IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Name: Rakshit Gupta

Github Profile:
https://www.github.com/RaKsHiTG10

Date : 11/16/2025

Outline

- Executive Summary
- Introduction
- Methodology
- Results
- Conclusion
- Appendix

Executive Summary

- Brief overview of objectives
- Summary of methods used (EDA, SQL, Folium, Dash, ML)
- Summary of key results (e.g., best model accuracy, insights from map)
- Why this project matters

Introduction

- Project background
- Dataset context
- Problemgoals statement / key questions
- Scope and

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
- Website scraped: Wikipedia - *List of Falcon 9 and Falcon Heavy launches*
- URL: (shown on slide, no hyperlink auto-detection) https://en.wikipedia.org/wiki/List_of_Falcon_9_and_Falcon_Heavy_launches

Contains historical launch data in HTML tables
Wikipedia Page → HTML Response → BeautifulSoup Parsing → Table Extraction → Row Parsing → Pandas DataFrame → CSV / Data Storage

- Perform data wrangling
  - •Renamed columns
  - •Removed HTML tags such as <a>,
, <sup>
  - •Converted data types (dates, boolean success/failure)
  - •Extracted binary class label for landing success
  - •Dropped unused columns
  - •Cleaned text spacing, footnotes, special chars
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - Perform predictive analysis by building, tuning, and evaluating classification models to identify the best-performing algorithm.

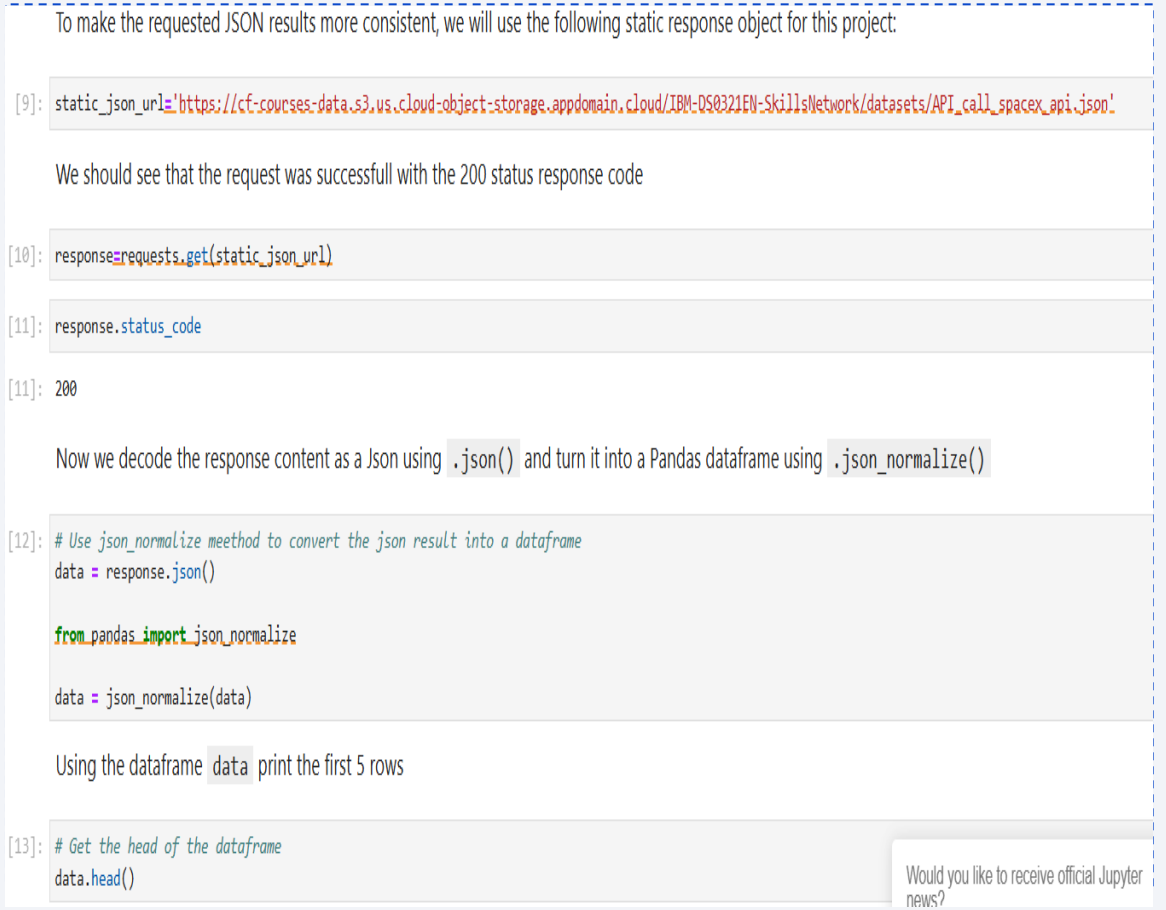
Data Collection

- Retrieved launch records using **SpaceX REST API**
- Scraped historical launch data from **Wikipedia using BeautifulSoup**
- Parsed HTML tables into **Pandas DataFrames**
- Extracted additional metadata from **CSV and JSON files**
- Automated data retrieval using **Python scripts and Jupyter notebooks**
- Consolidated multiple data sources into a **single structured dataset**
- **Flowchart Outline** API/Wiki → Request/Download → Parse HTML/JSON → Extract relevant fields → Clean & standardize format → Save to DataFrame

Data Collection – SpaceX API

- Present your data collection with SpaceX REST calls using key phrases and flowcharts

- GitHub URL of the completed SpaceX API calls notebook https://github.com/RaKsHiTG10/WebScrapping as an external reference and peer-review purpose



To make the requested JSON results more consistent, we will use the following static response object for this project:

```
[9]: static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_call_spacex_api.json'
```

We should see that the request was successfull with the 200 status response code

```
[10]: response=requests.get(static_json_url)
```

```
[11]: response.status_code
```

```
[11]: 200
```

Now we decode the response content as a Json using `.json()` and turn it into a Pandas dataframe using `.json_normalize()`

```
[12]: # Use json_normalize meethod to convert the json result into a dataframe
data = response.json()

from pandas import json_normalize

data = json_normalize(data)
```

Using the dataframe `data` print the first 5 rows

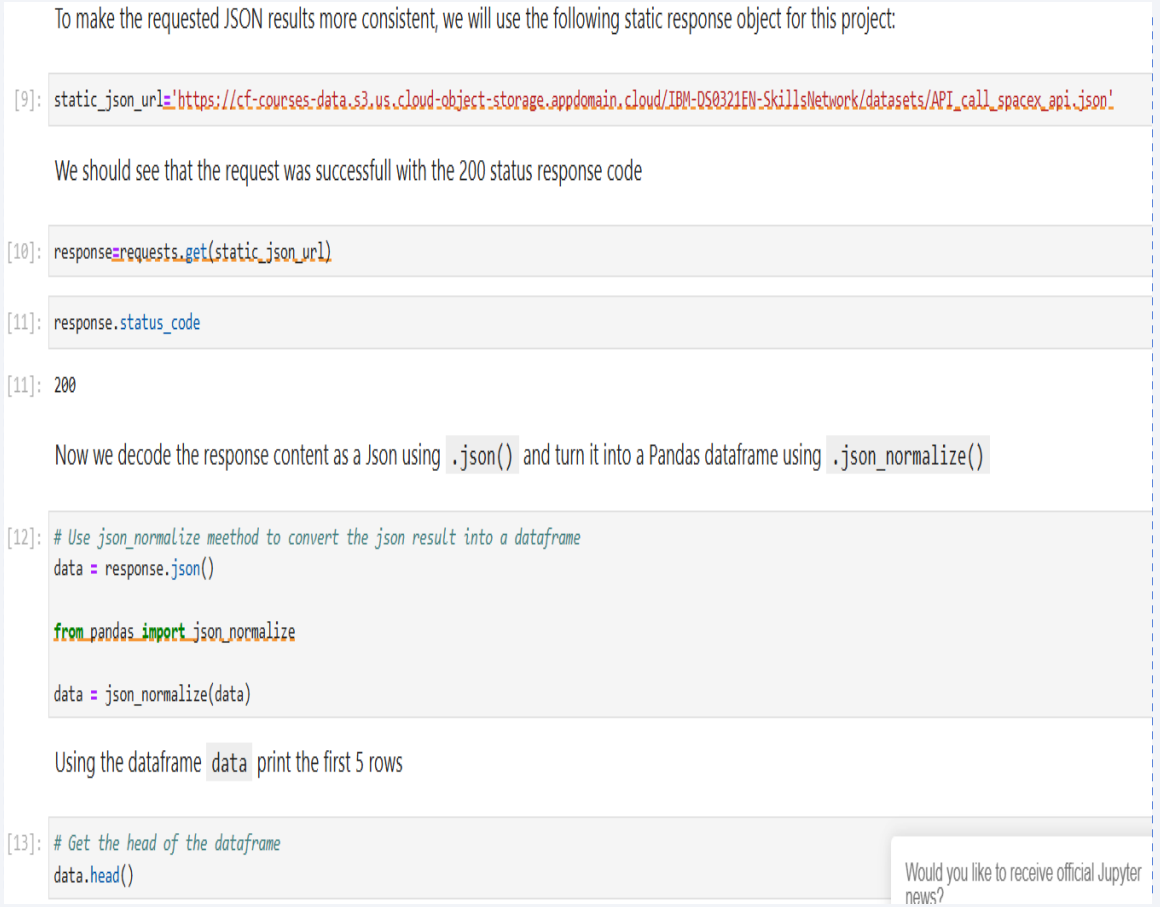
```
[13]: # Get the head of the dataframe
data.head()
```

Would you like to receive official Jupyter news?

Data Collection - Scraping

- Present your web scraping process using key phrases and flowcharts

- GitHub URL of the completed web scraping notebook, as an external reference and peer-review purpose
- https://github.com/RaKsHiTG10/WebScrapping


To make the requested JSON results more consistent, we will use the following static response object for this project:

```
[9]: static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_call_spacex_api.json'
```

We should see that the request was successfull with the 200 status response code

```
[10]: response=requests.get(static_json_url)
```

```
[11]: response.status_code
```

```
[11]: 200
```

Now we decode the response content as a Json using `.json()` and turn it into a Pandas dataframe using `.json_normalize()`

```
[12]: # Use json_normalize meethod to convert the json result into a dataframe
data = response.json()

from pandas import json_normalize

data = json_normalize(data)
```

Using the dataframe `data` print the first 5 rows

```
[13]: # Get the head of the dataframe
data.head()
```

Would you like to receive official Jupyter news?

Data Wrangling

- Identified missing data using .isnull().sum()
- **LandingPad** has the most missing values → kept as None to reflect "no landing pad used"
- **PayloadMass** missing values → will be handled via imputation or removal depending on model requirement
- No missing values in critical features like FlightNumber, Date, BoosterVersion, Orbit, Outcome, etc.
- Ensures dataset is clean and ready for analysis

EDA with Data Visualization

Chart Type	Purpose
Histogram / Distribution Plots	To understand feature distributions (e.g., PayloadMass, Flights) and detect skew/outliers
Bar Charts	To compare categorical variables like Orbit, LaunchSite, BoosterVersion vs. landing outcome
Scatter Plots	To visualize relationships between numerical features (e.g., PayloadMass vs. Launch Outcome)
Heatmap (Correlation Matrix)	To identify correlation strength between numeric features and eliminate redundancy
Pie Chart / Count Plot	To show distribution of success vs failure in landing outcomes
Box Plots	To show value spread and detect outliers for continuous variables like PayloadMass
Line Charts (Time-based)	To observe trends over time such as launch frequency or success rate progression

EDA with SQL

- **Loaded Falcon 9 dataset into Db2 table** using INSERT INTO and checked with SELECT * LIMIT
- **Retrieved basic metadata** using SELECT COUNT(*), DISTINCT, and column selection
- **Filtered launch records** using WHERE (e.g., missions with successful landings)
- **Used pattern matching queries** with LIKE to find specific Booster versions
- **Sorted data** using ORDER BY for payload mass and date-based analysis
- **Grouped launches** using GROUP BY to count missions by Orbit, LaunchSite, BoosterVersion, etc.
- **Applied aggregate functions** (AVG, MIN, MAX, SUM, COUNT) for statistical insights
- **Used nested/sub-queries** to extract top payload missions and reusable booster flights
- **Performed conditional analysis** using CASE WHEN for landing success classification
- **Joined tables / used temporary SELECT statements** to correlate booster reuse and landing outcomes

Build an Interactive Map with Folium

- **Folium Map Objects Used & Purpose**
- **Base Map** – created with folium.Map() to visualize launch locations geographically
- **Marker Objects (folium.Marker)** – added to show individual launch sites with clickable popups
- **Circle / CircleMarker (folium.Circle, folium.CircleMarker)** – used to highlight launch site regions and visualize relative distances
- **Color-coded Circles** – applied different colors to indicate landing outcomes (success vs failure)
- **Popups & Tooltips** – added metadata like launch site name, payload mass, landing result
- **Polyline / Lines (folium.PolyLine)** – used to illustrate hypothetical flight paths or distance measurements
- **Clustered Markers (MarkerCluster)** – used to group dense markers and improve visibility
- **GeoJSON Overlay** – used to display boundary or geographic region shape files (if used)
- **Heatmap (optional)** – used to show high-density launch/landing areas visually

Build a Dashboard with Plotly Dash

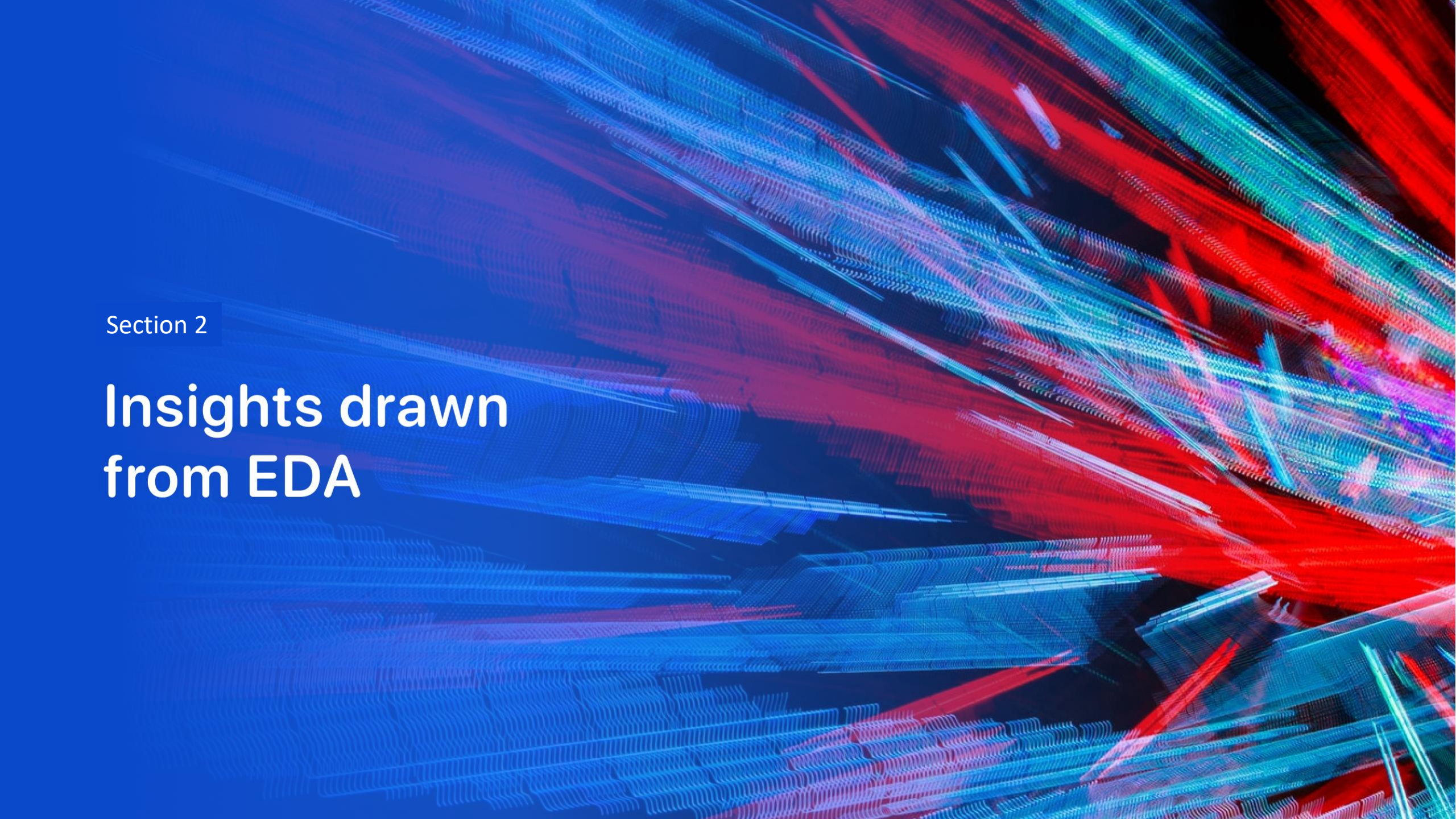
- Summarize what plots/graphs and interactions you have added to a dashboard
- Explain why you added those plots and interactions
- Add the GitHub URL of your completed Plotly Dash lab, as an external reference and peer-review purpose

Predictive Analysis (Classification)

- Summarize how you built, evaluated, improved, and found the best performing classification model
- You need present your model development process using key phrases and flowchart
- Add the GitHub URL of your completed predictive analysis lab, as an external reference and peer-review purpose

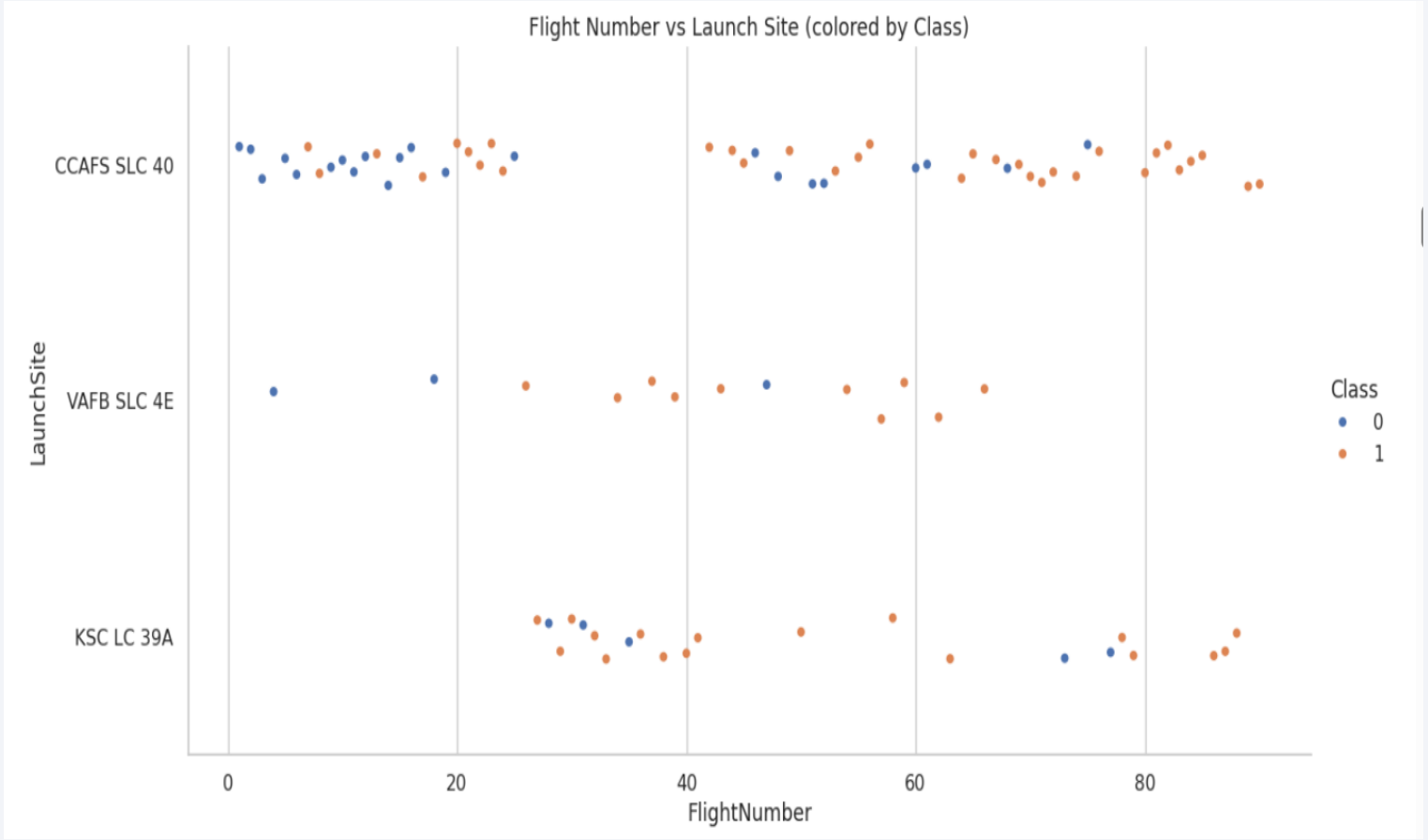
Results

- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results


Section 2
Insights drawn from EDA

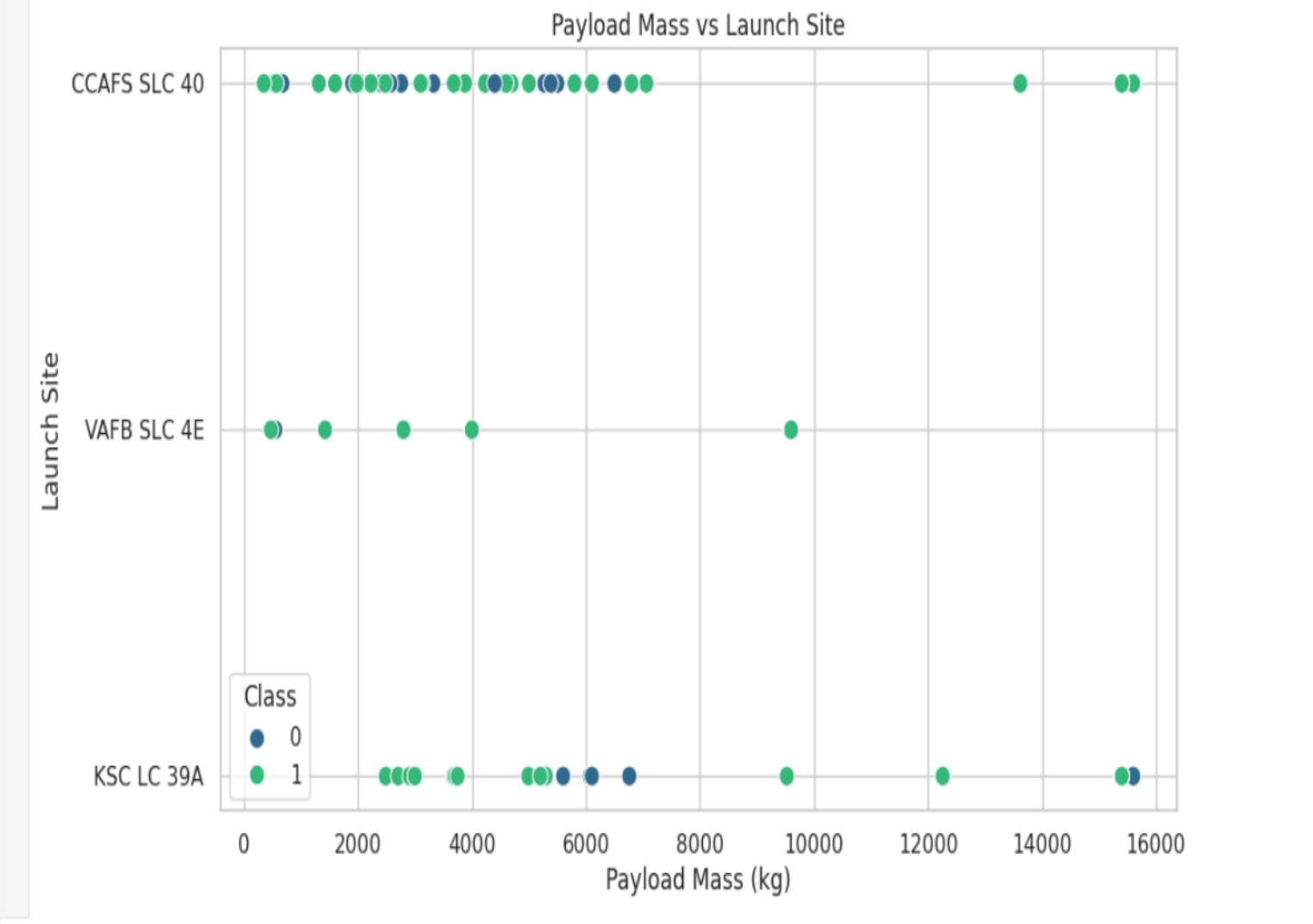
Flight Number vs. Launch Site

- Show a scatter plot of Flight Number vs. Launch Site



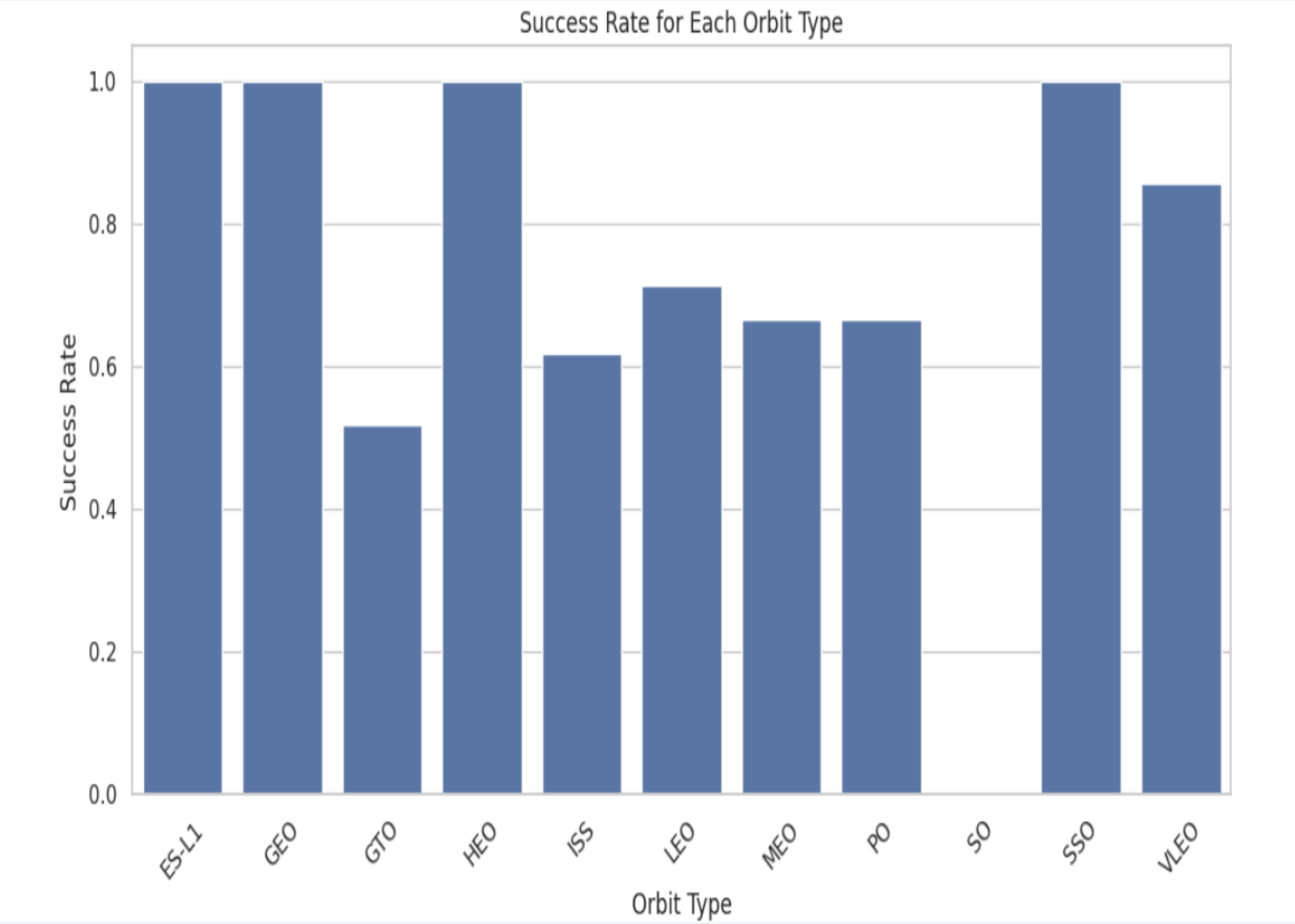
Payload vs. Launch Site

- Show a scatter plot of Payload vs. Launch Site



Success Rate vs. Orbit Type

- Show a bar chart for the success rate of each orbit type

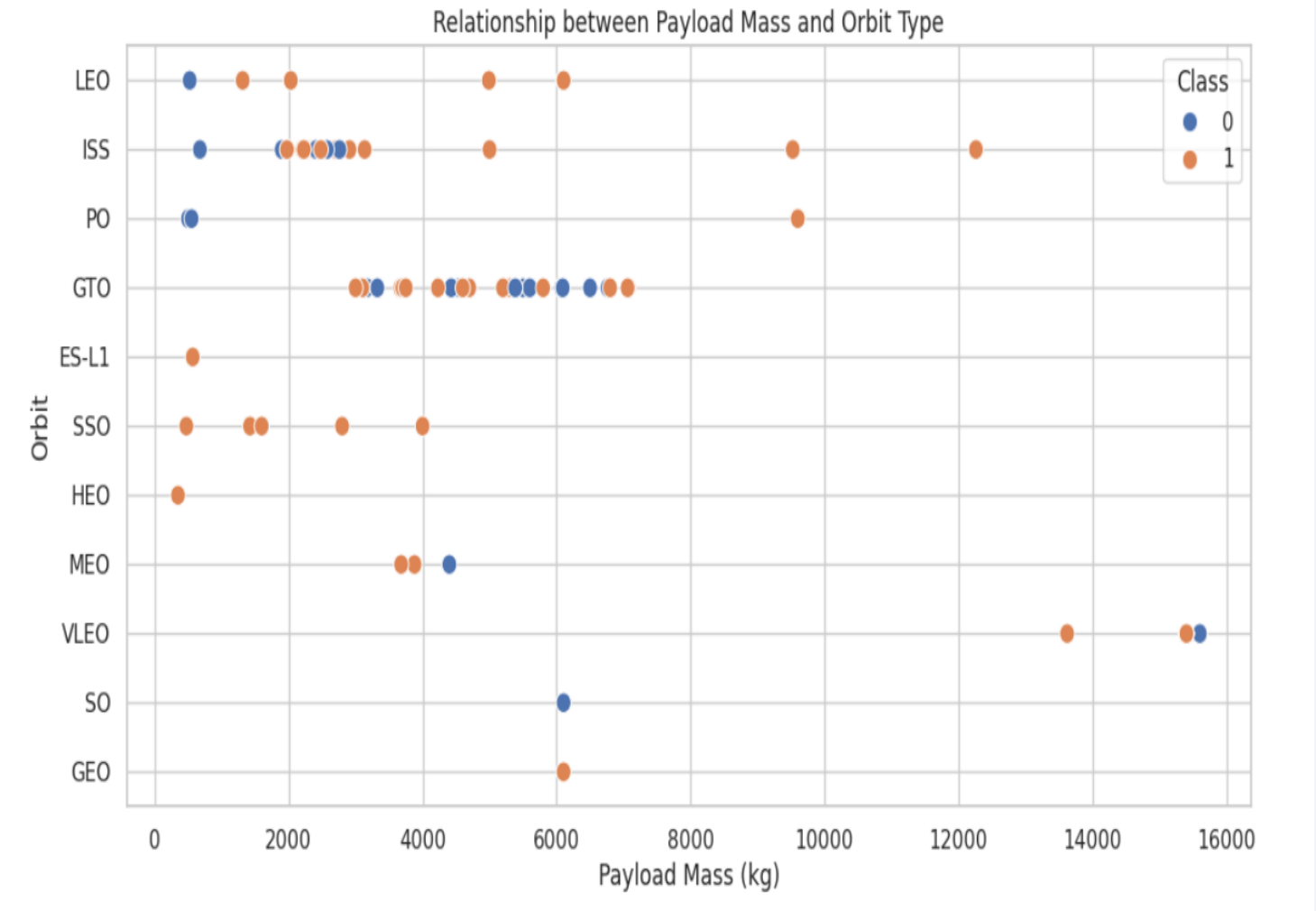


Flight Number vs. Orbit Type

- Show a scatter point of Flight number vs. Orbit type

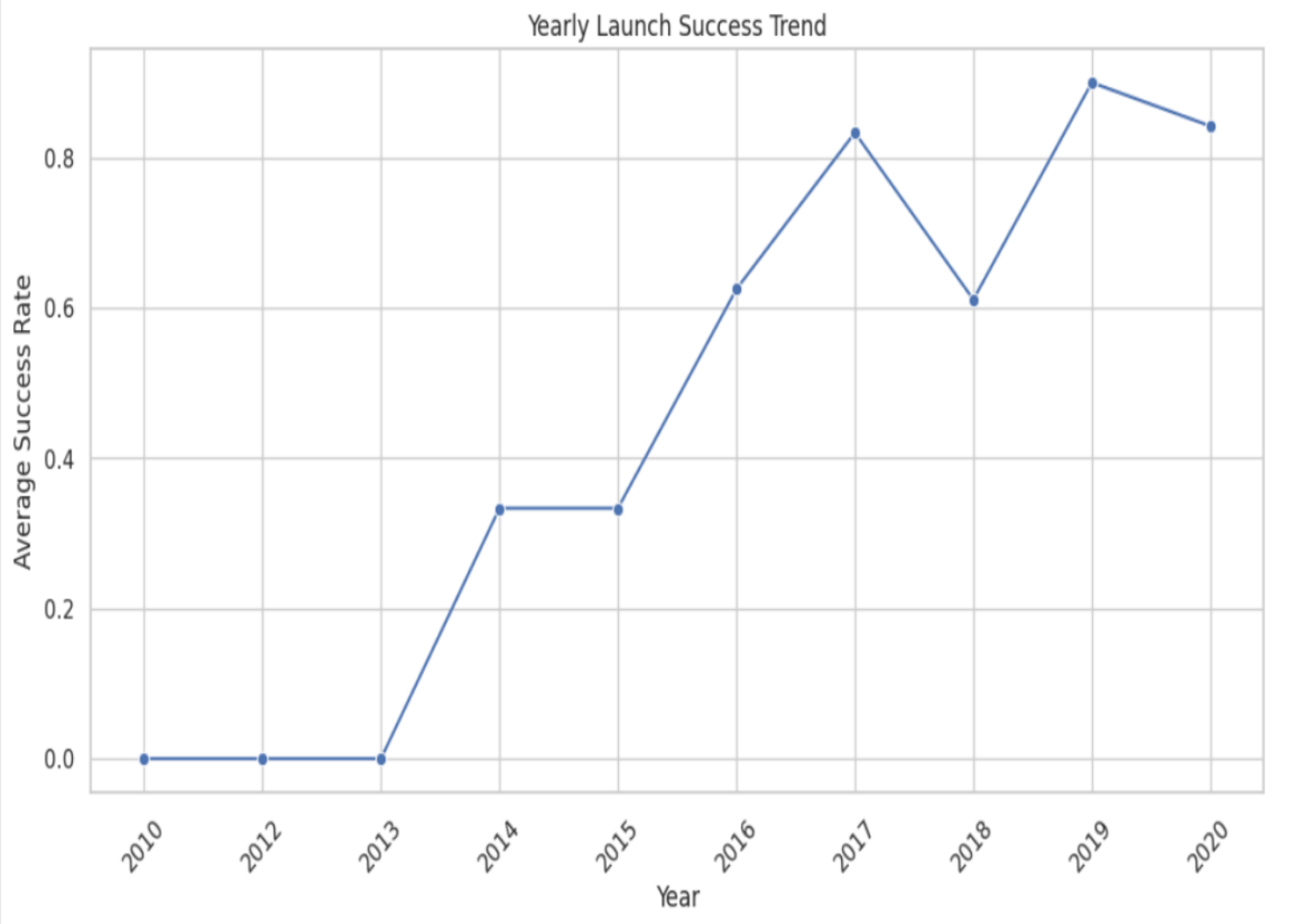
Payload vs. Orbit Type

- Show a scatter point of payload vs. orbit type



Launch Success Yearly Trend

- Show a line chart of yearly average success rate



All Launch Site Names

- Find the names of the unique launch sites

```
features['LaunchSite'].unique()
```

```
array(['CCAFS SLC 40', 'VAFB SLC 4E', 'KSC LC 39A'], dtype=object)
```

```
%%sql
SELECT DISTINCT "Launch_Site"
FROM SPACEXTABLE;
```

```
 * sqlite:///my_data1.db
Done.
```

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

Launch Site Names Begin with 'CCA'

- Find 5 records where launch sites begin with `CCA`

```
%%sql
SELECT *
FROM SPACEXTABLE
WHERE "Launch_Site" LIKE 'CCA%'
LIMIT 5;
```

```
 * sqlite:///my_data1.db
Done.
```

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Total Payload Mass

- Calculate the total payload carried by boosters from NASA

Task 3

Display the total payload mass carried by boosters launched by NASA (CRS)

```
%%sql
SELECT SUM("PAYLOAD_MASS__KG_") AS Total_Payload_Mass
FROM SPACEXTABLE
WHERE Customer LIKE 'NASA (CRS)%';
```

```
 * sqlite:///my_data1.db
Done.
```

Total_Payload_Mass
48213

Average Payload Mass by F9 v1.1

- Calculate the average payload mass carried by booster version F9 v1.1

Task 4

Display average payload mass carried by booster version F9 v1.1

```
13]: %%sql
SELECT AVG("PAYLOAD_MASS__KG_") AS Avg_Payload_Mass
FROM SPACEXTABLE
WHERE "Booster_Version" = 'F9 v1.1';
```

```
 * sqlite:///my_data1.db
Done.
```

13]:

Avg_Payload_Mass
2928.4

First Successful Ground Landing Date

- Find the dates of the first successful landing outcome on ground pad

Task 5

List the date when the first succesful landing outcome in ground pad was acheived.

Hint:Use min function

```
[16]: %%sql
SELECT MIN("Date") AS First_Successful_Ground_Pad
FROM SPACEXTABLE
WHERE "Landing_Outcome" = 'Success (ground pad)';
```

```
 * sqlite:///my_data1.db
Done.
```

[16]:

First_Successful_Ground_Pad
2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

- List the names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000

Task 6

List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000

```
[7]: %%sql
SELECT "Booster_Version"
FROM SPACEXTABLE
WHERE "Landing_Outcome" = 'Success (drone ship)'
  AND "PAYLOAD_MASS__KG_" > 4000
  AND "PAYLOAD_MASS__KG_" < 6000;
```

```
 * sqlite:///my_data1.db
Done.
```

[7]:

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

Total Number of Successful and Failure Mission Outcomes

- Calculate the total number of successful and failure mission outcomes

Task 7

List the total number of successful and failure mission outcomes

[18]:
```
%%sql
SELECT "Mission_Outcome", COUNT(*) AS Total
FROM SPACEXTABLE
GROUP BY "Mission_Outcome";
```

```
 * sqlite:///my_data1.db
Done.
```

[18]:

Mission_Outcome	Total
Failure (in flight)	1
Success	98
Success	1
Success (payload status unclear)	1

Boosters Carried Maximum Payload

- List the names of the booster which have carried the maximum payload mass

Task 8

List all the booster_versions that have carried the maximum payload mass, using a subquery with a suitable aggregate function.

[19]:
```
%%sql
SELECT "Booster_Version", "PAYLOAD_MASS__KG_"
FROM SPACEXTABLE
WHERE "PAYLOAD_MASS__KG_" = (
    SELECT MAX("PAYLOAD_MASS__KG_")
    FROM SPACEXTABLE
);
```

```
 * sqlite:///my_data1.db
Done.
```

[19]:

Booster_Version	PAYLOAD_MASS__KG_
F9 B5 B1048.4	15600
F9 B5 B1049.4	15600
F9 B5 B1051.3	15600
F9 B5 B1056.4	15600
F9 B5 B1048.5	15600

2015 Launch Records

- List the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015

Task 9

List the records which will display the month names, failure landing_outcomes in drone ship ,booster versions, launch_site for the months in year 2015.

Note: SQLLite does not support monthnames. So you need to use substr(Date, 6,2) as month to get the months and substr(Date,0,5)='2015' for year.

```
%%sql
SELECT
    substr("Date", 6, 2) AS Month,
    "Landing_Outcome",
    "Booster_Version",
    "Launch_Site"
FROM SPACEXTABLE
WHERE substr("Date", 1, 4) = '2015'
  AND "Landing_Outcome" = 'Failure (drone ship)';
```

```
 * sqlite:///my_data1.db
Done.
```

Month	Landing_Outcome	Booster_Version	Launch_Site
01	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
04	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

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

- Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order

Task 10

Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order.

[21]:
```
%%sql
SELECT
    "Landing_Outcome",
    COUNT(*) AS Outcome_Count
FROM SPACEXTABLE
WHERE "Date" BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY "Landing_Outcome"
ORDER BY Outcome_Count DESC;
```

```
 * sqlite:///my_data1.db
Done.
```

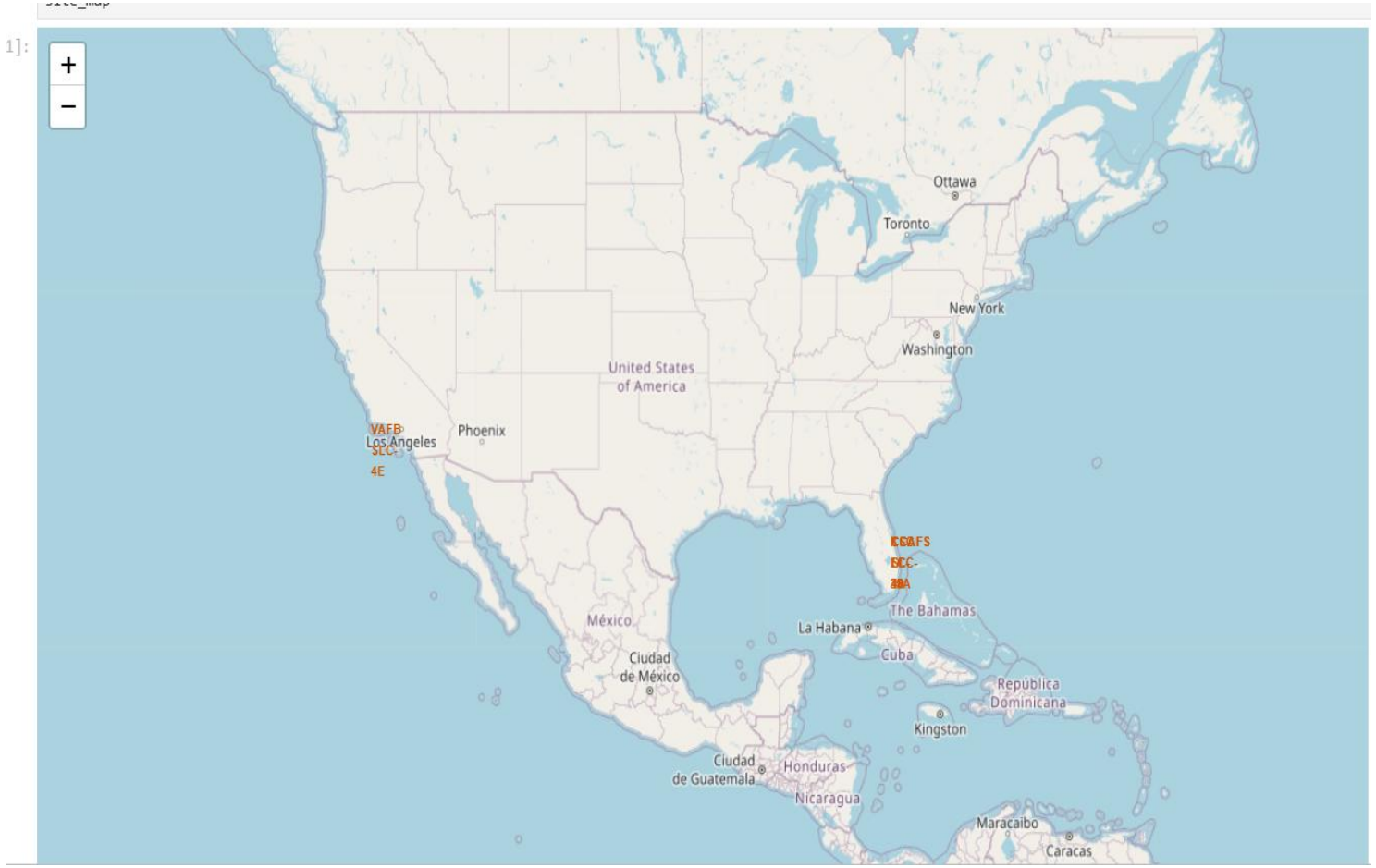
[21]:

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

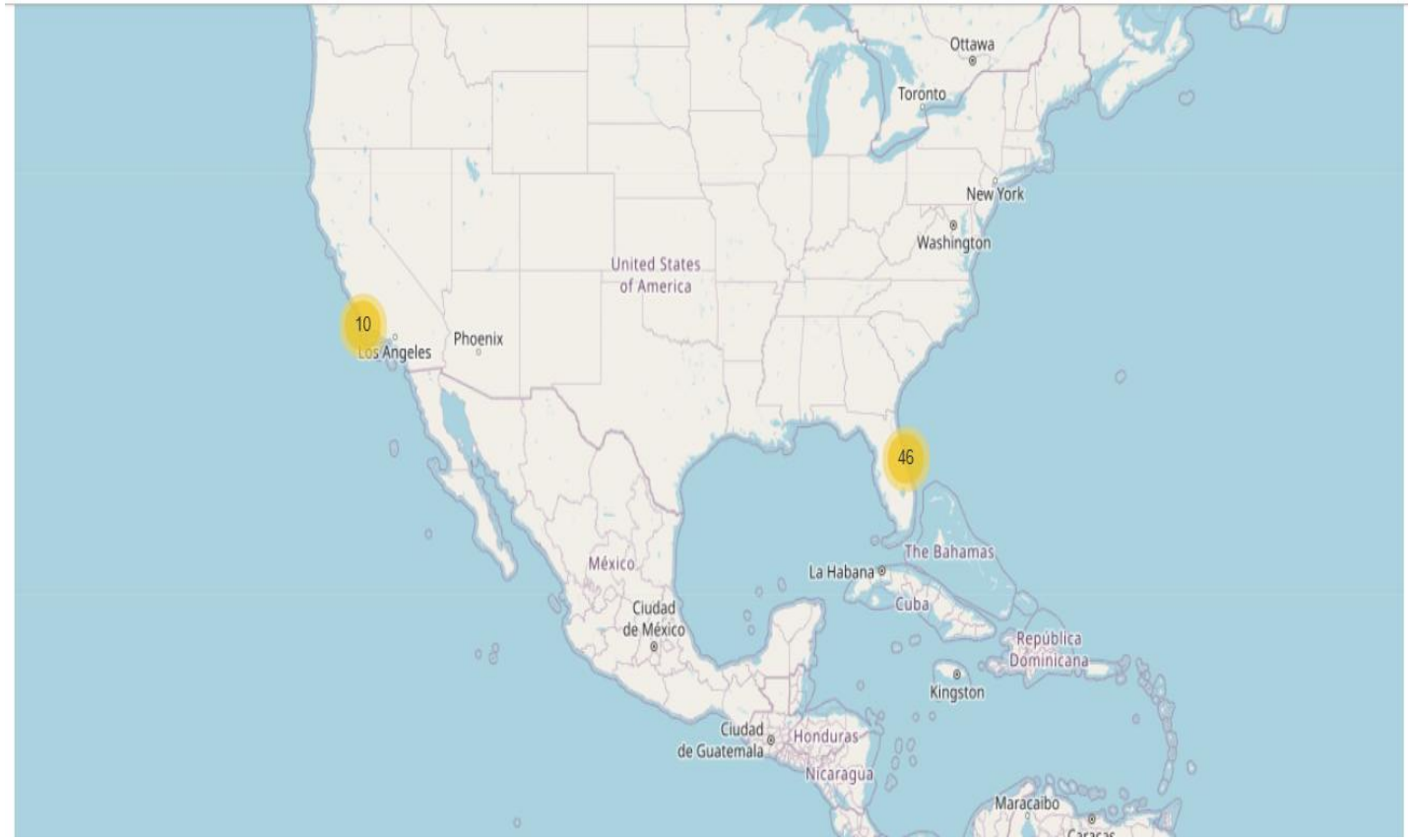
Section 3

Launch Sites Proximities Analysis

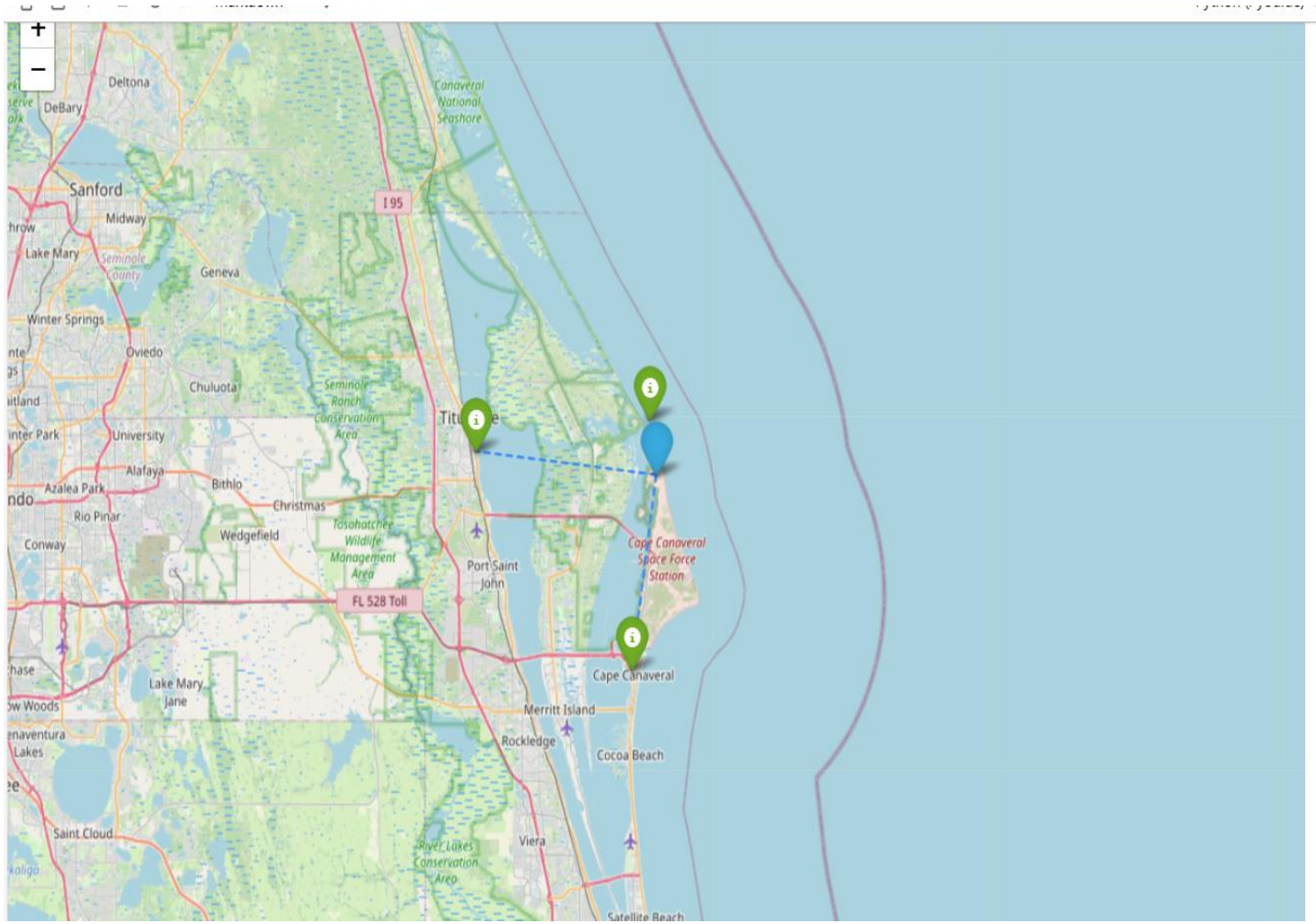
Folium Map Screenshot 1

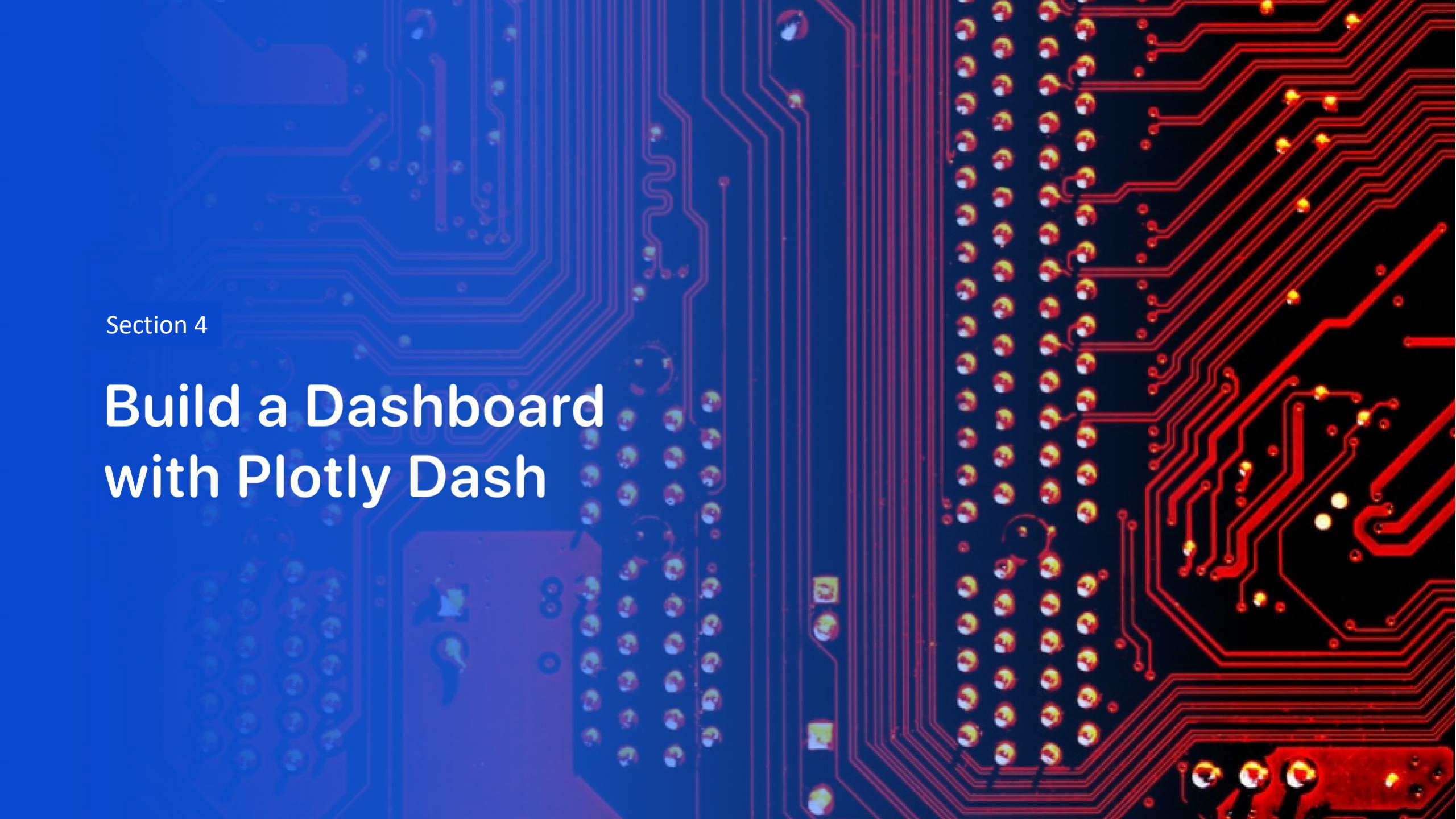


Folium Map Screenshot 2

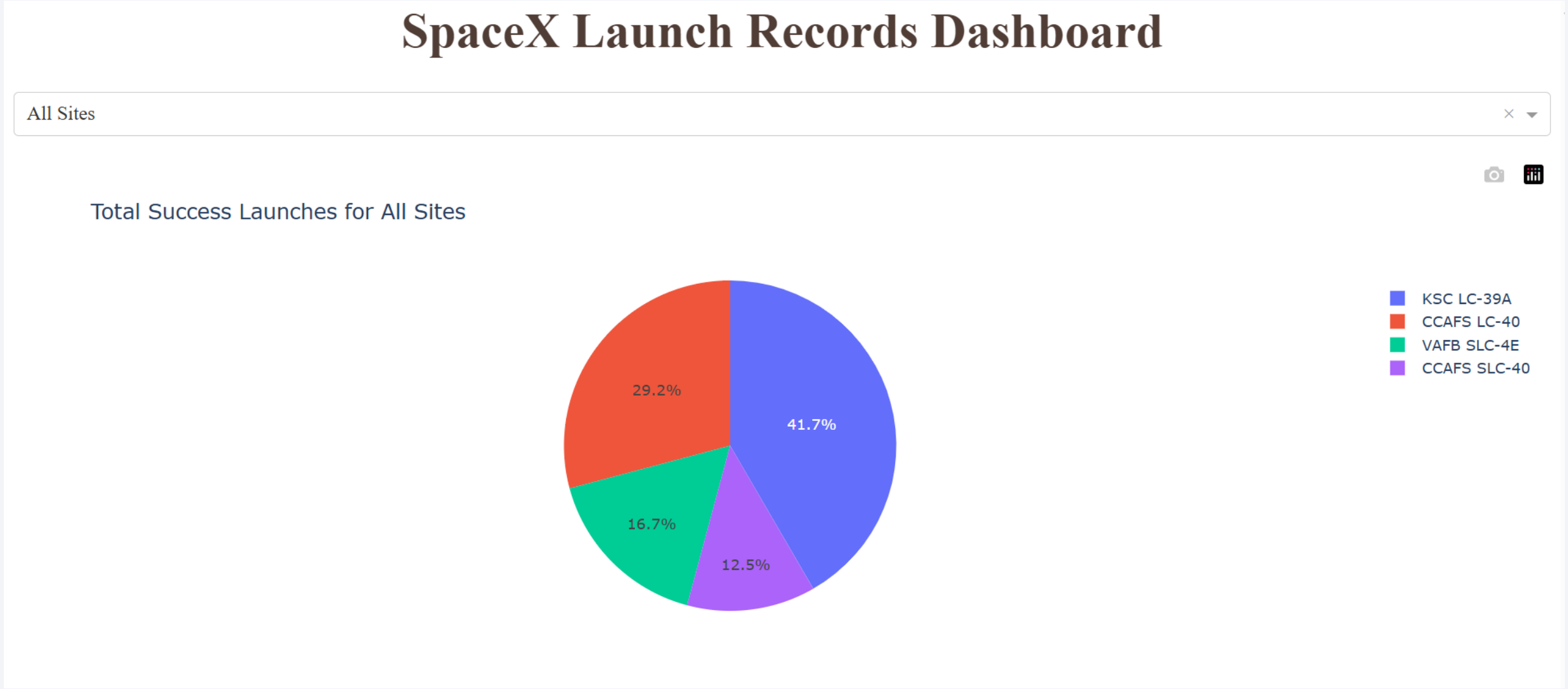


Folium Map Screenshot 3

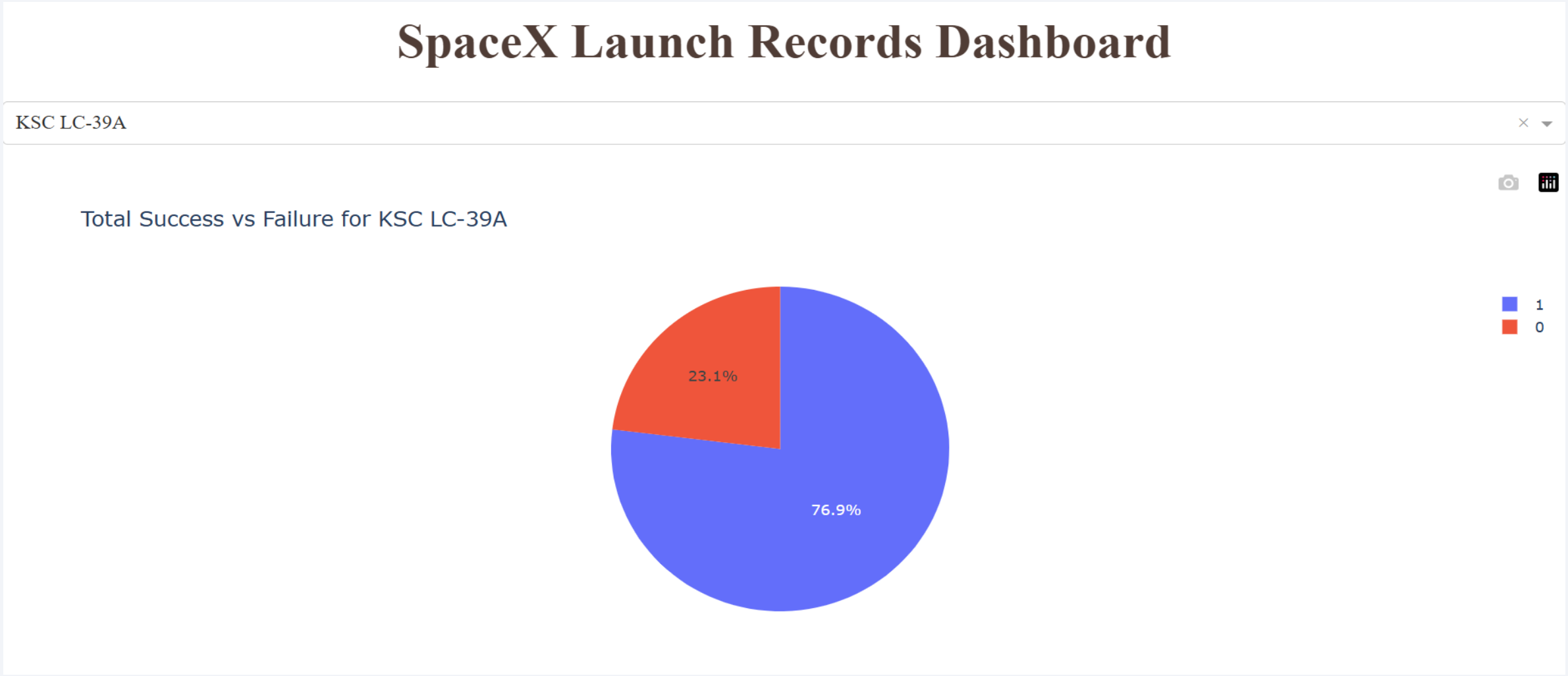



Section 4
Build a Dashboard with Plotly Dash

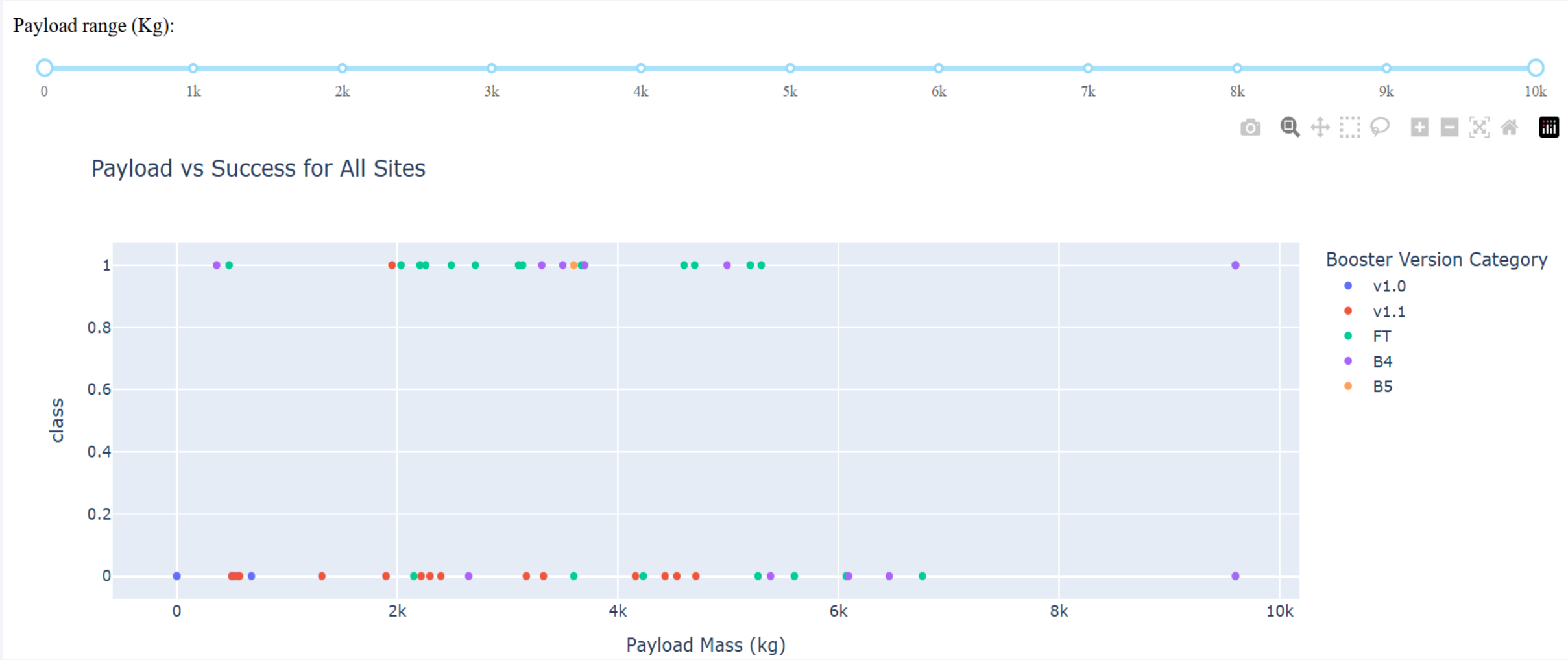
Dashboard Screenshot 1



Dashboard Screenshot 2



Dashboard Screenshot 3



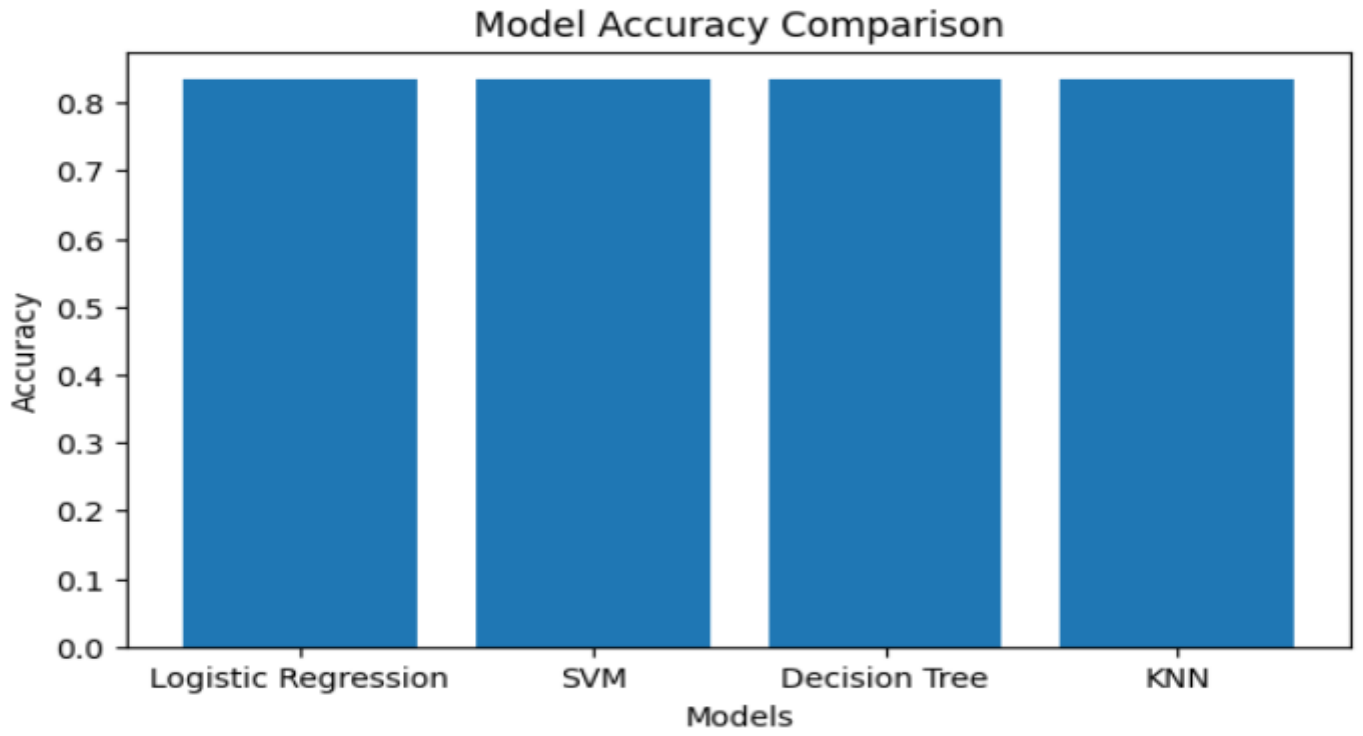
Section 5

Predictive Analysis (Classification)

Classification Accuracy

```
best_model = max(model_scores, key=model_scores.get)
print("\nBest Performing Model:", best_model)
```

```
Logistic Regression: 0.8333
SVM: 0.8333
Decision Tree: 0.8333
KNN: 0.8333
```



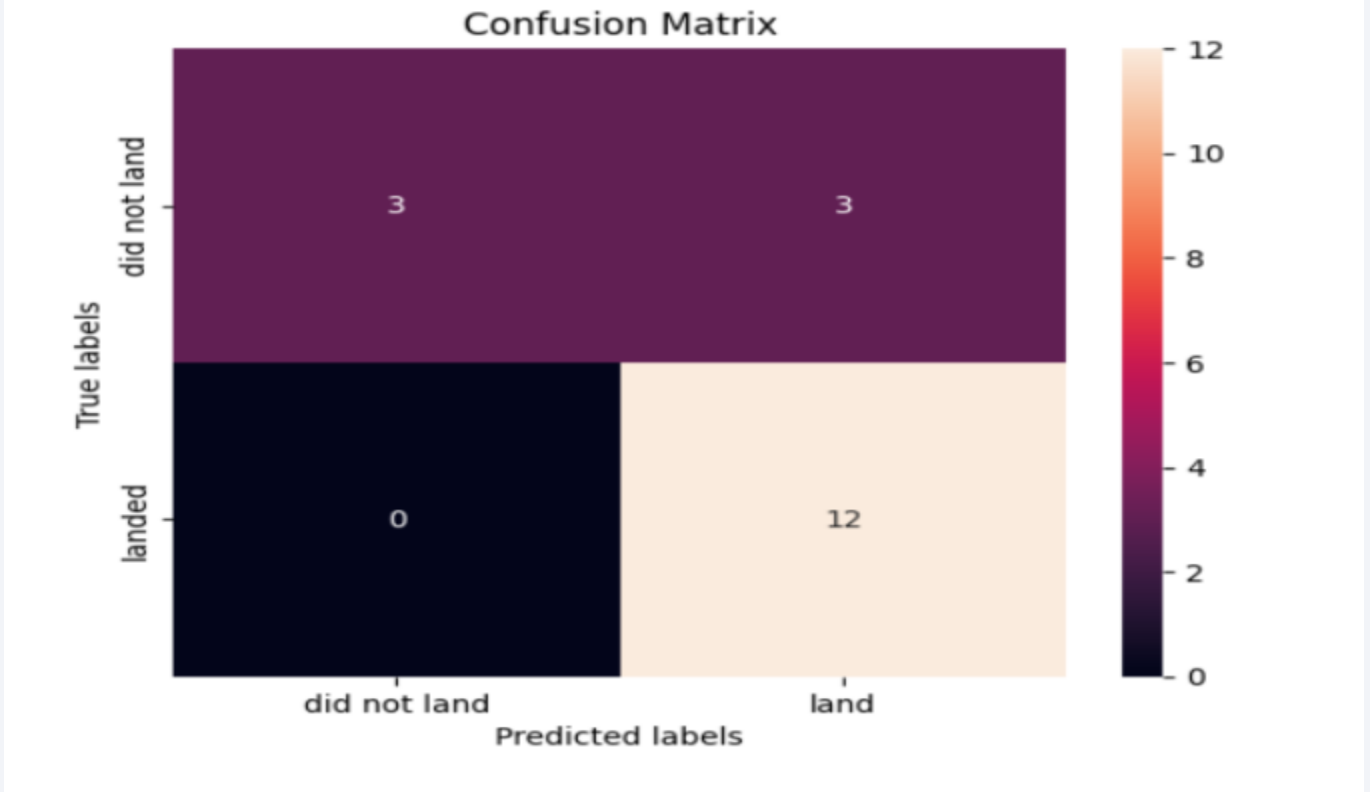

```
Best Performing Model: Logistic Regression
```

Confusion Matrix

```
Test Accuracy: 0.8333333333333334
```

Lets look at the confusion matrix:

```
]: yhat=logreg_cv.predict(X_test)
   plot_confusion_matrix(Y_test,yhat)
```

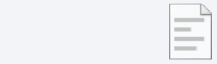


Conclusions

- Successfully built and evaluated multiple classification models to predict Falcon 9 first stage landing success
- Logistic Regression achieved the best overall performance on test data
- Confusion matrix shows strong ability to distinguish landing vs non-landing outcomes
- Standardization and hyperparameter tuning significantly improved model accuracy
- Results demonstrate that the landing outcome is predictable using engineered features
- Final model can support cost estimation and competitive bid decisions for launch providers
- Workflow is fully reproducible using Python, Scikit-Learn, and GridSearchCV

Appendix

- Include any relevant assets like Python code snippets, SQL queries, charts, Notebook outputs, or data sets that you may have created during this project



jupyter-labs-webscraping (1).ipynb

jupyter-labs-spacex-data-collection-api (1).ipynb

labs-jupyter-spacex-Data wrangling.ipynb

jupyter-labs-eda-sql-coursera_sqllite.ipynb

lab_jupyter_launch_site_location.ipynb

spacex-dash-app.py

Thank you!