July 14, 2024

C964: Computer Science Capstone

Task 2 parts A, B, C and D

Part A: Letter of Transmittal

May 01, 2024

Alexander Johnson
Chief Technology Officer
Scrappy Investors, Inc
1306 Scrappy Boulevard, Boise, Idaho

Dear Mr. Johnson,

Scrappy Investors' (SI) singular purpose is to proactively invest for busy clients. That and, of course, to make a profit. However, as clients request diversification by way of cryptocurrencies, we have begun to experience significant challenges using our legacy tools for price analysis. Our traditional methods and penchant for value investing have proven ineffective when applied to highly volatile crypto coins. To accommodate these new investment options and equip our market analysts, I propose we introduce machine learning to our cryptocurrency due diligence pipeline.

Where more traditional investment categories have well-established metrics for market analysis, cryptocurrency is far from traditional. This market is riddled with volatility. When in-house traders enter this arena without any tools to ground them, they tend to make rash decisions under pressure. I believe price prediction powered by machine learning will better inform our investors and give them the confidence they need to make decisive in-the-money trades.

Our goal at SI has been, and always will be, to buy low and sell high. And yet, this universal investing goal is so elusive that it has become a running joke. Its simplicity belies the tumultuous nature of public markets and a human tendency towards poor decision making under pressure. For cryptocurrencies, these dangers are especially pronounced. Traditional investors view cryptocurrencies as risky ways to burn money, driven by unpredictable social media trends and elusive technology advancements. Trading in such volatility, without traditional venues for due diligence like company valuations, gpd, and profit margins, means investors are left with vague metrics like "gut feelings" and "market sentiment" to drive their buying decisions. This unpredictability can lead to suboptimal trading and significant damage to our bottom line. By deploying machine learning to historical price data, SI can equip our traders with data-driven anchors from which to operate. Employees will have hard numbers, will reduce their emotional decision making, and will enhance trading confidence, thereby increasing SI's profits while minimizing losses. Moreover, this machine learning can go beyond price prediction and provide employees with automatic data visualization. They will be able to select a date-range and automatically generate histograms, line graphs, and data descriptions in real time.

As we take our first step into machine learning, I propose we begin slowly with a proof-of-concept pilot model. Doing so means we can test various hypotheses for appropriate modeling and minimize costs in the process. The pilot will be a recurrent neural network (more specifically, a Long Short-Term Memory Network), and be limited to indicating future closing values of Ethereum cryptocurrency based on historical data. The model itself will be wrapped in a simple user interface so traders can focus less on technology and more on leveraging information. The hope is that this will allow them to make faster, better decisions with minimal friction. If we are happy with the results of the pilot, we could include additional cryptocurrencies to our model with limited overhead and expand our model's utility from a position of advantage.

The total cost to implement the new model would be approximately $11,500 for the first year, to include: hardware, cloud infrastructure, development, and training. Most of the resources required to implement the project are already available in-house, and costs will largely come from our labor pool. All tools will be open source and will present nominal expenses, if any. Additional maintenance costs will be no more than $6,000 in the years following deployment, pending further feature additions if we are happy with the product.

Our project will be business focused from start to finish, and will follow the Waterfall Methodology. We will use the Ethereum cryptocurrency as our pilot currency, leveraging a public dataset containing daily closing prices from Ethereum's inception in 2015 to the first quarter of 2022.

The project will require approximately two months of development. First, we will prepare and normalize the dataset for use. Next, we will build out the Long Short-Term Memory Network and train it to acceptable accuracy. Third, we would implement our User Interface with a focus on functionality and simplicity. We would then evaluate the model in a simulated testing environment with the help of experienced beta testers. Finally, we would deploy our model for general use within SI and train our employees on its use.

Using my institutional knowledge of SI as well as my four years of software engineering experience and Bachelors of Computer Science, I will be the lead developer of this project.

I strongly believe machine learning will position SI to better navigate current markets and increase profits. I look forward to discussing this proposal further, and exploring how position Scrappy Investors to be resilient in an unpredictable future.

Sincerely,

James Revello

James Revello
Software Engineer
SID # 010649181
Scrappy Investors, Inc.

Part B: Project Proposal Plan

Project Summary

The Problem

Cryptocurrency trading is characterized largely by volatility. High opportunities for reward are tempered by similarly high risks and potential for loss. Traditional trading strategies, such as value investing, fail to account for and manage this volatility. Within SI, traders fall prey to emotionally-driven decisions in such an environment. Under pressure, employees see prices skyrocket and rush to buy for fear of missing out—only to be punished by buying the bubble. Still others think prices have hit rock bottom, placing inordinately high buy orders only to catch a falling knife.

As markets evolve and new currencies and technologies develop, fortunes are made and lost in the blink of an eye. This presents both great opportunity and great risk. Our startup investment company has largely managed to capitalize on rapid price fluctuations to maximize returns, operating on razor thin margins. However, high volatility environments push even our most experienced traders into rash decision making. To temper our traders, equip them for due diligence in untraditional environments, and anchor their decisions in predicted prices, we require a reliable machine learning model from which to operate.

The Solution

I propose SI utilize a predictive machine learning model to forecast price movements and equip traders to make more informed decisions while minimizing losses. Even if the model is marginally better than an emotionally vulnerable human's best guess, it is well worth the investment. The model's value lies not only in its ability to predict prices and perform data analysis, but to inspire confidence in investors using the tool. With this confidence, our employees will make decisions fueled by grounded, data-driven information, instead of vague gut feelings.

The predictive model proposed is a Long Short-Term Memory (LSTM) Network. Neural networks are based loosely on the human brain. A given network has layers of neurons which receive stimulus (input), process said stimulus in some way, and then pass the processed data to the next layer as output. With this constant input/processing/outputting, a neural network with sufficiently large sets of data can predict future outcomes automatically. The proposed LSTM is an evolution of previous artificial neural networks like the Recurrent Neural Network and is ideal for processing data over long periods of time.

LSTMs possess memory block which contain an input gate, output gate, and forget gate. The first gate controls what information is added from the input provided. The second gate processes the input and formulates output based on the memory it currently possesses. The third gate discards information over time and frees up memory. Essentially, LSTMs remember previous input and use that memory for sophisticated learning and higher levels of precision in future training. LSTMs are perfect for finding patterns over time based on historical data. In other words, they are perfect for SI's present problem.

With a simple forecast, we will minimize our investor's tendency toward rash ill-timed decisions and prevent substantial losses. We will provide them with the tools they need to make informed and confident buying and selling decisions. Moreover, the model will be accessed via an intuitive and simple user interface so our investors have less friction between them and a profitable trade.

The Client

Scrappy Investors Incorporated is a startup investment company that specializes in proactively trading in stocks and cryptocurrency for its customers. While SI is the institutional client served by this proposed model, and will no doubt benefit from more profits because of the LSTM, it is not our target audience. This in-house, bespoke neural network will be targeted first and foremost to our line-traders. The proposed design is tailored to our own employee population with our own preferences and workflows in mind—NOT the general market. Our traders are experts in finance and trading, but they don't have in-depth technological expertise. Our traders do not have time to hand-code their own solutions, they need something simple that solves a problem with minimal fuss. Most importantly, our traders need more information on which to base their trades. They are stuck with vague market sentiment and no hard numbers on which to value their buy and sell orders. Our traders need a tool to accurately inform their decisions based on past historical data.

Taking the above into account, our model will be built on open-source libraries and technology. It will contain a simple and intuitive user interface nestled within a Jupyter Notebook, so even our most non-technical investor will feel comfortable with its use. The model's UI will be designed such that it can aid in price analysis while minimizing frustration. Each cell will be activated via button to ensure intentional running of code. Predictions will be held to strict speeds so investors do not wait an inordinate amount of time for results. But most importantly, the model will provide accurate predictions of future prices based on past data. It will enable traders to make informed decisions based on tangible data rather than ethereal and fallible instincts. As a bonus, the model will allow traders to investigate past data at their own pace, delving into the available data set and comparing previous closing values if they so choose.

Deliverables

Our first major deliverable is the Ethereum price data we will ultimately use to train our proposed machine learning model. By collecting and processing this data for training, we will accumulate a bolus of historical data which data analysts can assess for future investment strategies. While the LSTM will be limited toward predicting closing prices, the data itself will be analyzed automatically and transformed into histograms, line graphs, boxplots, and more.

In addition to the training data, we will produce the LSTM model itself. It will be trained on the dataset and will be able to predict future Ethereum prices with sufficient training.

To access the LSTM based machine learning model, we will develop an easy-to-use user interface via Jupyter Notebook. This will be the client-facing front end our traders will use to access our model.

To train employees on the use of the new model, we will develop and deliver a user guide and recurring in-house training to ensure everyone is confident in its use.

Application Benefits

The proposed machine learning application will enhance our investor's trading accuracy by predicting Ethereum price movements. It will inform their various trades and provide another crucial variable on which to decide to buy or sell. The model will also mitigate emotion-based decision making. In lieu of knee-jerk reactions, traders will have objective, data driven insights on which to operate. Our trading model will effectively reduce the number of losing trades by anchoring our traders to hard data, thereby reducing risk. Taking all the above into account, the model will result in increased profits and decreased risk and loss.

Data Summary

As this proposal is a prototype, we will start with a simple Kaggle Dataset so we do not constantly update and change our primary testing variable. This means our data will NOT be current, a factor to keep in mind as we evaluate our results. Should the pilot prove successful, we will be able to update the program, with minimal effort, to pull current data from Yahoo Finance (ETH-USD) or upload able local CSV. Because we are using a publicly available Kaggle resource, all data is public, open-source information based on public markets. There are no legal or ethical concerns involved in its use.

The Dataset captures seven datapoints: Date (String), Market Open Price (Float), Daily High Price (Float), Daily Low Price (Float), Market Close Price (Float), Adjusted Close (Float), and Trade Volume (Int). We will additionally modify our data to split our Date string into three integers: Year, Month, and Day.

Methodology and Implementation

Our project will be governed by the Waterfall Methodology, while monolithic, we have known business requirements to drive our project in a narrow and focused direction. A strict structure for design and linear development will be more effective here than iterative production like in an Agile methodology would require. This is a highly structured methodology that puts the business needs at the forefront of our development. Additionally, due to the short timeframe of our project, we do not foresee a change in business requirements or regulations derailing our project. This largely negates one of Waterfall's largest weaknesses.

We will start by developing and defining the objectives and requirements of SI through meetings with stakeholders (in this case, our end users) to ensure we build something that will be used in the end.

We will then create a design that is tailored to the requirements. In this case, we know the design will need to be simple and user friendly. It will also need to be responsive. The initial information from the previous requirements phase will target more specific goals such as features requested in our meeting.

Next, we will implement the preceding steps by writing the code for our model. This is where we will clean and process our data, implement our model, and train it.

Following implementation, we will verify the code meets initial requirements through careful validation of accuracy (via error analysis and heatmapping). We will beta test with select employees so we have feedback from the target users on the model itself, as well as the user interface we have designed. This will be our opportunity to refine our project before deployment.

Finally, we will deploy the model and begin the maintenance phase. It his highly likely that dissemination to a wider user base will uncover issues which need fixed, such as dependencies and bugs. Our maintenance phase will also encompass the human element: training staff and updating user guides so all employees are proficient at using the new tool.

Timeline

Our timeline is as follows:

Milestone or deliverable	**Duration (hours or days)**	**Projected start date**	**Anticipated end date**
Meet with stakeholders. Define business requirements	1 Day	5/01/2024	5/02/2024
Data Acquisition and Analysis	1 Week	5/02/2024	5/09/2024
Data Preparation and Normalization	1 Week	5/09/2024	5/13/2024
Model Coding	1 Week	5/13/2024	5/20/2024
Model Training and Tuning	1 Week	5/20/2024	5/27/2024
Model Evaluation	1 Week	5/27/2024	6/03/2024
Testing in Simulated Market	1 Week	6/03/2024	6/10/2024
Integration to Live Trades	2 Weeks	6/10/2024	6/24/2024
User Interface and Feedback	2 Weeks	06/24/2024	7/08/2024
Final Testing and Signoff	2 Days	07/08/2024	7/10/2024
Ongoing Monitoring and Training	Continuous	7/10/2024	No End Date

Evaluation Plan

We will evaluate and verify our methods at each stage of development. During the data preparation stage, we will verify our data quality through Exploratory Data Analysis (EDA). During the model training phase, we will ensure proper normalization and shaping of data which will fit the LSTM network we will use from the Karas Library.

Finally, for the model evaluation phase, we'll compare our model predictions to actual market movements using mean absolute error (MAE) and root mean squared error (RMSE). We will also examine our Mean Absolute Percentage Error to visualize the error as a percentage. We will round out our analysis with heat maps of the results. We will also include a line graph which includes our original data set, the training data, and the testing data as separate lines.

Resources and Costs

Since our proposal is to build a prototype, the costs will be minimal by design. Should we wish to develop the model further, we will price more advanced technology and incur more cost as a result.

Our Hardware and Software costs:

1. Computers: $ 4,000.00 (our current hardware is not sufficient for futureproofing)
2. Cloud Servers: $5,500.00 per month (captured by our IaaS Cloud Infrastructure)
3. Software Licenses: $0 (open-source tools only)

*Labor and Time Costs:

1. Project team: 3 team members (data scientist, developer, UI/UX designer)
2. Estimated Labor Time: 720 hours
3. Estimated Labor Cost: $108,000 (assumes $150/hour)

Environment Costs

1. Deployment: $0.00 (we will share the files locally)
2. On Prem Hosting: $5,500 per month (included in cloud server costs)
3. In-House Maintenance: $6,000 annually

Cost Increase: $11,500

*Labor and Time Costs estimated by employee salaries. As SI has these employees available in-house, this would not be a cost increase. Labor costs included here for visibility in case the decision is made to outsource this work as well as considerations in prioritizing this effort over other projects.

Part C: Application

Our application is hosted in Google Collab and will have all source files and data files submitted in a zipped file. It is useable on a Windows 10 machine with any of the company installed browsers. Users will need to utilize their valid Google account to access Google Collab.

The mathematical algorithm applied to our data will be an unsupervised, sequential, linear regression model. The model will be a Long Short Term Memory Network (LSTM), an iteration of a Recurrent Neural Network. While LSTMs can be used for both classification and linear regression, we will be doing the latter. The data will be normalized and transformed, and the model will predict a numerical value.

The user interface will be a Jupiter Notebook with interactive graphs and buttons. It will have visualizations such as line graphs, histograms, heatmaps, describe functions, header visualizations, tail visualizations, & box plots.

Our data will be contained in a static .CSV file. Our source code will be hosted in Google Collab *and* provided in a static source file in case of accidental modification. The Google Collab will be accessible via link. The source files will be less than 200 MB.

Part D: Post-implementation Report

Solution Summary

Our company, Scrappy Investors(SI), has faced significant challenges responding to increased demand for cryptocurrency investments. Due to cryptocurrency's high volatility and lack of traditional modes for due diligence, traditional investment strategies have not been effective. Our traders often encountered hasty and emotional decisions when placed under pressure, and were punished by hefty losses as a result. The lack of reliable data-driven tools pushed our traders to rely on gut feelings and market sentiment in lieu of data analysis and long-term strategies.

To address this, we developed a machine learning application designed to predict cryptocurrency prices and automate data analysis. The model used historical data to forecast future prices, thereby enhancing confidence and grounding decisions in data. Beyond informing decisions, our model provided a user-friendly interface which provided easy access to predictions, reduced the need for instinctive trading, and enhanced employee confidence.

Data Summary

For the initial prototype model, the project incorporated static historical data in the form of a plain text (.CSV) file. This was done to have a single unchanging variable as input for our model so we could test our hypothesis without datasets constantly changing and updating. Likewise, the dataset was pre-cleaned and contained no null values or duplicate dates. Future projects will update the program to pull current data from Yahoo Finance (ETH-USD) for a more accurate model with the possible inclusion of additional currencies.

The dataset was sourced from kaggle.com and can be found by clicking HERE. It contained price data for Ethereum based in US Dollars and spanned from August 7, 2015 to March 25, 2022. All data was public, open-source information from public markets. Therefore, there were no legal or ethical concerns involved in its use.

The dataset captured seven datapoints: Date (String), Market Open Price (Float), Daily High Price (Float), Daily Low Price (Float), Market Close Price (Float), Adjusted Close (Float), and Trade Volume (Int). The first four entries are illustrated below.

	Date	Open	High	Low	Close	Adj Close	Volume
0	2017-11-09	308.644989	329.451996	307.056000	320.884003	320.884003	893249984
1	2017-11-10	320.670990	324.717987	294.541992	299.252991	299.252991	885985984
2	2017-11-11	298.585999	319.453003	298.191986	314.681000	314.681000	842300992
3	2017-11-12	314.690002	319.153015	298.513000	307.907990	307.907990	1613479936
4	2017-11-13	307.024994	328.415009	307.024994	316.716003	316.716003	1041889984

On initial observation, CLOSE and ADJUSTED CLOSE (see verification below). We dropped the adjusted close from our data frame to avoid duplicate/redundant input.

```
df[df['Close'] == df['Adj Close']].shape
```

```
(1598, 7)
```

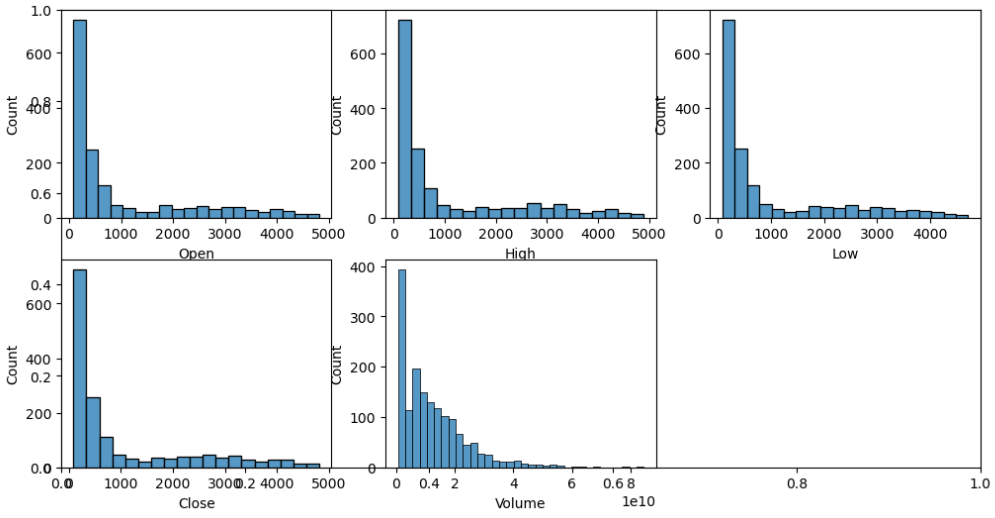
The dataset did not have any null values (see below). This meant our data was complete for each day without any missing entries. There were 1598 total entries. As seen in the lower right table below, our mean or *average* closing price for the dataset was 1027.55 US Dollars. This indicated our dataset had significant outliers, as our 50th percentile closing price was 386.45 USD.

```
df.isnull().sum()
```

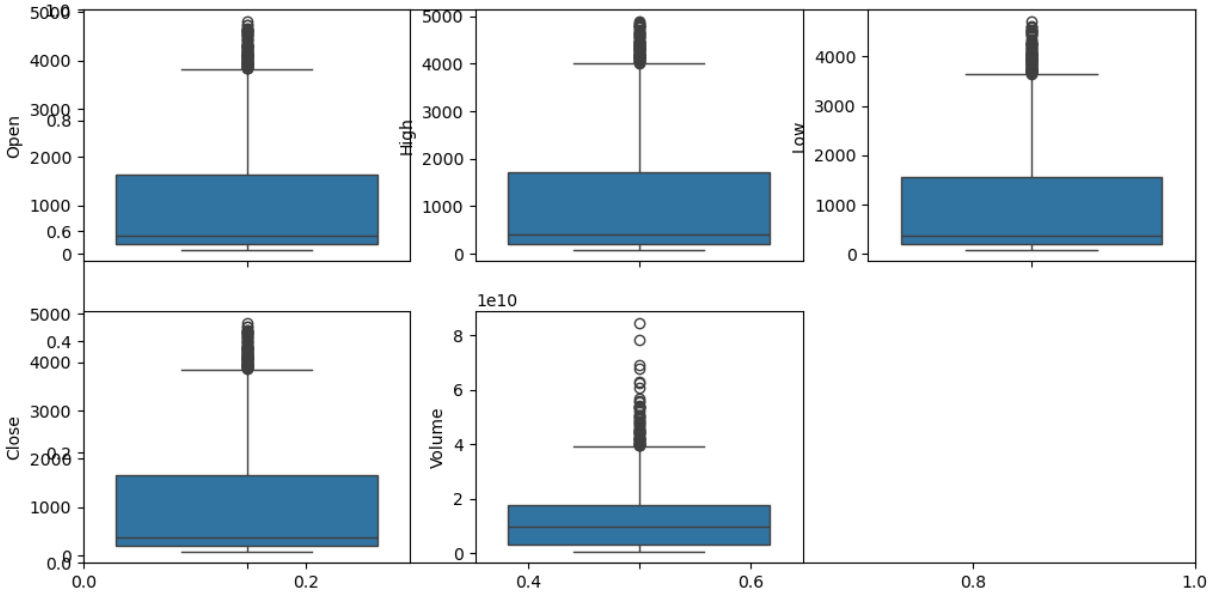
```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

	Open	High	Low	Close	Volume
count	1598.000000	1598.000000	1598.000000	1598.000000	1.598000e+03
mean	1026.060794	1061.282123	986.461636	1027.554834	1.245347e+10
std	1230.979724	1270.582395	1185.384747	1231.398723	1.121937e+10
min	84.279694	85.342743	82.829887	84.308296	6.217330e+08
25%	196.428421	201.416004	188.846440	196.605812	3.154440e+09
50%	386.373398	396.498703	375.446228	386.445556	9.525409e+09
75%	1647.892822	1721.577515	1568.554352	1659.367554	1.764118e+10
max	4810.071289	4891.704590	4718.039063	4812.087402	8.448291e+10

Based on these initial indicators of outliers, we expanded our investigation with histograms of each column (See below). The histograms indicated the data was right skewed. A right skew can be observed when the tail of a histogram extends to the right, as seen in our figures. This also meant the data was skewed *positive*. Based on the aggressiveness of the skew, we had *significant* outliers which was unsurprising given the volatility of Ethereum crypto. We took care to ensure our training set was large enough to accommodate for these outliers, and to normalize the data prior to input into the model.



To further examine outlier significance, we generated boxplots. While all the data captured outliers, the VOLUME outliers were relatively less extreme.

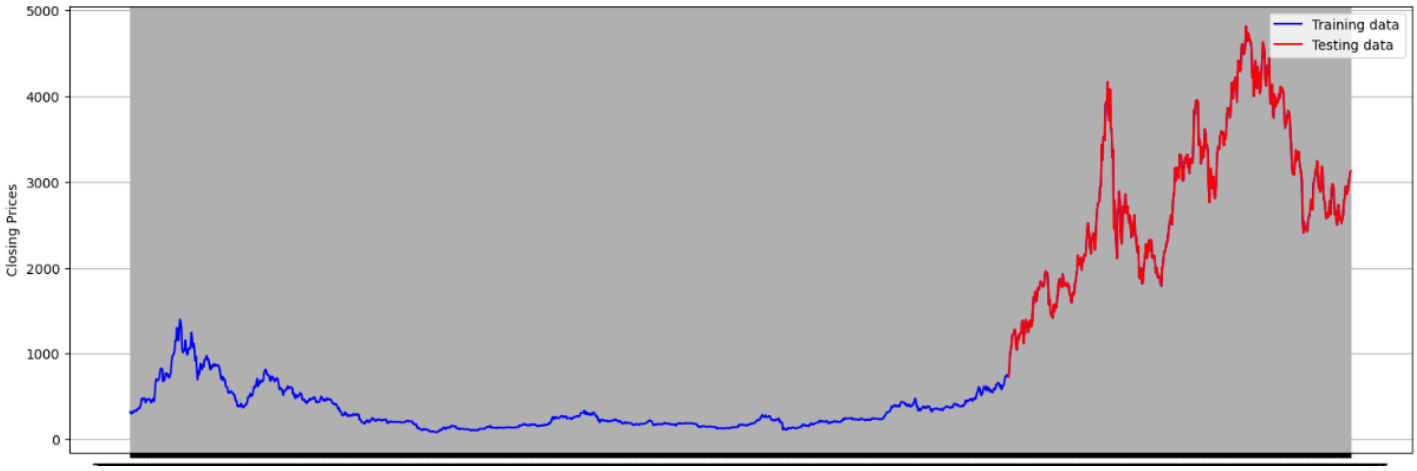


Concluding our exploratory data analysis, we noticed additional features we could use for training, effectively increasing available data points for our model. Our DATE column was a string containing three separate data values which could be split into Year, Month, and Day. Likewise, we used the separated month data to observe quarter end values.

	Date	Open	High	Low	Close	Volume	Year	Month	Day	Quarter End
0	2017-11-09	308.644989	329.451996	307.056000	320.884003	893249984	2017	11	9	0
1	2017-11-10	320.670990	324.717987	294.541992	299.252991	885985984	2017	11	10	0
2	2017-11-11	298.585999	319.453003	298.191986	314.681000	842300992	2017	11	11	0
3	2017-11-12	314.690002	319.153015	298.513000	307.907990	1613479936	2017	11	12	0
4	2017-11-13	307.024994	328.415009	307.024994	316.716003	1041889984	2017	11	13	0

Once we built our features, we split the data into training and testing sets for training and validation. Using the LSTM for rudimentary linear regression, we scale the data and transformed it to values between 0 and 1 so the model could process the same shape.

The training data was from 2017 to 2021 and the test data was from 2021 to 2022. This gave us an approximate 80/20 split of training and test data, seen in the line graph below. One concern we needed to be cognizant of: the training data was *significantly* lower in price than the test data. This pushed the predicted values lower than the actual values. With more current training data, this problem would resolve itself. At present, the most important metric by which to judge the model was to ensure the predictions followed the general shape of the dataset.



The model itself had a lookback period of 60 days, meaning it only considered the previous two months prior to its selected date. Likewise, the model had a dropout of .20, meaning it ignored .20 of its neurons and thus "forgot" 20% of the information from layer to layer. This was done to prevent overtraining and overmatching the data. This overtraining would inflate the accuracy of our model, inspiring false confidence that not be applicable to real world scenarios.

Machine Learning

The predictive model we used in our application was a LSTMN. Neural networks are based loosely on our understanding of the brain. A network has layers of neurons which receive stimulus (Input), process the stimulus in some way, then pass the data to the next layer as output. With this constant input/processing/outputting, a neural network can predict future outcomes based on its results. Our own LSTM was an evolution of previous artificial neural networks (Recurrent Neural Networks) and was better suited for sequence prediction because it uses information from long sequences of data.

Our LSTM model was initialized as a sequential model from the Karas library. We transformed our data into the appropriate input shape, then passed that shape into a variable called "input shape" since Karas preferred an object as input. Then we passed the input shape into input layer. Afterward, we had four layers with fifty neurons each and a dropout of .20. The neurons processed the data. The dropout ensured that 20% of the neurons 'dropped' and not provided to the next layer. This was done to prevent our model from overfitting our data (an overfitted model would not be attributable to real world circumstances and would provide a high "prediction" valuation which may be based on false assumptions). Finally, our "dense" layer had a single unit and returned only one output—our prediction.

When we compiled our model, we used the Adam optimizer because it was straightforward to implement and efficient (so we had fewer memory requirements). This is useful because our model lives in a Jupyter notebook and may be deployed in environments where memory is sparse or later shared via cloud in tools such as Google Collab. We then fit the model to our own data by feeding it the training values and performing ten epochs. Epochs are essentially the number of passes over the data (in this case, we passed over our dataset ten times with the model).

In summary, we used an LSTM because it was appropriate for our historical analysis which covered a large range of time and substantial dataset. We chose our dropout of 20% so the model would not overfit our data. We used the Adam

optimizer so it would be efficient and reliable without using inordinate amounts of memory. And we passed over our model ten times.

Validation

```
MSE: 0.01638794912319807
MAE: 0.10613616597148741
RMSE: 0.1280154253330358
MAPE: 0.13827275780147116
```

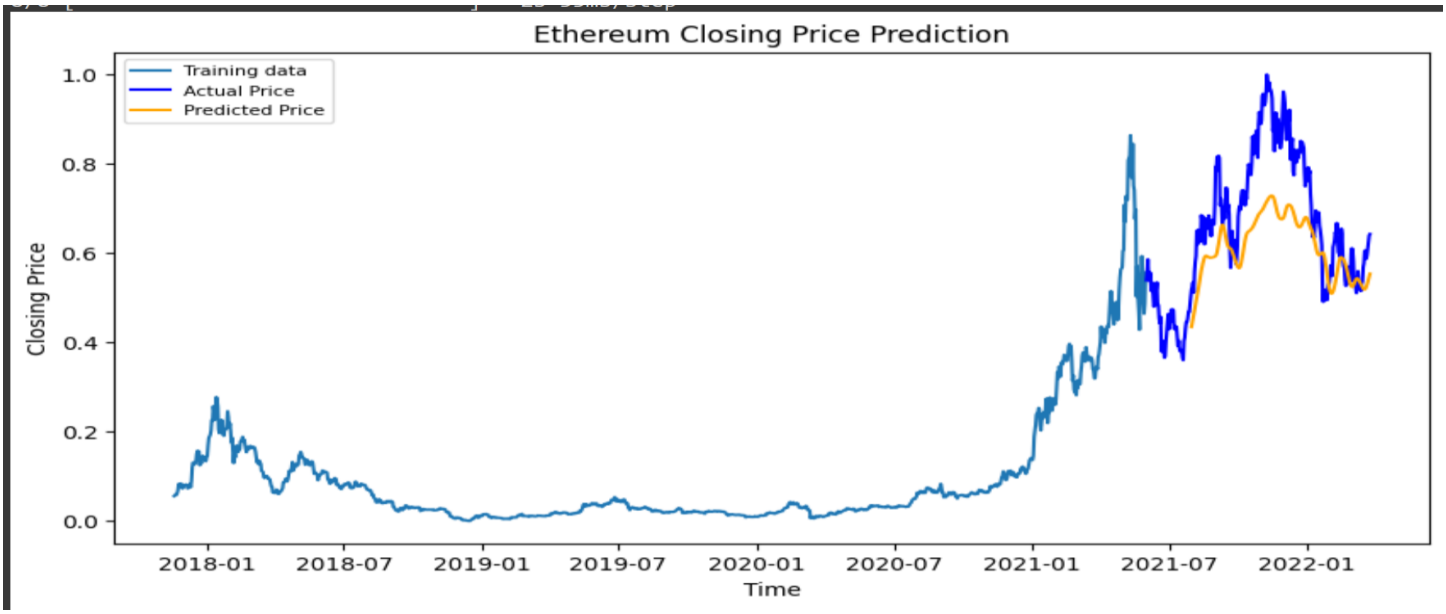
To evaluate our machine learning model, we used several different statistical metrics so we had diverse analysis of the results. First, we used Mean Squared Error (MSE) which averaged the squared difference between our predicted values and the actual ones. We used it because it gave us a sense of how far our predictions were from the true values. A lower MSE is predictive of model accuracy. Our prediction's mean squared error was .0163 in this case, which meant our prediction was relatively accurate. Note, MSE's are highly susceptible to outliers due to the squaring of values. This meant our data was normalized at significant data size.

We next used Mean Absolute Error (MAE), which provided the average magnitude of errors in our prediction. It gave a straightforward interpretation of the average error value between data which was helpful, but it could not tell us whether the differences were negative or positive. Our MAE was .1061, meaning our predictions were, on average, off by about 11 Cents. This was suspiciously accurate, and presented concerns we may have overtrained our data.

Third, we evaluated Root Mean Squared Error (RMSE). We had an RMSE of 0.1280 which meant our predictions deviated from actual values by about 13 cents. This metric was used because it penalized large errors more heavily than small ones.

Finally, we used a Mean Absolute Percentage Error. It illustrated that our predictions were off by a certain percentage (14% in this case).

These results were discouraging, because they indicated our model may have been overtrained. They are suspiciously accurate. But they are valuable nonetheless because they follow the shape of our data very well and can predict the next day's value will go up or down with confidence.



As we see in the figure, our model followed trends well and following the general shape of the line graph with relative accuracy. The largest problem with the model was its tendency to dramatically underestimate the price. Nearly every value was well under the actual price of the data. I believe this is due to the relatively low closing prices found in the training set. The testing set, on the other hand, saw rather dramatic increases in prices, the last two years of the data being especially high. For our next training, the model can fine-tune and trained on more relevant data with training and testing sets which are less disparate.

Visualizations

Our visualizations were found in the following locations:

1. Head and Tail illustrators in *Date Selector* Cell
2. Head, Description, Histogram, Boxplot, Line graph visualizations in *Analysis* Cell
3. Head illustrator in *Add Features* Cell
4. Line graph in *Train and Plot* Cell
5. Heatmap in *Model Evaluation* Cell
6. Closing Price Line graph in *User Prediction* Cell
7. Heatmap in *User Prediction Evaluation* Cell

User Guide

1. Set-Up Environment:
    a. Navigate to Google Collab and sign in / register.
    b. Open the Scrappy Predictor Jupiter Notebook.
2. Importing Packages and CSV
    a. Run the *Package Importer* cell and click the button.
    b. If the *'eth_data.csv'* file is not uploaded to the current instance (on first use, it won't be):
        i. Click the folder icon in left toolbar
        ii. Click the upload icon and select the 'eth_data.csv' file in browser
3. Running Cells

    a. Click the play button at the top left of introduction cell as test
    b. You should see green checkmark on success
    c. Click this green check for future cells you wish to run
4. Installing Dependencies
    a. Run the *Install Dependencies* cell and click the button
    b. Installation may take a moment.
    c. Repeating this step is not necessary for subsequent uses unless errors exist
5. Select Dates
    a. Run *Date Selector* cell
    b. Select a desired date range via the slider
    c. Click “Filter Data frame”
6. Data Analysis
    a. Run the *Data Analysis* cell
    b. Click buttons to see visualizations of your previously selected date range
7. Add Features
    a. Run the *Add Features* cell and click the button to do the following:
        i. Split Dates column into Year, Month, Day
        ii. Drop the redundant Adj Close column
8. Train and Plot Model
    a. Run the *Train and Plot* cell to train your model on entire dataset
9. Evaluate Model
    a. Run the *Evaluate Model* cell and click button
10. Predict Next Week’s Closing Prices
    a. Run *Price Predictor* cell and click button to see prediction for end of previously selected date range
11. Evaluate Prediction
    a. Run the *Prediction Evaluator* cell and click button to evaluate predictions for previously selected date range (not the whole model)

Reference Page

There are no external references made in this project.