Introduction to Financial Timeseries

Kannan Singaravelu, CQF

June 2023

1 Financial Data Preprocessing

A time series is a series of data points indexed in time order. Financial Data such as equity, commodity, and forex price series observed at equally spaced points in time are an example of such a series. It is a sequence of data points observed at regular time intervals and depending on the frequency of observations, a time series may typically be in ticks, seconds, minutes, hourly, daily, weekly, monthly, quarterly and annual.

The first step towards any data analysis would be to parse the raw data that involves extracting the data from the source and then cleaning and filling the missing data if any. While data comes in many forms, Python makes it easy to read time-series data using useful packages.

In this session, we will retrieve and store both end-of-day and intraday data using some of the popular python packages. These libraries aim to keep the API simple and make it easier to access historical data. Further, we will see how to read data from traditional data sources stored locally.

1.1 Load Libraries

We'll now import the required libraries that we'll use in this example. Refer `requirements.txt` for list of packages.

[ ]:
```python
# Import data manipulation libraries
import numpy as np
import pandas as pd

# Import yahoo finance library
import yfinance as yf

# Import cufflinks for visualization
import cufflinks as cf
# cf.set_config_file(offline=True, theme='pearl')

# Ignore warnings - optional
import warnings
warnings.filterwarnings('ignore')
```

1.2 Docstring or Signature

Getting information on function attributes and outputs.

```
[ ]: # help(yf.download)
     # yf.download?
```

1.3 Data Retrieval

Retrieving EOD, Intraday, Options data

1.3.1 Retrieving end-of-day data for single security

We'll retrieve historical data from yahoo finance using `yfinance` library

Example 1

```
[ ]: # Fetch the data by specifying the number of period
     df1 = yf.download('SPY', period='5d', progress=False)

     # Display the first five rows of the dataframe to check the results.
     df1
```

Example 2

```
[ ]: # Fetch data by specifying the the start and end dates
     df2 = yf.download('SPY', start='2022-06-01', end='2022-06-30', progress=False)

     # Display the first five rows of the dataframe to check the results.
     df2.head()
```

Example 3

```
[ ]: # Fetch data for year to date (YTD)
     df3 = yf.download('SPY', period='ytd', progress=False)

     # Display the last five rows of the dataframe to check the results.
     df3.tail()
```

1.3.2 Retrieving data for multiple securities

We'll retrieve historical price data of five Nasdaq-listed stocks from yahoo finance.

Example 4

```
[ ]: # Specify stocks
     # https://en.wikipedia.org/wiki/Dow_Jones_Industrial_Average
     dow_stocks = ['UNH', 'GS', 'HD', 'AMGN', 'MCD']
```

```
[ ]: # Fetch data for multiple stocks at once
     df4 = yf.download(dow_stocks, period='ytd', progress=False)['Adj Close']

     # Display dataframe
     df4.tail()
```

1.3.3 Retrieving multiple fields for multiple securities

We'll now retrieve multiple fields from yahoo finance.

Example 5

```
[ ]: # Fetch data for multiple fields using comprehension
     ohlcv = {symbol: yf.download(symbol, period='250d', progress=False) for symbol␣
      ↪in dow_stocks}
```

```
[ ]: ohlcv
```

```
[ ]: # Display NVDA stock data
     ohlcv['GS'].head()
```

```
[ ]: # Display GS adjusted close data
     ohlcv['GS']['Adj Close']
```

1.3.4 Retrieving intraday data

We'll now retrieve intraday data from yahoo finance.

Example 6

```
[ ]: # Retrieve intraday data for last five days
     df6 = yf.download('SPY', period='5d', interval='1m', progress=False)

     # Display dataframe
     df6
```

1.3.5 Retrieving option chain

We'll now retrieve option chain for SPY for March 2022 expiration from yahoo finance and filter the output to display the first seven columns.

Example 7

```
[ ]: spy = yf.Ticker('SPY')
```

```
[ ]: spy.option_chain
```

```
[ ]: # Get SPY option chain for Sept 30th expiration
     # https://finance.yahoo.com/quote/SPY/options?date=1680220800
     # spy = yf.Ticker('SPY')
     options = spy.option_chain('2023-06-30')
     options
```

```
[ ]: # Filter calls for strike above 390
     df7 = options.calls[options.calls['strike']>390]

     # Check the filtered output
     df7.iloc[:,:7]
```

1.3.6 Retrieving Hypertext Markup Language (HTML)

We'll now retrieve India's benchmark index NIFTY50 Index data from Wikipedia.

Example 8

[ ]:
```python
# Read data from wikipedia
nifty50 = pd.read_html('https://en.wikipedia.org/wiki/NIFTY_50')[2].Symbol.
 ↪to_list()

# Read five symbols
nifty50[:5]
```

1.4 Data Storage

Export files and store it in a local drive

1.4.1 Storing OHLCV data in Excel File

[ ]:
```python
# Dataframe to Excel
from pandas import ExcelWriter
```

[ ]:
```python
# Storing the fetched data in a separate sheet for each security
writer = ExcelWriter('data/stocks.xlsx')

# df.to_excel() - this is list comprehension - df.to_excel()
[pd.DataFrame(ohlcv[symbol].tz_localize(None)).to_excel(writer, symbol) for␣
 ↪symbol in dow_stocks]

# save file
writer.save()
```

1.4.2 Storing OHLCV data in a CSV File

[ ]:
```python
# Save ohlcv data for each securities in stockname.csv format
[pd.DataFrame(ohlcv[symbol]).to_csv('data/'+symbol+'.csv') for symbol in␣
 ↪dow_stocks]
print('*** file saved ***')
```

1.5 Data Loading

Importing files stored in the local drive

1.5.1 Reading Microsfot Excel File

We'll now read the Excel file stored locally using Pandas

[ ]:
```python
# Reading the fetched data in a spreadsheet
gs = pd.read_excel('data/stocks.xlsx', sheet_name='GS', index_col=0,␣
 ↪parse_dates=True)

# Display the last five rows of the data frame to check the results
gs.tail()
```

1.5.2 Reading CSV File

We'll now read the csv file stored locally using Pandas

```
[ ]: # Read CSV file
     gs = pd.read_csv('data/GS.csv', index_col=0, parse_dates=True)

     # Display the last five rows of the data frame to check the results
     gs.tail()
```

1.6 Interactive Visualization of Time Series

We use `cufflinks` for interactive visualization. It is one of the most feature rich third-party wrapper around Plotly by Santos Jorge. It binds the power of `plotly` with the flexibility of `pandas` for easy plotting.

When you import cufflinks library, all pandas data frames and series objects have a new method `.iplot()` attached to them which is similar to pandas' built-in `.plot()` method.

1.6.1 Plotting Line Chart

Next, we'll plot the time series data in the line format.

```
[ ]: df3['Close'][-30:].iplot(kind='line', title='Line Chart')
```

1.6.2 Plotting OHLC Data

Next, we'll plot the time series data in ohlc format.

```
[ ]: df3[-30:].iplot(kind='ohlc', title='Bar Chart')
```

1.6.3 Plotting Candlestick

Next, we'll plot an interactive candlestick chart.

```
[ ]: df3[-30:].iplot(kind='candle', title='Candle Chart')
```

1.6.4 Plotting Selected Stocks

Next, we'll compare the GS & HD data that we fetched from Yahoo Finance.

```
[ ]: # Use secondary axis
     df4[['GS', 'HD']].iplot(secondary_y='HD')
```

1.6.5 Plotting using Subplots

```
[ ]: # Use subplots
     df4[['GS', 'HD']].iplot(subplots=True)
```

1.6.6 Normalized Plot

```
[ ]: df4.normalize().iplot()
```

1.6.7 Visualising Return Series

We'll now plot historical daily log normal return series using just one line of code.

```
[ ]: # Calculating Log Normal Returns
     # Use numpy log function to derive log normal returns
     daily_returns = np.log(df4).diff().dropna()

     # Display the last five rows of the data frame to check the output
     daily_returns.head()
```

Log Normal Distribution A normal distribution is the most common and widely used distribution in statistics. It is popularly referred as a "bell curve" or "Gaussian curve". Financial time series though random in short term, follows a log normal distribution on a longer time frame.

Now that we have derived the daily log returns, we will plot this return distribution and check whether the stock returns follows log normality.

```
[ ]: # Plot log normal distribution of returns
     daily_returns.iplot(kind='histogram', title = 'Histogram of Daily Returns',␣
      ↪subplots=True)
```

Plotting Annual Returns

```
[ ]: # Plot Mean Annual Returns
     (daily_returns.mean()*252*100).iplot(kind='bar')
```

Plotting Annualized Volatility

```
[ ]: # Plot Mean Annualized Volatility
     (daily_returns.std()*np.sqrt(252)*100).iplot(kind='bar')
```

Plotting Correlation Correlation defines the similarity between two random variables. As an example we will check correlation between our Nasdaq listed stocks.

```
[ ]: # Plot correlation of returns
     daily_returns.corr().iplot(kind='heatmap', title="Correlation Matrix",␣
      ↪colorscale="Blues")
```

1.7 References

- YFinance Documentation
- Cufflinks Documentation
- Numpy Documentation
- Pandas Documentation
- Python Resources

June 2023, Certificate in Quantitative Finance.