**‘FUNDING SECURED at $420’**



*An extraction, transformation and loading team project focused on Tesla, SpaceX and the Twitter account of Elon Musk*

Tesla Evaluation

# Authors: John Michals, Jeremy Randolph, Kenneth Reed, Khrystyne Vaughan

# Purpose

After narrowly escaping a potentially devastating short position, the bank’s equity trading desk approached our business intelligence group with a request for data surrounding their largest equity position, Tesla. The traders, nearly burned by the now infamous, “Funding Secured” tweet by founder and CEO Elon Musk, requested all available information on Musk’s twitter activity, any available information on SpaceX, Musk’s aerospace transportation services company and direct influencer of Tesla stock price and finally, the Tesla stock price history since the company turned public in the summer of 2010.

The traders had several questions they were attempting to answer with our provided data. Those questions were:

* *Does price volatility increase, decrease or remain flat in the lead up to or the wake of a SpaceX launch?*
* *Does the outcome of a SpaceX launch have a direct effect to the price of the stock?*
* *Does the stock price react to the private twitter account of the founder and CEO?*
* *Does the popularity of a given tweet have a measurable effect on the stock price?*
* *Is the volume of Tesla stock traded subject to change when Musk tweets?*

Through several extraction, transformation and loading techniques, our team was able to provide the equity desk with the requested data in a tabular format stored on a MySQL database. The tables created provided all the relevant data that was reliably retrievable to provide the framework for which to answer the traders’ inquiries.

# Extraction

Stock data sets are some of the more readily available and trustworthy data sets published. Utilizing the data available on yahoo finance, a media property providing financial news and data for the Yahoo network, our team was able to export the Tesla stock data, up to the requested date of October 29th, 2018, to a csv file. The file included Tesla’s daily stock performance since June 29th, 2010 and included several columns that included, date, open, high, low, close, adjusted close, and volume. Below is a link to where the data was originally obtained:

Tesla Stock Price: <https://finance.yahoo.com/quote/TSLA/history/>

The SpaceX data was slightly more difficult to obtain but thankfully, our team is frequent visitors to Kaggle. Kaggle is the world’s largest community of data scientists and contains a plethora of available data sets. On Kaggle, our team was able to find a loaded csv file that contained several columns of data on SpaceX launches. The columns were, time, booster version, launch site, payload, payload mass (kg), orbit, customer, mission outcome and landing outcome. A link to where the data was obtained follows:

SpaceX Launch Data: <https://www.kaggle.com/scoleman/spacex-launch-data>

The Twitter data presented the most interesting and unique challenge for our group to extract. After all, there aren’t exactly user Twitter and Tweet tables floating around on Kaggle or Yahoo. However, by utilizing several Python libraries, namely Tweepy and pandas, as well as the Twitter API, our group was able to successfully extract the most recent Elon Musk tweets. Upon successful connection through the API, a loop was used to extract the tweets from Twitter, and then append each tweet into an array. Only selected fields were extracted, specifically, id, date, text, favorite and retweet counts. The below code highlights the Twitter API utilized, as well as the loop executed, to extract the tweets:

auth = tweepy.OAuthHandler(consumer\_key, consumer\_secret)

auth.set\_access\_token(access\_token, access\_token\_secret)

api = tweepy.API(auth, parser=tweepy.parsers.JSONParser())

*# Target User*

target\_user = "elonmusk"

*# Tweet Texts*

tweet\_texts = []

*# Create a loop to iteratively run API requests*

for x in range(1, 200): #20 per page

*# Get all tweets from home feed (for each page specified)*

public\_tweets = api.user\_timeline(target\_user, page=x)

*# Loop through all tweets*

for tweet in public\_tweets:

*# Print Tweet*

#print(tweet["text"],tweet["created\_at"])

*# Store Tweet in Array*

tweet\_texts.append([tweet["id"],tweet["created\_at"],tweet["text"],tweet["favorite\_count"],tweet["retweet\_count"]])

Through the loop, our group was able to capture 3,229 total Tweets. These tweets, Musk’s most recent, were loaded into a dataframe and columns were assigned. This data was the exported to a csv, although this csv was ultimately abandoned given its lack of relevance to the traders’ initial inquiries. However, the tweet data frame was further transformed in an effort to create a table that would better serve the traders’ interests. The additional transformations will be covered in the next section.

# Transform

Utilizing the Python library Pandas, inside of Jupyter Notebook, our first step was to import the newly extracted csvs for the stock prices and the SpaceX launch data. Below are the codes executed to perform these actions:

spacex\_file = "Resources/spacex\_launch\_data.csv"

teslatwo\_file="TSLA \_to\_29th.csv"

spacex\_df = pd.read\_csv(spacex\_file)

teslatwo\_df=pd.read\_csv(teslatwo\_file)

Now that our data was read into two independent Pandas Dataframes, our team then began a fairly straightforward transformation process. For SpaceX, the transformation included limiting the specific columns of data that were relevant for answering our equity traders’ initial questions. We then renamed the columns to correspond properly to the database that we eventually exported our data into (more on this topic in the next section). Our next step was to then perform a drop on any duplicate data that existed and lastly to set the index of the table to the id that was created. Once the transformation was finished, our team pushed the new dataframe to a csv file. Below is the code executed to perform these transformation steps and obtain our SpaceX final table:

spacex\_cols = ["Flight Number", "Date","Launch Site", "Payload", "Customer", "Mission Outcome", "Landing Outcome"]

spacex\_transform = spacex\_df[spacex\_cols].copy()

spacex\_transform = spacex\_transform.rename(columns={"Flight Number":"id",

"Date":"real\_date",

"Launch Site": "launch\_site",

"Payload":"payload",

"Customer":"customer",

"Mission Outcome":"mission\_outcome",

"Landing Outcome":"landing\_outcome"})

spacex\_transform.drop\_duplicates("id", inplace=True)

spacex\_transform.set\_index("id", inplace=True)

spacex\_transform.head()

The cleanup and transformation on the Tesla stock data was slightly more involved. Similar steps performed with the SpaceX dataframe were copied over for the Tesla stock data. Two additional columns were created and added to the dataframe from the remaining data to further add value to the equity traders’ future analysis.

The first column was the calculated range for a given day. This range, the low price for the given day subtracted from the high price for that same given day, was added to better illustrate to the equity team the intraday movement of the underlying stock price. The thought process behind building this column was that the traders would need a quick reference column to perform analysis on that would measure the intraday activity of Tesla stock. Additionally, to better approximate how the intraday movement related to the actual underlying price of Tesla stock, we calculated a “Range percent”, which was the range divided by the closing price. That value was then multiplied by one hundred to get a metric that would reside between zero and one hundred. The logic behind the calculated indicator is the greater the value in the range percent column, the more volatile that particular trading day was. This would allow our traders to attempt to answer if a given trading day was more or less volatile given a particular tweet by Elon Musk or a Spacex related event. The code for these two additional columns is below:

tesla\_transform["range"] = tesla\_transform["high\_price"].subtract(tesla\_transform["low\_price"], fill\_value=0)

tesla\_transform["rng\_perct\_close"] =(tesla\_transform["range"]/tesla\_transform["closing\_price"])\*100

Following the creation of these two data columns, the dataframe was ready for export and was pushed to a new csv file for ultimate loading into our database.

As mentioned in the extraction portion, the Twitter data needed additional transformations past loading it into a dataframe and changing column headings. The first, additional transformation was creating a column “Popular Sum” that would be comprised of the tweet column “Favorite Count” and “Retweet Count”. The purpose was to generate a singular measure of the popularity of a given Musk tweet. Following this transformation, our group added a date column utilizing the datetime library. Given that Musk often tweets several times a day, we had a potential problem in that our table included a number of tweets for the same day. Given that our traders were only interested in the most potentially impactful tweets, and not the lesser popular ones, we performed a sort on the dataframe by Date and Popular Sum, from most recent, and then dropped any duplicate dates. The code for these essential transformations is below:

tweet\_ETL\_df["Popular\_Sum"] = tweet\_ETL\_df["Favorite\_Count"] + tweet\_ETL\_df["Retweet\_Count"]

tweet\_ETL\_dt\_df['Date']=tweet\_ETL\_df.Tweet\_Date=pd.to\_datetime(tweet\_ETL\_df.Tweet\_Date).dt.strftime('%Y-%m-%d')

sorted\_df = tweet\_ETL\_dt\_df.sort\_values(['Date','Popular\_Sum'], ascending=False)

sorted\_distinct = sorted\_df.drop\_duplicates(['Date'])

This process left us with a table that highlighted Musk’s tweets, on singular specific days and displayed his most impactful tweets based on the Twitter communities liking and retweeting behavior. This was the table our traders needed and was then exported to a new csv, ultimately to be loaded into our SQL database.

# Load

For the loading portion of this process, our team chose to use MySQL and the MySQL client workbench as the preferred database for which to load in the transformed data. Our team chose this database, in part, because of our familiarity with its relational structure. Furthermore, we felt that our equity team would be able to better visualize the data in the SQL tables that were generated.

As previously mentioned in the above section, our team created three tables containing data on tweets, space launches and stock prices. As part of the cleanup, we renamed column headings. This was done to better align the data with the column headings that were used in the tables that were created in MySQL. Clarity during the importing process was paramount. We constructed three frameworks in total and a sample of the SpaceX MySQL skeleton is below:

|  |
| --- |
| CREATE TABLE SpaceX( |
|  | id INT NOT NULL, |
|  | real\_date DATE, |
|  | launch\_site VARCHAR(100), |
|  | payload VARCHAR(100), |
|  | customer VARCHAR(100), |
|  | mission\_outcome VARCHAR(100), |
|  | landing\_outcome VARCHAR(100), |
|  | Primary Key (id) |
|  | ); |

After the framework of each table was complete, our group utilized the MySQL Workbench import wizard to carefully sift through the data and import in the transformed csv files to our database. The import process was extremely smooth for the Tesla stock table as well as the SpaceX table. However, a problem arose during the importing of the tweet data because the database was having a difficult time handling the emoji characters that were in a few of the tweets. A workaround solution was created where the characters were modified and set to “Utf8mb4” as a format. This solved some of the issue, however, there were still several rows of tweet data that were not importing. The simple solution, at that point, was to manually find and replace the double quotations that were added from the csv extraction from pandas. Additionally, our group took the step of manually inserting several rows of data, seven in total, because of the emoji characters. A sample of such inserting is below:

|  |  |
| --- | --- |
| insert into elon\_pop  values(971224396890124288,"I just realized there is a jazz hands emoji 🤗",150523,16500,167023,"2018-03-07"); | |
|  | |
|  |

Following the manual insertion, the tweet table began to cooperate and Select statements from each of the tables were possible and functional. All tables were created with column headings that would appear obvious and intuitive to any data user that came along. It was felt that clarity, above all else, would rank highest in importance. Especially given that stock information is fairly straightforward to begin with, the other tables were labeled to be as intuitive as possible.

Of significant importance, all tables had a common column of “Date” constructed within them. This was a critical feature of our tables because it allowed the relational power of MySQL to be realized. Joins were possible with all three tables on date, the unique identifying key. This would allow the major analysis work to be performed by the equity team.

Our group chose not to forcefully join the tables at this time because we decided that the tables, independently, should be first analyzed by the equity team. They were powerful enough on their own and answered questions the equity team may have within those subject matters. However, we did provide the equity trade team with the query statement that would perform the triple join. Once that join was complete, the equity team could perform various queries to answer all the questions initially posed. Queries to analyze the specific tweet that corresponded with a very volatile price move or with a given launch outcome would all be possible on this newly joined super table.