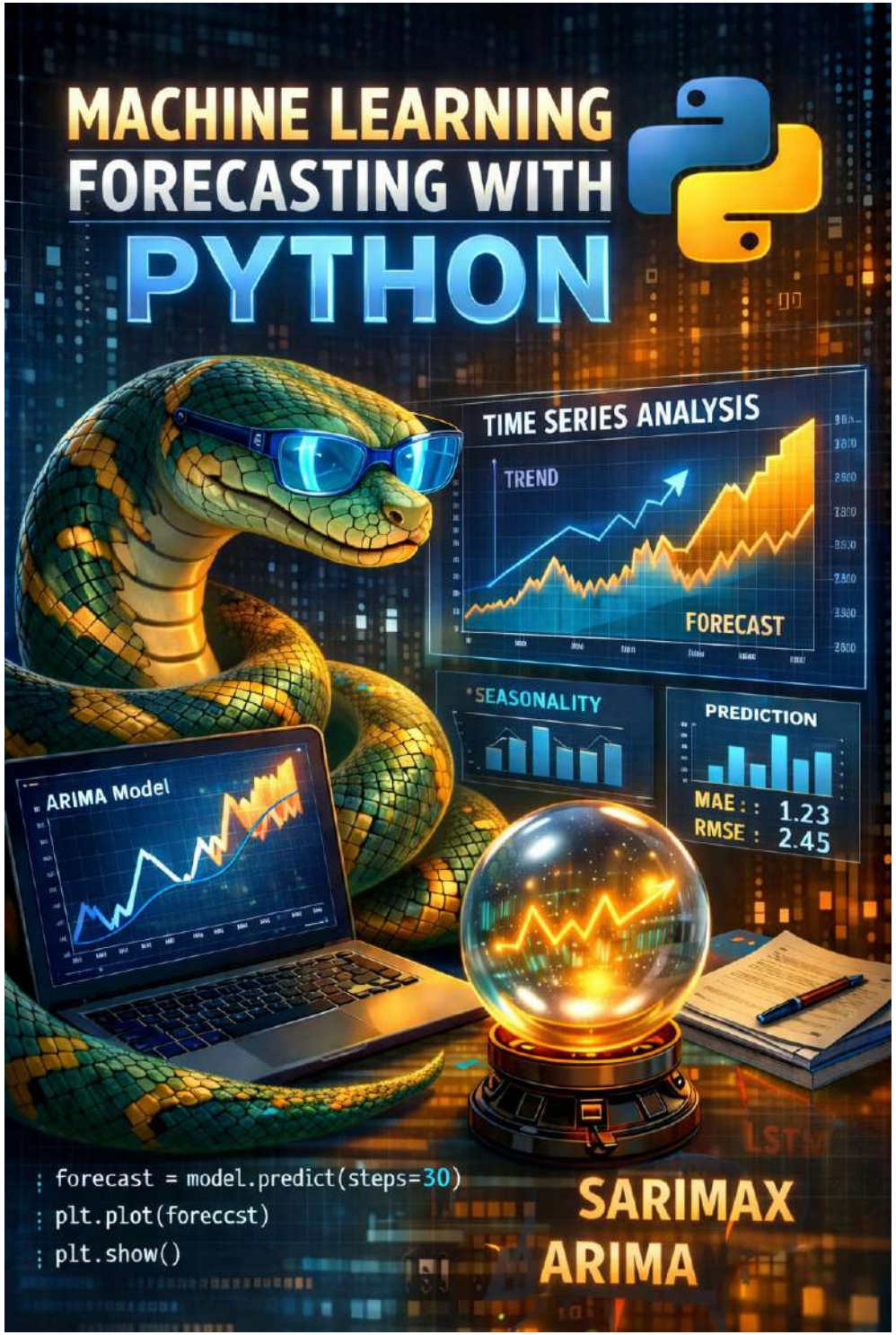
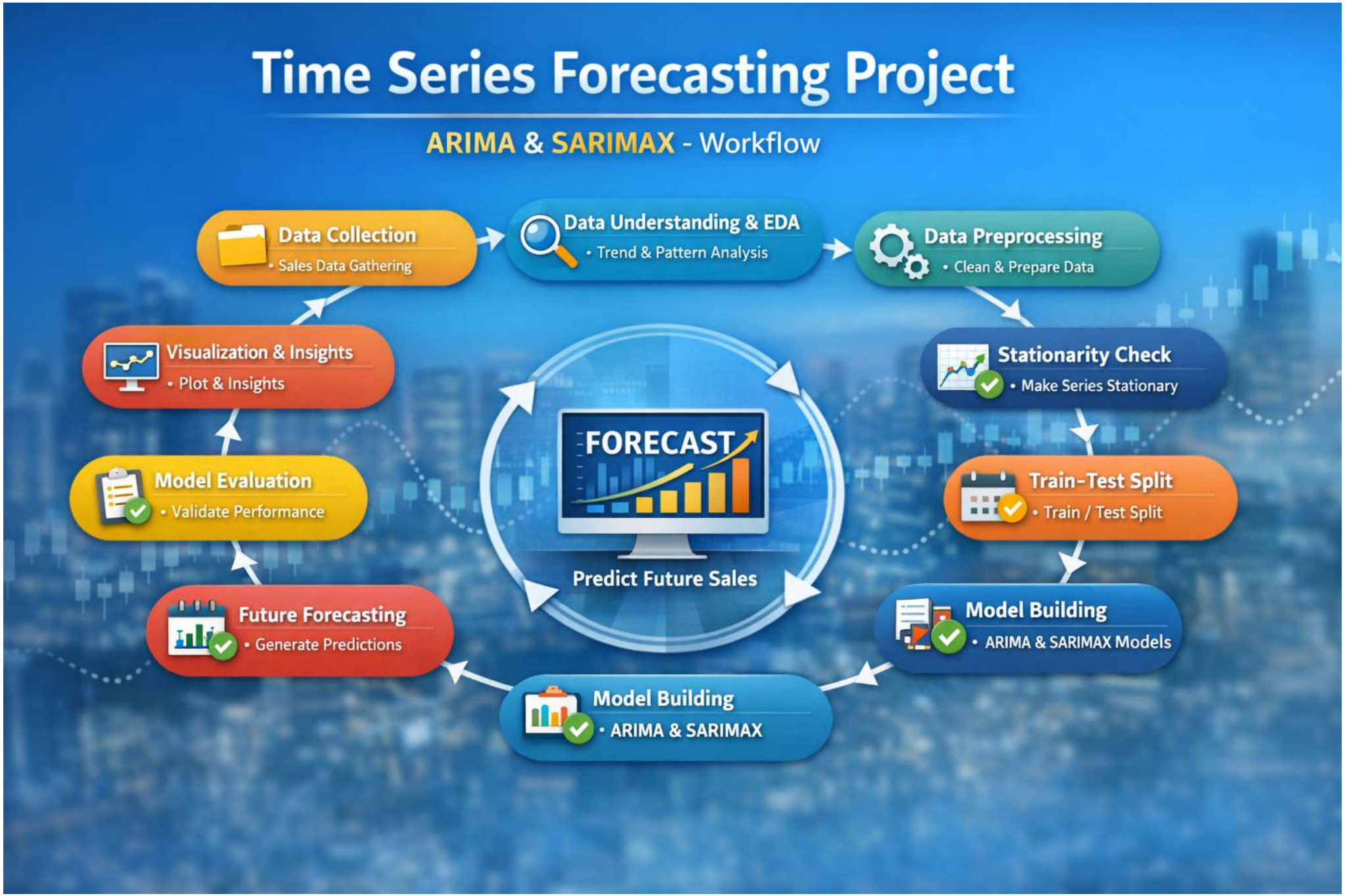

MACHINE LEARNING
FORECASTING WITH
PYTHON
TIME SERIES ANALYSIS
TREND
FORECAST
SEASONALITY
PREDICTION
MAE : 1.23
RMSE : 2.45
ARIMA Model
forecast = model.predict(steps=30)
plt.plot(foreccst)
plt.show()
SARIMAX
ARIMA
LSTM


Time Series Forecasting Project
ARIMA & SARIMAX - Workflow
Data Collection
• Sales Data Gathering
Data Understanding & EDA
• Trend & Pattern Analysis
Data Preprocessing
• Clean & Prepare Data
Stationarity Check
• Make Series Stationary
Train-Test Split
• Train / Test Split
Model Building
• ARIMA & SARIMAX Models
Model Building
• ARIMA & SARIMAX
Future Forecasting
• Generate Predictions
Model Evaluation
• Validate Performance
Visualization & Insights
• Plot & Insights
FORECAST
Predict Future Sales

ARIMA and Seasonal ARIMA

Autoregressive Integrated Moving Averages

The general process for ARIMA models is the following:

Visualize the Time Series Data

Make the time series data stationary

Plot the Correlation and AutoCorrelation Charts

Construct the ARIMA Model or Seasonal ARIMA based on the data

Use the model to make predictions

Let's go through these steps!

In [1]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline
```

In [3]:
```
df=pd.read_csv('perrin-freres-monthly-champagne-.csv')
```

In [4]:
```
df.head()
```

Out[4]:

	Month	Perrin Freres monthly champagne sales millions ?64-?72
0	1964-01	2815.0
1	1964-02	2672.0
2	1964-03	2755.0
3	1964-04	2721.0
4	1964-05	2946.0

In [5]:
```
df.tail()
```

Out[5]:

	Month	Perrin Freres monthly champagne sales millions ?64-?72
102	1972-07	4298.0
103	1972-08	1413.0
104	1972-09	5877.0
105	NaN	NaN
106	Perrin Freres monthly champagne sales millions...	NaN

In [6]:
```
## Cleaning up the data
df.columns=["Month","Sales"]
df.head()
```

Out[6]:

	Month	Sales
0	1964-01	2815.0
1	1964-02	2672.0
2	1964-03	2755.0
3	1964-04	2721.0
4	1964-05	2946.0

In [7]:
```
df.tail()
```

Out[7]:

	Month	Sales
102	1972-07	4298.0
103	1972-08	1413.0
104	1972-09	5877.0
105	NaN	NaN
106	Perrin Freres monthly champagne sales millions...	NaN

In [8]:
```
## Drop last 2 rows
df.drop(106,axis=0,inplace=True)
```

In [9]:
```
df.tail()
```

Out[9]:

	Month	Sales
101	1972-06	5312.0
102	1972-07	4298.0
103	1972-08	1413.0
104	1972-09	5877.0
105	NaN	NaN

In [11]:
```
df.drop(105,axis=0,inplace=True)
```

In [12]:
```
df.tail()
```

Out[12]:

	Month	Sales
100	1972-05	4618.0
101	1972-06	5312.0
102	1972-07	4298.0
103	1972-08	1413.0
104	1972-09	5877.0

In [13]:
```
# Convert Month into Datetime
df['Month']=pd.to_datetime(df['Month'])
```

In [14]:
```
df.head()
```

Out[14]:

	Month	Sales
0	1964-01-01	2815.0
1	1964-02-01	2672.0
2	1964-03-01	2755.0
3	1964-04-01	2721.0
4	1964-05-01	2946.0

In [15]:
```
df.set_index('Month',inplace=True)
```

In [16]:
```
df.head()
```

Out[16]:

	Sales
Month	
1964-01-01	2815.0
1964-02-01	2672.0
1964-03-01	2755.0
1964-04-01	2721.0
1964-05-01	2946.0

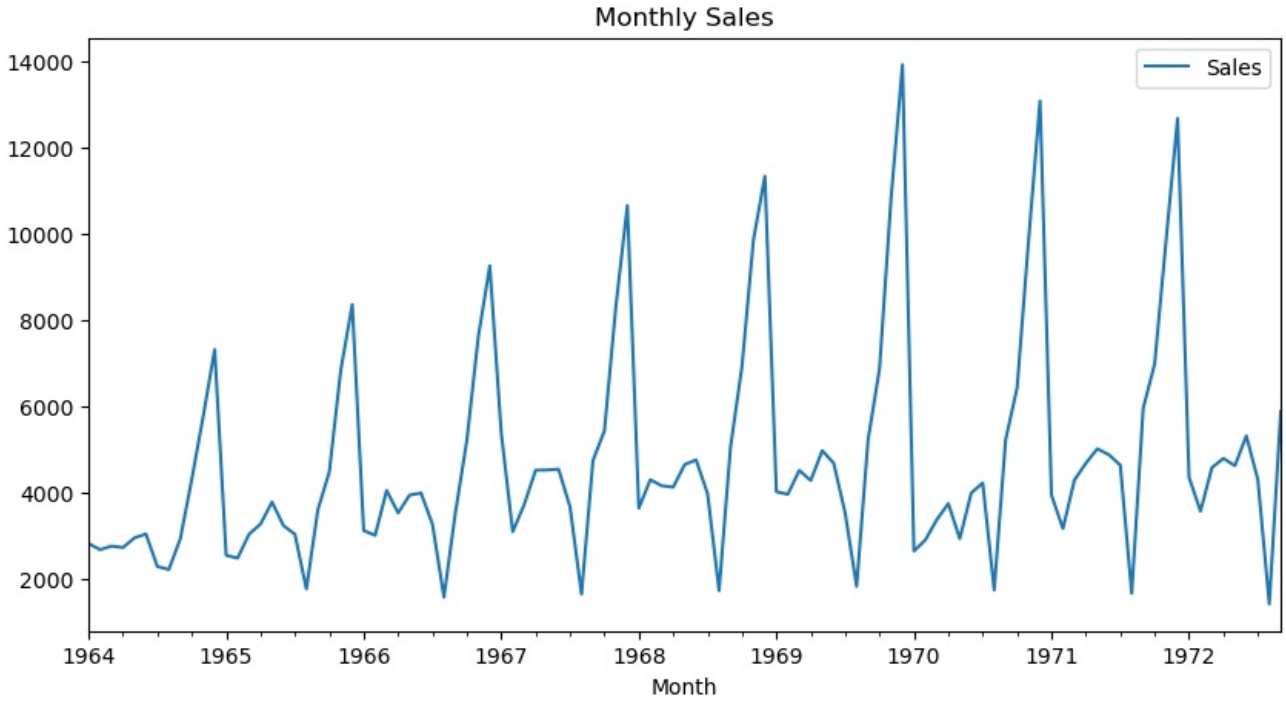
In [17]:
```
df.describe()
```

Out[17]:

	Sales
count	105.000000
mean	4761.152381
std	2553.502601
min	1413.000000
25%	3113.000000
50%	4217.000000
75%	5221.000000
max	13916.000000

Step 2: Visualize the Data

In [23]:
```
df.plot(figsize=(10,5), title="Monthly Sales")
plt.show()
```



In [24]:
```
### Testing For Stationarity

from statsmodels.tsa.stattools import adfuller
```

In [25]:
```
test_result=adfuller(df['Sales'])
```

In [26]:
```
#Ho: It is non stationary
#H1: It is stationary

def adfuller_test(sales):
    result=adfuller(sales)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations Used']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit ro
    else:
        print("weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationa
```

In [27]:
```
adfuller_test(df['Sales'])
```

```
ADF Test Statistic : -1.833593056327635
p-value : 0.36391577166023914
#Lags Used : 11
Number of Observations Used : 93
weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary
```

Differencing

In [28]:
```
df['Sales First Difference'] = df['Sales'] - df['Sales'].shift(1)
```

In [29]:
```
df['Sales'].shift(1)
```

Out[29]:
```
Month
1964-01-01       NaN
1964-02-01    2815.0
1964-03-01    2672.0
1964-04-01    2755.0
1964-05-01    2721.0
               ...
1972-05-01    4788.0
1972-06-01    4618.0
1972-07-01    5312.0
1972-08-01    4298.0
1972-09-01    1413.0
Name: Sales, Length: 105, dtype: float64
```

In [30]:
```
df['Seasonal First Difference']=df['Sales']-df['Sales'].shift(12)
```

In [31]:
```
df.head(15)
```

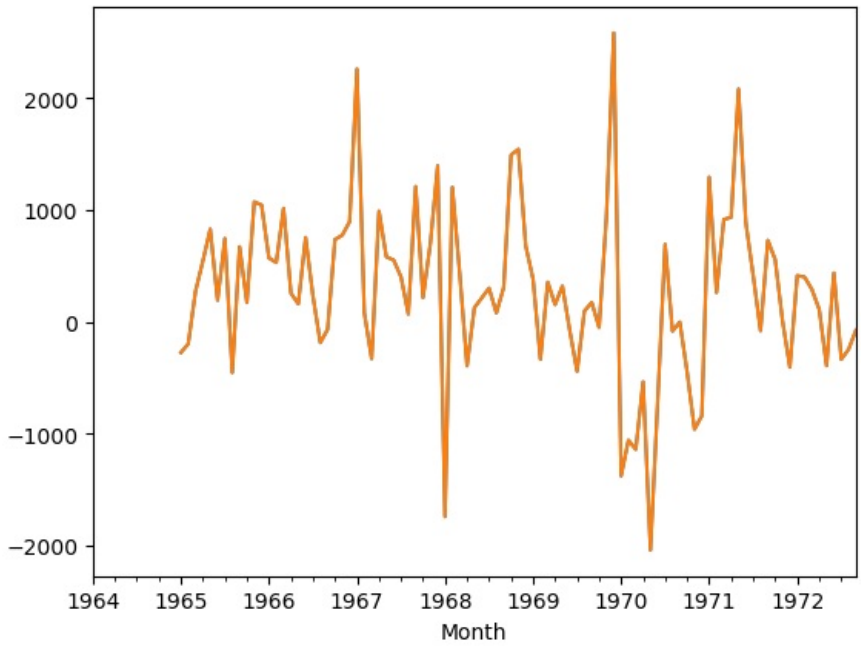
Out[31]:

Month	Sales	Sales First Difference	Seasonal First Difference
1964-01-01	2815.0	NaN	NaN
1964-02-01	2672.0	-143.0	NaN
1964-03-01	2755.0	83.0	NaN
1964-04-01	2721.0	-34.0	NaN
1964-05-01	2946.0	225.0	NaN
1964-06-01	3036.0	90.0	NaN
1964-07-01	2282.0	-754.0	NaN
1964-08-01	2212.0	-70.0	NaN
1964-09-01	2922.0	710.0	NaN
1964-10-01	4301.0	1379.0	NaN
1964-11-01	5764.0	1463.0	NaN
1964-12-01	7312.0	1548.0	NaN
1965-01-01	2541.0	-4771.0	-274.0
1965-02-01	2475.0	-66.0	-197.0
1965-03-01	3031.0	556.0	276.0

In [32]:
```
## Again test dickey fuller test
adfuller_test(df['Seasonal First Difference'].dropna())
```

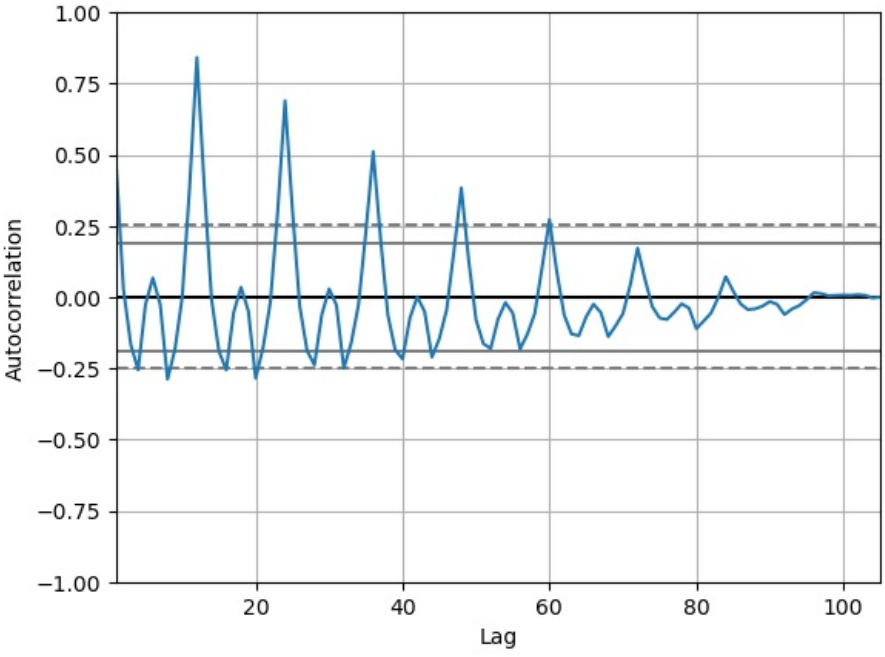
```
ADF Test Statistic : -7.626619157213163
p-value : 2.060579696813685e-11
#Lags Used : 0
Number of Observations Used : 92
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root and is statio
nary
```

In [34]:
```
import matplotlib.pyplot as plt

df['Seasonal First Difference'].plot()
plt.show()
```



Auto Regressive Model

In [37]:
```
from pandas.plotting import autocorrelation_plot
import matplotlib.pyplot as plt

autocorrelation_plot(df['Sales'])
plt.show()
```



This means:

✅ data has seasonality ✅ Past values affect future values

Final Thoughts on Autocorrelation and Partial Autocorrelation

Identification of an AR model is often best done with the PACF.

For an AR model, the theoretical PACF “shuts off” past the order of the model. The phrase “shuts off” means that in theory the partial autocorrelations are equal to 0 beyond that point. Put another way, the number of non-zero partial autocorrelations gives the order of the AR model. By the “order of the model” we mean the most extreme lag of x that is used as a predictor.

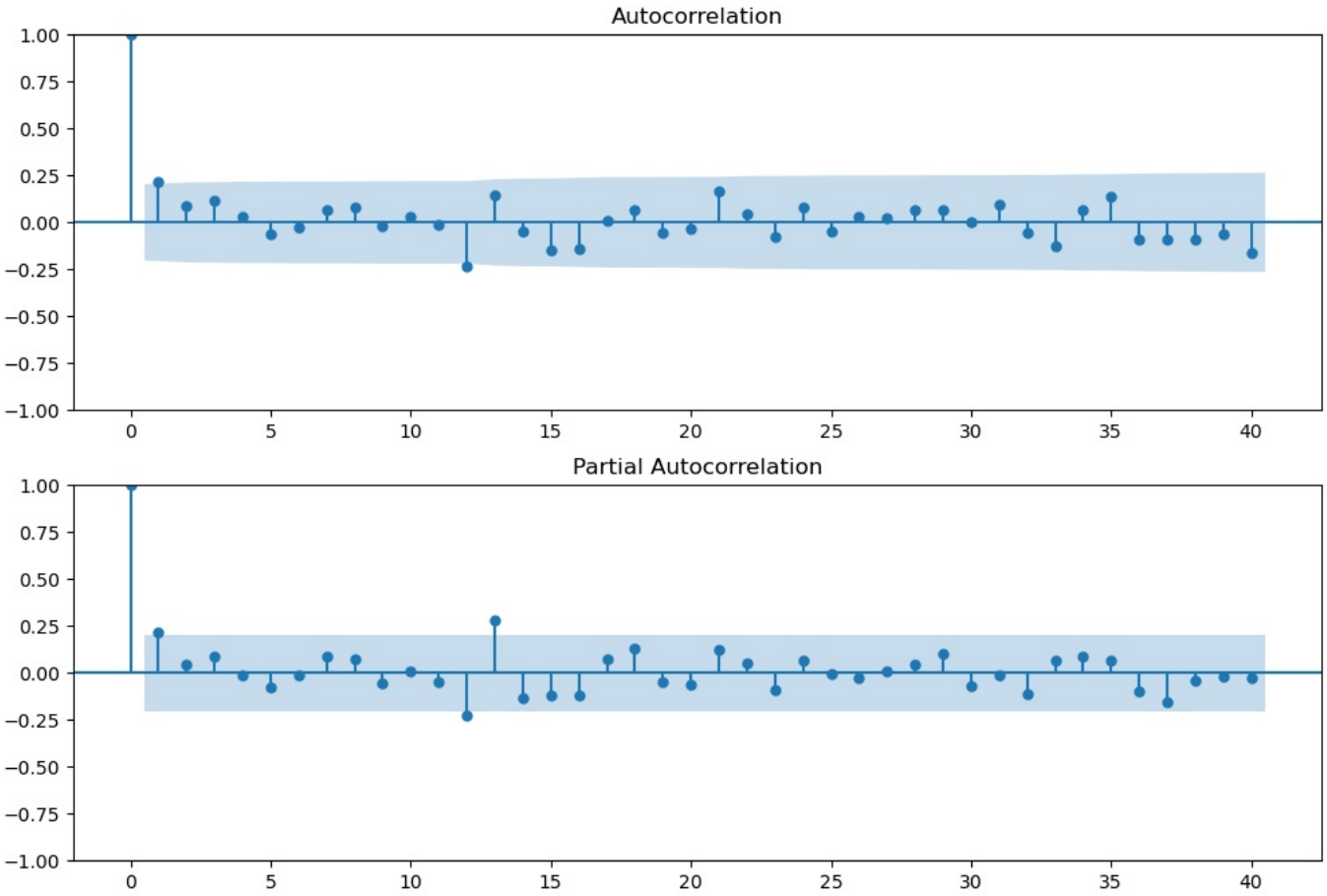
Identification of an MA model is often best done with the ACF rather than the PACF.

For an MA model, the theoretical PACF does not shut off, but instead tapers toward 0 in some manner. A clearer pattern for an MA model is in the ACF. The ACF will have non-zero autocorrelations only at lags involved in the model.

p,d,q p AR model lags d differencing q MA lags

In [38]:
```
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
```

In [42]:
```
import matplotlib.pyplot as plt
import statsmodels.api as sm

fig = plt.figure(figsize=(12,8))

ax1 = fig.add_subplot(211)
sm.graphics.tsa.plot_acf(
    df['Seasonal First Difference'].dropna(),
    lags=40,
    ax=ax1
)

ax2 = fig.add_subplot(212)
sm.graphics.tsa.plot_pacf(
    df['Seasonal First Difference'].dropna(),
    lags=40,
    ax=ax2
)

plt.show()
```



In [53]:
```
# For non-seasonal data
#p=1, d=1, q=0 or 1
from statsmodels.tsa.arima_model import ARIMA
```

In [54]:
```
df.index = pd.to_datetime(df.index)
```

In [55]:
```
df = df.asfreq('MS')
```

In [56]:
```
from statsmodels.tsa.arima.model import ARIMA

model = ARIMA(df['Sales'], order=(1,1,1))
model_fit = model.fit()
```

In [57]:
```
model_fit.summary()
```

Out[57]:

SARIMAX Results

Dep. Variable:	Sales	**No. Observations:**	105
Model:	ARIMA(1, 1, 1)	**Log Likelihood**	-952.814
Date:	Thu, 18 Dec 2025	**AIC**	1911.627
Time:	21:23:55	**BIC**	1919.560
Sample:	01-01-1964	**HQIC**	1914.841
	- 09-01-1972		
Covariance Type:	opg		

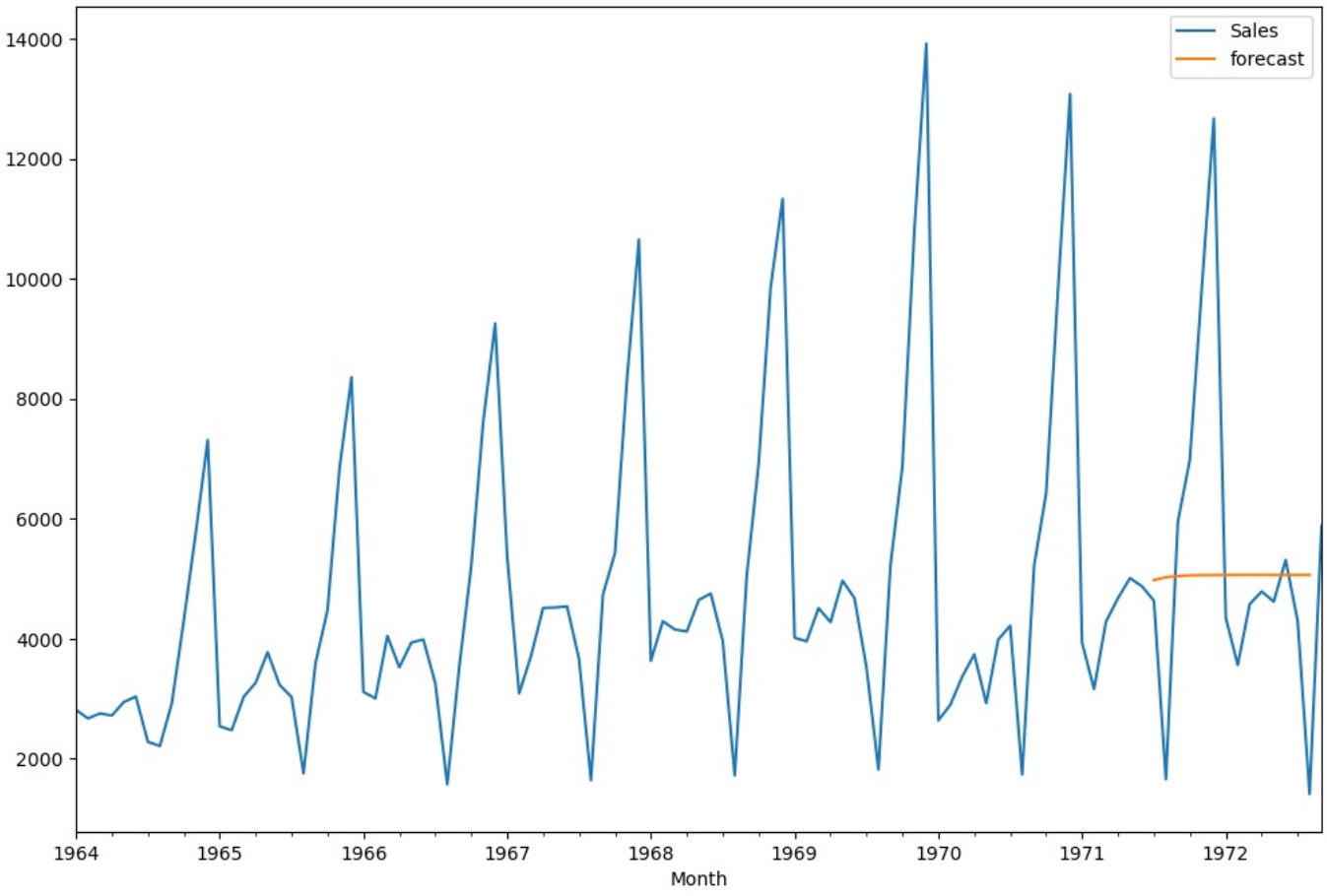
	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.4545	0.114	3.998	0.000	0.232	0.677
ma.L1	-0.9666	0.056	-17.305	0.000	-1.076	-0.857
sigma2	5.226e+06	6.17e+05	8.473	0.000	4.02e+06	6.43e+06

Ljung-Box (L1) (Q):	0.91	**Jarque-Bera (JB):**	2.59
Prob(Q):	0.34	**Prob(JB):**	0.27
Heteroskedasticity (H):	3.40	**Skew:**	0.05
Prob(H) (two-sided):	0.00	**Kurtosis:**	3.77

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [60]:
```
import matplotlib.pyplot as plt

df['forecast'] = model_fit.predict(start=90, end=103, dynamic=True)

df[['Sales', 'forecast']].plot(figsize=(12,8))
plt.show()
```

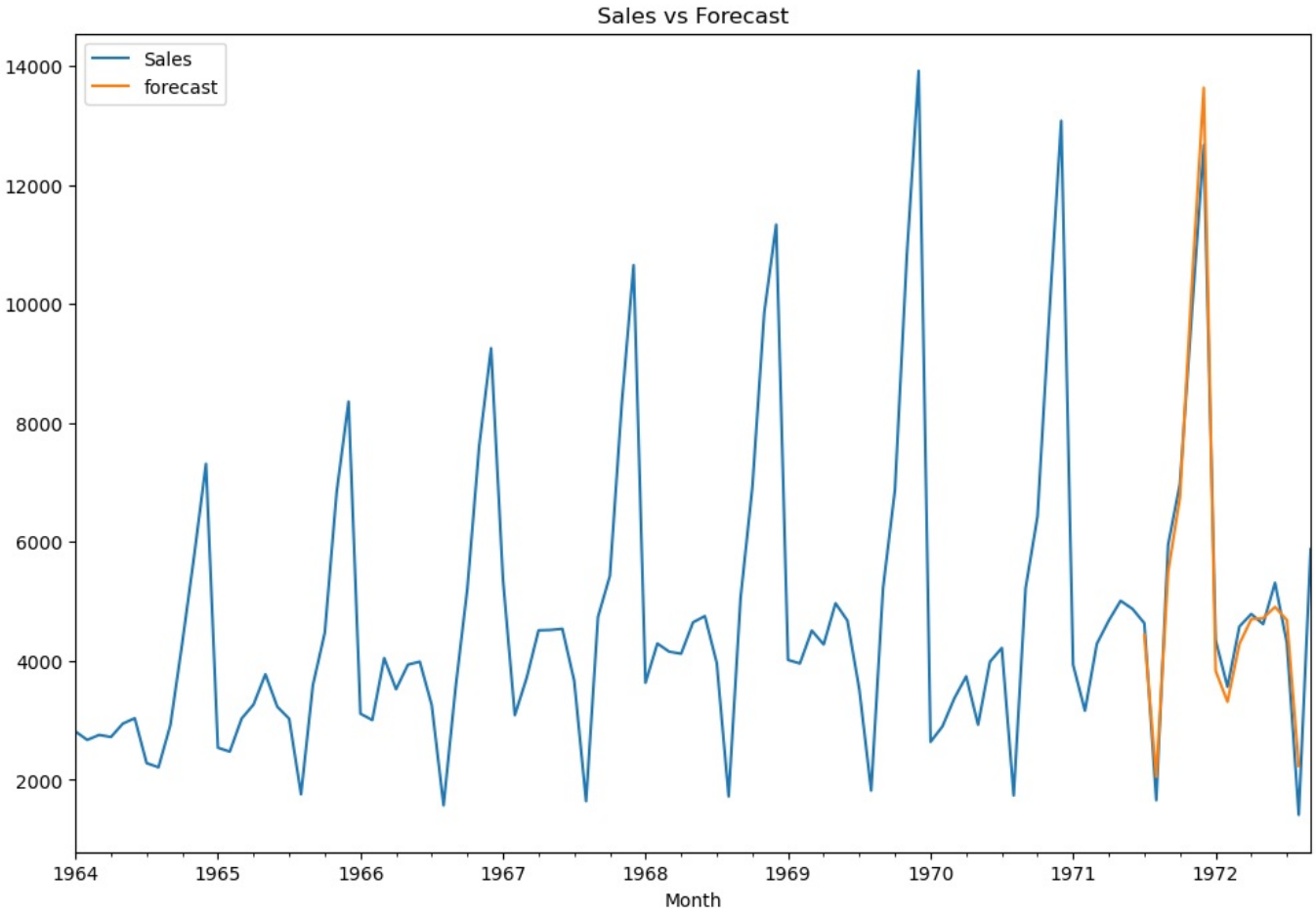


In [61]:
```
import statsmodels.api as sm
```

In [62]:
```
model=sm.tsa.statespace.SARIMAX(df['Sales'],order=(1, 1, 1),seasonal_order=(1,1,1,12))
results=model.fit()
```

In [64]:
```
%matplotlib inline
```

In [66]:
```
import matplotlib.pyplot as plt

df['forecast'] = results.predict(start=90, end=103, dynamic=True)

ax = df[['Sales', 'forecast']].plot(figsize=(12,8))
ax.set_title("Sales vs Forecast")

plt.show()
```



In [67]:
```
from pandas.tseries.offsets import DateOffset
future_dates=[df.index[-1]+ DateOffset(months=x)for x in range(0,24)]
```

In [68]:
```
future_datest_df=pd.DataFrame(index=future_dates[1:],columns=df.columns)
```

In [69]:
```
future_datest_df.tail()
```

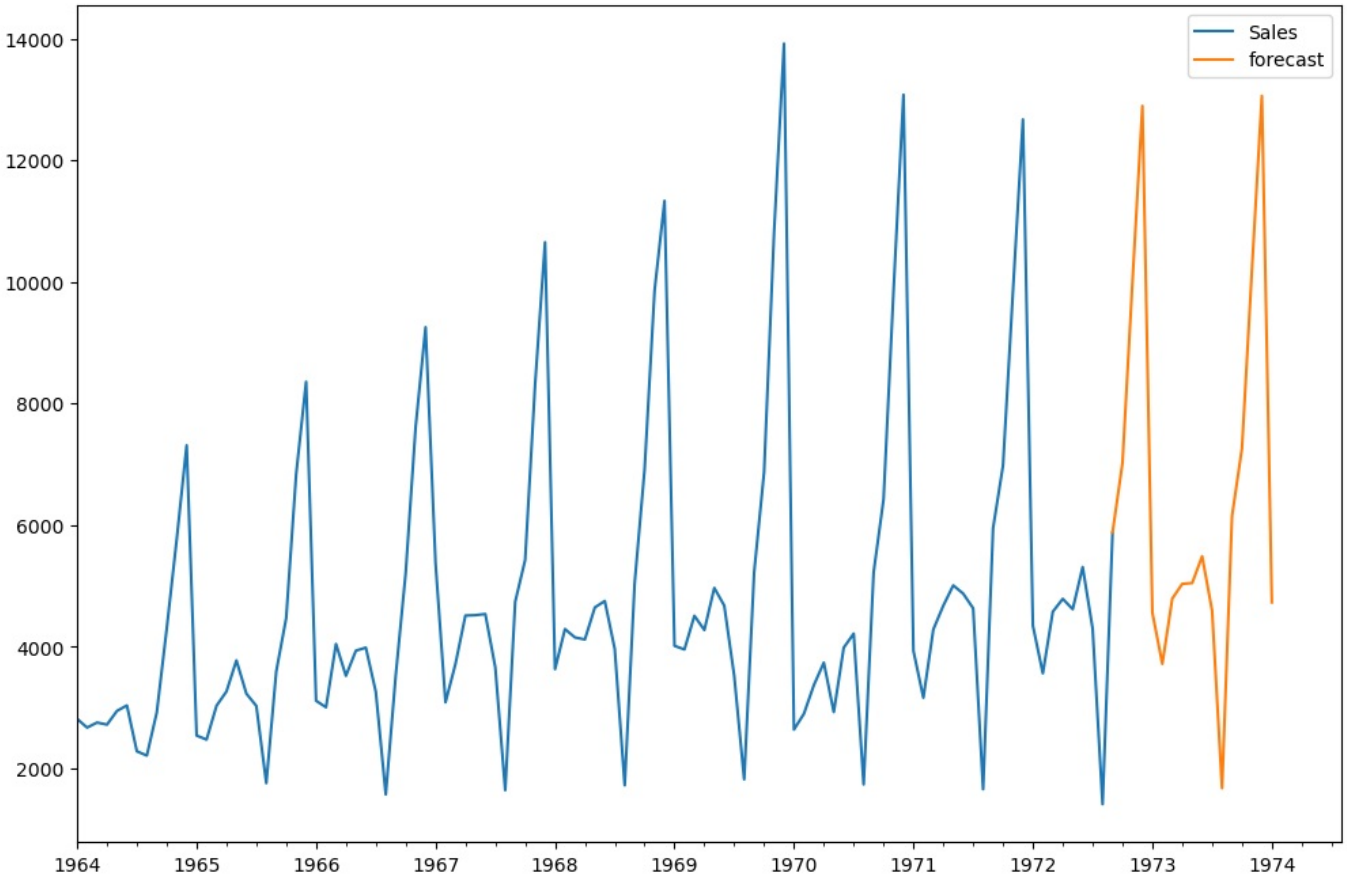
Out[69]:

	Sales	Sales First Difference	Seasonal First Difference	forecast
1974-04-01	NaN	NaN	NaN	NaN
1974-05-01	NaN	NaN	NaN	NaN
1974-06-01	NaN	NaN	NaN	NaN
1974-07-01	NaN	NaN	NaN	NaN
1974-08-01	NaN	NaN	NaN	NaN

In [75]:
```
future_df=pd.concat([df,future_datest_df])
```

```
C:\Users\asusl\AppData\Local\Temp\ipykernel_17052\1723908026.py:1: FutureWarning: The behavior of DataFrame conc
atenation with empty or all-NA entries is deprecated. In a future version, this will no longer exclude empty or
all-NA columns when determining the result dtypes. To retain the old behavior, exclude the relevant entries befo
re the concat operation.
  future_df=pd.concat([df,future_datest_df])
```

In [76]:
```
import matplotlib.pyplot as plt

future_df['forecast'] = results.predict(start=104, end=120, dynamic=True)

future_df[['Sales','forecast']].plot(figsize=(12,8))
plt.show()
```



Top 5 Project Insights

1. Sales data shows strong seasonality and trend, making time series forecasting models like ARIMA and SARIMAX suitable for prediction.

2. Stationarity checking and differencing were crucial, as ARIMA models perform best when the time series has stable statistical properties.

3. ACF and PACF analysis helped identify optimal AR and MA parameters, improving model structure and prediction accuracy.

4. SARIMAX performed better than basic ARIMA by capturing seasonal patterns present in the sales data.

5. Forecast results followed the overall sales trend, proving that time series models are effective baseline approaches for future sales planning.

Project Conclusion

In this project, time series forecasting was performed to predict future sales using ARIMA and SARIMAX models.

The data was analyzed for trends, seasonality, and stationarity, followed by proper preprocessing and model selection.

ARIMA successfully captured the overall sales pattern, while SARIMAX provided improved performance by handling seasonality.

The results demonstrate that time series models are effective for short-term forecasting based on historical data.

Future accuracy can be enhanced by incorporating external factors and more advanced forecasting techniques.

The data showed clear seasonality and trend. Proper preprocessing and parameter selection improved forecasting, and SARIMAX provided better results by handling seasonality compared to ARIMA

In [ ]:

THE END

THANK YOU!

We appreciate your time and attention!