Team Machos: Rebecca Mao & Shin Cho

API's, SQL's, and Visualizations

https://github.com/scho13/206_final_project_machos

Original Goals:

Our original goal was to use the Fiscal Data API and the Coinpaprika API. We wanted to use the Fiscal Data API to pull the U.S Government's financial reports. We wanted to collect the total net cost (in the billions) for a certain year. This net cost information will be pulled for the past 10 years (September 30, 2011-2021). From the Coin Paprika API, we will be collecting the "close" value (the value of the Bitcoin at the end of the day) for the past 10 years (September 30, 2011-2021). We then wanted to compare these two values and seeing if there is a correlation between the two.

Achieved Goals:

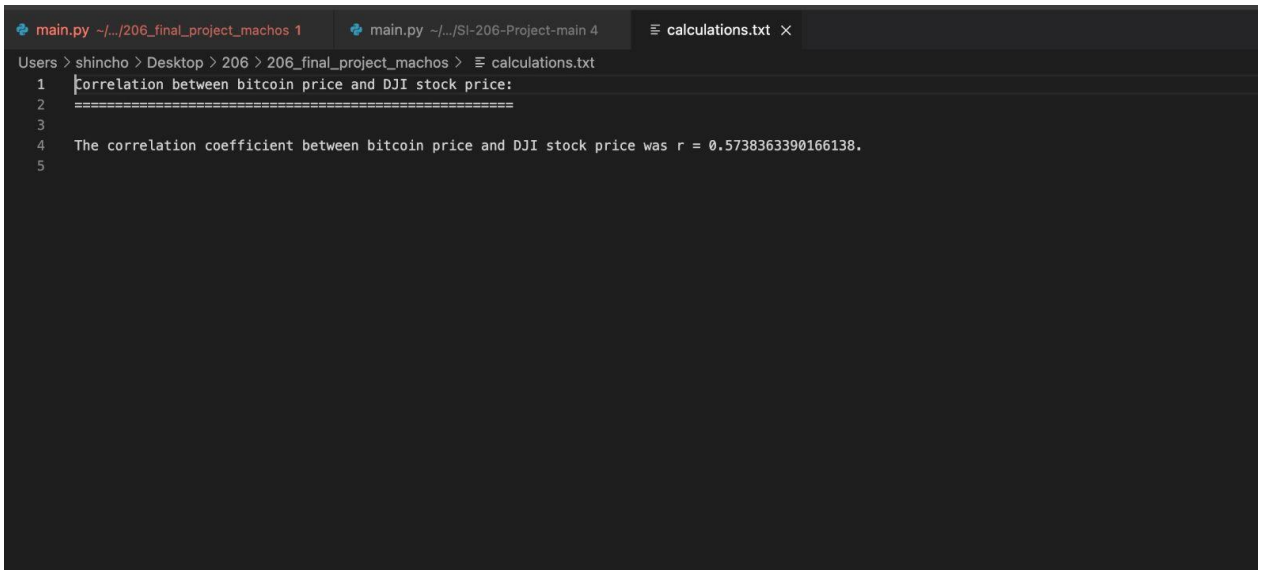
As we began to work on our project, we ended up having to change the direction our project was going due to limitations of one of our APIs. Since we needed at least 100 rows collected from each API, we had to increased the time frame when requesting data from the APIs. Yet, the Fiscal Data only has a financial report per year on September 30th. Bitcoin, on the other hand could be collected from any day but we only had data for Bitcoin starting in 2010. Thus, we'd be able to reach 100 rows of data for Bitcoin but not for the Fiscal Data. As a result, we decided to not use the Fiscal Data and instead found a stock API called Twelvedata API. This API collects the price of the Dow Jones Industrial Average stock price. We decided to use DJIA because it is a stock index of 30 of the most traded stocks in the United States. It's used as one way to measure the direction of the stock market. We set our time frame to be from January 1st, 2021 to December 31st, 2021. We stored all data using 25 rows at a time, resulting in 150 rows each for our 2 tables. We also created 2 visualizations.

Problems Faced:

We had difficulty in limiting the data to 25 rows at a time when inserting it into the database. We originally limited the amount of data returned when requesting from the APIs to 25 but found that once adding that into the database we were unable to requests the next 25 rows of data without having to change the request url. We solved this problem by pulling all the data from the API all at once and only appending 25 to a list. From the list we would then add the data into the database. Using "if" statements to check how many rows were in the database already, we would add an integer to add to how much the database would add in. Another problem we faced was that the Coinpaprika API collected bitcoin values everyday while the Twelvedata API only collected data from the weekdays and non-holidays since the stock market is closed on weekends and holidays. This meant that when we matches the rows based on the date, the bitcoin table had more data than the stock table. So, if we limited the rows to 100 exactly, the dates would not match up. We fixed this problem by adding in more rows to the database to ensure that we had enough data points from each table that had a shared key.

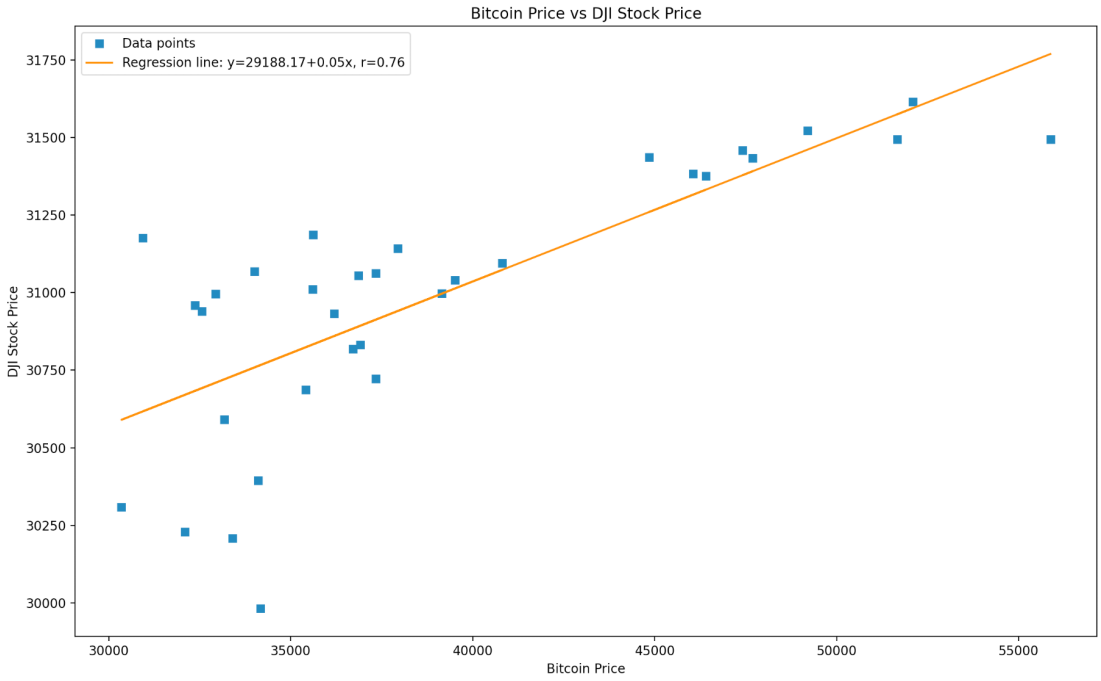
Calculations:

1. Data calculated from find the correlation coefficient between bitcoin price and DJI stock price in the 2021 year:

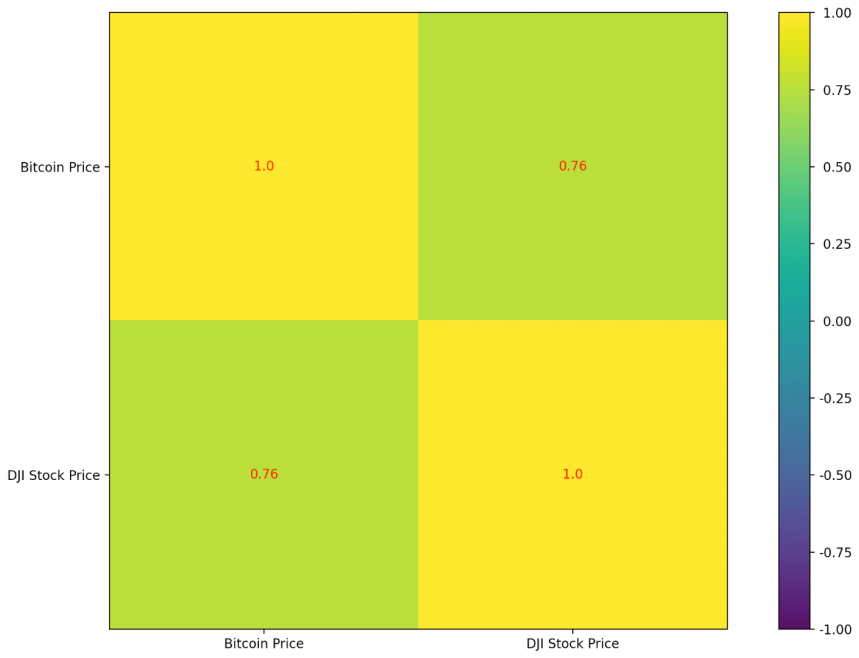


Visualizations:

1. Scatterplot with regression line of the correlation between Bitcoin Price and Dows Jones Industrial Average Stock Price in 2021



2. Heat Map of the correlation coefficient between Bitcoin Price and Dows Jones Industrial Average Stock Price in 2021



Instructions for Running Code:

To run our code, make sure you have some form of SQL reader installed on your computer so you are able to view the database and see the tables that we created.

1. Run through the main.py file 3 times in order to fill in the database to have 150 rows for each table.
2. Once you run the code, you will be able to see our 2 visualizations and the csv files with the calculations included.

Function Documentation:

Database:

a) def setUpDatabase(db_name):
   i) Takes the database name as an input. Sets up the database and creates a directory
   ii) Returns database curser and connector

DJIA Stock Data:

b) def stock_api():
   i) Takes nothing as input. Uses request module to collect DJIA stock prices from the Twelvedata API
   ii) Returns DJIA stock data for a specified date range in JSON formatting

c) def create_stock_table(cur, conn):
   i) Takes in database cursor and connector as inputs
   ii) Returns nothing. Creates a new table called ‘Stock’.

d) def add_into_stock_table(cur, conn, add):

i) Takes in database cursor, database connector, and specified integer as inputs. The function calls the stock_api() and takes the JSON formatted data. It loops through the data from a starting point to an end point(limit) and indexes the wanted data
ii) Indexes the date, opening stock price, high stock price, low stock price, and closing stock price
iii) It will then append the wanted data as a tuple to a list. Then, it will loop through the list of tuples and insert it into the Stock table.
iv) Each time it runs, an integer is added to the start and end point based on how many rows are already in the database. This ensures that adding into the database is limited to 25 rows each time the code is ran.
v) Returns nothing

Bitcoin Data:

e) def bitcoin_api():
   i) Takes in nothing as input. Uses requests module to collect bitcoin price data from Coinpaprika API.
   ii) Returns bitcoin data for a specified date range in JSON formatting
f) def create_bitcoin_table(cur, conn):
   i) Takes in database cursor and connector as inputs
   ii) Returns nothing. Creates a new table called 'Bitcoin'.
g) def add_into_bitcoin_table(cur, conn, add):
   i) Takes in database cursor, database connector, and specified integer as inputs. The function calls the stock_api() and takes the JSON formatted data. It loops through the data from a starting point to an end point(limit) and indexes the wanted data
   ii) Indexes the date, opening bitcoin price, high bitcoin price, low bitcoin price, and closing bitcoin price
   iii) It will then append the wanted data as a tuple to a list. Then, it will loop through the list of tuples and insert it into the Bitcoin table.
   iv) Each time it runs, an integer is added to the start and end point based on how many rows are already in the database. This ensures that adding into the database is limited to 25 rows each time the code is ran.
   v) Returns nothing

Calculations:

h) def join_tables(cur, conn):
   i) Takes in database cursor and database connector as inputs
   ii) Uses JOIN to return a list of tuples in the format (date, bitcoin closing price, stock closing price) where the dates are the same
i) def correlation_calc(list_of_tuple):
   i) Takes in the list of tuples from the join_tables() function as input. Calculates the correlation coefficient between the bitcoin closing price and DJIA stock closing price
   ii) Returns the correlation coefficient as an integer
j) def write_correlation_calc(filename, correlation):

i) Takes in a filename as a string and the correlation coefficient that is returned from the carrelation_calc() function.
ii) Returns a text file ('calculations.txt') that writes the correlation coefficient between bitcoin closing prices and DJIA closing prices.

Visualizations/Graphs:

k) def create_regression_line(list_of_tuple):
   i) Takes in the list of tuples returned from the join_tables() function as input.
   ii) Uses Matplotlib to create and return a scatter plot with a regression line of the correlation between bitcoin price and DJIA stock price.

l) def create_heat_map(list_of_tuple):
   i) Takes in the list of tuples returned from the join_tables() function as input.
   ii) Returns a heat map that uses a color scale to show the correlation coefficient between Bitcoin Price and Dows Jones Industrial Average Stock Price in 2021.

Main

m) def main():
   i) Takes in no input and returns nothing.
   ii) Calls the setUpDatabase(), create_bitcoin_table(), create_stock_table(), join_tables(), correlation_calc(), write_correlation_calc(), create_regression_line(), create_heat_map() functions
   iii) Limits the amount of data stored in the database to 25 rows at a time

Resources Used:

Date	Issue Decription	Location of Resource	Result
4/7	How to use Bitcoin API	https://api.coinpaprika.com/	Yes, allowed us to learn the documentation and find the opening, high, low, and closing bitcoin price for a specified date range.
4/7	How to use Fiscal Data API	https://fiscaldata.treasury.gov/api-documentation/	It was a helpful resourced to understand the U.S financial reports but we ended up not using this API.
4/12	How to use Stock API	https://twelvedata.com/docs#core-data	Yes, allowed us to learn the documentation and find the opening, high, low, and closing bitcoin price for a specified date range.

4/16	Was confused on what it meant to limit 25 rows at a time when adding it into a database and how to go about it	https://piazza.com/class/kxxn6pvwt194p7?cid=503	Yes, the instructor's response to the student's piazza post was very helpful and we utilized it by appending the data into a list and then adding it into the database.
4/18	Didn't know how to find how many rows were in our database	https://www.navicat.com/en/company/aboutus/blog/695-getting-row-counts-in-mysql-part-1#:~:text=To%20counts%20all%20of%20the,returned%20by%20a%20SELECT%20statement.	Yes, this resolved our issue as we learned the COUNT(*) will return the number of rows in a table.
4/21	Didn't know how to calculate the correlation coefficient and make a scatterplot with a regression line	https://realpython.com/numpy-scipy-pandas-correlation-python/	Yes, this resource gave a really good template to how we should outline our code to do the calculations
4/21	Was confused on how to create a heatmap that will show the correlation between our two data variables	https://likegeeks.com/python-correlation-matrix/	Yes, this resource was really helpful in explaining what a heat map is and how to structure it in code.