IBM Developer
SKILLS NETWORK

# Winning Space Race with Data Science

Samuel Ogunseye
April 27 2025

# Outline

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

# Executive Summary

- Summary of methodologies
  - SpaceX Data Collection with the SpaceX API
  - SpaceX Data Collection with Web Scraping
  - SpaceX Data Wrangling
  - SpaceX Exploratory Data Analysis using SQL
  - SpaceX Exploratory Data Analysis using Pandas and Matplotlib
  - SpaceX Interactive Visualizations of Launch Sites with Folium and Plotly Dash
  - SpaceX Landing Prediction with Machine Learning
- Summary of all results
  - Exploratory Data Analysis outcomes
  - Interactive Visualizations
  - Landing Prediction via Classification

# Introduction

- Project background and context

SpaceX advertises Falcon 9 rocket launches on its website with a cost of $62 million. Other providers cost upward of $165 million dollars each and much of the savings on SpaceX's rocket launches, is because SpaceX can reuse the first stage of its rockets. 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 SpaceX for a rocket launch.

- Problems you want to find answers

In this capstone project, we will predict if the Falcon 9 rocket first stage will land successfully using data collected from the SpaceX website.

Section 1

# Methodology

# Methodology

## Executive Summary

- Data collection methodology:
  - Request and parse the SpaceX API launch data.
  - Request the Falcon 9 historical launch records from its Wikipedia page.
  - Convert the parsed data into a Pandas dataframe.
  - Filter the dataframe to only include Falcon 9 launches and launch sites.
- Perform data wrangling
  - Deal with missing values.
  - Calculate the number of launches on each site.
  - Calculate the number and occurrence of each orbit and mission outcome of the orbits.

# Methodology

## Executive Summary

- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - Write Python code to conduct Exploratory Data Analysis by manipulating data in a Pandas dataframe.
  - Write and execute SQL queries to select and sort data.
  - Generate interactive maps, calculate distance, plot coordinates, and mark clusters by writing Python code using Folium.
  - Build an interactive dashboard that contains pie charts and scatter plots to analyze data with Plotly Dash.
  - Train and deploy different classification models / parameters.
  - Express hyperparameter grid search and the accuracy of different classification models / parameters.
  - Use Machine Learning to build a predictive model to aid launch predictions and answer the business problem.

# Data Collection

- Data was collected using the SpaceX API (a RESTful API) via a get request to the SpaceX API launch data using its url. This was then parsed into a JSON result and converted into a Pandas dataframe.
- Web scraping was used to request the Falcon 9 historical launch records from its Wikipedia page. A BeautifulSoup object was created from the HTML response. The column/variable names were extracted from the HTML table header and a Pandas dataframe was created from it by parsing the launch HTML tables.

# Data Collection – SpaceX API

- Data was collected using the SpaceX API (a RESTful API) via a get request to the SpaceX API launch data using its url. This was then parsed into a JSON result and converted into a Pandas dataframe.
- This is the GitHub URL of the completed SpaceX API calls notebook displayed here.

**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:

[9]:
```
# 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:

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]:
```
# We should see that the request was successfull with the 200 status response code.

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

# Use json_normalize method to convert the json result into a dataframe.

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

Using the dataframe `data` print the first 5 rows

# Data Collection - Scraping

- Web scraping was used to request the Falcon 9 historical launch records from its Wikipedia page. A BeautifulSoup object was created from the HTML response.
  The column/variable names were extracted from the HTML table header and a Pandas dataframe was created from it by parsing the launch HTML tables.
- This is the GitHub URL of the completed Web Scraping notebook displayed here.

**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.

[9]:
```
# TASK 1: Request the Falcon9 Launch Wiki page from its URL.

# use requests.get() method with the provided static_url
# assign the response to an object

response = requests.get(static_url)
```

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

[13]:
```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content.

soup = BeautifulSoup(response.content, 'html.parser')
```

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

[14]:
```
# Print the page title to verify if the BeautifulSoup object was created properly.

# Use soup.title attribute

soup.title
```

[14]:
```
<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

[15]:
```
# TASK 2: Extract all column/variable names from the HTML table header.

# 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

- After creating the Pandas dataframe, the missing values in the ***PayloadMass*** were replaced with the mean value. The resulting data was filtered using the ***BoosterVersion*** column to show only the Falcon 9 launches and the corresponding launch sites.
- The number of launches on each site was derived using ***value_counts()***. The same method was used to derive the number and occurrence of each orbit, and the number and occurrence of mission outcome of the orbits. The landing outcome and landing class were established from these.
- This is the GitHub URL of the completed Data Wrangling notebook [displayed here.]

**TASK 4: Create a landing outcome label from Outcome column**

Using the `Outcome`, create a list where the element is zero if the corresponding row in `Outcome` is in the set `bad_outcome`; otherwise, it's one. Then assign it to the variable `landing_class`:

[15]:
```
# TASK 4: Create a landing outcome label from the 'Outcome' column.

# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise

df['Class'] = df['Outcome'].apply(lambda x: 0 if x in bad_outcomes else 1)
df['Class'].value_counts()
```

[15]:
```
Class
1    60
0    30
Name: count, dtype: int64
```

This variable will represent the classification variable that represents the outcome of each launch. If the value is zero, the first stage did not land successfully; one means the first stage landed Successfully

[17]:
```
landing_class = df['Class']
df[['Class']].head(8)
```

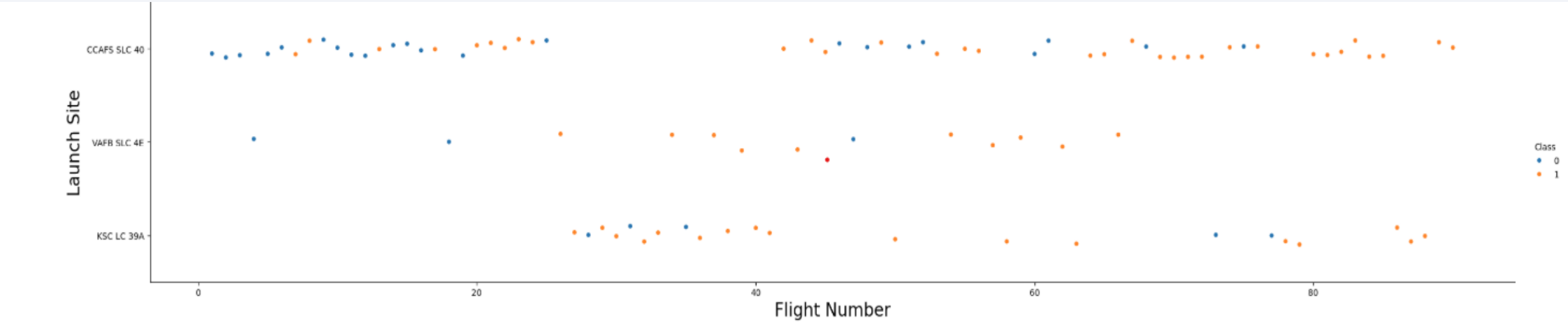
[17]:

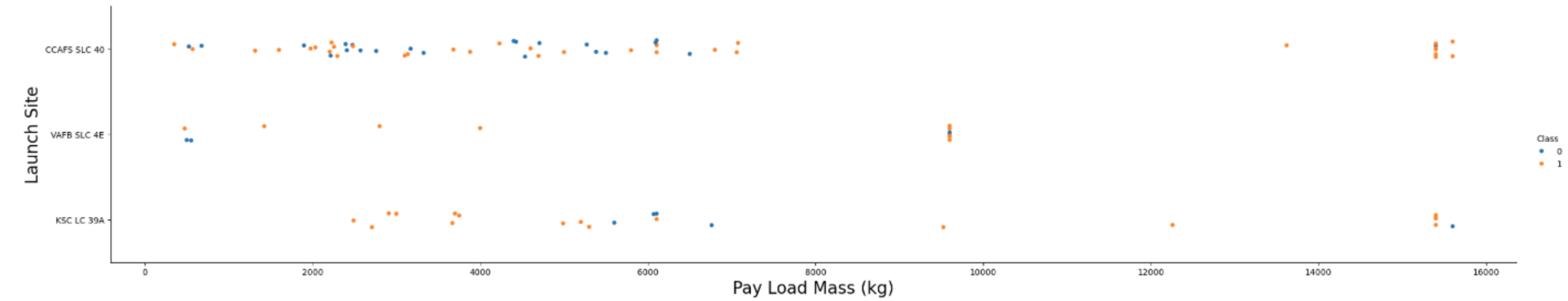
| | Class |
|---|---|
| 0 | 0 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 1 |
| 7 | 1 |

# EDA with Data Visualization

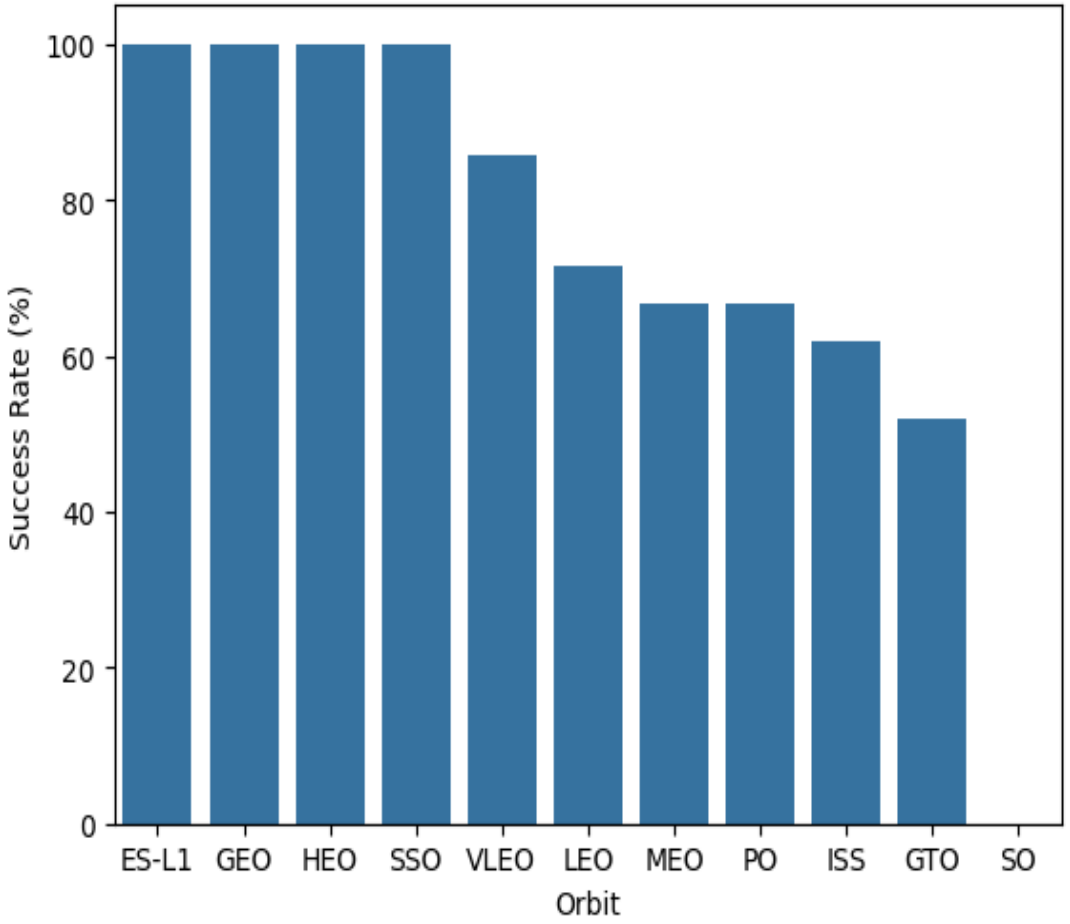
- Wrote Python code to conduct Exploratory Data Analysis by manipulating data in a Pandas dataframe.
- Used scatter plots to visualize the relationship between Flight Number and Launch Site, Payload Mass and Launch Site, FlightNumber and Orbit type, and Payload Mass and Orbit type.
- Used a bar chat to visualize the relationship between success rate of each orbit type.
- Used a line chart to visualize the launch success yearly trend.
- Deployed Features Engineering to obtain some insight about how the columns `Orbits`, `LaunchSite`, `LandingPad` and `Serial` would affect the launch success rate.
- This is the GitHub URL of the completed EDA with Data Visualization notebook displayed here.

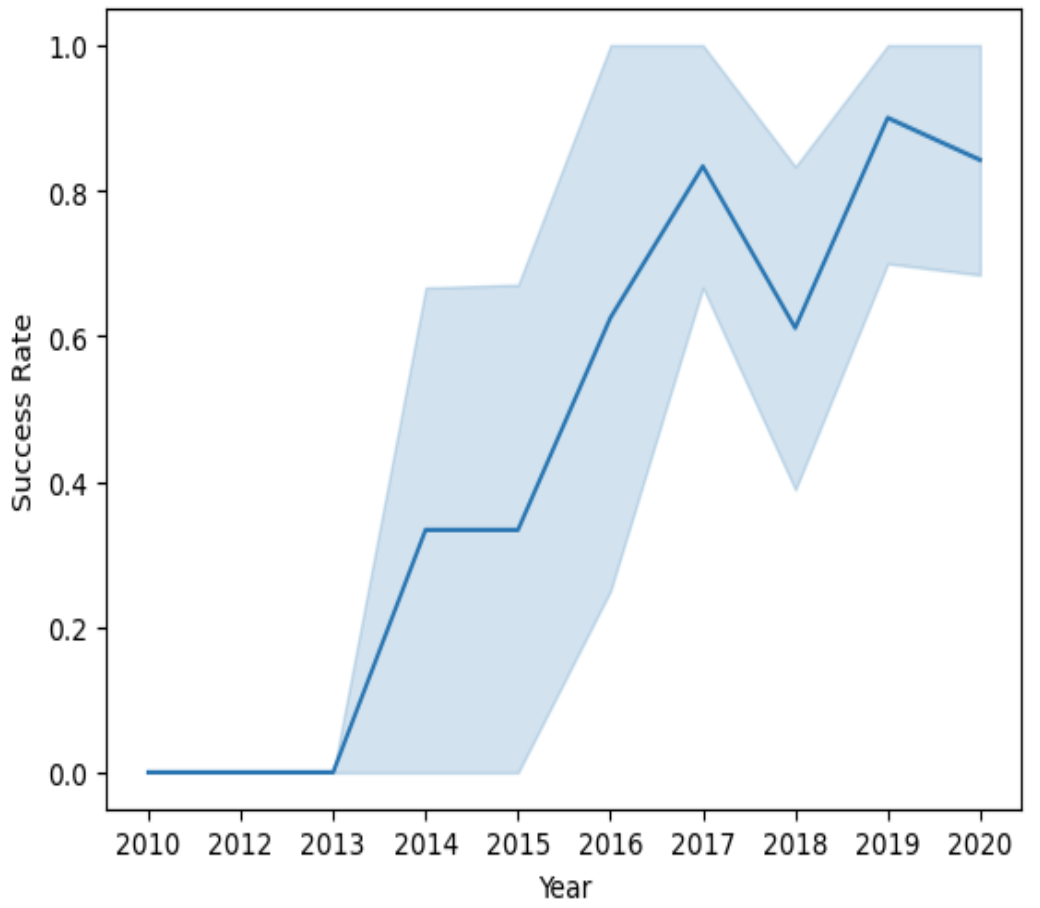
# EDA with Data Visualization



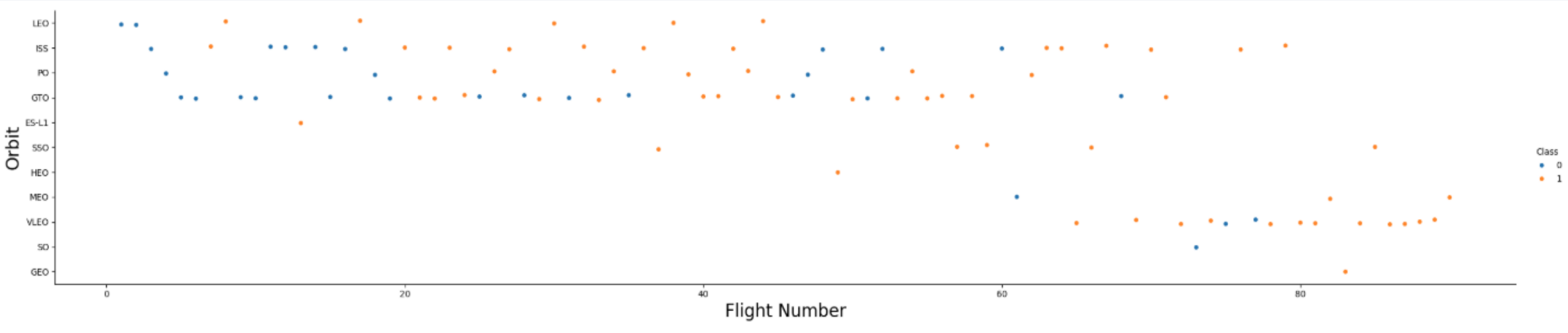



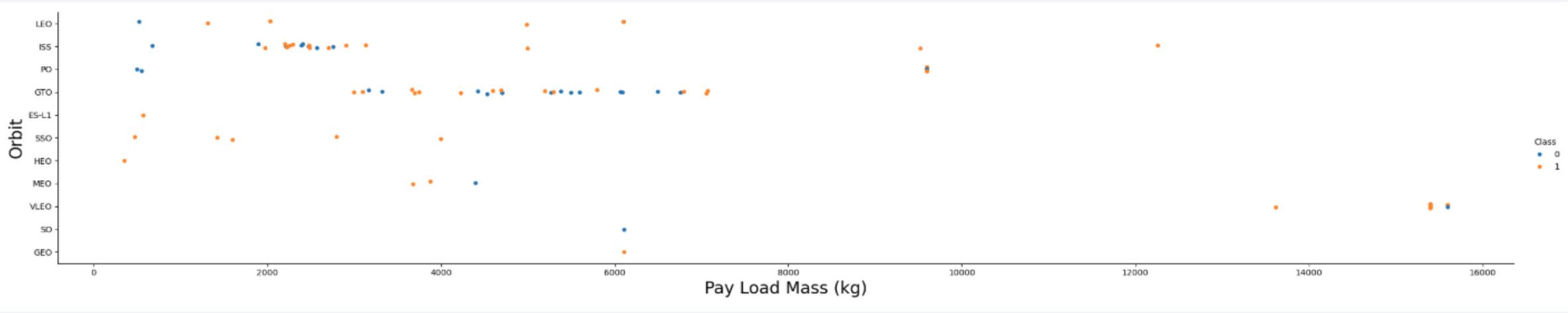
# EDA with Data Visualization



# EDA with Data Visualization



# EDA with SQL

- The following SQL queries were performed with Exploratory Data Analysis.
  - Display the names of the unique launch sites in the space mission.

```
[11]: # Task 1: Display the names of the unique launch sites in the space mission.

      %sql SELECT DISTINCT LAUNCH_SITE as "Launch_Sites" FROM SPACEXTBL;

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

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

```
[12]: # Task 2: Display 5 records where launch sites begin with the string 'CCA'.

      %sql SELECT * FROM 'SPACEXTBL' WHERE Launch_Site LIKE 'CCA%' LIMIT 5;

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

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

```
[13]: # Task 3: Display the total payload mass carried by boosters launched by NASA (CRS).

      %sql SELECT SUM(PAYLOAD_MASS__KG_) as "Total Payload Mass(Kgs)", Customer FROM 'SPACEXTBL' WHERE Customer = 'NASA (CRS)';

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

# EDA with SQL

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

```
[14]: # Task 4: Display average payload mass carried by booster version F9 v1.1.

%sql SELECT AVG(PAYLOAD_MASS__KG_) as "Average Payload Mass Kgs", Customer, Booster_Version FROM 'SPACEXTBL' WHERE Booster_Version LIKE 'F9 v1.1%';

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

- List the date when the first successful landing outcome in ground pad was achieved.

```
[15]: # Task 5: List the date when the first successful landing outcome in ground pad was achieved.

%sql SELECT MIN(DATE), Landing_Outcome FROM 'SPACEXTBL' WHERE "Landing _Outcome" = "Success (ground pad)";

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

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

```
[16]: # 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.

%sql SELECT DISTINCT Booster_Version, Payload FROM SPACEXTBL WHERE "Landing _Outcome" = "Success (drone ship)" AND PAYLOAD_MASS__KG_ > 4000 AND PAYLOAD_MASS__KG_ < 6000;

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

# EDA with SQL

- List the total number of successful and failure mission outcomes.

```
[17]: # Task 7: List the total number of successful and failure mission outcomes.

%sql SELECT "Mission_Outcome", COUNT("Mission_Outcome") as Total FROM SPACEXTBL GROUP BY "Mission_Outcome";

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

- List all the booster_versions that have carried the maximum payload mass. Use a subquery.

```
[18]: # Task 8: List all the booster_versions that have carried the maximum payload mass. Use a subquery.

%sql SELECT "Booster_Version", Payload, "PAYLOAD_MASS__KG_" FROM SPACEXTBL WHERE "PAYLOAD_MASS__KG_" = (SELECT MAX("PAYLOAD_MASS__KG_") FROM SPACEXTBL);

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

- 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.

```
[25]: # 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.

%sql SELECT substr(Date,6,2), substr(Date,0,5),"Booster_Version", "Launch_Site", Payload, "PAYLOAD_MASS__KG_", "Mission_Outcome", "Landing _Outcome" FROM SPACEXTBL WHERE substr(Date,6,2)='2015' AND "Landing _Outcome" = 'Failure (drone ship)';

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

# EDA with SQL

- 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]: # 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.

%sql SELECT * FROM SPACEXTBL WHERE "Landing _Outcome" LIKE 'Success%' AND (Date BETWEEN '2010-06-04' AND '2017-03-20') ORDER BY Date DESC;

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

- This is the GitHub URL of the completed EDA with SQL notebook displayed here.

# Build an Interactive Map with Folium

- Created Folium map to mark all the launch sites, mark the success/failed launches for each site on the map, and calculated the distances between a launch site to its proximities.
- Created and added map objects like `folium.Circle`, `folium.Marker`, and `folium.PolyLine` to the Folium map.
- `folium.Circle` was used to add a highlighted circle area with a text label on a specific coordinate; `folium.Marker` was used to generate marked launch sites; and `folium.PolyLine` was used to draw lines between the launch sites to selected points on the coastline on the Folium map respectively.
- This is the GitHub URL of the completed interactive map with Folium map displayed here.

# Build a Dashboard with Plotly Dash

- Built an interactive dashboard with Plotly Dash to add a Launch Site Drop-down Input Component; add a callback function to render success-pie-chart based on selected site dropdown; add a Range Slider to Select Payload; and add a callback function to render the success-payload-scatter-chart scatter plot.
  - The Launch Site Drop-down Input Component was added to enable the selection of one launch site and check its detailed success rate (class = 0 vs. class = 1).
  - The callback function to render success-pie-chart based on selected site dropdown was added to get the selected launch site from the site-dropdown. This was then used to render a pie chart visualizing launch success counts.
  - The Range Slider to Select Payload was added to find if variable payload is correlated to mission outcome. From a dashboard point of view, this will enable the easy selection of different payload range and show if some visual patterns can be identified.
  - The callback function to render the success-payload-scatter-chart scatter plot was added to visually observe how payload may be correlated with mission outcomes for selected site(s).
- This is the GitHub URL of the completed interactive dashboard with Plotly Dash displayed here.

# Predictive Analysis (Classification)

- I loaded the data as a Pandas dataframe and created a NumPy array from the column `Class` in data, by applying the method `to_numpy()` then assign it to the output variable `Y`, which is a Pandas series.
- I standardized the data, then reassigned it to the variable `X` using the transform function `preprocessing.StandardScaler()` derived from `sklearn`.
- Thereafter, I used the function `train_test_split` to split the data `X` and `Y` into training and test data. The parameter `test_size` was set to 0.2 and the `random_state` was set to 2. This function was derived from `sklearn.model_selection`.
- To establish the best performing Machine Learning model on the test data, I did the following to the Logistic Regression, Support Vector Machine (SVM), Decision Trees and K Nearest Neighbors (KNN) models:
  - I created an object for each model, then created a `GridSearchCV` object with cv = 10. From this, I fit the object to the training data to find the best parameters from the dictionary `parameters`.
  - After fitting the training data, I output the `GridSearchCV` object for each model. I then displayed the best parameters using the data attribute `best_params_` and the accuracy on the validation data using the data attribute `best_score_`.
  - Finally, I calculated the accuracy on the test data for each model using the method `score`, and plotted a `confusion matrix` for each model using the test and predicted outcomes.

# Predictive Analysis (Classification)

- The table below shows the Test Data Accuracy score for the Logistic Regression, Support Vector Machine (SVM), Decision Trees and K Nearest Neighbors (KNN) models. All the models have the same accuracy of 0.833333 on the test data i.e. they perform equally.

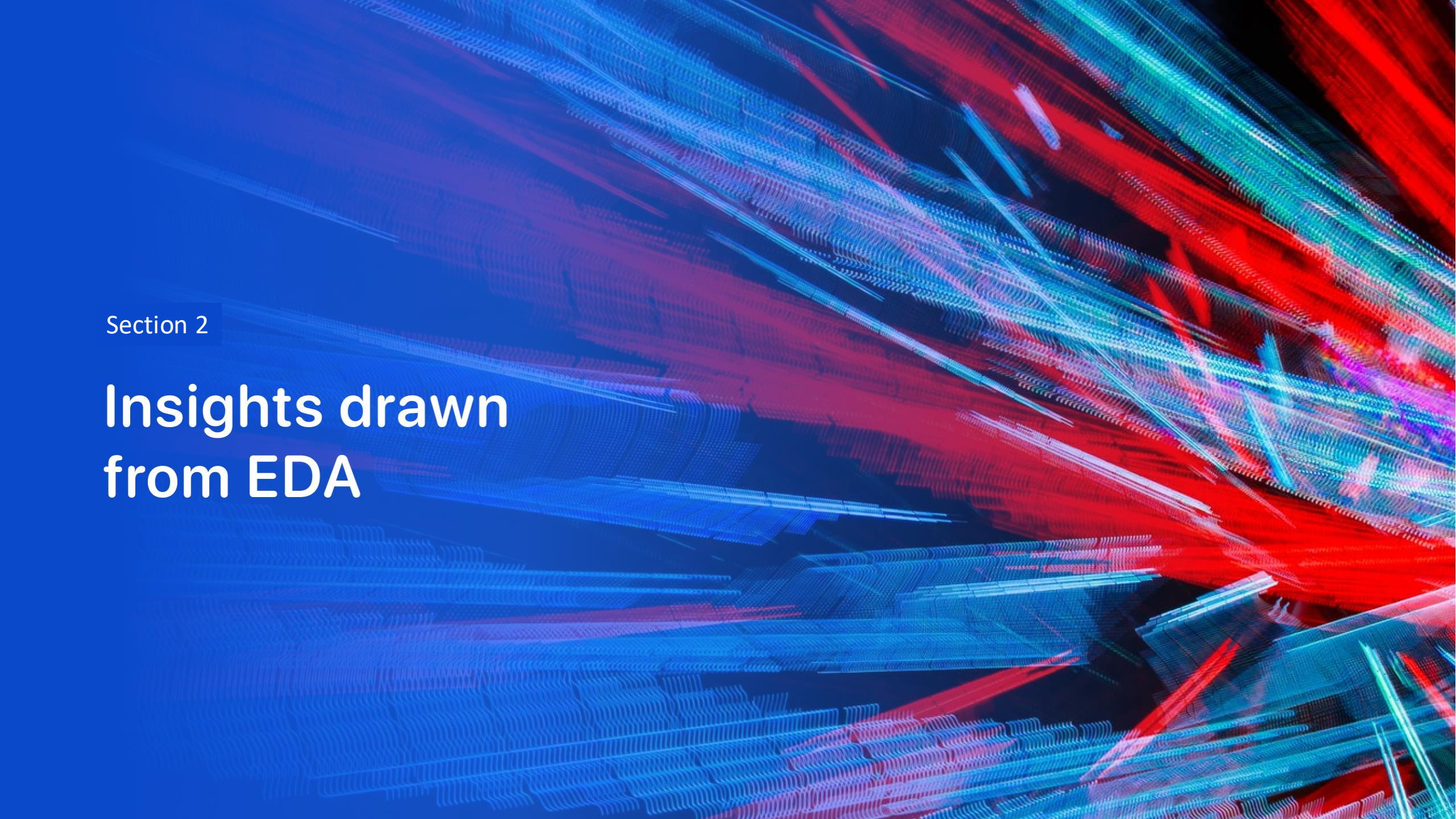
[40]:

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

- This is the GitHub URL of the completed Predictive Analysis (Classification) with Machine Learning displayed here.

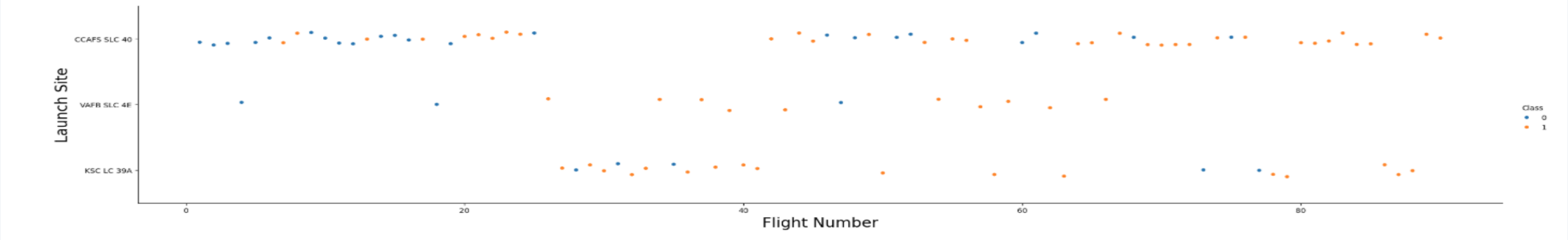
# Results

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

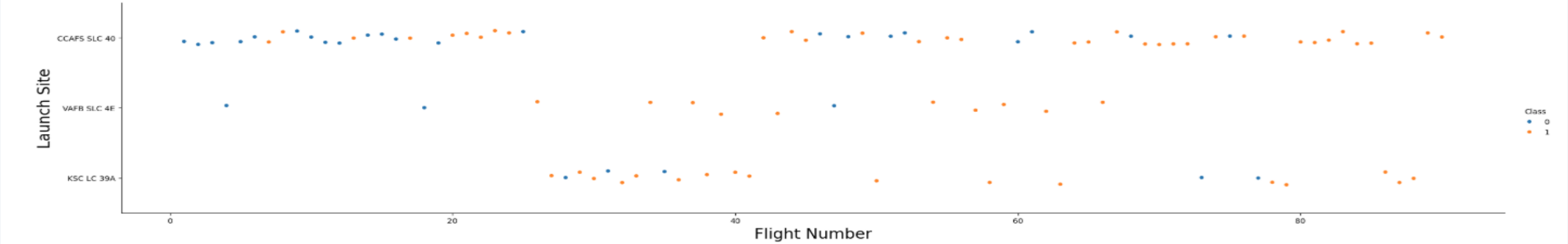

Section 2
Insights drawn from EDA

# Flight Number vs. Launch Site

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




- Show the screenshot of the scatter plot with explanations

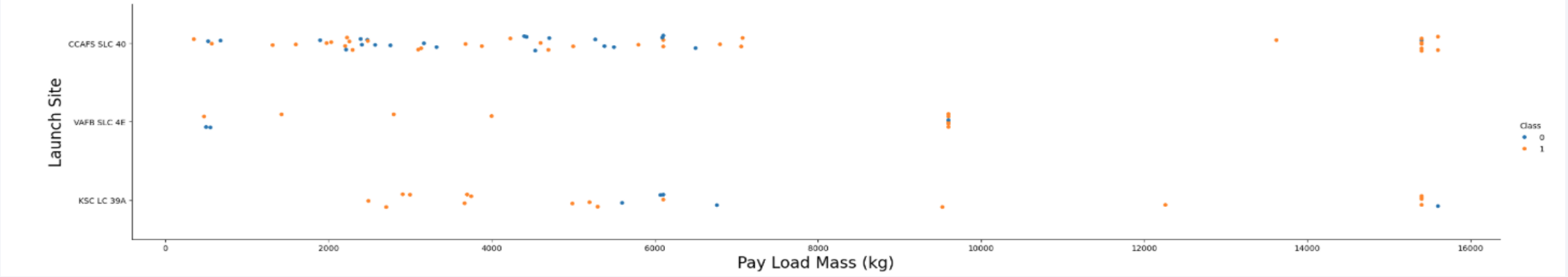



Now try to explain the patterns you found in the Flight Number vs. Launch Site scatter point plots.

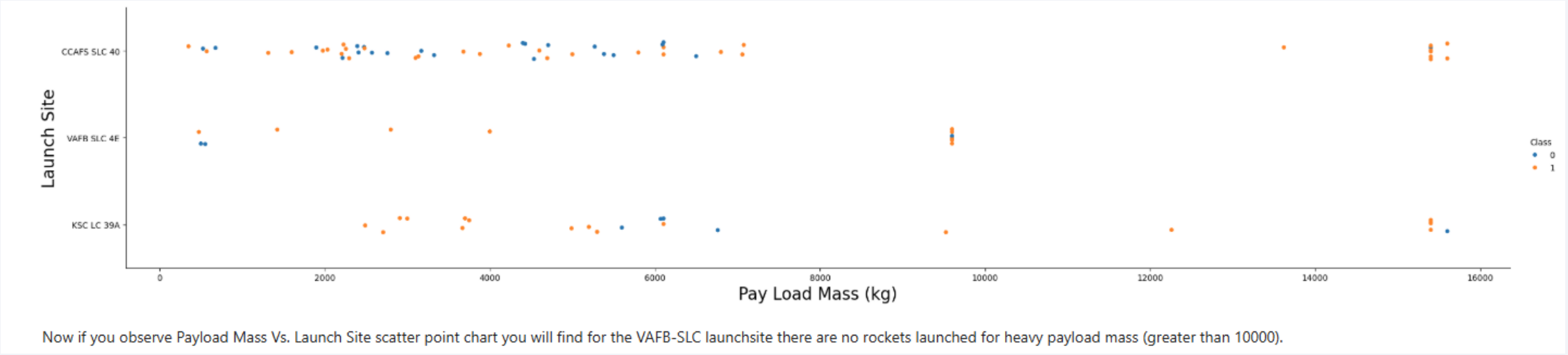
***As the flight number increases in each of the 3 launch sites, the success rate increases too. The success rate for the VAFB SLC 4E Launch Site is 100% after the 50th Flight Number. For the KSC LC 39A and CCAFS SLC 40 Launch Sites, they each achieve a 100% success rate after the 80th Flight Number.***

# Payload vs. Launch Site

- Show a scatter plot of Payload vs. Launch Site

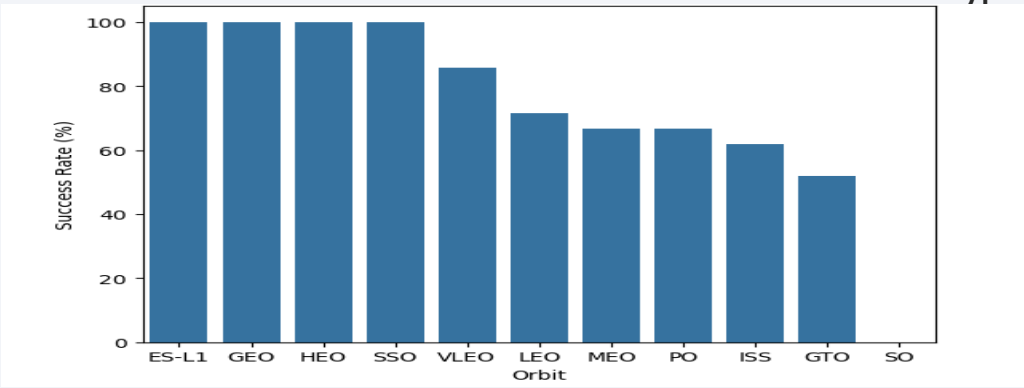



- Show the screenshot of the scatter plot with explanations

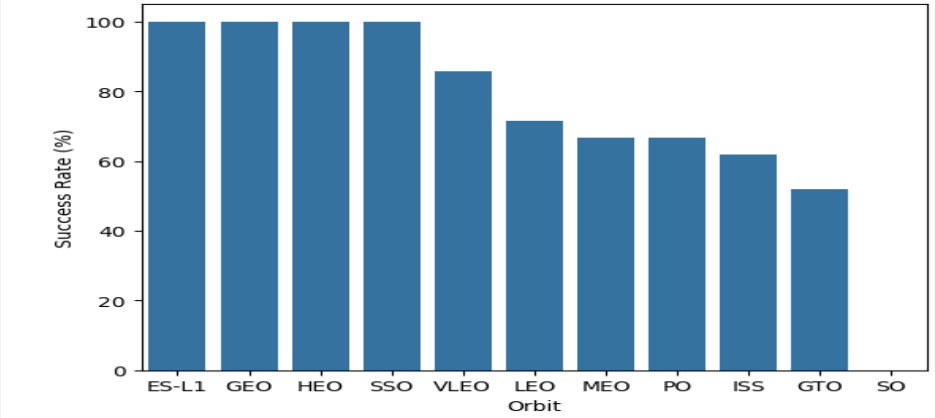


# Success Rate vs. Orbit Type

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




- Show the screenshot of the bar chat with explanations

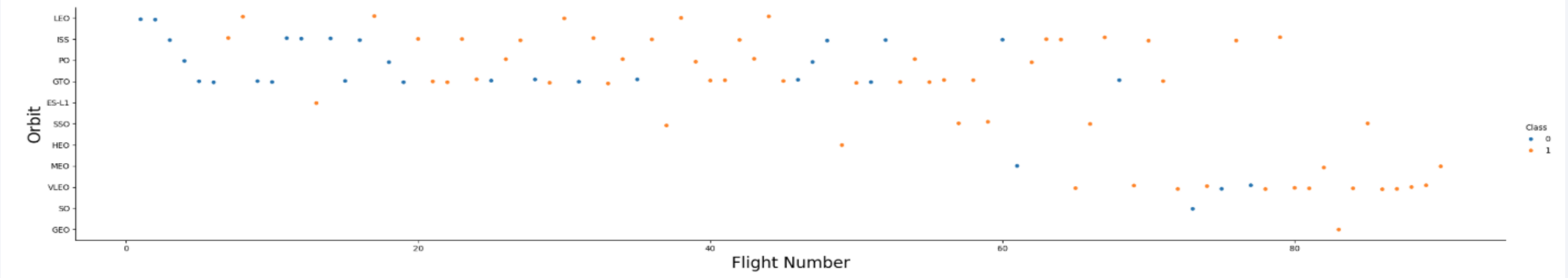



Analyze the plotted bar chart to identify which orbits have the highest success rates.

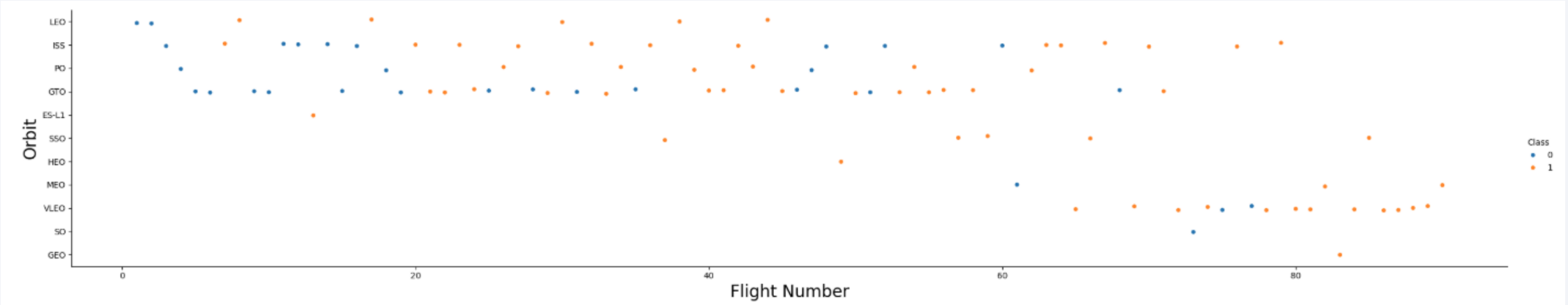
***The orbits ES-L1, GEO, HEO and SSO have the highest success rates of 100%. The VLEO, LEO, MEO, PO, ISS and GTO orbits have success rates ranging from just above 80% to around 50%. The SO orbit has a 0% success rate.***

# Flight Number vs. Orbit Type

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



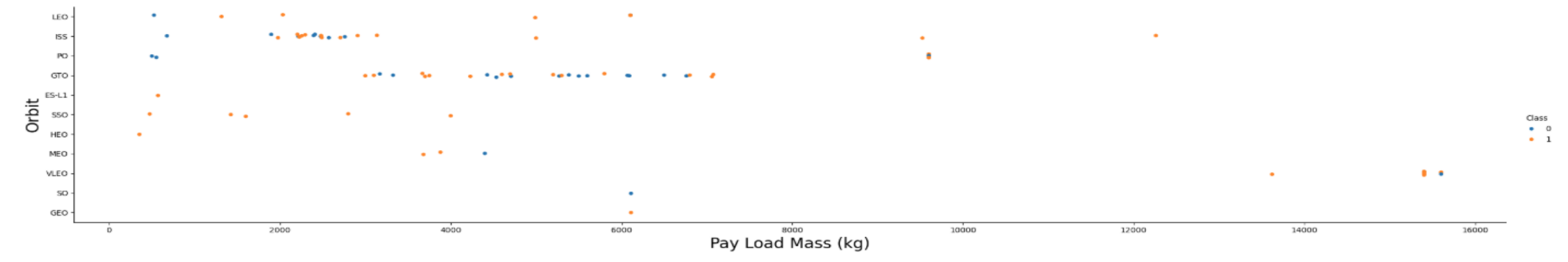

- Show the screenshot of the scatter plot with explanations



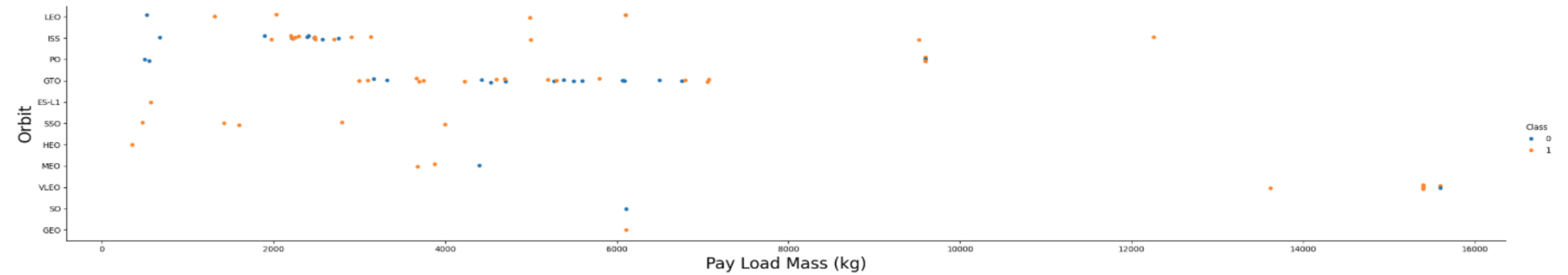

You can observe that in the LEO orbit, success seems to be related to the number of flights. Conversely, in the GTO orbit, there appears to be no relationship between flight number and success.

# Payload vs. Orbit Type

- Show a scatter point of payload vs. orbit type




- Show the screenshot of the scatter plot with explanations

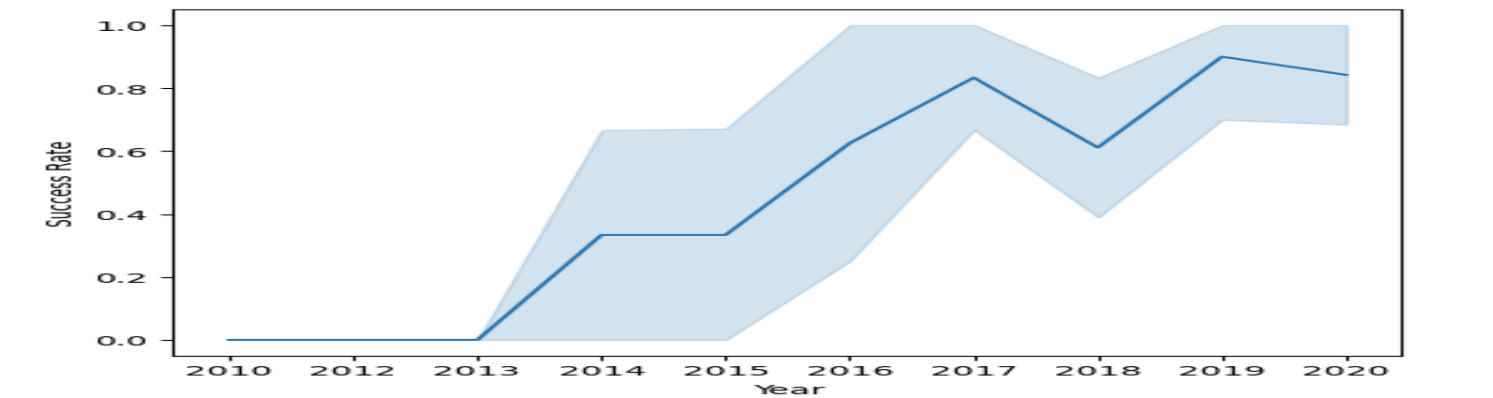



With heavy payloads the successful landing or positive landing rate are more for Polar, LEO and ISS.

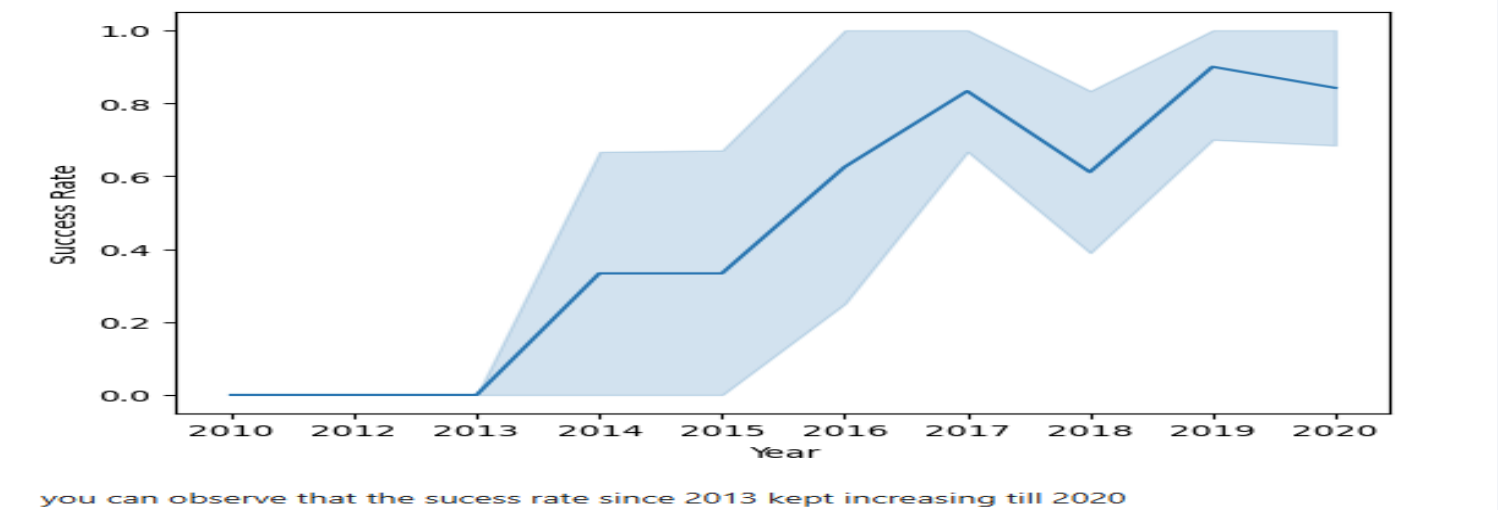
However, for GTO, it's difficult to distinguish between successful and unsuccessful landings as both outcomes are present.

# Launch Success Yearly Trend

- Show a line chart of yearly average success rate




- Show the screenshot of the line chart with explanations



# All Launch Site Names

- Find the names of the unique launch sites

### Task 1

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

In [11]:
```
# Task 1: Display the names of the unique launch sites in the space mission.

%sql SELECT DISTINCT LAUNCH_SITE as "Launch_Sites" FROM SPACEXTBL;
```

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

Out[11]:

| Launch_Sites |
|---|
| CCAFS LC-40 |
| VAFB SLC-4E |
| KSC LC-39A |
| CCAFS SLC-40 |

- I used the `SELECT DISTINCT` statement to return all the unique launch sites from the `LAUNCH_SITE` column of the `SPACEXTBL` table.

# Launch Site Names Begin with 'CCA'

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

**Task 2**

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

In [12]:
```
# Task 2: Display 5 records where launch sites begin with the string 'CCA'.
%sql SELECT * FROM 'SPACEXTBL' WHERE Launch_Site LIKE 'CCA%' LIMIT 5;
```

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

Out[12]:

| 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 |

- I used the `SELECT` statement to return launch sites from the `LAUNCH_SITE` column of the `SPACEXTBL` table. Then, I used the `WHERE` and `LIKE` commands with the `%` wildcard to display the launch sites that begin with the string `CCA`. I limited the result of this display to 5 launch sites with the `LIMIT 5` statement.

# 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)

In [13]:
```
# Task 3: Display the total payload mass carried by boosters launched by NASA (CRS).

%sql SELECT SUM(PAYLOAD_MASS__KG_) as "Total Payload Mass(Kgs)", Customer FROM 'SPACEXTBL' WHERE Customer = 'NASA (CRS)';
```

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

Out[13]:

| Total Payload Mass(Kgs) | Customer |
|---|---|
| 45596 | NASA (CRS) |

- I used the `SELECT` statement and `SUM` function to return the sum of the `PAYLOAD_MASS__KG_` column of the `SPACEXTBL` table. Then, I used the `WHERE Customer = 'NASA (CRS)'` command to display only the result from the column `Customer` for the specific customer `NASA (CRS)`.

# Average Payload Mass by F9 v1.1

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

```
[14]: # Task 4: Display average payload mass carried by booster version F9 v1.1.

%sql SELECT AVG(PAYLOAD_MASS__KG_) as "Average Payload Mass Kgs", Customer, Booster_Version FROM 'SPACEXTBL' WHERE Booster_Version LIKE 'F9 v1.1%';

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

**Task 4**

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

```
In [14]: # Task 4: Display average payload mass carried by booster version F9 v1.1.

%sql SELECT AVG(PAYLOAD_MASS__KG_) as "Average Payload Mass Kgs", Customer, Booster_Version FROM 'SPACEXTBL' WHERE Booster_

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

Out[14]:

| Average Payload Mass Kgs | Customer | Booster_Version |
|---|---|---|
| 2534.6666666666665 | MDA | F9 v1.1 B1003 |

- I used the `SELECT` statement and `AVG` function to return the average of the `PAYLOAD_MASS__KG_` column of the `SPACEXTBL` table. Then, I used the `Customer` and `Booster_Version` columns with the command `WHERE Booster_Version LIKE 'F9 v1.1%'` to display only the result for customers that had booster version F9 v1.1.

# 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 successful landing outcome in ground pad was achieved.

*Hint: Use min function*

In [15]:
```
# Task 5: List the date when the first successful landing outcome in ground pad was achieved.

%sql SELECT MIN(DATE), Landing_Outcome FROM 'SPACEXTBL' WHERE "Landing _Outcome" = "Success (ground pad)";
```

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

Out[15]:

| MIN(DATE) | Landing_Outcome |
|---|---|
| None | None |

- I used the `SELECT` statement and `MIN` function to return the earliest / first successful landing outcome on ground pad from the `Landing_Outcome` column of the `SPACEXTBL` table. Then, I used the command `WHERE "Landing _Outcome" = "Success (ground pad)"` to display only the result for first successful landing outcome on ground pad.

# 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

```
[16]: # 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.

%sql SELECT DISTINCT Booster_Version, Payload FROM SPACEXTBL WHERE "Landing _Outcome" = "Success (drone ship)" AND PAYLOAD_MASS__KG_ > 4000 AND PAYLOAD_MASS__KG_ < 6000;

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

**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 [16]: # Task 6: List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less
         %sql SELECT DISTINCT Booster_Version, Payload FROM SPACEXTBL WHERE "Landing _Outcome" = "Success (drone ship)" AND PAYLOAD_
 * sqlite:///my_data1.db
Done.
```

Out[16]:

| Booster_Version | Payload |
|---|---|

- I used the `SELECT DISTINCT` statement to return the ``Booster_Version` and `Payload` columns of the `SPACEXTBL` table. Then, I used the command `WHERE "Landing _Outcome" = "Success (drone ship)" AND PAYLOAD_MASS__KG_ > 4000 AND PAYLOAD_MASS__KG_ < 6000` to display only the result for successful drone ship landings with payload mass between 4000 and 6000.

# 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**

In [17]:
```
# Task 7: List the total number of successful and failure mission outcomes.

%sql SELECT "Mission_Outcome", COUNT("Mission_Outcome") as Total FROM SPACEXTBL GROUP BY "Mission_Outcome";
```

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

Out[17]:

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

- I used the `SELECT`, `COUNT` and `GROUP BY` statements to return the total number of successful and failure mission outcomes from the `Mission_Outcome` column of the `SPACEXTBL` table.

# Boosters Carried Maximum Payload

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

```
[18]: # Task 8: List all the booster_versions that have carried the maximum payload mass. Use a subquery.

%sql SELECT "Booster_Version", Payload, "PAYLOAD_MASS__KG_" FROM SPACEXTBL WHERE "PAYLOAD_MASS__KG_" = (SELECT MAX("PAYLOAD_MASS__KG_") FROM SPACEXTBL);

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

**Task 8**

**List all the booster_versions that have carried the maximum payload mass. Use a subquery.**

```
In [18]: # Task 8: List all the booster_versions that have carried the maximum payload mass. Use a subquery.
         %sql SELECT "Booster_Version", Payload, "PAYLOAD_MASS__KG_" FROM SPACEXTBL WHERE "PAYLOAD_MASS__KG_" = (SELECT MAX("PAYLOAD_

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

Out[18]:

| Booster_Version | Payload | PAYLOAD_MASS__KG_ |
|---|---|---|
| F9 B5 B1048.4 | Starlink 1 v1.0, SpaceX CRS-19 | 15600 |
| F9 B5 B1049.4 | Starlink 2 v1.0, Crew Dragon in-flight abort test | 15600 |
| F9 B5 B1051.3 | Starlink 3 v1.0, Starlink 4 v1.0 | 15600 |
| F9 B5 B1056.4 | Starlink 4 v1.0, SpaceX CRS-20 | 15600 |
| F9 B5 B1048.5 | Starlink 5 v1.0, Starlink 6 v1.0 | 15600 |
| F9 B5 B1051.4 | Starlink 6 v1.0, Crew Dragon Demo-2 | 15600 |
| F9 B5 B1049.5 | Starlink 7 v1.0, Starlink 8 v1.0 | 15600 |
| F9 B5 B1060.2 | Starlink 11 v1.0, Starlink 12 v1.0 | 15600 |
| F9 B5 B1058.3 | Starlink 12 v1.0, Starlink 13 v1.0 | 15600 |
| F9 B5 B1051.6 | Starlink 13 v1.0, Starlink 14 v1.0 | 15600 |
| F9 B5 B1060.3 | Starlink 14 v1.0, GPS III-04 | 15600 |
| F9 B5 B1049.7 | Starlink 15 v1.0, SpaceX CRS-21 | 15600 |

- I used a `SELECT` statement to return the `Booster_Version`, `Payload` and `PAYLOAD_MASS__KG_` columns of the `SPACEXTBL` table. Then I deployed a command `WHERE` and a subquery to list the names of the boosters which have carried the maximum payload mass.

# 2015 Launch Records

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

```
[25]: # 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.

%sql SELECT substr(Date,6,2), substr(Date,0,5),"Booster_Version", "Launch_Site", Payload, "PAYLOAD_MASS__KG_", "Mission_Outcome", "Landing _Outcome" FROM SPACEXTBL WHERE substr(Date,6,2)='2015' AND "Landing _Outcome" = 'Failure (drone ship)';

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

**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 [25]: # Task 9: List the records which will display the month names, failure landing_outcomes in drone ship ,booster versions, la
         %sql SELECT substr(Date,6,2), substr(Date,0,5),"Booster_Version", "Launch_Site", Payload, "PAYLOAD_MASS__KG_", "Mission_Out
          * sqlite:///my_data1.db
        Done.
```

Out[25]:

| substr(Date,6,2) | substr(Date,0,5) | Booster_Version | Launch_Site | Payload | PAYLOAD_MASS__KG_ | Mission_Outcome | "Landing _Outcome" |
|---|---|---|---|---|---|---|---|

- I used `substr` in the `SELECT` statement to derive the month and year from the `Date` column of the `SPACEXTBL` table. Commands were deployed on the columns `Booster_Version`, `Launch_Site`, `Payload`, `PAYLOAD_MASS_KG_`, `Mission_Outcome`, `Landing_Outcome`, to list the failed landing_outcomes in drone ship, their booster versions, and launch site names for the months in year 2015.

# 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

```
[21]: # 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.

%sql SELECT * FROM SPACEXTBL WHERE "Landing _Outcome" LIKE 'Success%' AND (Date BETWEEN '2010-06-04' AND '2017-03-20') ORDER BY Date DESC;

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

**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 [21]: # Task 10: Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-
         %sql SELECT * FROM SPACEXTBL WHERE "Landing _Outcome" LIKE 'Success%' AND (Date BETWEEN '2010-06-04' AND '2017-03-20') ORDE
 * sqlite:///my_data1.db
Done.
```

Out[21]:

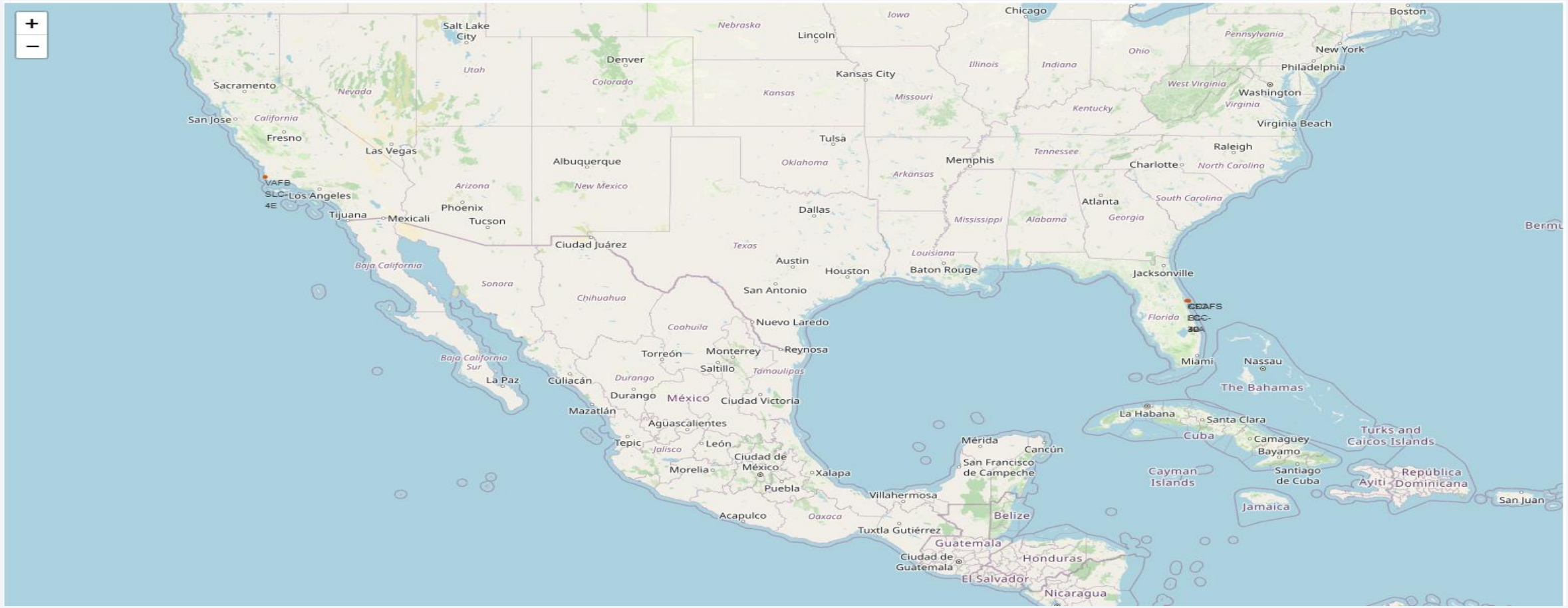
| Date | Time (UTC) | Booster_Version | Launch_Site | Payload | PAYLOAD_MASS__KG_ | Orbit | Customer | Mission_Outcome | Landing_Outcome |
|---|---|---|---|---|---|---|---|---|---|

- I used the `SELECT` statement to return the `Landing_Outcome` column of the `SPACEXTBL` table. Then, I used the command `WHERE "Landing _Outcome" LIKE 'Success%' AND (Date BETWEEN '2010-06-04' AND '2017-03-20') ORDER BY Date DESC;` to display only the landing outcomes between the date 2010-06-04 and 2017-03-20, in descending order.

Section 3

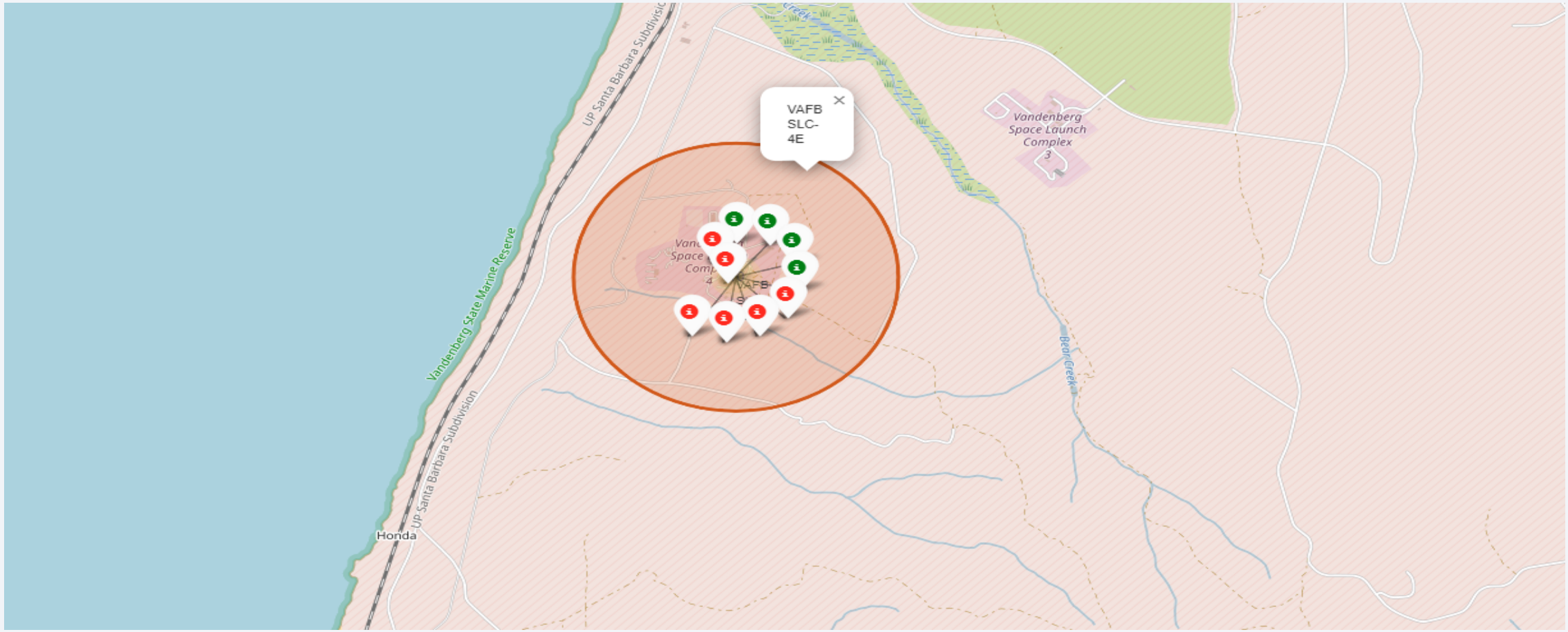
# Launch Sites Proximities Analysis

# Mark all the launch sites on a global map.



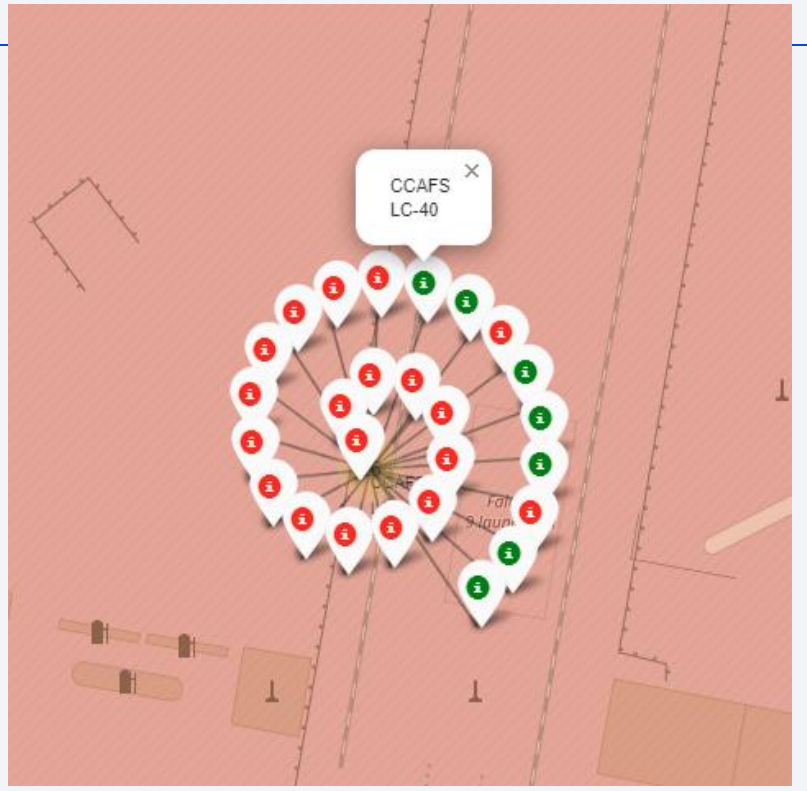

- The launch sites are relatively close to the Equator line, especially the ones in Florida. This appears to be deliberate to take advantage of the Earth's rotational speed around the Equator.
- All the launch sites are in close proximity to the coast. This can be for safety reasons, in case of failed launches and explosions. The debris will be in the ocean, instead of populated areas.

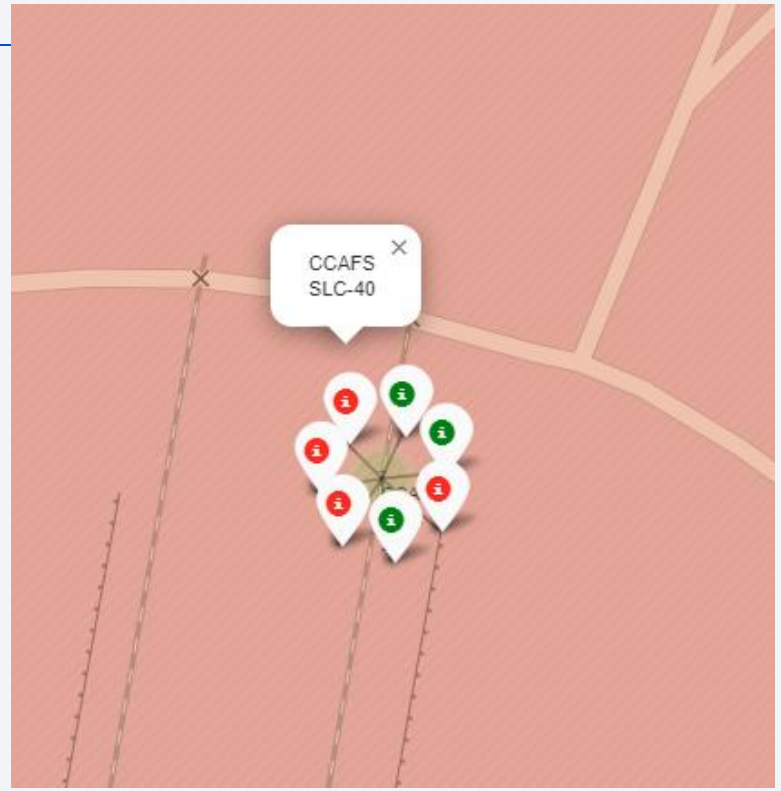
# Color-labeled launch outcomes for each launch site on the map.




- The VAFB SLC-4E launch site in California has a low success rate of 40% (i.e. 4 successes out of 10 launches). This is lower than the success rate for all the launch sites in Florida, except for the CCAFS LC-40 launch site with a very low success rate of 26.92% (i.e. 7 successes out of 26 launches).

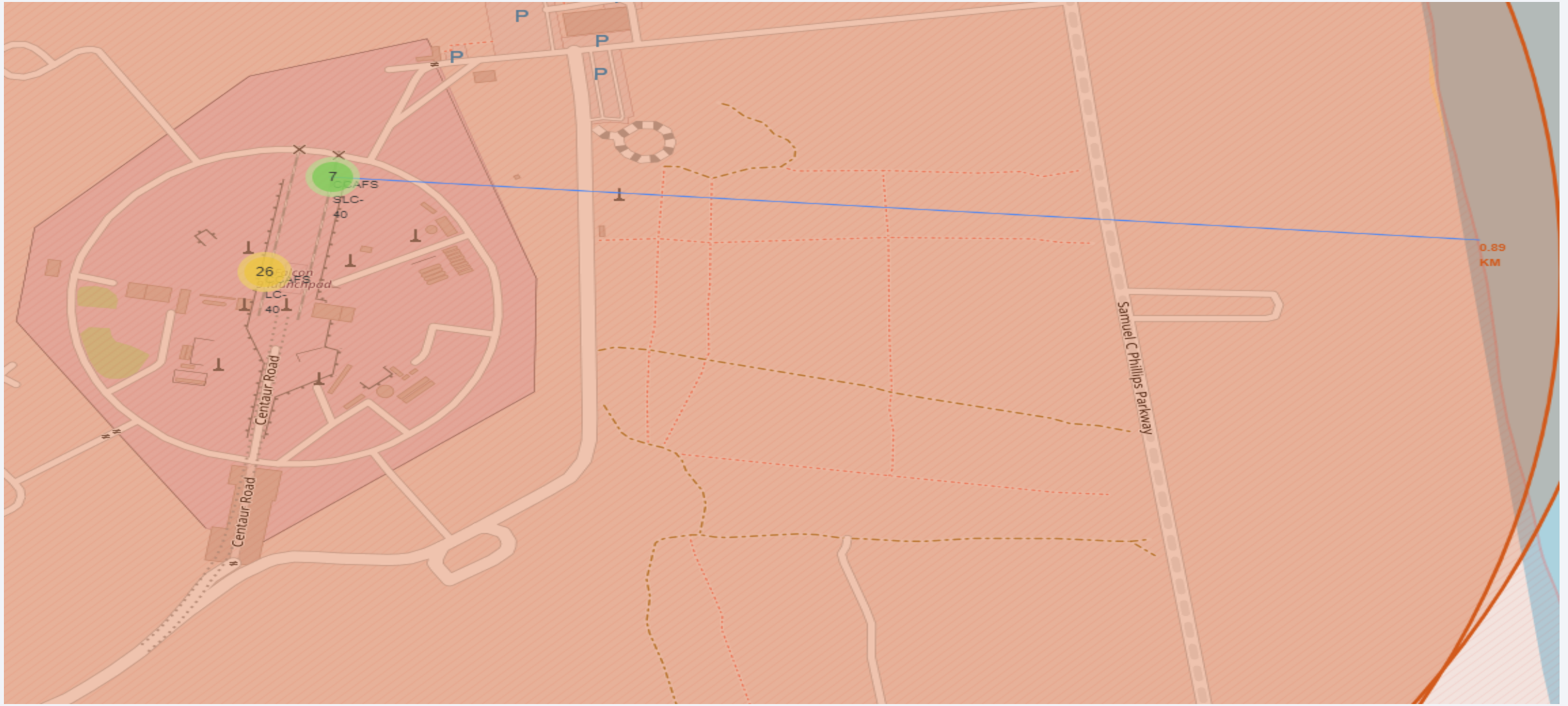
# Color-labeled launch outcomes for each launch site on the map.

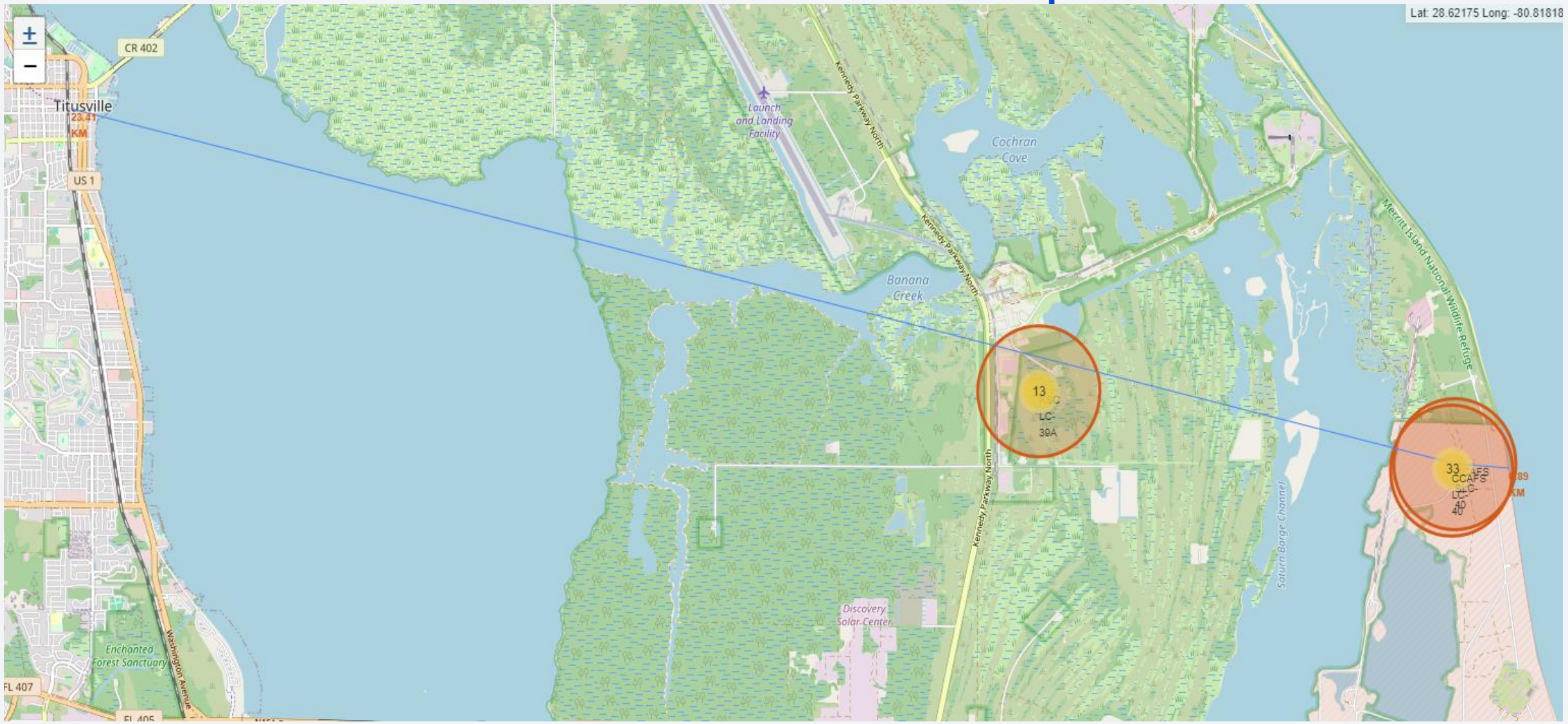

- The KSC LC-39A launch site has a high success rate of 76.92% (i.e. 10 successes out of 13 launches); the CCAFS LC-40 launch site has a very low success rate of 26.92% (i.e. 7 successes out of 26 launches); and the CCAFS SLC-40 launch site has a low success rate of 42.86% (i.e. 3 successes out of 7 launches). The launch sites are all in Florida.

# Distances between a launch site to its proximities.

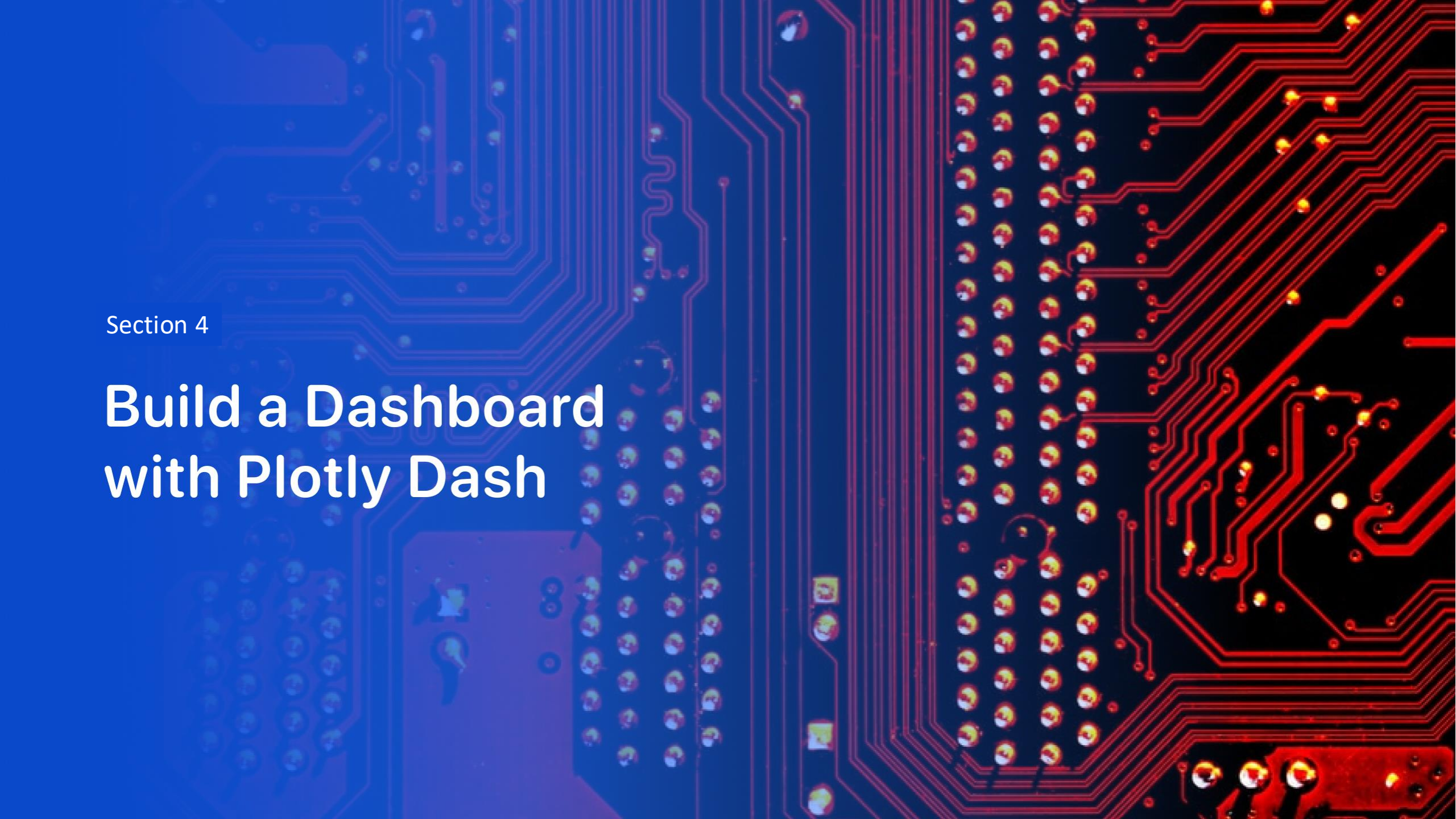



- The distance from the CCAFS LC-40 launch site to the coastline is 0.89 kilometre.

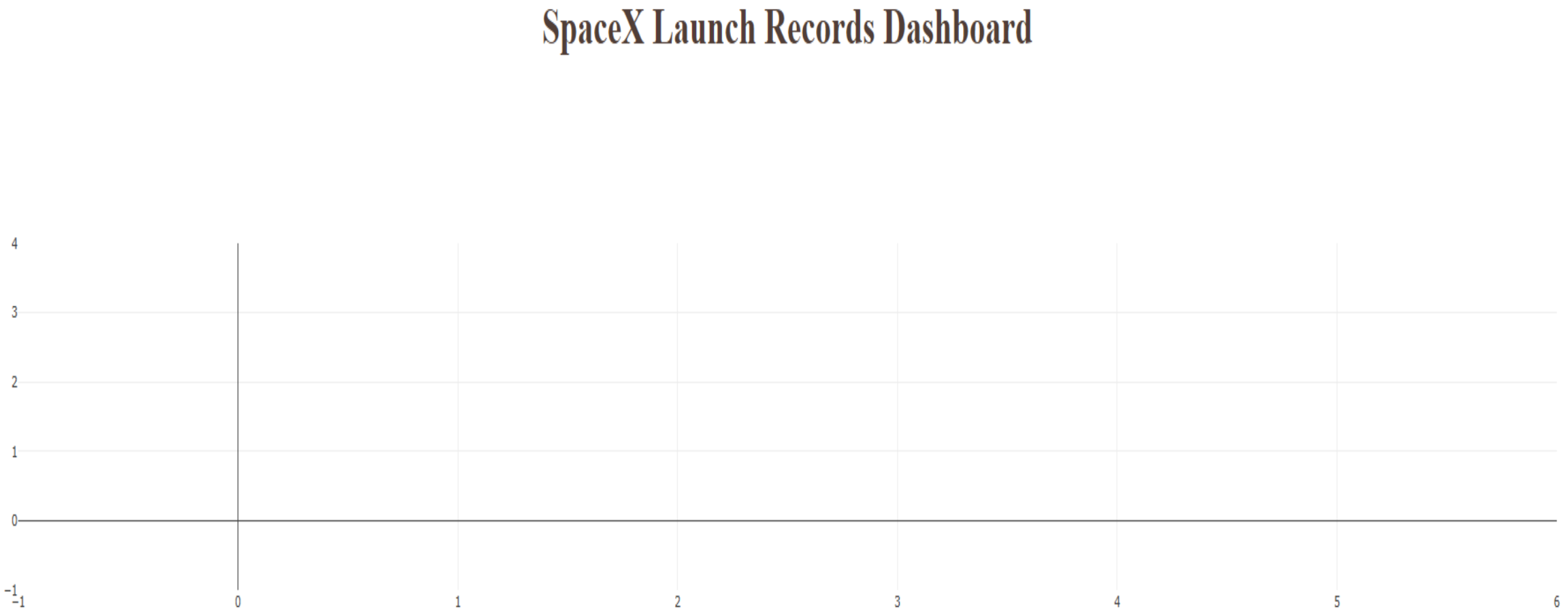
# Distances between a launch site to its proximities.



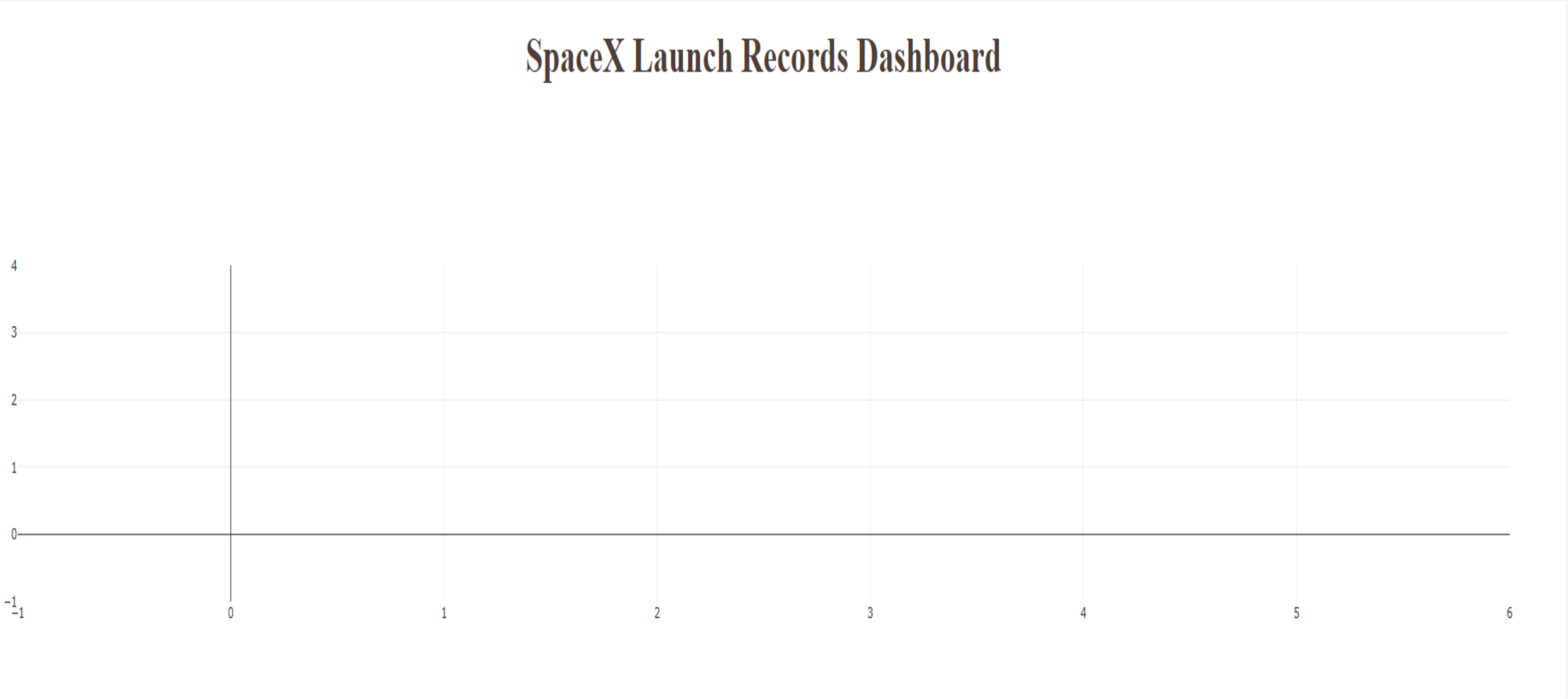

- The distance from the CCAFS LC-40 launch site to Titusville, a nearby city is 23.41 kilometres.


Section 4
Build a Dashboard with Plotly Dash

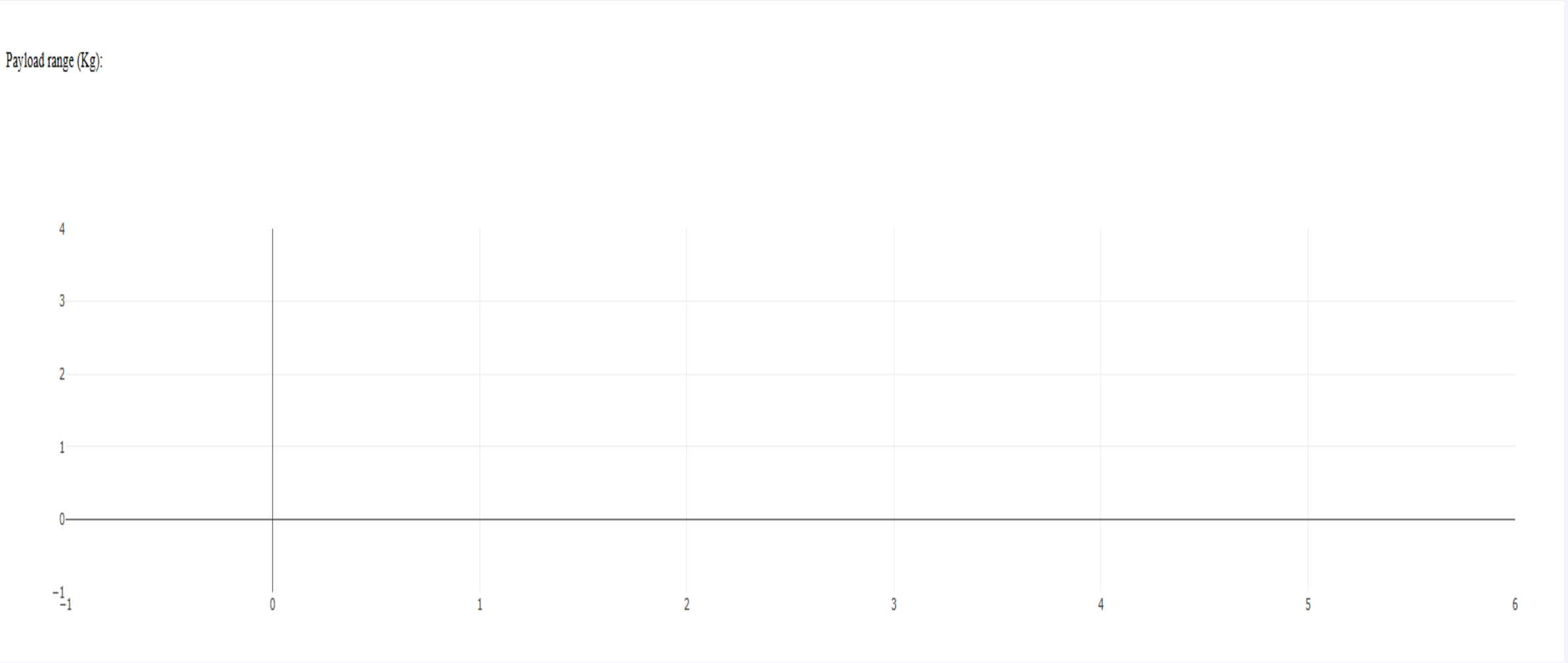
# Pie chart of launch success count for all sites.



# Pie chart for the launch site with highest launch success ratio.



# <Payload vs. Launch Outcome scatter plot for all sites>



Section 5

# Predictive Analysis (Classification)

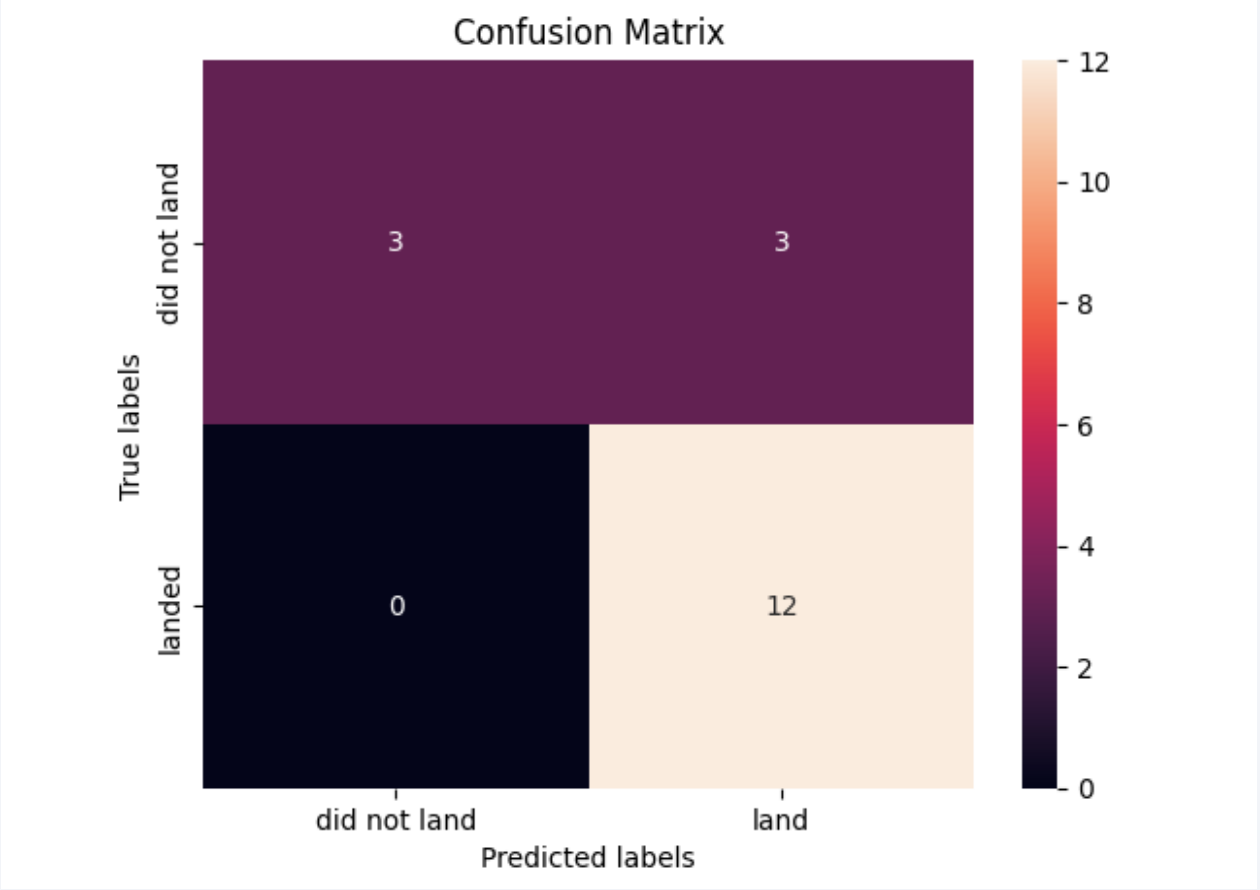
# Classification Accuracy

- The table below shows the Test Data Accuracy score for the Logistic Regression, Support Vector Machine (SVM), Decision Trees and K Nearest Neighbors (KNN) models. All the models have the same accuracy of 0.833333 on the test data i.e. they perform equally.

[40]:

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

# Confusion Matrix




- All the models had the same confusion matrices. These matrices equally defined all the different classes, with the major problem being the false positives expressed for all of them.

# Conclusions

- All the launch sites have different success rates. As the flight number increases in each of the 3 launch sites, the success rate increases too. The success rate for the VAFB SLC 4E Launch Site is 100% after the 50th flight. For the KSC LC 39A and CCAFS SLC 40 Launch Sites, they each achieve a 100% success rate after the 80th flight.
- From the Payload Mass Vs. Launch Site scatter point chart, you will find that for the VAFB-SLC Launch Site, there are no rockets launched for heavy payload mass (greater than 10,000 kilogrammes).
- The orbits ES-L1, GEO, HEO and SSO have the highest success rates of 100%. The VLEO, LEO, MEO, PO, ISS and GTO orbits have success rates ranging from just above 80% to around 50%. The SO orbit has a 0% success rate.
- In the LEO orbit, success seems to be related to the number of flights. Conversely, in the GTO orbit, there appears to be no relationship between flight number and success.

# Conclusions

- With heavy payloads the successful landing or positive landing rate are more for the Polar, LEO and ISS orbits. However, for the GTO orbit, it is difficult to distinguish between successful and unsuccessful landings as both outcomes are present.
- Finally, the success rate of the launches kept increasing from 2013 till 2020.

# Appendix

- This is the GitHub URL of the completed Data Science Capstone Project https://github.com/SamuelOlujuwon/SpaceX-Falcon-9-First-Stage-Landing-Prediction

# Thank you!