

Dhanvi Shah
CDS DS210: Section B1
Final Project Report

Data.....	2
Purpose:.....	2
Goals:.....	2
Code Walkthrough.....	2
Read-In (fn read).....	2
Adjacency (Adjacent Mod).....	3
Creating Graph (Mod Create).....	4
Recommendations:.....	5
Daily Expect:.....	5
Portfolio & Risk:.....	5
Making Groups (fn findcomp & fn groups).....	6
Recommendations (fn recommend).....	7
Tests.....	8
Preliminary Manual Testing:.....	9
Test #1:.....	10
Test #2:.....	11

Data

Purpose:

In general, asset prices are very difficult to predict, especially with relation to each other. However, barring any significant current events or market defining events, this project is meant to use similarity between years of data for all relevant asset classes, in order to provide recommendations between them.

By analyzing around four years of historical data across many different assets, I want to make predictions for the next four years. For example, throughout this project, we will see if the GBP is doing well 72 days after we start another project like this, then those that are heavily correlated with it may do well on the corresponding day.

In case the data does not load, it can be found:

https://github.com/julian-west/asset_price_correlations/tree/master/data

Goals:

1. Find related assets, specifically those increasing/decreasing together.
2. Take into account each users' risk tolerance and based on their current holdings, create a diversified or homogeneous portfolio.

Code Walkthrough

Read-In (fn read)

We read in our data using the read function in the main file. It takes in the local path of our csv file and turns it into a digestible format. For this, I involved the `csv::reader`, read in preliminary data like the headers (column names) and the baseline data (which is represented by the last row in this data, as it is in reverse chronological order). Due to the large number of vertices (originally over 30,000), I also only took a subset of this data at 30-record intervals. This subset is still greater than the required 1000-nodes for this project.

After collecting this preliminary data, I find the log-return value for each asset on each day (returns from the baseline value). This is because assets are usually [normalized and compared according to their log-return value](#). This value is also useful in creating edges according to correlation between each node.

After finding this value, I push the label (date & asset) and the corresponding log-return to a vector called "Nodes". I also create a Hashmap (Assetprices) with each asset type (determined by the column name) and the corresponding log returns throughout our time frame.

This function returns Nodes and Assetprices.

In assetprices, each record represents a date and the corresponding asset prices on that date. The following is a subset of how this read-in looks.

```
StringRecord(["11/11/2013", "72.59987148", "25.34", "143.5402519", "37.90317998", "58.27816456", "89.67617224", "41.72350643", "26.38576606", "27.42169952", "44.14917519", "24.93214548", "33.90027881", "157.64", "94.70556345", "132.67", "34.02084224", "98.53", "23.63583572", "123.87", "93.79867422", "54.57954382", "31.824066", "98.95313837", "20.6", "163.5629998", "105.8719507", "92.69904326", "34.26", "21.84", "49.24035645", "54.83630234", "776.32", "40.74803652", "77.98607181", "15.79063051", "31.59789939", "33.90058666", "100.1734809", "106.63"])
StringRecord(["11/8/2013", "72.66299396", "25.23", "143.4309642", "38.09662966", "58.15333373", "90.19967102", "42.00335922", "26.20553542", "27.2412936", "44.20577669", "24.7693676", "34.01584224", "157.93", "94.63575836", "132.13", "33.66898128", "98.58", "23.61638236", "124.279", "93.84504493", "54.6222241", "31.95050878", "99.11937264", "20.69", "163.5353144", "106.071745", "93.04997149", "34.01", "21.895", "49.09180524", "55.06997079", "777.92", "40.76187443", "77.84216948", "15.78684651", "31.56065967", "33.90929251", "100.1544727", "106.29"])
StringRecord(["11/7/2013", "73.14092132", "25.09", "141.937366", "38.18085895", "57.8145072", "90.89766939", "41.84223186", "26.09739703", "26.97068472", "43.85803482", "24.69702188", "33.73112076", "158.53", "94.77536855", "132.74", "33.48162674", "99.73", "23.48020882", "126.16", "94.91157129", "55.364861", "32.02953551", "100.0992799", "20.83", "161.3584101", "106.9850904", "95.34450227", "33.97", "21.77", "48.82965605", "54.68202702", "812.8", "40.01923985", "76.69994469", "15.42736654", "31.27670684", "33.97023347", "100.2114974", "107.01"])
StringRecord(["11/6/2013", "72.99664136", "25.27", "143.1850669", "38.88431235", "58.74182193", "91.11371651", "42.4273786", "26.38576606", "27.99899845", "44.69632308", "25.10396658", "34.2168221", "158.6", "95.17525199", "133.77", "34.03455111", "99.04", "24.10757977", "127.2", "94.66116945", "56.0477455", "31.96631412", "99.89804898", "21", "163.4246289", "106.8423802", "94.54366603", "34.189", "21.67", "49.61173447", "55.67444131", "785.024", "40.55430576", "77.9006298", "15.60143052", "31.69099868", "34.29234999", "100.1259603", "107.51"])
```

The following is a small output of Nodes, which represents all of our data in a digestible format for our program to read.

```
hina", 0.04673627060232971), ("12/4/2013", "Yen", -0.03558215987944808), ("12/4/2013", "Gold Miners", -0.12643748721816442), ("12/4/2013", "iold", -0.056634954269879134), ("12/4/2013", "10yr treasuries", -0.010309552774970372), ("12/4/2013", "US real estate", -0.0579475853987206), ("12/4/2013", "High yield Bond", 0.00347533551391985), ("12/4/2013", "Corp Bond", -0.004135363099700619), ("12/4/2013", "Silver", -0.1038857445072108), ("12/4/2013", "SNP 500", 0.019779197753580842), ("12/4/2013", "Bonds II", -0.013301605984679329), ("12/4/2013", "20+ Treasuries", -0.03022751627538565), ("12/4/2013", "Oil", 0.02176613628771129), ("12/4/2013", "USD", -0.0027624326959100726), ("12/4/2013", "Europe", -0.00696444021599475), ("12/4/2013", "Pacific", -0.011778313848785363), ("12/4/2013", "VXX", -0.09884330404494225), ("12/4/2013", "Materials", 0.0157084295269088856), ("12/4/2013", "Energy", 0.004057741371890436), ("12/4/2013", "Finance", 0.03249409245874562), ("12/4/2013", "Tech", 0.03141272284218295), ("12/4/2013", "Utilities", -0.021478377617904876), ("12/4/2013", "ST Corp Bond", 0.0010434068418226194), ("12/4/2013", "CHF", 0.010371423645849689)]
```

Adjacency (Adjacent Mod)

Since we found the nodes, all that's left before we are able to create the graph is finding the edges between these nodes. We do this through the `createadj` function in the Adjacent mod. It takes in the list of Nodes from the `read` function, the size of Nodes, and a threshold. This function creates a Hashmap, "Edges", and an (differently styled) Adjacency Matrix, "Matrix". Although the traditional adjacency matrix would have 2 columns and an indicator of whether these nodes share an edge, the matrix we use here has n columns and n rows and True/False values at `[i][j]` depending on if there is an edge there. The main part of this function is a nested for-loop. For each Node, it accesses all other nodes and sees if the correlation between these two assets' log returns is less than or equal to the threshold. (Concession: I originally planned to find cosine similarity between each asset but realized this is not applicable to the way I read in the data. Although distance between returns may not be the best indicator of correlation, I was able to find resources that supported that this may be a good comparison point nonetheless: [Here](#), and [Here](#))

Since we are measuring correlation by distance, as shown below, the lower the better. Therefore, the function then creates an edge between these two values by adding the `jlabel` as a value to the `ilabel` key in Edges. It also marks `[i][j]` as True, indicating an edge, in our Matrix.

```
let corr = (ivalue-jvalue).abs();
```

This function returns the Adjacency List (HashMap) and the Matrix.

This is a subset of the adjlist that shows the adjacency list for Gold Miners 11/17/2014):

```
... "11/17/2014",
\"Gold Miners\"": [\"11/8/2017\", \"Commodities\", \"11/8/2017\", \"GBP\", \"11/8/2017\", \"China Large Cap\", \"11/8/2017\", \"Euro\", \"11/8/2017\",
\"Yen\", \"11/8/2017\", \"Gold Miners\", \"11/8/2017\", \"Gold\", \"11/8/2017\", \"Silver\", \"11/8/2017\", \"Oil\", \"11/8/2017\", \"VXX\", \"11/
8/2017\", \"Energy\", \"11/8/2017\", \"CHF\", \"9/27/2017\", \"Commodities\", \"9/27/2017\", \"GBP\", \"9/27/2017\", \"China Large Cap\", \"9/27/2017
\", \"Euro\", \"9/27/2017\", \"Yen\", \"9/27/2017\", \"Gold Miners\", \"9/27/2017\", \"Gold\", \"9/27/2017\", \"Silver\", \"9/27/2017\", \"Oil\", \"
9/27/2017\", \"VXX\", \"9/27/2017\", \"Energy\", \"9/27/2017\", \"CHF\", \"8/15/2017\", \"Commodities\", \"8/15/2017\", \"UK\", \"8/15/2017\", \"GBP
\", \"8/15/2017\", \"China Large Cap\", \"8/15/2017\", \"Euro\", \"8/15/2017\", \"Yen\", \"8/15/2017\", \"Gold Miners\", \"8/15/2017\", \"Gold\", \"
8/15/2017\", \"Silver\", \"8/15/2017\", \"Oil\", \"8/15/2017\", \"VXX\", \"8/15/2017\", \"Energy\", \"8/15/2017\", \"CHF\", \"7/3/2017\", \"Commodit
ies\", \"7/3/2017\", \"UK\", \"7/3/2017\", \"GBP\", \"7/3/2017\", \"China Large Cap\", \"7/3/2017\", \"Euro\", \"7/3/2017\", \"Yen\", \"7/3/2017\",
\"Gold Miners\", \"7/3/2017\", \"Gold\", \"7/3/2017\", \"Silver\", \"7/3/2017\", \"Oil\", \"7/3/2017\", \"VXX\", \"7/3/2017\", \"Energy\", \"7/3/2
```

We also get the following adjacency matrix (same information but iterable easily with `[i][j]`):

```
se, false, true, false, false, false, true, false, true, true, true, false, true, false, false, false, false, true, false, true, false, true, false, true
, true, true, false, true, true, true, true, true, true, true, false, false, true, false, true, false, false, true, false, true, false, true, false, true
, false, true, false, false, false, false, true, false, true, true, false, true, true, true, false, true, true, true, true, true, true, true, true, true
, true, false, false, true, false, true, false, false, true, false, true, true, true, true, false, true, false, false, false, true, false, true, false, true, false
, true, false, true, true, true, true, true, true, true, true, true, true, true, true, false, true, true, true, true, true, true, true, true, true, true, f
alse, true, false, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, t
rue, true, true, true, true, false, false, true, false, true, false, true, false, true, true, true, true, false, true, false, true, true, true, true, true, t
rue, true, false, true, true, false, false, true, false, true, false, true, false, true, true, true, true, false, true, false, false, false, false, true,
false, true, false, true, true, false, true, true, false, false, true, true, true, true, false, true, true, true, false, true, true, true, true, false,
e, false, false, true, false, true, true, true, false, true, false, false, false, false, true, false, true, false, true, true, false, true, true, true, false,
true, true, true, true, true, true, true, true, false, false, false, false, true, false, false, false, true, false, true, true, true, true, false,
false, false, false, true, false, true, false, true, false, false, true, false, false, false, false, false, true, true, true, false, true, true, true, false, f
alse, false, false, true, false, false, false, true, false, false, false, false, false, false, false, false, true, true, true, false, true, true, true, false, f
alse, false, false, true, false, false, false, false, true, false, false, false, false, false, false, false, false, true, false, true, false, true, false,
false, true, false, false, false, false, false, true, true, true, false, true, true, true, false, false, false, false, false, true, false, true, false,
false, false, true, false, true, false, false, false, false, true, false, true, false, false, false, false, false, false, false, false, false, false, true, tr
ue, true, false, true, true, true, false, false, false, true, false, false, false, false, false, false, false, false, false, true, false, false, false, fa
lse, true, false, true, false, true, false, false, true, false, true, false, true, true, true, true, true, false, true, true, true, false, false, false, false,
```

Additionally, since the threshold is something that can be changed when calling the function, this adjacency list serves two purposes:

1. Calculating the assets with most similar performance (as traditionally used, when threshold is set to > 0.5)
2. Calculating the least correlated/negatively correlated assets. For example, if threshold is set to -0.5 , the adjacency list will provide values that are negatively correlated

Creating Graph (Mod Create)

To create the graph itself, we move to our create module. Here, I start with a Graph struct; this consists of the size of the graph (n/number of nodes), a list of vertices, and both the Adjacency list and Adjacency Matrix.

In our main function, we create the graph like this:

```
let mut graph = Graph::new(n, Nodes.clone(), adjmap.clone(), adjmat.clone());
let graph = graph.undirected();
```

We get Nodes as an output of our read function, n is the size of Nodes, and adjmap and adjmat both come from our adjacency function.

I create a graph using the new and undirected functions. New takes all of our inputs and feeds them into the struct. We don't have to do much for making sure our graph is undirected since the createadj function iterates over every node, finding bipartite edges. However, this undirected function makes our self.vertices easier to read by sorting them per the label.

Recommendations:

Daily Expect:

Our graph is now created. We can apply it to our preliminary objectives for this project from earlier. Although this is a more elementary output, our `dailyexpect` method takes in the graph and a minimum return value, and throws out two vectors, one with assets with positive returns, and another with negative. This function helps to address our first goal of finding similarities for assets across our timeframe. For the purposes of this project, I set the threshold = to 0.0. However, for someone who wants higher returns, this threshold can also be higher. This method is also helpful in seeing if the data was read in correctly, and if log returns were computed correctly. We will use it as a test later.

Portfolio & Risk:

Next, we come to our joint methods, portfolio and risk. Risk is a BFS algorithm that produces a vector of distances from each node to the rest, as measured by neighbors between the nodes in question. It starts with an empty queue, pushes our starting vertex to it, and as long as there is a queue, it explores the neighbors for the first value in this queue. It accesses this vertex in the adjacency matrix we made earlier (hence why it wasn't a traditional adjacency matrix), and if it is true that our current vertex shares an edge with this target, then the corresponding distance for this point is $1 + \text{distance to the value we accessed through our while loop}$. This value is then added to the queue and the process is repeated until there is nothing in the queue. This happens when we have either reached all nodes, no matter how far, or if no matter how many nodes we iterate over, there are some that are unreachable.

The portfolio method calls the risk method for all vertices. In principle, this is a nested for loop (but the second loop may not include all n in vertices if they don't have neighbors).

This is a snippet of the output. As we can see, it described the distance between the node representing Commodities on 1/10/2017 and the other nodes in our data. Sense check, this is a valid output because the distance back to commodities on 1/10/2017 is 0 while the distance to pacific on the same day is None. Pacific on 1/10/2017 is likely an outlier in a different component (which we will explore later).

```
Distance from Asset "1/10/2017", "Commodities"
"\1/10/2017\", \"10yr treasuries\"": Some(2)
"\1/10/2017\", \"20+ Treasuries\"": Some(7)
"\1/10/2017\", \"Bonds Global\"": Some(3)
"\1/10/2017\", \"Bonds II\"": Some(1)
"\1/10/2017\", \"CHF\"": Some(1)
"\1/10/2017\", \"China Large Cap\"": Some(1)
"\1/10/2017\", \"China\"": Some(2)
"\1/10/2017\", \"Commodities\"": Some(0)
"\1/10/2017\", \"Corp Bond\"": Some(2)
"\1/10/2017\", \"DOW\"": Some(1)
"\1/10/2017\", \"EAFE\"": Some(1)
"\1/10/2017\", \"Emerg Markets Bonds\"": Some(3)
"\1/10/2017\", \"Emerg Markets\"": Some(5)
"\1/10/2017\", \"Energy\"": Some(5)
"\1/10/2017\", \"Euro\"": Some(5)
"\1/10/2017\", \"Europe\"": Some(1)
"\1/10/2017\", \"Finance\"": Some(4)
"\1/10/2017\", \"France\"": Some(3)
"\1/10/2017\", \"GBP\"": Some(3)
"\1/10/2017\", \"Germany\"": Some(2)
"\1/10/2017\", \"Gold Miners\"": Some(2)
"\1/10/2017\", \"Gold\"": Some(1)
"\1/10/2017\", \"High yield Bond\"": Some(1)
"\1/10/2017\", \"Italy\"": Some(6)
"\1/10/2017\", \"Japan\"": Some(3)
"\1/10/2017\", \"Materials\"": Some(2)
"\1/10/2017\", \"Oil\"": Some(1)
"\1/10/2017\", \"Pacific\"": None
"\1/10/2017\", \"Pacifix ex Japan\"": Some(2)
"\1/10/2017\", \"SNP 500\"": Some(1)
"\1/10/2017\", \"ST Corp Bond\"": Some(1)
```

This is another snippet of the same output for further reference:

```
"11/25/2016\", \"Tech\"": Some(1)
"11/25/2016\", \"UR\"": Some(4)
"11/25/2016\", \"US real estate\"": Some(2)
"11/25/2016\", \"USD\"": Some(3)
"11/25/2016\", \"Utilities\"": Some(1)
"11/25/2016\", \"Yen\"": Some(2)
"11/25/2016\", \"Yen\"": Some(4)
"11/8/2017\", \"10yr treasuries\"": Some(2)
"11/8/2017\", \"20+ Treasuries\"": Some(8)
"11/8/2017\", \"Bonds Global\"": Some(1)
"11/8/2017\", \"Bonds II\"": Some(4)
"11/8/2017\", \"CHF\"": Some(3)
"11/8/2017\", \"China Large Cap\"": Some(2)
"11/8/2017\", \"China\"": Some(3)
"11/8/2017\", \"Commodities\"": Some(4)
"11/8/2017\", \"Corp Bond\"": Some(6)
"11/8/2017\", \"DOW\"": Some(2)
"11/8/2017\", \"EAFE\"": Some(4)
"11/8/2017\", \"Emerg Markets Bonds\"": Some(4)
"11/8/2017\", \"Emerg Markets\"": Some(5)
"11/8/2017\", \"Energy\"": Some(5)
"11/8/2017\", \"Euro\"": Some(5)
"11/8/2017\", \"Europe\"": Some(2)
"11/8/2017\", \"Finance\"": Some(4)
"11/8/2017\", \"France\"": Some(4)
"11/8/2017\", \"GBP\"": Some(4)
"11/8/2017\", \"Germany\"": Some(2)
"11/8/2017\", \"Gold Miners\"": Some(1)
"11/8/2017\", \"Gold\"": Some(2)
"11/8/2017\", \"High yield Bond\"": Some(2)
"11/8/2017\", \"Italy\"": Some(6)
"11/8/2017\", \"Japan\"": Some(1)
```

Making Groups (fn findcomp & fn groups)

Our next two methods are built very similarly in that they are independent methods but groups calls the for loop in findcomp for all nodes.

Findcomp also makes a queue and while there is a point in the queue, it will access all of the connections in the corresponding row in our adjacency matrix. If there is a connection to a different node from our point, and this new node does not have an associated component, it assigns a component that corresponds with which iteration of the for loop we are on (after which iteration we were able to access this node). It also adds this new node to the queue so that it can be explored.

Groups calls this function for all nodes that don't have a component associated with them. Each time it has to explore the neighbors of a new point, the "count variable" (the one that was assigned as the component number earlier), increments by 1, indicating that a new group is made.

There are 12 components.

```
Finished dev [unoptimized + debuginfo] target(s) in 0.50s
Running `C:\Users\dhanv\OneDrive\Desktop\2023-2024\Spring 24\DS 210
\final_proj\target\debug\final_proj.exe`
12 components:
```

However, as we can tell by this subset of the output (on the right), most points are in one group (group 1). Meanwhile some groups, like group 12, only have about 1 point in them, meaning that they may just be outlier assets in our assessment.

Recommendations (fn recommend)

For our final graph utilization, we utilize the recommend function (housed in our main file). This function resembles a page rank. It takes in a variable number, risklevel, of iterations that represents the risk level of the user (ie: one iteration = less risky, very similar asset types while 20 iterations could result in an asset with very little similarity).

This function outputs a hashmap. For each asset, it visits *risk-level* number of other assets. It also keeps track of the assets we've visited throughout this iteration of the for-loop in "visitedneigh" and makes sure to not visit them. It takes steps to the original asset's neighbors, and choose one at random. It will set this next neighbor as the current asset, and depending on how high the risk level is, add this asset to "visitedneigh" and will take another step. Once our secondary for loop is done (the length of which is set by risklevel), it will assign whatever current asset we have accessed as the value for this key. Although a bit difficult to read, this is a subset of the output:

```
{
  "Emerg Markets Bonds": "\5/12/2015", "\Corp Bond", "\5/12/2015", "\China Large Cap": "\8/22/2014", "\China Large Cap", "\12/31/2014", "\Corp Bond": "\1/28/2016", "\Pacific", "\12/14/2015", "\Yen": "\8/31/2016", "\China Large Cap", "\12/14/2015", "\Germany": "\8/6/2015", "\France", "\1/17/2014", "\EAFE": "\5/19/2017", "\ST Corp Bond", "\8/6/2015", "\USD": "\7/11/2014", "\20+ Treasuries", "\1/17/2014", "\Emerg Markets": "\8/6/2015", "\Europe", "\8/22/2014", "\Bonds Global": "\5/19/2017", "\Bonds II", "\3/4/2014", "\Gold Miners": "\5/12/2015", "\EAFE"
}
```

The key is before the semicolon and the value is after. However, it is important to remember that since these are calculated at random, this algorithm is a bit more unstable than the rest.

```
498 : Some(1)
499 : Some(12)
500 : Some(1)
501 : Some(1)
502 : Some(1)
503 : Some(1)
504 : Some(1)
505 : Some(1)
506 : Some(1)
507 : Some(1)
508 : Some(1)
509 : Some(1)
510 : Some(1)
511 : Some(1)
512 : Some(1)
513 : Some(1)
514 : Some(1)
515 : Some(1)
516 : Some(1)
517 : Some(1)
518 : Some(1)
519 : Some(1)
520 : Some(1)
521 : Some(1)
522 : Some(1)
523 : Some(1)
524 : Some(1)
525 : Some(1)
526 : Some(1)
527 : Some(1)
528 : Some(1)
529 : Some(1)
530 : Some(1)
531 : Some(1)
532 : Some(1)
533 : Some(1)
```

Tests

```
#[test]
fn test(){
  let testpath = r"C:\Users\dhanv\OneDrive\Desktop\2023-2024\Spring 24\DS 210\final_proj\final
  let (Assetstest, Nodestest) = read(testpath).expect("Couldn't Read!");
  let nt = Nodestest.len();
  let (testmap, testmat) = adjacent::createadj(Nodestest.clone(), 0.03, nt);
  let far_test = recommend(testmap.clone(), 1);
  let key = "\"2/13/2017\", \"Germany\"";
  let expvalue = "\"2/13/2017\", \"Pacifix ex Japan\"";
  match far_test.get(key){
    Some(value) => assert_eq!(value, expvalue),
    None => panic!("No key at all!")
  }
  let testgraph = Graph::new(nt, Nodestest.clone(), testmap.clone(), testmat.clone());
  let (testpos, testneg) = testgraph.dailyexpect(0.0);
  assert_eq!(testneg.iter().map(|(s, v)| (s.as_str(), *v)).collect::<Vec<_>>(),
  [("\"2/13/2017\", \"Commodities\", -0.47149904489571054),
  ( "\"2/13/2017\", \"Emerg Markets\", -0.02977940328378424),
  ( "\"2/13/2017\", \"Italy\", -0.1748501500584068),
  ( "\"2/13/2017\", \"France\", -0.016355540693393806),
  ( "\"2/13/2017\", \"UK\", -0.08006145084453031)]]);
  testgraph.portfolio();
  testgraph.groups()
}
```

This test looks like my main function to ensure there is no discrepancy in our final results. The only difference is that the test set is much smaller.

In order to test that my overall code works correctly, I extracted a small subset of this data into a testing csv file. It is only a few dates and a few assets so that I could compute the expected values by hand/excel and make sure my code works as expected. This is what my test data looks like (although it is also included in the github submission):

Date	Bon ds Glob al	Comm odities	DO W	Eme rg Mar kets	EAF E	Eme rg Mar kets Bon ds	Pacif ix ex Japa n	Ger man y	Italy	Japa n	Fran ce	UK
2/13/2017	79.4 6899	15.87	200. 8181	38.1 6121	59.4 1037	109. 2537	42.5 1091	26.9 0614	23.3 9388	51.2 5009	24.6 3444	31.2 9816

11/1/2013	73.0	25.43	141.3	39.3	58.4	91.6	42.1	26.3	27.8	44.4	25.0	33.9
	4173		9465	1471	5649	7045	8145	0917	6369	5105	4066	0698

They are far apart in date to ensure that our log return differences vary for each asset we analyze.

Preliminary Manual Testing:

Since this subset of data was small, I calculated the log returns using a different excel sheet and found the following:

Date	Bonds Global	Commodities	DOW	Emerg Markets	EAFE	Emerg Markets Bonds	Pacific ex Japan	Germany	Italy	Japan	France	UK
2/13/2017	0.0843 35961 01	-0.471 49904 49	0.3469 49297 2	-0.029 77919 442	0.0161 86073 43	0.1754 72619 8	0.0077 80197 928	0.0224 36965 74	-0.174 84995 78	0.1423 28790 4	-0.016 35543 986	-0.080 06158 302
11/1/2013	0	0	0	0	0	0	0	0	0	0	0	0

I also found the distance between each of these and found which ones would have edges based off of the maximum distance threshold in the code. This is what the output looks like (to make it simpler, the threshold is now 0.03 and the green values are less than the threshold)

	Bonds Global	Commodities	DOW	Emerg Markets	EAFE	Emerg Markets Bonds	Pacific ex Japan	Germany	Italy	Japan	France	UK
Bonds Global	0	0.5558 35005 9	0.2626 13336 1	0.1141 15155 4	0.0681 49887 59	0.0911 36658 79	0.0765 55763 09	0.0618 98995 28	0.2591 85918 9	0.0579 92829 37	0.1006 91400 9	0.1643 97544
Commodities		0	0.8184 48342	0.4417 19850 5	0.4876 85118 3	0.6469 71664 7	0.4792 79242 8	0.4939 36010 6	0.2966 49087 1	0.6138 27835 3	0.4551 43605	0.3914 37461 9
DOW			0	0.3767 28491 6	0.3307 63223 7	0.1714 76677 3	0.3391 69099 2	0.3245 12331 4	0.5217 99255	0.2046 20506 8	0.3633 04737	0.4270 10880 2

Emerg Marke ts					0.0459 65267 085	0.2052 51814 2	0.0375 59392 35	0.0522 16160 16	0.1450 70763 4	0.1721 07984 8	0.0134 23754 56	0.0502 82388 6
EAFE						0.2052 51814 02	0.0375 59392 35	0.0522 16160 16	0.1450 70763 4	0.1721 07984 8	0.0134 23754 56	0.0502 82388 6
Emerg Market s Bonds							0.1676 92421 09	0.1530 35654 1	0.3503 22577 6	0.0331 43829 42	0.1918 28059 7	0.2555 34202 8
Pacifix ex Japan								0.0146 56767 081	0.1826 30155 8	0.1345 48592 5	0.0241 35637 79	0.0878 41780 95
Germa ny									0.1972 86923 06	0.1198 91824 7	0.0387 92405 6	0.1024 98548 8
Italy										0.3171 78748 02		0.0947 88374 82
Japan											0.1586 84230 02	0.2223 90373 4
France												0.0637 06143 16
UK												0

Test #1:

The first test primarily serves to see if the code properly reads the file, creates the graph, and calculates the log returns correctly. It builds off of the two lists we made before of positive and negative returns. Since there were only a few negative returns in this test data, I asserted that this should be the value of the entire “testneg” list (was able to find the values, differently rounded from our excel sheet, from the Nodes map and create this assertion).

```
assert_eq!(testneg.iter().map(|(s, v)| (s.as_str(), *v)).collect::<Vec<_>>(),
[("\2/13/2017\", \"Commodities\"\", -0.47149904489571054),
("\2/13/2017\", \"Emerg Markets\"\", -0.02977940328378424),
("\2/13/2017\", \"Italy\"\", -0.1748501500584068),
("\2/13/2017\", \"France\"\", -0.016355540693393806),
("\2/13/2017\", \"UK\"\", -0.08006145084453031)]);
```

This test passed successfully:

```
running 1 test
test test ... ok
```

```
test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

Test #2:

My other test mostly tested if the pagerank style, recommend function, was working correctly and if edges were created correctly.

```
let far_test = recommend(testmap.clone(), 1);
let key = "\2/13/2017\", \"Germany\"";
let expvalue = "\2/13/2017\", \"Pacifix ex Japan\"";
match far_test.get(key){
    Some(value) => assert_eq!(value, expvalue),
    None => panic!("No key at all!")
}
```

```
running 1 test
test test ... ok
```

```
test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

To make the test more simple, I picked the Germany value as it only has one edge. I asserted that the corresponding value in the hashmap for this key should be the Pacifix ex Japan asset. This test passed.

References

I used the following resources to iterate through my csv file:

https://doc.rust-lang.org/rust-by-example/flow_control/match.html

<https://users.rust-lang.org/t/solved-how-to-fix-this-borrowed-value-does-not-live-long-enough/107105>

<https://users.rust-lang.org/t/how-to-get-first-column-from-library-rust-csv/4055/2>

This for understanding graphs, nodes, and edges:

[https://www.wsfcs.k12.nc.us/cms/lib/NC01001395/Centricity/Domain/1707/INTERATED%20unit 04.pdf](https://www.wsfcs.k12.nc.us/cms/lib/NC01001395/Centricity/Domain/1707/INTERATED%20unit%2004.pdf)

These resources for helping with Git:

<https://stackoverflow.com/questions/4181861/message-src-refspec-master-does-not-match-any-when-pushing-commits-in-git>

<https://stackoverflow.com/questions/16330404/how-to-remove-remote-origin-from-a-git-repository>

These for understanding my dataset and calculating log returns:

<https://gregorygundersen.com/blog/2022/02/06/log-returns/>

This for understanding the logic behind BFS:

<https://stackoverflow.com/questions/71189961/bfs-algorithm-tracking-the-path>

<https://stackoverflow.com/questions/13171038/how-to-find-the-distance-between-two-nodes-using-bfs>

<https://blog.logrocket.com/pathfinding-rust-tutorial-examples/>

For component marking:

[https://stackoverflow.com/questions/69010836/in-rust-how-can-i-create-a-function-which-will-accept-a-marker-component-as-a,](https://stackoverflow.com/questions/69010836/in-rust-how-can-i-create-a-function-which-will-accept-a-marker-component-as-a)

<https://dev.to/fushji/a-weekly-rust-pill-5-5d5j>

<https://towardsdatascience.com/implementing-a-connected-component-labeling-algorithm-from-scratch-94e1636554f>