Final Project Report

Angelina Knight, Sean Fishbein, Nicholas Ketchum

SI 206—The University of Michigan School of Information

December 14, 2020

LIST OF CONTENTS:

1. GOALS:

a. Original Goals:

- **i.** Keep it simple!
- **ii.** Display *correlations* between two key economic variables in the year 2020 (Dow Jones Industrial Average and Initial/Continuing Unemployment Claims) along with new COVID-19 cases using chart visualization.
- iii. Visualize Dow Jones Industrial daily highs and lows: totals and averages
- iv. Visualize new COVID-19 cases: percentage change
- v. Visualize initial and continuing unemployment claims: totals and percentages
- vi. Attempt to keep time periods similar except when to compare before/after statistics

b. **Achieved Goals:**

   i. We kept a simple plan where we developed our own functions to be executed sequentially, reducing potential communication issues.

   ii. Display correlations between two key economic variables in the year 2020 along with trends in COVID cases using chart visualization.

   iii. Capture data from three sources (2 API, 1 website) for comparison:

   1. COVID-19 daily new cases,
   2. Dow Jones Industrial daily highs and lows, and
   3. Federal Reserve Initial and Continued Unemployment claims.

   iv. Take all three data sources and calculate percentage movements alongside totals and save as comma separated value files.

   v. Graph all three data sources starting now and insert enough data to both meet requirements and provide real insights. *Note: Graphs don't exactly share the same time period; Dow and unemployment data have longer histories than COVID-19 records. Dow and unemployment data includes dates previous to COVID-19 measurements. This is to help spot differences between pre and post-COVID-19.*

   vi. Allow each execution to add more back-data to CSV and charts, excepting COVID cases that started records in March 2020.

2. PROBLEMS:

- Zoom communication was difficult, even when we complemented Zoom with other mechanisms (texts, calls, chats). It was difficult to illustrate ideas or organize conversation over technical issues. We had to build in small chunks that modularly fit together to avoid stepping on each other's toes, even though we used version control.
- From each of the sources we used, we pulled data in the form of dates and each source had a different way of formatting their dates. We had to figure out a way to reformat out dates into a universal string date format so we were consistent across all our database tables.

- We initially and unwittingly stored data in different sort orders in the database. We had to rewrite some code to reverse ordering.
- Chart axes had too-short or too-long ticks and intervals. We had to research how to optimize those. If we had more time we could write a formula to stretch intervals depending on how much data is graphed.

3. CALCULATION FILES:

a. covid-cases.csv
    i. Stores date and daily percentage change

b. dji.csv
    i. Stores date and daily closing prices and trading range

c. unemployment-ICSA-percentages.csv
    i. Stores date and daily percentage change of initial claims

d. unemployment-ICSA-total.csv
    i. Stores date and weekly total new initial claims

e. unemployment-CCSA-percentages.csv
    i. Stores date and daily percentage change of continued claims

f. unemployment-CCSA-total.csv
    i. Stores date and weekly total new continued claims

g. **Samples of all six computed CSV files** saved after six program executions:
    i. https://www.dropbox.com/sh/z99oqpath5q2qlu/AACnb4fm1jfB4q_PFnYmr6oAa?dl=0

4. VISUALIZATION CHARTS:

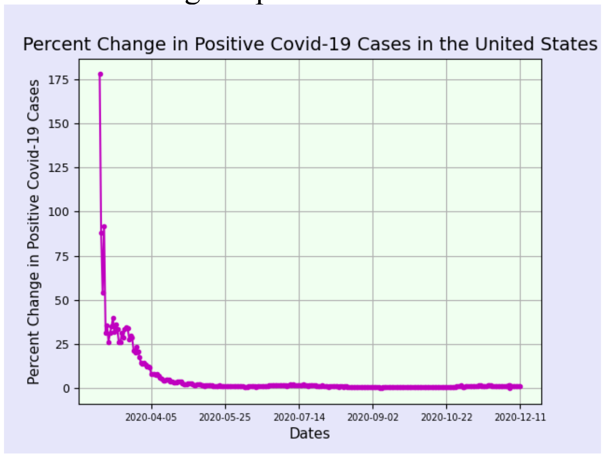
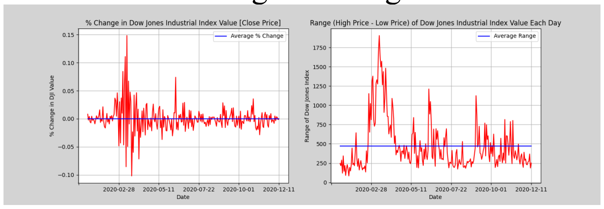
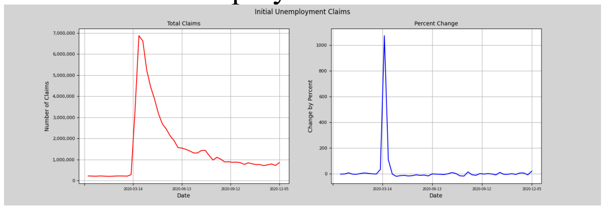
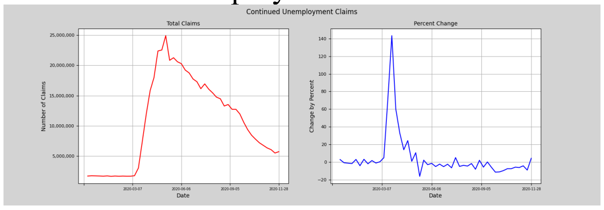
a. covid-cases-chart.png (1 visualization)
    i. Displays daily percentage change of new cases over time

b. dji-chart.png (2 visualizations)
    i. Displays daily percentage change, daily trading ranges, and averages of both over time

c. unemployment-ICSA-chart.png (2 visualizations)
    i. Displays weekly total new initial claims and percent change over time

d. unemployment-CCSA-chart.png (2 visualizations)

i. Displays weekly total new continued claims and percent change over time

Percent change in positive COVID-19 cases 	Percent change and ranges of DJI 
Total and percent change of initial unemployment claims 	Total and percent change of continuing unemployment claims 

5. **CODE DOCUMENTATION** (explain what each function does)

main.py

- This file calls all of our individual files, angie.py, sean.py, and nick.py.

angie.py *(COVID-19 Cases from covidtracking.com API)*

- **getData(url)**
  - This function takes a url, in a string, format as a parameter. The function first sends a request to the specified url, in this case is the Covid-19 API, for its information and saves this as a response object. Then the response object is converted into a python dictionary using json. It then iterates through the response

dictionary and adds the dates and positive cases data into their respective lists. Then all the dates are out into a universal format, for example 2020-01-01. Lastly, the two lists of dates and positive cases are returned as a tuple.

- **createDbTable(dbname)**
  - This function takes a name of a database in string format as its one parameter. The function first establishes a connection and cursor with the database using the parameter name given. Then using the cursor, if a table called positive_cases does not already exist, the function will create that table with the columns id, date, and positives.
- **getID(dbname)**
  - This function takes a name of a database in string format as its one parameter. The function first establishes a connection and cursor with the database using the parameter name given. An empty list is initialized and then iterates through all the ids selected from the positive_cases table in the database and appends them to the empty list. Then a try and except statement is used to set the variable iD to last id added into the table or, if the table is empty, the iD will b set to zero. Lastly, the function returns iD.
- **dataIntoDB(dbname, url)**
  - This function requires two parameters, the database name as a string and a url for the website you're requesting data from as a string. The function first establishes a connection and cursor with the database using the parameter name given. Then the previously defined function getID is called and saved to a variable iD. Next the previously defined function getData is called and that tuple it returns is split into two lists, one for the dates and one for the positive Covid-19 cases. A limit variable is set to 8 which will limit how many items are entered into the database per execution. A threshold variable is set which indicates at what point in the table we want to add the rest of the data all at once. Then the function checks if the iD is greater than the threshold, if so it inserts the rest of the data from our lists into the table. Otherwise, only 8 more datapoints are added into the table. Finally, the changes are committed to the data and the cursor and connection are closed.

- **calcPercChange(dbname)**
    - This function takes a name of a database in string format as its one parameter. The function first establishes a connection and cursor with the database using the parameter name given. The dates and positive cases are selected from the database, excluding the first datapoint because that day there are zero positive cases and we can not divide by zero. Then the function loops through the length of the list of data, minus one, calculates the percent change in positive cases for each day and appends the calculations and date to new empty lists. The two lists are appended to one output list and that output list is returned.
- **writeToCsv(outFileName, url)**
    - This function takes a string for what you want the name of the csv file to be and a url for the website you're requesting data from as a string. The previously defined functions getData and calcPercChange are called. A pathway to the intended directory for the file is established. A file for writing is opened and column headers, date and percent change in positive Covid-19 cases, are added. Then it loops through the calculated data and for each one writes a new row to the file, date and calculated percent change.
- **makeVis(dbname)**
    - This function takes a name of a database in string format as its one parameter. The previously defined function calcPercChange is called and the list it returns is split into two separate lists, dates and calculated values. Then using matplotlib module, a figure is created. The figure has the dates on the x-axis and calculated values on the y-axis. The styles of the figure is change by updating the colors, adding a title, and changing the font sizes. Tick marks are also added on the x-axis to make the information there more readable. Lastly, the figure is saved to the current directory.
- **angMain()**
    - This function takes no parameters and initializes variables that will be used as parameters for other functions in the file, like dbname and url. Then all the previously defined functions are called.

sean.py *(Dow Jones Industrial Data from Yahoo Finance Website)*

- **get_stock_data()**
  - Retrieves data from yahoo finance and puts the data into a pandas data frame format. It will change the pandas data frame into numerous lists and will change the date formats to match other partners dates format. This function will return a Dates, Volume, High Price, Low Price and Close_Price List
- **start_db(dbname)**
  - Takes parameter dbname which is the name of the database. It will create, if not already present, the tables to put into the database (Dates,Volumes,High Price, Low Price and Close_Price). This function does not return anything
- **check_ID(dbname)**
  - Its purpose is to see how many items are stored in the database. This function will make a list of all the ID #s in the database and will return the highest number. This number indicates how many items are in the list. The function will return the ID as an integer
- **insert_Data(dbname)**
  - This function will insert data from yahoo finance into the database. If the item count in the database (based on the ID) is less than 33.3, this function will insert 8 items at a time into the database. After the count is greater than 33.3, the remaining Yahoo Finance data (retrieved using the get_stock_data function) will be inserted into the database. This function does not return anything
- **calculate(dbname)**
  - Function that performs calculations from the data in the SQLITE database. It will select Close_Price Data and turn the data into a numpy array to make calculations easier. This function calculates the mean of Close_Price Data with numpy. Additionally, this function will select High and Low Price Data, turn each into numpy arrays, and make a numpy array list of High Price - Low Price for each date. Using numpy, this function calculates the mean Range (High Price - Low Price) from the given data. This function makes a list of the mean Range (the list contains the same number, with a length of the total rows in database. This makes it convenient to graph the mean.) The function also calculates percent change of

close price per day (compared to the previous day) using numpy arrays. Additionally, this function will calculate the mean percent change per day using numpy and makes a list of the mean percent difference (the list contains the same number, with a length of the total rows in database. This makes it convenient to graph the mean). This function returns the Close_Price Percent Change,Close Price Percent Change Mean, High Price - Low Price and mean high Price - Low Price lists.

- **write_to_CSV(outfile)**
    - This function will write to the outfile variable file the data from the calculate function. It will write the index names of the 4 rows and will write each list from the calculate function into the csv file
- **visualize(dname)**
    - This function will visualize calculated data. The first graph will use as the x axis the dates and will graph a series of the close_price percent change data. A grid, axis labels, legend and title are included. The function will also plot the mean close_price percent change. The second graph will use as the x axis the dates and will graph a series of the high price - low price differences. A grid, axis labels, legend and title are included. The function will also plot the mean range (high price - low price) .

nick.py *(Initial/Continued Unemployment Claims from Federal Reserve API)*

- **_uecGetOffset(conn, cur, series_id)** *(joined tables)*
    - Use: helper function that determines the row count in the database *for a specific claim type*. Tables are joined to limit claims by type.
    - Input: db connection (object), db cursor (object), and a Federal Reserve-defined “series_id” (string)
    - Returns: current row count of joined records of two related tables (int)
- **_uecGetMaxId(conn, cur, table, primary_key)**
    - Use: helper function that returns the current max primary_key of all rows in a given table

  - Input: db connection (object), db cursor (object), table (string), primary_key (string)
  - Returns: max primary key, or “0” if empty (int)
- **uecGraphClaims(series_id, datum, title)**
  - Use: visualizes data from any Federal Reserve API series
  - Input: Federal Reserve API “series_id” (string), list of dates and values (list), and a title (string)
  - Output: saves one PNG chart image in the current directory (file)
  - Returns: nothing
- **uecGetPercentChange(conn, cur, series_id)** *(joined tables)*
  - Use: calculates percent change of total initial and continuing unemployment claims.
  - Contains: **a database join** to identify weather claims being measured are initial claims and continued claims
  - Input: db connection (object), db cursor (object), and Federal Reserve API series_id (string)
  - Returns: a list containing two equal-length lists of dates and percentages (list)
- **uecReadCsv(inFileName)**
  - Use: reads calculation csv files and returns the data as a list
  - Input: inFileName (string)
  - Returns: a list containing two equal-length lists of dates and percentages (list)
- **uecWriteCsv(outFileName, datum)**
  - Use: writes calculated data to a csv file
  - Input: outFileName (string), datum (list)
  - Output: saves a single csv file in the current directory (file)
  - Returns: nothing
- **uecGetTotalChange(conn, cur, series_id)** *(joined tables)*
  - Use: queries database to calculate the changes between weekly measurement periods
  - Contains: **a database join** to identify weather claims being measured are initial claims and continued claims

  - Input: db connection (object), db cursor (object), and Federal Reserve API series_id (string)
  - Returns: a list of dates and claims (list)
- **uecInsertClaimClaims(conn, cur, series_id, datum)**
  - Use: inserts a list of one or more claims (a date and total new claims) for a particular Federal Reserve series into the database
  - Input: db connection (object), db cursor (object), and Federal Reserve API series_id (string), datum (list of dates and claims)
  - Output: saves data to the database
  - Returns: nothing
- **uecGetData(conn, cur, settings, series_id)**
  - Use: grabs series data from a rate-limited Federal Reserve API with a dictionary of "settings" acting as API parameters.
  - Input: db connection (object), db cursor (object), and Federal Reserve API series_id (string)
  - Returns: list of data (list)
- **uecInsertClaimTypes(conn, cur)**
  - Use: one-time insertion (only upon the first program execution) of two rows indicating two types of claims, relating claim types to another table containing claim totals: "Initial claims" and "Continued claims"
  - Input: db connection (object), db cursor (object)
  - Output: inserts two records into claim_types
  - Returns: nothing
- **uecCreateTables(conn, cur)**
  - Use: initializes two related tables (if not already existing): claim_claims and claim_types
  - Input: db connection (object), db cursor (object)
  - Output: creates two tables: claim_claims and claim_types
  - Returns: nothing
- **uecAppTask(conn, cur, settings, series_id, title)**

  - Use: initializes data retrieval, allows multiple series calculations and visualizations
  - Input: db connection (object), db cursor (object), series_id (string), title (string)
  - Output: called csv and visualization functions
  - Returns: nothing
- **uecApp(conn, cur)**
  - Use: calls functions to initialize other functions: creates two tables and inserts two initial claim_type rows, sets a db/api rate limit threshold, determines db/api limits beyond the 100-row db threshold, declares API parameters, and calls sequential "tasks" to calculate and visualize each series API call.
  - Input: db connection (object), db cursor (object)
  - Output: output from called functions (csv files, chart files, db rows)
- **uecMain()**
  - Use: sets up the database and cursor object and calls uecApp and closes the database connection
  - Input: none
  - Output: output from called uecApp function
  - Returns: nothing

6. DOCUMENTATION OF RESOURCES

Date	**Issue Description**	**Location of Resource**	**Result**
12/1/20	Needed to get data about positive Covid-19 cases	https://covidtracking.com/data/api	Yes

12/1/20	Needed API data about initial unemployment claims	https://fred.stlouisfed.org/series/ICSA	Yes
12/1/20	Needed website-scraped data for Dow Jones Index	https://finance.yahoo.com/quote/%5EDJI?p=^DJI	Yes
12/1/20	Needed API data about continued unemployment claims	https://fred.stlouisfed.org/series/CCSA	Yes
12/2/20	How to use numpy array	https://www.w3schools.com/python/python_ml_getting_started.asp	Yes
12/3/20	How to specify range of dates to scrape data	https://pandas-datareader.readthedocs.io/en/latest/remote_data.html	Yes
12/3/20	How to format date time series strings	https://www.w3schools.com/python/python_datetime.asp	Yes
12/9/20	How to style a matplotlib line graph with adding tick marks.	https://www.tutorialspoint.com/matplotlib/matplotlib_setting_ticks_and_tick_labels.htm	Yes

12/5/20	How to create a database in SQLite using python	https://www.w3schools.com/python/python_mysql_create_db.asp	Yes
12/10/20	How format a SQLite join statement using python and what does a join statement do	https://www.w3schools.com/python/python_mysql_join.asp	Yes
12/11/20	Reversing graph axis	https://stackoverflow.com/questions/2051744/reverse-y-axis-in-pyplot	Yes
12/11/20	Widening graph intervals	https://stackoverflow.com/questions/12608788/changing-the-tick-frequency-on-x-or-y-axis-in-matplotlib	Yes
12/11/20	Setting a sleep delay in Python to avoid frequency rate limitations	https://realpython.com/python-sleep/	Yes
12/12/20	Embarrassingly, needed to review how to calculate percent change (original calculation was very off)	https://www.mathsisfun.com/numbers/percentage-change.html	Yes

7. EXECUTION INSTRUCTIONS:

a. First, make sure the python version is 3.9.0 or higher.

b. Second, be sure these python modules are installed:
   i. csv
   ii. datetime
   iii. json
   iv. matplotlib
   v. numpy
   vi. os
   vii. pandas
   viii. pandas_datareader
   ix. requests
   x. sqlite3
   xi. time

c. Then, in a terminal window, run "python3 main.py" **FOUR** times. This populates the database no more than 25 rows at a time, for a total of 90 rows, which is below the 100 row limit. You should see in the same directory as main.py:

   i. One database:
      1. db.sqlite
   ii. Six *.csv files:
      1. covid-cases.csv
      2. dji.csv
      3. unemployment-ICSA-total.csv
      4. unemployment-ICSA-percentages.csv
      5. unemployment-CCSA-total.csv
      6. unemployment-CCSA-percentages.csv
   iii. Four *.png chart files:
      1. covid-cases-chart.png (one visualization)
      2. dji-chart.png (two visualizations)
      3. unemployment-ICSA-chart.png (two visualizations)

4. unemployment-CCSA-chart.png (two visualizations)

d. Again, run “python3 main.py” **ONE** time. This enters the next 25 records and the database will now contain 112 records, which is above the 100 row limit.
   i. Database, *.csv, *.png will automatically update and re-save.

e. Finally—once again—run “python3 main.py” **ONE MORE** time (this should be the sixth execution). The database, csv, and charts will populate significantly more data for better insights.
   i. Database, *.csv, *.png will automatically update and re-save.

8. ILLUSTRATION OF DATABASE GROWTH

a. Each run of the program up to five times will only insert 25 or fewer records into the database. After the fifth run, an arbitrary number of additional records will be inserted into the database.
b. Table of database table growth, in total row count:

Program execution	**DJI_data** (table row count)	**positive_cases** (table row count)	**claim_claims** (table row count)	**claim_types** (table row count)	**Total database** (row count)
1	8	8	6	2	24
2	16	16	12	2	46
3	24	24	18	2	68
4	32	32	24	2	90
5	40	40	30	2	112
6	Arbitrary	Arbitrary	Arbitrary	2	Arbitrary