Jorgo Qirjaj (jq16)
COMP614 - Stock Prediction

Python code

The CodeSkulptor file can be accessed using the link here.

3.A. Building a Model

Referring to ***m*** as in the *length of the input list*, ***n*** as the *input number*.

Q3.A.i: In terms of **m** and/or **n**, how many pieces of data should comprise each state in the resultant markov chain?

In Markov Chains, each state is the most recent sequence of data points that the model uses to predict the following value.

If we consider the order of the model to be *n*, then each **state** would have ***n*** number of data in it.

If we take a look at our example, it becomes clearer:

```
data = [1, 2, 1, 1, 2, 1, 2, 3]
```

If n = 2, then each state would be (1, 2), followed by (2, 1), then (1, 1) and so on until the ending state (1, 2). As we can see, each **state** has a total of **n = 2** elements**.**

Q3.A.ii: In terms of ***m*** and/or ***n***, what is the maximum possible number of states that you could end up with in your markov chain (*i.e.*, in the case where there are no repeats)? Why? Please be as clear and specific as possible in your explanation. You should only count states for which there is a "next" value in the input list.

Following the same logic as we did in the previous HW, it is clear that the max number of possible states for a markov chain will always be equal to the **number of states that can be created with a slice of length *n*,** from the input data, as long as there is a next value after the slice. Simply put, that would be:

```
m - n = len(data) - order
```

Looking at indexes for our code, if the input list has length **m**, then the first state starts at index 0 and goes up to the last element which has index **m - n - 1.**

We can think of it like slicking a window of length n (2), in that input list (data). Every time we move one step forward, we create a state. Then, we eventually stop once there is no element following that state. That means the total number of states is **m - n.**

If we do this for the data list given, then we can see it more clearly. If **m = 8 and n = 2**:

```
data = [1, 2, 1, 1, 2, 1, 2, 3]
```

The states, each of length n = 2, are (1, 2), (2, 1), (1, 1), (1, 2), (2, 1), (1, 2). The ending "slice" (2, 3) cannot be a state because there aren't any values following it. That gives us a total of **6 states:**

```
m - n = 8 - 2 = 6
```

Q3.A.iii: Without using iteration, how could you extract a single state of the correct length? Provide a Python expression for extracting the sequence of elements that will be used to comprise the state starting at position ***i*** in the original list. Your expression should work for any arbitrary value of ***i*** that is greater than or equal to zero and less than your answer to Question 3.A.ii.

In Python, we can **extract the states by using slicing** (as mentioned earlier too). Each slice would have length **n**. In code, since the starting position is ***i,*** that would look like:

```
data = [i : i+n]
```

This will return a state of length **n,** and it will include all the elements from position **i** to **i + n - 1.** As mentioned in the question itself, the expression will work for any i greater than or equal to 0 or less than (m - n), and makes sure that the state will have a next value.

If we do this for the data list given and an order **n = 2**, then we would have:

```
i = 0 ->  data[0 : 2] -> (1, 2)
i = 1 ->  data[1 : 3] -> (2, 1)
i = 2 ->  data[2 : 4] -> (1, 1)
              ...
i = 5 ->  data[5 : 7] -> (1, 2)
```

3.B. Making Predictions

Referring to ***m*** as the *number of previous pieces of data*, ***n*** as the *order of the model,* and ***p*** as the future data to be predicted.

Q.3.B.i: Explain how you will use the inputs (model and last) to predict the first piece of future data. In particular, you should address how you will access the appropriate probabilities and how you will use those probabilities to select the future datum. Please be as clear and precise as possible!

Let's understand what the inputs **model** and **last** are.

- "**Model"** is the markov_chain we created on step 1. This will be a dictionary that will map the **states** (consisting of n values) to the **probabilities** of what the next value could be.
- "**Last**" is the list containing the most recent **n** states.

Firstly, we should use the "last" input to **identify our current state**. We need each state to be represented as a tuple so we convert it like so:

```
state = tuple(last)
```

Then, we should access the transition (trs) probabilities for that state. If the **state** from above exists in the **model** dictionary, then we get the inner dictionary, which is basically a possible **next** bin (0-3) and respectively the **probability** of occurring.

```
trs = model[state]
```

To make the prediction, I will randomly select the next bin based on the **probabilities** from the **model**. I'll use **random.random()** to generate a random number between 0 and 1, then **compare it with the probabilities of each possible next value**. I'll go through each possible next value and keep adding up their probabilities until the **total** is **greater** than the **random number**. The first value that passes this point will be my predicted next state.

This follows basic probability rules, and so the transitions with higher probabilities are more likely to be chosen.

Q.3.B.ii: Imagine that **p** is greater than or equal to 2. How will you construct the state that will be used to make the second prediction? As a part of your explanation, please include a concrete example (*i.e.*, sample inputs + the first prediction generated + the state used to make the second prediction).

If **p >= 2**, that means we're predicting more than 1 value. So **after generating the 1st prediction, we need to use that value as part of the next state.**

To build the next state:

- Take the current (n - 1) values from the previous state
- Append the new predicted value to the end
- The new list (still of length n) becomes the new state
- That new state is used to make the next prediction and so on

Let's take an example to see this more clearly. If the order n = 2, and the data bins are [1, 2, 3, 2, 1], then:

- Last two values (2, 1) is the current state
- Model predicts the first future value to be 2
- Keep the most recent (n - 1) value (1) and append predicted value (2) to that
- This will form the second prediction and **new state will be (1, 2)**

The **new state (1, 2)** is used to **check the transition probabilities** in the model and predict the next value and so on. So **for every prediction, we drop the oldest value and keep the newly predicted one**.

Q. 3.B.iii: In the event that a state is not found in the model, what Python function call could you make to generate a random integer between 0 and 3, inclusive? Please indicate not only the name of the function, but the input(s) that you will use.

If a **state can't be found** in the model, that simply means the **Markov chain does not have that as a value**, so we won't be able to predict the next value.

For this case, we can use Python's built in function **random.randrange()** to make a random prediction where each possible value (0 - 3) has the same chance of being chosen. In code, that would look like this:

```
random.randrange(0, 4)
```

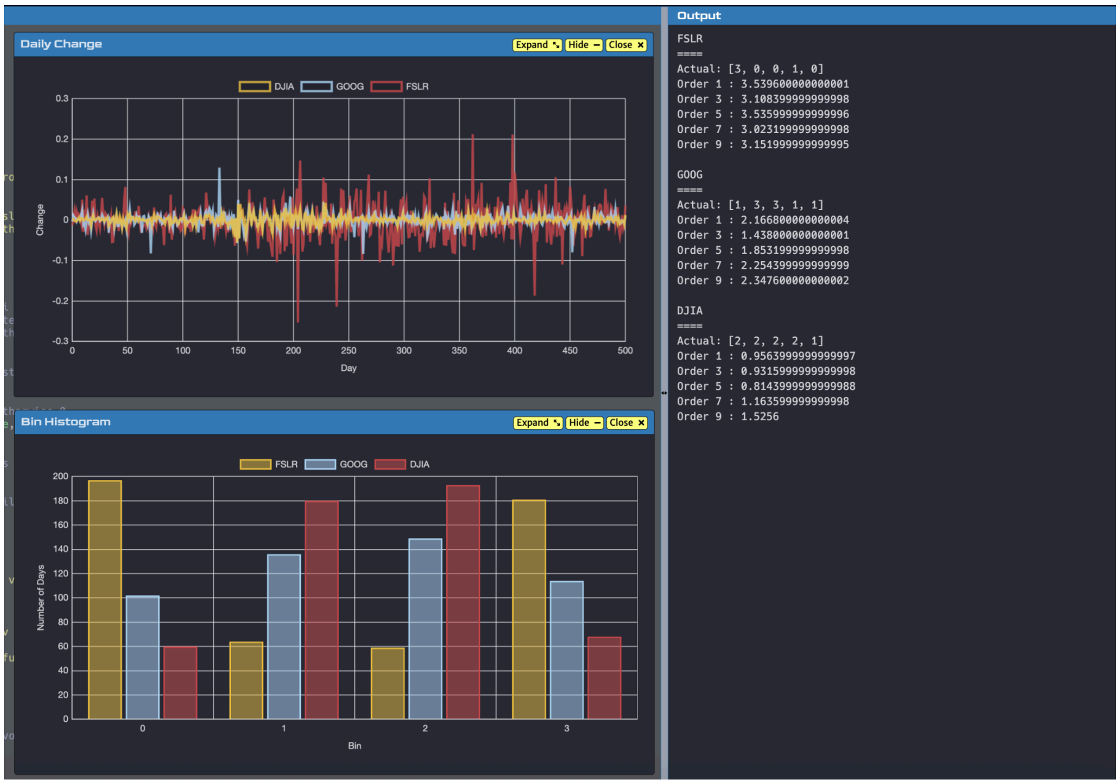
Since range is non-inclusive for the upper-limit, that will **return a random int between 0 and 3.**

4. Print Output

```
FSLR
====
Actual: [3, 0, 0, 1, 0]
Order 1 : 3.539600000000001
Order 3 : 3.1083999999999998
Order 5 : 3.5359999999999996
Order 7 : 3.0231999999999998
Order 9 : 3.1519999999999995

GOOG
====
Actual: [1, 3, 3, 1, 1]
Order 1 : 2.1668000000000004
Order 3 : 1.4380000000000001
Order 5 : 1.8531999999999998
Order 7 : 2.2543999999999999
Order 9 : 2.3476000000000002

DJIA
====
Actual: [2, 2, 2, 2, 1]
Order 1 : 0.95639999999999997
Order 3 : 0.93159999999999998
Order 5 : 0.81439999999999988
Order 7 : 1.1635999999999998
Order 9 : 1.5256
```



```
FSLR
====
Actual: [3, 0, 0, 1, 0]
Order 1 : 3.539600000000001
Order 3 : 3.1083999999999998
Order 5 : 3.5359999999999996
Order 7 : 3.0231999999999998
Order 9 : 3.1519999999999995

GOOG
====
Actual: [1, 3, 3, 1, 1]
Order 1 : 2.1668000000000004
Order 3 : 1.4380000000000001
Order 5 : 1.8531999999999998
Order 7 : 2.2543999999999999
Order 9 : 2.3476000000000002

DJIA
====
Actual: [2, 2, 2, 2, 1]
Order 1 : 0.95639999999999997
Order 3 : 0.93159999999999998
Order 5 : 0.81439999999999988
Order 7 : 1.1635999999999998
Order 9 : 1.5256
```

5. Discussion Questions

5.1. Which stock/index can you predict with the lowest error? Based on the plots of the day-to-day change in stocks and the histogram of bins (which should pop up when you run your code), can you guess why that stock/index is easiest to predict?

The stock that can be predicted with the lowest error is the **DJIA index**. Across all Markov chain orders (1, 3, 5, 7, 9), DJIA will consistently have the lowest MSE, every time we run the code, when compared to both FSLR and GOOGL. Looking at the "Daily Change" plot, FSLR has the most fluctuations, GOOGL is somewhere in the middle, but DJIA is more stable than all. The histogram graph also shows how the bin distribution is more balanced around the middle which makes it easier to see patterns.

5.2. Given that we have divided the day-to-day price change into 4 bins, how many possible states are there in an n-th order Markov chain for predicting the change in stock price?

Each day's price can fall into one of the 4 bins (0 - 3) we see in the histogram. Because of that, n-th order Markov chain can always have **4^n possible states (in theory)**. As we know, each state represents a unique sequence of what happened in the previous n days. As an example, order n = 2 would create 16 states, order n = 3 would create 64 states and so on. So **the higher the order, the more possible state combinations** exist.

On the other hand, in practice, like in our HW, the number of states also depends on the data we have. Since each state will use n data points, the number of states we would form in len(data) - n, as discussed earlier, would mean that the number of states decreases as n increases. Possible combinations grow with n, while observed states in the dataset become smaller.

5.3. The training data we gave you covers two years of data, with 502 data points per stock/index. With that data, is it possible to see all of the possible states in an nth order Markov chain? What are the constraints on n? How do you think it would affect the accuracy of the model if there were not enough data?

Due to limited data, **we can't see all the possible states in a high-order Markov chain**. For example, a 5th or 7th order model would require hundreds of unique combinations of the previous days' bins, but we don't have enough data to cover all of them.

The key limitation is between data length and the model order, because as order n increases the number of states becomes smaller *(len(data) - n).* So higher-order models have more detailed state definitions but much less data to see the most precise probability. **Therefore, if we don't have enough data the prediction will become less reliable.**

Reflection

Q1. The main concepts this week were Markov chain logic and using randomness with the goal of predicting. This HW also helped me understand how to use probability decisions in coding, combined with loops and lists which we have used actively for the past couple of HWs. All these concepts (logic, statistics, programming) are important because they are at the backbone of data science and what I believe will help me to create ML models later in the course.

Q2. The tech and skills highlighted in this HW go beyond just understanding, analyzing and predicting the stock market. These skills can apply to other problems that involve pattern and probabilistic thinking. I believe this will also help me for other data science courses to create and experiment with predictive ML models.

Q3. This week I actually wrote the write-up draft before I worked on the code and that helped a lot! The pre-write Qs helped me understand each part of the HW and how to approach it strategically. Maybe something I would do differently would be the testing, since it was limited/harder to do this time, unless I had built out all the smaller functions.

Q4. I feel a lot more confident with the topics we used this week and using them for complex problems. Specifically, I feel that I understood how to use statistical thinking and the cumulative random selection to solve complex predictive problems and think that I can help my peers by explaining this with other examples like weather prediction.