Benjamin Holmes

Cryptocurrency Trend Tracker via Social Media

2022 1

Benjamin Holmes

Contents

Benjamin Holmes

Benjamin Holmes

Analysis of the problem

Problem Identification

This project aims to help small-time investors make quantitatively informed decisions on cryptocurrency investments. Opinion based insight is currently a major factor in cryptocurrency investments. As qualitative data, this comes with risks. Through the social media platform Twitter, which allows users to ‘tweet’ updates, opinions, news and other information, this project will collect data from this platform, analyze the data, and finally present the findings in an accessible manner, all using computational methods. This project is important to small-time investors because of the lack of large, reliable data on the market that is free and accessible to use. There is also no simple and small scale software that can be applied quickly and effectively to a lesser known crypto coin, which may allow for trending, new and ‘hot’ coins to be found in their early days for maximum profit. This, in turn, calls for an encompassing solution that offers these services via a computational approach. Using this approach allows for formulating problems in a way that enables us to use a computer and other tools to help solve them as well as logically organizing and analyzing data.

Stakeholders

Stakeholder Backgrounds

Joe is a 6th former who has a part-time job at a local tennis club. Living with his parents whilst he is still at school, he has minimal expenses in his day-to-day life. This leaves him with cash in his bank account that sits and collects small amounts of interest. Recently, Joe has become more aware of creating multiple sources of passive income - in order to make his money work for him. As Joe doesn’t have the amount of capital needed to start investing through brokerage firms, he has begun to look for smaller, less complex ways of investing. Eventually, he came across the cryptocurrency market - through exchanges such as Binance or Coinbase he can easily control his cash: where it is invested, when it is invested and how much is invested. Despite the more volatile market, perhaps requiring more personal attention and awareness of the crypto market, he started to enjoy the process of managing his crypto coins. Feeling responsible for his own wealth, he wants to be more certain of his choices of crypto coin to put money into. Especially with recent sudden movements in the market due to fast increases and decreases in popularity of different coins, he wants to make sure he can also quickly spot these trends in order to buy and sell crypto to maximize profits and minimize losses. He currently uses various forums and websites such as Reddit where he reads people's opinions on different coins and bases his investments on these assessments combined with his own qualitative analysis of the coin. He would prefer something that uses a more mathematical approach to statistically tell him whether he should invest in a coin depending on its popularity.

Andy is a social media analyst who helps businesses use social media to help determine the popularity of their product. He is interested in how social media can affect people's decision

Benjamin Holmes

making in spending money. Seeing the interest in cryptocurrencies rise, he has decided to invest a small amount of money into Bitcoin to see what would happen. Having a full-time job, he doesn't have the time to be constantly checking the Bitcoin price and tracking how well his investment is doing. Despite not monitoring the coin all the time, he would like to be able to know if there is a drastic change in popularity that may affect his investment. Whether this is a sudden upwards or downwards trend, he would like to be aware so that he could either invest more into, or sell, his Bitcoin.

Stakeholder Interviews

Question	Joe	Andy
How and when would you use this software?	"I think I would use this software to diversify my choices on investments. For example, before making a trade, I would use this software to almost confirm a decision. Perhaps also to check out potential investments and see if there are any coins that have a massive increase in interest"	"As I don't frequently check how my investments are doing I feel this software will be mostly used to either notify me of a sudden spike in popularity of a coin, or, to be able to double check the popularity of a coin before I buy""
How would you access this software?	"I carry around a laptop so however really. I mean ideally a website rather than a program which I'll have to load up each time I want to check"	"Preferably on my phone on an app - that's quite hard though. I'd settle for a usable website that loads quickly"
How should inputs be delivered?	"Ideally I should be able to enter a keyword or coin name and results come from that alone - perhaps a date time frame as well so I can track long or short term trends"	"Minimal inputs really - I'd like to be able to see a cover page of the most popular or trending coins so I can quickly see what's going on. Past that, maybe a search feature to find a rarer or more specific coin."
Who will use this software?	"I would definitely use it but I reckon my friends interested in crypto could definitely get on board - basically newbies and people my age"	"I think people like me will be thankful for a simple software like this - a quick check once in a while to tell them what's going on. Hopefully giving us the chance to catch some of

Benjamin Holmes

		these massive sudden spikes early. In other words, longer term investors who are just exploring this avenue of investing and aren't too personally invested ”
How should outputs be delivered?	“Nicely presented and simple would be best. Lots of graphs that clearly show changes in popularity with perhaps some explanations of certain terms for the complete beginners”	“Smooth and well designed minimalism - to be able to show what needs to be shown in the most accessible way. This means no long paragraphs and pictures - pure crypto content laid out professionally”
What should the end result be?	“Overall I think the final product should be a simple website with popularity data compared with real price crypto graphs. This information should be easily accessible in every way with proper information that I can use to make decisions”	“At the end of the project I’d like to see a working program that collects data, analyzes it and presents it extremely well. It should be quick to load as well as provide insightful data that can’t be found through simply looking through people’s opinions on Reddit”

Researching the problem

Similar problems and solutions

Currently there is no simple software that would allow users to easily see collected popularity data on various crypto coins - let alone fast, free, reliable and accessible. There is no proper middle ground between price tracking websites such as CoinMarketCap and cryptocurrency exchanges such as Binance. Currently users rely on their own market research: articles, word of mouth, online opinion. Obviously, money can still be made this way. However, by qualitatively analyzing large amounts of these ‘online opinions’, a much broader and reliable evaluation of specific coins can be gathered.

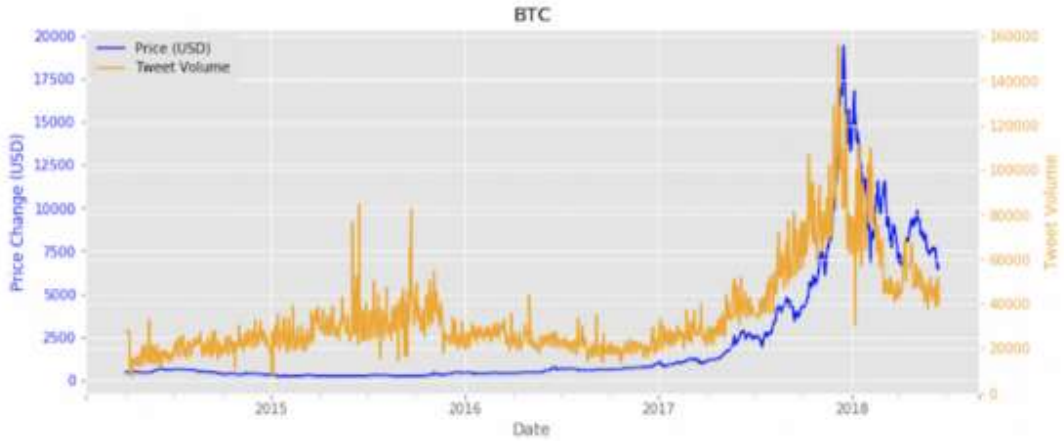
The main group to benefit from this software will be students ages (18-22) who are wanting to invest small amounts of money - as risk free as possible. It will also help beginners, in general, wanting to explore and experiment with cryptocurrency trading. By making investing in the market simple and based less on personal research, it will allow small-scale traders to be confident and comfortable when investing as they know their investment is based on numerical data. Furthermore, it will allow for longer term investors to keep an eye on their investments

without having to put aside time in a busy schedule to ensure their coin is safe. By gathering large volumes of data points, it also means that extreme opinions can be treated as anomalies, therefore preventing the skewing of a person's unconscious opinion on a crypto coin. This will allow for logic-based investing which is safer for protecting capital. As this system is aiming to be accessible to beginners, it will moreover be important to include explanations and definitions to allow for better user understanding in the final software.

Although some academic research has been done on the relationship between online popularity and real-life market price, it has not resulted in open-source software that can be used by small-time investors to better understand the popularity of specific coins. For example, one study observed 62.48% accuracy when making predictions based on bitcoin-related tweet sentiment and historical bitcoin price. This was done by downloading historic prices of Bitcoin and collecting Bitcoin-related Tweets, followed by performing tweet sentiment analysis and calculating the relationship between Tweet sentiment and price direction. The Random Forest Regression binary classification model was then applied, with diverse inputs and evaluated outputs in order to compare an error accounted graph.

Another paper discussed whether the increase in the cost of Bitcoin is linked to the number of tweets or the results of Web Search media. They compared cost patterns with Google Trend Data, tweet volume, and in particular with positive tweets. They gathered more than 1,900,000 tweets, analyzed them then marked the positive and negative tweets. They found a correlation between Bitcoin price and the number of tweets at the same time, between Bitcoin Price and Positive Tweets, and Bitcoin Price with Google Trend.

Benjamin Holmes



Demographic Questionnaire

In order to identify public opinions on a new software being developed to aid investment decisions in the cryptocurrency market, I conducted a questionnaire that gathered a range of information that allowed me to better understand the demographics that this software will be catering to. I also interviewed stakeholders to elicit requirements for the software. The reason I chose a questionnaire to gather information was due to wanting a larger set of data that could be collected quickly and reflect a general population well. By offering multiple choices it narrows down responses and makes it an easier process for participants to complete - especially due to the fact questionnaires are emailed directly to potential participants.
As a flowchart type style, the questionnaire gradually narrows down to a smaller group who would actually use the software, whilst finding useful information and reasons why other groups wouldn't use the software. This allows for simpler analysis - as percentages of age, financial status and type of investor.

Interviews with stakeholders allows for a much more personal elicitation of requirements. By asking questions designed to provoke a broad range of responses it means that a detailed requirement specification can be built, which suits the needs of two different types of user.

How old are you?(501 people)
18-22: 328
23-40: 127
41-60: 35
61+: 11

Do you have extra money you would be willing to invest? (501 people asked)
Yes: 259

Benjamin Holmes

Would you invest this money in cryptocurrency?
Yes: 105

Why do you invest in cryptocurrency?
I'm experimenting: 37
I want to make money in the long term: 19
I want to make quick money: 29
I want to diversify my portfolio: 20

How do you currently make decisions on where to invest in cryptocurrency?
Youtube: 14
Reddit: 21
Twitter: 33
Friends and Family: 24
Articles: 10
Someone else invests for me: 3

Would you use a service that uses social media data to analyse other investors' opinions on various crypto coins?
Yes: 80
No: 25

No: 154

Why wouldn't you invest in cryptocurrency?
It's too risky: 41
I don't know how: 53
I don't have the time: 22
I wouldn't know what to invest in: 38

Specifying the proposed solution

From the questionnaire we can see that while roughly half of those who responded have money they would be willing to invest, only 40% of this population would invest this money in cryptocurrencies. This is probably due to the unreliability of the market - cryptocurrencies have large fluctuations in value and the risk of losing money is high, even though some investors have had high returns. Perhaps this pattern of staying away from risk comes from limited funds of the youth which, as we can see, make up roughly 65% of the surveyed population.

When people were asked why they wouldn't invest in cryptocurrencies, 26% responded by saying it's too risky, 34% didn't know how to invest, and 24% wouldn't know what to invest in.

Benjamin Holmes

Fairly evenly distributed between these three, it reiterates the uncertainty stemming from crypto currency investments. This may be due to a stigma of fortunes being made but also dramatically lost, when investing as a newbie or veteran in crypto. A well designed software should be able to reduce these uncertainties by providing extra information aimed at beginners, as well as making graphs and data presented in such a way that more clearly demonstrates which coins are the best for the purpose the investor intends.

Moving on to reasons why people invest in cryptocurrency, the survey suggests that most people are experimenting. As a fairly new market it is not surprising that most people will be trying out trading rather than treating it as a regular source of income. The second highest vote was to make quick money - this is almost an opposite to the risk free strategy of the youth. Seeking quick money often results in more risk despite the potential for greater rewards. Software that is useful for the whole cryptocurrency investing community should be able to spot and display quick money opportunities, as well as reliable, long term investments.

Looking at the ‘sources of information’ question, the survey showed that most of the current crypto investors rely on opinion based qualitative analysis to make green trades. With Reddit and Twitter being the most common social media sources, surprisingly 23% of people said their trading information came from family and friends. This shows how people speak about a coin, whether on a social media platform or amongst their social circle, has a strong influence on investment decisions. By providing a service that uses real data to predict crypto prices, it should allow for more success and reliability of trades for everyone - whether they gather their insights online or offline.

Finally, 76% of investors I surveyed would use a software I intend to produce. Although it is positive that ¾ of this population would use this software, I could have improved upon this questionnaire by asking why the remaining ¼ wouldn’t. Speculation could suggest they feel confident in their current methods of information gathering, or, perhaps, they are unsure that data from social media accurately reflects the current market. To meet the needs of the group that says it wouldn’t use this software due confidence in other methods, this project should clearly demonstrate its value so that it may be used by them in the future, at least in combination with other methods, to produce maximum results.

The two stakeholders I interviewed provided useful insight into two different types of users, allowing for a wider range of system requirements. For example Joe, as a hobbyist, is interested in lots of information provided quickly and accurately.Andy, however, is looking for software that spots price changes reliably - that can call him back into the market to change his investments at a moment's notice.

User Interface Stakeholder Requirements

Benjamin Holmes

Requirement	Explanation
Easily accessible - Definitions of terms - Labeled Graphs - Instruction Manual	The user can easily decipher what is going on, with descriptions for beginners
Quick, responsive design - Fast load up - Minimal large files	So the user can easily check up on prices with quick loading times
Search feature - Coin tags or names - Date input style lenient	To look up specific coins within a time frame
Home page - Big displays of trending coins - Links to other areas of the site	Displays trending/'hot' coins that a user should highly consider investing into based off popularity
Graphical comparison - Easy to read graphs with time frame - Side by side or graphical overlay - Option to see real price graphs only	Real crypto graphs compared with popularity graphs to easily see how they compare
Simple design - Good colour scheme - Layout	To make content easier to comprehend, without distracting and off putting UI
Software Compatibility - Up-to-date system - Runs online with modern website features	Allows the user to access the site anywhere from any search engine or device

From the information collected from the public and the two stakeholders, I was able to elicit success criteria for this project. These will be specific, measurable, achievable, realistic and timebound in order to be clear about objectives and completion. In this case, the timebound nature of the project is the deadline, and the criteria being reached will be shown through screenshots.

Success Criteria

Criteria	Justification
1 Connect Python to Twitter API **1.1** Get access tokens and consumer keys **1.2** Install Libraries **1.3** Pull tweets	The best way to use Twitter as a platform to collect data, efficiently and without breaching terms of service is through its custom made API designed for that purpose. Other ways

	could include web scrolling and cursor movement, however this is slow and is limited by Twitter's front end performance speeds.
2 Clean Data from Twitter **2.1** Remove URLS **2.2** Remove strange syntax	As Twitter data is scrapped from the public, it does not come raw with only the necessary elements. This calls for a cleansing of symbols and other features that compose a tweet, fundamentally written by a human. By cleaning this data it allows for the program to go on to analyze more effectively, by-passing unused characters.
3 Calculate Sentiment Polarity **3.1** Install Textblob **3.2** Analyze Tweets	In order to gauge the public's opinion on a topic, the emotion found behind text, in this case, a tweet, must be found. This can be used to quantify qualitative data into a medium that is easier for a program to manipulate and present in a way that is accessible to users.
4 Sentiment Values **4.1** Average daily values **4.2** Store **4.3** Plot	Sentiment values must be stored for future use so that long term data collection can occur over time. By calculating an average daily value, it limits the amount of data processes the computer has to do and therefore speeds up display time. Although this will truncate extreme values, this is helpful to provide a more general view of a coin's popularity in a day.
5 Get real crypto prices **5.1** Choose time frame **5.2** Choose coin **5.3** Calculate average of open, high, low and close **5.4** Plot	Collecting real crypto prices, through Python means that data is easily accessible for other aspects of integration, in further development, which compares the two and makes predictions. By using an average of open, high, low and close it summarizes data into one data point instead of four. This speeds up plotting time which is important if many graphs will be accessed quickly
6 Compare Plots	Comparing plots allows for a combination of graphs to see if popularity reflects the coin or visa versa. This will result in the software being used to make educated decisions on investments
7 Create User Interface **7.1** Search feature	The front end is important, especially for stakeholders as it is where the software is interacted with. By implementing various

Benjamin Holmes

7.2 Trending feature **7.3** Design	features and designs, it will match what stakeholders are looking for in a product. Simultaneously it will provide ease of use for any user wanting to use this software.

Limitations

Twitter allows only small time frames to be scraped from its API. This means that either only small amounts of data can be collected at a time, or data must be collected over time periodically. This results in code left running in order to continuously update the graph plotter and averages.To help manage the sheer volume of data requests, limits are placed by Twitter on the number of requests that can be made in a time interval. This is known as a rate limit and allows for Twitter to provide the reliable and scalable API that the developer community relies on.

Due to the amount of data markets hold, this code will only be able to reflect less accurate versions of data points, and therefore rougher and less precise graphs. Millions of dollars are traded every minute, however it is inefficient for a programme running over long periods to be collecting this data in such small time intervals. This limits precision but still gives a clear picture of the coin.

Design of the solution

Decomposing the problem

In order to decompose this project, modular design will be used. This takes large problems and divides them into sub-problems. By splitting this project into smaller sections, the task becomes easier to solve and manage. This is paired with the added benefit of being able to test smaller sections more regularly and therefore limiting errors.

To identify the classic transformation of data into information IPSO (Input, Process, Storage, Output) charts can be used. I choose this method to display the processes behind the system. For each output, I look at data that is required, how it is processed,and any associated storage. IPSO charts are useful for prompting other aspects of design. By considering two sections of the program separately it allows for better understanding and modularity as they can work separately.

Similarly I constructed two data flow charts that lay out the decomposition of both modules into meaningful parts.

Benjamin Holmes

Twitter Scraper ISPO Chart

Input	Process	Storage	Output
(User input) Date: - Start - End	Asks the user for input as a basic value stored as a variable. Then converts this date into a readable format for the pull request through splitting then mapping.	User set global variable	N/A
(User input) Coin Type: - Label - Name	Asks the user for a crypto coin name e.g bitcoin or a crypto coin label e.g BTC. Uses this to form a pull request from the Twitter API.	User set global variable	N/A
API connection setter: - Consumer Key - Consumer Secret Key - Access Token - Secret Access Token	Uses Tweepy to authenticate consumer keys, then sets access tokens. Goes on to apply these settings to a variable named api which manages these keys under a wait on rate limit	Managing global variable	N/A
Request Tweets Function: - Start Date - End Date - Number of Tweets to pull - Search query - Language	Start and End dates are passed as parameters from the previously collected user inputs. As the only parameters, the rest of the inputs are called from global variables. Uses Start Date, End Date, User inputted coin type (search query) and Language (programmer set to 'en' for English) in	Stored in a temporary list with the length of Number of pulled tweets x content of the individual tweets. As a tweet limit is 280 the size of the list will be < (Number of Tweets to Pull) x (280). This can also be looked at as bits which will be this value x8 as there are 8 bits in a character.	List of tweets

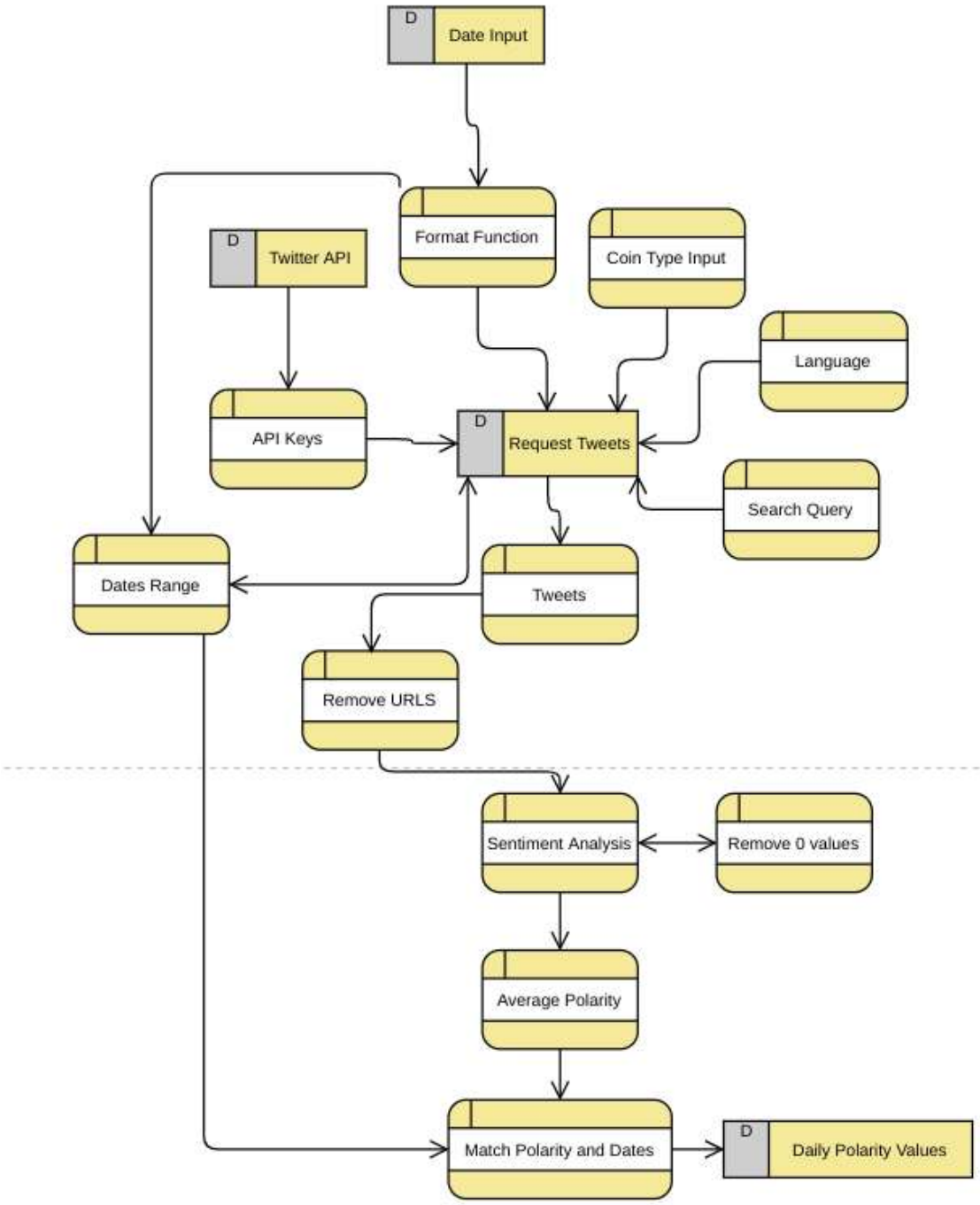
	order to form a pull request that scraps tweets through the Tweepy Cursor and stores them in a list format.		
URL cleaner: - URL remover function - Tweet list	The URL remover returns blank space instead of a range of symbols or characters that can be associated with URLS ([^0-9A-Za-z \t])|(\w+:\/\/\S+). This function then can be used on the list of tweets by passing it as a parameter then returning a cleaned list.	Local variable	List of tweets except with cleaned tweets
Sentiment Analysis: - Cleaned Tweet List	Uses Textblob to analyze tweets then links each tweet it analyzed with the relevant polarity value for each tweet in the cleaned tweet list. Then it goes on to form a table with columns for the tweet and polarity value. Next removes tweets and values where the polarity value is 0 in order to prevent average skewing.	Data Frame Table with columns of Polarity and Tweet	N/A
Average Daily Polarity: - Sentiment Table values	Collects the polarity column of the table and adds them to a list of polarity values. Then sums the list of polarity values and divides by the length of the polarity value list to create the average polarity list	List	N/A

Benjamin Holmes

Date range: - Start and End Dates in suitable format	Uses datetime to loop through dates in a user-given range. This is then appended to a list as separate dates. E.g. Start: 21-09-21 End: 25-09-21 Would create a list with: 21-09-21 22-09-21 23-09-21 24-09-21 25-09-21	List	Prints Dates
Average Polarity Per Date: - Range of dates list - Average polarity list	Loops through the length of the date range list and runs the polarity finder function using the start date and end date as one day per loop. So for a range of dates the loop finds the polarity of each day by using the dates either side of that day as start and end points	List	Prints polarity values per day

Benjamin Holmes

Twitter Scraper Data Flow Diagram



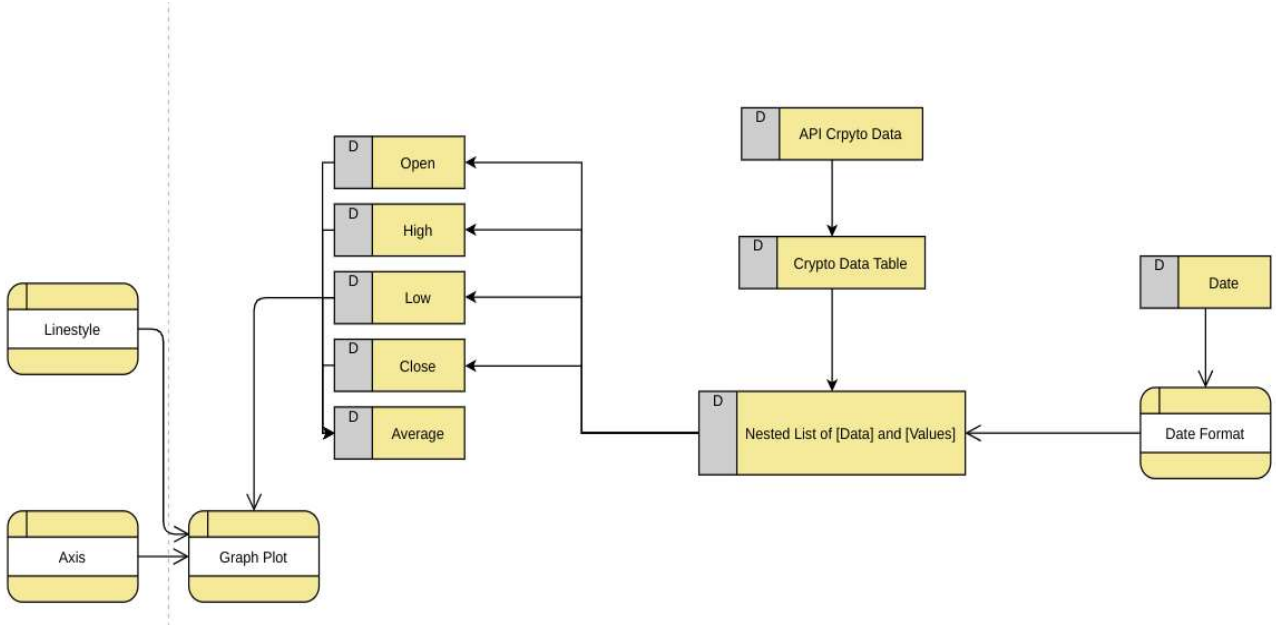
Benjamin Holmes

Current Crypto Price ISPO Chart

Input	Process	Storage	Output
Crypto Price Finder Function: - Symbol of crypto coin - Start Date - End Date	Uses programmer input to find real crypto prices in a time frame and produce a graph.	Currently it stores data as lists that are collected each run.	Graph of prices over time (shows open, high, low, close and average prices)
API request and data storage: - Coin Symbol - Start Date - End Date	Uses messari API to collect data values and store the json file in a Data Frame table with two columns.	Table	N/A
Table to List Loop: - Open, High, Low, Close, Average Lists - Date list	Loops through the table and appends the relevant information to the relevant list, matched to the format-changed dates. Also loops through all lists involved to find the average of all data values which is added to average list	Multiple Nested Lists	N/A
Graph: - Open, High, Low, Close, Average Lists - Date list - Line style - Line width - Axis Labels	Uses data from all lists to plot them on the same line graph, against date. Programmer also inputs linestyle, linewidth and axis.	Can be saved as image files	Graph of prices over time (shows open, high, low, close and average prices)

Benjamin Holmes

Current Crypto Price Data Flow Diagram



Justification of Decomposition

Decomposition Chart/Diagram	Justification of Layout
Twitter Scraper ISPO Chart	Has 9 inputs that store data in either lists, temporary tables or global and local variables. This is due to the processing nature of the code where data is given, scraped and processed, then outputted in format. The input sections of the code are a series of smaller tasks that can be worked on separately and chronologically. This links to success criteria points **1** through **4.**
Twitter Scraper Data Flow Diagram	Demonstrates how the code as a whole acts as a function with many entry points, methods occurring, then one output. This is important as it is very modular and therefore can be applied to any keyword selection of tweets all producing a similar style output.
Current Crypto Price ISPO Chart	A very simple data collection and graphical presentation program that gathers data from

	Messari API and frames it in a way that best displays it. Similarly to the Twitter Scraper ISPO Chart, each input section represents a section of code that can be worked on modularly. This links to success criteria point **5**.
Current Crypto Price Data Flow Diagram	This clearly shows how the data is collected, split by the nature of the group of data, then repurposed into a singular graphical presentation. This is important as it is very similar to the Twitter Scraper's graph, meaning they can be compared fairly easily.

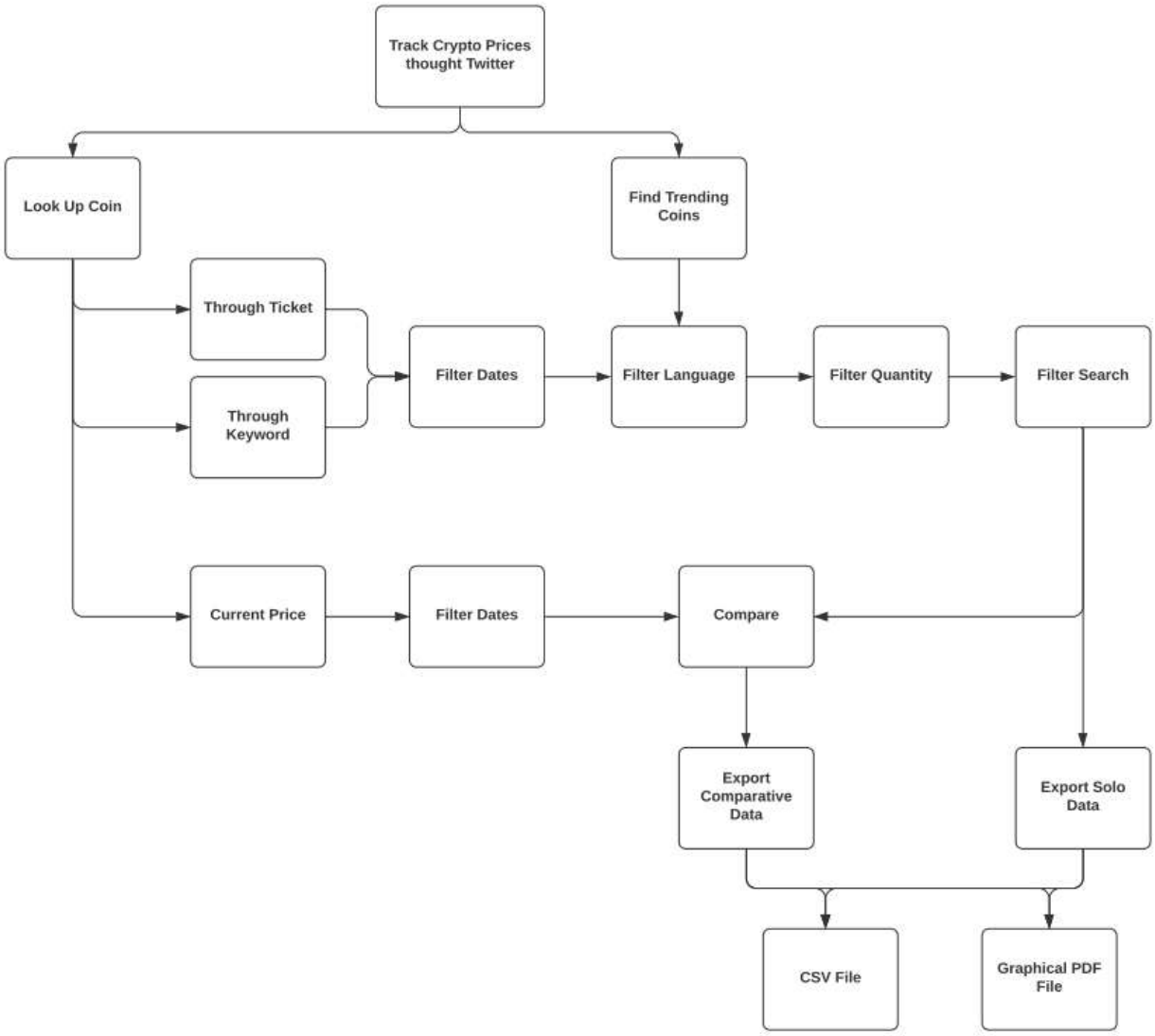
Data Dictionary and Validation

The following table contains all the data that could be entered into the system by the user, and how I plan to validate it to ensure that all fields are correctly inputted and stored. The erroneous data for some fields is a blank field - in cases where the validation check is only allowing the user to select correct fields from a list or directory, there is no way to test this.

Field	**Data Type**	**Length**	**Validation Check**	**Validation Description**	**Valid Data**	**Erroneous Data**
Date Start	String	10 characters	Presence	String in the format YYYY-MM-DD	2020-09-24	Any string outside format or blank
Date End	String	10 characters	Presence	String in the format YYYY-MM-DD	2020-09-30	Any string outside format or blank
Crypto Ticket	String	3-7 Characters	Presence	Capitalisation	BTC	Blank or lower case
Language	String	2 Characters	Presence	Lower case matching language key	en	Numbers
Search Keyword	String	1-30 Characters	Presence	Numbers, symbols, upper/lower case	bitcoin	(Blank)

Benjamin Holmes

Search Type	String	1-30 Characters	List	Allows selection from a list of types of searching	Trending	(Blank)



Describing the solution

Top-Down Design

Benjamin Holmes

Stepwise Refinement

This is a more detailed list of the processes involved in the new system, mirroring the system flow chart. It allows me to break down each task into its simplest processes, and organize the data that has been inputted to store.

1. Import Libraries
    1.1. Import Matplotlib as plt
    1.2. Import Datetime
2. Set Keys
    2.1. Consumer Key
        2.1.1. Authenticate Identification
    2.2. Consumer Secret Key
        2.2.1. Request access to resources from Twitter
    2.3. Access Token
        2.3.1. Set Access Privileges Unique to user
    2.4. Secret Access Token
        2.4.1. Unlock Privileges through the OAuth 2.0 Protocol
3. Date Setting
    3.1. Enter start and end dates
        3.1.1. Validation
4. Pull Tweets
    4.1. If Pulling Trending Tweets
        4.1.1.
    4.2. If Pulling Tweets On Keyword
        4.2.1. User entered keyword
            4.2.1.1. Validation
        4.2.2. Pull relevant tweets
5. Clean Data
    5.1. Split into separate strings
        5.1.1. Remove unwanted characters
    5.2. Join
6. Sentiment Analysis
    6.1. Analyze individual tweets
        6.1.1. Assign Polarity Score
        6.1.2. Store in temporary table

6.2. Calculate average polarity score based on the time frame and number of tweets pulled

7. Graph
   7.1. Find every date in the range of dates
      7.1.1. Plot on x axis
   7.2. Plot average polarity on y axis
   7.3. Graph Style
      7.3.1. Line Types
      7.3.2. Labels
8. Compare
   8.1. Import real crypto prices

Usability Features

In order to reach the user requirement specification and decide on the best techniques for usability evaluations throughout the project, this project will work towards an effective, efficient, error tolerant, engaging and easy to learn/use software. This will be demonstrated in various ways through the output of the software and its attached user interface.

Algorithm Design

Crawling Data from Tweets on Keyword

Explanation
This function loops through a list of additional key words, concatenates them to the main search query and then uses the twitter scraper object to collect data. It then adds data points to a relevant list.
Pseudo-code
Keyword List [ ] *Query Word " "* *FOR each Keyword in Keyword List THEN* - *Full Keyword = Query Word + Keyword List [Keyword]* - *FUNCTION Scrape Twitter(Keyword, Number of Tweets, Dates)* *END FOR*

Cleaning Tweets

Benjamin Holmes

Explanation
This function loops through the column of tweets and replaces certain symbols, numbers and letters with nothing.
Pseudo-code
Clean Tweets [ ] *FOR each Tweet in “Tweet” Column THEN* - *Tweet = Replace Usernames with “ ”* - *Tweet = Replace Hashtags with “ ”* - *Tweet = Replace Hyperlinks with “ ”* - *ADD Tweet to Clean Tweets [ ]* *END FOR*

Qualitative Score from Data

Explanation
This function uses the Polarity column and decides whether the score means the tweet is Positive, Negative or Neutral.
Pseudo-code
FUNCTION Get Analysis (Score) - *IF Score < 0 THEN*   - *OUTPUT “Negative”* - *END IF* - *IF Score > 0 THEN*   - *OUTPUT “Positive”* - *END IF* - *ELSE THEN*   - *OUTPUT “Neutral”* - *END ELSE* *END FUNCTION* *Analysis Column = Get Analysis (Polarity Column)*

Benjamin Holmes

Manipulating Live Crypto Data

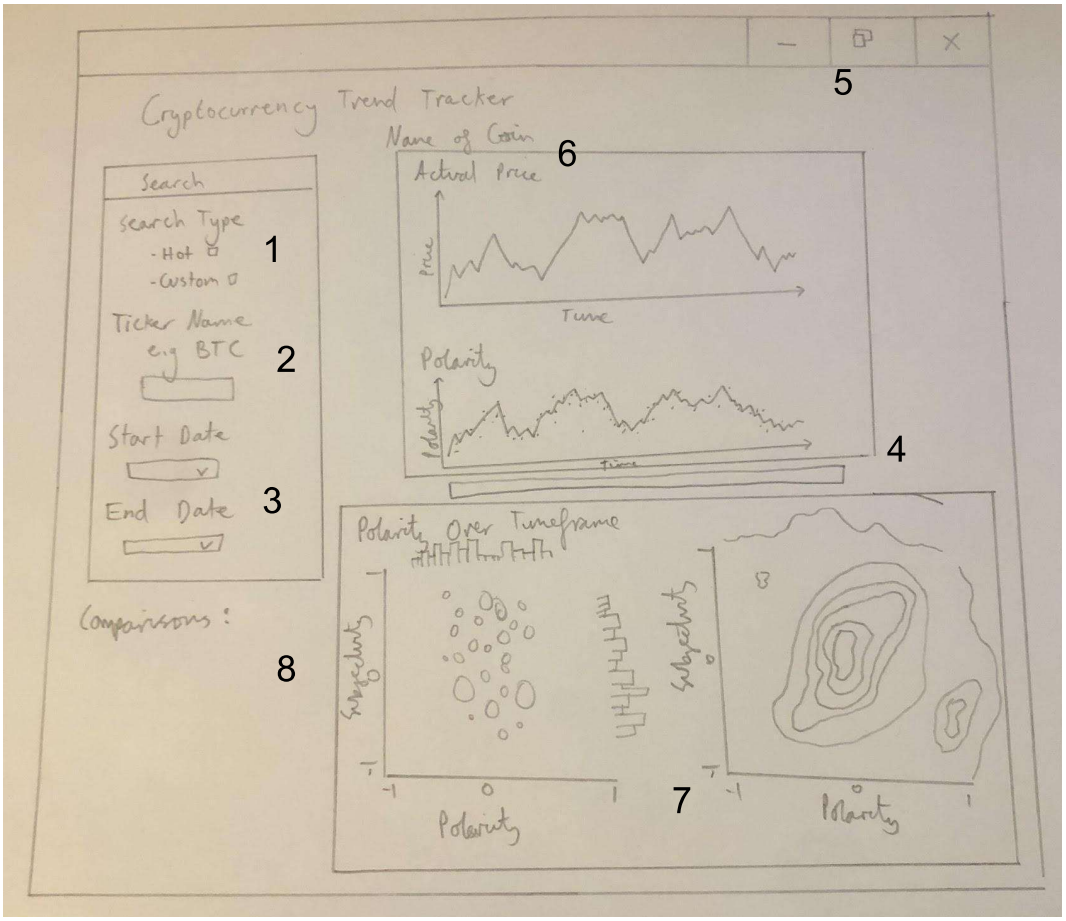
Explanation
This function uses the Polarity column and decides whether the score means the tweet is Positive, Negative or Neutral.
Pseudo-code
IMPORT Messari Data *CREATE Data Table with “Date” and “Values” Columns* *Open [ ]* *High [ ]* *Close [ ]* *Low [ ]* *Date [ ]* *Average [ ]* *FOR Values in Date Table THEN* - *Changed Dates = Split Date column into Day/Month/Year and remove Minutes/Seconds* - *ADD Changed Dates to Date [ ]* - *ADD Date Table [Values] [1] to Open [ ]* - *ADD Date Table [Values] [2] to High [ ]* - *ADD Date Table [Values] [3] to Close [ ]* - *ADD Date Table [Values] [4] to Low [ ]* *END FOR* *FOR Values in Date Table THEN* - *ADD (Open [Values] + High[Values] + Close[Values] + Low[Values])/4 to Average[ ]* *END FOR*

User Interface Design

In order to visualize the usability features in the design stage of this project a hand drawn diagram can be used to demonstrate envisioned aspects in the front end of the software. These

Benjamin Holmes

drawings are my first outlines of what the interface of the system should look like. They will most likely not be the final designs, but they are the first generation of each form and a starting point for later concepts.



1) This allows the user to decide whether they to want the software to find trending coins, or if they want to initiate a custom search based on various search queries
2) Selection of coin (ticker name). This would only be able to be used if custom search is selected.
3) Selection of end and start date of the search. This would only be able to be used if custom search is selected.

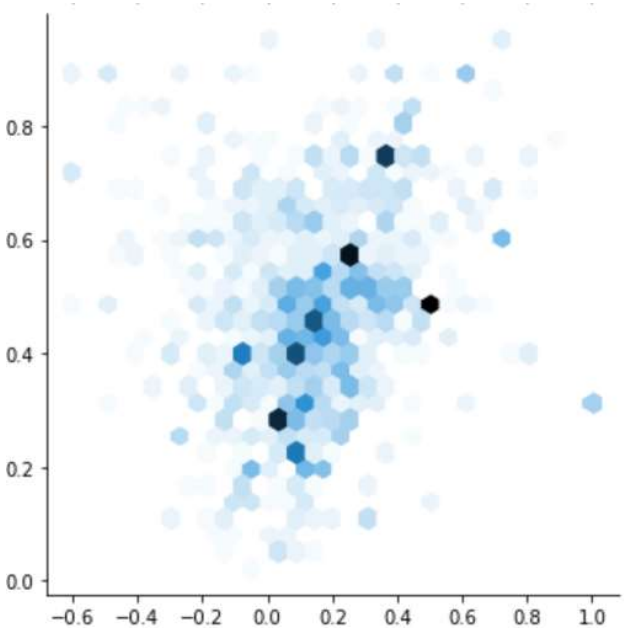
4) Time slice slider allows for the user to analyze times between their initial start and endpoint dates. This highlights a section of the graph and zooms in accurately allowing for more detail.
5) Presented on a web client that allows users to easily access as well as minimize, full screen and close the application. Similar to other applications this makes the program easily accessible and intuitive.
6) The sub-title of the page contains the name of the coin the user is searching. All data below is relevant only to that coin.
7) Other graphical visualizations of the coin over time. This includes variations on subjectivity as well as polarity scores through sentiment analysis. By presenting this data in a visual way - simple graphs - it allows quick evaluation of each coin.
8) Computational comparison section. Although up for development later in the project, this would include image analysis and machine learning scores based on similarity of graphs.

Output Design

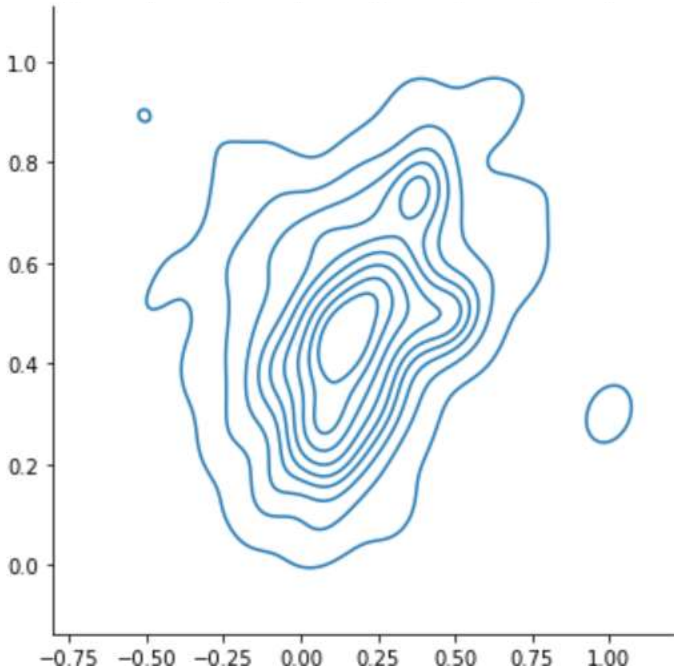
Most output from this program will be graphical. Therefore it is important to use a combination of good graph building libraries. This would need to include data entry from CSV files, regular axis, axis labeling, title labeling, legend, color coding and options of graph types (e.g. line, scatter, bar, Kernel Density Estimation, joint plots and hex plots). As this program aims to find correlation, it is important to select graph types that show this correlation in a way that is readable and evident. For example, jumbled data may result in line graphs becoming ineffective, whilst scatter plots show polarity density well as individual data points. Bar graphs would also be ineffective due to the polarity scores going to 6 decimal places. As it is very unlikely that lots of tweets will have exactly the same polarity a bar graph would display as long and flat with values often only reaching one data point high. For this reason Kernel Density Estimation and hex plots become useful as they can represent polarity scores as not only quantitative values but also in a color gradient or line density based on frequency around point values.

Example of Hex plot:

Benjamin Holmes



Example of Kernel Density Estimation plot:



Benjamin Holmes

OOP Class Design

As a linear process, class use is limited as new objects with various differences do not need to be instantiated frequently. Classes in the system are used occasionally in order to create an object that scrapes tweets based on certain parameters. The class is instantiated each runtime when declared by user inputs. Past run time, my software does not need to store this defined object, as this is passed to a database that is appended to each runtime.

Twitter Scraper
q: String (Query) lang: String start_date: Datetime end_date: Datetime tweet_mode: String host: String wait_on_rate_limit: Boolean count: Integer geocode: Integer Concatenation api_key: String api_secret: String access_token: String access_token_secret: String
search_tweets() items() rate_limit_status() set_access_token() OAuthHandler()

Database Design

As this project deals with large amounts of stored data that needs to be searched frequently in order to plot, an CSV table that exports to an SQL database is appropriate for this system. The CSV file will act as a temporary storage template during runtime. At the end of the program all data in this file will append to a unique larger SQL file that collects data dependent on the Search Query but over a wider range of dates (from previous runtimes). This is important as the

Benjamin Holmes

Twitter API limits the frequency of searches in a given time, as well as only being able to search up to a week in the past. Therefore by collecting data on given coins over time it enables the program to have plottable data that extends past a week in history. As information and analysis on different coins cannot be mixed, data from different coins will be kept differentiated with key ticker names. Advantages of transferring temporary twitter data to a larger database include security, faster access/retrieval, organization as well as greater capacity - especially over time.

CSV Table Structure:

	A	B	C	D	E	F	G
1		Created_at	User_id	Tweets	Subjectivity	Polarity	Analysis
2	0	2022-01-29 23:50	43354683	Bitcoin s volatility won t shake the confidence of institutional investors	0.566	0.88	Neutral
3	1	2022-01-29 23:49	1.40389E+18	2 Bitcoin BTC is maturing Despite increased exuberance as BTC scaled tc	0.448	0.102	Positive
4	2	2022-01-29 23:43	1.47765E+18	crypto lowers the barrier to entry for new cryptocurrency investors Try	0.454545455	0.136363636	Positive
5	3	2022-01-29 23:41	1.37119E+18	Woaaah Bitcoin is amazing Did you see it Didn t you Good Don t look at i	0.8	0.166666667	Positive
6	4	2022-01-29 23:36	1.44696E+18	Anthony Scaramucci founder of investment firm SkyBridge Capital addr	0.6375	0.1375	Positive
7	5	2022-01-29 23:36	1.18765E+18	George Soros is a legendary hedge fund manager who is widely conside	0.65	0.4	Positive
8	6	2022-01-29 23:35	8.47882E+17	This is kind of storytelling more than objective fact If bitcoin is at 100k i	0.534285714	0.302857143	Positive
9	7	2022-01-29 23:28	2853296417	Today s Deposit Bonus 0 005 BTC It is only for first 50 investors per day fa	0.305555556	0.064393939	Positive
10	8	2022-01-29 23:23	8.29722E+17	Norway Y all don t understand how this works Those are just brokerage	0.98	0.6	Positive

SQL Table Structure:

```
CREATE TABLE twitter_data(
    PRIMARY KEY       INTEGER  NOT NULL PRIMARY KEY
   ,ticker    VARCHAR(7) NOT NULL
   ,Created_at   VARCHAR(25) NOT NULL
   ,User_id      VARCHAR(20) NOT NULL
   ,Tweets       VARCHAR(285) NOT NULL
   ,Subjectivity VARCHAR(6) NOT NULL
   ,Polarity     VARCHAR(6) NOT NULL
   ,Analysis     VARCHAR(8) NOT NULL
);
```

The Primary Key in the SQL database is a combination of a NOT NULL constraint and the index INTEGER. This allows for unique identification. An example of lines of data from this table would be (inserted from the CSV file):

```
INSERT INTO twitter_data(primary_key, ticker,Created_at,User_id,Tweets,Subjectivity,Polarity,Analysis) VALUES (1,BTC,'2022-01-29 23:50:18+00:00','43354683','
INSERT INTO twitter_data(primary_key,ticker ,Created_at,User_id,Tweets,Subjectivity,Polarity,Analysis) VALUES (2,BTC,'2022-01-29 23:49:35+00:00','140389421931
INSERT INTO twitter_data(primary_key,ticker ,Created_at,User_id,Tweets,Subjectivity,Polarity,Analysis) VALUES (3,BTC,'2022-01-29 23:43:48+00:00','147765486395
INSERT INTO twitter_data(primary_key,ticker,Created_at,User_id,Tweets,Subjectivity,Polarity,Analysis) VALUES (4,BTC,'2022-01-29 23:41:39+00:00','137119458958
INSERT INTO twitter_data(primary_key,ticker ,Created_at,User_id,Tweets,Subjectivity,Polarity,Analysis) VALUES (5,BTC,'2022-01-29 23:36:11+00:00','144696380561
```

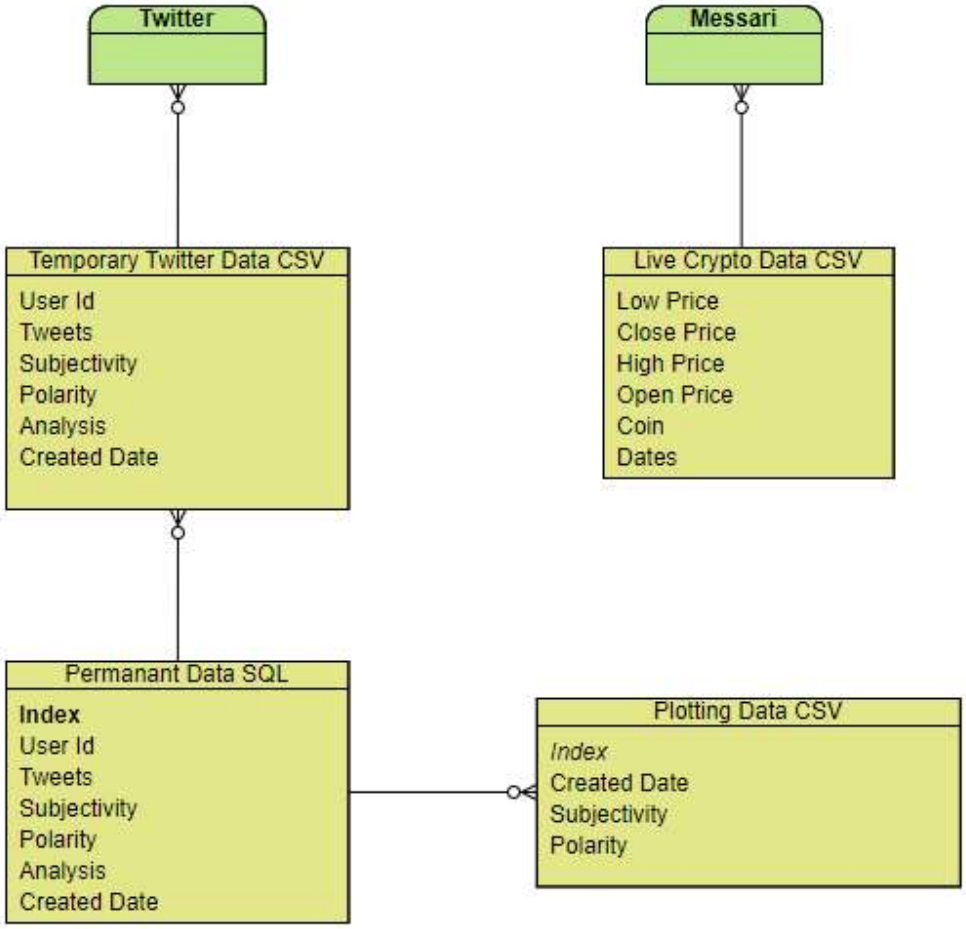
```
0.88','Neutral');
uggests holders are focused on the long term Long term investors hold 13 5m BTC with more than 500k addresses exhibiting long term holding behavior','0.4
way to buy Altbase Token Bitcoin Altcoins and memecoins from any smartphone anywhere in the world','0.4545454545454453','0.13636363636363635','Positive'
te me but I don t care we need investors','0.7999999999999999','0.16666666666666666','Positive');
that Bitcoin has great potential Bitcoin forex Binance Robinhood ATLAS blockchain BNB Cardano CeekVR DeFi Ethereum raca Crypto cryptocurrency','0.6375','
```

Benjamin Holmes

Separately, real time crypto data will be gathered using an API linked to the online database Messari, that provides data insights and live pricing. This will then be stored in a simple CSV file as the data is stored server side, it can be called and used on command.

Database Relations:

Benjamin Holmes



The diagrams illustrate the relationships between the normalized tables in the system. The entity names are column names. We can tell the data has been normalized because there are no many-to-many relationships, no null values, related data is linked with a primary key and other tables are linked with a forgien key. These databases are backend and therefore are not related to any output data the user would be able to save after using the system to gather information.

Identification of Storage Media

When the system is used, the main output will be graphically displayed and hosted by a web application. The user could download the actual system from a secure online location - this

would require an internet connection as well as access to administrative rights when downloading files on a home computer. On a slow connection, this would be impractical. It also means the file would have to remain online and accessible indefinitely in case a reinstall is required.

The system could also be installed from a CD-ROM (CD-RW discs are unnecessary because data will only be read from and not written to the disc after the executable is initially copied) which can store 194 megabytes of data. The disc could then be kept to install the system on other devices. However, as the disc would only contain the installation file, a large amount of space that cannot be overwritten would be wasted. There is also the consideration that as the pressure on laptops and netbooks to become more portable and smaller in dimension increases, many machines will no longer have an internal CD/DVD drive.

Ideally, the storage of the executable file would be on a USB flash-drive, for several reasons. The file itself wouldn't create any wasted space on the flash drive, as the rest of the drive could still be read to and written from as usual allowing for output CSV data files the user may want to be exported into the same directory. USB ports, as well as being industry standard and portable, also have fast installation speeds allowing for easy sharing, downloading and accessibility between stakeholders.The user could access this USB physically after development, or from sharing files via email.

Security and Integrity of Data

As there is no personal data stored about users (or anonymous Twitter accounts) within the system, there is no need for any kind of encryption. Users won't have access to the system framework as it would only be accessible on the front end of a web application, so having a restricted access profile is not a concern. Storing the tweets and analyzed data scores in CSV files combined with an externally hosted database means there is less chance a user could gain access to raw data.

To protect the integrity of the stored data, all data entry will be controlled by strict validation rules. Wherever possible - filtering questions by certain parameters – the user will select their options from drop-down menus, calendars or selection buttons. This minimizes free text input which, as well as saving time, stops typographic errors which may cause the system to crash or incorrect data to be stored. Changes between databases will follow the ACID (Atomic, Consistent, Isolated, Durable) framework with null values being disregarded. Although true values will sometimes take on the value of 0 there is no way to differentiate the difference between faulty or true values, therefore suggesting complete cleansing of these values to avoid data skewing.

Benjamin Holmes

Describing the approach to testing

Testing Plan

To ensure the system will be able to correctly handle all user inputs (both correct and incorrect) as well as correctly navigate between different search queries and execute algorithms correctly, I have planned a testing process. The following table is designed to test the actual outcome vs. the expected results for every process,user input and system output in the case of typical (correct and expected) data, erroneous (would cause the system to throw an exception, e.g. Incorrect data types) data and boundary data (less expected data, e.g. Blank fields or boundary data).

Test ID	Description	TEB (Typical, Erroneous, Boundary)	Expected Outcome
01	Check the API keys are valid and can link to Twitter	**T**: 4 valid keys linked to twitter developer account **E**: Blank fields/wrong keys **B**: Other irrelevant keys are passed	OAuthHandler is set and the API is connected
02	Check Twitter Scraper class is instantiated as an object correctly	**T**: All attributes are correctly passed **E**: Missing field/Wrong data type	The object is created and all relevant attributes are valid
03	Check search query extensions are being used correctly	**T**: Each value in the list is concatenated to the main keyword **E**:Missing extra keywords	All keyword variations are created
04	Check tables are created and is storing data properly	**T**: Data is in correct columns **E**: Retweets	Data is stored in a logical structure
05	Check data is cleaned	**T**: Only characters and numbers in a string are in a tweet **E**: Usernames, Hashtags, Symbols **B**: Empty space	Data is reduced by removing unnecessary content that can't be analyzed

		between characters	
06	Check Polarity is calculated	**T**: A float value **E**: Null values **B**: Long decimals	Data column is created with a respective score
07	Check Subjectivity is calculated	**T**: A float value **E**: Null values **B**: Long decimals	Data column is created with a respective score
08	Check Analysis is calculated	**T**: Positive, Negative, Neutral **E**: Missing values	Data column is created with a respective word
09	Check Start Date is selected correctly	**T**: String in the format YYYY-MM-DD **E**: Any string outside format or blank	Selection of calendar date
10	Check End Date is selected correctly	**T**: String in the format YYYY-MM-DD **E**: Any string outside format or blank	Selection of calendar date
11	Check Crypto Ticket is entered correctly	**T**: Letter Ticker Tag **E**: Number/Symbols/ Blank **B**: 3-7 Capital letters long	Ticker can be found on Twitter and the Messari database allowing for search
12	Check language is a valid language	**T**: Two letter language tag **E**: Numbers/ Blank	The language tag can be found in the Twitter languages supported list and it is searchable
13	Check search query is valid	**T**: Numbers, symbols, upper/lower case **E**: Blank **B**: Up to 20 characters	The search word is able to find results on Twitter
14	Check search type is selected	**T**: An option is selected **E**: Blank	One of the search type options is selected so that search can begin
15	Check data is plotted correctly	**T**: Graph with regular axis and consistent	A graph that shows some sort of

		data points **E**: Mixed data points	correlation

This test data is chosen to allow for linear development of the system with regular and thorough checkpoints that test for robustness, syntax errors, logical errors and runtime errors. This is important as tests build upon each other, so a well tested base ensures that the system as a whole is well developed. By checking user input fields are correctly filled in the right format, it allows the system to run as it is intended to, without any errors being thrown, which would therefore halt the program. One example of this validation test is having character limits on the ticker input field. This works due to the fact that coin tickers have a small number of capital characters that won't exceed a certain length (7 Characters), allowing for data not fitting this requirement to easily show an error message to prompt re-entering.

Developing the solution

Iterative development process and testing to inform development

Developing this project will take several iterations. Non-working/sub-optimal sections of each iteration will continue into the next iteration, with more testing. This allows for already efficient code to be used whilst code that still needs to be changed can have more development time. By following the success criteria's ordered numerical list, as well as the testing plans checkpoints it will ensure that the code is produced to fulfill stakeholder requirements whilst also passing validation tests.

Recording this development is fundamental to showing the interactive development process. I will be using annotated screenshots, as well as tables for testing to prove the system works and is efficient. Whilst screenshots are the easiest method of evidence recording, I will make certain that they are cropped and are visible.

During this project I will be using Jupyter Notebook for development. It is an open source web application that allows users to create and share documents that integrate live code, equations, computational output, visualizations, and other multimedia resources, along with explanatory text

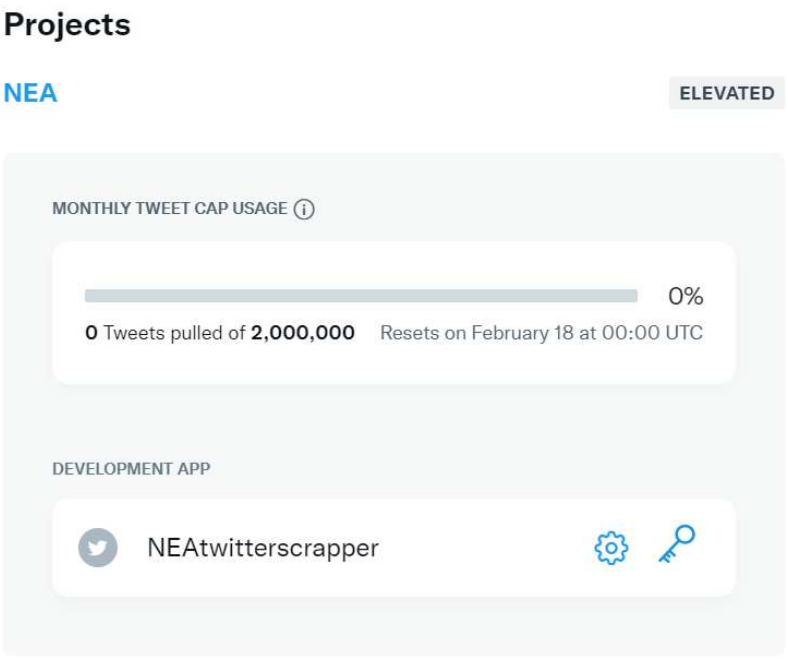
Benjamin Holmes

in a single document. It also can be used for all sorts of data science tasks including data cleaning and transformation, numerical simulation, exploratory data analysis, data visualization, statistical modeling, machine learning and deep learning. As this is particularly relevant to my project it has lots of benefits.

Furthermore it is a form of interactive computing, an environment in which users execute code, see what happens, modify, and repeat in an iterative method. This is particularly useful for showing iterative development and step-by-step processes.

First Iteration

As part of Criteria **1** we must first get a Twitter developer account in order to link the API to the program (**1.1**)

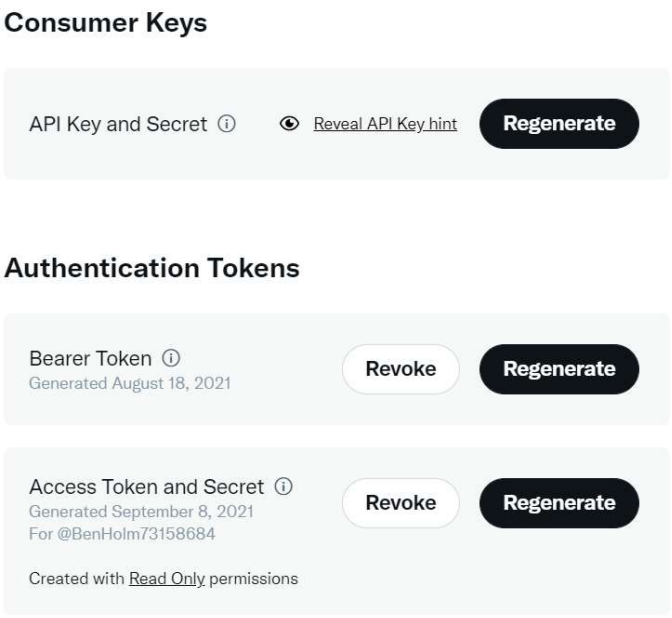


The developer account allows you to set up and manage your own Projects and Apps. Projects create the permissions for allowing Apps to connect to Twitter API v2 endpoints. Only Apps

Benjamin Holmes

within Projects can access Twitter API v2. From the Dashboard in the developer portal, developers can find App IDs; edit an App's setting, permissions, and callback URLs; and generate and revoke keys and tokens. It is these keys and tokens that are needed to create a twitter scraper.

As the Twitter API handles enormous amounts of data, data is secured through authentication. This is in the form of Bearer Tokens, Access Tokens and Consumer Keys.



The OAuth 2.0 Bearer Token authenticates requests on behalf of the developer App. As this method is specific to the App, it does not involve any users. This method is typically for developers that need read-only access to public information. In this case, as we are reading and collecting tweets, this read-only access is necessary.

Having got the 4 tokens, we can begin to implement them in code.

```
consumer_key= 'Gdcp26KHtuIagI6C7c8XIawWq'
consumer_secret= 'VL8HEIRG1hpV0FCIspiZ6KVsEExRV4cPrav6CswEp9mMxUnzDa'
access_token= '1427974872186097665-zfrVCIXiHw9jUtXyCkEEfBsxkYtxJM'
access_token_secret= 'iMDSW8ZOzfqdLit1Yb55t9TNihGHTC0ooe0qWhMu1h93o'
```

After installing the relevant libraries as laid out in Criteria **1.2**, the authentication handler must be set to allow access to Twitter's database via the credentials that have just been acquired. This is done through the Twitter scraper object. As there are no errors Test **01** has proven successful.

Benjamin Holmes

```
auth = tw.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tw.API(auth, wait_on_rate_limit=True)
```

The wait_on_rate_limit attribute being set to True decides whether or not to automatically wait for rate limits to replenish. The rate limit is the maximum amount of data Twitter allows a user to scrape in a given timeframe.

In order to fulfill Criteria **1.3** we create a function that instantiates the Twitter Scraper class and saves the object to a variable called “tweets”. It is important to have this instantiation within a function as it means it can be replicated easily with different parameters being passed to change the attributes. This is in line with a modular approach to programming. In this case we pass the start and end dates of the search into the function.

```
def requesttweets(startdate,enddate):
  tweets = tw.Cursor(api.search,
                     q=new_search,
                     lang="en",
                     since=startdate,
                     until=enddate,
                     result_type="recent"
                     ).items(numpulltweets)
```

When this function is called the following error is produced:

```
requesttweets("2022-02-04","2022-02-07")
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [17], in <module>
----> 1 requesttweets("2022-02-04","2022-02-07")

Input In [16], in requesttweets(startdate, enddate)
      1 def requesttweets(startdate,enddate):
----> 2   tweets = tw.Cursor(api.search,
      3                      q=new_search,
      4                      lang="en",
      5                      since=startdate,
      6                      until=enddate,
      7                      result_type="recent"
      8                      ).items(numpulltweets)

AttributeError: 'API' object has no attribute 'search'
```

This is due to frequent documentation updates. With the new standard being api.search_tweets. Employing this change, the code still produces a NameError as there are undefined attributes being used in the instantiation. Language is pre-set to “en” so Test **12** is untestable this iteration.

```
Input In [18], in requesttweets(startdate, enddate)
      1 def requesttweets(startdate,enddate):
      2   tweets = tw.Cursor(api.search_tweets,
----> 3                      q=new_search,
      4                      lang="en",
      5                      since=startdate,
      6                      until=enddate,
```

Benjamin Holmes

Therefore we need to define some starting variables within the code. As this is the first iteration I will not rely on user input for most variables, however I will implement user chosen dates in order to test formating.

```
new_search = "bitcoin -filter:retweets" #this searches on the keyword bitcoin but removes retweets
numpulltweets = 10 #keep this low during testing to avoid exceed rate limit
startdateinput = input("Start Date (YYYY-MM-DD): ")
yearS, monthS, dayS = map(int, startdateinput.split('-'))
date1 = datetime.date(yearS, monthS, dayS)
enddateinput = input("End Date (YY-MM-DD): ")
yearE, monthE, dayE = map(int, enddateinput.split('-'))
date2 = datetime.date(yearE, monthE, dayE)
```

The search query and number of tweets to be scraped are now defined (untestable Test **13**) . The start and end dates receive user values from on screen display as seen below. The dates also have to be formatted into the datetime library format in order to be manipulated later.

```
Start Date (YYYY-MM-DD): 2022-02-04
End Date (YYYY-MM-DD): 2022-02-07
```

As there is user input here, and although this is not the final format the date input will be, in order to test the validation catching, I use try and except validation. This is part of Test **02** although it falls under Test **09** and Test **10** as the dates are attributes of the Twitter Scraper object.

```
format = "%Y-%m-%d"

while True:
    try:
        startdateinput = input("Start Date (YYYY-MM-DD): ")
        datetime.datetime.strptime(startdateinput, format)
        print("This is the correct date string format.")
    except ValueError:
        print("This is the incorrect date string format. It should be YYYY-MM-DD")
        continue
    else:
        break

while True:
    try:
        enddateinput = input("End Date (YYYY-MM-DD): ")
        datetime.datetime.strptime(enddateinput, format)
        print("This is the correct date string format.")
    except ValueError:
        print("This is the incorrect date string format. It should be YYYY-MM-DD")
        continue
    else:
        break
```

This ensures that the dates are in the correct format and loops until the user enters it correctly.

Benjamin Holmes

Testing **09** :

```
Start Date (YYYY-MM-DD): 123456
This is the incorrect date string format. It should be YYYY-MM-DD
Start Date (YYYY-MM-DD): !"£$%^
This is the incorrect date string format. It should be YYYY-MM-DD
Start Date (YYYY-MM-DD): abcdefg
This is the incorrect date string format. It should be YYYY-MM-DD
Start Date (YYYY-MM-DD): 2000-01-40
This is the incorrect date string format. It should be YYYY-MM-DD
Start Date (YYYY-MM-DD): 2222-12-10
This is the correct date string format.
```

The date field handles incorrect data correctly and only accepts valid dates within the limits of a calendar.

```
Start Date (YYYY-MM-DD):
This is the incorrect date string format. It should be YYYY-MM-DD
Start Date (YYYY-MM-DD): 20020202
This is the incorrect date string format. It should be YYYY-MM-DD
Start Date (YYYY-MM-DD): 2002/02/02
This is the incorrect date string format. It should be YYYY-MM-DD
```

It also handles blank fields and boundary data correctly. Other attributes in this iteration are not imputed by the user so testing those values at this stage will be ineffective

Testing **10**:

```
End Date (YYYY-MM-DD): 123456
This is the incorrect date string format. It should be YYYY-MM-DD
End Date (YYYY-MM-DD): !"£$%^
This is the incorrect date string format. It should be YYYY-MM-DD
End Date (YYYY-MM-DD): abcdefg
This is the incorrect date string format. It should be YYYY-MM-DD
End Date (YYYY-MM-DD): 2000-01-40
This is the incorrect date string format. It should be YYYY-MM-DD
End Date (YYYY-MM-DD): 2222-12-10
This is the correct date string format.
```

The date field handles incorrect data correctly and only accepts valid dates within the limits of a calendar.

```
End Date (YYYY-MM-DD):
This is the incorrect date string format. It should be YYYY-MM-DD
End Date (YYYY-MM-DD): 20020202
This is the incorrect date string format. It should be YYYY-MM-DD
End Date (YYYY-MM-DD): 2002/02/02
This is the incorrect date string format. It should be YYYY-MM-DD
```

It also handles blank fields and boundary data correctly. Other attributes in this iteration are not imputed by the user so testing those values at this stage will be ineffective

Benjamin Holmes

Next, the tweets scraped need to be cleaned of unnecessary data. By creating a function that can be applied to any set of data I ensure that code is repeatable.

```
def remove_url(txt):
    return " ".join(re.sub("([^0-9A-Za-z \t])|(\w+:\/\/\S+)", "", txt).split())
```

This function splits the data, replaces unwanted characters (usernames, links, hashtags, symbols, emojis) with empty space, then rejoins the strings into one final clean string. To check

```
tweets_no_urls = [remove_url(tweet.text) for tweet in tweets]
```

this we can use Test **05** after applying it to the data set.

Test **05**:

Before the cleaning of data is applied, the tweets look like this:

```
@LarryCohen83 @spectatorindex It feels too volatile to make a good benchmark.  Unless you're arguing that Bitcoin i… https://t.
co/AW87ObFr4N
Bitcoin missed!!
Ethereum missed!!
BNB missed!!
Shiba Inu missed!!
#Shibnobi don't miss
Time is everything, being l… https://t.co/aPhlUQQKAH
@SadafJadran Definitely me and you can't live without Twitter 🤩 and thank you @verified for going to verify… https://t.co/uNol
9DW7p6
$EQOS is my Bitcoin stock play on this most recent Bitcoin Rally.
 Bought on Friday. https://t.co/0zjpU4OGdz
About to start, don't miss out on this #Bitcoin adoption upgrade in Canada https://t.co/b1IGhVqd2U
Find out all about bitcoin. What is it? Where does it come from? How does it work?  But 'The secrets to crypto curr… https://t.
co/bj3LWrtqGe
@CryptoBoj I see #Bitcoin as ultimately becoming a reserve currency for banks, playing much the same role as gold d… https://t.
co/8yBXMlkcXU
But like how do you even bring it up in a conversation 🤣 "yo check out this bitcoin I've got here" https://t.co/CvzbcT1a3F
👋 Who's someone in the #Bitcoin, #Blockchain, #Crypto, #DeFi, #NFT #NFTCommunity that we should be following/know a… https://
t.co/hp43slYsKf
@snappycaster @RAC The transactions I do with Bitcoin would typically take days or a week in the legacy banking sys… https://t.
co/R9VZpljRgp
```

After applying the function:

```
['LarryCohen83 spectatorindex It feels too volatile to make a good benchmark Unless youre arguing that Bitcoin i', 'Bitcoin mis
sedEthereum missedBNB missedShiba Inu missedShibnobi dont missTime is everything being l', 'SadafJadran Definitely me and you c
ant live without Twitter and thank you verified for going to verify', 'EQOS is my Bitcoin stock play on this most recent Bitcoi
n Rally Bought on Friday', 'About to start dont miss out on this Bitcoin adoption upgrade in Canada', 'Find out all about bitco
in What is it Where does it come from How does it work But The secrets to crypto curr', 'CryptoBoj I see Bitcoinas ultimately b
ecoming a reserve currency for banks playing much the same role as gold d', 'But like how do you even bring it up in a conversa
tion yo check out this bitcoin Ive got here', 'Whos someone in the Bitcoin Blockchain Crypto DeFi NFT NFTCommunity that we shou
ld be followingknow a', 'snappycaster RAC The transactions I do with Bitcoin would typically take days or a week in the legacy
banking sys']
```

Benjamin Holmes

As expected there are no usernames, hashtags or symbols within the list. There are however double spaces, which is a boundary condition.

The preprocessing of the text data is an essential step as it makes the raw text ready for mining, i.e., it becomes easier to extract information from the text and apply algorithms to it. If we skip this step then there is a higher chance that there is noisy and inconsistent data. The objective of this step is to clean noise that is less relevant to find the sentiment of tweets such as punctuation, special characters, numbers, and terms which don't carry much weightage in context to the text.

The next stage involves extracting numeric features from the Twitter text data . This feature space is created using all the unique words present in the entire data. So, if data is preprocessed well, then it is able to get a better quality feature space.

For this first iteration I will use the library TextBlob (Criteria **3.1**) in order to make sentiment analysis simpler. The process involves reading each tweet and determining a score of polarity. This polarity score determines how positive or negative the emotion behind the tweet is, where the range is from -1 to 1 (0 is neutral).

```
sentiment_objects = [TextBlob(tweet) for tweet in clean_tweets]
```

This line loops through each tweet and applies the TextBlob library to it (Criteria **3.2**) . To see an example output the first value can be observed using list methods.

```
print(sentiment_objects[0].polarity, sentiment_objects[0])
```

```
0.7 LarryCohen83 spectatorindex It feels too volatile to make a good benchmark Unless youre arguing that Bitcoin i
```

As seen above the tweet to the right has a polarity score of 0.7. In order to better manipulate all the data I will append this to a list.

```
sentiment_values = [[tweet.sentiment.polarity, str(tweet)] for tweet in sentiment_objects]
print(sentiment_values[0])
```

```
[0.7, 'LarryCohen83 spectatorindex It feels too volatile to make a good benchmark Unless youre arguing that Bitcoin i']
```

Here, the index 0 value is the polarity score. It is important to note that the data types are cast differently. The polarity score becomes a float value - this is fundamental for plotting data - whereas the tweet itself becomes a string.

As Tests **07** and **08** will be looked at in a later iteration, I can only check Test **06** is currently valid. Before I do this I will create a data table to order the data points better.

Benjamin Holmes

```
sentiment_df = pd.DataFrame(sentiment_values, columns=["polarity", "tweet"])
sentiment_df.head()
```

	polarity	tweet
0	0.700000	LarryCohen83 spectatorindex It feels too volat...
1	0.000000	Bitcoin missedEthereum missedBNB missedShiba I...
2	0.068182	SadafJadran Definitely me and you cant live wi...
3	0.250000	EQOS is my Bitcoin stock play on this most rec...
4	0.000000	About to start dont miss out on this Bitcoin a...

The dataframe with headings “polarity” and “tweet” is created to contain data more logically and systematically. This is important as it is quicker to read a dataframe than a list. This will improve times to plot the data. Each value has its respective index number starting from 0. Here it is clear there are still null values in the polarity data. As mentioned before, although this could be a true null value, it is more common it is not, therefore it is more sensible to remove all null values entirely to prevent data misinterpretation through a graph.

```
sentiment_df = sentiment_df[sentiment_df.polarity != 0]
sentiment_df
```

	polarity	tweet
0	0.700000	LarryCohen83 spectatorindex It feels too volat...
2	0.068182	SadafJadran Definitely me and you cant live wi...
3	0.250000	EQOS is my Bitcoin stock play on this most rec...
6	0.216667	CryptoBoj I see Bitcoinas ultimately becoming ...
9	-0.166667	snappycaster RAC The transactions I do with Bi...

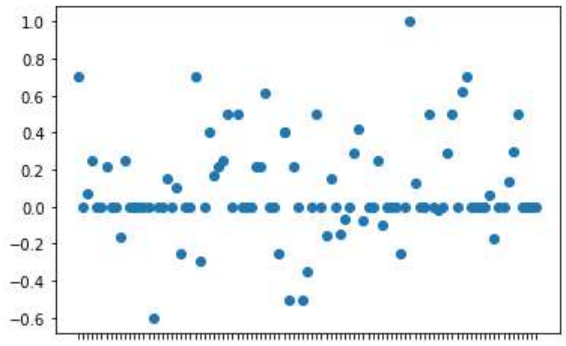
Test **06**:

To test the Polarity values are float values with no null values I will first print a few more data entries to gain a broader scope. I will also test the theory that false null values are more common than true null values. This can be done by plotting the data and seeing if there is an overwhelming number of zero values. Obviously I am testing live data so there will be some polarity bias to certain values that are overall truly measuring sentiment.

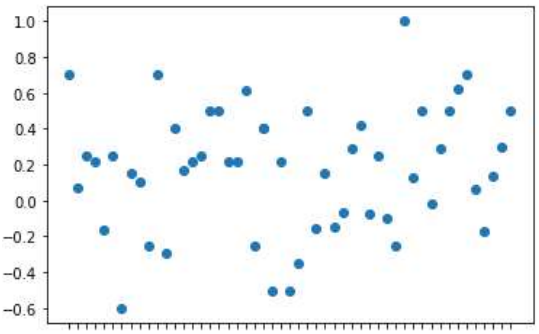
Benjamin Holmes

	polarity	tweet
0	0.700000	LarryCohen83 spectatorindex It feels too volat...
1	0.000000	Bitcoin missedEthereum missedBNB missedShiba I...
2	0.068182	SadafJadran Definitely me and you cant live wi...
3	0.250000	EQOS is my Bitcoin stock play on this most rec...
4	0.000000	About to start dont miss out on this Bitcoin a...
...	...	...
95	0.500000	GregAbbottTX Hmm I recall a couple bitcoin min...
96	0.000000	Colewherld RealShibaDoge To the moon shiba 20 ...
97	0.000000	Bitcoin has completed its proofitdoesntwork
98	0.000000	CoinCornerDanny I retweeted this because I am ...
99	0.000000	BTC 42378 whos feeling Bullish for cryptocurre...

Plotting this table:



This clearly shows the number of false null values. When I remove the null values, the graph looks much cleaner with easier data points to work with as there is less zero bias:



Benjamin Holmes

In this first iteration I will use a method to plot polarity values over time by calculating the average polarity of each day. This will meet Criteria **4.1**. Clearly this has limitations as it only shows sentiment day by day, however with complications of time formating at this stage in the development process it makes sense to get a rough idea of how polarity changes over time. To find the average polarity of each day, I first set the start and end dates to the same value - this collects data on that day only. Next, I divide the sum of the list of polarity values by the length of the list of polarity values. In order to systematically add this to a CSV file to collect data over time, I then create a dataframe with the respective headers (Criteria **4.2**).

```
avgpolarity = (sum(polarityvalues))/(len(polarityvalues))
dfavg = pd.DataFrame(columns=['Date', 'Average_Polarity'])
dfavg = dfavg.append({'Date': date2, 'Average_Polarity': avgpolarity}, ignore_index=True)
```

The ignore_index parameter removes the index to make appending to the CSV file less messy. By appending to the CSV file each runtime, with a new date being searched it bypasses the kernel erasing and allows us to store values permanently. This allows us to plot all the values later. The average polarity and date in the dataframe have the following format:

```
0.34648156737442454
```

	Date	Average_Polarity
0	2022-02-08	0.346482

With each iteration of the code, with a new date each time, data can begin to be collected. The reason for this is due to API rate limits restricting historic searching - the API can only pull tweets from at most 7 days ago. By regularly scraping and saving I can begin to build my own custom database that is not limited by Twitter's API. The CSV file after several scrapes:

Date	Average Polarity
02/02/2022	0.22114757
03/02/2022	0.21559007
04/02/2022	0.24757346
05/02/2022	0.26166042
06/02/2022	0.13039611
07/02/2022	0.17684694
08/02/2022	0.34648157

After I have collected sufficient data, I am able to read the file

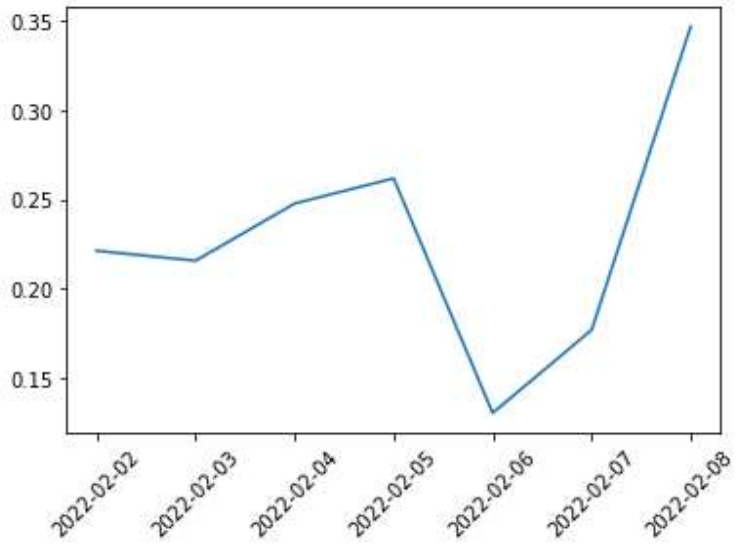
Benjamin Holmes

```
graph_data = pd.read_csv("date_avgpolarity1.csv")
```

And then plot the data values

```
fig, ax = plt.subplots()
ax.plot(graph_data.Date, graph_data.Average_Polarity)
ax.tick_params(axis='x', rotation=45)
plt.show()
```

This graph has the Date on the x-axis and the Average Polarity on the y-axis. I have set the rotation of the dates to 45° to ensure visibility of the whole date. This code produces an acceptable, although limited in values, graph. This meets Criteria **4.3**.



In order to see how this plot compares to real values, the live crypto prices must be collected, formatted and plotted. Messari can be used as a source in any public-facing application that is created for specific users or viewers without copying it directly. The database is also freely accessible and detailed so suits this system well.

As its own function called “get_crypto_price”, it takes 3 parameters: symbol, start and end (Criteria **5.1**/**5.2**). Within the function, the API is called and stored in a data frame.

```
def get_crypto_price(symbol, start, end):
    api_url = f'https://data.messari.io/api/v1/markets/binance-{symbol}-usdt/metrics/price/time-series?start={start}&end={end}&ir
    raw = requests.get(api_url).json()
```

Benjamin Holmes

As I am dealing with multiple parameters being entered into the URL string, I've chosen to use f-Strings rather than .format(). f -String expressions are evaluated at runtime and then formatted using the __format__ protocol. This means they are less prone to error, and faster, than other ways of formatting.

Next, I save the collected data to a data frame with two columns

```
df = pd.DataFrame(raw['data']['values'])
lista = raw['data']['values']
```

This produces the following table:

```
[[1640995200000, 46216.93, 47954.63, 46208.37, 47722.65, 19311.570649999998], [1641081600000, 47722.65, 47990, 46654, 47286.18,
18094.83401], [1641168000000, 47283.71, 47570, 45696, 46446.09, 27241.595119999998], [1641254400000, 46446.09, 47557.54, 45500,
45832.02, 33716.54109999995], [1641340800000, 45832.01, 47069.81, 42500, 43451.13, 50968.715949999954], [1641427200000, 43451.1
4, 43816, 42430.58, 43082.31, 38351.18905000001], [1641513600000, 43082.31, 43145.83, 40610, 41566.48, 54134.73333000003], [164
1600000000, 41569.49, 42300, 40501, 41679.74, 32691.90577000002], [1641686400000, 41679.74, 42786.7, 41200.02, 41864.62, 22532.
50093], [1641772800000, 41864.62, 42248.5, 39750, 41822.49, 39265.932019999986], [1641859200000, 41822.49, 43100, 41268.93, 427
29.29, 36845.45996000001], [1641945600000, 42729.29, 44322, 42450, 43902.66, 33158.07532], [1642032000000, 43902.65, 44500, 423
11.22, 42560.11, 34323.63224999999], [1642118400000, 42558.35, 43448.78, 41725.95, 43059.96, 31201.746700000014], [164220480000
0, 43059.97, 43800, 42555, 43084.3, 21672.19261], [1642291200000, 43084.3, 43475, 42581.79, 43071.65, 20394.170290000002], [164
2377600000, 43071.66, 43176.18, 41540.42, 42201.62, 27282.85607], [1642464000000, 42201.63, 42691, 41250, 42352.12, 28961.26065
999999], [1642550400000, 42352.12, 42559.13, 41138.56, 41660.01, 31261.178690000008], [1642636800000, 41660, 43505, 40553.31, 4
0680.92, 41227.005710000005], [1642723200000, 40680.91, 41100, 35440.45, 36450.01, 87467.16662599986], [1642809600000, 36445.3
1, 36835.22, 34008, 35071.42, 89230.16109500006], [1642896000000, 35071.42, 36499, 34601.01, 36244.55, 43410.457630000004], [16
42982400000, 36244.55, 37550, 32917.17, 36660.35, 89595.41550000003], [1643068800000, 36660.35, 37545.14, 35967.33, 36958.32, 3
6837.25550000003], [1643155200000, 36958.32, 38919.98, 36234.63, 36809.34, 68956.83265000003], [1643241600000, 36808.04, 37234.
47, 35507.01, 37160.1, 51936.398370000024], [1643328000000, 37160.11, 38000, 36155.01, 37716.56, 41672.55452000001], [164341440
0000, 37716.56, 38720.74, 37268.44, 38166.84, 26050.168540000002], [1643500800000, 38166.83, 38359.26, 37351.63, 37881.76, 2136
4.185450000004], [1643587200000, 37881.75, 38744, 36632.61, 38466.9, 36429.50015000005]]
```

Looking at the first list in the 2D array, the first value (index 0) is the time series format version of the date and time, this will need to be converted later. The next four values are the open , high, low and close price. The last value (index 5) is the volume. In order to find the average value of each of these price values I add each respective value to its own list then find the sum and divide by the length (Criteria **5.3**).

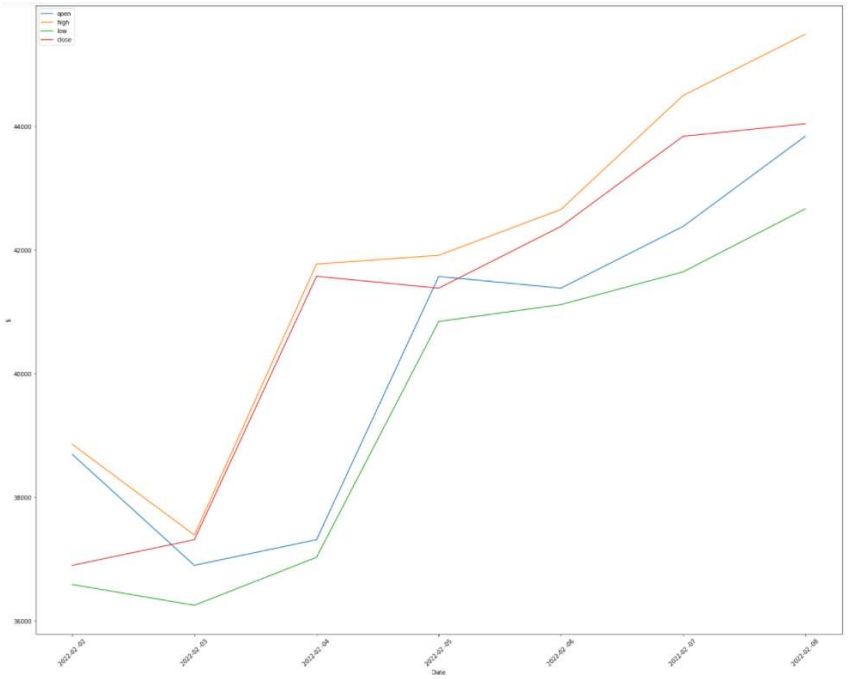
Benjamin Holmes

```
open = []
high = []
low = []
close = []
date = []
avg = []
for i in range (0,(len(lista))):
    date.append(datetime.datetime.fromtimestamp((lista[i][0])/1000).strftime('%Y-%m -%d'))
    open.append(lista[i][1])
    high.append(lista[i][2])
    low.append(lista[i][3])
    close.append(lista[i][4])

for i in range(0,len(lista)):
    avg.append((lista[i][1]+lista[i][2]+lista[i][3]+lista[i][4])/4)
```

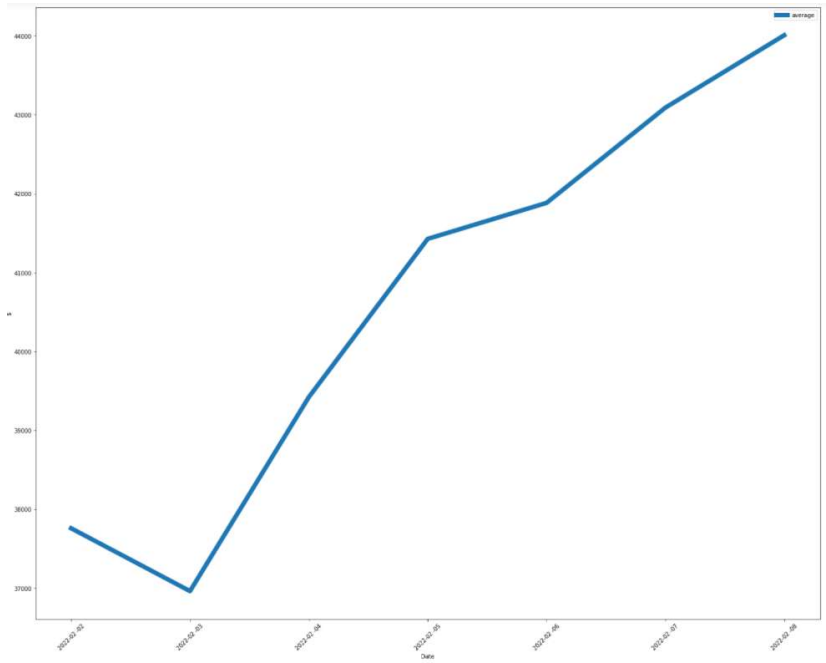
Appending to the average list must be done in a separate loop, despite the same conditions, as the lists have to be complete to find the average value. The date is formatted by first dividing the time series by 1000 - this is due to the original format usually including minutes and seconds. After this the date is split into its respective parts and separated by a dash.

To visualize open, high, low and close values on a graph I can plot them easily as they are already in list format (Criteria **5.4**).

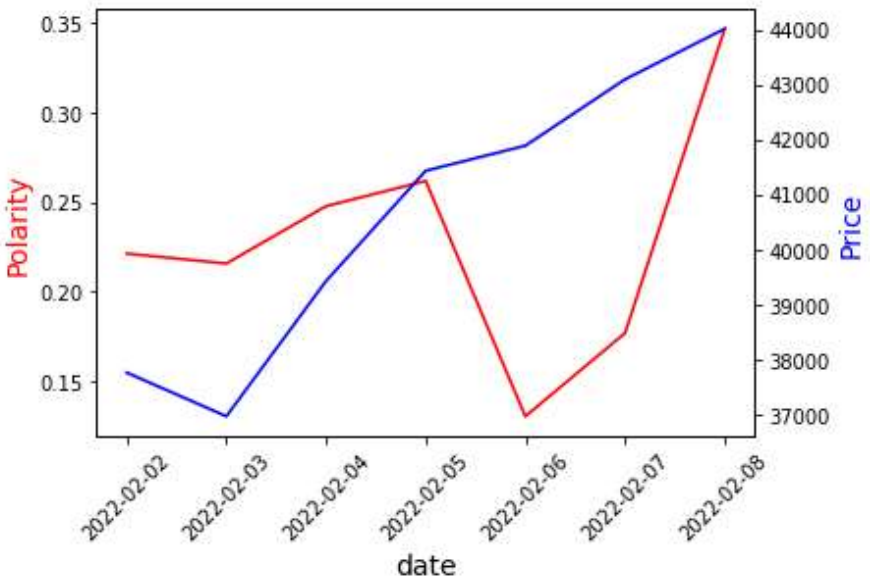


The average line can then be plotted:

Benjamin Holmes



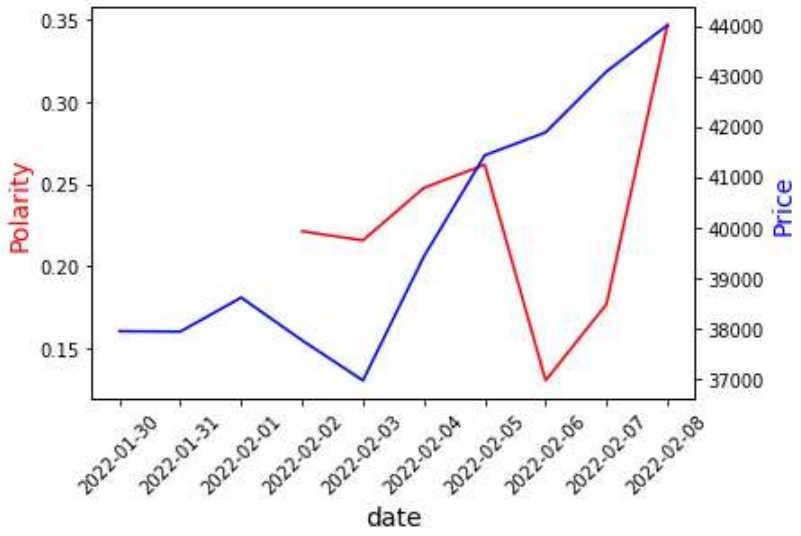
As both plots both share the same x-axis, I can use the code below to plot them on the same graph, allowing for a visual comparison. This is part of Criteria **6**



From this data we can see some, although minimal, correlation. It is important to remember the limited data set this represents, in terms of period of time, as well as number of tweets scrapped (<100) this may not give a full view of all the data. Furthermore, as part of this project is to look

Benjamin Holmes

at relationships between Twitter sentiment and real price, I can extend the date of the real price to see if there is a link between tweet polarity after changes in Bitcoin price.



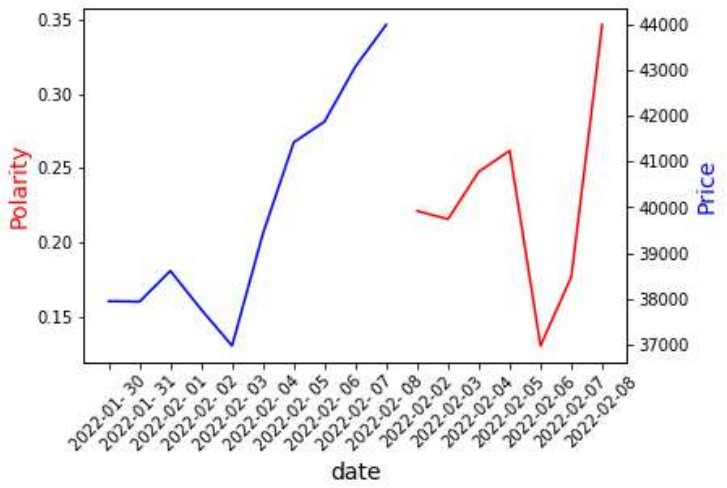
Whilst this could be coincidental, here we can see that tweet polarity seems to follow Bitcoin price - just with two days delay. Low price seems to result in low polarity and high price seems to result in high sentiment after a couple of days. From these findings I can begin to think about shifting data, as well as causation and effect when producing graphs for the stakeholder.

Test **15**:

Both graphs have regular x-axis spacing and a linear relationship. Sharing the x-axis, it is important those values are exactly the same otherwise graphs will plot separately. I found this during development due to a space in the datetime formatting causing ‘different’ dates. This is seen below:

```
date.append(datetime.datetime.fromtimestamp((lista[i][0])/1000).strftime('%Y-%m- %d'))
```

Benjamin Holmes



Once this error had been fixed, both graphs were able to share the axis. The scale of the y-axis only shows a small section of the graph, as not all values from 0 to the current price are needed, just like - in this case - all polarity values from -1 to 1.

First Iteration Review

This iteration aimed to provide a base level and provide the majority of functionality to the system. Despite not providing proper user output or developing a user interface, it has produced the framework for later iterations.

Success Criteria Developments:

Criteria	Criteria Met?	Next Iterations	Testing
1 Connect Python to Twitter API **1.1** Get access tokens and consumer keys **1.2** Install Libraries **1.3** Pull tweets	Yes	Minor changes to pulling tweets format	**01**: Valid **02**: Valid **03**: Working but untestable at this stage
2 Clean Data from Twitter **2.1** Remove URLS **2.2** Remove strange syntax	Mostly	More efficient cleaning and other types of data to exclude	**04**: Valid **05**: Valid
3 Calculate Sentiment Polarity **3.1** Install Textblob **3.2** Analyze Tweets	Partly	Polarity values have been calculated but subjectivity and analysis still need to	**06**: Valid **07**:N/A **08**:N/A

		be calculated.	
4 Sentiment Values **4.1** Average daily values **4.2** Store **4.3** Plot	Mostly	Changes to storage of values	**09**:Valid
5 Get real crypto prices **5.1** Choose time frame **5.2** Choose coin **5.3** Calculate average of open, high, low and close **5.4** Plot	Yes	Minimal	**10**:Valid
6 Compare Plots	Partly	More computational depth to comparison	**15**: Valid
7 Create User Interface **7.1** Search feature **7.2** Trending feature **7.3** Design	No	Create	**11**:N/A **12**:N/A **13**:N/A **14**:N/A

Second Iteration

In this iteration I will aim to continue to develop the system. This should involve making code more efficient as well as more computational. I also want to explore other types of plots in this iteration, whilst also beginning to develop a basic user interface. It is important to build upon

Benjamin Holmes

aspects already completed in the first iteration, as well as focus on areas laid out in the first iteration's review, as goals for 'next iterations'.

Firstly, I will rework the object that pulls the tweets. This is because I want values to be inputtable through just one function. Also, as I continue to work with daily polarity values, I only need one date rather than a start and end date. This one date should encompass the whole day of values (up to the rate limit or otherwise specified number).

```
def apiSearch(kw, number_of_tweets, tweets, user_id, created_at, date ):
    for i in tw.Cursor(api.search_tweets, q = kw , lang ="en", until = date ,tweet_mode = "extended").items(number_of_tweets):
        tweets.append(i.full_text)
        user_id.append(i.user.id)
        created_at.append(i.created_at)
```

Here, I initialize a Twitter Scraper object with search query, language, date, tweet mode and number of tweets attributes. Within the apiSearch function the data from this pull request is searched and tweets, dates and user id are added to respective lists.

In order to develop Criteria **1.3** as well as check Test **03**, the following code combines keywords through a list to develop search queries based on a stem word.

```
keyword_list = ['Investors', 'Price', 'Interest', 'Cryptocurrency']
keyword = 'Bitcoin'
for i in range (len(keyword_list)):
    kw = str(keyword + ' ' + keyword_list[i])
    apiSearch(kw, number_of_tweets, tweets, user_id, created_at, date)
```

It is within this code that the 'Crawling Data from Tweets on Keyword' is used. By pulling the tweets within this loop it ensures every variation of keyword is searched upon. To ensure this is occurring we use Test **03**.

Test **03:**

Benjamin Holmes

```
keyword_list = ['Investors', 'Price', 'Interest', 'Cryptocurrency']
keyword = 'Bitcoin'
for i in range (len(keyword_list)):
    kw = str(keyword + ' ' + keyword_list[i])
    print(kw)
```

```
Bitcoin Investors
Bitcoin Price
Bitcoin Interest
Bitcoin Cryptocurrency
```

As each value in the list is concatenated to the main keyword this is an example of typical data. In order to test erroneous data I must leave the keyword list empty and see if the loop only searches on the main keyword. With the code above, if the additional keyword list is empty, the length of the list will be 0 so the loop with that condition will not run - producing no values.

If I want the system to search upon the main key word if there are no additional keywords I must use an if statement.

```
keyword_list = []
keyword = 'Bitcoin'
if not keyword_list:
    kw = keyword
    print(kw)
else:
    for i in range (len(keyword_list)):
        kw = str(keyword + ' ' + keyword_list[i])
        print(kw)
```

Here, the program checks if the list is equal to not true or false (as empty lists have a boolean value of false). If this condition is met it only prints the main keyword, otherwise the original statement will run. The output with an empty list is seen below:

```
Bitcoin
```

The output with values within the list is seen below. Notice that the keyword is not printed by itself.

```
Bitcoin Investors
Bitcoin Price
Bitcoin Interest
Bitcoin Cryptocurrency
```

As missing additional keywords are dealt with this test is valid in this non-user input variation of the iteration.

As a continuation of cleaning data, it is important to remove retweets. This can be done after creating a dataframe.

Benjamin Holmes

```
df = pd.DataFrame({"Created_at": created_at,"User_id": user_id,"Tweets":tweets})
df = df[~df.Tweets.str.contains("RT")]
df = df.reset_index(drop = True)
```

I also want to ensure that data is thoroughly cleaned, whilst also being efficient. To do this I will create a new cleaner function that has more functionality, following the 'Cleaning Tweets' algorithm.

```
clean_tweets = []
def tweet_cleaner(tweet_column):
    for tweet in tweet_column:
        tweet = re.sub("@[A-Za-z0-9]+","",tweet) # Remove mentions @
        clean_tweets.append(tweet)
    return clean_tweets
clean_tweets = tweet_cleaner(df['Tweets'])
df['Tweets'] = clean_tweets
df
```

Rather than looping through a list like last iteration, this code enables the program to filter through a dataframe column which is much quicker for large amounts of data. To check this development is working properly I will use Test **05**

Test **05**

To test if the data frame contains the correct cleaned data I will print the output.

24	2022-02-05 22:41:03+00:00	574690409	But sir, 3% annual interest. Put your bitcoin...
25	2022-02-05 22:30:36+00:00	1315568371753259008	#Crypto Whales vs. #Stablecoins \n\n Last ...
26	2022-02-05 22:22:21+00:00	1480665645327036417	Google Trends worldwide search interest for 'N...
27	2022-02-05 23:59:15+00:00	997855218011463680	4 hour top movers report #blockchain #crypto #...
28	2022-02-05 23:59:11+00:00	817001274776416256	Bitcoin And The Downslope Of Community Results...
29	2022-02-05 23:59:08+00:00	1421551417953853445	Cryptocurrency monitored and organized by pee...
30	2022-02-05 23:59:03+00:00	1487747806827790337	Check out our new Instagram _NFT \n\n \n\n...

This clearly still includes emojis, hashtags and other unwanted characters. To fix this I change the conditions to ones that will replace these characters with empty spaces.

This produces the following output:

Benjamin Holmes

```
clean_tweets = []
def tweet_cleaner(tweet_column):
    for tweet in tweet_column:
        tweet = re.sub("([^0-9A-Za-z \t])|(\w+:\/\/\S+)", "",tweet)
        clean_tweets.append(tweet)
    return clean_tweets
clean_tweets = tweet_cleaner(df['Tweets'])
df['Tweets'] = clean_tweets
df
```

24	2022-02-05 22:41:03+00:00	574690409	FriarHass But sir 3 annual interest Put your b...
25	2022-02-05 22:30:36+00:00	1315568371753259008	Crypto Whales vs Stablecoins Last 24 hours o...
26	2022-02-05 22:22:21+00:00	1480665645327036417	Google Trends worldwide search interest for NF...
27	2022-02-05 23:59:15+00:00	997855218011463680	4 hour top movers report blockchain crypto cry...
28	2022-02-05 23:59:11+00:00	817001274776416256	Bitcoin And The Downslope Of CommunityResults
29	2022-02-05 23:59:08+00:00	1421551417953853445	BitcoinMagazine Cryptocurrency monitored and o...
30	2022-02-05 23:59:03+00:00	1487747806827790337	Check out our new Instagram RatCamNFT fintec...

Therefore this code is valid and works very efficiently to clean all the data in the dataframe.

After this I can move on to implementing the ‘Qualitative Score from Data’ algorithm. In this iteration, it involves first creating functions that use TextBlob to find the polarity and subjectivity scores in a modular way.

These values must then be added to new columns in the data frame.

```
def getSubjectivity(text):
    return TextBlob(text).sentiment.subjectivity

def getPolarity(text):
    return TextBlob(text).sentiment.polarity
```

The algorithm then focuses on determining qualitative values depending on these scores. This allows for more variation in graphical representation later on as data can be grouped. In this algorithm, data is labeled either ‘Positive’, ‘Negative’ or ‘Neutral’. It is then added to another column in the data frame. As data is managed atomically, each value is lined up with the tweet that it represents, no data should be unordered or misplaced.

Benjamin Holmes

So that data can accumulate and it is not reliant on the Twitter API over time, I then add this data frame to a CSV file which can be forwarded to the SQL database.

```
# Function compute negative, positive
def getAnalysis(score):
    if score < 0:
        return 'Negative'
    elif score > 0:
        return 'Positive'
    else:
        return 'Neutral'

df['Analysis'] = df['Polarity'].apply(getAnalysis)
```

Importantly, the first line is involved in the creation of the CSV file so only needs to be called

```
df.to_csv('twitter123.csv') #initial creation and first interation
with open('twitter123.csv', 'a') as f:
    df.to_csv(f, header=False)
#append data to csv rather than rewrite so data can accumulate
```

once. If this line of code is used repeatedly, data will not be able to accumulate as the standard for the .to_csv() method is ‘w’ which means rewrite. For this reason, every runtime apart from the first will use the ‘with open’ method of appending to a CSV file. The header is set to False to dismiss new headings being created each time a data frame is appended - this would alter data reading.

In order to read the accumulated data once a graph plot is needed, I save the contents into a global variable sample_data. The benefit of this is the program can access the global variable from all the functions or modules within the program.

Next, as part of test **06** and **07**, I must ensure that all null values are removed from the dataframe. This can be done by updating the sample_data to include the column of

```
sample_data = pd.read_csv("twitter123.csv")
```

```
sample_data = sample_data[sample_data.Polarity != 0]
sample_data = sample_data[sample_data.Subjectivity != 0]
```

polarity/subjectivity values that do not equal 0.

In order to check this is completed properly I use tests **06** and **07.**

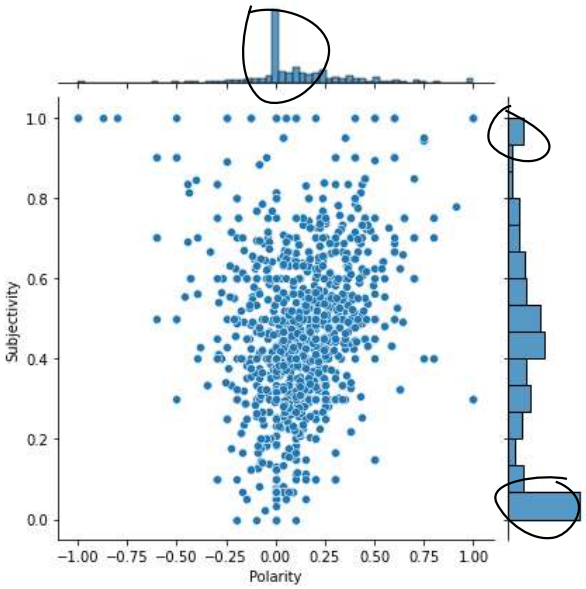
Benjamin Holmes

Test **06** and **07**:

First I will check the CSV file before filtering out the null values:

Subjectivity	Polarity
0.000000	0.000000
0.448000	0.102000
0.454545	0.136364
0.800000	0.166667
0.637500	0.137500
...	...
0.333333	0.250000
0.000000	0.000000
0.000000	0.000000
0.329167	0.120833
1.000000	0.600000

Clearly, absolute null values are much more common than other values - as mentioned in the first iteration. What I also must consider is subjectivity values of absolute one, also being more common than other values. In order to test this I will plot a graph of polarity and subjectivity.



Using the bar chart as well as the scatter graph evidently shows the emphasis on polarity 0 values, as well as subjectivity 0 and 1 values. For this reason it is significant to remove all three groups to reduce data bias. To remove the subjectivity values of 1 I employ a similar process as removing the null values.

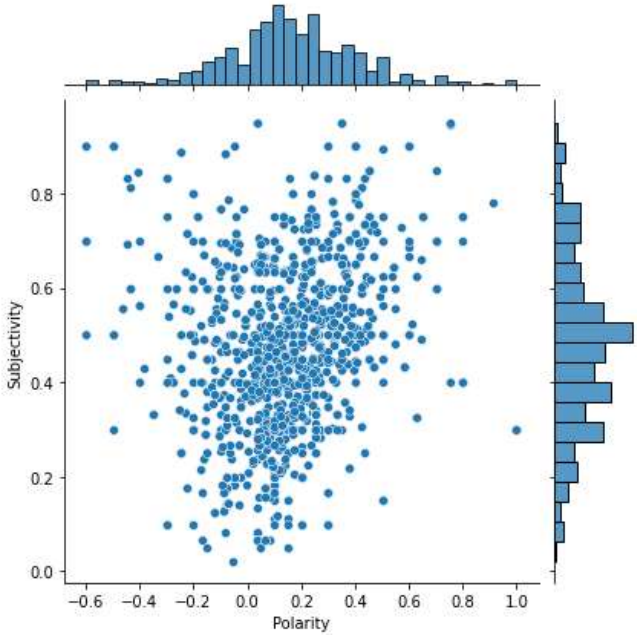
```
sample_data = sample_data[sample_data.Subjectivity != 1]
```

After removing all the null and one values the CSV files looks much cleaner:

Subjectivity	Polarity
0.448000	0.102000
0.454545	0.136364
0.800000	0.166667
0.637500	0.137500
0.650000	0.400000

Benjamin Holmes

Therefore, plotting the previous graph again, except with the new data:



This shows much less extreme extremities of the bar chart. The reason the middle portion of the charts produce a peak is simply due to probability favoring middle positions in a normal distribution - this does not call for data removal.

As all erroneous data has been removed tests **06** and **07** are valid.

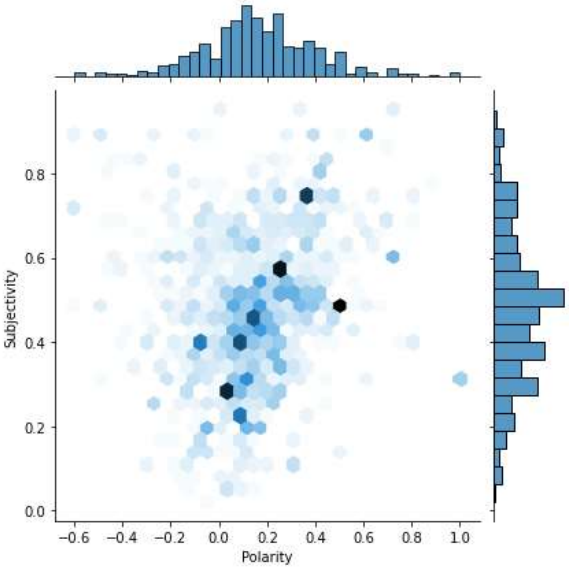
Moving on to plotting data, I can also plot subjectivity values against polarity values first as a hex plot, and secondly as a Kernel Density Estimation. This is good for data visualization - allowing stakeholders to better view data, ultimately affecting decision making.

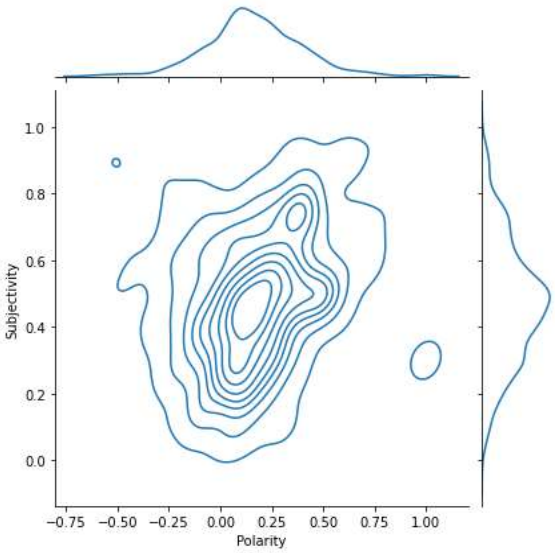
Benjamin Holmes

Hex Plot:

On this graph we can see that the darker gradient hexagons represent denser values.

Kernel Density Estimation:





Test **15**:

Checking the graphs above have all data plotted correctly, we can see regular intervals,

with minimal anomalies, labeled axis and appropriate scales. This shows test **15** is valid.

As the back end of the second iteration is now mostly complete, I will begin work on the user interface (Criteria **7**). As laid out in the User Interface Design section, this should be a web application that displays the relevant information the stakeholder would need to make informed decisions. In this second iteration, the user interface will only begin development.

To create this application I have chosen to use Streamlit as there are many advantages that suit this system:

- It embraces Python scripting without the need for html
- Accessible and intuitive to use (user end)
- No callbacks are needed since widgets are treated as variables
- Data caching simplifies and speeds up computation pipelines.

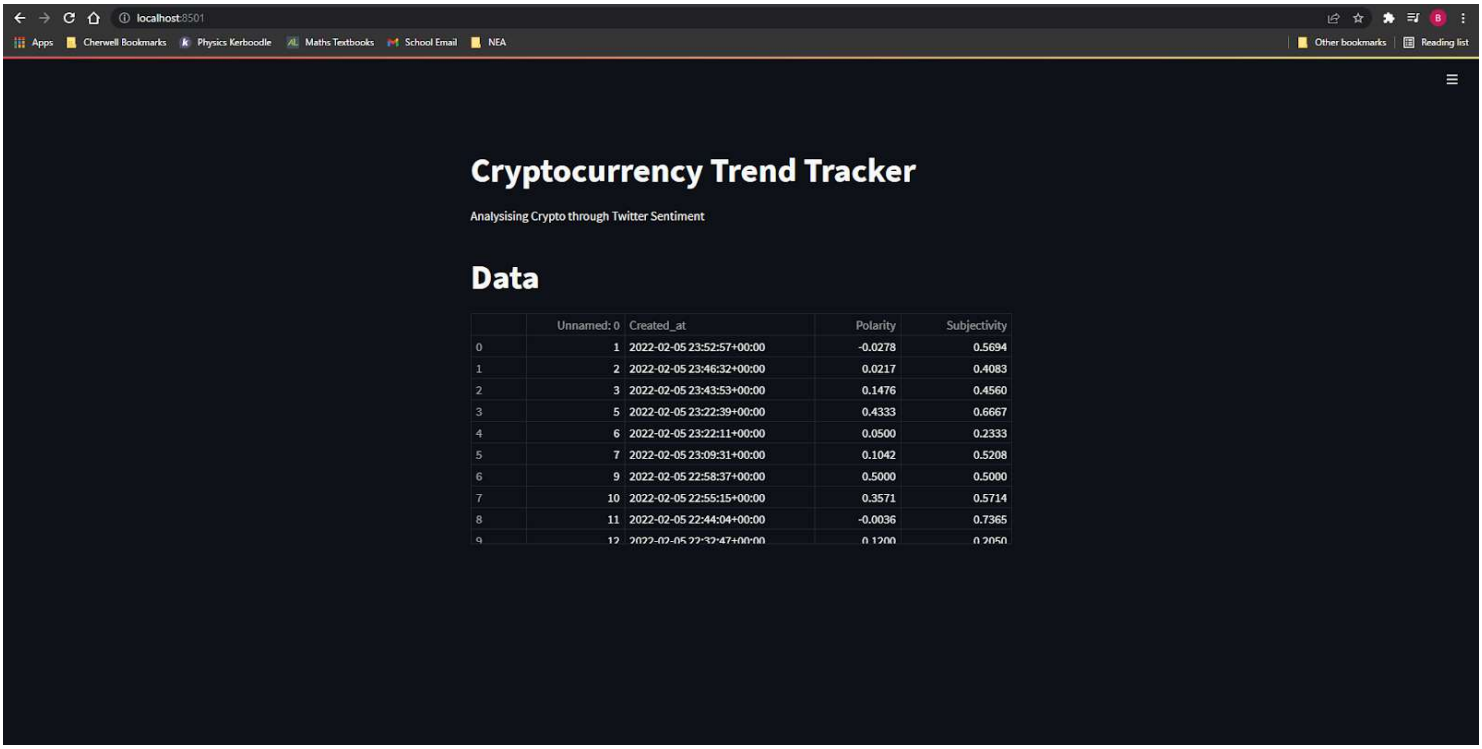
To first set up the local host I install Streamlit and import the file I want to display.

```
st.write("""
# Cryptocurrency Trend Tracker
Analysising Crypto through Twitter Sentiment
""")

df = pd.read_csv("UIdata.csv")
st.title("Data")  # add a title
st.write(df)
```

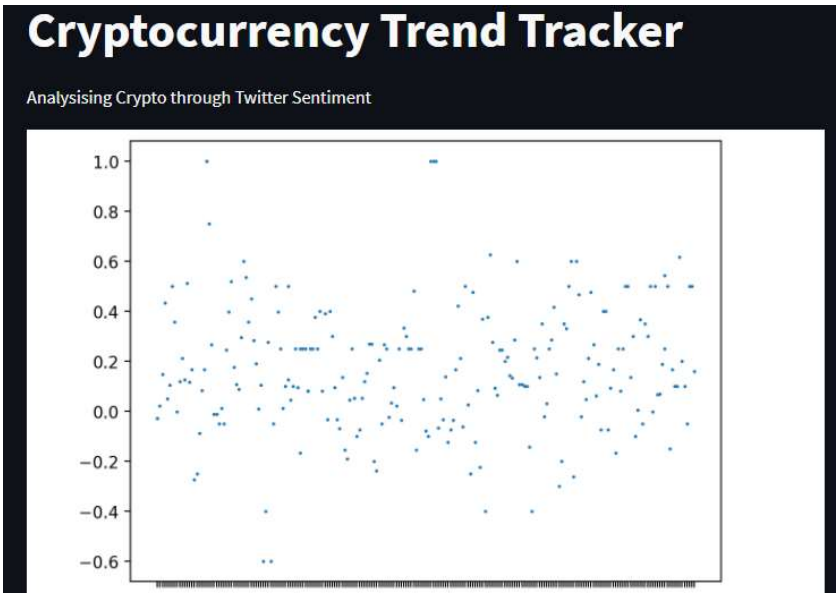
This produces the web application:

Benjamin Holmes



	Unnamed: 0	Created_at	Polarity	Subjectivity
0	1	2022-02-05 23:52:57+00:00	-0.0278	0.5694
1	2	2022-02-05 23:46:32+00:00	0.0217	0.4083
2	3	2022-02-05 23:43:53+00:00	0.1476	0.4560
3	5	2022-02-05 23:22:39+00:00	0.4333	0.6667
4	6	2022-02-05 23:22:11+00:00	0.0500	0.2333
5	7	2022-02-05 23:09:31+00:00	0.1042	0.5208
6	9	2022-02-05 22:58:37+00:00	0.5000	0.5000
7	10	2022-02-05 22:55:15+00:00	0.3571	0.5714
8	11	2022-02-05 22:44:04+00:00	-0.0036	0.7365
9	12	2022-02-05 22:32:47+00:00	0.1200	0.2050

From here, I can begin customizing the display, inputs, outputs and interactive features. First I want to know I can plot graphs I have made in the previous step. This can be done with a pyplot extension.



Benjamin Holmes

Next, the input sidebar section must be created in its first form. This should have sections for the search type, ticker name, start date and end date. Each of these inputs should be entered in an appropriate way and formatted to suit the program. First I will start with creating the side bar, then making the search type section.

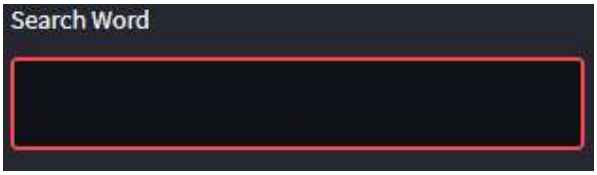
```
st.sidebar.markdown("## Search Twitter")
select_event = st.sidebar.selectbox('Search Type',
                                    ['Custom', 'Trending'])
```

This part of the code shows a selectbox being created, after inheriting attributes of the sidebar object as well as the streamlit (st) class. The select box has a title then two options which should appear as a drop down menu, to satisfy Criteria **7.2**. This creates the input section seen below:

This selection type works better than tick boxes (that were in the original user interface design) as it allows for sticky selection once searching the entire query further down the side bar. Next I need to create the search input section.

```
enter_search = st.sidebar.text_input("Search Word")
```

This creates the text box below:

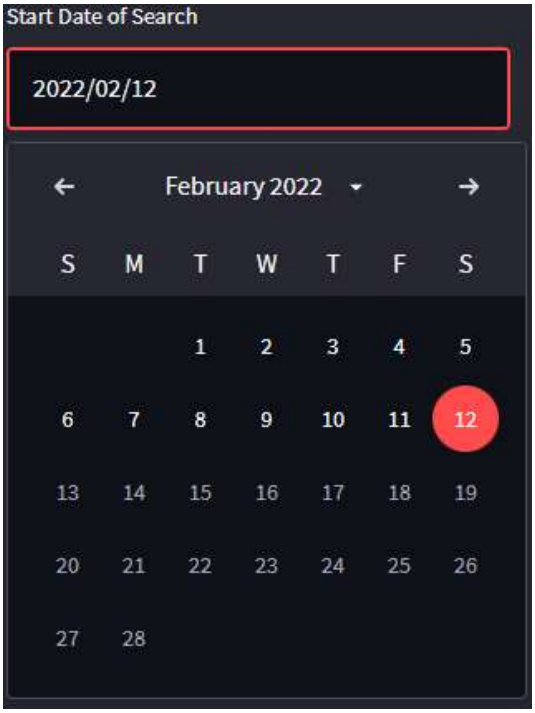


Benjamin Holmes

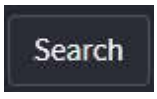
After this I can use a calendar widget to create start and end date inputs, I must limit these dates to only go up to the current date, and only go back as far as the data collected goes. This can be done by changing attributes of the date_input object.#

```
select_start_date = st.sidebar.date_input('Start Date of Search',
                                          min_value=(datetime.datetime(2022, 2, 1)),
                                          max_value=(date.today()))
select_end_date = st.sidebar.date_input('End Date of Search',
                                        min_value=(datetime.datetime(2022, 2, 1)),
                                        max_value=(date.today()))
```

It is still valid if the dates of the start and end dates are the same as that will result in a search over that 24 hour period. The code above produces the following display for both the start and end date inputs:



In order to send the imputed data to the system a final user input must be used to complete the ‘form’. In this case I use a search button:



The first iteration of this user interface as a whole can be seen below:

Benjamin Holmes

As we can see this is a basic development of the overall system user interface. As a base for the final iteration we can go on to link this display to the code and develop design to be more user friendly. Even including other capabilities such as a download button to copy the displayed data on to a users own system.

Second Iteration Review

The second iteration aimed at building upon first iteration concepts, as well as developing new methods to meet the success criteria. Part of this involved beginning the creation of a user interface. Despite this not being fully complete, it lays the foundation for the final iteration to complete the build of this system. With most other areas being developed to work effectively, this iteration has fulfilled most areas of the success criteria to an acceptable level.

Criteria	Criteria Met?	Next Iterations	Testing
1 Connect Python to Twitter API **1.1** Get access tokens and consumer keys **1.2** Install Libraries **1.3** Pull tweets	Yes	Minor changes	**01**: Valid **02**: Valid **03**: Valid
2 Clean Data from Twitter **2.1** Remove URLS	Yes	Minor changes	**04**: Valid **05**: Valid

2.2 Remove strange syntax			
3 Calculate Sentiment Polarity **3.1** Install Textblob **3.2** Analyze Tweets	Yes	Minor changes	**06**: Valid **07**:Valid **08**:Valid
4 Sentiment Values **4.1** Average daily values **4.2** Store **4.3** Plot	Mostly	Minor changes	**09**:Valid
5 Get real crypto prices **5.1** Choose time frame **5.2** Choose coin **5.3** Calculate average of open, high, low and close **5.4** Plot	Yes	Minor changes	**10**:Valid
6 Compare Plots	Partly	More computational depth to comparison	**15**: Valid
7 Create User Interface **7.1** Search feature **7.2** Hot feature **7.3** Design	Partly	Link with program and further improve design	**11**:Untestable at this stage **12**:Untestable at this stage **13**:Untestable at this stage **14**:Untestable at this stage

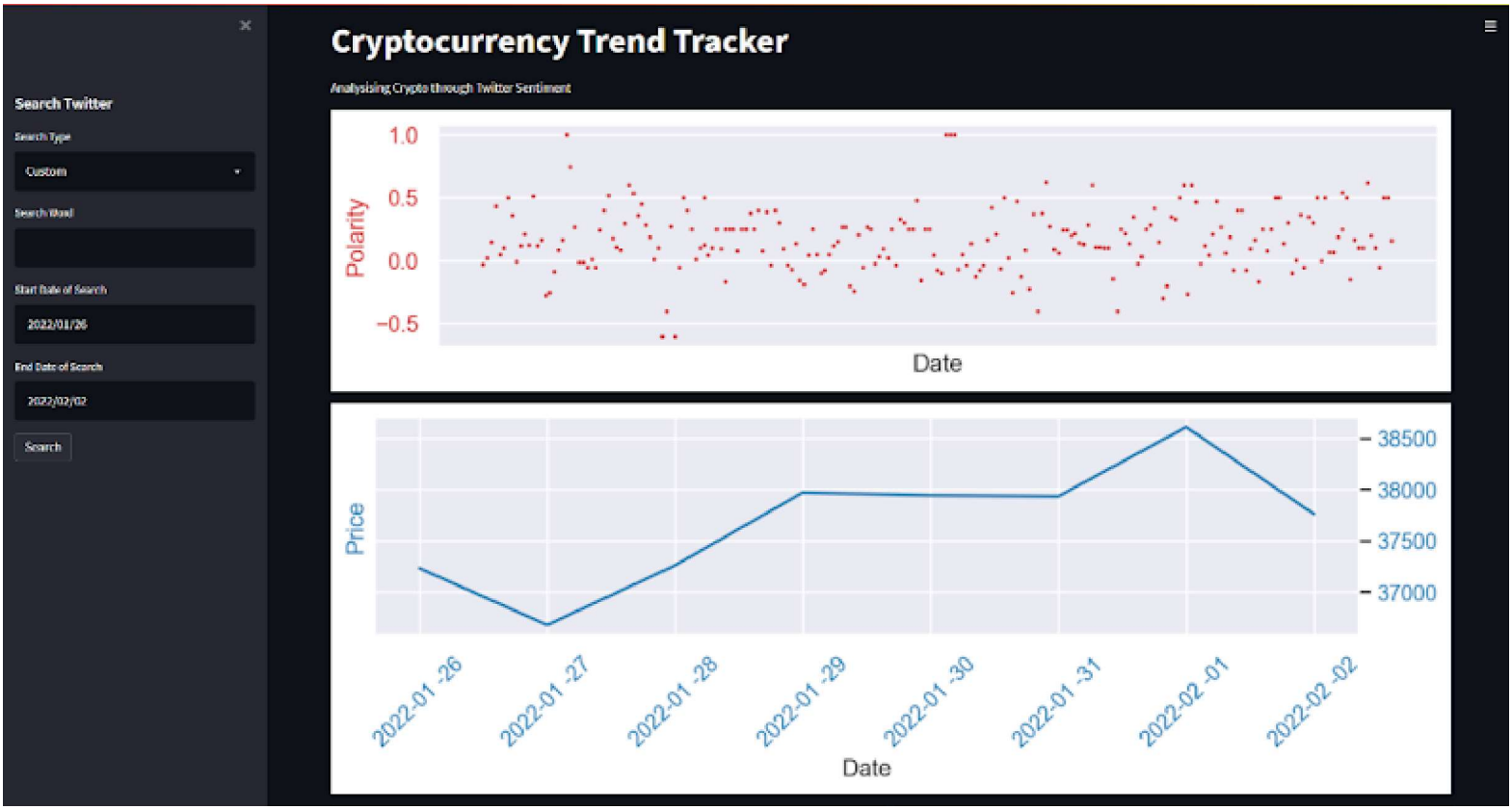
Benjamin Holmes

Final Iteration

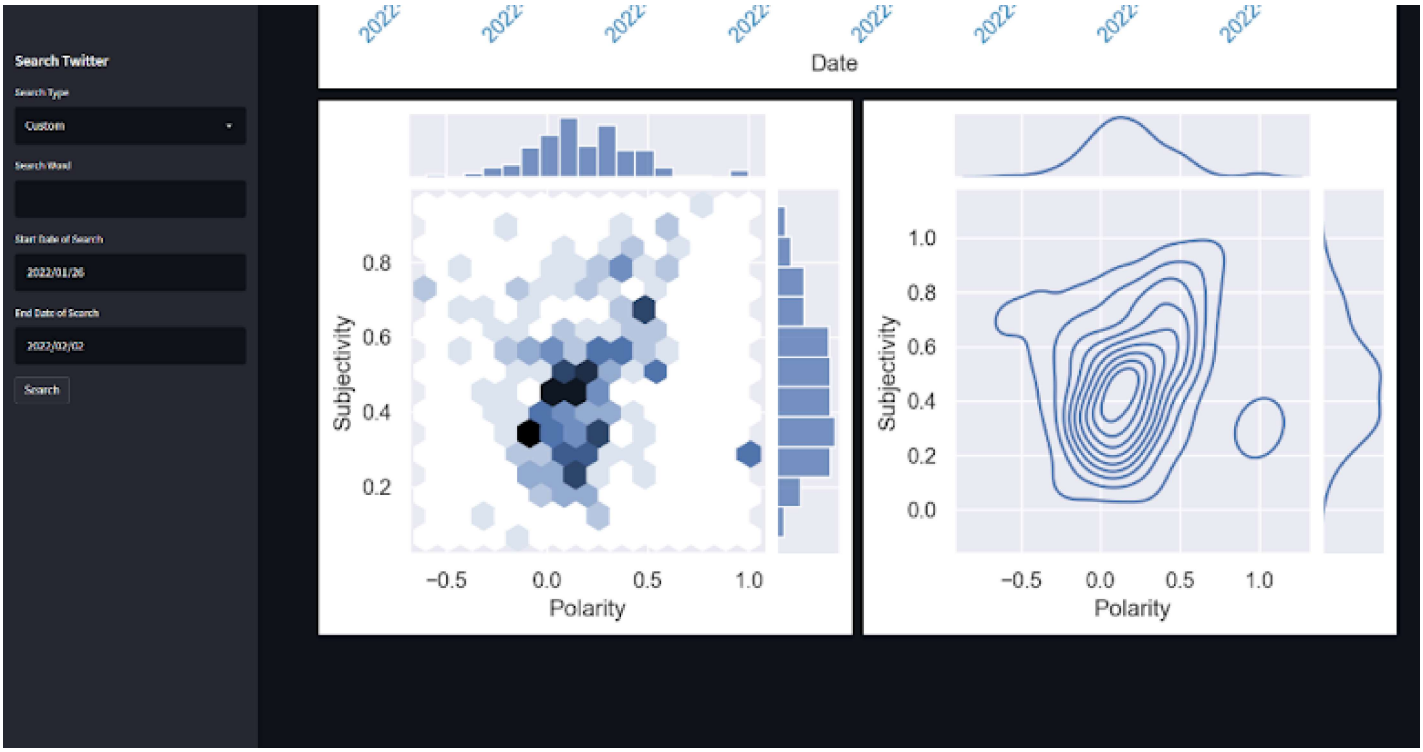
In this final iteration, I will integrate the backend data and code to the user interface, as the streamlit application is fairly intuitive, this stage does not require much evidence as it is a matter of copying modules of code from the first two iterations into a master program that is integrated with the temporary CSV files, the user interface, as well as the permanent SQL database. This iteration will also involve improving visual design for user experience as well as testing user input fields.

After adding more graphs and connecting the backend the result looks more like the vision set out in the User Interface Design Section.

Benjamin Holmes



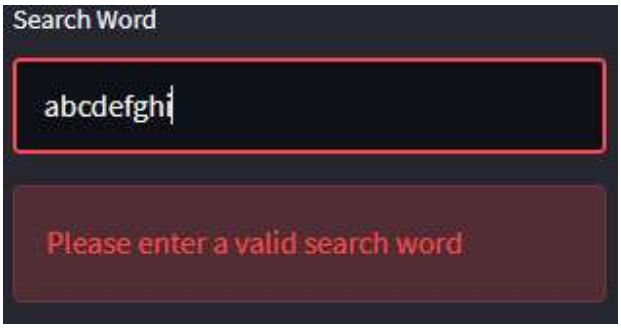
Benjamin Holmes



In order to test and validate user inputs I must use tests **11** and **12**.

Test **11**:

This test ensures that the search query/ ticket is entered correctly. This needs to be a capital letter string with 3-7 characters. As expected, if I enter a valid token, there will be no success message, only a successful search. Entering an invalid token, however, produces the following error.

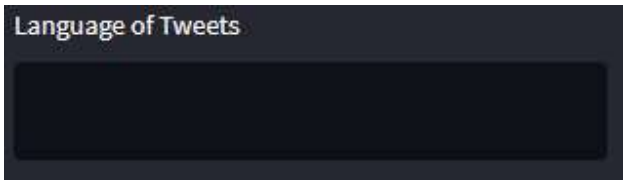


The same occurs if the section is left blank whilst set on custom mode.

Benjamin Holmes

Test **12**:

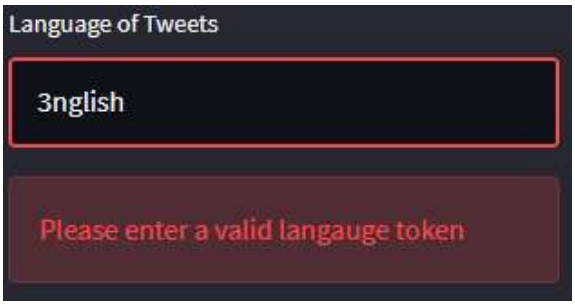
In order to test whether the language is inputted correctly, I must first create this field.



Next I must validate whether the two character string is within the list of valid languages, Twitter can search on. This is from the list below:

```
langs = {'ar': 'Arabic', 'bg': 'Bulgarian', 'ca': 'Catalan', 'cs': 'Czech', 'da': 'Danish', 'de': 'German',
         'el': 'Greek', 'en': 'English', 'es': 'Spanish', 'et': 'Estonian', 'fa': 'Persian', 'fi': 'Finnish',
         'fr': 'French', 'hi': 'Hindi', 'hr': 'Croatian', 'hu': 'Hungarian', 'id': 'Indonesian', 'is': 'Icelandic',
         'it': 'Italian', 'iw': 'Hebrew','ja': 'Japanese', 'ko': 'Korean', 'lt': 'Lithuanian', 'lv': 'Latvian',
         'ms': 'Malay', 'nl': 'Dutch', 'no': 'Norwegian', 'pl': 'Polish', 'pt': 'Portuguese', 'ro': 'Romanian',
         'ru': 'Russian', 'sk': 'Slovak', 'sl': 'Slovenian', 'sr': 'Serbian', 'sv': 'Swedish', 'th': 'Thai',
         'tl': 'Filipino', 'tr': 'Turkish', 'uk': 'Ukrainian', 'ur': 'Urdu',
         'vi': 'Vietnamese', 'zh_CN': 'Chinese (simplified)', 'zh_TW': 'Chinese (traditional)'}
```

Similarly to the token input section, if an incorrectly formatted string is now entered into this text box an error message now appears.



Benjamin Holmes

In the final section of this iteration I want to develop Criteria **6** and compare the real price line graph and the polarity scatter graph. In order to do this computationally, I create an image classifier that can tell how similar two images, or in this case graphs, are. For that, there is no need for any complicated libraries like TensorFlow or image classification models. There are two ways to find if an image is similar to another image. First is to look at Mean Square Error (MSE) and the second is Structural Similarity Index (SSIM). The equation for these indexes are seen below:

$$MSE = \frac{1}{m\,n}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$

$$\text{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

MSE will calculate the mean square error between each pixel for the two images we are comparing. Whereas SSIM will do the opposite and look for similarities within pixels; i.e. if the pixels in the two images line up and or have similar pixel density values. The only issue is that MSE tends to have arbitrarily high numbers so it is harder to standardize it. While generally the higher the MSE the least similar they are, if the mse between picture sets differ appears randomly, it will be harder for us to tell anything. SSIM on the other hand puts everything in a scale of -1 to 1. A score of 1 meant they are very similar and a score of -1 meant they are very different. In the code below I implement these indexes:

Comparison function:

```
def compare_images(imageA, imageB):
# compute the mean squared error and structural similarity index for the images
    m = mse(imageA, imageB)
    s = ssim(imageA, imageB)
    return m,s
```

MSE function:

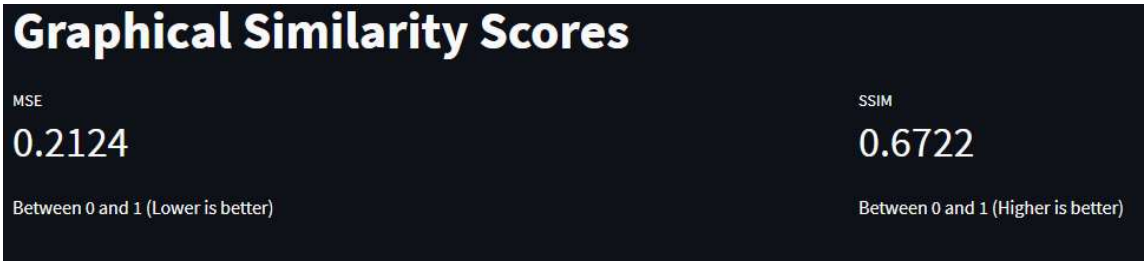
Benjamin Holmes

SSIM Function:

```
def ssim(imageA, imageB, cs_map=False):
    imageA = imageA.astype(numpy.float64)
def mse(imageA, imageB):
    err = np.sum((imageA.astype("float") - imageB.astype("float")) ** 2)
    err /= float(imageA.shape[0] * imageA.shape[1])
    return err
    K2 = 0.03
    L = 255 #bitdepth of image
    C1 = (K1*L)**2
    C2 = (K2*L)**2
    mu1 = signal.fftconvolve(window, imageA, mode='valid')
    mu2 = signal.fftconvolve(window, imageB, mode='valid')
    mu1_sq = mu1*mu1
    mu2_sq = mu2*mu2
    mu1_mu2 = mu1*mu2
    sigma1_sq = signal.fftconvolve(window, imageA*imageA, mode='valid') - mu1_sq
    sigma2_sq = signal.fftconvolve(window, imageB*imageB, mode='valid') - mu2_sq
    sigma12 = signal.fftconvolve(window, imageA*imageB, mode='valid') - mu1_mu2
    if cs_map:
        return (((2*mu1_mu2 + C1)*(2*sigma12 + C2))/((mu1_sq + mu2_sq + C1)*
                    (sigma1_sq + sigma2_sq + C2)),
                (2.0*sigma12 + C2)/(sigma1_sq + sigma2_sq + C2))
    else:
        return ((2*mu1_mu2 + C1)*(2*sigma12 + C2))/((mu1_sq + mu2_sq + C1)*
                    (sigma1_sq + sigma2_sq + C2))
```

The above function uses the scipy library. Once comparing using the first function, two values will be produced. These show how related the two graphs are and can be used to judge how accurate the sentiment analysis is in order to test how reliable the polarity graph is. If the user determines the polarity graph is similar enough to the real prices (using these scores), they may choose to use the polarity graph to invest in a coin that seems to have a high sentiment - usually resulting in a higher future real price. This can be used to make money.

Displaying these scores on the user interface, after running the code, can be seen below:



Final Iteration Review

In this iteration I completed the tasks laid out to be done in the second iteration's review. After testing inputs and further comparing graphs computationally, the success criteria and tests during development are mostly achieved. The final test table below shows met requirements.

Benjamin Holmes

Criteria	Criteria Met?	Next Iterations	Testing
1 Connect Python to Twitter API **1.1** Get access tokens and consumer keys **1.2** Install Libraries **1.3** Pull tweets	Yes	N/A	**01**: Valid **02**: Valid **03**: Valid
2 Clean Data from Twitter **2.1** Remove URLS **2.2** Remove strange syntax	Yes	N/A	**04**: Valid **05**: Valid
3 Calculate Sentiment Polarity **3.1** Install Textblob **3.2** Analyze Tweets	Yes	N/A	**06**: Valid **07**:Valid **08**:Valid
4 Sentiment Values **4.1** Average daily values **4.2** Store **4.3** Plot	Mostly	N/A	**09**:Valid
5 Get real crypto prices **5.1** Choose time frame **5.2** Choose coin **5.3** Calculate average of open, high, low and close **5.4** Plot	Yes	N/A	**10**:Valid
6 Compare Plots	Yes	N/A	**15**: Valid
7 Create User Interface **7.1** Search feature **7.2** Trending feature **7.3** Design	Mostly	N/A	**11**:Valid **12**:Valid **13**:Valid **14**:Valid

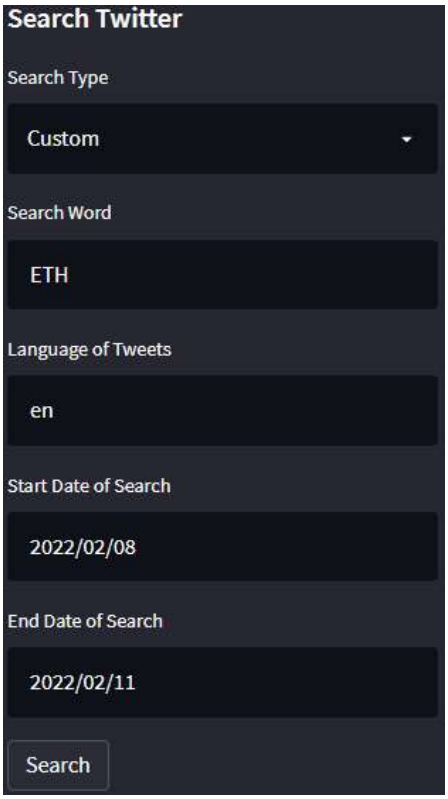
Benjamin Holmes

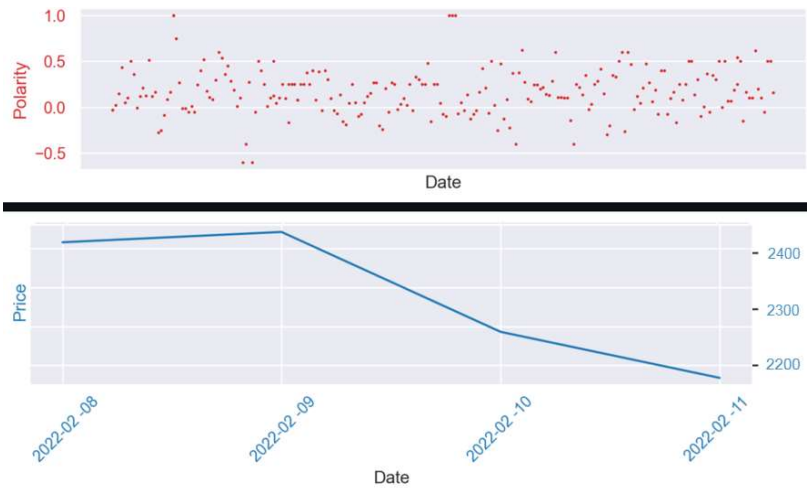
Evaluation

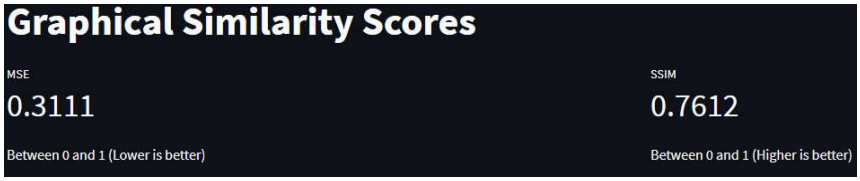
Testing to inform evaluation

To ensure the project works as desired as a whole, I can try various different inputs to see expected outputs.

ETH:

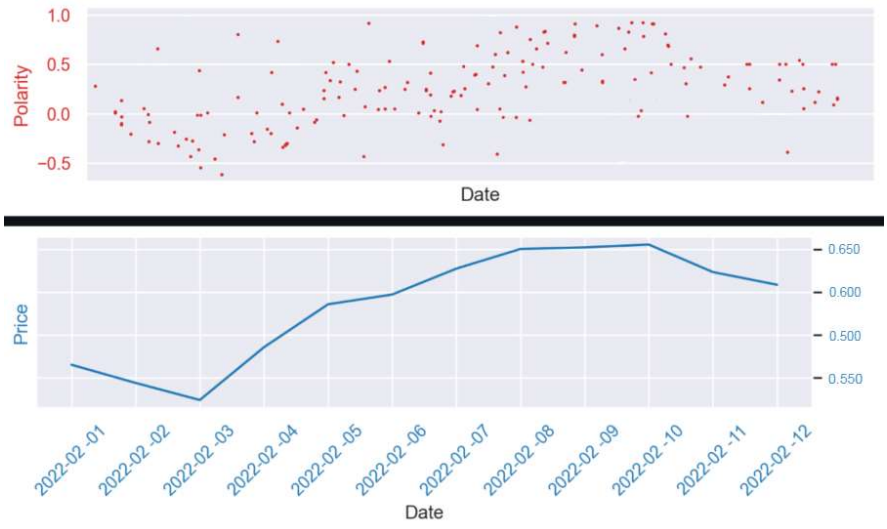


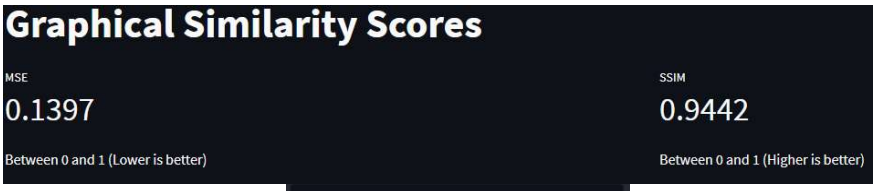




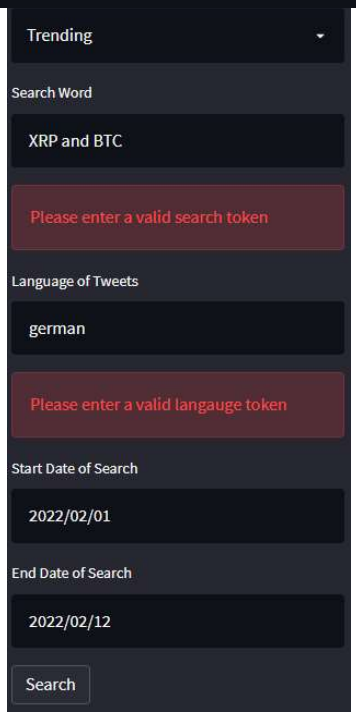
XRP:

Benjamin Holmes





Testing input validation as a whole:

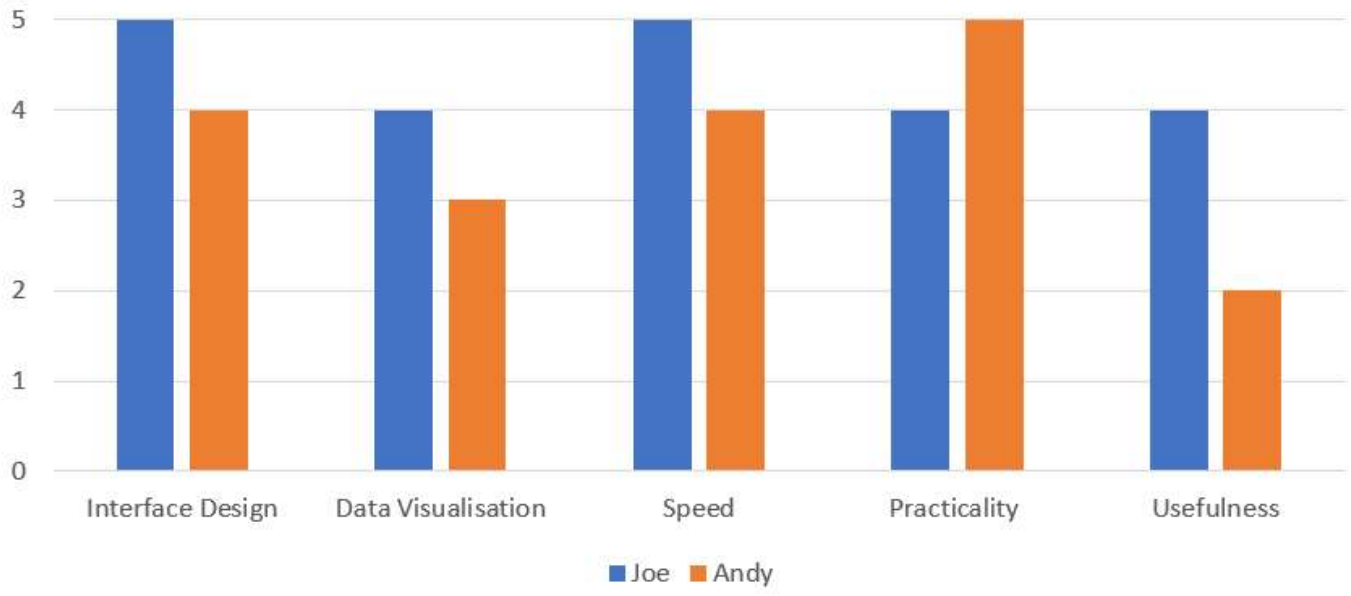


We can see that the search word must be a valid coin token whilst the language must suit the two character language format. If this is not the case error messages appear, and the search does not occur until the input is entered correctly.

Benjamin Holmes

As part of managing usability features laid out in the design stage, it was important to ensure that the system was effective, efficient, error tolerant, engaging and easy to learn/use software. This can now be demonstrated through the simplicity and robustness of the user interface. It does not allow invalid input data to be entered - through complete validation, no inputs will crash the system. The system is also efficient, with a new search query taking less than roughly 4 seconds - on average - to pull, sort and display the data. This further demonstrates the program's effectiveness. The system is also very intuitive, with only 5 input fields and one search button, it limits the amount of knowledge a user must know before performing a search.

In order to confirm these usability features' success, the stakeholders mentioned in the analysis stage can test and give feedback on the system. I gave the installation files, a simple explanation and a feedback questionnaire to each of the stakeholders . In order to get an objective assessment of how the users rated the system after the initial use, I asked them to rate the interface design, data visualization, speed, practicality and usefulness out of five (five being the highest). I then asked for additional feedback on any aspect of the system, and have summarized their thoughts below.



In the table below, Joe and Andy also provide additional feedback on desirable feature changes.

Joe	Andy

Benjamin Holmes

• Better way of finding trending coins • Suggestions generated from Twitter • More detail allowed in keyword input	• Better integration onto mobile display • Other methods of comparison included • Graphical time delay principles • Greater data set to allow further analysis of historical prices and polarity

The changes above would be the objectives for the second pass through the systems development life cycle.

Success of the solution

Overall this project has completed its aims and objectives to a high standard, to a point where the system is usable and could be used for its intended purpose. By following clearly the success criteria and also the various testing plans throughout the development, it allowed me to ensure each target was being met. Below is further analysis of how each point of the success criteria was achieved or worked towards.

Criteria	Extent of Criteria Met	Testing
1 Connect Python to Twitter API **1.1** Get access tokens and consumer keys **1.2** Install Libraries **1.3** Pull tweets	**1.1** Completely as developer account was created and the keys are set **1.2** Completely but could be extended with future development if new features are added **1.3** Completely as tweets are scraped as envisioned and are able to stored	**01**: Working and functional **02**: Working and functional **03**: Working and functional
2 Clean Data from Twitter **2.1** Remove URLS **2.2** Remove strange syntax	**2.1** Completely as there are no urls in the cleaned database **2.2** Completely, although extremely unique/new symbols may slip through this filter	**04**: Working and functional **05**: Working and functional
3 Calculate Sentiment Polarity	**3.1** Completely **3.2** Completely as polarity,	**06**: Working and functional **07**: Working and functional

3.1 Install Textblob **3.2** Analyze Tweets	subjectivity and analysis are collected from each tweet, this could be developed in further iterations into a more complex machine learning algorithm where test data is used	**08**: Working and functional
4 Sentiment Values **4.1** Average daily values **4.2** Store **4.3** Plot	**4.1** Completely although not used in the final version **4.2** Completely as data is stored securely in both CSV files and a permanent SQL file **4.3** Completely as data is plotted in numerous different ways that allow for easiest visualization	**09**: Working and functional
5 Get real crypto prices **5.1** Choose time frame **5.2** Choose coin **5.3** Calculate average of open, high, low and close **5.4** Plot	**5.1** Mostly as users can choose a time frame within the search feature, except this cannot be altered post search **5.2** Completely as search words or ticker tokens can be entered and they are validated to match a specific coin **5.3** Completely although not used in the final version **5.4** Completely as a graph is plotted that matches the polarity graph's timeframe	**10**: Working and functional
6 Compare Plots	**6** Completely although more types of comparison could be added to make the software more analytical	**15**: Working and functional
7 Create User Interface **7.1** Search feature **7.2** Trending feature **7.3** Design	**7.1** Completely as search words or ticker tokens can be entered and they are validated to match a specific coin **7.2** Mostly as the function works although not the extent where is is usable for real application, this could be	**11**: Working and functional **12**: Working and functional **13**: Working and functional **14**: Working and functional

	improved in further development **7.3** Completely as there is a aesthetic user interface that displays data presentably, design however can always be customized later dependent on developers preferences	

Describing the final product

The final product represents iterations of development and plenty of research into optimizing the system. Although not without its flaws, the system has progressed from the designs laid out in the design section and envisioned in the analysis section. Key differences between the design product and the final product is the emphasis on and implementation of more complex comparison algorithms (page 76), better user interface design (page 66) and also advanced database use (page 32). These developments came throughout this project in order to better support the success criteria. With more time the developments below could be implemented, whilst also carrying out code optimisation.

Maintenance and development

The largest limitation with the system is the rate limit determined by the Twitter API (as initially laid out on page 13). This literally limits the amount of data that can be scraped in a given time period, and means that data has to be continuously updated and collected to make a viable system. Through development I have also noticed irregularities in the twitter data, that whilst collection times are unchangeable through the API, sometimes data from specific times only are collected. This obviously reduces the accuracy of the system if we want whole day averages of data. Another drawback is the reduced development time spent on the trending feature, which stakeholders commented on. For this to work to the standard users would need for practicality, a greater range of coins and more detail on each must be produced as output.

Aside from ensuring libraries and code function is up-to-date, in terms of maintaining the system, the main area where continuous input is needed is scraping enough tweets regularly and adding them to the database in order to always have up to date data. This also allows for more historical analysis as the sample size increases. For the maintenance to occur, users would have to ensure that the system is used frequently.

Future variations could change the algorithm so that tweets are scraped more efficiently, whilst also being stored in a more computational way, this would speed up processing times. Along

with improving the trending section of the system, new criteria for interactive graphical display could be added. This would add more depth to the analysis done by users as data visuals could be altered without having to update the entire search. This links to the limitation above of the number of pull requests available. Another option further development could consider, is by passing the API entirely and using a cursor to manually scroll through Twitter and collect data. The reason this was not done in this project is due to the time required to develop a fast cursor to do this. Often these cursors are limited by Twitter's own user interface as it is not using Twitter back end to scrape data. Nonetheless, if this was modified in another development cycle it could produce a faster and more reliable system.

If stakeholders wish to add new features or changes this can be easily done as the program is heavily modular and the separate subroutines can be swapped out with the program still running smoothly. The important variables are global to the combination of programs, so any new features could either integrate this into a procedure, or, use local variables without interfering with other functions.

As I designed the software in 4 parts; the twitter scraper, the live crypto data, the SQL database and the user interface, if the stakeholders wanted a new design this could easily be integrated by swapping out features of unwanted sections that are combined into the main program. Also, as the code and development is mostly annotated, along with inputs and outputs being commented, a new developer could comfortably interact with the subroutines.