VIETNAM NATIONAL UNIVERSITY
UNIVERSITY OF INFORMATION TECHNOLOGY
INFORMATION SYSTEMS FACULTY

REPORT LAB 5
SUBJECT: DATA ANALYSIS

Lecturer : Assoc. Prof. Nguyen Dinh Thuan
Instructor : TA. Nguyen Minh Nhut
Class : STAT3013.O12.CTTT
Group 11 : Pham Thi Thuy Trang - 21522697
: Tran Ngoc Khoi Nguyen - 21522398
: Chau Nguyen Thanh Tai - 21522561

Ho Chi Minh City, October 2023

TABLE OF CONTENT

TEACHER'S COMMENTS

……
……
……
……
……
……
……

WORK DISTRIBUTION

Works \ Members	Pham Thi Thuy Trang (Leader)	Tran Ngoc Khoi Nguyen	Chau Nguyen Thanh Tai

Problem statement	✓	✓	✓
Build the report template	✓		✓
1. Mean Absolute Error (MAE)			✓
2. Mean Absolute Percentage Error (MAPE)			✓
3. Mean Squared Error (MSE)	✓		
4. Root Mean Squared Error (RMSE)	✓		
5. Coefficient of Determination (R-squared)		✓	
6. Adjusted R-squared		✓	
Summarize and edit reports	✓	✓	✓
Completion (%)	100%	100%	100%

A. TASK 1:

I. Mean Absolute Error (MAE):

Definiton:

Absolute Error is the amount of error in your measurements. It is the difference between the measured value and "true" value.

This can be caused by your scale **not measuring the exact amount you are trying to measure**.[1]

For example, your scale may be accurate to the nearest pound. If you weigh 89.6 lbs, the scale may "round up" and give you 90 lbs. In this case the absolute error is 90 lbs – 89.6 lbs = .4 lbs.

The formula for the absolute error (Δx) is:

$$(\Delta x) = x_i - x$$

Where:

x_i is the measurement,

x is the true value.

Mean Absolute Error (MAE) is a statistical measure of errors between paired observations expressing the same phenomenon. It is calculated by taking the average of the absolute differences between the predicted and actual values.[1], [2]

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|x_i - x|$$

Where:

n = the number of errors,

$|x_i - x|$ = the absolute errors.

II. Mean Absolute Percentage Error (MAPE):

Definition:

The **mean absolute percentage error** (MAPE) — also called the mean absolute percentage deviation (MAPD) — measures accuracy of a forecast system. It measures this accuracy as a percentage, and can be calculated as the average absolute percent error for each time period minus actual values divided by actual values.

The mean absolute percentage error (MAPE) is the most common measure used to forecast error, probably because the variable's units are scaled to percentage units, which makes it easier to understand [3]. It works best if there are no extremes to the data (and no zeros). It is often used as a loss function in regression analysis and model evaluation.

The formula for MAPE is:

$$MAPE = \frac{1}{n}\left|\frac{A_t - F_t}{A_t}\right|$$

Where:

n is the number of fitted points.

A_t is the actual value.

F_t is the forecast value.

MAE is a scale-dependent accuracy measure and therefore cannot be used to make comparisons between predicted values that use different scales . It is a common measure of forecast error in time series analysis.

III. Mean Squared Error (MSE):

Mean Squared Error (MSE) is a common metric used to measure the average squared difference between the actual (observed) values and the predicted values in a regression analysis. It is a way to quantify the accuracy of a model's predictions.

The formula for Mean Squared Error is:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

[4]

where:

- *n* is the number of data points,
- *yi* is the actual (observed) value of the dependent variable for the *i*-th data point,
- *y^i* is the predicted value of the dependent variable for the *i*-th data point.

Interpretation:

- A lower MSE indicates a better model fit, meaning the predictions are closer to the actual values.
- A higher MSE indicates a worse model fit, meaning the predictions are further away from the actual values.
- MSE is always a non-negative value.

IV. Root Mean Squared Error (RMSE):

RMSE stands for Root Mean Squared Error, and it is another metric commonly used to evaluate the performance of a regression model, similar to Mean Squared Error (MSE). The key difference is that RMSE takes the square root of the average of the squared differences between the predicted and actual values. This is done to make the metric more interpretable in the same units as the target variable.

The formula for RMSE is as follows:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

[5]

where:

- *n* is the number of data points,
- *yi* is the actual (observed) value of the dependent variable for the *i*-th data point,
- *y^i* is the predicted value of the dependent variable for the *i*-th data point.

Calculation:

- Calculate the squared differences between predicted and actual values for each observation.
- Calculate the average of the squared differences.
- Take the square root of the average to obtain the RMSE.

Interpretation:

- Lower RMSE indicates a better model fit. The predicted values are closer to the actual values.
- Higher RMSE indicates a worse model fit. The predicted values are further away from the actual values.
- RMSE has the same units as the target variable, making it easier to interpret than MSE.

V. Coefficient of Determination (R-squared):

The coefficient of determination, or R^2 is a measure that provides information about the goodness of fit of a model. In the context of regression, it is a statistical measure of how well the regression line approximates the actual data. It is therefore important when a statistical model is used either to predict future outcomes or in the testing of hypotheses.

$$R^2 = 1 - \frac{SSE}{SST} = \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y}_i)^2}$$

Where:
The SSE (sum of squares due to error) is the sum of the residuals squared,
SST (the total sum of squared) is the sum of the distance the data is away from the mean all squared. As it is a percentage it will take values between 0 and 1. [6]

Or we can calculate it by another formular:

$$R^2 = \frac{SSR}{SST} = \frac{\sum(\hat{y}_i - \bar{y}_i)^2}{\sum(y_i - \bar{y}_i)^2}$$

With:
The SSR (sum of squares due to regression). [7]

The use of squared multiple correlation coefficient, R^2, in choosing model is a common goal in econometrics analysis; it is a classical model selection criterion which has been widely used for centuries and it is still popular till today. [8]

VI. Adjusted R-squared:

The R-Squared (R2) value is commonly reported when performing multiple linear regression. It quantifies the proportion of variance of the dependent variable that can be accounted for by the regression model in the sample, which is commonly abbreviated as the proportion of variance explained. It is well known that the R^2 value systematically overestimates the amount of variance explained in the population, which is arguably the more relevant quantity. To estimate the amount of variance explained in the population, the so-called adjusted R^2 is typically used. [9]

$$Adjusted\ R^2 = R_a^2 = 1 - \frac{SSE(n-2)}{SST(v)}$$

Where:
v= n−m

- v is the residual degrees of freedom which indicate the number of independent pieces of information involving the n data points thatare required to calculate the sum of squares,
- n is the number of response values estimated from the response values,
- m is the number of fitted coefficients estimated from the response values. [10]

Or we can simplify it by this formula:

$$R_a^2 = 1 - (1 - R^2)\frac{n-1}{n-p-1}$$

Where:

➔ R^2: The R^2 of the model
➔ n: The number of observations
➔ k: The number of predictor variables [11]

B. TASK 2:

I. Mean Absolute Error (MAE):

1. Using Excel:

b) Lab 3:

Step 1: Caluclate the log of each independent variable

	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
	am	Dominent pc	CO	NO2	O3	PM10	PM2.5	SO2	b0	LOG CO	LOG NO2	LOG O3	LOG PM10	LOG PM2.5	LOG SO2
	h/P	pm25	3	9.00	34	66	85	10	1	1.09861	2.19722	3.52636	4.18965	4.44265	2.30259
	h/P	pm25	3	6.00	22	58	88	10	1	1.09861	1.79176	3.09104	4.06044	4.47734	2.30259
	hΔ	pm25	2	4.00	25	34	42	11	1	0.69315	1.38629	3.21888	3.52636	3.73767	2.39790
	hΔ	pm25	2	5.00	17	37	44	11	1	0.69315	1.60944	2.83321	3.61092	3.78419	2.39790
	hΔ	pm25	2	3.00	25	37	49	11	1	0.69315	1.09861	3.21888	3.61092	3.89182	2.39790
	hΔ	pm25	2	3.00	25	37	49	11	1	0.69315	1.09861	3.21888	3.61092	3.89182	2.39790
	hΔ	pm25	3	9.00	32	23	28	11	1	1.09861	2.19722	3.46574	3.13549	3.33220	2.39790
	hΔ	pm25	3	7.00	28	24	30	11	1	1.09861	1.94591	3.33220	3.17805	3.40120	2.39790

Figure 1: Calculate logs

Step 2: Choose Data Analysis -> Click Regression -> Click OK.

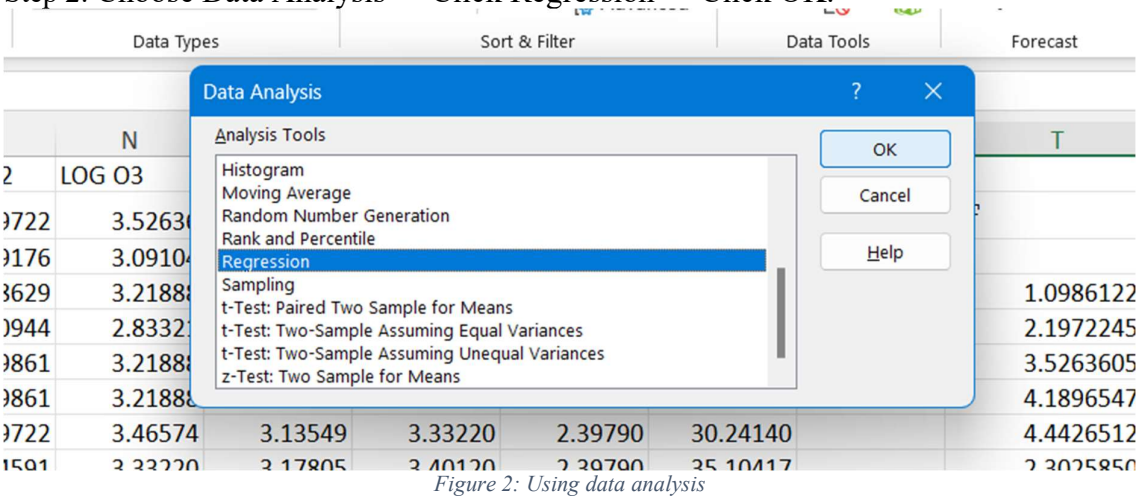


Figure 2: Using data analysis

Step 3: Then, we choose 'AQI' column to be the input Y range and Log columns to be the input X range.

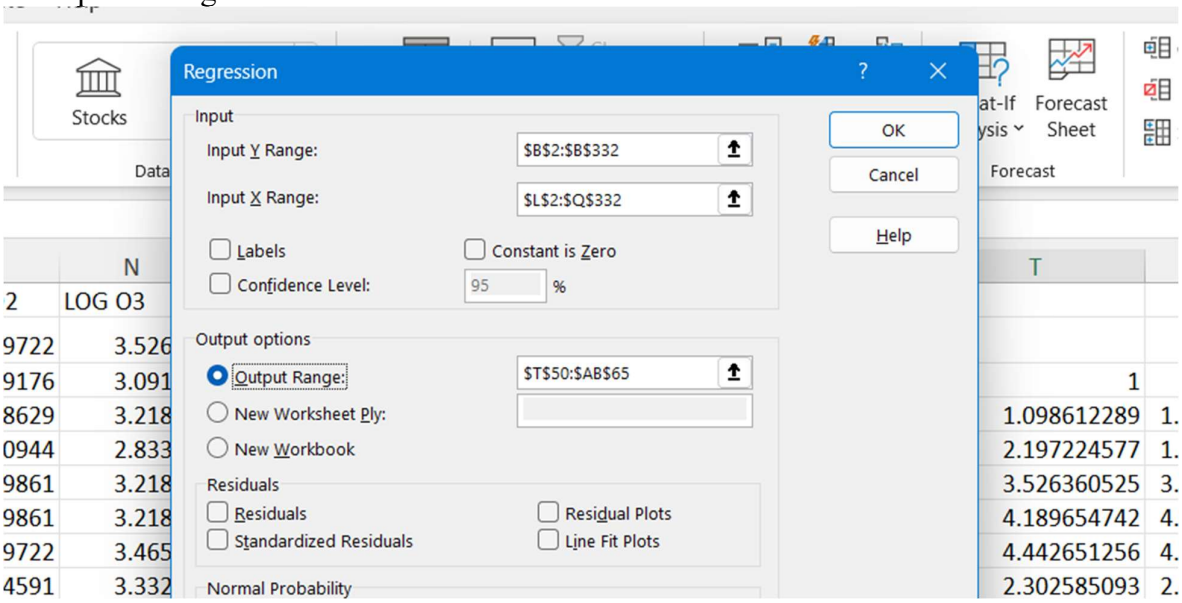


Figure 3: Input X range and Y range

Here is the coefficients table:

SUMMARY OUTPUT

Regression Statistics	
Multiple R	0.90846434
R Square	0.82530745
Adjusted R Square	0.82207241
Standard Error	21.5085165
Observations	331

ANOVA

	df	*SS*	*MS*	*F*	*Significance F*
Regression	6	708120.742	118020.124	255.114505	1.617E-119
Residual	324	149887.675	462.616281		
Total	330	858008.417			

	Coefficients	*tandard Erro*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*Lower 95.0%*	*Upper 95.0%*
Intercept	-137.60494	8.79917908	-15.638384	1.8299E-41	-154.91567	-120.2942	-154.91567	-120.2942
X Variable 1	9.59725741	1.88891689	5.08082566	6.3601E-07	5.88116709	13.3133477	5.88116709	13.3133477
X Variable 2	-5.1583116	1.72653898	-2.9876601	0.00302584	-8.5549538	-1.7616694	-8.5549538	-1.7616694
X Variable 3	-2.7194593	1.43438485	-1.8959063	0.05886203	-5.5413429	0.10242433	-5.5413429	0.10242433
X Variable 4	20.7017642	3.44282508	6.01301657	4.9091E-09	13.9286505	27.474878	13.9286505	27.474878
X Variable 5	33.6588847	3.24385795	10.3761895	5.7333E-22	27.2772016	40.0405679	27.2772016	40.0405679
X Variable 6	0.41412145	1.48805478	0.27829718	0.78096176	-2.5133477	3.34159063	-2.5133477	3.34159063

Figure 4: Coefficients table

Step 4: We begin to predict value

=$U$66+$U$67*L2+$U$68*M2+$U$69*N2+$U$70*O2+P2*$U$71+Q2*$U$72

	R	S	T	U	V	W	X	Y	Z
	Result								
259	89.23645	MAE	X^T						
259	91.00436	15.63367	1	1	1	1	1	1	1
790	52.94353	MAPE	1.098612289	1.09861229	0.69314718	0.69314718	0.69314718	0.69314718	1.09861229
790	56.15758	0.04723	2.197224577	1.79175947	1.38629436	1.60943791	1.09861229	1.09861229	2.19722458
790	61.36651		3.526360525	3.09104245	3.21887582	2.83321334	3.21887582	3.21887582	3.4657359
790	61.36651		4.189654742	4.06044301	3.52636052	3.61091791	3.61091791	3.61091791	3.13549422
790	30.24140		4.442651256	4.47733681	3.73766962	3.78418963	3.8918203	3.8918203	3.33220451
790	35.10417		2.302585093	2.30258509	2.39789527	2.39789527	2.39789527	2.39789527	2.39789527
790	71.15122								

Figure 5: Predict the value

Step 5: We calculate MAE based on the formula:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|x_i - x|$$

=SUM(ABS(B2:B332-R2:R332))/COUNT(R2:R332)

	Q	R	S	T	U	V	W	X	Y	Z
	OG SO2	Result								
	2.30259	89.23645	MAE	X^T						
	2.30259	91.00436	15.63367	1	1	1	1	1	1	1
	2.39790	52.94353	MAPE	1.098612289	1.09861229	0.69314718	0.69314718	0.69314718	0.69314718	1.09861229
	2.39790	56.15758	0.04723	2.197224577	1.79175947	1.38629436	1.60943791	1.09861229	1.09861229	2.19722458
	2.39790	61.36651		3.526360525	3.09104245	3.21887582	2.83321334	3.21887582	3.21887582	3.4657359
	2.39790	61.36651		4.189654742	4.06044301	3.52636052	3.61091791	3.61091791	3.61091791	3.13549422
	2.39790	30.24140		4.442651256	4.47733681	3.73766962	3.78418963	3.8918203	3.8918203	3.33220451
	2.39790	35.10417		2.302585093	2.30258509	2.39789527	2.39789527	2.39789527	2.39789527	2.39789527
	2.39790	71.15122								

Figure 6: Calculate MAE

b) Lab 4:

Step 1: We calculate lags.

S	T	U	V	W	X
date	close	Lag1	Lag2	Lag3	
5/1/2018	174516	175578	174842	172881	
3/1/2018	171655	174516	175578	174842	
9/1/2018	169202	171655	174516	175578	
10/1/2018	172472	169202	171655	174516	
11/1/2018	171655	172472	169202	171655	
12/1/2018	171573	171655	172472	169202	

Figure 7: Calculate lags

Step 2: Choose Data Analysis -> Click Regression -> Click OK.

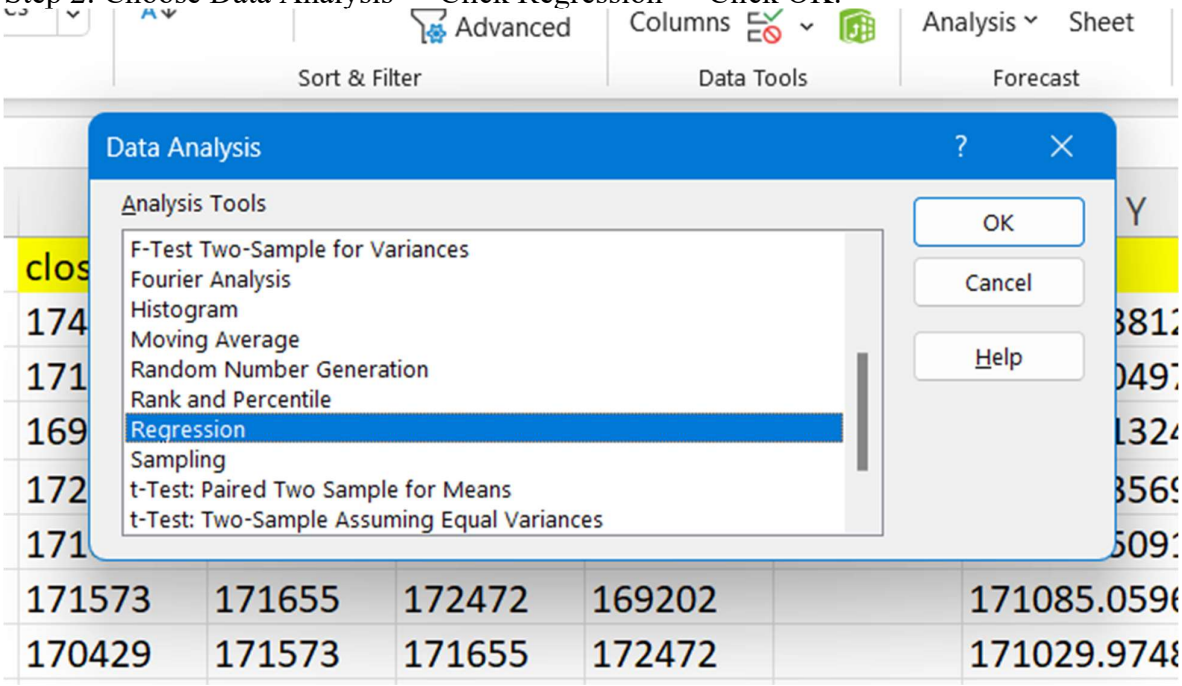


Figure 8: Using data analysis

Step 3: Then, we choose 'close' column to be the input Y range and Lag1, Lag2, Lag3 columns to be the input X range.

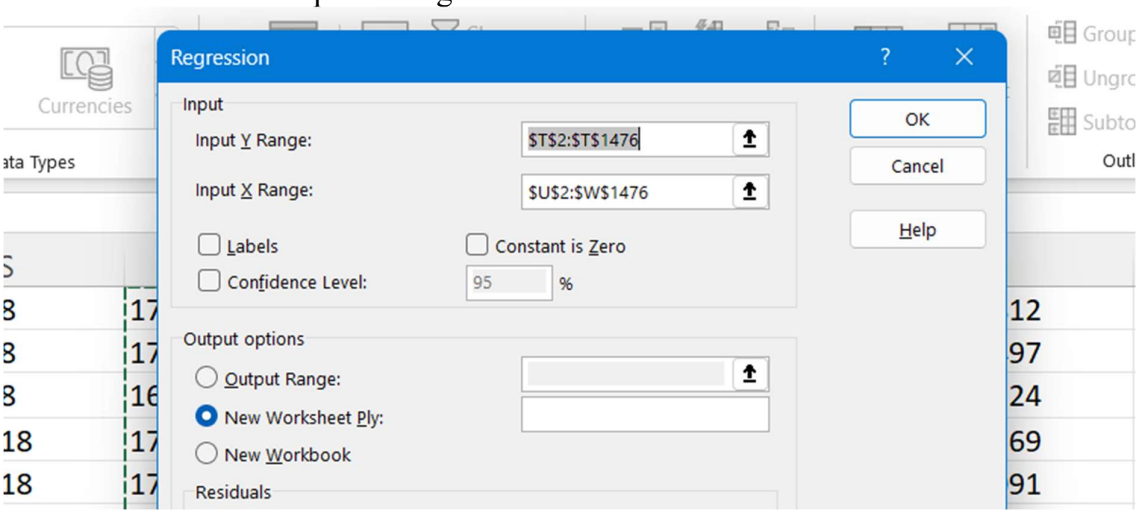


Figure 9: Input Y range and X range

Here is the coefficients table:

	A	B	C	D	E	F	G	H	I	J
1	SUMMARY OUTPUT									
2										
3	*Regression Statistics*									
4	Multiple R	0.99741								
5	R Square	0.994826								
6	Adjusted R	0.994816								
7	Standard E	1718.214								
8	Observatic	1475								
9										
10	ANOVA									
11		*df*	*SS*	*MS*	*F*	*Significance F*				
12	Regressior	3	8.35E+11	2.78E+11	94285.36	0				
13	Residual	1471	4.34E+09	2952260						
14	Total	1474	8.39E+11							
15										
16		*Coefficients*	*tandard Err*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*ower 95.0%*	*pper 95.0%*	
17	Intercept	507.1998	180.4302	2.811058	0.005003	153.2718204	861.1277	153.2718	861.1277	
18	X Variable	0.987788	0.02607	37.89032	8.4E-220	0.936650402	1.038926	0.93665	1.038926	
19	X Variable	-0.00154	0.036651	-0.04215	0.966384	-0.073439212	0.070349	-0.07344	0.070349	
20	X Variable	0.007548	0.025968	0.290641	0.771366	-0.043391711	0.058487	-0.04339	0.058487	
21										

Figure 10: Coefficient table.

Step 4: we start to predict

=507.2 +0.98779*U2 -0.0015*V2+0.00755*W2

	Q	R	S	T	U	V	W	X	Y
			date	close	Lag1	Lag2	Lag3		Predict
			5/1/2018	174516	175578	174842	172881		174984.3812
			8/1/2018	171655	174516	175578	174842		173949.0497
			9/1/2018	169202	171655	174516	175578		171130.1324
			10/1/2018	172472	169202	171655	174516		168703.3569
			11/1/2018	171655	172472	169202	171655		171915.5091
			12/1/2018	171573	171655	172472	169202		171085.0596
			15/01/2018	170429	171573	171655	172472		171029.9748

Figure 11:Predicting

Step 5: We calculate MAE based on the formula:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|x_i - x|$$

=SUM(ABS((T2:T1476)-(Y2:Y1476)))/COUNT(T2:T1476)

	P	Q	R	S	T	U	V	W	X	Y	Z	AA
				date	close	Lag1	Lag2	Lag3		Predict		
				5/1/2018	174516	175578	174842	172881		174984.3812		MAPE
				8/1/2018	171655	174516	175578	174842		173949.0497		93469.6486
				9/1/2018	169202	171655	174516	175578		171130.1324		MAE
				10/1/2018	172472	169202	171655	174516		168703.3569		1068.334771
				11/1/2018	171655	172472	169202	171655		171915.5091		

Figure 12: Calculate MAE

2. Using R language:

a) Lab 3:

Step 1: Import and Read dataset.

```
> nls_data
   Station.ID AQI.index                                Station.name Dominent.pollutant CO NO2
1       13250         1 Lào Cai/KCN TALOONG1, Vietnam                pm10 43   1
2       13250         1 Lào Cai/KCN TALOONG1, Vietnam                pm10 53   1
3       13250         1 Lào Cai/KCN TALOONG1, Vietnam                pm10 38   1
4       13250         1 Lào Cai/KCN TALOONG1, Vietnam                pm10 34   2
5       13250         1 Lào Cai/KCN TALOONG1, Vietnam                pm10 39   3
6       13026         1  Hà Nội/Chi cục BVMT, Vietnam                pm25  6  27
7       13026         1  Hà Nội/Chi cục BVMT, Vietnam                pm25  6  27
8       13026         1  Hà Nội/Chi cục BVMT, Vietnam                pm25  6  27
9       13026         1  Hà Nội/Chi cục BVMT, Vietnam                pm25  6  27
10      13026         1  Hà Nội/Chi cục BVMT, Vietnam                pm25  6  27
11      13026         1  Hà Nội/Chi cục BVMT, Vietnam                pm25  6  27
```

Figure 13: Import dataset.

Step 2: Calculate logs and use lm() function.

```
> nls_data$log_CO <- log10(nls_data$CO)
> nls_data$log_NO2 <- log10(nls_data$NO2)
> nls_data$log_O3 <- log10(nls_data$O3)
> nls_data$log_PM10 <- log10(nls_data$PM10)
> nls_data$log_PM2.5 <- log10(nls_data$PM2.5)
> nls_data$log_SO2 <- log10(nls_data$SO2)
> independent_vars <- c('log_CO', 'log_NO2', 'log_O3', 'log_PM10', 'log_PM2.5', 'log_SO2')
> reg = lm(AQI.index ~ log_CO + log_NO2 + log_O3 + log_PM10 + log_PM2.5 + log_SO2, data = nls_data)
> summary(reg)
```

Figure 14: Calculate logs and use lm() function.

Here is the result:

```
Call:
lm(formula = AQI.index ~ log_CO + log_NO2 + log_O3 + log_PM10 +
    log_PM2.5 + log_SO2, data = nls_data)

Residuals:
    Min      1Q  Median      3Q     Max
-46.487 -15.289  -5.708   8.103 101.066

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -137.6049     8.7992 -15.638  < 2e-16 ***
log_CO        22.0985     4.3494   5.081 6.36e-07 ***
log_NO2      -11.8775     3.9755  -2.988  0.00303 **
log_O3        -6.2618     3.3028  -1.896  0.05886 .
log_PM10      47.6676     7.9274   6.013 4.91e-09 ***
log_PM2.5     77.5024     7.4693  10.376  < 2e-16 ***
log_SO2        0.9535     3.4264   0.278  0.78096
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 21.51 on 324 degrees of freedom
Multiple R-squared:  0.8253,    Adjusted R-squared:  0.8221
F-statistic: 255.1 on 6 and 324 DF,  p-value: < 2.2e-16
```

Figure 15: The result

Step 3: Next, we import library “Metrics”

```
> library(Metrics)
> nls_data$Predict = predict(reg)
> nls_data$Actual = AQI.index
> nls_data
```

Figure 16: Import lbrabry "Metrics"

Step 4: Calculate MAE

```
 [ reached 'max' / getOption("max.print") -- omitted 276 rows ]
> mape <- mape(nls_data$Actual,nls_data$Predict)
> mape
[1] 0.5312956
> mae <- mae(nls_data$Actual,nls_data$Predict)
> mae
[1] 15.63367
```

Figure 17: Calculate MAE

b) Lab 4:

Step 1: Read the dataset

```
> # Read data from a CSV file
> df <- read.csv(file.choose(), sep = ",")
```

Figure 18: Raed the dataset.

Step 2: Set row names to index

```
>
> # Set row names to the 'date' column
> rownames(df) <- df$date
```

Figure 19: Set row names to index.

Step 3: Convert data to numeric format

```
>
> # Convert 'close' column to numeric format
> df$close <- as.numeric(gsub(",", ".", df$close))
>
```

Figure 20: convert data.

Step 4: Create time series object

```
>
> # Create a time series object
> data <- ts(na.omit(df$close), frequency = 1, start = c(02, 01, 2018))
>
```

Figure 21: Create time series object.

Step 5: Plot data

```
>
> # Plot the time series data
> plot(data)
```

Figure 22: Plot data.

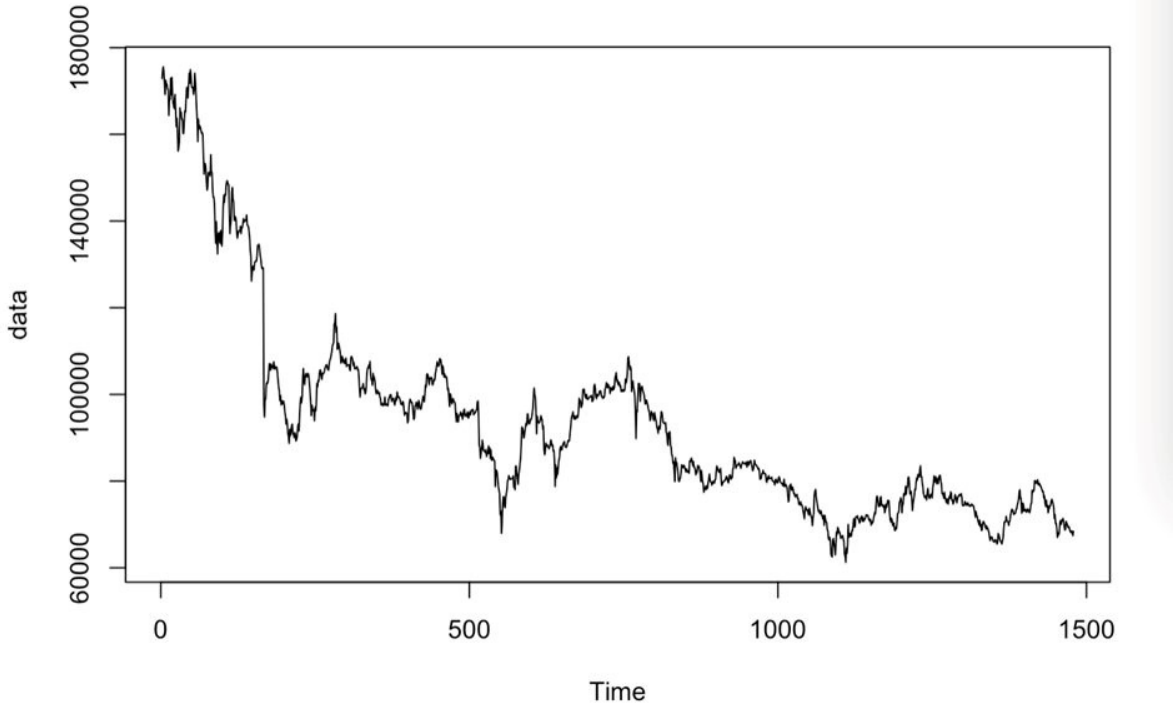


Figure 23: The result chart.

Step 6: Calculate the Dickey-Fuller test

```
> # Perform Augmented Dickey-Fuller test for stationarity
> adf.test(data)

        Augmented Dickey-Fuller Test

data:  data
Dickey-Fuller = -3.0937, Lag order = 11, p-value = 0.1153
alternative hypothesis: stationary
```

Figure 24: Calculate Dickey-Fuller Test.

Step 7: Split data to train & Test

```
>
> # Split the data into training and testing sets
> train <- head(df, 1034)
> test <- tail(df, 443)
```

Figure 25: Split data to train and test.

Step 8: Create time series objects

```
>
> # Create time series objects
> train_ts <- ts(train$close, frequency = 1, start = c(02, 01, 2018))
> test_ts <- ts(test$close, frequency = 1, start = c(25, 02, 2022))
>
```

Figure 26: Create time series object.

Step 9: Find the fit ARIMA model by auto.arima

```
> # Use auto.arima with the time series object
> arima_model <- forecast::auto.arima(train_ts, trace = TRUE)

 Fitting models using approximations to speed things up...

 ARIMA(2,1,2) with drift         : Inf
 ARIMA(0,1,0) with drift         : 18545.04
 ARIMA(1,1,0) with drift         : 18546.91
 ARIMA(0,1,1) with drift         : 18547.05
 ARIMA(0,1,0)                    : 18545.53
 ARIMA(1,1,1) with drift         : 18548.88

 Now re-fitting the best model(s) without approximations...

 ARIMA(0,1,0) with drift         : 18560.16

 Best model: ARIMA(0,1,0) with drift
```

Figure 27: Find the fit ARIMA model.

Step 10: Train the data with ARIMA model (0,1,0)

```
>
> # Fit ARIMA model to the training set
> fitARIMAtrain <- arima(train_ts, order = c(0, 1, 0))
>
```

Figure 28: Train data.

Step 11: Forecast for the testing data

```
> # Forecast for the testing set
> forecast_values <- forecast(fitARIMAtrain, h = length(test_ts))
>
```

Figure 29: Forecast for the testing data.

Step 12: Plot the forecast data

```
>
> # Plot the forecast for the testing set
> plot(forecast_values)
>
```

Figure 30: Plot the forecast data.

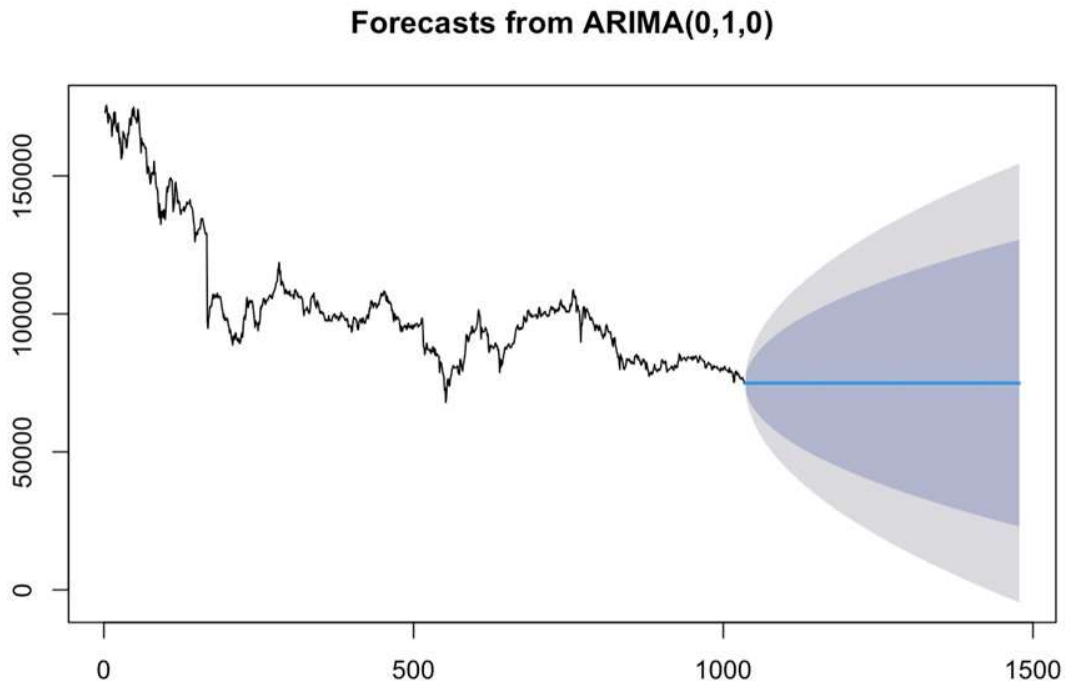


Figure 31: Forecast data chart.

Step 13: Extract the forcasted values

```
>
> # Extract the forecasted values
> predicted_values <- forecast_values$mean
>
```

Figure 32: Extract the forecasted values.

Step 14: Calculate the MSE

```
+   # Create a data frame with actual and predicted values
+   merged_data <- merge(data.frame(actual = test_ts), data.frame(predicted = predicted_values), by =
"row.names", all = TRUE)
+
+   # Rename columns
+   colnames(merged_data) <- c("date", "actual", "predicted")
+
+   # Calculate MAE and MAPE
+   mae <- mean(abs(merged_data$actual - merged_data$predicted), na.rm = TRUE)
+   mape <- mean(abs((merged_data$actual - merged_data$predicted) / merged_data$actual) * 100, na.rm =
TRUE)
+
+   # Check for NaN in MAE and MAPE
+   if (!is.finite(mae) || is.nan(mae) || !is.finite(mape) || is.nan(mape)) {
+     cat("Warning: NaN detected in MAE or MAPE.\n")
+   } else {
+     # Print MAE and MAPE
+     cat("MAE: ", mae, "\n")
+     cat("MAPE: ", mape, "\n")
+   }
+
+     # Plot actual vs. predicted values
+     plot(merged_data$date, merged_data$actual, type = "l", col = "blue", ylab = "Close Price", xlab =
"Time", main = "Actual vs. Predicted")
+     lines(merged_data$date, merged_data$predicted, col = "red")
+     legend("topright", legend = c("Actual: Blue", "Predicted: Red"), col = c("blue", "red"))
+   }
MAE:  3716.452
MAPE:  5.259202
```

Figure 33: MAE calculation.

Explain:

The MAE value of 3716.452 indicates the average absolute difference between the actual and predicted values. A larger MAE suggests a larger absolute error in your predictions. On the other hand, the MAPE value of 5.259202 indicates the average percentage difference between the actual and predicted values. A smaller MAPE is generally considered better, and a value of 5.259202 suggests that, on average, the predictions are off by approximately 5.26%.

3. Using Python languages:

a) Lab3:

Step 1: Import libraries and read dataset.

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
```

[52] ✓ 0.0s

```
data = pd.read_csv("lab3.csv")
data
```

[53] ✓ 0.0s

	Station ID	AQI index	Dominent pollutant	CO	NO2	O3	PM10	PM2.5	SO2	b0
0	12958	85	pm25	3	9	34	66	85.0	10	1
1	12958	88	pm25	3	6	22	58	88.0	10	1
2	13012	42	pm25	2	4	25	34	42.0	11	1
3	13012	44	pm25	2	5	17	37	44.0	11	1
4	13012	49	pm25	2	3	25	37	49.0	11	1
...	...	...	...	...	...	...	...	...	...	...
326	12976	67	pm25	5	12	8	53	67.0	9	1

Figure 34: Import and read dataset.

Step 2: Calculate logs of independent variables

```
data['log(CO)'] = np.log(data['CO'])
data['log(NO2)'] = np.log(data['NO2'])
data['log(O3)'] = np.log(data['O3'])
data['log(PM10)'] = np.log(data['PM10'])
data['log(PM2.5)'] = np.log(data['PM2.5'])
data['log(SO2)'] = np.log(data['SO2'])
```

54] ✓ 0.0s

Figure 35: Logs of independent variables

Step 4: Taking a dependent variable is graduation and independent variables.

```
x = np.array(data[['log(CO)', 'log(NO2)', 'log(O3)', 'log(PM10)', 'log(PM2.5)', 'log(SO2)']]).reshape((-1, 6))
y = np.array(data['AQI index']).reshape((-1, 1))
```

] ✓ 0.0s

Figure 36:Taking a dependent variable and independent variables.

Step 5: Wer use scikit-learn's LinearRegression class to create a linear regression model ('reg') and then fit it to the input data 'x' and the target variable 'y'. The fit method is used to train the model on the provided data.

```
reg = LinearRegression()
reg.fit(x,y)
```

[56] ✓ 0.0s

```
▾ LinearRegression
LinearRegression()
```

Figure 37: Create the LinearRegression model.

```
reg.intercept_
```

[57] ✓ 0.0s

```
array([-137.60493727])
```

```
reg.coef_
```

[58] ✓ 0.0s

```
array([[ 9.59725741, -5.15831159, -2.7194593 , 20.70176424, 33.65888474,
         0.41412145]])
```

```
reg.score(x,y)
```

[59] ✓ 0.0s

```
0.8253074538923281
```

Figure 38: Take the values of the estimates of the regression coefficients.

The fit() function in Python is used to train a linear regression model. This function will find the regression coefficients of the model so that the sum of squared errors between the predicted values and the actual values is minimized.

- The first parameter x: The array containing the values of the independent variable.
- The second parameter y: The array containing the values of the dependent variable.

Gaining the R2 by using the .score() function and take the values of the estimates of the regression coefficients with .intercept_ and .coef_.

Step 6: Creating a regression table using MLR through OLS.

```
y = data['AQI index']

# For independent variables (x), you can select the desired columns
x = data[['log(CO)', 'log(NO2)', 'log(O3)', 'log(PM10)', 'log(PM2.5)', 'log(SO2)']]

# If you want to add a constant term to the independent variables, you can use the following:
x = sm.add_constant(x)

reg = sm.OLS(y,x).fit()
print(reg.summary())
```

[60] ✓ 0.0s

Figure 39: Create a regression table.

Here is the result:

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              AQI index   R-squared:                       0.825
Model:                            OLS   Adj. R-squared:                  0.822
Method:                 Least Squares   F-statistic:                     255.1
Date:                Sun, 10 Dec 2023   Prob (F-statistic):          1.62e-119
Time:                        16:13:17   Log-Likelihood:                -1481.8
No. Observations:                 331   AIC:                             2978.
Df Residuals:                     324   BIC:                             3004.
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       -137.6049      8.799    -15.638      0.000    -154.916    -120.294
log(CO)        9.5973      1.889      5.081      0.000       5.881      13.313
log(NO2)      -5.1583      1.727     -2.988      0.003      -8.555      -1.762
log(O3)       -2.7195      1.434     -1.896      0.059      -5.541       0.102
log(PM10)     20.7018      3.443      6.013      0.000      13.929      27.475
log(PM2.5)    33.6589      3.244     10.376      0.000      27.277      40.041
log(SO2)       0.4141      1.488      0.278      0.781      -2.513       3.342
==============================================================================
Omnibus:                      121.452   Durbin-Watson:                   1.273
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              368.084
Skew:                           1.684   Prob(JB):                     1.18e-80
...
==============================================================================
```

Figure 40: Result

Step 6: We use scikit-learn's metrics functions to calculate Mean Absolute Error (MAE) for the predictions made by a linear regression model.

```
from sklearn.metrics import mean_absolute_percentage_error,mean_absolute_error
y_predict = reg.predict(x)
mape = mean_absolute_percentage_error(y,y_predict)
mae = mean_absolute_error(y,y_predict)
print("MAPE: ",mape)
print("MAE: ",mae)
```

[62] ✓ 0.0s

```
MAPE:  0.531295579500056
MAE:  15.63367475842379
```

Figure 41: Calculate MAE

Explain:
The relatively high MAPE value (0.5313) suggests that, on average, the percentage difference between the actual and predicted values is considerable. This indicates that the model's predictions have a relatively large relative error.
The MAE value (15.6337) provides information about the average magnitude of errors, and a value of approximately 15.63 indicates that, on average, the model's predictions deviate from the actual values by about 15.63 units.

b) Lab 4:

Step 1: Import necessary libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
```

✓ 5.2s

+ Code + Markdown

Figure 42: Import libraries.

Step 2: We load data from CSV file into Jupiter notebook and split data into 3 parts: train data, test data and validate data with 7:2:1 scale.

```
df = pd.read_csv('vnmvn.csv')
df = df[['close']]
df = df.dropna() # Drop missing values
df = df.reset_index(drop=True) # Reset the index

# Split the data into training, testing, and validation sets by 7:2:1
train_size = int(0.7 * len(df))
test_size = int(0.2 * len(df))
val_size = len(df) - train_size - test_size

train_data = df[:train_size]
test_data = df[train_size:train_size+test_size]
val_data = df[train_size+test_size:]
```

✓ 0.0s

Figure 43: Split data

Step 3: Find the best ARI function using “auto_arima” support function in python and then fit the data with the model. The best ARIMA model is ARIMA (0,1,0).

```
# Training process
x_train = np.array(train_data.index).reshape(-1, 1)
y_train = np.array(train_data['close'])

# Find the best ARIMA model using auto_arima
from pmdarima.arima import auto_arima
model = auto_arima(y_train, trace=True, error_action='ignore', suppress_warnings=True)

# Fit the model
model.fit(y_train)
✓ 1.8s
```

Figure 44: Find the best ARIMA model

Here is the result:

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.59 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=18560.152, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=18561.221, Time=0.02 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=18561.227, Time=0.03 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=18560.656, Time=0.01 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=18563.234, Time=0.06 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0] intercept
Total fit time: 0.724 seconds
```

ARIMA
ARIMA(0,1,0)(0,0,0)[0] intercept

Figure 45: Result

Step 4: Assign testing data and validating data to some variables to calculate.

```
# make predictions on the testing set
x_test = np.array(test_data.index).reshape(-1, 1)
y_test = np.array(test_data['close'])
y_pred = model.predict(n_periods=len(y_test))

# make predictions on the validate set
x_val= np.array(val_data.index).reshape(-1, 1)
y_val = np.array(val_data['close'])
y_pred_val =  model.predict(n_periods=len(y_val))
[4] ✓ 0.0s
```

Figure 46: Assign testing data

Step 5: We use scikit-learn's metrics functions to calculate Mean Absolute Error (MAE) for the predictions made by ARIMA model.

```
# calculate the MAPE
valid_mape = np.mean(np.abs((y_val - y_pred_val) / y_val)) * 100
test_mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
print("Validation MAPE:", valid_mape)
print("Testing MAPE:", test_mape)

# calculate the MAE
valid_mae = np.mean(np.abs(y_pred_val - y_val))
test_mae = np.mean(np.abs(y_pred - y_test))
print("Validation MAE:", valid_mae)
print("Testing MAE:", test_mae)
```

[5] ✓ 0.0s

```
Validation MAPE: 9.42634997618378
Testing MAPE: 17.802781729100502
Validation MAE: 6855.259643833731
Testing MAE: 13419.157165405752
```

Figure 47: Calculate MAE

II. Mean Absolute Percentage Error (MAPE):

1. Using Excel:

a) Lab 3:

We can rely on the data calculated from MAE to calculate MAPE when A_t (AQI index) is the actual value and F_t (Result) is the forecast value.

Then we apply the MAPE's formula:

$$MAPE = \frac{1}{n}\left|\frac{A_t - F_t}{A_t}\right|$$

=1/COUNT(R2:R332)*SUM(ABS(B2:B332-R2:R332)/COUNT(R2:R332))

Q	R	S	T	U	V	W	X	Y	Z
G SO2	Result								
2.30259	89.23645	MAE	X^T						
2.30259	91.00436	15.63367	1	1	1	1	1	1	1
2.39790	52.94353	MAPE	1.098612289	1.09861229	0.69314718	0.69314718	0.69314718	0.69314718	1.09861229
2.39790	56.15758	0.04723	2.197224577	1.79175947	1.38629436	1.60943791	1.09861229	1.09861229	2.19722458
2.39790	61.36651		3.526360525	3.09104245	3.21887582	2.83321334	3.21887582	3.21887582	3.4657359
2.39790	61.36651		4.189654742	4.06044301	3.52636052	3.61091791	3.61091791	3.61091791	3.13549422
2.39790	30.24140		4.442651256	4.47733681	3.73766962	3.78418963	3.8918203	3.8918203	3.33220451
2.39790	35.10417		2.302585093	2.30258509	2.39789527	2.39789527	2.39789527	2.39789527	2.39789527
2.39790	71.15122								

Figure 48: Calculate MAPE in Lab3

b) Lab 4:

We can rely on the data calculated from MAE to calculate MAPE when A_t (close) is the actual value and F_t (predict) is the forecast value.

Then we apply the MAPE's formula:

$$MAPE = \frac{1}{n}\left|\frac{A_t - F_t}{A_t}\right|$$

=1/COUNT(T2:T1476)*SUM(ABS((T2:T1476)-(Y2:Y1476)/(T2:T1476)))

P	Q	R	S	T	U	V	W	X	Y	Z	AA
			date	close	Lag1	Lag2	Lag3		Predict		
			5/1/2018	174516	175578	174842	172881		174984.3812		MAPE
			8/1/2018	171655	174516	175578	174842		173949.0497		93469.6486
			9/1/2018	169202	171655	174516	175578		171130.1324		MAE
			10/1/2018	172472	169202	171655	174516		168703.3569		1068.334771
			11/1/2018	171655	172472	169202	171655		171915.5091		

Figure 49: Calculate MAPE in Lab4.

2. Using R language:

a) Lab 3:

We perform the calculation steps as we did in the MAE section and here are the results obtained:

```
 [ reached 'max' / getOption("max.print") -- omitted 276 rows ]
> mape <- mape(nls_data$Actual,nls_data$Predict)
> mape
[1] 0.5312956
> mae <- mae(nls_data$Actual,nls_data$Predict)
> mae
[1] 15.63367
```

Figure 50: Using R to calculate MAPE in Lab3.

b) Lab 4:

We perform the calculation steps as we did in the MAE section and here are the results obtained:

```
+     legend("topright", legend = c("Actual: Blue", "Predicted: Red"), col = c("blue", "red")
+   }
MAE:  3716.452
MAPE:  5.259202
> }
```

Figure 51: Using R to calculate MAPE in Lab4.

Explain:

While the MAPE value of 5.259202 indicates the average percentage difference between the actual and predicted values. A smaller MAPE is generally considered better, and a value of 5.259202 suggests that, on average, the predictions are off by approximately 5.26%.

3. Using Python languages:

a) Lab 3:

We perform the calculation steps as we did in the MAE section and here are the results obtained:

```
from sklearn.metrics import mean_absolute_percentage_error,mean_absolute_error
y_predict = reg.predict(x)
mape = mean_absolute_percentage_error(y,y_predict)
mae = mean_absolute_error(y,y_predict)
print("MAPE: ",mape)
print("MAE: ",mae)
```

```
[62]  ✓ 0.0s

...  MAPE:  0.531295579500056
     MAE:  15.63367475842379
```

Figure 52:Using Python to calculate MAPE in Lab3.

Explain:

The relatively high MAPE value (0.5313) suggests that, on average, the percentage difference between the actual and predicted values is considerable. This indicates that the model's predictions have a relatively large relative error.

b) Lab 4:

We perform the calculation steps as we did in the MAE section and here are the results obtained:

```
# calculate the MAPE
valid_mape = np.mean(np.abs((y_val - y_pred_val) / y_val)) * 100
test_mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
print("Validation MAPE:", valid_mape)
print("Testing MAPE:", test_mape)

# calculate the MAE
valid_mae = np.mean(np.abs(y_pred_val - y_val))
test_mae = np.mean(np.abs(y_pred - y_test))
print("Validation MAE:", valid_mae)
print("Testing MAE:", test_mae)
```

[5] ✓ 0.0s

```
Validation MAPE: 9.42634997618378
Testing MAPE: 17.802781729100502
Validation MAE: 6855.259643833731
Testing MAE: 13419.157165405752
```

Figure 53: Using Python to calculate MAPE in Lab4.

Explain:
Validation MAPE (9.43%):
A MAPE of 9.43% on the validation set indicates that, on average, the model's predictions have an error of approximately 9.43% relative to the actual values in the validation set. A lower MAPE suggests a relatively better performance in terms of percentage accuracy.
Testing MAPE (17.80%):
A MAPE of 17.80% on the testing set suggests that, on average, the model's predictions have an error of approximately 17.80% relative to the actual values in the testing set. A higher MAPE in the testing set compared to the validation set may indicate a potential decrease in performance when applied to new, unseen data.

III. Mean Squared Error (MSE):

1. Using Excel:

a) Lab 3:

Step 1: We need to clean data in order to regression, (remove blank data, select rows of data whose dominant pollutants are PM10 or PM2.5).

Step 2: We calculate log of the dependent variables.

K2 =LN(E2)

	I	J	K	L	M	N	O	P	Q
1	?.5	SO2	LOG CO	LOG NO2	LOG O3	LOG PM10	LOG PM2.5	LOG SO2	
2	85	10	1,09861	2,19722	3,52636	4,18965	4,44265	2,30259	
3	88	10	1,09861	1,79176	3,09104	4,06044	4,47734	2,30259	
4	42	11	0,69315	1,38629	3,21888	3,52636	3,73767	2,39790	
5	44	11	0,69315	1,60944	2,83321	3,61092	3,78419	2,39790	
6	49	11	0,69315	1,09861	3,21888	3,61092	3,89182	2,39790	
7	49	11	0,69315	1,09861	3,21888	3,61092	3,89182	2,39790	
8	28	11	1,09861	2,19722	3,46574	3,13549	3,33220	2,39790	
9	30	11	1,09861	1,94591	3,33220	3,17805	3,40120	2,39790	
10	60	11	1,38629	2,30259	3,43399	3,76120	4,09434	2,39790	
11	131	11	1,60944	2,56495	2,63906	4,31749	4,87520	2,39790	
12	142	11	1,60944	3,52636	3,17805	4,34381	4,95583	2,39790	
13	116	11	1,94591	2,99573	3,17805	4,15888	4,75359	2,39790	
14	139	117	3,63759	0,69315	2,19722	4,00733	4,93447	4,76217	

Figure 54:Handling log of dependent variables

Step 3: Choose [Data Analysis] on the Data tab.

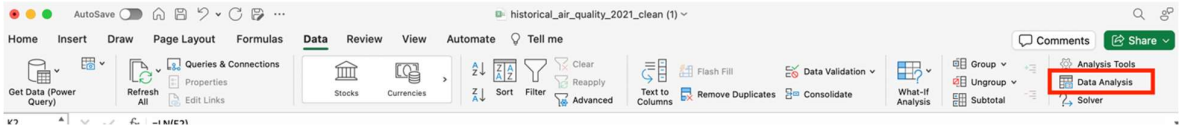


Figure 55: Position of the DA tool on the toolbar.

Step 4: In [Data Analysis] dialog, choose Regression.

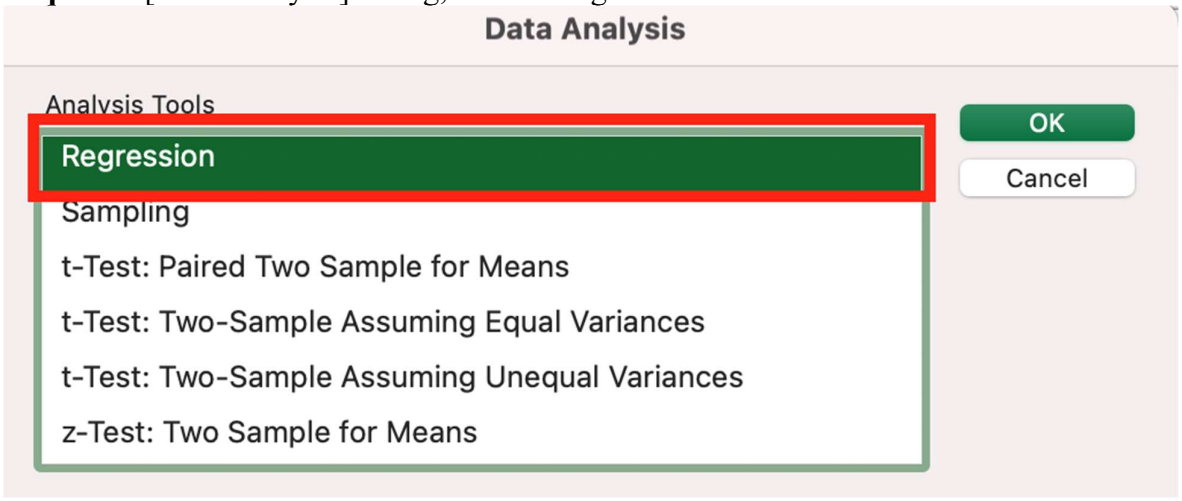


Figure 56: DA dialog box.

Step 5: In [Regression] dialog, enter the input range X and input range Y. After that, choose the location of the output range.

- Input Y range:

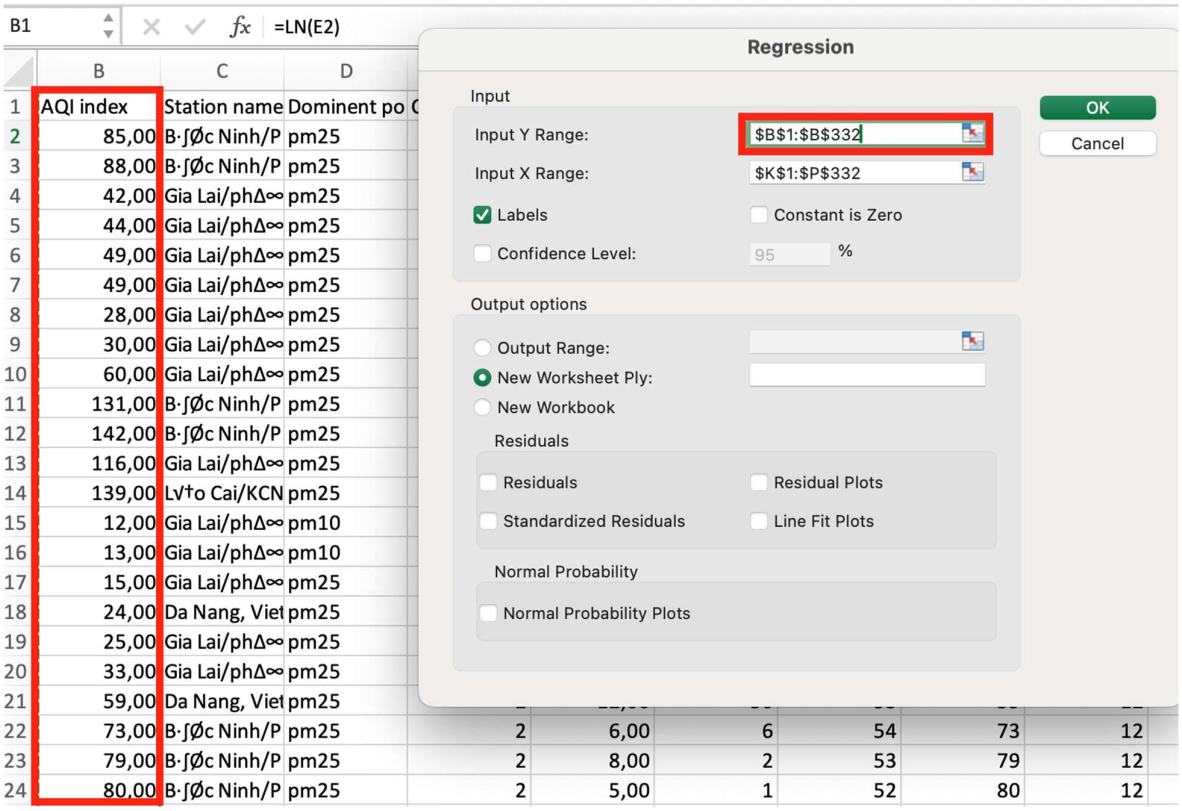


Figure 57: Choosing range step in Regression dialog.

- Input X range:

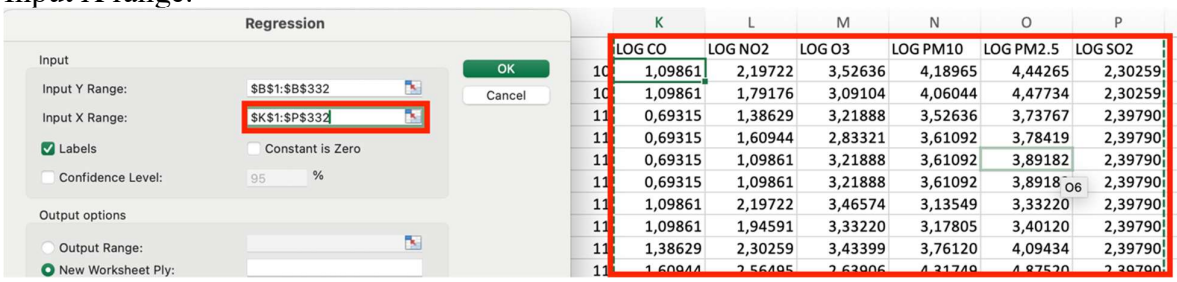


Figure 58:Range of input.

- Input the output range:

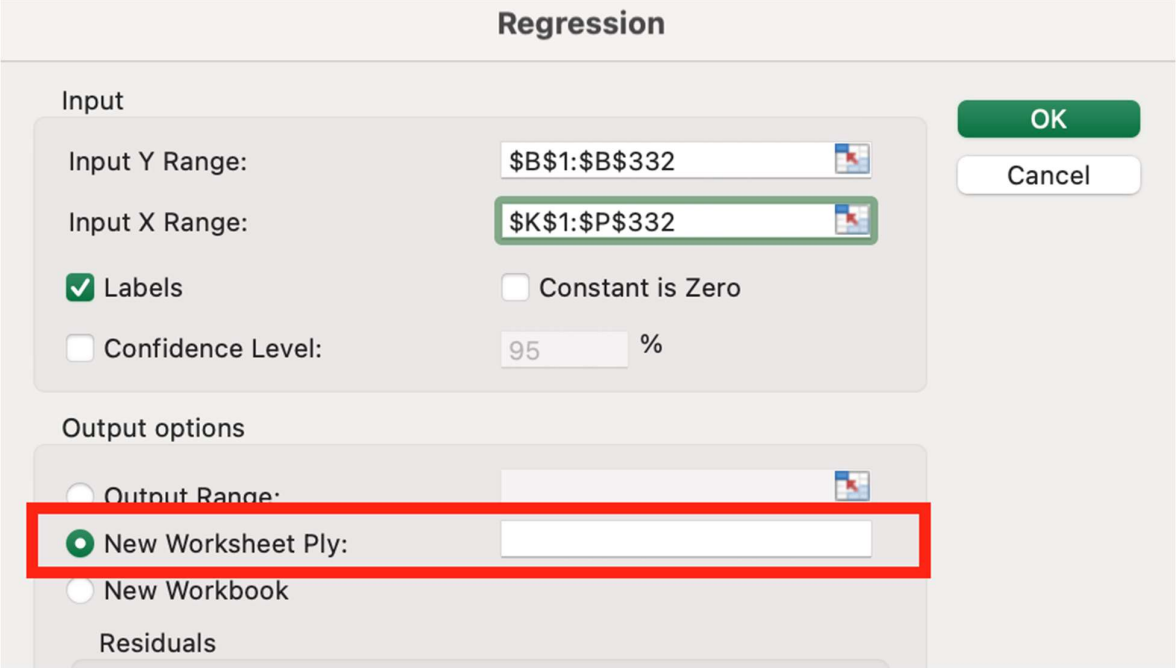


Figure 59: Position of output.

Step 6: Click OK and we will have the result like picture below.

SUMMARY OUTPUT

Regression Statistics	
Multiple R	0,90846434
R Square	0,82530745
Adjusted R S	0,82207241
Standard Err	21,5085165
Observations	331

ANOVA

	df	*SS*	*MS*	*F*	*Significance F*
Regression	6	708120,742	118020,124	255,114505	1,617E-119
Residual	324	149887,675	462,616281		
Total	330	858008,417			

	Coefficients	*tandard Erro*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*Lower 95,0%*	*Upper 95,0%*
Intercept	-137,60494	8,79917908	-15,638384	1,8299E-41	-154,91567	-120,2942	-154,91567	-120,2942
LOG CO	9,59725741	1,88891689	5,08082566	6,3601E-07	5,88116709	13,3133477	5,88116709	13,3133477
LOG NO2	-5,1583116	1,72653898	-2,9876601	0,00302584	-8,5549538	-1,7616694	-8,5549538	-1,7616694
LOG O3	-2,7194593	1,43438485	-1,8959063	0,05886203	-5,5413429	0,10242433	-5,5413429	0,10242433
LOG PM10	20,7017642	3,44282508	6,01301657	4,9091E-09	13,9286505	27,474878	13,9286505	27,474878
LOG PM2.5	33,6588847	3,24385795	10,3761895	5,7333E-22	27,2772016	40,0405679	27,2772016	40,0405679
LOG SO2	0,41412145	1,48805478	0,27829718	0,78096176	-2,5133477	3,34159063	-2,5133477	3,34159063

Figure 60: The result of Regression tool.

From the above results we can draw the following conclusions:

Regression equation:

AQI = -127.60494 + 9.59725741*log10(CO))-5.1583116*log10(NO2) – 2.7194593 *log10(O3) + 20.7017642 * log10(PM10)+ 0.4142145* log10(PM2.5)

Step 7: Calculate AQI by the equation

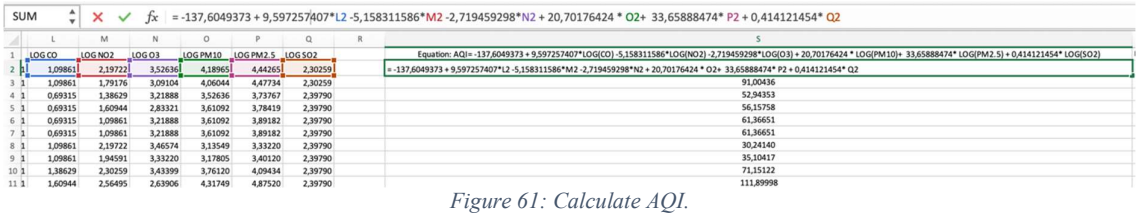


Figure 61: Calculate AQI.

Step 8: Calculate the Error: Y-Y^

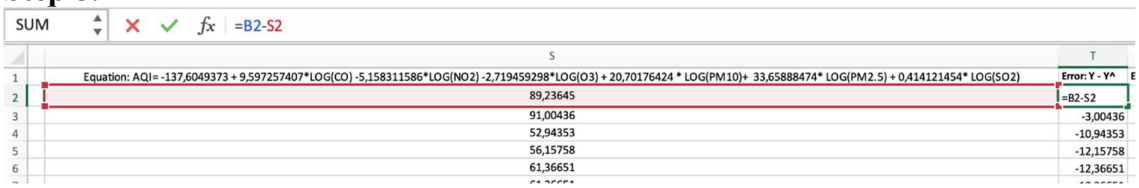


*Figure 62: Calculating Y - Y^*

Step 9: Calculate the Error^2: (Y-Y^)^2

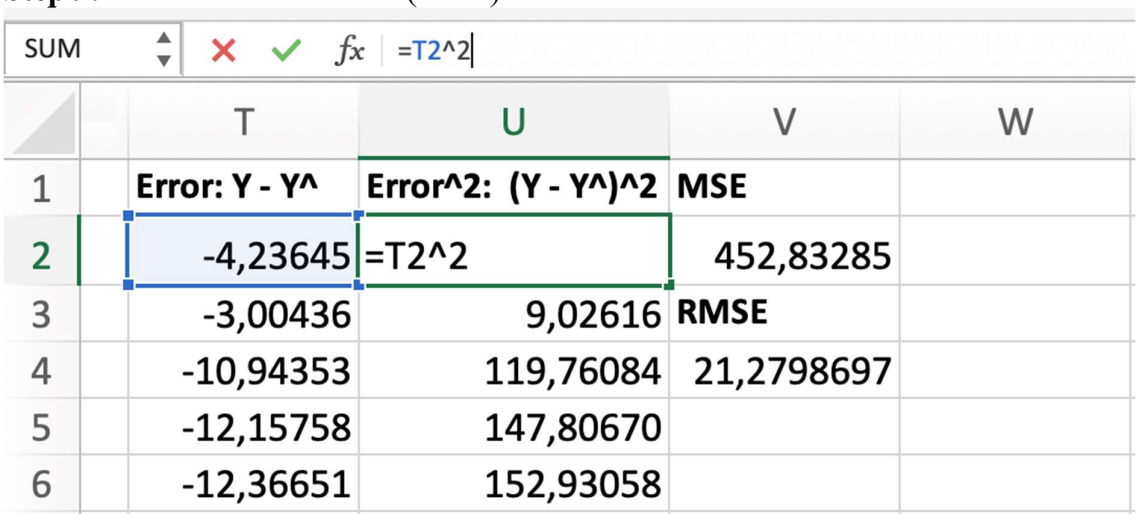


Figure 63:Calculating (Y-Y^)^2.

Step 10: Calculate MSE

SUM =SUM(U2:U332)/COUNT(U2:U332)

	T	U	V	W
1	Error: Y - Y^	Error^2: (Y - Y^)^2	MSE	
2	-4,23645	17,94747	U332)	
3	-3,00436	9,02616	RMSE	
4	-10,94353	119,76084	21,2798697	
5	-12,15758	147,80670		
6	-12,36651	152,93058		
7	-12,36651	152,93058		
8	-2,24140	5,02388		
9	-5,10417	26,05259		

Figure 64: Calculating MSE.

=> With an MSE = 452.83285, your model's mean squared error is relatively low, indicating a good fit to the data.

b) Lab 4:
We have ARIMA (3, 0, 0) model. Means that:

$$Y_t = \mu + \varphi_1(Y_{t-1}) + \varphi_2(Y_{t-2}) + \varphi_3(Y_{t-3})$$

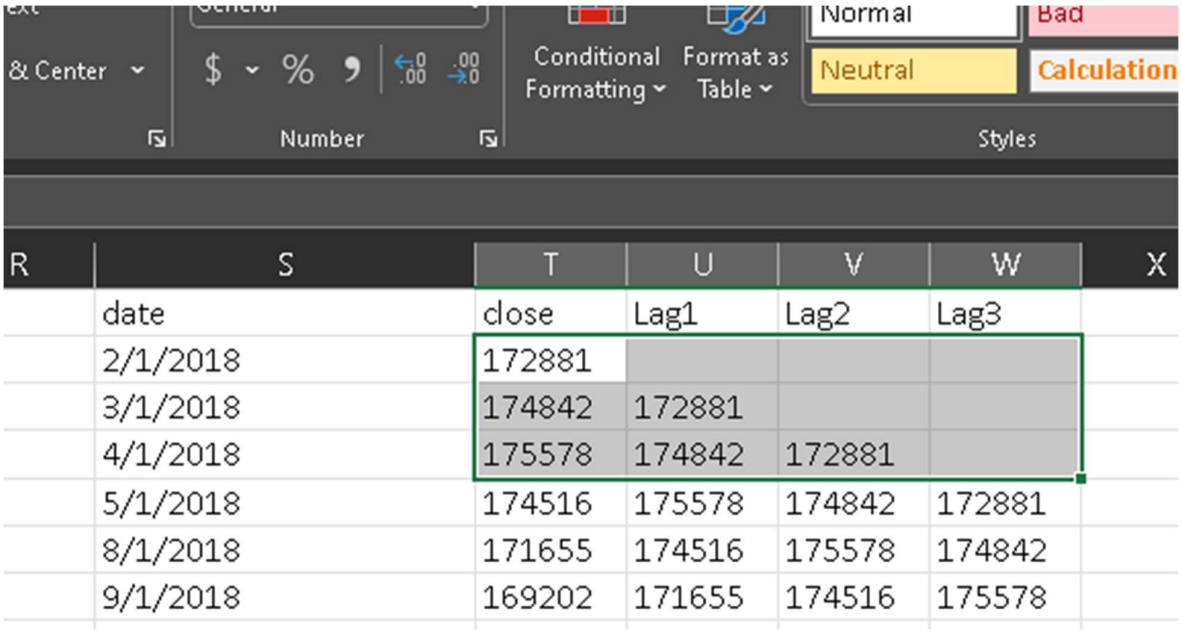
Figure 65: Equation of ARIMA.

Step 1: Calculate the lag to find the AR(q).

S	T	U	V	W	X
date	close	Lag1	Lag2	Lag3	
2/1/2018	172881				
3/1/2018	174842	172881			
4/1/2018	175578	174842	172881		
5/1/2018	174516	175578	174842	172881	
8/1/2018	171655	174516	175578	174842	
9/1/2018	169202	171655	174516	175578	
10/1/2018	172472	169202	171655	174516	
11/1/2018	171655	172472	169202	171655	

Figure 66: Calculate Lag 1,2,3.

Step 2: Delete all rows which have empty value.



R	S	T	U	V	W	X
	date	close	Lag1	Lag2	Lag3	
	2/1/2018	172881				
	3/1/2018	174842	172881			
	4/1/2018	175578	174842	172881		
	5/1/2018	174516	175578	174842	172881	
	8/1/2018	171655	174516	175578	174842	
	9/1/2018	169202	171655	174516	175578	

Figure 67: Delete row which has empty value.

Step 3: After deleting rows which have empty value, we have the final dataset.

S	T	U	V	W
date	close	Lag1	Lag2	Lag3
5/1/2018	174516	175578	174842	172881
8/1/2018	171655	174516	175578	174842
9/1/2018	169202	171655	174516	175578
10/1/2018	172472	169202	171655	174516
11/1/2018	171655	172472	169202	171655
12/1/2018	171573	171655	172472	169202

Figure 68: Data after deleting empty values.

Step 4: Choose Data Analysis -> Click Regression -> Click OK.

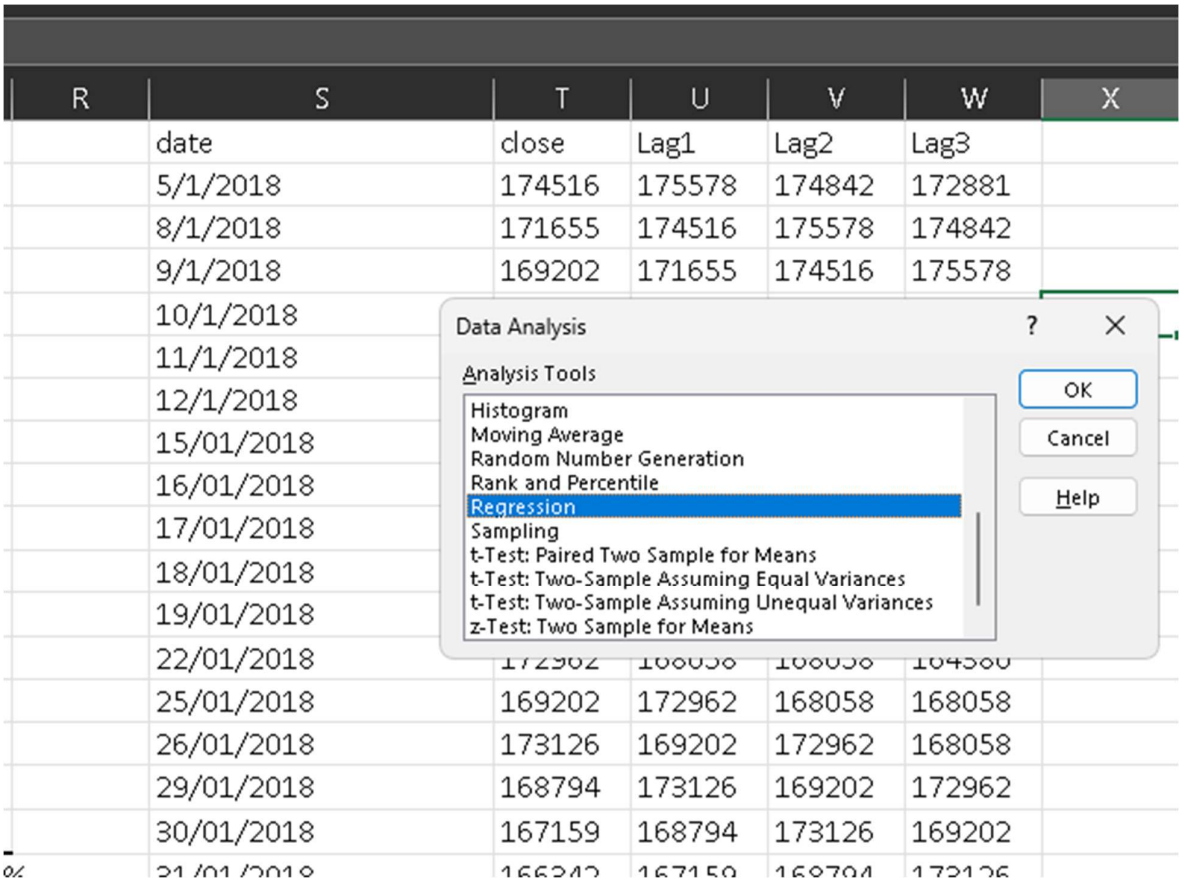
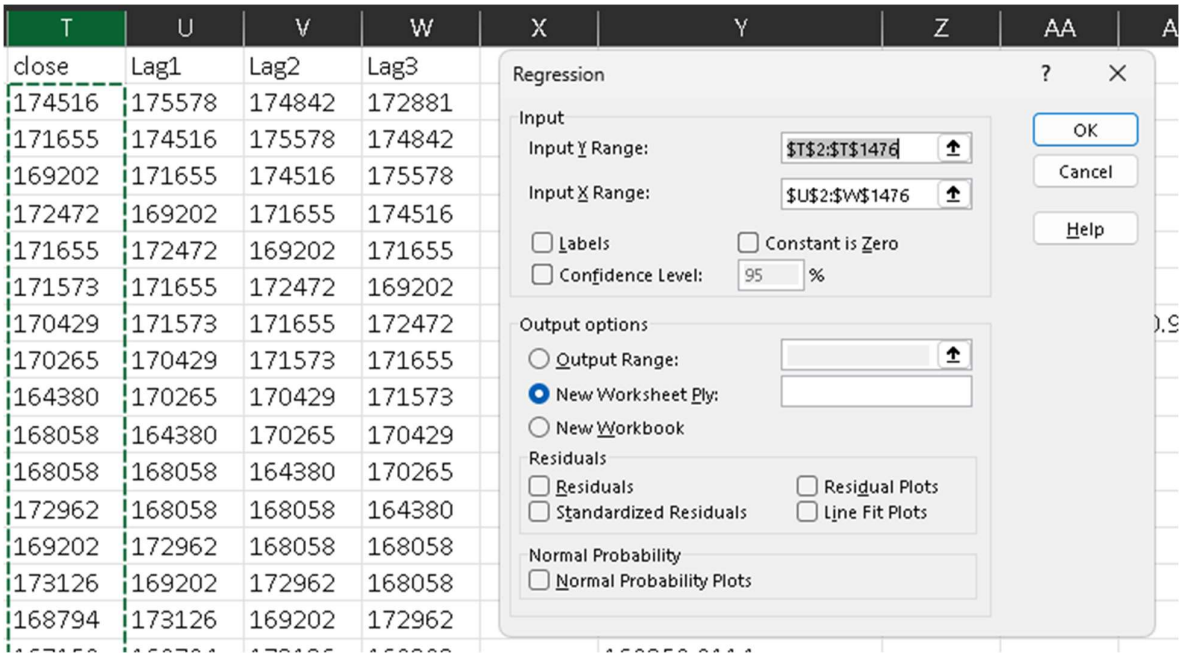


Figure 69: Dialog of regression.

Step 5: Then, we choose 'close' column to be the input Y range.
Lag1, Lag2, Lag3 columns to be the input X range.



close	Lag1	Lag2	Lag3
174516	175578	174842	172881
171655	174516	175578	174842
169202	171655	174516	175578
172472	169202	171655	174516
171655	172472	169202	171655
171573	171655	172472	169202
170429	171573	171655	172472
170265	170429	171573	171655
164380	170265	170429	171573
168058	164380	170265	170429
168058	168058	164380	170265
172962	168058	168058	164380
169202	172962	168058	168058
173126	169202	172962	168058
168794	173126	169202	172962

Figure 70: Enter X & Y values to regression.

Step 6: Click "OK" then we have the coefficients table.

SUMMARY OUTPUT

Regression Statistics	
Multiple R	0.99741
R Square	0.99483
Adjusted F	0.99482
Standard E	1718.21
Observati	1475

ANOVA

	df	*SS*	*MS*	*F*	*gnificance F*
Regressior	3	8.4E+11	2.78355E+11	94285.4	0
Residual	1471	4.3E+09	2952259.587		
Total	1474	8.4E+11			

	Coefficients	*andard Err*	*t Stat*	*P-value*	*Lower 95%*	*Jpper 95%*	*ower 95.0%*	*pper 95.0%*
Intercept	507.2	180.43	2.811057742	0.005	153.272	861.128	153.272	861.128
Lag 1	0.98779	0.02607	37.8903196	8E-220	0.93665	1.03893	0.93665	1.03893
Lag 2	-0.0015	0.03665	-0.042150597	0.96638	-0.0734	0.07035	-0.0734	0.07035
Lag3	0.00755	0.02597	0.290641491	0.77137	-0.0434	0.05849	-0.0434	0.05849

Figure 71: Result of regression.

Step 7: After having all necessary values, we start to predict based on this equation:

$$Y_t = 507,2 + 0.98779Y_{t-1} - 0.0015Y_{t-2} + 0.00755Y_{t-3}$$

Figure 72: Equation of ARIMA

S	T	U	V	W	X	Y
date	close	Lag1	Lag2	Lag3		Predict
5/1/2018	174516	175578	174842	172881		174984.3812
8/1/2018	171655	174516	175578	174842		173949.0497
9/1/2018	169202	171655	174516	175578		171130.1324
10/1/2018	172472	169202	171655	174516		168703.3569
11/1/2018	171655	172472	169202	171655		171915.5091
12/1/2018	171573	171655	172472	169202		171085.0596
15/01/2018	170429	171573	171655	172472		171029.9748
16/01/2018	170265	170429	171573	171655		169893.8977

Figure 73: Result of prediction.

Step 8: Calculate Error: Y – Y^

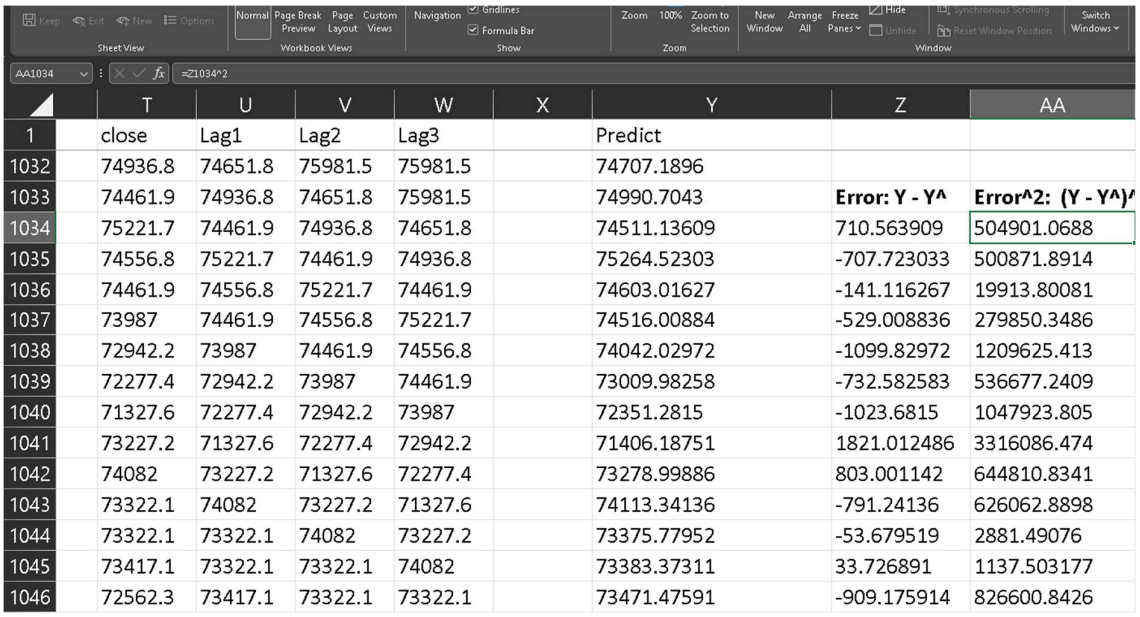
Sheet View | Workbook Views | Show | Zoom | Window

Z1034 | =T1034-Y1034

	T	U	V	W	X	Y	Z
1	close	Lag1	Lag2	Lag3		Predict	
1032	74936.8	74651.8	75981.5	75981.5		74707.1896	
1033	74461.9	74936.8	74651.8	75981.5		74990.7043	**Error: Y - Y^**
1034	75221.7	74461.9	74936.8	74651.8		74511.13609	710.563909
1035	74556.8	75221.7	74461.9	74936.8		75264.52303	-707.723033
1036	74461.9	74556.8	75221.7	74461.9		74603.01627	-141.116267
1037	73987	74461.9	74556.8	75221.7		74516.00884	-529.008836
1038	72942.2	73987	74461.9	74556.8		74042.02972	-1099.82972
1039	72277.4	72942.2	73987	74461.9		73009.98258	-732.582583
1040	71327.6	72277.4	72942.2	73987		72351.2815	-1023.6815
1041	73227.2	71327.6	72277.4	72942.2		71406.18751	1821.012486
1042	74082	73227.2	71327.6	72277.4		73278.99886	803.001142
1043	73322.1	74082	73227.2	71327.6		74113.34136	-791.24136
1044	73322.1	73322.1	74082	73227.2		73375.77952	-53.679519
1045	73417.1	73322.1	73322.1	74082		73383.37311	33.726891
1046	72562.3	73417.1	73322.1	73322.1		73471.47591	-909.175914

Figure 74: Calculating Y - Y^.

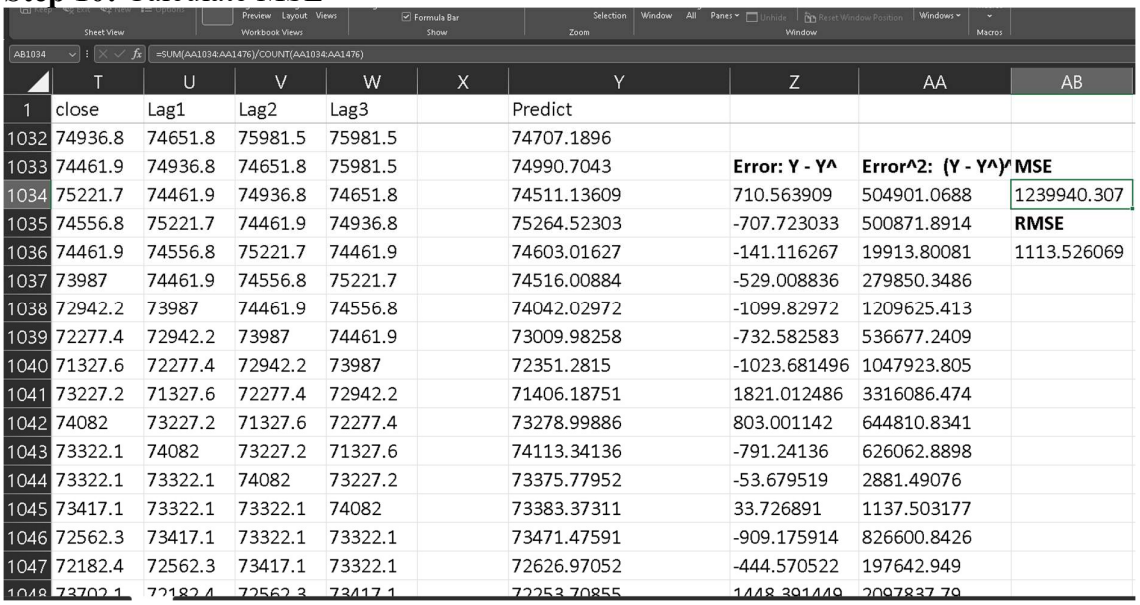
Step 9: Calculate Error^2: (Y – Y^) ^2



	T	U	V	W	X	Y	Z	AA
1	close	Lag1	Lag2	Lag3		Predict		
1032	74936.8	74651.8	75981.5	75981.5		74707.1896		
1033	74461.9	74936.8	74651.8	75981.5		74990.7043	**Error: Y - Y^**	**Error^2: (Y - Y^)^**
1034	75221.7	74461.9	74936.8	74651.8		74511.13609	710.563909	504901.0688
1035	74556.8	75221.7	74461.9	74936.8		75264.52303	-707.723033	500871.8914
1036	74461.9	74556.8	75221.7	74461.9		74603.01627	-141.116267	19913.80081
1037	73987	74461.9	74556.8	75221.7		74516.00884	-529.008836	279850.3486
1038	72942.2	73987	74461.9	74556.8		74042.02972	-1099.82972	1209625.413
1039	72277.4	72942.2	73987	74461.9		73009.98258	-732.582583	536677.2409
1040	71327.6	72277.4	72942.2	73987		72351.2815	-1023.6815	1047923.805
1041	73227.2	71327.6	72277.4	72942.2		71406.18751	1821.012486	3316086.474
1042	74082	73227.2	71327.6	72277.4		73278.99886	803.001142	644810.8341
1043	73322.1	74082	73227.2	71327.6		74113.34136	-791.24136	626062.8898
1044	73322.1	73322.1	74082	73227.2		73375.77952	-53.679519	2881.49076
1045	73417.1	73322.1	73322.1	74082		73383.37311	33.726891	1137.503177
1046	72562.3	73417.1	73322.1	73322.1		73471.47591	-909.175914	826600.8426

Figure 75: Calculating (Y-Y^)^2.

Step 10: Calculate MSE



	T	U	V	W	X	Y	Z	AA	AB
1	close	Lag1	Lag2	Lag3		Predict			
1032	74936.8	74651.8	75981.5	75981.5		74707.1896			
1033	74461.9	74936.8	74651.8	75981.5		74990.7043	**Error: Y - Y^**	**Error^2: (Y - Y^)^**	**MSE**
1034	75221.7	74461.9	74936.8	74651.8		74511.13609	710.563909	504901.0688	1239940.307
1035	74556.8	75221.7	74461.9	74936.8		75264.52303	-707.723033	500871.8914	**RMSE**
1036	74461.9	74556.8	75221.7	74461.9		74603.01627	-141.116267	19913.80081	1113.526069
1037	73987	74461.9	74556.8	75221.7		74516.00884	-529.008836	279850.3486	
1038	72942.2	73987	74461.9	74556.8		74042.02972	-1099.82972	1209625.413	
1039	72277.4	72942.2	73987	74461.9		73009.98258	-732.582583	536677.2409	
1040	71327.6	72277.4	72942.2	73987		72351.2815	-1023.681496	1047923.805	
1041	73227.2	71327.6	72277.4	72942.2		71406.18751	1821.012486	3316086.474	
1042	74082	73227.2	71327.6	72277.4		73278.99886	803.001142	644810.8341	
1043	73322.1	74082	73227.2	71327.6		74113.34136	-791.24136	626062.8898	
1044	73322.1	73322.1	74082	73227.2		73375.77952	-53.679519	2881.49076	
1045	73417.1	73322.1	73322.1	74082		73383.37311	33.726891	1137.503177	
1046	72562.3	73417.1	73322.1	73322.1		73471.47591	-909.175914	826600.8426	
1047	72182.4	72562.3	73417.1	73322.1		72626.97052	-444.570522	197642.949	
1048	73702.1	72182.4	72562.3	73417.1		72253.70855	1448.391449	2097837.79	

Figure 76: Calculating MSE.

=> The mean square error (MSE) is 1,239,940,307, which shows a relatively high mean square difference between the predicted value and the actual value, thus proving that the ARIMA model is not the best fit for the dataset.

2. Using R language:

a) Lab 3:

Step 1: Import the dataset.

```
> nls_data <- read.csv(file.choose(), sep = ";")
>
> nls_data
    Station.ID AQI.index                                Station.name Dominent.pollutant CO NO2 O3 PM10 PM2.5
1        13250         1 Lào Cai/KCN TALOONG1, Vietnam                pm10 43   1  2    1    23
2        13250         1 Lào Cai/KCN TALOONG1, Vietnam                pm10 53   1  2    1    23
3        13250         1 Lào Cai/KCN TALOONG1, Vietnam                pm10 38   1  3    1    18
4        13250         1 Lào Cai/KCN TALOONG1, Vietnam                pm10 34   2  2    1    21
5        13250         1 Lào Cai/KCN TALOONG1, Vietnam                pm10 39   3  2    1     2
6        13026         1  Hà Nội/Chi cục BVMT, Vietnam                pm25  6  27 1    1     1
7        13026         1  Hà Nội/Chi cục BVMT, Vietnam                pm25  6  27 1    1     1
8        13026         1  Hà Nội/Chi cục BVMT, Vietnam                pm25  6  27 1    1     1
```

Figure 77: Import data.

Step 2: Using attach to interact with dataset.

```
> attach(nls_data)
```

Figure 78: Attach data.

Step 3: Calculate log of dependent variables.

```
> nls_data$log_CO <- log(nls_data$CO)
> nls_data$log_NO2 <- log(nls_data$NO2)
> nls_data$log_O3 <- log(nls_data$O3)
> nls_data$log_PM10 <- log(nls_data$PM10)
> nls_data$log_PM2.5 <- log(nls_data$PM2.5)
> nls_data$log_SO2 <- log(nls_data$SO2)
```

Figure 79: Calculate log.

Step 4: Define a vector independent_vars containing the names of the log-transformed variables:

```
> independent_vars <- c('log_CO', 'log_NO2', 'log_O3', 'log_PM10', 'log_PM2.5', 'log_SO2')
```

Figure 49. Define a vector independent_vars

Step 5: The “lm” function and choosing the fit values to performance multivariable linear regression.

```
> reg = lm(AQI.index ~ log_CO + log_NO2 + log_O3 + log_PM10 + log_PM2.5 + log_SO2, data = nls_data)
> summary(reg)
```

Figure 80: Performance multivariablee linear regression.

```
Call:
lm(formula = AQI.index ~ log_CO + log_NO2 + log_O3 + log_PM10 +
    log_PM2.5 + log_SO2, data = nls_data)

Residuals:
    Min      1Q  Median      3Q     Max
-46.487 -15.289  -5.708   8.103 101.066

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -137.6049     8.7992 -15.638  < 2e-16 ***
log_CO        22.0985     4.3494   5.081 6.36e-07 ***
log_NO2      -11.8775     3.9755  -2.988  0.00303 **
log_O3        -6.2618     3.3028  -1.896  0.05886 .
log_PM10      47.6676     7.9274   6.013 4.91e-09 ***
log_PM2.5     77.5024     7.4693  10.376  < 2e-16 ***
log_SO2        0.9535     3.4264   0.278  0.78096
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 21.51 on 324 degrees of freedom
Multiple R-squared:  0.8253,    Adjusted R-squared:  0.8221
F-statistic: 255.1 on 6 and 324 DF,  p-value: < 2.2e-16
```

Figure 81: Summary Output.

Step 6: Calculate residuals

```
> # Tính residuals (phần dư)
> residuals <- residuals(reg)
```

Figure 82: Caluclating the residuals.

Step 7: Calculate MSE

```
> # Tính Mean Squared Error (MSE)
> mse <- mean(residuals^2)
> cat("MSE:", mse, "\n")
MSE: 452.8329
```

Figure 83: Calculating the MSE.

With an MSE = 452.8329, your model's mean squared error is relatively low, indicating a good fit to the data.

b) Lab 4:

Step 1: Import the dataset

```
> # Read data from a CSV file
> df <- read.csv(file.choose(), sep = ",")
```

Figure 84: Import the dataset.

Step 2: Set row names to index

```
> # Set row names to the 'date' column
> rownames(df) <- df$date
```

Figure 85: Set row names.

Step 3: Convert data to numeric format.

```
> # Convert 'close' column to numeric format
> df$close <- as.numeric(gsub(",", ".", df$close))
```

Figure 86: Convert the data to numeric format.

Step 4: Create a time series object.

```
> # Create a time series object
> data <- ts(na.omit(df$close), frequency = 1, start = c(02, 01, 2018))
```

Figure 87: Create time series object.

Step 5: Plot data

```
> # Plot the time series data
> plot(data)
```

Figure 88: Plot data.

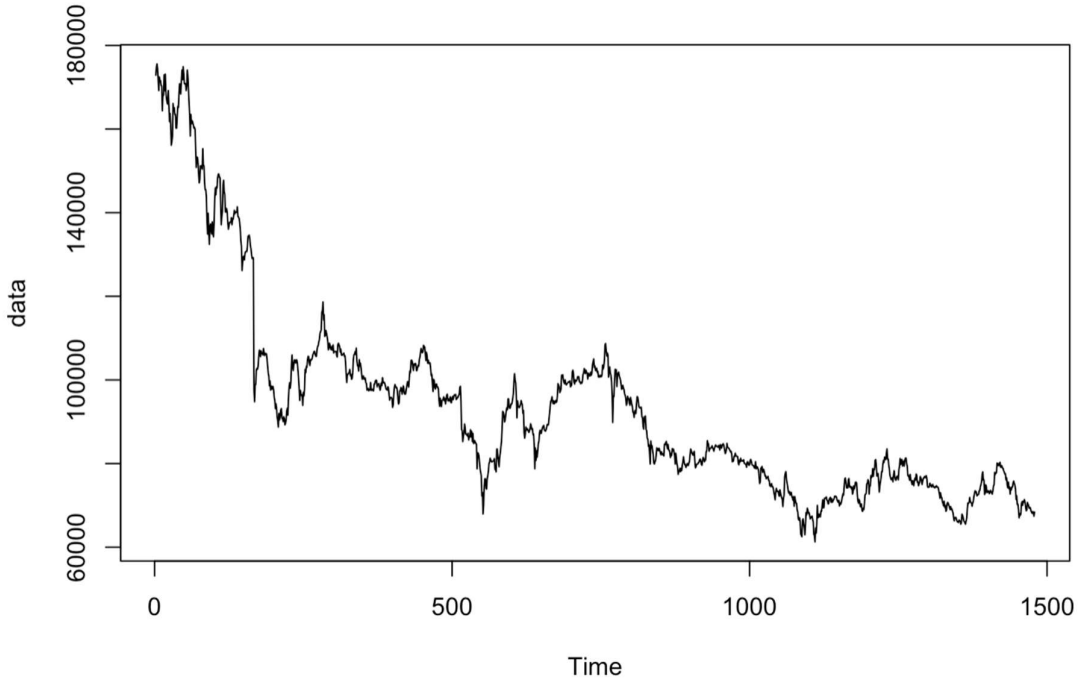


Figure 89: Result chart.

Step 6: Calculate the Dickey-Fuller test.

```
> # Perform Augmented Dickey-Fuller test for stationarity
> adf.test(data)

        Augmented Dickey-Fuller Test

data:  data
Dickey-Fuller = -3.0937, Lag order = 11, p-value = 0.1153
alternative hypothesis: stationary
```

Figure 90: Calculating Dickey-Fuller.

Step 7: Split data to train & Test

```
> # Split the data into training and testing sets
> train <- head(df, 1034)
> test <- tail(df, 443)
```

Figure 91: Split dataset.

Step 8: Create time series objects.

```
> # Create time series objects
> train_ts <- ts(train$close, frequency = 1, start = c(02, 01, 2018))
> test_ts <- ts(test$close, frequency = 1, start = c(25, 02, 2022))
```

Figure 92: Create time series objects.

Step 9: Find the fit ARIMA model by auto.arima

```
> # Use auto.arima with the time series object
> arima_model <- forecast::auto.arima(train_ts, trace = TRUE)

 Fitting models using approximations to speed things up...

 ARIMA(2,1,2) with drift         : Inf
 ARIMA(0,1,0) with drift         : 18545.04
 ARIMA(1,1,0) with drift         : 18546.91
 ARIMA(0,1,1) with drift         : 18547.05
 ARIMA(0,1,0)                    : 18545.53
 ARIMA(1,1,1) with drift         : 18548.88

 Now re-fitting the best model(s) without approximations...

 ARIMA(0,1,0) with drift         : 18560.16

 Best model: ARIMA(0,1,0) with drift
```

Figure 93: Find the fit ARIMA model.

Step 10: Train the data with ARIMA model (0,1,0)

```
> # Fit ARIMA model to the training set
> fitARIMAtrain <- arima(train_ts, order = c(0, 1, 0))
```

Figure 94: Training data.

Step 11: Forecast for the testing data.

```
> # Forecast for the testing set
> forecast_values <- forecast(fitARIMAtrain, h = length(test_ts))
```

Figure 95: Forecasting the testing set.

Step 12: Plot the forecast data.

```
> # Plot the forecast for the testing set
> plot(forecast_values)
```

Figure 96: Plot forecast data.

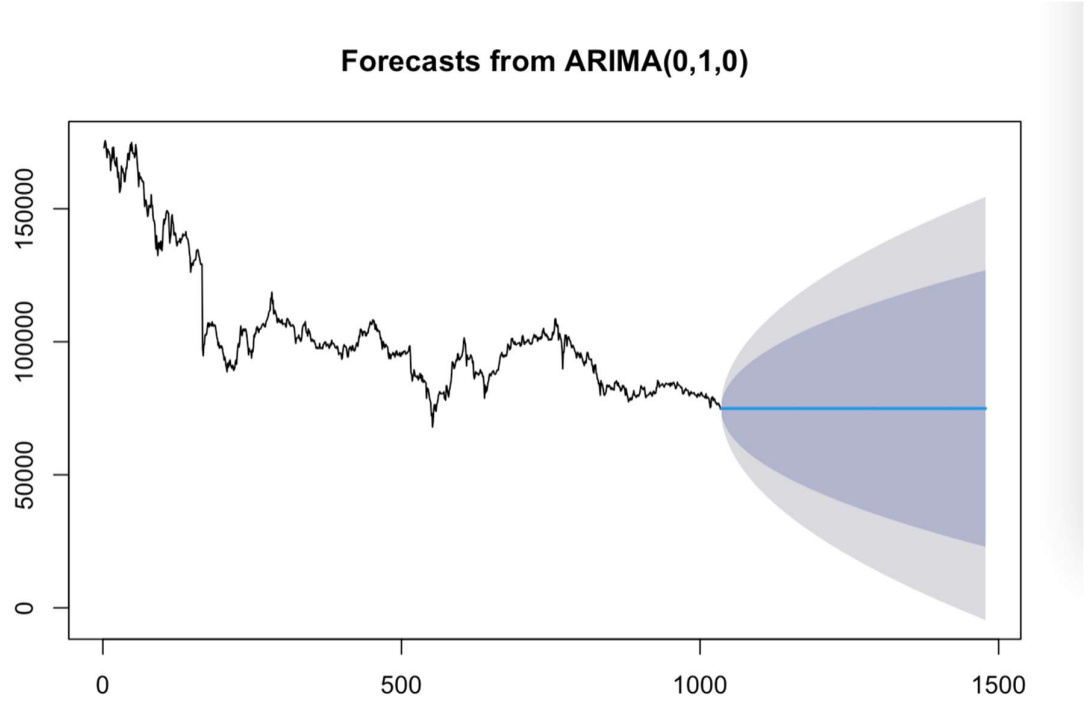


Figure 97: Result of forecast.

Step 13: Extract the forecasted values.

```
> # Extract the forecasted values
> predicted_values <- forecast_values$mean
```

Figure 98: Extract the forecasted values.

Step 14: Calculate the MSE

```
> # Check for missing or non-finite values
> if (any(is.na(test_ts)) || any(is.na(predicted_values)) ||
+     any(!is.finite(test_ts)) || any(!is.finite(predicted_values))) {
+     cat("Warning: Missing or non-finite values detected in actual or predicted values.\n")
+ } else {
+     # Create a data frame with actual and predicted values
+     merged_data <- merge(data.frame(actual = test_ts), data.frame(predicted = predicted_values), by = "row.names", all = TRUE)
+
+     # Rename columns
+     colnames(merged_data) <- c("date", "actual", "predicted")
+
+     # Calculate MSE and RMSE
+     mse <- mean((merged_data$actual - merged_data$predicted)^2, na.rm = TRUE)
+     rmse <- sqrt(mse)
+
+     # Check for NaN in MSE and RMSE
+     if (!is.finite(mse) || is.nan(mse) || !is.finite(rmse) || is.nan(rmse)) {
+         cat("Warning: NaN detected in MSE or RMSE.\n")
+     } else {
+         # Print MSE and RMSE
+         cat("MSE: ", mse, "\n")
+         cat("RMSE: ", rmse, "\n")
+
+         # Plot actual vs. predicted values
+         plot(merged_data$date, merged_data$actual, type = "l", col = "blue", ylab = "Close Price", xlab = "Time", main = "Actual vs. Predicted")
+         lines(merged_data$date, merged_data$predicted, col = "red")
+         legend("topright", legend = c("Actual: Blue", "Predicted: Red"), col = c("blue", "red"))
+     }
+ }
MSE:  21511117
RMSE:  4638.008
```

Figure 99: Calculating MSE.

=> The mean square error (MSE) is 21511117, which shows a relatively high mean square difference between the predicted value and the actual value, thus proving that the ARIMA model is not the best fit for the dataset.

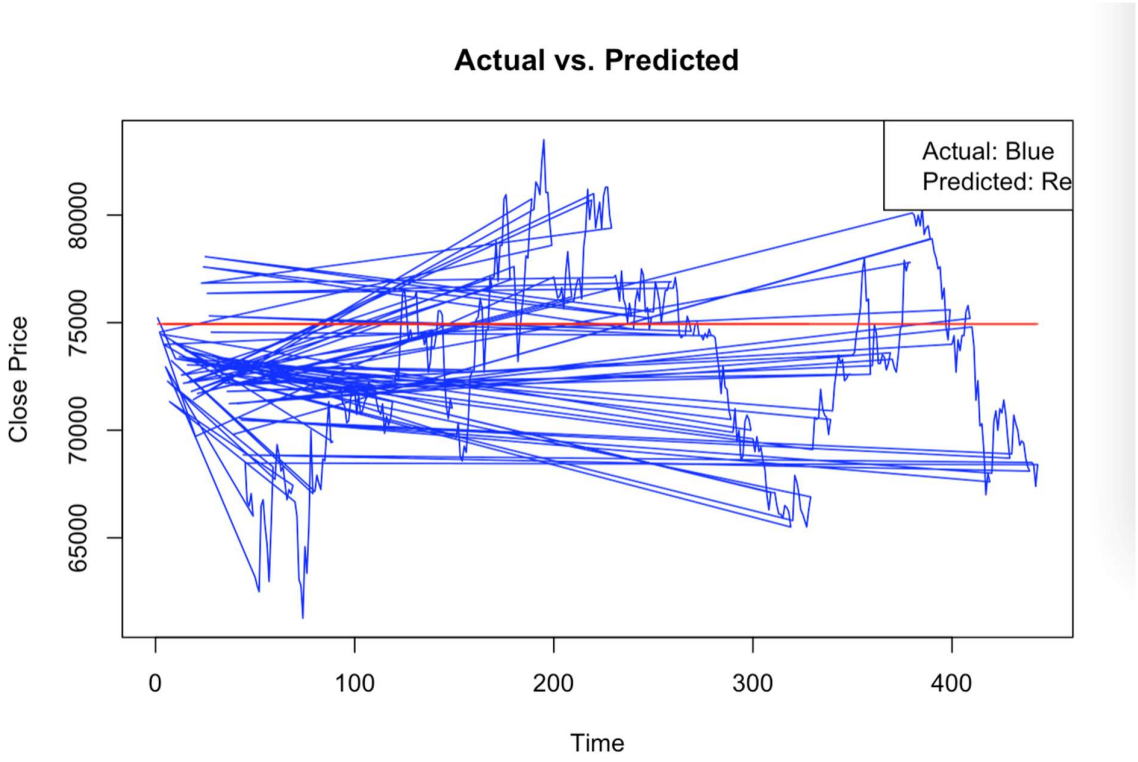


Figure 100: Result of testing the accuracy of models.

3. Using Python language:

a) Lab 3:

Step 1: Import the libraries to use.

```
[1]: import pandas as pd
     import numpy as np
     from sklearn.linear_model import LinearRegression
     import statsmodels.api as sm
```

Figure 101: Import the libraries.

Step 2: Import dataset.

```
[2]: data = pd.read_csv("historical_air_quality_2021_clean.csv", sep=";")
```

```
[3]: data
```

[3]:

	Station ID	AQI index	Station name	Dominent pollutant	CO	NO2	O3	PM10	PM2.5	SO2
0	12958	85.0	Bắc Ninh/Phong Cốc, Vietnam	pm25	3.0	9.0	34.0	66.0	85.0	10.0
1	12958	88.0	Bắc Ninh/Phong Cốc, Vietnam	pm25	3.0	6.0	22.0	58.0	88.0	10.0
2	13012	42.0	Gia Lai/phường Thống Nhất - Pleiku, Vietnam	pm25	2.0	4.0	25.0	34.0	42.0	11.0
3	13012	44.0	Gia Lai/phường Thống Nhất - Pleiku, Vietnam	pm25	2.0	5.0	17.0	37.0	44.0	11.0
4	13012	49.0	Gia Lai/phường Thống Nhất - Pleiku, Vietnam	pm25	2.0	3.0	25.0	37.0	49.0	11.0
...	...	...	...	...	...	...	...	...	...	...
326	12976	67.0	Bắc Ninh/Châu Khê, Vietnam	pm25	5.0	12.0	8.0	53.0	67.0	9.0
327	12976	67.0	Bắc Ninh/Châu Khê, Vietnam	pm25	5.0	12.0	8.0	53.0	67.0	9.0
328	12976	67.0	Bắc Ninh/Châu Khê, Vietnam	pm25	5.0	12.0	8.0	53.0	67.0	9.0
329	12976	83.0	Bắc Ninh/Châu Khê, Vietnam	pm25	6.0	7.0	8.0	54.0	83.0	9.0
330	13250	47.0	Lào Cai/KCN TALOONG1, Vietnam	pm25	34.0	3.0	3.0	7.0	47.0	99.0

331 rows × 10 columns

Figure 102: Import dataset.

Step 3: Calculate log of dependent variables.

```
[4]: data['log(CO)'] = np.log(data['CO'])
     data['log(NO2)'] = np.log(data['NO2'])
     data['log(O3)'] = np.log(data['O3'])
     data['log(PM10)'] = np.log(data['PM10'])
     data['log(PM2.5)'] = np.log(data['PM2.5'])
     data['log(SO2)'] = np.log(data['SO2'])
```

Figure 103: Calculate log of dependent variables.

Step 4: Get the dependent & independent variables

```
[5]: x = np.array(data[['log(CO)', 'log(NO2)', 'log(O3)', 'log(PM10)', 'log(PM2.5)', 'log(SO2)']]).reshape(
     y = np.array(data['AQI index']).reshape((-1, 1))
```

Figure 104: Get dependent & independent variables.

Step 5: Using Linear Regression() to gain the model follow X and Y variables and taking result.

```
[6]: reg = LinearRegression()
     reg.fit(x,y)
```

```
[6]: ▾ LinearRegression
     LinearRegression()
```

Figure 105: Using Linear Regression to gain the model.

The fit() function in Python is used to train a linear regression model. This function will find the regression coefficients of the model so that the sum of squared errors between the predicted values and the actual values is minimized.

- The first parameter x: The array containing the values of the independent variable.
- The second parameter y: The array containing the values of the dependent variable.

Step 6: Creating a regression table using MLR through OLS.

```
[6]: y = data['AQI index']

# For independent variables (x), you can select the desired columns
x = data[['log(CO)', 'log(NO2)', 'log(O3)', 'log(PM10)', 'log(PM2.5)', 'log(SO2)']]

# If you want to add a constant term to the independent variables, you can use the following:
x = sm.add_constant(x)

reg = sm.OLS(y,x).fit()
```

Figure 106: Creating a regression table.

Step 7: Calculate the residuals.

```
[7]: # Tính residuals (phần dư)
residuals = y - reg.predict(x)
```

Figure 107: Calculating the residuals.

Step 8: Calculate MSE

```
[8]: # Tính Mean Squared Error (MSE)
mse = np.mean(residuals**2)
print('MSE:', mse)

MSE: 452.8328547833633
```

Figure 108: Calculating MSE.

With an MSE = 452.83285, your model's mean squared error is relatively low, indicating a good fit to the data.

c) Lab 4:

Step 1: Import the libraries to use.

```
[1]: # Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
```

Figure 109: Import libraries.

Step 2: Import the dataset and split the data into

```
[2]: # Read dataset file
df = pd.read_csv('vnmvn.csv')
df = df[['close']]
df = df.dropna() # Drop missing values
df = df.reset_index(drop=True) # Reset the index

# Split the data into training, testing, and validation sets by 7:2:1
train_size = int(0.7 * len(df))
test_size = int(0.3 * len(df))

train_data = df[:train_size]
test_data = df[train_size:train_size+test_size]
```

Figure 110: Read dataset & split data.

Step 3: Train the data and find the fit ARIMA model.

```
[3]: # Training process
x_train = np.array(train_data.index).reshape(-1, 1)
y_train = np.array(train_data['close'])

# Find the best ARIMA model using auto_arima
from pmdarima.arima import auto_arima
model = auto_arima(y_train, trace=True, error_action='ignore', suppress_warnings=True)

# Fit the model
model.fit(y_train)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.70 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=18560.152, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=18561.221, Time=0.03 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=18561.227, Time=0.03 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=18560.656, Time=0.01 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=18563.234, Time=0.07 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0] intercept
Total fit time: 0.849 seconds
```

```
[3]: ARIMA
ARIMA(0,1,0)(0,0,0)[0] intercept
```

Figure 111: Find the fit ARIMA model.

Step 4: Predict for the testing data.

```
[4]: last_index = df.index[-1]
last_data = pd.RangeIndex(start=last_index + 1, stop=last_index + test_size + 1, step=1)

# Create an array of consecutive integers starting from last_index
x_predict = np.array(range(last_index + 1, last_index + test_size + 1)).reshape(-1, 1)

# Toggle output scrolling onverted to an integer
test_size = int(test_size)

# Predict the closing prices for the next period based on the length of the test data
y_predict = model.predict(n_periods=test_size)

# Print the predicted closing prices for the next period
print('Predicted closing prices for the next period:')
print(y_predict)
```

```
Predicted closing prices for the next period:
[74841.98470474 74747.16940949 74652.35411423 74557.53881897
 74462.72352372 74367.90822846 74273.0929332  74178.27763795
 74083.46234269 73988.64704743 73893.83175218 73799.01645692
```

Figure 112: Predict test data.

Step 5: Calculate MSE

```
[5]: # Assuming 'test_data' is a NumPy array
test_data = np.reshape(test_data, (test_data.shape[0], 1))

# Reshape y_predict to match the shape of test_data
y_predict = np.reshape(y_predict, (y_predict.shape[0], 1))

# Calculate the MSE
mse = np.mean((y_predict - test_data)**2)
print('MSE:', mse)

# Calculate the RMSE
rmse = np.sqrt(np.mean((y_predict - test_data)**2))
print('RMSE:', rmse)
```

```
MSE: 539440210.3792304
RMSE: 23225.852199203164
```

Figure 113: Calculating MSE.

Step 6: Plot the predict data.

```
[14]: # Create an index for the actual test data
actual_index = np.arange(len(test_data))
# Plotting the actual values and predicted values for the test set
plt.plot(actual_index, test_data, label='Actual Test Data')
plt.plot(actual_index, y_predict, label='Predicted Test Data')
plt.legend()
plt.xlabel('Index')
plt.ylabel('Close Price')
plt.title('Actual vs. Predicted Test Data')
plt.show()
```

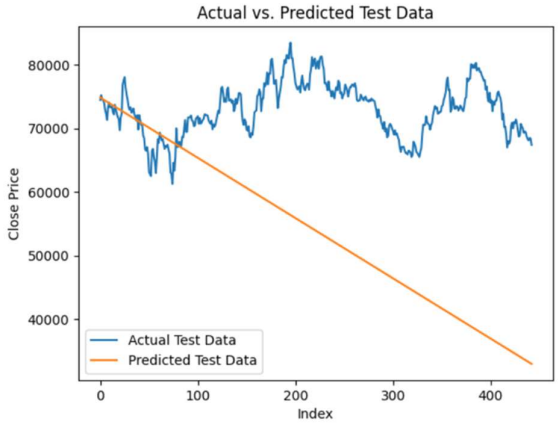


Figure 114: Plot result.

=> The mean square error (MSE) is 539440210.3792304, which shows a relatively high mean square difference between the predicted value and the actual value, thus proving that the ARIMA model is not the best fit for the dataset.

IV. Root Mean Squared Error (RMSE):

1. Using Excel:

a) Lab 3:

Calculate the same with MSE and then SQRT(MSE) we will have the RMSE.

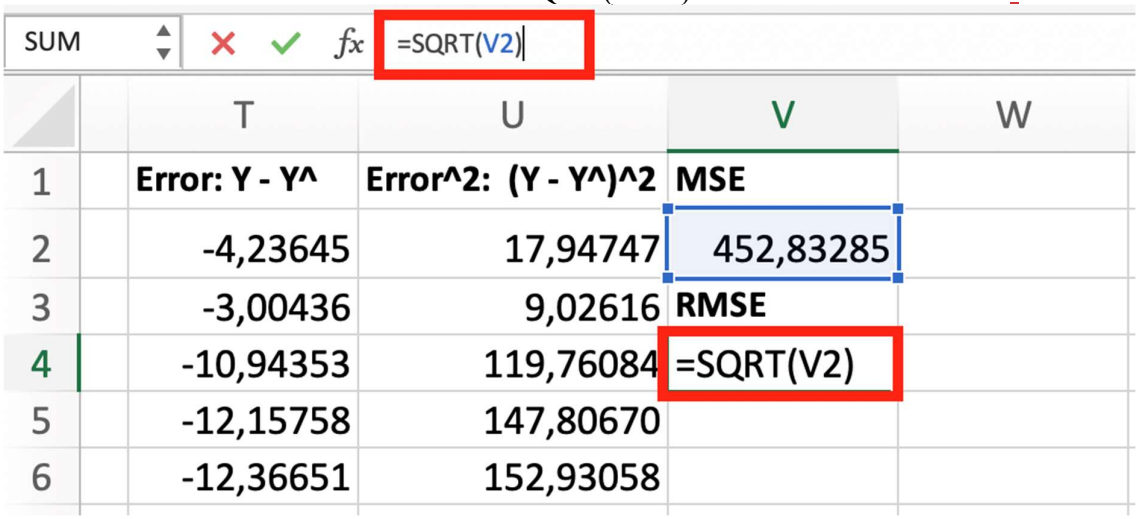
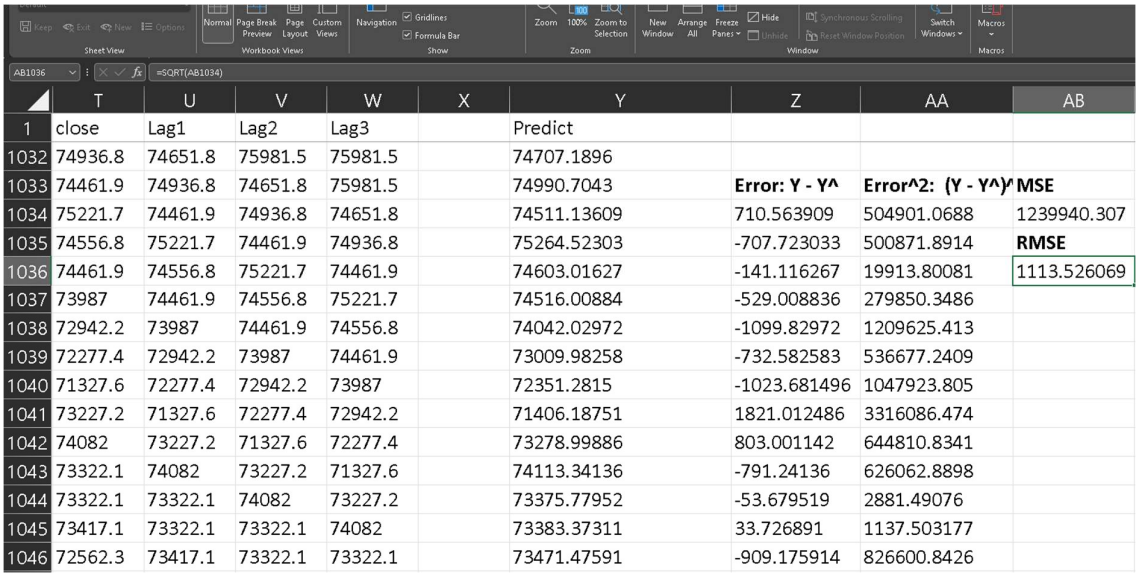


Figure 115: Calculating RMSE.

➔ The corresponding RMSE = 21.2798697 represents the average magnitude of prediction errors, and a lower RMSE generally suggests better overall model accuracy.

b) Lab 4:

Calculate the same with MSE and then SQRT(MSE) we will have the RMSE.



	T	U	V	W	X	Y	Z	AA	AB
1	close	Lag1	Lag2	Lag3		Predict			
1032	74936.8	74651.8	75981.5	75981.5		74707.1896			
1033	74461.9	74936.8	74651.8	75981.5		74990.7043	**Error: Y - Y^**	**Error^2: (Y - Y^)^**	**MSE**
1034	75221.7	74461.9	74936.8	74651.8		74511.13609	710.563909	504901.0688	1239940.307
1035	74556.8	75221.7	74461.9	74936.8		75264.52303	-707.723033	500871.8914	**RMSE**
1036	74461.9	74556.8	75221.7	74461.9		74603.01627	-141.116267	19913.80081	1113.526069
1037	73987	74461.9	74556.8	75221.7		74516.00884	-529.008836	279850.3486	
1038	72942.2	73987	74461.9	74556.8		74042.02972	-1099.82972	1209625.413	
1039	72277.4	72942.2	73987	74461.9		73009.98258	-732.582583	536677.2409	
1040	71327.6	72277.4	72942.2	73987		72351.2815	-1023.681496	1047923.805	
1041	73227.2	71327.6	72277.4	72942.2		71406.18751	1821.012486	3316086.474	
1042	74082	73227.2	71327.6	72277.4		73278.99886	803.001142	644810.8341	
1043	73322.1	74082	73227.2	71327.6		74113.34136	-791.24136	626062.8898	
1044	73322.1	73322.1	74082	73227.2		73375.77952	-53.679519	2881.49076	
1045	73417.1	73322.1	73322.1	74082		73383.37311	33.726891	1137.503177	
1046	72562.3	73417.1	73322.1	73322.1		73471.47591	-909.175914	826600.8426	

Figure 116: Calculating RMSE.

=> The Root Mean Square Error (RMSE) value is 1113.526069, which indicates that the average level of prediction errors in the model is high, so the model does not fit the best with the dataset, and needs to be changed to another model.

2. Using R language:

a) Lab 3:

Calculate the same with MSE and then SQRT(MSE) we will have the RMSE.

```
> # Tính Root Mean Squared Error (RMSE)
> rmse <- sqrt(mse)
> cat("RMSE:", rmse, "\n")
RMSE: 21.27987
```

Figure 117: Calculating RMSE

➔ The corresponding RMSE = 21.27987 represents the average magnitude of prediction errors, and a lower RMSE generally suggests better overall model accuracy.

b) Lab 4:

Calculate the same with MSE and then SQRT(MSE) we will have the RMSE.

```
> # Check for missing or non-finite values
> if (any(is.na(test_ts)) || any(is.na(predicted_values)) ||
+     any(!is.finite(test_ts)) || any(!is.finite(predicted_values))) {
+     cat("Warning: Missing or non-finite values detected in actual or predicted values.\n")
+ } else {
+     # Create a data frame with actual and predicted values
+     merged_data <- merge(data.frame(actual = test_ts), data.frame(predicted = predicted_values), by = "row.names", all = TRUE)
+
+     # Rename columns
+     colnames(merged_data) <- c("date", "actual", "predicted")
+
+     # Calculate MSE and RMSE
+     mse <- mean((merged_data$actual - merged_data$predicted)^2, na.rm = TRUE)
+     rmse <- sqrt(mse)
+
+     # Check for NaN in MSE and RMSE
+     if (!is.finite(mse) || is.nan(mse) || !is.finite(rmse) || is.nan(rmse)) {
+         cat("Warning: NaN detected in MSE or RMSE.\n")
+     } else {
+         # Print MSE and RMSE
+         cat("MSE: ", mse, "\n")
+         cat("RMSE: ", rmse, "\n")
+
+         # Plot actual vs. predicted values
+         plot(merged_data$date, merged_data$actual, type = "l", col = "blue", ylab = "Close Price", xlab = "Time", main = "Actual vs. Predicted")
+         lines(merged_data$date, merged_data$predicted, col = "red")
+         legend("topright", legend = c("Actual: Blue", "Predicted: Red"), col = c("blue", "red"))
+     }
+ }
MSE:  21511117
RMSE:  4638.008
```

Figure 118: Calculating RMSE.

The Root Mean Square Error (RMSE) value is 4638.008, which indicates that the average level of prediction errors in the model is high, so the model does not fit the best with the dataset and needs to be changed to another model.

3. Using Python languages:

a) Lab 3:

Calculate the same with MSE and then SQRT(MSE) we will have the RMSE.

```
[9]: # Tính Root Mean Squared Error (RMSE)
     rmse = np.sqrt(mse)
     print('RMSE:', rmse)

     RMSE: 21.279869707856843
```

Figure 119: Calculating RMSE.

a. The corresponding RMSE = 21.2798697 represents the average magnitude of prediction errors, and a lower RMSE generally suggests better overall model accuracy.

b) Lab 4:

Calculate the same with MSE and then SQRT(MSE) we will have the RMSE.

```
[5]: # Assuming 'test_data' is a NumPy array
     test_data = np.reshape(test_data, (test_data.shape[0], 1))

     # Reshape y_predict to match the shape of test_data
     y_predict = np.reshape(y_predict, (y_predict.shape[0], 1))

     # Calculate the MSE
     mse = np.mean((y_predict - test_data)**2)
     print('MSE:', mse)

     # Calculate the RMSE
     rmse = np.sqrt(np.mean((y_predict - test_data)**2))
     print('RMSE:', rmse)

     MSE: 539440210.3797304
     RMSE: 23225.852199203164
```

Figure 120: Calculating RMSE.

The Root Mean Square Error (RMSE) value is 23225.852199, which indicates that the average level of prediction errors in the model is high, so the model does not fit the best with the dataset, and needs to be changed to another model.

V. Coefficient of Determination (R-squared):

1. Using Excel:

a. Lab 3:

Step 1: We set “AQI Index” " to the y value that represents the dependent value in regression.

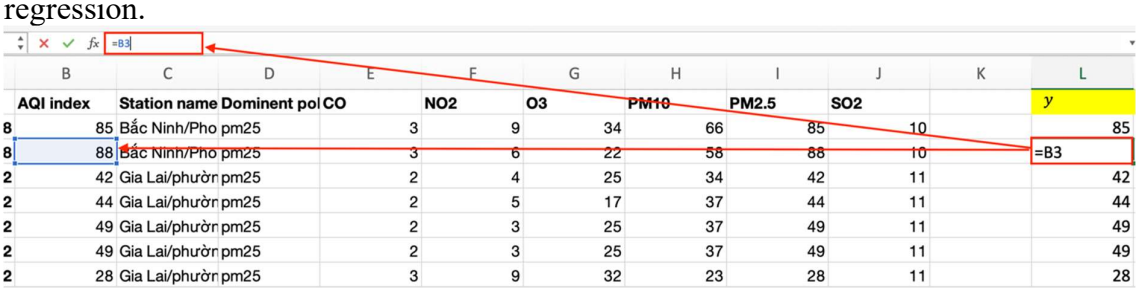


Figure 121. Setting y value

Step 2: Calculating the mean value $\bar{y}$.

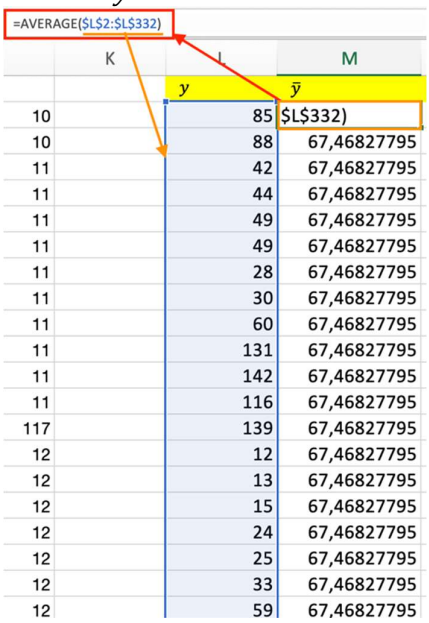


Figure 122. Calculating $\bar{y}$

Step 3: Calculating ANOVA to find equation through Regression tool:

SUMMARY OUTPUT

Regression Statistics	
Multiple R	0,995162572
R Square	0,990348544
Adjusted R S	0,990169813
Standard Erro	5,05556491
Observations	331

ANOVA

	df	*SS*	*MS*	*F*	*Significance F*
Regression	6	849727,386	141621,231	5541,01063	0
Residual	324	8281,03065	25,5587366		
Total	330	858008,417			

	Coefficients	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*Lower 95.0%*	*Upper 95.0%*
Intercept	-1,23933967	0,7382779	-1,6786899	0,09417663	-2,6917632	0,21308388	-2,6917632	0,21308388
X Variable 1	0,203076481	0,0348984	5,81907635	1,4218E-08	0,1344204	0,27173256	0,1344204	0,27173256
X Variable 2	-0,092177376	0,03554588	-2,5931944	0,00994023	-0,1621072	-0,0222475	-0,1621072	-0,0222475
X Variable 3	0,027025305	0,03070846	0,88006045	0,37947885	-0,0333878	0,08743846	-0,0333878	0,08743846
X Variable 4	0,26950697	0,01559847	17,2777876	7,2397E-48	0,23881991	0,30019403	0,23881991	0,30019403
X Variable 5	0,835309642	0,01139817	73,2845388	9,592E-204	0,81288588	0,85773341	0,81288588	0,85773341
X Variable 6	-0,006345849	0,01032987	-0,6143201	0,53943498	-0,0266679	0,01397624	-0,0266679	0,01397624

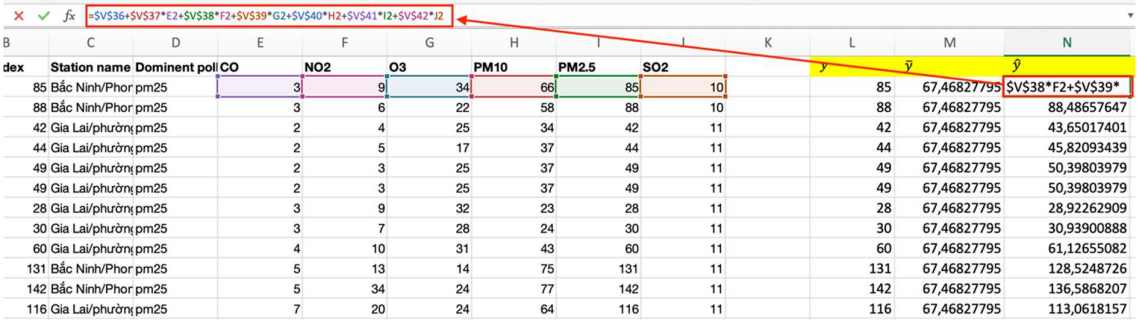
Equation:

Y(t) =-1,23+0,230*CO-0,09*NO3+0,027*O3+0,269*PM10+0,835*PM2.5-0,006*SO2

Figure 123. Result of Regression tool

b. The equation is: $Y(t)= 507.2 + 0.98779*r_{(t-1)} -0.0015*r_{(t-2)} + 0.00755*r_{(t-3)}$
c. The $R^2 = 0,994826381$.

Step 4: We calculate the $\hat{y}$ (predict value) based on the equation above:



dex	Station name	Dominent poll	CO	NO2	O3	PM10	PM2.5	SO2		y	ȳ	ŷ
85	Bắc Ninh/Phor	pm25	3	9	34	66	85	10		85	67,46827795	$V$38*F2+$V$39*
88	Bắc Ninh/Phor	pm25	3	6	22	58	88	10		88	67,46827795	88,48657647
42	Gia Lai/phường	pm25	2	4	25	34	42	11		42	67,46827795	43,65017401
44	Gia Lai/phường	pm25	2	5	17	37	44	11		44	67,46827795	45,82093439
49	Gia Lai/phường	pm25	2	3	25	37	49	11		49	67,46827795	50,39803979
49	Gia Lai/phường	pm25	2	3	25	37	49	11		49	67,46827795	50,39803979
28	Gia Lai/phường	pm25	3	9	32	23	28	11		28	67,46827795	28,92262909
30	Gia Lai/phường	pm25	3	7	28	24	30	11		30	67,46827795	30,93900888
60	Gia Lai/phường	pm25	4	10	31	43	60	11		60	67,46827795	61,12655082
131	Bắc Ninh/Phor	pm25	5	13	14	75	131	11		131	67,46827795	128,5248726
142	Bắc Ninh/Phor	pm25	5	34	24	77	142	11		142	67,46827795	136,5868207
116	Gia Lai/phường	pm25	7	20	24	64	116	11		116	67,46827795	113,0618157

Figure 124. Calculating predict value based on equation

Step 5: Calculating SSE:
We have SEE = $\sum(y_i - \hat{y}_i)^2$.
Starting with $(y_i - \hat{y}_i)$:

=L2-N2

L	M	N	O	P
y	$\bar{y}$	$\hat{y}$	$y-\bar{y}$	$y-\hat{y}$
85	67,46827795	88,18447484	17,53172205	=L2-N2
88	67,46827795	88,48657647	20,53172205	-0,486576468
42	67,46827795	43,65017401	-25,46827795	-1,650174011
44	67,46827795	45,82093439	-23,46827795	-1,820934388
49	67,46827795	50,39803979	-18,46827795	-1,398039789
49	67,46827795	50,39803979	-18,46827795	-1,398039789
28	67,46827795	28,92262909	-39,46827795	-0,922629093

Figure 125. Calculating $(y_i - \hat{y}_i)$

Continue with $(y_i - \hat{y}_i)^2$

=P2^2

P	Q
$y-\hat{y}$	$(y-\hat{y})^2$
-3,184474837	=P2^2
-0,486576468	0,236756659
-1,650174011	2,723074267
-1,820934388	3,315802045
-1,398039789	1,954515251

Figure 126. Calculating the square of Error value

Now we can find SSE:

=SUM(Q2:Q332)

Q	R	S	T	U	V
$(y-\hat{y})^2$	$(y-\bar{y})^2$	$(\hat{y}-\bar{y})^2$		Equation:	
10,14087999	307,3612782	429,1608136		Y(t) =-1,23+0,230*CO-0,09*NO3	
0,236756659	421,5516105	441,7688728		sumError	
2,723074267	648,6331815	567,302075		sumSSE	=SUM(Q2:Q332)
3,315802045	550,7600697	468,6074831		sumSST	858008,4169
1,954515251	341,0772903	291,3930307		sumSSR	849727,3863
1,954515251	341,0772903	291,3930307			
0,851244442	1557,744964	1485,767046			
0,881737673	1403,871852	1334,387499			

Figure 127. Calculating SSE by sum function

Step 6: Calculating SST:
First, we need the $(y - \bar{y})^2$

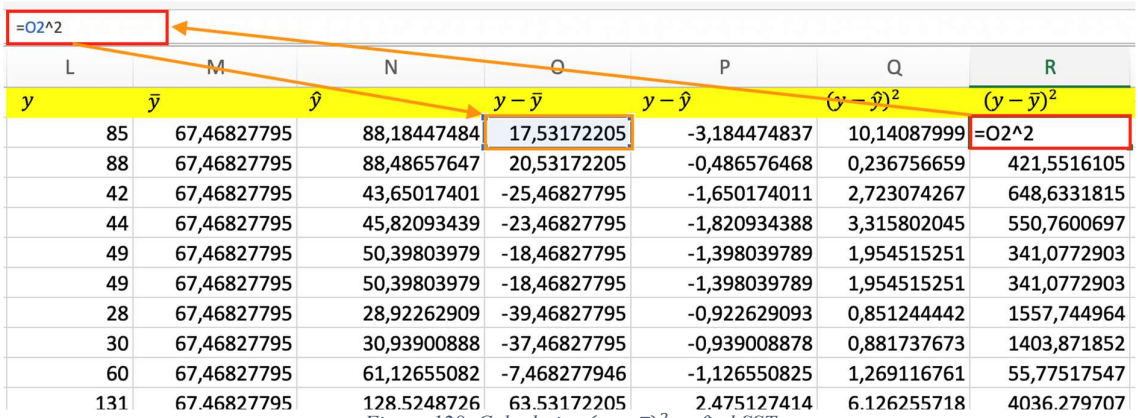


Figure 128. Calculating $(y - \bar{y})^2$ to find SST

After that, we can find SST value by using sum function of the $(y - \bar{y})^2$ with SST= $\sum(y - \bar{y})^2$:

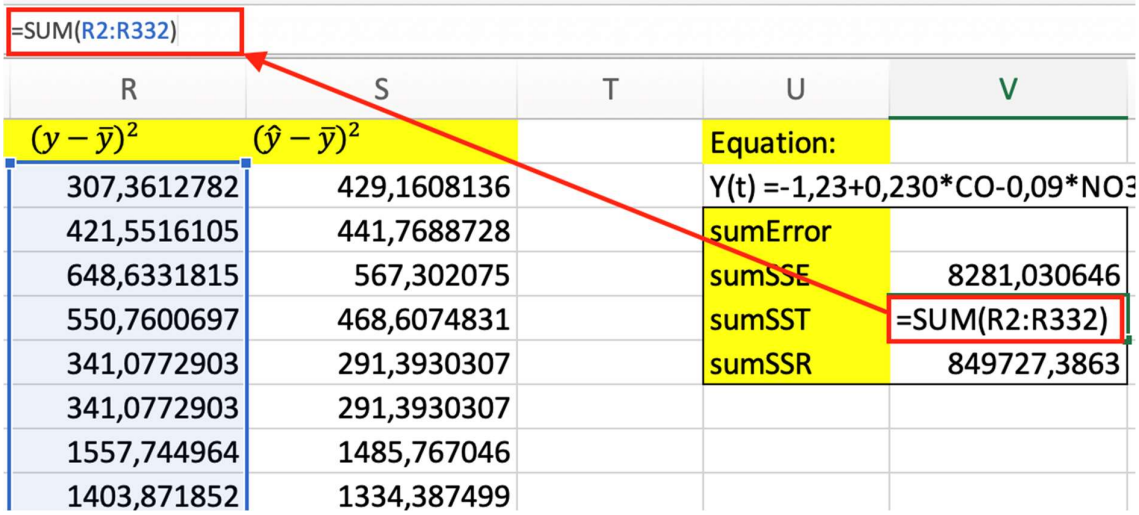


Figure 129. Finding SST

Step 7: Calculating SSR:

We need calculate $(\hat{y}_i - \bar{y})^2$ before finding SSR since SSR = $\sum(\hat{y}_i - \bar{y}_i)^2$

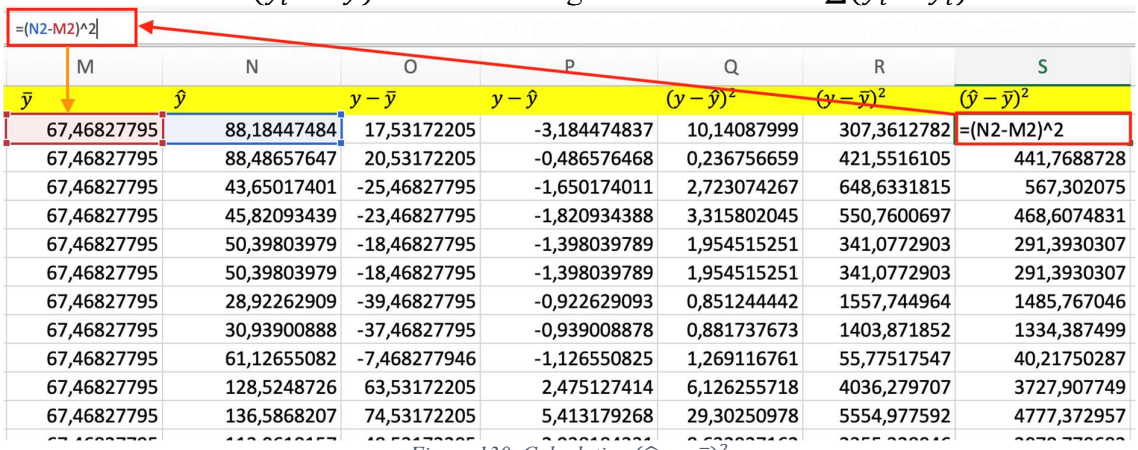


Figure 130. Calculating $(\hat{y}_i - \bar{y})^2$

After having $(\hat{y}_i - \bar{y})^2$ value, we find SSR by sum fuction:

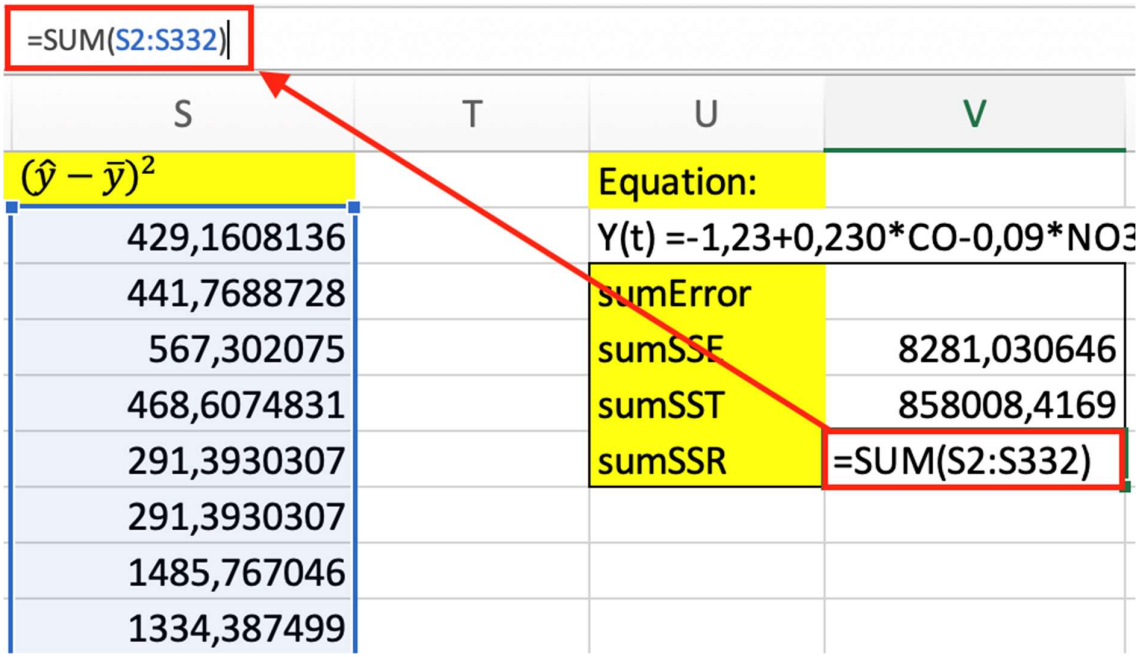


Figure 131. Calculating SSR

Step 8: Finding R^2

We have two formula:

b) $R^2 = 1 - \frac{SSE}{SST} = \frac{\sum(y_i-\hat{y}_i)^2}{\sum(y_i-\bar{y}_i)^2}$

c) $R^2 = \frac{SSR}{SST} = \frac{\sum(\hat{y}_i-\bar{y}_i)^2}{\sum(y_i-\bar{y}_i)^2}$

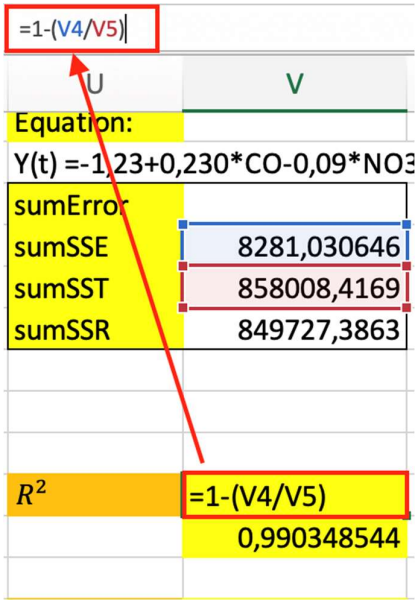


Figure 132. The first formula using SSE and SST

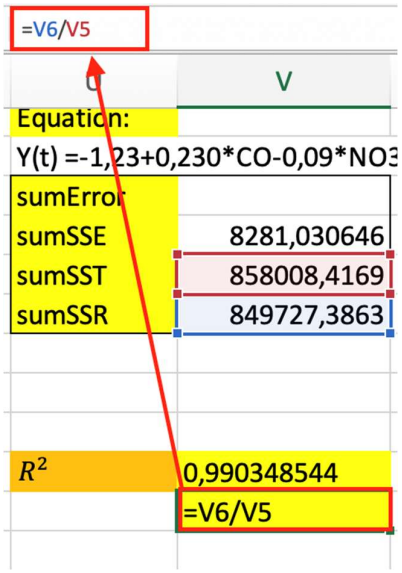


Figure 133. The second formula using SSR and SST

After checking with the R^2 value in ANOVA result, we can have conclusion that both of these two formula are accurate.

a. The value $R^2 = 0{,}990348544$ is very high, close to 1. This shows that the equation has the ability to explain the variation of the dependent variable very well. Specifically, the equation explains 99.03% of the variation in the dependent variable.

However, we also need to note that the R^2 value can be affected by the number of independent variables in the model. As the number of independent variables increases, the R^2 value may increase, even if those independent variables are not statistically significant. Therefore, we need to consider other indicators, such as the p-value of the regression coefficients, to evaluate the model fit.

b. Lab 4

Step 1: We set "close" to the y value that represents the dependent value in regression.

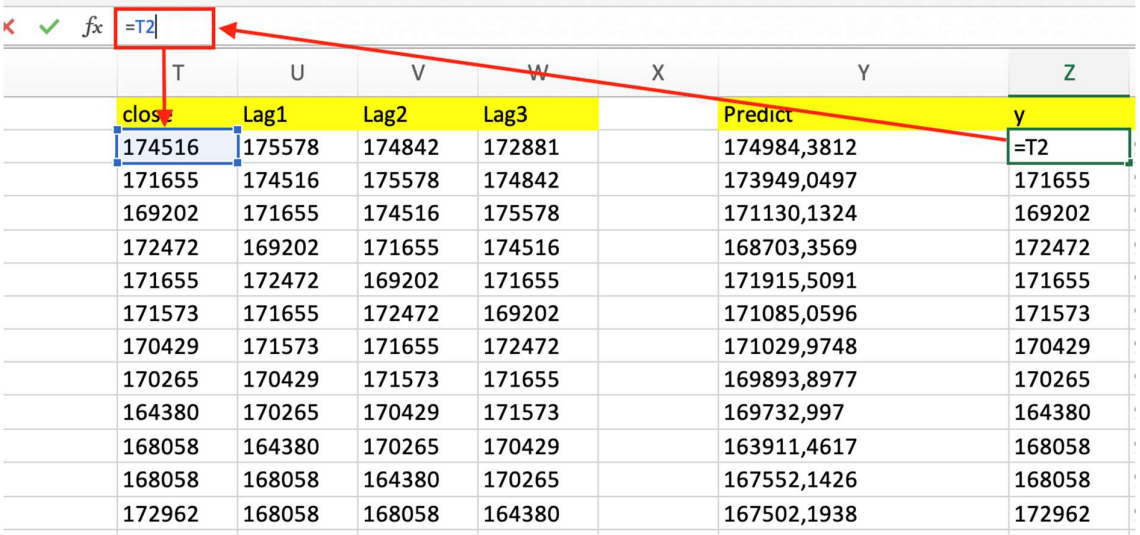


Figure 134. Setting y value from close column

Step 2: Calculating the mean value $\bar{y}$.

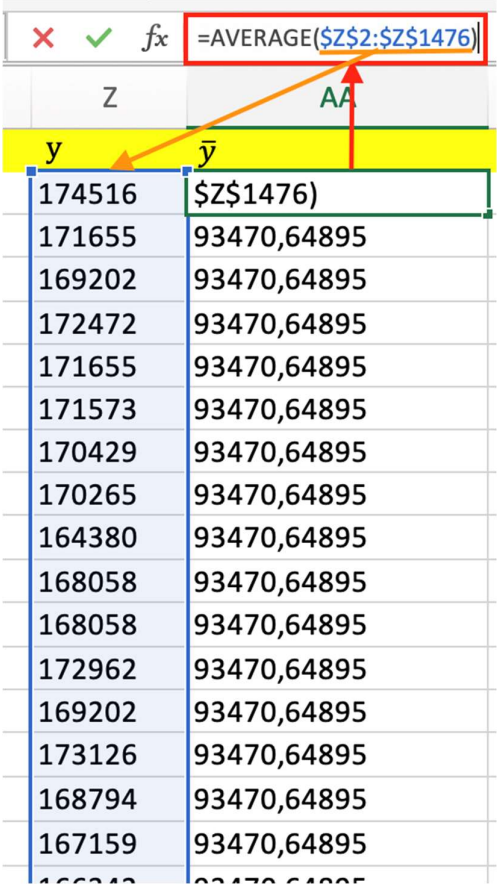


Figure 135. Calculating the mean of value y

Step 3: Calculating ANOVA to find equation through Regression tool:

SUMMARY OUTPUT

Regression Statistics	
Multiple R	0,99741
R Square	0,994826
Adjusted R	0,994816
Standard Er	1718,214
Observatior	1475

ANOVA

	df	*SS*	*MS*	*F*	*ignificance F*
Regression	3	8,35E+11	2,78355E+11	94285,36	0
Residual	1471	4,34E+09	2952259,587		
Total	1474	8,39E+11			

	Coefficients	*andard Errc*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*.ower 95.0%*	*Jpper 95.0%*
Intercept	507,1998	180,4302	2,811057742	0,005003	153,2718	861,1277	153,2718	861,1277
Lag 1	0,987788	0,02607	37,8903196	8,4E-220	0,93665	1,038926	0,93665	1,038926
Lag 2	-0,001545	0,036651	-0,042150597	0,966384	-0,073439	0,070349	-0,073439	0,070349
Lag3	0,007548	0,025968	0,290641491	0,771366	-0,043392	0,058487	-0,043392	0,058487

Figure 136. ANOVA result

c. The equation is: $Y(t)= 507.2 + 0.98779*r_{(t-1)} -0.0015*r_{(t-2)} + 0.00755*r_{(t-3)}$

d. The $R^2 = 0{,}994826381$.

Step 4: We calculate the $\hat{y}$ (predict value) based on the equation above:

=507,2 +0,98779*U2 -0,0015*V2+0,00755*W2

Y	Z	AA	AB
Predict	y	$\bar{y}$	$\hat{y}$
174984,3812	174516	93470,64895	174984,3812
173949,0497	171655	93470,64895	173949,0497
171130,1324	169202	93470,64895	171130,1324
168703,3569	172472	93470,64895	168703,3569
171915,5091	171655	93470,64895	171915,5091
171085,0596	171573	93470,64895	171085,0596
171029,9748	170429	93470,64895	171029,9748
169893,8977	170265	93470,64895	169893,8977
169732,997	164380	93470,64895	169732,997
163911,4617	168058	93470,64895	163911,4617
167552,1426	168058	93470,64895	167552,1426

Figure 137: Calculate y ^from equation (predict) after finding ANOVA step.

Step 5: Calculating SSE:
We have SEE = $\sum(y_i - \hat{y}_i)^2$.
Starting with $(y_i - \hat{y}_i)$:

=Z2-AB2

Z	AA	AB	AC	AD
y	$\bar{y}$	$\hat{y}$	$y - \bar{y}$	Error = $y - \hat{y}$
174516	93470,64895	174984,3812	81045,35105	=Z2-AB2
171655	93470,64895	173949,0497	78184,35105	-2294,04974
169202	93470,64895	171130,1324	75731,35105	-1928,13235
172472	93470,64895	168703,3569	79001,35105	3768,64312
171655	93470,64895	171915,5091	78184,35105	-260,50913
171573	93470,64895	171085,0596	78102,35105	487,94045
170429	93470,64895	171029,9748	76958,35105	-600,97477
170265	93470,64895	169893,8977	76794,35105	371,10234
164380	93470,64895	169732,997	70909,35105	-5352,997
168058	93470,64895	163911,4617	74587,35105	4146,53835

Figure 138. Calculating Error value due to y and ŷ

Continue with $(y_i - \hat{y}_i)^2$

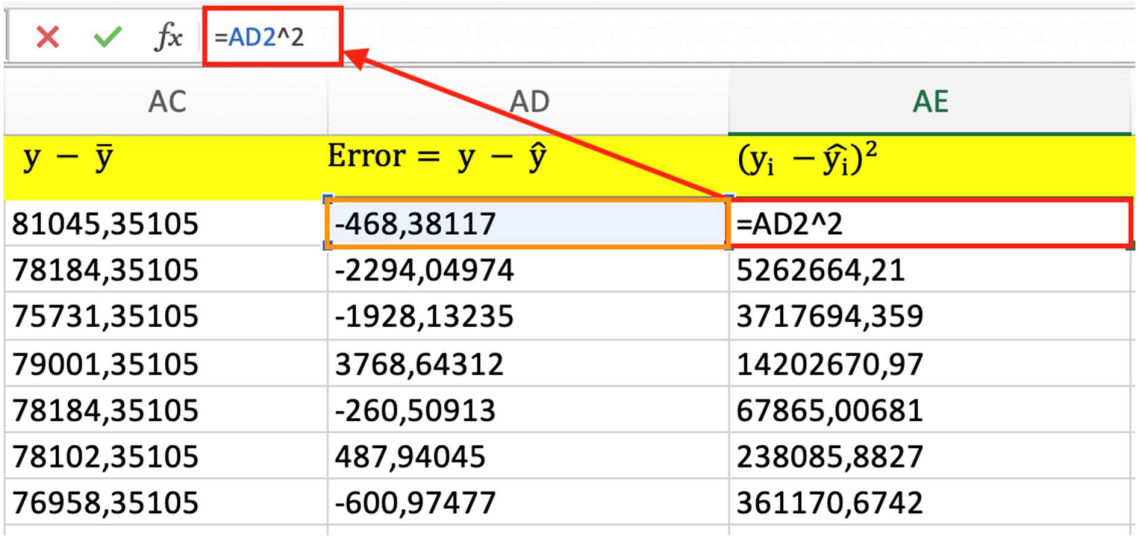


Figure 139. Calculating the square of Error value

Now we can find SSE:

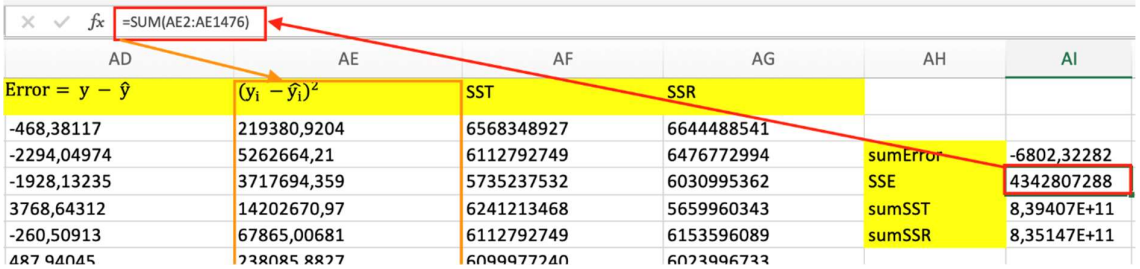


Figure 140. Calculating SSE by sum function

Step 6: Calculating SST:

First, we need the $(y - \bar{y})^2$

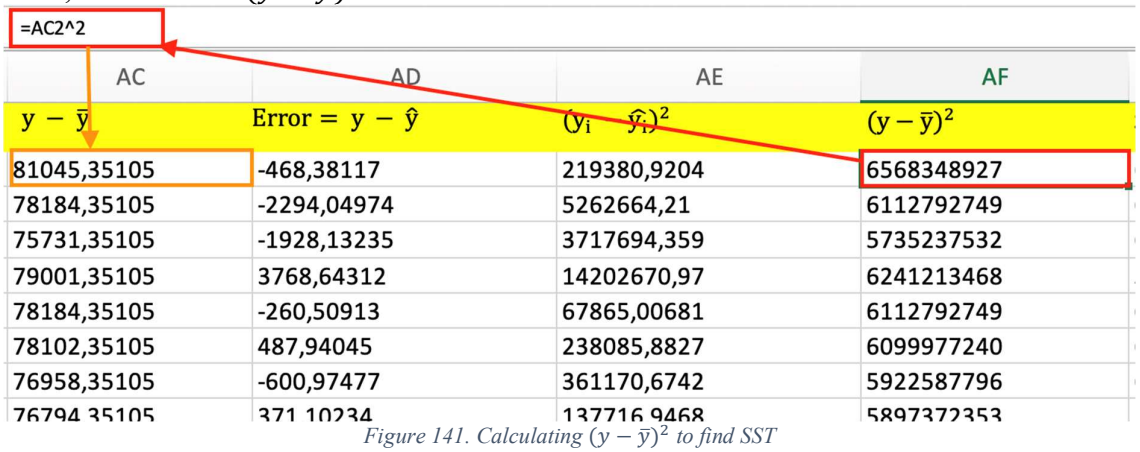


Figure 141. Calculating $(y - \bar{y})^2$ to find SST

After that, we can find SST value by using sum function of the $(y - \bar{y})^2$ with SST= $\sum(y - \bar{y})^2$:

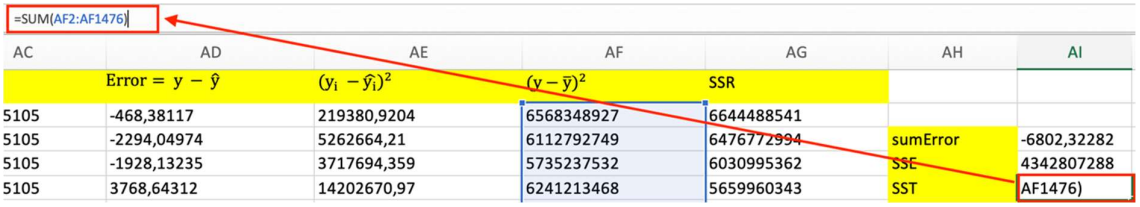


Figure 142. Finding SST

Step 7: Calculating SSR:

We need calculate $(\hat{y}_i - \bar{y})^2$ before finding SSR since SSR = $\sum(\hat{y}_i - \bar{y}_i)^2$

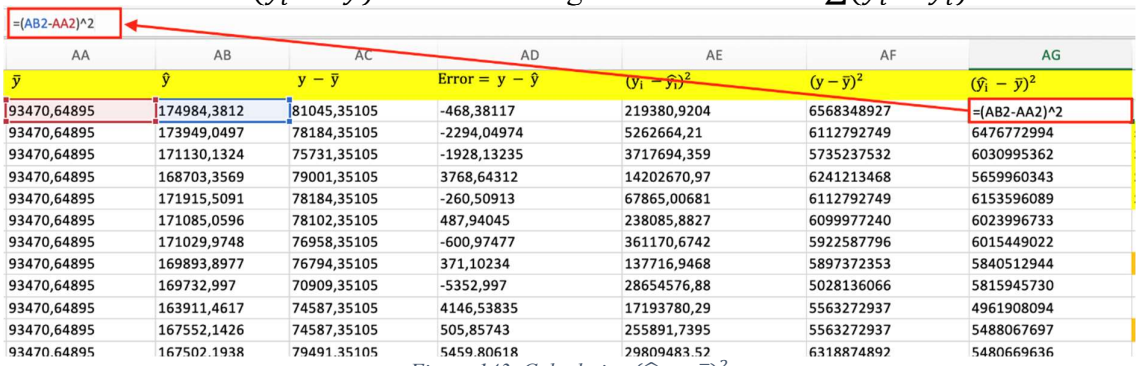


Figure 143. Calculating $(\hat{y}_i - \bar{y})^2$

After having $(\hat{y}_i - \bar{y})^2$ value, we find SSR by sum fuction:

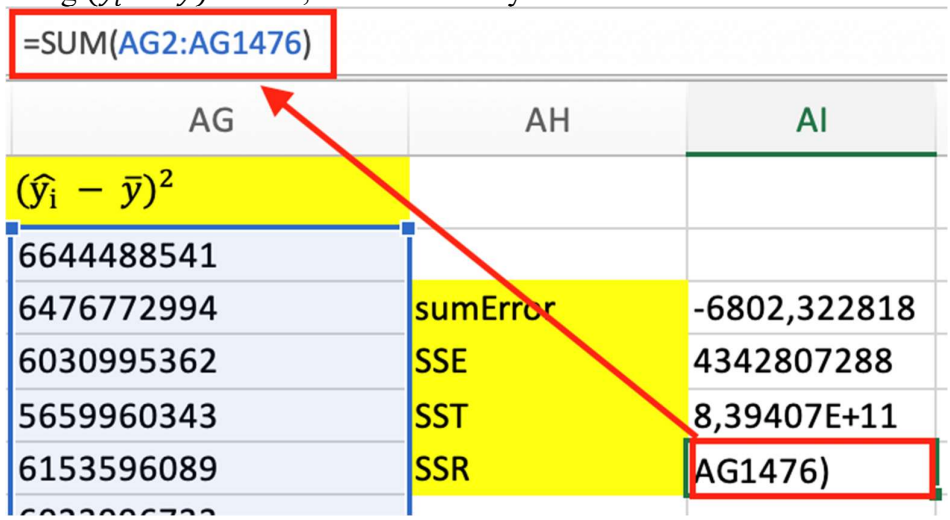


Figure 144. Calculating SSR

Step 8: Finding R^2

We have two formula:

d) $R^2 = 1 - \frac{SSE}{SST} = \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y}_i)^2}$

e) $R^2 = \frac{SSR}{SST} = \frac{\sum(\hat{y}_i - \bar{y}_i)^2}{\sum(y_i - \bar{y}_i)^2}$

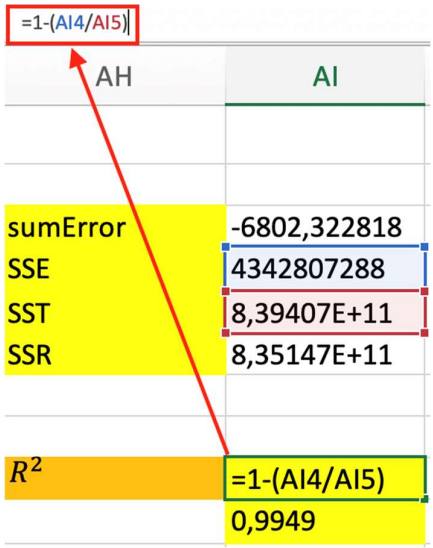


Figure 145. The first formula using SSE and SST

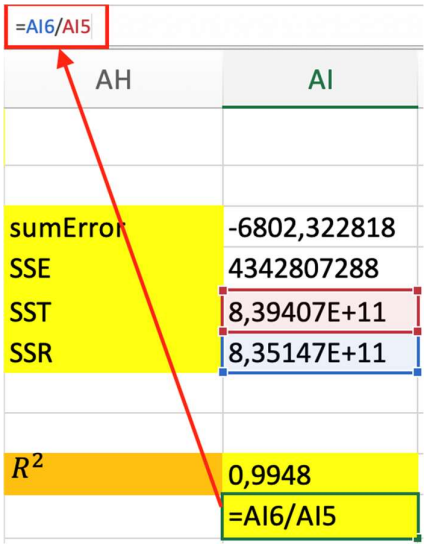


Figure 146. The second formula using SSR and SST

After checking with the R^2 value in ANOVA result, we can have conclusion that both of these two formula are accurate.

a. The value $R^2 = 0.994826381$ is very high, close to 1. This shows that the equation has the ability to explain the variation of the dependent variable very well. Specifically, the equation explains 99.48% of the variation in the dependent variable.

However, we also need to note that the R^2 value can be affected by the number of independent variables in the model. As the number of independent variables increases, the R^2 value may increase, even if those independent variables are not statistically significant. Therefore, we need to consider other indicators, such as the p-value of the regression coefficients, to evaluate the model fit.

2. Using R language:

a) Lab 3:

First, we need to build a model to study the relationship between log variables and AQI.index and then summary it to see the regression parameter.

With:

f) nls_data$log_CO <- log10(nls_data$CO): This command creates a new variable named log_CO in dataset nls_data. This variable contains the base 10 logarithm value of the original CO variable.

g) The next four commands do the same for the variables NO2, O3, PM10, PM2.5, and SO2, creating the corresponding log variables.

h) independent_vars <- c('log_CO', 'log_NO2', 'log_O3', 'log_PM10', 'log_PM2.5', 'log_SO2') identify independent variables.

i) reg = lm(AQI.index ~ log_CO + log_NO2 + log_O3 + log_PM10 + log_PM2.5 + log_SO2, data= nls_data): This command creates a linear regression model named reg. The model's dependent variable is AQI.index, and the independent variables are log variables listed in the independent_vars vector. The data used to build the model is nls_data.

```
> attach(nls_data)
> nls_data$log_CO <- log10(nls_data$CO)
> nls_data$log_NO2 <- log10(nls_data$NO2)
> nls_data$log_O3 <- log10(nls_data$O3)
> nls_data$log_PM10 <- log10(nls_data$PM10)
> nls_data$log_PM2.5 <- log10(nls_data$PM2.5)
> nls_data$log_SO2 <- log10(nls_data$SO2)
> independent_vars <- c('log_CO', 'log_NO2', 'log_O3', 'log_PM10', 'log_PM2.5', 'log_SO2')
> reg = lm(AQI.index ~ log_CO + log_NO2 + log_O3 + log_PM10 + log_PM2.5 + log_SO2, data= nls_da
ta)
> summary(reg)

Call:
lm(formula = AQI.index ~ log_CO + log_NO2 + log_O3 + log_PM10 +
    log_PM2.5 + log_SO2, data = nls_data)

Residuals:
    Min      1Q  Median      3Q     Max
-46.487 -15.289  -5.708   8.103 101.066

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -137.6049     8.7992 -15.638  < 2e-16 ***
log_CO        22.0985     4.3494   5.081 6.36e-07 ***
log_NO2      -11.8775     3.9755  -2.988  0.00303 **
log_O3        -6.2618     3.3028  -1.896  0.05886 .
log_PM10      47.6676     7.9274   6.013 4.91e-09 ***
log_PM2.5     77.5024     7.4693  10.376  < 2e-16 ***
log_SO2        0.9535     3.4264   0.278  0.78096
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 21.51 on 324 degrees of freedom
Multiple R-squared:  0.8253,    Adjusted R-squared:  0.8221
```

Figure 147. Build a model to study the relationship between log variables and AQI.index

To check the R^2 by small-scale way, we can use the formula as mentioned above.

```
> SST <- sum((AQI.index - mean(AQI.index))^2)
> SST
[1] 858008.4
```

Figure 148. Calculating and showing the SST

```
> SSE <- sum((AQI.index - fitted(reg))^2)
> SSE
[1] 149887.7
```

Figure 149. Calculating and showing the SSE

```
> SSR <- sum((fitted(reg) - mean(AQI.index))^2)
> SSR
[1] 708120.7
```

Figure 150. Calculating and showing the SSR

We still have two way to calculate R^2:

```
> R_squared <- 1 - (SSE / SST)
> R_squared
[1] 0.8253075
```

Figure 151. The first method to calculate the R squared

```
> R_squared_2 <- SSR/SST
> R_squared_2
[1] 0.8253075
```

Figure 152. The second method to calculate the R squared

After calculating, we check to see if the returned parameters are the same or not. If it is the same then we have followed the correct method.

➔ The equation of MNLR of this data set: AQI = -137,6049 + 22,09 * CO – 11,87*NO_2 – 6,26*O_3 + 47,66*PM10 + 77,5*PM2,5 + 0,95*SO_2 fitted the model 82,53% with R squared testing model method.

b) Lab 4:

We separate the rows close, lag1, lag2, lag3 calculated in Lab 4 into a new file. After reading and attaching the data file, we create a multiple linear regression model and take “close” as the dependent y value. Then, use summary command to see the parameter of this data set’s regression.

```
> attach(lab4_data)
> reg = lm(close~ Lag1+Lag2+Lag3,data=lab4_data)
> summary(reg)

Call:
lm(formula = close ~ Lag1 + Lag2 + Lag3, data = lab4_data)

Residuals:
     Min       1Q   Median       3Q      Max
-31478.8   -709.6      3.6    666.6   8416.4

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 507.199768 180.430221   2.811    0.005 **
Lag1          0.987788   0.026070  37.890   <2e-16 ***
Lag2         -0.001545   0.036651  -0.042    0.966
Lag3          0.007548   0.025968   0.291    0.771
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1718 on 1471 degrees of freedom
Multiple R-squared:  0.9948,    Adjusted R-squared:  0.9948
F-statistic: 9.429e+04 on 3 and 1471 DF,  p-value: < 2.2e-16
```

Figure 153. Regression result of the data set

a. The equation is: $Y(t)= 507.2 + 0.98779*r_{(t-1)} -0.0015*r_{(t-2)} + 0.00755*r_{(t-3)}$
b. $R^2 = 0,9948$

Let's start by calculating R square by hand to check again.

```
> SST <- sum((close - mean(close))^2)
> SST
[1] 839407357869
```

Figure 154. Calculating SST

```
> SSE <- sum((close - fitted(reg))^2)
> SSE
[1] 4342773852
```

Figure 155. Calculating SSE

```
> SSR <- sum((fitted(reg) - mean(close))^2)
> SSR
[1] 835064584017
```

Figure 156. Calculating SSR

We still have two way to calculate R^2:

```
> R_squared <- 1 - (SSE / SST)
> R_squared
[1] 0.9948264
```

Figure 157. Calculating R squared by method 1

```
> R_squared_2 <- SSR / SST
> R_squared_2
[1] 0.9948264
```

Figure 158. Calculating R squared by method 2

After calculating, we check to see if the returned parameters are the same or not. If it is the same then we have followed the correct method.

- The equation of this data set: $Y(t)= 507.2 + 0.98779*r_{(t-1)} -0.0015*r_{(t-2)} + 0.00755*r_{(t-3)}$ fitted the model 99,48%% with R squared testing model method.
- The value $R^2 = 0.994826381$ is very high, close to 1. This shows that the equation has the ability to explain the variation of the dependent variable very well.

Specifically, the equation explains 99.48% of the variation in the dependent variable. However, we also need to note that the R^2 value can be affected by the number of independent variables in the model. As the number of independent variables increases, the R^2 value may increase, even if those independent variables are not statistically significant. Therefore, we need to consider other indicators, such as the p-value of the regression coefficients, to evaluate the model fit.

3. Using Python languages:

a) Lab 3:

Step 1: Import and read the data

```
In [4]: import pandas as pd
        import numpy as np
        from scipy import stats
        import statsmodels.api as sm
        import matplotlib.pyplot as plt
        from sklearn.linear_model import LinearRegression
        from sklearn.metrics import r2_score

In [9]: df = pd.read_csv('historical_air_quality_2021_clean.csv',delimiter=";")
        df
```

Out[9]:

	Station ID	AQI index	Station name	Dominent pollutant	CO	NO2	O3	PM10	PM2.5	SO2
0	12958	85.0	Bắc Ninh/Phong Cốc, Vietnam	pm25	3.0	9.0	34.0	66.0	85.0	10.0
1	12958	88.0	Bắc Ninh/Phong Cốc, Vietnam	pm25	3.0	6.0	22.0	58.0	88.0	10.0
2	13012	42.0	Gia Lai/phường Thống Nhất - Pleiku, Vietnam	pm25	2.0	4.0	25.0	34.0	42.0	11.0
3	13012	44.0	Gia Lai/phường Thống Nhất - Pleiku, Vietnam	pm25	2.0	5.0	17.0	37.0	44.0	11.0
4	13012	49.0	Gia Lai/phường Thống Nhất - Pleiku, Vietnam	pm25	2.0	3.0	25.0	37.0	49.0	11.0
...	...	...	...	...	...	...	...	...	...	...
326	12976	67.0	Bắc Ninh/Châu Khê, Vietnam	pm25	5.0	12.0	8.0	53.0	67.0	9.0
327	12976	67.0	Bắc Ninh/Châu Khê, Vietnam	pm25	5.0	12.0	8.0	53.0	67.0	9.0
328	12976	67.0	Bắc Ninh/Châu Khê, Vietnam	pm25	5.0	12.0	8.0	53.0	67.0	9.0
329	12976	83.0	Bắc Ninh/Châu Khê, Vietnam	pm25	6.0	7.0	8.0	54.0	83.0	9.0
330	13250	47.0	Lào Cai/KCN TAI LOONG1, Vietnam	pm25	34.0	3.0	3.0	7.0	47.0	99.0

Figure 159. Import libraries and dataset

Step 2: Training the model.

- Transform data by taking logarithms of relevant variables.
- Extract independent and dependent variables from DataFrame and convert them to the appropriate format.
- Use the LinearRegression model to understand the relationship between independent and dependent variables.
- Provides access to model parameters such as intercepts and coefficients.

```
In [42]: # Calculating log of dependent variables:
         df['log(CO)'] = np.log(df['CO'])
         df['log(NO2)'] = np.log(df['NO2'])
         df['log(O3)'] = np.log(df['O3'])
         df['log(PM10)'] = np.log(df['PM10'])
         df['log(PM2.5)'] = np.log(df['PM2.5'])
         df['log(SO2)'] = np.log(df['SO2'])
         # Get the dependent & independent variables:
         x = np.array(df[['log(CO)', 'log(NO2)', 'log(O3)', 'log(PM10)', 'log(PM2.5)', 'log(SO2)']]).reshape((-1, 6))
         y = np.array(df['AQI index']).reshape((-1, 1))
         # Using Linear Regression() to gain the model follow X and Y variables and taking result:
         reg = LinearRegression()
         reg.fit(x,y)

Out[42]: ▾ LinearRegression
         LinearRegression()
```

Figure 160. Training the model before finding R squared

Step 3: Finding R squared

```
In [43]: # R-squared
         r_squared = reg.score(x, y)

         print("R-squared:", r_squared)

         R-squared: 0.8253074538923281
```

Figure 161. Finding and showing the R squared

reg.score(x, y): The score function of the reg model is used to evaluate the fit of the model to the data. It calculates R-squared, a statistical index between 0 and 1, indicating the proportion of variation in the dependent variable that is explained by the independent variables.

➔ The equation of MNLR of this data set: AQI = -137,6049 + 22,09 * CO – 11,87*NO_2 – 6,26*O_3 + 47,66*PM10 + 77,5*PM2,5 + 0,95*SO_2 fitted the model 82,53% with R squared testing model method.

b) Lab 4:

Step 1:

```
In [1]: import pandas as pd
        import numpy as np
        from scipy import stats
        import statsmodels.api as sm
        import matplotlib.pyplot as plt
        from sklearn.linear_model import LinearRegression
        from sklearn.metrics import r2_score

In [3]: df=pd.read_csv('lab5.csv')
        df
```

Out[3]:

	close	Lag1	Lag2	Lag3
0	174516.0	175578.0	174842.0	172881.0
1	171655.0	174516.0	175578.0	174842.0
2	169202.0	171655.0	174516.0	175578.0
3	172472.0	169202.0	171655.0	174516.0
4	171655.0	172472.0	169202.0	171655.0
...	...	...	...	...
1470	68100.0	68300.0	68700.0	69400.0
1471	68500.0	68100.0	68300.0	68700.0
1472	68400.0	68500.0	68100.0	68300.0
1473	67400.0	68400.0	68500.0	68100.0
1474	68400.0	67400.0	68400.0	68500.0

Figure 162. Import necessary libraries and read dataset

Step 2: Training data

- Extract independent and dependent variables from DataFrame and convert them to the appropriate format.
- Use the LinearRegression model to understand the relationship between independent and dependent variables.
- Provides access to model parameters such as intercepts and coefficients.

```
In [4]: # Get the dependent & independent variables:
        x = np.array(df[['Lag1','Lag2','Lag3']]).reshape((-1, 3))
        y = np.array(df['close']).reshape((-1, 1))
        # Using Linear Regression() to gain the model follow X and Y variables and taking result:
        reg = LinearRegression()
        reg.fit(x,y)

Out[4]: ▾ LinearRegression
        LinearRegression()
```

Figure 163. Training data for using to calculate R squared

Step 3:

```
In [5]: # R-squared
        r_squared = reg.score(x, y)

        print("R-squared:", r_squared)

        R-squared: 0.9948263810036525
```

Figure 164. Finding and showing the R squared

reg.score(x, y): The score function of the reg model is used to evaluate the fit of the model to the data. It calculates R-squared, a statistical index between 0 and 1, indicating the proportion of variation in the dependent variable that is explained by the independent variables.

The equation of this data set: $Y(t)= 507.2 + 0.98779*r_{(t-1)} -0.0015*r_{(t-2)} + 0.00755*r_{(t-3)}$

fitted the model 99,48% with R squared testing model method.

VI. Adjusted R-squared:

1. Using Excel:

a) Lab 3:

After calculating R^2, we can move to the stage finding Adjusted R^2.

Step 1: Based on the formula, we can find n by count function. With the range (L2:L332) is the range of y value (AQI Index). And k is the number of independent variables we have above (k=6).

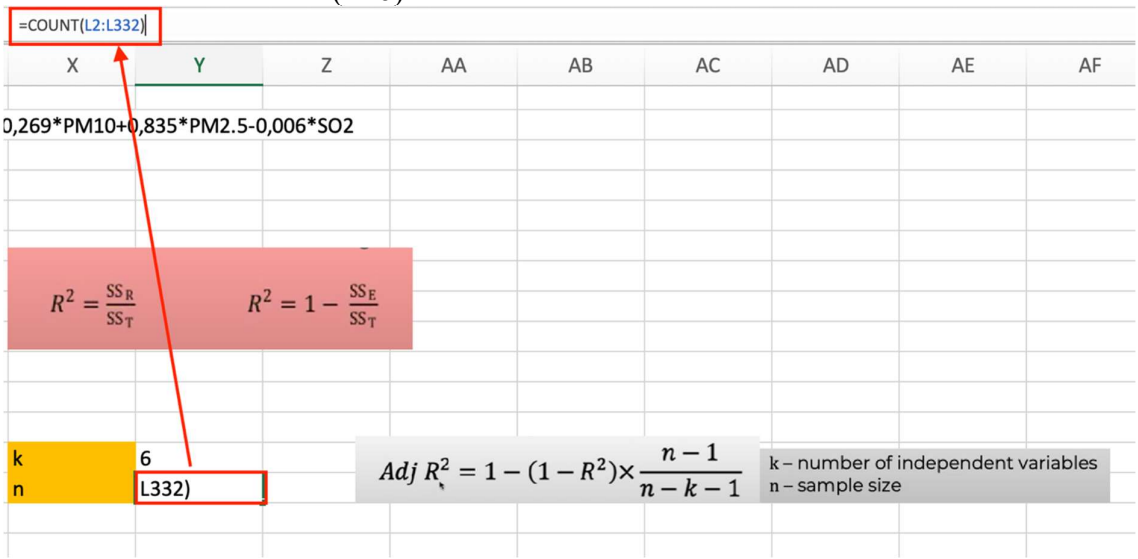


Figure 165. Calculate the number of y values

Step 2: Finding Adjusted R^2

We can find Adjusted R^2 by using the command below based on the formula:

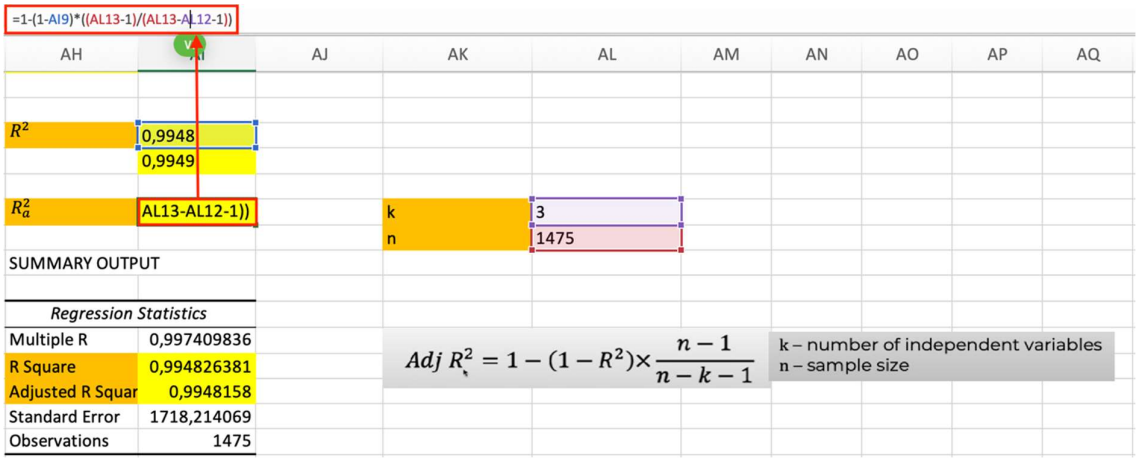


Figure 166. Calculating adjusted R squared

After that, we need to check if the result is accurated or not.

The Adjusted R^2= 0.9948158 is very high, close to 1. This shows that the method is able to explain the variation of the dependent variable very well. Specifically,

the solution method achieves 99.48% of the variation in the dependent variable, after adjusting the number of independent variables in the model.

With such a high R_a^2 value, we can be confident that this method is a suitable model to expect the dependent variables. The method can be used to predict the value of the dependent variable with high accuracy.

b) Lab 4:

After calculating R^2, we can move to the stage finding Adjusted R^2.

Step 1: Based on the formula, we can find n by count function. With the range (Z2:Z1476) is the range of y value (close). And k is the number of independent variables we have above (k=3).

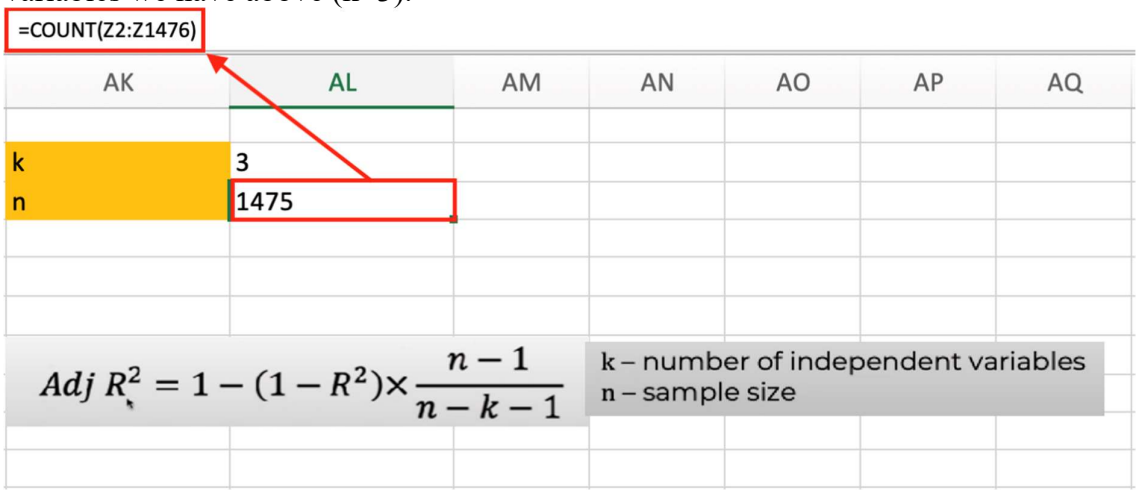


Figure 167. Calculate the number of y values

Step 2: Finding Adjusted R^2

We can find Adjusted R^2 by using the command below based on the formula:

$$R_a^2 = 1 - (1 - R^2)\frac{n-1}{n-p-1}$$

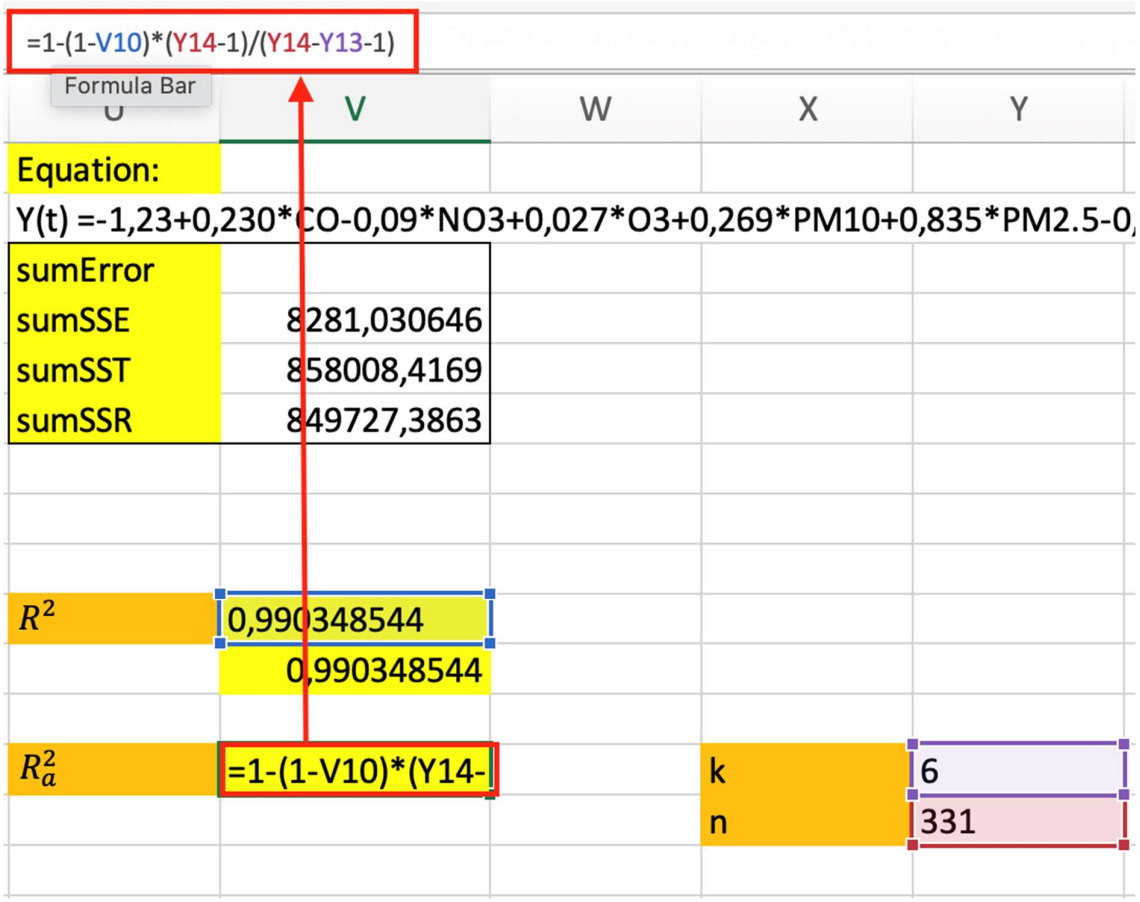


Figure 168. Calculating adjusted R squared

After that, we need to check if the result is accurated or not.

The Adjusted R^2= 0,990169813 is very high, close to 1. This shows that the method is able to explain the variation of the dependent variable very well. Specifically, the solution method achieves 99.01% of the variation in the dependent variable, after adjusting the number of independent variables in the model.

With such a high R_a^2 value, we can be confident that this method is a suitable model to expect the dependent variables. The method can be used to predict the value of the dependent variable with high accuracy.

2. Using R language:

a) Lab 3:

First, we need to build a model to study the relationship between log variables and AQI.index and then summary it to see the regression parameter.

With:

- nls_data$log_CO <- log10(nls_data$CO): This command creates a new variable named log_CO in dataset nls_data. This variable contains the base 10 logarithm value of the original CO variable.

- The next four commands do the same for the variables NO2, O3, PM10, PM2.5, and SO2, creating the corresponding log variables.

- independent_vars <- c('log_CO', 'log_NO2', 'log_O3', 'log_PM10', 'log_PM2.5', 'log_SO2') identify independent variables.

- reg = lm(AQI.index ~ log_CO + log_NO2 + log_O3 + log_PM10 + log_PM2.5 + log_SO2, data= nls_data): This command creates a linear regression model named reg. The model's dependent variable is AQI.index, and the independent variables are log variables listed in the independent_vars vector. The data used to build the model is nls_data.

```
> attach(nls_data)
> nls_data$log_CO <- log10(nls_data$CO)
> nls_data$log_NO2 <- log10(nls_data$NO2)
> nls_data$log_O3 <- log10(nls_data$O3)
> nls_data$log_PM10 <- log10(nls_data$PM10)
> nls_data$log_PM2.5 <- log10(nls_data$PM2.5)
> nls_data$log_SO2 <- log10(nls_data$SO2)
> independent_vars <- c('log_CO', 'log_NO2', 'log_O3', 'log_PM10', 'log_PM2.5', 'log_SO2')
> reg = lm(AQI.index ~ log_CO + log_NO2 + log_O3 + log_PM10 + log_PM2.5 + log_SO2, data= nls_da
ta)
> summary(reg)

Call:
lm(formula = AQI.index ~ log_CO + log_NO2 + log_O3 + log_PM10 +
    log_PM2.5 + log_SO2, data = nls_data)

Residuals:
    Min      1Q  Median      3Q     Max
-46.487 -15.289  -5.708   8.103 101.066

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -137.6049     8.7992 -15.638  < 2e-16 ***
log_CO        22.0985     4.3494   5.081 6.36e-07 ***
log_NO2      -11.8775     3.9755  -2.988  0.00303 **
log_O3        -6.2618     3.3028  -1.896  0.05886 .
log_PM10      47.6676     7.9274   6.013 4.91e-09 ***
log_PM2.5     77.5024     7.4693  10.376  < 2e-16 ***
log_SO2        0.9535     3.4264   0.278  0.78096
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 21.51 on 324 degrees of freedom
Multiple R-squared:  0.8253,    Adjusted R-squared:  0.8221
```

Figure 169. Build a model to study the relationship between log variables and AQI.index

To check the adjusted R^2 by small-scale way, we can use the formula as mentioned above.

```
> n <- length(AQI.index)
> p <- length(reg$coefficients)
```

Figure 170. Finding n and k

Where:

- n is the length (number of elements of the dependent variable AQI.index),
- k or p returns the number of independent variables in the model.

After that, we can use the result of R squared that we have found to calculate the adjusted R square as below:

```
> adjusted_R_squared <- 1 - (1 - R_squared) * ((n-1)/(n-p-1))
> adjusted_R_squared
[1] 0.8215215
```

Figure 171. Calculating Adjusted R squared

After calculating, we check to see if the returned parameters are the same or not. If it is the same then we have followed the correct method.

➔ The equation of MNLR of this data set: AQI = -137,6049 + 22,09 * CO – 11,87*NO_2 – 6,26*O_3 + 47,66*PM10 + 77,5*PM2,5 + 0,95*SO_2 fitted the model 82,15% with adjusted R squared testing model method.

b) Lab 4:

First, we need to build a model to study the relationship between log variables and "close" and then summary it to see the regression parameter.

```
> attach(lab4_data)
> reg = lm(close~ Lag1+Lag2+Lag3,data=lab4_data)
> summary(reg)

Call:
lm(formula = close ~ Lag1 + Lag2 + Lag3, data = lab4_data)

Residuals:
     Min       1Q   Median       3Q      Max
-31478.8   -709.6      3.6    666.6   8416.4

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 507.199768 180.430221   2.811    0.005 **
Lag1          0.987788   0.026070  37.890   <2e-16 ***
Lag2         -0.001545   0.036651  -0.042    0.966
Lag3          0.007548   0.025968   0.291    0.771
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1718 on 1471 degrees of freedom
Multiple R-squared:  0.9948,    Adjusted R-squared:  0.9948
F-statistic: 9.429e+04 on 3 and 1471 DF,  p-value: < 2.2e-16
```

Figure 172. Build a model to study the relationship between log variables and close

To check the adjusted R^2 by small-scale way, we can use the formula as mentioned above.

```
> n <- length(close)
> p <- length(reg1$coefficients)
```

Figure 173. Finding n and p

Where:

- n is the length (number of elements of the dependent variable close),
- k or p returns the number of independent variables in the model.

After that, we can use the result of R squared that we have found to calculate the adjusted R square as below:

```
> adjusted_R_squared <- 1 - (1 - R_squared) * ((n-1)/(n-p-1))
> adjusted_R_squared
[1] 0.9948123
```

Figure 174. Calculating Adjusted R squared

After calculating, we check to see if the returned parameters are the same or not. If it is the same then we have followed the correct method.

The equation of Lab 4 data set: $Y(t)= 507.2 + 0.98779*r_{(t-1)} -0.0015*r_{(t-2)} + 0.00755*r_{(t-3)}$ fitted the model 99,48% with adjusted R squared testing model method.

3. Using Python languages:

a) Lab 3:

Step 1: Import and read the data

```
In [4]: import pandas as pd
        import numpy as np
        from scipy import stats
        import statsmodels.api as sm
        import matplotlib.pyplot as plt
        from sklearn.linear_model import LinearRegression
        from sklearn.metrics import r2_score

In [9]: df = pd.read_csv('historical_air_quality_2021_clean.csv',delimiter=";")
        df
```

Out[9]:

	Station ID	AQI index	Station name	Dominent pollutant	CO	NO2	O3	PM10	PM2.5	SO2
0	12958	85.0	Bắc Ninh/Phong Cốc, Vietnam	pm25	3.0	9.0	34.0	66.0	85.0	10.0
1	12958	88.0	Bắc Ninh/Phong Cốc, Vietnam	pm25	3.0	6.0	22.0	58.0	88.0	10.0
2	13012	42.0	Gia Lai/phường Thống Nhất - Pleiku, Vietnam	pm25	2.0	4.0	25.0	34.0	42.0	11.0
3	13012	44.0	Gia Lai/phường Thống Nhất - Pleiku, Vietnam	pm25	2.0	5.0	17.0	37.0	44.0	11.0
4	13012	49.0	Gia Lai/phường Thống Nhất - Pleiku, Vietnam	pm25	2.0	3.0	25.0	37.0	49.0	11.0
...	...	...	...	...	...	...	...	...	...	...
326	12976	67.0	Bắc Ninh/Châu Khê, Vietnam	pm25	5.0	12.0	8.0	53.0	67.0	9.0
327	12976	67.0	Bắc Ninh/Châu Khê, Vietnam	pm25	5.0	12.0	8.0	53.0	67.0	9.0
328	12976	67.0	Bắc Ninh/Châu Khê, Vietnam	pm25	5.0	12.0	8.0	53.0	67.0	9.0
329	12976	83.0	Bắc Ninh/Châu Khê, Vietnam	pm25	6.0	7.0	8.0	54.0	83.0	9.0
330	13250	47.0	Lào Cai/KCN TAI OONG1 Vietnam	pm25	34.0	3.0	3.0	7.0	47.0	99.0

Figure 175. Import libraries and dataset

Step 2: Training the model.

- Transform data by taking logarithms of relevant variables.
- Extract independent and dependent variables from DataFrame and convert them to the appropriate format.

- Use the LinearRegression model to understand the relationship between independent and dependent variables.
- Provides access to model parameters such as intercepts and coefficients.

```
In [42]: # Calculating log of dependent variables:
         df['log(CO)'] = np.log(df['CO'])
         df['log(NO2)'] = np.log(df['NO2'])
         df['log(O3)'] = np.log(df['O3'])
         df['log(PM10)'] = np.log(df['PM10'])
         df['log(PM2.5)'] = np.log(df['PM2.5'])
         df['log(SO2)'] = np.log(df['SO2'])
         # Get the dependent & independent variables:
         x = np.array(df[['log(CO)', 'log(NO2)', 'log(O3)', 'log(PM10)', 'log(PM2.5)', 'log(SO2)']]).reshape((-1, 6))
         y = np.array(df['AQI index']).reshape((-1, 1))
         # Using Linear Regression() to gain the model follow X and Y variables and taking result:
         reg = LinearRegression()
         reg.fit(x,y)

Out[42]: LinearRegression
         LinearRegression()
```

Figure 176. Training the model before finding R squared

Step 3: Finding Adjusted R Squared

```
In [44]: # Calculate adjusted R-squared
         n_features = x.shape[1]
         n_samples = x.shape[0]
         adjusted_r_squared = 1 - (1 - r_squared) * (n_samples - 1) / (n_samples - n_features - 1)

         print("Adjusted R-squared:", adjusted_r_squared)

         Adjusted R-squared: 0.822072406742186
```

Figure 177. Finding and showing the adjusted R squared

n_features = x.shape[1]:
+ n_features is a variable that stores the number of independent variables (features) in the model.
+ x.shape[1] accesses the size of the second dimension of array x (number of columns). Because x contains independent variables, the number of columns corresponds to the number of independent variables in the model.

n_samples = x.shape[0]:
+ n_samples is a variable that stores the number of data samples (samples) used to train the model.
+ x.shape[0] accesses the size of the first dimension of array x (number of rows). Since x contains data for samples, the number of rows corresponds to the number of samples in the model.

The command to calculate R_a^2 is based on the formula above.
Python's adjusted R squared calculation results are quite different from R and Excel because the calculation model of each library is available in Python.
➔ The equation of MNLR of this data set: AQI = -137,6049 + 22,09 * CO – 11,87*NO_2 – 6,26*O_3 + 47,66*PM10 + 77,5*PM2,5 + 0,95*SO_2 fitted the model 82,2% with adjusted R squared testing model method.

b) Lab 4:

Step 1:

```
In [1]: import pandas as pd
        import numpy as np
        from scipy import stats
        import statsmodels.api as sm
        import matplotlib.pyplot as plt
        from sklearn.linear_model import LinearRegression
        from sklearn.metrics import r2_score

In [3]: df=pd.read_csv('lab5.csv')
        df
```

Out[3]:

	close	Lag1	Lag2	Lag3
0	174516.0	175578.0	174842.0	172881.0
1	171655.0	174516.0	175578.0	174842.0
2	169202.0	171655.0	174516.0	175578.0
3	172472.0	169202.0	171655.0	174516.0
4	171655.0	172472.0	169202.0	171655.0
...	...	...	...	...
1470	68100.0	68300.0	68700.0	69400.0
1471	68500.0	68100.0	68300.0	68700.0
1472	68400.0	68500.0	68100.0	68300.0
1473	67400.0	68400.0	68500.0	68100.0
1474	68400.0	67400.0	68400.0	68500.0

Figure 178. Import necessary libraries and read dataset

Step 2: Training data

- Extract independent and dependent variables from DataFrame and convert them to the appropriate format.
- Use the LinearRegression model to understand the relationship between independent and dependent variables.
- Provides access to model parameters such as intercepts and coefficients.

```
In [4]: # Get the dependent & independent variables:
        x = np.array(df[['Lag1','Lag2','Lag3']]).reshape((-1, 3))
        y = np.array(df['close']).reshape((-1, 1))
        # Using Linear Regression() to gain the model follow X and Y variables and taking result:
        reg = LinearRegression()
        reg.fit(x,y)

Out[4]: ▾ LinearRegression
        LinearRegression()
```

Figure 179. Training data for using to calculate R squared

Step 3:

```
In [6]: # Calculate adjusted R-squared
        n_features = x.shape[1]
        n_samples = x.shape[0]
        adjusted_r_squared = 1 - (1 - r_squared) * (n_samples - 1) / (n_samples - n_features - 1)

        print("Adjusted R-squared:", adjusted_r_squared)

        Adjusted R-squared: 0.994815829775244
```

Figure 180. Finding and showing the adjusted R squared

n_features = x.shape[1]:

+ n_features is a variable that stores the number of independent variables (features) in the model.

+ x.shape[1] accesses the size of the second dimension of array x (number of columns). Because x contains independent variables, the number of columns corresponds to the number of independent variables in the model.

n_samples = x.shape[0]:
+ n_samples is a variable that stores the number of data samples (samples) used to train the model.
+ x.shape[0] accesses the size of the first dimension of array x (number of rows). Since x contains data for samples, the number of rows corresponds to the number of samples in the model.

The command to calculate R_a^2 is based on the formula above.

After calculating, we check to see if the returned parameters are the same or not. If it is the same then we have followed the correct method.
=> The equation of Lab 4 data set: $Y(t)= 507.2 + 0.98779*r_{(t-1)} -0.0015*r_{(t-2)} + 0.00755*r_{(t-3)}$ fitted the model 99,48% with adjusted R squared testing model method.

REFERENCES

[1] Kotz, S.; et al., eds. (2006), Encyclopedia of Statistical Sciences, Wiley.
[2] Everitt, B. S.; Skrondal, A. (2010), The Cambridge Dictionary of Statistics, Cambridge University Press.
[3] MAPE - Mean Absolute Percentage Error. Working with Planning. Retrieved May 27, 2022.
[4] Mean Squared Error (MSE), By Jim Frost.
[5] Barnston, A., (1992). "Correspondence among the Correlation [root mean square error] and Heidke Verification Measures; Refinement of the Heidke Score." Notes and Correspondence, Climate Analysis Center. Available from here.
[6] Coefficient of Determination, R-squared, Newcastle University. Numeracy, Maths and Statistics - Academic Skills Kit (ncl.ac.uk)
[7] Osama E. Gouda, Salah H. El-Hoshy. (2021). Diagnostic technique for analysing the internal faults within power transformers based on sweep frequency response using adjusted R-square methodology - Gouda - 2020 - IET Science, Measurement & Technology - Wiley Online Library (page 3). IET Science, Measurement & Technology. doi: 10.1049/iet-smt.2020.004
[8] Shokrya Saleh. (2014). MODEL SELECTION VIA ROBUST VERSION OF R-SQUARED (page 1). Journal of Mathematics and Statistics 10 (3): 414-420, 2014. doi:10.3844/jmssp.2014.414.420
jmssp.2014.414-libre.pdf (d1wqtxts1xzle7.cloudfront.net)
[9] Julian Karch (2020). Improving on Adjusted R-Squared.
https://doi.org/10.1525/collabra.343
[10] Osama E. Gouda, Salah H. El-Hoshy. (2021). Diagnostic technique for analysing the internal faults within power transformers based on sweep frequency response using adjusted R-square methodology - Gouda - 2020 - IET Science, Measurement & Technology - Wiley Online Library (page 4). IET Science, Measurement & Technology. doi: 10.1049/iet-smt.2020.004
[11] How to Interpret Adjusted R-Squared (With Examples)
How to Interpret Adjusted R-Squared (With Examples) - Statology

- **Link of documents (Excel, R, Python):** Files of Excel, R and Python
- **Raw dataset:**
  Lab 3: Dataset on Air Quality in Vietnam in 2021 - Dataset OD Mekong Datahub (opendevelopmentmekong.net)
  Lab 4: https://vn.investing.com/equities/vietnam-dairy-products-jsc-historical-data