Project 8: Python Basics

Instructions

Project 8 is due on November 13th by 10:00 pm. This is a hard deadline, so no exceptions. You must push your local repository back to GitHub before the deadline. Your repository must contain:

- The Python code you used to complete the project;
- A notebook file named `main.ipynb` that generates all required plots;
- A `report.pdf` file with your answers to the project questions. The report must also contain an Appendix with the code used to solve the project;
- All plots in the report must be self-contained. Self-contained means that a reader who only sees your figure (image and caption, but not the surrounding text) can understand what you are plotting. This translates to all plots having axis titles, correct units on the axis, and a caption that summarizes what is plotted.

This project makes use of stock data. Refer to the Data page for instructions on how to download the data and which files to download (requires Duke login). You must complete all exercises for both of your stocks using the data at the 5-minutes sampling frequency, unless stated otherwise.

You can obtain the repository for this project by clicking **on this link**.

Questions

The purpose of this project is to practice the basics of Python in the context of financial econometrics.

Exercise 1 - Realized Variance

On this exercise you will use prices stored in a list to compute variance estimates.

A.

Consider a list of prices for a given stock at some given day:

```
prices = [
    142.19,
    142.27,
    142.29,
    142.62,
```

```
    142.70,
    142.64,
    142.61,
    142.62,
    142.63,
    142.67,
    142.62,
    142.50,
    142.45,
    142.57,
    142.54,
    142.32,
    142.17,
    142.03,
    141.85,
    141.30,
    141.36,
    141.01,
    140.78,
    140.96,
    141.16,
    141.38,
]
```

If stocks sampled every 5-minutes lead to about 78 observations per day, what would be the sampling frequency of the prices above? (Use `len` to answer this question.)

B.

The built-in module `math` provides common mathematical functions, like logarithm. You can see what functions are available using `dir`:

```
import math
print("Functions available in the math module:")
for fun in dir(math):
    if not fun.startswith('__'):
        print(fun)
```

Always read a function's documentation before using it:

```
help(math.log)
```

The full documentation of the `math` module is available on the Mathematical Functions documentation page.

Compute the log-returns from the prices above. Hint: use a for-loop over indices, and start at index 1 instead of 0 (use `range`).

C.

Compute the average return and the standard deviation of the returns. Hint: Read the documentation of the `statistics` module.

D.

Compute the realized variance from the returns. Report the annualized value. Hint: Define a function for doing the computation.

E. (Optional)

Consider the two comparisons below:

```
print(0.1 + 0.2 == 0.3)
print(sum([0.1, 0.2]) == 0.3)
```

Why do they evaluate to `False`? Is it a Python-specific issue? Could this issue affect the computation of the realized variance?

F. (Optional)

What is the difference between the function `sum` and the function `math.fsum`? Use `sum` and `math.fsum` to compute the realized variance from the returns. Compare the results.

G.

Compute the bipower variance from the returns. Report the annualized value. Hint: Look at the function `abs` and at `math.pi`.

Exercise 2 - Reading Prices from a File

In this exercise you will read from one of your `csv` files and parse the data. You will then compute the realized variance estimators for all days in the sample.

A.

Consider one of your 5-minute data files. Read the data and split it in 3 lists: one for dates, one for times, and one for prices.

Hints:

- Use `open` to read the entire file into memory, you will obtain a single very long string.
- Then use the string method `split` to split the long string into lines.
- For each line, use `split` again to get the data for each column.
- Append the data to the right list, and consider converting the value to a more appropriate type (like `int` or `float`).
- Finally, put everything in a function.

B.

Determine programatically how many observations are available each day and how many days are available in the sample. Report the values.

Hint: Compare the first date to all dates you have. After you figure out a solution, write it into a function.

C.

Transform the list of all prices into a list of lists, where each sublist contains all prices for a given day. For example, if we had two prices per day:

```
p = [1, 2, 3, 4, 5, 6]
```

Then the transformed list should be:

```
transformed_p = [[1, 2], [3, 4], [5, 6]]
```

Hint: Use the information about `N` and `T` and the list method `append`.

D.

Obtain a list of lists for returns, where each sublist contains the returns for a day.

E.

Compute the realized variance for each day in the sample.

Hint: Use the list of returns split by day and the functions you built in the first exercise.

F.

Report the average and median of the realized variances. Remember to annualize the values.

Exercise 3 - Plotting

In this exercise you will plot the realized variance estimates for all days. You will also adjust the x-axis to display the correct values (dates).

A.

The package `matplotlib` (not a built-in, but comes pre-installed with Anaconda) can be used to create nice plots in Python. It works a similar way to Matlab's plotting capabilities. The main plotting functions are available in the subpackage named `pyplot`. We can immport this package via:

```
import matplotlib.pyplot
```

However, we do not want to type `matplotlib.pyplot` everytime we need to plot something. To simplify our life, we will give `matplotlib.pyplot` an alias, i.e., we will refer it by a shorter name. This can be done by:

```
import matplotlib.pyplot as plt
```

Now, `plt` is an alias for `matplotlib.pyplot`.

Read the documentation for `plt.plot` and plot the realized variance for all days (do not worry about having the right values in the x-axis).

Hint: For the x-axis you can use a list of integers (consider `range` to create such list).

B.

Matplotlib offers several ways of customizing a plot. For example, it is possible to add a grid, change colors, change the line width, add Latex to labels, and so on. An easy way of changing the overall style of plots is with the method `plt.style.context`. This method can be used with a context manager (similar to the way we used a context manager for opening files), and it takes as input the name of the style you want to use. For example:

```
with plt.style.context("ggplot"):
    # run plotting commands here
    # plt.plot(x, y)
```

The full list of styles and their names is available on the style sheets reference page.

Use the style `"ggplot"` to plot the realized variance.

C.

To modify the x-axis so that it correctly displays timestamps, we need to use the first two columns of the data set to obtain the timestamps for each return. Python has a built-in module named `datetime` which supplies a class for manipulating dates and times. We will use the class `datetime` of the module `datetime` to create objects that represent the date (year, month and day) and time (hour, minute and second) of each return.

We can create a `datetime.datetime` instance in several different ways:

```
from datetime import datetime
# Create a date
print(datetime(2017, 11, 20))          # November 20th, 2017
# Create a date with time
print(datetime(2017, 11, 20, 9, 27))  # 9:27 AM (24 hour clock)
print(datetime(2017, 11, 20, 23, 55))  # 11:55 PM (24 hour clock)
# We can also create a datetime from a string
print(datetime.strptime('20171120 0927', '%Y%m%d %H%M'))
```

The string `'%Y%m%d %H%M'` is a formatting string so that the method `strptime` can understand what the number `'20171120 0927'` means. You must read this part of the documentation page to understand what is the meaning of the format codes, like `%Y`.

Why did I type `'0927'` instead of `'927'` when I called the method `strptime` above? What do the other formatting codes I used mean?

D.

Use the first two columns of your stock file (stored in the lists `dates` and `times`) to generate a new list of lists. Each sublist should have the date and time information as a string for each of the prices on a given day.

To combine the date and time information in a string, you can simply join the two strings. For example:

```
date = 20070103
time = 935
dt = str(date) + ' ' + '0' + str(time)
```

Alternatively, you can combine the integers firsrt, and then transform them into a string.

Hint: To go over two values from different lists at the same time in a for-loop, use the built-in function `zip`. Try it out on a small list to see how it works, and remember to read its documentation. Then, use a for-loop with tuple unpacking.

E.

Given the list of lists containing the date and time information for each price observation, use the function `datetime` from the module also named `datetime` to obtain a datetime object representing the information. Do this for all dates and times for every single day in the sample.

Hint: Use the method `strptime` from `datetime`.

F.

We can use the datetime objects to correctly display the dates in plots. Plot the realized variance and use the datetimes in the x-axis.

Hint: Since we have one realized variance per day, you need to use a single datatime object per day.

G.

Read the documentation on `matplotlib.pyplot` and add a title and labels to the realized variance plot.