IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

IBM Data Science Capstone Project:
Space X Falcon 9 Landing Analysis

Michelle Paniagua
28-04-2025

Outline

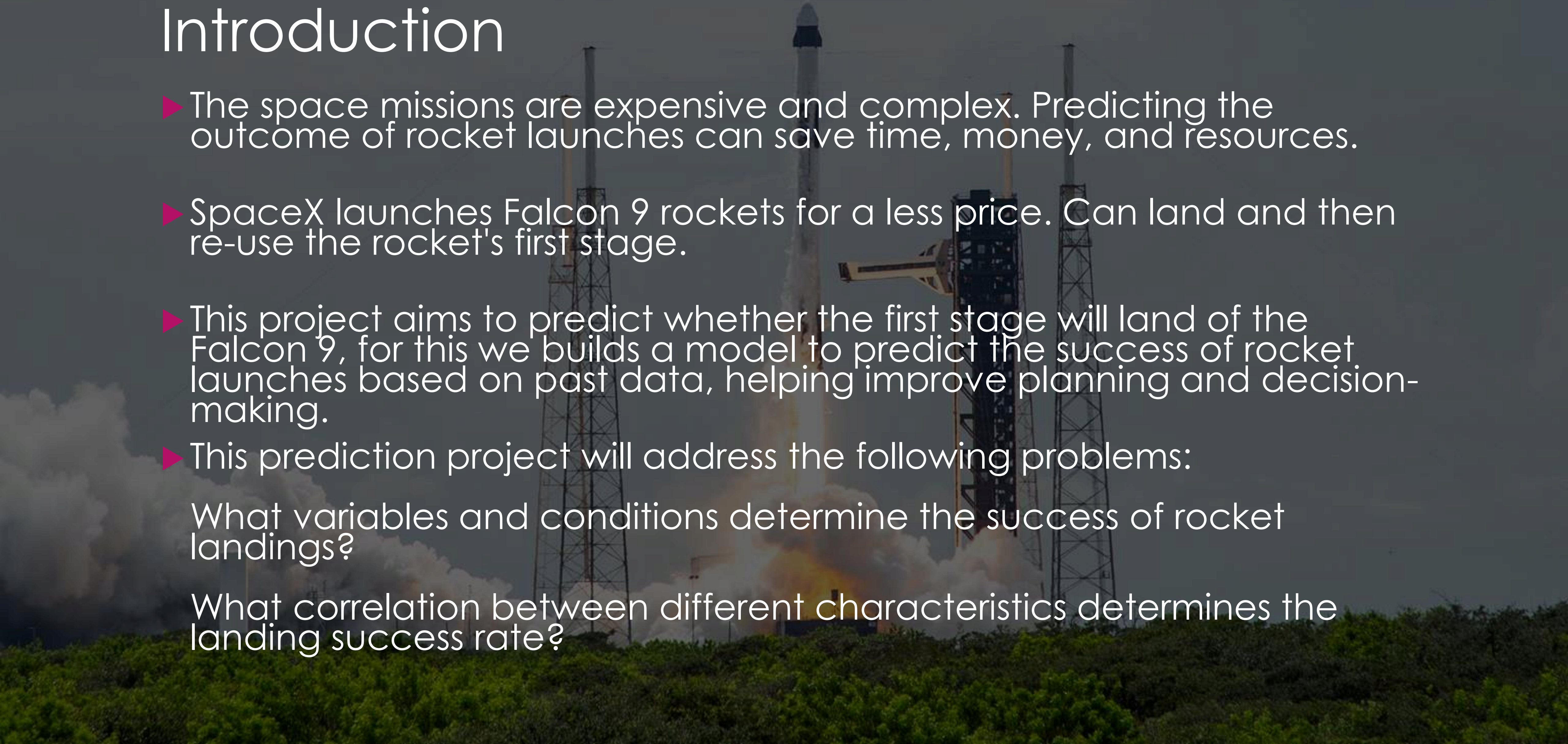
Executive Summary

- Summary of methodologies
- Data Collection
- Exploratory Data Analysis
- Interactive Visual Analytics
- Predictive Analysis (Classification)

Summary of all results

- Exploratory Data Analysis (EDA) results
- Geospatial analytics
- Interactive dashboard
- Predictive analysis of classification models

Introduction



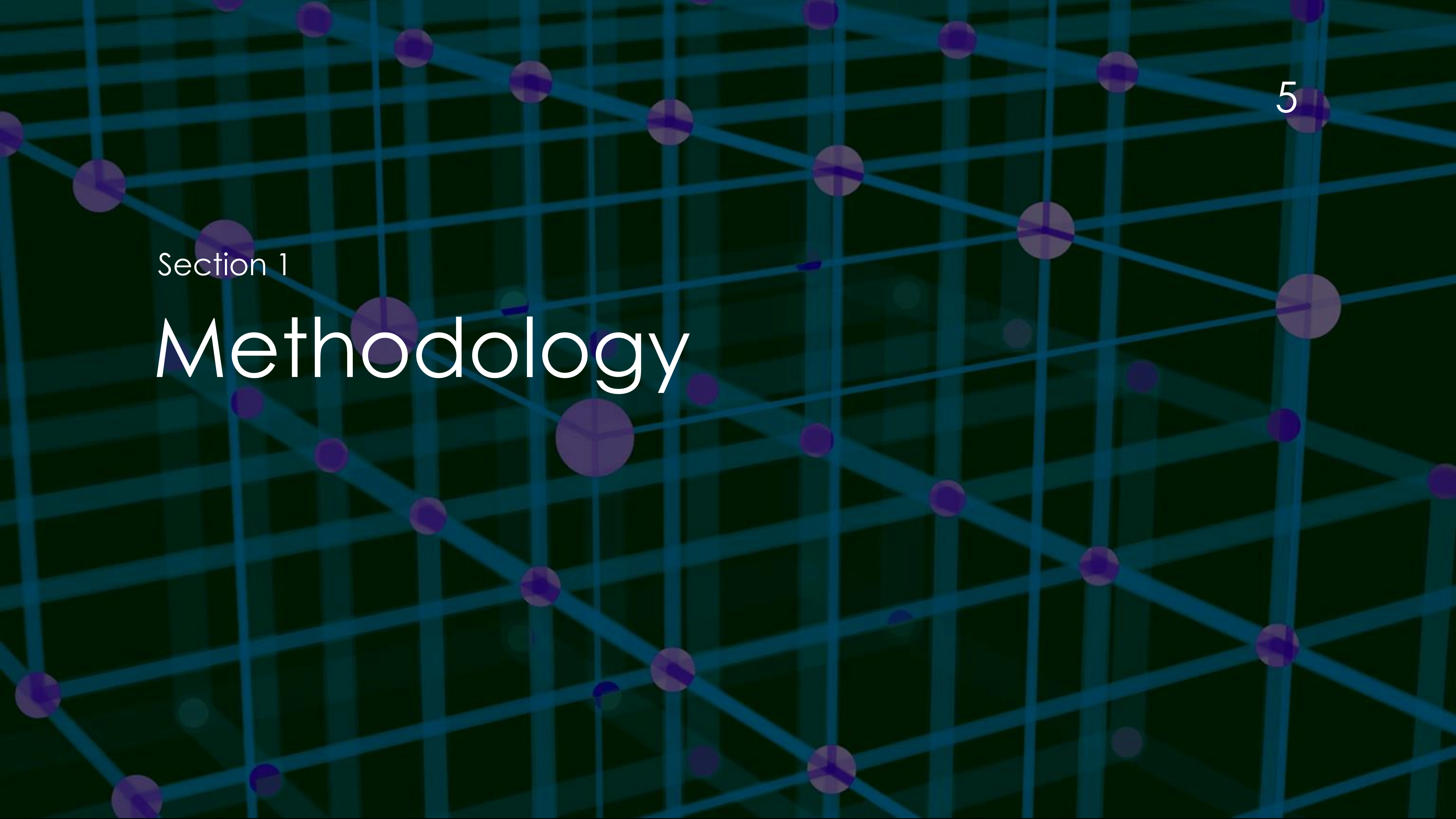
- The space missions are expensive and complex. Predicting the outcome of rocket launches can save time, money, and resources.
- SpaceX launches Falcon 9 rockets for a less price. Can land and then re-use the rocket's first stage.
- This project aims to predict whether the first stage will land of the Falcon 9, for this we builds a model to predict the success of rocket launches based on past data, helping improve planning and decision-making.
- This prediction project will address the following problems:

  What variables and conditions determine the success of rocket landings?

  What correlation between different characteristics determines the landing success rate?

Section 1

Methodology



Methodology Summary

1. Data collection:

- REST API
- Web Scraping

2. Perform data wrangling

- The .fillna() method was used to remove null values.
- The .value_counts() method was used to determine the number of launches at each site, the number of orbits in each location, and mission results by orbit type.
- A landing result label was created. A 0 indicates a failure to land, and a 1 indicates a successful landing.

3. Perform exploratory data analysis (EDA) using visualization and SQL

- SQL queries are used to manipulate and evaluate the dataset.
- Pandas and Matplotlib libraries are used to generate visualizations and observe relationships between variables and identify patterns.

4. Perform interactive visual analytics using Folium and Plotly Dash

- A geospatial analysis was performed using Folium.
- An interactive dashboard was created using Plotly Dash.

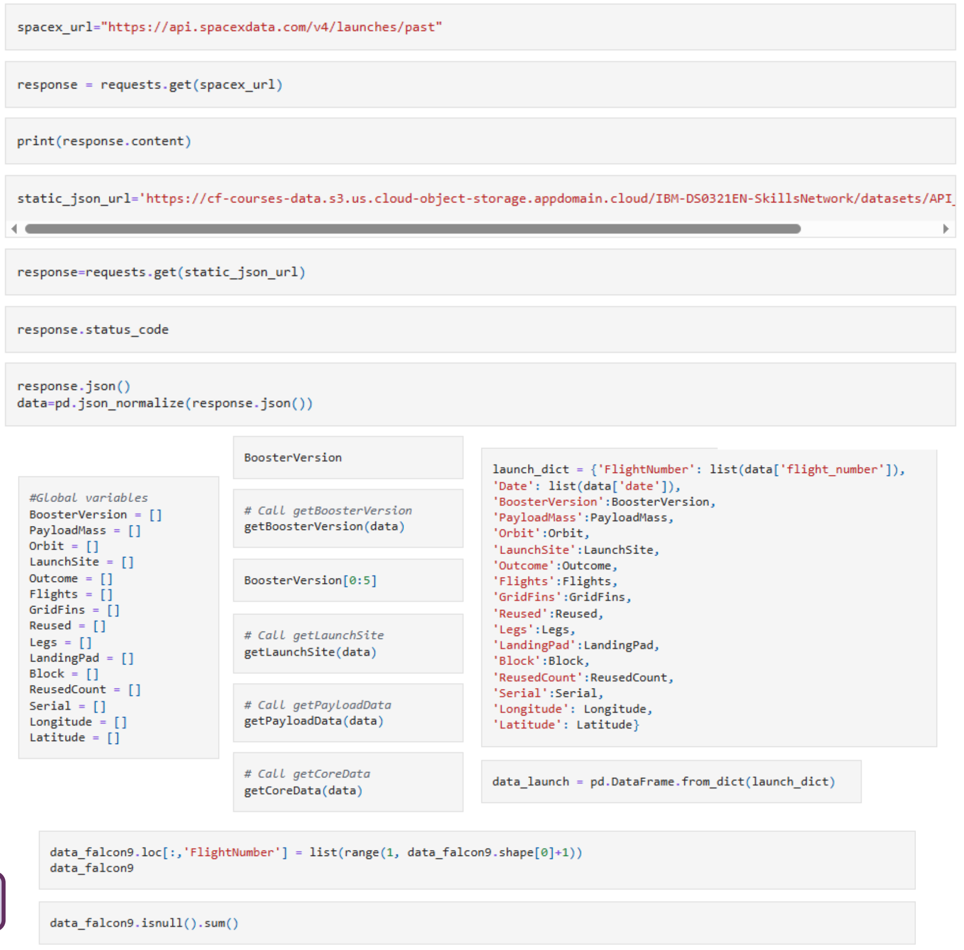
5. Perform predictive analysis using classification models

- The Scikit-Learn library was used to preprocess and standardize the data, splitting the data into training and test data. Classification models were trained and hyperparameters were identified using GridSearchCV.
- Confusion matrices for the classification models were plotted.
- The accuracy of the classification models was evaluated.

Data Collection – SpaceX API

- 1. Sent a GET request to the Rocket Launch REST API ➡️ Retrieved data on rockets, payloads, launch sites, and launch results.
- 2. Converted the API to JSON format and loaded it into a Pandas DataFrame. ➡️ To explore and manipulate the data.
- 3. Performed data cleaning:
  Managed missing values.
  Defined lists to organize the data.
  Extracted key columns such as rocket name, launch date, payload mass, etc.
- 4. Created the dataset:
  A dictionary with the extracted lists.
  Created a new Pandas DataFrame from the dictionary.
- 5. Filtered the dataset to contain only Falcon 9 launches.
  Reset the FlightNumber index.
  Replace missing PayloadMass values with the average payload mass.

https://github.com/Michu-code/Class-IBM/blob/main/1.%20jupyter-labs-spacex-data-collection-api.ipynb



```
spacex_url="https://api.spacexdata.com/v4/launches/past"
```

```
response = requests.get(spacex_url)
```

```
print(response.content)
```

```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_
```

```
response=requests.get(static_json_url)
```

```
response.status_code
```

```
response.json()
data=pd.json_normalize(response.json())
```

```
#Global variables
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
ReusedCount = []
Serial = []
Longitude = []
Latitude = []
```

```
BoosterVersion
```

```
# Call getBoosterVersion
getBoosterVersion(data)
```

```
BoosterVersion[0:5]
```

```
# Call getLaunchSite
getLaunchSite(data)
```

```
# Call getPayloadData
getPayloadData(data)
```

```
# Call getCoreData
getCoreData(data)
```

```
launch_dict = {'FlightNumber': list(data['flight_number']),
'Date': list(data['date']),
'BoosterVersion':BoosterVersion,
'PayloadMass':PayloadMass,
'Orbit':Orbit,
'LaunchSite':LaunchSite,
'Outcome':Outcome,
'Flights':Flights,
'GridFins':GridFins,
'Reused':Reused,
'Legs':Legs,
'LandingPad':LandingPad,
'Block':Block,
'ReusedCount':ReusedCount,
'Serial':Serial,
'Longitude': Longitude,
'Latitude': Latitude}
```

```
data_launch = pd.DataFrame.from_dict(launch_dict)
```

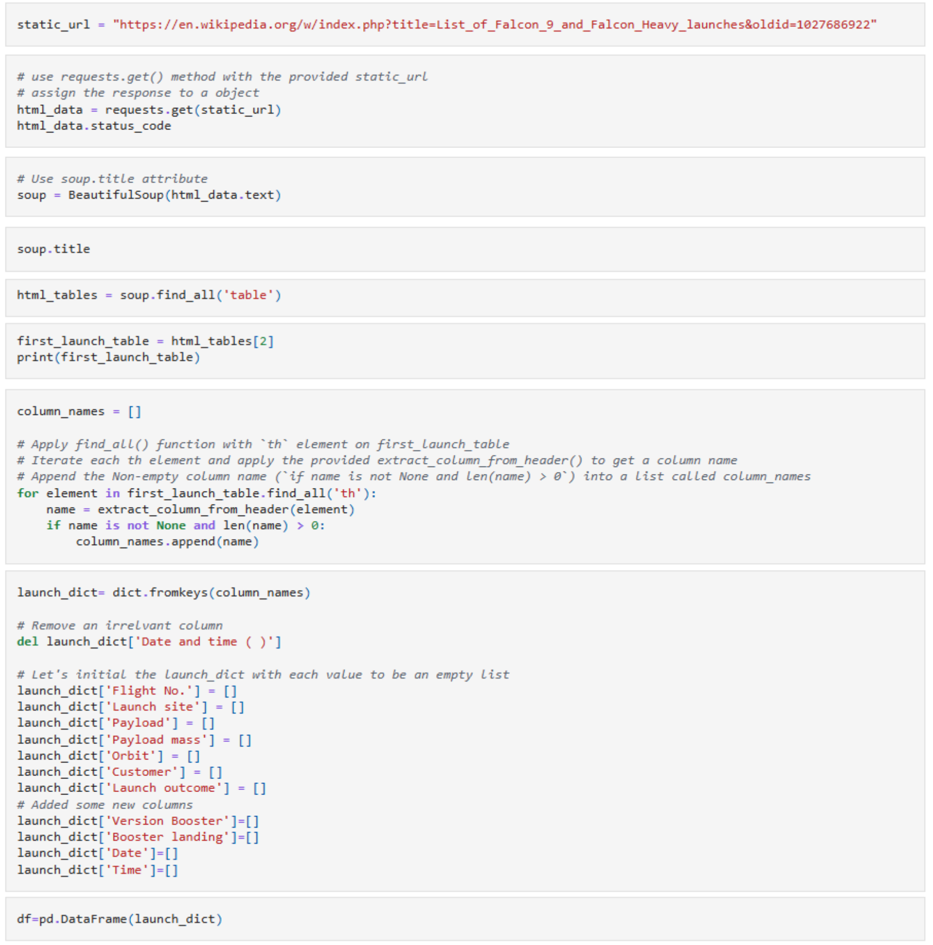
```
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9
```

```
data_falcon9.isnull().sum()
```

Data Collection - Web Scraping

- 1. A request was sent to a URL containing the launch history. ➡ The HTML content of the page was retrieved to navigate through the HTML elements.
- 2. A BeautifulSoup object was created from the HTML to allow navigation through the HTML elements.
- 3. The HTML tables containing launch logs were searched and extracted. ➡ The tables with launch data, payloads, and results were located.
- 4. The columns from the tables were extracted and a dictionary was initialized as keys ➡ The data storage was prepared.
- 5. Analysis functions were used to populate the dictionary with row data from the tables. ➡ Cases of missing data or irregular formats were handled.
- 6. The dictionary was converted into a DataFrame for analysis or export.

https://github.com/Michu-code/Class-IBM/blob/main/2.%20jupyter-labs-webscraping.ipynb



```
static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"
```

```
# use requests.get() method with the provided static_url
# assign the response to a object
html_data = requests.get(static_url)
html_data.status_code
```

```
# Use soup.title attribute
soup = BeautifulSoup(html_data.text)
```

```
soup.title
```

```
html_tables = soup.find_all('table')
```

```
first_launch_table = html_tables[2]
print(first_launch_table)
```

```
column_names = []

# Apply find_all() function with `th` element on first_launch_table
# Iterate each th element and apply the provided extract_column_from_header() to get a column name
# Append the Non-empty column name (`if name is not None and len(name) > 0`) into a list called column_names
for element in first_launch_table.find_all('th'):
    name = extract_column_from_header(element)
    if name is not None and len(name) > 0:
        column_names.append(name)
```

```
launch_dict= dict.fromkeys(column_names)

# Remove an irrelvant column
del launch_dict['Date and time ( )']

# Let's initial the launch_dict with each value to be an empty list
launch_dict['Flight No.'] = []
launch_dict['Launch site'] = []
launch_dict['Payload'] = []
launch_dict['Payload mass'] = []
launch_dict['Orbit'] = []
launch_dict['Customer'] = []
launch_dict['Launch outcome'] = []
# Added some new columns
launch_dict['Version Booster']=[]
launch_dict['Booster landing']=[]
launch_dict['Date']=[]
launch_dict['Time']=[]
```

```
df=pd.DataFrame(launch_dict)
```

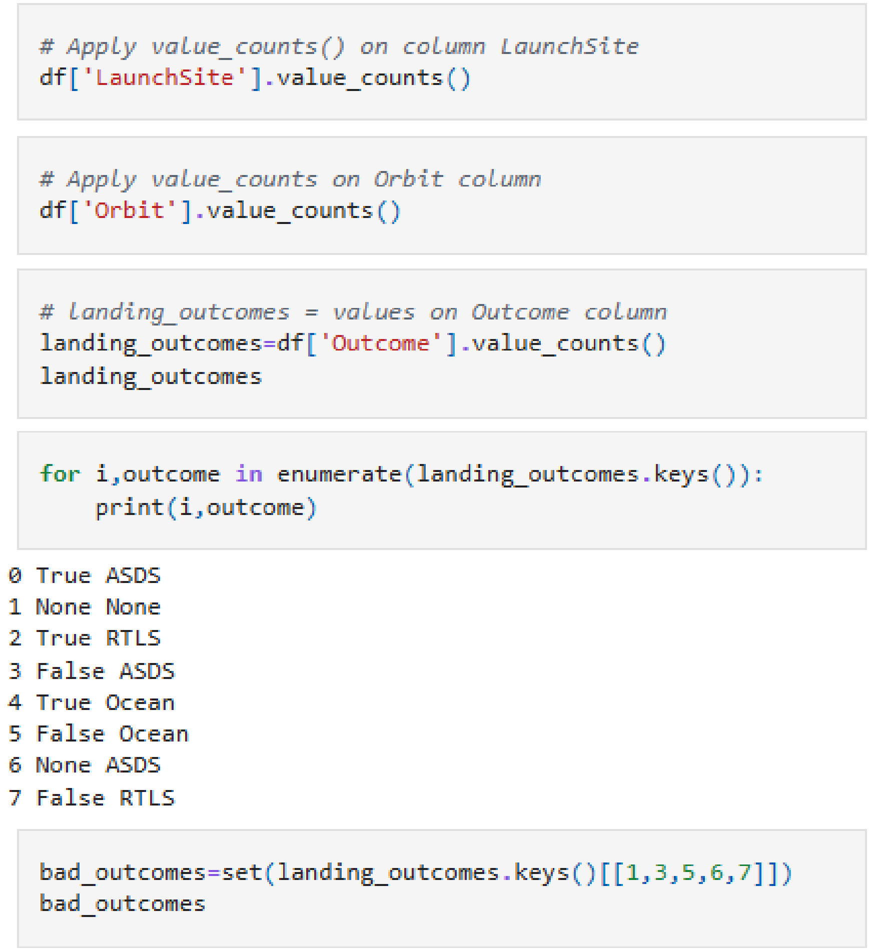
Data Wrangling

- The rocket launch dataset has multiple launch sites and target orbits. The outcome of each mission is recorded in the "Outcome" column, including different types of landings, whether in the ocean, on a drone ship, or on a land platform.

Data exploration:

- 1.Identify missing data:
  Calculated missing values percentage for each column.
  Found missing values mainly in the LandingPad column.
- 2. Data type:
  Identified categorical and numerical columns (like FlightNumber as integer, orbit as object).
- 3. Analyze launch sites:
  Used .value_counts() to find the number of launches per site.
- 4. Analyze orbit types:
  Used .value_counts() to determine the distribution of orbit types.
- 5. Analyze mission outcomes:
  Used .value_counts() to explore the different landing outcomes.

https://github.com/Michu-code/Class-IBM/blob/main/3.%20labs-jupyter-spacex-Data%20wrangling.ipynb



```
# Apply value_counts() on column LaunchSite
df['LaunchSite'].value_counts()
```

```
# Apply value_counts on Orbit column
df['Orbit'].value_counts()
```

```
# landing_outcomes = values on Outcome column
landing_outcomes=df['Outcome'].value_counts()
landing_outcomes
```

```
for i,outcome in enumerate(landing_outcomes.keys()):
    print(i,outcome)
```

```
0 True ASDS
1 None None
2 True RTLS
3 False ASDS
4 True Ocean
5 False Ocean
6 None ASDS
7 False RTLS
```

```
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
bad_outcomes
```

Data Wrangling

Data Labeling:

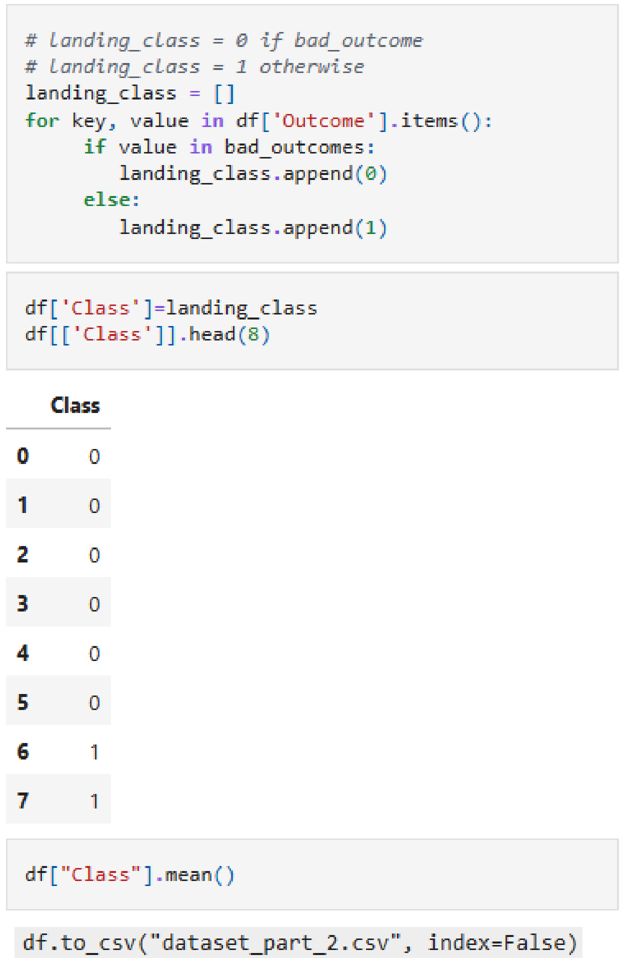
- 6. Create landing outcome labels:
  Defined unsuccessful outcomes (bad_outcomes set)
  Created a class column:
  1 = successful landing
  0 = failed landing

- 7. Calculate success rate:
  Computed the mean of the Class column to find overall landing success rate.

- 8. Export data:
  Saved the cleaned dataset as dataset_part_2.csv.

Landing Outcome Details:
True Ocean – Successful landing in the ocean
False Ocean – Failed landing in the ocean
True RTLS – Successful landing on a ground pad
False RTLS – Failed landing on a ground pad
True ASDS – Successful landing on a drone ship
False ASDS – Failed landing on a drone ship
None ASDS / None None – No landing attempt or missing data

https://github.com/Michu-code/Class-IBM/blob/main/3.%20labs-jupyter-spacex-Data%20wrangling.ipynb



```python
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
landing_class = []
for key, value in df['Outcome'].items():
    if value in bad_outcomes:
        landing_class.append(0)
    else:
        landing_class.append(1)
```

```python
df['Class']=landing_class
df[['Class']].head(8)
```

```
   Class
0      0
1      0
2      0
3      0
4      0
5      0
6      1
7      1
```

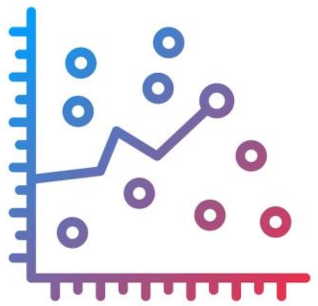
```python
df["Class"].mean()
```

```python
df.to_csv("dataset_part_2.csv", index=False)
```

EDA with Data Visualization

Scatter Plot

- Explore relationships between two numerical variables.
- Identify trends, patterns, or clusters in the data.



Bar chart

- Compare a numerical value across categories.
- Visualize differences in success rates among different orbit types.

Line Chart

- Show trends over time.
- Track the progress or change of a metric across years.

EDA with SQL

This process involves using SQL queries to explore and retrieve specific information from a dataset related to space missions. These queries include:

- 1. Listing the names of launch sites across missions.
- 2. Displaying five records where launch sites begin with "CCA."
- 3. Calculating the total payload mass carried by rockets launched by NASA (CRS).
- 4. Obtaining the average payload weight of the F9 v1.1 booster.
- 5. Finding the date of the first successful landing on land.
- 6. Identifying the names of successful boosters on unmanned spacecraft that carried payloads between 4,000 and 6,000 kg.
- 7. Counting the total number of successful and unsuccessful missions.
- 8. Determining which booster versions carried the largest payloads.
- 9. List the failed unmanned spacecraft landings in 2015, including booster versions and launch sites.
- 10. Classify the landings (as "Failure" or "Success") over a date range from June 4, 2010, to June 4, 2013, in descending order.

https://github.com/Michu-code/Class-IBM/blob/main/4.%20jupyter-labs-eda-sql-coursera_sqllite.ipynb

Build an Interactive Map with Folium

This brief explains how different objects were created and added to an interactive map to visualize launch data.

- All launch sites were marked on the map using a Folium Map object.
- For each site, a circle and a marker were added to identify its location.
- To display successful and failed launches at each site, the events were grouped together, as many shared the same coordinates.
- Before grouping, each marker was assigned a color: green for successful launches (class = 1) and red for failed launches (class = 0).
- A MarkerCluster was used to group the launches, adding a marker for each with a text icon indicating the distance, whose color matched the launch status.
- To analyze the proximity of the launch sites, the distances between points were calculated using their latitude and longitude coordinates.
- Markers were created to show these distances and lines were drawn (using folium.PolyLine) to visualize the connections between sites and their proximities on the map.

https://github.com/Michu-code/Class-IBM/blob/main/6.%20lab_jupyter_launch_site_location.ipynb

Build a Dashboard with Plotly Dash

Two types of charts were incorporated into a Plotly Dash panel to facilitate interactive data visualization.

First, the corresponding libraries were imported to develop the web application in Python. These include dash_html_components, dash_core_components, dash.dependencies, and plotly.express.

- **Pie chart**

Using px.pie(), which illustrates the number of successful launches at each site, allowing us to identify which are the most successful.

Additionally, dcc.Dropdown() was added to allow filtering the chart using a dropdown menu and thus analyze the success or failure rate for a specific site.

- **Scatter chart**

Using px.scatter(), which shows the relationship between the launch result (success or failure) and the payload mass in kilograms.

Finally, RangeSlider() was used to provide a slider bar and provide an interactive chart. It can also be filtered by mass ranges and different propellant types.

https://github.com/Michu-code/Class-IBM/blob/main/7.%20spacex_app.py

Predictive Analysis (Classification)

To develop and find the best-performing classification model, we followed a structured process that included several steps:

Data preparation

First, we loaded the data using Numpy and Pandas, performed the necessary transformations such as standardization and preprocessing, and then split the dataset into training and test data using train_test_split().

→

Model development

Next, we selected different machine learning algorithms suitable for our problem. For each, we created a GridSearchCV object along with a dictionary of hyperparameters to tune. We trained the models using the training data and tuned the hyperparameters to optimize their performance.

→

Model evaluation

To evaluate each model, we reviewed the best hyperparameters (best_params_) and accuracy metrics (score and best_score). We also graphed and analyzed the confusion matrix to better understand how they performed.

→

Selecting the best model

Finally, we compared the accuracy scores of all the tested algorithms. The model with the highest score was considered the best performer.

This process allowed us not only to create different models but also to systematically evaluate them and select the one that best suited our data and objectives.

https://github.com/Michu-code/Class-IBM/blob/main/8.%20Machine%20Learning%20Prediction.ipynb

Results

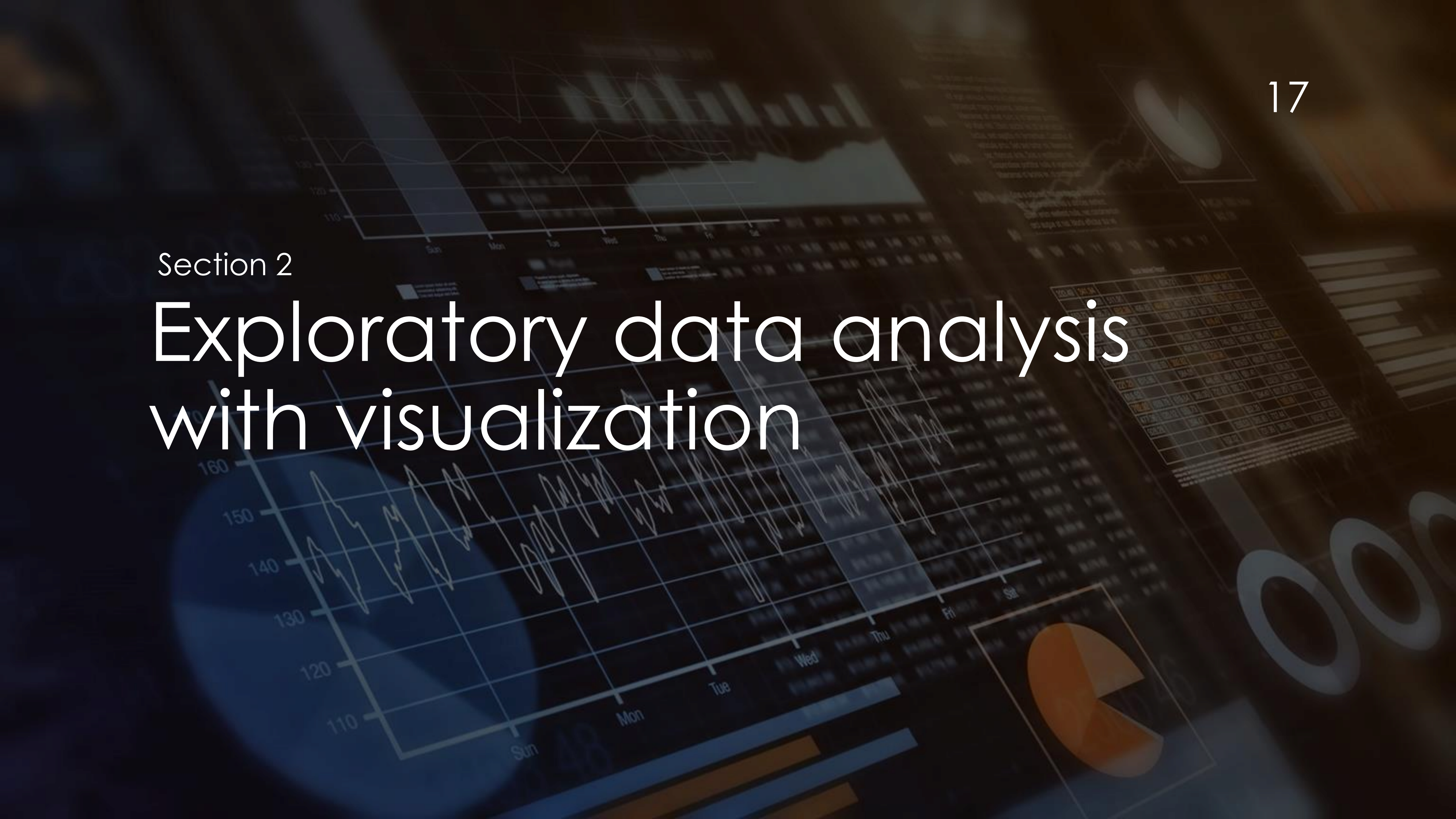
EXPLORATORY DATA ANALYSIS RESULTS

INTERACTIVE ANALYTICS DEMO IN SCREENSHOTS

PREDICTIVE ANALYSIS RESULTS

Section 2

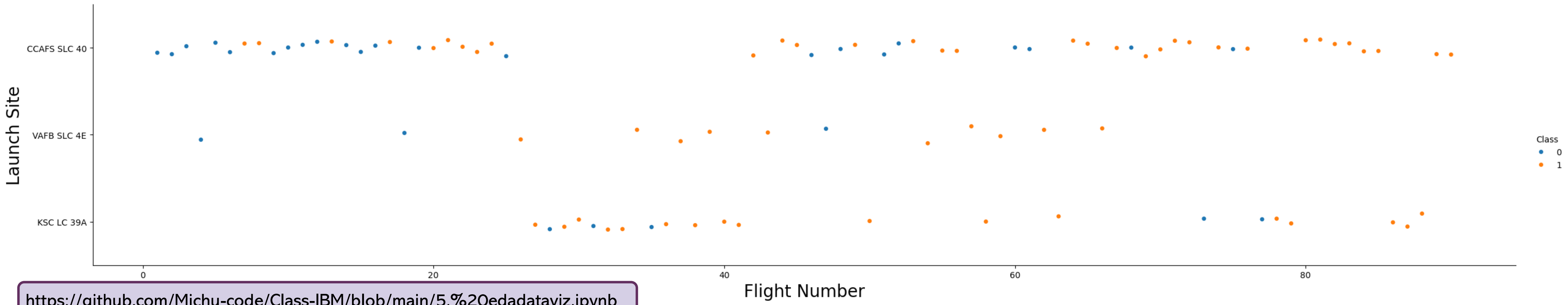
Exploratory data analysis with visualization



Flight Number vs. Launch Site

Scatter Plot:

- Show the relationship between flight number and launch success for different sites
- As the number of flights increases, the success rate at a launch site also increases.
- Most early flights (Flight Number < 30) were launched from CCAFS SLC 40 and generally had lower success rates.
- VAFB SLC 4E shows a similar trend, with early launches being less successful.
- KSC LC 39A did not have early flights; launches from this site were generally more successful from the start.
- Above a flight number of around 30, there are significantly more successful landings (Class = 1).

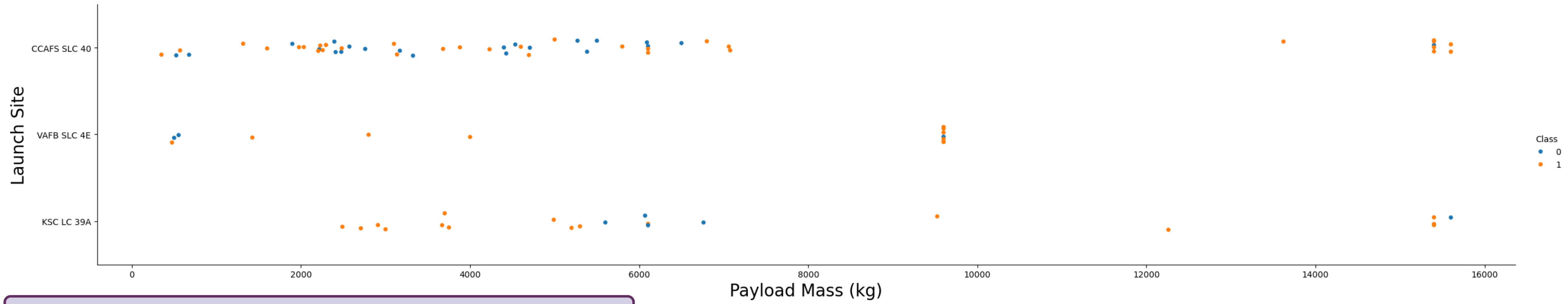


https://github.com/Michu-code/Class-IBM/blob/main/5.%20edadataviz.ipynb

Payload vs. Launch Site

Scatter Plot

- Heavier payloads (above 7000 kg) are rare, and when present, they generally show higher success rates.
- There is no clear, consistent correlation between payload mass and success rate at any specific site.
- CCAFS SLC 40 tends to have lighter payloads compared to other sites, although there are some heavier outliers.
- Overall, higher payloads at CCAFS SLC 40 appear to be associated with better landing success.

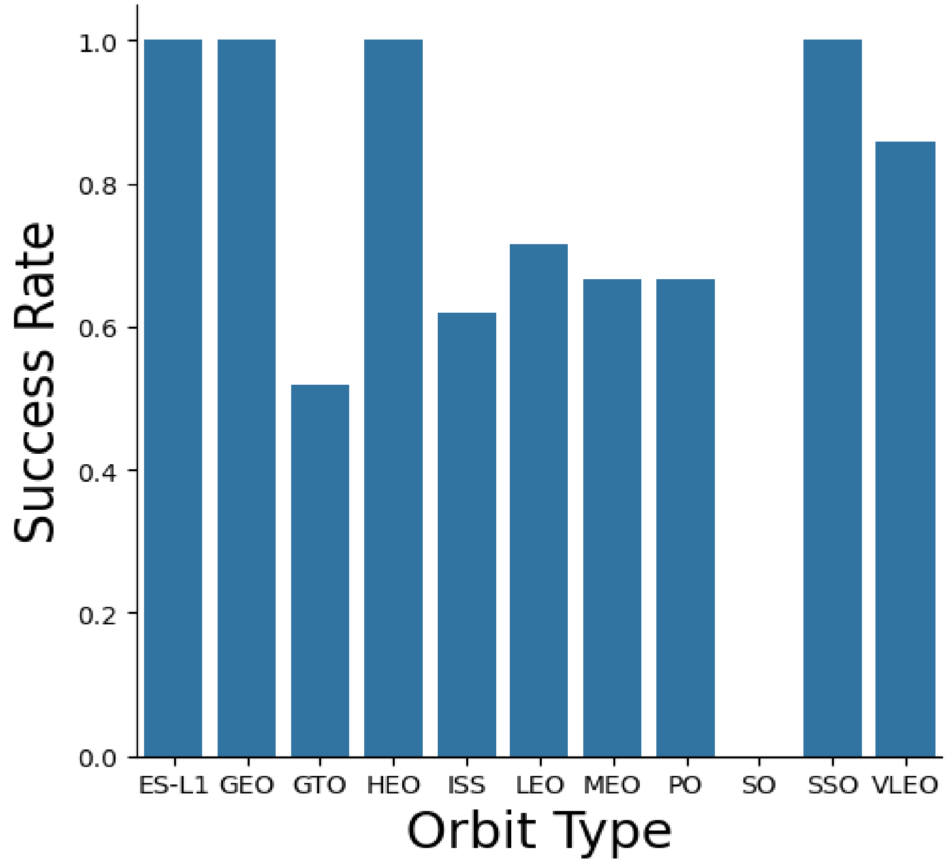


https://github.com/Michu-code/Class-IBM/blob/main/5.%20edadataviz.ipynb

Success Rate vs. Orbit Type

Bar Chart

- Compares the success rates across different orbit types.
- Certain orbits like ES-L1, GEO, HEO, and SSO achieved a 100% success rate.
- VLEO also showed a strong success rate.
- The SO (Heliocentric Orbit) had the lowest success rate, with 0% success.
- This suggests that mission target orbit plays a critical role in determining landing success.

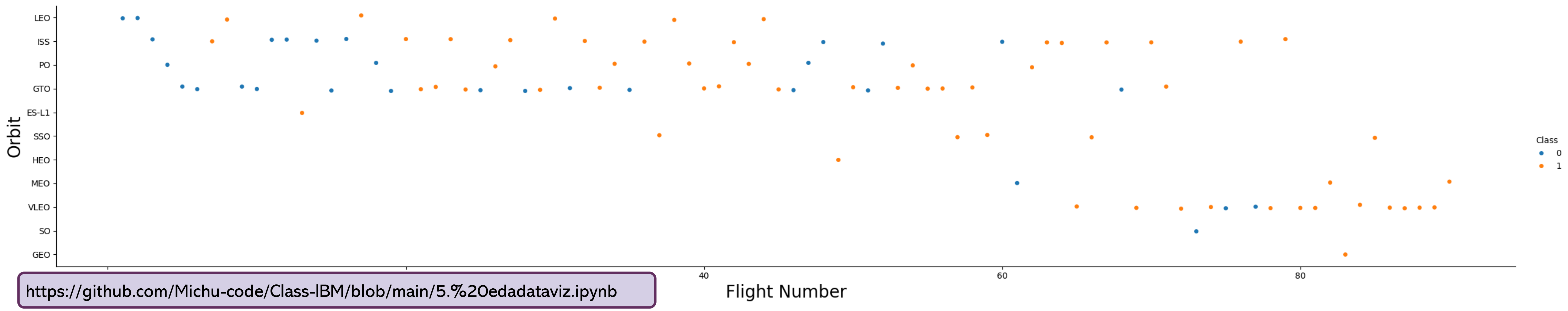


https://github.com/Michu-code/Class-IBM/blob/main/5.%20edadataviz.ipynb

Flight Number vs. Orbit Type

Scatter Plot

- Explores how launch experience (flight number) affects success across different orbit types.
- Orbits like GEO, HEO, and ES-L1 had 100% success rates, but with only one launch each.
- The SSO orbit shows a more impressive trend: multiple flights with consistent success.
- LEO orbits show a clear pattern where more flights lead to higher success rates.
- For GTO orbits, no strong relationship between flight number and success was observed.

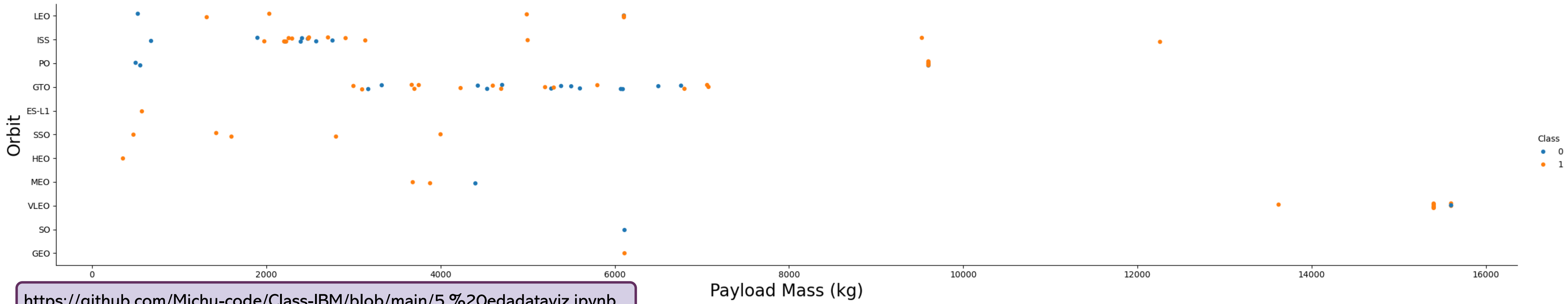


https://github.com/Michu-code/Class-IBM/blob/main/5.%20edadataviz.ipynb

Payload vs. Orbit Type

Scatter Plot

- Purpose: Investigates the link between payload mass and orbit type success.
- Heavy payloads tend to be launched into PO, ISS, and LEO orbits.
- In these orbits, higher payloads are associated with greater landing success.
- For GTO orbits, the relationship between payload mass and success rate is unclear.
- VLEO launches also involve heavier payloads, which aligns with mission needs.

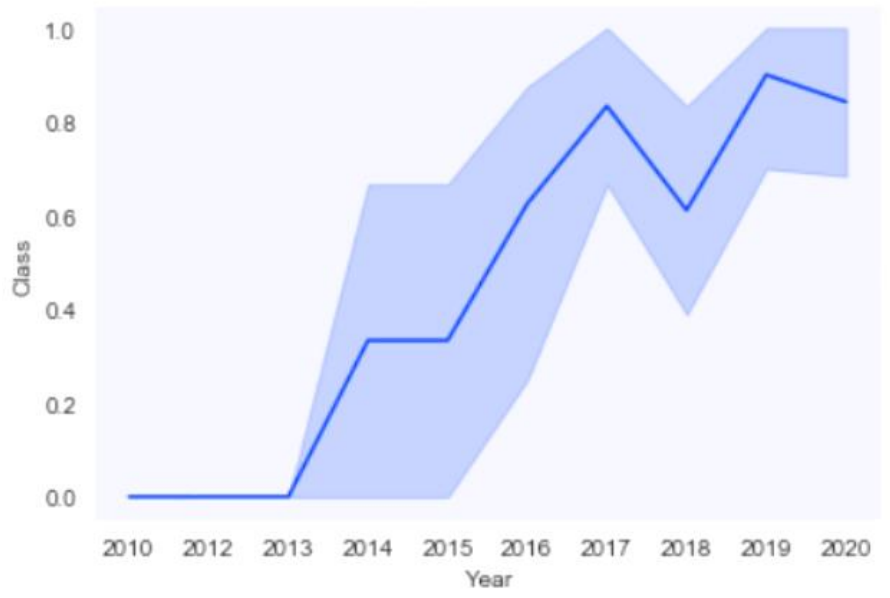


https://github.com/Michu-code/Class-IBM/blob/main/5.%20edadataviz.ipynb

Launch Success Yearly Trend

Line Chart

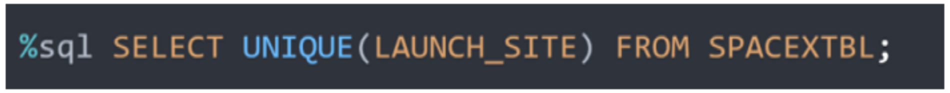
- Tracks the yearly trend of landing success over time.
- From 2010 to 2013, no landings were successful (success rate = 0%).
- After 2013, the success rate steadily increased, despite minor dips in 2018 and 2020.
- After 2016, there was consistently a greater than 50% chance of success.
- The continuous improvement reflects SpaceX's learning curve and technological advancements.



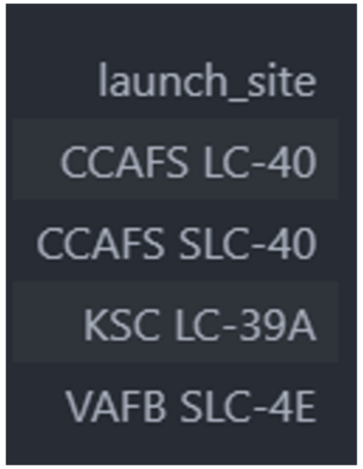
https://github.com/Michu-code/Class-IBM/blob/main/5.%20edadataviz.ipynb

All Launch Site Names

- To find the names of the launch sites, the DISTINCT keyword is used and it returns the values from the LAUNCH_SITE column of the SPACEXTBL table.



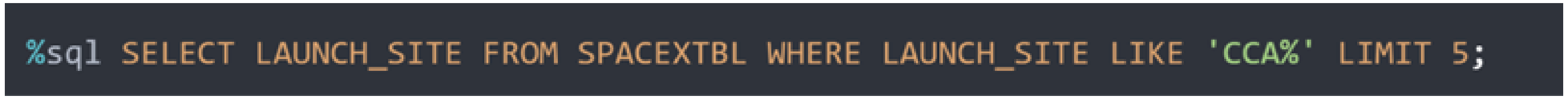
```
%sql SELECT UNIQUE(LAUNCH_SITE) FROM SPACEXTBL;
```



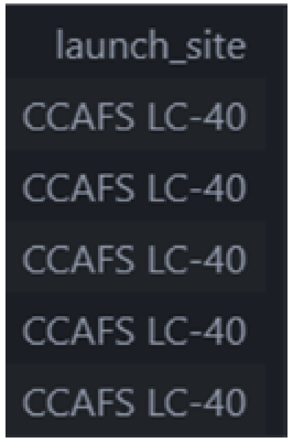
launch_site
CCAFS LC-40
CCAFS SLC-40
KSC LC-39A
VAFB SLC-4E

Launch Site Names Begin with 'CCA'

- We used a SQL query to find 5 records where the launch site name begins with 'CCA'.
- The LIKE 'CCA%' wildcard was used to filter launch sites starting with those letters, and LIMIT 5 was applied to retrieve only the first 5 matching records.



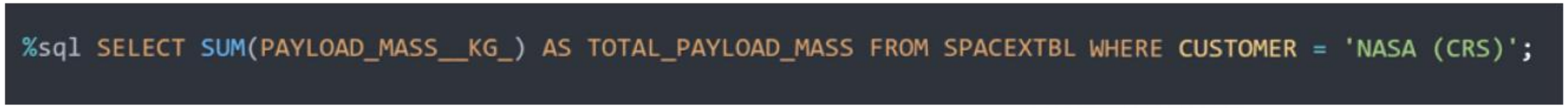
```
%sql SELECT LAUNCH_SITE FROM SPACEXTBL WHERE LAUNCH_SITE LIKE 'CCA%' LIMIT 5;
```



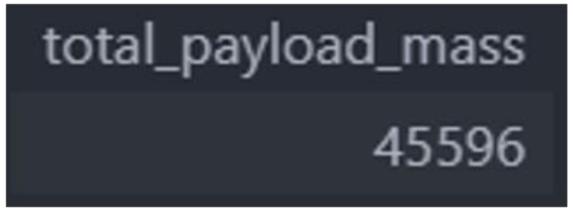
launch_site
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40

Total Payload Mass

- We calculated the total payload carried by boosters from NASA.
- Using the SUM keyword on the payload mass column, filtered for NASA (CRS) missions, we found the total payload to be 45596.



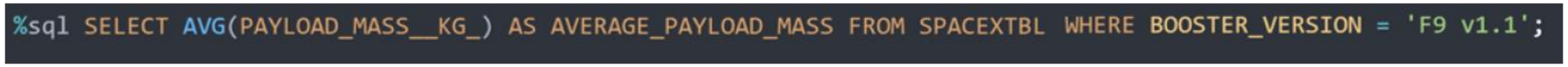
```
%sql SELECT SUM(PAYLOAD_MASS__KG_) AS TOTAL_PAYLOAD_MASS FROM SPACEXTBL WHERE CUSTOMER = 'NASA (CRS)';
```



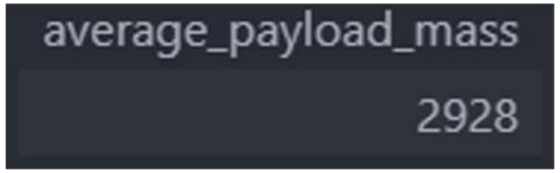
total_payload_mass
45596

Average Payload Mass by F9 v1.1

- We calculated the average payload mass carried by the F9 v1.1 booster version.
- Using the AVG keyword on the payload mass column and filtering by booster version, the average payload mass was found to be approximately 2928.4



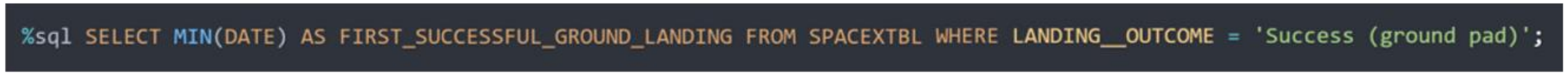
```
%sql SELECT AVG(PAYLOAD_MASS__KG_) AS AVERAGE_PAYLOAD_MASS FROM SPACEXTBL WHERE BOOSTER_VERSION = 'F9 v1.1';
```



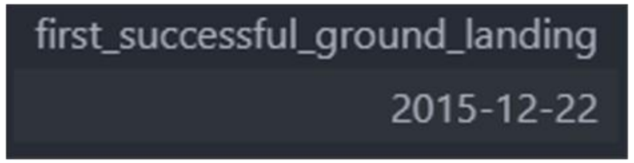
average_payload_mass
2928

First Successful Ground Landing Date

- We determined the date of the first successful landing on a ground pad.
- Using the MIN keyword on the date column, filtered for successful ground pad outcomes, the first successful landing was observed on December 22, 2015.



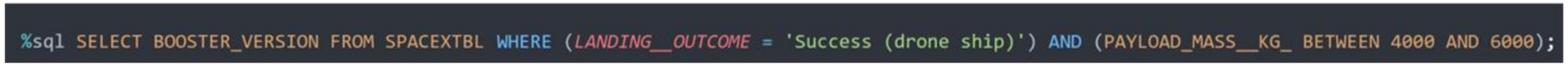
```
%sql SELECT MIN(DATE) AS FIRST_SUCCESSFUL_GROUND_LANDING FROM SPACEXTBL WHERE LANDING__OUTCOME = 'Success (ground pad)';
```



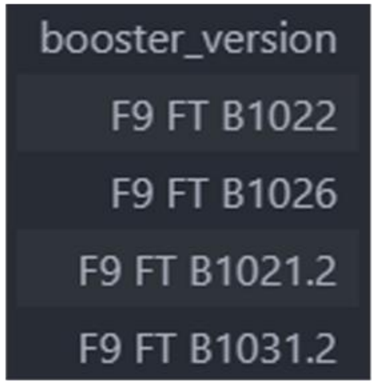
first_successful_ground_landing
2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

- We filtered the data to find boosters that successfully landed on a drone ship with payload mass between 4000 kg and 6000 kg.
- The WHERE clause with an AND condition and BETWEEN was used to select only the boosters meeting both criteria.



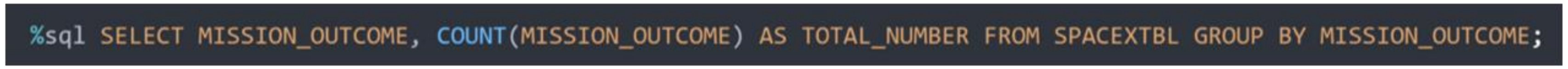
```
%sql SELECT BOOSTER_VERSION FROM SPACEXTBL WHERE (LANDING__OUTCOME = 'Success (drone ship)') AND (PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000);
```



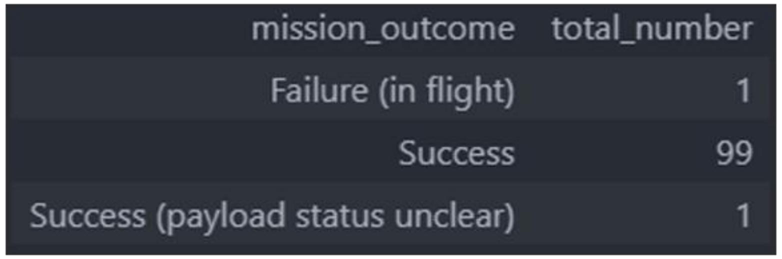
booster_version
F9 FT B1022
F9 FT B1026
F9 FT B1021.2
F9 FT B1031.2

Total Number of Successful and Failure Mission Outcomes

- We calculated the total number of successful and failed mission outcomes.
- The COUNT keyword was used to tally outcomes, and the GROUP BY keyword helped group results by mission success or failure categories.



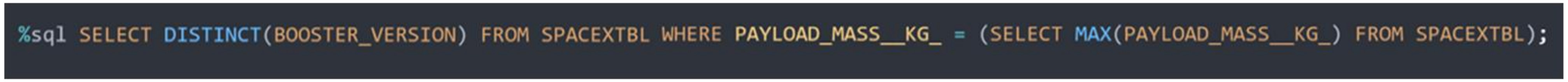
```
%sql SELECT MISSION_OUTCOME, COUNT(MISSION_OUTCOME) AS TOTAL_NUMBER FROM SPACEXTBL GROUP BY MISSION_OUTCOME;
```



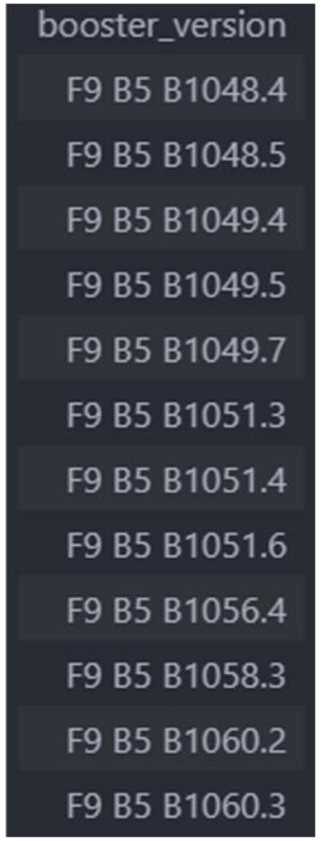
mission_outcome	total_number
Failure (in flight)	1
Success	99
Success (payload status unclear)	1

Boosters Carried Maximum Payload

- We listed the boosters that carried the maximum payload mass.
- A subquery using the MAX() function was applied in the WHERE clause, and the DISTINCT keyword was used to retrieve unique booster versions that achieved this.



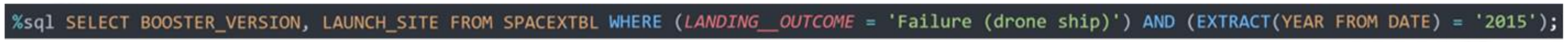
```
%sql SELECT DISTINCT(BOOSTER_VERSION) FROM SPACEXTBL WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTBL);
```



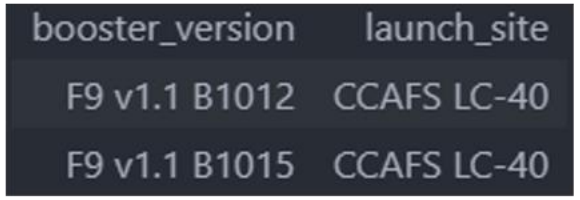
booster_version
F9 B5 B1048.4
F9 B5 B1048.5
F9 B5 B1049.4
F9 B5 B1049.5
F9 B5 B1049.7
F9 B5 B1051.3
F9 B5 B1051.4
F9 B5 B1051.6
F9 B5 B1056.4
F9 B5 B1058.3
F9 B5 B1060.2
F9 B5 B1060.3

2015 Launch Records

- We retrieved failed landing outcomes on drone ships for the year 2015.
- A combination of WHERE, LIKE, AND, and BETWEEN clauses was used to filter for drone ship failures during that specific year.



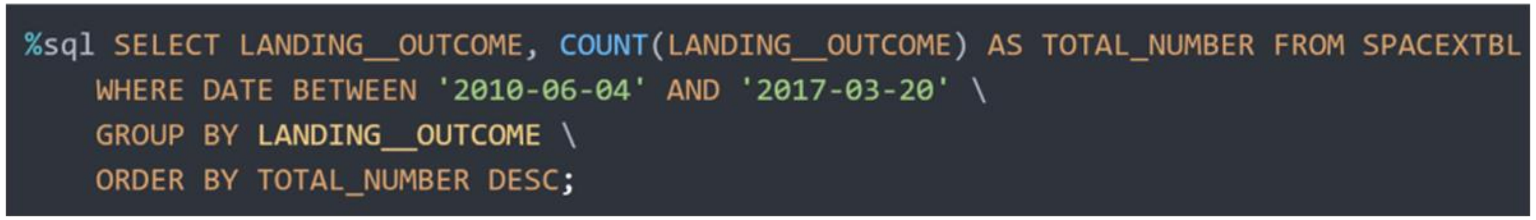
```
%sql SELECT BOOSTER_VERSION, LAUNCH_SITE FROM SPACEXTBL WHERE (LANDING__OUTCOME = 'Failure (drone ship)') AND (EXTRACT(YEAR FROM DATE) = '2015');
```



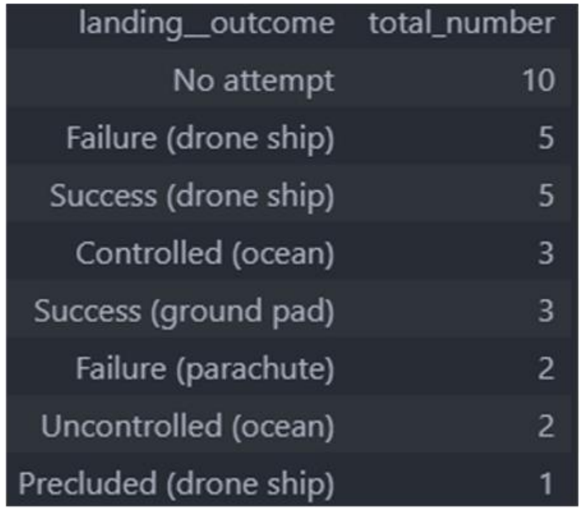
booster_version	launch_site
F9 v1.1 B1012	CCAFS LC-40
F9 v1.1 B1015	CCAFS LC-40

Rank Landing Outcomes Between 2010-06-04 and 2017-03-20

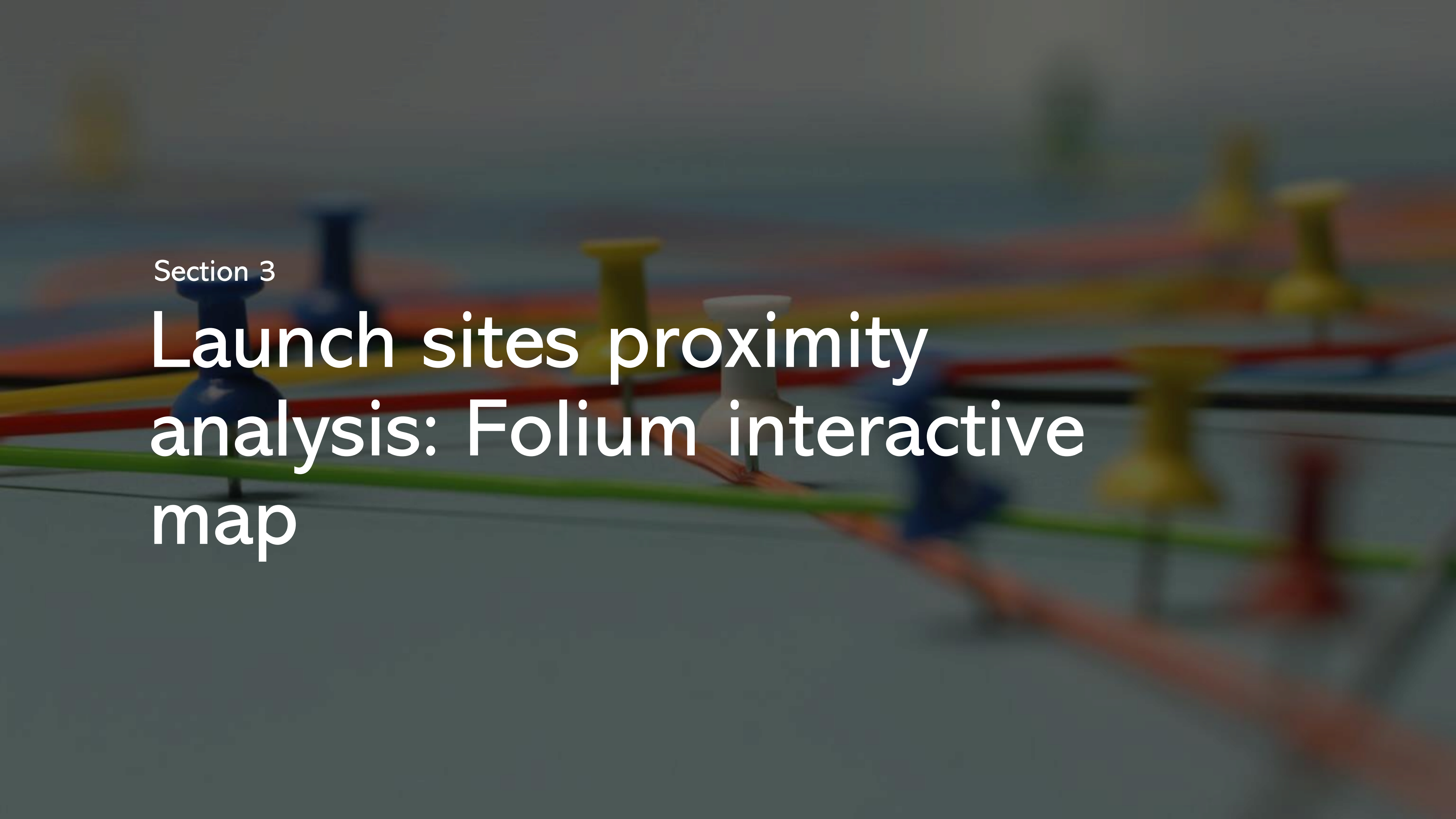
- We ranked the number of different landing outcomes between 2010-06-04 and 2017-03-20.
- Using WHERE with BETWEEN, we filtered the timeframe, then applied GROUP BY to group the outcomes and ORDER BY DESC to rank them from most to least frequent.



```
%sql SELECT LANDING__OUTCOME, COUNT(LANDING__OUTCOME) AS TOTAL_NUMBER FROM SPACEXTBL
    WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20' \
    GROUP BY LANDING__OUTCOME \
    ORDER BY TOTAL_NUMBER DESC;
```



landing__outcome	total_number
No attempt	10
Failure (drone ship)	5
Success (drone ship)	5
Controlled (ocean)	3
Success (ground pad)	3
Failure (parachute)	2
Uncontrolled (ocean)	2
Precluded (drone ship)	1

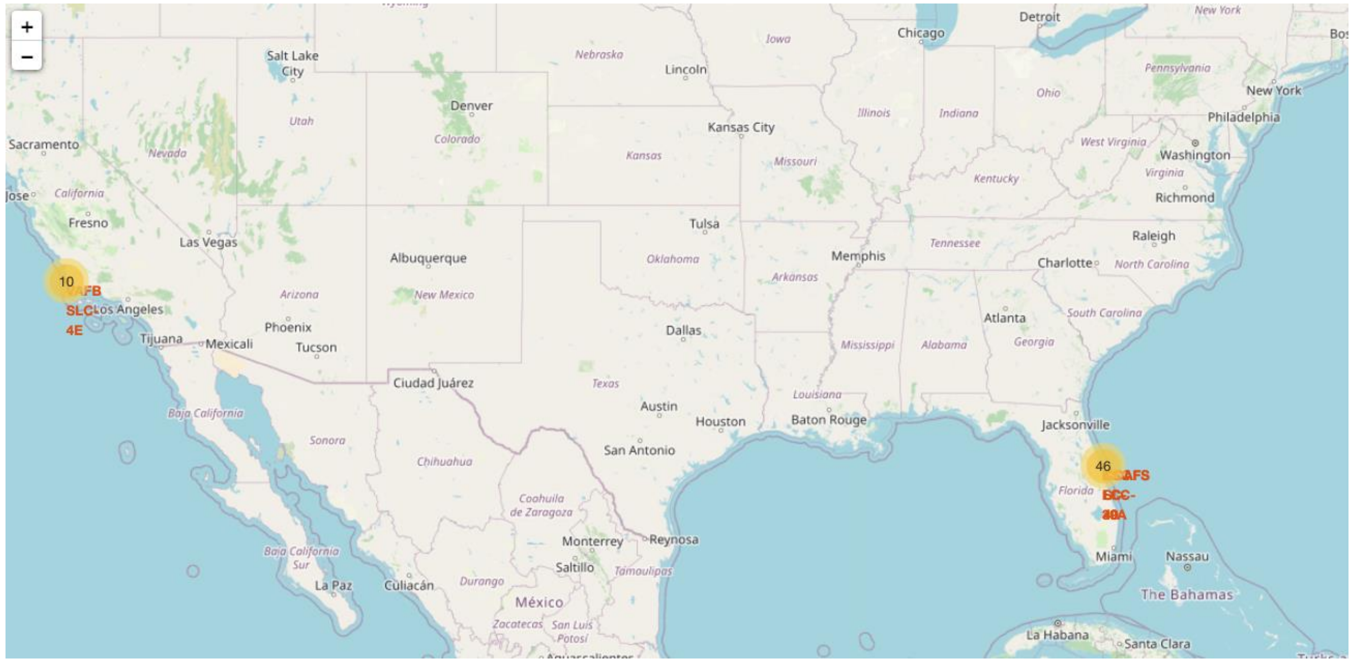


Section 3

Launch sites proximity analysis: Folium interactive map

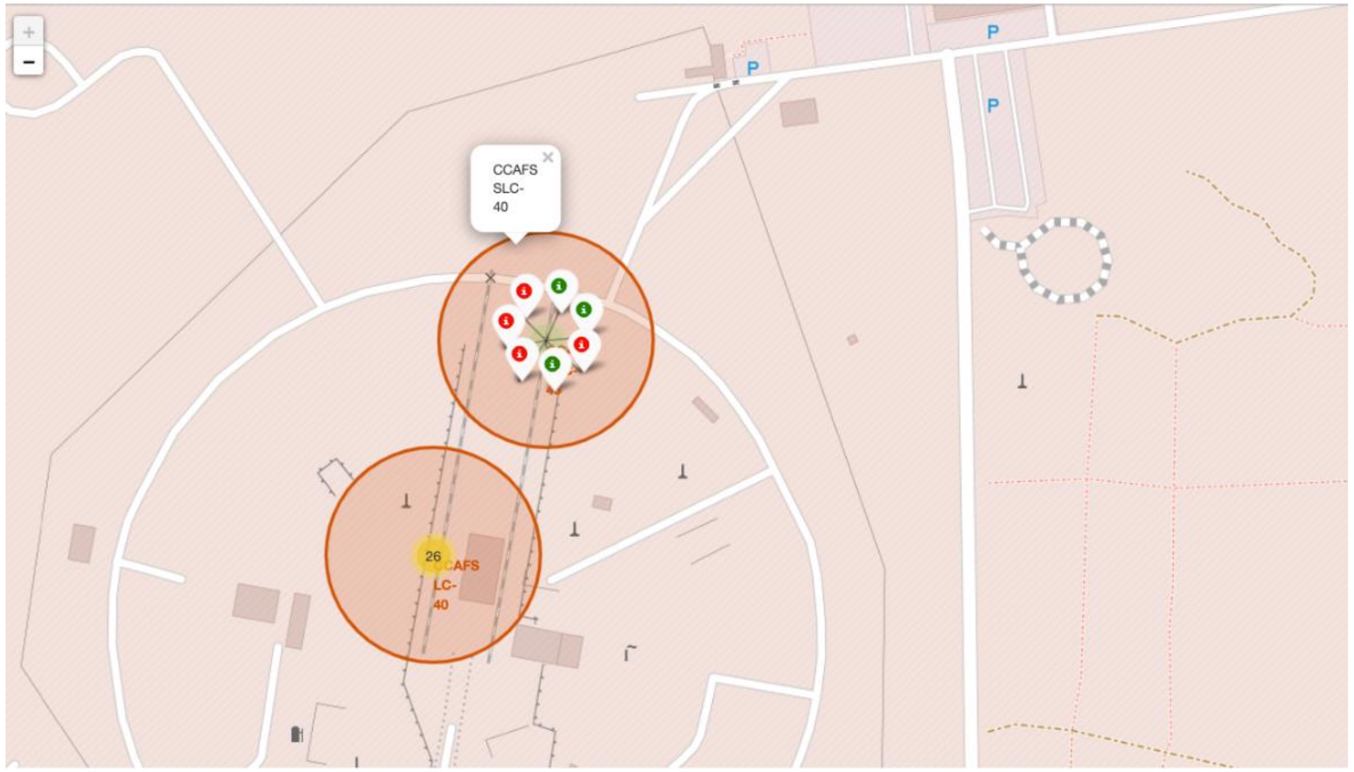
Folium map with launch sites' location markers on a global map

- This is the map of SpaceX launch sites are on coasts of the United States of America. The markers is in Florida and California



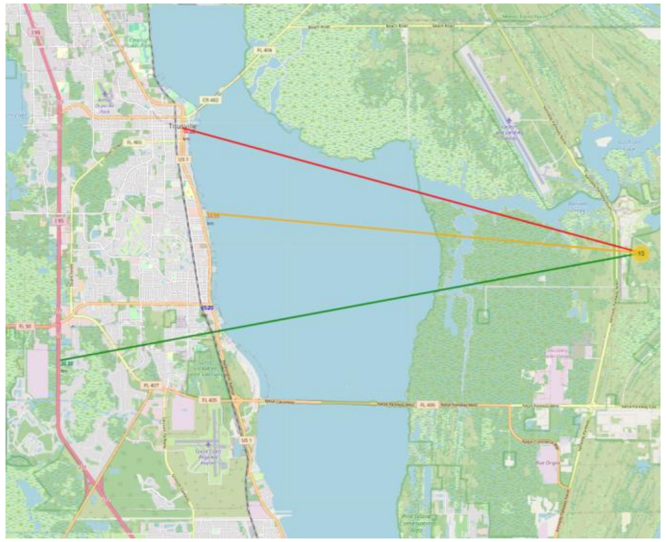
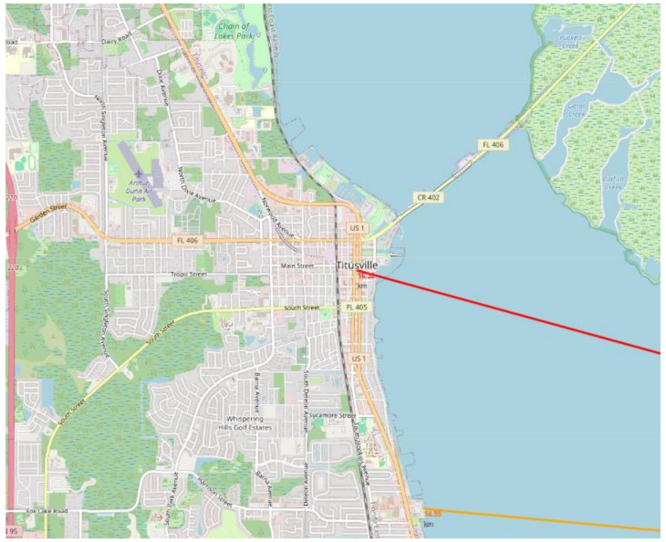
Launch sites with color labels

- Launches, on this almost CCAFS SLC-40, have been marked with green icons for successful launches and red for failed ones.



Proximity of launch point to other places of interest

- Using the Florida launch site, we can understand the location of the launch sites.
- Launch sites near highways: 20.28 km
- Launch sites near cities: 16.32 km from Titusville, Florida
- Launch sites near the coast: 14.99 km

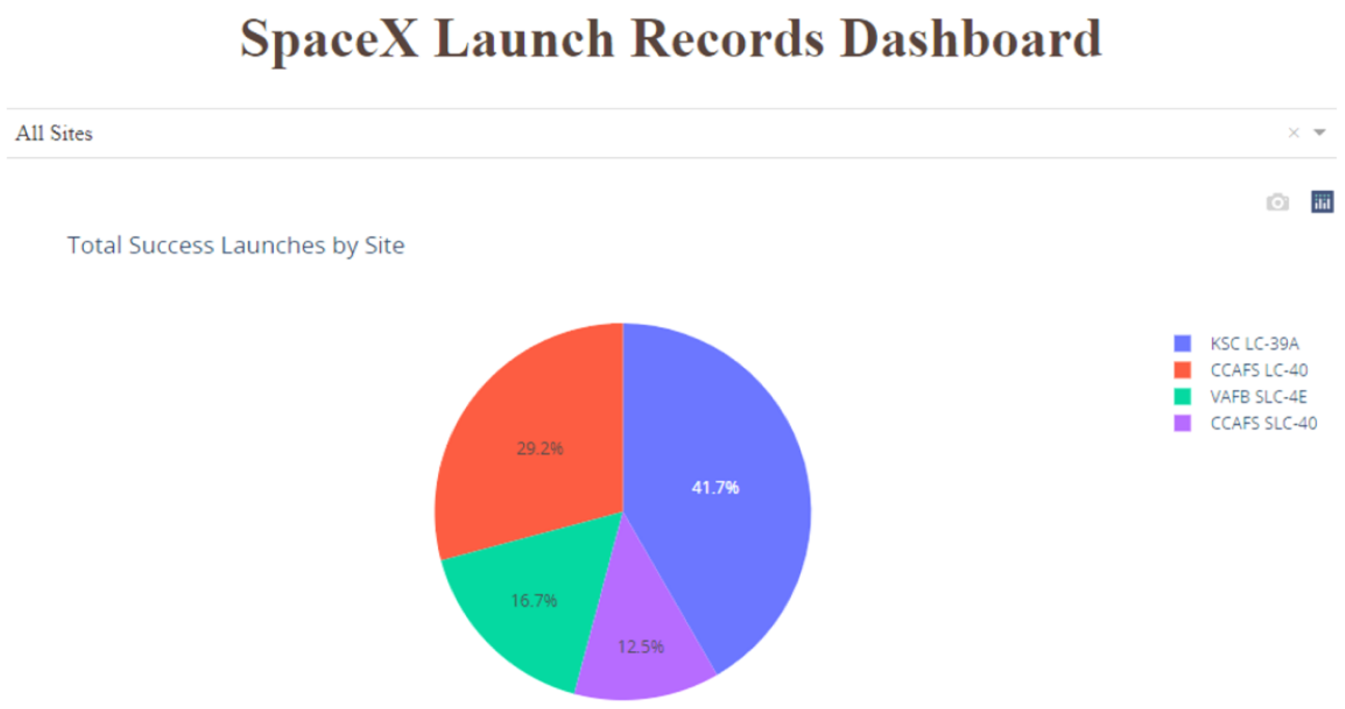



Section 4

Interactive dashboard with Plotly Dash

Launch success count for all sites

- The graph shows the launch sites, where KSC LC-39 had the most successful launches with a total of 41.7% successful launches.

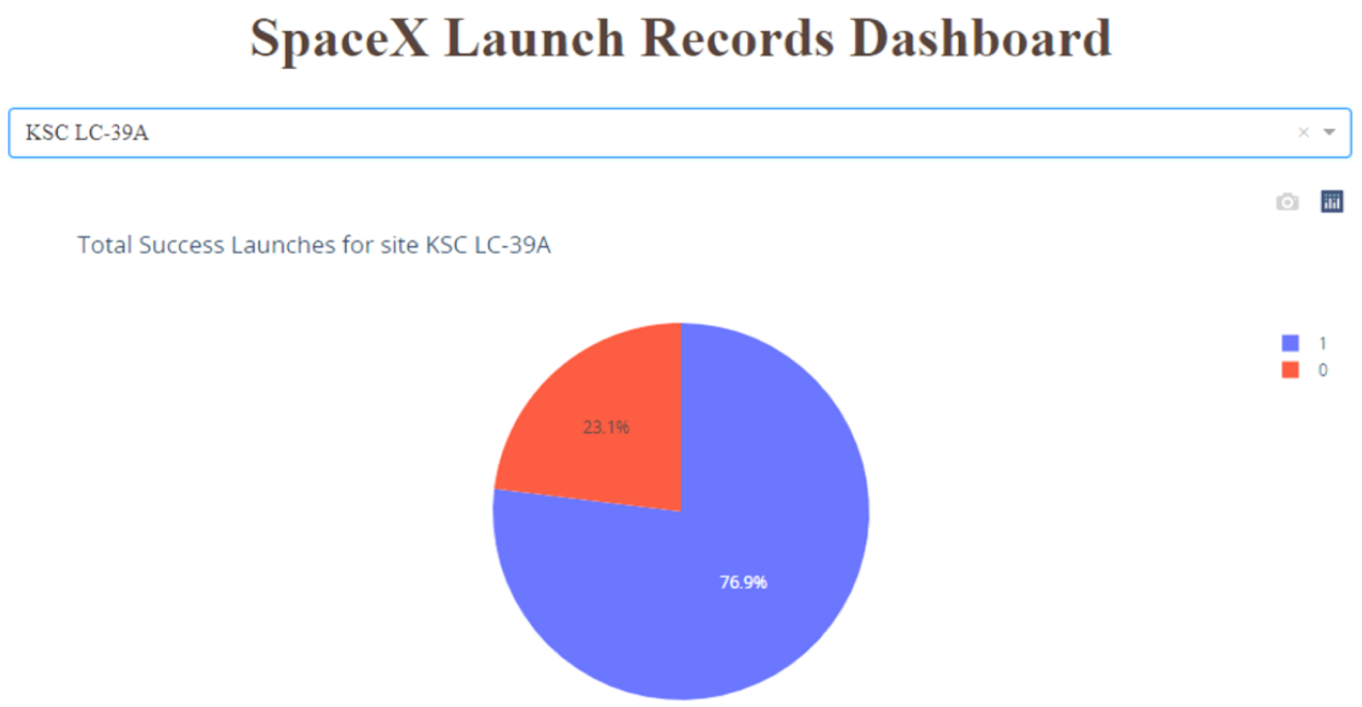


SpaceX Launch Records Dashboard

All Sites

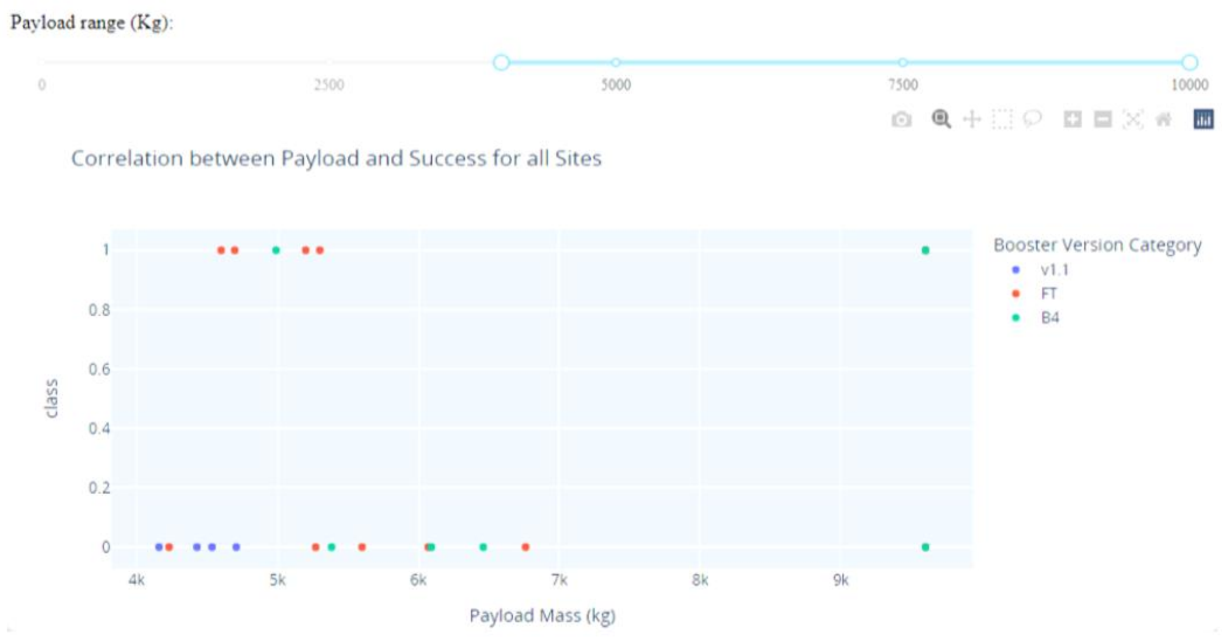
Launch site with highest launch success ratio

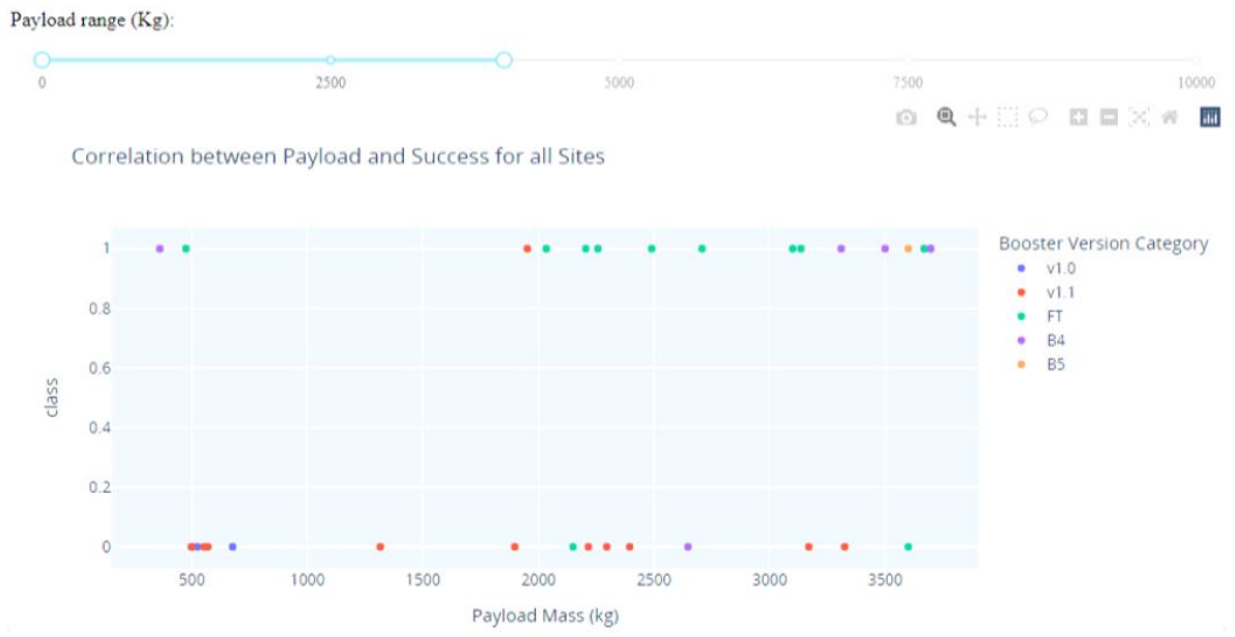
- The KSC LC-39 A launch site also had the highest rate of successful launches, with a success rate of 76.9%. While its launch failure rate is 23.1%.



Payload vs. Launch Outcome scatter plot for all sites

- The success rates for low weights payloads is higher than the heavy weighted payloads
- The booster types v1.0 and B5 have not been launched with massive payloads.



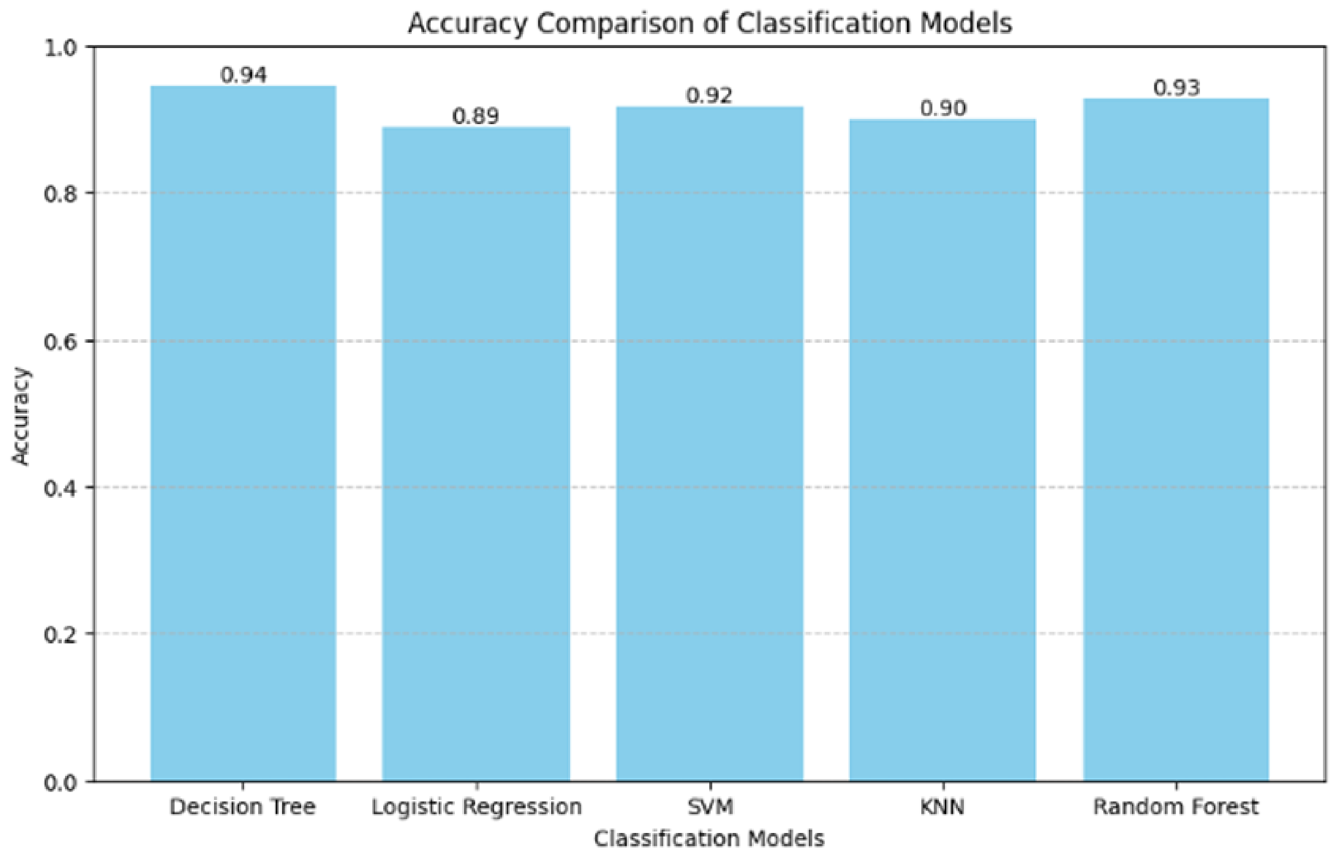


Section 5

Predictive analysis - Classification

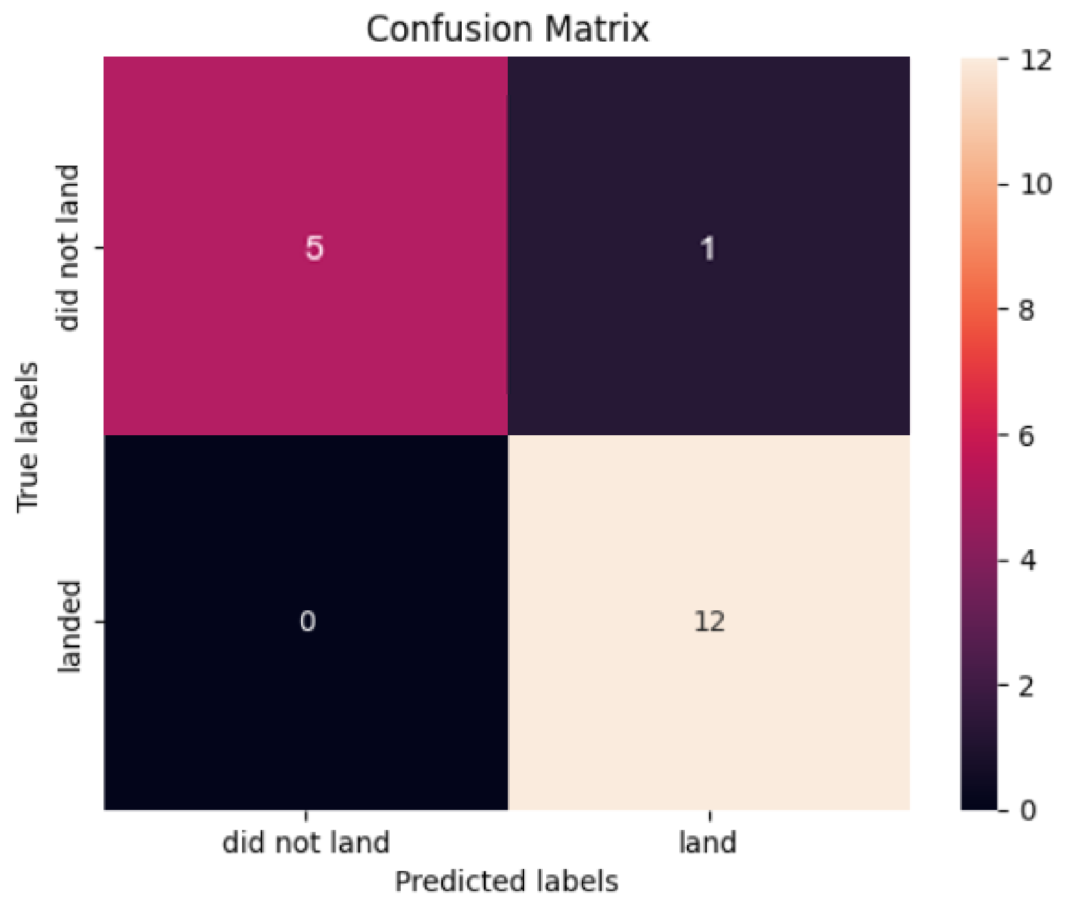
Classification Accuracy

- The decision tree is the model with the highest classification accuracy.
- The decision tree model achieved the highest classification accuracy with an accuracy score of approximately 94.44%.



Confusion Matrix

- We selected the Decision Tree model as the best performing classifier based on its highest accuracy of approximately 94.44%.
- True Positives (TP) = 12
  Correctly predicted successful landings.
- True Negatives (TN) = 5
  Correctly predicted failed landings.
- False Positives (FP) = 0
  No unsuccessful landings incorrectly predicted as successful.
- False Negatives (FN) = 1
  One successful landing incorrectly predicted as a failure.
- Only one misclassification out of 18 total predictions. No false positives, which means the model is very reliable when predicting success.



Conclusions

Conclusions

Most early flights before 2013 were unsuccessful, but after 2013, success rates steadily increased (despite small dips in 2018 and 2020) Since 2016, there has always been a greater than 50% chance of success.

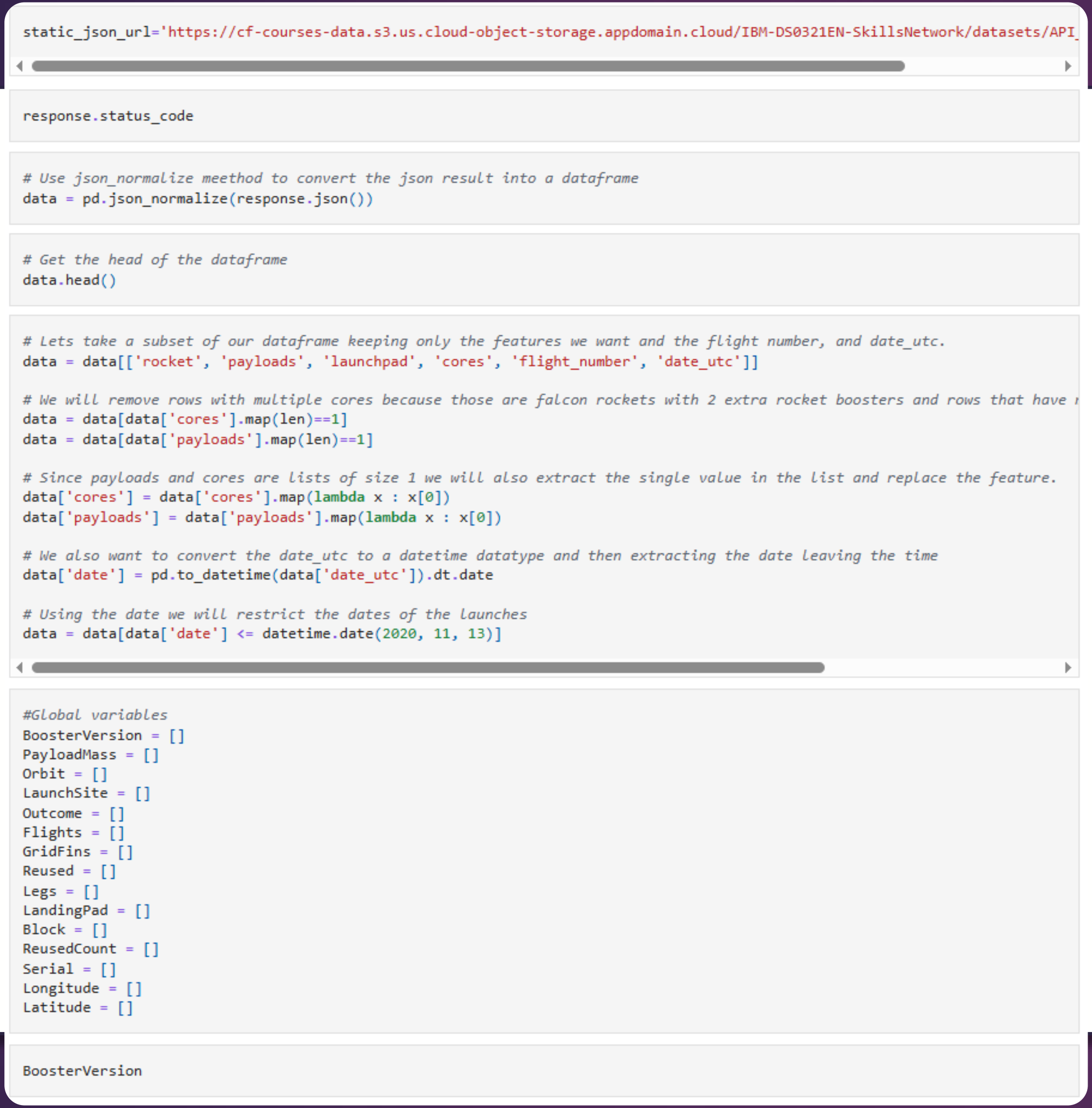
Orbit types such as ES-L1, GEO, HEO, and SSO achieved a 100% success rate. SSO is the most notable, due to its higher number of launches. And heavy payloads are most successful in orbits such as PO, ISS, and LEO. VLEO launches are associated with heavier payloads.

The launch site KSC LC-39A had the most successful launches with a 41.7% of all successful landings, with a success rate of 76.9%

The success rates for massive payloads of more 4000 kg were lower than for lighter payloads.

The Decision Tree is the best machine learning models and achieved the highest classification accuracy at approximately 94.44%. Is making it the best performing model for predicting Falcon 9 first-stage landing success.

Appendix - API



```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_
```

```
response.status_code
```

```
# Use json_normalize meethod to convert the json result into a dataframe
data = pd.json_normalize(response.json())
```

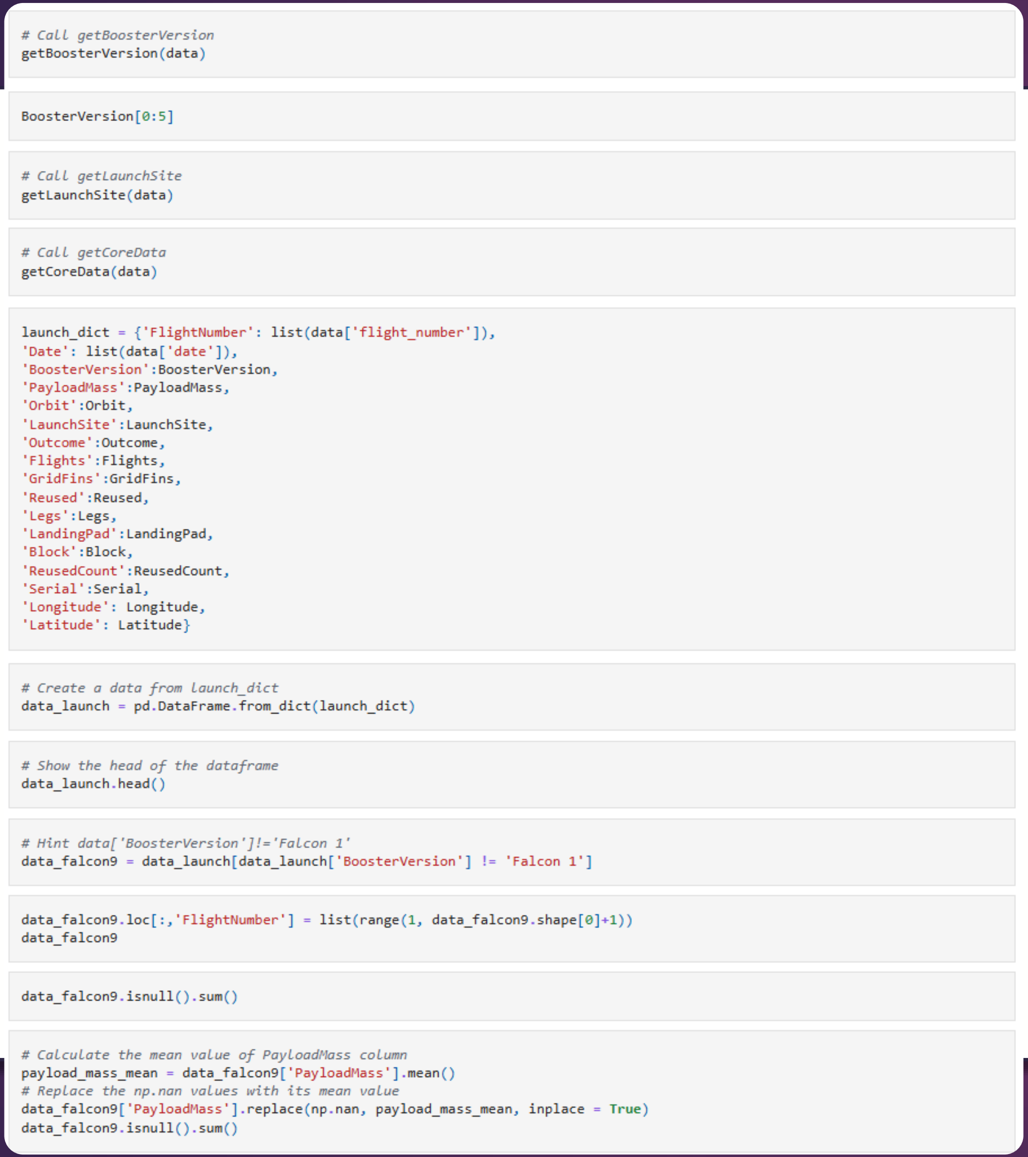
```
# Get the head of the dataframe
data.head()
```

```
# Lets take a subset of our dataframe keeping only the features we want and the flight number, and date_utc.
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]

# We will remove rows with multiple cores because those are falcon rockets with 2 extra rocket boosters and rows that have
data = data[data['cores'].map(len)==1]
data = data[data['payloads'].map(len)==1]

# Since payloads and cores are lists of size 1 we will also extract the single value in the list and replace the feature.
data['cores'] = data['cores'].map(lambda x : x[0])
data['payloads'] = data['payloads'].map(lambda x : x[0])

# We also want to convert the date_utc to a datetime datatype and then extracting the date leaving the time
data['date'] = pd.to_datetime(data['date_utc']).dt.date

# Using the date we will restrict the dates of the launches
data = data[data['date'] <= datetime.date(2020, 11, 13)]
```

```
#Global variables
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
ReusedCount = []
Serial = []
Longitude = []
Latitude = []
```

```
BoosterVersion
```

Appendix - API



```
# Call getBoosterVersion
getBoosterVersion(data)
```

```
BoosterVersion[0:5]
```

```
# Call getLaunchSite
getLaunchSite(data)
```

```
# Call getCoreData
getCoreData(data)
```

```
launch_dict = {'FlightNumber': list(data['flight_number']),
'Date': list(data['date']),
'BoosterVersion':BoosterVersion,
'PayloadMass':PayloadMass,
'Orbit':Orbit,
'LaunchSite':LaunchSite,
'Outcome':Outcome,
'Flights':Flights,
'GridFins':GridFins,
'Reused':Reused,
'Legs':Legs,
'LandingPad':LandingPad,
'Block':Block,
'ReusedCount':ReusedCount,
'Serial':Serial,
'Longitude': Longitude,
'Latitude': Latitude}
```

```
# Create a data from launch_dict
data_launch = pd.DataFrame.from_dict(launch_dict)
```

```
# Show the head of the dataframe
data_launch.head()
```

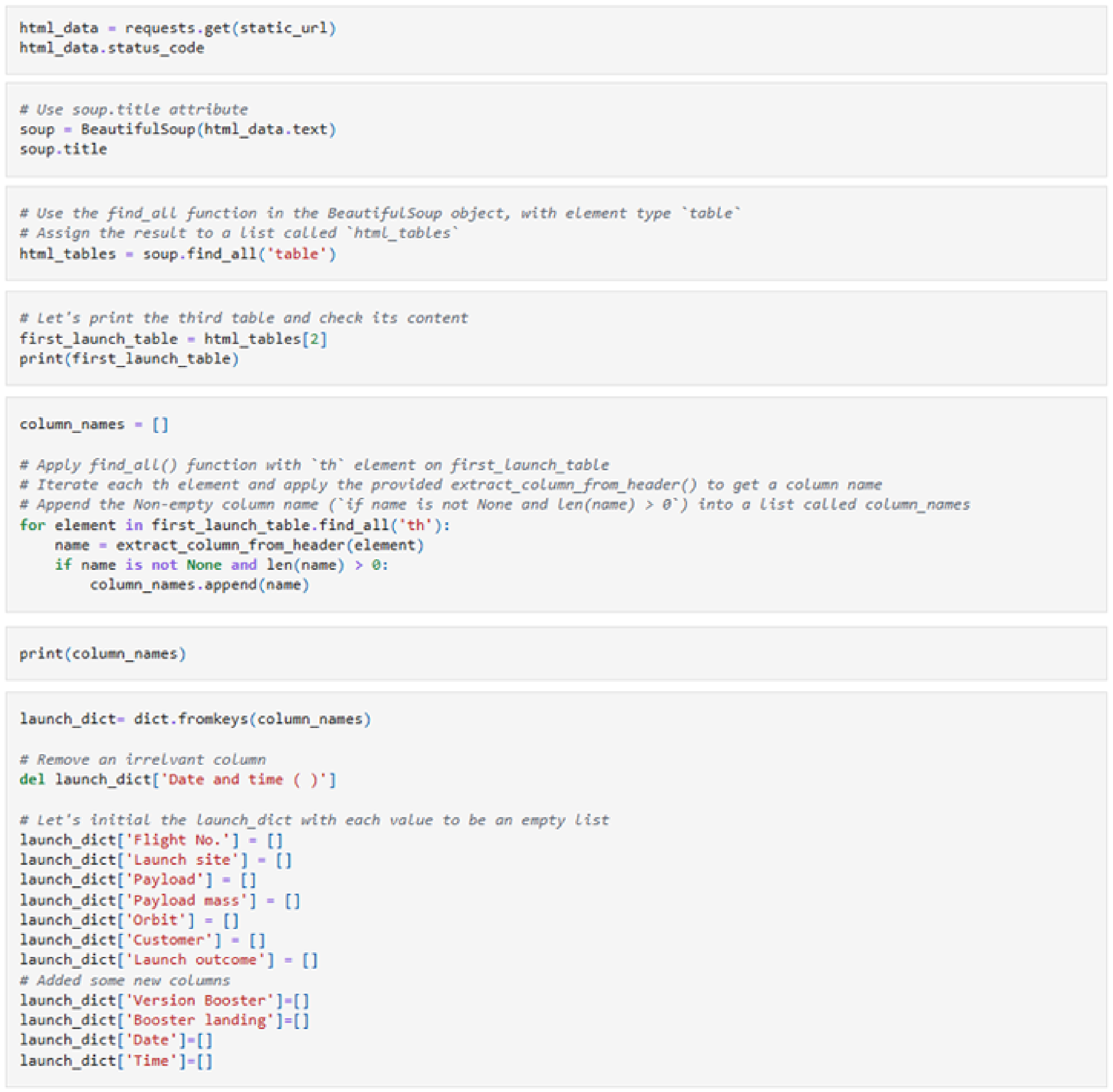
```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = data_launch[data_launch['BoosterVersion'] != 'Falcon 1']
```

```
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9
```

```
data_falcon9.isnull().sum()
```

```
# Calculate the mean value of PayloadMass column
payload_mass_mean = data_falcon9['PayloadMass'].mean()
# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'].replace(np.nan, payload_mass_mean, inplace = True)
data_falcon9.isnull().sum()
```

Appendix - WEB SCRAPING



```
html_data = requests.get(static_url)
html_data.status_code
```

```
# Use soup.title attribute
soup = BeautifulSoup(html_data.text)
soup.title
```

```
# Use the find_all function in the BeautifulSoup object, with element type `table`
# Assign the result to a list called `html_tables`
html_tables = soup.find_all('table')
```

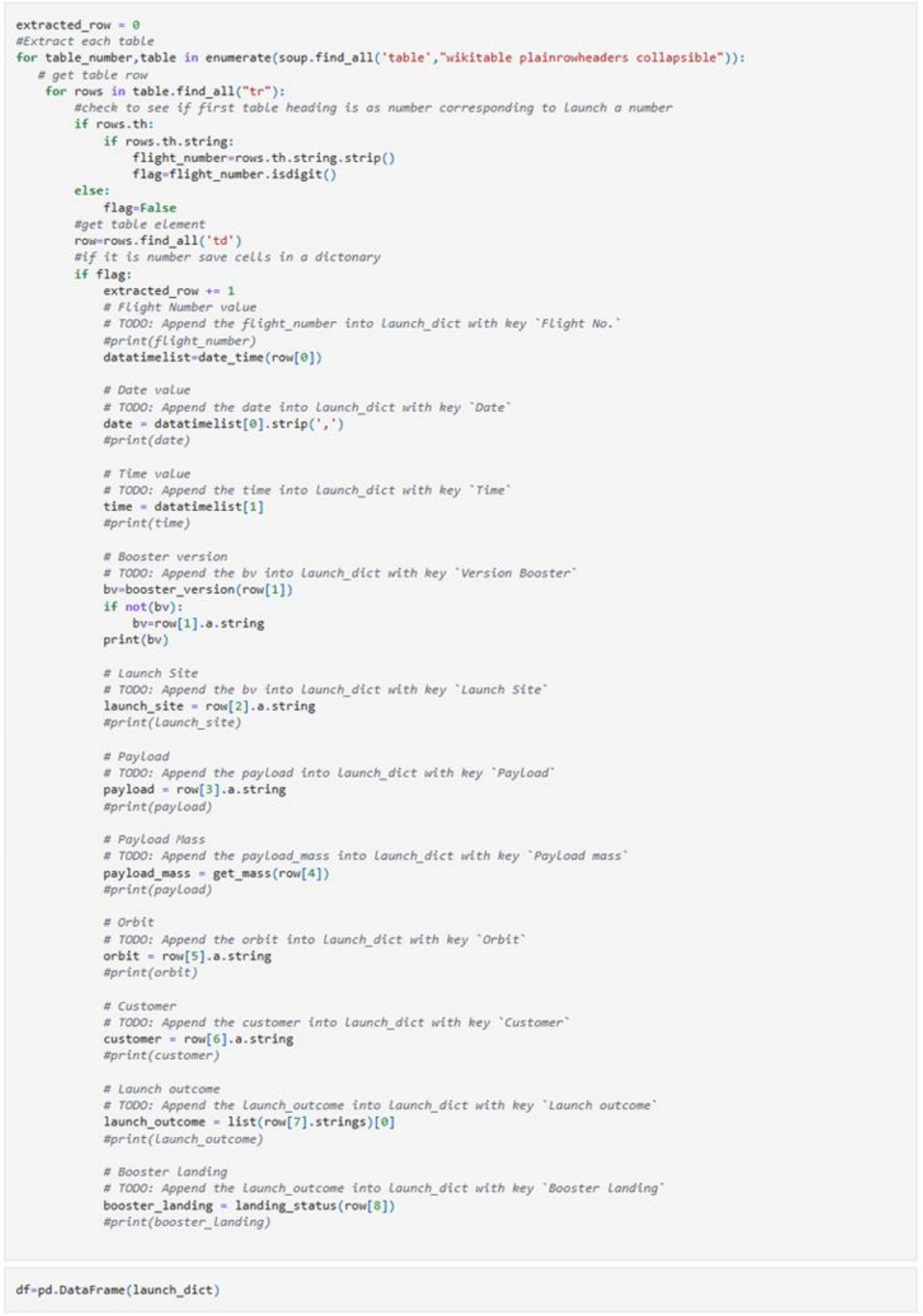
```
# Let's print the third table and check its content
first_launch_table = html_tables[2]
print(first_launch_table)
```

```
column_names = []

# Apply find_all() function with `th` element on first_launch_table
# Iterate each th element and apply the provided extract_column_from_header() to get a column name
# Append the Non-empty column name (`if name is not None and len(name) > 0`) into a list called column_names
for element in first_launch_table.find_all('th'):
    name = extract_column_from_header(element)
    if name is not None and len(name) > 0:
        column_names.append(name)
```

```
print(column_names)
```

```
launch_dict= dict.fromkeys(column_names)

# Remove an irrelvant column
del launch_dict['Date and time ( )']

# Let's initial the launch_dict with each value to be an empty list
launch_dict['Flight No.'] = []
launch_dict['Launch site'] = []
launch_dict['Payload'] = []
launch_dict['Payload mass'] = []
launch_dict['Orbit'] = []
launch_dict['Customer'] = []
launch_dict['Launch outcome'] = []
# Added some new columns
launch_dict['Version Booster']=[]
launch_dict['Booster landing']=[]
launch_dict['Date']=[]
launch_dict['Time']=[]
```

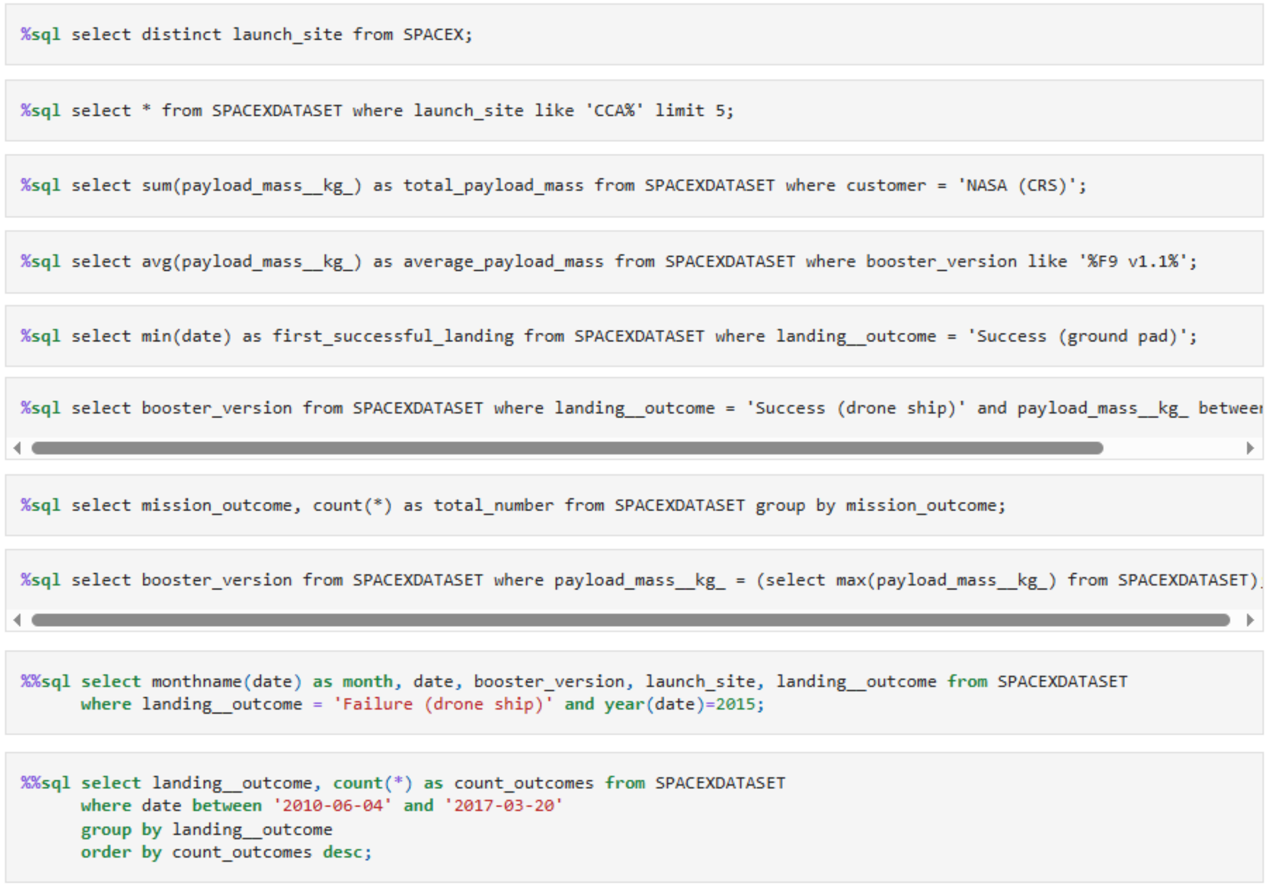
Appendix - WEB SCRAPING



```
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders collapsible")):
   # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number corresponding to launch a number
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
        else:
            flag=False
        #get table element
        row=rows.find_all('td')
        #if it is number save cells in a dictonary
        if flag:
            extracted_row += 1
            # Flight Number value
            # TODO: Append the flight_number into launch_dict with key `Flight No.`
            #print(flight_number)
            datatimelist=date_time(row[0])

            # Date value
            # TODO: Append the date into launch_dict with key `Date`
            date = datatimelist[0].strip(',')
            #print(date)

            # Time value
            # TODO: Append the time into launch_dict with key `Time`
            time = datatimelist[1]
            #print(time)

            # Booster version
            # TODO: Append the bv into launch_dict with key `Version Booster`
            bv=booster_version(row[1])
            if not(bv):
                bv=row[1].a.string
            print(bv)

            # Launch Site
            # TODO: Append the bv into launch_dict with key `Launch Site`
            launch_site = row[2].a.string
            #print(launch_site)

            # Payload
            # TODO: Append the payload into launch_dict with key `Payload`
            payload = row[3].a.string
            #print(payload)

            # Payload Mass
            # TODO: Append the payload_mass into launch_dict with key `Payload mass`
            payload_mass = get_mass(row[4])
            #print(payload)

            # Orbit
            # TODO: Append the orbit into launch_dict with key `Orbit`
            orbit = row[5].a.string
            #print(orbit)

            # Customer
            # TODO: Append the customer into launch_dict with key `Customer`
            customer = row[6].a.string
            #print(customer)

            # Launch outcome
            # TODO: Append the launch_outcome into launch_dict with key `Launch outcome`
            launch_outcome = list(row[7].strings)[0]
            #print(launch_outcome)

            # Booster landing
            # TODO: Append the launch_outcome into launch_dict with key `Booster landing`
            booster_landing = landing_status(row[8])
            #print(booster_landing)
```

```
df=pd.DataFrame(launch_dict)
```

Appendix - SQL



```
%sql select distinct launch_site from SPACEX;
```

```
%sql select * from SPACEXDATASET where launch_site like 'CCA%' limit 5;
```

```
%sql select sum(payload_mass__kg_) as total_payload_mass from SPACEXDATASET where customer = 'NASA (CRS)';
```

```
%sql select avg(payload_mass__kg_) as average_payload_mass from SPACEXDATASET where booster_version like '%F9 v1.1%';
```

```
%sql select min(date) as first_successful_landing from SPACEXDATASET where landing__outcome = 'Success (ground pad)';
```

```
%sql select booster_version from SPACEXDATASET where landing__outcome = 'Success (drone ship)' and payload_mass__kg_ betwee
```

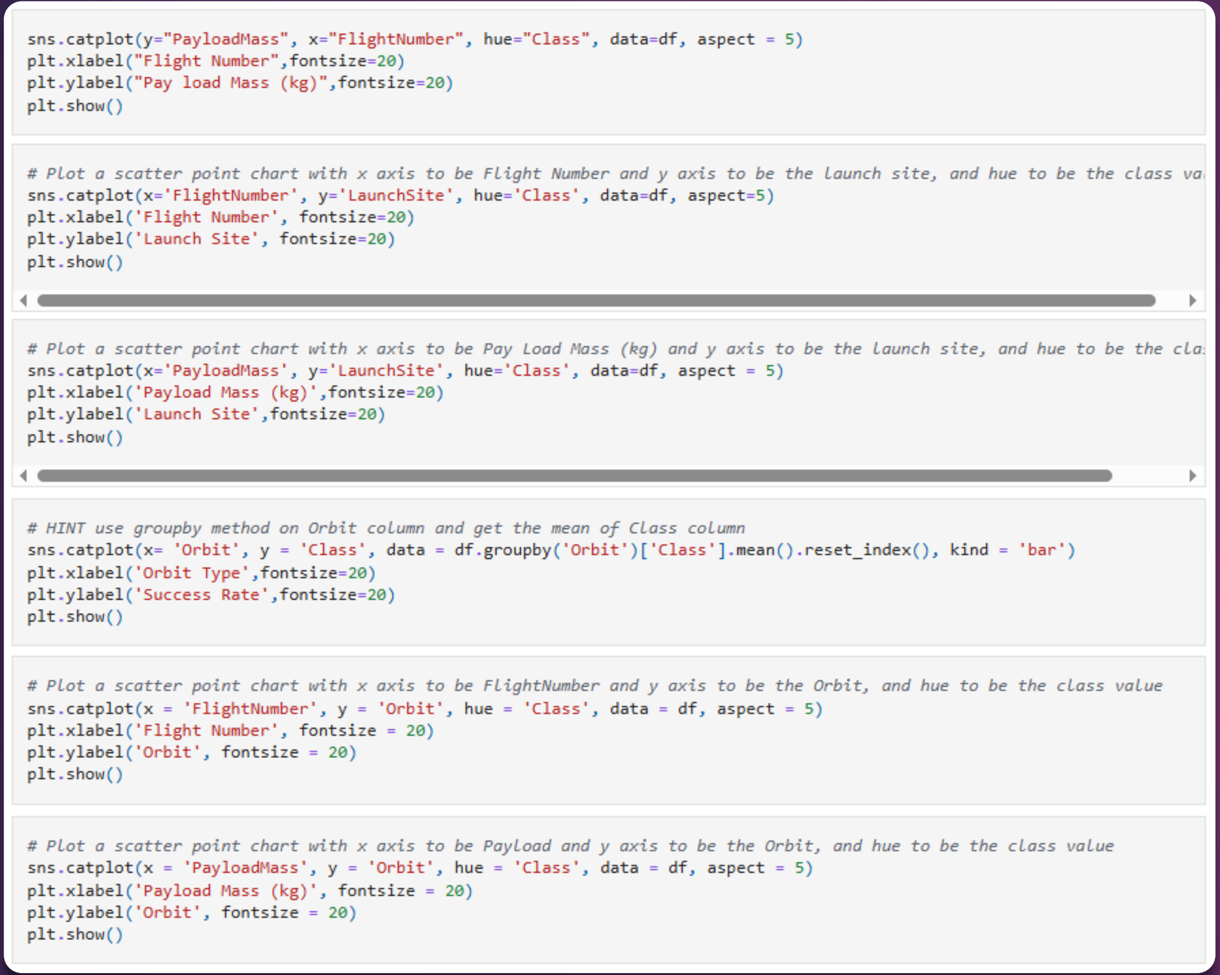
```
%sql select mission_outcome, count(*) as total_number from SPACEXDATASET group by mission_outcome;
```

```
%sql select booster_version from SPACEXDATASET where payload_mass__kg_ = (select max(payload_mass__kg_) from SPACEXDATASET)
```

```
%%sql select monthname(date) as month, date, booster_version, launch_site, landing__outcome from SPACEXDATASET
      where landing__outcome = 'Failure (drone ship)' and year(date)=2015;
```

```
%%sql select landing__outcome, count(*) as count_outcomes from SPACEXDATASET
      where date between '2010-06-04' and '2017-03-20'
      group by landing__outcome
      order by count_outcomes desc;
```
]

Appendix – EDA Charts



```
sns.catplot(y="PayloadMass", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Pay load Mass (kg)",fontsize=20)
plt.show()
```

```
# Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to be the class va
sns.catplot(x='FlightNumber', y='LaunchSite', hue='Class', data=df, aspect=5)
plt.xlabel('Flight Number', fontsize=20)
plt.ylabel('Launch Site', fontsize=20)
plt.show()
```

```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch site, and hue to be the cla
sns.catplot(x='PayloadMass', y='LaunchSite', hue='Class', data=df, aspect = 5)
plt.xlabel('Payload Mass (kg)',fontsize=20)
plt.ylabel('Launch Site',fontsize=20)
plt.show()
```

```
# HINT use groupby method on Orbit column and get the mean of Class column
sns.catplot(x= 'Orbit', y = 'Class', data = df.groupby('Orbit')['Class'].mean().reset_index(), kind = 'bar')
plt.xlabel('Orbit Type',fontsize=20)
plt.ylabel('Success Rate',fontsize=20)
plt.show()
```

```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class value
sns.catplot(x = 'FlightNumber', y = 'Orbit', hue = 'Class', data = df, aspect = 5)
plt.xlabel('Flight Number', fontsize = 20)
plt.ylabel('Orbit', fontsize = 20)
plt.show()
```

```
# Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, and hue to be the class value
sns.catplot(x = 'PayloadMass', y = 'Orbit', hue = 'Class', data = df, aspect = 5)
plt.xlabel('Payload Mass (kg)', fontsize = 20)
plt.ylabel('Orbit', fontsize = 20)
plt.show()
```