**Aggressive trading strategy using options**

Here, we’ll use a simple options strategy: buying call options when anticipating a bullish move and put options for bearish moves. We will use sentiment analysis on news and social media to help predict these moves.

For simplicity, let’s assume we are using a broker with a Python API such as Alpaca. Note that the exact code will depend on the specifics of your broker’s API:

pip install alpaca-trade-api

import alpaca\_trade\_api as tradeapi

# Initialize the Alpaca API

api = tradeapi.REST('<APCA-API-KEY-ID>', '<APCA-API-SECRET-KEY>',

base\_url='https://paper-api.alpaca.markets')

# Define the stock symbol symbol = 'TSLA'

contract = api.get\_option\_contracts(symbol)

# Function to buy a call option

def buy\_call(api, symbol, contract): order = api.submit\_order(

symbol=symbol,

10 Tesla’s Financial Journey: AI Analysis and Bias Unveiled

qty=1, side='buy', type='limit',

time\_in\_force='gtc', limit\_price=contract.ask\_price

)

print(f"Call option order submitted. ID: {order.id}")

# Function to buy a put option

def buy\_put(api, symbol, contract): order = api.submit\_order(

symbol=symbol, qty=1, side='buy', type='limit',

time\_in\_force='gtc', limit\_price=contract.bid\_price

)

print(f"Put option order submitted. ID: {order.id}")

# Example usage

buy\_call(api, symbol, contract) buy\_put(api, symbol, contract)

**IMPORTANT**

Replace <APCA-API-KEY-ID> and <APCA-API-SECRET-KEY> with your actual Alpaca API key and secret.

Let’s look at what this code does:

* **Imports Alpaca trade API**: Alpaca is an online brokerage that offers a modern platform for trading and has its own Python library that allows you to interact with their platform programmatically. We start by importing this library, which is a collection of code that we can use to do things such as place trades.
* **Defines the Alpaca API**: Here, we are connecting to Alpaca’s trading platform by using their API. This is like establishing a secure line of communication between our Python code and Alpaca’s trading services.
* **Defines the option contract**: An option contract represents 100 shares of a stock. Defining the option contract involves specifying details such as the stock (Tesla, in our case), the price at which you have the right to buy or sell the stock (the strike price), and the date when the option expires.
* **Buys a call option**: A call option gives us the right (but not the obligation) to buy the stock at the strike price. This is what we do when we anticipate that Tesla’s stock price will go up. We’re effectively betting on a bullish market movement.
* **Buys a put option**: Conversely, a put option gives us the right (but not the obligation) to sell the stock at the strike price. We do this when we expect Tesla’s stock price to go down or when we’re anticipating a bearish market movement.

To decide whether we should buy a call or put option, we are using sentiment analysis on news and social media as well as earnings call transcripts. Sentiment analysis involves using algorithms to determine whether the sentiment toward Tesla in news articles and social media posts is positive or negative. If the sentiment is bullish (positive), we might buy a call option. If it’s bearish (negative), we might buy a put option.

It’s important to remember, though, that this is a simplified overview; actual trading involves more complexity and risk. Always ensure that you understand a strategy fully and consider consulting a financial advisor before implementing it.

**Conservative trading strategy using position trading**

For a more conservative strategy, we can use a simple long position strategy where we buy shares of Tesla when certain fundamental conditions are met. We will assume that the alpaca-trade-api package is not installed in the following example. If you have already installed it, please remove the first line in this Python code example:

pip install alpaca-trade-api

import alpaca\_trade\_api as tradeapi

# Initialize the Alpaca API

api = tradeapi.REST('<Your-API-Key>', '<Your-Secret-Key>', base\_ url='https://paper-api.alpaca.markets')

# Define the stock symbol symbol = 'TSLA'

try:

# Place a buy order api.submit\_order(

symbol=symbol, qty=1, side='buy', type='market',

time\_in\_force='day'

12 Tesla’s Financial Journey: AI Analysis and Bias Unveiled

)

# Place a sell order api.submit\_order(

symbol=symbol, qty=1, side='sell', type='market',

time\_in\_force='day'

)

# List current positions positions = api.list\_positions() for position in positions:

print(f"{position.symbol} {position.qty}")

except Exception as e:

print(f"An error occurred: {e}")

* **API key and secret key**: Ensure you replace <Your-API-Key> and <Your-Secret-Key>

with the actual Alpaca API key and secret key.

* **Market orders**: The code is currently placing market orders. Make sure the market is open when you run this code, or else the orders may not execute.
* **time-in-force**: You’ve set time\_in\_force to gtc (good-til-cancelled). This is fine for limit orders, but for market orders, you may consider using ‘day’ to specify that the order is only valid during the trading day.
* **Error handling**: The code does not include any error handling. You might want to add some try-except blocks.

The following is an explanation of this Python code snippet:

* **Imports Alpaca trade API**: This code will import Alpaca’s library into your Python script.
* **Establishes Alpaca API connection**: This will set up a connection to Alpaca’s API using your personal API keys.
* **Defines the stock**: This is where you specify the stock you’re interested in trading (Tesla, in this case).
* **Buys Tesla shares**: When you’re ready to buy shares, you can place an order using Alpaca’s submit\_order function. Here’s how you could place a simple market order to buy one share of Tesla:

api.submit\_order( symbol=symbol,

Case study: Tesla Inc. 13

qty=1, side='buy', type='market',

time\_in\_force='day'

* **Sells Tesla shares**: Similarly, when you’re ready to sell shares, you could use the same submit\_ order function:

api.submit\_order( symbol=symbol, qty=1, side='sell', type='market',

time\_in\_force='day'

)

* print: Lastly, the print statement is used to output the result of a particular action in your code. For instance, if you want to print the list of your current positions, you could use the list\_positions function and print it as follows:

positions = api.list\_positions() for position in positions:

print(f"{position.symbol} {position.qty}")

Keep in mind that this code is illustrative and assumes that you have API keys from Alpaca. Always remember to protect your API keys and don’t share them with others.

In the next section, we will highlight how to pull in news and earnings call transcript data to help pull in data that can be reviewed by ChatGPT to determine sentiment (positive, neutral, or negative). This can signal whether or not we should buy, hold, or sell based on the sentiment indicators.

**News and market sentiment integration for trading strategies: aggressive and conservative**

In this section, we will delve into the pivotal role of news and market sentiment in shaping intelligent trading strategies and learning how to interpret and integrate real-time data and market indicators into investment decisions. You will master the art of leveraging cutting-edge tools and analytics to predict market trends, understand investor behaviors, and enhance your trading performance and financial acumen.

This section will provide a step-by-step process:

1. Install libraries newsapi and Beautifulsoup4.
2. Use an API from NewsAPI to fetch Tesla news articles and BeautifulSoup to pull the Tesla earnings transcripts.
3. Perform sentiment analysis with ChatGPT (including comments in articles and Q&A in earnings transcripts).
4. Save the data into a CSV file(s).
5. Import the data into Power BI.
6. Create visualizations to use for your Tesla aggressive and conservative trading decisions.

Using a Python library called NewsAPI to get news data and a library called TextBlob for simple sentiment analysis, NLTK library. or even pre-trained models from transformers library (such as BERT or GPT-3.5) to evaluate sentiment from news and social media. For the integration of the data into Power BI, we will outline a simple method using CSV files. Let’s dive into these steps in detail:

* 1. Install the required libraries.

In Python, you need to install the following libraries using pip:

**pip install newsapi-python pip install requests**

* 1. Get Tesla news and earnings call data from Tesla articles and Tesla earnings call transcripts. To extract news articles about Tesla, use the following code:

from newsapi import NewsApiClient

# Initialize the News API client

newsapi = NewsApiClient(api\_key='your-newsapi-key')

try:

# Fetch news articles related to Tesla all\_articles = newsapi.get\_everything(q='Tesla',

from\_param='2022-10-

01',

to='2022-12-31',

sort\_by='relevancy')

# Display articles

for article in all\_articles['articles']: print(article['title'], article['url'],

article['content'])

except Exception as e:

print(f"An error occurred: {e}")

IMPORTANT: Replace 'your-newsapi-key' with your actual News API key.

The date range above requires a paid membership to obtain all the Tesla news articles from 2022-10 to 2022-12-31. See the following instructions to access the paid membership at the News API [website](https://newsapi.org/).

1. Navigate to the News API website. Go to News API and click on **Get API key**.
2. Sign up or log in. If you don’t have an account, you’ll need to create one. If you already have an account, log in.
3. Select a plan. News API offers multiple plans, including a free tier with limited access and paid memberships for more comprehensive access. In this case, a paid plan will be required to run the Python code shown previously.
4. If you’ve selected a paid plan, you’ll be prompted to enter your payment information.
5. Once the account setup is complete, you’ll be provided with an API key. This is what you’ll use to access the service programmatically.

For earnings call transcripts, let’s use the Financial Modeling Prep API as an example. First, we can use it to extract the pag[e content and then parse the da](https://financialmodelingprep.com/developer/docs/)ta:

* 1. Navigate to the Financial Modeling Prep website.
  2. Register or log in. If you are new, you’ll need to create an account. If you already have an account, log in.
  3. Choose a plan. Go to the pricing section and choose Ultimate Plan. Follow the steps for payment to activate your subscription.
  4. Once your account is set up and the subscription is active, go to your dashboard to generate an API key:

import requests import json

# Initialize API endpoint and API key api\_key = "your\_api\_key\_here"

api\_endpoint = f"https://financialmodelingprep.com/api/v3/your\_ endpoint\_here?apikey={api\_key}"

# Payload or parameters for date range (Modify as per actual API documentation)

params = {

"from": "2022-10-01",

"to": "2022-12-31"

}

try:

# Make the API request

16 Tesla’s Financial Journey: AI Analysis and Bias Unveiled

response = requests.get(api\_endpoint, params=params) response.raise\_for\_status()

# Parse the JSON data

data = json.loads(response.text)

# Extract and print the data (Modify as per actual API response)

# For demonstration, assuming data is a list of dictionaries with a 'transcript' key

for item in data:

print(item.get("transcript", "Transcript not available"))

except Exception as e:

print(f"An error occurred: {e}")

Replace "your\_api\_key\_here" and "your\_endpoint\_here" with your actual API key and the API endpoint you’re interested in. Also, adjust the params according to the API’s actual documentation.

**Note**

The provided Python code is a general template and may not work out of the box due to API- specific requirements and data structures. Always refer to the API documentation for accurate and up-to-date information.

As for parsing the Q&A part and comments, the structure of the HTML would dictate how to isolate that section. If it’s consistently structured across transcripts, you could simply adjust your selectors to grab that specific part of the page.

Here’s a Python code snippet that assumes you have the earnings call transcript in a string format. It looks for the line where Martin Viecha, VP of investor relations at Tesla, announces the start of the Q&A section in the earnings call. Then, it separates investor questions from the answers from the management team in the earnings call transcript:

def parse\_transcript(transcript):

lines = transcript.split('\n') # Assume the transcript uses newline characters to separate lines

in\_qa\_section = False questions = [] answers = [] current\_q = "" current\_a = ""

for line in lines:

# Check if the Q&A section starts

if "Martin Viecha" in line and "investor question" in line.lower():

in\_qa\_section = True

continue # Skip this line and move to the next line

if in\_qa\_section:

# Assume that a line starting with "Q:" signifies a

question

if line.startswith("Q:"):

# Save the previous Q&A pair before moving on to

the next question

if current\_q and current\_a: questions.append(current\_q.strip()) answers.append(current\_a.strip())

current\_q = line[2:].strip() # Skip "Q:" and

save the rest

current\_a = "" # Reset the answer string

else:

# Accumulate lines for the current answer current\_a += " " + line.strip()

# Save the last Q&A pair if it exists if current\_q and current\_a:

questions.append(current\_q.strip()) answers.append(current\_a.strip())

return questions, answers

# Sample transcript (Replace this string with your actual transcript data)

sample\_transcript = """

Martin Viecha: We will now start the investor question part of the earnings call.

Q: What is the outlook for next quarter? Elon Musk: We expect to grow substantially. Q: What about competition?

Elon Musk: Competition is always good for the market. """

questions, answers = parse\_transcript(sample\_transcript) print("Questions:")

for q in questions:

print(q)

print("\nAnswers:") for a in answers:

print(a)

This is a simple example and might not handle all the intricacies of real-world earnings call transcripts. For example, some earnings calls might have multiple people answering a single question, the VP of investor relations might be different in the future, or the Q&A format may vary on the earnings call.

Note that this assumes the transcript is well-formatted and follows the patterns coded into the function. You might need to adapt the code to fit the specific formatting and structure of the transcripts you’re working with.

5. Save the data as a CVS file(s).

Now, you can save the news articles and earnings call transcripts data as a CSV file. You can easily save the data as a CSV file using the pandas library. Here’s how you might modify the previous scripts to save the data into a CSV file:

For the NewsAPI data, use the following code:

import pandas as pd

from newsapi import NewsApiClient

newsapi = NewsApiClient(api\_key='your-newsapi-key')

# You can adjust the dates and sort type as per your requirements

all\_articles = newsapi.get\_everything(q='Tesla',

from\_param='2022-10-01', to='2022-12-31',

sort\_by='relevancy')

# Create a DataFrame to store the article data df = pd.DataFrame(all\_articles['articles'])

# Save the DataFrame to a CSV file df.to\_csv('newsapi\_data.csv')

B). For the Earnings Call Transcript data from the Financial Modeling Prep API:

import requests import json

import pandas as pd

# Initialize API endpoint and API key

api\_endpoint = "https://financialmodelingprep.com/api/v3/your\_ earnings\_call\_endpoint\_here"

api\_key = "your\_api\_key\_here"

# Payload or parameters for date range and Tesla's ticker symbol params = {

"from": "2022-10-01",

"to": "2022-12-31",

"ticker": "TSLA", "apikey": api\_key

}

try:

# Make the API request

response = requests.get(api\_endpoint, params=params) response.raise\_for\_status()

# Parse the JSON data

data = json.loads(response.text)

# Extract the transcript, assuming it's in a key called 'transcript'

# (Modify as per actual API response) transcript\_data = data.get("transcript", [])

# Convert the transcript data to a DataFrame

df = pd.DataFrame(transcript\_data, columns=['Transcript'])

# Save the DataFrame to a CSV file df.to\_csv('Tesla\_earnings\_call\_transcript.csv', index=False)

except Exception as e:

print(f"An error occurred: {e}")

**Important note**

In this code snippet, we included "ticker": "TSLA" to the params dictionary to specify that we’re interested in Tesla’s earnings call transcripts. This assumes that the API uses a parameter named ticker to specify the company. You may need to consult Financial Modeling Prep’s API documentation to confirm the exact parameter name and usage.

The reasons why we chose to save the raw data in the CSV files as opposed to saving it after the sentiment analysis was completed are as follows:

* **Reusability of raw data**: If you believe the raw data could be useful for other analyses in the future, it might be a good idea to save it as is. This way, you can always go back to the original data and perform different or additional analyses as required.
* **Computational resources**: If you’re dealing with a large amount of data and limited computational resources, it might be more efficient to perform sentiment analysis on-the- fly as the data comes in and then save the results. This way, you don’t need to store large amounts of raw data and then process it all at once.
* **Iterative improvement**: If you plan to improve or change your sentiment analysis method over time, saving the raw data would be beneficial. You can re-run your new and improved analysis on the original data at any time.

6. Perform sentiment analysis.

Once you have the news and earnings call data, we can perform sentiment analysis on it using TextBlob.

Here’s an outline of the process using the TextBlob library in Python for Tesla news articles:

from textblob import TextBlob# Function to calculate sentiment def calculate\_sentiment(text: str):

blob = TextBlob(text)

return blob.sentiment.polarity

# Let's assume you have a list of news articles

news\_articles = [...] # replace with your list of news articles

# Calculate sentiment for each article

sentiments = [calculate\_sentiment(article) for article in news\_ articles]

# You could then save these sentiments to a CSV file along with the articles:

import pandas as pd

df = pd.DataFrame({ 'Article': news\_articles, 'Sentiment': sentiments,

})

df.to\_csv('article\_sentiments.csv', index=False)

This will create a CSV file named article\_sentiments.csv, which contains each article along with its sentiment score.

You can then import this CSV file into Power BI to create visualizations.

Case study: Tesla Inc. 21

For the news articles, consider separating the text by speaker and then running sentiment analysis. This could provide insights into how different people commenting on the article are perceived or whether different individuals have different sentiment in their speech.

Here’s an outline of the process using the TextBlob library in Python for Tesla earnings call transcripts:

from textblob import TextBlob import pandas as pd

# Function to calculate sentiment def calculate\_sentiment(text: str):

blob = TextBlob(text)

return blob.sentiment.polarity

# Assuming 'transcript' is a list of strings where each string is an earnings call transcript

transcripts = [...] # replace with your list of earnings call transcripts

# Calculate sentiment for each transcript

sentiments = [calculate\_sentiment(transcript) for transcript in transcripts]

# Save these sentiments to a CSV file along with the transcripts:

df = pd.DataFrame({ 'Transcript': transcripts, 'Sentiment': sentiments,

})

df.to\_csv('transcript\_sentiments.csv', index=False)

This code will create a new CSV file named transcript\_sentiments.csv, which includes each earnings call transcript along with its sentiment score. As with the news articles, you can then import this CSV file into Power BI to create visualizations.

For the earnings call transcript, consider separating the text by speaker and then running sentiment analysis. This could provide insights into how different people (e.g., CEO, CFO, investor relations, Wall Street analysts) are perceived or whether different individuals have different sentiments in their speech.

Again, it’s important to note that TextBlob provides a simple form of sentiment analysis.

For more nuanced analysis, consider using more sophisticated models from libraries such as mers, i.e., GPT 3.5.

Integrate the sentiment analysis with a trading strategy.

The sentiment analysis data can be used as a signal in a trading strategy. For instance, a significant increase in positive sentiment could be a signal to buy, while an increase in negative sentiment could be a signal to sell or short.

Please consider these as illustrative examples rather than ready-to-use code.

Let’s assume you have two Python scripts, one for sentiment analysis (sentiment\_analysis. py) and another for decision making and trade execution (trade\_execution.py).

For the sentiment analysis script (sentiment\_analysis.py), here’s a simplified version of a script that performs sentiment analysis and saves the results:

from newsapi import NewsApiClient from textblob import TextBlob import pandas as pd

import os

def get\_sentiment(text): analysis = TextBlob(text)

if analysis.sentiment.polarity > 0: return 'positive'

elif analysis.sentiment.polarity == 0: return 'neutral'

else:

return 'negative'

newsapi = NewsApiClient(api\_key='YOUR\_API\_KEY')

data = newsapi.get\_everything(q='Tesla', language='en')

articles = data['articles']

sentiments = [get\_sentiment(article['description']) for article in articles]

df = pd.DataFrame({'Article': articles, 'Sentiment': sentiments})

# Save to CSV

df.to\_csv('sentiment\_scores.csv', index=False)

For the decision making and trade execution script (trade\_execution.py), here’s a simplified version of a script that reads the sentiment scores, makes decisions, and executes trades:

import pandas as pd

import alpaca\_trade\_api as tradeapi import os

api = tradeapi.REST('APCA-API-KEY-ID', 'APCA-API-SECRET-KEY',

base\_url='https://paper-api.alpaca.markets') df = pd.read\_csv('sentiment\_scores.csv')

# Analyze the sentiment scores and make a decision positive\_articles = df[df['Sentiment'] == 'positive'].shape[0] negative\_articles = df[df['Sentiment'] == 'negative'].shape[0]

# Placeholder for your trading strategy if positive\_articles > negative\_articles:

decision = 'buy'

elif negative\_articles > positive\_articles: decision = 'sell'

else:

decision = 'hold'

# Execute the decision if decision == 'buy':

api.submit\_order( symbol='TSLA', qty=1, side='buy', type='market',

time\_in\_force='gtc'

)

elif decision == 'sell': api.submit\_order(

symbol='TSLA', qty=1, side='sell', type='market',

time\_in\_force='gtc'

)

Case study: Tesla Inc. 25

To run these scripts at specific intervals, you might use a task scheduler. For example, on Unix-based systems, you might use cron. Here’s a sample cron job that runs sentiment\_ analysis.py every day at 8 AM and trade\_execution.py every day at 9 AM:

# Edit your crontab file with crontab -e and add the following lines:

# Run sentiment\_analysis.py at 8 AM every day

0 8 \* \* \* cd /path/to/your/scripts && /usr/bin/python3 sentiment\_analysis.py

# Run trade\_execution.py at 9 AM every day

0 9 \* \* \* cd /path/to/your/scripts && /usr/bin/python3 trade\_ execution.py

In the Windows environment, you can use Task Scheduler to accomplish the same task. Remember to replace /path/to/your/scripts with the actual path to your scripts and

/usr/bin/python3 with the path to your Python interpreter.

9. Involve ChatGPT in the process.

Including ChatGPT in this process could provide an additional layer of analysis to support your trading strategy. Specifically, ChatGPT can be used to provide additional insights from the news articles or transcripts and help with decision-making.

For instance, instead of a simple positive, neutral, or negative sentiment analysis, you could use ChatGPT to generate a summary of each article or transcript. This summary could be analyzed for more nuanced sentiment, such as enthusiasm for a new Tesla product or concerns about supply chain issues.

To implement this, you would need to feed the text of each article or transcript into ChatGPT and then analyze the resulting output.

The following is example Python code:

import openai

from textblob import TextBlob openai.api\_key = 'your-openai-key'

def get\_summary(text):

response = openai.Completion.create( engine="text-davinci-002", prompt=text,

temperature=0.3, max\_tokens=100

)

return response.choices[0].text.strip()

26 Tesla’s Financial Journey: AI Analysis and Bias Unveiled

def get\_sentiment(text): analysis = TextBlob(text)

if analysis.sentiment.polarity > 0: return 'positive'

elif analysis.sentiment.polarity == 0: return 'neutral'

else:

return 'negative'

# Let's assume we have a list of articles

articles = ["Article 1 text...", "Article 2 text...", "..."]

summaries = [get\_summary(article) for article in articles] sentiments = [get\_sentiment(summary) for summary in summaries]

# You can now proceed to save the summaries and sentiments and use them in your decision-making process

**Important note**

Remember that this is a simplified example and the actual implementation might require handling various edge cases and API limits.

Moreover, incorporating ChatGPT into your process might require an adjustment of the sentiment analysis, as you’re moving from analyzing the entire articles to analyzing summaries generated by

GPT-4. You’ll also need to account for the costs associated with using the OpenAI API.

**Financial visualizations—data extraction to Power BI visualizations**

Here are the steps to build the financial visualizations discussed in the previous section. We will take you through the process of extracting the data, saving it, and then extracting it to create each visualization.

Use Python to download the 10-Q and 10-K reports from the EDGAR database on the SEC’s website. Here’s a basic Python script using the requests library to download a single file:

**import requests**

**def download\_file(url, filename):**

**response = requests.get(url)**

**open(filename, 'wb').write(response.content)**

**#** [**URL to the file (link you get from the SEC's EDGAR database)**](https://www.sec.gov/Archives/edgar/data/1318605/000156459021004599/0001564590-21-004599-index.htm)

[**url = 'https://www.sec.gov/Archives/**](https://www.sec.gov/Archives/edgar/data/1318605/000156459021004599/0001564590-21-004599-index.htm)**edgar/ data/1318605/000156459021004599/0001564590-21-004599-index.htm'**

**# Path where you want to store the file filename = 'tesla\_10k.html'**

**download\_file(url, filename)**

This script simply downloads a file from a given URL and saves it to the specified location. You need to replace the url variable with the URL of the 10-K or 10-Q report that you want to download.

Keep in mind that you would need to repeat this process for each 10-K and 10-Q company report you want to include in the Power BI visualizations to compare to Tesla. We would recommend adding from the upcoming list to complete your comparison analysis through SEC filings. **Instructions**

You will need to locate the **Central Index Key** (**CIK**) number for any company from which you want to find SEC filings, such as 10-K annual reports or 10-Q quarterly reports. The CIK number is a unique identifier assigned by the **US Securities and Exchange Commission** (**SEC**) to corporations who are obligated to disclose financial information with the SEC.

Here’s a concise guide on how to obtain a CIK number for [a public company.](https://www.sec.gov/edgar/searchedgar/companysearch.html) [SEC’s EDGAR database:](https://www.sec.gov/edgar/searchedgar/companysearch.html)

* [Go to the SEC’s](https://www.sec.gov/edgar/searchedgar/companysearch.html) EDGAR database: [https://www.sec.gov/edgar/searchedgar/](https://www.sec.gov/edgar/searchedgar/companysearch.html) [companysearch.html](https://www.sec.gov/edgar/searchedgar/companysearch.html)
* In the **Company Name** field, type the name of the company you’re interested in. The search results will display the company’s CIK number alongside its name.

Google or Bing online search:

* + You can obtain a CIK number by conducting a simple online search. Type the company’s name followed by CIK number into the search engine of your choice (e.g., Google CIK number).

Company’s website or filings:

* + Public companies often include their CIK number on their official website, especially in the investor relations section or in their SEC filings.
  + General Motors (GM) SEC CIK Number: 0001467858

GM is a traditional automaker that’s investing heavily in electrification and autonomous driving technologies. Its Chevrolet Bolt and upcoming GMC Hummer EV and Cadillac Lyriq are direct competitors to Tesla’s models.

* Ford (F) SEC CIK Number: 0000037996

Ford’s Mustang Mach-E and the upcoming all-electric F-150 Lightning show the company’s commitment to electrification. Ford is a legacy car manufacturer similar to GM and is in the midst of a transition to the EV market.

* Rivian (RIVN) SEC CIK Number: 0001809779

A pure-play EV company, Rivian is a US-based EV manufacturer backed by Ford and Amazon, which recently went public and is a direct competitor to Tesla in the electric truck market.

* NIO Inc. (NIO) SEC CIK Number: 0001736541

While not US-based (it’s a Chinese company), NIO is listed on the NYSE. NIO is a manufacturer of premium EVs and is often referred to as the “Tesla of China."

* XPeng Inc. (XPEV) SEC CIK Number: 0001821684

Another Chinese EV manufacturer listed on the NYSE, XPeng is focused on developing affordable EVs and advanced autonomous driving technologies.

* Lucid Group (LCID) SEC CIK Number: 0001736874

Lucid Motors is an American EV manufacturer that recently went public. Its first model, the Lucid Air, is a luxury electric sedan that competes with Tesla’s Model S.

By comparing Tesla with both traditional automakers (GM, Ford) and pure-play EV companies (Rivian, NIO, XPeng, Lucid), the visualizations should provide a comprehensive view of Tesla’s performance in the rapidly evolving EV market.

Automating this for multiple companies over multiple years would involve building a more sophisticated process that can navigate the SEC’s EDGAR database, which is beyond the scope of this response. You can also reference the SEC API process provided in *Chapter 1*.

Once you have these files, you would then need to process them to extract the relevant financial data. This could be done using Python’s built-in string methods or regular expressions for simple cases or with a library such as BeautifulSoup for more complex HTML processing.

As an alternative to *step 1* (CSV file option), you can extract data from a company’s 10-K and 10-Q reports for analysis involves web scraping from SEC’s EDGAR database, HTML/XML parsing, and handling CSV files for data storage. Here’s a basic script that demonstrates these steps:

**import os import requests**

**from bs4 import BeautifulSoup import csv**

30 Tesla’s Financial Journey: AI Analysis and Bias Unveiled

**# Set the URL for the company's filings page on EDGAR**

**company\_url =** [**"https://www.sec.gov/cgi-bin/browse-edgar?action=getcomp**](http://www.sec.gov/cgi-bin/browse-edgar?action=getcomp) **any&CIK=0001318605&type=&dateb=&owner=exclude&count=40"**

**# Download the page**

**response = requests.get(company\_url) page\_content = response.content**

**# Parse the page with BeautifulSoup**

**soup = BeautifulSoup(page\_content, 'html.parser')**

**# Find all document links on the page**

**doc\_links = soup.find\_all('a', {'id': 'documentsbutton'})**

**# If no such id exists, find links by text (this assumes that the text 'Documents' is consistent)**

**if not doc\_links:**

**doc\_links = soup.find\_all('a', string='Documents')**

**# Loop through the document links for doc\_link in doc\_links:**

**# Get the URL of the document page**

**doc\_page\_url = 'https://www.sec.gov' + doc\_link.get('href')**

**# Download the document page**

**response = requests.get(doc\_page\_url) doc\_page\_content = response.content**

**# Parse the document page**

**soup = BeautifulSoup(doc\_page\_content, 'html.parser')**

**# Find the link to the 10-K or 10-Q file**

**filing\_link = soup.find\_all('a', {'href': lambda href: (href and ("10-K" in href or "10-Q" in href))})**

**# If a filing link was found if filing\_link:**

**# Get the URL of the 10-K or 10-Q file**

**filing\_url = 'https://www.sec.gov' + filing\_link[0]. get('href')**

**# Download the file**

**response = requests.get(filing\_url)**

**filing\_content = response.content**

**# Parse the file content (as text for simplicity) soup = BeautifulSoup(filing\_content, 'html.parser')**

**# Find all tables in the file tables = soup.find\_all('table')**

**# Loop through the tables and save each as a CSV file for i, table in enumerate(tables):**

**with open(f'{doc\_link.text}\_{i}.csv', 'w', newline='') as**

**f:**

**writer = csv.writer(f)**

**for row in table.find\_all('tr'): writer.writerow([col.text for col in row.find\_**

**all('td')])**

I. Extract Power BI visualization data for each visualization.

The next step is to identify which tables contain the data you need and to extract that data into CSV files. Here’s a simple Python script to illustrate this process:

import csv

# List of tables parsed from the 10-K or 10-Q file tables = [...]

# The indices of the tables containing the data we need market\_share\_table\_index = ... operating\_efficiency\_ratio\_table\_index = ... revenue\_growth\_table\_index = ... gross\_margin\_table\_index = ... rd\_investment\_table\_index = ... geographic\_revenue\_distribution\_table\_index = ...

# List of the table indices table\_indices = [

market\_share\_table\_index, operating\_efficiency\_ratio\_table\_index, revenue\_growth\_table\_index, gross\_margin\_table\_index, rd\_investment\_table\_index, geographic\_revenue\_distribution\_table\_index

]

# List of names for the CSV files

csv\_names = [

"market\_share.csv", "operating\_efficiency\_ratio.csv", "revenue\_growth.csv", "gross\_margin.csv", "rd\_investment.csv", "geographic\_revenue\_distribution.csv"

]

# Loop through the table indices for i in range(len(table\_indices)):

# Get the table

table = tables[table\_indices[i]]

# Open a CSV file

with open(csv\_names[i], 'w', newline='') as f: writer = csv.writer(f)

# Loop through the rows in the table for row in table.find\_all('tr'):

# Write the row to the CSV file writer.writerow([col.text for col in row.find\_

all('td')])

You’ll need to manually inspect the 10-K and 10-Q documents to determine which tables contain the data you need (market\_share\_table\_index, operating\_efficiency\_ ratio\_table\_index, etc., in the script). Once you’ve identified those tables, this script will extract the data from them and save it into separate CSV files.

However, this is still a simplified example. In practice, the data may need cleaning or reshaping before it can be used for visualizations. You may also need to extract data from other parts of the document besides tables. Furthermore, some of the data you’re interested in, such as market share or operating efficiency ratio, might not be directly reported in the 10-K or 10-Q. In these cases, you would need to calculate these metrics from the available data or find alternative

data sources.

I. Vehicle range and performance data:

Let’s use a hypothetical example of extracting electric vehicle data from a website such as Inside EVs, which contains the specifications of various electric vehicles. Please remember that

this example is only for educational purposes, and you should always respect the website’s terms and conditions and data privacy regulations.

This Python example will utilize BeautifulSoup and Requests, two widely used libraries for web scraping.

Before you begin, you need to install these libraries if you haven’t already. You can install them via pip:

**pip install beautifulsoup4 requests pandas**

Here is a simple Python script to scrape EV data:

import requests

from bs4 import BeautifulSoup import pandas as pd

def scrape\_data(url):

response = requests.get(url)

soup = BeautifulSoup(response.text, 'html.parser')

table = soup.find('table') # Assumes only one table on the

page

headers = []

for th in table.find('tr').find\_all('th'): headers.append(th.text.strip())

rows = table.find\_all('tr')[1:] # Exclude header

data\_rows = [] for row in rows:

data = []

for td in row.find\_all('td'): data.append(td.text.[strip())](https://insideevs.com/guides/electric-car-range-charging-time/)

[data\_rows.append(data)](https://insideevs.com/guides/electric-car-range-charging-time/)

[return pd.DataFr](https://insideevs.com/guides/electric-car-range-charging-time/)ame(data\_rows, columns=headers)

url = 'https://insideevs.com/guides/electric-car-range-charging- time/' # Example URL, please check if scraping is allowed

df = scrape\_data(url)

df.to\_csv('ev\_data.csv', index=False) # Save the data to a CSV file

The following is an explanation of the Python code snippet:

* Import the necessary libraries. You need these libraries to send HTTP requests, parse HTML, and manipulate data in a tabular format:

import requests

from bs4 import BeautifulSoup import pandas as pd

* Define a function for data scraping. This function takes a URL as input, sends a GET request to that URL, parses the HTML response to find the data table, extracts the headers and rows from the table, and returns the data as a pandas DataFrame:

def scrape\_data(url):

# Send a GET request to the URL response = requests.get(url)

# Parse the HTML content of the page with BeautifulSoup soup = BeautifulSoup(response.text, 'html.parser')

# Find the data table in the HTML (assuming there's only one table)

table = soup.find('table')

# Extract table headers headers = []

for th in table.find('tr').find\_all('th'): headers.append(th.text.strip())

# Extract table rows

rows = table.find\_all('tr')[1:] # Exclude header row

data\_rows = [] for row in rows:

data = []

for td in row.find\_all('td'): data.append(td.text.strip())

data\_rows.append(data)

# Create a DataFrame with the data and return it return pd.DataFrame(data\_rows, columns=headers)

* Use the function to scrape data and save it as a CSV file. Here, you input the URL of the web page you want to scrape data from, call th[e scrape\_data function to get the data as a data](https://insideevs.com/guides/electric-car-range-charging-time/) [frame, and th](https://insideevs.com/guides/electric-car-range-charging-time/)en save the data frame to a CSV file:

url = 'https://insideevs.com/guides/electric-car-range-charging- time/' # Example URL, please check if scraping is allowed

df = scrape\_data(url)

df.to\_csv('ev\_data.csv', index=False) # Save the data to a CSV file

**Note**

This code assumes that the web page has a single table containing the data we need. If the webpage structure is different, you’ll need to adjust the code accordingly. Always respect the website’s rules and regulations, as well as any relevant data privacy and legal aspects. However, remember that this is a simple example and might not work with all websites, especially those using JavaScript to load data or having complex structures. For such scenarios, you might have to resort to more sophisticated techniques and tools, like Selenium or Scrapy.

II. Infrastructure data

Let’s also look at the process to extract the data for our Tesla and competitors’ infrastructure (charging station) visualization.

One approach could be to use APIs of charging station databases for the infrastructure part. Let’s consider the Open Charge Map’s public API. The following Python script demonstrates how to retrieve information about charging stations in the United States:

import requests import pandas as pd

api\_key = "your\_api\_key" # replace with your API key country\_code = "US" # for United States

url = f"https://api.openchargemap.io/v3/poi/?key={api\_ key}&countrycode={country\_code}&output=json"

38 Tesla’s Financial Journey: AI Analysis and Bias Unveiled

response = requests.get(url)

# make sure the request was successful assert response.status\_code == 200

# convert to JSON

data = response.json()

# create a pandas DataFrame df = pd.json\_normalize(data)

# print the DataFrame print(df)

Convert the JSON file that holds the Tesla and competitor infrastructure data to a CSV file.

The infrastructure data extracted from the Open Charge Map API is provided in JSON format, which we then converted into a pandas DataFrame (essentially a table) in Python for easier handling. This data frame can be saved to a CSV file if desired with the following line of code:

df.to\_csv('infrastructure\_data.csv', index=False)

The vehicle range and performance data has already been manually compiled from various online sources and saved in a CSV format; it can be read in Python using the pandas read\_csv

function mentioned in *step 1*, so no additional work is required.

**KPI visualizations–data extraction to Power BI visualization**

For this section, we will be providing the steps to create visualizations to provide a way for the reader to evaluate Tesla KPIs.

The KPI visualizations are as follows:

* **Vehicle deliveries**: This is a bar chart with a line chart overlay that shows vehicle deliveries by quarter and a stacked bar chart that shows breakdown vehicle deliveries by model (e.g., Model S, Model 3, Model X, Model Y)
* **Energy storage and solar deployments**: This is a bar chart with a line chart overlay that displays energy storage deployments and solar installations by quarter

Fortunately, this data can be found in the Tesla annual and quarterly reports we extracted earlier when we were pulling data from the SEC website. Extract vehicle delivery and energy storage and solar deployment data from the Tesla SEC CSV files that were saved during the Tesla financial visualization process. All you need to do is find the CSV files that have already been created and follow these steps:

1. Here’s a generic Python script for reading a CSV file and extracting the needed data:

import pandas as pd

# Load the CSV file

df = pd.read\_csv('tesla\_report.csv')

# Extract the data needed for visualizations

vehicle\_deliveries = df[['Quarter', 'Model S Deliveries', 'Model

3 Deliveries', 'Model X Deliveries', 'Model Y Deliveries']]

energy\_storage\_and\_solar\_deployments = df[['Quarter', 'Energy Storage Deployments', 'Solar Installations']]

# Save the extracted data into new CSV files

vehicle\_deliveries.to\_csv('vehicle\_deliveries.csv', index=False)

energy\_storage\_and\_solar\_deployments.to\_csv('energy\_storage\_and\_ solar\_deployments.csv', index=False)

In this code snippet:

* The pandas library is imported.
* The pd.read\_csv() function is used to read the CSV file. Replace tesla\_report.csv

with the name of your actual CSV file.

* The needed columns for each visualization are extracted into new data frames.
* The to\_csv() function is used to save these new data frames into new CSV files, which can then be imported into Power BI.

Please modify the column names in the script to match the exact column names in your CSV files. Also, replace tesla\_report.csv with the path of your CSV file.

This script assumes that you have a single CSV file with all the data needed. If the data is spread across multiple files (for instance, one file per report), you’ll need to load each file separately,

and possibly concatenate the results.

As we delve into the realm of financial data extraction, the following Python code demonstrates a practical way to fetch the latest 10-K filings directly from the SEC for a specified company (in this case, Tesla). By leveraging the power of requests and JSON libraries, we craft a function that retrieves, processes, and presents key data points, serving as a fundamental step in our financial analysis journey:

Import requests Import json

def get\_latest\_10k\_data(cik):

# Define the base URL for the SEC data API base\_url = "https://data.sec.gov/submissions/"

# Define the URL for the company's latest 10-K data url = f"{base\_url}CIK{cik}.json"

# Get the JSON content from the URL Response = requests.get(url)

data = json.loads(response.text)

# Find the data for the latest 10-K filing for filing in data['filings']:

if filing['form'] == '10-K': return filing

return None

# Get the data for Telsa's latest 10-K filing tesla\_cik = '0001318605'

tesla\_10k\_data = get\_latest\_10k\_data(tesla\_cik)

# Now you have a dictionary containing the data for Tesla's latest 10-K filing

# the structure of this will data will depend on the current format of the SEC's website

To use the SEC API to pull financial data, you would follow a similar process to the one described in the previous Python code examples. Instead of using BeautifulSoup to parse HTML from the EDGAR website, you would send a GET request to the appropriate API endpoint and then parse the returned JSON data.

As an alternative, I can provide a Python script that would help you download the 10-K filings of Tesla. You can then manually search for the geographic distribution information:

import requests import os

def download\_10k(cik, doc\_link):

# Define the base URL for the SEC EDGAR database base\_url = ["https://www.sec.gov/Archives/"](http://www.sec.gov/Archives/)

# Combine the base\_url with the doc\_link to get the full URL of the 10-K filing

url = base\_url + doc\_link

# Get the content from the URL Response = requests.get(url)

#Save the content to a .txt file

with open(cik + '.txt', 'wb') as f: f:write(response.content)

#Define the CIK for Tesla Tesla\_cik = '0001318605'

# Define the doc\_link for the latest 10-K filing of Tesla

# This can be found on the EDGAR database and will need to be updated Tesla\_doc\_link = 'edgar/data/1318605/0001564590-21-004599.txt'

# Download the 10-K filing Download\_10k(tesla\_cik, tesla\_doc\_link)

After running this script, you’ll have a text file named 0001318605.txt (Tesla’s CIK) in your current directory that contains the latest 10-K filing of Tesla. You can then open this file and manually search for the geographic distribution information.