Multivariate Modeling (DATS 6450)

Term Project Report

Instructor: Dr. Reza Jafari
Student: Srijon Mukhopadhyay

Due Date: 22nd April, 2020

1. Abstract:

With numerous forecasting models and algorithms at hand, it is not enough just to apply models to the data. Every model has its own area of functionality where it performs best, and it is essential to know how a model performs to utilize it efficiently. With this in mind, I've decided to write my report on the comparison of performance of three different models that are extensively used when applied to climate data and see how they perform.

I've chosen a data set that contains the daily records of temperature statistics in Delhi from the years 2013 to 2017. Climate change is an important fear plaguing all of us and in the context of India, Delhi is one of the main areas of interest, with temperatures displaying more and more extreme behaviors of the years. I would like to analyze the climate records of Delhi and ascertain how its humidity has changed over the years, with increasing pollution.

2. Introduction:

The purpose of this project is to evaluate different models and forecasting methods and ascertain which one performs best. There are multiple steps necessary to first prepare the data for the application of models and then making forecasts. It is also important to look at the forecasts and calculate residuals. It is from the residual analysis that we understand how a model is behaving and what steps may be taken to improve its accuracy.

The following steps are followed in this report:

- The dataset in use is described and cleaned
- It is made sure that the dependent variable is stationary (if not, then transforming methods are used to make it stationary)
- The best time series decomposition is chosen
- Holt-Winters Method is applied on the dependent variable and forecasts are made
- Feature engineering is used to determine which features are most important for the model
- A multiple linear regression model is fit on the data
- An ARMA model representing the dataset is used
- The ARMA model parameters are estimated
- A final ARMA model is selected
- A comparison is made between the ARMA and Regression Model
- Predictions are made using the final model and plotted against the true values

3. Description of the Dataset:

I have used a dataset consisting of the daily climate observations in Delhi from 2013 to 2017.

It has 1575 observations and four columns - the Mean Temperature (‘meantemp’), Humidity (‘humidity), Wind Speed (‘wind_speed’) and the Mean Pressure (‘meanpressure’).

The dataset is for the daily climate observations, so each time step represents one day.

Humidity (‘humidity’) is my dependent variable in this data set.

Reading in the dataset and displaying the first ten rows:

	date	meantemp	humidity	wind_speed	meanpressure
0	2013-01-01	10.000000	84.500000	0.000000	1015.666667
1	2013-01-02	7.400000	92.000000	2.980000	1017.800000
2	2013-01-03	7.166667	87.000000	4.633333	1018.666667
3	2013-01-04	8.666667	71.333333	1.233333	1017.166667
4	2013-01-05	6.000000	86.833333	3.700000	1016.500000
5	2013-01-06	7.000000	82.800000	1.480000	1018.000000
6	2013-01-07	7.000000	78.600000	6.300000	1020.000000
7	2013-01-08	8.857143	63.714286	7.142857	1018.714286
8	2013-01-09	14.000000	51.250000	12.500000	1017.000000
9	2013-01-10	11.000000	62.000000	7.400000	1015.666667

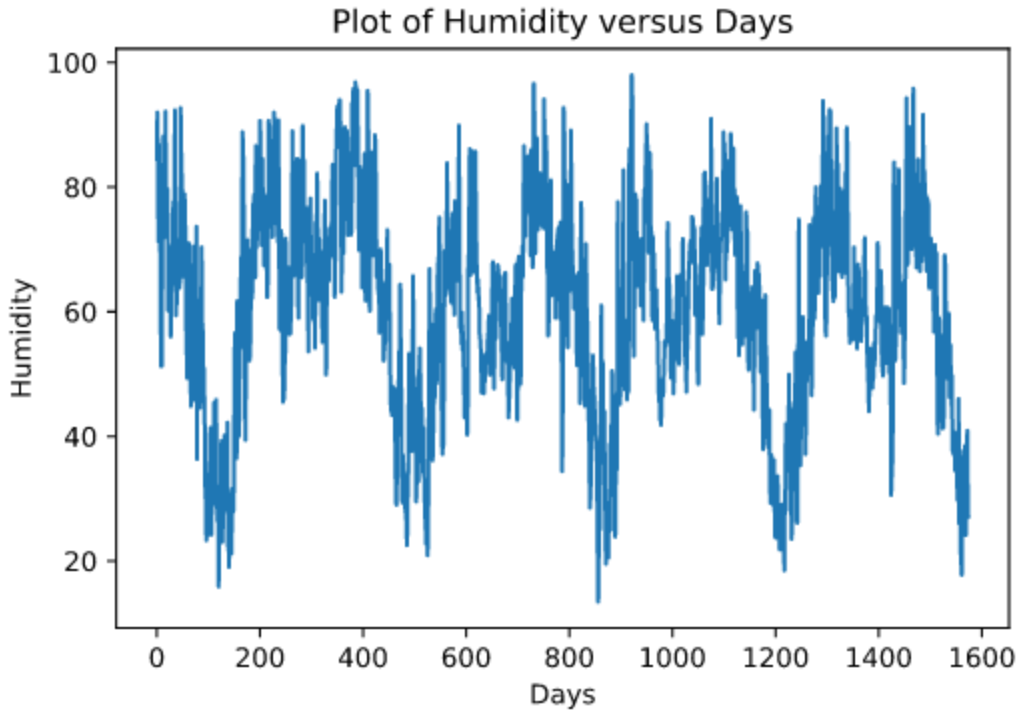
Displaying the last ten rows:

	date	meantemp	humidity	wind_speed	meanpressure
1565	2017-04-15	31.222222	30.444444	5.966667	1002.444444
1566	2017-04-16	31.000000	34.250000	2.100000	1003.250000
1567	2017-04-17	32.555556	38.444444	5.366667	1004.444444
1568	2017-04-18	34.000000	27.333333	7.811111	1003.111111
1569	2017-04-19	33.500000	24.125000	9.025000	1000.875000
1570	2017-04-20	34.500000	27.500000	5.562500	998.625000
1571	2017-04-21	34.250000	39.375000	6.962500	999.875000
1572	2017-04-22	32.900000	40.900000	8.890000	1001.600000
1573	2017-04-23	32.875000	27.500000	9.962500	1002.125000
1574	2017-04-24	32.000000	27.142857	12.157143	1004.142857

Checking for missing values and looking at the structure of the datatset:

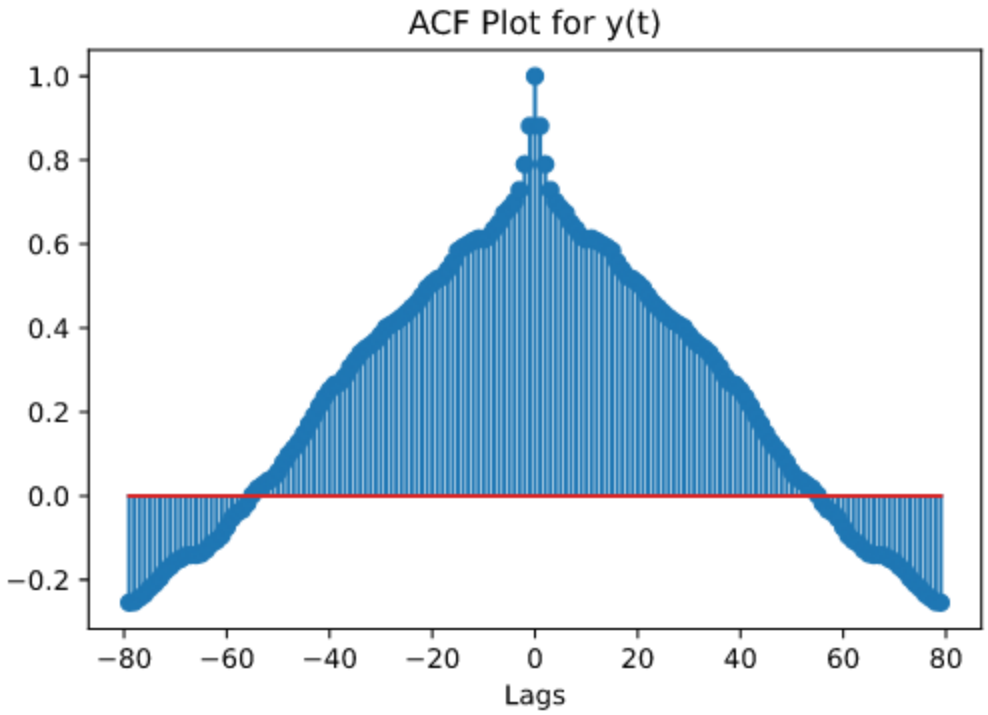
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1575 entries, 0 to 1574
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   date          1575 non-null   object 
 1   meantemp      1575 non-null   float64
 2   humidity      1575 non-null   float64
 3   wind_speed    1575 non-null   float64
 4   meanpressure  1575 non-null   float64
dtypes: float64(4), object(1)
memory usage: 61.6+ KB
```

Plotting the humidity versus the time:

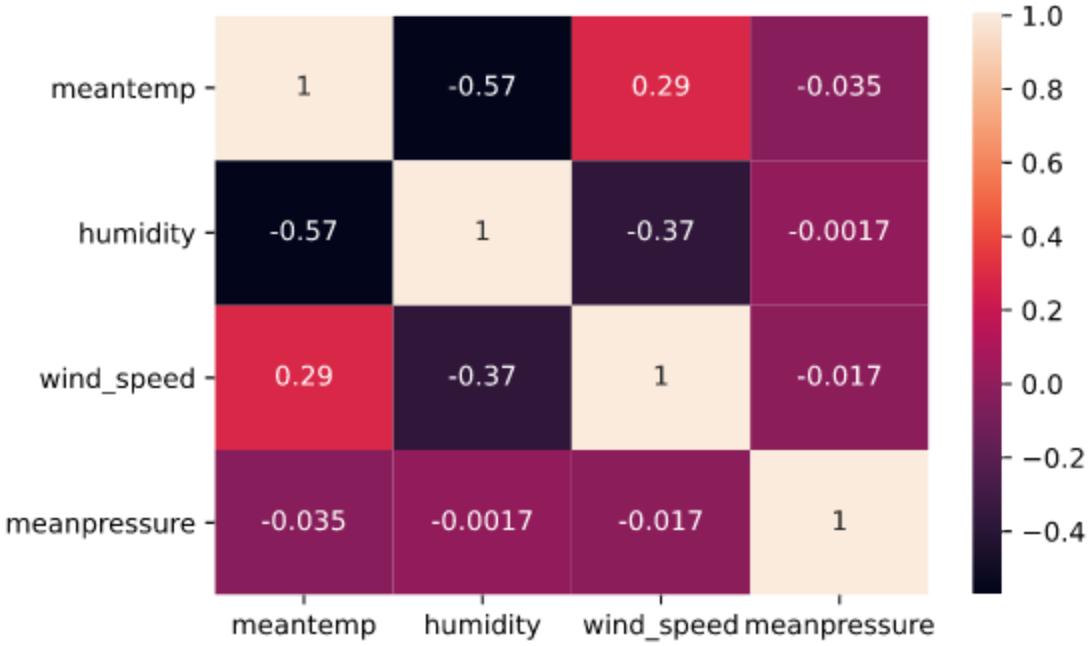


To the naked eye, this looks like a stationary plot. I'll have to perform an ADF test later on to be sure, though.

ACF plot for the dependent variable with 80 lags:



Correlation matrix using seaborn:



Here, I don't see any high values of correlations between my independent variables. They're average at best, for example, meantemp has a correlation of 0.29 with wind_speed, which is below average. Meanpressure has an almost negligible value of correlation with all the variables, including my dependent variable, so I have a feeling I'll have to remove that feature when I perform feature engineering. Let's see.

I can thus wager that multicollinearity will not be a problem with my regression model.

The dataset is split into training and test sets, with 80%-20% split.

4. Stationarity:

Performing the Augmented Dickey Fuller Test on the dependent variable to check for stationarity:

```
ADF Statistic: -3.638618

 p-value: 0.005064

 Critical Values:
     1%: -3.435
     5%: -2.863
     10%: -2.568
```

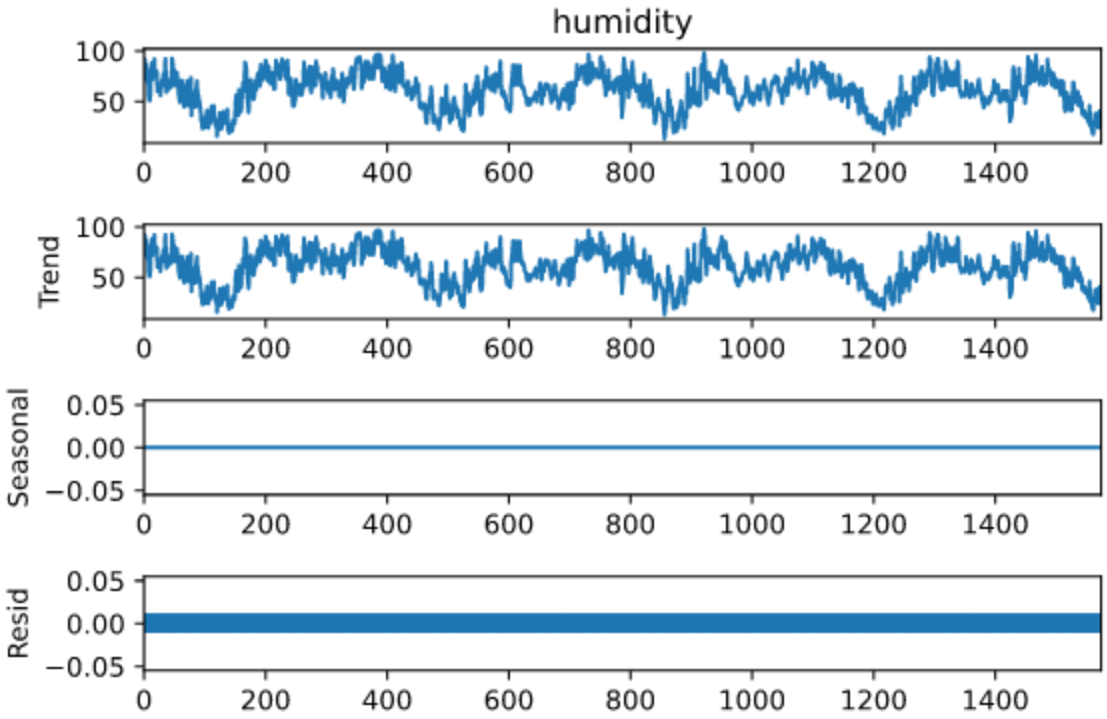
Here, the ADF test statistic is negative. This implies stationarity in the data.

The p-value is less than the 95% critical threshold of 0.05. I can now reject the null hypothesis that states that implies non stationarity in the data, in favor of

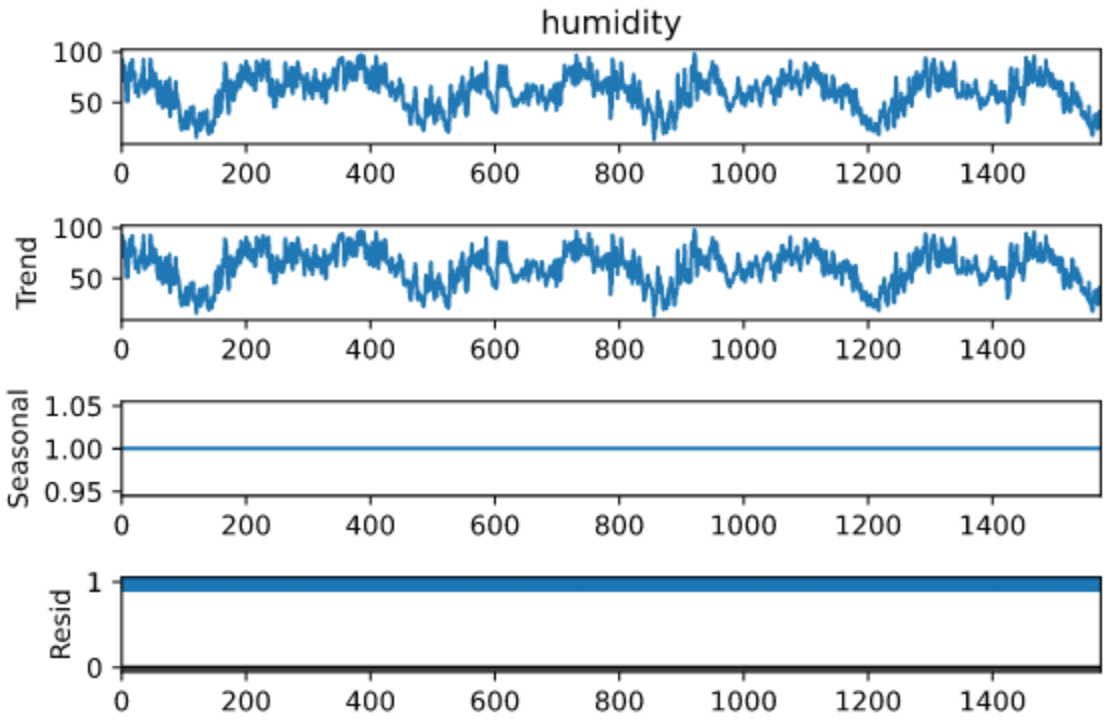
the alternative hypothesis. The dependent variable is stationary.

5. Time Series Decomposition:

Plotting the time series decomposition of the additive model:



Plotting the time series decomposition of the multiplicative model:

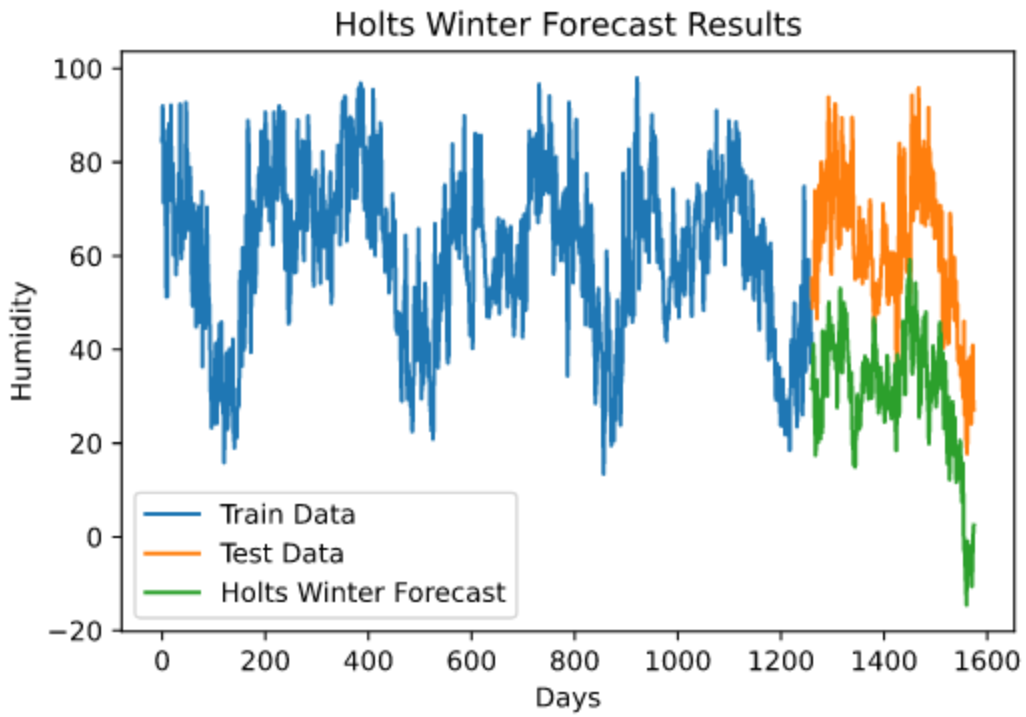


It is clear, looking at the seasonal components and residual variances that an additive model would best represent my data.

6. Holt-Winters Method:

I fit my Holts-Winter model using an additive model and specifying the seasonal trend.

Plotting the behavior:



The model is able to capture some of the seasonality. I would say this performs well. I'll look at the statistics of the residuals.

Mean:

```
The mean of the residuals for Holts Winter is:
30.5144712197015 34
```

The estimator is biased.

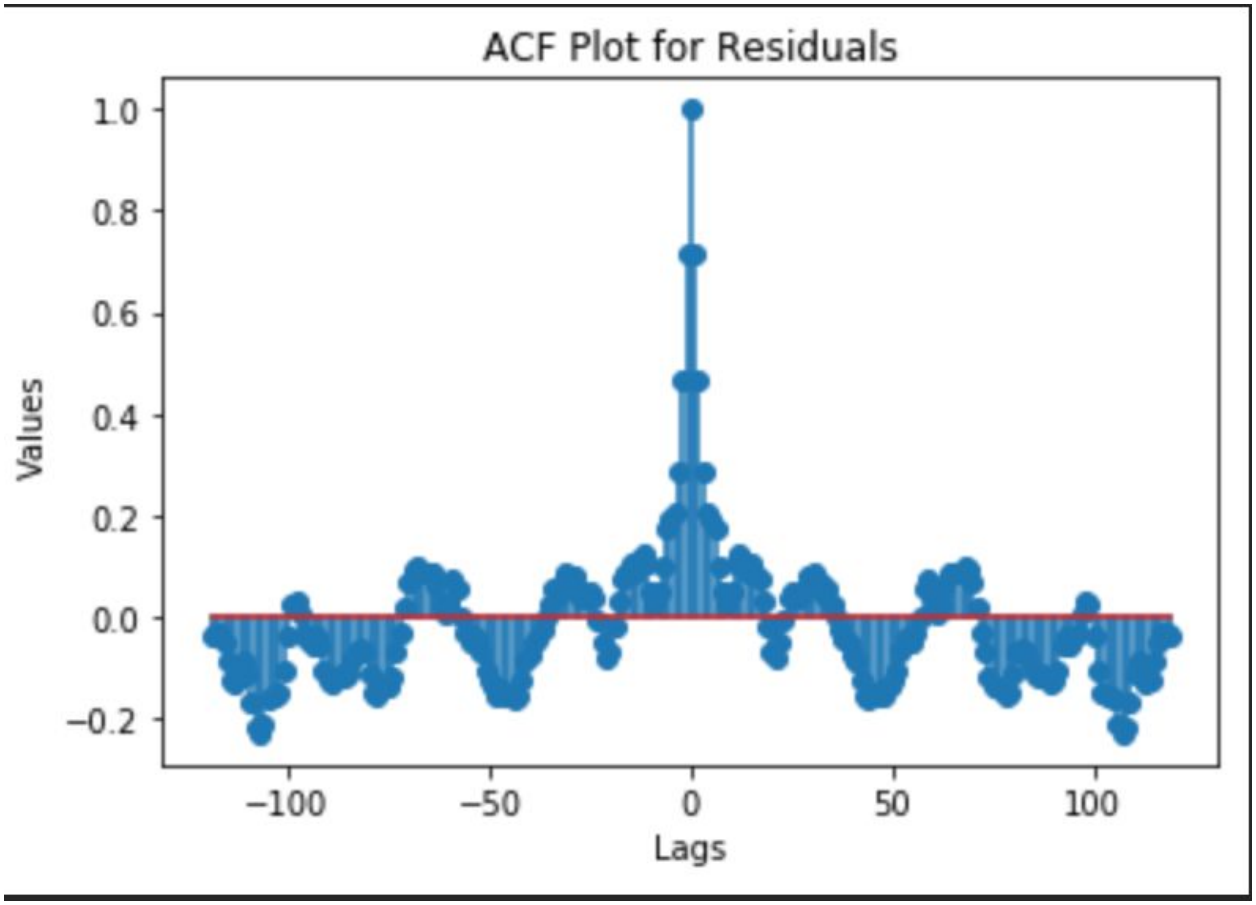
Variance of the residuals:

```
The variance of the residuals is: 168.9033662663944
```

RMSE of Holts Residuals:

```
The RMSE of the Holts-Winter model is: 33.16679544490824
```

ACF of the residuals:



7. Feature Selection:

I'm going to use the OLS function to run a fit on the model to see what coefficients I get and also check the t-test results. I will use this to get rid of

any features that do not contribute to the model, to reduce any chances of overfitting.

I've already seen earlier that there is no multicollinearity among my independent variables so that's great.

Running the OLS function and finding my first fit of the regression model:

OLS Regression Results

Dep. Variable:	humidity	**R-squared:**	0.429
Model:	OLS	**Adj. R-squared:**	0.428
Method:	Least Squares	**F-statistic:**	314.9
Date:	Tue, 21 Apr 2020	**Prob (F-statistic):**	2.00e-152
Time:	13:10:03	**Log-Likelihood:**	-5025.6
No. Observations:	1260	**AIC:**	1.006e+04
Df Residuals:	1256	**BIC:**	1.008e+04
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	99.4624	2.407	41.319	0.000	94.740	104.185
meantemp	-1.3025	0.052	-25.180	0.000	-1.404	-1.201
wind_speed	-0.7221	0.083	-8.705	0.000	-0.885	-0.559
meanpressure	-0.0016	0.002	-0.809	0.419	-0.005	0.002

So all my independent variables, besides meanpressure, have good p-values above the 95% critical threshold value of 0.05. This means that I can reject the null hypothesis that the coefficients are all equal to zero.

The independent variable ‘meanpressure’, however, has a p-value higher than the critical threshold of 0.05. I will thus fail to reject the null hypothesis that the coefficients are equal to zero.

As I suspected earlier,I will need to run another fit of the OLS function with the ‘meanpressure’ feature removed.

8. Developing the Regression model:

I’ll run another OLS fit on my data with the meanpressure variable removed:

OLS Regression Results

Dep. Variable:	humidity	**R-squared:**	0.429
Model:	OLS	**Adj. R-squared:**	0.428
Method:	Least Squares	**F-statistic:**	472.2
Date:	Tue, 21 Apr 2020	**Prob (F-statistic):**	1.13e-153
Time:	13:15:04	**Log-Likelihood:**	-5026.0
No. Observations:	1260	**AIC:**	1.006e+04
Df Residuals:	1257	**BIC:**	1.007e+04
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	97.8212	1.295	75.534	0.000	95.280	100.362
meantemp	-1.3013	0.052	-25.171	0.000	-1.403	-1.200
wind_speed	-0.7219	0.083	-8.703	0.000	-0.885	-0.559

With the meanpressure variable removed, all my independent variables now have p-values below the 95% critical threshold value of 0.05. This means that I can reject the null hypothesis that the coefficients are all equal to zero. My model thus has significant coefficients.

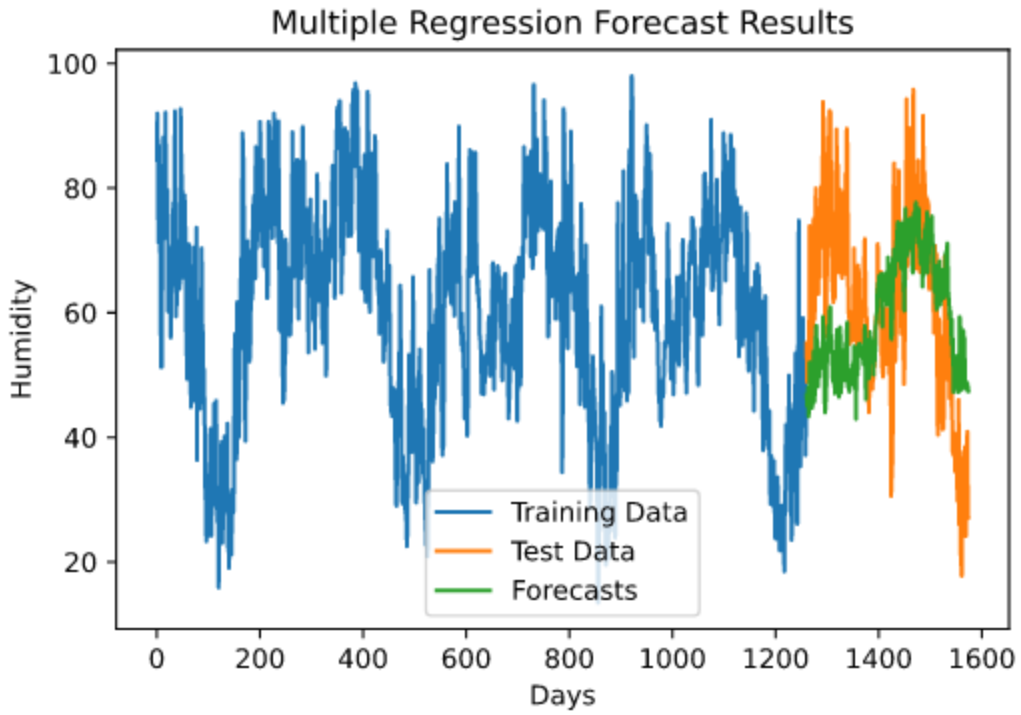
The p-value of the F statistic is much lower than the 95% critical threshold value, implying that I can reject the null hypothesis that states that an intercept only model would provide the same output that my current model with its parameters is generating.

The R-squared and the Adjusted R-squared value is lower than 60%. This is not a very good score and doesn't account for a lot of the variability in the data. I'll need to check and analyze the linear relationship between the forecasts and test values.

The AIC and BIC values seem to be quite low. AIC values represent the amount of information lost by my model, so a negligible AIC implies that my regression model is able to capture all of the information in my data. This is good.

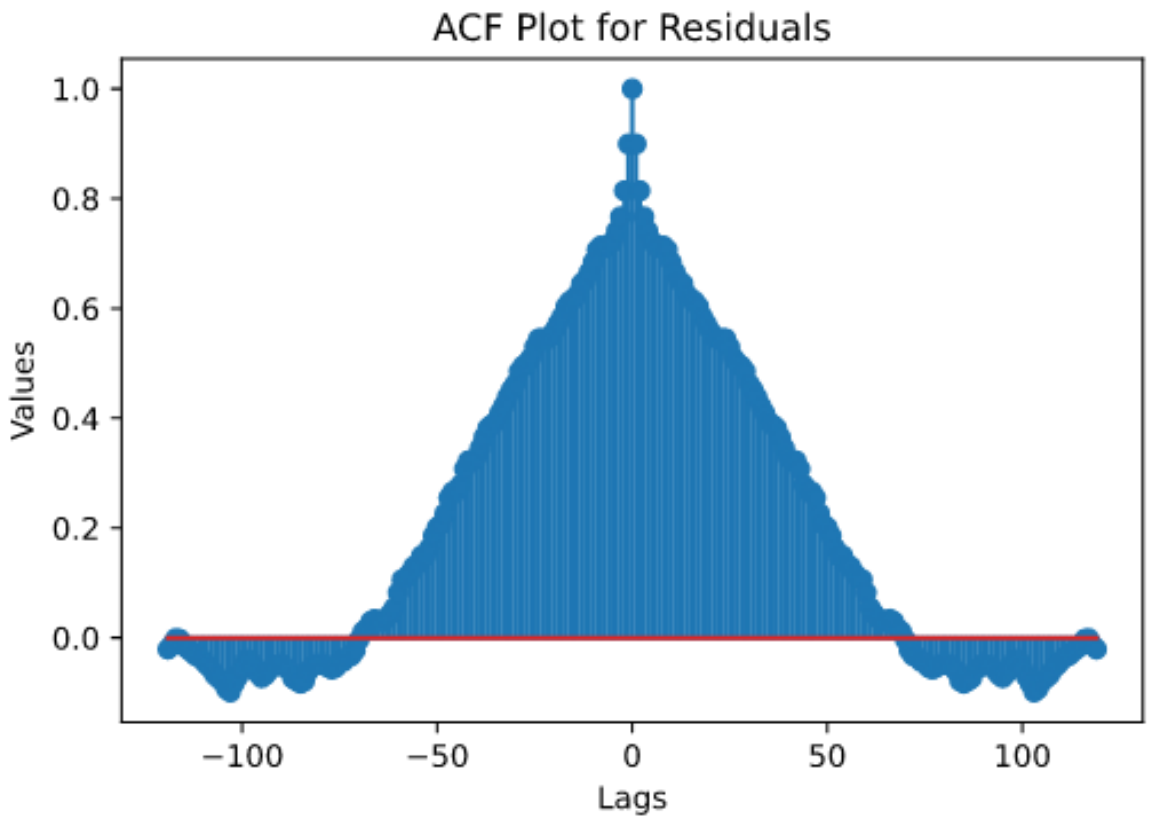
I'll use my model to make forecasts on the test data so I can compare the residuals.

Looking at how the forecasts did:



The forecasts seem to do better than the results of the Holt Winters model. It captures the seasonality better. I'll look at some of the statistics of the residuals.

Plotting the ACF plot for the residuals over 120 lags:



Displaying the Q-value of the residuals:

```
The Q value of the residuals is: 2299.806634822249
```

Printing the variance of the residuals:

```
The variance of the residuals is: 214.64387403927174
```

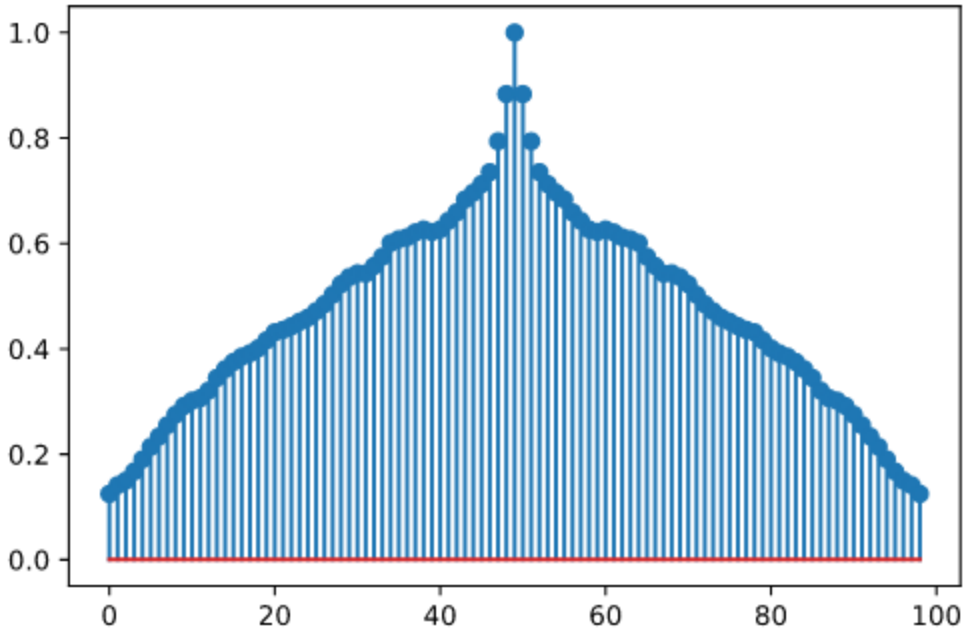
Printing the mean of the residuals:

```
The mean of the residuals is: 2.0783728373105155
```

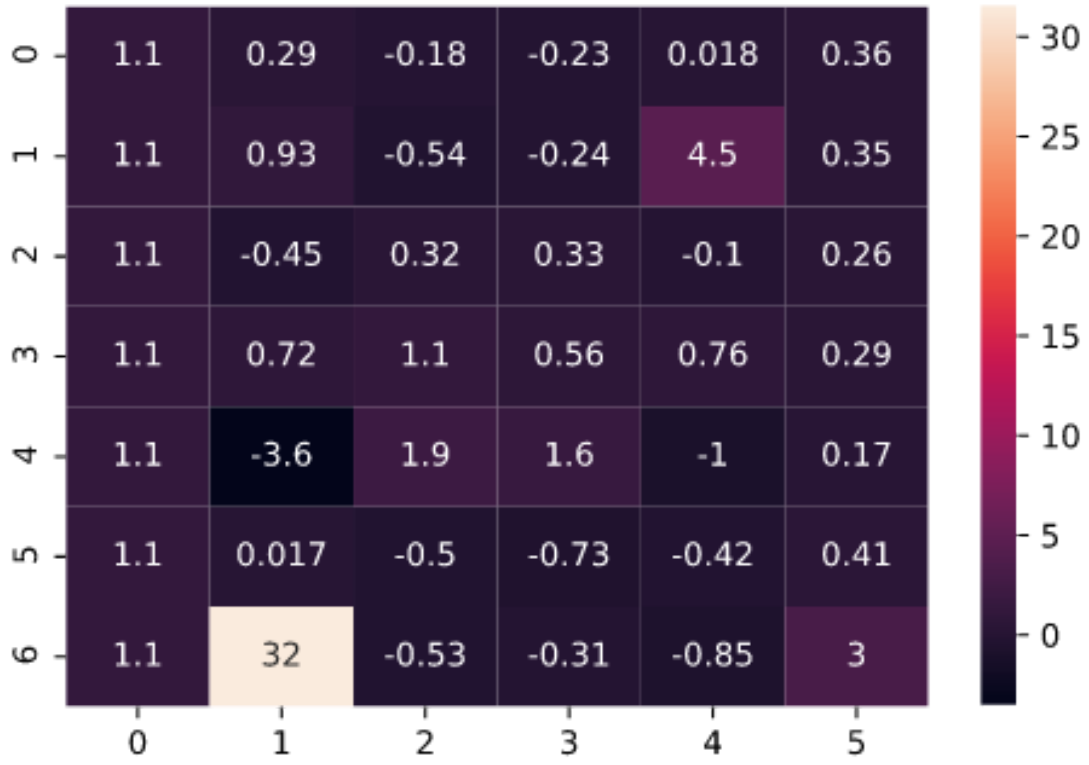
Since the mean of the residuals is greater than 0, the residuals are thus not white. This implies a biased estimator.

9. ARMA Model Determination:

The theoretical acf of the train set is found and then plotted below, for 50 lags:



Implementing a GPAC table from this autocorrelation function which I can pick two orders to work with:



I'll estimate two models to best represent this dataset: ARMA(1,0) model, with n_a of 1 and no n_b and ARMA (1,5) , with n_a of 1 and n_b of 5.

I'll import and run the statsmodel package to generate the model. The model generated will estimate the parameter.

10. Estimating the ARMA Model Parameter:

Working with ARMA(1,0):

The estimated parameter:

```
The AR coefficient a0 is 0.9912973690557185
```

Running the fit:

```
                            ARMA Model Results
==============================================================================
Dep. Variable:               humidity   No. Observations:                 1260
Model:                     ARMA(1, 0)   Log Likelihood               -4455.612
Method:                       css-mle   S.D. of innovations              8.295
Date:                Tue, 21 Apr 2020   AIC                           8915.225
Time:                        19:54:59   BIC                           8925.503
Sample:                             0   HQIC                          8919.087

==============================================================================
                     coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1.humidity     0.9913      0.004    277.365      0.000       0.984       0.998
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.0088           +0.0000j            1.0088            0.0000
-----------------------------------------------------------------------------
```

11. Model Selection:

Looking at the confidence intervals of my ARMA(1,0):

	0	1
ar.L1.humidity	0.984292	0.998302

So the confidence intervals don’t include 0. This is great, it means the parameters estimated are significant.

Estimated variance of the error:

```
The estimated variance of the residuals for ARMA(1,0) model
is: 252.05360081594355
```

Covariance matrix:

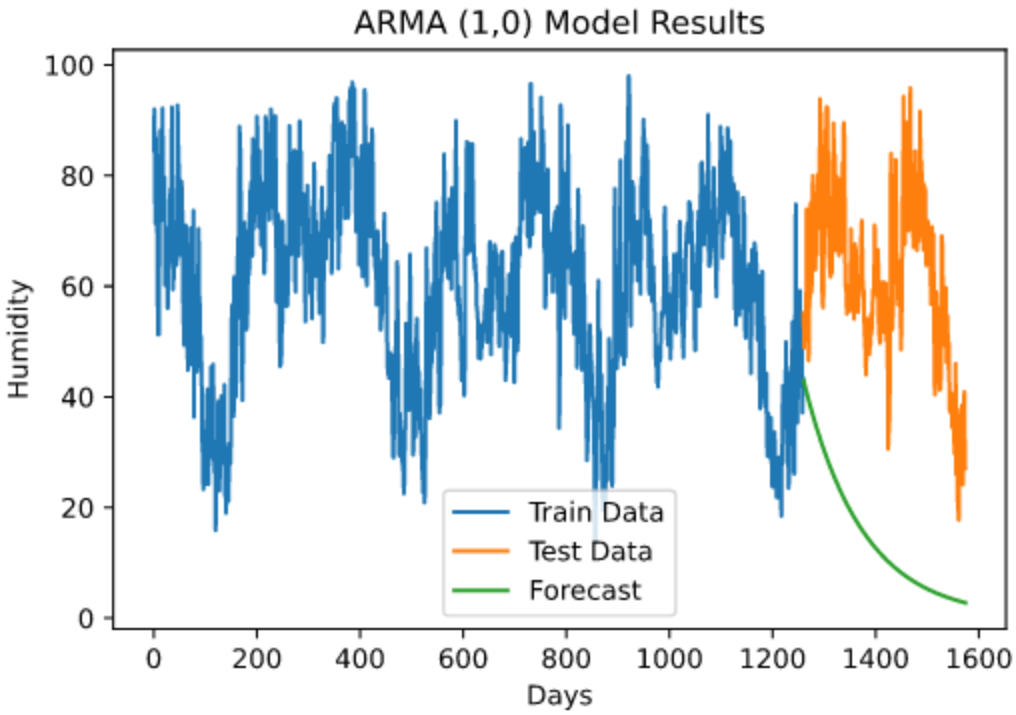
	ar.L1.humidity
ar.L1.humidity	0.000013

Checking the Chi - Square Diagnostic test:

```
Residuals are not white
```

Since the residuals are not white, it means this is a biased estimator as well! I'm going to try and see if my other ARMA model fares better.

Plot of the forecasts:



The model struggles to represent the stationarity of the model.

Repeating the same process for ARMA(2,5):

Estimating the parameters:

```
The AR coefficient a0 is 0.2845733790072036
The MA coefficient b0 is 0.49607898319740296
The MA coefficient b1 is -0.3554872101041734
The MA coefficient b2 is -0.322675563866437
The MA coefficient b3 is -0.17673904297403314
The MA coefficient b4 is -0.07858453023975867
The AR coefficient a1 is 0.7140088146320064
```

Running the fit:

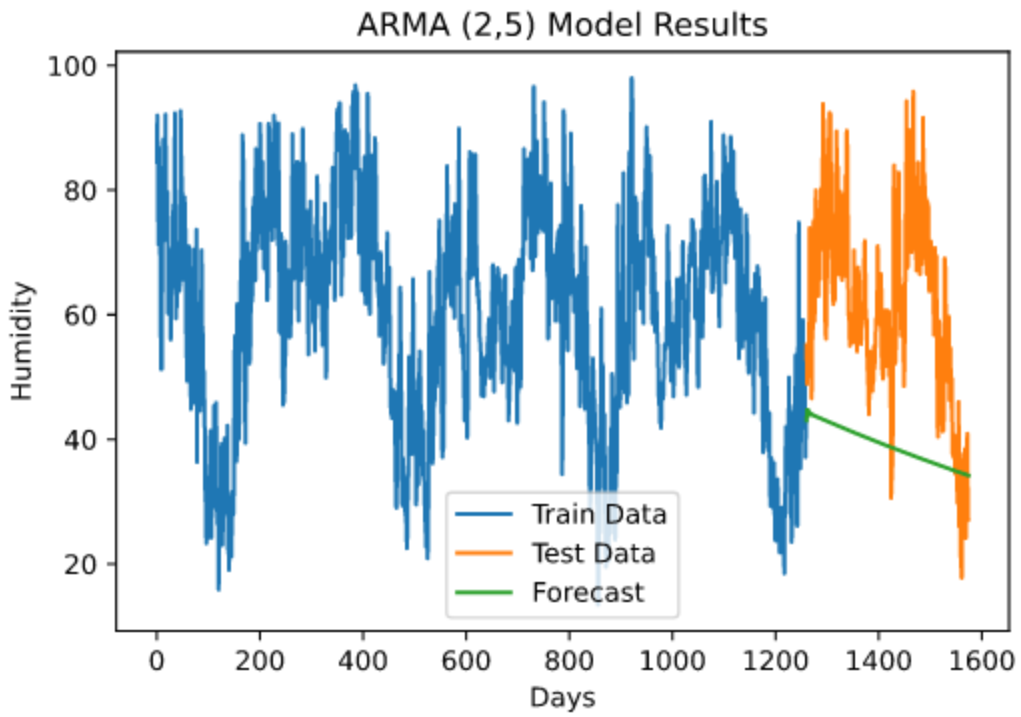
```
                            ARMA Model Results
==============================================================================
Dep. Variable:               humidity   No. Observations:                 1260
Model:                     ARMA(1, 5)   Log Likelihood               -4384.940
Method:                       css-mle   S.D. of innovations              7.841
Date:                Tue, 21 Apr 2020   AIC                           8783.879
Time:                        20:10:18   BIC                           8819.851
Sample:                             0   HQIC                          8797.397

=======================================================================================
                      coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------------
ar.L1.humidity     0.9992      0.001   1186.726      0.000       0.998       1.001
ma.L1.humidity    -0.2212      0.028     -7.814      0.000      -0.277      -0.166
ma.L2.humidity    -0.1988      0.029     -6.873      0.000      -0.255      -0.142
ma.L3.humidity    -0.1783      0.028     -6.295      0.000      -0.234      -0.123
ma.L4.humidity    -0.0491      0.028     -1.737      0.082      -0.105       0.006
ma.L5.humidity    -0.0260      0.028     -0.936      0.349      -0.080       0.028
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.0008           +0.0000j            1.0008            0.0000
MA.1            1.1872           -0.0000j            1.1872           -0.0000
MA.2           -1.6809           -1.4930j            2.2482           -0.3844
MA.3           -1.6809           +1.4930j            2.2482            0.3844
MA.4            0.1412           -2.5293j            2.5332           -0.2411
MA.5            0.1412           +2.5293j            2.5332            0.2411
-----------------------------------------------------------------------------
```

Looking at the confidence intervals for my ARMA(2,5):

	0	1
ar.L1.humidity	−0.038337	0.607484
ar.L2.humidity	0.391407	1.036611
ma.L1.humidity	0.171341	0.820817
ma.L2.humidity	−0.449552	−0.261423
ma.L3.humidity	−0.415580	−0.229771
ma.L4.humidity	−0.264757	−0.088721
ma.L5.humidity	−0.134524	−0.022645

I’m not happy with these intervals. I can see for the first AR component that the confidence intervals include zero. This means the estimated parameter is not significant.

Looking at a plot of the forecast:



This seems to do better than the ARMA(1,0) model but still isn't able to capture the seasonality of the model.

Mean of the residuals:

```
The residual mean of the ARMA(2,5) model is:
22.27887229807198
```

The mean implies that my model is a biased estimator.

Variance:

```
Estimated variance of residuals of ARMA (2,5) is:
210.94756243531518
```

RMSE:

```
The RMSE of the residuals: 26.595031741081165
```

Chi-Square Test:

```
Residuals are not white
```

Unfortunately both my models fail the Chi Square Diagnostic Test.

I took a closer look at the GPAC table and tried all the combinations and noticed that none of them passed the Chi square test. I also tried subtracting the mean and fitting the model and adding the mean back to the predictions to relax the Q score. Relaxing the constraints for my model did not improve the results of the diagnostic test either.

12. Final Model Selection:

	Residual Mean	Residual Variance	RMSE
Holts Winter	30.514388	168.903787	33.166725
Multiple Regression	2.078373	214.643874	14.797416
ARMA (1,0)	46.664282	252.053601	49.291063
ARMA(2,5)	22.278894	210.947494	26.595048

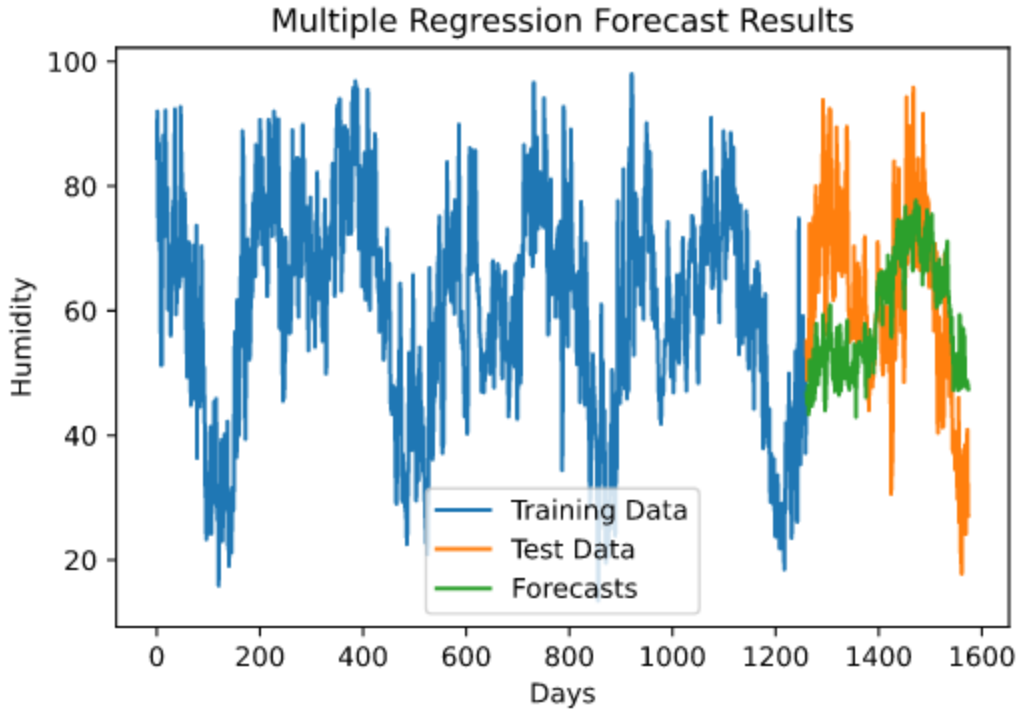
Looking at the table above that reflects how my models performed, I would choose the Multiple Regression model for the following reasons:

- While it's a biased estimator, it is able to capture the seasonality of the data better than the Holts Winter and ARMA models.
- The residuals of the regression model have a much lower mean than the residual means of the Holts Winter residuals

- The Regression model has a much lower value of RMSE than the other two models

13. Predictions:

Here's another look at how the OLS function did with the predictions:



The model captures some of the seasonality in the components. It does better than the other models, as we saw earlier.

14. Summary and Conclusions:

- One reason for the ARMA models not passing the chi square test could be the ARMA model lacking an intercept

- All three models struggled a bit, most probably due to the nature of the data. My data records daily observations for a long period of time (four years) and thus involve multiple seasonal patterns and complex seasonality

- A way to fix this and may be extend this project could be to implement dynamic harmonic regression algorithms that may be able to account for the complex seasonality of the model

- The regression model would definitely do better with more features. It may be suffering from underfitting

15. Appendix:

Main code:

```
from config import *
import statsmodels.tsa.holtwinters as ets
from statsmodels.tsa.seasonal import seasonal_decompose
from scipy.stats import chi2

#Loading the data from my dataset
delhi_climate = pd.read_csv("daily_climate_delhi.csv")

#Displaying head:
delhi_climate.head(10)

#Displaying tail:
delhi_climate.tail(10)

#Checking for NAN values:
delhi_climate.info()
```

```
#3.a Plot of the dependent variable versus time
plt.plot(delhi_climate.humidity)
plt.ylabel("Humidity")
plt.xlabel("Days")
plt.title("Plot of Humidity versus Days")
plt.show()

#3.b ACF of the dependent variable:
acf_plot(80,delhi_climate.humidity)


#3.c Correlation Matrix with SNS

corr = delhi_climate.corr()
print(corr)
sns.heatmap(corr, annot = True)


#3.e Splitting data set into training and test data sets

X = delhi_climate[['meantemp', 'wind_speed', 'meanpressure']]
Y = delhi_climate['humidity']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size = 0.8, test_size = 0.2,
shuffle = False)



#4. Stationarity:
#ADF Stationarity Test


adf_test(delhi_climate.humidity)

# Decomposing the data with additive model

results_additive = seasonal_decompose(delhi_climate.humidity, model = "additive", freq = 1)
results_additive.plot()

plt.show()

#Decomposing data with multiplicative model:
results_multiplicative = seasonal_decompose(delhi_climate.humidity, model =
"multiplicative", freq = 1)
results_multiplicative.plot()

plt.show()

#6 Holt-Winters Method
holt_winters = ets.ExponentialSmoothing(Y_train, trend= None, seasonal='add',
```

```
seasonal_periods=365).fit()
holt_winters_forecast = holt_winters.predict(start = 1260, end = 1574)

#Displaying the forecasts
display(holt_winters_forecast)



#Residuals for Holt Winters
residuals_holts = Y_test - holt_winters_forecast

holts_res_mean = np.mean(residuals_holts)
holts_res_mean
print("The mean of the residuals for Holts Winter is: {}".format(holts_res_mean))
#ACF of Holts Winter forecasts

acf_plot_residuals(120, residuals_holts)


#Variance of Residuals:
print("The variance of the residuals is: {}".format(np.var(residuals_holts)))
holts_var = np.var(residuals_holts)

#RMSE for Holts Winter
holts_rmse = np.sqrt(np.mean(residuals_holts ** 2 ))
holts_rmse
print("The RMSE of the Holts-Winter model is: {}".format(holts_rmse))


#Plotting Holts Winter Forecasts
plt.plot(Y_train, label = 'Train Data')
plt.plot(Y_test, label = 'Test Data')
plt.plot(holt_winters_forecast, label = 'Holts Winter Forecast')
plt.xlabel("Days")
plt.ylabel("Humidity")
plt.title("Holts Winter Forecast Results")
plt.legend()
plt.show()




#8 Multiple Linear Regression Model

X_train_fit = sm.add_constant(X_train)
multiple_regression_first_fit = sm.OLS(Y_train, X_train_fit).fit()
multiple_regression_first_fit.summary()
```

```
#Removing the meanpressure feature and running another fit:

X_removed_feature = X_train_fit.drop(['meanpressure'], axis = 1)
multiple_regression_second_fit = sm.OLS(Y_train, X_removed_feature).fit()
multiple_regression_second_fit.summary()


#Predictions on OLS model
X_test_fit = sm.add_constant(X_test).drop(['meanpressure'], axis = 1)


y_hat_OLS = multiple_regression_second_fit.predict(X_test_fit)


#Plotting the forecasts
plt.plot(Y_train, label = 'Training Data')
plt.plot(Y_test, label = 'Test Data')
plt.plot(y_hat_OLS, label = 'Forecasts')
plt.legend()
plt.xlabel("Days")
plt.ylabel("Humidity")
plt.title("Multiple Regression Forecast Results")
plt.show()


#Residuals

residuals_OLS = Y_test - y_hat_OLS


#RMSE of Residuals from OLS

np.sqrt(np.mean(residuals_OLS ** 2))
ols_rmse = np.sqrt(np.mean(residuals_OLS ** 2))
#ACF of Residuals from OLS model

acf_plot_residuals(120, list(residuals_OLS))

#Q value of Residuals:
print("The Q value of the residuals is: {}".format(find_Q_value(Y_test, y_hat_OLS, 15)))

#Variance of Residuals
print("The variance of the residuals is: {}".format(np.var(residuals_OLS)))

ols_var = np.var(residuals_OLS)
#Mean of OLS Residuals
print("The mean of the residuals is: {}" .format(np.mean(residuals_OLS)))
ols_res_mean = np.mean(residuals_OLS)
```

```
#ARMA Model Determination

ry = theoretical_acf(50, Y_train)
ry1 = ry[::-1]
ry2 = np.concatenate((np.reshape(ry1,50), ry[1:]))

#AutoCorrelation plot for Theoretical ACF
plt.stem(ry2)
plt.show()

#gpac table
sns.heatmap(gpac(ry2,7,7), annot = True)


#Trying ARMA(1,0):
#Trying order with na = 1, nb =0:
#ARMA Parameter Estimation:

model_arma_10 = arma_model_parameter_estimation(Y_train,1,0)


#Confidence Intervals for ARMA(1,0):
model_arma_10.conf_int()

#Estimated Covariance:
model_arma_10.cov_params()

#Prediction
y_hat_arma10 = model_arma_10.predict(start = 1260, end  = 1574)

residuals_arma10 = Y_test - y_hat_arma10

#Plot of forecast
plt.plot(Y_train,label = 'Train Data')
plt.plot(Y_test, label = 'Test Data')
plt.plot(y_hat_arma10, label = 'Forecast')
plt.title("ARMA (1,0) Model Results")
plt.xlabel("Days")
plt.ylabel("Humidity")
plt.legend()
plt.show()


#Estimated Variance:
print("The estimated variance of the residuals for ARMA(1,0) model is:
{}".format(np.var(Y_test - y_hat_arma10)))
arma10_var = np.var(residuals_arma10)

#Mean of Residuals:
arma10_mean = np.mean(residuals_arma10)
```

```
arma10_mean
print("Mean of residuals for ARMA(1,0) model: {}".format(arma10_mean))


#ACF
acf_plot_residuals(120, residuals_arma10)


#Chi_Square Test
Q10 = find_Q_value(Y_test, y_hat_arma10, 20)
Q10
dof = 20 - 1 - 0
alpha = 0.01
if Q10 < chi2.ppf(0.99, dof):
   print("Residuals are white")
else:
   print("Residuals are not white")



residuals_arma10 = Y_test - y_hat_arma10

#RMSE of ARMA(1,0):
np.sqrt(np.mean(residuals_arma10 ** 2))

arma10_rmse = np.sqrt(np.mean(residuals_arma10 ** 2))
#Trying ARMA(2,5):
#Trying order with na = 2, nb =5:
#ARMA Parameter Estimation:

model_arma_25 = arma_model_parameter_estimation(Y_train,2,5)

#Confidence intervals ARMA(1,5):
model_arma_25.conf_int()

#Covariance parameters
model_arma_25.cov_params()

#Forecasts
y_hat_arma25 = model_arma_25.predict(start = 1260, end  = 1574)

residuals_arma25 = Y_test - y_hat_arma25


#Plot of forecast
plt.plot(Y_train,label = 'Train Data')
plt.plot(Y_test, label = 'Test Data')
plt.plot(y_hat_arma25, label = 'Forecast')
plt.title("ARMA (2,5) Model Results")
plt.xlabel("Days")
```

```
plt.ylabel("Humidity")
plt.legend()
plt.show()

#Mean of Residuals:
arma25_mean = np.mean(residuals_arma25)
arma25_mean
print("The residual mean of the ARMA(2,5) model is: {}".format(arma25_mean))
#Estimated Variance

arma25_var = np.var(residuals_arma25)
arma25_var
print("Estimated variance of residuals of ARMA (2,5) is: {}".format(arma25_var))

#ACF
acf_plot_residuals(120, residuals_arma25)


#RMSE residuals
arma25_rmse = np.sqrt(np.mean(residuals_arma25 ** 2))
arma25_rmse
print("The RMSE of the residuals: {}".format(arma25_rmse))

#Chi_Square Test
Q25 = find_Q_value(Y_test, y_hat_arma25, 20)
dof = 20 - 2 - 5
alpha = 0.01
if Q25 < chi2.ppf(0.99, dof):
   print("Residuals are white")
else:
   print("Residuals are not white")


#Trying after subtracting mean (ARMA(1,0)):
Y_subtracted_mean = np.subtract(Y_train, np.mean(Y_train))

model_mean_10 = arma_model_parameter_estimation(Y_subtracted_mean,1,0)

#confidence intervals:
model_mean_10.conf_int()

#forecasts:
y_mean_10 = model_mean_10.predict(start = 1260, end = 1574)
y_mean_10 = np.add(y_mean_10, np.mean(Y_train))

plt.plot(Y_train,label = 'Train Data')
plt.plot(Y_test, label = 'Test Data')
plt.plot(y_mean_10, label = 'Forecast')
plt.title("Results of Subtracted Mean Predictions")
plt.xlabel("Days")
```

```
plt.ylabel("Humidity")
plt.legend()
plt.show()


#Chi Square
#Chi_Square Test
Q_mean_10 = find_Q_value(Y_test, y_mean_10, 20)
dof = 20 - 1 - 0
alpha = 0.01
if Q_mean_10 < chi2.ppf(0.99, dof):
   print("Residuals are white")
else:
   print("Residuals are not white")


#Trying after subtracting mean (ARMA(2,5)):

model_mean_25 = arma_model_parameter_estimation(Y_subtracted_mean, 2, 5)

#Confidence Intervals
model_mean_25.conf_int()

#forecasts:
y_mean_25 = model_mean_25.predict(start = 1260, end = 1574)
y_mean_25 = np.add(y_mean_25, np.mean(Y_train))

plt.plot(Y_train,label = 'Train Data')
plt.plot(Y_test, label = 'Test Data')
plt.plot(y_mean_25, label = 'Forecast')
plt.title("Results of Subtracted Mean Predictions")
plt.xlabel("Days")
plt.ylabel("Humidity")
plt.legend()
plt.show()

#Chi_Square Test
Q_mean_25 = find_Q_value(Y_test, y_mean_25, 20)
dof = 20 - 1 - 0
alpha = 0.01
if Q_mean_10 < chi2.ppf(0.99, dof):
   print("Residuals are white")
else:
   print("Residuals are not white")




Table = pd.DataFrame(data = {
```

```
  'Residual Mean': [holts_res_mean,ols_res_mean,arma10_mean,arma25_mean],
  'Residual Variance': [holts_var,ols_var,arma10_var,arma25_var],
  'RMSE':[holts_rmse,ols_rmse,arma10_rmse,arma25_rmse]



}, index = ['Holts Winter', 'Multiple Regression', 'ARMA (1,0)', 'ARMA(2,5)']
)

display(Table)
```

Background code containing packages and functions:

```
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from scipy import signal
from sklearn.model_selection import train_test_split
from numpy.linalg import inv
import pandas as pd
import seaborn as sns
import statsmodels.api as sm
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.api import ExponentialSmoothing, SimpleExpSmoothing, Holt

#ADF test
def adf_test(x):
   adf_list = adfuller(x)
   print('\n ADF Statistic: %f' % adf_list[0])
   print('\n p-value: %f' % adf_list[1])
   print('\n Critical Values:')
   for key, value in adf_list[4].items():
       print('\t%s: %.3f' % (key, value))




#Correlation Coefficient
def correlation_coefficient_cal(x, y):
   num = 0
   denom_x_sq = 0
   denom_y_sq = 0

   for i in range(0, len(x)):
       a =  (x[i] - np.mean(x)) * (y[i] - np.mean(y))
       num += a

   for i in range(len(x)):
```

```
            b = np.square(x[i] - np.mean(x))
            denom_x_sq += b
            c = np.square(y[i] - np.mean(y))
            denom_y_sq += c
    denom = np.sqrt(denom_x_sq) * np.sqrt(denom_y_sq)
    r = num / denom
    return r

#Partial Correlation
def partial_correlation(x, y, confound):
    num = correlation_coefficient_cal(x, y)  - (correlation_coefficient_cal(x, confound) *
correlation_coefficient_cal(y, confound))
    den = np.sqrt(1 - np.square(correlation_coefficient_cal(x, confound))) * np.sqrt(1 -
np.square(  correlation_coefficient_cal (y, confound))  )
    return num/den

#Partial Correlation Hypothesis Test
def hypothesis_test(number_samples, number_confounding, partial_correlation_value):
    alpha = 0.05
    degrees_freedom = number_samples - 2 - number_confounding
    critical_t_value = t.ppf(1-alpha, df = degrees_freedom)
    t_correlation = np.abs(partial_correlation_value * np.sqrt( degrees_freedom / (1 -
np.square(  partial_correlation_value)  )))

    if t_correlation > critical_t_value:
        print("The t value for the hypothesis test is: {}".format(t_correlation))
        print("The t vale at the critical threshold of 0.05 is: {}".format(critical_t_value))
        print("Thus the correlation is statistically significant")
    else:
        print("The t value for the hypothesis test is: {}".format(t_correlation))
        print("The t vale at the critical threshold of 0.05 is: {}".format(critical_t_value))
        print("Thus the correlation is not significant")

    return t_correlation , critical_t_value

#Auto Correlation
def auto_correlation(x, k):
    num = 0
    denom = 0
    if k > 0:
        for i in range(k, len(x)):
            a = (x[i] - np.mean(x)) * (x[i-k] - np.mean(x))
            num += a
        for i in range(0, len(x)):
            a = np.square(x[i] - np.mean(x))
            denom += a
    if k < 0:
        k = k * -1
        for i in range(k, len(x)):
```

```
            a = (x[i] - np.mean(x)) * (x[i-k] - np.mean(x))
            num += a
        for i in range(0, len(x)):
            a = np.square(x[i] - np.mean(x))
            denom += a
    elif k == 0:
        return 1
    acr = num/denom
    return acr

#Auto Correlation Plot

def acf_plot(number_lags, array_to_plot):
    lags = np.arange(-1 * (number_lags - 1), number_lags, 1)
    acf_list = []
    for i in lags:
        acf_list.append(auto_correlation(array_to_plot, i))
    plt.stem(lags, acf_list)
    plt.title("ACF Plot for y(t)")
    plt.xlabel("Lags")
    plt.show()

#Plot for acf
def theoretical_acf(lags, x):
    acf_list = []
    for i in range(lags):
        acf_list.append(auto_correlation(x,i))
    return acf_list
#GPac table
def gpac(theoretical_acf, j_value, k_value):
    gpac = np.zeros((j_value,k_value-1))
    for k in range(1,k_value):
        for j in range(0,j_value):
            den = np.zeros((k,k))
            for row in range(k):
                for col in range(k):
                    den[row][col] = theoretical_acf[abs(j+row-col)]
            num = den.copy()
            for row in range(k):
                num[row][k-1] = theoretical_acf[j+row+1]

            det_num = np.linalg.det(num)
            det_den = np.linalg.det(den)
            gpac[j][k-1] = det_num/det_den
    return gpac

#Stats Model ARMA
def arma_model_parameter_estimation(y,na,nb):
    model = sm.tsa.ARMA(y,(na,nb)).fit(trend = 'nc', disp = 0)
    for i in range(na):
```

```
        print("The AR coefficient a{}".format(i), "is", model.params[i])
        for i in range(nb):
            print("The MA coefficient b{}".format(i), "is", model.params[i+na])
    print(model.summary())
    return model

def auto_correlation_list(array_plot):
  acr_list = []
  for i in range(0, len(array_plot)):
      acr_list.append(auto_correlation(array_plot, i))
  return acr_list

#Qvalue
def find_Q_value(x, x_pred, lags):
  list_errors = x - x_pred
  acr_error_list = theoretical_acf(lags, list_errors)
  sum_square_ac = 0
  for i in range(1, len(acr_error_list)):
     sum_square_ac = sum_square_ac +  np.square(acr_error_list[i])
  Q = len(list_errors) * sum_square_ac
  return Q


def average_method(x, k):
    x_pred = []
    for i in range(0, len(x)):
        if i == k:
            x_pred.append(np.mean(x[:k]))
            break
        else:
            x_pred.append(x[i])
    return x_pred

#Plot for average method: predicted and true values
def average_method_plot(x, x_pred):
    plt.plot(x_pred)
    plt.legend(["Original Values"])
    plt.plot(x[:len(x_pred)])
    plt.legend(["Predicted Model"])
    plt.title("Plot of Predicted Model and Actual Values")
    plt.show()

#Implementing the drift Method
def drift_method(x, k):
  x_pred = []
  for i in range(0, len(x)):
      if i == k:
          x_pred.append(x[i-1]+((x[i-1]-x[0])/(i-1)))
          break
      else:
```

```
            x_pred.append(x[i])
    return x_pred


#Implementing the Naive Method
def naive_method(x, k):
    x_pred = []
    for i in range(0, len(x)):
        if i == k:
            x_pred.append(x[i-1])
            break
        else:
            x_pred.append(x[i])
    return x_pred

#Plot for naive method: predicted and true values
def naive_method_plot(x, x_pred):
    plt.plot(x_pred)
    plt.legend(["Original Values"])
    plt.plot(x[:len(x_pred)])
    plt.legend(["Predicted Model"])
    plt.title("Plot of Predicted Model and Actual Values")
    plt.show()

def simple_exponential_method(x,  alpha, initial_condition):
    x_pred = []
    for i in range(1,len(x)):
      if i==1:
        x_pred.append((alpha*x[i-1])+initial_condition * (1-alpha))
      else:
        x_pred.append((alpha*x[i-1]) + ((1-alpha)*x_pred[i-2]))
    x_pred.insert(0, x[0])
    return x_pred

#Plot for simple_exponential method: predicted and true values
def simple_exponential_method_plot(x, x_pred):
 plt.plot(x_pred)
 plt.legend(["Original Values"])
 plt.plot(x[:len(x_pred)])
 plt.legend(["Predicted Model"])
 plt.title("Plot of Predicted Model and Actual Values")
 plt.show()



#Writing function for forecast error for average and drift:
def forecast_error(x, x_pred):
  error = x[len(x_pred)-1] - x_pred[-1]
  return error
```

```
#Writing function for variance of errors:
def forecast_error_variance(x, *args):
   var_list = []
   for elem in args:
       var_list.append(forecast_error(x, elem))
   return np.var(var_list)


#Writing function for sum square errors:
def sum_square_error(x, *args):
   error_square = 0
   sum_error_square = 0
   for elem in args:
       for i in range(0, len(x)):
           if x[i] != elem[i]:
               sum_error_square  = sum_error_square + np.square(x[i] - elem[i])
               break
   return sum_error_square


#Writing a function for MSE calculation:
def mean_squared_error(x, *args):
   error_square = 0
   sum_error_square = 0
   for elem in args:
       for i in range(0, len(x)):
           if x[i] != elem[i]:
               sum_error_square  = sum_error_square + np.square(x[i] - elem[i])
               break
   return (sum_error_square / len(args))

def find_Q_value(x, x_pred, lags):
  list_errors = list(x - x_pred)
  acr_error_list = theoretical_acf(lags, list_errors)
  sum_square_ac = 0
  for i in range(1, len(acr_error_list)):
     sum_square_ac = sum_square_ac +  np.square(acr_error_list[i])
  Q = len(list_errors) * sum_square_ac
  return Q

def acf_plot_residuals(number_lags, array_to_plot):
   array_to_plot = list(array_to_plot)
   lags = np.arange(-1 * (number_lags - 1), number_lags, 1)
   acf_list = []
   for i in lags:
       acf_list.append(auto_correlation(array_to_plot, i))
   plt.stem(lags, acf_list)
   plt.title("ACF Plot for Residuals")
   plt.ylabel("Values")
```

```
    plt.xlabel("Lags")
    plt.show()
```