0 Food for Thought

What is the goal of the overall project? How does it relate to *Machine Learning*?
What *technologies* does the project require to achieve its goals? What is the purpose of each?
How do we install the required dependencies?
How do we access *Environmental Variables* in Python?
How do we add *command-line arguments* to a Python program using *ArgParse*?
How do you connect to the *Twitter API* using *Tweepy*?
How do you send text to *Google Cloud Natural Language API* and receive back a sentiment score?
How do you submit a *SPARQL* query to *Wikidata FreeBase Database* and receive a response?
How do you create a multi-thread *StreamListener* from the *Tweepy* library? How does *StreamListener* work?

1 Introduction

This project introduces the concepts of *natural language processing* using *Google Cloud Natural Language API* in order to analyze the sentiment of tweets from Twitter containing a reference to a given stock market ticker symbol, for example *TSLA* or a given set of keywords relating to the company in question. When enough tweets have been captured and analyzed, a *net-sentiment analysis* could later be performed in order to determine the overall sentiment of a given company at a given time. However, given this projects complexity and length, this extra layer is beyond the scope of this lesson.

2 Technologies

This project will utilize several technologies for the *Python* programming language in order to achieve its desired goals. First, we will utilize the *Twitter API* in order to quickly access tweets in real time or from Twitter's archives. These tweets will be analyzed by another technology, *Google Cloud Natural Language API*, which will analyze the individual tweet's sentiment. Finally, we will use the *Wikidata Query Service* in order to access information on the company data based on its ticker symbol found within the tweet. In order to make this project easier and simpler to use, we will also use *ArgParse* for adding command-line arguments and processing to the overall *stockticker* program.

3 Setup Authentication

3.1 Twitter

Log in to your Twitter account and create a new application. Under the Keys and Access Tokens tab for your app you'll find the Consumer Key and Consumer Secret. Export both to environment variables:

```
export TWITTER_CONSUMER_KEY="<YOUR_CONSUMER_KEY>"
export TWITTER_CONSUMER_SECRET="<YOUR_CONSUMER_SECRET>"
```

If you want the tweets to come from the same account that owns the application, simply use the Access Token and Access Token Secret on the same page. If you want to tweet from a different account, follow the steps to obtain an access token. Then export both to *environment variables*:

```
export TWITTER_ACCESS_TOKEN="<YOUR_ACCESS_TOKEN>"
export TWITTER_ACCESS_TOKEN_SECRET="<YOUR_ACCESS_TOKEN_SECRET>"
```

3.2 Google

Follow the Google Application Default Credentials instructions to create, download, and export a service account key.

```
export GOOGLE_APPLICATION_CREDENTIALS="/path/to/credentials-file.json"
```

You also need to enable the Cloud Natural Language API for your Google Cloud Platform project.

4 Install dependencies

There are a few library dependencies, which you can install using pip:

```
$ pip3 install -r requirements.txt
```

The contents of *requirements.txt*:

```
google
google-cloud-language
tweepy
py-dotenv
requests
```

5 Setup `.env`

.env is the file used to store API keys

```
TWITTER_CONSUMER_KEY=
TWITTER_CONSUMER_SECRET=
TWITTER_ACCESS_TOKEN=
TWITTER_ACCESS_TOKEN_SECRET=
GOOGLE_APPLICATION_CREDENTIALS=
```

6 How *tickerminer.py* will be used

The desired usage of the final program *stockmine.py* will be used in the following ways:

```
$ python3 tickerminer.py --keywords TSLA,'Elon Musk',Musk,Tesla,SpaceX
```

The program will take a parameter *keywords*. It will then use that set of words to prob tweets using the *Twitter API* for any tweet that contains those keywords. If one or many of the keywords exists, then it is analyzed for its sentiment and develops a score between -1 and 1.

7 Accessing *Environmental Variables* contained within `.env`

```
# Read API keys
try:
    read_dotenv(os.path.join(os.path.dirname(__file__), '.env'))
except FileNotFoundError:
    print("\n'.env' does not exist. Please create the file & add the necessary API keys to it.")
```

```
    exit(1)

# The keys for the Twitter app we're using for API requests
# (https://apps.twitter.com/app/13239588). Read from environment variables.
TWITTER_CONSUMER_KEY = getenv('TWITTER_CONSUMER_KEY')
TWITTER_CONSUMER_SECRET = getenv('TWITTER_CONSUMER_SECRET')

# The keys for the Twitter account we're using for API requests.
# Read from environment variables.
TWITTER_ACCESS_TOKEN = getenv('TWITTER_ACCESS_TOKEN')
TWITTER_ACCESS_TOKEN_SECRET = getenv('TWITTER_ACCESS_TOKEN_SECRET')
```

The above snippet allows us to access the environmental variables from `.env`.

8 Parsing CLI Arguments using *ArgParse*

Since this program, *tickerminer* takes an argument *keywords*, we must provide a means for the program to handle this logic. To do so, we will use the Python library *ArgParse* like so:

```
if __name__ == "__main__":
    # parse CLI arguments
    parser = argparse.ArgumentParser()
    parser.add_argument("--keywords", metavar="KEYWORDS",
     help="Use keywords to search for in Tweets instead of feeds. "
          "Separated by commas, case insensitive, spaces are ANDs commas are ORs. "
          "Example: TSLA,'Elon Musk',Musk,Tesla,SpaceX")
    args = parser.parse_args()

    # Print help if no arguments are given.
    if len(sys.argv) == 1:
        parser.print_help()
```

9 Handling CLI argument *--keywords*

```
# python3 tickerminer.py --keywords <KEYWORDS>
if args.keywords:
    twitter = Twitter()
    twitter.start_streaming(args, twitter_callback)
```

9.1 twitter_callback()

This is the main wrapper function for handling each step of the process to analyze a given tweet.

```
def twitter_callback(tweet):
    """Analyzes tweets"""

    # Start analysis.
```

```
    analysis = Analysis()

    # strip out hash-tags for language processing.
    text = re.sub(r"[#|@$]\S+", "", tweet['text']).strip()
    tweet['text'] = text

    # Find any mention of companies in tweet.
    companies = analysis.find_companies(tweet)

    # if no company's are found, skip.
    if not companies:
        return

    # Analyze a tweet & obtain its sentiment.
    results = analysis.analyze(companies)

    print(results)
```

10 Creating *analysis.py*

analysis.py is used to analyze a given tweet using *Google Cloud Natural Language API* as well as the *Wikidata Query Service* for finding a given company within the tweet. *language_client* is the language service client to the *Google Cloud Natural Language API*, which is used to send text for sentiment analysis.

```
class Analysis:
    """A helper for analyzing company data in text."""

    def __init__(self):
        self.language_client = language.LanguageServiceClient()
```

The function `find_companies()` is used to find any mention of a given company within the body of a given tweet. This function utilizes the *Wikidata Query Service* in order to match a given Freebase ID (mid) to a given company's ticker symbol. It then returns the dictionary of companies and their data along with their respective tweet the company is associated with.

```
    def find_companies(self, tweet):
        """Finds any mention of companies in a tweet."""

        if not tweet:
            return None

        text = tweet['text']
        if not text:
            return None

        # Run entity detection.
        document = language.types.Document(
            content=text,
```

```
            type=language.enums.Document.Type.PLAIN_TEXT,
            language="en"
        )
        entities = self.language_client.analyze_entities(document).entities
        # print("Found entities: %s" % entities_tostring(entities))

        # Collect all entities which are publicly traded companies, i.e.
        # entities which have a known stock ticker symbol.
        companies = []

        for entity in entities:
            # Use the Freebase ID of the entity to find company data. Skip any
            # entity which doesn't have a Freebase ID (unless we find one via
            # the Twitter handle).
            name = entity.name
            metadata = entity.metadata
            try:
                mid = metadata["mid"]
            except KeyError:
                continue

            company_data = get_company_data(mid)

            # Skip any entity for which we can't find any company data.
            if not company_data:
                continue

            for company in company_data:
                # Append & attach metadata associated with a company.
                company["tweet"] = text
                company["url"] = get_tweet_link(tweet)

                print("Examining tweet: %s" % company['tweet'])

                # Add to the list unless we already have the same entry.
                names = [existing["name"] for existing in companies]
                if not company["name"] in names:
                    companies.append(company)

                break

        return companies
```

The function `analyze()` is used to *analyze* the given dictionary of companies returned from `find_companies()` in order to extract (with help from the `extract_sentiment()` function) and append the sentiment score to the results dictionary. This is part of the data cleansing process.

```
    def analyze(self, companies):
        """Attach a sentiment score to each company found."""
```

```
        results = {}

        for company in companies:
            # Extract and add a sentiment score.
            sentiment = self.extract_sentiment(company['tweet'])
            results[company['symbol']] = {'sentiment': sentiment}

        return results
```

The function `extract_sentiment()` is utilized to find and return the sentiment score from a given tweet using the *Google Cloud Natural Language API*.

```
    def extract_sentiment(self, text):
        """Extracts a sentiment score [-1, 1] from text using Google Natural Language API."""

        if not text:
            return 0

        document = language.types.Document(
            content=text,
            type=language.enums.Document.Type.PLAIN_TEXT,
            language="en"
        )
        sentiment = self.language_client.analyze_sentiment(document).document_sentiment

        return sentiment.score
```

11 Creating *wiki.py*

wiki.py is used to fetch a given company's ticker data from Wikidata *Query Service*. The *MID_TO_TICKER_QUERY* was constructed with the help of the *StackOverFlow* answer found here. More information about *SPARQL* can be found at the following resources:

SPARQL query: https://www.w3.org/TR/sparql11-query/

Constructing SPARQL queries: https://medium.com/wallscope/constructing-sparql-queries-ca63b8b9ac02

***11.1 WIKIDATA_QUERY_URL* & MID_TO_TICKER_QUERY**

```
# The URL for a GET request to the Wikidata API. The string parameter is the
# SPARQL query.
WIKIDATA_QUERY_URL = "https://query.wikidata.org/sparql?query=%s&format=JSON"

# A Wikidata SPARQL query to find stock ticker symbols and other information
# for a company. The string parameter is the Freebase ID of the company.
MID_TO_TICKER_QUERY = (
    'SELECT ?companyLabel ?rootLabel ?tickerLabel ?exchangeNameLabel'
```

```
    ' WHERE {'
    '  ?entity wdt:P646 "%s" .'  # Entity with specified Freebase ID.
    '  ?entity wdt:P176* ?manufacturer .'  # Entity may be product.
    '  ?manufacturer wdt:P1366* ?company .'  # Company may have restructured.
    '  { ?company p:P414 ?exchange } UNION'  # Company traded on exchange ...
    '  { ?company wdt:P127+ / wdt:P1366* ?root .'  # ... or company has owner.
    '    ?root p:P414 ?exchange } UNION'  # Owner traded on exchange or ...
    '  { ?company wdt:P749+ / wdt:P1366* ?root .'  # ... company has parent.
    '    ?root p:P414 ?exchange } .'  # Parent traded on exchange.
    '  VALUES ?exchanges { wd:Q13677 wd:Q82059 } .'  # Whitelist NYSE, NASDAQ.
    '  ?exchange ps:P414 ?exchanges .'  # Stock exchange is whitelisted.
    '  ?exchange pq:P249 ?ticker .'  # Get ticker symbol.
    '  ?exchange ps:P414 ?exchangeName .'  # Get name of exchange.
    '  FILTER NOT EXISTS { ?company wdt:P31 /'  # Exclude newspapers.
    '                              wdt:P279* wd:Q11032 } .'
    '  FILTER NOT EXISTS { ?company wdt:P31 /'  # Exclude news agencies.
    '                              wdt:P279* wd:Q192283 } .'
    '  FILTER NOT EXISTS { ?company wdt:P31 /'  # Exclude news magazines.
    '                              wdt:P279* wd:Q1684600 } .'
    '  FILTER NOT EXISTS { ?company wdt:P31 /'  # Exclude radio stations.
    '                              wdt:P279* wd:Q14350 } .'
    '  FILTER NOT EXISTS { ?company wdt:P31 /'  # Exclude TV stations.
    '                              wdt:P279* wd:Q1616075 } .'
    '  FILTER NOT EXISTS { ?company wdt:P31 /'  # Exclude TV channels.
    '                              wdt:P279* wd:Q2001305 } .'
    '  SERVICE wikibase:label {'
    '   bd:serviceParam wikibase:language "en" .'  # Use English labels.
    '  }'
    ' } GROUP BY ?companyLabel ?rootLabel ?tickerLabel ?exchangeNameLabel'
    ' ORDER BY ?companyLabel ?rootLabel ?tickerLabel ?exchangeNameLabel')
```

11.2 `make_wikidata_request()`

This function simply makes a request to the *Wikidata SPARQL API* and returns the results to its caller. It is the first step in the process of obtaining ticker information on a given company as it is used in the next function, `get_company_data()`.

```
def make_wikidata_request(query):
    """Makes a request to the Wikidata SPARQL API."""

    query_url = WIKIDATA_QUERY_URL % quote_plus(query)

    response = get(query_url)

    try:
        response_json = response.json()
    except ValueError:
        return None
```

```
    try:
        results = response_json["results"]
        bindings = results["bindings"]
    except KeyError:
        return None

    return bindings
```

11.3 `get_company_data()`

This function looks up the stock ticker information for a company via its *Freebase ID*, which is an identifier for a page in the *Freebase database*. The Freebase database is part of the *Wikidata* database and utilizes *SPARQL* as the query language to query against it. The function returns the stock ticker information for a company.

```
def get_company_data(mid):
    """Looks up stock ticker information for a company via its Freebase ID (MID)."""

    query = MID_TO_TICKER_QUERY % mid
    response = make_wikidata_request(query)

    if not response:
        return None

    # Collect the data from the response.
    companies = []
    for data in response:
        try:
            name = data["companyLabel"]["value"]
        except KeyError:
            name = None

        try:
            root = data["rootLabel"]["value"]
        except KeyError:
            root = None

        try:
            symbol = data["tickerLabel"]["value"]
        except KeyError:
            symbol = None

        try:
            exchange = data["exchangeNameLabel"]["value"]
        except KeyError:
            exchange = None

        company = {"name": name,
                   "symbol": symbol,
                   "exchange": exchange}

        # Add the root if there is one.
        if root and root != name:
            company["root"] = root

        # Add to the list unless we already have the same entry.
        if company not in companies:
            companies.append(company)

    return companies
```

12 Creating *twitter.py*

twitter.py is used to authenticate and retrieve tweets from Twitter using *Tweepy*. Of course, this is a seperate python

program which requires access to our *Environment Variables*, and thus we must also read in the *Environment Variables* from the `.env` file like we did in **section 7** before defining `class Twitter`.

```
class Twitter:
    """A helper for talking to Twitter APIs."""

    def __init__(self):
        self.twitter_auth = OAuthHandler(TWITTER_CONSUMER_KEY,
                                         TWITTER_CONSUMER_SECRET)
        self.twitter_auth.set_access_token(TWITTER_ACCESS_TOKEN,
                                           TWITTER_ACCESS_TOKEN_SECRET)

        self.twitter_api = API(auth_handler=self.twitter_auth,
                               retry_count=60,
                               retry_delay=1,
                               retry_errors=[400, 401, 500, 502, 503, 504],
                               wait_on_rate_limit=True,
                               wait_on_rate_limit_notify=True)
```

12.1 `start_streaming()`

This function creates a stream of tweets, where each tweet matches the set of keywords passed from the command-line argument *--keywords*.

```
    def start_streaming(self, args, callback):
        """Starts streaming tweets and returning data to the callback."""

        self.twitter_listener = TwitterListener(callback=callback)

        twitter_stream = Stream(self.twitter_auth, self.twitter_listener)

        if args.keywords:
            try:
                # Search for tweets containing a list of keywords.
                keywords = args.keywords.split(',')

                print("Searching for tweets containing %s" % keywords)
                twitter_stream.filter(track=keywords, languages=['en'])
            except TweepError:
                print("Twitter API error %s" % TweepError)
            except KeyboardInterrupt:
                twitter_stream.disconnect()
                sys.exit(0)
```

13 Creating *twitterlistener.py*

twitterlistener.py contains the class *TwitterListener* which extends from the parent class *StreamListener* from *Tweepy*. In *Tweepy*, an instance of *tweepy.Stream* establishes a streaming session and routes messages to *StreamListener* instance. The `on_data` method of a stream listener receives all messages and calls functions according to the message type. The default

StreamListener can classify most common twitter messages and routes them to appropriately named methods, but these methods are only stubs.

Therefore using the streaming api has three steps.

1. Create a class inheriting from StreamListener
2. Using that class create a Stream object
3. Connect to the Twitter API using the Stream.

13.1 Creating a StreamListener

This simple stream listener prints status text. The on_data method of *Tweepy's StreamListener* conveniently passes data from statuses to the on_status method. Create class *TwitterListener* inheriting from *StreamListener* and overriding `on_status`. Additionally, we connect the `tweet_callback` function from the *stockticker.py* program so that way we can sanitize the data prior to passing it off to the main tweet handler function. Since this is a multi-thread application, we will utilize a queue for handling each thread.

```
class TwitterListener(StreamListener):
    def __init__(self, callback):
        super().__init__()

        # The tweet_callback function defined in stockticker.py
        self.callback = callback
        self.error_status = None
        self.start_queue()
```

13.2 `start_queue()`

The function `start_queue()` creates a queue and starts each worker thread.

```
    def start_queue(self):
        """Creates a queue and starts the worker threads."""

        self.queue = Queue()
        self.stop_event = Event()
        self.workers = []
        for worker_id in range(1):
            worker = Thread(target=self.process_queue, args=[worker_id])
            worker.daemon = True
            worker.start()
            self.workers.append(worker)
```

13.2 `stop_queue()`

The function `stop_queue()` shuts down the queue and each respective worker threads gracefully by setting the `stop_event` and joining each individual thread until all threads have been gracefully terminated.

```
def stop_queue(self):
```

```
        """Shuts down the queue and worker threads."""

        # First stop the queue.
        if self.queue:
            self.queue.join()

        # Then stop the worker threads.
        if self.workers:
            self.stop_event.set()
            for worker in self.workers:
                # Block until the thread terminates.
                worker.join()
```

13.3 `process_queue()`

The function `process_queue()` continuously processes tasks within the queue until the `stop_event` has been set.

```
def process_queue(self, worker_id):
        """Continuously processes tasks within the queue."""

        while not self.stop_event.is_set():
            try:
                data = self.queue.get(block=True, timeout=1)
                self.handle_data(data)
                self.queue.task_done()
            except Empty:
                continue
```

13.4 `on_error()`

The function `on_error()` handles and API errors that arises when a worker thread processes its data.

```
    def on_error(self, status):
        """Handles any API errors."""

        self.error_status = status
        self.stop_queue()
        return False
```

13.5 *`get_error_status()`*

The function `get_error_status()` returns the API error_status, if there was one.

```
    def get_error_status(self):
        """Returns the API error status, if there was one."""

        return self.error_status
```

13.6 `on_data()`

The function `on_data()` puts a task to process the new data onto the queue of worker threads, only stopping if the `stop_event` has been set.

```
    def on_data(self, data):
        """Puts a task to process the new data on the queue."""

        # Stop streaming if requested.
        if self.stop_event.is_set():
            return False

        # Put the task on the queue and keep streaming.
        self.queue.put(data)
        return True
```

13.7 `handle_data()`

The function `handle_data()` preforms sanity-checks and extracts the data before sending it to the `tweet_callback` function.

```
    def handle_data(self, data):
        """Sanity-checks and extracts the data before sending it to the
        callback.
        """

        try:
            tweet = loads(data)
        except ValueError:
            return

        # Call the callback.
        self.callback(tweet)
```

14 Conclusion

At the outset of this lesson, we introduce the concepts of *natural language processing* using *Google Cloud Natural Language API* in order to analyze the sentiment of tweets from Twitter containing a reference to a given stock market ticker symbol, for example *TSLA* or a given set of keywords relating to the company in question. We learned how to use the Python libraries, *ArgParse* for command-line argument handling, *Tweepy* for accessing the *Twitter API*, *py-dotenv* for handling `.env` files and *Environmental Variables*, *google-cloud-language* for connecting to the *Google Cloud Natural Language API* and extracting a sentiment score of a given text. Looking to the future, when enough tweets have been captured and analyzed, a net-sentiment analysis could later be performed in order to determine the overall sentiment of a given company at a given time. This can be really useful statistic in not only understanding how the broader market interprets the actions of a given company, but could be a means to choosing a company to invest in the stock market.