CS917 Coursework 1

Deadline: Monday week 6 (4 November) at 12.00 noon. Please read the entire sheet before starting on Part A.

Background

Through lectures and exercise sheets you have gained some useful experience of the multi-paradigm programming language Python. In this coursework we would like you to use this knowledge to solve a number of real-world problems based on the analysis of company stock data.

The dataset that you will be using is taken from the Financial Times Stock Exchange 100 Index (the FTSE 100). This lists the 100 companies registered on the London Stock Exchange with the highest market capitalisation. The FTSE 100 is seen as a gauge of the prosperity of businesses regulated by UK company law and there is much interest (and speculation) in the volatility of this index as the UK moves through various BREXIT scenarios.

Data

The data that you need for this coursework is in a single CSV file called **ftse100.csv**, which can be found on the module website. Within the CSV file there is data on the value of stock prices for each company on the FTSE 100. Each row of the data file is formatted as follows:

1. ‘date’: the day of the stock information, format dd/mm/yyyy
2. ‘time’: the time of the stock information, format hh:mm
3. ‘code’: The FTSE100 company code
4. ‘name’: Full name of the company
5. ‘currency’: The currency the company is valued in
6. ‘price’: The valuation of the stock at the date and time
7. ‘diff’: The price difference between the current and last recorded value
8. ‘per_diff’: The percentage difference between the current and last recorded value

The data has been collected in 15 minute intervals from Monday 14 October to Friday 18 October.

Part A (25 Marks)

In this first part you are required to define 5 functions.

The first of these functions is

```
daily_movement(data, code, date) -> float
```

which given the data, a stock code, and a date (in the format dd/mm/yyyy) will return a positive or negative floating point number (to 2 decimal places) that is the price movement of that stock on that date.

The second function

```
daily_high(data, code, date) -> float
```

which given the data, a stock code, and a date will return a positive floating point number (to 2 decimal places) that is the highest price for that stock on the given date.

The third function

```
daily_low(data, code, date) -> float
```

provides the equivalent functionality to that above but now returns the lowest price for that stock on the given date.

The fourth function

```
daily_avg(data, code, date) -> float
```

provides the average price (to 2 decimal places) for a given stock on a given date.

Finally, the fifth function

```
percentage_change(data, code, date) -> float
```

should return the percentage change for a particular company code on a particular date.

The code skeleton below can be found on the module website and you should add your solutions to a copy of this file. If you use this code skeleton then marking should be very easy using our ready-made test harness.

Skeleton files for all parts are provided. Below is an example of the skeleton code provided for part A:

```
"""
```

```
    Part A
    Please provide definitions for the following functions
"""

# daily_movement(data, code, date) -> float
# daily_high(data, code, date) -> float
# daily_low(data, code, date) -> float
# daily_avg(data, code, date) -> float
# percentage_change(data, code, date) -> float

# Replace the body of this main function for your testing purposes
if __name__ == "__main__":
    # Start the program

    # Example variable initialization
    # data is always the ftse100.csv read in using a DictReader
    data = []
    with open("ftse100.csv", "r") as f:
        reader = csv.DictReader(f)
        data = [r for r in reader]
    # code is always a string value
    code = "III"
    # date is always a string formatted 'dd/mm/yyyy'
    date = "14/10/2019"

    # access individual rows from data using list indices
    first_row = data[0]
    # to access row values, use relevant column heading in csv
    print(f"code = {first_row['code']}")
    print(f"price = {first_row['price']}")
    print(f"date = {first_row['date']}")

    pass
```

Details of the marking scheme that we will use can be found below.

Part B (25 Marks)

In this exercise you are first expected to create a portfolio. This is simply a list of company codes which a particular trader is interested in. You must ensure that each code in the portfolio is a valid FTSE 100 company (of course), and a portfolio must contain at least 1 and at most 100 companies.

```
    create_portfolio() -> [code]
```

To implement this function, you will need to ask for inputs from the user using the input functions from previous labs. Each input should ask for a single company code. When all the company codes have been entered for this portfolio, input 'EXIT' to exit the input loop, and then return the portfolio.

Next you are required to write a function that will find the best *x* investments in a portfolio for a particular period:

```
best_investments(data,portfolio,x,start_date,end_date) -> [code]
```

The function should take the following parameters: The FTSE 100 data; The portfolio in question (which is a list of company `code`s); the number of investments that the trader is interested in (this must be a number between 1 and the number of companies in the portfolio inclusive); and a start and end date, each in the format dd/mm/yyyy.

There are a lot of opportunities for error here: *x* must be less than or equal to the portfolio size; the start date must be less than the end date etc. In **all** cases where it is not possible to return a valid answer, or where you think the function should return an exception, your function should return an empty list of codes, e.g. `[ ]`.

In a similar fashion, now define the function:

```
worst_investments(data,portfolio,x,start_date,end_date) -> [code]
```

The parameters are to be defined in the same way as above, and the function should return `[ ]` in all scenarios where it is not possible to calculate a valid answer.

A code skeleton for Part B can also be found on the module website, so please add your solutions to this. Details of the marking scheme that we will use can be found below.

Part C (25 Marks)

As Data Analysts, we should be capable of interrogating and understanding our data. Many of us, however, find it quite difficult to understand hundreds or possibly thousands of lines of csv. To help with this one might want to visualise the data in a graph.

In this exercise, you will be utilising the `matplotlib.pyplot` library to visualise the trends of the stock prices in the data file. You will again find a code skeleton on the module website which you should use when developing your solutions.

You need to implement two functions for Part C.The first function is

```
plot_company(data, code, start_date, end_date)
```

Which, given the code of a particular company and a start and end date (in the format dd/mm/yyyy), will output a line graph of the stock over the time period from the start date to the end date. The function should not return a value, but should instead output the plot to a file called **plot1.png**.

The second function

```
plot_portfolio(data, portfolio, start_date, end_date)
```

is similar to the first function, except that it should plot lines for each stock in the portfolio. Due to the large variance in share prices, it may not be feasible to plot some companies on the same graph. Luckily for us the `matplotlib` package comes with a subplot function, allowing us to plot multiple graphs in the same figure.

Using the subplots function, create and save a plot containing multiple graphs. To make coding easier, the maximum number of subplots expected within one figure will be 6.

The function should not return a value, but should instead output the plot to a file called **plot2.png**.

Each graph generated by `plot_company` and `plot_portfolio` should have a title, a legend of all of the `code`s present in the graph, suitable scales for both axes, and labelled axes with units identified.

Part D (25 Marks)

In this exercise you are required to create a new `Company` class. This class will encapsulate the code you created in Part A so that `Company` objects can be created. Each `Company` will have a `data` variable which will store entries from **ftse100.csv** related to this company. The specification for the new class is as follows:

`Company`:
`#Instance variables`

- `code`
- `name`
- `currency`
- `data`

`#Functions`

- `daily_movement(date) -> float`
- `daily_high(date) -> float`
- `daily_low(date) -> float`
- `daily_avg(date) -> float`
- `percentage_change(date) -> float`

If we want to meaningfully participate in the stock market, it is not enough to just analyse data, we also need to be able to predict future stock prices for a given company. To do this we will implement a linear regression model to identify the trend and predict the next day's stock prices.

There are various maths libraries in Python that might help you do this, but we would like you to implement this feature by hand. Define a function called `predict_next_average`, which takes an instance of the Company class, calculates the average `price` for the 5 days of data in **ftse100.csv** and uses this to predict what you think the average `price` will be for the next day of trading.

```
predict_next_average(company) -> float
```

For those not familiar with linear regression models, the algorithm to generate a simple linear model is available below. In this algorithm, ***m*** is the gradient of a straight line and ***b*** is the *y* intercept. Applying this model to our dataset, for our needs, you will need to assign ***x*** to the day (Monday is day 0, Tuesday is day 1 etc.) and ***y*** is the average `price`.

$$m = \frac{\sum_{i=1}^{n} (x_i - \overline{X})(y_i - \overline{Y})}{\sum_{i=1}^{n} (x_i - \overline{X})^2}$$

$$b = \overline{Y} - m\overline{X}$$

The resulting model should result in ***y***=***mx***+***b*** which will generate a straight line from which we can extrapolate the `price` of day 5 in our sequence (which in reality would be the following Monday, as the FTSE 100 is closed over the weekend).

For many analysis techniques, it is not enough to predict the next stock prices or averages. Instead we will want to classify companies based on how the stocks have evolved over the past 5 days. Next implement a function that will return a string classifier that will identify if a stock's value is 'increasing', 'decreasing', 'volatile' or 'other':

```
classify_trend(company) -> str
```

To do this, perform a linear regression on the `daily high` and `daily low` for a given company and determine whether the highs and lows are increasing or decreasing. You will most likely need to use the linear regression algorithm you implemented for predicting the average price, so it may be useful to make the regression its own function.

The classification system works as follows: If the **daily highs are increasing** and the **daily lows are decreasing**, this means that the stock prices have been fluctuating over the past 5 days so **assign 'volatile'** to your result string. If the **daily highs and daily lows are both increasing**, this likely means that the overall stock prices are increasing so **assign 'increasing'**. Likewise if the **daily highs and lows are both decreasing** then **assign 'decreasing'**. We currently only care about these 3 classifications so if a company shares **do not follow any of the above trends assign 'other'** to your result string.

The marking scheme for Part D can also be found below.

Coursework Submission and Marking

Deadline: Monday week 6 (4 November) at 12.00 noon. Coursework in the department is nearly always submitted through Tabula. The advantage of using this system is that you can be assured that the work has been submitted, a secure record of the work is kept, feedback is easy to distribute and, where appropriate, code can be automatically run and checked for correctness. Instructions on how to register on Tabular and the steps to follow to submit your work will be posted on the module webpage shortly. Please note the university requires that late penalties apply, so if you do submit your work late (by even 5 minutes!) you will be penalised.

You are required to submit four separate files for this coursework: **parta.py**, **partb.py**, **partc.py**, **partd.py**. Each of these files will be run on the FTSE 100 test data so that it can be checked for correctness. We will also judge each solution on coding style and how well you have made use of the various features of Python that we have covered in the lectures.

The marking scheme for the coursework is as follows:

Part A

The **parta.py** file will have the following functions tested:

```
1. daily_movement(data, code, date) -> float
2. daily_high(data, code, date) -> float
3. daily_low(data, code, date) -> float
4. daily_avg(data, code, date) -> float
5. percentage_change(data, code, date) -> float
```

Each function will be tested on four (`code`, `date`) combinations and checked against our own solutions. Thus twenty tests will be run in total for Part A and there are 20 possible marks available for these tests. In addition, marks will be available for coding style (2 marks) and how well you have made use of the various language features of Python (3 marks).

In your feedback for this exercise you will be told how many of the twenty tests your code passes, and also how many marks were awarded for coding style and use of language features.

Part B

The **partb.py** file will have the following functions tested:

1. `create_portfolio() -> [code]`
2. `best_investments(data,portfolio,x,start_date,end_date) -> [code]`
3. `worst_investments(data,portfolio,x,start_date,end_date) -> [code]`

All three functions will be tested to ensure they follow the specification. This includes scenarios where the functions should return `[]` as an answer. This means that your solution should be able to handle incorrect input (as outlined above) as well as input that will yield a result.

`create_portfolio` is worth 4 marks and will be subject to 4 tests which will test its ability to handle both correct and incorrect inputs.

`best_investments` and `worst_investments` will be subject to 8 tests each (i.e. 8 marks for each function with 1 mark per test). These tests will consist of both valid and invalid inputs. For these tests, the portfolios supplied will always be valid, however the value of `x`, and the `start_date` and `end_date` may be invalid and therefore require a `[]` output.

Like Part A, 2 marks will be assigned for coding style and 3 marks for the use of python language features.

Part C

For Part C, the code in your **partc.py** file will be run with 5 sets of test input. The `plot_company` function will be tested with two valid FTSE 100 company codes; the `plot_portfolio` function will be tested with three valid test portfolios. The resulting graphs for each of the five tests (in `plot1.png` and `plot2.png`) will be visually inspected and marks will be allocated for each of the following features:

- The correctness of the graph (1 mark)

- The graph has a title (1 mark)
- The *x* and *y* axis both have labels and units (1 mark)
- A legend to identify the data (1 mark)
- That appropriate scales are calculated for the *x* and *y* axis (1 mark)

Thus your code will be expected to generate 5 graphs, each of which is worth 5 marks.

Part D

The **partd.py** file will have the following functions tested:

1. `company.daily_movement(date) -> float`
2. `company.daily_high(date) -> float`
3. `company.daily_low(date) -> float`
4. `company.daily_avg(date) -> float`
5. `company.percentage_change(date) -> float`

The implementation of the Company class is worth 5 marks. Therefore each of these required class functions will be allocated one mark each.

We will then test the two functions:

1. `predict_next_average(company) -> float`
2. `classify_trend(company) -> str`

The `predict_next_average` function will be tested on 5 random companies from the FTSE 100. Each correct answer will result in 2 marks each; thus 10 marks in total.

The `classify_trend` function will be tested on 5 random companies, but not necessarily the same as those above, and each test will be worth 2 marks; total of 10 marks.

Please note that no marks will be assigned for solutions that use additional imported libraries not covered in the lectures to solve these questions.

Working Independently and Completion

It is important that you complete this coursework independently. Warwick has quite strict rules about this, and if a submitted solution is deemed to be a copy of another then there are harsh penalties. The Tabula system has built-in checking software that is quite sophisticated: if variable names are changed but the structure of the code is the same, then it will spot this; if you reorder the code, then it will spot this also; if you change the comments, then the checking software will know. So rather than trying to fool the system - which you will not be able to do - it is best just to complete the work by yourself. You do not need to do all the questions to get a decent mark, so just do what you can on your own … and then you will do fine.

This coursework is designed to have stretch goals. Part D in particular is quite complicated and we are not expecting everyone to do this. If you are able to get Parts A-C done well, then you will likely score between 65% and 75%, which is a good mark which you should be pleased with.

Good luck.