Time Series Analysis with Pandas

Time series data consists of data points indexed in time order, often used in fields like finance, weather forecasting, and economics. Pandas provides extensive tools for manipulating, analyzing, and visualizing time series data.

---

1. Importing Libraries

python
Copy code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

---

2. Creating Time Series Data

Using `pd.date_range`

python
Copy code

```
# Create a time range
dates = pd.date_range(start='2023-01-01', periods=10, freq='D')

# Create a DataFrame with time series data
data = pd.DataFrame({
    'Date': dates,
    'Value': np.random.randint(10, 100, len(dates))
})
data.set_index('Date', inplace=True)  # Set Date as the index
print(data)
```

Reading Time Series Data

python
Copy code

```
# Example: Reading time series data from a CSV
# CSV format:
# Date,Value
# 2023-01-01,45
# 2023-01-02,50
```

```
data = pd.read_csv('time_series.csv', parse_dates=['Date'],
index_col='Date')
print(data.head())
```

---

3. Exploring Time Series Data

Basic Exploration

python
Copy code

```
# First few rows
print(data.head())

# Summary statistics
print(data.describe())

# Check for missing values
print(data.isnull().sum())
```

Resampling Data

python
Copy code

```
# Resample to monthly frequency (sum of values)
monthly_data = data.resample('M').sum()
print(monthly_data)

# Resample to weekly frequency (mean of values)
weekly_data = data.resample('W').mean()
print(weekly_data)
```

---

4. Plotting Time Series Data

python
Copy code

```
# Plot the data
data.plot(figsize=(10, 5), title='Time Series Data', ylabel='Value',
xlabel='Date')
plt.grid()
plt.show()
```

---

5. Time Series Operations

Shifting Data

- **Lagging**: Shifting data backward.
- **Leading**: Shifting data forward.

python
Copy code
```
# Shift data by one time step
data['Lag1'] = data['Value'].shift(1)

# Shift data forward by one time step
data['Lead1'] = data['Value'].shift(-1)

print(data.head())
```

Rolling Statistics

python
Copy code
```
# Calculate a 3-day rolling mean
data['RollingMean'] = data['Value'].rolling(window=3).mean()

# Calculate a 7-day rolling sum
data['RollingSum'] = data['Value'].rolling(window=7).sum()

print(data.head())
```

---

6. Handling Missing Data

python
Copy code
```
# Fill missing values with the previous value
data.fillna(method='ffill', inplace=True)

# Fill missing values with the next value
data.fillna(method='bfill', inplace=True)

# Drop rows with missing values
data.dropna(inplace=True)
```

---

7. Time Indexing and Slicing

python
Copy code

```
# Select data for a specific date
print(data.loc['2023-01-05'])

# Select data for a range of dates
print(data.loc['2023-01-03':'2023-01-07'])

# Filter by month or year
print(data['2023-01'])  # January 2023
print(data['2023'])     # Entire year 2023
```

---

8. Time Series Decomposition

Use statsmodels for decomposing time series into trend, seasonality, and residuals.

python
Copy code

```
from statsmodels.tsa.seasonal import seasonal_decompose

result = seasonal_decompose(data['Value'], model='additive',
period=7)
result.plot()
plt.show()
```

---

9. Autocorrelation and Partial Autocorrelation

Autocorrelation helps identify repeating patterns in time series.

python
Copy code

```
from pandas.plotting import autocorrelation_plot

# Plot autocorrelation
autocorrelation_plot(data['Value'])
plt.show()
```

---

10. Forecasting with Pandas

Naive Forecast

python
Copy code

```
# Use the last observed value as the forecast
data['NaiveForecast'] = data['Value'].shift(1)
```

Simple Moving Average Forecast

python
Copy code

```
# Use a rolling mean for forecasting
data['SMAForecast'] = data['Value'].rolling(window=3).mean()
```

---

11. Advanced Forecasting Techniques

For more advanced forecasting, use models like ARIMA, SARIMA, or machine learning approaches.

Example: ARIMA Model

python
Copy code

```
from statsmodels.tsa.arima.model import ARIMA

# Fit ARIMA model
model = ARIMA(data['Value'], order=(1, 1, 1))
model_fit = model.fit()

# Forecast
forecast = model_fit.forecast(steps=10)
print(forecast)
```

---

12. Saving Processed Data

Save the processed data for future analysis.

python
Copy code

```
data.to_csv('processed_time_series.csv')
```