Introduction to Financial Time Series

Kannan Singaravelu, CQF

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.

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

# 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)
```

1.2 Docstring or Signature

Getting information on function attributes and outputs.

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

2 Data Retrieval

Retrieving EOD, Intraday, Options data

2.1 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('AAPL', 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('AAPL', start='2024-01-01', end='2024-01-31', progress=False)

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

[ ]:
```
# Display last five rows
df2.tail()
```

Example 3

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

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

2.2 Retrieving data for multiple securities

We'll retrieve historical price data of five 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()
```

2.3 Retrieving multiple fields for multiple securities

We'll now retrieve multiple fields from yahoo finance.

Example 5

[ ]:
```
# Group the stocks
df5 = yf.download(dow_stocks, start="2017-01-01", end="2017-01-30",␣
↪group_by="ticker", progress=False)
```

[ ]:
```
# Display GS stock data
df5['GS']
```

2.4 Retrieving intraday data

We'll now retrieve intraday data from yahoo finance.

Example 6

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

# Display dataframe
df6
```

2.5 Retrieving option chain

We'll now retrieve option chain for SPY for March 2024 expiration from yahoo finance and filter the output to display the first seven columns.

Example 7

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

[ ]:
```
# Use option chain method
spy.option_chain
```

[ ]:
```
# Get SPY option chain for March 28th expiration
# https://finance.yahoo.com/quote/SPY240328C00475000?p=SPY240328C00475000
options = spy.option_chain('2024-03-28')
options
```

[ ]:
```
# Filter calls for strike above 490
df7 = options.calls[options.calls['strike']>490]

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

2.6 Retrieving Hypertext Markup Language (HTML)

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

Example 8

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

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

3 Database Storage & Retrieval

Let's store the data in the database and load back for manipulation.

3.1 Storing Nifty50 data in SQLite Database

[ ]:
```
# Import & create database
import sqlalchemy
engine = sqlalchemy.create_engine('sqlite:///India')
```

[ ]:
```
# Fetch data from yahoo using list comprehension
data = [yf.download(symbol+'.NS', period='250d', progress=False).reset_index()␣
 ↪for symbol in nifty50]

# save it to database
for frame, symbol in zip(data, nifty50):
    frame.to_sql(symbol, engine, if_exists='replace', index=False)
```

3.2 Reading Sqlite Database

We'll now read the ohlcv data stored locally in the database using Pandas

[ ]:
```
# Query from database
result = pd.read_sql_query('SELECT * FROM  "ADANIENT" WHERE DATE >␣
 ↪"2024-01-01"', engine, index_col='Date')

# Display the results
result.tail()
```

4 Data Manipulation

Next, we'll see some methods used in data wrangling. This is critical if you work on financial data or time series.

4.1 Filter & Query Methods

We'll now see some examples of subset selection using Panda's filter and query methods.

[ ]:
```
# Query from database
df = pd.read_sql_query('SELECT * FROM  "TITAN"', engine, index_col='Date')

# Convert to datetime format
df.index = pd.to_datetime(df.index)
```

[ ]:
```
# Get first few days of data
df.first('3D')
```

[ ]:
```
# Get last few days of data
df.last('3D')
```

[ ]:
```
# Filter based on column
df.filter(['Close'])
```

[ ]:
```
# Filter based on row or index
df.filter(like='2023-10-23', axis=0)
```

[ ]:
```
# Query for a specific condition. Ex: Close price > 3800
df.query('Close>3800')
```

[ ]:
```
# Query for condition where the difference between High and Low is greater than␣
↪100
df.query('High-Low>100')
```

[ ]:
```
# Query for a multiple conditions. Ex: Close > 3700 and High-Low > 150
df.query('Close>3700 & High-Low>150')
```

[ ]:
```
# Query for condition where Close>Open price
df.query('Close>Open')
```

[ ]:
```
# Query for condition where Open price is equal to Low price of the day
df.query('Open==Low')
```

4.2 Resampling data

We'll now resample the frequency of time series.

[ ]:
```
# https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.
↪html#offset-aliases
# Resampling to derive weekly values from daily time series
df_weekly = df.resample('W').last()

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

[ ]:
```
# Resampling to a specific day of the week: Thu
df_weekly_thu = df.resample('W-THU').last()

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

[ ]:
```
# Resampling to derive monthly values from daily time series
df_monthly = df.resample('M').last()

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

5 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.

5.1 Plotting Line Chart

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

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

5.2 Plotting OHLC Data

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

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

5.3 Plotting Candlestick

Next, we'll plot an interactive candlestick chart.

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

5.4 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')
```

5.5 Plotting using Subplots

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

5.6 Normalized Plot

[ ]:
```
# Normalize plot
df4.normalize().iplot()
```

5.7 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()
```

5.7.1 Plotting Annual Returns

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

5.7.2 Plotting Annualized Volatility

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

5.7.3 Plotting Rolling Returns

[ ]:
```
# To calculate 5 days rolling returns, simply sum daily returns for 5 days as␣
↪log returns are additive
rolling_return = daily_returns.rolling(5).sum().dropna()

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

[ ]:
```
# Plot Rolling Returns
rolling_return.iplot(title='5-Days Rolling Returns')
```

5.8 Time Series Statistics

Statistics is a branch of mathematics that deals with collecting, interpreting, organization and interpretation of data. The two main categories of statistics are descriptive statistics and inferential statistics.

Descriptive statistics help us to understand the data in a meaningful way and is an important part of data analysis. While inferential statistics allows us to infer trends and derive conclusion from it.

[ ]:
```
# Analysing the daily returns data
daily_returns.describe().T
```

5.8.1 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)
```

5.8.2 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")
```

6 References

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

Python Labs by Kannan Singaravelu.