Python Time Series Analysis Cheatsheet

1. Importing Essential Libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima.model import ARIMA
```

2. Creating and Loading Time Series Data

Creating Datetime Objects

```
# Single date
date = pd.to_datetime("2024-01-01")

# Date range
dates = pd.date_range(start="2024-01-01", end="2024-01-10", freq='D')
```

Reading Time Series Data

```
# Read CSV with date as index
data = pd.read_csv("stock_prices.csv",
                   parse_dates=["Date"],
                   index_col="Date")
```

3. Time Series Indexing and Slicing

```
# Access specific date
single_day = data.loc["2024-01-01"]

# Date range slice
date_range = data.loc["2024-01-01":"2024-01-05"]
```

4. Visualization Techniques

Basic Line Plot

```
data["Close"].plot(
    figsize=(10, 6),
    title="Stock Prices",
    xlabel="Date",
    ylabel="Price ($)"
)
plt.show()
```

Rolling Statistics Visualization

```
# Rolling Mean
data["Rolling_Mean"] = data["Close"].rolling(window=20).mean()
data[["Close", "Rolling_Mean"]].plot(
    figsize=(10, 6),
    title="Rolling Mean"
)
plt.show()
```

5. Time Series Decomposition

```
# Decompose time series
result = seasonal_decompose(
    data["Close"],
    model="additive",
    period=30
)
result.plot()
plt.show()
```

6. Stationarity Analysis

Augmented Dickey-Fuller Test

```
from statsmodels.tsa.stattools import adfuller

result = adfuller(data["Close"])
print("ADF Statistic:", result[0])
print("p-value:", result[1])
```

Differencing

```
# Make series stationary
data["Diff"] = data["Close"].diff()
```

7. Correlation Analysis

Autocorrelation Function (ACF)

```
from statsmodels.graphics.tsaplots import plot_acf

plot_acf(data["Close"].dropna(), lags=20)
plt.show()
```

Partial Autocorrelation Function (PACF)

```
from statsmodels.graphics.tsaplots import plot_pacf

plot_pacf(data["Close"].dropna(), lags=20)
plt.show()
```

8. Forecasting with ARIMA

Build and Fit ARIMA Model

```
# ARIMA(p,d,q) model
model = ARIMA(data["Close"], order=(1, 1, 1))
result = model.fit()
print(result.summary())

# Forecast next 10 periods
forecast = result.forecast(steps=10)
print(forecast)
```

9. Practical Analysis Techniques

Volatility Analysis

```
data["Volatility"] = data["Close"].pct_change().rolling(window=20).std()
data[["Close", "Volatility"]].plot(
    figsize=(10, 6),
    secondary_y="Volatility"
)
plt.show()
```

Event Window Analysis

```
event_date = "2024-01-15"
window = 5
event_window = data.loc[
    event_date - pd.Timedelta(days=window):
    event_date + pd.Timedelta(days=window)
]
event_window["Close"].plot(title="Event Window Analysis")
plt.show()
```

Key Concepts

- **Time Series**: Sequence of data points indexed in time order
- **Stationarity**: Constant mean and variance over time
- **Decomposition**: Separating series into trend, seasonality, and residual

- **ARIMA**: AutoRegressive Integrated Moving Average model for forecasting

Common Parameters

- **Rolling Window**: Number of periods for rolling calculations
- **ARIMA Order (p,d,q)**:
  - p: Autoregressive terms
  - d: Differencing order
  - q: Moving average terms