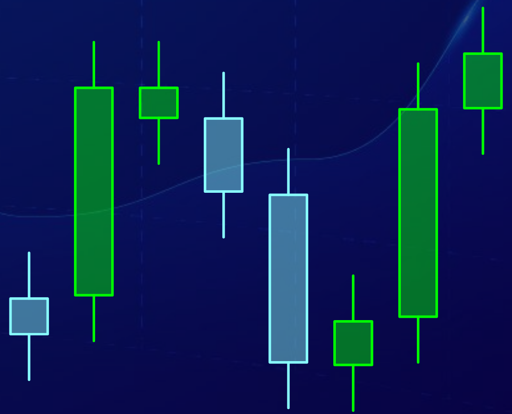
AMZN STOCK ANALYSIS: ETL PIPELINE

Presented by the Pipeline Pandas

Amrita | Nick | Jasmine | Alejandro

Table of Contents



01

Overview

Overview of Project

By using semi-structured data, specifically about daily changes in Amazon stock prices, we will create a data pipeline locally, utilizing extraction, transforming, loading and analysis to achieve clean, structured data as well as insightful analysis.

The deliverable of this product will be a SQL database with transformed and cleaned data as well as statistical insights.

02

Checkpoint 1: Extraction

Extraction

For extraction, we choose to go with a dataset from https://www.alphavantage.co/ using their stock market API.

Using the requests module in Python, we first retrieved the dataset from the URL, with some parameters.

```python
def api_call(output_size, api_key):
    return rq.get(f'https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol=AMZN&outputsize={output_size}&apikey={api_key}').json()
```

✓ 1.0s

This would efficiently return a JSON file of the dataset.
Later in the code, we can then make a call to api_call to retrieve the JSON and store it in a variable.

```python
data = api_call('full', 'KVUJT3KI90X21VGB')
```

From this data, we can then further specify that we want the actual data from within the JSON.

```
time_series_data = data["Time Series (Daily)"]
```

After this, we can then use this to finally convert into a dataframe.

```
df = pd.DataFrame(time_series_data).T
```

03

Checkpoint 2: Transformation

Upon converting the data into json we converted all the values into floats.

This will detect and transform any values which are received as strings, ensuring minimal data loss while maintaining data quality and consistency.

```
stmnt1 = '''create table if not exists amzn (
    date date primary key,
    open float,
    high float,
    low float,
    close float,
    volume int
    )
'''
```

Although we found our data source to be very clean, we decided to quickly convert any leftover 'string' values in the dataset. This would not delete any irregular data but convert it and turn all of it into a more desired data type

```python
df = pd.DataFrame(time_series_data).T
df.rename(columns={'1. open': 'open', '2. high': 'high', '3. low': 'low', '4. close': 'close', '5. volume': 'volume'}, inplace=True)
df = df.reset_index().rename(columns={'index': 'date'})
df
```

✓ 0.0s

Followed by the transformation code which set the new columns for our Dataframe

	date	open	high	low	close	volume
0	2024-02-16	168.7400	170.4200	167.1700	169.5100	48107744
1	2024-02-15	170.5800	171.1700	167.5900	169.8000	49855196
2	2024-02-14	169.2100	171.2100	168.2800	170.9800	42815544
3	2024-02-13	167.7300	170.9500	165.7500	168.6400	56345122
4	2024-02-12	174.8000	175.3900	171.5400	172.3400	51050440

	open	high	low	close	volume
date					
2024-02-16	168.7400	170.4200	167.1700	169.5100	48107744
2024-02-15	170.5800	171.1700	167.5900	169.8000	49855196
2024-02-14	169.2100	171.2100	168.2800	170.9800	42815544
2024-02-13	167.7300	170.9500	165.7500	168.6400	56345122
2024-02-12	174.8000	175.3900	171.5400	172.3400	51050440

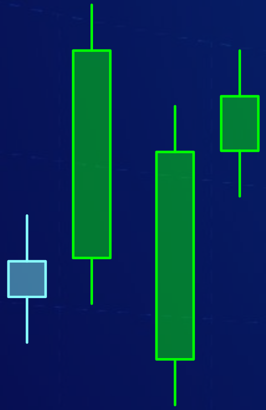
Finally changing the dataframe from the index as primary key —> to the date, as each date is already unique.
Keeping our dataframe to essential information only.

Adding a final column

Using the pandas .rolling() function we added a useful piece of data.

- Collects an average value of the closing average
- The rolling mean at each point is the average of the current value and the two preceding values within the window.
- Valuable information for traders

```
# Calculate 30-day moving average
df2['30_MA'] = df2['close'].rolling(window=30).mean()
```



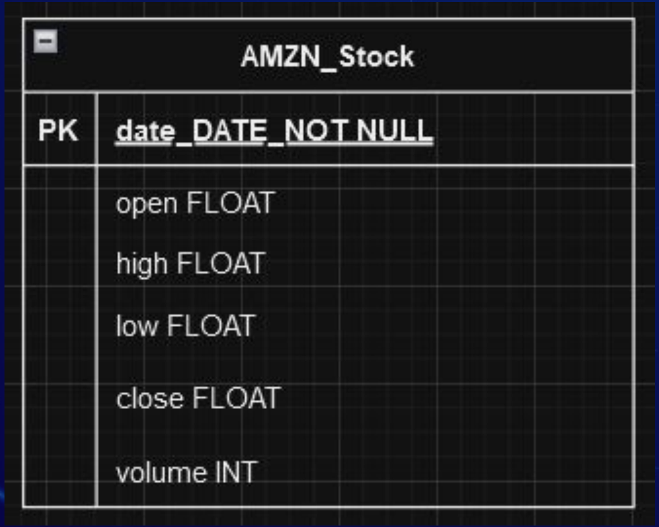
04

Checkpoint 3: Loading

Data Loading

Schema

Given the nature of the retrieved data, a simple schema was judged to be sufficient. Primary key would be the date, as that was guaranteed to be a unique value.



Selective Insertion

Most of this checkpoint's complexity lay in the mechanics of only inserting non-duplicate data into the SQLite table every time the code was run to avoid the 'unique key' error.

Depending on the state of the table ie. empty/not empty, the parameters of the API call would change.

The output size is either 'full' (entire dataset) or 'compact' (last 100 days' worth):

```
def api_call(output_size, api_key):
    return rq.get(f'https://www.alphavantage.co
```

If database table is empty, full dataset for AMZN stock is retrieved via api_call() function

If there is already data in the table, only last 100 rows of data will be retrieved

Either way, retrieved data is then converted to a dataframe.

```python
is_empty = True
if conn.execute(f"SELECT COUNT(*) FROM amzn;").fetchone()[0] == 0:
    data = api_call('full', 'KVUJT3KI90X21VGB')
else:
    data = api_call('compact', 'KVUJT3KI90X21VGB')
    is_empty = False

time_series_data = data["Time Series (Daily)"]
```

```python
if is_empty:
    df.to_sql('amzn', conn, index=False, if_exists='append')
    print("All data inserted.")
else:
    max_date = conn.execute("SELECT MAX(date) FROM amzn;").fetchone()[0]
    df_filtered = df[pd.to_datetime(df['date']) > pd.to_datetime(max_date)]
    if df_filtered.empty:
        print("No new data to insert.")
    else:
        df_filtered.to_sql('amzn', conn, index=False, if_exists='append')
        conn.commit()
        print("New data inserted.")
```

If table empty, entire dataframe will be inserted via SQLite

If there is already data in table, only the most recent, non-duplicate entries will be inserted

By comparing every entry date to the latest date already present in the table, redundant data and unique key error are no longer issues

SQL ▾ ‹ 1 / 123 › 1 - 50 of 6113

date	open	high	low	close	volume
2024-02-16	168.74	170.42	167.17	169.51	48107744
2024-02-15	170.58	171.17	167.59	169.8	49855196
2024-02-14	169.21	171.21	168.28	170.98	42815544
2024-02-13	167.73	170.95	165.75	168.64	56345122
2024-02-12	174.8	175.39	171.54	172.34	51050440
2024-02-09	170.9	175.0	170.5803	174.45	56985986
2024-02-08	169.65	171.43	168.88	169.84	42316454

	open	high	low	close	volume
date					
2024-02-20	167.8300	168.7100	165.7400	167.0800	41980326
2024-02-16	168.7400	170.4200	167.1700	169.5100	48107744
2024-02-15	170.5800	171.1700	167.5900	169.8000	49855196
2024-02-14	169.2100	171.2100	168.2800	170.9800	42815544
2024-02-13	167.7300	170.9500	165.7500	168.6400	56345122
...	...	...	...	...	...
1999-11-05	3.2375	3.2750	3.1125	3.2470	11091400
1999-11-04	3.3595	3.3595	3.0500	3.1530	16759200
1999-11-03	3.4095	3.4250	3.2500	3.2905	10772100
1999-11-02	3.4875	3.5000	3.2530	3.3220	13243200
1999-11-01	3.4030	3.5940	3.3155	3.4565	12824100

Initial insertion

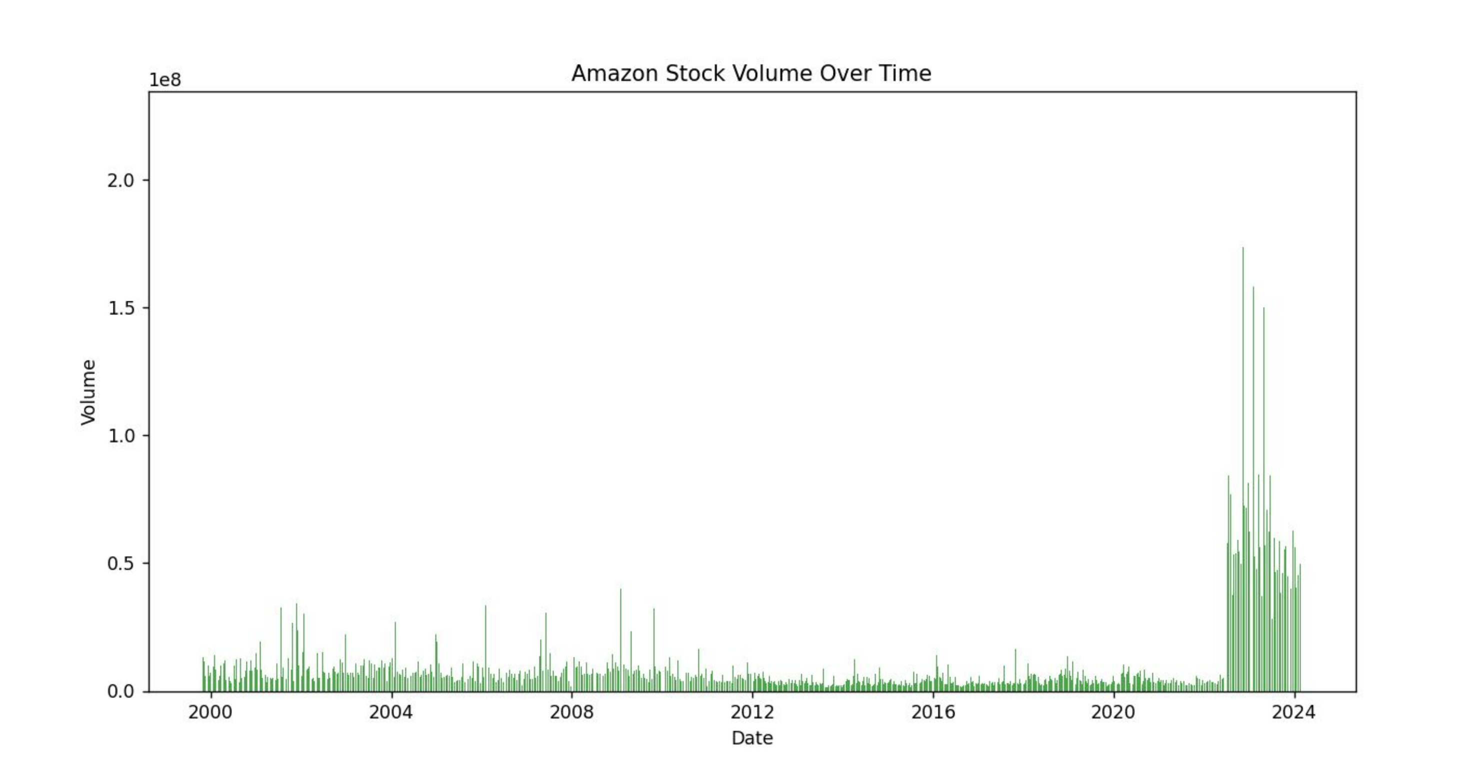
Data update

Success!

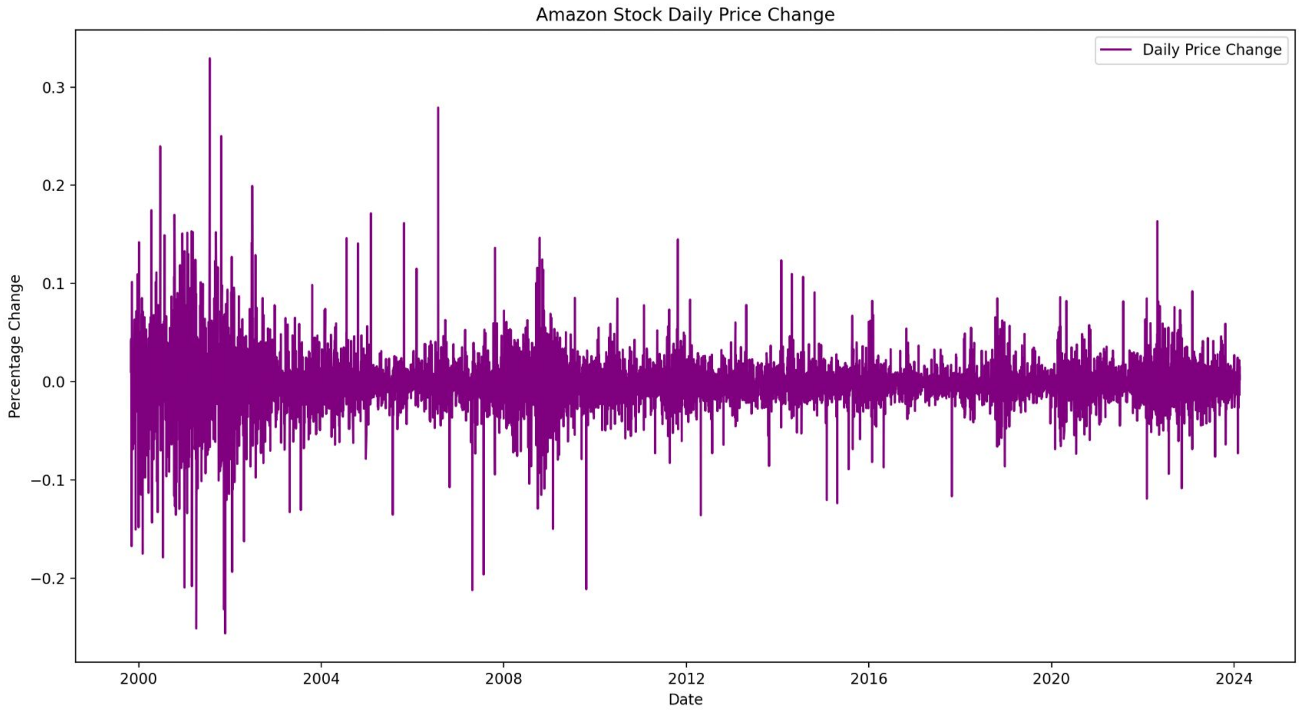
05

Checkpoint 4: Analysis

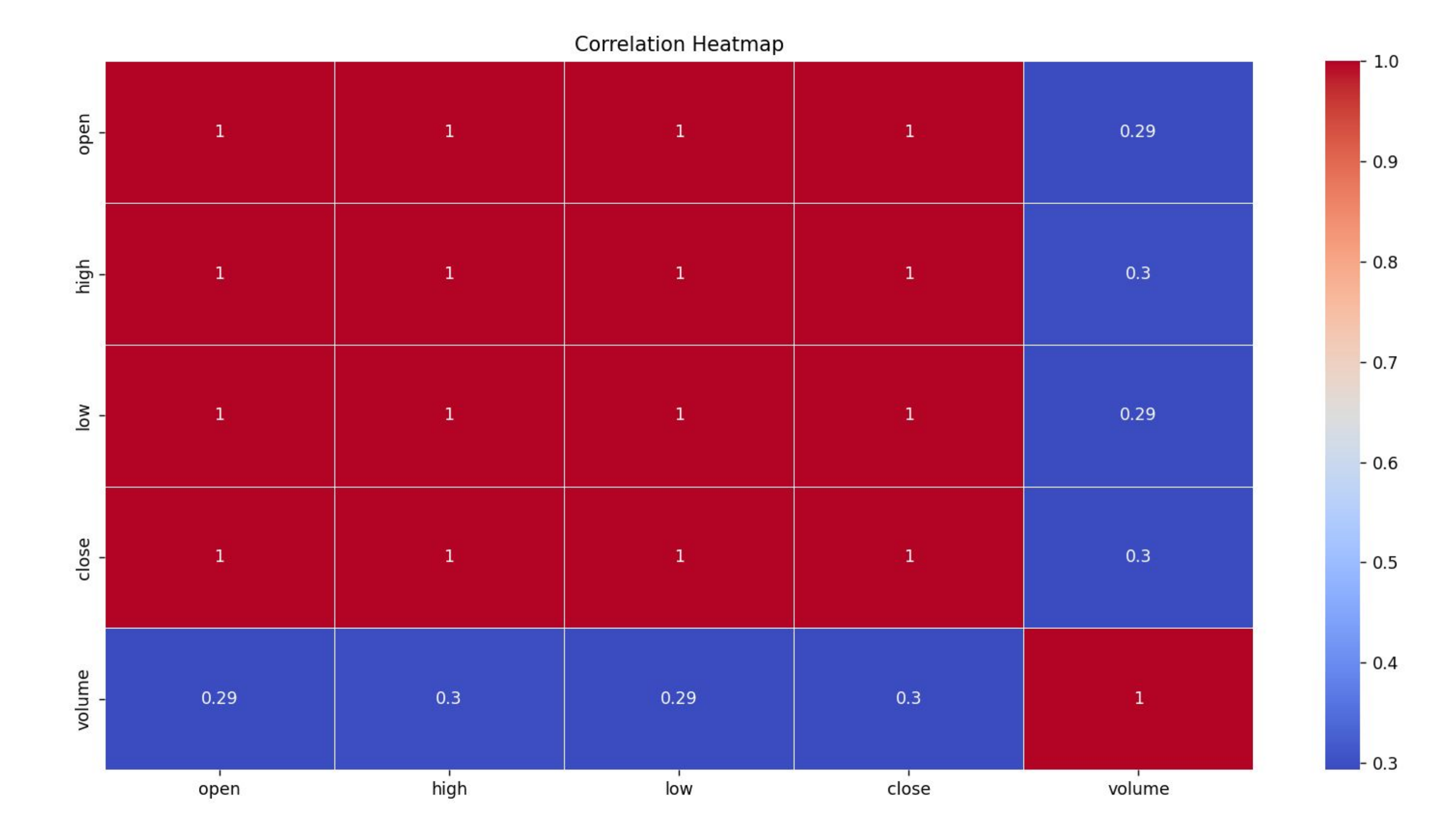

Amazon Stock Closing Prices Over Time
Closing Price
30-day Moving Average
175
150
125
100
75
50
25
0
2000
2004
2008
2012
2016
2020
2024
Date

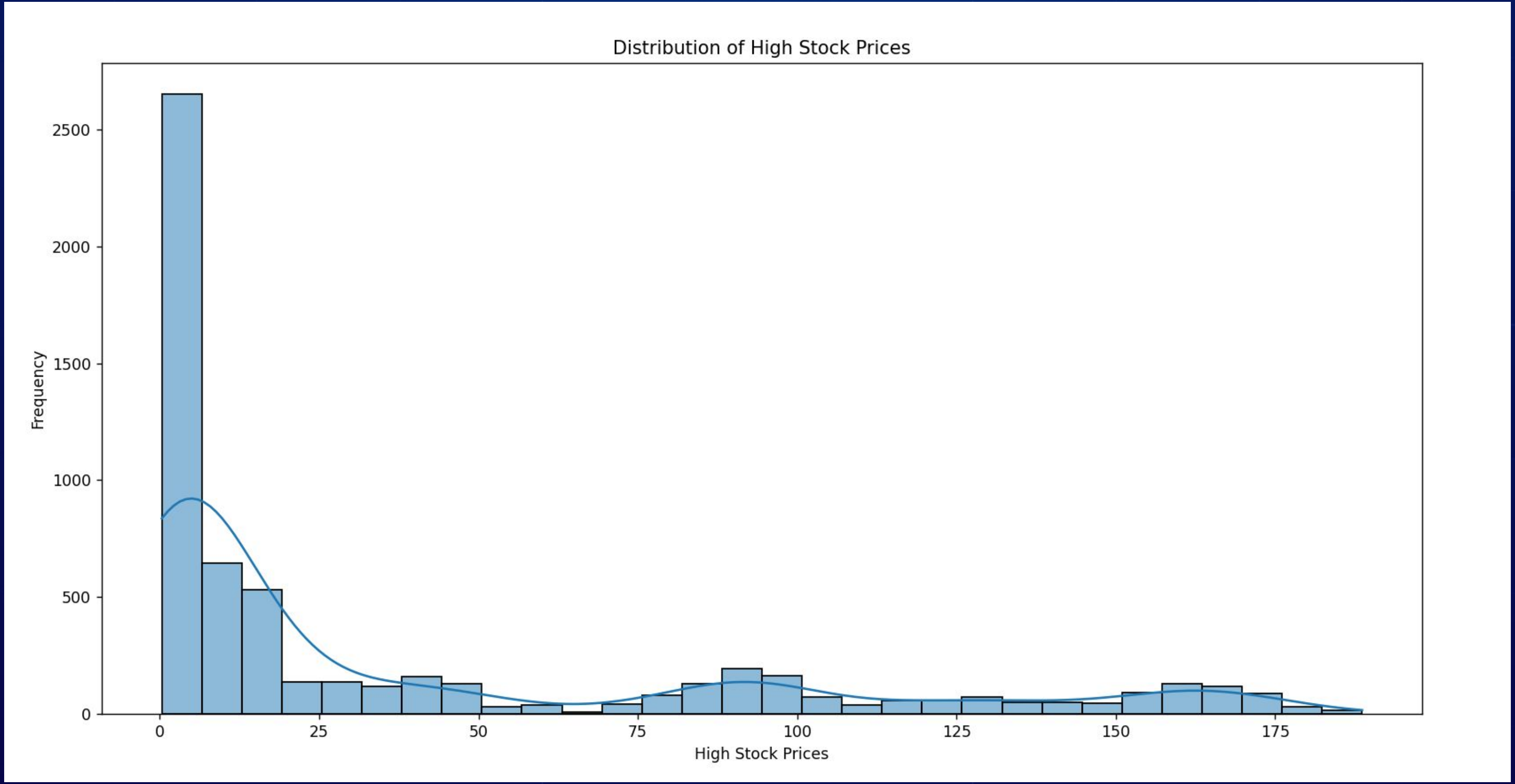


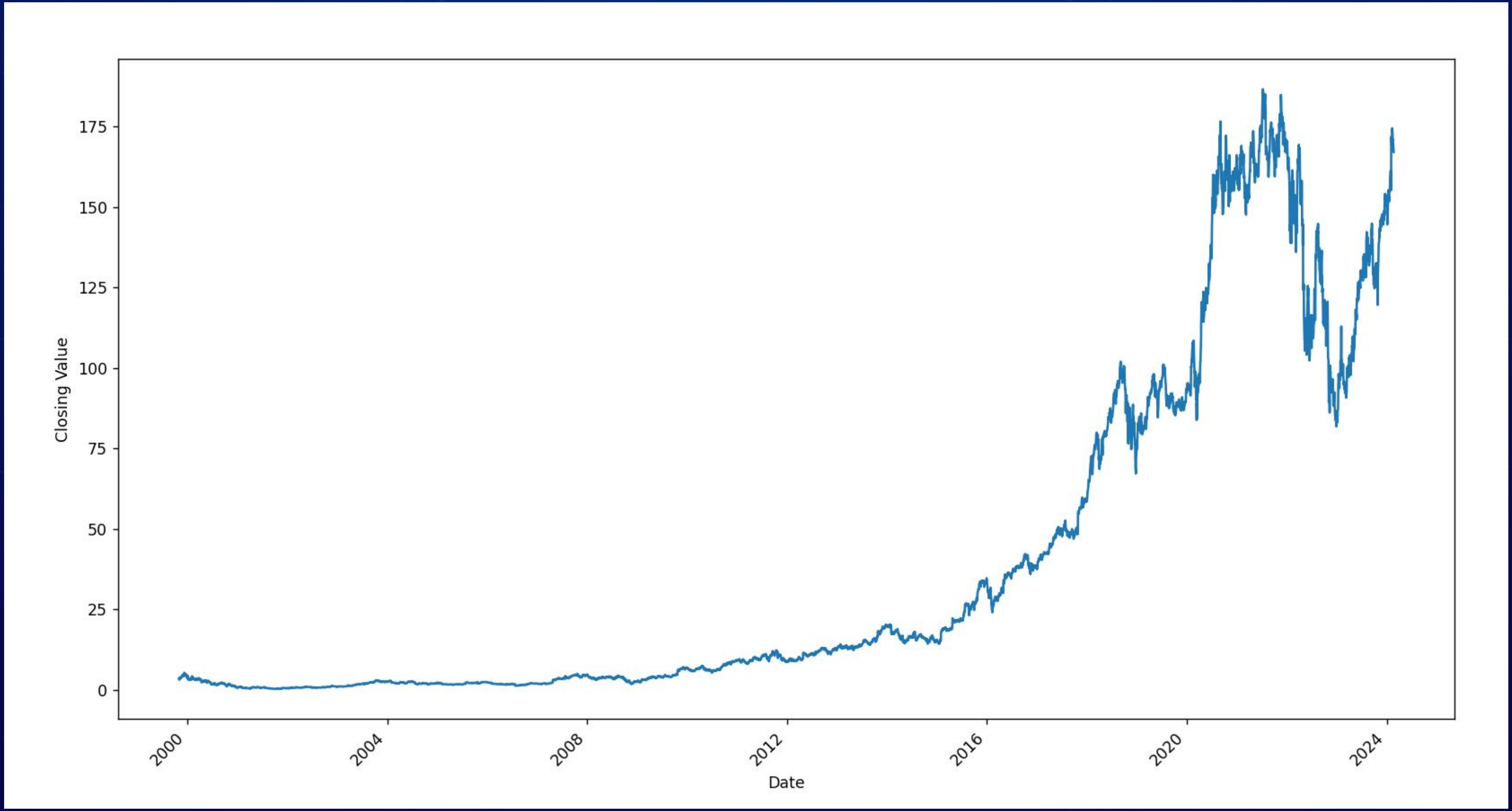
Amazon Stock Volume Over Time
1e8
2.0
1.5
1.0
0.5
0.0
Volume
2000
2004
2008
2012
2016
2020
2024
Date

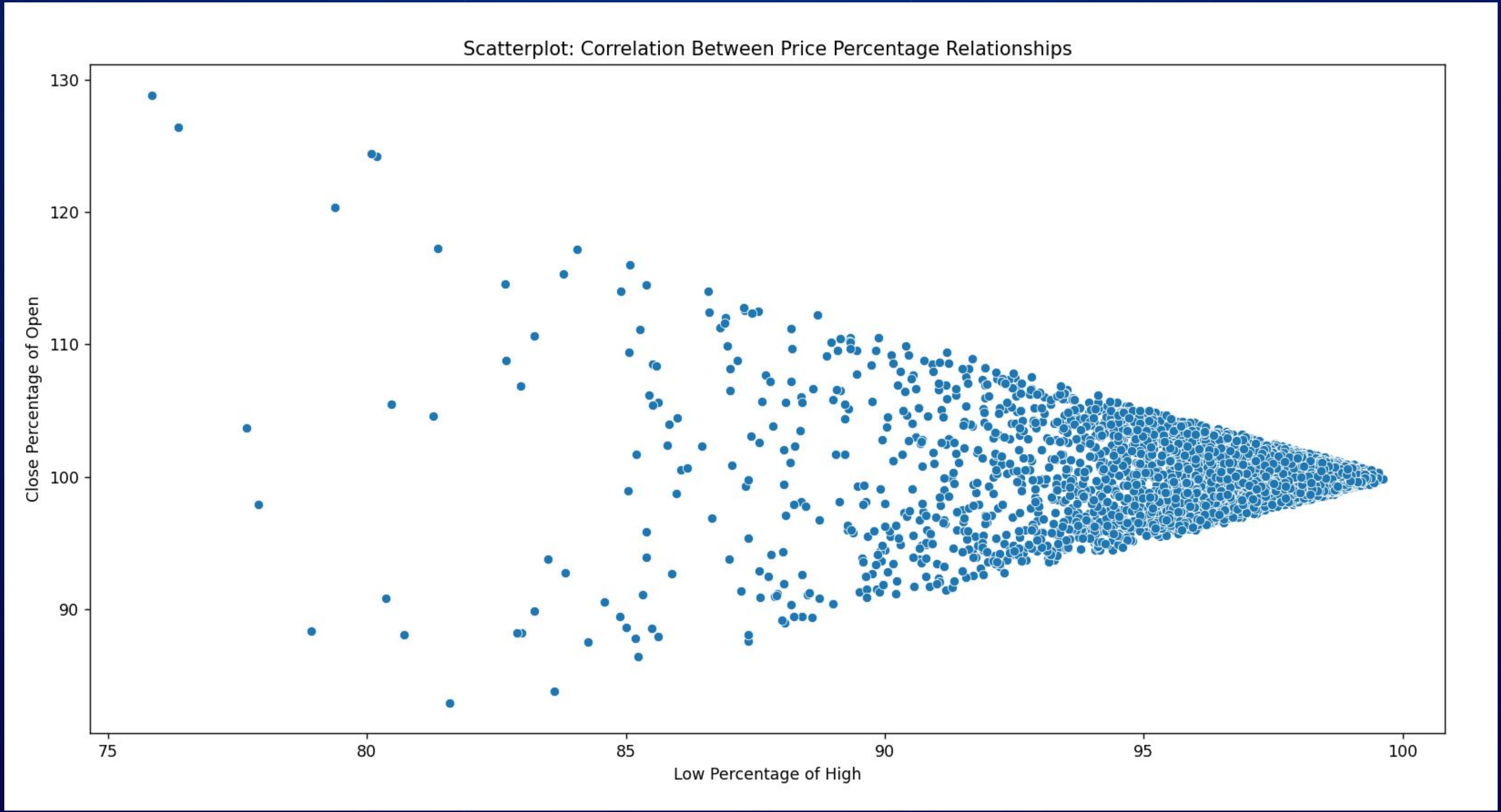


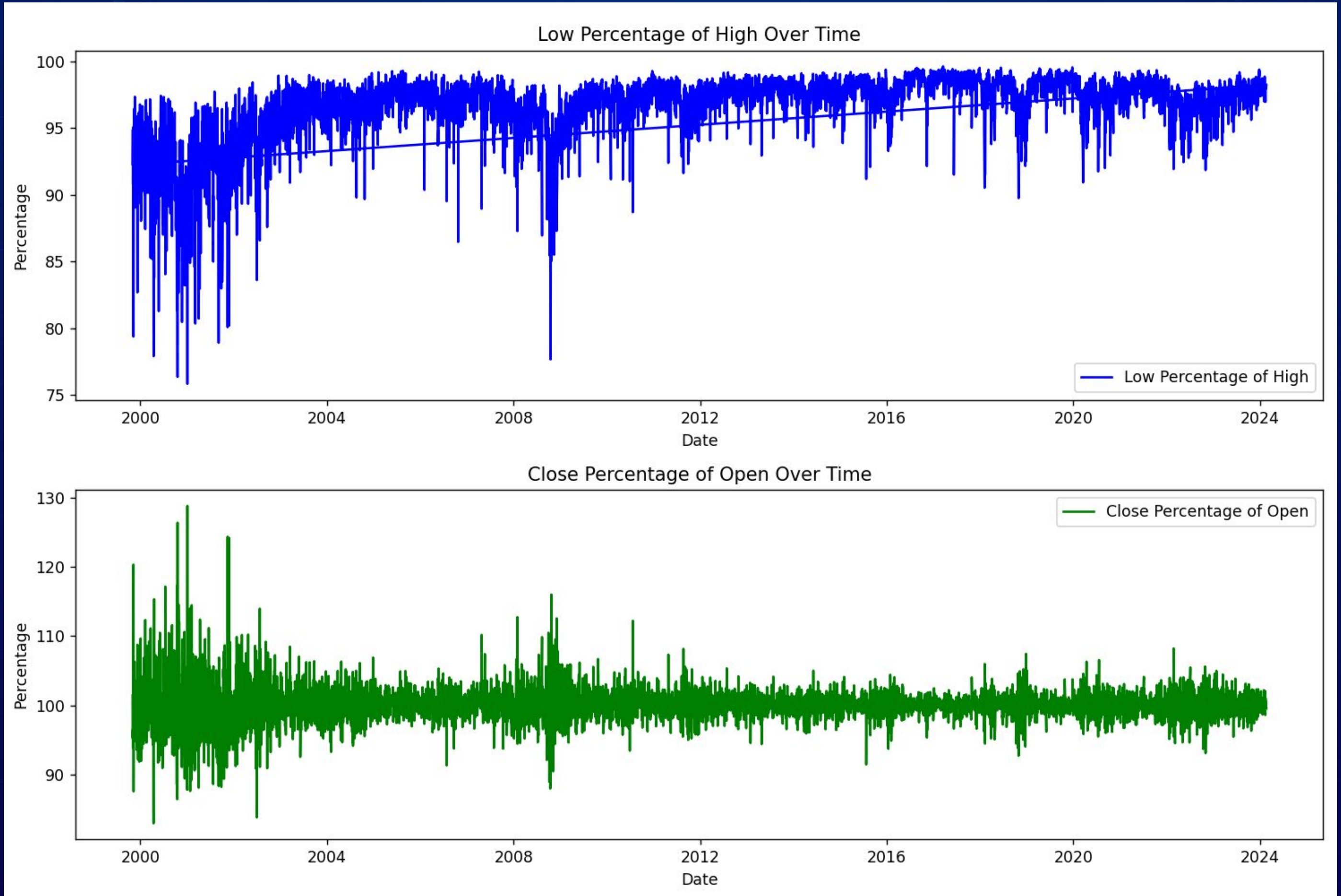
Amazon Stock Daily Price Change
Daily Price Change
Percentage Change
0.3
0.2
0.1
0.0
−0.1
−0.2
2000
2004
2008
2012
2016
2020
2024
Date

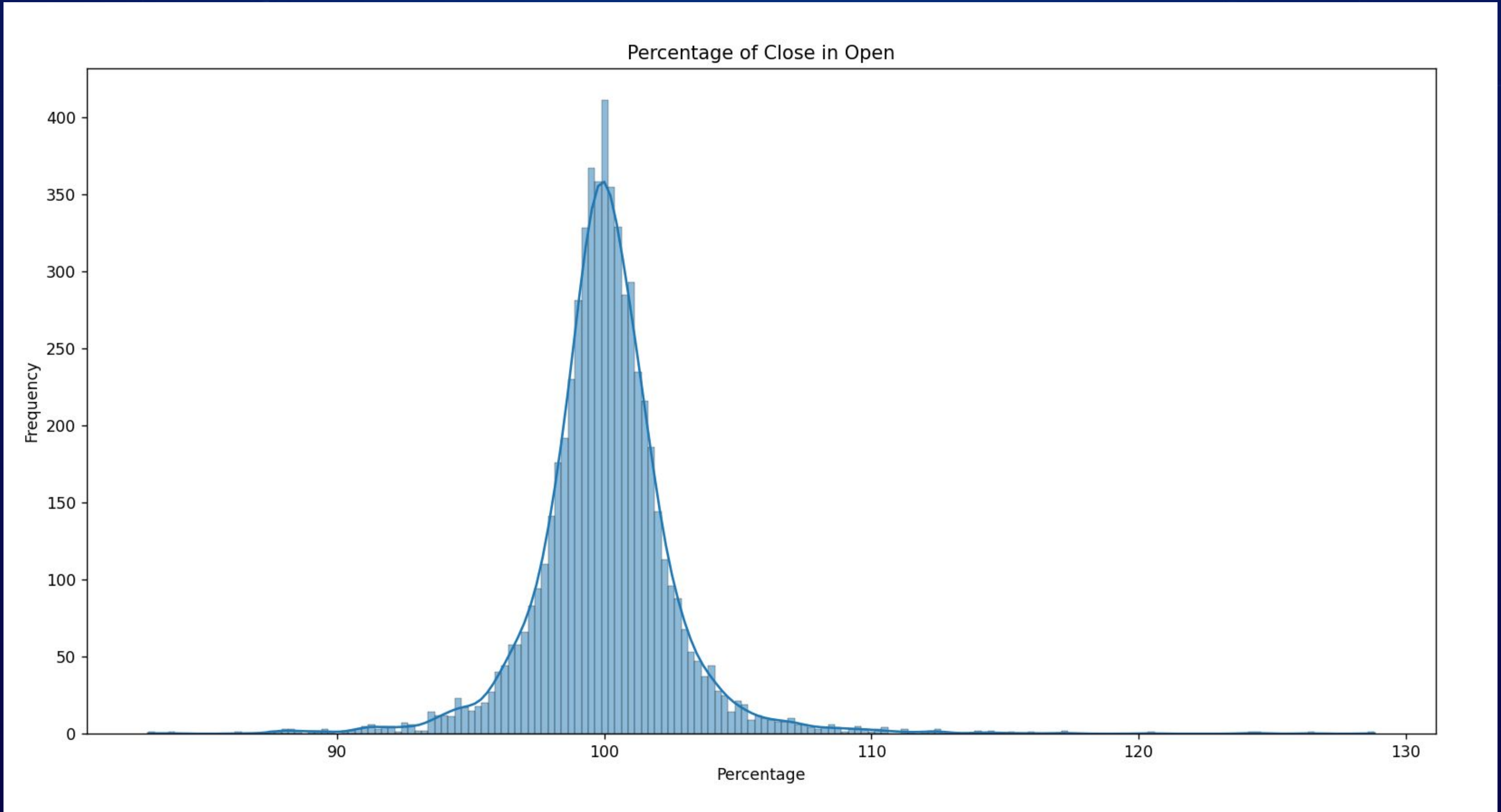

Correlation Heatmap
open
high
low
close
volume
1
1
1
1
0.29
1
1
1
1
0.3
1
1
1
1
0.29
1
1
1
1
0.3
0.29
0.3
0.29
0.3
1
open
high
low
close
volume
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3

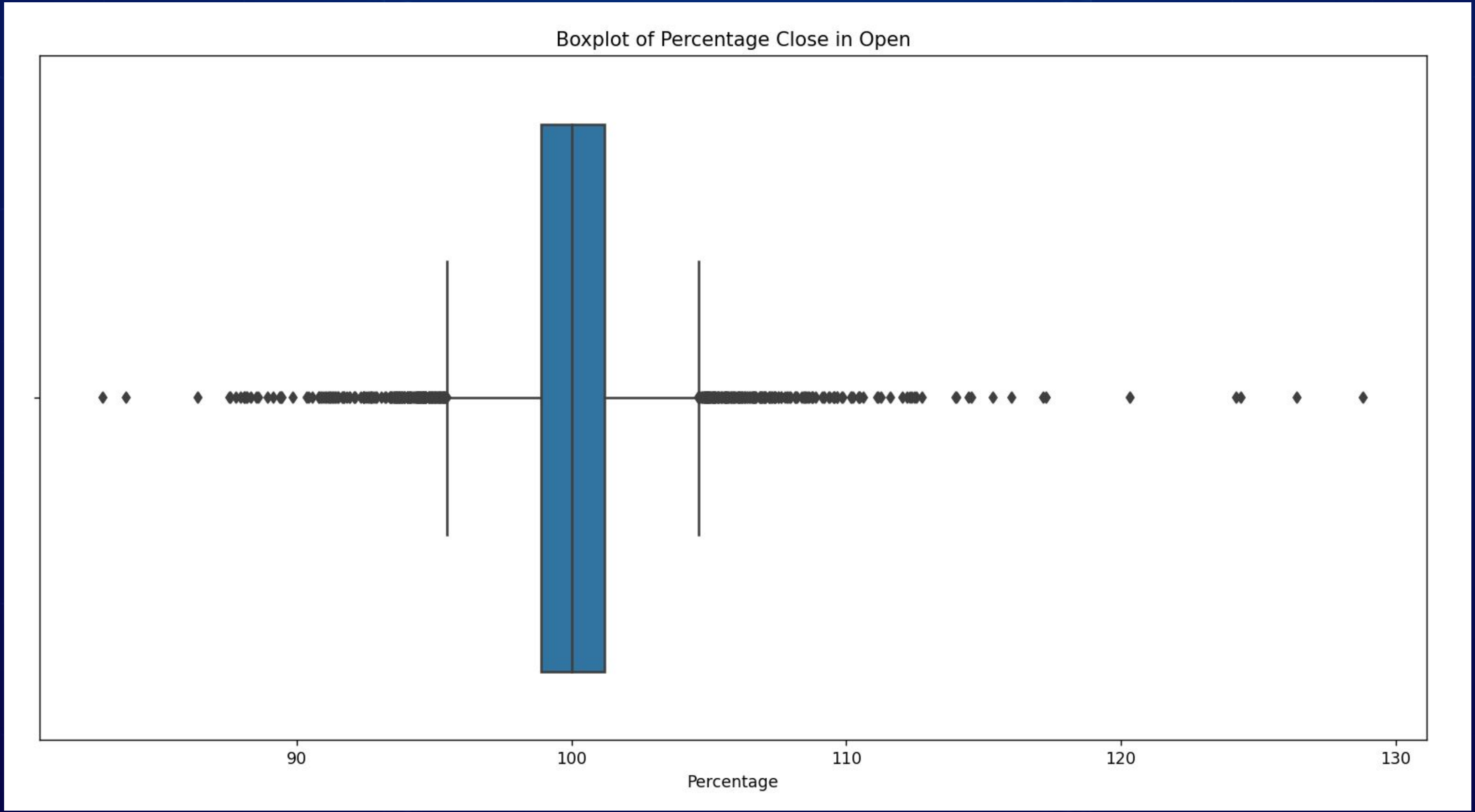

Distribution of High Stock Prices
Frequency
0
500
1000
1500
2000
2500
0
25
50
75
100
125
150
175
High Stock Prices


Closing Value
175
150
125
100
75
50
25
0
2000
2004
2008
2012
2016
2020
2024
Date


Scatterplot: Correlation Between Price Percentage Relationships
Close Percentage of Open
Low Percentage of High
130
120
110
100
90
75
80
85
90
95
100


Low Percentage of High Over Time
Percentage
Date
Low Percentage of High
Close Percentage of Open Over Time
Percentage
Date
Close Percentage of Open


Percentage of Close in Open
Frequency
Percentage
0
50
100
150
200
250
300
350
400
90
100
110
120
130


Boxplot of Percentage Close in Open
90
100
110
120
130
Percentage

06

Lessons learned

Lesson #1

The Problem

Had to make the code infinitely reusable without flooding the database with duplicates or getting the 'unique key' error

Took current state of table into account, implemented checks to determine parameter of API call

Filtered dataframe to only latest date(s) not already present in table

The Lesson

Take a goal-oriented approach to solve problems, don't be afraid to come at it from different angles

Thoroughly account for as many possible use-case scenarios as possible

Lesson #2

The Problem

Raw stock price data saw an enormous drop-off after 6/5/22

AMZN had underwent a stock split (by a factor of 20) at that time, dramatically skewing the data from that point onwards; full graphs from 1999-present showed a massive, misleading drop in price

We divided all data from before 6/5/22 by 20 prior to visualization and analysis

The Lesson

Always look into significant discrepancies or outliers in raw data and account for them, otherwise the analysis could be rendered worthless

Many times, external research should be conducted

Lesson #3

The Problem

The nature of this project structure lent itself to team members finishing certain checkpoints very quickly. Leaving some team members done with their jobs in order for others to begin..

The Lesson

Most people don't want to go and ask team members for help, and feel like their burdening others. Let them know you are bored! And want to help!

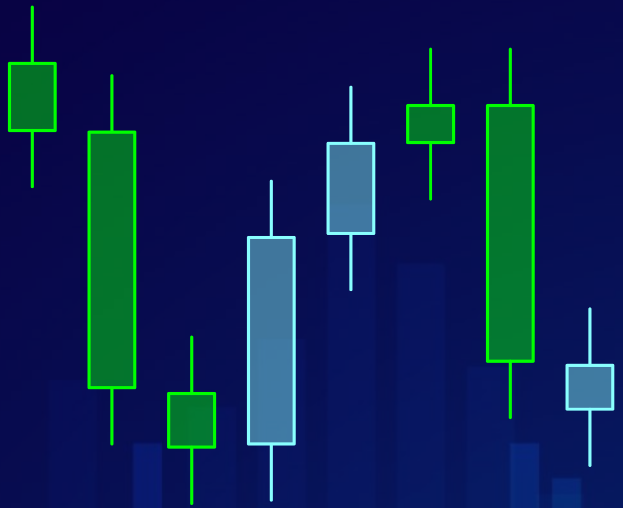
Lesson #4

The Problem

1. Initial challenge was deciding which aspects of the data to graph.
2. Once the visuals were created, extracting meaningful insights demanded a thorough understanding of the dataset
3. Selecting the right visuals was crucial.

The Lesson

1. Deeper understanding of the dataset for better analysis
2. Experimenting with different graphs and refining them based on insights for overall quality of the visuals



END

CREDITS: This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics & images by **Freepik**