Recipe: Stock Prediction

Recipe: markov_chain

Input: *data*, a sequence of int numbers 0-3 and an integer *order* where order>1

Output: Return a *order*th order Markov Chain

Create an empty map *chain* that will map a sequence of values the length of order to a map from next possible values in data to a value representing a real number of chance of that value appearing next

For each index of data-order do
    Initialize *current*, a sequence representing the current values from index to the index+order not inclusive of data

    Initialize *next,* an integer representing the value of index+order from data

    If current not in chain do
        Set $chain_{current}$ to an empty map
    If next not in $chain_{current}$ do
        $chain_{current, next} \leftarrow 1$
    If next in $chain_{current}$ do
        $chain_{current, next} \leftarrow chain_{current, next} + 1$
For each value in chain do
    Initialize *total*, an integer representing the sums of the map of $chain_{value}$
    For each subvalue in $chain_{value}$ do
        $chain_{value, subvalue} \leftarrow chain_{value, subvalue}$ / total
Return chain

Recipe: predict

Inputs: *model*, a map representing a Markov chain, *last*, a sequence (with length of the order of the Markov chain) representing the previous states, *num*, an integer representing the number of desired future states

Output: a sequence of integers that are the next num states

Initialize *next*, a sequence to store the possible next values
Initialize *vals*, a sequence of the last values from last
Initialize possible, a sequence of ints 0-3 in the case model is empty

For each index 0,1,...,num do
    If vals does not exist as a key in model do
        next← a random int from possible
        val ← next

If vals in model do

Initialize *total*, a real number representing total percent of $model_{vals}$

total⟵0

Initialize current, a sequence representing tuple of $model_{vals,item}$ and the percent of it appearing

Current⟵ set to empty

For each key in $model_{vals}$ do

total⟵ total+$model_{vals,key}$

current⟵(key,total)

Initialize randtot, a real number representing total* a random number from [0,1)

For each possible and tot respectively in current do

If randtot<= tot do

Append possible to next

Append possible vals

Return next

Recipe: mse

Inputs: *result*, a list of integers or real numbers representing the actual output, *expected*, a list of integers or real numbers representing the predicted output

Outputs: a real number that is the mean squared error between the two data sets

Initialize *total*, a real number representing the total of the mean squared error

total⟵0

For each index in 1,2,..., length of expected do

total⟵total+$(result_{index}-expected_{index})^2$

Return total/length of result

Results:

```
FSLR
====
Actual: [3, 0, 0, 1, 0]
Order 1 : 3.5891999999999998
Order 3 : 3.1632
Order 5 : 3.100000000000002
Order 7 : 3.148400000000001
Order 9 : 3.0923999999999997

GOOG
====
Actual: [1, 3, 3, 1, 1]
Order 1 : 2.226400000000001
```

```
Order 3 : 1.4455999999999999
Order 5 : 2.346800000000003
Order 7 : 2.298799999999999
Order 9 : 2.338799999999998

DJIA
====
Actual: [2, 2, 2, 2, 1]
Order 1 : 0.9931999999999984
Order 3 : 0.9431999999999989
Order 5 : 0.7831999999999997
Order 7 : 1.4528
Order 9 : 1.493200000000001
```

Discussion:

1. The best order for each stock is FSLR:9, GOOG:2, DJIA:5. The difference in best order stems from the data sets and their different volatility; the predictions of the different orders will have different errors.
2. DJIA order 5 works the best out of the stocks/indexes. The different stocks have different errors for the same orders because of the difference in volatility between the stocks. So DJIA would have the most consistent values showing why it has the lowest error.
3. The possible states would be 4^k where k represents order so for any order we can find the possible states
4. With 502 data points it's only reasonable to see the 4th order because 4^4=256 while 4^5=1024 which is more states than total data points so most states won't be seen. So as the states get higher without enough data the error would get higher because there are more states than data points to work off of so the predicted would start to overfit leading to error.