = []

    for i in range(interval, len(dataset)):

        _value_ = dataset[i] - dataset[i - interval]

        diff.append(_value_)
    return diff



def acf_function(dataset, lags):
    """
    :param dataset: The data used in calculating the ACF values.
    :param lags: The number of lags specified for calculating the ACF values.
    :return: The ACF values associated with the given dataset at a specified lag value.
    # Values for t are adjusted to account for the indexing differences between Python and the handwritten ACF.
    """
    t = lags + 1

    max_t = len(dataset)

    dataset_mean = np.mean(dataset)

    # Numerator
    numerator = 0
    while t < max_t + 1:
        numerator += (dataset[t - 1] - dataset_mean) * (dataset[t - lags - 1] - dataset_mean)
        t += 1

    t = 1

    # Denominator
    denominator = 0
    while t < max_t + 1:
        denominator += (dataset[t - 1] - dataset_mean) ** 2
        t += 1
    acf_value = numerator / denominator

    return acf_value


def acf_compiler(data, lags):
    """
    :param data: The data used in calculating the ACF values.
    :param lags: The number of lags specified for calculating the ACF values.
    :return: A Numpy array containing the two-sided experimental ACF values.
    """
    acf_values = []

    for k in range(lags + 1):
        acf_values.append(acf_function(data, k))

    acf_values = np.asarray(acf_values)
    acf_values_transform = acf_values[::-1]
    exp_acf_values_ = np.concatenate((np.reshape(acf_values_transform, lags + 1), acf_values[1:]))

    return exp_acf_values_


# ========================================================================================================================
# Pre-processing procedures; see 'q5_description.py' for additional comments

df = pd.read_csv('PRSA_data_2010.1.1-2014.12.31.csv', header=0)

df['_date_'] = pd.to_datetime(df['year']*10000 + df['month']*100 + df['day'], format='%Y%m%d')
```

```
df['date'] = df['_date_'] + df['hour'].astype('timedelta64[h]')


df = df.drop(columns=['No', 'year', 'month', 'day', 'hour', '_date_'])


df = df.set_index('date')


df = df.loc['2010-01-02':'2011-01-01']


df = df.interpolate(method='time')


df['cbwd'] = df['cbwd'].astype('category')

df['cbwd'] = df['cbwd'].cat.codes


pm = df.loc[:, 'pm2.5']


predictors = df.drop(columns=['pm2.5', 'cbwd'])


predictors = sm.add_constant(predictors)


response = pm.copy(deep=True)


X_train, x_test, Y_train, y_test = train_test_split(predictors, response, shuffle=False, test_size=0.2)


# ========================================================================================================================

# Creating the SARIMA models


model_1 = SARIMAX(endog=Y_train, order=(2, 0, 0), seasonal_order=(1, 0, 0, 24), freq='H').fit()


print('Summary of SARIMA (2, 0, 0) x (1, 0, 0, 24):\n')
print(model_1.summary())


model_2 = SARIMAX(endog=Y_train, order=(2, 0, 0), seasonal_order=(2, 0, 0, 24), freq='H').fit()


print('Summary of SARIMA (2, 0, 0) x (2, 0, 0, 24):\n')


print(model_2.summary())


model_3 = SARIMAX(endog=Y_train, order=(1, 1, 0), seasonal_order=(0, 1, 1, 24), freq='H').fit()


print('Summary of SARIMA (1, 1, 0) x (0, 1, 1, 24):\n')


print(model_3.summary())


# ======================================================================================================================

# Part A: Diagnostic tests


# =========================================================

# Confidence intervals

# =========================================================


print(100*'-')


sarima_model_1_conf_int = model_1.conf_int()

sarima_model_1_conf_int = sarima_model_1_conf_int.values.tolist()

sarima_model_1_conf_int = [val for sublist in sarima_model_1_conf_int for val in sublist]


print('The confidence intervals of SARIMA (2, 0, 0) x (1, 0, 0, 24) are as follows:\n')

print('Coefficient a1:', sarima_model_1_conf_int[0], '<=', model_1.params[0], '<=', sarima_model_1_conf_int[1])

print('Coefficient a2:', sarima_model_1_conf_int[2], '<=', model_1.params[1], '<=', sarima_model_1_conf_int[3])
print('Coefficient a24:', sarima_model_1_conf_int[4], '<=', model_1.params[2], '<=', sarima_model_1_conf_int[5])


sarima_model_2_conf_int = model_2.conf_int()

sarima_model_2_conf_int = sarima_model_2_conf_int.values.tolist()

sarima_model_2_conf_int = [val for sublist in sarima_model_2_conf_int for val in sublist]


print('The confidence intervals of SARIMA (2, 0, 0) x (2, 0, 0, 24) are as follows:\n')

print('Coefficient a1:', sarima_model_2_conf_int[0], '<=', model_2.params[0], '<=', sarima_model_2_conf_int[1])

print('Coefficient a2:', sarima_model_2_conf_int[2], '<=', model_2.params[1], '<=', sarima_model_2_conf_int[3])

print('Coefficient a24:', sarima_model_2_conf_int[4], '<=', model_2.params[2], '<=', sarima_model_2_conf_int[5])

print('Coefficient a48:', sarima_model_2_conf_int[6], '<=', model_2.params[3], '<=', sarima_model_2_conf_int[7])


sarima_model_3_conf_int = model_3.conf_int()

sarima_model_3_conf_int = sarima_model_3_conf_int.values.tolist()

sarima_model_3_conf_int = [val for sublist in sarima_model_3_conf_int for val in sublist]


print('The confidence intervals of SARIMA (1, 1, 0) x (0, 1, 1, 24) are as follows:\n')

print('Coefficient a1:', sarima_model_3_conf_int[0], '<=', model_3.params[0], '<=', sarima_model_3_conf_int[1])

print('Coefficient ma24:', sarima_model_3_conf_int[2], '<=', model_3.params[1], '<=', sarima_model_3_conf_int[3])


# =========================================================

# Zero/Pole cancellations

# =========================================================


print(100*'-')


# SARIMA (2, 0, 0) x (1, 0, 0, 24)


num = [1, 0]

den = [1, model_1.params[0], model_1.params[1], model_1.params[2]]
```

```
zeros = np.roots(num)

poles = np.roots(den)



print('\nZeros of SARIMA (2, 0, 0) x (1, 0, 0, 24):', zeros)

print('Poles of SARIMA (2, 0, 0) x (1, 0, 0, 24):', poles)



# SARIMA (2, 0, 0) x (2, 0, 0, 24)



num = [1, 0]

den = [1, model_2.params[0], model_2.params[1], model_2.params[2], model_2.params[3]]



zeros = np.roots(num)

poles = np.roots(den)



print('\nZeros of SARIMA (2, 0, 0) x (2, 0, 0, 24):', zeros)

print('Poles of SARIMA (2, 0, 0) x (2, 0, 0, 24):', poles)



# SARIMA (1, 1, 0) x (0, 1, 1, 24)



num = [1, model_3.params[1]]

den = [1, model_3.params[0]]



zeros = np.roots(num)

poles = np.roots(den)



print('\nZeros of SARIMA (1, 1, 0) x (0, 1, 1, 24):', zeros)

print('Poles of SARIMA (1, 1, 0) x (0, 1, 1, 24):', poles)



# There are no zero/pole cancellations; thank goodness!
# =========================================================

# Chi-square tests

# =========================================================


print(100*'-')


lags = 55


# SARIMA (2, 0, 0) x (1, 0, 0, 24)


na = 26

nb = 0


osa_predictions_model_1 = []


for i in range(len(Y_train.index)):

    if i == 0:

        osa_predictions_model_1.append(model_1.params[0] * Y_train[i])

    elif 0 < i < 23:

        osa_predictions_model_1.append(model_1.params[0] * Y_train[i] + model_1.params[1] * Y_train[i - 1])

    elif i == 23:

        osa_predictions_model_1.append(model_1.params[0] * Y_train[i] + model_1.params[1] * Y_train[i - 1] + model_1.params[2] *
Y_train[i - 23])

    elif i == 24:

        osa_predictions_model_1.append(model_1.params[0] * Y_train[i] + model_1.params[1] * Y_train[i - 1] + model_1.params[2] *
Y_train[i - 23] - model_1.params[0] * model_1.params[2] * Y_train[i - 24])

    else:

        osa_predictions_model_1.append(model_1.params[0] * Y_train[i] + model_1.params[1] * Y_train[i - 1] + model_1.params[2] *
Y_train[i - 23] - model_1.params[0] * model_1.params[2] * Y_train[i - 24] - model_1.params[1] * model_1.params[2] * Y_train[i -
25])
```

```
e = Y_train[1:] - osa_predictions_model_1[:-1]


# re = acf_compiler(data=e, lags=lags)

re = sm.tsa.stattools.acf(e, nlags=lags)


Q = len(Y_train) * np.sum(np.square(re))


lb_value, p_value = sm.stats.acorr_ljungbox(e, lags=lags)


print('The lb value is', lb_value)

print('The p-value is', p_value)


DOF = lags - na - nb


alpha = 0.05


chi_critical = chi2.ppf(1 - alpha, DOF)


if Q < chi_critical:

    print('\nQ value:', Q)

    print('Critical Q:', chi_critical)

    print('The residuals are white.')


else:

    print('\nQ value:', Q)

    print('Critical Q:', chi_critical)

    print('The residuals are not white.')


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

# # SARIMA (2, 0, 0) x (2, 0, 0, 24)
# # ------------------------------------------------------

na = 50
nb = 0

osa_predictions_model_2 = []

for i in range(len(Y_train.index)):
    if i == 0:
        osa_predictions_model_2.append(model_2.params[0] * Y_train[i])
    elif 0 < i < 23:
        osa_predictions_model_2.append(model_2.params[0] * Y_train[i] + model_2.params[1] * Y_train[i - 1])
    elif i == 23:
        osa_predictions_model_2.append(model_2.params[0] * Y_train[i] + model_2.params[1] * Y_train[i - 1] + model_2.params[2] *
Y_train[i - 23])
    elif i == 24:
        osa_predictions_model_2.append(model_2.params[0] * Y_train[i] + model_2.params[1] * Y_train[i - 1] + model_2.params[2] *
Y_train[i - 23] - model_2.params[0] * model_2.params[2] * Y_train[i - 24])
    elif 25 < i < 47:
        osa_predictions_model_2.append(model_2.params[0] * Y_train[i] + model_2.params[1] * Y_train[i - 1] + model_2.params[2] *
Y_train[i - 23] - model_2.params[0] * model_2.params[2] * Y_train[i - 24] - model_2.params[1] * model_2.params[2] * Y_train[i -
25])
    elif i == 47:
        osa_predictions_model_2.append(model_2.params[0] * Y_train[i] + model_2.params[1] * Y_train[i - 1] + model_2.params[2] *
Y_train[i - 23] - model_2.params[0] * model_2.params[2] * Y_train[i - 24] - model_2.params[1] * model_2.params[2] * Y_train[i -
25] + model_2.params[3] * Y_train[i - 47])
    elif i == 48:
        osa_predictions_model_2.append(model_2.params[0] * Y_train[i] + model_2.params[1] * Y_train[i - 1] + model_2.params[2] *
Y_train[i - 23] - model_2.params[0] * model_2.params[2] * Y_train[i - 24] - model_2.params[1] * model_2.params[2] * Y_train[i -
25] + model_2.params[3] * Y_train[i - 47] - model_2.params[0] * model_2.params[3] * Y_train[i - 48])
    else:
        osa_predictions_model_2.append(model_2.params[0] * Y_train[i] + model_2.params[1] * Y_train[i - 1] + model_2.params[2] *
Y_train[i - 23] - model_2.params[0] * model_2.params[2] * Y_train[i - 24] - model_2.params[1] * model_2.params[2] * Y_train[i -
25] + model_2.params[3] * Y_train[i - 47] - model_2.params[0] * model_2.params[3] * Y_train[i - 48] - model_2.params[1] *
model_2.params[3] * Y_train[i - 49])


e = Y_train[1:] - osa_predictions_model_2[:-1]


# re = acf_compiler(data=e, lags=lags)

re = sm.tsa.stattools.acf(e, nlags=lags)


Q = len(Y_train) * np.sum(np.square(re))


lb_value, p_value = sm.stats.acorr_ljungbox(e, lags=lags)


print('The lb value is', lb_value)

print('The p-value is', p_value)


DOF = lags - na - nb


alpha = 0.05


chi_critical = chi2.ppf(1 - alpha, DOF)


if Q < chi_critical:

    print('\nQ value:', Q)

    print('Critical Q:', chi_critical)

    print('The residuals are white.')


else:

    print('\nQ value:', Q)

    print('Critical Q:', chi_critical)

    print('The residuals are not white.')
```

```
# # -------------------------------------------------------

# # SARIMA (1, 1, 0) x (0, 1, 1, 24)

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



na = 26

nb = 24



osa_predictions_model_3 = []



for i in range(len(Y_train.index)):

    if i == 0:

        osa_predictions_model_3.append(Y_train[i] + model_3.params[0] * Y_train[i])

    elif 0 < i < 23:

        osa_predictions_model_3.append(Y_train[i] + model_3.params[0] * Y_train[i] - model_3.params[0] * Y_train[i - 1])

    elif i == 23:

        osa_predictions_model_3.append(Y_train[i] + model_3.params[0] * Y_train[i] - model_3.params[0] * Y_train[i - 1] +
Y_train[i - 23] + model_3.params[1] * Y_train[i - 23])

    elif i == 24:

        osa_predictions_model_3.append(Y_train[i] + model_3.params[0] * Y_train[i] - model_3.params[0] * Y_train[i - 1] +
Y_train[i - 23] + model_3.params[1] * Y_train[i - 23] - Y_train[i - 24] - model_3.params[1] * osa_predictions_model_3[i - 23] -
model_3.params[0] * Y_train[24])

    else:

        osa_predictions_model_3.append(Y_train[i] + model_3.params[0] * Y_train[i] - model_3.params[0] * Y_train[i - 1] +
Y_train[i - 23] + model_3.params[1] * Y_train[i - 23] - Y_train[i - 24] - model_3.params[1] * osa_predictions_model_3[i - 23] -
model_3.params[0] * Y_train[24] + model_3.params[0] * Y_train[i - 25])



e = Y_train[1:] - osa_predictions_model_3[:-1]



# re = acf_compiler(data=e, lags=lags)

re = sm.tsa.stattools.acf(e, nlags=lags)



Q = len(Y_train) * np.sum(np.square(re))
```

```
lb_value, p_value = sm.stats.acorr_ljungbox(e, lags=lags)


print('The lb value is', lb_value)

print('The p-value is', p_value)


DOF = lags - na - nb


alpha = 0.05


chi_critical = chi2.ppf(1 - alpha, DOF)


if Q < chi_critical:

    print('\nQ value:', Q)

    print('Critical Q:', chi_critical)

    print('The residuals are white.')


else:

    print('\nQ value:', Q)

    print('Critical Q:', chi_critical)

    print('The residuals are not white.')


# ======================================================================================================================

# Part B: Variance of the error and covariance


# The variance of the error was found be reading the standard errors of the coefficients and squaring them.


# SARIMA (2, 0, 0) x (1, 0, 0, 24)


print('\nSARIMA (2, 0, 0) x (1, 0, 0, 24):\n')
print('Coefficient a1:', np.square(0.003))

print('Coefficient a2:', np.square(0.003))

print('Coefficient a24:', np.square(0.012))


# SARIMA (2, 0, 0) x (2, 0, 0, 24)


print('\nSARIMA (2, 0, 0) x (2, 0, 0, 24):\n')


print('Coefficient a1:', np.square(0.003))

print('Coefficient a2:', np.square(0.003))

print('Coefficient a24:', np.square(0.013))

print('Coefficient a48:', np.square(0.006))


# SARIMA (1, 1, 0) x (0, 1, 1, 24)


print('\nSARIMA (1, 1, 0) x (0, 1, 1, 24):\n')


print('Coefficient a1:', np.square(0.003))

print('Coefficient ma24:', np.square(0.014))


# Covariances of the parameter estimates were found by printing the covariance matrices.


print('\nCovariance matrices')


print('\nSARIMA (2, 0, 0) x (1, 0, 0, 24):\n', model_1.cov_params())


print('\nSARIMA (2, 0, 0) x (2, 0, 0, 24):\n', model_2.cov_params())


print('\nSARIMA (1, 1, 0) x (0, 1, 1, 24):\n', model_3.cov_params())
```

```
# =======================================================================================================================

# Part C: Determining whether the model is a biased or unbiased estimator


# After talking with professor Jafari, he said a model is unbiased if the mean of the residuals is non-zero or not

# very close to zero.


model_1_residual_mean = np.mean(osa_predictions_model_1)


model_2_residual_mean = np.mean(osa_predictions_model_2)


model_3_residual_mean = np.mean(osa_predictions_model_3)


print('\nMean of residuals for SARIMA (2, 0, 0) x (1, 0, 0, 24):', model_1_residual_mean)


print('Mean of residuals for SARIMA (2, 0, 0) x (2, 0, 0, 24):', model_2_residual_mean)


print('Mean of residuals for SARIMA (1, 1, 0) x (0, 1, 1, 24):', model_3_residual_mean)


# All three models appear to be biased estimators, but this could be due to errors in calculating one-step ahead

# predictions.


# =======================================================================================================================

# Part D: Check the variance of the residual errors vs the forecast errors


# This part cannot be done without first computing h-step ahead forecasts and comparing the values with the test set to

# compute forecast errors. That step is done below. Upon comparing results, the best model of this project will apply

# h-step ahead predictions using the forecast function, not .forecast().


model_1_residual_var = np.var(osa_predictions_model_1)
```

```
model_2_residual_var = np.var(osa_predictions_model_2)


model_3_residual_var = np.var(osa_predictions_model_3)


model_1_forecasts = model_1.forecast(steps=len(y_test))


model_1_forecast_errors = y_test - model_1_forecasts

model_1_forecast_errors_var = np.var(model_1_forecast_errors)


model_2_forecasts = model_2.forecast(steps=len(y_test))

model_2_forecast_errors = y_test - model_2_forecasts

model_2_forecast_errors_var = np.var(model_2_forecast_errors)


model_3_forecasts = model_3.forecast(steps=len(y_test))

model_3_forecast_errors = y_test - model_3_forecasts

model_3_forecast_errors_var = np.var(model_3_forecast_errors)


print('\nVariance of residuals for SARIMA (2, 0, 0) x (1, 0, 0, 24):', model_1_residual_var)

print('Variance of forecast errors for SARIMA (2, 0, 0) x (1, 0, 0, 24):', model_1_forecast_errors_var)


print('\nVariance of residuals for SARIMA (2, 0, 0) x (2, 0, 0, 24):', model_2_residual_var)

print('Variance of forecast errors for SARIMA (2, 0, 0) x (2, 0, 0, 24):', model_2_forecast_errors_var)


print('\nVariance of residuals for SARIMA (1, 1, 0) x (0, 1, 1, 24):', model_3_residual_var)

print('Variance of forecast errors for SARIMA (1, 1, 0) x (0, 1, 1, 24):', model_3_forecast_errors_var)


# ========================================================================================================================

# Part E: If you find that the ARIMA or SARIMA model better represents the dataset, then you can find the model

# accordingly. You are not constraint only to use of ARMA model.
# Because my data is seasonal ARMA (i.e. SARIMA), I decided to test different SARIMA models for this project. Applying

# non-seasonal ARMA models would make no sense, given the seasonal component present in the ACF and PACF plots.


# ======================================================================================================================

# Additional: Plotting the one-step ahead predictions against the training set


# SARIMA (2, 0, 0) x (1, 0, 0, 24)


osa_predictions_model_1 = pd.Series(osa_predictions_model_1, index=Y_train.index)

osa_predictions_model_2 = pd.Series(osa_predictions_model_2, index=Y_train.index)

osa_predictions_model_3 = pd.Series(osa_predictions_model_3, index=Y_train.index)


plt.figure()


plt.plot(Y_train, label='Training data')

plt.plot(y_test, label='Testing data')

plt.plot(osa_predictions_model_1, label='One-step ahead predictions')

plt.plot(model_1_forecasts, label='H-step ahead forecasts')


plt.title('Plot of OSA predictions - SARIMA (2, 0, 0) x (1, 0, 0, 24)')

plt.xlabel('Time (in hours)')

plt.ylabel('PM 2.5 Concentration (µg / cu m)')

plt.legend(loc='upper right')


plt.show()


plt.figure()


plt.plot(Y_train[:50], label='Training data')
plt.plot(osa_predictions_model_1[:50], label='One-step ahead predictions')


plt.title('OSA predictions - SARIMA (2, 0, 0) x (1, 0, 0, 24) (first 50 samples)')

plt.xlabel('Time (in hours)')

plt.ylabel('PM 2.5 Concentration (µg / cu m)')

plt.legend(loc='upper right')


plt.xticks(rotation=45)


plt.show()


# SARIMA (2, 0, 0) x (2, 0, 0, 24)


plt.figure()


plt.plot(Y_train, label='Training data')

plt.plot(y_test, label='Testing data')

plt.plot(osa_predictions_model_2, label='One-step ahead predictions')

plt.plot(model_2_forecasts, label='H-step ahead forecasts')


plt.title('Plot of OSA predictions - SARIMA (2, 0, 0) x (2, 0, 0, 24)')

plt.xlabel('Time (in hours)')

plt.ylabel('PM 2.5 Concentration (µg / cu m)')

plt.legend(loc='upper right')


plt.show()


plt.figure()


plt.plot(Y_train[:50], label='Training data')
plt.plot(osa_predictions_model_2[:50], label='One-step ahead predictions')


plt.title('OSA predictions - SARIMA (2, 0, 0) x (2, 0, 0, 24) (first 50 samples)')

plt.xlabel('Time (in hours)')

plt.ylabel('PM 2.5 Concentration (µg / cu m)')

plt.legend(loc='upper right')


plt.xticks(rotation=45)


plt.show()


# SARIMA (1, 1, 0) x (0, 1, 1, 24)


plt.figure()


plt.plot(Y_train, label='Training data')

plt.plot(y_test, label='Testing data')

plt.plot(osa_predictions_model_3, label='One-step ahead predictions')

plt.plot(model_3_forecasts, label='H-step ahead forecasts')


plt.title('Plot of OSA predictions - SARIMA (1, 1, 0) x (0, 1, 1, 24)')

plt.xlabel('Time (in hours)')

plt.ylabel('PM 2.5 Concentration (µg / cu m)')

plt.legend(loc='upper right')


plt.show()


plt.figure()


plt.plot(Y_train[:50], label='Training data')
plt.plot(osa_predictions_model_3[:50], label='One-step ahead predictions')



plt.title('OSA predictions - SARIMA (1, 1, 0) x (0, 1, 1, 24) (first 50 samples)')

plt.xlabel('Time (in hours)')

plt.ylabel('PM 2.5 Concentration (µg / cu m)')

plt.legend(loc='upper right')



plt.xticks(rotation=45)



plt.show()
```

q14_base_models.py

```python
import pandas as pd

import statsmodels.api as sm

from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt

import numpy as np



# ========================================================================================================================

# Final Project: Base models

# Benjamin Lee

# Data Science 6450-15: Time Series Modeling & Analysis

# 16 December 2020

# ========================================================================================================================

# Applicable functions




def one_step_ahead_average_method(training_data):

    """

    :param training_data: The training set used in this assignment.

    :return: A list containing the one-step ahead predictions using the average forecast method.

    """

    predictions = [0]

    for value in range(len(training_data) - 1):

        predictions.append(round(sum(training_data[:value + 1]) / (value + 1), 3))

    return predictions




def h_step_ahead_average_method(training_data, testing_data):

    """

    :param training_data: The training set used in this assignment.
    :param testing_data: The testing set used in this assignment.

    :return: A list containing the h-step ahead forecasts using the average forecast method.

    """

    forecast_value = round(sum(training_data[::]) / len(training_data), 3)

    forecasts = [forecast_value] * len(testing_data)

    return forecasts




def one_step_ahead_naive_method(training_data):

    """

    :param training_data: The training set used in this assignment.

    :return: A list containing the one-step ahead predictions using the naive forecast method.

    """

    predictions = [0]

    for value in range(len(training_data) - 1):

        predictions.append(training_data[value])

    return predictions




def h_step_ahead_naive_method(training_data, testing_data):

    """

    :param training_data: The training set used in this assignment.

    :param testing_data: The testing set used in this assignment.

    :return: A list containing the h-step ahead forecasts using the average forecast method.

    """

    forecast_value = training_data[-1]

    forecasts = [forecast_value] * len(testing_data)

    return forecasts
```

```
def one_step_ahead_drift_method(training_data):

    """

    :param training_data: The training set used in this assignment.

    :return: A list containing the one-step ahead forecasts using the drift forecast method.

    """

    predictions = [0, training_data[0]]

    for value in range(1, len(training_data) - 1):

        predictions.append(round(training_data[value] + ((training_data[value] - training_data[0]) / value), 3))

    return predictions




def h_step_ahead_drift_method(training_data, testing_data):

    """

    :param training_data: The training set used in this assignment.

    :param testing_data: The testing set used in this assignment.

    :return: A list containing the h-step ahead forecasts using the drift forecast method.

    """

    forecasts = []

    for value in range(1, len(testing_data) + 1):

        forecasts.append(round(training_data[-1] + value * ((training_data[-1] - training_data[0]) / (len(training_data) - 1)),
3))

    return forecasts




def one_step_ahead_ses_method(training_data, alpha):

    """

    :param training_data: The training set used in this assignment.

    :param alpha: The value of the smoothing parameter.

    :return: A list containing the one-step ahead forecasts using the simple exponential smoothing forecast method.

    """

    predictions = [training_data[0]]
    for value in range(len(training_data) - 1):

        predictions.append(round(alpha * training_data[value] + (1 - alpha) * predictions[value], 3))

    return predictions




# ========================================================================================================================

# Pre-processing procedures; see 'q5_description.py' for additional comments



df = pd.read_csv('PRSA_data_2010.1.1-2014.12.31.csv', header=0)



df['_date_'] = pd.to_datetime(df['year']*10000 + df['month']*100 + df['day'], format='%Y%m%d')



df['date'] = df['_date_'] + df['hour'].astype('timedelta64[h]')



df = df.drop(columns=['No', 'year', 'month', 'day', 'hour', '_date_'])



df = df.set_index('date')



df = df.loc['2010-01-02':'2011-01-01']



df = df.interpolate(method='time')



df['cbwd'] = df['cbwd'].astype('category')

df['cbwd'] = df['cbwd'].cat.codes



pm = df.loc[:, 'pm2.5']



predictors = df.drop(columns=['pm2.5', 'cbwd'])



predictors = sm.add_constant(predictors)
```

```
response = pm.copy(deep=True)


X_train, x_test, Y_train, y_test = train_test_split(predictors, response, shuffle=False, test_size=0.2)


lags = 50


# ========================================================================================================================

# Average method


osa_predictions_average_method = one_step_ahead_average_method(Y_train)

osa_predictions_average_method = pd.Series(osa_predictions_average_method, index=Y_train.index)


hsa_forecasts_average_method = h_step_ahead_average_method(Y_train, y_test)

hsa_forecasts_average_method = pd.Series(hsa_forecasts_average_method, index=y_test.index)


plt.figure()


plt.plot(Y_train, label='Training Dataset')

plt.plot(y_test, label='Testing Dataset')

plt.plot(osa_predictions_average_method, label='One-step ahead predictions\n(Average Method)')

plt.plot(hsa_forecasts_average_method, label='H-step ahead predictions\n(Average Method)')


plt.title('Plot of Average method predictions and forecasts vs original data')

plt.xlabel('Time (in hours)')

plt.ylabel('PM 2.5 Concentration (μg / cu m)')

plt.legend(loc='upper right')


plt.show()
plt.figure()



plt.plot(Y_train[:50], label='Training Dataset')

plt.plot(osa_predictions_average_method[:50], label='One-step ahead predictions\n(Average Method)')



plt.title('Plot of Average method vs original data (first 50 samples)')

plt.xlabel('Time (in hours)')

plt.ylabel('PM 2.5 Concentration (µg / cu m)')

plt.legend(loc='upper right')



plt.xticks(rotation=45)



plt.show()



# Residuals



residuals = []



for i in range(len(Y_train.index)):

    residuals.append(Y_train[i] - osa_predictions_average_method[i])



print('The mean of the residuals for the average method are', np.mean(residuals))



# Forecast errors



forecast_errors = []



for i in range(len(y_test.index)):

    forecast_errors.append(y_test[i] - hsa_forecasts_average_method[i])
```

```
print('\nThe variance of the residuals for the average method is', np.var(residuals))

print('The variance of the forecast errors for the average method is', np.var(forecast_errors))



re = sm.tsa.stattools.acf(residuals, nlags=lags)



Q = len(Y_train) * np.sum(np.square(re))



print('The Q-value of the residuals is', Q)



# Analysis: The mean of the residuals is somewhat close to zero (much closer than the SARIMA models), but is still

# likely a biased estimator due to its non-zero value. Additionally, the overall curves of the one-step ahead

# predictions and H-step ahead forecasts for the average are terrible in comparison to the original training set. The

# variance of the forecast errors is much higher than the variance of the one-step ahead predictions. This is likely

# not the best model for this dataset. Additionally, Q-value is enormous; the auto-correlations definitely don't come

# from a white noise series.



# ========================================================================================================================

# Naive method



osa_predictions_naive_method = one_step_ahead_naive_method(Y_train)

osa_predictions_naive_method = pd.Series(osa_predictions_naive_method, index=Y_train.index)



hsa_forecasts_naive_method = h_step_ahead_naive_method(Y_train, y_test)

hsa_forecasts_naive_method = pd.Series(hsa_forecasts_naive_method, index=y_test.index)



plt.figure()



plt.plot(Y_train, label='Training Dataset')

plt.plot(y_test, label='Testing Dataset')

plt.plot(osa_predictions_naive_method, label='One-step ahead predictions\n(Naive Method)')
plt.plot(hsa_forecasts_naive_method, label='H-step ahead predictions\n(Naive Method)')



plt.title('Plot of Naive method predictions and forecasts vs original data')

plt.xlabel('Time (in hours)')

plt.ylabel('PM 2.5 Concentration (µg / cu m)')

plt.legend(loc='upper right')



plt.show()



plt.figure()



plt.plot(Y_train[:50], label='Training Dataset')

plt.plot(osa_predictions_naive_method[:50], label='One-step ahead predictions\n(Naive Method)')



plt.title('Plot of Naive method vs original data (first 50 samples)')

plt.xlabel('Time (in hours)')

plt.ylabel('PM 2.5 Concentration (µg / cu m)')

plt.legend(loc='upper right')



plt.xticks(rotation=45)



plt.show()



# Residuals



residuals = []



for i in range(len(Y_train.index)):

    residuals.append(Y_train[i] - osa_predictions_naive_method[i])
```

```
print('\nThe mean of the residuals for the naive method are', np.mean(residuals))


# Forecast errors


forecast_errors = []


for i in range(len(y_test.index)):

    forecast_errors.append(y_test[i] - hsa_forecasts_naive_method[i])


print('\nThe variance of the residuals for the naive method is', np.var(residuals))

print('The variance of the forecast errors for the naive method is', np.var(forecast_errors))


re = sm.tsa.stattools.acf(residuals, nlags=lags)


Q = len(Y_train) * np.sum(np.square(re))


print('The Q-value of the residuals is', Q)


# Analysis: The naive method performs great on the training set; the values of the training set and the one-step ahead

# predictions practically overlap. Additionally, the mean of the residuals is very close to zero; this means the naive

# model is likely an unbiased estimator. However, the results of the h-step ahead predictions are awful. The h-step

# ahead predictions do not match the testing set at all, and the variance of the forecast errors are extremely large.

# This is likely not the best model for this dataset. The Q-value is large, but right on par with the SARIMA methods;

# the auto-correlations likely don't come from a white noise series.


# ======================================================================================================================

# Drift method


osa_predictions_drift_method = one_step_ahead_drift_method(Y_train)

osa_predictions_drift_method = pd.Series(osa_predictions_drift_method, index=Y_train.index)
```

```
hsa_forecasts_drift_method = h_step_ahead_drift_method(Y_train, y_test)

hsa_forecasts_drift_method = pd.Series(hsa_forecasts_drift_method, index=y_test.index)



plt.figure()



plt.plot(Y_train, label='Training Dataset')

plt.plot(y_test, label='Testing Dataset')

plt.plot(osa_predictions_drift_method, label='One-step ahead predictions\n(Drift Method)')

plt.plot(hsa_forecasts_drift_method, label='H-step ahead predictions\n(Drift Method)')



plt.title('Plot of Drift method predictions and forecasts vs original data')

plt.xlabel('Time (in hours)')

plt.ylabel('PM 2.5 Concentration (µg / cu m)')

plt.legend(loc='upper right')



plt.show()



plt.figure()



plt.plot(Y_train[:50], label='Training Dataset')

plt.plot(osa_predictions_drift_method[:50], label='One-step ahead predictions\n(Drift Method)')



plt.title('Plot of Drift method vs original data (first 50 samples)')

plt.xlabel('Time (in hours)')

plt.ylabel('PM 2.5 Concentration (µg / cu m)')

plt.legend(loc='upper right')



plt.xticks(rotation=45)
plt.show()

# Residuals

residuals = []

for i in range(len(Y_train.index)):
    residuals.append(Y_train[i] - osa_predictions_drift_method[i])

print('\nThe mean of the residuals for the Drift method are', np.mean(residuals))

# Forecast errors

forecast_errors = []

for i in range(len(y_test.index)):
    forecast_errors.append(y_test[i] - hsa_forecasts_drift_method[i])

print('\nThe variance of the residuals for the Drift method is', np.var(residuals))
print('The variance of the forecast errors for the Drift method is', np.var(forecast_errors))

re = sm.tsa.stattools.acf(residuals, nlags=lags)

Q = len(Y_train) * np.sum(np.square(re))

print('The Q-value of the residuals is', Q)

# Analysis: The results of the drift method are very similar to the naive method. The only big change is the h-step
# ahead forecasts, whose plot displays a slightly downward trend. The h-step ahead forecasts using the drift method are
# terrible; this is also likely not the best model for the dataset. The Q-value is large, but right on par with the
# SARIMA methods; the auto-correlations likely don't come from a white noise series.



# ========================================================================================================================

# Simple exponential smoothing



osa_predictions_ses_method = one_step_ahead_ses_method(Y_train, alpha=0.5)

osa_predictions_ses_method = pd.Series(osa_predictions_ses_method, index=Y_train.index)



hsa_forecasts_ses_method = [osa_predictions_ses_method[-1] for _ in range(len(y_test))]

hsa_forecasts_ses_method = pd.Series(hsa_forecasts_ses_method, index=y_test.index)



plt.figure()



plt.plot(Y_train, label='Training Dataset')

plt.plot(y_test, label='Testing Dataset')

plt.plot(osa_predictions_ses_method, label='One-step ahead predictions\n(SES Method)')

plt.plot(hsa_forecasts_ses_method, label='H-step ahead predictions\n(SES Method)')



plt.title('Plot of Simple Exponential Smoothing vs original data')

plt.xlabel('Time (in hours)')

plt.ylabel('PM 2.5 Concentration (µg / cu m)')

plt.legend(loc='upper right')



plt.show()



plt.figure()



plt.plot(Y_train[:50], label='Training Dataset')

plt.plot(osa_predictions_ses_method[:50], label='One-step ahead predictions\n(SES Method)')
plt.title('Plot of SES Method vs original data (first 50 samples)')

plt.xlabel('Time (in hours)')

plt.ylabel('PM 2.5 Concentration (µg / cu m)')

plt.legend(loc='upper right')


plt.xticks(rotation=45)


plt.show()


# Residuals


residuals = []


for i in range(len(Y_train.index)):

    residuals.append(Y_train[i] - osa_predictions_ses_method[i])


print('\nThe mean of the residuals for the SES method are', np.mean(residuals))


# Forecast errors


forecast_errors = []


for i in range(len(y_test.index)):

    forecast_errors.append(y_test[i] - hsa_forecasts_ses_method[i])


print('\nThe variance of the residuals for the SES method is', np.var(residuals))

print('The variance of the forecast errors for the SES method is', np.var(forecast_errors))


re = sm.tsa.stattools.acf(residuals, nlags=lags)
```

```
Q = len(Y_train) * np.sum(np.square(re))



print('The Q-value of the residuals is', Q)



# Analysis: Simple exponential works very well on the training set, and the mean of the residuals is practically zero.

# However, the plot of h-step ahead predictions vs the testing set is terrible. h-step ahead predictions do not match

# the testing set at all. This is likely not a good model for this dataset. The Q-value is a bit larger than the other

# base models, as well as the SARIMA methods; the auto-correlations likely don't come from a white noise series.
```

q15_h_step.py

```
import pandas as pd

import statsmodels.api as sm

from sklearn.model_selection import train_test_split

from statsmodels.tsa.statespace.sarimax import SARIMAX

import matplotlib.pyplot as plt

import numpy as np



# ========================================================================================================================

# Final Project: Final model h-step prediction

# Benjamin Lee

# Data Science 6450-15: Time Series Modeling & Analysis

# 16 December 2020

# ========================================================================================================================

# Pre-processing procedures; see 'q5_description.py' for additional comments



df = pd.read_csv('PRSA_data_2010.1.1-2014.12.31.csv', header=0)



df['_date_'] = pd.to_datetime(df['year']*10000 + df['month']*100 + df['day'], format='%Y%m%d')



df['date'] = df['_date_'] + df['hour'].astype('timedelta64[h]')



df = df.drop(columns=['No', 'year', 'month', 'day', 'hour', '_date_'])



df = df.set_index('date')



df = df.loc['2010-01-02':'2011-01-01']



df = df.interpolate(method='time')
```

```
df['cbwd'] = df['cbwd'].astype('category')

df['cbwd'] = df['cbwd'].cat.codes



pm = df.loc[:, 'pm2.5']



predictors = df.drop(columns=['pm2.5', 'cbwd'])



predictors = sm.add_constant(predictors)



response = pm.copy(deep=True)



X_train, x_test, Y_train, y_test = train_test_split(predictors, response, shuffle=False, test_size=0.2)



# ========================================================================================================================

# Creating the SARIMA (2, 0, 0) x (1, 0, 0, 24) model



model = SARIMAX(endog=Y_train, order=(2, 0, 0), seasonal_order=(1, 0, 0, 24), freq='H').fit()



# ========================================================================================================================

# One-step ahead predictions



osa_predictions = []



for i in range(len(Y_train.index)):

    if i == 0:

        osa_predictions.append(model.params[0] * Y_train[i])

    elif 0 < i < 23:

        osa_predictions.append(model.params[0] * Y_train[i] + model.params[1] * Y_train[i - 1])

    elif i == 23:

        osa_predictions.append(model.params[0] * Y_train[i] + model.params[1] * Y_train[i - 1] + model.params[2] * Y_train[i - 
23])
```

```
    elif i == 24:

        osa_predictions.append(model.params[0] * Y_train[i] + model.params[1] * Y_train[i - 1] + model.params[2] * Y_train[i - 
23] - model.params[0] * model.params[2] * Y_train[i - 24])

    else:

        osa_predictions.append(model.params[0] * Y_train[i] + model.params[1] * Y_train[i - 1] + model.params[2] * Y_train[i - 
23] - model.params[0] * model.params[2] * Y_train[i - 24] - model.params[1] * model.params[2] * Y_train[i - 25])



hsa_forecasts = []



for i in range(len(y_test)):

    if i == 0:

        hsa_forecasts.append(model.params[0] * Y_train[i] + model.params[1] * Y_train[i - 1] + model.params[2] * Y_train[i - 23] 
- model.params[0] * model.params[2] * Y_train[i - 24] - model.params[1] * model.params[2] * Y_train[i - 25])

    elif i == 1:

        hsa_forecasts.append(model.params[0] * osa_predictions[i - 1] + model.params[1] * Y_train[i - 1] + model.params[2] * 
Y_train[i - 22] - model.params[0] * model.params[2] * Y_train[i - 23] - model.params[1] * model.params[2] * Y_train[i - 24])

    elif 1 < i < 25:

        hsa_forecasts.append(model.params[0] * osa_predictions[i - 1] + model.params[1] * osa_predictions[i - 2] + 
model.params[2] * Y_train[i - (23 - i)] - model.params[0] * model.params[2] * Y_train[i - (24 - i)] - model.params[1] * 
model.params[2] * Y_train[i - (25 - i)])

    elif i == 25:

        hsa_forecasts.append(model.params[0] * osa_predictions[i - 1] + model.params[1] * osa_predictions[i - 2] + 
model.params[2] * osa_predictions[i - 25] - model.params[0] * model.params[2] * Y_train[i - (24 - i)] - model.params[1] * 
model.params[2] * Y_train[i - (25 - i)])

    elif i == 26:

        hsa_forecasts.append(model.params[0] * osa_predictions[i - 1] + model.params[1] * osa_predictions[i - 2] + 
model.params[2] * osa_predictions[i - 25] - model.params[0] * model.params[2] * osa_predictions[i - 26] - model.params[1] * 
model.params[2] * Y_train[i - 26])

    else:

        hsa_forecasts.append(model.params[0] * osa_predictions[i - 1] + model.params[1] * osa_predictions[i - 2] + 
model.params[2] * osa_predictions[i - 25] - model.params[0] * model.params[2] * osa_predictions[i - 26] - model.params[1] * 
model.params[2] * osa_predictions[i - 27])



# =======================================================================================================================

# Plotting the h-step ahead forecasts
```

```
osa_predictions = pd.Series(osa_predictions, index=Y_train.index)


hsa_forecasts = pd.Series(hsa_forecasts, index=y_test.index)


plt.figure()


plt.plot(Y_train, label='Training data')

plt.plot(y_test, label='Testing data')

plt.plot(osa_predictions, label='One-step ahead predictions')

plt.plot(hsa_forecasts, label='H-step ahead forecasts')


plt.title('Plot of OSA predictions - SARIMA (2, 0, 0) x (1, 0, 0, 24)')

plt.xlabel('Time (in hours)')

plt.ylabel('PM 2.5 Concentration (µg / cu m)')

plt.legend(loc='upper left')


plt.show()


# ========================================================================================================================

# Variance of the forecast errors


forecast_errors = y_test - hsa_forecasts


print('The variance of the forecast errors is', np.var(forecast_errors))
```