Time Series Analysis

- Data collected in a time sequence (daily, weekly, monthly).

Examples:

- Stock prices 📈 (daily)
- Retail sales 🛒 (monthly)
- Temperature 🌡️ (hourly)

Helps understand patterns (trend, seasonality) and predict future values.

Import & Setup

```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima.model import ARIMA
from pandas.plotting import autocorrelation_plot
```

Date Handling

```
 Convert to datetime
df['Order_Date'] = pd.to_datetime(df['Order_Date'])

# Set as index for time-based operations
df = df.set_index('Order_Date')

# Extract features
df['Year'] = df.index.year
df['Month'] = df.index.month
df['Weekday'] = df.index.day_name()

print(df.head())
```

Makes data time-aware so you can group/slice easily.

	Product_ID	Sale_Date	Sales_Rep	Region	Sales_Amount	Quantity_Sold	Product_Category	Unit_Cost	Unit_Price	Customer_Type	Discount	Payment_Method
0	1052	2023-02-03	Bob	North	5053.97	18	Furniture	152.75	267.22	Returning	0.09	Cash
1	1093	2023-04-21	Bob	West	4384.02	17	Furniture	3816.39	4209.44	Returning	0.11	Cash
2	1015	2023-09-21	David	South	4631.23	30	Food	261.56	371.40	Returning	0.20	Bank Transfer
3	1072	2023-08-24	Bob	South	2167.94	39	Clothing	4330.03	4467.75	New	0.02	Credit Card
4	1061	2023-03-24	Charlie	East	3750.20	13	Electronics	637.37	692.71	New	0.08	Credit Card

Filtering & Slicing (Simple)

```
df['2023']              # All 2023 data
df['2023-07']             # July 2023
df['2023-07-01':'2023-07-15']  # First 15 days of July
```

Very useful in business reports.

```
            Product_ID   Sale_Date Sales_Rep Region  Sales_Amount  \
Order_Date
2023-02-03        1052  2023-02-03       Bob  North       5053.97
2023-04-21        1093  2023-04-21       Bob   West       4384.02
2023-09-21        1015  2023-09-21     David  South       4631.23
2023-08-24        1072  2023-08-24       Bob  South       2167.94
2023-03-24        1061  2023-03-24   Charlie   East       3750.20

            Quantity_Sold Product_Category  Unit_Cost  Unit_Price  \
Order_Date
2023-02-03             18        Furniture     152.75      267.22
2023-04-21             17        Furniture    3816.39     4209.44
2023-09-21             30             Food     261.56      371.40
2023-08-24             39         Clothing    4330.03     4467.75
2023-03-24             13      Electronics     637.37      692.71

           Customer_Type  Discount Payment_Method Sales_Channel  \
Order_Date
2023-02-03     Returning      0.09           Cash        Online
2023-04-21     Returning      0.11           Cash        Retail
2023-09-21     Returning      0.20  Bank Transfer        Retail
2023-08-24           New      0.02    Credit Card        Retail
2023-03-24           New      0.08    Credit Card        Online

           Region_and_Sales_Rep  Year  Month   Weekday
Order_Date
2023-02-03            North-Bob  2023      2    Friday
2023-04-21             West-Bob  2023      4    Friday
2023-09-21          South-David  2023      9  Thursday
2023-08-24            South-Bob  2023      8  Thursday
2023-03-24         East-Charlie  2023      3    Friday
```

Resampling & Aggregation

```
# Daily sales
daily_sales = df['Sales'].resample('D').sum()

# Weekly profit
weekly_profit = df['Profit'].resample('W').mean()

# Monthly sales trend
monthly_sales = df['Sales'].resample('M').sum()
```

Converts irregular data → meaningful summaries.

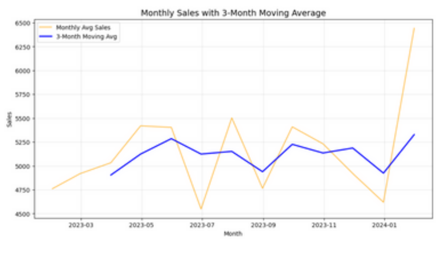
```
der_Date
23-01-01   -159489.45666
23-01-08    -84268.10217
23-01-15   -153331.53100
23-01-22   -130210.11833
23-01-29    -96882.90708
23-02-05    -97360.57615
23-02-12   -117043.98178
23-02-19   -122710.23800
23-02-26   -140670.20538
23-03-05   -122007.71040
23-03-12   -110404.57500
23-03-19   -144755.77700
23-03-26    -91785.06210
23-04-02   -144229.44809
23-04-09   -106290.28500
23-04-16   -131194.30500
23-04-23   -155851.29529
23-04-30   -130951.28000
23-05-07   -183020.90300
23-05-14   -132528.61681
23-05-21   -187257.70230
23-05-28   -111243.35550
23-06-04    -99668.58375
```

Rolling Statistics (Trend Smoothing)

```
# 7-day moving average
df['Sales_MA7'] = df['Sales'].rolling(7).mean()

df[['Sales','Sales_MA7']].plot(figsize=(10,5), title="Sales with 7-Day Moving Avg")
plt.show()
```

Removes noise, highlights underlying trend.



Exponential Moving Average (Faster Reaction)

```
df['Sales_EMA'] = df['Sales'].ewm(span=12, adjust=False).mean()
```

Unlike simple MA, EMA reacts quicker to recent changes.

```
Order_Date
2023-02-03    5053.970000
2023-04-21    4950.900769
2023-09-21    4901.720651
2023-08-24    4481.139012
2023-03-24    4368.686857
                  ...
2023-04-15    4497.510844
2023-09-07    4531.179945
2023-04-27    5007.875338
2023-12-20    4488.120670
2023-08-16    4555.168260
Name: Sales_EMA, Length: 1000, dtype: float64
```

Growth Analysis (MoM & YoY)

```
# Month-over-Month %
df['MoM'] = df['Sales'].pct_change(periods=1) * 100# Year-over-Year %
df['YoY'] = df['Sales'].pct_change(periods=12) * 100
```

Key business KPI for tracking growth.

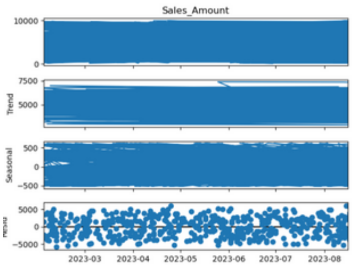
```
Order_Date
2023-02-03          NaN
2023-04-21   -13.255916
2023-09-21     5.638889
2023-08-24   -53.188678
2023-03-24    72.984492
                ...
2023-04-15   -47.942156
2023-09-07    -0.370098
2023-04-27    61.770942
2023-12-20   -78.643066
2023-08-16   202.179850
Name: MoM, Length: 1000, dtype: float64
```

Seasonal Decomposition (Trend + Seasonality)

```
result = seasonal_decompose(df['Sales'], model='additive', period=12)
result.plot()
plt.show()
```

Breaks series into:

- Trend → long-term growth
- Seasonality → repeating cycles (holidays, seasons)
- Residual → noise



Cumulative Sum

```
df['Cumulative_Sales'] = df['Sales'].cumsum()
```

Tracks total growth over time.

```
Order_Date
2023-04-15    5000365.77
2023-09-07    5005082.13
2023-04-27    5012711.83
2023-12-20    5014341.30
2023-08-16    5019265.23
Name: Cumulative_Sales, dtype: float64
```

Lag Features (Feature Engineering for ML)

```
df['Sales_Lag1'] = df['Sales'].shift(1)  # yesterday's sales
df['Sales_Lag7'] = df['Sales'].shift(7)  # last week's sales
```

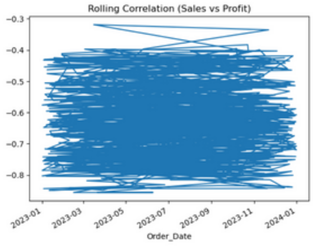
Helps models learn from past values.

```
Order_Date
2023-02-03        NaN
2023-04-21        NaN
2023-09-21        NaN
2023-08-24        NaN
2023-03-24        NaN
                ...
2023-04-15    4912.69
2023-09-07    9215.32
2023-04-27     496.59
2023-12-20    2985.46
2023-08-16    2154.66
Name: Sales_Lag7, Length: 1000, dtype: float64
```

Rolling Correlation (Sales vs Profit)

```
df['Sales'].rolling(30).corr(df['Profit']).plot(title="Rolling Correlation (Sales vs
Profit)")
plt.show()
```

Measures relationship strength over time.



Stationarity Test (Needed for Forecasting)

```
from statsmodels.tsa.stattools import adfuller

result = adfuller(df['Sales'])
print("ADF Statistic:", result[0])
print("p-value:", result[1])
```

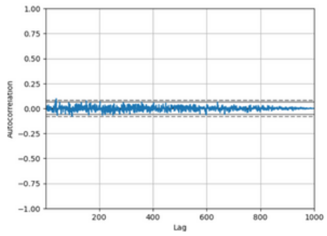
If p < 0.05 → stationary (good for ARIMA).

```
ADF Statistic: -30.416398670618502
p-value: 0.0
```

Autocorrelation (Check Repetition)

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

Shows seasonal lags (e.g., sales spike every 7 days).



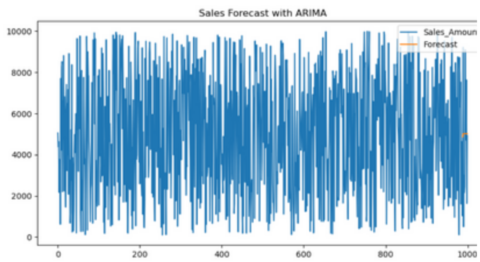
Forecasting with ARIMA (Basic)

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

df['Forecast'] = fit.predict(start=len(df)-12, end=len(df)+6, dynamic=True)

df[['Sales','Forecast']].plot(figsize=(10,5), title="Sales Forecast with ARIMA")
plt.show()
```

Predicts future values based on past trends.



Train-Test Forecast Validation

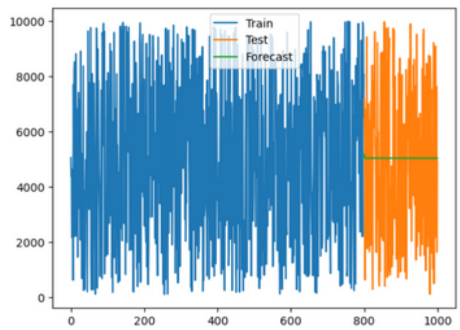
```
train = df['Sales'][:int(0.8*len(df))]
test = df['Sales'][int(0.8*len(df)):]

model = ARIMA(train, order=(1,1,1))
fit = model.fit()

forecast = fit.predict(start=len(train), end=len(df)-1, dynamic=False)

plt.plot(train, label="Train")
plt.plot(test, label="Test")
plt.plot(forecast, label="Forecast")
plt.legend()
plt.show()
```

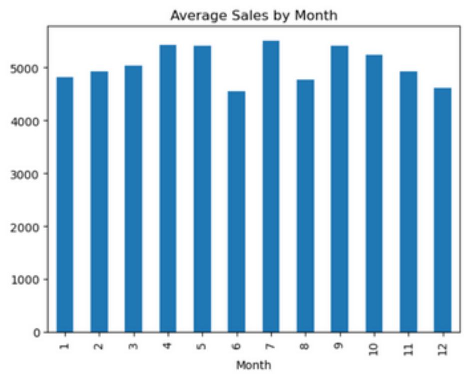
Compares predicted vs actual.



Seasonal Plot (Month vs Avg Sales)

```
df.groupby(df.index.month)['Sales'].mean().plot(kind='bar')
plt.title("Average Sales by Month")
plt.show()
```

Identifies best & worst sales months.



Heatmap of Seasonality

```
import seaborn as sns

pivot = df.pivot_table(values="Sales", index=df.index.year, columns=df.index.month, aggfunc='sum')
sns.heatmap(pivot, annot=True, fmt=".0f", cmap="YlGnBu")
plt.title("Year-Month Sales Heatmap")
plt.show()
```

Visualizes seasonal patterns across years.