IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Artyom
Ashigov
2024-09-09

Outline

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

Executive Summary

- Summary of methodologies
  - Data Collection through API
  - Data Collection with Web Scraping
  - Data Wrangling
  - Exploratory Data Analysis with SQL
  - Exploratory Data Analysis with Data Visualization
  - Interactive Visual Analytics with Folium
  - ML Prediction
- Summary of all results
  - Exploratory Data Analysis result
  - Interactive analytics in screenshots
  - Predictive Analytics result

Introduction

- **Project background and context**

Space X advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is because Space X can reuse the first stage. Therefore, if we can determine if the first stage will land, we can determine the cost of a launch. This information can be used if an alternate company wants to bid against space X for a rocket launch. This goal of the project is to create a machine learning pipeline to predict if the first stage will land successfully.

- **Problems you want to find answers**

- What factors determine if the rocket will land successfully?

- The interaction amongst various features that determine the success rate of a successful landing.

- What operating conditions needs to be in place to ensure a successful landing program.

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - Data was collected using SpaceX API and web scraping from Wikipedia
- Perform data wrangling
  - One-hot encoding was applied to categorical features
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - How to build, tune, evaluate classification models

Data Collection

The data was collected using various methodologies:

- Data collection was done using get request to the SpaceX API.
- Next, we decoded the response content as a Json using .json() function call and turn it into a pandas dataframe using .json_normalize().
- We then cleaned the data, checked for missing values and fill in missing values where necessary.
- In addition, we performed web scraping from Wikipedia for Falcon 9 launch records with BeautifulSoup.
- The objective was to extract the launch records as HTML table, parse the table and convert it to a pandas dataframe for future analysis.

Data Collection – SpaceX API

- We used the get request to the SpaceX API to collect data, clean the requested data and did some basic data wrangling and formatting.

- https://github.com/artyomashigov/ibm_ds_capstone/blob/main/jupyter-labs-spacex-data-collection-api.ipynb

1. Get request for rocket launch data using API

In [6]:
```
spacex_url="https://api.spacexdata.com/v4/launches/past"
```

In [7]:
```
response = requests.get(spacex_url)
```

2. Use json_normalize method to convert json result to dataframe

In [12]:
```
# Use json_normalize method to convert the json result into a dataframe

# decode response content as json
static_json_df = res.json()
```

In [13]:
```
# apply json_normalize
data = pd.json_normalize(static_json_df)
```

3. We then performed data cleaning and filling in the missing values

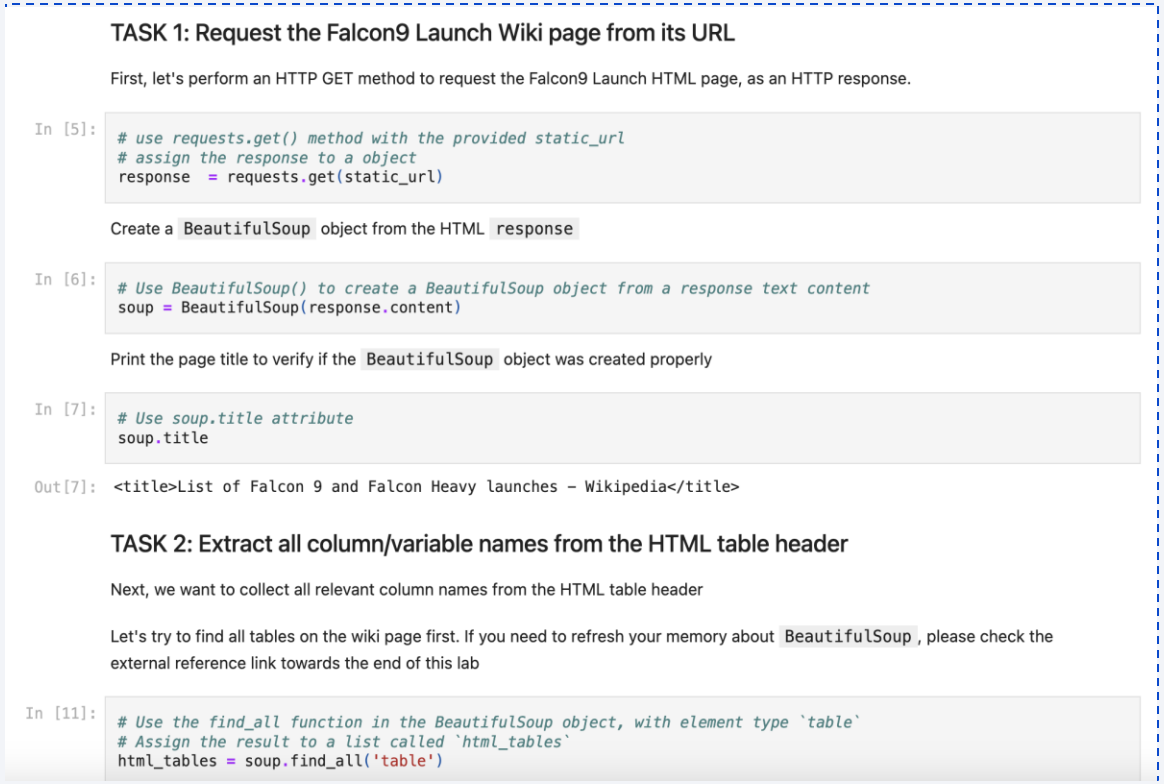
In [30]:
```
rows = data_falcon9['PayloadMass'].values.tolist()[0]

df_rows = pd.DataFrame(rows)
df_rows = df_rows.replace(np.nan, PayloadMass)

data_falcon9['PayloadMass'][0] = df_rows.values
data_falcon9
```

Data Collection - Scraping

We applied web scrapping to webscrap Falcon 9 launch records with BeautifulSoup

We parsed the table and converted it into a pandas dataframe.

- https://github.com/artyomashigov/ibm_ds_capstone/blob/main/jupyter-labs-webscraping.ipynb



TASK 1: Request the Falcon9 Launch Wiki page from its URL

First, let's perform an HTTP GET method to request the Falcon9 Launch HTML page, as an HTTP response.

In [5]:
```
# use requests.get() method with the provided static_url
# assign the response to a object
response  = requests.get(static_url)
```

Create a `BeautifulSoup` object from the HTML `response`

In [6]:
```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(response.content)
```

Print the page title to verify if the `BeautifulSoup` object was created properly

In [7]:
```
# Use soup.title attribute
soup.title
```

Out[7]:
```
<title>List of Falcon 9 and Falcon Heavy launches – Wikipedia</title>
```

TASK 2: Extract all column/variable names from the HTML table header

Next, we want to collect all relevant column names from the HTML table header

Let's try to find all tables on the wiki page first. If you need to refresh your memory about `BeautifulSoup`, please check the external reference link towards the end of this lab

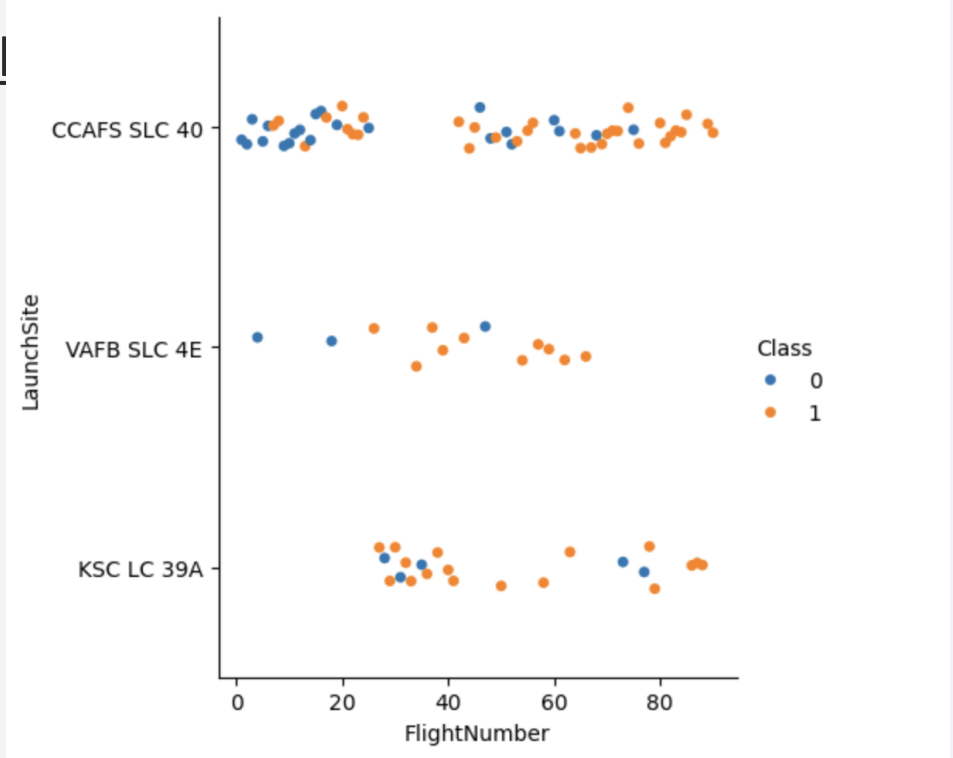
In [11]:
```
# Use the find_all function in the BeautifulSoup object, with element type `table`
# Assign the result to a list called `html_tables`
html_tables = soup.find_all('table')
```

Data Wrangling

1. We performed exploratory data analysis and determined the training labels.
2. We calculated the number of launches at each site, and the number and occurrence of each orbits
3. We created landing outcome label from outcome column and exported the results to csv.
4. Link: https://github.com/artyomashigov/ibm_ds_capstone/blob/main/labs-jupyter-spacex-Data%20wrangling.ipynb

EDA with Data Visualization

- We explored the data by visualizing the relationship between flight number and launch Site, payload and launch site, success rate of each orbit type, flight number and orbit type, the launch success yearly trend.
- https://github.com/artyomashigov/ibm_ds_capstone/bl



EDA with SQL

- We applied EDA with SQL to get insight from the data. We wrote queries to find out for instance:

- The names of unique launch sites in the space mission.

- The total payload mass carried by boosters launched by NASA (CRS)

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

- The total number of successful and failure mission outcomes

- The failed landing outcomes in drone ship, their booster version and launch site names.

- https://github.com/artyomashigov/ibm_ds_capstone/blob/main/jupyter-labs-eda-sql-coursera_sqllite.ipynb

Build an Interactive Map with Folium

- We marked all launch sites, and added map objects such as markers, circles, lines to mark the success or failure of launches for each site on the folium map.
- We assigned the feature launch outcomes (failure or success) to class 0 and 1.i.e., 0 for failure, and 1 for success.
- https://github.com/artyomashigov/ibm_ds_capstone/blob/main/lab_jupyter_launch_site_location.ipynb

Build a Dashboard with Plotly Dash

- We built an interactive dashboard with Plotly dash
- We plotted pie charts showing the total launches by a certain sites
- We plotted scatter graph showing the relationship with Outcome and Payload Mass (Kg) for the different booster version.

https://github.com/artyomashigov/ibm_ds_capstone/blob/main/spacex_dash_app.py

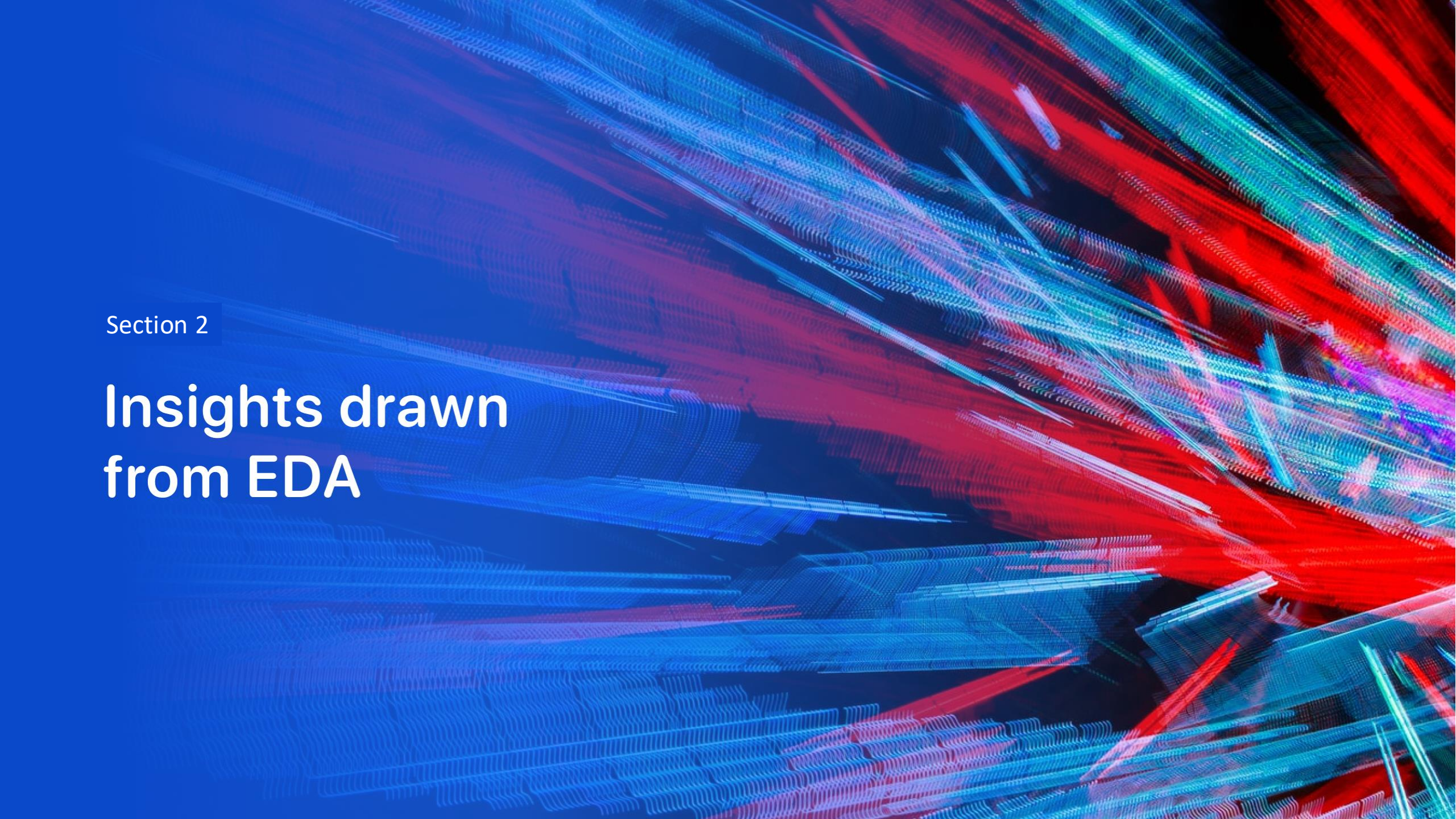
Predictive Analysis (Classification)

- We loaded the data using numpy and pandas, transformed the data, split our data into training and testing.
- We built different machine learning models and tune different hyperparameters using GridSearchCV.
- We used accuracy as the metric for our model, improved the model using feature engineering and algorithm tuning.
- We found the best performing classification model.

https://github.com/artyomashigov/ibm_ds_capstone/blob/main/SpaceX_Machine%20Learning%20Prediction_Part_5.ipynb

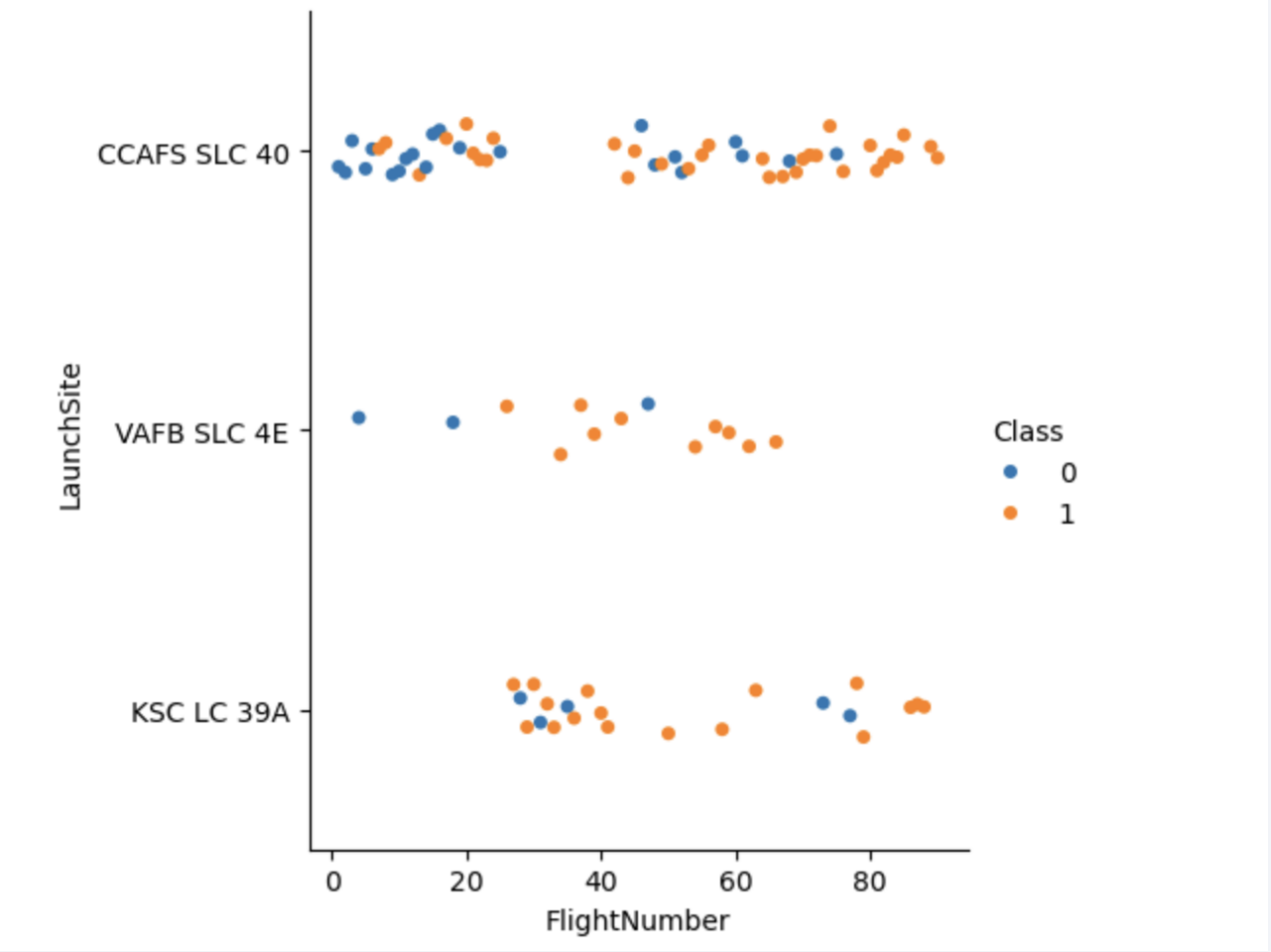
Results

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


Section 2
Insights drawn from EDA

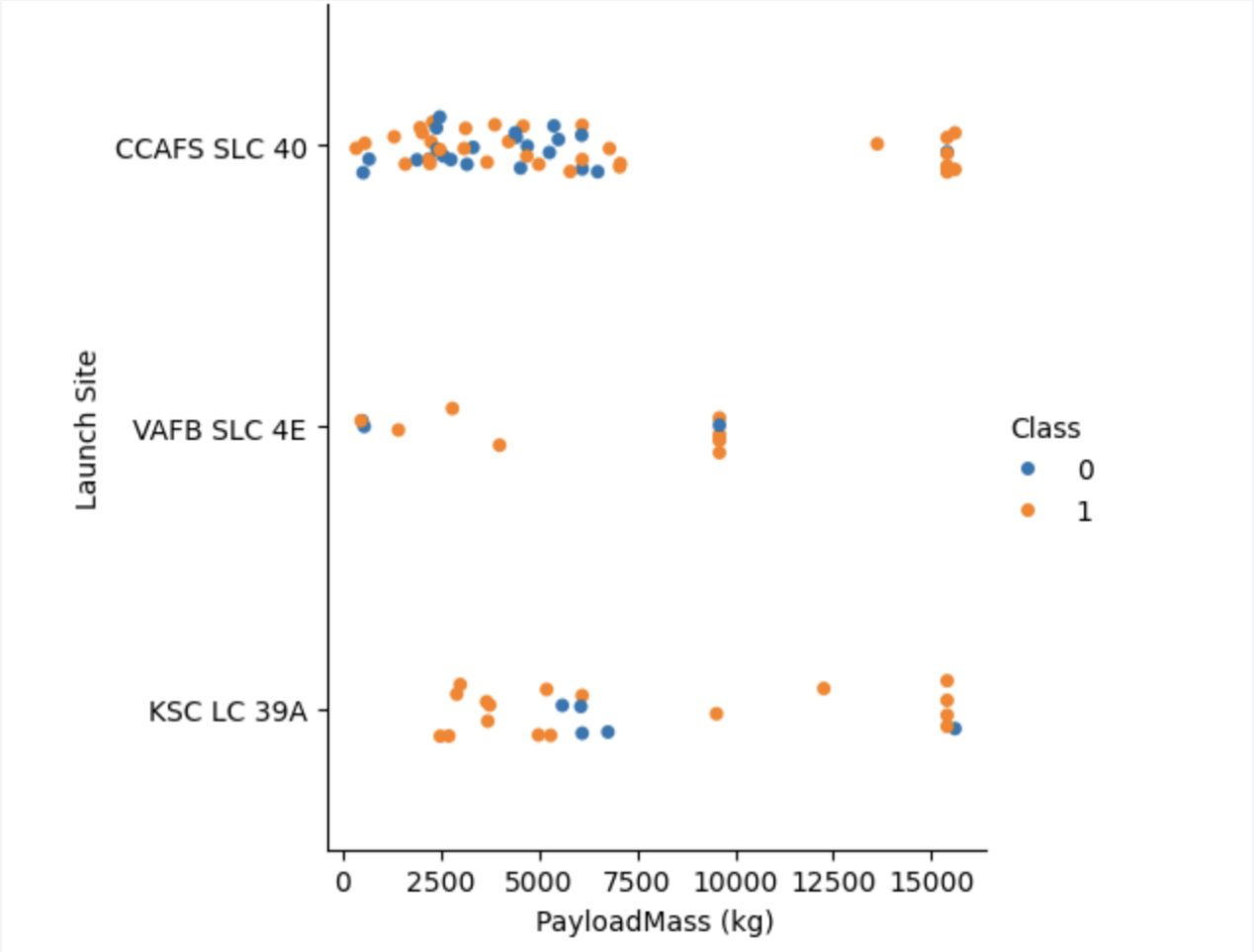
Flight Number vs. Launch Site

- From the plot, we found that the larger the flight amount at a launch site, the greater the success rate at a launch site.



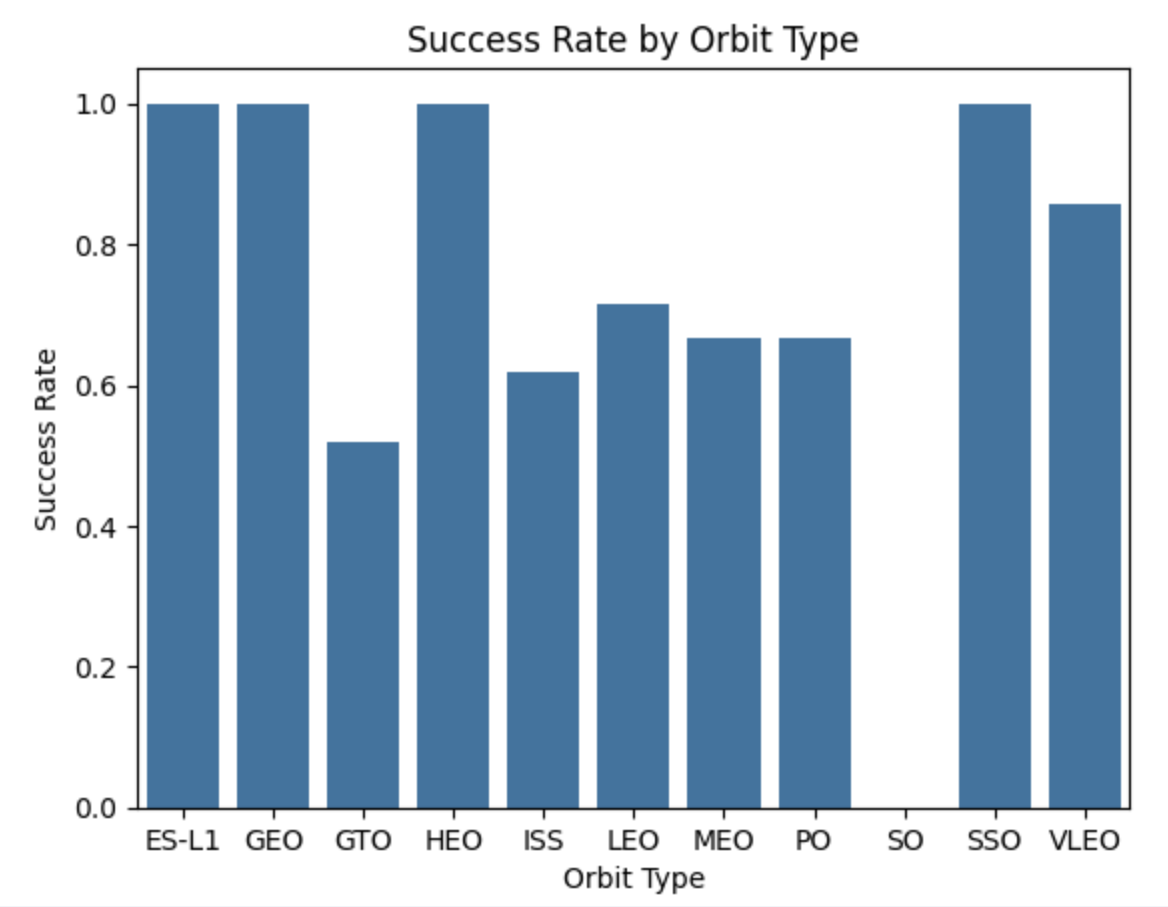
Payload vs. Launch Site

- The greater the payload the better is succes rate for the first launch site in the graph



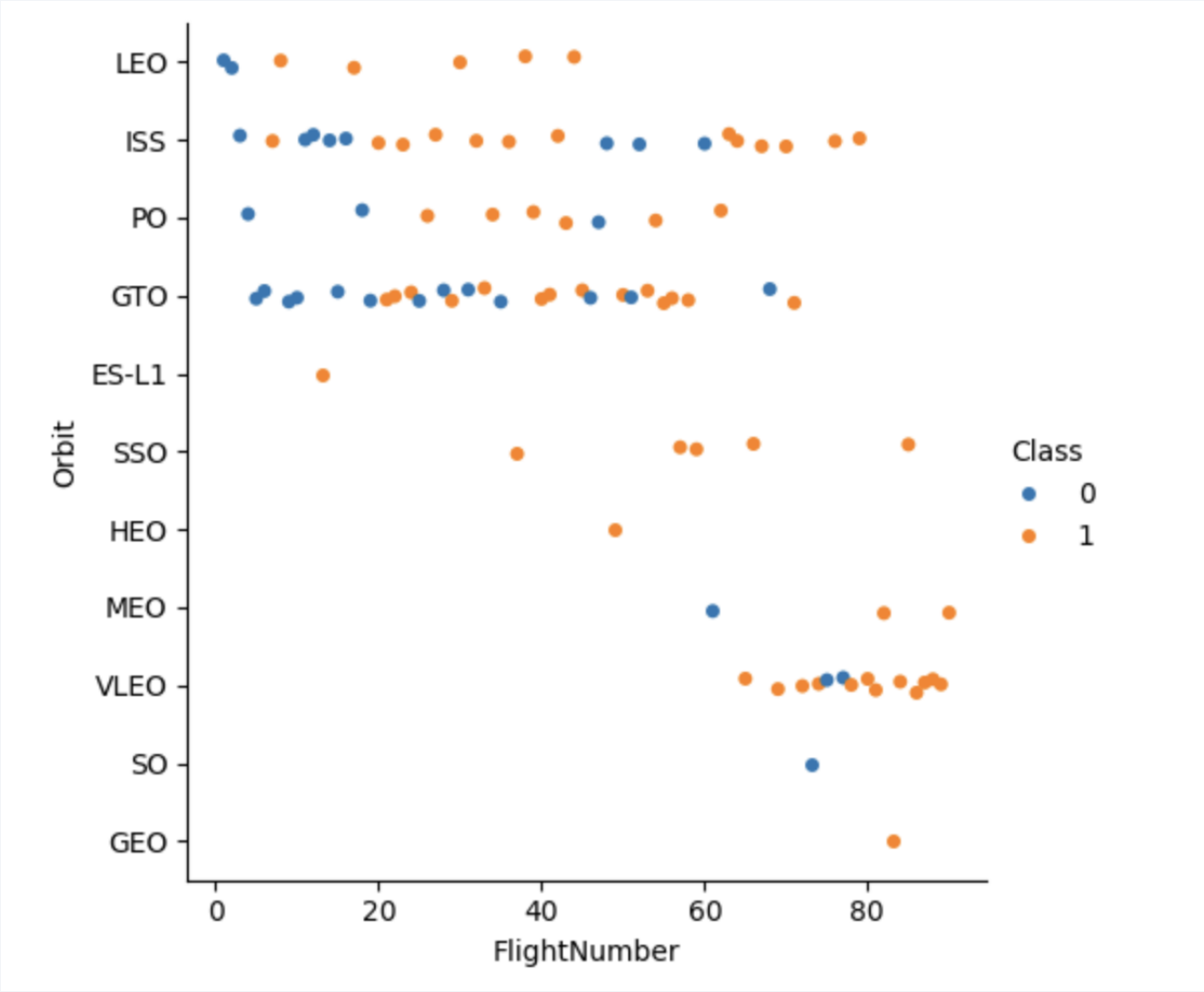
Success Rate vs. Orbit Type

- ES-L1, GEO, HEO and SSO have 100% success rate



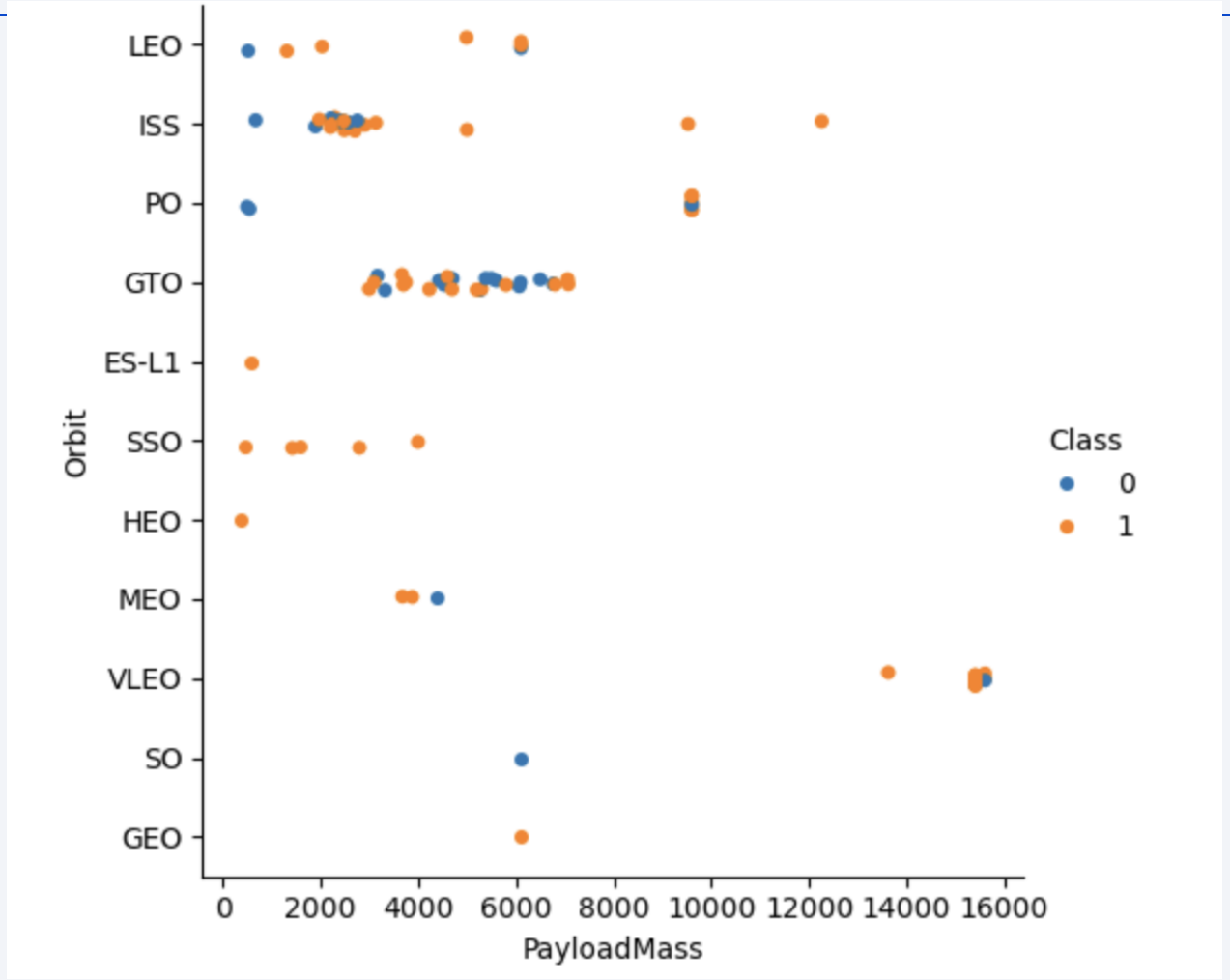
Flight Number vs. Orbit Type

- The plot visualizes the relationship between flight numbers and orbits, where the color represents the class of success or failure. Orange dots (Class 1) indicate successful launches, while blue dots (Class 0) show failures across different orbits.



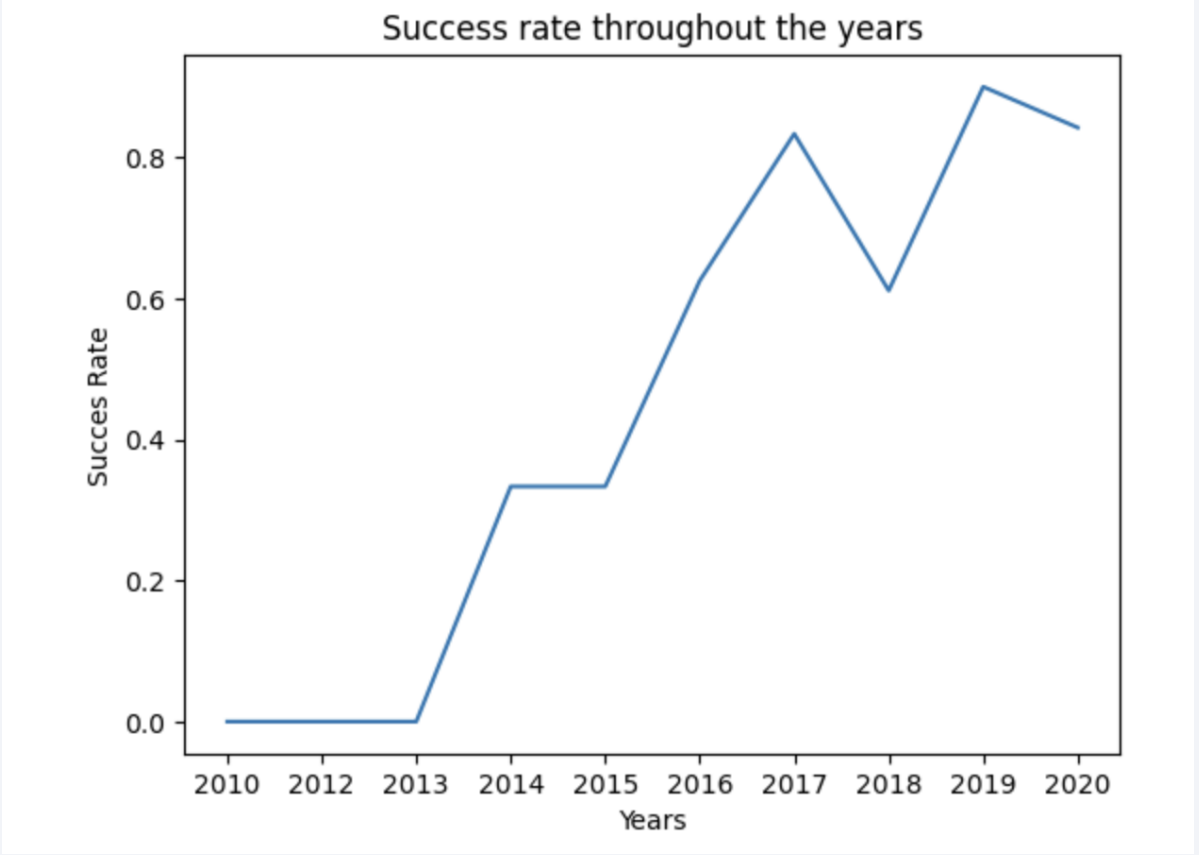
Payload vs. Orbit Type

- Most successful missions (Class 1) are spread across various orbits and payload masses, while failures (Class 0) are more scattered and less frequent.



Launch Success Yearly Trend

- The success rate has steadily increased from 2014 onwards, peaking around 2019 before experiencing a slight decline in 2020.



All Launch Site Names

- The SQL query retrieves the distinct launch site names from the SPACEXTABLE database, displaying four unique launch sites used in space missions.

Task 1

Display the names of the unique launch sites in the space mission

In [11]:
```
%sql SELECT DISTINCT("Launch_Site") FROM SPACEXTABLE
```

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

Out[11]:

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

Launch Site Names Begin with 'CCA'

Task 2

Display 5 records where launch sites begin with the string 'CCA'

In [16]:
```
%sql SELECT * FROM SPACEXTABLE WHERE "Launch_Site" LIKE "CCA%" LIMIT 5
```

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

Out[16]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (p
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 (p
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	N
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	N
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	N

Total Payload Mass

Task 3

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

In [18]:
```
%sql SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEXTABLE WHERE Customer = "NASA (CRS)"
```

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

Out[18]:

SUM(PAYLOAD_MASS__KG_)
45596

Average Payload Mass by F9 v1.1

Task 4

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

In [19]:
```
%sql SELECT AVG(PAYLOAD_MASS__KG_) FROM SPACEXTABLE WHERE "Booster_Version" LIKE "F9 v1.1%"
```

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

Out[19]:

AVG(PAYLOAD_MASS__KG_)
2534.6666666666665

First Successful Ground Landing Date

Task 5

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

Hint:Use min function

In [20]:
```
%sql SELECT min(Date) FROM SPACEXTABLE WHERE "Landing_Outcome" = "Success"
```

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

Out[20]:

min(Date)
2018-07-22

Successful Drone Ship Landing with Payload between 4000 and 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

In [24]:
```
%sql SELECT DISTINCT("Booster_Version") FROM SPACEXTABLE WHERE "Mission_Outcome" = "Success" and PAYLOAD_MASS__KG
```

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

Out[24]:

Booster_Version
F9 v1.1
F9 v1.1 B1011
F9 v1.1 B1014
F9 v1.1 B1016
F9 FT B1020
F9 FT B1022
F9 FT B1026
F9 FT B1030
F9 FT B1021.2
F9 FT B1032.1
F9 B4 B1040.1
F9 FT B1031.2
F9 FT B1032.2
F9 B4 B1040.2
F9 B5 B1046.2
F9 B5 B1047.2

F9 B5 B1048.3

F9 B5 B1051.2

F9 B5B1060.1

F9 B5 B1058.2

F9 B5B1062.1

Total Number of Successful and Failure Mission Outcomes

Task 7

List the total number of successful and failure mission outcomes

In [38]:

```
%%sql
SELECT
COUNT(CASE WHEN "Mission_Outcome" = 'Success' THEN 1 ELSE NULL END) AS successful,
COUNT(CASE WHEN "Mission_Outcome" != 'Success' THEN 1 ELSE NULL END) AS failures
FROM SPACEXTABLE
```

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

Out[38]:

successful	failures
98	3

Boosters Carried Maximum Payload

Task 8

List the names of the booster_versions which have carried the maximum payload mass. Use a subquery

In [40]:
```
%%sql
SELECT DISTINCT("Booster_Version") FROM SPACEXTABLE WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM
```

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

Out[40]:

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

2015 Launch Records

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.

In [45]:
```
%%sql
SELECT substr(Date,6,2) as month, "Booster_Version", "Launch_Site", "Landing_Outcome"
    FROM SPACEXTABLE WHERE substr(Date, 0, 5)  = '2015' and
    "Landing_Outcome" LIKE "Failure%"
```

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

Out[45]:

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

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

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.

In [54]:

```
%%sql
SELECT "Landing_Outcome", COUNT("Landing_Outcome") as Number
    FROM SPACEXTABLE WHERE Date BETWEEN "2010-06-04" AND "2017-03-20"
    GROUP BY Landing_Outcome ORDER BY Number DESC
```

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

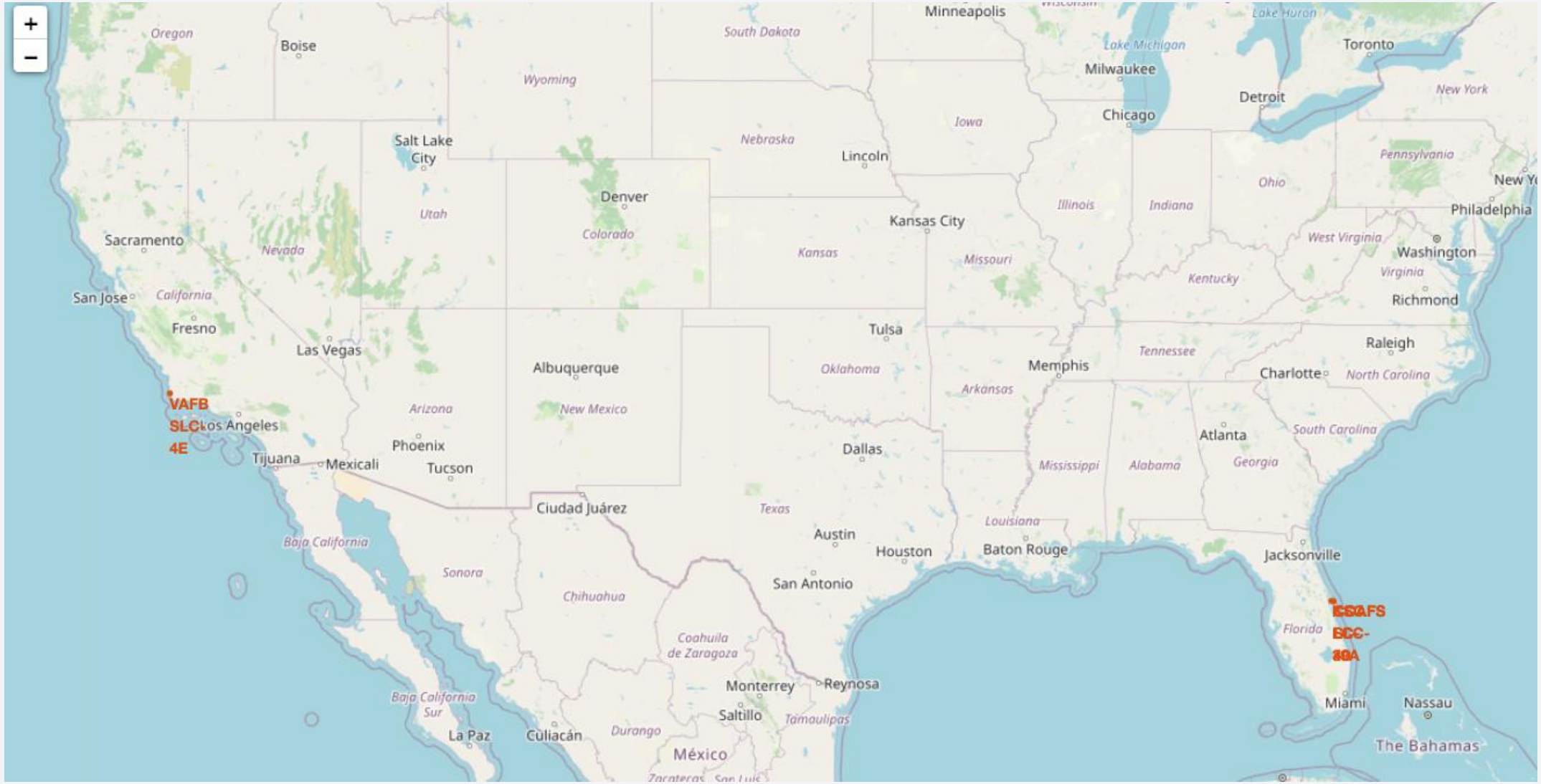
Out[54]:

Landing_Outcome	Number
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

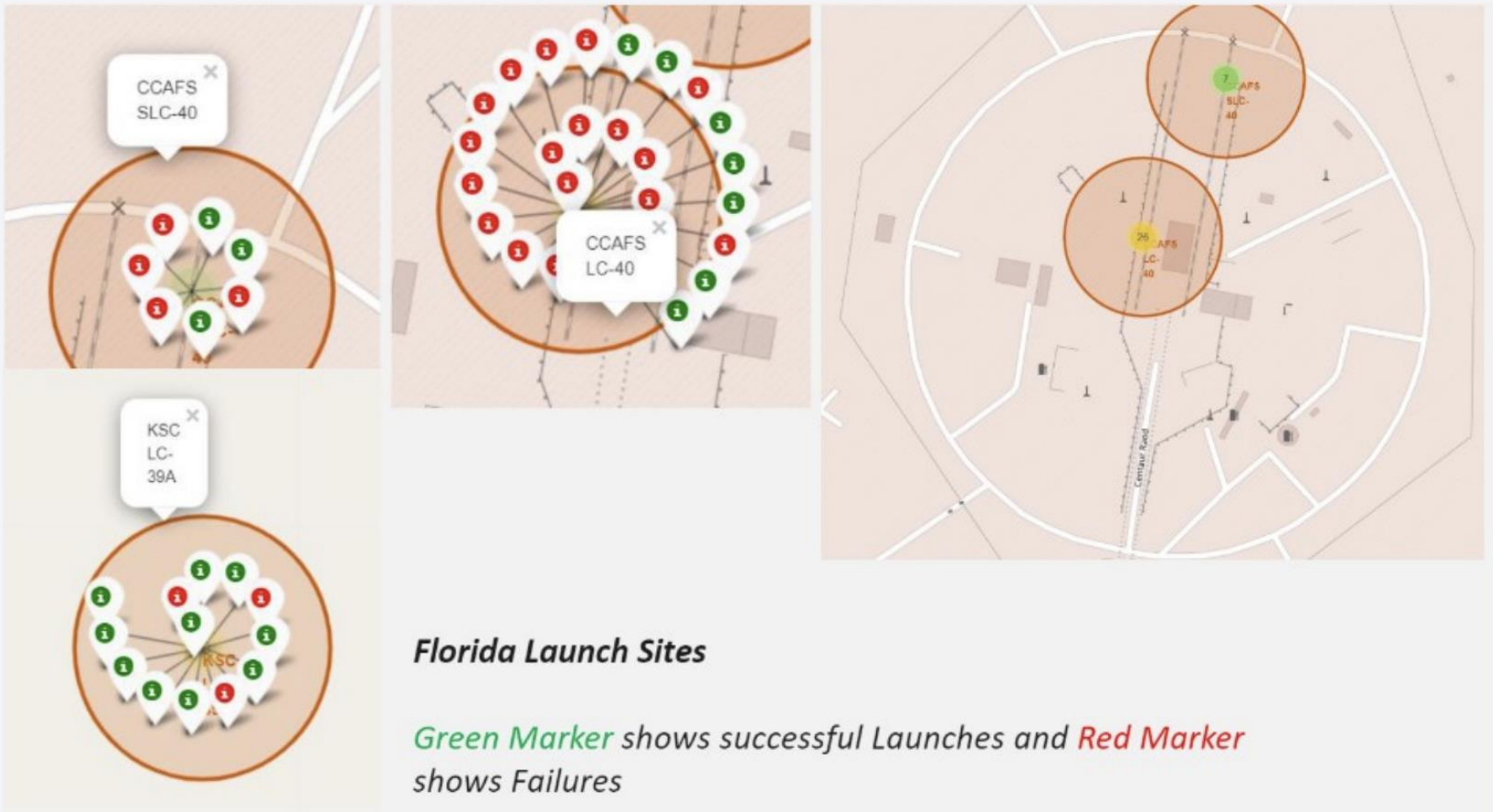
Section 3

Launch Sites Proximities Analysis

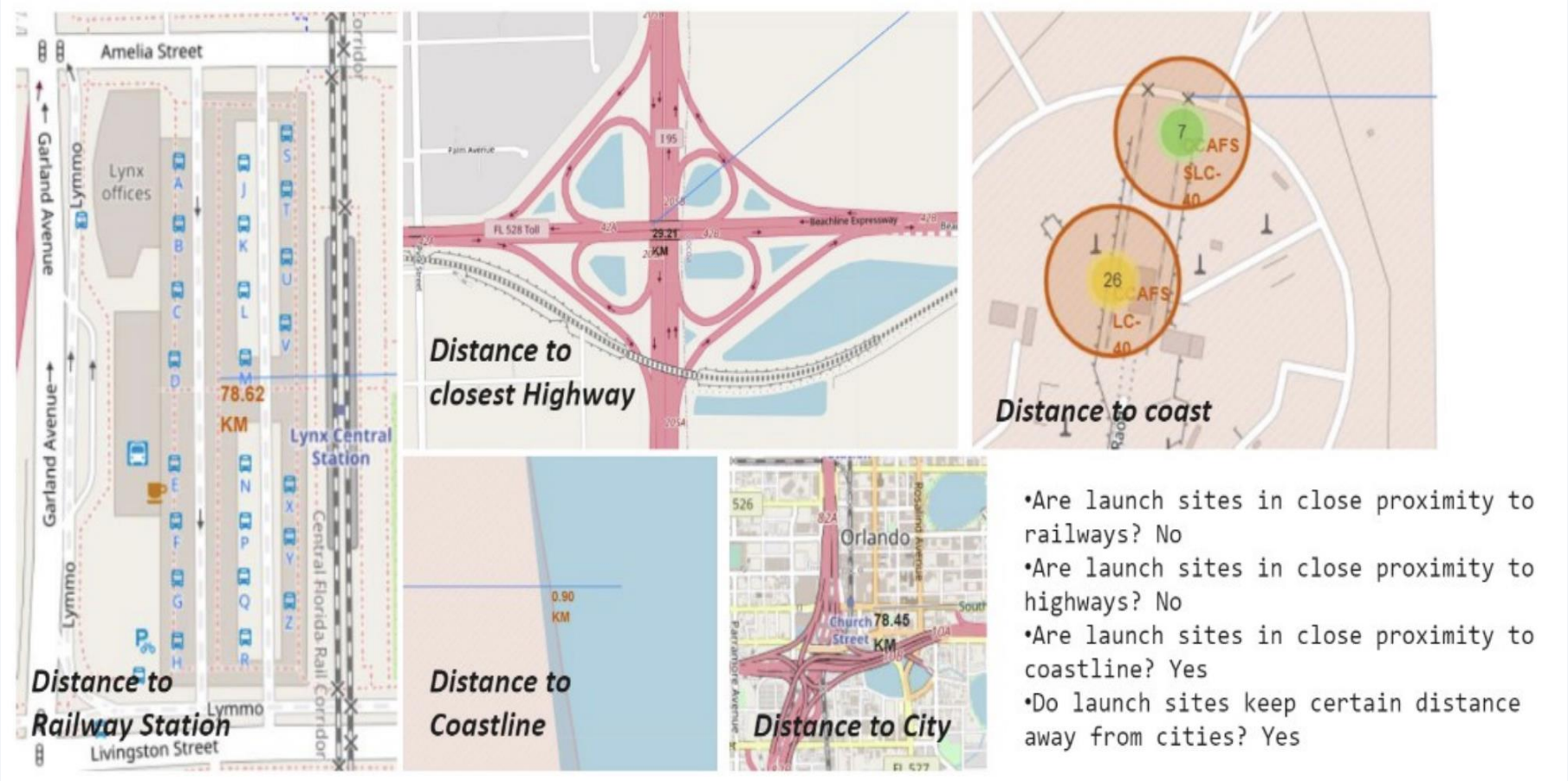
<Folium Map Screenshot 1>

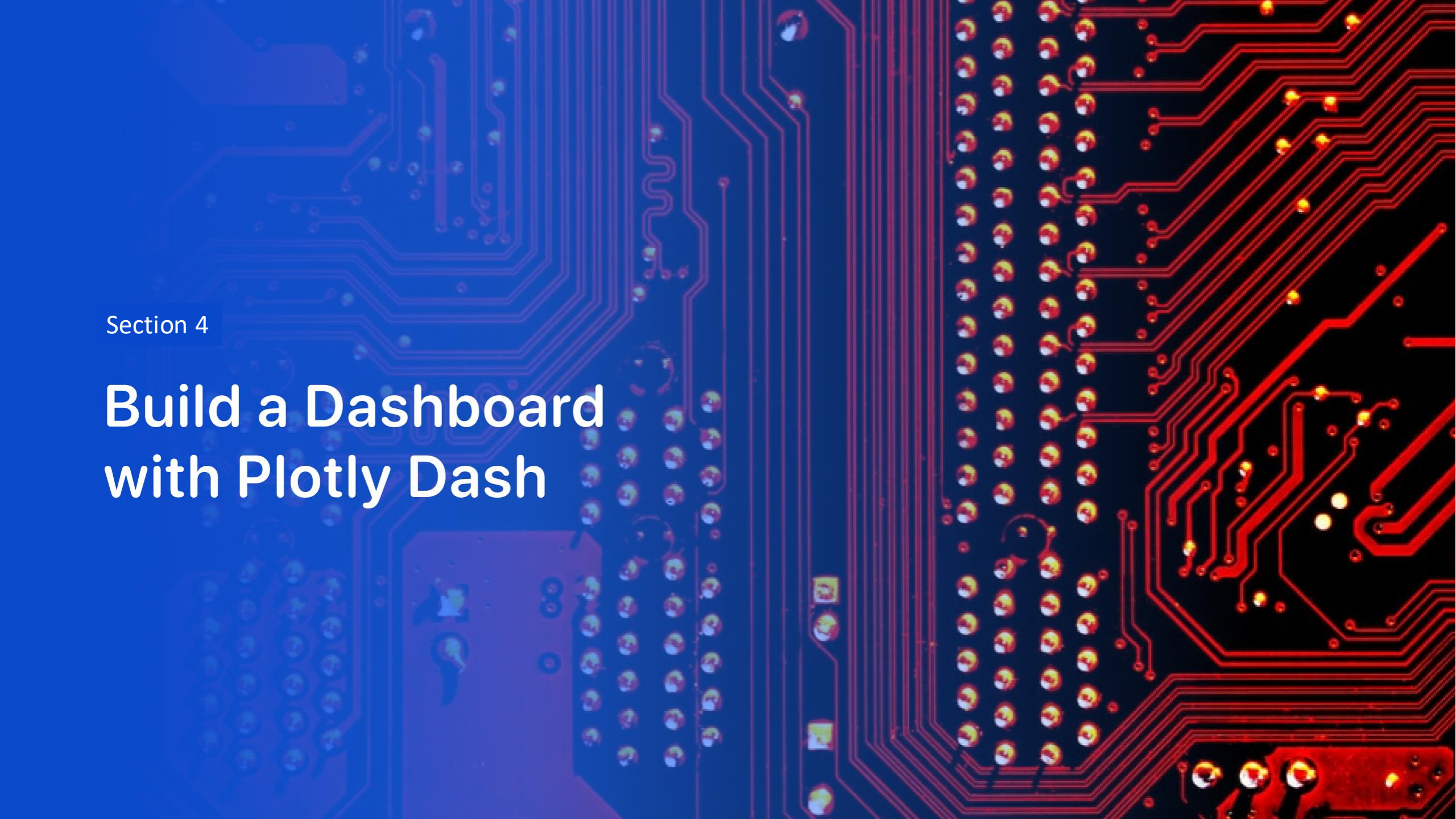


<Folium Map Screenshot 2>

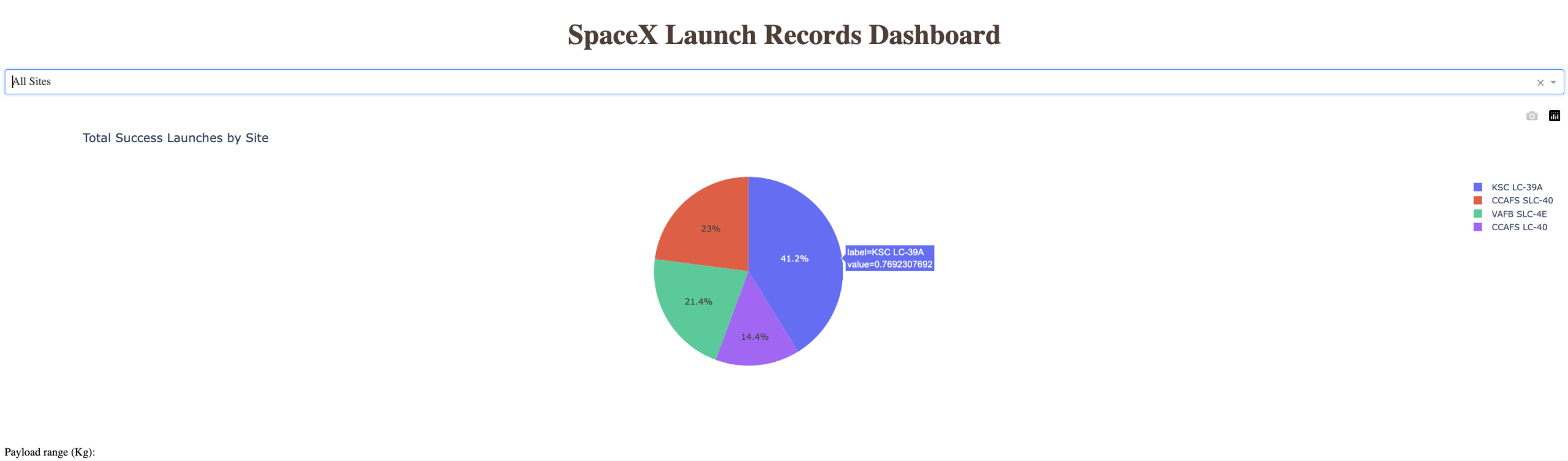


<Folium Map Screenshot 3>

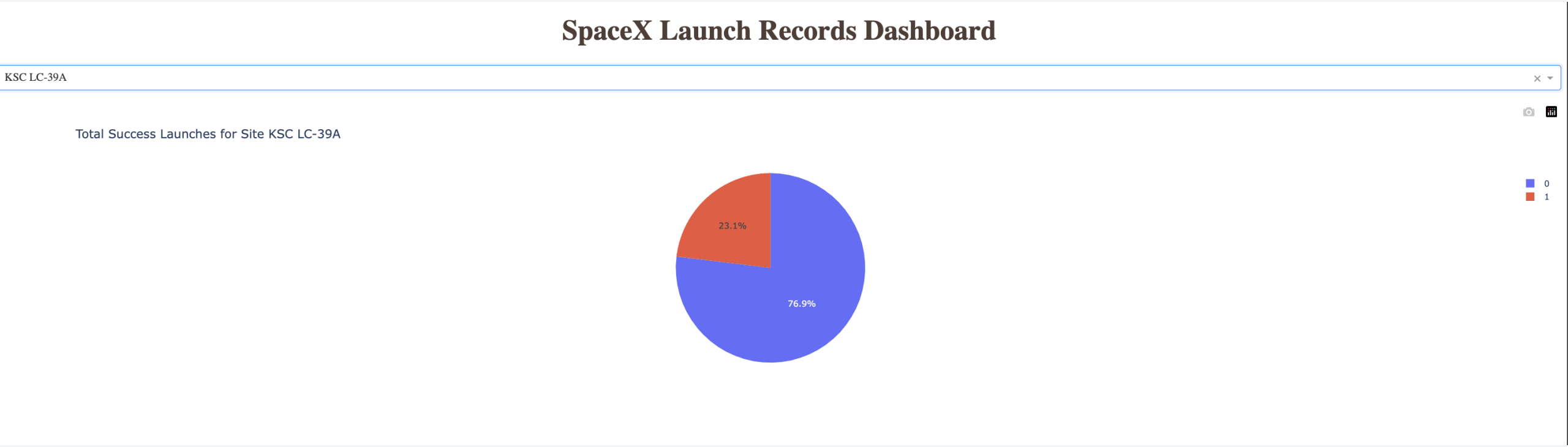



Section 4
Build a Dashboard with Plotly Dash

<Dashboard Screenshot 1>

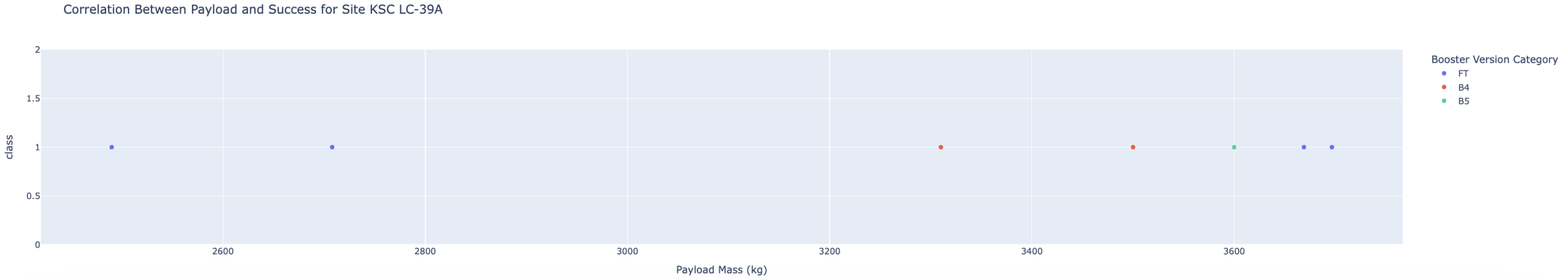


<Dashboard Screenshot 2>

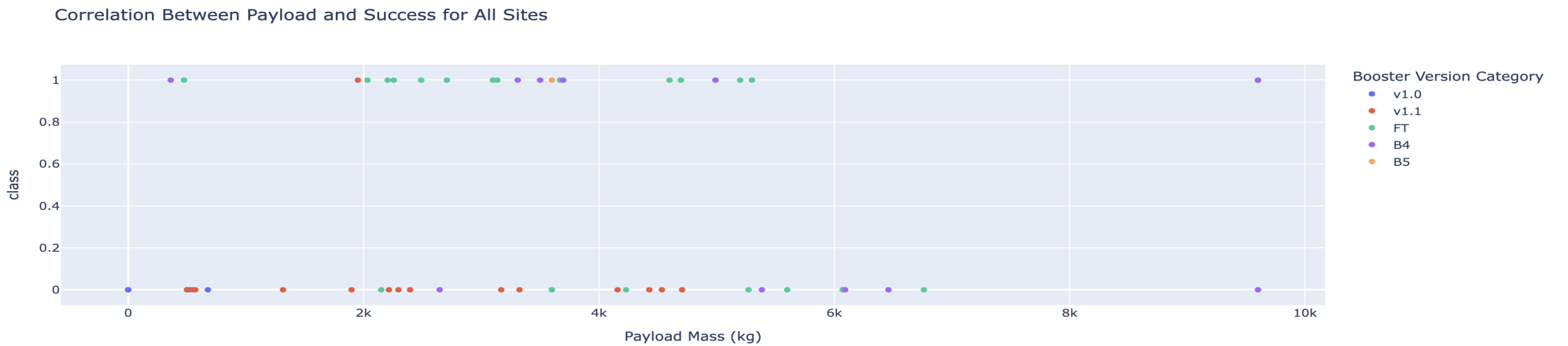


<Dashboard Screenshot 3>

Payload range (Kg):



Payload range (Kg):



Section 5

Predictive Analysis (Classification)

Classification Accuracy

- Logistic Regression had the best accuracy for test set

TASK 12

Find the method performs best:

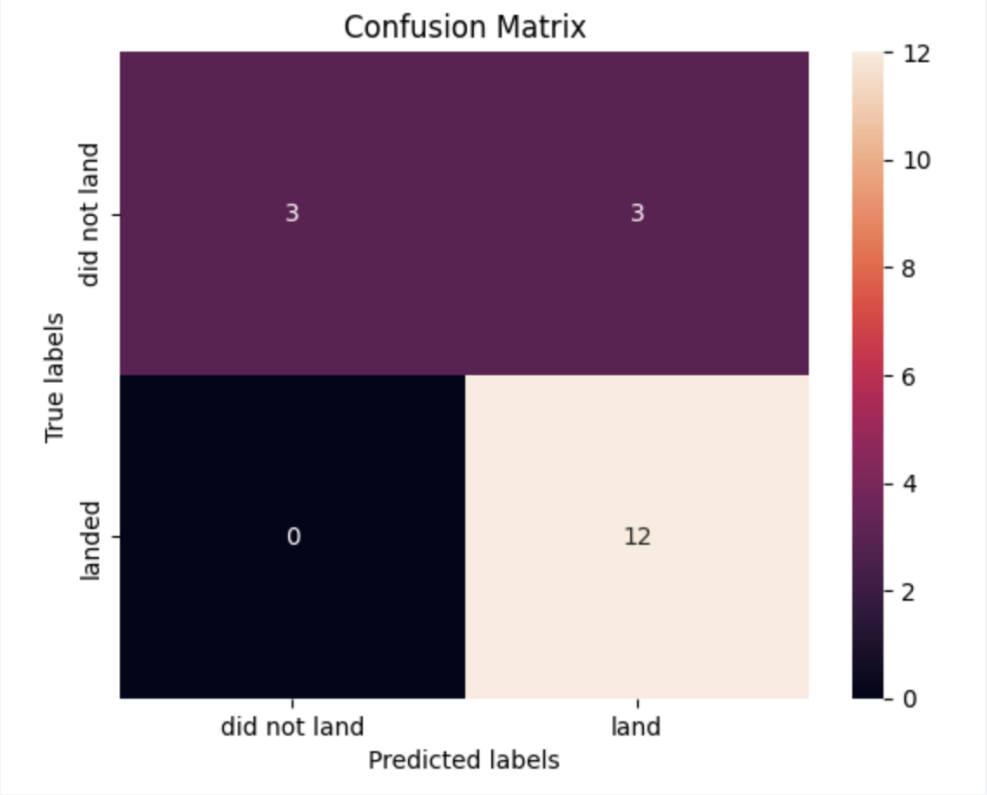
[50]:

```
a = {'accuracy_lr': accuracy, 'accuracy_knn': accuracy_knn, 'accuracy_svc': accuracy_svc, 'accuracy_tree': accura

# find the key corresponding to the maximum value
best_method = max(a, key=a.get)

# print the best accuracy and the corresponding method
print(a[best_method])
print(best_method)
```

```
0.8333333333333334
accuracy_lr
```

Confusion Matrix

- This logistic regression's confusion matrix.
- This confusion matrix shows a classification result where 12 instances of “landed” were correctly predicted, 3 instances of “did not land” were misclassified as “land,” and 3 instances of “did not land” were correctly classified. There are no misclassifications of “landed” as “did not land.”
-



Conclusions

- Factors Affecting Landings: Variables such as payload mass, launch site, and booster type played significant roles in determining the likelihood of successful landings. Heavier payloads or specific booster configurations showed lower landing success probabilities.
- The best model was logisticr regression with 83% accuracy
- Specific models with higher payload mass had higher success rate

Appendix

- Tested Random Forest ensemble method, but got similar results to Logistic Regression Model

Random Forest extra

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Define the parameter grid for RandomForestClassifier
parameters = {
    'n_estimators': [50, 100, 200],  # number of trees in the forest
    'criterion': ['gini', 'entropy'],  # function to measure the quality of a split
    'max_depth': [2*n for n in range(1, 10)],  # maximum depth of the tree
    'max_features': ['auto', 'sqrt'],  # number of features to consider for the best split
    'min_samples_leaf': [1, 2, 4],  # minimum number of samples required to be at a leaf node
    'min_samples_split': [2, 5, 10],  # minimum number of samples required to split an internal node
    'bootstrap': [True, False]  # whether bootstrap samples are used when building trees
}

# Create a RandomForestClassifier object
forest = RandomForestClassifier()

# Create a GridSearchCV object to tune the hyperparameters
forest_cv = GridSearchCV(estimator=forest, param_grid=parameters, cv=10)

# Fit the object to find the best parameters
forest_cv.fit(X_train, Y_train)

# Output the best parameters found by GridSearchCV
print(f"Best parameters: {forest_cv.best_params_}")

# Evaluate on test data
print(f"Best RandomForest accuracy on test set: {forest_cv.score(X_test, Y_test)}")
```

```
Best parameters: {'bootstrap': True, 'criterion': 'entropy', 'max_depth': 6, 'max_features': 'sqrt', 'min_samples_
leaf': 1, 'min_samples_split': 2, 'n_estimators': 50}
Best RandomForest accuracy on test set: 0.8333333333333334
```

Thank you!