Mechanical Engineering Department
ME7194D Project (Part 1)

Demand Forecasting and Inventory Management of Medicines in a Healthcare Centre

UNDER THE GUIDENCE OF PROFESOR R. SRIDHARAN

NAME – SOURABH GHAGHRE
ROLL NO.- M220550ME

Outlines

- Self Evaluation
- Data Collection Information
- Information Regarding Dataset
- Data Preprocessing
- Model Building
  - Theoretical Concept of Each Model
  - Mathematical Concept Behind The Model
  - Pseudo Code For Each Model
  - Python Code For Each Model
  - Output of Each Model
- Conclusion
- Future Plans
- Recent Literature Review

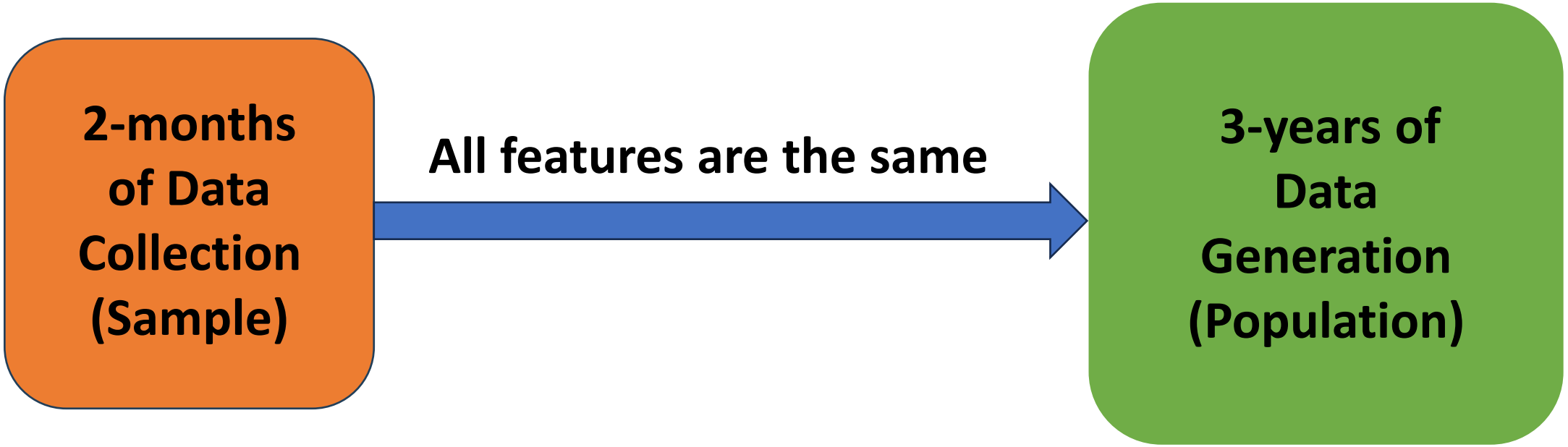
Self Evaluation

What did I do in last one month?

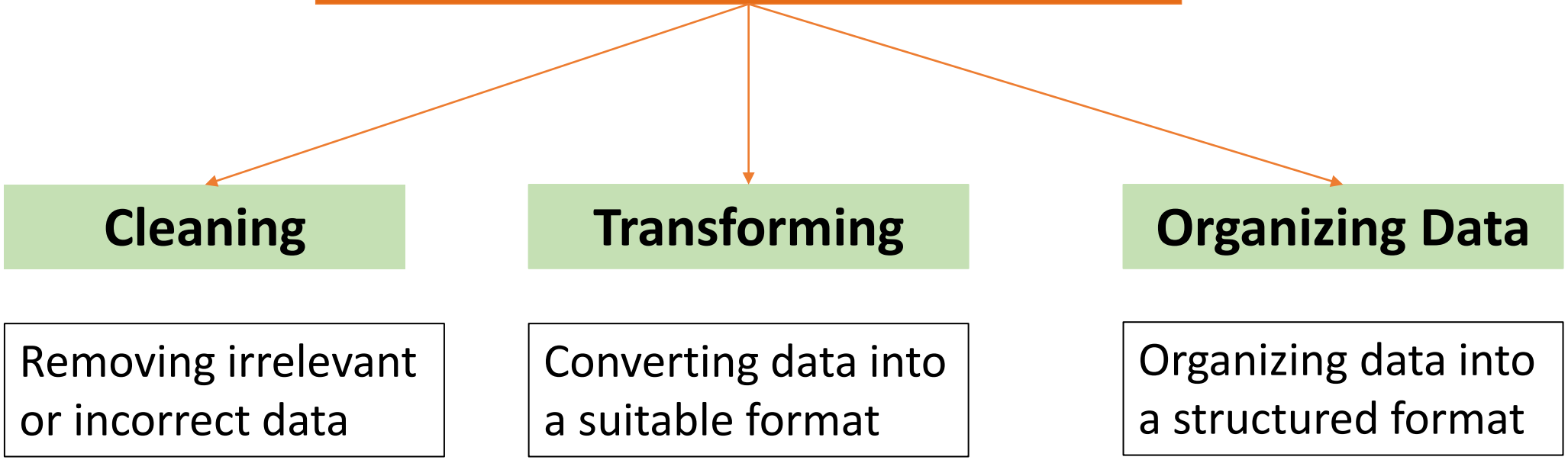
1. Collected data from NITC healthcare centre (collected 2 months of data).
2. Have done some literature review
3. Developed machine learning models using Python on Jupyter Notebook.
4. Analyzed the results obtained.

Data Collection Information

Symptoms/dignosised	DATE	SEX	AGE	MP1	QUANTITY	MP2	QUANTITY MP2	MP3	QUANTITY MP3	MP4
LEG PAIN	10-08-2022	F	20-01-1900	ZERODL	3	PANTOP DSR	3			
WOUND DRESSING	05-08-2022	M	20-01-1900	ZERODOL	3	PANTOP DSR	3			
VIRAL FEVER	11-08-2022	M	19-01-1900	ASCORYL SYRUP	1	DOLO 600	9	LEVOCET	3	
FEVER/COUGH	10-08-2022	F	19-01-1900	DOLO 650	9	MOX 500	3	ASCORYL SYRUP	1	
FEVER	11-08-2022									
FEVER/LOOSE MOTION	23-08-2022	M	21-01-1900	CEFIXIME 200	6	DOLO 600	9	ENUFF 100	6	ORS
LEG PAIN	19-08-2022	F	21-01-1900	ZERODOL	2	PANTOP DSR	2			
ACUTE GASTIRIST/LOOSE STOOL	17-08-2022	M	20-01-1900	PANTOP DSR	3	ORS	2			
URI	17-08-2022	M	19-01-1900	MOX 500	6	CPM	6			
COUGH	12-08-2022	F	19-01-1900	TUSQ	1					
SKIN INFECTION	02-08-2022	M	20-01-1900	CANDID CREAM	1	LEVOCET	3			
URI	16-08-2022	M	19-01-1900	ASCORYL SYRUP	1	DOLO 600	6	MOX 500	6	
URI	23-08-2022	M	20-01-1900	PARACETAMOL 500	6	MONTEE LC	2	SYRUP ASCORYL	1	MOXCLAV
VIRAL FEVER	01-08-2022	M		DOLO 650	6	LEVOCET	2			
COUGH	03-08-2022	M		ASCORYL SYRUP	1	LEVOCET	2			
SHOULDER PAIN	15-08-2022	M	20-01-1900	ZERODOL	6	OME Z	3	B COMPLEX	5	
FEVER/COLD	05-08-2022	M	19-01-1900	PARACETAMOL 500	6	LEVOCET	2			
URI	07-08-2022									
WOUND DRESSING	03-08-2022	F		BETADIN	1	PANTOP DSR	3			
URI	10-08-2022	M	19-01-1900	ASCORYL SYRUP	1	MOX CLAVE	3	LEVOCET	2	PERACETOMOL
R/W	12-08-2022									
VIRAL FEVER	03-08-2022	M		PARACETAMOL 500	6	MOTEE LC	2			
SINUS	01-08-2022	M	19-01-1900	AZEE 500	3					
ABDOMINAL PAIN	06-08-2022	M	20-01-1900	CIPLOX TZ	3	OME	3			
VIRAL FEVER	19-08-2022	M	20-01-1900	DOLO 650	9	MOX 500	9	MONTEE LC	3	
URI	17-08-2022	F	20-01-1900	AZEE 500	2	PARACETOMOL 500	6			
FOOT PAIN	16-08-2022	F	20-01-1900	MOX CLAV	6					
REFER KMCT	19-08-2022	M	20-01-1900	NA						


2-months of Data Collection (Sample)
All features are the same
3-years of Data Generation (Population)

DATA PREPROCESSING



Data Preprocessing By Python Codes

```
In [310]: df.shape
```

```
Out[310]: (42520, 14)
```

```
In [311]: df.nunique()
```

```
Out[311]: ROLL NO.          42520
          DATE               1127
          GENDER                2
          AGE                  10
          DIGNOSIS             10
          MEDICINE_1            9
          QUANTITY_1            4
          MEDICINE_2            7
          QUANTITY_2            6
          MEDICINE_3            4
          QUANTITY_3            4
          MEDICINE_ 4           3
          QUANTITY_ 4           2
          TOTAL_QUANTITY       10
          dtype: int64
```

```
In [312]: df = df.drop(['ROLL NO.'], axis = 1)
```

```
In [313]: df.columns = [col.lower().replace(' ', '') for col in df.columns]
```

```
In [314]: df.columns
```

```
Out[314]: Index(['date', 'gender', 'age', 'dignosis', 'medicine_1', 'quantity_1',
                 'medicine_2', 'quantity_2', 'medicine_3', 'quantity_3', 'medicine_4',
                 'quantity_4', 'total_quantity'],
                dtype='object')
```

Contd..

Each Features Specific Information

```
In [315]: df.gender.value_counts()
```

```
Out[315]: F    21289
          M    21231
          Name: gender, dtype: int64
```

```
In [316]: df.date.min(), df.date.max()
```

```
Out[316]: (Timestamp('2021-08-01 00:00:00'), Timestamp('2024-08-31 00:00:00'))
```

```
In [317]: df.dignosis.unique()
```

```
Out[317]: array(['COLD', 'URTI', 'SKIN_INFECTION', 'ABDOMINAL_PAIN', 'LEG_PAIN',
                 'ROUTINE_CHEKUP', 'LOOSE_MOTION', 'ACIDITY', 'VIRAL FEVER',
                 'COUGH'], dtype=object)
```

```
In [318]: df.medicine_1.unique()
```

```
Out[318]: array(['Mox 500', 'Dolo 650', 'CANDID_CREAM', 'CIFIXIM', 'Zerodol', nan,
                 'ENUFF', 'Pantop_40', 'Paracetamol', 'Ascoryl_Syrup'], dtype=object)
```

```
In [319]: df.medicine_2.unique()
```

```
Out[319]: array(['PARACETOMOL', 'ASCORYL_SYRUP', 'LEVOCET', 'PANTOP_40', nan, 'ORS',
                 'AZEE', 'DOLO_650'], dtype=object)
```

Groupby -It is used for grouping data based on some criterion, typically a column or a set of columns in a DataFrame.

CODE

```
import pandas as pd

# Example DataFrame
data = {'Category': ['A', 'B', 'A', 'B', 'C'],
        'Value': [10, 20, 30, 40, 50]}
df = pd.DataFrame(data)

# Group by the 'Category' column
grouped_df = df.groupby('Category')

# Calculate the mean of each group
mean_values = grouped_df.mean()

# Print the result
print(mean_values)
```

EXAMPLE

```
Original DataFrame:
  Category  Value
0        A     10
1        B     20
2        A     30
3        B     40
4        C     50

Grouped and Summed DataFrame:
Category     Value
A                 40
B                 60
C                 50
```

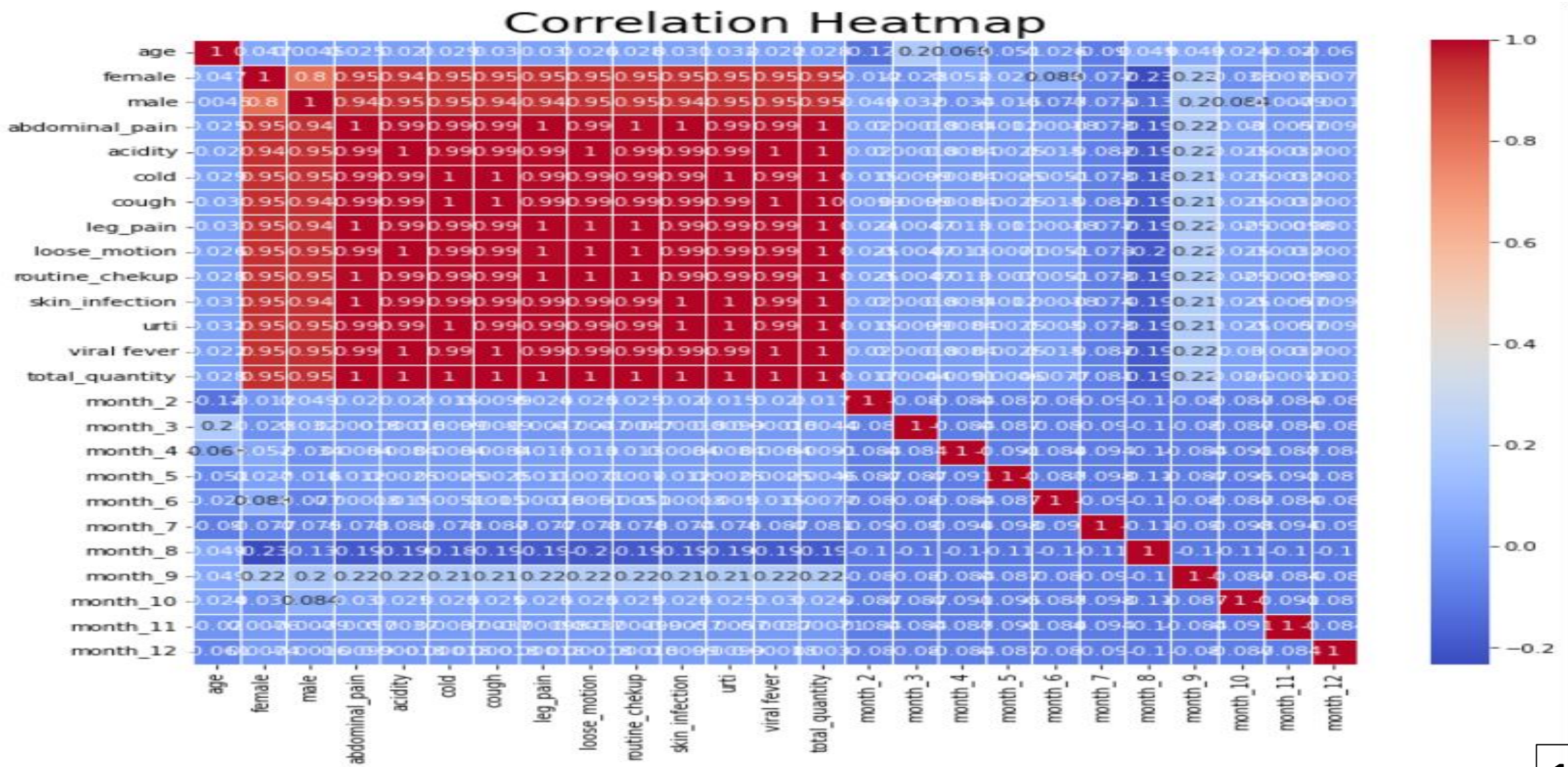
weekly observation in the below table

```
In [515]: df_clean
```

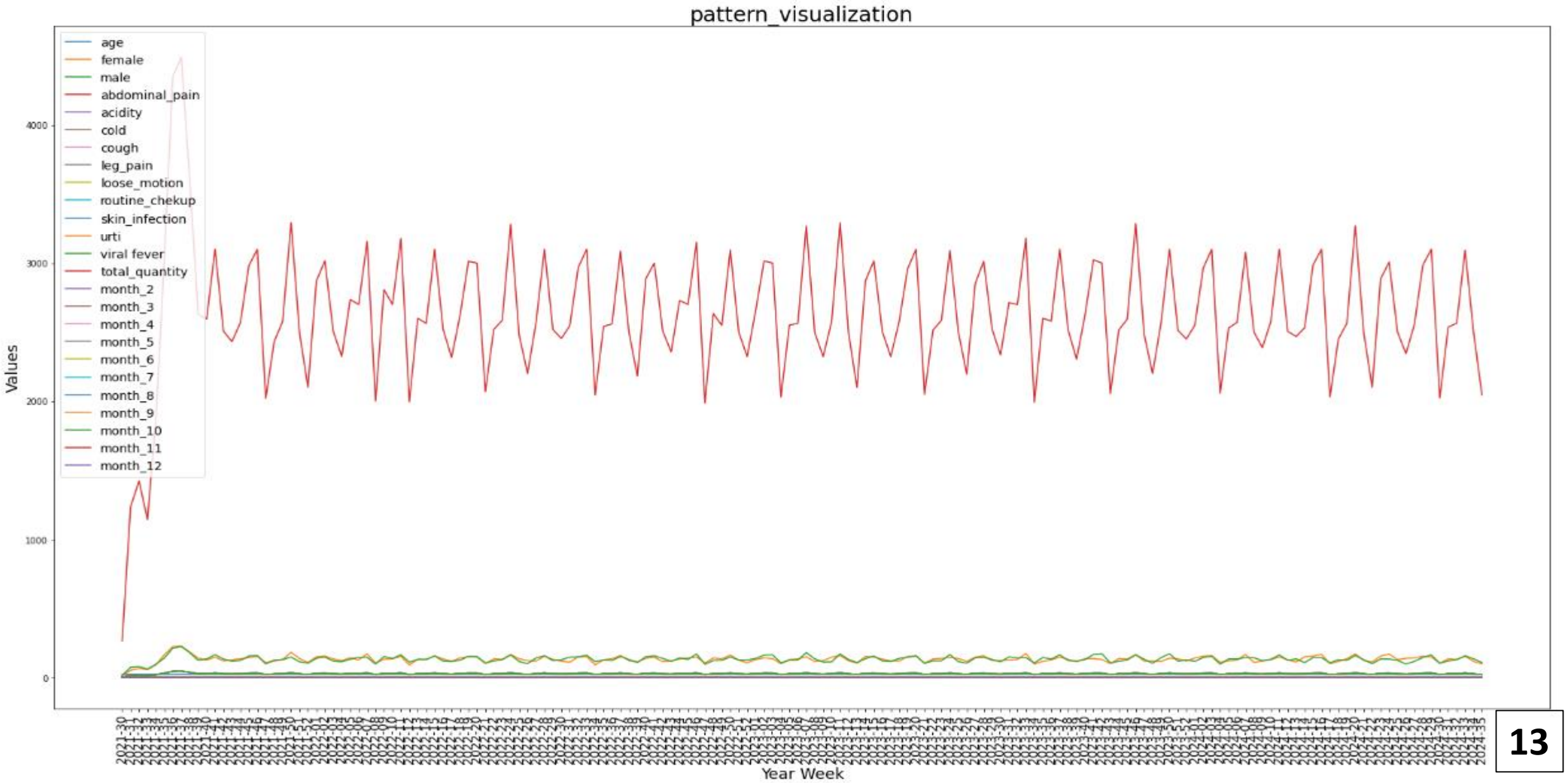
Out[515]:

	year_week	age	female	male	abdominal_pain	acidity	cold	cough	leg_pain	loose_motion
0	2021-30	22.370370	13	14	3	2	3	2	3	3
1	2021-31	22.219512	51	72	12	13	12	13	12	12
2	2021-32	22.297872	64	77	14	14	15	14	14	14
3	2021-33	22.594828	56	60	12	11	11	11	12	12
4	2021-34	22.451613	91	95	18	19	19	19	18	18
...	...	...	...	...	...	...	...	...	...	...
157	2024-31	22.415020	134	119	25	26	25	26	25	25
158	2024-32	22.700389	129	128	26	25	26	25	26	26
159	2024-33	22.472492	153	156	31	31	31	31	31	30
160	2024-34	22.200000	116	134	25	25	25	25	25	25
161	2024-35	22.808824	97	107	20	21	20	21	20	21

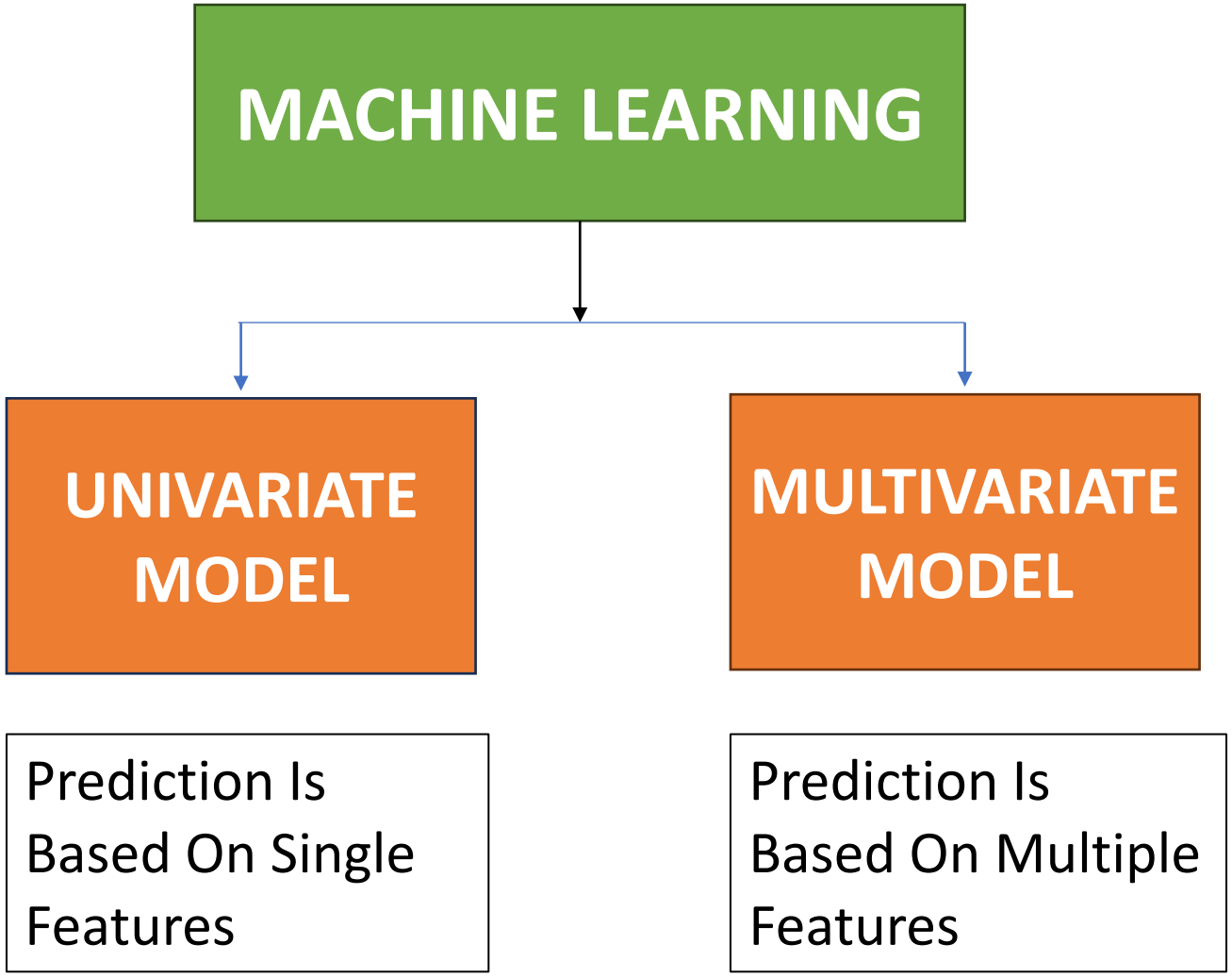
Correlation Matrix

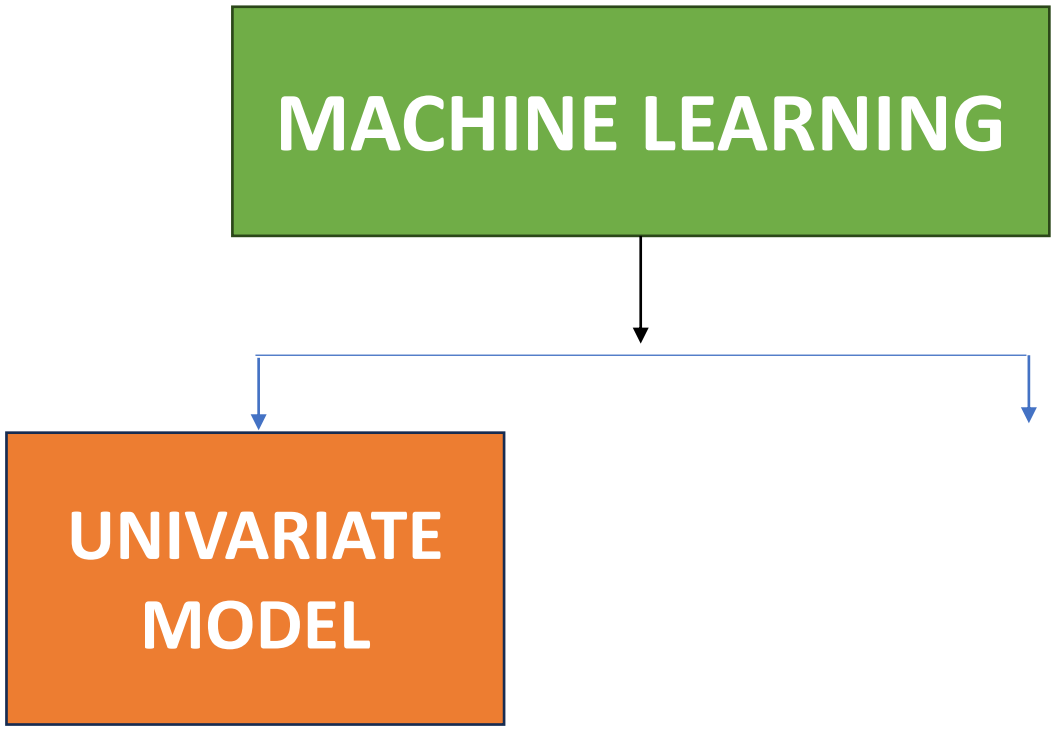


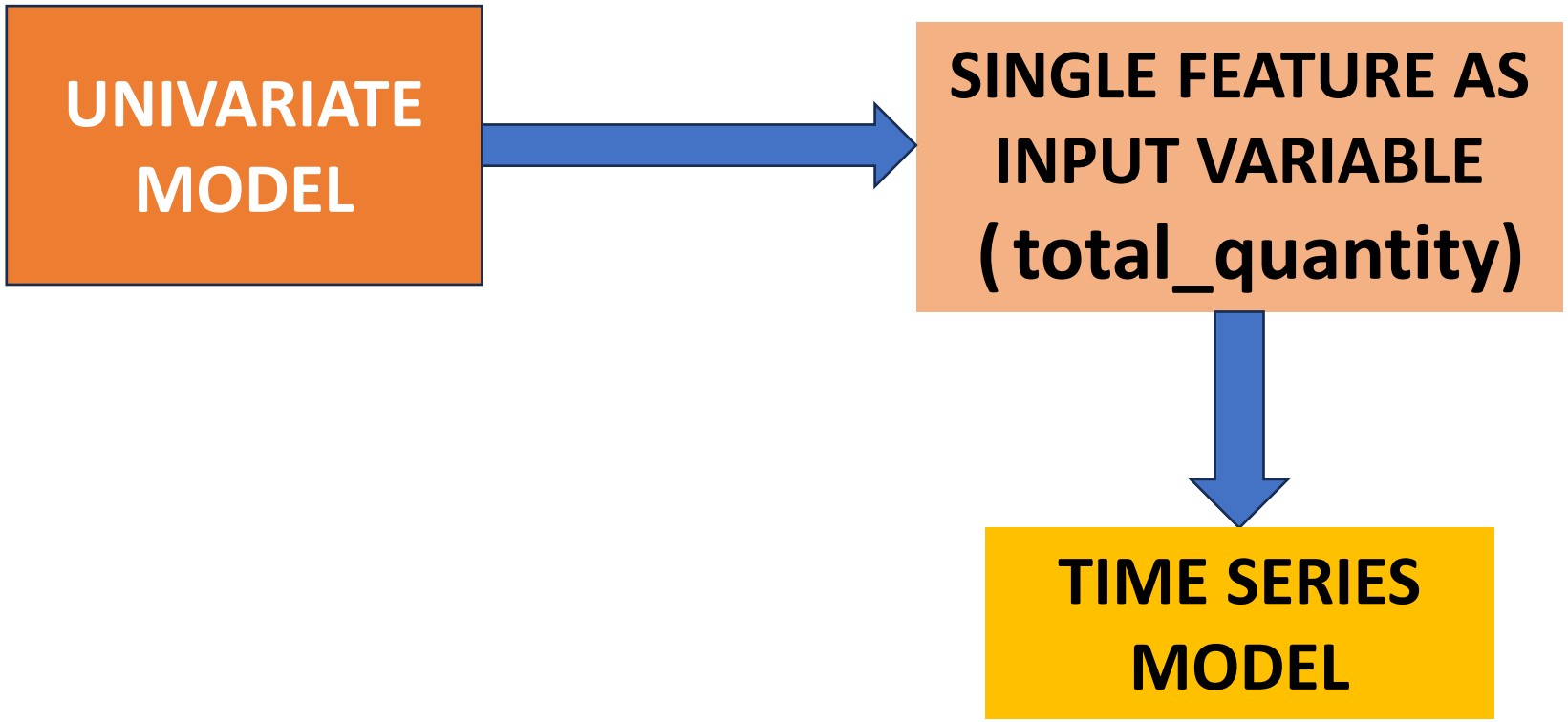
Pattern Visualization of My Dataset



MODEL BUILDING


MACHINE LEARNING
UNIVARIATE MODEL
MULTIVARIATE MODEL
Prediction Is Based On Single Features
Prediction Is Based On Multiple Features


MACHINE LEARNING
UNIVARIATE MODEL
Prediction Is Based on Single Features



- Auto Regressive Integrated Moving Average(ARIMA)
- Moving Average(MA)
- Simple Exponential Smoothing(SES)
- Holt Linear (HL)
- Holt Winter (HW)

Before implementing a model, the question arises:
why time series models?

Time series models are used for several reasons:

1. primarily to analyze, interpret, and make predictions based on **temporal data**.
2. Time series models help in identifying and understanding patterns, trends, and seasonality in data that change over time.
3. One of the main objectives of time series modeling is to forecast future values based on historical data.

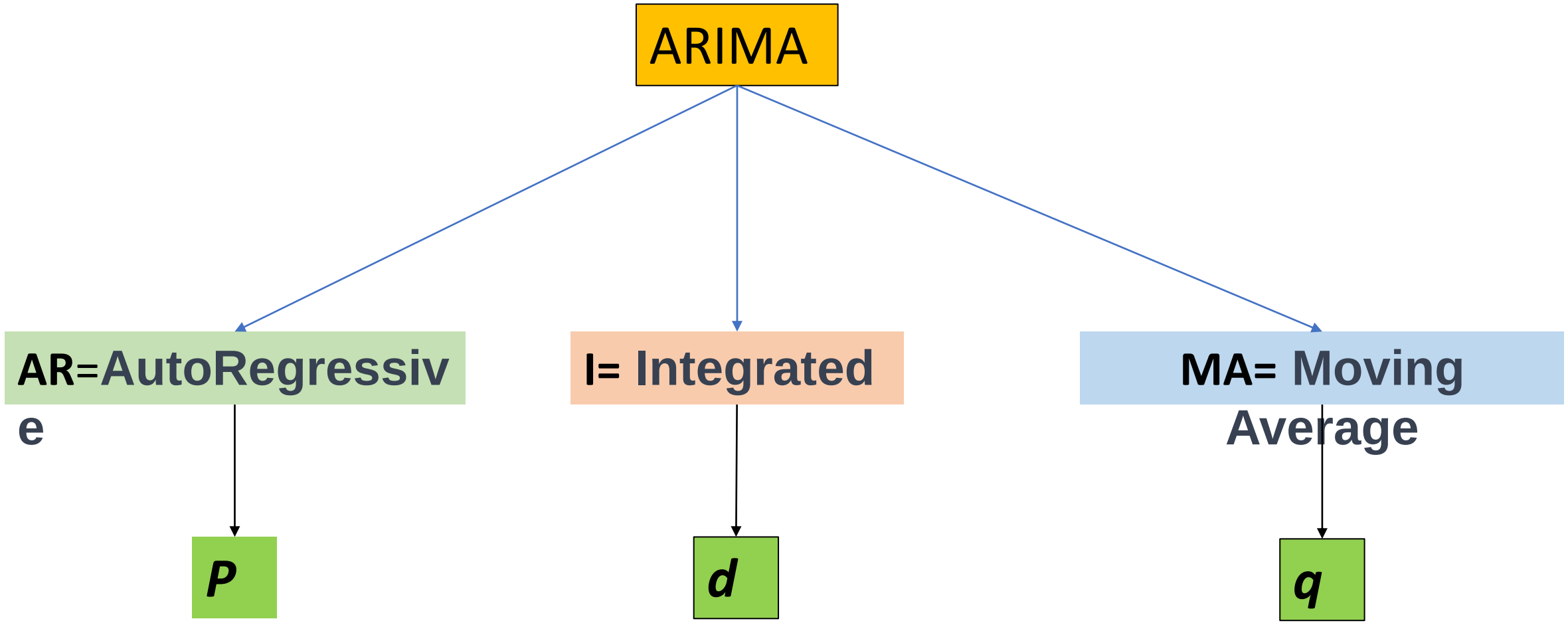
Note: Temporal data refers to data that is collected and recorded over time, typically in a sequential or chronological order

ARIMA MODEL

Why ARIMA MODEL?

Because:

1. Dataset is time dependent.
2. Dataset is stationary.
3. Input variable (total_quantity) is showing Auto-correlation.


ARIMA
AR=AutoRegressive
I= Integrated
MA= Moving Average
P
d
q

ARIMA Model (AutoRegressive Integrated Moving Average)

Model Components:

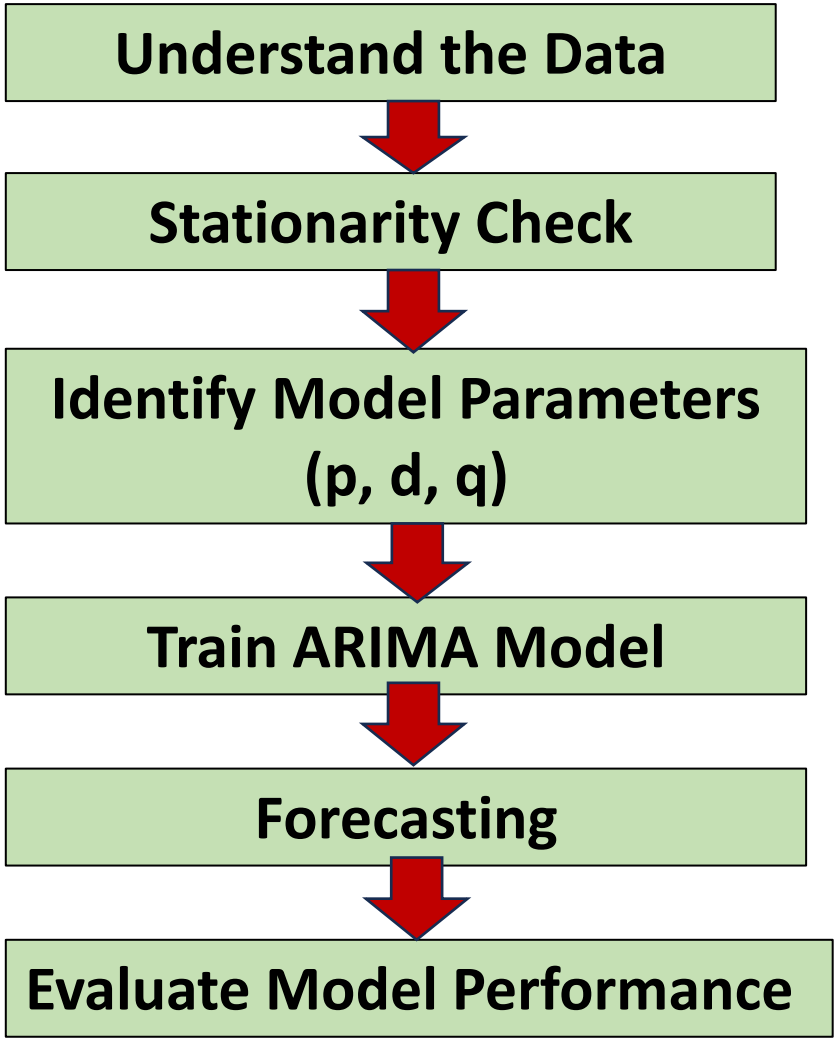
1.AutoRegressive (AR) Component: This captures the **linear relationship between an observation and its lagged values**. The order of the AR component is denoted by *p*.

2.Integrated (I) Component: This represents the differencing of the time series to make it **stationary**. The order of differencing is denoted by *d*.

3.Moving Average (MA) Component: This models the **dependency between an observation and a residual error** from a moving average model. The order of the MA component is denoted by *q*.

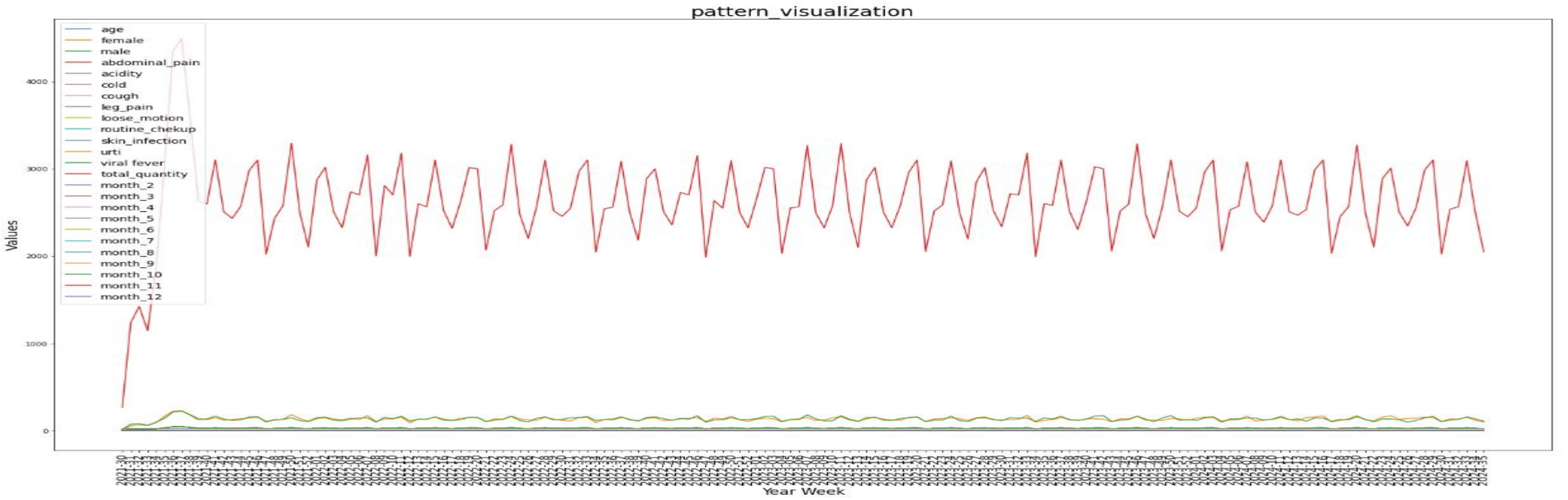
NOTES: lag refers to the time difference between observations in a time series.

Applying an ARIMA model involves a systematic procedure that includes the following steps:



Understand the Data

- Explore the time series data to understand its patterns, trends, and seasonality.
- Plot the data to inspect any evident patterns visually.



INFERENCE: Fully time-dependent, zig-zag pattern

Stationarity Check

- Stationary tends to – 1. Mean constant
  2. Standard deviation constant
- Check if the time series is stationary using visual inspection and statistical tests like the **Augmented Dickey-Fuller (ADF) test**.
- If the series is not stationary, apply differencing until stationarity is achieved.

Augmented Dickey-Fuller (ADF) test

Test is a statistical test used to determine whether a unit root is present in a time series dataset.

A **unit root** is an indication that a time series is non-stationary. The presence of a unit root implies that the time series exhibits a **random walk and has a non-constant mean**, which makes it challenging to predict future values

NOTES: The ADF statistic is the primary test statistic. It measures how strongly the time series is influenced by a unit root. **The more negative the ADF statistic, the stronger the evidence against the presence of a unit root, means data is stationary.**

Contd..

Hypothesis Testing For Finding A Stationarity

The null hypothesis is

H_0: Non-stationary ($\varphi = 1$)

The alternative hypothesis is

H_1: Stationary ($\varphi < 1$)

Φ= unit root

p-value:

p-value < **α** reject H_0,

p-value >= **α** do not reject H_0

Significance Level (α): 0.05

Contd.. I performed the code in my data set, codes are

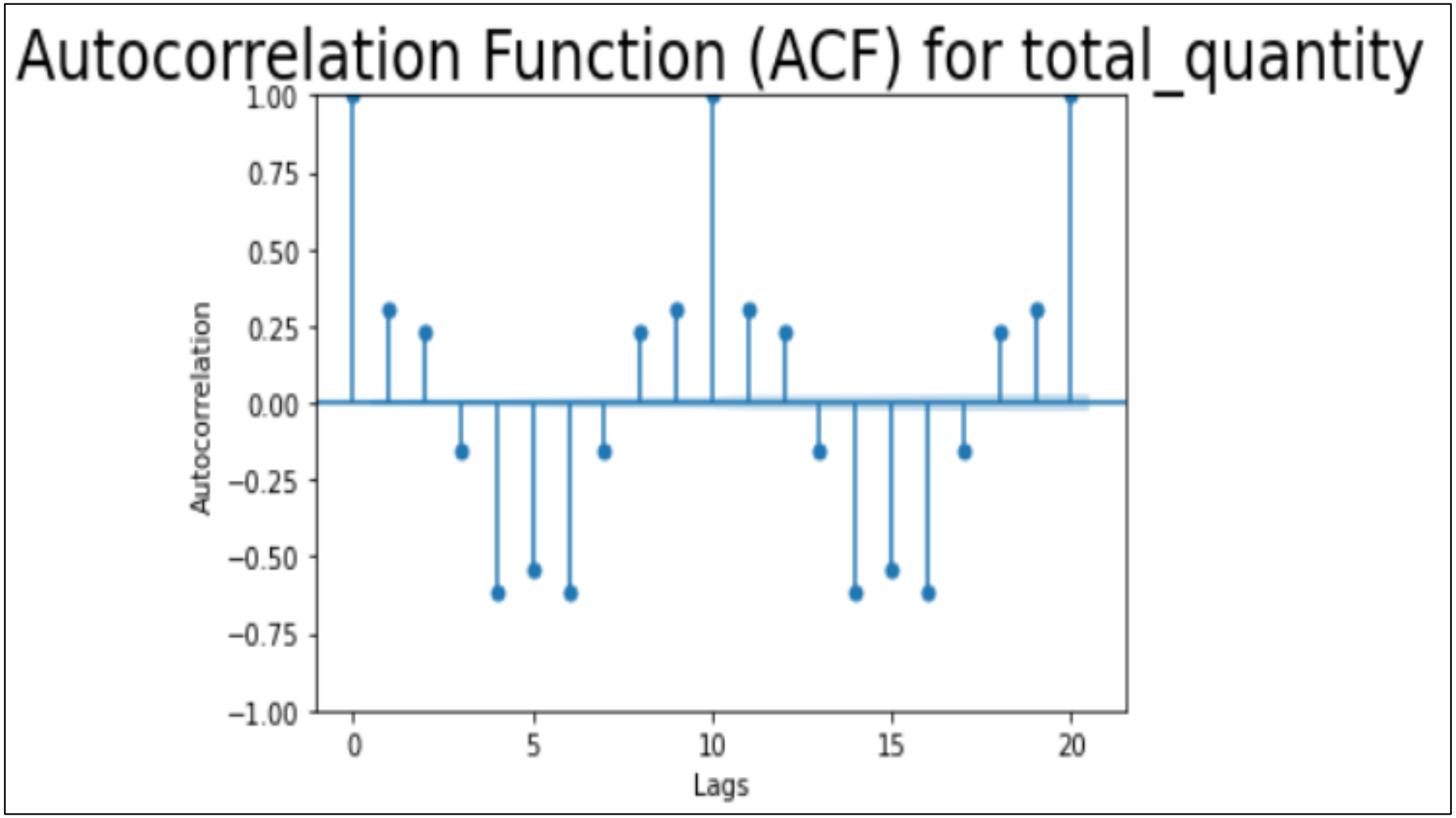
```
from statsmodels.tsa.stattools import adfuller
# Perform Augmented Dickey-Fuller test
adf_result = adfuller(df_clean['total_quantity'], autolag='AIC')

# Print the results
print("ADF Statistic:", adf_result[0])
print("p-value:", adf_result[1])
print("Critical Values:", adf_result[4])

# Interpret the results
if adf_result[1] <= 0.05:
    print("The time series is likely stationary.")
else:
    print("The time series is likely non-stationary.")
```

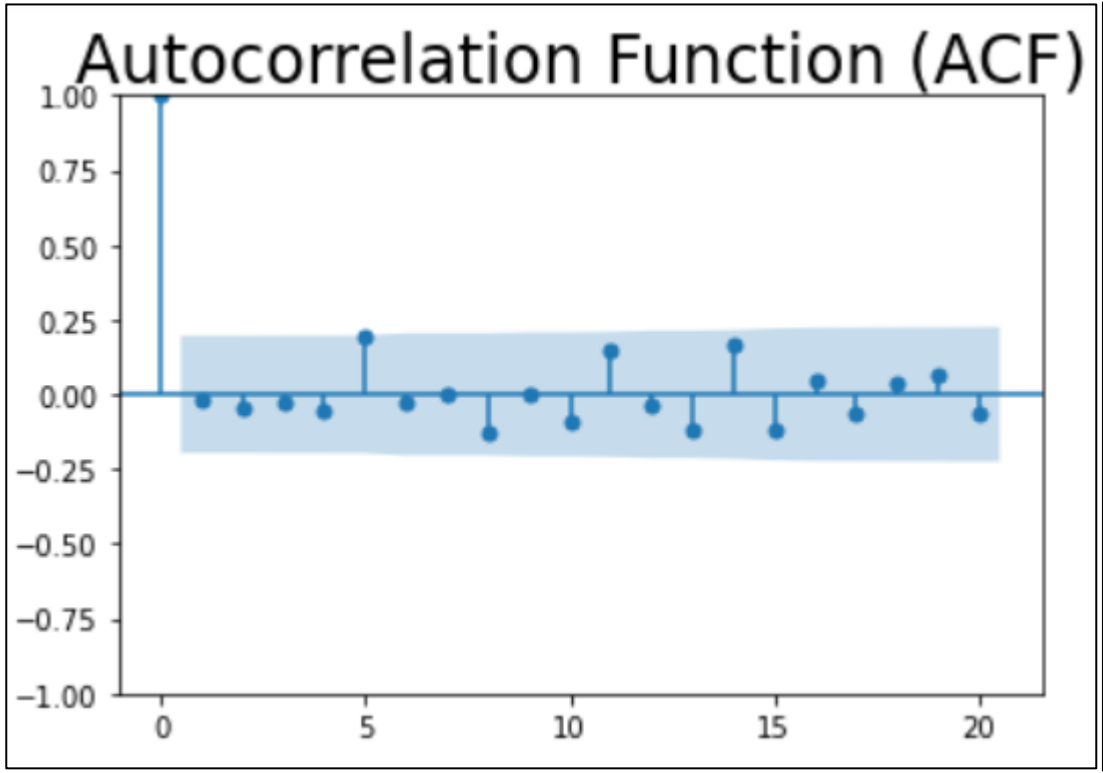
```
ADF Statistic: -7.673963682440713
p-value: 1.5666522937096034e-11
Critical Values: {'1%': -3.475018024295416, '5%': -2.881140802842043, '10%': -2.577221358046935}
The time series is likely stationary.
```

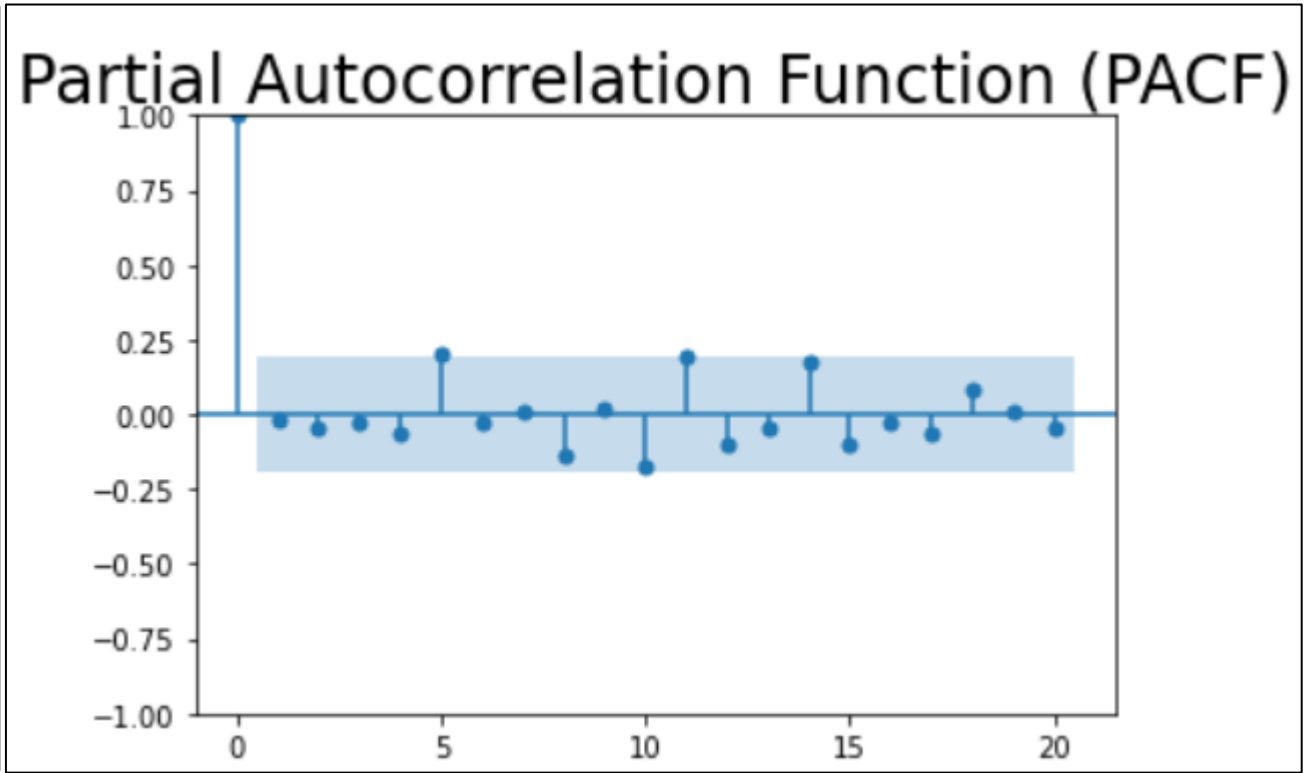
INFERENCE: p-value< **α,** means reject the null hypothesis, means alternative hypothesis is acceptable, means unit root less than 1, **data is stationary**


Autocorrelation Function (ACF) for total_quantity
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Autocorrelation
0
5
10
15
20
Lags

Identify Model Parameters (*p*, d, q):

- p: Order of the autoregressive (AR) component (lag order for autocorrelation).
- d: Order of differencing needed to achieve stationarity.
- q: Order of the moving average (MA) component (lag order for moving average).
- Plot autocorrelation function (ACF) and partial autocorrelation function (PACF) to identify potential values for p and q.





Train ARIMA Model

- Split the data into training and testing sets.
- Use the training set to fit the ARIMA model.

Pseudo-Code for ARIMA MODEL

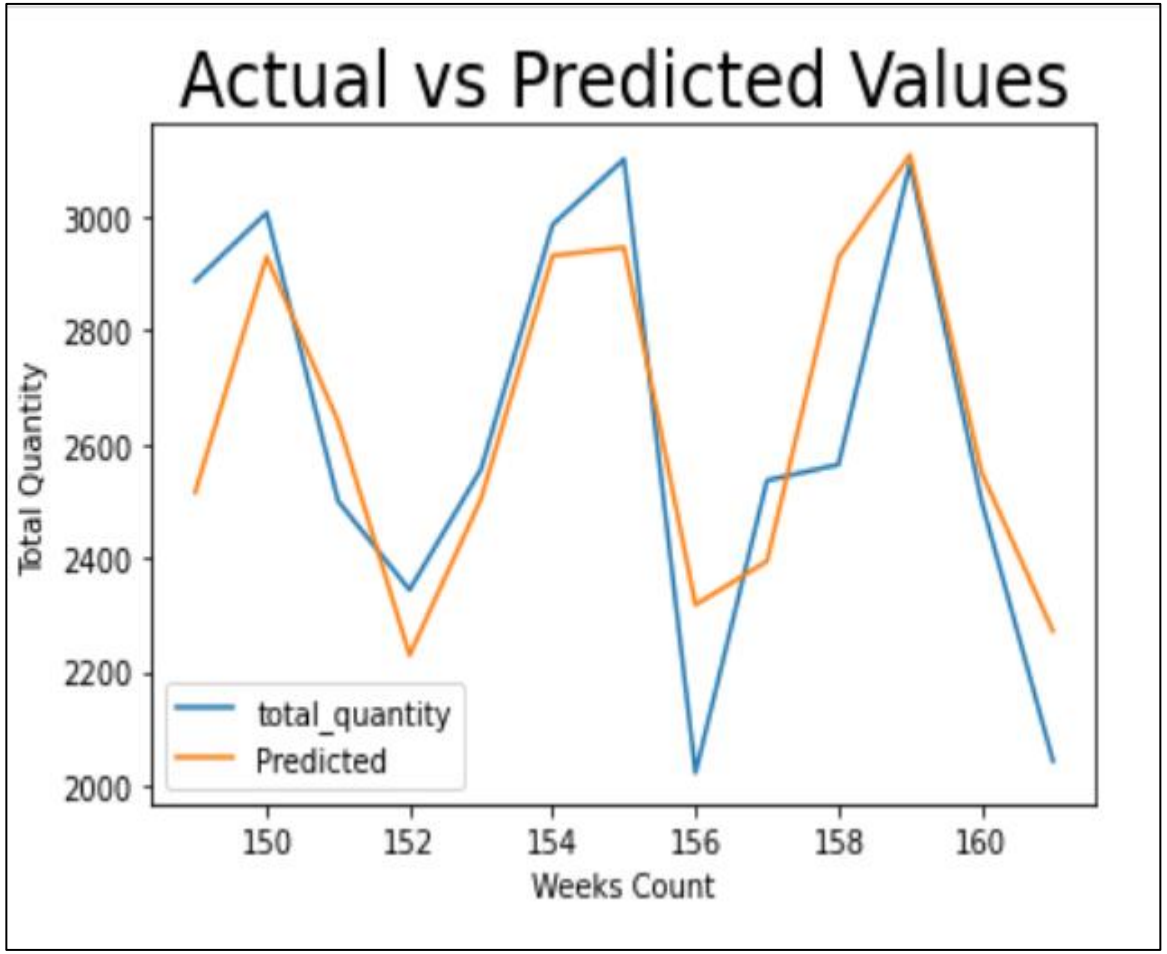
```
function fit_arima_model(data, order=(p, d, q)):
    model = ARIMA(data, order=order)
    fitted_model = model.fit(disp=0)
    return fitted_model
function forecast_arima_model(fitted_model, steps):
    forecast, stderr, conf_int = fitted_model.forecast(steps=steps)
    return forecast
```

Python code for ARIMA model

```
def run_fcast(ser,model,h,period):

    if model == "arima":
        print("Running Arima.........")
#         stepwise_model_arima = auto_arima(ser, start_p=0, start_q=0,max_p=2, max_q=2, m=period,seasonal=False,max_d=1,
#                               trace=False,
#                               error_action='ignore',
#                               suppress_warnings=True,
#                               stepwise=True)
#         fcast = np.round(stepwise_model_arima.predict(n_periods=h)).astype(int)
        fcast = []
        history = ser.tolist()
        for i in range(0,h):
            model = ARIMA(history, order = (5,0,4))
            model_fit = model.fit()
            out = model_fit.forecast()
            yhat = out[0]
            fcast.append(yhat)
            history.append(yhat)
        return fcast;
```

ARIMA Output




```
arima_fcast=univariate_forecast(train,'arima',test)
arima_fcast
```

```
Running Arima.........
```

	year_week	total_quantity	Predicted
149	2024-23	2886	2515.160967
150	2024-24	3006	2928.539606
151	2024-25	2500	2639.545482
152	2024-26	2344	2229.638764
153	2024-27	2556	2503.625467
154	2024-28	2985	2930.529543
155	2024-29	3100	2945.116324
156	2024-30	2023	2317.640260
157	2024-31	2536	2394.437882
158	2024-32	2564	2927.037659
159	2024-33	3092	3107.455161
160	2024-34	2500	2550.499655
161	2024-35	2044	2271.640619

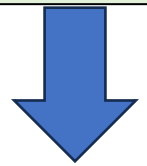
Performance metrics

```
arima_fcast_mse = mean_squared_error(arima_fcast['total_quantity'], arima_fcast['Predicted'])
arima_fcast_mape = mean_absolute_percentage_error(arima_fcast['total_quantity'], arima_fcast['Predicted'])*100
arima_fcast_rmse = sqrt(arima_fcast_mse)
metrics_table = pd.DataFrame({
    'Metric': ['Mean Squared Error (MSE)', 'Root Mean Squared Error (RMSE)', 'Mean Absolute Percentage Error (MAPE)'],
    'Value': [arima_fcast_mse,arima_fcast_rmse,arima_fcast_mape]
})

# Display the table
print(metrics_table)
```

```
                             Metric         Value
0              Mean Squared Error (MSE)  38386.969018
1       Root Mean Squared Error (RMSE)    195.925927
2  Mean Absolute Percentage Error (MAPE)      6.363061
```

Studied forecasting techniques in Manufacturing and Planning Control (MPC) last year. Now, these techniques are implemented using Python.



- **Moving Average (MA)**
- **Simple Exponential smoothing (SES)**
- **Holt Linear (HL)**
- **Holt Winter (HW)**

The mathematical concepts behind traditional forecasting methods using-

"Supply Chain Engineering Models and Applications" by A. Ravi Ravindran and Donald Paul Warsing as my reference book.

Moving Average

Here the entire historical data is used by computing the average of all the past demand for the forecast.

$$F_{n+1} = \sum_{t=1}^{n} \frac{D_t}{n}$$

where

F_{n+1} is the forecast of demand for period (n + 1)

D=Demand

n= number of periods

Simple exponential smoothing(SES)

The Simple Exponential Smoothing (SES) method is a popular technique in time series analysis for forecasting based on the exponentially weighted moving average of past observations. It assigns exponentially decreasing weights to historical data, giving more importance to recent observations.

$$F_{n+1} = \alpha D_n + (1 - \alpha) F_n$$

α is the smoothing parameter (0< α<1)

Pseudo-Code SES

```
function simple_exponential_smoothing(historical_DATA, alpha):
    forecasts = []
    forecasts.append(historical_DATA[0])  # Initial forecast is set equal
to the first observation

    for t in range(1, len(historical_DATA)):
        forecast_t = alpha * historical_DATA[t-1] + (1 - alpha) *
forecasts[t-1]
        forecasts.append(forecast_t)

    return forecasts
```

Machine Learning Codes For Exponential Moving Average

```
elif model == "simexp":
    print("Running Simple Exp.............")
    fcast = np.round(SimpleExpSmoothing(np.asarray(ser)).fit(smoothing_level=0.2,optimized=False).forecast(h)).astype(int)
    fcast[fcast<0] = 0
    return fcast;
```

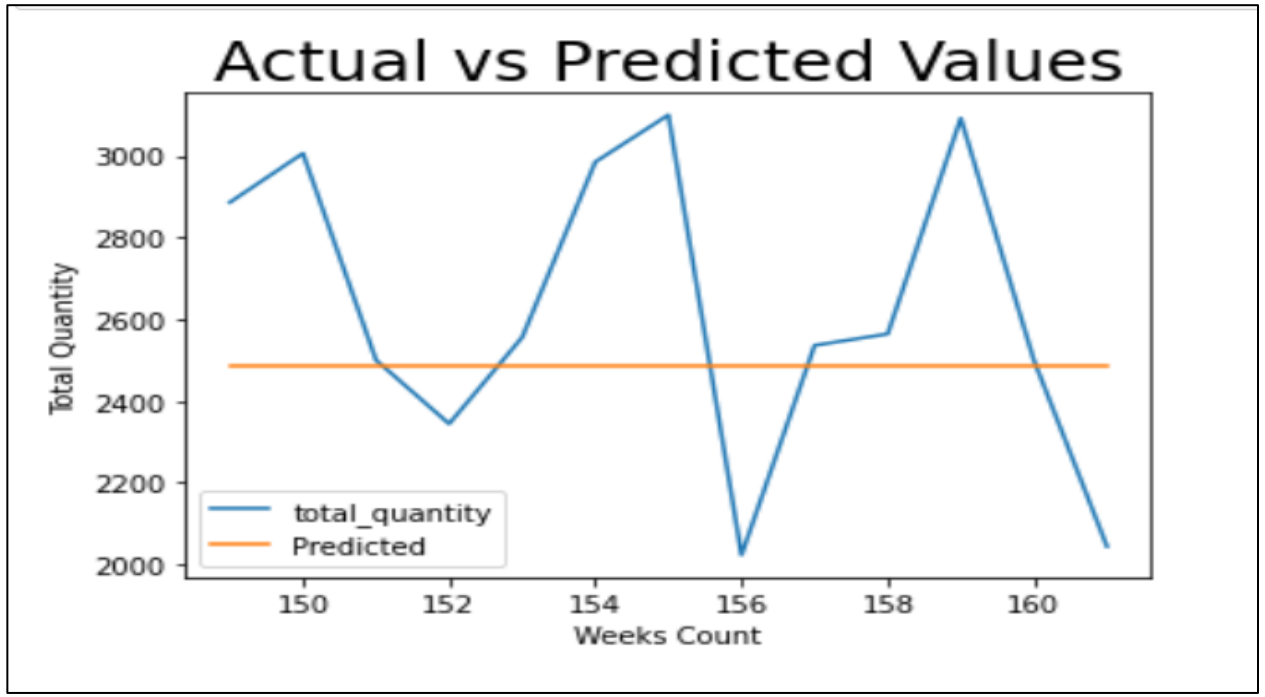
Performance metrics

```
simexp_fcast_mse = mean_squared_error(simexp_fcast['total_quantity'], ma_fcast['Predicted'])
simexp_fcast_rmse = sqrt(simexp_fcast_mse)
simexp_fcast_mape = mean_absolute_percentage_error(simexp_fcast['total_quantity'], simexp_fcast['Predicted'])*100
# Create a DataFrame to store the metrics
metrics_table = pd.DataFrame({
    'Metric': ['Mean Squared Error (MSE)', 'Root Mean Squared Error (RMSE)', 'Mean Absolute Percentage Error (MAPE)'],
    'Value': [ma_fcast_mse, ma_fcast_rmse, ma_fcast_mape]
})

# Display the table
print(metrics_table)
```

```
                             Metric          Value
0               Mean Squared Error (MSE)  143927.307692
1         Root Mean Squared Error (RMSE)     379.377527
2  Mean Absolute Percentage Error (MAPE)      11.303470
```

Simple exponential Moving Average Output




```
In [821]: ma_fcast = univariate_forecast(train, 'ma', test)
          ma_fcast
```

```
Running Moving Average.....
```

Out[821]:

	year_week	total_quantity	Predicted
149	2024-23	2886	2485
150	2024-24	3006	2485
151	2024-25	2500	2485
152	2024-26	2344	2485
153	2024-27	2556	2485
154	2024-28	2985	2485
155	2024-29	3100	2485
156	2024-30	2023	2485
157	2024-31	2536	2485
158	2024-32	2564	2485
159	2024-33	3092	2485
160	2024-34	2500	2485
161	2024-35	2044	2485

Holt Linear(HL)

Holt's Linear Exponential Smoothing (or simply Holt's Linear) is a method for time series forecasting that extends the Simple Exponential Smoothing (SES) method to handle data with a trend component. It's particularly useful when there is a linear trend in the time series.

Holt's method is also known as **double exponential smoothing or trend adjusted exponential smoothing method.**

Key Concepts of Holt's Linear Exponential Smoothing:

Components:

- Holt's Linear considers two main components of a time series: level (L_t) and trend (T_t).
- Level (L_t) represents the smoothed value of the series.
- Trend (T_t) represents the rate of change in the series.

Smoothing Parameters (α and β):

- α and β are smoothing parameters that control the weights assigned to the observed value and the previous level/trend.

Forecast Calculation

The forecast (*Ft*) is a combination of the level and trend components:

$$F_{t+1} = L_{t+1} + T_{t+1}$$

Under the exponential smoothing method, the estimate of the level for (t + 1) is given by

$$L_{t+1} = \alpha D_t + (1 - \alpha) F_t$$

The same approach is used to estimate the trend factor for (t + 1) using another smoothing constant β as follows

$$T_{t+1} = \beta [L_{t+1} - L_t] + (1 - \beta) T_t$$

Pseudo-Code Holt Linear

```
function holt_linear_exponential_smoothing(data, alpha, beta):
    # Initialize level and trend
    level = data[0]
    trend = data[1] - data[0]
    # Initialize forecasts list
    forecasts = [data[0]]
    # Iterate through the data
    for i in range(1, len(data)):
        # Calculate forecast
        forecast = level + trend
        forecasts.append(forecast)
        # Update level and trend
        level = alpha * data[i] + (1 - alpha) * (level + trend)
        trend = beta * (level - forecasts[i-1]) + (1 - beta) * trend
        return forecasts
```

Machine Learning Codes for Holt linear

```
elif model == "holtlinear":
    print("Running Holt Linear.....")
    fcast = np.round(Holt(ser).fit(smoothing_level = 0.2,smoothing_slope = 0.1).forecast(h)).astype(int)
    fcast[fcast<0] = 0
    return fcast;
```

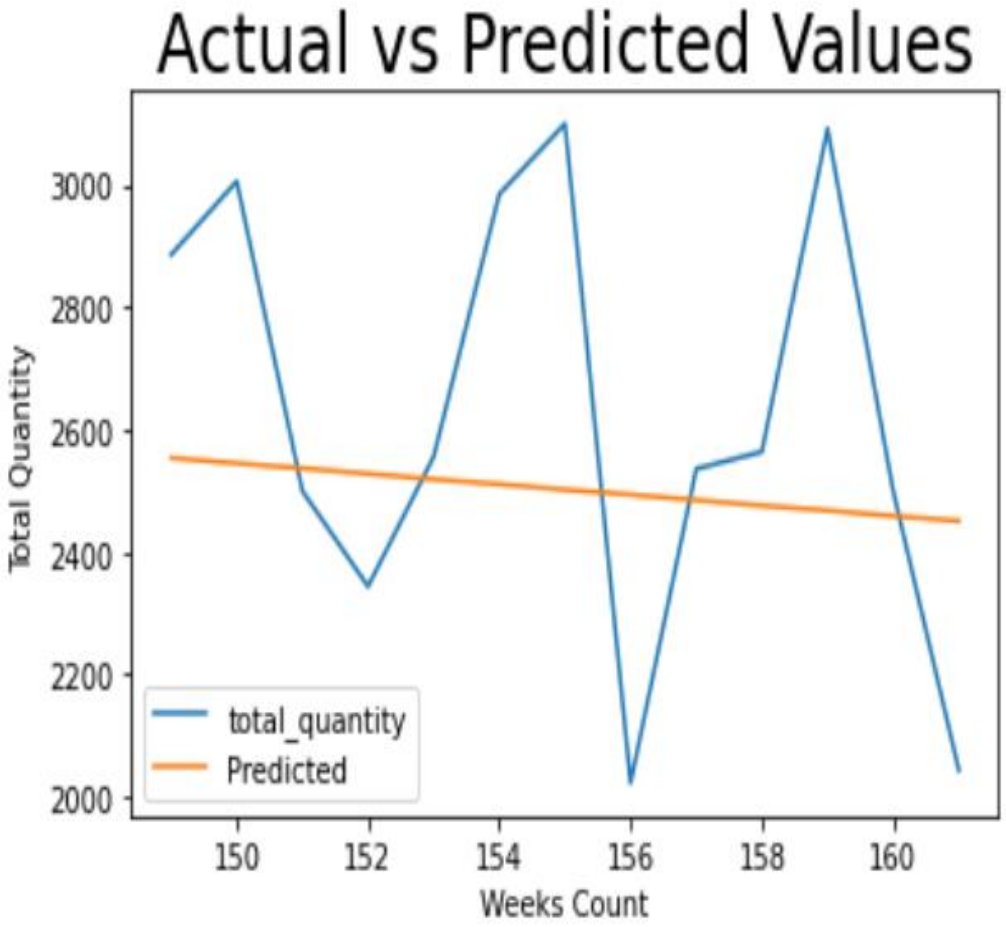
Performance metrics

```
hl_fcast_mse = mean_squared_error(hl_fcast['total_quantity'], hl_fcast['Predicted'])
hl_fcast_rmse = sqrt(hl_fcast_mse)
hl_fcast_mape = mean_absolute_percentage_error(hl_fcast['total_quantity'], hl_fcast['Predicted'])*100
# Create a DataFrame to store the metrics
metrics_table = pd.DataFrame({
    'Metric': ['Mean Squared Error (MSE)', 'Root Mean Squared Error (RMSE)', 'Mean Absolute Percentage Error (MAPE)'],
    'Value': [hl_fcast_mse, hl_fcast_rmse, hl_fcast_mape]
})

# Display the table
print(metrics_table)
```

```
                             Metric          Value
0              Mean Squared Error (MSE)  133116.230769
1       Root Mean Squared Error (RMSE)     364.850971
2  Mean Absolute Percentage Error (MAPE)      11.018878
```

Holt Linear Output




```
hl_fcast = univariate_forecast(train, 'holtlinear', test)
hl_fcast
```

```
Running Holt Linear.....
```

	year_week	total_quantity	Predicted
149	2024-23	2886	2554
150	2024-24	3006	2545
151	2024-25	2500	2537
152	2024-26	2344	2528
153	2024-27	2556	2519
154	2024-28	2985	2511
155	2024-29	3100	2502
156	2024-30	2023	2494
157	2024-31	2536	2485
158	2024-32	2564	2476
159	2024-33	3092	2468
160	2024-34	2500	2459

Holt-Winters Method(HW) (Triple Exponential Smoothing)

The Holt-Winters method is an extension of the Holt method that introduces seasonality into the forecasting model. It's suitable for time series data with a trend and seasonality.

Model Components:

- Level (l): Represents the average value in the time series.
- Trend (b): Represents the average rate of change in the time series.
- Seasonality (s): Represents the repeating patterns or cycles in the time series.

Equations

$$F_{t+1} = (L_{t+1} + T_{t+1})SI_{t+1} \quad \text{.....(1)}$$

$$L_{t+1} = \alpha\left(\frac{D_t}{SI_t}\right) + (1-\alpha)(L_t + T_t) \quad \text{.....(2)}$$

$$T_{t+1} = \beta(L_{t+1} - L_t) + (1-\beta)T_t \quad \text{.....(3)}$$

$$SI_{t+p} = \gamma\left(\frac{D_t}{L_t}\right) + (1-\gamma)SI_t \quad \text{.....(4)}$$

where α, β, and γ are smoothing constants between 0 and 1 for level, trend and seasonality respectively

Pseudo-Code Holt Winter

```
def holt_winters_exponential_smoothing(data, alpha, beta, seasonality_index):
    # Initialize level, trend, and seasonality
    level = data[0]
    trend = data[1] - data[0]
    seasonality = [data[i] / level for i in range(len(data))]  # Initialize seasonality index
    # Initialize forecasts list
    forecasts = [data[0]]
    # Iterate through the data
    for i in range(1, len(data)):
        # Calculate forecast
        forecast = (level + trend) * seasonality[i]
        forecasts.append(forecast)
        # Update level, trend, and seasonality
        level = alpha * data[i] / seasonality[i] + (1 - alpha) * (level + trend)
        trend = beta * (level - forecasts[i-1]) + (1 - beta) * trend
        seasonality[i] = gamma * data[i] / level + (1 - gamma) * seasonality[i]
    return forecasts
```

Machine Learning Codes For Holt Winter

```
elif model == "holtwinter":
    print("Running Holt Winters.....")
    fcast = np.round(ExponentialSmoothing(ser ,seasonal_periods=52 ,trend='add',
                            seasonal='add').fit(optimized = True).forecast(h)).astype(int)
    fcast[fcast<0] = 0
    return fcast;
```

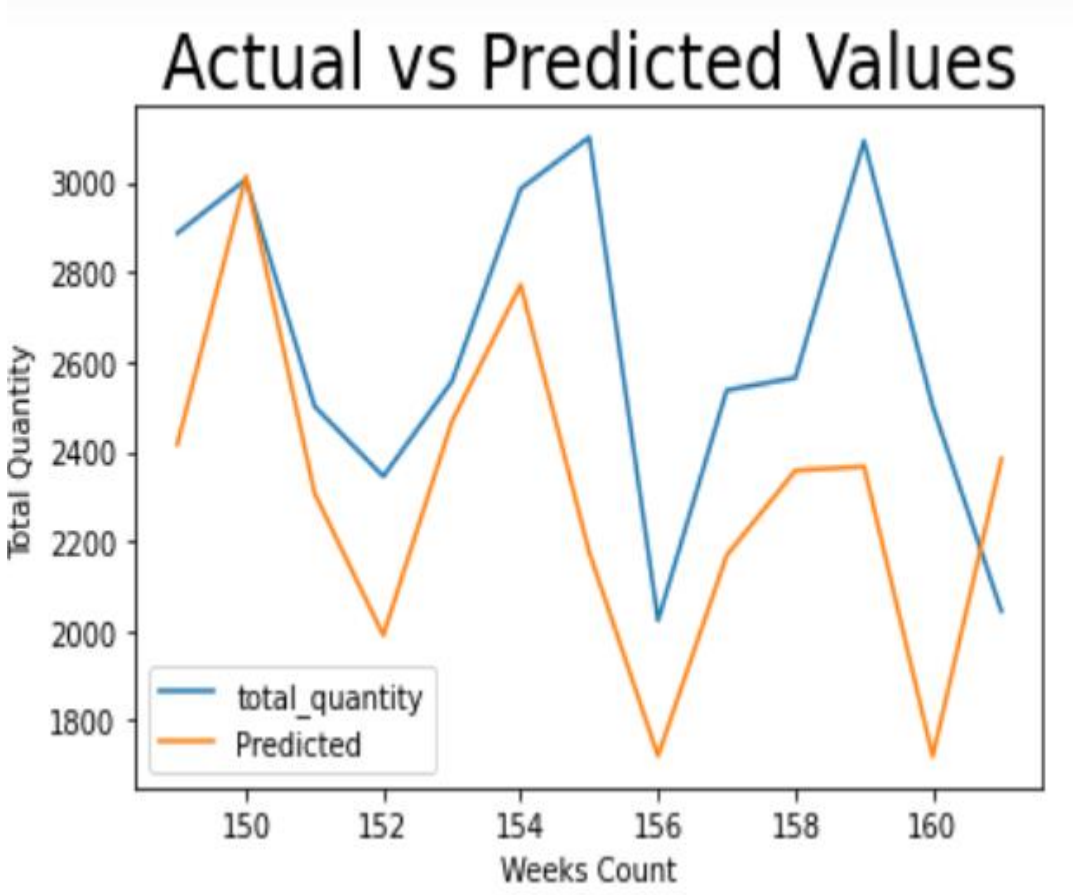
Performance metrics

```
hw_fcast_mse = mean_squared_error(hw_fcast['total_quantity'], hw_fcast['Predicted'])
hw_fcast_rmse = sqrt(hw_fcast_mse)
hw_fcast_mape = mean_absolute_percentage_error(hw_fcast['total_quantity'], hw_fcast['Predicted'])*100
# Create a DataFrame to store the metrics
metrics_table = pd.DataFrame({
    'Metric': ['Mean Squared Error (MSE)', 'Root Mean Squared Error (RMSE)', 'Mean Absolute Percentage Error (MAPE)'],
    'Value': [hw_fcast_mse, hw_fcast_rmse, hw_fcast_mape]
})

# Display the table
print(metrics_table)
```

```
                             Metric          Value
0              Mean Squared Error (MSE)  216608.384615
1       Root Mean Squared Error (RMSE)     465.412059
2  Mean Absolute Percentage Error (MAPE)      14.516564
```

Output of Holt Winter




```
hw_fcast = univariate_forecast(train, 'holtwinter', test)
hw_fcast
```

```
Running Holt Winters.....
```

	year_week	total_quantity	Predicted
149	2024-23	2886	2415
150	2024-24	3006	3013
151	2024-25	2500	2306
152	2024-26	2344	1990
153	2024-27	2556	2467
154	2024-28	2985	2771
155	2024-29	3100	2174
156	2024-30	2023	1721
157	2024-31	2536	2168
158	2024-32	2564	2357
159	2024-33	3092	2366
160	2024-34	2500	1720
161	2024-35	2044	2383

Results

S.NO	MODEL	RMSE	MAPE	ACCURACY
1.	ARIMA	195.925927	6.36	93.64
2.	HOLT LINEAR	364.850971	11.018878	88.99
3.	SIMPLE EXPONENTIAL SMOOTHING	379.377527	11.30	88.70
4.	HOLT WINTER	465.412059	14.516564	85.49

CONCLUSION

1. In this research, four methods (ARIMA, SES, HL, HW) were used on the stationary dataset.
2. The results show that the ARIMA model gives the best accuracy in predicting the total quantity of medicine.
3. As the dataset does not have any trend, no significant change in accuracy was observed in the Holt linear and simple exponential smoothing method.
4. Also, due to no seasonality impact on the dataset, the Holt winter results are quite similar to the other two methods.

Future Plan

1. Continue data collection.
2. Work on multivariate forecasting models.
3. Develop codes for inventory management.

Recent Literature Review

Author Name(year)
AlRuthia et al (2023)

Title of the work/Journal Name
Local causes of essential medicines shortages from the perspective of supply chain professionals in Saudi Arabia/Saudi Pharmaceutical Journal

Objective of work
To survey interruptions in the supply of essential drugs in the pharmaceutical supply chain

Findings from the work
- The public centralized pharmaceutical procurement must reform its procurement and purchasing practices.
- Reform healthcare institutions' inventory management and demand forecasting to avoid frequent and unfortunate shortages of essential medicines.

Gaps identified/Scope for future work
This research used descriptive questionnaires instead of quantitative study.

Recent Literature Review

Author Name(year)	Title of the work/Journal Name	Objective of work
Merkuryeva et al (2018)	Demand forecasting in pharmaceutical supply chains: A case study / ICTE in Transportation and Logistics	To study an integrated procedure for in-market product demand forecasting and purchase order generation in the pharmaceutical supply chain

Findings from the work

- The public centralized pharmaceutical procurement must reform its procurement and purchasing practices.
- Reform healthcare institutions' inventory management and demand forecasting to avoid frequent and unfortunate shortages of essential medicines.

REFERENCES

- AlRuthia, Y., Almutiri, N.M., Almutairi, R.M., Almohammed, O., Alhamdan, H., El-Haddad, S.A. and Asiri, Y.A., 2023. Local causes of essential medicines shortages from the perspective of supply chain professionals in Saudi Arabia. *Saudi Pharmaceutical Journal*, *31*(6), pp.948-954.
- Merkuryeva, G., Valberga, A. and Smirnov, A., 2019. Demand forecasting in pharmaceutical supply chains: A case study. *Procedia Computer Science*, *149*, pp.3-10.
- Dey, B., Roy, B., Datta, S. and Ustun, T.S., 2023. Forecasting ethanol demand in India to meet future blending targets: A comparison of ARIMA and various regression models. *Energy Reports*, *9*, pp.411-418.
- Ravindran, A. R., & Warsing, D. P. (2013). *Supply Chain Engineering Models and Applications*. Taylor & Francis Group, an Informa business.

THANK YOU