



CPSC 597 Project

Cryptocurrency Market Predictor

Project Advisor

Professor Lidia Morrison

Submitted by:

Brandon Michael Burdick

CWID: 889272894

Department of Computer Science and Engineering

California State University, Fullerton

May 17, 2022

Table of Contents

Abstract	3
1.0 Introduction	4
1.1 Background	4
1.2 Motivation	5
1.3 Related Work	6
1.3.1 Cryptocurrency Price Predictions Using Tweet Volumes and Sentiment Analysis	6
1.3.2 Stochastic Neural Networks for Cryptocurrency Price Prediction	7
1.4 Target Audience	7
2.0 Problem Statement	8
3.0 Features	9
3.1 Jupyter Notebook	11
3.2 Back-end API	12
3.3 React Front-end	15
4.0 Development Environment, Tools & Datasets	19
5.0 Implementation	22
5.1 Jupyter Notebook	22
5.1.1 Import Statements Cell & How to Run a Cell	23
5.1.2 Reading in Crypto Price Dataset	24
5.1.3 Searching for Tweets using Twint	26
5.1.4 sift_tweet and get_sentiment Functions	28
5.1.5 Data preparation and Preprocessing	29
5.1.6 Creating, training and testing the Model	33
5.1.7 Pull live data to make Prediction	35
5.2 Back-end API	41
5.2.1 Fast API Implementation	41
5.3 User Interface	43
5.3.1 React Imports & App declaration	44
5.3.2 Select Component	45
5.3.4 Button Component	46
5.3.5 Pie Chart Component	47
6.0 Conclusions and Future Work	48
References	50

Abstract

Cryptocurrencies have proven to be much more than just a buzzword since Satoshi Nakamoto introduced the concept for a decentralized currency in 2008. Investors, computer scientists, financial technology enthusiasts and everyday citizens have all participated in discussions regarding cryptocurrency and its viability as a potential asset for the future. However, the risk that comes along with investing funds in such a volatile market tends to inhibit everyday people from getting involved with this internet currency. This paper proposes the creation of a tool that can assist users in timing their investments to maximize their potential profits and minimize their losses through the use of a trained Machine Learning Model designed to predict the future trends of specified currencies. The paper starts by outlining the specifics of the underlying problem, the objectives of the project, goals for the project, and the timeline for accomplishing the creation of this tool.

1.0 Introduction

1.1 Background

For years, organizations and researchers have been utilizing Machine Learning Algorithms (ML) as a means to predict stock market trends. Companies like Vantage Point have built an entire business model around providing a service that they claim can predict market trends with “up to 87.4% accuracy,” (VantagePoint, 2021) as stated on their website. Their tool assists investors in making sound investments that anticipate market change through the use of Machine Learning. Thien Hai Nguyen, Kiyooki Shirai and Julien Velcin outlined their process of using sentiment analysis solely as a means to predict stock prices in their paper as well. While their findings and model were only able to predict trends with only 54.41% accuracy (Nguyen, T.H. et al., 2015), this paper assisted in opening up the idea of utilizing sentiment analysis as a means to predict stock market trends. Which is what Trang-Thi Ho and Yennuan Huang were able to do in their paper that proposed a method that utilized Convolutional Neural Networks and CandleStick charts to predict prices of high volume stocks in their paper in 2021. They were able to design a network that predicted the path of the Apple stock with an accuracy of 75.6% (Ho & Huang, 2021).

Since its inception in 2008, Satoshi Nakamoto's proposed decentralized currency has sparked controversy and inspiration among those in the fintech atmosphere (Nakamoto, S., 2008). Originally the bar to entry for cryptocurrency required users to understand the technological aspects of how a blockchain works, how to construct their own mining rigs for creating blocks and earning Bitcoin, and the public-key cryptography involved in owning and trading this online currency. However, cryptocurrency has become more easily available to the everyday person in recent years with the creation of exchange platforms like Coinbase, Binance

and Robinhood, where users can simply connect their payment information and purchase the cryptocurrencies that peak their interest. It is no longer a requirement or risk to be technologically savvy enough to acquire your own software/hardware wallet and purchase these currencies from other cryptocurrency owners through the use of untrustworthy outlets like Craigslist forums and local meetups to trade crypto (unless you mined it yourself). Although, with this increased accessibility comes an increased amount of risk, as many users might just be buying into crypto projects that they've only heard about in passing and neglected to research themselves.

1.2 Motivation

My motivation for working on this project comes from my passion for studying the fields of Blockchain Technology and Machine Learning and my interests in the Stock and Cryptocurrency markets. Since its inception I believe the blockchain data structure has proven its versatility and viability in the technologic space. Because of it, cryptocurrencies have given people a platform where they can exchange their currencies in a cryptographically secure and time-efficient manner across the globe. Due to these advantages, cryptocurrency has seen tremendous growth in recent years. We are seeing a paradigm shift from what was once perceived as something that could never work, to now having trillions of dollars invested into it, with thousands of projects to speak for its viability (Fang et. al., 2020). However, just because the accessibility has increased doesn't necessarily mean everyone wants to get involved. When you purchase cryptocurrencies, you're still risking your investment due to the volatility of the currency you're investing into. So my goal is to provide investors interested in cryptocurrency with a tool to help reassure them with probabilistic significance when it would be a good or bad

time to invest, based on a ML model's prediction, similar to the way this is done in the Stock Market atmosphere.

1.3 Related Work

1.3.1 Cryptocurrency Price Predictions Using Tweet Volumes and Sentiment Analysis

This paper outlines the researchers process of utilizing Twitter and Google trend volumes to train a machine learning model for predicting the prices of cryptocurrencies (mainly Ethereum and Bitcoin). In their paper they discuss previous research on the topic, and highlight research done by psychological researchers regarding how “decisions are influenced by emotions” (Abraham et al., 2018) and how social media can showcase both people's emotions and their decisions. Leading to the conclusion that they could use the sentiment scanned from social media posts to determine the sentiment people have towards various cryptocurrencies. They discuss and outline how they acquired the tweets using Tweepy (twitter python tool), clean those tweets and format them into something that can be used for making predictions. They also collected google trends data to acquire the frequency that cryptocurrencies are queried in any 90 day increment, so they can determine whether or not there was a correlation between the sentiment analysis recorded from twitter, and the sheer amount of times those topics were being googled on a global scale. Jethin Abraham and his colleagues concluded that when prices lowered twitter sentiments weren't a reliable indicator, but both google trends and tweet volume were heavily correlated with price. Their research helps pave the way to determining whether or not there exists a correlation between social media sentiment and the rise and fall of cryptocurrency prices, which they concluded does indeed exist.

1.3.2 Stochastic Neural Networks for Cryptocurrency Price Prediction

In this paper, the researchers outline previous research efforts on the topic, then the concept of Stochasticity and how it applies to the financial market, then outlines the details of the models they trained for predicting the Cryptocurrency prices and their overall conclusions. They start off by describing how Regression has an accuracy of 66-77% when observing tweet volume and google trends and attempting to make a prediction solely off of that. They move into discussing how a Multilayer Perceptron has been used to predict either an impending upward or downward trend with an accuracy of 64%, followed by RNN implementations by two researchers who only achieved an accuracy of 63% on Bitcoin predictions, and 56% on Ethereum. At the end they outline their Stochastic Neural Network's ability to make predictions on the trends of the cryptocurrency market, and the average relative improvement over other Neural Networks to be an increase of about 1.56% (P. Jay et al., 2020).

1.4 Target Audience

The ideal target audience for this application are mainly two sets of individuals: those who want to get involved in the investing in the cryptocurrency atmosphere, and those fascinated by the world of machine learning. The hope is that this application can provide some form of assistance to those who might want to time their investments well, as well as those who would like to view the code and contribute to it, as it will be open source on Github, and documented in a jupyter notebook.

For those interested in investing in cryptocurrency, the goal of this application is to assist them in validating their investments with probabilistic reassurance. Many people feel as though they lack the investing experience and technical knowledge to assure themselves they're placing their money in a sound investment, especially in something as complicated as cryptocurrency. It

goes without saying that the conceptual understanding of what a blockchain is, how it functions, and how it even guarantees cryptographic security of your funds is definitely not easy to understand. Even with a technical background, the core concepts of how this decentralized data structure works are hard to grasp for most. Since this is the case, it defeats one of the main purposes of cryptocurrency that Nakamoto outlines in his paper proposing a decentralized currency, since a main goal was to put the power of financial control into the hands of the people (Nakamoto, 2008). We need more tools assisting the general public in feeling comfortable with becoming involved in the world of digital currency in order for this decentralized system to thrive, and I hope this tool can assist in providing that.

For those interested in the Machine Learning aspect of the project all of the code will be publicly available on my Github. There exists a Jupyter notebook for the purposes of the ML aspects of this project, that outlines the retrieval/creation, cleaning and organizing of the price dataset & live tweets pulled, as well as the training and use of the model on the live price & tweet data. The hope for making this project open-source is that other like-minded individuals who wish to assist in the improvement of this project can do so by simply viewing the code and providing insight to other potential implementations and findings of their own using my project as a baseline/starting point for their own purpose.

2.0 Problem Statement

In F. Fang et al.'s comprehensive survey of cryptocurrency trading, they outline the exponential increase in cryptocurrency market cap, where on December 20, 2019 the market cap was around 190 billion U.S. dollars, and by April 2021 it had grown to over 2 Trillion U.S. dollars (Fang et al., 2022). A large reason for this increase in market cap during this time period

is due to new investors becoming involved in the cryptocurrency atmosphere. However, these new investors invested their funds into cryptocurrencies without knowing exactly whether or not their investment is a safe bet. It's extremely difficult to predict the future of a currency based on its current standing, with or without an in-depth knowledge of the market.

With that being said, there is definitely a level of strategy and luck that goes into timing investments when it comes to cryptocurrency, like the stock market. Cryptocurrency maintains a reputation of being an extremely volatile intangible asset, where the price can fluctuate greatly. Many people have attempted to predict when markets are about to boom or crash, and attempt to time their investments to maximize profits similar to that of investing into businesses through publicly traded stocks. However, due to human error and our inability to process and assess large amounts of data due to our inability to accurately assess probability when calculating risk, we can turn to technology to attempt to make our predictions for us. But unlike stocks, if someone were to invest into crypto projects that end up plummeting in price, overall it hurts the chances of people being more I believe we can utilize similar machine learning methods to train a model so it can make price predictions about cryptocurrencies with much better accuracy than human investors.

The goal of this project is to utilize Machine Learning to create a model that can assist users in timing their investments, by analyzing previous market trends and user sentiment on social media to predict whether a cryptocurrency will increase or decrease in value over a given time-frame.

3.0 Features

BEFORE YOU BEGIN:

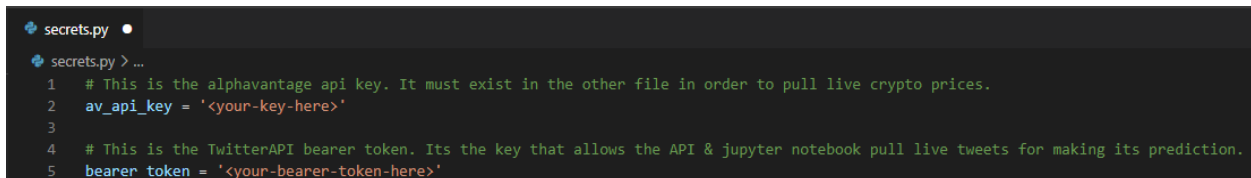
It is *imperative* that you verify you either have access to the **secrets.py** file submitted with the project, or you have created your own **secrets.py** file and stored the necessary API keys inside of it as these keys are **required** to run the project. If you do not have the provided file, then feel free to use the following provided guides for generating each respective key necessary for running the project:

- [Guide for Twitter API Key/Bearer Token](#)
- [Location for generating AlphaVantage API Key](#)

Once you have access to your bearer token and AV API key, clone/download the repository to your local and create a “secrets.py” file with the following two declared variables inside of it:

- **av_api_key** = “your-api-key-here”
- **bearer_token** = “your-bearer-token-here”

The variables in this file **must** be strings, and they **must** be named as specified above (screenshot below as an example).

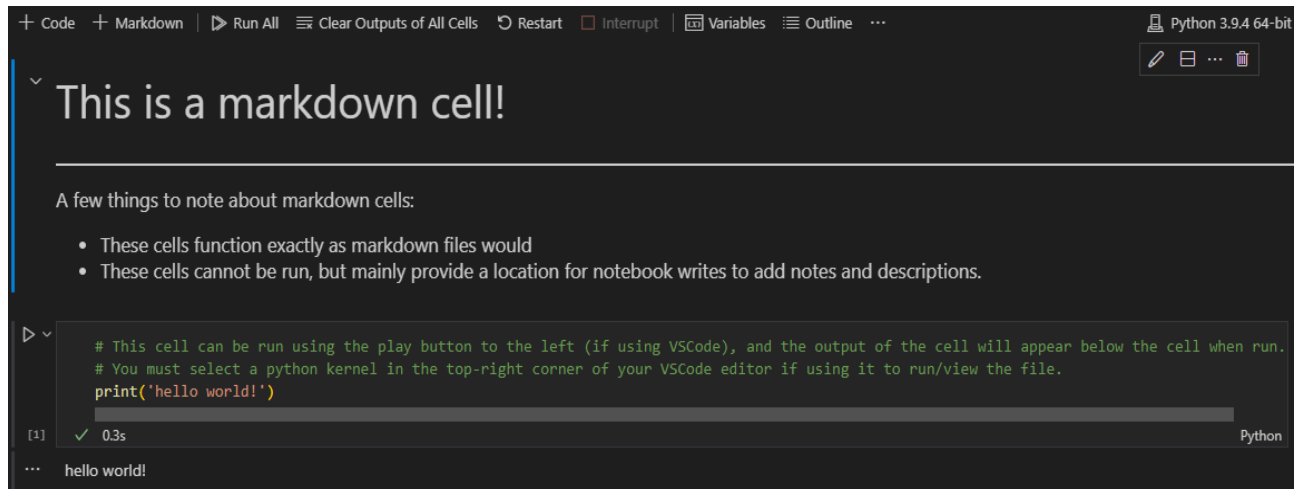


```
secrets.py
1 # This is the alphavantage api key. It must exist in the other file in order to pull live crypto prices.
2 av_api_key = '<your-key-here>'
3
4 # This is the TwitterAPI bearer token. Its the key that allows the API & jupyter notebook pull live tweets for making its prediction.
5 bearer_token = '<your-bearer-token-here>'
```

Additionally, there is a “requirements.txt” file inside the Github repository and project submission that provides all the proper python libraries and versions needed to run the project. I **strongly** recommend [creating a python virtual environment](#) and importing this file to install all dependencies required for this project.

3.1 Jupyter Notebook

In order to to run any cells, you have to click the play symbol (▶) in the top left corner. Also, unless otherwise specified, you must run each cell sequentially in the Jupyter notebook to properly run the project. Some cells are dependent on the functions/variables created in previous cells in order to work properly. Additionally, when the user has successfully run any cell, you can see the runtime for the cell and the green checkmark (✓) in the bottom left corner of the cell, signifying that everything ran smoothly and there are no errors to report. The screenshot below shows an example markdown cell & python cell for reference:



The jupyter notebook provided with this project submission is a feature mainly intended for the use of graders, researchers and other machine learning/cryptocurrency enthusiasts. The notebook provides a baseline for my approach to solving the problem by applying Linear Discriminant Analysis to the dataset created in an attempt to classify an hour worth of cryptocurrency data as increasing/decreasing.

3.2 Back-end API

The API is a feature that can be used to quickly pull predictions using a tool like Postman or cURL in the case that the user doesn't want to use the front-end application provided, or progress through the entirety of the Jupyter notebook. The API is also available for users to connect to their own front-end application if they so choose, and provides an exposed endpoint that they can utilize themselves. The code that is used in the API for the purposes of making the prediction is taken directly from the Jupyter notebook, so if anyone who wants to utilize this API for their own front-end application or add additional endpoints to it is interested in how that section works and wants to make modifications to it, they can refer to the Jupyter notebook documentation or the documentation in the Implementation section of this document.

The API can be run with the uvicorn ASGI web server implementation. The process of installing uvicorn is described [here](#). Downloading the tools needed for using the API is as simple as running two separate “pip install” commands (assuming the user has the [pip package installer](#) on their machine):

- **pip install fastapi** - installs fastapi library
- **pip install “uvicorn[standard]”** - installs uvicorn to function as server for API

Once all of that is installed/configured, users must simply run the following command in their terminal:

```
python -m uvicorn CryptoPredictonAPI:app
```

and they will see the following output:

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER

Microsoft Windows [Version 10.0.19044.1645]
(c) Microsoft Corporation. All rights reserved.

C:\Users\WaKaBurd\Documents\GitHub\CryptoPredictionTool>python -m uvicorn CryptoPredictionAPI:app
INFO:      Started server process [18444]
INFO:      Waiting for application startup.
INFO:      Application startup complete.
INFO:      Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)

```

Now the only endpoint that's exposed for this service is located at:

“`http://127.0.0.1:8000/prediction_generator/<coin_name>`”, where users must use either a command line tool like cURL or an application like Postman to hit it and receive the output from the prediction on the desired coin. As an example, here's how one would hit the URL and the expected response when using the Postman Application:

The screenshot shows the Postman application interface. At the top, a GET request is configured to `http://127.0.0.1:8000/prediction_generator/BCH`. Below the request bar, the 'Query Params' section is empty. The 'Body' tab is selected, showing a JSON response. The status bar at the bottom indicates a 200 OK response with a 990 ms response time and 181 B of data.

KEY	VALUE	DESCRIPTION
Key	Value	Description

```

1  {"1": 0.1501188504863964, "2": 0.8498811495136036}

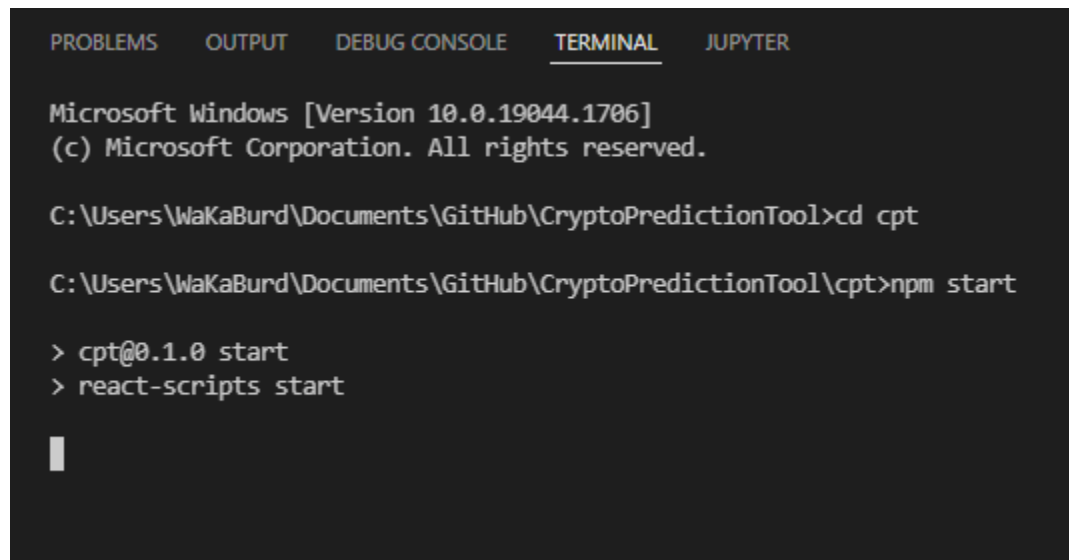
```

The user of the postman application is sending a GET request to the prediction_generator endpoint at port 8000, searching for the specified coin “BCH” in this example. Note the response body shows a 200 OK message, and it responds with 2 values in JSON format. The first value “0.150..” assumes the probability of an increase in price over the next hour, while the second value “0.849..” indicates the probability of a decrease in price over the next hour.

3.3 React Front-end

The front-end itself is provided for users/crypto enthusiasts who might not be interested in the development aspects of this project, and instead are solely interested in seeing what kind of predictions the predictor is capable of making. In order to use the predictor however you must have the node package manager tool installed on your machine, and have finished configuring/running the back-end API for this project (see above section for instructions).

The front-end provided uses the javascript React framework, and is located inside the “cpt” directory in the public Github repository/project submission. In order to run the React application, one must simply download the project to their local machine and must first have the back-end API running in a separate terminal. Once those two tasks are complete, they must open an additional terminal, swap directories to go into the “cpt” directory of the project, and run the “**npm start**” command. If all works properly, the user should see the following output:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER

Microsoft Windows [Version 10.0.19044.1706]
(c) Microsoft Corporation. All rights reserved.

C:\Users\WaKaBurd\Documents\Github\CryptoPredictionTool>cd cpt

C:\Users\WaKaBurd\Documents\Github\CryptoPredictionTool\cpt>npm start

> cpt@0.1.0 start
> react-scripts start

█
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER

Compiled successfully!

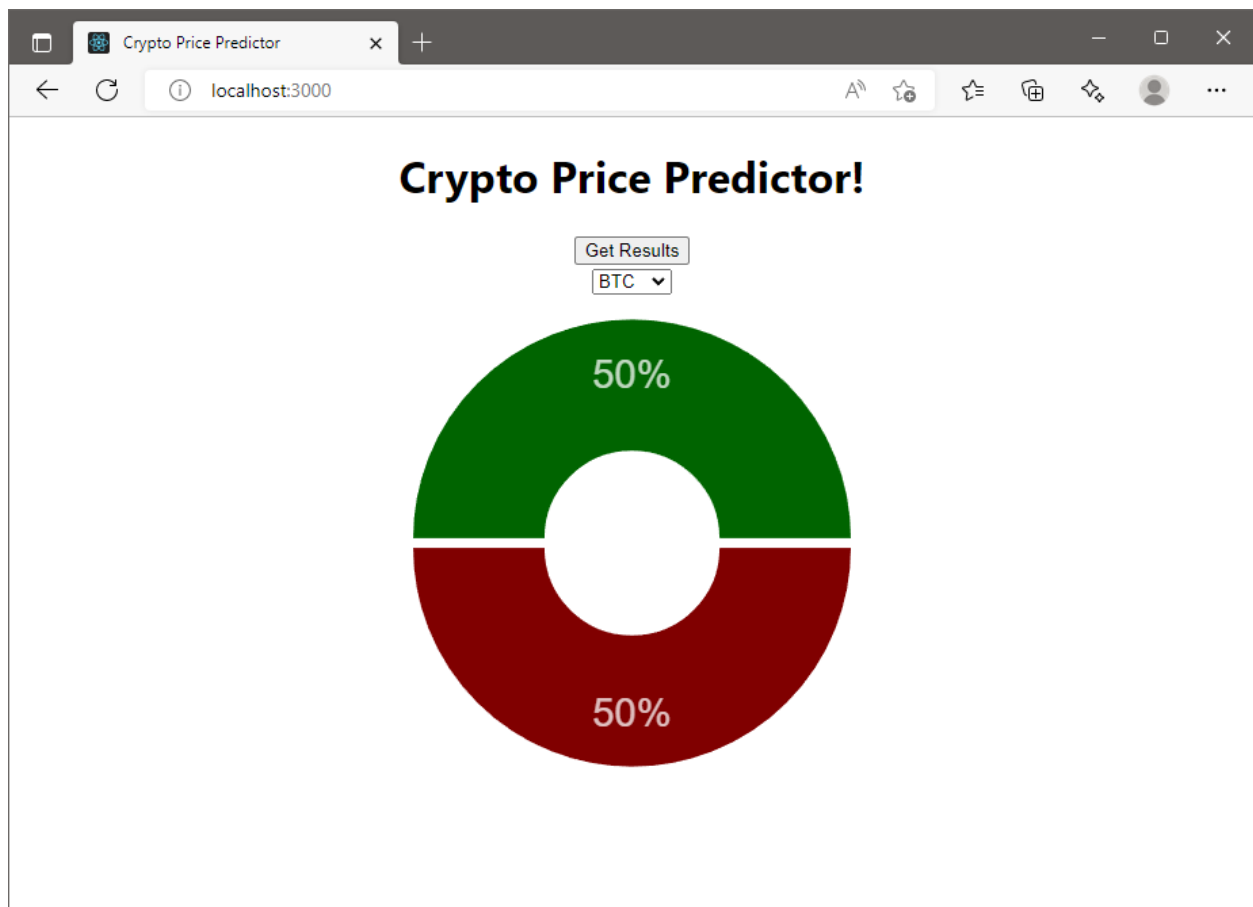
You can now view cpt in the browser.

Local:      http://localhost:3000
On Your Network: http://192.168.1.9:3000

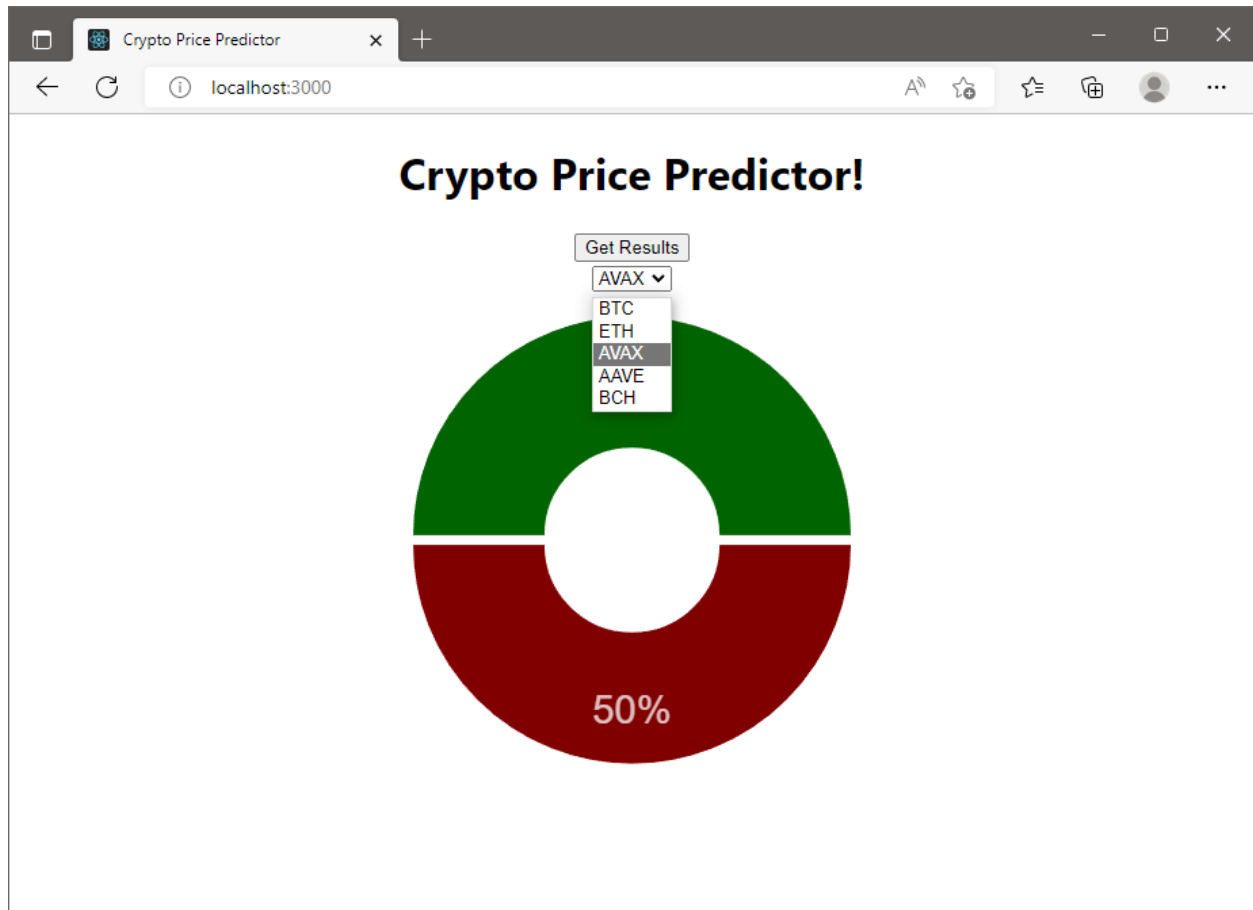
Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

From there, a new web browser should be opened automatically that looks like the following:



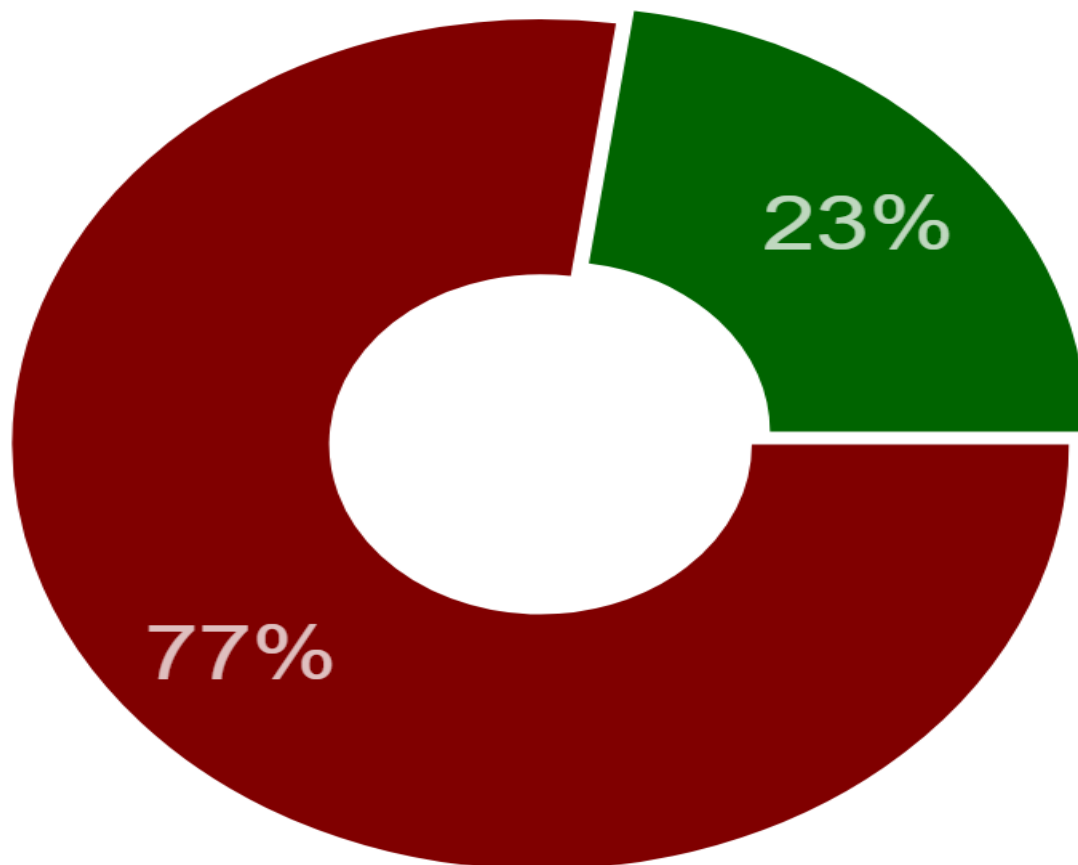
From there, the user can begin interacting with the application by clicking on the dropdown table and selecting one of the 5 available cryptocurrencies the front-end offers.



Once the user has made their selection, all they have to do is click the “Get Results” button and wait approximately 2-5 seconds while the application requests a prediction value. When the prediction is received by the application, it will be reflected through the pie chart, showing the probabilities of an increase/decrease in price in the next hour based on the model’s prediction.

Crypto Price Predictor!

Get Results
AVAX ▾



4.0 Development Environment, Tools & Datasets

Visual Studio Code:

The entirety of this project was developed using the visual studio code text editor and many of its community/developer-offered extensions.

Python Programming Language:

This project uses python version 3.9.4

ReactJS:

ReactJS v18.1.0 was used for the creation of this project

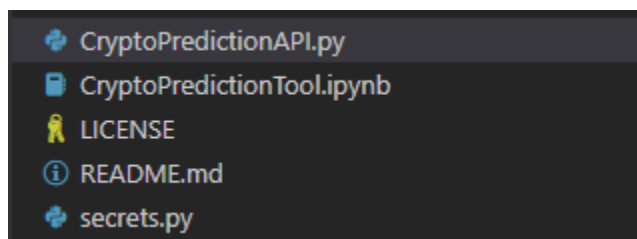
FastAPI:

The back-end API created for this project utilized the python [FastAPI library](#).

AlphaVantage API:

For pulling live cryptocurrency price data the [AlphaVantage API](#) was used.

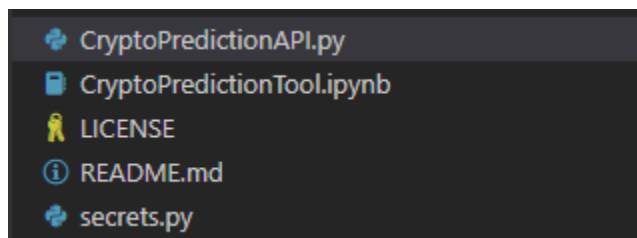
NOTE: If you plan on running this project you *must* generate your own free AlphaVantageAPI key (see “How to Use” section of this document for instructions) **OR** use the API keys provided in the “secrets.py” file provided in the project submission. The file must be in the same directory as both the API & Jupyter Notebook to properly work (screenshot below as an example).



Twint & Twitter Developer API:

The Twint library is a free-to-use python library that has zero limitations on the number of tweets users can pull, and was used in this project to extract old tweets by the thousands for the purpose of training the model. This tool outperforms the Twitter API because of its ability to pull tweets with little limitations where the Twitter Developer API falls short. The Twitter Developer API is used in this project to pull live tweets for making our predictions, as it excels in this regard. While we can only pull 150 tweets every 15 minutes, the granularity offered by this tool far surpasses the twint tool, making it great for our needs.

NOTE: If you plan on running this project you *must* generate your own free Twitter API key & bearer token(see “How to Use” section of this document for instructions) **OR** use the API keys provided in the “secrets.py” file provided in the project submission. The file must be in the same directory as both the API & Jupyter Notebook to properly work (screenshot below as an example).



Textblob & VADER Sentiment Analysis Tool:

Textblob is a sentiment analysis tool that can quickly and accurately determine the sentiment of plain-text sentences (polarity and subjectivity). The VADER Sentiment Analyzer also provides sentiment analysis of tweets, but is designed to understand the sentiment incorporated into slang & emojis that are commonly used on the internet.

Pandas & SciKit Learn libraries:

The Pandas library provides a powerful tool for the purposes of data manipulation and analysis. Using the data frames provided by pandas and tools provided there-in allow me to adeptly construct and organize the datasets required for making predictions. The SciKit library provides tools for the purpose of data analysis and machine learning, making the creation of models used in this project significantly easier and more streamlined.

Jupyter Notebook:

The Jupyter Notebook tool was used to create the majority of the project, and offers a plethora of documentation on how to interact with the project, the purposes of each cell, and a location for other developers to view how the data is retrieved, formatted and fed into the model.

GitHub:

GitHub was used to maintain version and source control for this project, and for organizing out the tickets required for completing the application. The entirety of the project is available on my personal Github account under the repository titled [CryptoPredictionTool](#).

Bitstamp price Datasets:

The price datasets offered on the [Bitstamp website](#) assisted with the completion of this project as well. They have hourly and daily breakdowns of cryptocurrency prices available for download in a CSV format (this project uses the hourly datasets).

5.0 Implementation

5.1 Jupyter Notebook

The majority of the development for this project was done using the Python programming language in a Jupyter Notebook. Jupyter Notebook is a web application that allows developers to easily create and share computational documents, which makes it ideal for the purpose of creating machine learning projects and applications. The notebook consists of code cells and markdown cells, allowing users to run smaller portions of code and viewing output while writing organized, formatted notes and explanations about their code directly into the document using Markdown. Each of these cells compartmentalizes a different portion of the project, whether it be reading in the data, fetching tweets from Twitter (using either Twint or the TwitterAPI), pulling live Crypto prices, or organizing all of the required data into a properly formatted Pandas dataframe for the purpose of training the model to make predictions. For the remainder of this section, I'll be walking through each Jupyter notebook cell for documentation purposes, and outlining the expected output.

5.1.1 Import Statements Cell & How to Run a Cell

Import Statements

```
import pandas as pd                # Pandas dataframe library
import pandas_datareader as pdr    # Pandas datareader that allows me to lookup & store live crypto prices from yahoo finance.
import numpy as np                 # Numpy
import matplotlib.pyplot as plt    # Pyplot used to create visuals/graphics based on data
from alpha_vantage.timeseries import TimeSeries # Library used for pulling live price data from alphavantage api

from datetime import datetime, timedelta, timezone # Datetime library.
import warnings
warnings.simplefilter(action='ignore', category=ResourceWarning) # Suppresses warnings to limit size of output cells.
warnings.filterwarnings('ignore')

import glob                        # For changing/finding proper directory
import os                         # For changing/finding proper directory (when opening files)
import requests                   # For sending HTTP requests in order to hit necessary API endpoints.
import twint                      # Twitter web scraping tool with more features than the regular twitter API
import nest_asyncio               # Import required for twint usage, allows for the use of asynchronous computing
nest_asyncio.apply()

import re                         # Regex for string cleaning (used for Textblob Sentiment Analysis)
from textblob import TextBlob     # Textblob used for sentiment analysis of cleaned data.
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer # Sentiment analysis tool that works great on determining social media sentiment.

from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split # Used for splitting data
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis # Used for implementing LDA

os.chdir(r'C:\Users\WaKaBurd\Documents\GitHub\CryptoPredictionTool\archive') # Directory where the stopwords.txt file is located.
stopwords_file = open("stopwords.txt", "r+") # This file is used for sifting the tweets fetched before feeding them into Textblob for Sentiment Analysis
stopwords = list(stopwords_file.read().split('\n'))
```

[4] ✓ 0.4s

Python

For the import cell specifically, the purpose of this cell is simple and straightforward; import all required libraries & documents. Each import statement has its purpose for being imported listed at the end of the line in the form of a comment. Some of these imports will require the user to have PIP installed on their machine (pip installation guide can be found [here](#)), and they'll need to run a “pip install <package>” command in order to verify they have the proper libraries installed in their environment. As an example, for installing the “pandas” library in your python environment, you'll need to run the command: “pip install pandas” in your environment, and your pandas library should be installed as needed and good to go.

5.1.2 Reading in Crypto Price Dataset

Reading in crypto price dataset

Section below reads csv files into pandas dataframes for interacting with. Also compiles list of coin names for twitter searching.

```
path = r'c:\Users\WaKaBurd\Documents\GitHub\CryptoPredictionTool\prices\HourlyPrices'
extension = 'csv'
os.chdir(path)
hourly_csv_files = glob.glob('*.*'.format(extension))

# Compile list of all coin names for searching on twitter later
hourly_coins = []

for coin in hourly_csv_files:
    vals = coin.split("_")
    coin_name = vals[0]
    hourly_coins.append(coin_name)

# compile list of pandas dataframes for use later.
hourly_coin_data = []

for file in hourly_csv_files:
    df = pd.read_csv(file)
    hourly_coin_data.append(df)
```

[3] ✓ 0.6s

This cell navigates to the specified path where the user has stored their price dataset for the hourly prices read in from the HourlyPrices directory in the project repository, and from there compiles a list of the coins we have data for (based off of the name of the file) and compiles the csv files into our hourly_coin_data list comprised of 1 pandas df per CSV file.

Also, to give you an idea of what the CSV file contains, the below screenshot showcases the columns and values stored in each of these tables. The main columns that are used for this project are the date, open, high, low, close, & both volume columns for the purpose of training the model. It's also important to notice that the date column has the datetimes split by hour for the purposes of my project, as I'll be building my model and making my predictions based on one hour of data (both tweets and financial data).

Just to give you an idea of what this looks like...

```
hourly_coin_data[0].head()
```

[5] ✓ 0.7s

	unix	date	symbol	open	high	low	close	Volume AAVE	Volume USD
0	1650931200	2022-04-26 00:00:00	AAVE/USD	169.23	170.68	168.38	169.48	174.216470	29526.207266
1	1650927600	2022-04-25 23:00:00	AAVE/USD	169.68	170.57	169.68	170.10	1.814157	308.588094
2	1650924000	2022-04-25 22:00:00	AAVE/USD	168.21	169.51	168.21	169.51	0.383062	64.932829
3	1650920400	2022-04-25 21:00:00	AAVE/USD	168.46	168.46	168.46	168.46	3.677922	619.582701
4	1650916800	2022-04-25 20:00:00	AAVE/USD	169.83	169.90	169.83	169.90	11.501266	1954.065088

5.1.3 Searching for Tweets using Twint

This cell consists of a single “search_coins” function, which is expecting a list as a function argument that should consist of all the names of coins the user wants to search. The function navigates to the proper directory to store the search results, creates/enters the directory with the same name as the coin to be searched using the Twint tool. In order to use Twint, a configuration needs to be created and ran to pull the tweets I need from their database. The configuration I used consists of the following parameters and values:

- **Limit** (number of tweets to be retrieved): 100 (for 100 tweets)
- **Lang** (language you’d like the tweets to be in): “en(lish)”
- **Pandas** (enables the storing of tweets in a pandas dataframe): true
- **Search** (value we’re searching for): *coin* (e.g. BTC, ETC, AVAX, etc.)
- **Since** (search start date): *from_date* (start date for our search)
- **Until** (search end date): *to_date* (end date for our search)
- **Output** (output file location): *<coin>_<from_date>_<to_date>_search_result.csv*

Scrape Twitter for data on all coins supplied by dataset

Below section of code searches through Twint tweet database for any tweets associated with the each of the provided cryptocurrency acronyms.

```
# Function for iterating through coins list and storing findings in .csv files
def search_coins(coins):

    for coin in coins:
        path = r'c:\Users\WaKaBurd\Documents\GitHub\CryptoPredictionTool\search_results'
        os.chdir(path)
        os.chdir(coin)

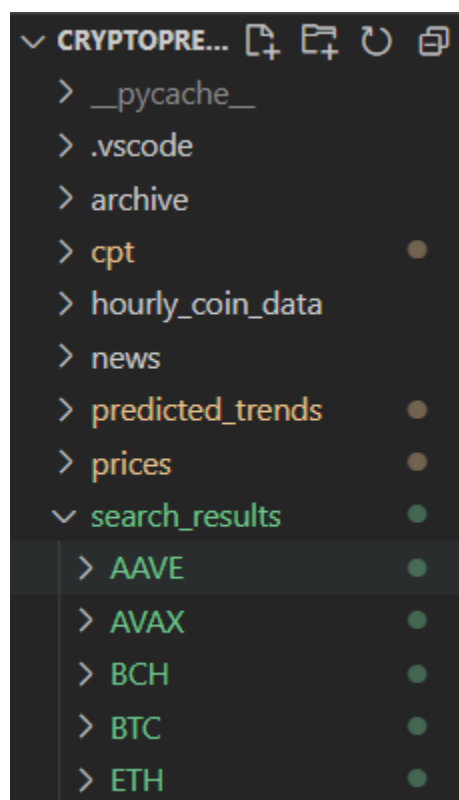
        print('performing twitter search for coin:', coin)

        from_date = '2022-04-27'
        to_date = '2022-04-28'
        print(f'searching {from_date} to {to_date}')

        c = twint.Config()
        c.Limit = 100
        c.Lang = "en"
        c.Pandas = True
        c.Search = coin
        c.Hide_output = True
        c.Since = from_date
        c.Until = to_date
        c.Store_csv = True
        c.Output = coin + '_' + from_date + '_' + to_date + '_search_result.csv'
        twint.run.Search(c)
    search_coins(hourly_coins)
```

The screenshot below on the left-hand side showcases the output for the cell, signifying that it was able to properly fetch tweets for the requested date. The screenshot below on the right-hand side shows the 5 directories that have been modified in the “search_results” folder, allowing users to view the tweets in the CSV format if they so choose.

```
[8] ✓ 13.4s
... performing twitter search for coin: AAVE
searching 2022-05-09 to 2022-05-10
performing twitter search for coin: AVAX
searching 2022-05-09 to 2022-05-10
performing twitter search for coin: BCH
searching 2022-05-09 to 2022-05-10
performing twitter search for coin: BTC
searching 2022-05-09 to 2022-05-10
performing twitter search for coin: ETH
searching 2022-05-09 to 2022-05-10
```



5.1.4 sift_tweet and get_sentiment Functions

```

# Need to create function for cleaning the tweets so we can derive the subjectivity and polarity using textblob.
def sift_tweet(tweet, stop_words):
    cleaned_tweet = tweet
    cleaned_tweet = re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\w+)", "", tweet) # regex to remove all @username, emojis, and links from tweets.
    for word in cleaned_tweet:
        if word in stop_words: cleaned_tweet.replace(word, '')
    return cleaned_tweet

# Function for allowing me to generate the sentiment intensity of the text passed in.
def get_sentiment(text):
    sia = SentimentIntensityAnalyzer()
    sentiment = sia.polarity_scores(text)
    return sentiment

```

[14]

The “sift_tweet” function used to clean tweets and remove all special characters, links, emojis and hashtags. After that it iterates through each remaining word and removes words that exist in the stopwords list (read in from stopwords.txt file earlier on) to remove all words that do not carry any sentiment value. This is so the Textblob tool can properly assess the polarity and subjectivity of the tweets that are fetched and stored. The “get_sentiment” function takes in a tweet as is, and runs the VADER SentimentIntensityAnalyzer on the tweet to determine the sentiment of a tweet. The great thing about this tool is that it’s configured to understand and assign sentiment values to text that is commonly posted on the internet (such as hashtags, emojis and other slang), making it an ideal tool for analyzing the sentiment of tweets.

5.1.5 Data preparation and Preprocessing

```
os.chdir(r'C:\Users\WaKaBurd\Documents\GitHub\CryptoPredictionTool\search_results')
tweet_dfs = []
grouped_tweets = []

# Read Tweets into a DF from the CSVs
for coin in hourly_coins:

    os.chdir(r'C:\Users\WaKaBurd\Documents\GitHub\CryptoPredictionTool\search_results')
    os.chdir(coin)
    csv_names = glob.glob('*.{format(extension)}')
    coin_pds = []
    for file in csv_names:
        tweet_pd = pd.read_csv(file)
        tweet_pd.sort_values(by='date')
        coin_pds.append(tweet_pd)
    tweet_dfs.append(coin_pds)
```

The next few screenshots outline the implementation for data preparation and preprocessing before feeding it into the model for training and predicting values. The start of the cell reads all of the CSV files into dataframes, sorts them by date and appends them to a list so we can use them for preprocessing.

```
# for i in range(len(tweet_dfs)):
# This is just so i can the data i need to train a model for aave and avax. I'll do all 5 when i want to showcase something but for now i c
for i in range(1):
    i = 0
    print('lookin at coin number:', i)
    hourly_coin_data[i]['date'] = pd.to_datetime(hourly_coin_data[i]['date'])
    hourly_coin_data[i]['joined_tweets'] = ""
    hourly_coin_data[i]['compound'] = 0.0
    hourly_coin_data[i]['positive'] = 0.0
    hourly_coin_data[i]['negative'] = 0.0
    hourly_coin_data[i]['neutral'] = 0.0

    #print(hourly_coin_data[i])
    for j in range(len(tweet_dfs[i])):
        tweet_dfs[i][j]['created_at'] = pd.to_datetime(tweet_dfs[i][j]['created_at'])

        for day in range(1,31):
            #print('checking day:', day)
            for hour in range(24):
                tweet_time_mask = (tweet_dfs[i][j]['created_at'].dt.hour >= hour) & (tweet_dfs[i][j]['created_at'].dt.hour < hour + 1) & \
                    (tweet_dfs[i][j]['created_at'].dt.day >= day) & (tweet_dfs[i][j]['created_at'].dt.day < day + 1)
                price_time_mask = (hourly_coin_data[i]['date'].dt.hour >= hour) & (hourly_coin_data[i]['date'].dt.hour < hour + 1) & \
                    (hourly_coin_data[i]['date'].dt.day >= day) & (hourly_coin_data[i]['date'].dt.day < day + 1)

                hour_view = tweet_dfs[i][j][tweet_time_mask]
                if hour_view.empty:
                    continue

                hour_view['cleaned_tweet'] = hour_view['tweet'].apply(lambda x: sift_tweet(str(x).lower(), stopwords))

                joined_tweets = ' '.join(hour_view['tweet'])
                joined_clean_tweets = ' '.join(hour_view['cleaned_tweet'])

                SIA = get_sentiment(joined_tweets)
                compound = SIA['compound'] # Score representing sum(lexicon ratings)
                pos = SIA['pos']
                neg = SIA['neg']
                neu = SIA['neu']
```

From there, I append empty cells for each of the various columns I'll be requiring in my finalized dataframe that will be used for training my model. The date column needs to be converted to a pandas datetime for the purpose of filtering it into hourly gaps using a time mask, so we can only view tweets from a single hour and associate that hour with the hourly price data as that's how it is split. The "joined_tweets" column is for us to essentially combine all of the tweets into a single cell that can then be fed into the Textblob and VADER Sentiment Analyzer for analysis. The compound, positive, negative, and neutral values are the result of the VADER Sentiment Analyzer, where each value provides a different type of sentiment assessment as follows:

- **compound**: sum of pos, neg and neutral score that is then normalized between (-1..1)
- **pos**: analyzes the sentiment of the text and determines how much of it was positive
- **neg**: analyzes the sentiment of the text and determines how much of it was negative
- **neu**: analyzes the sentiment of the text and determines how much of it was neutral

Past that, the nested for loop iterates through each day of the month and each hour of the day and scans for all tweets that exist in our data set within that time frame. If the tweets exist we can gather sentiment values for them, if not we continue through the loop, appending to our data frame of hourly coin data.

Dropping the rows that don't contain a polarity score. The only reason they wouldn't have this would be because they didn't have any tweets stored in their row for that hour.

```
[18] hourly_coin_data[0] = hourly_coin_data[0][hourly_coin_data[0]['polarity'].notna()] Python
```

Now, I have to iterate through remaining rows and append a price change label. This label signifies whether or not in that hour the price of the coin went up or down. This is what the model is going to be responsible for predicting.

```
[19] for i in range(len(hourly_coin_data)):
    hourly_coin_data[i].reset_index()
    hourly_coin_data[i]['price_change'] = np.nan
    for index, row in hourly_coin_data[i].iterrows():
        if row.open > row.close:
            hourly_coin_data[i].at[index, 'price_change'] = 0
        else:
            hourly_coin_data[i].at[index, 'price_change'] = 1 Python
```

From there, now that we have all of our tweets analyzed and grouped into their respective hours, we need to drop the rows that didn't have tweets from that time frame, and then assign a label to the remaining rows under the newly added “price_change” column. This column represents whether or not the price of the coin increased during that hour. This is crucial to record because this is the label my LDA classifier will be responsible for predicting, where 1 corresponds to an increase in price during that hour, and 0 corresponds to a decrease in price over that hour.

hourly_coin_data[0].head()

[27] Python

...	Unnamed: 0	open	high	low	Volume USD	compound	positive	negative	neutral	polarity	subjectivity	price_change
0	32	169.31	169.78	166.84	4483.910281	0.9920	0.107	0.068	0.825	-0.067892	0.365060	1.0
1	33	170.40	170.70	168.83	3966.333567	0.9164	0.098	0.080	0.821	0.018946	0.396833	0.0
2	34	169.27	170.87	167.84	14345.829974	0.9834	0.069	0.034	0.898	0.101900	0.487789	1.0
3	35	171.35	171.35	171.35	0.000000	-0.9548	0.080	0.084	0.836	0.027616	0.444126	1.0
4	36	171.56	172.03	171.35	2540.947375	0.9974	0.110	0.058	0.833	0.016304	0.485740	0.0

Now that we have all the price data and its corresponding sentiment data stored in a single, combined dataframe with a label to represent the price change during that hour, I can move into training the model to make its classification using that data (NOTE: the unnamed:0 column is the old indices from its previous table, this column will be dropped before being used for training).

5.1.6 Creating, training and testing the Model

```

# These are all the columns we actually want to keep for the purposes of training & using the model.
model_cols = ['open', 'high', 'low', 'Volume USD', 'compound', 'positive', 'negative', 'neutral', 'polarity', 'subjectivity', 'price_change']
os.chdir(r'C:\Users\WaKaBurd\Documents\GitHub\CryptoPredictionTool\hourly_coin_data')

for i in range(1):
    # for i in range(len(hourly_coin_data)):
        model_df = hourly_coin_data[i][model_cols]
        model_df.to_csv(f'model_df_{i}.csv')

        # Feature Dataset
        x = model_df
        # Target Dataset
        y = np.array(model_df['price_change'])
        x.drop(['price_change'], axis=1, inplace=True)
        np.asarray(x)

        # split into test & train
        x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)

        # Create LDA model
        model = LinearDiscriminantAnalysis().fit(x_train, y_train)
        predictions = model.predict(x_test)
        print(classification_report(y_test, predictions))

```

[25] Python

Now that we have the sentiment values associated with our fetched tweets and the price values for our cryptocurrencies stored by hour, we're ready and able to move into creating the model. The "model_cols" list contains all of the column names that I'm going to be keeping for the purpose of training our model. From there, I went through each of the created datasets, and grabbed only the columns I needed and placed them in a separate model dataframe. I also saved this dataframe for future testing purposes, because at this point all of the time-costly calculations are done, and I can use the dataset to quickly create another model as needed. Once I had my model dataframe, I simply extracted my target dataset and my feature dataset from the dataframe, then used SciKit's *train_test_split* function to split the values into training & testing sets respectively. From there, I used the *LinearDiscriminantAnalysis* function with the training datasets to construct a model for testing purposes, and test the accuracy of the set by having it predict the values in my *x_test* dataset.

```
print(classification_report(y_test, predictions))
```

[5] ✓ 0.1s

...	precision	recall	f1-score	support
0.0	1.00	0.64	0.78	11
1.0	0.88	1.00	0.94	30
accuracy			0.90	41
macro avg	0.94	0.82	0.86	41
weighted avg	0.91	0.90	0.89	41

The classification report tool will compare the predictions from the test set against the *y_test* dataset created (the actual target values from the corresponding *x_test* dataset) and generate the following values:

- **Precision:** what percentage of your predictions were correct
- **Recall:** what percentage of the positive cases were caught
- **F1-score:** what percentage of the positive predictions were correct
- **Support:** number of predictions

As we can see, from the 41 total predictions on the test set, the model achieved an overall accuracy of 90%. One thing to note about this is the size of this test set. Being that I used 20% of the total dataset (only 41 hours of tweets & prices) to test the model's accuracy, that's a small amount of total training data (approx 200 hours of data for training). As time goes on and I'm able to collect more hours of tweets and corresponding hours of price data for training and testing purposes, the overall size of the training/testing set will increase and allow me to create a more trustable and stable model.

5.1.7 Pull live data to make Prediction

Now that I have a trained model on the dataset I've created, I can move on to making predictions on live data. In order to do that all I need are two things: live tweets and live price data from the past hour.

Pulling live tweets using Twitter API:

```
def send_request(url, headers, params, next_token=None):  
    params['next_token'] = next_token  
    response = requests.request('GET', url, headers=headers, params=params)  
    print('Endpoint response code:' + str(response.status_code))  
    if (response.status_code != 200):  
        raise Exception(response.status_code, response.text)  
    return response.json()
```

Starting with the tweets, I'm going to be using the TwitterAPI for pulling live tweets as its much more stable and easy to use than Twint. Twint shines when I need to pull a large amount of data from up to 10 days prior, while the Twitter API is designed for pulling a smaller amount of tweets (max of 150 every 15 minutes) from a more recent time period. So, I created this *send_request* function that accepts the proper arguments for the purpose of querying the TwitterAPI endpoint, and notify me if there is any issues in hitting the endpoint by outputting the json response in the form of an HTTP status code.

```

def pull_live_tweets(coin):

    # Pull tweets from the last hour
    path = r'c:\Users\WaKaBurd\Documents\GitHub\CryptoPredictionTool\predicted_trends'
    os.chdir(path)
    #os.chdir(coin)

    print('performing twitter search for coin:', coin)

    # 1 hour ago
    from_date = datetime.now(timezone.utc) - timedelta(hours = 1)
    to_date = datetime.now(timezone.utc) + timedelta(seconds=-30)

    iso_from_date = from_date.isoformat()
    iso_to_date = to_date.isoformat()

    from_date = from_date.strftime('%Y-%m-%d %H:%M:%S')
    to_date = to_date.strftime('%Y-%m-%d %H:%M:%S')

    print(f'searching {from_date} to {to_date}')

    bearer_token = '<your-token-here>'

    headers = {
        "Authorization": "Bearer {}".format(bearer_token)
    }

    url = 'https://api.twitter.com/2/tweets/search/recent'

```

From there, I created a function specifically for pulling live tweets where I just need to pass in the coin name, and it will construct a set of parameters in JSON format from 1 hour before the function is called to 30 seconds prior (has to be at most 30 seconds before the time the command is run, so `timedelta` is used to give us the required gap), convert them into ISO format, group the headers and params together and hit the Twitter API endpoint using the `send_request` function (see below).

```

url = 'https://api.twitter.com/2/tweets/search/recent'

params = {
    'query': coin,
    'start_time': iso_from_date,
    'end_time': iso_to_date,
    'max_results': 100,
    'next_token': {}
}

json_response = send_request(url, headers, params)
return json_response

# Pull tweets on topic from last 30 minutes
fetched_tweets = pull_live_tweets('AVAX lang:en')
fetched_tweets_df = pd.DataFrame(fetched_tweets['data'])
fetched_tweets_df.to_csv('recently_fetched_tweets.csv')

```

```

performing twitter search for coin: AVAX lang:en
searching 2022-05-03 23:11:51 to 2022-05-04 00:11:21
Endpoint response code:200

```

The bottom three lines of the cell are essentially just returning the JSON response from the twitter API to the *fetched_tweets* variable, storing that json into a pandas dataframe, then saving those tweets for future use. You can see the output from the cell when its run as well, showcasing that the search was made on the AVAX coin, from the specified time frame and received a response code of 200 (HTTP OK).

Pulling live price data using AlphaVantage API:

```
# Uses AlphaVantage API with their CRYPTO_INTRADAY endpoint.

av_api_key = '<your-api-key-here>'
path = r'c:\Users\WaKaBurd\Documents\GitHub\CryptoPredictionTool\prices\LivePrices'
os.chdir(path)

def get_prices(coin):
    url = f'https://www.alphavantage.co/query?function=CRYPTO_INTRADAY&symbol={coin}&market=USD&interval=1min&apikey={av_api_key}&datatype=csv'
    req = requests.get(url)
    data = req.content
    csv_file = open(f'{coin}_prices.csv', 'wb')
    csv_file.write(data)
    csv_file.close()
    return

get_prices('AVAX') # Get the prices from the specified coin

# format that data into a dataframe
live_prices = pd.read_csv('AVAX_prices.csv') # read in live prices csv
kept_prices = live_prices.head(60) # keep only the last 60 minutes.
high = kept_prices['high'].max(axis=0) # Find the max value in the last 60 minutes
low = kept_prices['low'].min(axis=0) # find the lowesst value in the last 60 minutes
open_price = kept_prices['open'].values[59] # Price from 60 minutes ago. (opening price of the last hour)
volume = kept_prices['volume'].sum(axis=0) # summate the total volume traded from the last hour

live_coin_data = pd.DataFrame([[open_price, high, low, volume]], columns=['open', 'high', 'low', 'volume'])
```

After the tweets are gathered the next requirement is to grab the live price values for the cryptocurrency in question, and in order to do that I had to use the AlphaVantage API. The API requires an API key similar to the Twitter API, so in order to use it the user must input their own personal API key. The code in this section is fairly simple, the *get_prices* function sends an API request to the AlphaVantage API and requests a response in the form of a CSV, then that response is read into a pandas dataframe, and we only keep the 60 latest prices (corresponding to the last 60 minutes). From there, we grab the open price for that hour, the lowest price for that hour, the highest price for that hour, and the overall volume traded over that hour and append it to *live_coin_data* dataframe.

```

# Run textblob on tweets for polarity & subjectivity
combined_tweets = ' '.join(fetched_tweets_df['text'])

# Clean tweet so we can use textblob on it.
fetched_tweets_df['cleaned_tweet'] = fetched_tweets_df['text'].apply(lambda x: sift_tweet(str(x).lower(), stopwords))
combined_cleaned_tweets = ' '.join(fetched_tweets_df['cleaned_tweet'])

# Get sentiment values on tweets using VADER sentiment analyzer
sia = get_sentiment(combined_tweets)
compound = sia['compound']          # Score representing sum(lexicon ratings)
pos = sia['pos']
neg = sia['neg']
neu = sia['neu']

live_coin_data.loc[live_coin_data.index[0], 'compound'] = compound
live_coin_data.loc[live_coin_data.index[0], 'pos'] = pos
live_coin_data.loc[live_coin_data.index[0], 'neg'] = neg
live_coin_data.loc[live_coin_data.index[0], 'neu'] = neu
live_coin_data.loc[live_coin_data.index[0], 'polarity'] = TextBlob(combined_cleaned_tweets).sentiment[0]
live_coin_data.loc[live_coin_data.index[0], 'subjectivity'] = TextBlob(combined_cleaned_tweets).sentiment[1]

# make the prediction
model.predict(live_coin_data)
prob = model.predict_proba(live_coin_data)

```

Once the live tweets and live prices are gathered, a sentiment analyzer needs to be run on the tweets using Textblob (for polarity and subjectivity) and the Sentiment Intensity Analyzer (for compound, positive, negative & neutral) they are combined into a single pandas dataframe for feeding into the model so it can classify it with a label signifying whether or not the price will increase/decrease over the next hour. That prediction is stored in the *prob* variable, as we use it in to showcase the probabilistic chance that the model believes it will have that classification at the end of the hour.

```
print(prob[0])
```

[103] ✓ 0.5s

... [0.94941277 0.05058723]

This output is at the bottom of the final cell in the jupyter notebook, and signifies the probability of an increase/decrease in price over the hour according to all of the values entered into the model. The first value signifies the probability that the crypto will decrease in price over the next hour, and the second value signifies the probability that the cryptocurrency will increase in price.

5.2 Back-end API

The API provided was created using FastAPI. Realistically there's only one endpoint I needed to implement for this API, and its purpose is to allow users to easily interact with it through the React front-end I created (explained in the next section). The entirety of the API exists inside the "CryptoPredictionAPI.py" file.

5.2.1 Fast API Implementation

The content of the API file is very similar to the documentation for the previous jupyter notebook, because it recycles a large amount of the same functionality, with a twist of adding the declaration of a FastAPI app at the top. If you open the file, the top will consist of the same import statements as the jupyter notebook, with the addition of 2 FastAPI import statements (one for the library itself, and one for handling CORS). Immediately after that the FastAPI app is declared, with its origin locations and the declared middleware allotted with its allowed origins. This is required for the purpose of the front-end application, as it is built into the react front-end and assists in guaranteeing there are no issues when sending requests across origins.

```

35  from fastapi import FastAPI
36  from fastapi.middleware.cors import CORSMiddleware
37
38
39
40
41  # Run app with command: python -m uvicorn CryptoPredictionAPI:app --reload
42  app = FastAPI()
43
44  origins = [
45      "http://localhost:3000",
46      "http://localhost:8080"
47  ]
48
49  app.add_middleware(
50      CORSMiddleware,
51      allow_origins=origins,
52      allow_credentials=True,
53      allow_methods=["*"],
54      allow_headers=["*"],
55  )

```

```

58  #|=====|
59  #| Additional files and keys:|
60  #|=====|
61
62
63  # Read in stopwords file to list for sifting tweets later on.
64  os.chdir(r'C:\Users\WaKaBurd\Documents\GitHub\CryptoPredictionTool\archive')
65  stopwords_file = open("stopwords.txt", "r+")
66  stopwords = list(stopwords_file.read().split('\n'))
67  av_api_key = '<your-av-api-key>'
68  bearer_token = '<your-twitter-api-bearer-token>'

```

Additionally, just past the previous section of code is a small section for importing important files and declaring API keys. The user of the program needs to modify the directory stated here to mimic their directory in their own personal repository, and also needs to enter their AlphaVantage API key & TwitterAPI bearer token in order for the API to be able to properly query for live tweets and live crypto data using the APIs in this project.

```

301 @app.get("/prediction_generator/{coin}")
302 def generate_prediction(coin: Optional[str] = None):
303     fetched_tweets = pull_live_tweets(f'{coin} lang:en')
304     fetched_tweets_df = pd.DataFrame(fetched_tweets['data'])
305     fetched_tweets_df.to_csv('recently_fetched_tweets.csv')
306
307     # look up prices for that coin from the last hour
308     path = r'C:\Users\WaKaBurd\Documents\GitHub\CryptoPredictionTool\prices\LivePrices'
309     os.chdir(path)
310     url = f'https://www.alphavantage.co/query?function=CRYPTO_INTRADAY&symbol={coin}&market=USD&interval=1min&apikey={av_api_key}&datatype=csv'
311     req = requests.get(url)
312     data = req.content
313     csv_file = open(f'{coin}_prices.csv', 'wb')
314     csv_file.write(data)
315     csv_file.close()
316
317     # use old model/generate model for that coin
318     model = gen_model()
319
320     # make prediction
321     prediction = make_live_prediction(fetched_tweets_df, model)
322     d = dict(enumerate(prediction.flatten(), 1))
323     resp = json.dumps(d)
324
325     # return prediction
326     return resp

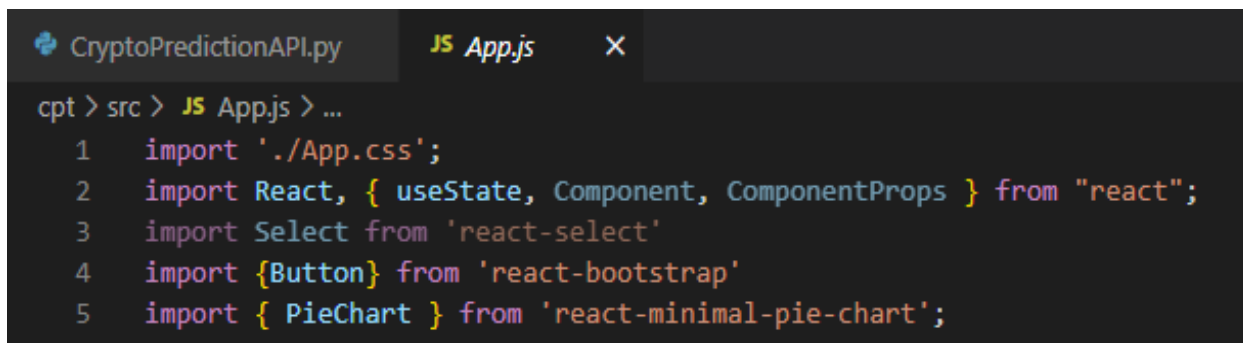
```

The above screenshot is from the bottom section of the API python file, and essentially just provides the logic for the API's functionality. Line 301 in the screenshot declares the endpoint and request type, and binds the request to the *generate_prediction* function, which will take the coin provided in the endpoint as its only argument parameter. From there, it will pull the live tweets, store them in a dataframe, grab the live prices, generate the model (using the models created and stored during the jupyter notebook execution), and then make a live prediction on the fetched tweets. It then takes that prediction, stores it in a dictionary, converts it to a JSON and returns it to the requester as a JSON response.

5.3 User Interface

The UI created and supplied alongside this project has a very simple goal: present the user with a way to interact with the project and obtain the necessary information as quickly and easily as possible. In order to accomplish that, a pie chart was created as it can clearly and quickly indicate the probability supplied from the back-end API. The front-end application was created using the React javascript library, as it is a very quick and easy way to get powerful application interfaces assembled and deployed.

5.3.1 React Imports & App declaration

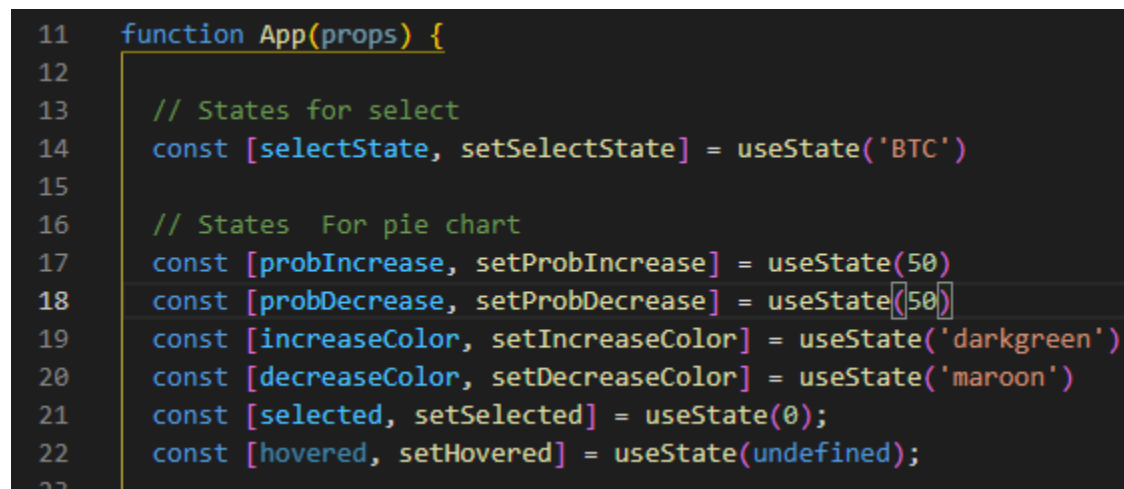


```

cpt > src > JS App.js > ...
1  import './App.css';
2  import React, { useState, Component, ComponentProps } from "react";
3  import Select from 'react-select'
4  import {Button} from 'react-bootstrap'
5  import { PieChart } from 'react-minimal-pie-chart';

```

For developing the front-end application I only imported a few React components, namely the Button, Select & PieChart components. I also defined the majority of the formatting for my page and components inside the app.css file (as you likely would for any web-application).



```

11  function App(props) {
12
13      // States for select
14      const [selectState, setSelectState] = useState('BTC')
15
16      // States For pie chart
17      const [probIncrease, setProbIncrease] = useState(50)
18      const [probDecrease, setProbDecrease] = useState(50)
19      const [increaseColor, setIncreaseColor] = useState('darkgreen')
20      const [decreaseColor, setDecreaseColor] = useState('maroon')
21      const [selected, setSelected] = useState(0);
22      const [hovered, setHovered] = useState(undefined);
23

```

React hooks were utilized to monitor and modify the values being stored and interacted with on the components. Just to briefly describe how to interpret these hooks, looking at the above screenshot the select component only uses one hook, defaulting the value to “BTC” that monitors which option the user has selected and stores it in the “selectState” variable, which is bound to the *setSelectState* function. The Pie chart component uses six hooks to monitor the

values, colors, and user interaction (whether they've selected a slice of the pie or are hovering over it) of the chart. The probabilities for whether a price will increase/decrease are both set to 50 to begin, and the slices of the pie are dark green and dark red respectively.

5.3.2 Select Component

```
<div>
  <select
    className="MySelect"
    value={selectState}
    onChange={handleSelectChange}
  >
    <option value='BTC'>BTC</option>
    <option value='ETH'>ETH</option>
    <option value='AVAX'>AVAX</option>
    <option value='AAVE'>AAVE</option>
    <option value='BCH'>BCH</option>
  </select>
</div>
```

```
function handleSelectChange(e) {
  setSelectState(e.target.value)
  console.log('selection made')
}
```

The two above screenshots showcase the code for the Select component and how the select component handles changes in real-time. The first screenshot outlines the class name, value to be stored relating to the component, and an onChange event handler that specifies what function to call when an event occurs. The only event that occurs when the Select component is when a user selects a value from the select dropdown (which are listed in the varying option values on the lines after the onChange declaration line). So when a selection is made using this component, the function receives the event, modifies the **selectState** variable using the *setSelectState* function, and logs the change so the user can verify the selection was changed

properly. This makes sure the value stored within the MySelect component matches the currently selected value so when the “GetResults” button is pressed, the *getPrediction()* function is called.

5.3.4 Button Component

```
async function getPrediction() {  
  console.log('getting predictions...')  
  
  let jsondata;  
  await fetch('http://127.0.0.1:8000/prediction_generator/' + selectState).then(  
    function(u){ return u.json();}  
  ).then(  
    function(json){  
      jsondata = json;  
    }  
  )  
  
  var json = JSON.parse(jsondata)  
  
  setProbIncrease(json['1'] * 100)  
  setProbDecrease(json['2'] * 100)  
}
```

The *getPrediction* function is how the front-end application interacts with the back-end API to retrieve live values and update the probability live as the user interacts with the application. This is accomplished through the use of a *fetch()* command, where we pass in the URL for the prediction_generator endpoint, with the currently selected state appended to the end of the string. It awaits the return of a JSON formatted message from the API (the probabilities), then parses that returned information in to a format that can be interpreted by the React app and uses the *setProbIncrease* and *setProbDecrease* functions to set the returned probabilities and update the chart (returned information must be multiplied by 100 to convert it from a decimal value to a % value).

5.3.5 Pie Chart Component

```
<div className="MyPieChart">
  <PieChart
    style={{
      fontFamily:
        '"Nunito Sans", -apple-system, Helvetica, Arial, sans-serif',
      fontSize: '8px',
    }}
    data={[
      { title: "ProbIncrease", value:probIncrease, color: increaseColor, key: 'increase'},
      { title: "ProbDecrease", value:probDecrease, color: decreaseColor, key: 'decrease'}
    ]}
  />
</div>
```

This screenshot showcases the code required to style and render the Pie Chart on the front-end application. The majority of the code for this component is formatting (which isn't shown as its simple CSS and formatting commands), but the one thing to note is the value variable stored inside the data block associated with the PieChart component. Notice that it refers to either the **probIncrease/probDecrease** variable associated with the hook from earlier. This way, it will always display the most recently stored/retrieved variable from either the declaration of the hook itself, or set in the *getPrediction* function above. This use of states allows the React app to update the Pie Chart component in real-time, instead of having to refresh the web-page to showcase when a new prediction is retrieved.

6.0 Conclusions and Future Work

As a whole, completing this project a ton was learned in the process but the goal line was reached. I was able to create a working prediction generator for select cryptocurrencies that can be used to assist newcomers to the cryptocurrency atmosphere in timing their investments. The probabilities provided by the Linear Discriminant Analysis model, the datasets created, and the tools created for interacting with the project from both a newcomer's perspective as well as a researcher's perspective show its potential for use. However, while it may have a great potential use in the crypto atmosphere, there is also great potential for improvement.

When I first began this project, I believed the Twint tweet scraping tool was going to allow me to grab more tweets than I would even need, and the Kaggle datasets available for tweets about cryptocurrency would also assist in filling some of the gaps. But, in using it I came to the realization that I can only pull tweets that are up to 10 days old if I'm searching based on a keyword using Twint. Also, the dataset was very limited in the amount of tweets it had available from an hourly/daily standpoint (only 43 days and 120 total hours of tweets). This limits the amount of tweets I was able to gather/use for training & testing the model, as I need both tweets from those hours as well as the corresponding price data from that hour-long timeframe. The creation of an automated twitter scraping tool, that pulls information every so often about cryptocurrencies of interest and stores them into a MongoDB instance would be a great addition to this project, and solve issues related to the quantity of data available.

Another improvement or future implementation for this project would be to attempt to use other classification methods and identify which model yields the most accurate predictions. Originally my hope was to use a recurrent neural network to predict the exact price of a currency over a few hours/days, but in feeding the data to the model I came to the conclusion that this

approach was ultimately inconclusive, and provided essentially useless results. After a few days/weeks of attempting to tune this model, I came to the conclusion that the tool would likely be easier to use & produce more useful predictions through the use of a classification algorithm. I settled on using Linear Discriminant Analysis, but I would like to implement a Support Vector Machine approach as well to attempt to classify more difficult points to place. This might improve the overall accuracy of the data, especially in the case of outlying data points that are likely to occur as a result of a highly volatile cryptocurrency market.

These implementations could easily be implemented in the back-end API and front-end application as well; providing the user with separate endpoints altogether that could allow the user to request information from when using a different form of model. This would also allow me to add more charts/information to my front-end to give it a more professional look and feel. Also, hosting the front-end application on a web-hosted service like AWS would increase the overall reach and ease-of-use of this project tremendously.

All things considered, I'm very proud of my work on this project, and look forward to working on it further with any spare time I have over the next few months/years until I feel as though I've done all I can do with it. I'm grateful for the opportunity to have been able to create this application (as it's something I've always wanted to do). I am extremely excited to see if I can expose this application to the cryptocurrency community and allow investors or researchers alike to interact with it and provide feedback, so I can improve upon what I've created and continue to modify it as the cryptocurrency sphere continues to grow and flourish.

References

- Abraham, J., Higdon, D., Nelson, J., & Ibarra, J. (2018). Cryptocurrency price prediction using tweet volumes and sentiment analysis. *SMU Data Science Review*, 1(3), 1.
- Fang, F., Ventre, C., Basios, M., Kanthan, L., Martinez-Rego, D., Wu, F., & Li, L. (2022). Cryptocurrency trading: a comprehensive survey. *Financial Innovation*, 8(1), 1-59.
- Ho, T. T., & Huang, Y. (2021). Stock Price Movement Prediction Using Sentiment Analysis and CandleStick Chart Representation. *Sensors*, 21(23), 7957.
- Jay, P., Kalariya, V., Parmar, P., Tanwar, S., Kumar, N., & Alazab, M. (2020). Stochastic neural networks for cryptocurrency price prediction. *Ieee access*, 8, 82804-82818.
- Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, 21260.
- Nguyen, T. H., Shirai, K., & Velcin, J. (2015). Sentiment analysis on social media for stock movement prediction. *Expert Systems with Applications*, 42(24), 9603-9611.
- VantagePoint. (2021, May 5). Proven Accuracy. Retrieved March 1, 2022 from <https://www.vantagepointsoftware.com/accuracy>.
- Xi, D., O'Brien, T. I., & Irannezhad, E. (2020). Investigating the investment behaviors in cryptocurrency. *The Journal of Alternative Investments*, 23(2), 141-160.