Project 3
NASA API
Asteroids data

- Maliha Mir
- Olive McKenzie
- Marta Rychel
- Brad Smart

Data sources

- The project focuses on the data from asteroids

(01 .01.2023 – 08.01.2023)

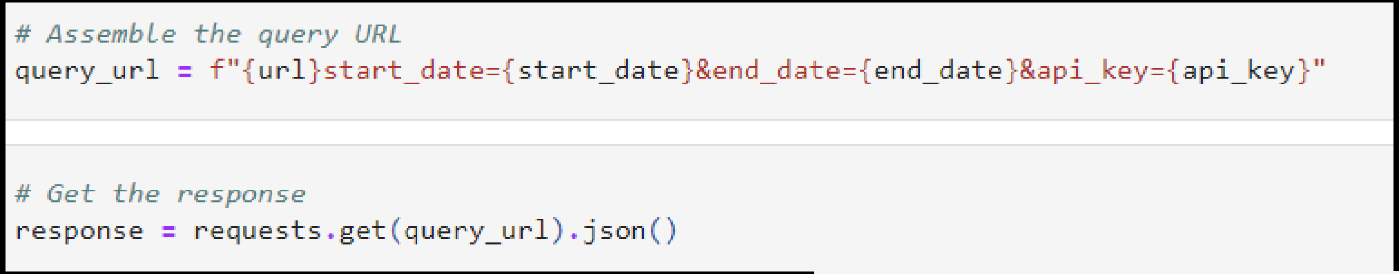
- Data has been extracted from :

➢ https://api.nasa.gov/neo/rest/v1/feed?start_date=2015-09-07&end_date=2015-09-14&api_key=DEMO_KEY

- Look in to asteroids to see how many are potentially hazardous, and check the properties.

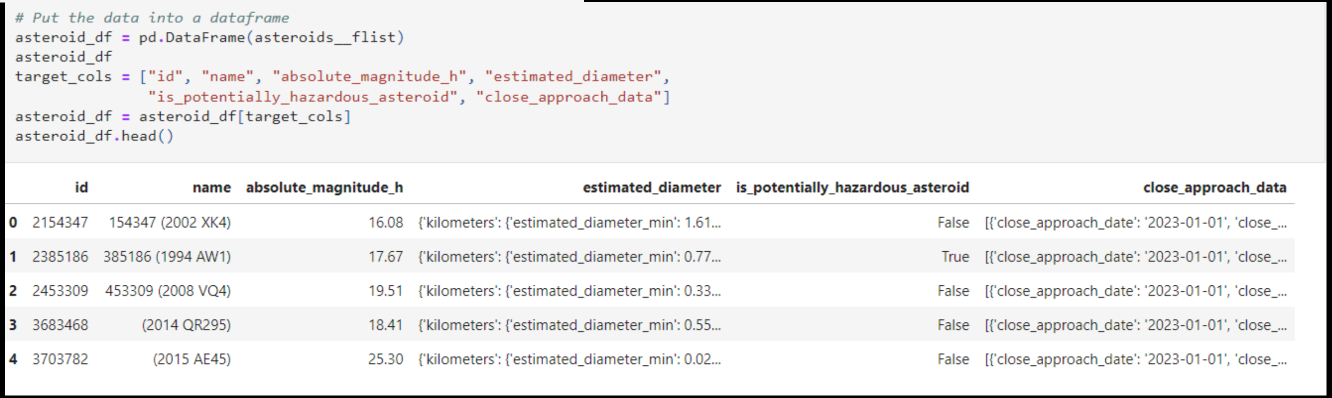
Data Collection

- As previously mentioned we used an API call to collect the data from the NASA API. We then used pandas to clean the data and then create a final dataframe. We had to extract the data out of the estimated diameter and close approach data columns



```
# Assemble the query URL
query_url = f"{url}start_date={start_date}&end_date={end_date}&api_key={api_key}"
```

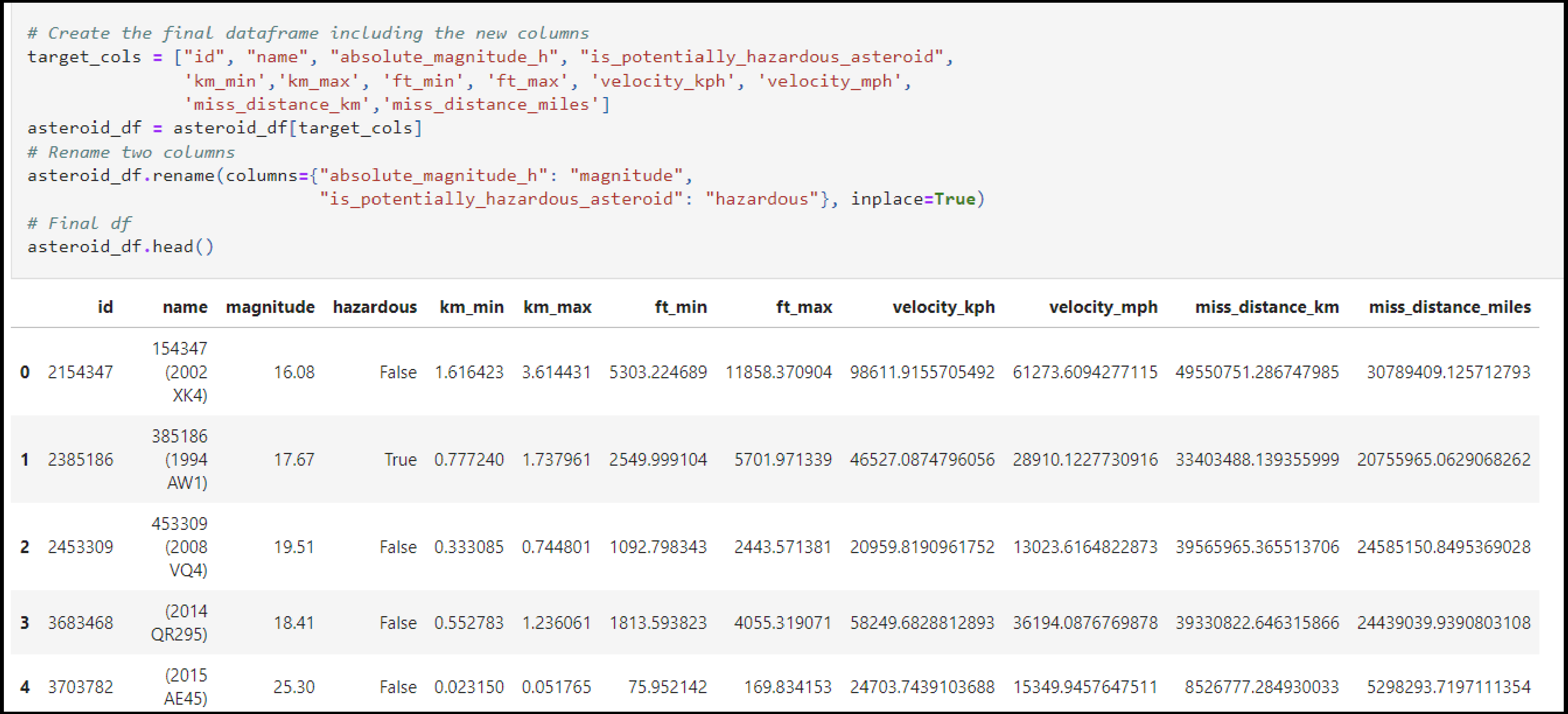
```
# Get the response
response = requests.get(query_url).json()
```



```
# Put the data into a dataframe
asteroid_df = pd.DataFrame(asteroids__flist)
asteroid_df
target_cols = ["id", "name", "absolute_magnitude_h", "estimated_diameter",
               "is_potentially_hazardous_asteroid", "close_approach_data"]
asteroid_df = asteroid_df[target_cols]
asteroid_df.head()
```

	id	name	absolute_magnitude_h	estimated_diameter	is_potentially_hazardous_asteroid	close_approach_data
0	2154347	154347 (2002 XK4)	16.08	{'kilometers': {'estimated_diameter_min': 1.61...	False	[{'close_approach_date': '2023-01-01', 'close_...
1	2385186	385186 (1994 AW1)	17.67	{'kilometers': {'estimated_diameter_min': 0.77...	True	[{'close_approach_date': '2023-01-01', 'close_...
2	2453309	453309 (2008 VQ4)	19.51	{'kilometers': {'estimated_diameter_min': 0.33...	False	[{'close_approach_date': '2023-01-01', 'close_...
3	3683468	(2014 QR295)	18.41	{'kilometers': {'estimated_diameter_min': 0.55...	False	[{'close_approach_date': '2023-01-01', 'close_...
4	3703782	(2015 AE45)	25.30	{'kilometers': {'estimated_diameter_min': 0.02...	False	[{'close_approach_date': '2023-01-01', 'close_...

Final Dataframe

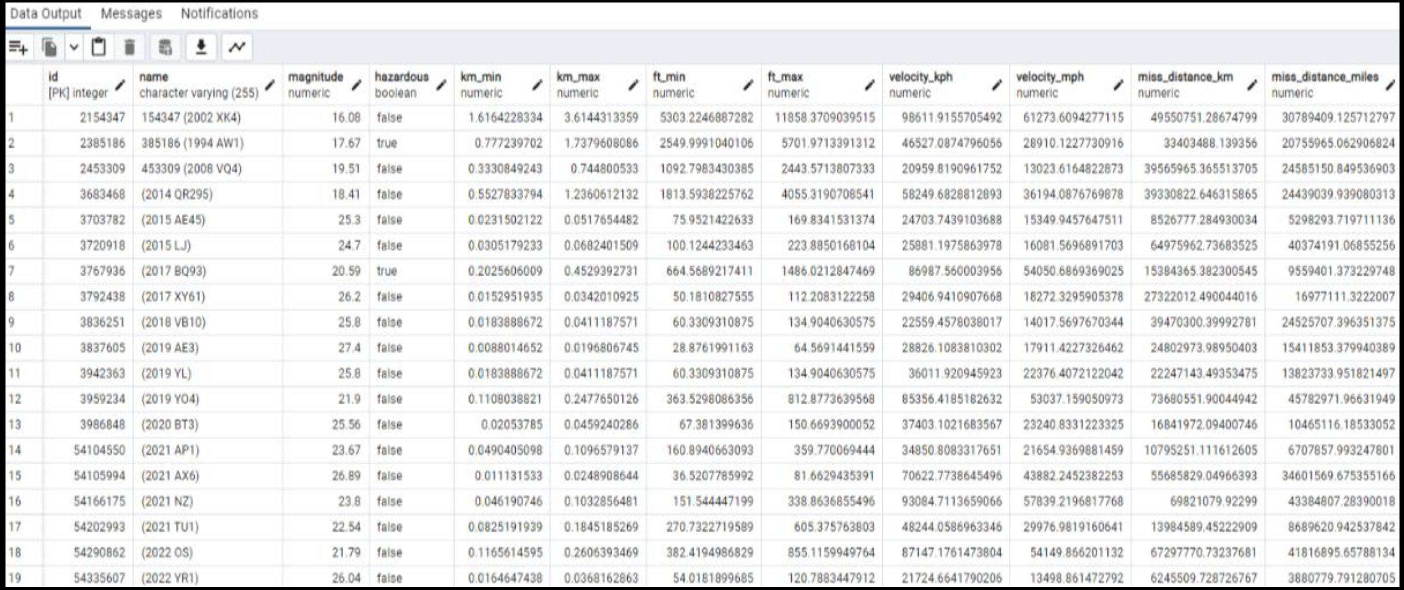


```
# Create the final dataframe including the new columns
target_cols = ["id", "name", "absolute_magnitude_h", "is_potentially_hazardous_asteroid",
               'km_min','km_max', 'ft_min', 'ft_max', 'velocity_kph', 'velocity_mph',
               'miss_distance_km','miss_distance_miles']
asteroid_df = asteroid_df[target_cols]
# Rename two columns
asteroid_df.rename(columns={"absolute_magnitude_h": "magnitude",
                            "is_potentially_hazardous_asteroid": "hazardous"}, inplace=True)
# Final df
asteroid_df.head()
```

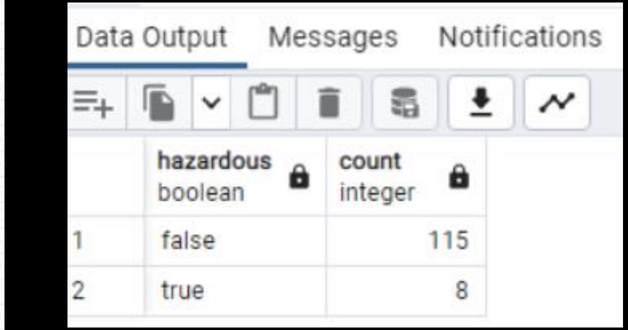
	id	name	magnitude	hazardous	km_min	km_max	ft_min	ft_max	velocity_kph	velocity_mph	miss_distance_km	miss_distance_miles
0	2154347	154347 (2002 XK4)	16.08	False	1.616423	3.614431	5303.224689	11858.370904	98611.9155705492	61273.6094277115	49550751.286747985	30789409.125712793
1	2385186	385186 (1994 AW1)	17.67	True	0.777240	1.737961	2549.999104	5701.971339	46527.0874796056	28910.1227730916	33403488.139355999	20755965.0629068262
2	2453309	453309 (2008 VQ4)	19.51	False	0.333085	0.744801	1092.798343	2443.571381	20959.8190961752	13023.6164822873	39565965.365513706	24585150.8495369028
3	3683468	(2014 QR295)	18.41	False	0.552783	1.236061	1813.593823	4055.319071	58249.6828812893	36194.0876769878	39330822.646315866	24439039.9390803108
4	3703782	(2015 AE45)	25.30	False	0.023150	0.051765	75.952142	169.834153	24703.7439103688	15349.9457647511	8526777.284930033	5298293.7197111354

Database technology used

- Data was loaded into a relational database for storage. ‘PGAdmin 4’ was used to create PostgreSQL tables that included the headers from the dataframe.
- We used PostgreSQL to store our data so that run queries and to be able to create additional tables, and to help us inspect the data.



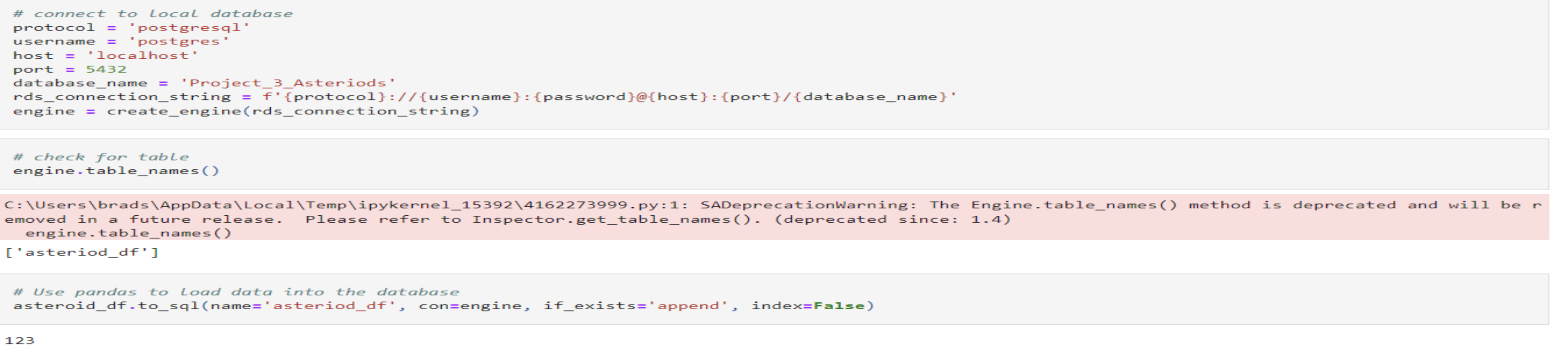
	id [PK] integer	name character varying (255)	magnitude numeric	hazardous boolean	km_min numeric	km_max numeric	ft_min numeric	ft_max numeric	velocity_kph numeric	velocity_mph numeric	miss_distance_km numeric	miss_distance_miles numeric
1	2154347	154347 (2002 XK4)	16.08	false	1.6164228334	3.6144313359	5303.2246887282	11858.3709039515	98611.9155705492	61273.6094277115	49550751.28674799	30789409.125712797
2	2385186	385186 (1994 AW1)	17.67	true	0.777239702	1.7379608086	2549.9991040106	5701.9713391312	46527.0874796056	28910.1227730916	33403488.139356	20755965.062906824
3	2453309	453309 (2008 VQ4)	19.51	false	0.3330849243	0.744800533	1092.7983430385	2443.5713807333	20959.8190961752	13023.6164822873	39565965.365513705	24585150.849536903
4	3683468	(2014 QR295)	18.41	false	0.5527833794	1.2360612132	1813.5938225762	4055.3190708541	58249.6828812893	36194.0876769878	39330822.646315865	24439039.939080313
5	3703782	(2015 AE45)	25.3	false	0.0231502122	0.0517654482	75.9521422633	169.8341531374	24703.7439103688	15349.9457647511	8526777.284930034	5298293.719711136
6	3720918	(2015 LJ)	24.7	false	0.0305179233	0.0682401509	100.1244233463	223.8850168104	25881.1975863978	16081.5696891703	64975962.73683525	40374191.06855256
7	3767936	(2017 BQ93)	20.59	true	0.2025606009	0.4529392731	664.5689217411	1486.0212847469	86987.560003956	54050.6869369025	15384365.382300545	9559401.373229748
8	3792438	(2017 XY61)	26.2	false	0.0152951935	0.0342010925	50.1810827555	112.2083122258	29406.9410907668	18272.3295905378	27322012.490044016	16977111.3222007
9	3836251	(2018 VB10)	25.8	false	0.0183888672	0.0411187571	60.3309310875	134.9040630575	22559.4578038017	14017.5697670344	39470300.39992781	24525707.396351375
10	3837605	(2019 AE3)	27.4	false	0.0088014652	0.0196806745	28.8761991163	64.5691441559	28826.1083810302	17911.4227326462	24802973.98950403	15411853.379940389
11	3942363	(2019 YL)	25.8	false	0.0183888672	0.0411187571	60.3309310875	134.9040630575	36011.920945923	22376.4072122042	22247143.49353475	13823733.951821497
12	3959234	(2019 YO4)	21.9	false	0.1108038821	0.2477650126	363.5298086356	812.8773639568	85356.4185182632	53037.159050973	73680551.90044942	45782971.96631949
13	3986848	(2020 BT3)	25.56	false	0.02053785	0.0459240286	67.381399636	150.6693900052	37403.1021683567	23240.8331223325	16841972.09400746	10465116.18533052
14	54104550	(2021 AP1)	23.67	false	0.0490405098	0.1096579137	160.8940663093	359.770069444	34850.8083317651	21654.9369881459	10795251.111612605	6707857.993247801
15	54105994	(2021 AX6)	26.89	false	0.011131533	0.0248908644	36.5207785992	81.6629435391	70622.7738645496	43882.2452382253	55685829.04966393	34601569.675355166
16	54166175	(2021 NZ)	23.8	false	0.046190746	0.1032856481	151.544447199	338.8636855496	93084.7113659066	57839.2196817768	69821079.92299	43384807.28390018
17	54202993	(2021 TU1)	22.54	false	0.0825191939	0.1845185269	270.7322719589	605.375763803	48244.0586963346	29976.9819160641	13984589.45222909	8689620.942537842
18	54290862	(2022 OS)	21.79	false	0.1165614595	0.2606393469	382.4194986829	855.1159949764	87147.1761473804	54149.866201132	67297770.73237681	41816895.65788134
19	54335607	(2022 YR1)	26.04	false	0.0164647438	0.0368162863	54.0181899685	120.7883447912	21724.6641790206	13498.861472792	6245509.728726767	3880779.791280705



	hazardous boolean	count integer
1	false	115
2	true	8

Load the data on to the DB

- A localhost connection to a PostgreSQL server was created and a connection made to it. The connection was made via an engine on Jupyter Notebook that could talk to the database.



```
# connect to local database
protocol = 'postgresql'
username = 'postgres'
host = 'localhost'
port = 5432
database_name = 'Project_3_Asteriods'
rds_connection_string = f'{protocol}://{username}:{password}@{host}:{port}/{database_name}'
engine = create_engine(rds_connection_string)
```

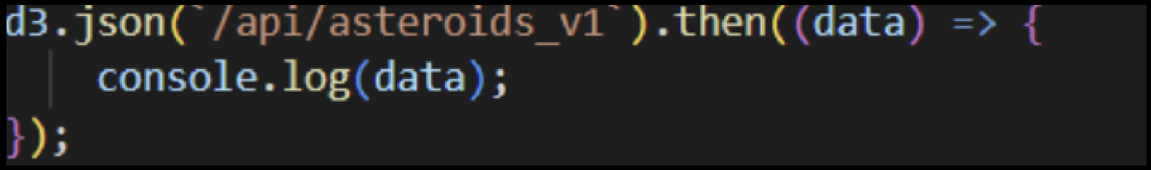
```
# check for table
engine.table_names()
```

```
C:\Users\brads\AppData\Local\Temp\ipykernel_15392\4162273999.py:1: SADeprecationWarning: The Engine.table_names() method is deprecated and will be r
emoved in a future release.  Please refer to Inspector.get_table_names(). (deprecated since: 1.4)
  engine.table_names()
['asteriod_df']
```

```
# Use pandas to load data into the database
asteroid_df.to_sql(name='asteriod_df', con=engine, if_exists='append', index=False)
```

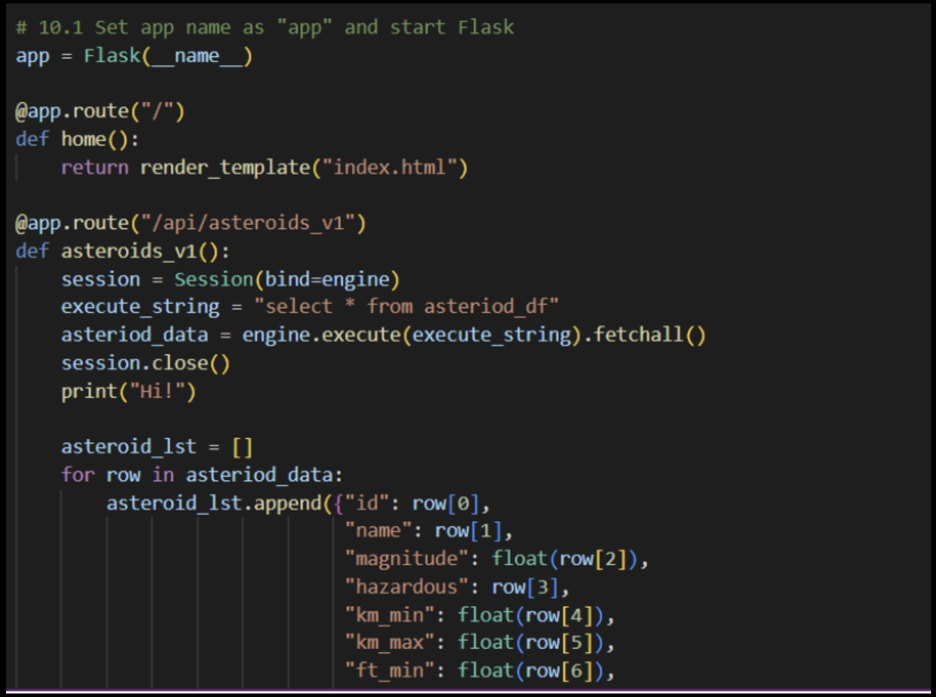
```
123
```

Flask app.py

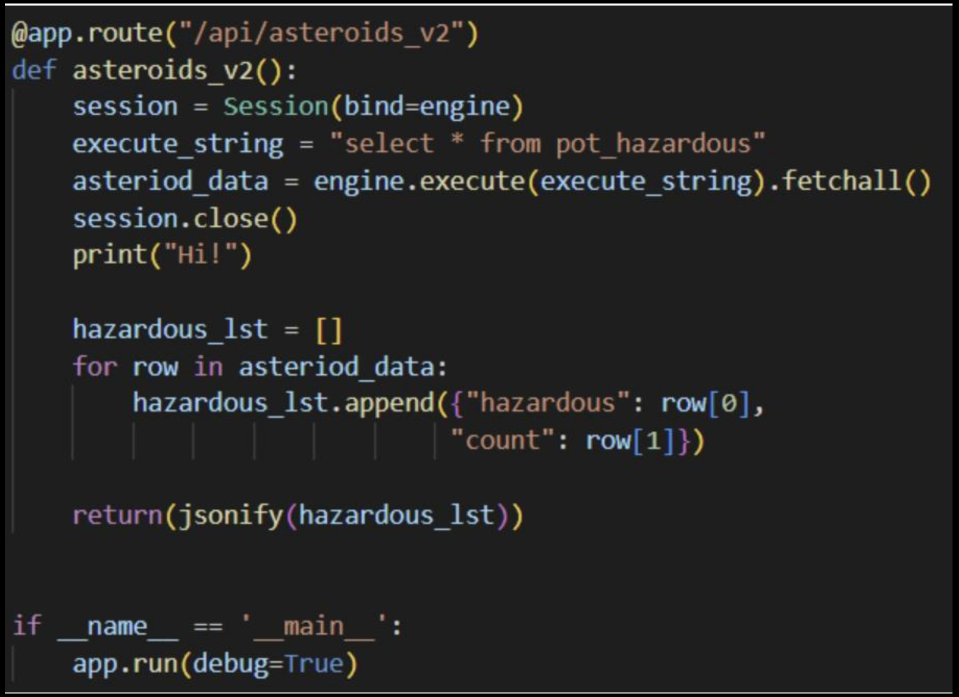


```
d3.json(`/api/asteroids_v1`).then((data) => {
    console.log(data);
});
```

- We have then used Flask to be able to use d3 to get the data into our javascript file to be able to create the visualisations for our dashboard

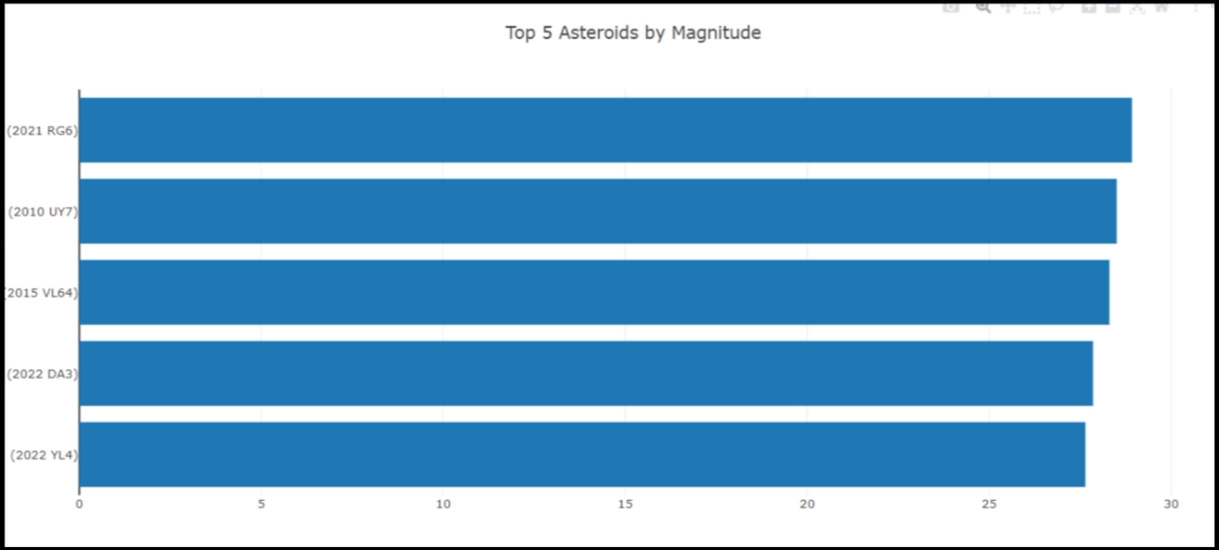


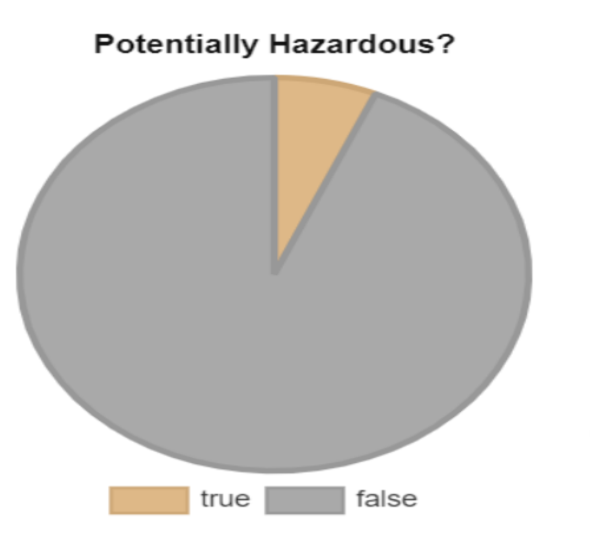
```
# 10.1 Set app name as "app" and start Flask
app = Flask(__name__)

@app.route("/")
def home():
    return render_template("index.html")

@app.route("/api/asteroids_v1")
def asteroids_v1():
    session = Session(bind=engine)
    execute_string = "select * from asteriod_df"
    asteriod_data = engine.execute(execute_string).fetchall()
    session.close()
    print("Hi!")

    asteroid_lst = []
    for row in asteriod_data:
        asteroid_lst.append({"id": row[0],
                             "name": row[1],
                             "magnitude": float(row[2]),
                             "hazardous": row[3],
                             "km_min": float(row[4]),
                             "km_max": float(row[5]),
                             "ft_min": float(row[6]),
```

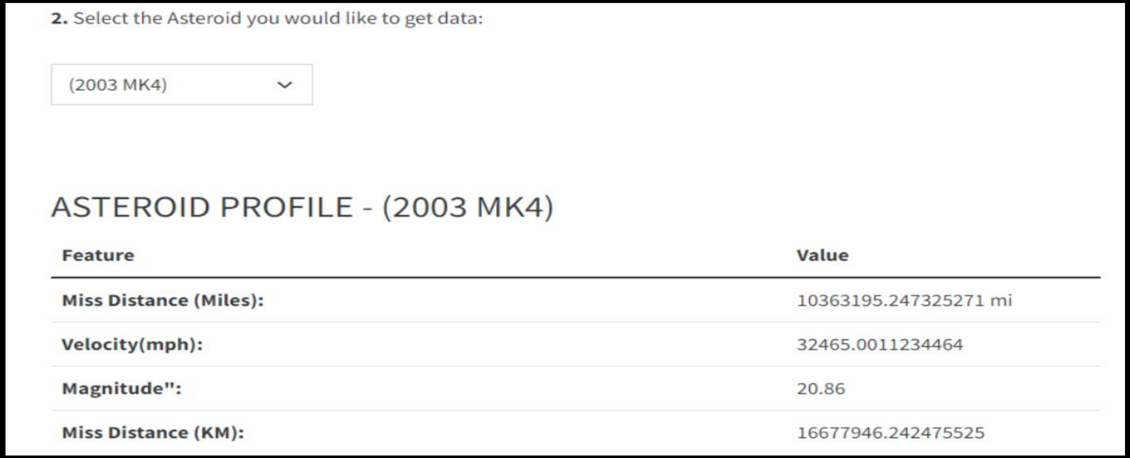


```
@app.route("/api/asteroids_v2")
def asteroids_v2():
    session = Session(bind=engine)
    execute_string = "select * from pot_hazardous"
    asteriod_data = engine.execute(execute_string).fetchall()
    session.close()
    print("Hi!")

    hazardous_lst = []
    for row in asteriod_data:
        hazardous_lst.append({"hazardous": row[0],
                              "count": row[1]})

    return(jsonify(hazardous_lst))


if __name__ == '__main__':
    app.run(debug=True)
```

Visualization





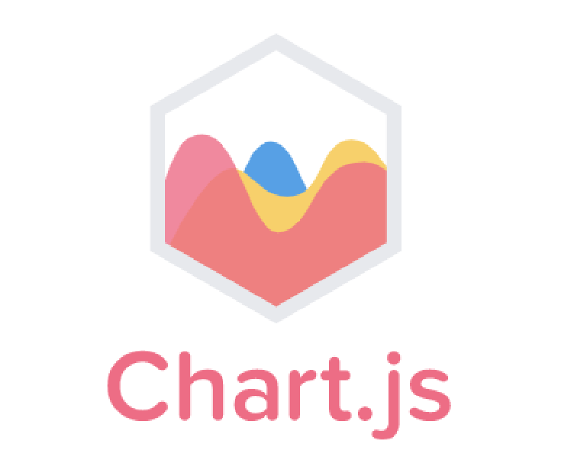

2. Select the Asteroid you would like to get data:

(2003 MK4)

ASTEROID PROFILE - (2003 MK4)

Feature	Value
Miss Distance (Miles):	10363195.247325271 mi
Velocity(mph):	32465.0011234464
Magnitude":	20.86
Miss Distance (KM):	16677946.242475525

JavaScript libraries



- We used 2 libraries in our project
- Plotly to create bar chart
- D3 to select data and create the table
- Chart.js to create pie chart

What we could do differently ?
Challenges

- 1. Next time we would like to spend more time to visualise the relationships between variables in our data.
- 2. Create more charts to be able to visualise the data
- 3. Faced challenges in overcoming bugs in our code and linking our java script to our html

Thank you for your attention