Exercises

Level 8: Tabular Data

All the exercises in this level should be done using Anaconda/JupyterLab. Each exercise should be in its own Notebook, with subparts as individual cells. You should submit the actual Notebooks (.ipynb) files from Jupyter.

Note that many of these exercises are deliberately open-ended to allow you to inject your own creativity into the data that you choose to analyze.

We will expand on these exercises in level 9 (data visualization).

8.1: NumPy/SciPy

Math Functions

1) Create a program that does the following:
   - **a.** Create an ndarray containing 1,000,000 random numbers.
   - **b.** Calculate the mean, median, mode, and standard deviation of the array. Compare the time spent running mean and standard deviation, vs. the previously implemented versions (from level 1). Is there a significant speedup? Why?
   - **c.** Calculate the 10,20,30,…100 quantiles of the array.

2) Provide examples of the many NumPy math functions and compare against their built-in Python counterparts.

Matrices

You may use NumPy or SciPy (or both!) for these problems.

3) Create two ndarrays and calculate the inner, outer, dot products, and the correlation matrix.

4) Create two 5x5 matrices of numbers:
   - **a.** Multiply both matrices together.
   - **b.** Compute the eigenvalues of the matrices.
   - **c.** Calculate the correlation matrix for each of the two matrices.

5) Solve the following system of linear equations (hint: use linalg.solve):

$$\begin{gathered} 7x - 3y + 4z = 15 \\ -3x - 11y + 6z = 98 \\ 2x + y - 4z = -11 \end{gathered}$$

6) In this exercise, we will create a matrix of random numbers that have a predefined correlation. In practice, this is really useful when running Monte Carlo simulations over data that has a known correlation. For example, if we know a set of underlying Stock prices are correlated and we wish to simulate how a portfolio of their Option prices react: We can price each Option using a Monte Carlo simulation individually; however, doing so individually will not give us insight into the portfolio value. Instead, we can create an array of random, correlated Stock prices. There are two ways to do this (you should attempt both), with the following steps:

   a. Pick 10 stocks from any stock exchange and get their last 10 closing prices – put these into an ndarray, and calculate the covariance matrix.

   b. Use numpy.random's multivariate_normal to create a correlated array of random numbers, based on the above covariance matrix.

   c. Use np.random's standard_normal function to create an array of uncorrelated random numbers. Use the following formula to create a correlated array of random numbers:

```
L = linalg.cholesky(covarianceMatrix)
corr = DotProduct(L, uncorr)
```

Note that in practice, for options pricing Monte Carlo simulations, we'd use a lognormal distribution, but for simplicity here, we use a normal distribution.

Financial Functions

7) We already used numpy-financial in the midterm project; specifically, the IRR function. We had directly implemented the following formulas in the project. NumPy-Financial gives us some canned functionality to do the same. Create a program that does the following:

   a. Computes the present value for a given rate, number of periods, and monthly payment.

   b. Calculates the monthly payment for a given rate, number of periods, and present value.

   c. Calculates the number of periods for a given rate, payment, and present value.

   d. Calculates the rate for a given number of periods, monthly payment, and present value.

8.2: Tabular Data

Loading Data

1) Use the yfinance library to download two years of stock data for your favorite stock ticker into a DataFrame and display it.

2) Use the sportsreference library to download some of your favorite sports-team data and display it.

3) Use the quandl library to download the same data as exercise 1.

4) Use the pandas-datareader library to load the same data as exercise 1 above. Save the DataFrame into a .csv file. Save this CSV file for the exercises below.

DataFrames

5) Use the data from exercise 4 above – load the CSV into a DataFrame. Perform the following operations:

- **a.** Display the first five and last five rows.
- **b.** Display a table containing only dates that have a volume of at least 10,000,000.
- **c.** Display the total volume.
- **d.** Display the average daily volume.
- **e.** Add a column containing the daily 'return', using the following formula:

$$r = \frac{(S_1 - S_0)}{S_0}$$

$S_0 = Previous\ Stock\ Close$
$S_1 = Stock\ Close$

We will ignore dividends here for simplicity.

- **f.** Calculate the average daily return.
- **g.** Calculate the volume-weighted average daily return.
- **h.** Add a column containing the 1-week (5-day) moving average volume.
- **i.** Add a column containing the 1-week (5-day) moving average daily return.

6) Load two years of stock data for every ticker in the current S&P 500. Perform the following:

- **a.** 'Stack' the table to get it into row form, and demonstrate 'unstack' to get it back into the original form.
- **b.** Melt and unmelt the table (using pivot), as shown in the lecture.
- **c.** Demonstrate usage of pivot_table.
- **d.** Create a derived table (a.k.a a 'summary table'), containing the total volume (in millions) per year, of each ticker.
- **e.** Create a derived table containing the total volume (in millions) of each ticker.
- **f.** Add a column to the original table containing the daily return per ticker (using the above formula).
- **g.** Add a column to the original table containing the 1-week (5-day) moving average daily return, per ticker.
- **h.** Add a column to the original table containing the rolling cumulative sum of volume per ticker.

7) Use the same data as exercise 6, but using a fresh notebook. Perform the following:

- **a.** Download a table (cross-sectional) containing reference data for each ticker. Reference data should include the full company name and any other useful column data you can find.
- **b.** Create a 'transactions' table, which has three columns: **Date**, **Ticker**, **Amount**, and **BuySell**. Add a bunch of transactions (at least ten) of your favorite stocks – both buys and sells (all long positions, no shorts, no day trading). Transactions should span a period of two years. Keep them self-consistent but somewhat arbitrary.
- **c.** Create a function that takes a date and returns a (cross-sectional) table containing a snapshot of your portfolio at that point in time (**Ticker**, **Position**).
  - i. The function should join the table with your stock reference data, to enrich with the full company name, etc.
  - ii. The function should join the table with the stock data table, to retrieve the open price, volume, and return for the given date, for each ticker.
- **d.** Create a 'scaffold' table, which is the concatenation of the above function, for every business date over the two-year period. Be sure to add a 'Date' column.
- **e.** 'PnL' (Profit and Loss) is your total portfolio profit/loss at any given point in time. For a given stock, the daily marked-to-market PnL is simply the **return*position**. Add a column containing the daily PnL for each ticker.
- **f.** Add a column containing the running cumulative PnL per ticker.
- **g.** Add a column containing the running cumulative PnL for your entire portfolio.
- **h.** Create a summary table of your average daily PnL, total PnL, and overall yield per ticker.
  - i. 'Yield' is PnL/Position.
  - ii. You should exclude days when you had no position in that stock, from the average.
- **i.** Display your total PnL at the end of the two years.

We will use the results from this exercise for visualization in level 9.