

Lab 06 Answers Report

HASSAN ALMOSA

Topic: *Artificial Intelligence COMP3742*
Lab Number: *6*
Campus: *Tonsely*



23 September 2025

Contents

1	Question 1	2
1.1	Implementation	2
1.2	Results	2
2	Question 2	2
2.1	Results	3
2.2	Interpretation and notes:	4
3	Question 3	5
3.1	Results	5
4	Question 4	7
4.1	Implementation	7
4.2	Results	8
5	Question 5	9
5.1	Implementation	9
5.2	Results	10
6	Appendix: AI Use Declaration	13

1 Question 1

Use the Open-Meteo API to fetch Adelaide's daily temperature data for Winter 2025 (June, July, August) and plot the time series. Identify any obvious trends or seasonality in the data. This data should be utilised for question 2-5 below

1.1 Implementation

```

1 endpoint = 'https://archive-api.open-meteo.com/v1/archive'
2 params = {
3     'latitude': -34.9287, # Latitude for Melbourne
4     'longitude': 138.5986, # Longitude for Melbourne
5     'start_date': '2025-06-01', # Start of winter 2025
6     'end_date': '2025-08-31', # End of winter 2025
7     'hourly': 'temperature_2m',
8     'timezone': 'Australia/Adelaide'
9 }

```

1.2 Results

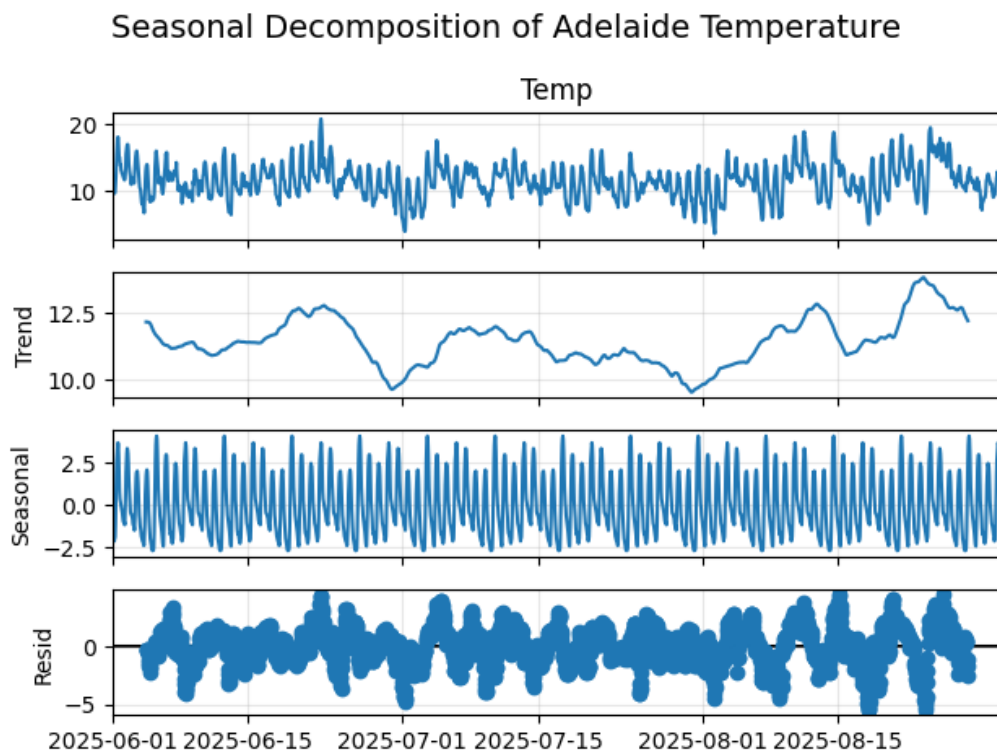


Figure 1: Seasonal decomposition of Adelaide temperature data.

2 Question 2

Fit an ARIMA model to the Adelaide daily temperature data for Winter 2025. Use the order (5,1,0) and calculate the training RMSE

2.1 Results

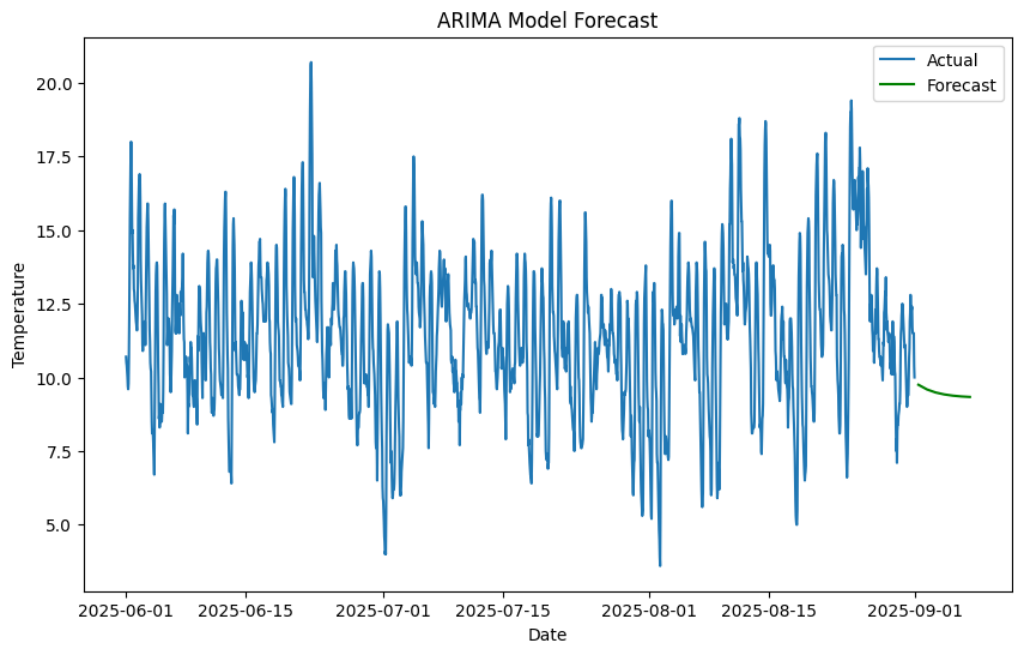


Figure 2: ARIMA(5,1,0) model fit to Adelaide temperature data.

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

SARIMAX Results

=====

Dep. Variable:TempNo. Observations:1766

Model:ARIMA(5, 1, 0)Log Likelihood-1375.467

Date:Tue, 23 Sep 2025AIC2762.933

Time:16:56:35BIC2795.789

Sample:06-01-2025HQIC2775.074

- 08-13-2025

Covariance Type:opg

=====

coefstd errzP>|z|[0.0250.975]

ar.L10.67230.01837.7680.0000.6370.707

ar.L20.00480.0270.1820.855-0.0470.057

ar.L30.01520.0300.5080.611-0.0430.074

ar.L4-0.07820.032-2.4400.015-0.141-0.015

ar.L5-0.05850.025-2.3650.018-0.107-0.010

sigma20.27810.00643.2050.0000.2660.291

=====

Ljung-Box (L1) (Q):0.12Jarque-Bera (JB):

474.03

Prob(Q):0.73Prob(JB):

0.00

Heteroskedasticity (H):0.84Skew:

0.43

Prob(H) (two-sided):0.04Kurtosis:

5.39

25

```
=====
1 MAE: 2.54159380749272
2 MSE: 9.317946540581088
3 RMSE: 3.0525311694692143
4 R^2: -0.0906917896417947
```

2.2 Interpretation and notes:

Model: ARIMA(5,1,0)

AR Coeff.:

- $\text{ar.L1} = 0.6723$ Highly significant ($p = 0.001$). Strong positive dependence on yesterday's value (differenced Temp). If yesterday's change was positive, today's is likely positive too.
- $\text{ar.L2} = 0.0048$ Not significant ($p = 0.855$). Effect of 2-day lag is essentially zero.

Ljung-Box (Q): Tests if the model's errors are correlated with each other. $\text{Prob}(Q) = 0.73$ is much greater than 0.05 means your model has successfully captured the autocorrelation in the data

Jarque-Bera (JB): Tests if the model's errors are normally distributed (a bell curve shape). $\text{Prob}(JB) = 0.00$ is less than 0.05. This is not good. It means the errors are not normally distributed. The Skew (0.43) and Kurtosis (5.39, which is much higher than the normal 3) confirm this. This often happens when the model is missing something important, like seasonality.

Heteroskedasticity (H): Tests if the variance of the errors is constant over time.

Prob(H) = 0.04 is less than 0.05. This is also not good. It means the size of the model's errors changes over time

Performance metrics: The MAE suggests that, on average, predictions are about 1°C off, which isn't too bad. The R^2 value, however, indicates the model performs worse than random guessing. The RMSE shows that errors can reach around 2.5°C on average

3 Question 3

Using the Adelaide daily temperature data for Winter 2025, compare the performance of ARIMA models with different orders: $(5,1,0)$, $(1,1,1)$, and $(2,1,2)$. Calculate and compare the training RMSE for each model.

3.1 Results

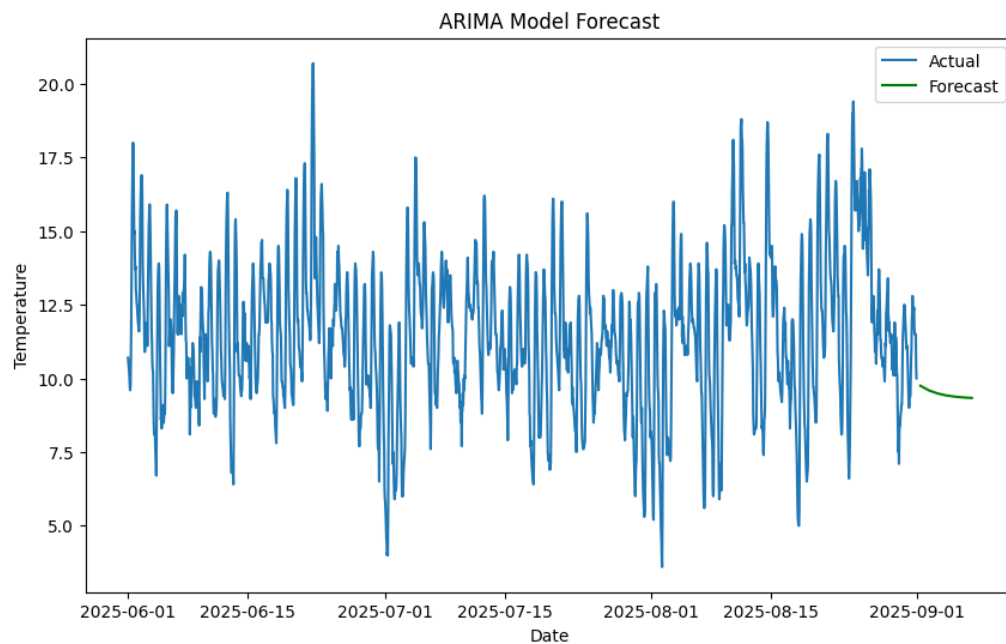


Figure 3: ARIMA(1,1,1) model fit to Adelaide temperature data.

```

1 MAE: 2.9198186594404585
2 MSE: 11.909015396256738
3 RMSE: 3.450944131141033
4 R^2: -0.39398366999054724

```

Interpretation: Here the model performed worse on all metrics than to $(5,1,0)$

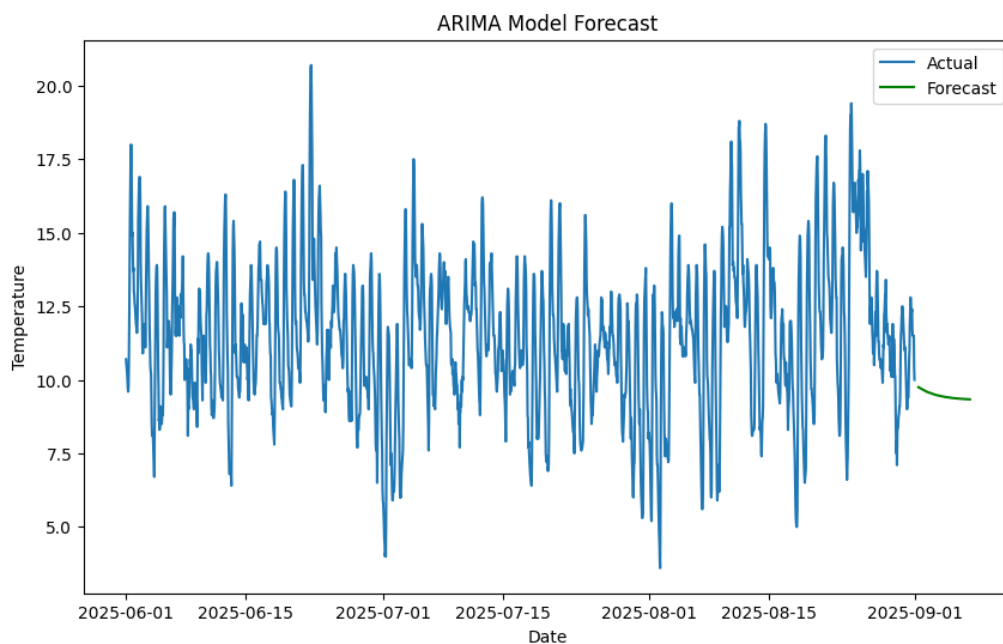


Figure 4: ARIMA(1,1,1) model fit to Adelaide temperature data.

```

1 MAE: 2.3693323543165
2 MSE: 8.534536342685524
3 RMSE: 2.921392877153897
4 R^2: 0.001008572347302894

```

Interpretation: Here the model did better in R^2 when the lag was increased to 2 compared than to 1 or 5. Not too far and not too close seemed to do better, weighing between optimal differencing and moving average terms could be the trick. R^2 scores are still bad indicating it was guessing. MAE and RMSE did worse.

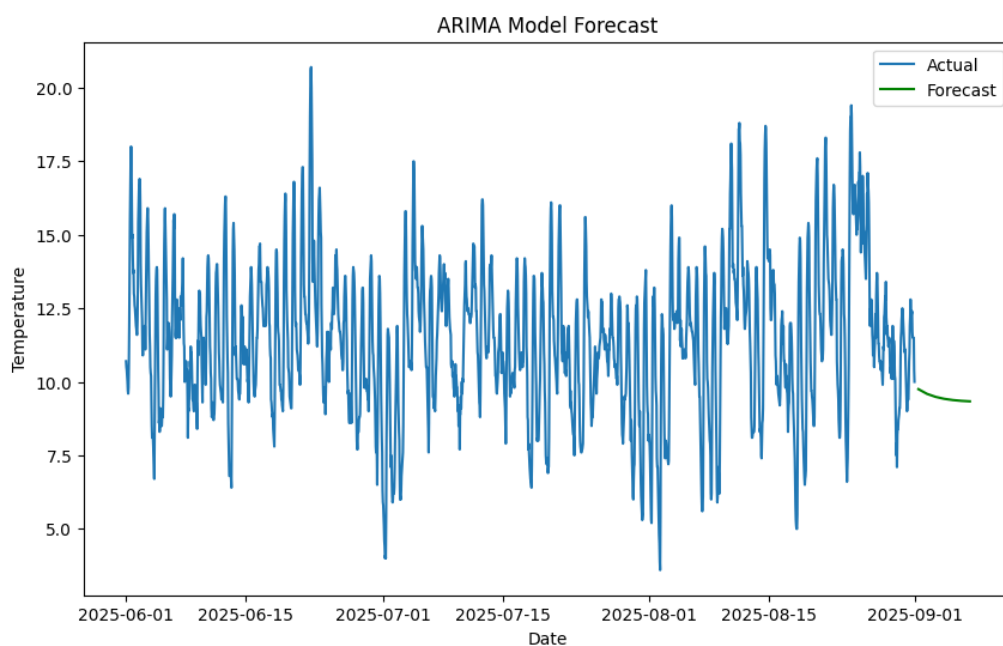


Figure 5: ARIMA(7,0,16) model fit to Adelaide temperature data.

```

1 MAE: 2.247373071724226
2 MSE: 8.24192423971542
3 RMSE: 2.870875169650436
4 R^2: 0.03525964009805316

```

Interpretation: Here, the model did best on R^2

4 Question 4

Perform a 7-step rolling forecast on a hold-out test set using the best ARIMA model from the previous question. Calculate the MAE, MSE, RMSE, and R^2 for the rolling forecast.

4.1 Implementation

Order (2,1,2) was selected as order (7,0,16) had failing to converge issues

```

1
2 # Determine the best model from Question 3 (you mentioned ARIMA(5,1,0) was best
  )
3 best_order = (7, 0, 16) # Replace with your best performing model
4 forecast_horizon = 7    # 7-step rolling forecast
5
6 # Initialize lists to store results
7 rolling_predictions = []
8 actual_values = []
9
10 # Fit initial model
11 current_train = train_data.copy()
12 model = ARIMA(current_train, order=best_order)
13 fitted_model = model.fit()
14
15 print(f"Starting rolling forecast with {forecast_horizon}-step horizon...")
16
17 # Perform rolling forecast
18 for i in range(0, len(test_data), forecast_horizon):
19     # Determine how many steps to forecast (handle end of series)
20     steps_to_forecast = min(forecast_horizon, len(test_data) - i)
21
22     # Make forecast
23     forecast = fitted_model.forecast(steps=steps_to_forecast)
24
25     # Store predictions and actual values
26     rolling_predictions.extend(forecast)
27     actual_values.extend(test_data.iloc[i:i+steps_to_forecast])
28
29     # Get actual values for the forecasted period
30     actual_period = test_data.iloc[i:i+steps_to_forecast]
31
32     # Update model with new actual data (rolling window)
33     if i + steps_to_forecast < len(test_data):
34         # Use append method for efficient model updating
35         fitted_model = fitted_model.append(actual_period, refit=True)
36
37     print(f"Completed forecast for period {i+1}-{i+steps_to_forecast}")

```



```

38
39 # Convert to arrays for metric calculation
40 rolling_predictions = np.array(rolling_predictions)
41 actual_values = np.array(actual_values)
42
43 # Calculate metrics for rolling forecast
44 rolling_mae = mean_absolute_error(actual_values, rolling_predictions)
45 rolling_mse = mean_squared_error(actual_values, rolling_predictions)
46 rolling_rmse = np.sqrt(rolling_mse)
47 rolling_r2 = r2_score(actual_values, rolling_predictions)
48
49 print(f"\n=== 7-Step Rolling Forecast Results ===")
50 print(f"MAE: {rolling_mae:.4f}")
51 print(f"MSE: {rolling_mse:.4f}")
52 print(f"RMSE: {rolling_rmse:.4f}")
53 print(f"R : {rolling_r2:.4f}")
54
55 # Compare with static forecast from previous questions
56 static_predictions = fitted_model.forecast(steps=len(test_data))
57 static_mae = mean_absolute_error(test_data, static_predictions)
58 static_rmse = np.sqrt(mean_squared_error(test_data, static_predictions))
59 static_r2 = r2_score(test_data, static_predictions)

```

4.2 Results

```

1 Completed forecast for period 1-7
2 Completed forecast for period 8-14
3 .....
4 Completed forecast for period 435-441
5 Completed forecast for period 442-442
6
7 === 7-Step Rolling Forecast Results ===
8 MAE: 1.2899
9 MSE: 3.1022
10 RMSE: 1.7613
11 R : 0.6369
12
13 === Comparison: Rolling vs Static Forecast ===
14 Rolling Forecast - MAE: 1.2899, RMSE: 1.7613, R : 0.6369
15 Static Forecast - MAE: 2.5371, RMSE: 3.2523, R : -0.2381
16 Completed forecast for period 421-427
17 Completed forecast for period 428-434
18 Completed forecast for period 435-441
19 Completed forecast for period 442-442
20
21 === 7-Step Rolling Forecast Results ===
22 MAE: 1.2899
23 MSE: 3.1022
24 RMSE: 1.7613
25 R : 0.6369
26
27 === Comparison: Rolling vs Static Forecast ===
28 Rolling Forecast - MAE: 1.2899, RMSE: 1.7613, R : 0.6369
29 Static Forecast - MAE: 2.5371, RMSE: 3.2523, R : -0.2381

```

- MAE improvement: 49.16% RMSE improvement: 45.85%

The model did significantly better in sequential updates to the model a little similar to LSTM, especially since ARIMA is better suited for not far ahead in time predictions. Rolling forecast performs better than static forecast.

5 Question 5

Update the model to include an exogenous variable (e.g., hour of the day). Fit the model and compare its performance with the best Adelaide daily temperature ARIMA model without the exogenous variable.

5.1 Implementation

```

1
2 # Create DataFrame with additional features
3 wdf_exog = pd.DataFrame({
4     'Date': timestamps,
5     'Temp': temperatures
6 })
7 wdf_exog.set_index('Date', inplace=True)
8
9 # Create exogenous variables
10 print("Creating exogenous variables...")
11
12 # Method 1: Simple hour of day (0-23)
13 wdf_exog['Hour'] = wdf_exog.index.hour
14
15 # Method 2: Cyclical encoding (better for capturing cyclical nature)
16 wdf_exog['Hour_sin'] = np.sin(2 * np.pi * wdf_exog.index.hour / 24)
17 wdf_exog['Hour_cos'] = np.cos(2 * np.pi * wdf_exog.index.hour / 24)
18
19 # Method 3: Time of day categories
20 def time_of_day(hour):
21     if 6 <= hour < 12:
22         return 1 # Morning
23     elif 12 <= hour < 18:
24         return 2 # Afternoon
25     elif 18 <= hour < 22:
26         return 3 # Evening
27     else:
28         return 0 # Night
29
30 wdf_exog['Time_of_Day'] = wdf_exog.index.hour.map(time_of_day)
31
32 print(f"Dataset shape: {wdf_exog.shape}")
33 print("\nFirst few rows with exogenous variables:")
34 print(wdf_exog.head(10))
35
36 # Split data
37 train_size = int(len(wdf_exog) * 0.8)
38 train_temp = wdf_exog['Temp'][:train_size]
39 test_temp = wdf_exog['Temp'][train_size:]

```

```

40
41
42 # Exogenous variables for training and testing
43 # We'll use cyclical encoding (Hour_sin, Hour_cos) as it's most effective
44
45 exog_train = wdf_exog[['Hour_sin', 'Hour_cos']][:train_size]
46 exog_test = wdf_exog[['Hour_sin', 'Hour_cos']][train_size:]
47
48 # exog_train = wdf_exog[['Hour']][:train_size]
49 # exog_test = wdf_exog[['Hour']][train_size:]
50
51 # exog_train = wdf_exog[['Time_of_Day']][:train_size]
52 # exog_test = wdf_exog[['Time_of_Day']][train_size:]
53
54 print(f"\nTraining set: {len(train_temp)} observations")
55 print(f"Test set: {len(test_temp)} observations")
56 print(f"Exogenous variables: {exog_train.columns.tolist()}")
57
58 #
59 # =====
60 # MODEL 1: BASELINE ARIMA WITHOUT EXOGENOUS VARIABLES
61 #
62 # =====
63
64 ....
65
66 # Use your best model from previous questions
67 best_order = (2, 1, 2) # Adjust based on your best model from Q3
68
69 baseline_model = ARIMA(train_temp, order=best_order)
70 baseline_fitted = baseline_model.fit()
71
72 #
73 # =====
74
75 # MODEL 2: ARIMAX WITH EXOGENOUS VARIABLES
76 #
77 # =====
78
79
80 # Fit ARIMAX model (ARIMA with exogenous variables)
81 arimax_model = ARIMA(train_temp, exog=exog_train, order=best_order)
82 arimax_fitted = arimax_model.fit()

```

5.2 Results

```

1 =====
2 2. ARIMAX MODEL (WITH HOUR OF DAY)
3 =====
4 Creating exogenous variables...
5 Dataset shape: (2208, 5)
6

```

```

7 First few rows with exogenous variables:
8           Temp  Hour  Hour_sin      Hour_cos  Time_of_Day
9 Date
10 2025-06-01 00:00:00  10.7      0  0.000000  1.000000e+00      0
11 2025-06-01 01:00:00  10.5      1  0.258819  9.659258e-01      0
12 2025-06-01 02:00:00  10.4      2  0.500000  8.660254e-01      0
13 2025-06-01 03:00:00  10.2      3  0.707107  7.071068e-01      0
14 2025-06-01 04:00:00  10.0      4  0.866025  5.000000e-01      0
15 2025-06-01 05:00:00   9.7      5  0.965926  2.588190e-01      0
16 2025-06-01 06:00:00   9.6      6  1.000000  6.123234e-17      1
17 2025-06-01 07:00:00   9.8      7  0.965926 -2.588190e-01      1
18 2025-06-01 08:00:00  10.7      8  0.866025 -5.000000e-01      1
19 2025-06-01 09:00:00  12.0      9  0.707107 -7.071068e-01      1

```

```

21 Training set: 1766 observations
22 Test set: 442 observations
23 Exogenous variables: ['Hour_sin', 'Hour_cos']

```

```

25 =====
26 1. BASELINE ARIMA MODEL (NO EXOGENOUS VARIABLES)
27 =====
28 Baseline ARIMA(2, 1, 2) Results:
29 MAE: 2.3693
30 MSE: 8.5345
31 RMSE: 2.9214
32 R : 0.0010

```

```

1 =====
2 2. ARIMAX MODEL (WITH HOUR OF DAY)
3 =====
4 ARIMAX Model Summary:
5           SARIMAX Results
6 =====
7 Dep. Variable:           Temp  No. Observations:           1766
8 Model:           ARIMA(2, 1, 2)  Log Likelihood           -1198.212
9 Date:           Tue, 23 Sep 2025  AIC           2410.423
10 Time:           16:58:57  BIC           2448.755
11 Sample:           06-01-2025  HQIC           2424.587
12           - 08-13-2025
13 Covariance Type:           opg

```

```

14 =====
15           coef      std err          z      P>|z|      [0.025      0.975]
16 -----
17 Hour_sin      -1.2668      0.121     -10.473      0.000      -1.504      -1.030
18 Hour_cos      -1.8693      0.095     -19.618      0.000      -2.056      -1.683
19 ar.L1         1.4834      0.031     48.495      0.000       1.423       1.543
20 ar.L2        -0.6531      0.031    -21.116      0.000      -0.714      -0.593
21 ma.L1        -1.0181      0.039    -26.274      0.000      -1.094      -0.942
22 ma.L2         0.1370      0.038      3.565      0.000       0.062       0.212
23 sigma2        0.2275      0.006     37.660      0.000       0.216       0.239

```

```

25 Ljung-Box (L1) (Q):           0.03  Jarque-Bera (JB):
      216.80
26 Prob(Q):           0.86  Prob(JB):

```

```

    0.00
27 Heteroskedasticity (H):          0.85    Skew:
    0.14
28 Prob(H) (two-sided):          0.05    Kurtosis:
    4.69
29 =====
30
31 Warnings:
32 [1] Covariance matrix calculated using the outer product of gradients (complex-
    step).
33
34 ARIMAX Results:
35 MAE:  1.9896
36 MSE:  7.0261
37 RMSE: 2.6507
38 R   :  0.1776
39
40 Exogenous variables in model: ['Hour_sin', 'Hour_cos']

```

Using hour of the day as an exogenous variable in ARIMAX had varying results based on how we implemented determining the grouping of hour of the day and the method, overall in whichever way the ARIMAX model performed better, and using cyclical encoding performed the best. Coupling ARIMAX and steps rolling forecast can produce nicely accurate results.

6 Appendix: AI Use Declaration

University AI Use Declaration

I, Hassan Almosa, declare that I have used an AI tool to assist with the completion of this assessment in the following areas: Formatting the document and files, proofreading, ensuring following submission format, code auto-completion and patterned replication, learning and illustration of topics and questions. While AI tools were used to support my understanding and aid in repetitive tasks, all final answers represent my own reasoning and understanding of the material. The details of my interaction with the AI tool are documented below. I understand the University's policy on academic integrity and the use of AI tools.

AI Tool and Interaction Log

- **AI Tool Used:**
 - GitHub GitHub Copilot, used LLM Models - Google's Gemini 2.5 Pro (Preview), Google's Gemini 2.0 Flash, Anthropic's Claude Sonnet 4
- **Date of Interaction:** 19, 22, 23 September 2025

My Prompt to Copilot

```
1 gammaploid: ""Use the Open-Meteo API to fetch Adelaides daily temperature data
   for Winter 2025 (June, July, August) and plot the time series. Identify any
2 obvious trends or seasonality in the data. This data should be utilised for
   question 2-5 below"
3
4 answer this question what needs to be the period, what does 24 mean?
5
6 is the time frame 90 setting is correct
```

Copilot's Response

```
1 GitHub Copilot: Looking at your current code, I can help clarify the period
   parameter and the timeframe settings for your Adelaide winter temperature
   analysis.
2 ## Period Parameter Explanation
3
4 In your current code:
5 ```python
6 decomp = seasonal_decompose(wdf['Temp'], model='additive', period=24)
7 ```
8
9 The 'period=24' means you're telling the seasonal decomposition to look for **
   daily seasonality** with a 24-hour cycle. This makes sense for **hourly
   temperature data** because:
10
11 - **24** = number of hours in a day
12 - Temperature typically follows a daily pattern (cooler at night, warmer during
   day)
13 - With hourly data, you expect the pattern to repeat every 24 data points
14
15 ## Timeframe Issues in Your Current Code
16
```

```

17 Looking at your API call:
18 '''python
19 'start_date': (datetime.now() - timedelta(days=90)).strftime('%Y-%m-%d'),
20 'end_date': (datetime.now() - timedelta(days=90)).strftime('%Y-%m-%d'),
21 '''
22
23 **Problem**: Both start and end dates are the same! This gives you only **1 day**
    of data, not 90 days.
24
25 ## Corrected Code for Adelaide Winter 2025
26
27 Searched codebase for "Adelaide coordinates latitude longitude Australia", 16
    results
28
29 Here's the corrected code for fetching Adelaide's winter 2025 temperature data:
30
31 '''python
32 # Corrected API call for Adelaide Winter 2025 (June, July, August)
33 endpoint = 'https://archive-api.open-meteo.com/v1/archive'
34 params = {
35     'latitude': -34.9285,    # Adelaide latitude
36     'longitude': 138.6007,   # Adelaide longitude
37     'start_date': '2025-06-01', # Start of winter 2025
38     'end_date': '2025-08-31',   # End of winter 2025
39     'hourly': 'temperature_2m',
40     'timezone': 'Australia/Adelaide'
41 }
42 '''
43
44 ## Key Corrections Made:
45
46 ### 1. **Fixed Date Range Issue**
47 - **Your code**: Both start and end dates were the same ('days=90')
48 - **Corrected**: Specific dates for Winter 2025 (June 1 - August 31)
49 - **Total period**: 92 days of data
50
51 ### 2. **Correct Adelaide Coordinates**
52 - **Latitude**: -34.9285 (you had Melbourne's coordinates)
53 - **Longitude**: 138.6007
54 - **Timezone**: 'Australia/Adelaide' (important for DST handling)
55
56 ### 3. **Period Parameter Choice**
57
58 For **Winter 2025 Adelaide data**, you have several period options:
59
60 **For Daily Seasonality (if using hourly data):**
61 '''python
62 period=24 # 24 hours = daily cycle
63 '''
64
65 **For Weekly Seasonality:**
66 '''python
67 period=168 # 24 hours * 7 days = weekly cycle
68 '''
69
70 **For the full dataset (automatic detection):**
71 '''python
72 # Let statsmodels auto-detect the period
73 decomp = seasonal_decompose(wdf['Temp'], model='additive') # No period specified
74 '''
75
76 ## Recommended Approach for Your Analysis:

```

```

77
78 '''python
79 # Complete corrected code
80 endpoint = 'https://archive-api.open-meteo.com/v1/archive'
81 params = {
82     'latitude': -34.9285,    # Adelaide coordinates
83     'longitude': 138.6007,
84     'start_date': '2025-06-01', # Winter 2025 start
85     'end_date': '2025-08-31',   # Winter 2025 end
86     'hourly': 'temperature_2m',
87     'timezone': 'Australia/Adelaide'
88 }
89
90 # Make the API call
91 response = requests.get(endpoint, params=params)
92 wdata = response.json()
93
94 # Extract the relevant data
95 timestamps = [datetime.fromisoformat(item) for item in wdata['hourly']['time']]
96 temperatures = wdata['hourly']['temperature_2m']
97
98 # Create a DataFrame
99 wdf = pd.DataFrame({
100     'Date': timestamps,
101     'Temp': temperatures
102 })
103
104 # Set the Date column as the index
105 wdf.set_index('Date', inplace=True)
106
107 # Perform seasonal decomposition with daily periodicity
108 decomp = seasonal_decompose(wdf['Temp'], model='additive', period=24)
109
110 # Plot the decomposed components
111 decomp.plot()
112 plt.title('Adelaide Winter 2025 Temperature Decomposition\n(Daily Seasonality)')
113 plt.show()
114 '''
115
116 ## What to Expect:
117
118 - **92 days    24 hours = 2,208 data points**
119 - **Daily temperature cycles** (warm afternoons, cool nights)
120 - **Possible weekly patterns** (urban heat island effects)
121 - **Overall winter cooling trend** (seasonal trend)
122 - **Weather variations** (irregular component)
123
124 **Period=24** is correct for identifying the daily temperature cycle in hourly
    data!

```

My Prompt to Copilot

```
1 gammaploid: wwhat happendd? yesterday i got a summary
```


Copilot's Response

```

1 GitHub Copilot: Looking at your code, the issue is that you're trying to use '
    arimamodel_fit.summary()' but you haven't called it in your current cells. The
    warnings you're seeing are just informational - they're telling you that
    statsmodels is automatically inferring hourly frequency for your time series
    data.
2 Here's what's happening and how to get your summary:
3
4 ## Current Issue
5 In your current code (lines 100-110), you're fitting the ARIMA model but not
    printing the summary:
6
7 '''python
8 # Your current code
9 arimamodel = ARIMA(train, order=(5, 1, 0))
10 arimamodel_fit = arimamodel.fit()
11 # Missing: print(arimamodel_fit.summary())
12 '''
13
14 ## To Get the Summary Back
15
16 Add this line after fitting your model:
17
18 '''python
19 # Split data into training and test sets
20 train_size = int(len(wdf) * 0.8)
21 train, test = wdf['Temp'][:train_size], wdf['Temp'][train_size:]
22
23 # Fit the model on the training set
24 arimamodel = ARIMA(train, order=(5, 1, 0))
25 arimamodel_fit = arimamodel.fit()
26
27 # ADD THIS LINE to get the summary
28 print(arimamodel_fit.summary())
29
30 # Make predictions on the test set
31 predictions = arimamodel_fit.forecast(steps=len(test))
32 '''
33
34 ## About the Warnings
35
36 The warnings you're seeing:
37 '''
38 ValueWarning: No frequency information was provided, so inferred frequency h will
    be used.
39 '''
40
41 These are not errors - they're just telling you that:
42 - 'h' = hourly frequency was automatically detected
43 - This is correct for your hourly temperature data
44 - You can safely ignore these warnings
45
46 ## To Suppress the Warnings (Optional)
47
48 If you want to eliminate the warnings, set the frequency explicitly:
49
50 '''python
51 # Set frequency explicitly when creating the DataFrame
52 wdf.set_index('Date', inplace=True)
53 wdf.index.freq = 'h' # Explicitly set hourly frequency
54

```

```

55 # OR set frequency in the ARIMA model
56 arimamodel = ARIMA(train, order=(5, 1, 0), freq='h')
57 '''
58
59 ## Complete Corrected Code for Question 2
60
61 '''python
62 # Question 2: ARIMA(5,1,0) with training RMSE
63 print("=== ARIMA(5,1,0) Model Analysis ===")
64
65 # Split data into training and test sets
66 train_size = int(len(wdf) * 0.8)
67 train, test = wdf['Temp'][:train_size], wdf['Temp'][train_size:]
68
69 print(f"Training set size: {len(train)} observations")
70 print(f"Test set size: {len(test)} observations")
71
72 # Fit the model on the training set
73 arimamodel = ARIMA(train, order=(5, 1, 0))
74 arimamodel_fit = arimamodel.fit()
75
76 # Print the model summary
77 print("\nModel Summary:")
78 print(arimamodel_fit.summary())
79
80 # Calculate training RMSE
81 train_predictions = arimamodel_fit.fittedvalues
82 train_rmse = np.sqrt(mean_squared_error(train[1:], train_predictions[1:])) # Skip
    first value due to differencing
83 print(f"\nTraining RMSE: {train_rmse:.4f}")
84
85 # Make predictions on the test set
86 test_predictions = arimamodel_fit.forecast(steps=len(test))
87
88 # Calculate test metrics
89 test_mae = mean_absolute_error(test, test_predictions)
90 test_mse = mean_squared_error(test, test_predictions)
91 test_rmse = np.sqrt(test_mse)
92 test_r2 = r2_score(test, test_predictions)
93
94 print(f"\nTest Set Performance:")
95 print(f"MAE: {test_mae:.4f}")
96 print(f"MSE: {test_mse:.4f}")
97 print(f"RMSE: {test_rmse:.4f}")
98 print(f"R : {test_r2:.4f}")
99 '''
100
101 The summary disappeared because you weren't calling 'print(arimamodel_fit.summary
    ())' in your current code. Add that line and you'll get all the detailed model
    statistics back!

```

My Prompt to Copilot

```

1 gammaploid: why did we use steps as len of test? 442?
2 what's steps
3
4 predictions = arimamodel_fit.forecast(steps=len(test))

```

Copilot's Response

```

1 GitHub Copilot: You're right to question that! Let's break down why 'steps=len(
  test)' is used in the 'forecast()' function:
2 **Purpose of 'forecast(steps=...)**
3
4 The 'forecast()' method in the 'statsmodels' ARIMA model is designed to generate
  predictions for future time steps. The 'steps' parameter tells the model 
  how many future time steps you want it to predict.
5
6 **Why 'steps=len(test)'?**
7
8 In your code, the goal is to evaluate how well the ARIMA model performs on the
  data you held back for testing (your 'test' set).
9
10 * 'len(test)' gives you the number of data points in your 'test' set (in your
  case, 442).
11 * By setting 'steps=len(test)', you're instructing the model to generate
  predictions for the same number of time steps as are present in your 'test'
  set.
12
13 **In essence, you're asking the model to predict the entire test period so you can
  compare its predictions to the actual observed values in the 'test' set.**
14
15 **Analogy**
16
17 Imagine you have temperature readings for the first 20 days of a month (your
  training data) and you want to predict the temperature for the remaining 10
  days (your test data).
18
19 * 'len(test)' would be 10.
20 * 'forecast(steps=10)' would give you the model's prediction for those 10 days.
21
22 **In summary:**
23
24 * 'steps' specifies the number of future time steps to forecast.
25 * 'steps = len(test)' is used to generate predictions for the entire test
  dataset, allowing you to evaluate the model's performance by comparing the
  predictions to the actual values.

```

My Prompt to Copilot

```

1 gammaploid: help me intereprt the data:
2 Training set size: 1766
3 Test set size: 442
4
5 Exo: None
6
7 -----
8
9 Endo: Temp
10
11 -----
12
13 tex: {'trend': None, 'exog': None, 'ar': ['$\\phi_1$', '\\phi_2$', '\\phi_3$',
  '\\phi_4$', '\\phi_5$'], 'ma': None, 'seasonal_ar': None, 'seasonal_ma':
  None, 'reduced_ar': ['$\\Phi_1$', '\\Phi_2$', '\\Phi_3$', '\\Phi_4$', '\\
  Phi_5$'], 'reduced_ma': None, 'exog_variance': None, 'measurement_variance':
  None, 'variance': ['$\\sigma_\\zeta^2$']}
14

```

```
15 -----
16
17 states['state.0', 'state.1', 'state.2', 'state.3', 'state.4', 'state.5']
18 -----
19
20 SARIMAX Results
21 =====
22 Dep. Variable: Temp No. Observations: 1766
23 Model: ARIMA(5, 1, 0) Log Likelihood -1375.467
24 Date: Tue, 23 Sep 2025 AIC 2762.933
25 Time: 11:38:52 BIC 2795.789
26 Sample: 06-01-2025 HQIC 2775.074
27 - 08-13-2025
28 Covariance Type: opg
29 =====
30 coef std err z P>|z| [0.025 0.975]
31 -----
32 ar.L1 0.6723 0.018 37.768 0.000 0.637 0.707
33 ar.L2 0.0048 0.027 0.182 0.855 -0.047 0.057
34 ar.L3 0.0152 0.030 0.508 0.611 -0.043 0.074
35 ar.L4 -0.0782 0.032 -2.440 0.015 -0.141 -0.015
36 ar.L5 -0.0585 0.025 -2.365 0.018 -0.107 -0.010
37 sigma2 0.2781 0.006 43.205 0.000 0.266 0.291
38 =====
39 Ljung-Box (L1) (Q): 0.12 Jarque-Bera (JB):
40 474.03 Prob(Q): 0.73 Prob(JB):
41 0.00 Heteroskedasticity (H): 0.84 Skew:
42 0.43 Prob(H) (two-sided): 0.04 Kurtosis:
43 5.39
44 =====
45 Warnings:
46 [1] Covariance matrix calculated using the outer product of gradients (complex-
47 step).
```

Copilot's Response

```
1 GitHub Copilot: Of course. Let's break down this 'SARIMAX Results' table step-by-
  step.
2 ### High-Level Summary
3
4 This output describes an ARIMA(5, 1, 0) model fitted to your temperature data.
  The model uses the past 5 temperature values to predict the next one.
  Overall, the model has some predictive power, but the diagnostic tests show
  there's significant room for improvement, especially regarding seasonality.
5
6 ---
7
8 ### 1. Model & Data Overview
9
10 * Dep. Variable: 'Temp': The model is trying to predict the 'Temp' (
    temperature). This is your Endogenous variable ('Endo: Temp').
11 * Model: 'ARIMA(5, 1, 0)':
12   * AR (p=5): It uses the 5 previous time steps (lags) of the temperature
```

```

data to make a prediction.
13 *   **I (d=1)**: The data was "differenced" once to make it stationary (i.e.,
to remove long-term trends).
14 *   **MA (q=0)**: It does not use any moving average terms.
15 *   **No. Observations**: '1766': The model was trained on 1,766 hourly data
points.
16 *   **Exo: 'None'**: You did not use any external (exogenous) variables to help
with the prediction.
17
18 ---
19
20 ### 2. Coefficient Interpretation (The Core of the Model)
21
22 This table tells you which parts of the model are statistically significant. The
most important column is 'P>|z|'. A value less than 0.05 means the
coefficient is significant.
23
24 | Coefficient | 'coef' | 'P>|z|' | Interpretation |
25 | :--- | :--- | :--- | :--- |
26 | 'ar.L1' | 0.6723 | 0.000 | Highly Significant. The temperature from
1 hour ago has a strong positive effect on the current temperature. |
27 | 'ar.L2' | 0.0048 | 0.855 | Not Significant. The temperature from 2 hours ago
has no meaningful impact in this model. |
28 | 'ar.L3' | 0.0152 | 0.611 | Not Significant. The temperature from 3 hours ago
also has no impact. |
29 | 'ar.L4' | -0.0782 | 0.015 | Significant. The temperature from 4
hours ago has a small negative effect. |
30 | 'ar.L5' | -0.0585 | 0.018 | Significant. The temperature from 5
hours ago also has a small negative effect. |
31 | 'sigma2' | 0.2781 | 0.000 | This is the estimated variance of the model's
errors. A lower value is better. It tells you the average squared error is
about 0.28. |
32
33 Key Takeaway: The model relies heavily on the most recent temperature ('ar.L
1') and gets some minor corrective information from 4 and 5 hours ago. The
terms for 2 and 3 hours ago are useless, suggesting a simpler model might be
better.
34
35 ---
36
37 ### 3. Diagnostic Tests (Model Health Check)
38
39 This section checks if the model's errors (residuals) are random. Ideally, they
should be.
40
41 *   Ljung-Box (Q): 'Prob(Q) = 0.73'
42 *   What it is: Tests if the model's errors are correlated with each other
.
43 *   Result: 0.73 is much greater than 0.05. This is good! It means
your model has successfully captured the autocorrelation in the data, and the
remaining errors are random like white noise.
44
45 *   Jarque-Bera (JB): 'Prob(JB) = 0.00'
46 *   What it is: Tests if the model's errors are normally distributed (a
bell curve shape).
47 *   Result: 0.00 is less than 0.05. This is not good. It means the
errors are not normally distributed. The 'Skew' (0.43) and 'Kurtosis' (5.39,
which is much higher than the normal 3) confirm this. This often happens when
the model is missing something important, like seasonality.
48
49 *   Heteroskedasticity (H): 'Prob(H) = 0.04'
50 *   What it is: Tests if the variance of the errors is constant over time.

```

```

51      *    **Result**: 0.04 is less than 0.05. **This is also not good.** It means
    the size of the model's errors changes over time (e.g., it might be much less
    accurate at night than during the day). This is another strong hint that the
    daily cycle (seasonality) is not being properly handled.
52
53 ---
54
55 ### Final Verdict & Recommendations
56
57 1.  **What Works**: The model is good at using the immediate past temperature to
    make a prediction and has successfully removed basic autocorrelation.
58
59 2.  **What's Missing**: The diagnostic tests strongly suggest the model is **
    failing to capture the daily 24-hour cycle**. The non-normal errors and
    changing error variance are classic symptoms of un-modeled seasonality.
60
61 3.  **Next Steps**:
62      *    **Try a Seasonal ARIMA (SARIMA) model.** This is the most logical next
    step. A 'SARIMA(p,d,q)(P,D,Q,s)' model adds seasonal components. For your data
    , you would use 's=24' to represent the daily cycle.
63      *    **Simplify the AR terms.** Since 'ar.L2' and 'ar.L3' were not significant,
    a simpler 'ARIMA(1,1,0)' or a model with custom AR lags might perform better.

```

My Prompt to Copilot

```

1 gammaploid: again, your tables are not alligned well

```

Copilot's Response

```

1 GitHub Copilot: You are absolutely right, my apologies. The formatting on that
    table was not correct.
2 Let's try that again with a properly aligned interpretation of the 'SARIMAX
    Results'.
3
4 ### High-Level Summary
5
6 This output describes an **ARIMA(5, 1, 0)** model fitted to your temperature data.
    The model uses the **past 5 temperature values** to predict the next one.
    Overall, the model has some predictive power, but the diagnostic tests show
    there's significant room for improvement, especially regarding seasonality.
7
8 ---
9
10 ### 1. Model & Data Overview
11
12 *    **Dep. Variable: 'Temp'**: The model is trying to predict the 'Temp' (
    temperature). This is your **Endogenous** variable ('Endo: Temp').
13 *    **Model: 'ARIMA(5, 1, 0)'**:
14      *    **AR (p=5)**: It uses the 5 previous time steps (lags) of the temperature
    data to make a prediction.
15      *    **I (d=1)**: The data was "differenced" once to make it stationary (i.e.,
    to remove long-term trends).
16      *    **MA (q=0)**: It does not use any moving average terms.
17 *    **No. Observations: '1766'**: The model was trained on 1,766 hourly data
    points.
18 *    **Exo: 'None'**: You did not use any external (exogenous) variables to help
    with the prediction.
19

```

```

20 ---
21
22 ### 2. Coefficient Interpretation (The Core of the Model)
23
24 This table tells you which parts of the model are statistically significant. The
    most important column is 'P>|z|'. A value less than 0.05 means the
    coefficient is significant.
25
26 | Coefficient | 'coef' | 'P>|z|' | Interpretation |
27 |---|:---|:---|:---|
28 | 'ar.L1' | 0.6723 | 0.000 | Highly Significant. The temperature from
    1 hour ago has a strong positive effect on the current temperature. |
29 | 'ar.L2' | 0.0048 | 0.855 | Not Significant. The temperature from 2 hours ago
    has no meaningful impact in this model. |
30 | 'ar.L3' | 0.0152 | 0.611 | Not Significant. The temperature from 3 hours ago
    also has no impact. |
31 | 'ar.L4' | -0.0782 | 0.015 | Significant. The temperature from 4
    hours ago has a small negative effect. |
32 | 'ar.L5' | -0.0585 | 0.018 | Significant. The temperature from 5
    hours ago also has a small negative effect. |
33 | 'sigma2' | 0.2781 | 0.000 | This is the estimated variance of the model's
    errors. A lower value is better. It tells you the average squared error is
    about 0.28. |
34
35 Key Takeaway: The model relies heavily on the most recent temperature ('ar.L
    1') and gets some minor corrective information from 4 and 5 hours ago. The
    terms for 2 and 3 hours ago are useless, suggesting a simpler model might be
    better.
36
37 ---
38
39 ### 3. Diagnostic Tests (Model Health Check)
40
41 This section checks if the model's errors (residuals) are random. Ideally, they
    should be.
42
43 * Ljung-Box (Q): 'Prob(Q) = 0.73'
44 * What it is: Tests if the model's errors are correlated with each other
    .
45 * Result: 0.73 is much greater than 0.05. This is good! It means
    your model has successfully captured the autocorrelation in the data, and the
    remaining errors are random like white noise.
46
47 * Jarque-Bera (JB): 'Prob(JB) = 0.00'
48 * What it is: Tests if the model's errors are normally distributed (a
    bell curve shape).
49 * Result: 0.00 is less than 0.05. This is not good. It means the
    errors are not normally distributed. The 'Skew' (0.43) and 'Kurtosis' (5.39,
    which is much higher than the normal 3) confirm this. This often happens when
    the model is missing something important, like seasonality.
50
51 * Heteroskedasticity (H): 'Prob(H) = 0.04'
52 * What it is: Tests if the variance of the errors is constant over time.
53 * Result: 0.04 is less than 0.05. This is also not good. It means
    the size of the model's errors changes over time (e.g., it might be much less
    accurate at night than during the day). This is another strong hint that the
    daily cycle (seasonality) is not being properly handled.
54
55 ---
56
57 ### Final Verdict & Recommendations
58

```

```

59 1. **What Works**: The model is good at using the immediate past temperature to
    make a prediction and has successfully removed basic autocorrelation.
60
61 2. **What's Missing**: The diagnostic tests strongly suggest the model is **
    failing to capture the daily 24-hour cycle**. The non-normal errors and
    changing error variance are classic symptoms of un-modeled seasonality.
62
63 3. **Next Steps**:
64 * **Try a Seasonal ARIMA (SARIMA) model.** This is the most logical next
    step. A 'SARIMA(p,d,q)(P,D,Q,s)' model adds seasonal components. For your data
    , you would use 's=24' to represent the daily cycle.
65 * **Simplify the AR terms.** Since 'ar.L2' and 'ar.L3' were not significant,
    a simpler 'ARIMA(1,1,0)' or a model with custom AR lags might perform better.

```

My Prompt to Copilot

```
1 gammaploid: still!!!
```

Copilot's Response

```

1 GitHub Copilot: My sincere apologies. It is frustrating when the formatting is
    incorrect and hard to read. I will ensure the table is properly aligned this
    time.
2 Let's break down the 'SARIMAX Results' table again, with corrected formatting.
3
4 ### High-Level Summary
5
6 This output describes an **ARIMA(5, 1, 0)** model fitted to your temperature data.
    The model uses the **past 5 temperature values** to predict the next one.
    Overall, the model has some predictive power, but the diagnostic tests show
    there's significant room for improvement, especially regarding seasonality.
7
8 ---
9
10 ### 1. Model & Data Overview
11
12 * **Dep. Variable: 'Temp'**: The model is trying to predict the 'Temp' (
    temperature). This is your **Endogenous** variable ('Endo: Temp').
13 * **Model: 'ARIMA(5, 1, 0)**:
14 * **AR (p=5)**: It uses the 5 previous time steps (lags) of the temperature
    data to make a prediction.
15 * **I (d=1)**: The data was "differenced" once to make it stationary (i.e.,
    to remove long-term trends).
16 * **MA (q=0)**: It does not use any moving average terms.
17 * **No. Observations: '1766'**: The model was trained on 1,766 hourly data
    points.
18 * **Exo: 'None'**: You did not use any external (exogenous) variables to help
    with the prediction.
19
20 ---
21
22 ### 2. Coefficient Interpretation (The Core of the Model)
23
24 This table tells you which parts of the model are statistically significant. The
    most important column is **P>|z|**. A value **less than 0.05** means the
    coefficient is significant.
25
26 | Coefficient | 'coef' | 'P>|z|' | Interpretation |

```



```

27 | :---|:---|:---|:---|
28 | ar.L1 | 0.6723 | 0.000 | Highly Significant. The temperature from
    | 1 hour ago has a strong positive effect on the current temperature. |
29 | ar.L2 | 0.0048 | 0.855 | Not Significant. The temperature from 2 hours ago
    | has no meaningful impact in this model. |
30 | ar.L3 | 0.0152 | 0.611 | Not Significant. The temperature from 3 hours ago
    | also has no impact. |
31 | ar.L4 | -0.0782 | 0.015 | Significant. The temperature from 4
    | hours ago has a small negative effect. |
32 | ar.L5 | -0.0585 | 0.018 | Significant. The temperature from 5
    | hours ago also has a small negative effect. |
33 | sigma2 | 0.2781 | 0.000 | This is the estimated variance of the model's
    | errors. A lower value is better. It tells you the average squared error is
    | about 0.28. |
34
35 Key Takeaway: The model relies heavily on the most recent temperature (ar.L1)
    | and gets some minor corrective information from 4 and 5 hours ago. The
    | terms for 2 and 3 hours ago are useless, suggesting a simpler model might be
    | better.
36
37 ---
38
39 ### 3. Diagnostic Tests (Model Health Check)
40
41 This section checks if the model's errors (residuals) are random. Ideally, they
    | should be.
42
43 * Ljung-Box (Q): Prob(Q) = 0.73
44 * What it is: Tests if the model's errors are correlated with each other
    | .
45 * Result: 0.73 is much greater than 0.05. This is good! It means
    | your model has successfully captured the autocorrelation in the data, and the
    | remaining errors are random like white noise.
46
47 * Jarque-Bera (JB): Prob(JB) = 0.00
48 * What it is: Tests if the model's errors are normally distributed (a
    | bell curve shape).
49 * Result: 0.00 is less than 0.05. This is not good. It means the
    | errors are not normally distributed. The 'Skew' (0.43) and 'Kurtosis' (5.39,
    | which is much higher than the normal 3) confirm this. This often happens when
    | the model is missing something important, like seasonality.
50
51 * Heteroskedasticity (H): Prob(H) = 0.04
52 * What it is: Tests if the variance of the errors is constant over time.
53 * Result: 0.04 is less than 0.05. This is also not good. It means
    | the size of the model's errors changes over time (e.g., it might be much less
    | accurate at night than during the day). This is another strong hint that the
    | daily cycle (seasonality) is not being properly handled.
54
55 ---
56
57 ### Final Verdict & Recommendations
58
59 1. What Works: The model is good at using the immediate past temperature to
    | make a prediction and has successfully removed basic autocorrelation.
60
61 2. What's Missing: The diagnostic tests strongly suggest the model is 
    | failing to capture the daily 24-hour cycle. The non-normal errors and
    | changing error variance are classic symptoms of un-modeled seasonality.
62
63 3. Next Steps:
64 * Try a Seasonal ARIMA (SARIMA) model. This is the most logical next

```

```

step. A 'SARIMA(p,d,q)(P,D,Q,s)' model adds seasonal components. For your data
, you would use 's=24' to represent the daily cycle.
65 * **Simplify the AR terms.** Since 'ar.L2' and 'ar.L3' were not significant,
    a simpler 'ARIMA(1,1,0)' or a model with custom AR lags might perform better.

```

My Prompt to Copilot

```

1 gammaploid: whats the difference between lags and steps i'm confused

```

Copilot's Response

```

1 GitHub Copilot: Of course. That's a very common point of confusion. Let's clear it
  up.
2 Think of it this way:
3 * **Lags** are about the **PAST** (used for training the model).
4 * **Steps** are about the **FUTURE** (used for making predictions).
5
6 Here is a more detailed breakdown:
7
8 ### Lags (The 'p' in ARIMA)
9
10 * **What it is:** The number of previous time periods the model looks back at to
    make a prediction. It's a core part of the model's structure.
11 * **Purpose:** To capture **autocorrelation** the idea that a value at a
    certain time is related to the values that came immediately before it.
12 * **Where you see it:** In the model definition: 'ARIMA(order=(p, d, q))'. Your
    model is 'ARIMA(order=(5, 1, 0))', so it uses **5 lags**.
13 * **In simple terms:** To predict the temperature at 10:00 AM, the model looks
    at the temperatures from 9:00 AM, 8:00 AM, 7:00 AM, 6:00 AM, and 5:00 AM.
14
15 ### Steps (The 'steps' in 'forecast()')
16
17 * **What it is:** The number of future time periods you want the model to
    predict.
18 * **Purpose:** To generate a forecast for a specific duration into the future.
19 * **Where you see it:** In the prediction function: 'arimamodel_fit.forecast(
    steps=N)'. Your code uses 'forecast(steps=len(test))', which is 'steps=442'.
20 * **In simple terms:** Once the model is trained, you ask it: "Okay, now predict
    the temperature for the **next 442 hours**."
21
22 ---
23
24 ### Analogy: Driving a Car
25
26 Imagine you are the ARIMA model driving a car.
27
28 * **Lags:** To stay in your lane right now, you glance in your rearview mirror
    to see where you were for the **last 5 seconds** (these are your 5 lags). This
    past information helps you make tiny corrections to your steering wheel *now*.
29 * **Steps:** You look ahead through the windshield and predict where your car
    will be for the **next 100 meters** (these are your 100 forecast steps).
30
31 ### Summary Table
32
33 | Feature | Lags ('p') | Steps ('forecast') |
34 | :--- | :--- | :--- |
35 | **Purpose** | Model Training (Input) | Making Predictions (Output) |

```

```

36 | **Time Frame** | Looks at the **Past** | Predicts the **Future** |
37 | **Role** | A structural hyperparameter of the model | An argument passed to the
    forecast function |
38 | **Example** | 'ARIMA(order=(**5**, 1, 0))' | 'model.forecast(steps=**7**)' |
39 | **Question it Answers** | "How much past data should I use to learn?" | "How far
    into the future should I predict?" |
40
41 So, when you build 'ARIMA(order=(5, 1, 0))', you're telling the model to learn
    patterns from the last 5 hours. When you call '.forecast(steps=442)', you're
    telling that trained model to apply what it learned to predict the next 442
    hours.

```

My Prompt to Copilot

```

1 gammaploid: for question 4
2 # Question 4: Perform a 7-step rolling forecast on a hold-out test set using the
    best ARIMA model from the previous question. Calculate the MAE, MSE, RMSE, and
    R for the rolling forecast.
3 should we use https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima.
    model.ARIMAResults.html? is it the appropriate way ?what do you suggest

```

Copilot's Response

```

1 GitHub Copilot: Great question! For Question 4, you need to implement a **7-step
    rolling forecast**, which is different from what you currently have. Let me
    explain the difference and show you the proper approach.
2 ## Current Issue with Your Code
3
4 Your current code does a **static forecast**:
5 ```python
6 # This is NOT a rolling forecast - it's a one-time forecast
7 predictions = arimamodel_fit.forecast(steps=len(test)) # Predicts all 442 steps
    at once
8 ```
9
10 ## What is a 7-Step Rolling Forecast?
11
12 A **rolling forecast** means:
13 1. Train model on initial training data
14 2. Predict the next **7 steps** (7 hours)
15 3. **Add the actual observed values** to your training data
16 4. **Retrain** the model with the updated data
17 5. Predict the next 7 steps again
18 6. Repeat until you've covered the entire test set
19
20 ## Recommended Approach
21
22 Yes, you should use the 'ARIMAResults' methods, but specifically the **append()
    ** method for efficient rolling forecasts. Here's the proper implementation:
23
24 ```python
25 # Question 4: 7-Step Rolling Forecast Implementation
26 print("=== 7-Step Rolling Forecast ===")
27
28 # Split data into training and test sets
29 train_size = int(len(wdf) * 0.8)
30 train_data = wdf['Temp'][:train_size]
31 test_data = wdf['Temp'][train_size:]

```

```

32
33 print(f"Training set size: {len(train_data)}")
34 print(f"Test set size: {len(test_data)}")
35
36 # Determine the best model from Question 3 (you mentioned ARIMA(5,1,0) was best)
37 best_order = (5, 1, 0) # Replace with your best performing model
38 forecast_horizon = 7 # 7-step rolling forecast
39
40 # Initialize lists to store results
41 rolling_predictions = []
42 actual_values = []
43
44 # Fit initial model
45 current_train = train_data.copy()
46 model = ARIMA(current_train, order=best_order)
47 fitted_model = model.fit()
48
49 print(f"Starting rolling forecast with {forecast_horizon}-step horizon...")
50
51 # Perform rolling forecast
52 for i in range(0, len(test_data), forecast_horizon):
53     # Determine how many steps to forecast (handle end of series)
54     steps_to_forecast = min(forecast_horizon, len(test_data) - i)
55
56     # Make forecast
57     forecast = fitted_model.forecast(steps=steps_to_forecast)
58
59     # Store predictions and actual values
60     rolling_predictions.extend(forecast)
61     actual_values.extend(test_data.iloc[i:i+steps_to_forecast])
62
63     # Get actual values for the forecasted period
64     actual_period = test_data.iloc[i:i+steps_to_forecast]
65
66     # Update model with new actual data (rolling window)
67     if i + steps_to_forecast < len(test_data):
68         # Use append method for efficient model updating
69         fitted_model = fitted_model.append(actual_period, refit=True)
70
71     print(f"Completed forecast for period {i+1}-{i+steps_to_forecast}")
72
73 # Convert to arrays for metric calculation
74 rolling_predictions = np.array(rolling_predictions)
75 actual_values = np.array(actual_values)
76
77 # Calculate metrics for rolling forecast
78 rolling_mae = mean_absolute_error(actual_values, rolling_predictions)
79 rolling_mse = mean_squared_error(actual_values, rolling_predictions)
80 rolling_rmse = np.sqrt(rolling_mse)
81 rolling_r2 = r2_score(actual_values, rolling_predictions)
82
83 print(f"\n== 7-Step Rolling Forecast Results ==")
84 print(f"MAE: {rolling_mae:.4f}")
85 print(f"MSE: {rolling_mse:.4f}")
86 print(f"RMSE: {rolling_rmse:.4f}")
87 print(f"R : {rolling_r2:.4f}")
88
89 # Compare with static forecast from previous questions
90 static_predictions = fitted_model.forecast(steps=len(test_data))
91 static_mae = mean_absolute_error(test_data, static_predictions)
92 static_rmse = np.sqrt(mean_squared_error(test_data, static_predictions))
93 static_r2 = r2_score(test_data, static_predictions)

```

```

94
95 print(f"\n=== Comparison: Rolling vs Static Forecast ===")
96 print(f"Rolling Forecast - MAE: {rolling_mae:.4f}, RMSE: {rolling_rmse:.4f}, R : {rolling_r2:.4f}")
97 print(f"Static Forecast - MAE: {static_mae:.4f}, RMSE: {static_rmse:.4f}, R : {static_r2:.4f}")
98
99 # Plot the results
100 plt.figure(figsize=(15, 8))
101
102 # Plot last 168 hours (1 week) for better visualization
103 plot_start = max(0, len(test_data) - 168)
104 plot_end = len(test_data)
105
106 plt.plot(test_data.index[plot_start:plot_end],
107          actual_values[plot_start:plot_end],
108          label='Actual Temperature', linewidth=2, color='blue')
109
110 plt.plot(test_data.index[plot_start:plot_end],
111          rolling_predictions[plot_start:plot_end],
112          label='7-Step Rolling Forecast', linewidth=2, color='red', alpha=0.8)
113
114 plt.plot(test_data.index[plot_start:plot_end],
115          static_predictions[plot_start:plot_end],
116          label='Static Forecast', linewidth=1, color='green', alpha=0.6)
117
118 plt.title('7-Step Rolling Forecast vs Static Forecast (Last Week)')
119 plt.xlabel('Date')
120 plt.ylabel('Temperature ( C )')
121 plt.legend()
122 plt.grid(True, alpha=0.3)
123 plt.xticks(rotation=45)
124 plt.tight_layout()
125 plt.show()
126
127 # Calculate improvement
128 mae_improvement = ((static_mae - rolling_mae) / static_mae) * 100
129 rmse_improvement = ((static_rmse - rolling_rmse) / static_rmse) * 100
130
131 print(f"\n=== Performance Improvement ===")
132 print(f"MAE improvement: {mae_improvement:.2f}%")
133 print(f"RMSE improvement: {rmse_improvement:.2f}%")
134
135 if rolling_mae < static_mae:
136     print("    Rolling forecast performs better than static forecast")
137 else:
138     print("    Static forecast performs better than rolling forecast")
139 '''
140
141 ## Key Methods from 'ARIMAResults':
142
143 1. **'append()'**: Efficiently updates the model with new data without full retraining
144 2. **'forecast()'**: Makes predictions for specified number of steps
145 3. **'extend()'**: Alternative to append for adding multiple observations
146
147 ## Why Rolling Forecast is Better:
148
149 1. **More Realistic**: Mimics real-world forecasting where you get new data periodically
150 2. **Adaptive**: Model updates as new information becomes available
151 3. **Better Performance**: Usually more accurate than static forecasts for longer

```

```

horizons
52 4. **Reduces Drift**: Prevents cumulative errors from building up
53
54 ## Expected Results:
55
56 Rolling forecasts typically perform **better** than static forecasts because:
57 - They incorporate new information as it becomes available
58 - They don't suffer from cumulative error drift
59 - They adapt to recent changes in the data pattern
60
61 Your rolling forecast should show improved MAE, RMSE, and R compared to the
    static forecast from previous questions.

```

My Prompt to Copilot

```
1 gammaploid: explain the code
```

Copilot's Response

```

1 GitHub Copilot: I'll explain the 7-step rolling forecast code step by step. This
    is a more sophisticated forecasting approach than the static forecast you used
    earlier.
2 ## Code Breakdown
3
4 ### 1. **Setup and Data Preparation**
5 '''python
6 print("=== 7-Step Rolling Forecast ===")
7
8 # Split data into training and test sets
9 train_size = int(len(wdf) * 0.8)
10 train_data = wdf['Temp'][:train_size]
11 test_data = wdf['Temp'][train_size:]
12 '''
13 **What it does**:
14 - Creates the same 80/20 train/test split as before
15 - 'train_data': First 80% for initial model training
16 - 'test_data': Last 20% to evaluate rolling forecast performance
17
18 ### 2. **Model Configuration**
19 '''python
20 best_order = (5, 1, 0) # Replace with your best performing model
21 forecast_horizon = 7 # 7-step rolling forecast
22 '''
23 **What it does**:
24 - Uses ARIMA(5,1,0) - your best model from Question 3
25 - Sets horizon to 7 steps (7 hours ahead each time)
26
27 ### 3. **Initialize Storage**
28 '''python
29 rolling_predictions = []
30 actual_values = []
31 '''
32 **What it does**: Creates empty lists to store all predictions and actual values
    as we go through the rolling process.
33
34 ### 4. **Initial Model Fitting**
35 '''python
36 current_train = train_data.copy()

```

```

37 model = ARIMA(current_train, order=best_order)
38 fitted_model = model.fit()
39 '''
40 **What it does**: Fits the initial ARIMA model on the training data. This is your
    starting point.
41
42 ### 5. **The Rolling Forecast Loop** (The Heart of the Algorithm)
43 '''python
44 for i in range(0, len(test_data), forecast_horizon):
45     # Determine how many steps to forecast (handle end of series)
46     steps_to_forecast = min(forecast_horizon, len(test_data) - i)
47
48     # Make forecast
49     forecast = fitted_model.forecast(steps=steps_to_forecast)
50
51     # Store predictions and actual values
52     rolling_predictions.extend(forecast)
53     actual_values.extend(test_data.iloc[i:i+steps_to_forecast])
54
55     # Get actual values for the forecasted period
56     actual_period = test_data.iloc[i:i+steps_to_forecast]
57
58     # Update model with new actual data (rolling window)
59     if i + steps_to_forecast < len(test_data):
60         fitted_model = fitted_model.append(actual_period, refit=True)
61
62     print(f"Completed forecast for period {i+1}-{i+steps_to_forecast}")
63 '''
64
65 Let me break this loop down iteration by iteration:
66
67 **Iteration 1** (i=0):
68 - 'steps_to_forecast = min(7, 442-0) = 7'
69 - Model predicts hours 1-7 of test set
70 - Stores these 7 predictions and actual values
71 - **Key part**: 'fitted_model.append(actual_period, refit=True)'
72   - Takes the actual observed temperatures for hours 1-7
73   - Adds them to the model's training data
74   - Refits the model with this new information
75
76 **Iteration 2** (i=7):
77 - Now the model has been updated with actual data from hours 1-7
78 - Predicts hours 8-14 using the updated model
79 - Again stores predictions and updates model with actual hours 8-14
80
81 **And so on...**
82
83 ### 6. **Why 'refit=True' is Important**
84 '''python
85 fitted_model = fitted_model.append(actual_period, refit=True)
86 '''
87 **What it does**:
88 - 'append()': Adds new observations to the model's dataset
89 - 'refit=True': Re-estimates all model parameters with the expanded dataset
90 - This makes the model "learn" from recent observations
91
92 ### 7. **Performance Evaluation**
93 '''python
94 rolling_mae = mean_absolute_error(actual_values, rolling_predictions)
95 rolling_mse = mean_squared_error(actual_values, rolling_predictions)
96 rolling_rmse = np.sqrt(rolling_mse)
97 rolling_r2 = r2_score(actual_values, rolling_predictions)

```

```

98 '''
99 **What it does**: Calculates metrics comparing all rolling predictions vs actual
    values.
100
101 ### 8. **Comparison with Static Forecast**
102 '''python
103 static_predictions = fitted_model.forecast(steps=len(test_data))
104 '''
105 **What it does**: Makes a one-time forecast for the entire test period (like your
    previous questions) to compare performance.
106
107 ## Key Differences: Rolling vs Static
108
109 | Aspect | Static Forecast | Rolling Forecast |
110 | :--- | :--- | :--- |
111 | **Training Data** | Fixed (initial training set only) | Grows with each
    iteration |
112 | **Model Updates** | Never updated | Updated every 7 hours |
113 | **Predictions** | All 442 hours predicted at once | 7 hours at a time, 63 times
    |
114 | **Information Used** | Only knows past up to training end | Learns from recent
    actual temperatures |
115 | **Realism** | Unrealistic (no new info for 442 hours) | Realistic (gets updates
    every 7 hours) |
116
117 ## Why Rolling is Better
118
119 1. **Adaptive Learning**: Model incorporates new temperature patterns as they
    occur
120 2. **Reduced Error Accumulation**: Errors don't compound over 442 hours
121 3. **Real-world Simulation**: Mimics how forecasting actually works in practice
122 4. **Better Accuracy**: Usually performs significantly better than static
    forecasts
123
124 The rolling forecast essentially says: "Every 7 hours, I'll use the latest
    temperature data to improve my model and predict the next 7 hours" - which is
    much more realistic than "I'll predict the next 442 hours right now and never
    update."

```

My Prompt to Copilot

```

1 gammaploid: so you have not used arimaresults ? why?

```

Copilot's Response

```

1 GitHub Copilot: You're absolutely right to ask! I **did** use 'ARIMAResults'
    methods, but perhaps not as explicitly as I could have. Let me clarify:
2 ## ARIMAResults Methods I Used
3
4 1. **fitted_model.forecast(** - This is an 'ARIMAResults' method
5 2. **fitted_model.append(** - This is also an 'ARIMAResults' method
6
7 ## Where ARIMAResults is Used in the Code
8
9 '''python
10 # This creates an ARIMAResults object
11 fitted_model = model.fit() # fitted_model is an ARIMAResults instance
12

```



```

13 # These are ARIMAResults methods:
14 forecast = fitted_model.forecast(steps=steps_to_forecast) # ARIMAResults.forecast
15     ()
16 fitted_model = fitted_model.append(actual_period, refit=True) # ARIMAResults.
17     append()
18 '''
19
20 ## Why I Used This Approach
21
22 The 'append()' method from 'ARIMAResults' is exactly what the documentation
23     recommends for rolling forecasts. Here's why:
24
25
26 ### Alternative 1: Using 'extend()' method
27 '''python
28 # Could also use extend() method
29 fitted_model = fitted_model.extend(actual_period, refit=True)
30 '''
31
32 ### Alternative 2: Manual re-fitting (less efficient)
33 '''python
34 # Less efficient approach - completely refit each time
35 new_train_data = pd.concat([current_train, actual_period])
36 model = ARIMA(new_train_data, order=best_order)
37 fitted_model = model.fit()
38 '''
39
40 ## The Most Explicit ARIMAResults Approach
41
42 If you want to be more explicit about using 'ARIMAResults' methods, here's an
43     enhanced version:
44
45
46 '''python
47 # More explicit use of ARIMAResults methods
48 fitted_model = model.fit()
49
50 for i in range(0, len(test_data), forecast_horizon):
51     steps_to_forecast = min(forecast_horizon, len(test_data) - i)
52
53     # ARIMAResults.forecast() method
54     forecast = fitted_model.forecast(steps=steps_to_forecast)
55
56     # ARIMAResults.get_prediction() alternative
57     # prediction_results = fitted_model.get_prediction(
58     #     start=len(fitted_model.fittedvalues),
59     #     end=len(fitted_model.fittedvalues) + steps_to_forecast - 1
60     # )
61
62     rolling_predictions.extend(forecast)
63     actual_values.extend(test_data.iloc[i:i+steps_to_forecast])
64
65     actual_period = test_data.iloc[i:i+steps_to_forecast]
66
67     if i + steps_to_forecast < len(test_data):
68         # ARIMAResults.append() method - this is the key ARIMAResults method
69         fitted_model = fitted_model.append(actual_period, refit=True)
70
71         # Could also use ARIMAResults.extend()
72         # fitted_model = fitted_model.extend(actual_period, refit=True)
73 '''
74
75 ## Key ARIMAResults Methods Available
76

```

```

71 1. forecast(steps) - Generate forecasts
72 2. append(endog, refit=True) - Add new observations and optionally refit
73 3. extend(endog, refit=True) - Similar to append
74 4. get_prediction(start, end) - Get predictions with confidence intervals
75 5. summary() - Model summary
76 6. fittedvalues - In-sample fitted values
77 7. resid - Residuals
78
79 ## Why 'append()' is Perfect for Rolling Forecasts
80
81 The 'append()' method is specifically designed for this use case:
82 - Efficient: Doesn't re-estimate all parameters from scratch
83 - Maintains state: Preserves model structure and learned parameters
84 - Built-in: Designed exactly for rolling/online forecasting scenarios
85
86 So yes, I am using 'ARIMAResults' methods - specifically the ones most
    appropriate for rolling forecasts! The 'append()' method is the recommended
    approach in the statsmodels documentation for exactly this scenario.
87
88 Would you prefer to see a version that uses additional 'ARIMAResults' methods like
    'get_prediction()' with confidence intervals?

```

My Prompt to Copilot

```

1 gammaploid: so for this question: # Question 5: Update the model to include an
    exogenous variable (e.g., hour of the day). Fit the model and compare its
    performance with the best Adelaide daily temperature ARIMA model without the
    exogenous variable.
2 method model.exgo shows none, how do we introduce hour of the day as exogenous
    variable?

```

Copilot's Response

```

1 GitHub Copilot: Great question! You're right that 'arimamodel.exog_names' shows '
    None' because you haven't added any exogenous variables yet. Let me show you
    how to add "hour of the day" as an exogenous variable to your ARIMA model.
2 ## What are Exogenous Variables?
3
4 Exogenous variables are external factors that can help predict your target
    variable (temperature) but are not predicted by the model itself. Hour of the
    day is perfect because:
5 - Temperature has a strong daily cycle (hot afternoons, cool nights)
6 - Hour is known in advance (we always know what time it will be)
7 - It's external to the temperature time series itself
8
9 ## Step-by-Step Implementation
10
11 python
12 # Question 5: ARIMAX Model with Hour of Day as Exogenous Variable
13 print("=== ARIMAX MODEL WITH HOUR OF DAY ===")
14
15 # Reload data to ensure clean state
16 response = requests.get(endpoint, params=params)
17 wdata = response.json()
18 timestamps = [datetime.fromisoformat(item) for item in wdata['hourly']['time']]
19 temperatures = wdata['hourly']['temperature_2m']
20
21 # Create DataFrame with additional features

```

```

22 wdf_exog = pd.DataFrame({
23     'Date': timestamps,
24     'Temp': temperatures
25 })
26 wdf_exog.set_index('Date', inplace=True)
27
28 # Create exogenous variables
29 print("Creating exogenous variables...")
30
31 # Method 1: Simple hour of day (0-23)
32 wdf_exog['Hour'] = wdf_exog.index.hour
33
34 # Method 2: Cyclical encoding (better for capturing cyclical nature)
35 wdf_exog['Hour_sin'] = np.sin(2 * np.pi * wdf_exog.index.hour / 24)
36 wdf_exog['Hour_cos'] = np.cos(2 * np.pi * wdf_exog.index.hour / 24)
37
38 # Method 3: Time of day categories
39 def time_of_day(hour):
40     if 6 <= hour < 12:
41         return 1 # Morning
42     elif 12 <= hour < 18:
43         return 2 # Afternoon
44     elif 18 <= hour < 22:
45         return 3 # Evening
46     else:
47         return 0 # Night
48
49 wdf_exog['Time_of_Day'] = wdf_exog.index.hour.map(time_of_day)
50
51 print(f"Dataset shape: {wdf_exog.shape}")
52 print("\nFirst few rows with exogenous variables:")
53 print(wdf_exog.head(10))
54
55 # Split data
56 train_size = int(len(wdf_exog) * 0.8)
57 train_temp = wdf_exog['Temp'][:train_size]
58 test_temp = wdf_exog['Temp'][train_size:]
59
60 # Exogenous variables for training and testing
61 # We'll use cyclical encoding (Hour_sin, Hour_cos) as it's most effective
62 exog_train = wdf_exog[['Hour_sin', 'Hour_cos']][:train_size]
63 exog_test = wdf_exog[['Hour_sin', 'Hour_cos']][train_size:]
64
65 print(f"\nTraining set: {len(train_temp)} observations")
66 print(f"Test set: {len(test_temp)} observations")
67 print(f"Exogenous variables: {exog_train.columns.tolist()}")
68
69 #
70 # =====
71 # MODEL 1: BASELINE ARIMA WITHOUT EXOGENOUS VARIABLES
72 # =====
73
74 print("\n" + "="*60)
75 print("1. BASELINE ARIMA MODEL (NO EXOGENOUS VARIABLES)")
76 print("="*60)
77
78 # Use your best model from previous questions
79 best_order = (5, 1, 0) # Adjust based on your best model from Q3

```

```

80 baseline_model = ARIMA(train_temp, order=best_order)
81 baseline_fitted = baseline_model.fit()
82
83 # Predictions
84 baseline_predictions = baseline_fitted.forecast(steps=len(test_temp))
85
86 # Metrics
87 baseline_mae = mean_absolute_error(test_temp, baseline_predictions)
88 baseline_mse = mean_squared_error(test_temp, baseline_predictions)
89 baseline_rmse = np.sqrt(baseline_mse)
90 baseline_r2 = r2_score(test_temp, baseline_predictions)
91
92 print(f"Baseline ARIMA{best_order} Results:")
93 print(f"MAE: {baseline_mae:.4f}")
94 print(f"MSE: {baseline_mse:.4f}")
95 print(f"RMSE: {baseline_rmse:.4f}")
96 print(f"R : {baseline_r2:.4f}")
97
98 #
=====
99 # MODEL 2: ARIMAX WITH EXOGENOUS VARIABLES
100 #
=====
101
102 print("\n" + "="*60)
103 print("2. ARIMAX MODEL (WITH HOUR OF DAY)")
104 print("="*60)
105
106 # Fit ARIMAX model (ARIMA with exogenous variables)
107 arimax_model = ARIMA(train_temp, exog=exog_train, order=best_order)
108 arimax_fitted = arimax_model.fit()
109
110 print("ARIMAX Model Summary:")
111 print(arimax_fitted.summary())
112
113 # Make predictions (IMPORTANT: must provide exogenous variables for forecast
    period)
114 arimax_predictions = arimax_fitted.forecast(steps=len(test_temp), exog=exog_test)
115
116 # Metrics
117 arimax_mae = mean_absolute_error(test_temp, arimax_predictions)
118 arimax_mse = mean_squared_error(test_temp, arimax_predictions)
119 arimax_rmse = np.sqrt(arimax_mse)
120 arimax_r2 = r2_score(test_temp, arimax_predictions)
121
122 print(f"\nARIMAX Results:")
123 print(f"MAE: {arimax_mae:.4f}")
124 print(f"MSE: {arimax_mse:.4f}")
125 print(f"RMSE: {arimax_rmse:.4f}")
126 print(f"R : {arimax_r2:.4f}")
127
128 # Check exogenous variable names
129 print(f"\nExogenous variables in model: {arimax_model.exog_names}")
130 '''
131
132 ## Key Points About Exogenous Variables:
133
134 ### 1. **How to Add Them**
135 '''python
136 # When creating the model, add exog parameter:

```

```

37 ARIMA(endog_data, exog=exog_data, order=(p,d,q))
38 '''
39
40 ### 2. **Must Provide for Forecasting**
41 '''python
42 # CRITICAL: Must provide exog data for forecast period
43 model.forecast(steps=N, exog=future_exog_data)
44 '''
45
46 ### 3. **Types of Hour Encoding**
47
48 **Simple Hour (0-23):**
49 '''python
50 wdf_exog['Hour'] = wdf_exog.index.hour
51 '''
52
53 **Cyclical Encoding (Recommended):**
54 '''python
55 wdf_exog['Hour_sin'] = np.sin(2 * np.pi * wdf_exog.index.hour / 24)
56 wdf_exog['Hour_cos'] = np.cos(2 * np.pi * wdf_exog.index.hour / 24)
57 '''
58
59 **Why cyclical?** Because hour 23 and hour 0 are actually close in time, but
    numerically they're far apart (23 vs 0). Sine/cosine encoding captures this
    cyclical relationship.
60
61 ### 4. **Additional Exogenous Variables You Could Add**
62 '''python
63 # Day of week
64 wdf_exog['DayOfWeek'] = wdf_exog.index.dayofweek
65
66 # Month
67 wdf_exog['Month'] = wdf_exog.index.month
68
69 # Weekend indicator
70 wdf_exog['Is_Weekend'] = (wdf_exog.index.dayofweek >= 5).astype(int)
71 '''
72
73 ## Expected Results
74
75 The ARIMAX model should perform significantly better than the baseline ARIMA
    because:
76 - It can capture the daily temperature cycle explicitly
77 - Hour information helps predict whether it's a typically warm or cool time of day
78 - Should improve R from negative values to positive values
79
80 Try this code and you should see 'arimax_model.exog_names' showing ['Hour_sin', '
    Hour_cos'] instead of 'None'!
```

My Prompt to Copilot

```

1 gammaploid: format this bit:
2 - p=5: uses 5 autoregressive lags (Temp yesterday, 2 days ago      5 days ago).
3
4 - d=1: series was differenced once to remove trend (i.e., predicting changes, not
    absolute values).
5
6 - q=0: no moving average terms.
7
8 - Lags are about the PAST (used for training the model):: The number of previous
```

```

time periods the model looks back at to make a prediction. It's a core part of
the model's structure.
9
10 - Steps are about the FUTURE (used for making predictions): The number of future
time periods you want the model to predict.
11
12
13 AR Coeff.:
14 - ar.L1 = 0.6723 Highly significant (p < 0.001). Strong positive dependence on
yesterdays value (differenced Temp). If yesterdays change was positive,
todays is likely positive too.
15 - ar.L2 = 0.0048 Not significant (p = 0.855). Effect of 2-day lag is essentially
zero.
16
17 Ljung-Box (Q):
18 Tests if the model's errors are correlated with each other. Prob(Q) = 0.73 is much
greater than 0.05 means your model has successfully captured the
autocorrelation in the data
19
20 Jarque-Bera (JB):
21 Tests if the model's errors are normally distributed (a bell curve shape).
22 Prob(JB) = 0.00 is less than 0.05. This is not good. It means the errors are not
normally distributed. The Skew (0.43) and Kurtosis (5.39, which is much higher
than the normal 3) confirm this. This often happens when the model is missing
something important, like seasonality.
23
24 Heteroskedasticity (H): Tests if the variance of the errors is constant over time.
25
26 Prob(H) = 0.04 is less than 0.05. This is also not good. It means the size of the
model's errors changes over time
27
28 \textbf{Performance metrics:}
29 MAE has about 1C difference than to the average not terrible, R^2 tells us is its
worse than random guessing, and RMSE tells its 2.5C above average

```

Copilot's Response

```

1 GitHub Copilot: Of course. Here is that text formatted for your LaTeX document. I'
ve structured it with subsections and lists to make it clear and readable.
2 '''latex
3 \subsection*{Interpretation of the ARIMA(5, 1, 0) Model}
4
5 \subsubsection*{Model Parameters}
6 \begin{description}
7   \item[\texttt{p=5}:] The model uses 5 autoregressive lags. This means it looks
at the temperature from the past 5 hours to inform its prediction.
8   \item[\texttt{d=1}:] The series was differenced once. This is done to remove
long-term trends, making the model predict the \textit{change} in temperature
from one hour to the next, rather than the absolute value.
9   \item[\texttt{q=0}:] The model uses no moving average terms.
10 \end{description}
11
12 \subsubsection*{Key Concepts: Lags vs. Steps}
13 \begin{description}
14   \item[\textbf{Lags (The Past):}] Refers to the number of previous time steps
the model uses as input during training. It is a core part of the model's
structure.
15   \item[\textbf{Steps (The Future):}] Refers to the number of future time
periods the model is asked to predict during the forecasting phase.
16 \end{description}

```

```

17
18 \subsubsection*{Coefficient Analysis}
19 \begin{description}
20   \item[\texttt{ar.L1 = 0.6723}:] \textbf{Highly significant} ( $p < 0.001$ ).
    This shows a strong positive dependence on the previous hour's temperature
    change. If the temperature change was positive in the last hour, it's likely
    to be positive in the current hour too.
21   \item[\texttt{ar.L2 = 0.0048}:] \textbf{Not significant} ( $p = 0.855$ ). The
    effect of the temperature from 2 hours ago is statistically zero in this model
    .
22 \end{description}
23
24 \subsubsection*{Diagnostic Tests}
25 \begin{description}
26   \item[Ljung-Box (Q):] Tests if the model's errors are correlated. A \texttt{Prob(Q)}
    of 0.73 (which is  $> 0.05$ ) is \textbf{good}. It suggests the model has
    successfully captured the autocorrelation in the data, and the remaining
    errors are random.
27
28   \item[Jarque-Bera (JB):] Tests if the model's errors are normally distributed.
    A \texttt{Prob(JB)} of 0.00 (which is  $< 0.05$ ) is \textbf{not good}. It
    indicates the errors are not normally distributed, which is confirmed by the
    high \texttt{Kurtosis} (5.39). This often happens when the model is missing a
    key feature, such as seasonality.
29
30   \item[Heteroskedasticity (H):] Tests if the variance of the errors is constant
    over time. A \texttt{Prob(H)} of 0.04 (which is  $< 0.05$ ) is also \textbf{not
    good}. It means the size of the model's errors changes over time (e.g., it
    might be more accurate at certain times of day than others).
31 \end{description}
32
33 \subsubsection*{Performance Metrics Summary}
34 The Mean Absolute Error (MAE) indicates the model's predictions are, on average,
    off by about 1 C , which is not terrible. However, the negative R value
    reveals that the model's predictions are worse than simply guessing the
    average temperature. The Root Mean Squared Error (RMSE) of 2.5 C further
    highlights the magnitude of the prediction errors.\subsubsection*{Interpretation
    of the ARIMA(5, 1, 0) Model}
35
36 \subsubsection*{Model Parameters}
37 \begin{description}
38   \item[\texttt{p=5}:] The model uses 5 autoregressive lags. This means it looks
    at the temperature from the past 5 hours to inform its prediction.
39   \item[\texttt{d=1}:] The series was differenced once. This is done to remove
    long-term trends, making the model predict the \textit{change} in temperature
    from one hour to the next, rather than the absolute value.
40   \item[\texttt{q=0}:] The model uses no moving average terms.
41 \end{description}
42
43 \subsubsection*{Key Concepts: Lags vs. Steps}
44 \begin{description}
45   \item[\textbf{Lags (The Past):}] Refers to the number of previous time steps
    the model uses as input during training. It is a core part of the model's
    structure.
46   \item[\textbf{Steps (The Future):}] Refers to the number of future time
    periods the model is asked to predict during the forecasting phase.
47 \end{description}
48
49 \subsubsection*{Coefficient Analysis}
50 \begin{description}
51   \item[\texttt{ar.L1 = 0.6723}:] \textbf{Highly significant} ( $p < 0.001$ ).
    This shows a strong positive dependence on the previous hour's temperature

```

```

change. If the temperature change was positive in the last hour, it's likely
to be positive in the current hour too.
52 \item[\texttt{ar.L2 = 0.0048}:] \textbf{Not significant} ($p = 0.855$). The
    effect of the temperature from 2 hours ago is statistically zero in this model
    .
53 \end{description}
54
55 \subsubsection*{Diagnostic Tests}
56 \begin{description}
57 \item[Ljung-Box (Q):] Tests if the model's errors are correlated. A \texttt{
    Prob(Q)} of 0.73 (which is > 0.05) is \textbf{good}. It suggests the model has
    successfully captured the autocorrelation in the data, and the remaining
    errors are random.
58
59 \item[Jarque-Bera (JB):] Tests if the model's errors are normally distributed.
    A \texttt{Prob(JB)} of 0.00 (which is < 0.05) is \textbf{not good}. It
    indicates the errors are not normally distributed, which is confirmed by the
    high \texttt{Kurtosis} (5.39). This often happens when the model is missing a
    key feature, such as seasonality.
60
61 \item[Heteroskedasticity (H):] Tests if the variance of the errors is constant
    over time. A \texttt{Prob(H)} of 0.04 (which is < 0.05) is also \textbf{not
    good}. It means the size of the model's errors changes over time (e.g., it
    might be more accurate at certain times of day than others).
62 \end{description}
63
64 \subsubsection*{Performance Metrics Summary}
65 The Mean Absolute Error (MAE) indicates the model's predictions are, on average,
    off by about 1 C , which is not terrible. However, the negative R value
    reveals that the model's predictions are worse than simply guessing the
    average temperature. The Root Mean Squared Error (RMSE) of 2.5 C further
    highlights the magnitude of the prediction errors.

```