Modeling:

Input:

Data, a sequence of numbers (corresponding to the bins 0-3)

Order, the number of the model to be create, which must be at least 1

Outputs:

a nth-order Markov chain, which has a tuple of the past states as the key and a map whose keys are the bins for next day and values are the probabilities of the change in stock price being in the bin the following day as the value.

1. Define a function called markov\_chain() with data and order as its two arguments.
2. Inside the function, create an empty map called model
3. Iterate the number over the set[0,the difference between the length of given sequence called data and given order) and assign the values to a variable called index
4. Inside the iteration, iterate all elements in the sequence data over the set [index, the sum of index and order]. Then push all iterated elements into a sequence called history.

Transform the type of elements from sequence to tuple

Assign the value of the element (whose index equal to the sum of the variable index and given order) to the variable called future in the sequence named data.

Check whether the map has the key history. If output is false(which means the map doesn’t have such a key), create a default map for that key(which means the default value of the key should be zero).

Iterate the element (this element is also a map) who has key named history over the outer map, the value of the element (this is new element whose key is future in the inner map) of the map named model adds one each time of the iteration in order to count the frequency of each variable called future.

1. Iterate over the map named model

Inside the iteration, assign 0 to the variable named total\_frequency. Iterate all values in the map named model and assign values to the variable named currency

Inside the second iteration, when at each time of iteration, add to the variable total\_frequency the value of the value in map whose key is the currency inside another map whose key is the variable named key\_value.

1. Iterate values whose corresponding keys are key\_value in the map named model and assign values of these values to the variable named new\_currency.

At each time for iteration divide the value of the value whose corresponding key is new\_currency inside the map which is the value of another map called model and has the key as key\_value by the variable named total frequency(transform the type of frequency into decimals before dividing).

1. Return the map named model

(Before the second function Prediction() with model, last , num as its arguments, I define my own function weight\_number() with a map called dictionary as the argument. The following part is its recipe)

Weight\_number(dictionary)

Input: Dictionay, which is the given map

Output: The value of a new map which will be used as weighted random choices generated from given map

1. Define the function named weight\_number with the given map named dictionary as its argument.
2. Iterate every pair of key and corresponding value and assign these pairs to the variable called dict\_item.
3. Create an empty map called dict1
4. Create a new sequence named middle with zero as its element.
5. Assign zero to the variable named weight
6. Iterate all numbers over the set [0, length of the map named dict\_item) and assign their values to the variable new\_index

Inside the iteration, at each time of the iteration, add to the variable weight the value of a map (this map is the value whose key is new\_index of another map called dict\_item) whose key is one.

Push the variable weight into the sequence called middle

The value of the value whose corresponding key is the sum of new\_index and 1 in the map dict1 equals to the value of the value whose corresponding key is zero in the map which is the value of another map called dict\_item with corresponding key as new\_index.

1. Push one into the sequence middle()
2. Randomly generate a new number and assign the value to the variable randonumber
3. Iterate all numbers over the set[0, length of the sequence middle) and assign their values to the variable named new\_index

Inside the iteration, check whether the variable randonumber is both larger than or equal to the element whose index equals to new\_index in the sequence middle and smaller than the element whose index equals to the sum of one and new\_index in the sequence middle. If the output is true, return the value of the element whose key is the sum of new\_index and one in the map caclled dict1.

Predict

Input:

Model, a given model which is a Markov chain(a map in fact) in the form produced by the function markov\_chain() with data and order as its two arguments.

Last, a sequence of numbers (0-3) whose length is the same as the order of the model.

Num, the number of states to predict in the future

Output:  
Result, a sequence of numbers (0-3) of length that equals to the value of num.

1. Transform the type of the sequence last into the sequence and assign the sequence to another sequence called list1.
2. Create an empty sequence called result
3. While the length of the result is smaller than the value of given parameter called num, transform the type of list1 from sequence to tuple and assign the value to the variable named tuple1

Check whether tuple1 is not an element of the sequence model generated by calling the function markov\_chain with data and order as its arguments. If the output is true (which means tuple1 is not an value in the map model), randomly generate an integer number in the range (0 ,3] and then assign the value to the variable called final. If the output is false (which means tuple1 is an value in the map model), call the function named weight\_number() whose argument is the value of an element whose key is tuple1 in the generated markov\_chain (a map) named model.

1. Push the variable final into the sequence result
2. Delete zero from the sequence list1
3. Push the variable final into the sequence
4. Return the sequence result

Discussion:  
Result:  
FSLR

====

Actual: [3, 0, 0, 1, 0]

Order 1 : 3.7632

Order 3 : 3.1152

Order 5 : 3.478

Order 7 : 3.126

Order 9 : 3.0132

GOOG

====

Actual: [1, 3, 3, 1, 1]

Order 1 : 2.2188

Order 3 : 1.4592

Order 5 : 1.826

Order 7 : 2.2296

Order 9 : 2.3416

DJIA

====

Actual: [2, 2, 2, 2, 1]

Order 1 : 0.9616

Order 3 : 0.9692

Order 5 : 0.8056

Order 7 : 1.1484

Order 9 : 1.4948

1. For FSLR, the order of the model that works best is Order 9

For GOOG, the order of the model that works best is Order 3

For DJIA, the order of the model that works best is Order 5

The reason that orders are not same may be because when we predict the next num values given the model and the last values, we need to randomly predict the next day's change if the state is not in the Markov chain. As a result, each time the number may be different, which lead to possible difference for each order.

1. Stock DJIA can be predicted with the lowest error. Since DJIA has the smallest daily change according to the daily change graph, I think the main reason that it is very easy to predict because it has a small range. In another word, it doesn’t vary much every day.
2. Since we have nth order markov chain and 4 bins. Therefore for each order of markov chain, we have 4 possibilities. As a result, the total result should be 4^n.
3. It depends on the value of n. Therefore in order to make these all 502 data enough for prediction of all possible states, 4^n must less than 502, therefore n must less than 5. If n<5(and n must at least be 1), then the data is enough, otherwise, the data is not enough.

If we don’t have enough data, our prediction may be affected much by some extreme values (the values that are too big or to small), therefore it will be not accurate enough.