IBM Developer
SKILLS NETWORK

# Winning Space Race with Data Science

Dhanapal Nagarajan
March 15, 2025

# Outline

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

# Executive Summary

- Summary of methodologies
  - Data collection using a SpaceX API
  - Data Collection with Web Scraping
  - Data Wrangling
  - Exploratory Data Analysis using SQL
  - EDA DataViz Using Python Pandas and Matplotlib
  - Launch Sites Analysis with Folium-Interactive Visual Analytics and Ploty Dash
  - Machine Learning Landing Prediction
- Summary of all results
  - EDA results
  - Interactive Visual Analytics and Dashboards
  - Predictive Analysis(Classification)

# Introduction

- Project background and context

  SpaceX promotes Falcon 9 rocket launches on its website at a cost of $62 million, while other providers charge upwards of $165 million per launch. The significant savings are largely due to SpaceX's ability to reuse the first stage. Thus, by predicting whether the first stage will successfully land, we can estimate the cost of a launch. This information is valuable for any company looking to compete with SpaceX for rocket launch contracts.

- Problems you want to find answers

  In this capstone project, we will predict the successful landing of the Falcon 9 first stage using data from Falcon 9 rocket launches as advertised on SpaceX's website.

Section 1

# Methodology

# Methodology

## Executive Summary

- Data collection methodology:
  - Describe how data was collected
- Perform data wrangling
  - Describe how data was processed
- 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

- Description of how SpaceX Falcon9 data was collected:

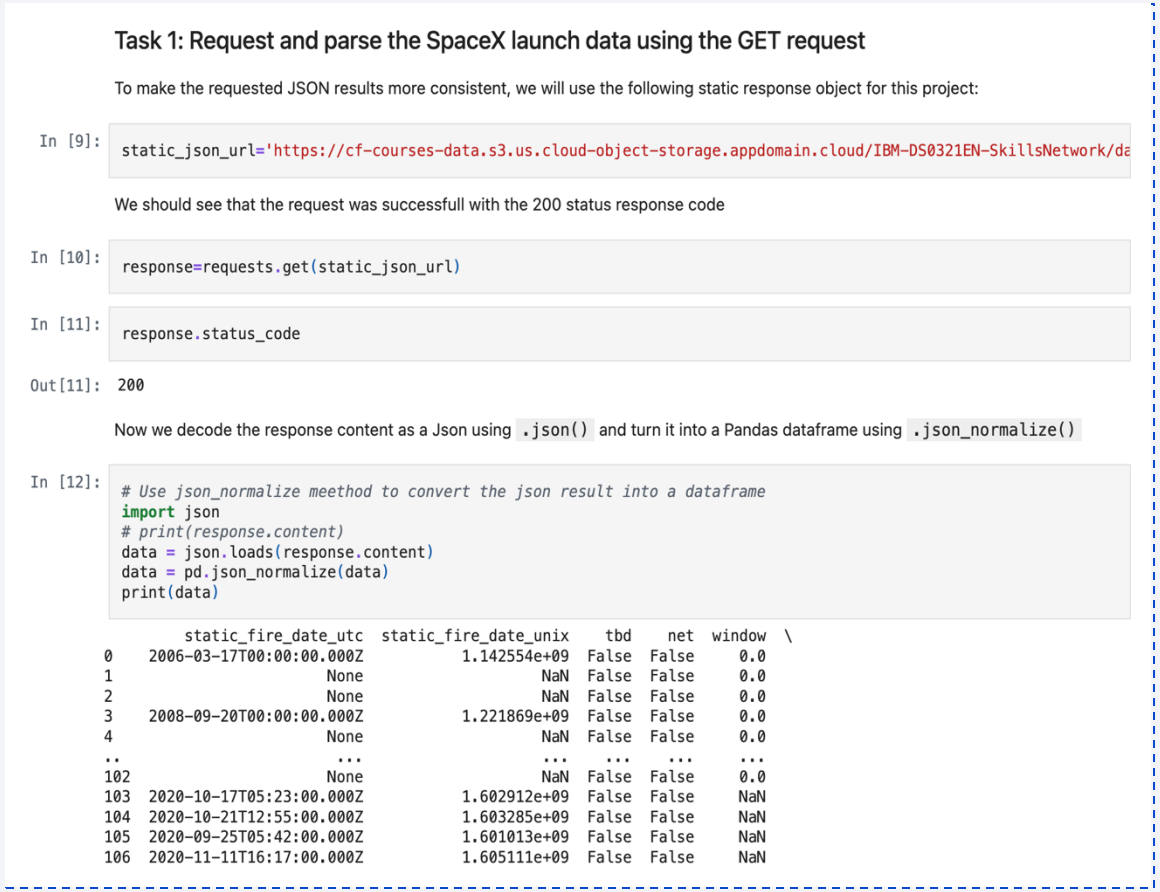
  - Data Collection from SpaceX API: We collected data by sending a request to the SpaceX API. This involved creating helper functions to make the process easier and to extract launch information using identification numbers.

  - Data Conversion: We used these functions to get the data and convert it into a format that was easy to work with. This involved converting the data into a Pandas data frame for analysis.

  - Web Scraping from Wikipedia: We also used web scraping to collect historical launch records of the Falcon 9 from Wikipedia. We extracted the data from tables on the page and turned it into a format that was easy to analyze.

# Data Collection – SpaceX API

- Data was collected using the SpaceX API (a RESTful API) by making a GET request to the SpaceX API. The SpaceX launch data was then requested and parsed using a GET request, and the response content was decoded as a JSON result, which was subsequently converted into a Pandas DataFrame
- The GitHub URL of the completed SpaceX API calls notebook
- URL: https://github.com/DhanapalInv/SpaceX-Falcon-9-1st-stage-Success-Landing-Prediction/blob/main/1.%20SpaceX-data-collection-api.ipynb



Task 1: Request and parse the SpaceX launch data using the GET request

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

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

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

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

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

Out[11]: 200

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

In [12]:
```
# Use json_normalize meethod to convert the json result into a dataframe
import json
# print(response.content)
data = json.loads(response.content)
data = pd.json_normalize(data)
print(data)
```

| | static_fire_date_utc | static_fire_date_unix | tbd | net | window | \ |
|---|---|---|---|---|---|---|
| 0 | 2006-03-17T00:00:00.000Z | 1.142554e+09 | False | False | 0.0 | |
| 1 | None | NaN | False | False | 0.0 | |
| 2 | None | NaN | False | False | 0.0 | |
| 3 | 2008-09-20T00:00:00.000Z | 1.221869e+09 | False | False | 0.0 | |
| 4 | None | NaN | False | False | 0.0 | |
| .. | ... | ... | ... | ... | ... | ... |
| 102 | None | NaN | False | False | 0.0 | |
| 103 | 2020-10-17T05:23:00.000Z | 1.602912e+09 | False | False | NaN | |
| 104 | 2020-10-21T12:55:00.000Z | 1.603285e+09 | False | False | NaN | |
| 105 | 2020-09-25T05:42:00.000Z | 1.601013e+09 | False | False | NaN | |
| 106 | 2020-11-11T16:17:00.000Z | 1.605111e+09 | False | False | NaN | |

# Data Collection - Scraping

- We used web scraping to collect historical launch records of the Falcon 9 from Wikipedia. We used tools called BeautifulSoup and request to extract the records from tables on the Wikipedia page and then turned them into a data frame.
- The GitHub URL of the completed web scraping notebook below.
- URL: https://github.com/Dhanapalnv/SpaceX-Falcon-9-1st-stage-Success-Landing-Prediction/blob/main/2.%20SpaceX-webscraping.ipynb

**Task 1: Request and parse the SpaceX launch data using the GET request**

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

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

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

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

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

Out[11]: 200

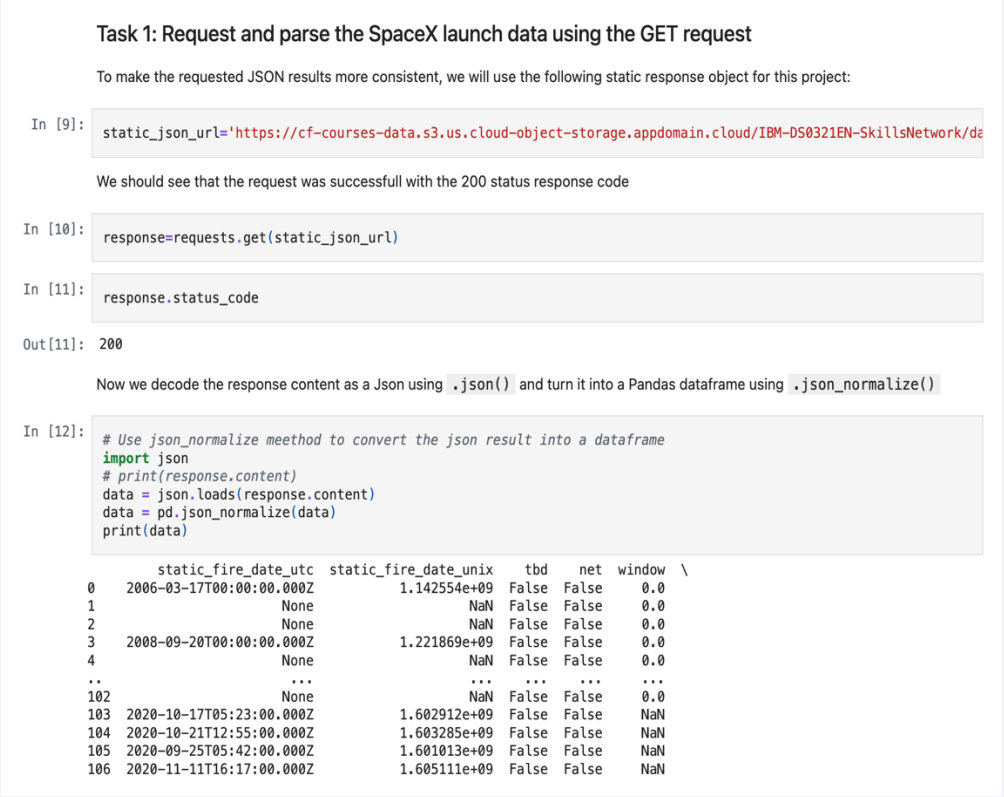
Now we decode the response content as a Json using `.json()` and turn it into a Pandas dataframe using `.json_normalize()`

In [12]:
```
# Use json_normalize meethod to convert the json result into a dataframe
import json
# print(response.content)
data = json.loads(response.content)
data = pd.json_normalize(data)
print(data)
```

| | static_fire_date_utc | static_fire_date_unix | tbd | net | window | \ |
|---|---|---|---|---|---|---|
| 0 | 2006-03-17T00:00:00.000Z | 1.142554e+09 | False | False | 0.0 | |
| 1 | None | NaN | False | False | 0.0 | |
| 2 | None | NaN | False | False | 0.0 | |
| 3 | 2008-09-20T00:00:00.000Z | 1.221869e+09 | False | False | 0.0 | |
| 4 | None | NaN | False | False | 0.0 | |
| .. | ... | ... | ... | ... | ... | ... |
| 102 | None | NaN | False | False | 0.0 | |
| 103 | 2020-10-17T05:23:00.000Z | 1.602912e+09 | False | False | NaN | |
| 104 | 2020-10-21T12:55:00.000Z | 1.603285e+09 | False | False | NaN | |
| 105 | 2020-09-25T05:42:00.000Z | 1.601013e+09 | False | False | NaN | |
| 106 | 2020-11-11T16:17:00.000Z | 1.605111e+09 | False | False | NaN | |

# Data Wrangling

- Data Preparation: After organizing the data into a table, we filtered it to only include Falcon 9 launches. We then fixed missing information in the landing pad and payload mass columns by filling in the missing payload mass with the average value.
- Data Analysis: We also looked closely at the data to find patterns and decide what to focus on when training models. This step helped us understand the data better and prepare it for further analysis.
- The GitHub URL of the completed data wrangling notebooks.
- URL: https://github.com/DhanapalInv/SpaceX-Falcon-9-1st-stage-Success-Landing-Prediction/blob/main/3.%20SpaceX-Data_wrangling.ipynb



Task 1: Request and parse the SpaceX launch data using the GET request

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

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

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

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

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

Out[11]: 200

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

In [12]:
```
# Use json_normalize meethod to convert the json result into a dataframe
import json
# print(response.content)
data = json.loads(response.content)
data = pd.json_normalize(data)
print(data)
```

```
        static_fire_date_utc  static_fire_date_unix    tbd    net  window  \
0    2006-03-17T00:00:00.000Z           1.142554e+09  False  False     0.0
1                        None                    NaN  False  False     0.0
2                        None                    NaN  False  False     0.0
3    2008-09-20T00:00:00.000Z           1.221869e+09  False  False     0.0
4                        None                    NaN  False  False     0.0
..                        ...                    ...    ...    ...     ...
102                      None                    NaN  False  False     0.0
103  2020-10-17T05:23:00.000Z           1.602912e+09  False  False     NaN
104  2020-10-21T12:55:00.000Z           1.603285e+09  False  False     NaN
105  2020-09-25T05:42:00.000Z           1.601013e+09  False  False     NaN
106  2020-11-11T16:17:00.000Z           1.605111e+09  False  False     NaN
```

# EDA with Data Visualization

- Data Analysis & Feature Engineering: Performed data analysis and feature engineering using Pandas and Matplotlib.
- Key Activities:
  - Exploratory Data Analysis: Conducted exploratory data analysis.
  - Preparing Data & Feature Engineering: Prepared the data and engineered its features.
- Visualizations:
  - Scatter Plots: Used scatter plots to visualize the relationships between:
    - Flight Number and Launch Site.
    - Payload and Launch Site.
    - Flight Number and Orbit Type.
    - Payload and Orbit Type.
  - Bar Chart: Used a bar chart to visualize the success rate of each orbit type.
  - Line Plot: Used a line plot to visualize the yearly trend of launch success.
- Add the GitHub URL of your completed EDA with SQL notebook: https://github.com/Dhanapalnv/SpaceX-Falcon-9-1st-stage-Success-Landing-Prediction/blob/main/4.%20SpaceX-EDA_DataViz_Using_Pandas_and_Matplotlib.ipynb

# EDA with SQL

- The following SQL queries were performed for EDA
  - Display the names of the unique launch sites in the space mission
  - Display 5 records where launch sites begin with the string 'CCA'
  - Display the total payload mass carried by boosters launched by NASA (CRS)
  - Display average payload mass carried by booster version F9 v1.1
  - List the date when the first succesful landing outcome in ground pad was acheived.
  - List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000
  - List the total number of successful and failure mission outcomes
  - List the names of the booster_versions which have carried the maximum payload mass. Use a subquery
  - 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.
  - 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.
  - the GitHub URL of your completed EDA with SQL notebook : https://github.com/DhanapalInv/SpaceX-Falcon-9-1st-stage-Success-Landing-Prediction/blob/main/5.%20SpaceX-EDA-sql_sqllite.ipynb

# Build an Interactive Map with Folium

- Launch Site Mapping: A Folium map was created to mark all the launch sites. Map objects such as markers, circles, and lines were added to visually indicate the success or failure of launches at each site, providing a clear overview of launch performance.
- Launch Outcome Classification: A set of launch outcomes was created, where failures were coded as 0 and successes as 1. This classification system allowed for easy analysis and comparison of launch results across different sites.
- The GitHub URL of your completed interactive map with Folium map : https://github.com/DhanapalInv/SpaceX-Falcon-9-1st-stage-Success-Landing-Prediction/blob/main/6.%20SpaceX-EDA_Launch_Sites_Locations_Analysis_with_Folium.ipynb

In [5]:
```
## Task 1: Mark all launch sites on a map
```

First, let's try to add each site's location on a map using site's latitude and longitude coordinates

The following dataset with the name `spacex_launch_geo.csv` is an augmented dataset with latitude and longitude added for each site.

In [6]:
```
# Download and read the `spacex_launch_geo.csv`
from js import fetch
import io

URL = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/spa
resp = await fetch(URL)
spacex_csv_file = io.BytesIO((await resp.arrayBuffer()).to_py())
spacex_df=pd.read_csv(spacex_csv_file)
```

Now, you can take a look at what are the coordinates for each site.

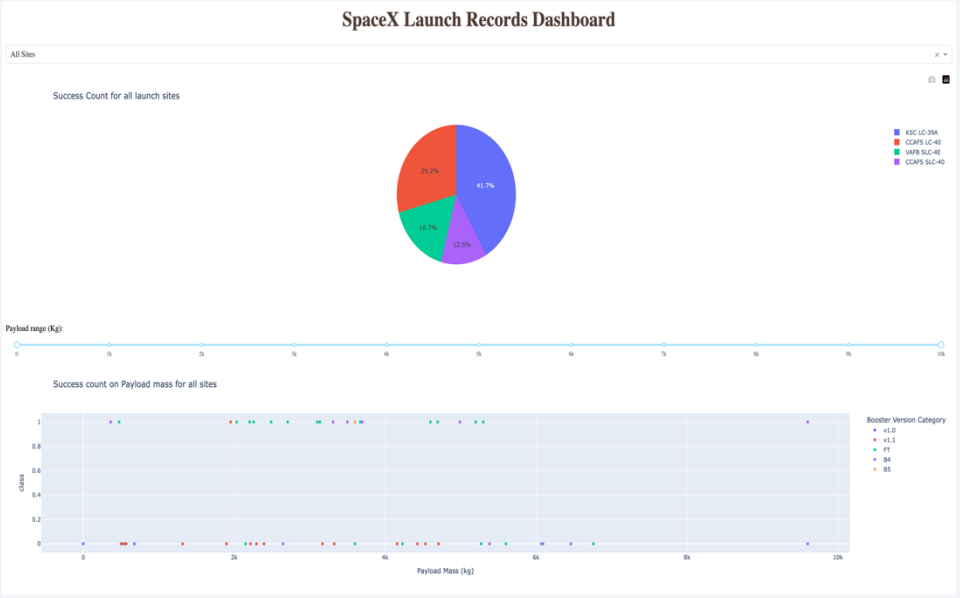
In [7]:
```
# Select relevant sub-columns: `Launch Site`, `Lat(Latitude)`, `Long(Longitude)`, `class`
spacex_df = spacex_df[['Launch Site', 'Lat', 'Long', 'class']]
launch_sites_df = spacex_df.groupby(['Launch Site'], as_index=False).first()
launch_sites_df = launch_sites_df[['Launch Site', 'Lat', 'Long']]
launch_sites_df
```

Out[7]:

| | Launch Site | Lat | Long |
|---|---|---|---|
| 0 | CCAFS LC-40 | 28.562302 | -80.577356 |
| 1 | CCAFS SLC-40 | 28.563197 | -80.576820 |
| 2 | KSC LC-39A | 28.573255 | -80.646895 |
| 3 | VAFB SLC-4E | 34.632834 | -120.610745 |

# Build a Dashboard with Plotly Dash

- Interactive Dashboard Application: An interactive dashboard application was built using Plotly Dash. This involved:
  - Adding a Launch Site Drop-down Input Component.
  - Implementing a callback function to render a success pie chart based on the selected site from the dropdown.
  - Adding a Range Slider to select the payload.
  - Implementing a callback function to render a success-payload scatter plot.
- The GitHub URL of your completed Plotly Dash lab : https://github.com/Dhanapalnv/SpaceX-Falcon-9-1st-stage-Success-Landing-Prediction/blob/main/7.%20SpaceX_Dashboard_with_Ploty_Dash_app.py



# Predictive Analysis (Classification)

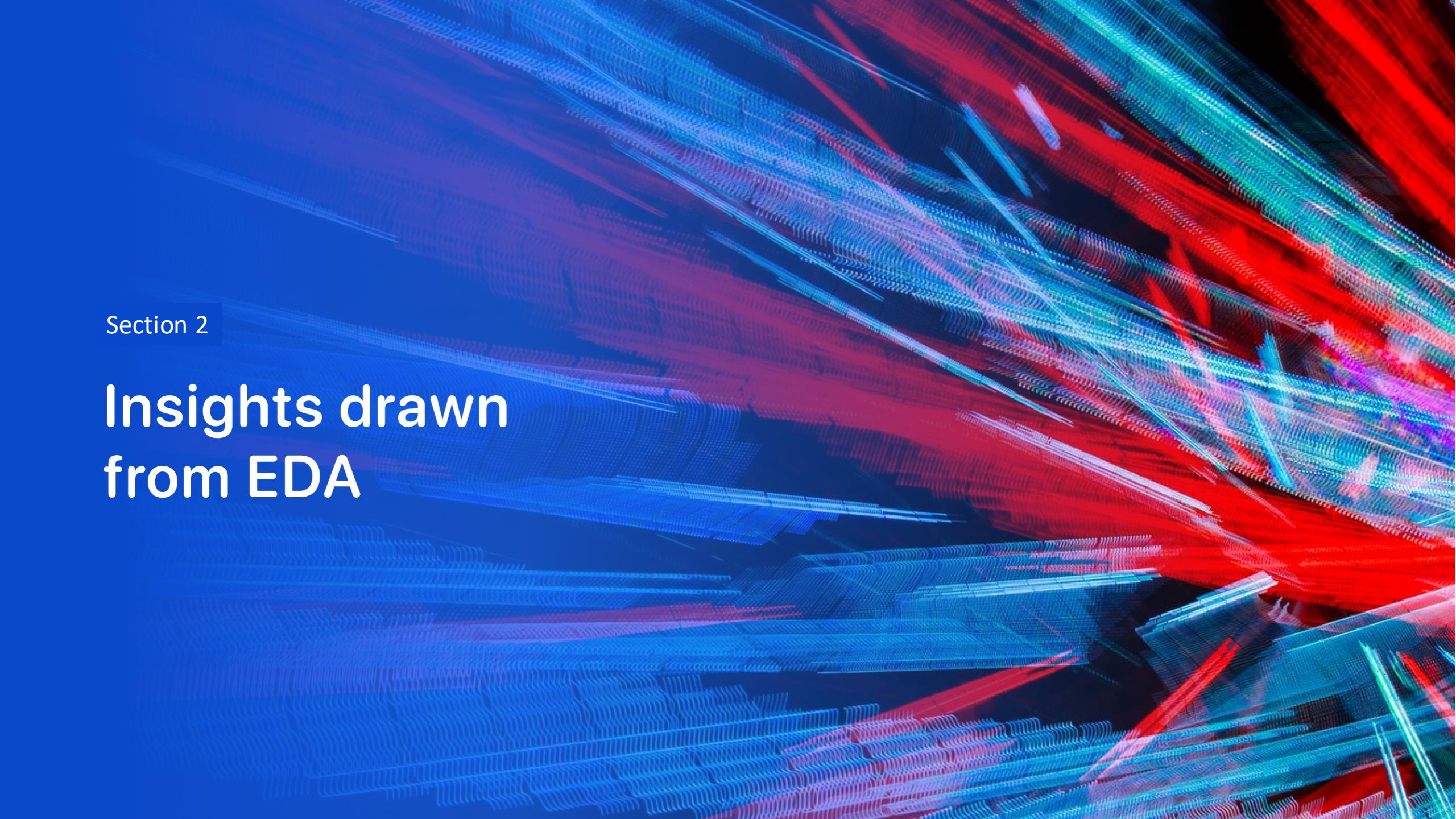
- Building and Evaluating the Best Classification Model
- Data Preparation:
  - Loaded data into a Pandas DataFrame.
  - Performed exploratory data analysis.
  - Created a NumPy array for the outcome variable.
  - Standardized the feature dataset using StandardScaler.
  - Split data into training and testing sets (80% for training, 20% for testing).
- Model Selection and Evaluation:
  - Created objects for SVM, Classification Trees, k-Nearest Neighbors, and Logistic Regression.
  - Used GridSearchCV to find the best hyperparameters for each model.
  - Evaluated each model's performance using validation data and identified the best parameters.
  - Calculated test accuracy for each model and plotted confusion matrices.
  - Compared test accuracy scores to determine the best-performing model.
  - URL: https://github.com/DhanapalInv/SpaceX-Falcon-9-1st-stage-Success-Landing-Prediction/blob/main/8.%20SpaceX_Machine_Learning_Prediction.ipynb

Result:

| | 0 |
|---|---|
| **Method** | Test Data Accuracy |
| **Logistic_Reg** | 0.833333 |
| **SVM** | 0.833333 |
| **Decision Tree** | 0.833333 |
| **KNN** | 0.833333 |

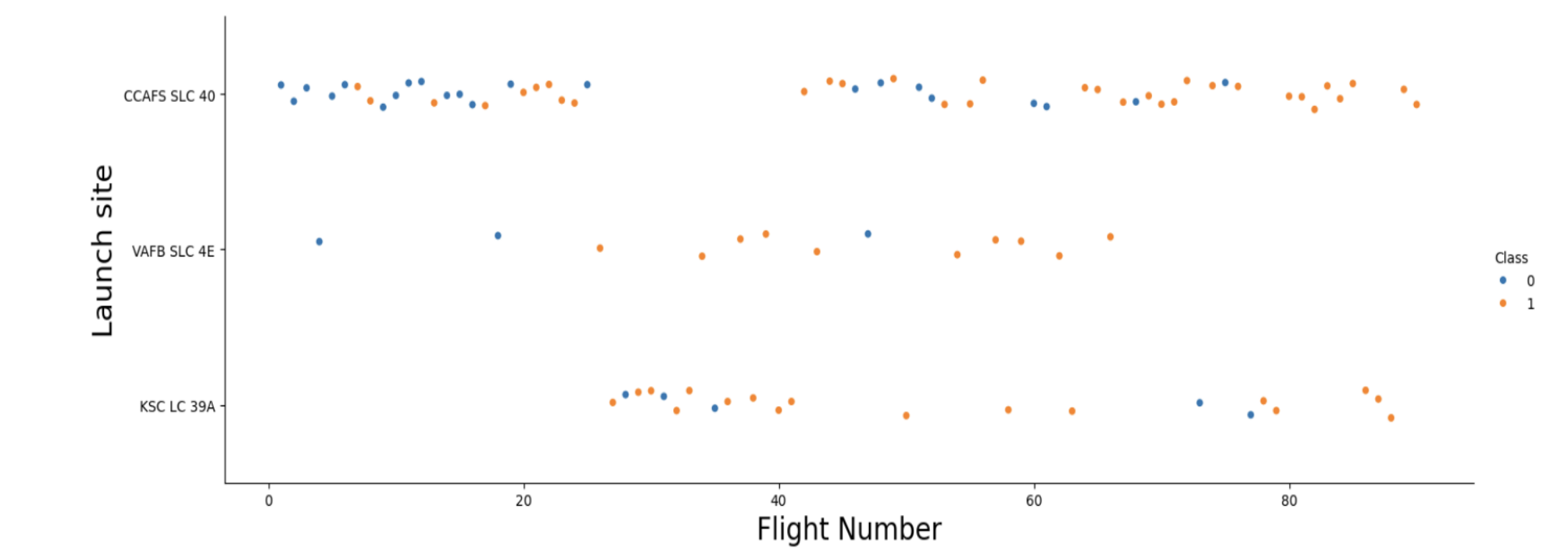
# Results

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


Section 2
Insights drawn from EDA

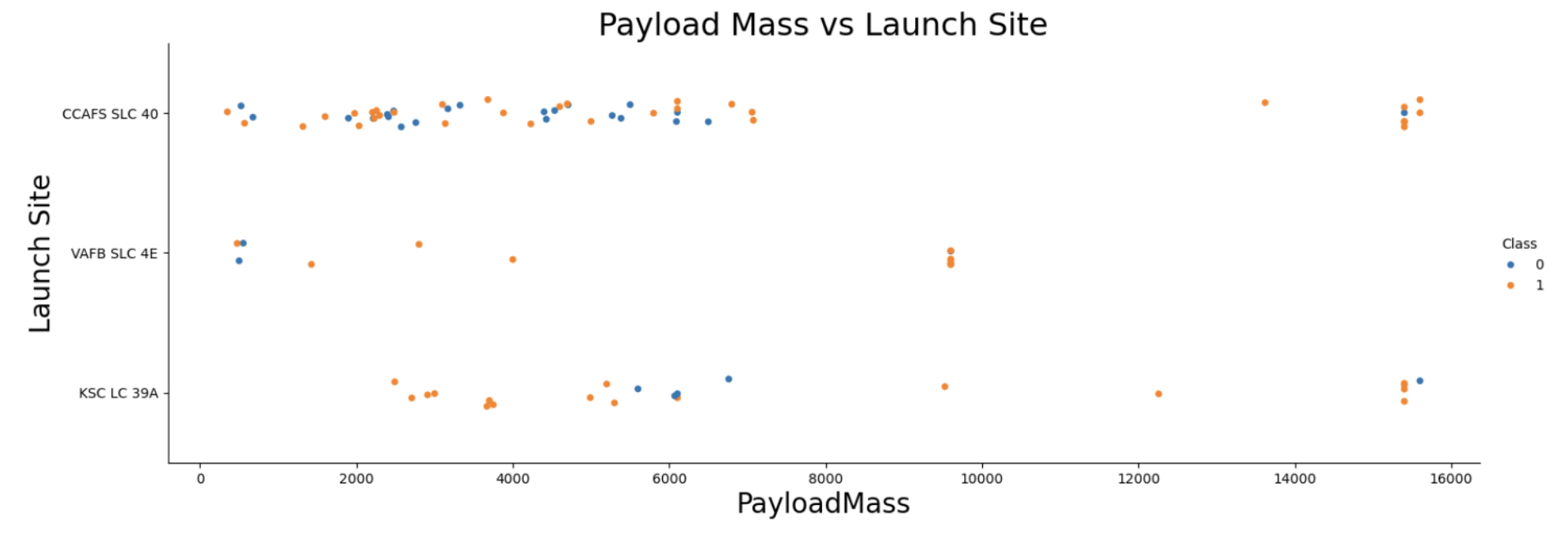
# Flight Number vs. Launch Site

- Scatter plot of Flight Number vs. Launch Site



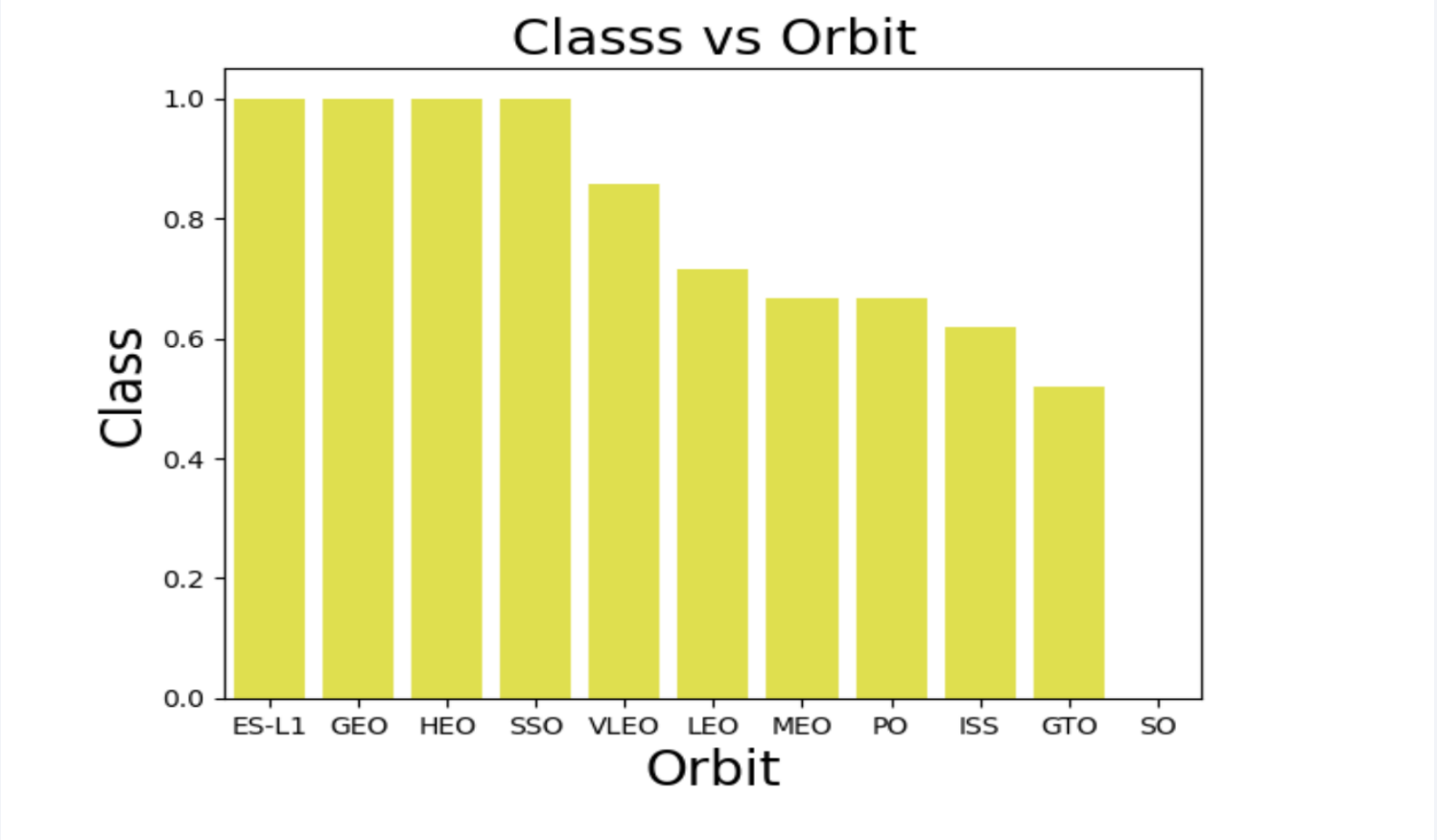
# Payload vs. Launch Site

- Scatter plot of Payload vs. Launch Site



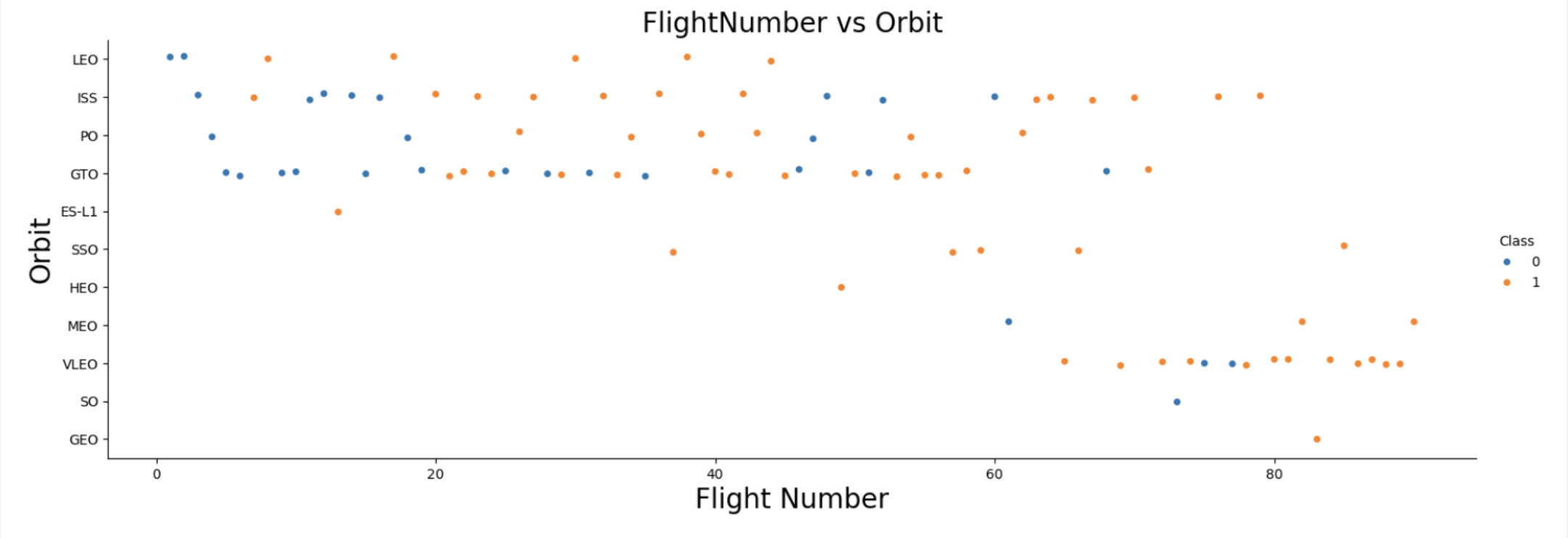
# Success Rate vs. Orbit Type

- Bar chart for the success rate of each orbit type



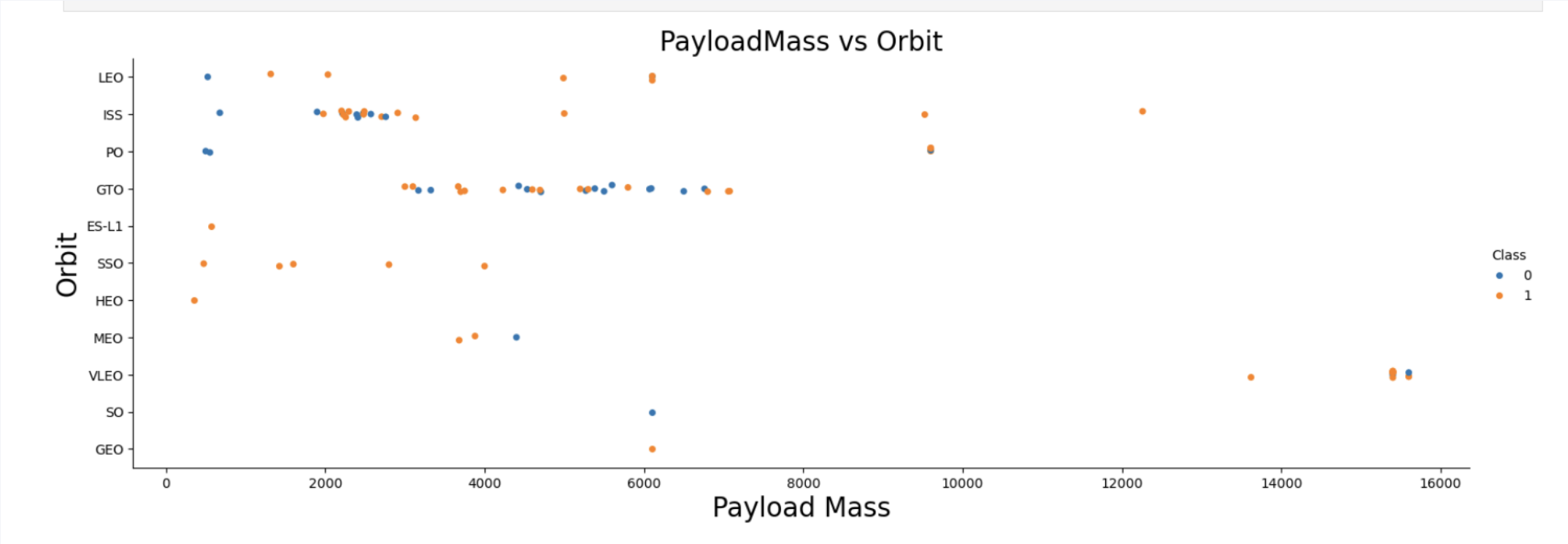
# Flight Number vs. Orbit Type

- Scatter point of Flight number vs. Orbit type



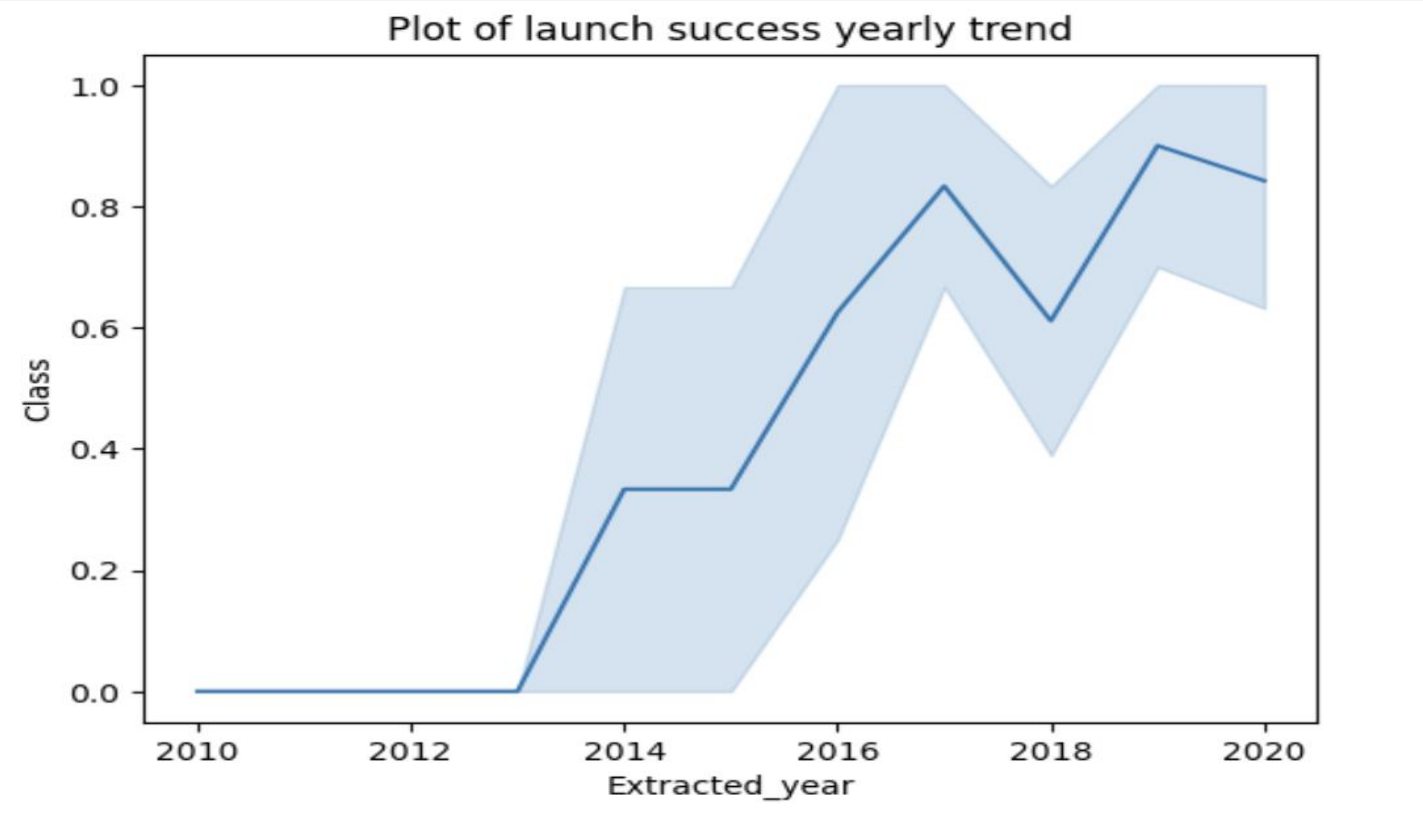
# Payload vs. Orbit Type

- Scatter point of payload vs. orbit type



# Launch Success Yearly Trend

- Line chart of yearly average success rate



# All Launch Site Names

- Names of the unique launch sites
- Run the *DISTINCT "Launch_Site"* which provide a unique launch sites

```
In [12]: %sql select DISTINCT "Launch_Site" from SPACEXTABLE
 * sqlite:///my_data1.db
Done.
```

Out[12]:

| 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`
- Used a like and wildcard operator along with Limit function

```
In [13]: %sql select * from SPACEXTABLE where Launch_Site like "CCA%" limit 5;
 * sqlite:///my_data1.db
Done.
```

Out[13]:

| 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

- Calculate the total payload carried by boosters from NASA
- To identify the total payload mass used SUM operator on the column PAYLOAD_MASS__KG_

### Task 3

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

In [14]:
```
%sql select sum(PAYLOAD_MASS__KG_) as "Total payload mass carried by boosters launched by NASA (CRS) in KG's"  from
```

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

Out[14]:

| Total payload mass carried by boosters launched by NASA (CRS) in KG's |
|---|
| 45596 |

# Average Payload Mass by F9 v1.1

- Calculate the average payload mass carried by booster version F9 v1.1
- Used AVG and WHERE with a wildcard operator.

### Task 4

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

In [15]:
```
s "Average payload mass carried by booster version F9 v1.1" from SPACEXTABLE where Booster_Version like "F9 v1.1%"
```

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

Out[15]:

| Average payload mass carried by booster version F9 v1.1 |
|---|
| 2534.6666666666665 |

### Task 5

# First Successful Ground Landing Date

- Find the dates of the first successful landing outcome on ground pad
- Used a MIN and Limit 1 to get a first successful landing outcome

```
In [16]: succesful landing outcome in ground pad" from SPACEXTABLE where "Landing_Outcome" = "Success (ground pad)" LIMIT 1
 * sqlite:///my_data1.db
Done.
```

Out[16]:

| first succesful landing outcome in 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

- Used a WHERE and AND parameter between and AND

```
In [19]: 'ersion from SPACEXTABLE where landing_Outcome = 'Success (drone ship)' and PAYLOAD_MASS__KG_ between 4000 and 6000

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

Out[19]:

| 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
- Used count * and groupby mission_outcome

```
In [20]: %sql select mission_outcome, count(*) as total_number from SPACEXTABLE group by mission_outcome;
 * sqlite:///my_data1.db
Done.
```

Out[20]:

| Mission_Outcome | total_number |
|---|---|
| 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
- Used Subquery on the Where to achieve this.

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

In [25]:
```
elect booster_version from SPACEXTABLE WHERE payload_mass__kg_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTABLE);
```

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

Out[25]:

| 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

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

- Used a substr(Date,0,5)='2015' in the where claus.

```
In [30]: ersion, Launch_Site  from SPACEXTABLE where (landing_Outcome = 'Failure (drone ship)') and substr(Date,0,5)='2015';
 * sqlite:///my_data1.db
Done.
```

Out[30]:

| month | Date | Landing_Outcome | Booster_Version | Launch_Site |
|---|---|---|---|---|
| 01 | 2015-01-10 | Failure (drone ship) | F9 v1.1 B1012 | CCAFS LC-40 |
| 04 | 2015-04-14 | 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.

In [45]:
```
uccess (ground pad)") AND Date between '2010-06-04' and '2017-03-20' group by Landing_Outcome order by Date DESC ;
```

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

Out[45]:

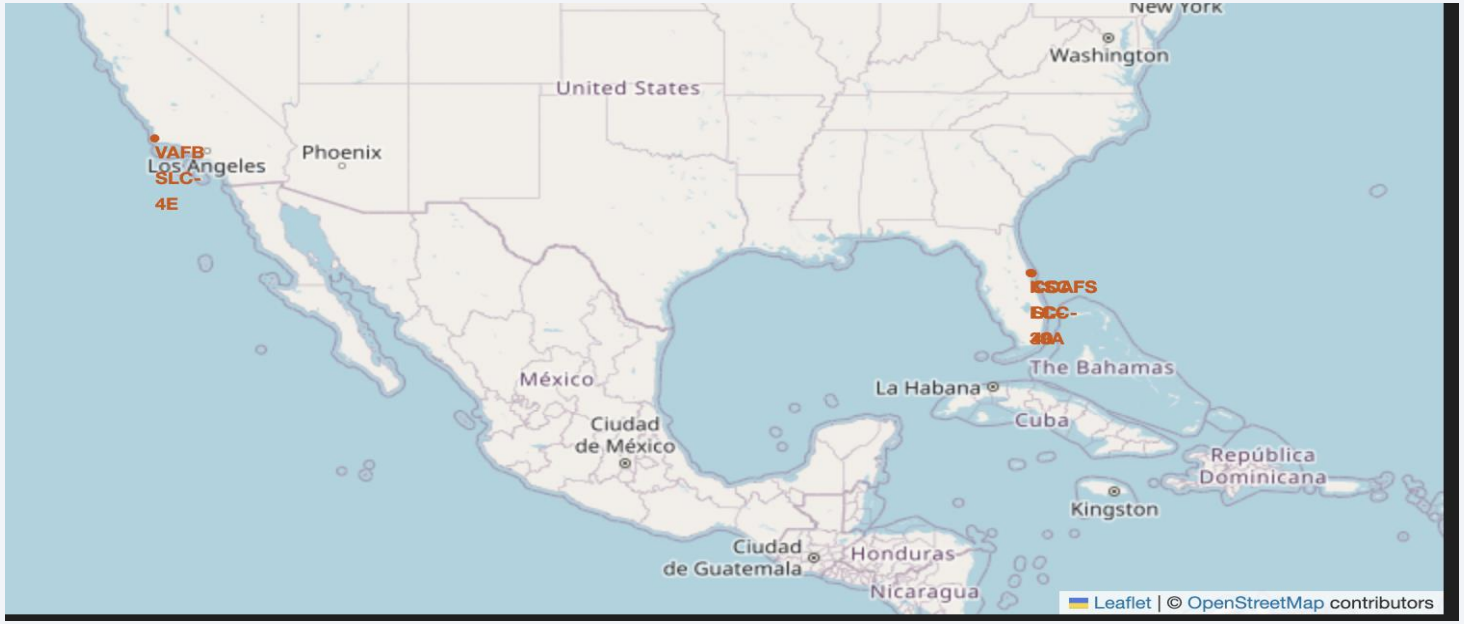
| Landing_Outcome | count (*) |
|---|---|
| Success (ground pad) | 3 |
| Failure (drone ship) | 5 |

Section 3

# Launch Sites Proximities Analysis

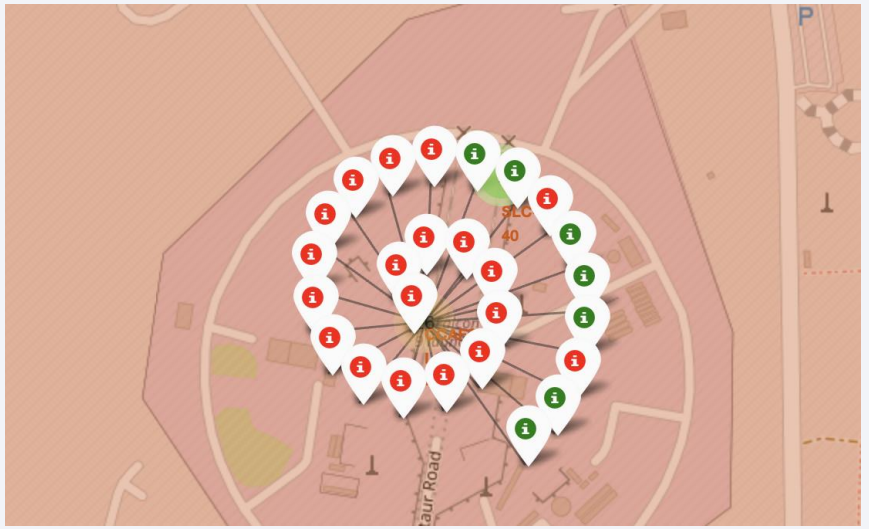
# All launch site

- Explore the generated folium map and make a proper screenshot to include all launch sites' location markers on a global map
- Used a folium.Map and location coordinated



# Outcome of each site and folium marker

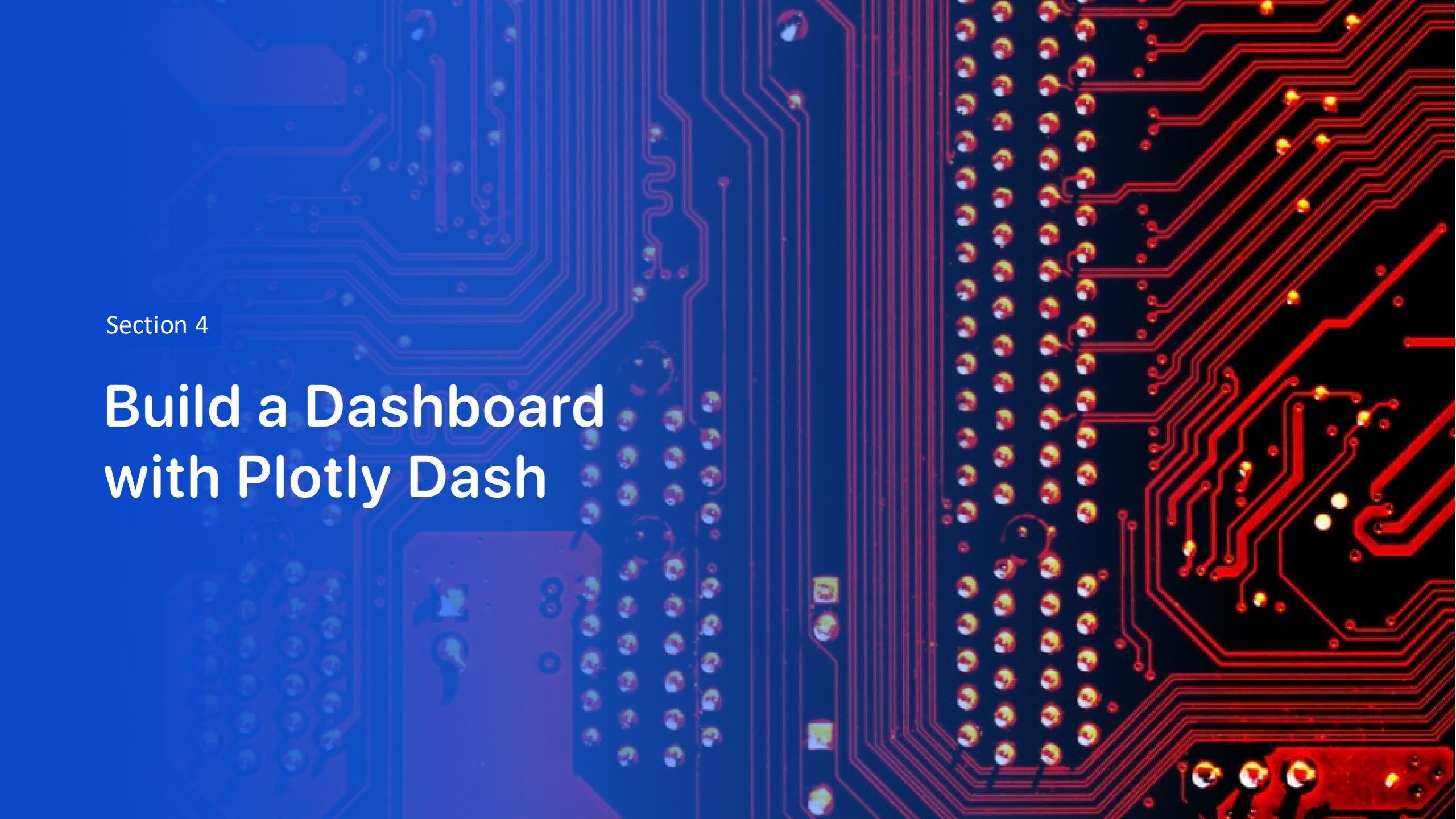
- Explore the folium map and make a proper screenshot to show the color-labeled launch outcomes on the map


- Used the folium.child operator to mark with launch site outcome

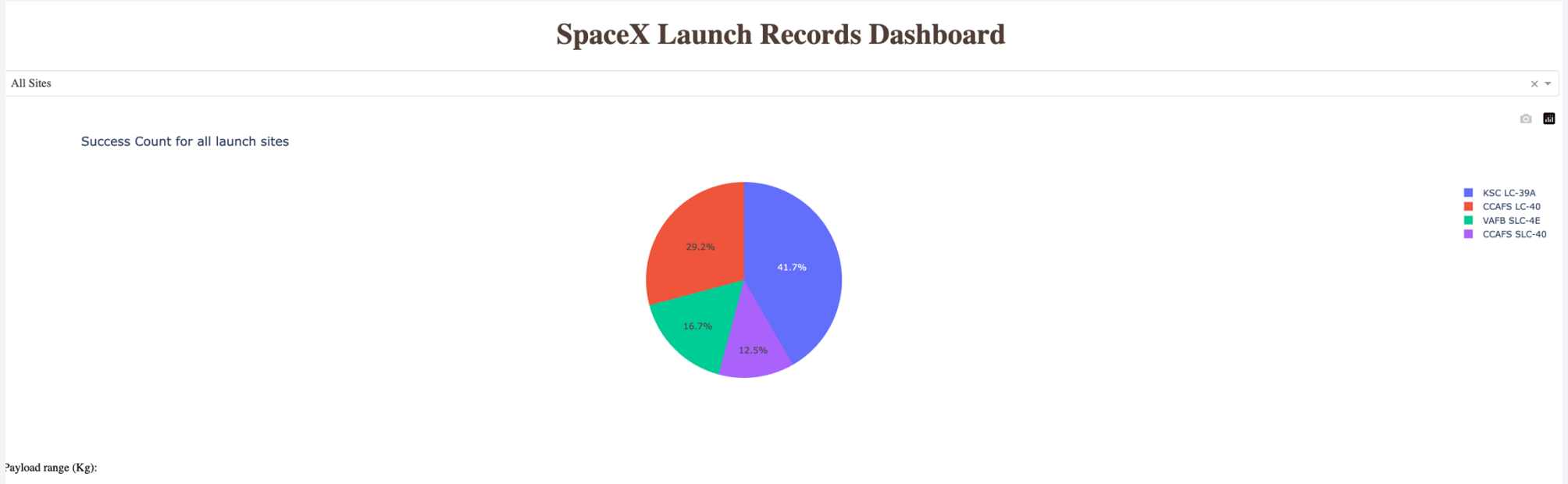
# ‘PolyLine’ between a launch site to the selected coastline point

- Used a folium.PolyLine and to find a between a launch site to the selected coastline point and the distance is 0.86 KM


Section 4
Build a Dashboard with Plotly Dash

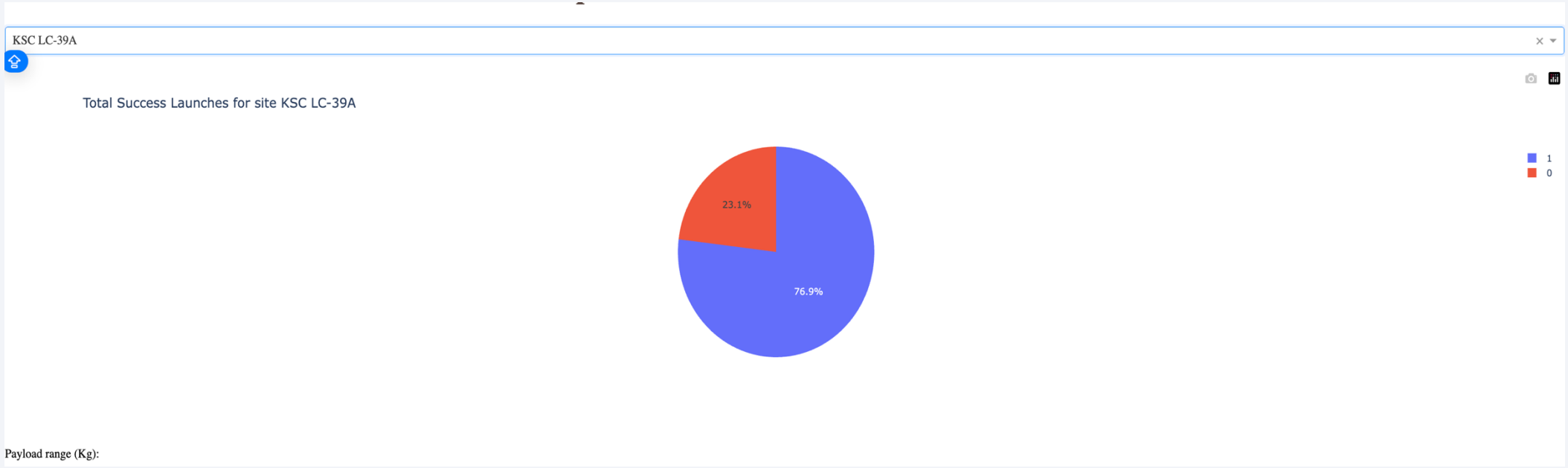
# Piechart of launch success count for all sites

- Screenshot of launch success count for all sites, in a piechart
- Launch site KSC LC-39A has the highest launch success rate at 42% followed by CCAFS LC-40 at 29%, VAFB SLC-4E at 17% and lastly launch site CCAFS SLC-40 with a success rate of 13%



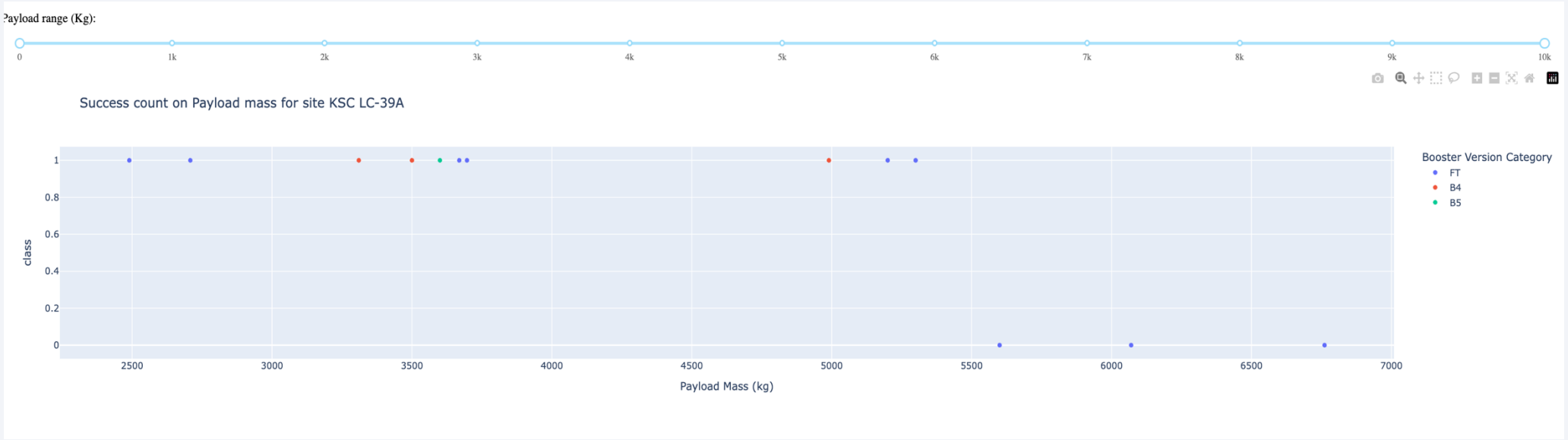
# Piechart for the launch site with highest launch success ratio

- Screenshot of the piechart for the launch site with highest launch success ratio
- Launch site KSC LC-39A had the 1st highest success ratio of 76% success against 23% failed launches



# Payload vs Launch Outcome for all sites

- Screenshots of Payload vs. Launch Outcome scatter plot for all sites, with different payload selected in the range slider
- For Launch site CCAFS LC-40 the booster version FT has the largest success rate from a payload mass of >2000kg



Section 5

# Predictive Analysis (Classification)

# Classification Accuracy

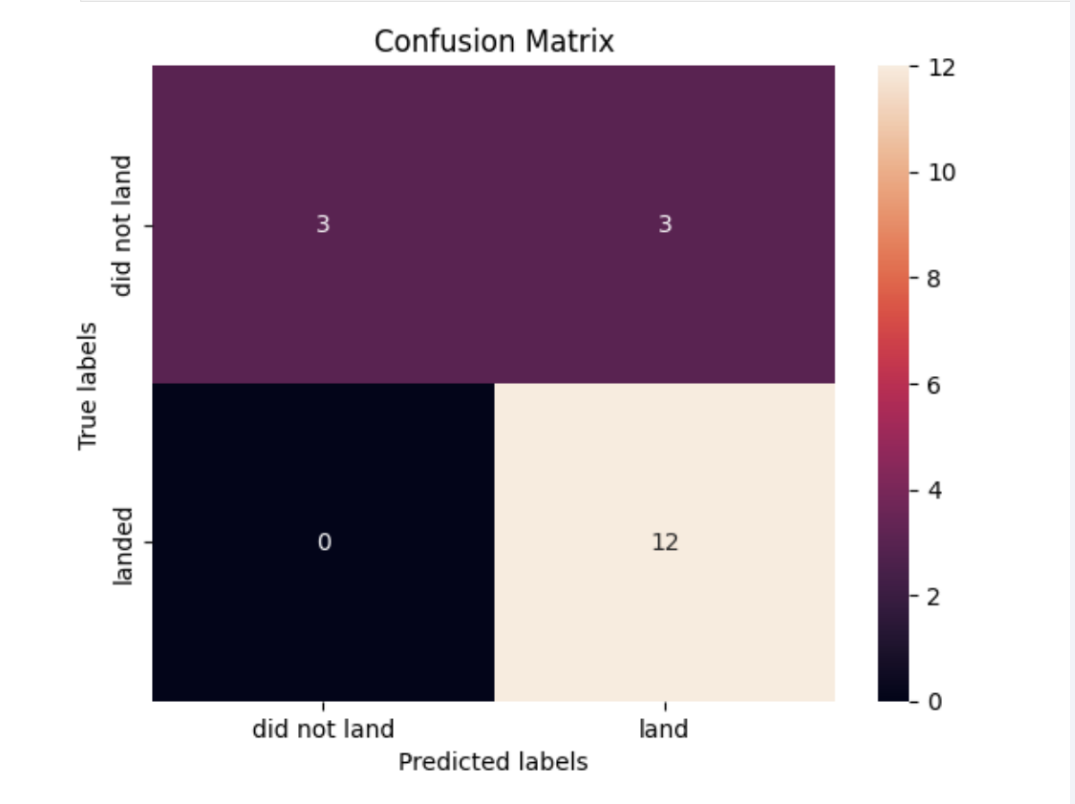
- Visualize the built model accuracy for all built classification models, in a bar chart

| | 0 |
|---|---|
| **Method** | Test Data Accuracy |
| **Logistic_Reg** | 0.833333 |
| **SVM** | 0.833333 |
| **Decision Tree** | 0.833333 |
| **KNN** | 0.833333 |

- All model has the same level of accuracy as 83%

# Confusion Matrix

- Confusion matrix of the best performing model with an explanation
- All four classification models had the same confusion matrices and equally distinguished between classes, but they all struggled with false positives.



# Conclusions

- **Launch Site Success Rates:** Different launch sites have different success rates. For example, CCAFS LC-40 has a success rate of 60%, while KSC LC-39A and VAFB SLC 4E have a success rate of 77%.
- **Success Rate Over Time:** As the number of flights increases at each launch site, so does the success rate. For instance, VAFB SLC 4E reaches a 100% success rate after the 50th flight, and both KSC LC-39A and CCAFS LC-40 achieve 100% success rates after the 80th flight.
- **Payload and Launch Site:** Looking at the payload vs. launch site chart, you'll notice that VAFB SLC 4E hasn't launched any rockets with heavy payloads (over 10,000 kg).
- **Orbit Success Rates:** Orbits like ES-L1, GEO, HEO, and SSO have the highest success rates at 100%. However, the SO orbit has the lowest success rate, with some having a 0% success rate.
- **LEO vs. GTO Success Rates:** In LEO orbit, success seems related to the number of flights. In contrast, there doesn't appear to be a relationship between flight number and success in GTO orbit.
- **Payload and Landing Success:** With heavy payloads, successful landings are more common in Polar, LEO, and ISS orbits. However, in GTO orbits, it's harder to distinguish between successful and unsuccessful landings.
- **Success Rate Over Years:** The overall success rate has been increasing since 2013 until 2020.

# 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

# Thank you!