IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Name : Avik Das

Date : 13-08-2025

GitHub Link : https://github.com/Avik-Das-567/IBM-Applied-Data-Science-Capstone

Outline

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

Executive Summary

Summary of methodologies :

- Data Collection API
- Data Collection with Web Scraping
- Data Wrangling
- Exploratory Data Analysis (EDA) with SQL
- Exploratory Data Analysis with Visualization
- Interactive Visual Analytics with Folium
- Build an Interactive Dashboard with Plotly Dash
- Predictive Analysis (Classification) with Machine Learning

Summary of all results :

- Data Collection Results
- EDA Results
- Interactive Visual Analytics Results
- Predictive Analysis Result

Introduction

Project background and context :

- In this capstone, we will predict if the Falcon 9 first stage will land successfully. SpaceX 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 SpaceX 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 SpaceX for a rocket launch.

Problems we want to find answers :

- Using Exploratory Data Analysis (EDA), we will collect data on the Falcon 9 first-stage landings. We will use a RESTful API and web scraping. We will also convert the data into a dataframe and then perform some data wrangling.
- Creating Interactive Visual Analytics & Dashboard : We will build a dashboard to analyze launch records interactively with Plotly Dash. We will then build an interactive map to analyze the launch site proximity with Folium.
- Next, we'll perform Predictive Analysis (Classification) : We will use Machine Learning to determine if the first stage of Falcon 9 will land successfully. We will split our data into training data and test data to find the best Hyperparameter for SVM, Classification Trees, and Logistic Regression. Then find the method that performs best using test data.

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - Data Collection API
  - Data Collection with Web Scraping
- Perform data wrangling
  - The data processing involved data wrangling and Exploratory Data Analysis (EDA)
- 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 via SpaceX API

Data was collected by **requesting SpaceX launch data from an API**.

Key Data Collection Steps:

- **Initial API Request:** A GET request was made to the SpaceX API to get historical launch data.
- **JSON to DataFrame:** The JSON response was converted into a Pandas DataFrame.
- **Preliminary Wrangling:** Selected key features, extracted single values from lists in cores and payloads, and filtered data up to November 13, 2020.
- **Data Enrichment:** Used IDs from rocket, payloads, launchpad, and cores to make additional API requests to gather more detailed information (e.g., booster name, launch site details, payload mass/orbit, core landing outcomes).
- **Dataset Construction:** All collected data was combined into a new Pandas DataFrame.
- **Filtering:** The DataFrame was filtered to include only Falcon 9 launches, and FlightNumber was reset.
- **Missing Values:** Missing PayloadMass values were replaced with the column's mean.
- **Export:** The cleaned dataset was saved as a CSV file.

Data Collection – SpaceX API

GitHub URL of the completed SpaceX API calls notebook :

https://github.com/Avik-Das-567/IBM-Applied-Data-Science-Capstone/blob/main/Module%201%20-%20Introduction/1.%20Data%20Collection%20API%20Lab/data-collection-api-lab.ipynb

Flowchart :

- **Objective**: Predict Falcon 9 landing success.
- **Initial Data Get**: GET SpaceX API launch data, confirm 200 status code, convert JSON to DataFrame.
- **Initial Cleaning**: Filter for single cores/payloads; restrict dates to on or before 2020-11-13.
- **Detailed Extraction**: Use helper functions to re-query API with IDs (rocket, launchpad, payloads, cores) for full details (e.g., booster name, landing outcome).
- **Final Dataset**: Combine extracted data into a DataFrame and filter to include only Falcon 9 launches.
- **Missing Values**: Replace np.nan in PayloadMass with its mean; retain None in LandingPad.
- **Export**: Save processed data to dataset_part_1.csv.

Data Collection via Web Scraping

Data sets were collected via **web scraping** Falcon 9 historical launch records from a Wikipedia page. The process leveraged Python libraries, notably **BeautifulSoup** for HTML parsing and **requests** for HTTP operations.

Data collection steps are as follows:

- **HTTP GET Request** to retrieve HTML of the Wikipedia page.
- **HTML Parsing** using BeautifulSoup to create a parseable object.
- **Table Identification:** The third HTML table on the page, containing launch records, was targeted.
- **Column Name Extraction** from the table's header elements (<th>).
- **Data Dictionary Initialization** with extracted column names and additional fields ('Version Booster', 'Booster landing', 'Date', 'Time').
- **Row-by-Row Data Extraction:** Each table row (<tr>) representing a launch was iterated, and specific data points (e.g., Flight No., Date, Time, Payload, Booster Landing status) were extracted and populated into the dictionary.
- **Pandas DataFrame Creation** from the populated dictionary.
- **CSV Export:** The DataFrame was saved as spacex_web_scraped.csv.

Data Collection – Web Scraping

GitHub URL of the completed web scraping notebook :

https://github.com/Avik-Das-567/IBM-Applied-Data-Science-Capstone/blob/main/Module%201%20-%20Introduction/2.%20Data%20Collection%20with%20Web%20Scraping%20lab/data-collection-with-web-scraping.ipynb

Flowchart :

- Importing necessary packages (requests, BeautifulSoup, pandas, etc.) and defining helper functions for data extraction.
- Requesting the HTML page from the specified URL.
- Parsing the HTML content to create a BeautifulSoup object.
- Identifying the target HTML table (the third one) containing the launch records.
- Extracting and cleaning column names from the table header.
- Iterating through table rows to extract data for each launch record and populating a dictionary.
- Converting the dictionary into a Pandas DataFrame.
- Exporting the DataFrame to a CSV file.

Data Wrangling

The data wrangling process :

1. **Data Loading & Initial Analysis:** The Space X dataset was loaded, and initial inspections were performed to identify missing values and column data types.

2. **Exploratory Data Analysis (EDA):** This involved analyzing patterns such as the number of launches per LaunchSite, and the occurrences of different Orbit types and Outcome types.

3. **Outcome Conversion to Training Labels:** The primary task was converting mission outcomes into numerical labels.
    - "Bad outcomes" (e.g., False ASDS, None None, False Ocean, False RTLS, None ASDS) were identified as unsuccessful landings.
    - A new Class column was created, labeled 0 for unsuccessful landings and 1 for successful landings.

4. **Success Rate Calculation:** The overall success rate was determined from the new Class column.

5. **Data Export:** The wrangled dataset was then exported as dataset_part_2.csv for subsequent analysis.

Data Wrangling

GitHub URL of the Data Wrangling Notebook :

https://github.com/Avik-Das-567/IBM-Applied-Data-Science-Capstone/blob/main/Module%201%20-%20Introduction/3.%20Data%20Wrangling/data-wrangling.ipynb

Flowchart

Load Data

↓

Initial Inspection (Missing Values, Data Types)

↓

EDA (Analyze LaunchSite, Orbit, Outcome Counts)

↓

Define Bad Outcomes

↓

Convert Outcome to Class (0: Unsuccessful, 1: Successful)

↓

Calculate Success Rate

↓

Export Data

EDA with Data Visualization

The Exploratory Data Analysis (EDA) focused on predicting Falcon 9 first stage landing success, a key factor for reducing launch costs. Using Pandas, Matplotlib, and Seaborn, several charts were plotted to understand data relationships:

- **Flight Number vs. Payload Mass with Outcome Overlay:** An initial scatter plot using sns.catplot was used to observe how FlightNumber and PayloadMass affect launch outcome, revealing increased success with higher flight numbers and even for massive payloads.
- **TASK 1: Flight Number vs. Launch Site:** A scatter point chart (sns.catplot with kind="strip") visualized the relationship between FlightNumber and LaunchSite, with Class (success/failure) as hue, to drill down into detailed launch records per site.
- **TASK 2: Payload Mass vs. Launch Site:** A scatter point chart (sns.scatterplot) was plotted to observe the relationship between PayloadMass and LaunchSite, colored by Class, noting, for example, that VAFB-SLC did not launch heavy payloads (over 10000 kg).
- **TASK 3: Success Rate of each Orbit Type:** A bar chart (sns.barplot) displayed the average Success Rate for each Orbit type to identify which orbits have the highest success rates.
- **TASK 4: Flight Number vs. Orbit type:** A scatter point chart (sns.scatterplot) showed the relationship between FlightNumber and Orbit type, using Class as hue, to see if success varied with flight number within specific orbits (e.g., LEO showed a relationship, GTO did not).
- **TASK 5: Payload Mass vs. Orbit type:** Another scatter point chart (sns.scatterplot) revealed the relationship between PayloadMass and Orbit type, highlighting that heavy payloads showed higher success rates for Polar, LEO, and ISS orbits.
- **TASK 6: Launch Success Yearly Trend:** A line chart (sns.lineplot) was used to visualize the average launch Success Rate over Year, showing a consistent increase in success from 2013 to 2020.

EDA with Data Visualization

GitHub URL of my completed EDA with data visualization notebook:

https://github.com/Avik-Das-567/IBM-Applied-Data-Science-Capstone/blob/main/Module%202%20-%20Exploratory%20Data%20Analysis/2.%20EDA%20with%20Visualization%20Lab/eda-with-visualization.ipynb

EDA with SQL

Summary of the SQL queries performed:

- Identified unique launch sites: The query found four distinct launch sites - 'CCAFS LC-40', 'VAFB SLC-4E', 'KSC LC-39A', and 'CCAFS SLC-40'.
- Retrieved 5 records for launch sites starting with ‘CCA’.
- Calculated total payload mass for 'NASA (CRS)' customers: The sum was 48213 KG.
- Determined average payload mass for 'F9 v1.1' booster version: The average was approximately 2534.67 KG.
- Found the earliest date of a successful ground pad landing: This was '2015-12-22'.
- Listed booster versions with successful drone ship landings and payload mass between 4000 KG and 6000 KG.
- Counted successful and failure mission outcomes.
- Identified booster versions that carried the maximum payload mass.
- Filtered failure landing outcomes on drone ships in 2015, displaying month, booster version, and launch site.
- Ranked landing outcome counts in descending order for dates between '2010-06-04' and '2017-03-20'.

EDA with SQL

GitHub URL of my completed EDA with SQL notebook:

https://github.com/Avik-Das-567/IBM-Applied-Data-Science-Capstone/blob/main/Module%202%20-%20Exploratory%20Data%20Analysis/1.%20EDA%20with%20SQL/eda-with-sql.ipynb

Build an Interactive Map with Folium

In this Lab, the following map objects were created and added to a Folium map:

- **Initial Map:** A **folium.Map** was created, centered at NASA Johnson Space Center, with a yellow folium.map.Marker and folium.Circle to set the initial geographic context.
- **Launch Sites:** For each launch site, a **folium.Circle** (radius 1000m) with a popup and a folium.Marker with a label were added. These were included to visually mark all launch sites on the map using their coordinates, making their locations intuitive and allowing users to assess proximity to the Equator or coast.
- **Launch Outcomes:** Individual **folium.Marker** objects were created for each launch record, colored green for successful launches (class=1) and red for failed launches (class=0). These were added to a MarkerCluster to simplify the map, especially where multiple launches occurred from the same coordinates. This allows for visualizing and identifying launch sites with high success rates.
- **Proximity Analysis:**
  - A **MousePosition** plugin was added to enable users to easily find coordinates for points of interest by hovering their mouse.
  - **folium.Marker** objects were created at specific proximity points (coastline, railway, highway, city) displaying the calculated distance (in KM) from a launch site.
  - **folium.PolyLine** objects were drawn between the launch site and each of these proximity points (e.g., blue for coastline, gray dashed for railway, green for highway, red dashed for city). These objects were crucial for exploring and analyzing the geographical proximities of launch sites, helping to discover factors influencing their optimal location.

Build an Interactive Map with Folium

GitHub URL of my completed interactive map with Folium map :

https://github.com/Avik-Das-567/IBM-Applied-Data-Science-Capstone/blob/main/Module%203%20-%20Interactive%20Visual%20Analytics%20%26%20Dashboards/1.%20Interactive%20Visual%20Analytics%20with%20Folium%20lab/interactive-visual-analytics-with-folium.ipynb

Build a Dashboard with Plotly Dash

The dashboard features the following plots/graphs and interactions:

- **Pie Chart (success-pie-chart)** :
  - **Description**: Shows launch success counts (total by site, or success/failure for a specific site).
  - **Why Added**: To visualize launch success rates and identify top-performing sites.
  - **Interaction**: Controlled by the Launch Site Dropdown.
- **Launch Site Dropdown (site-dropdown)** :
  - **Description**: Allows selection of "All Sites" or a specific launch site.
  - **Why Added**: Enables site-specific or aggregated data analysis for success rates.
  - **Interaction**: Inputs to both the Pie Chart and Scatter Plot.
- **Range Slider (payload-slider)** :
  - **Description**: Selects a payload mass range (0-10000 Kg).
  - **Why Added**: To identify patterns between payload mass and mission outcomes.
  - **Interaction**: Filters data for the Scatter Plot.
- **Scatter Plot (success-payload-scatter-chart)** :
  - **Description**: Plots 'Payload Mass (kg)' vs. 'class' (outcome), colored by 'Booster Version Category'.
  - **Why Added**: To observe payload-outcome correlation and compare Booster Version success rates.
  - **Interaction**: Controlled by both the Launch Site Dropdown and Payload Range Slider.

Build a Dashboard with Plotly Dash

GitHub URL of my completed Plotly Dash lab :

https://github.com/Avik-Das-567/IBM-Applied-Data-Science-Capstone/blob/main/Module%203%20-%20Interactive%20Visual%20Analytics%20%26%20Dashboards/2.%20Interactive%20Dashboard%20with%20Plotly%20Dash/spacex_dash_app.py

Predictive Analysis (Classification)

The goal was to build a **machine learning pipeline** to predict Falcon 9 first stage landings to determine launch costs.

The process involved:

- **Data Preparation**:
  - **Defined target variable (Y)** from 'Class' column.
  - **Standardized features (X)** using StandardScaler().
  - **Split data** into 80% training and 20% test sets (X_train, Y_train, X_test, Y_test).
- **Model Building & Evaluation**:
  - Evaluated **Logistic Regression**, **Support Vector Machine (SVM)**, **Decision Tree Classifier**, and **K Nearest Neighbors (KNN).**
  - For each model, **GridSearchCV** with 10-fold cross-validation (**cv=10**) was used on training data to find **best hyperparameters** and validation accuracy.
  - Calculated **test set accuracy** for each tuned model.
  - All four models—Logistic Regression, SVM, Decision Tree, and KNN—achieved a **test set accuracy of 0.8333**.
  - **Confusion matrices** were plotted to analyze errors.
- **Best Performing Model**:
  - **Logistic Regression** was identified as the best performing model with an accuracy of 0.8333, matching the other models' test accuracy.

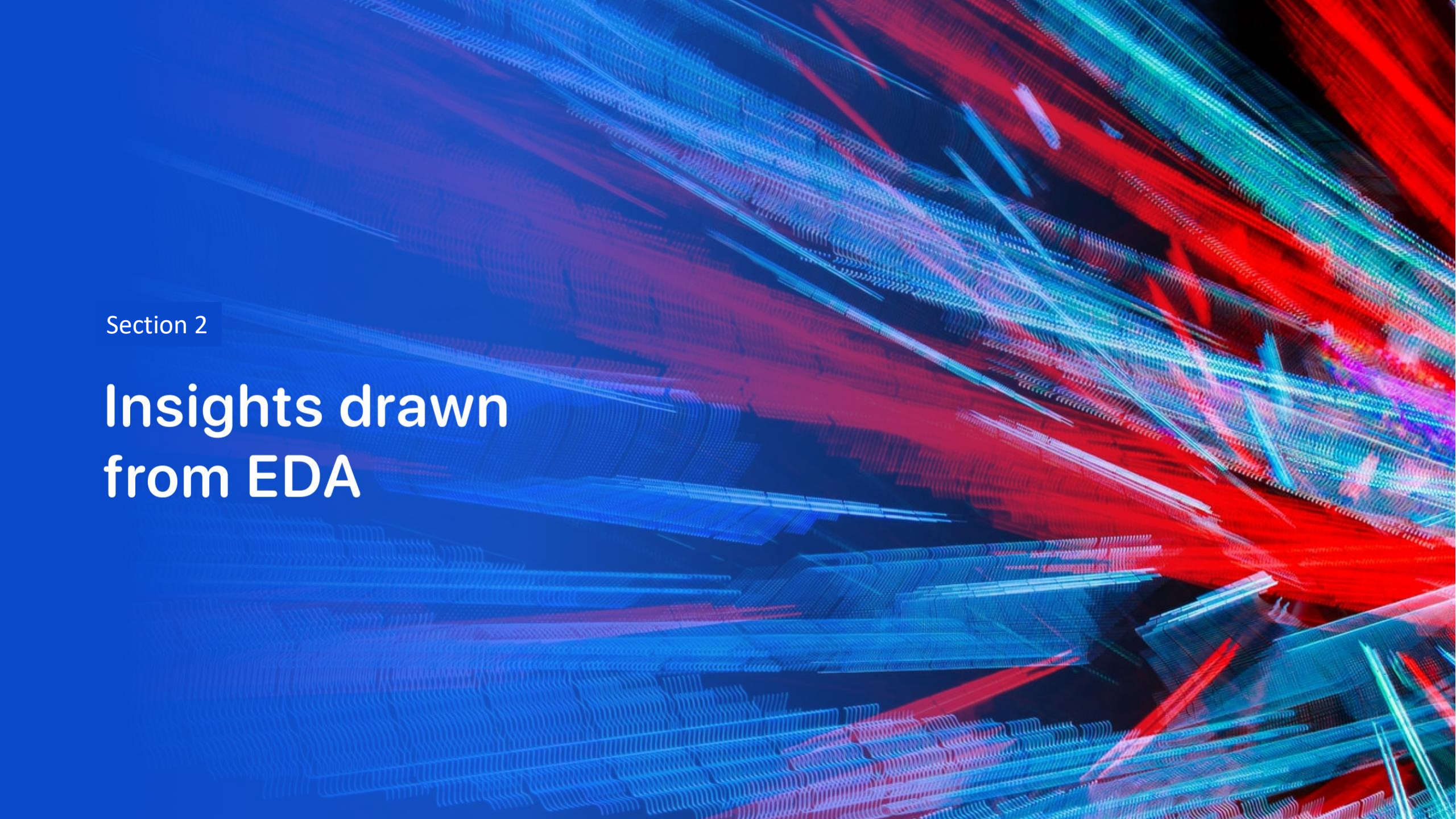
Predictive Analysis (Classification)

GitHub URL of my completed predictive analysis lab:

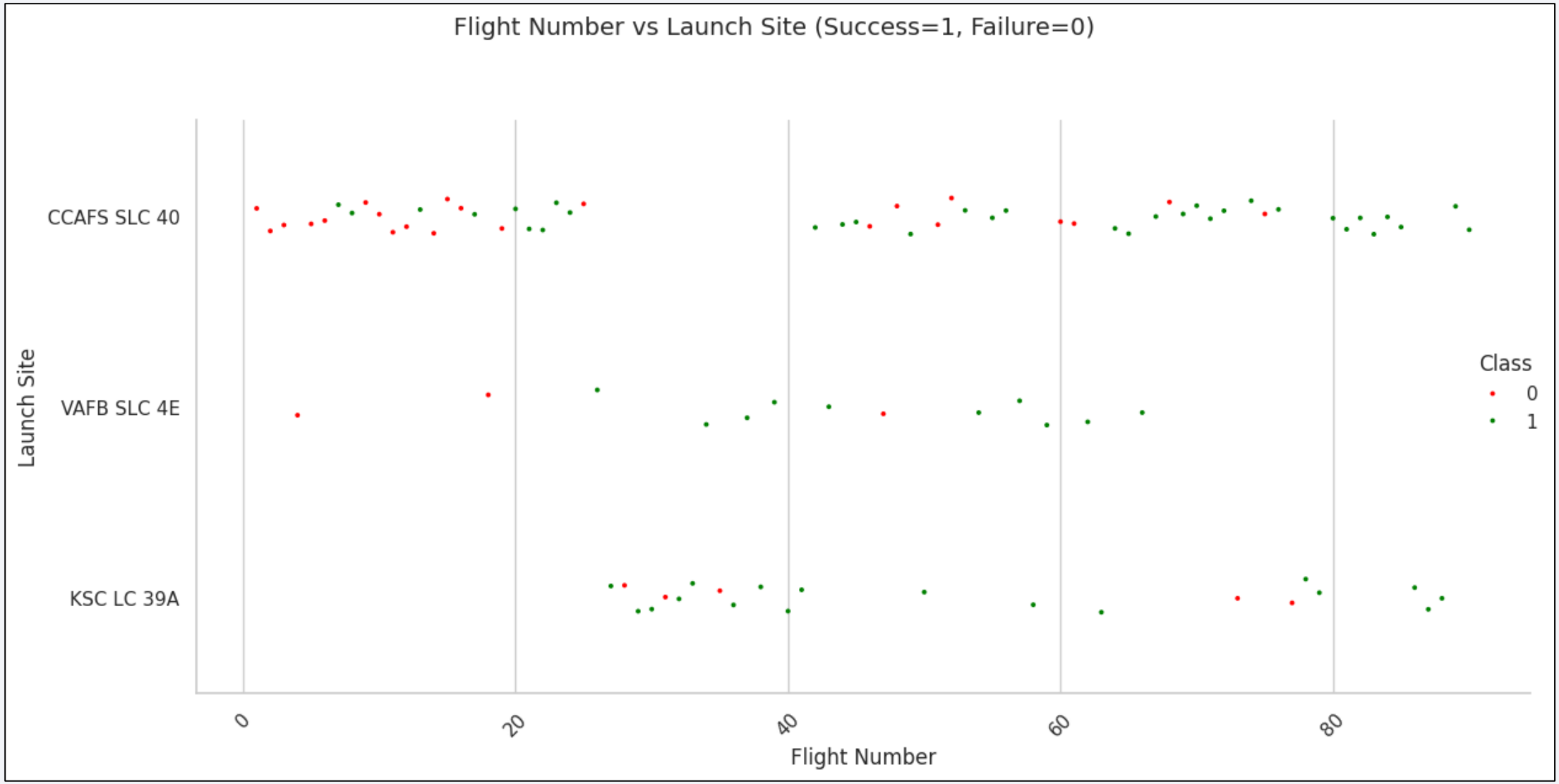
https://github.com/Avik-Das-567/IBM-Applied-Data-Science-Capstone/blob/main/Module%204%20-%20Predictive%20Analysis%20(Classification)/SpaceX-Machine-Learning-Prediction.ipynb

Results

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


Section 2
Insights drawn from EDA

Flight Number vs. Launch Site



Flight Number vs. Launch Site

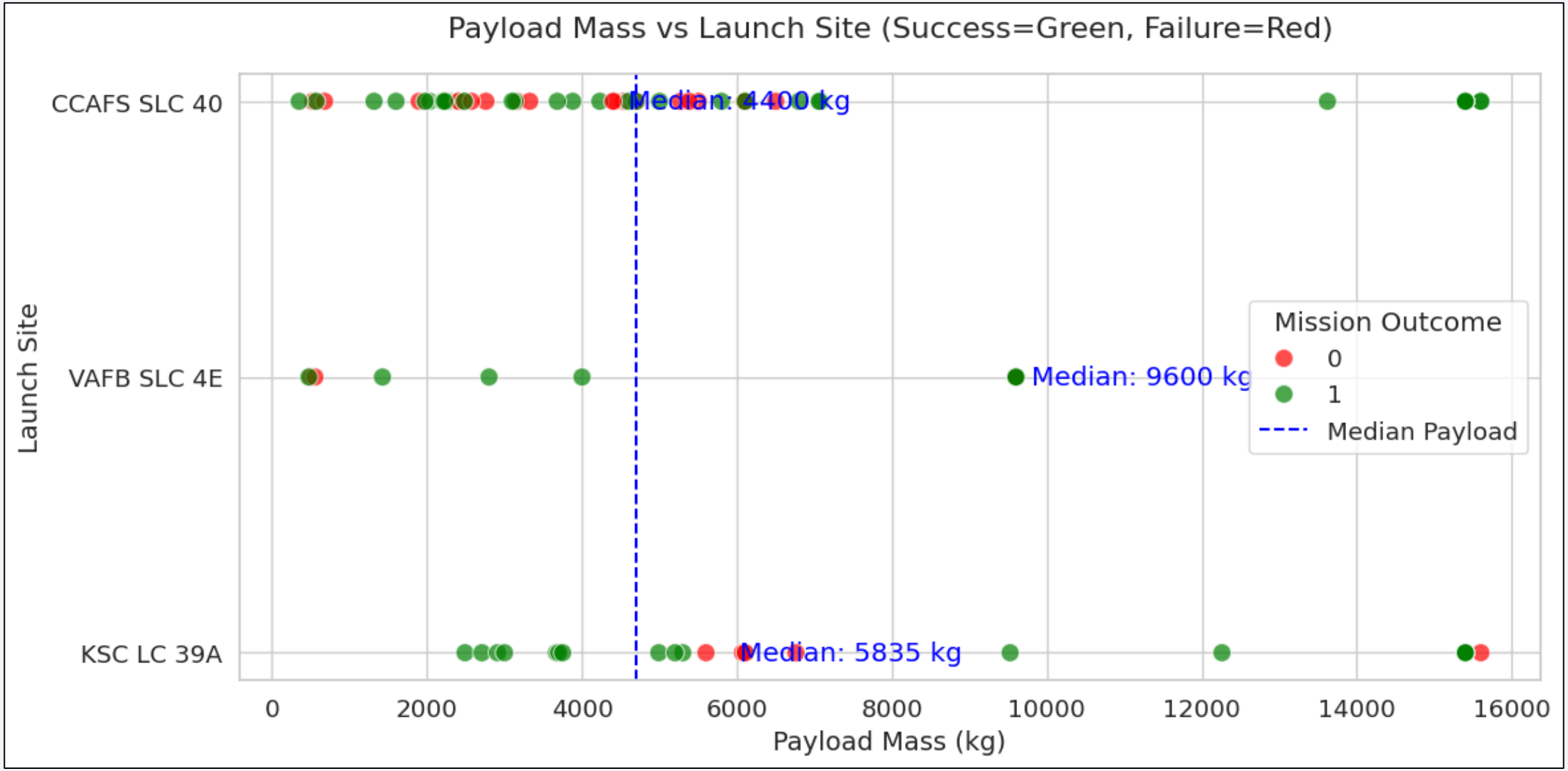
The **Flight Number vs Launch Site** scatter plot shows that:

- **Higher flight numbers** (later launches) tend to have more **successful landings** (green points).
- Some sites, like **CCAFS SLC-40**, show early failures but improved success over time.
- **VAFB SLC-4E** and **KSC LC-39A** have fewer launches, so their trend is less clear.

It basically suggests that experience over time at each site increases landing success.

```
# Set the style for better visuals
sns.set(style="whitegrid")

# Create the categorical plot
g = sns.catplot(
    x="FlightNumber",
    y="LaunchSite",
    hue="Class",
    data=df,
    kind="strip",  # Creates scatter points
    height=6,      # Adjust height
    aspect=2,      # Width to height ratio
    palette={1: "green", 0: "red"},  # Success=green, Failure=red
    s=8            # Point size
)

# Customize the plot
g.set_axis_labels("Flight Number", "Launch Site", fontsize=12)
g.fig.suptitle("Flight Number vs Launch Site (Success=1, Failure=0)", y=1.05, fontsize=14)

# Improve x-axis ticks to show all flight numbers
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Payload vs. Launch Site



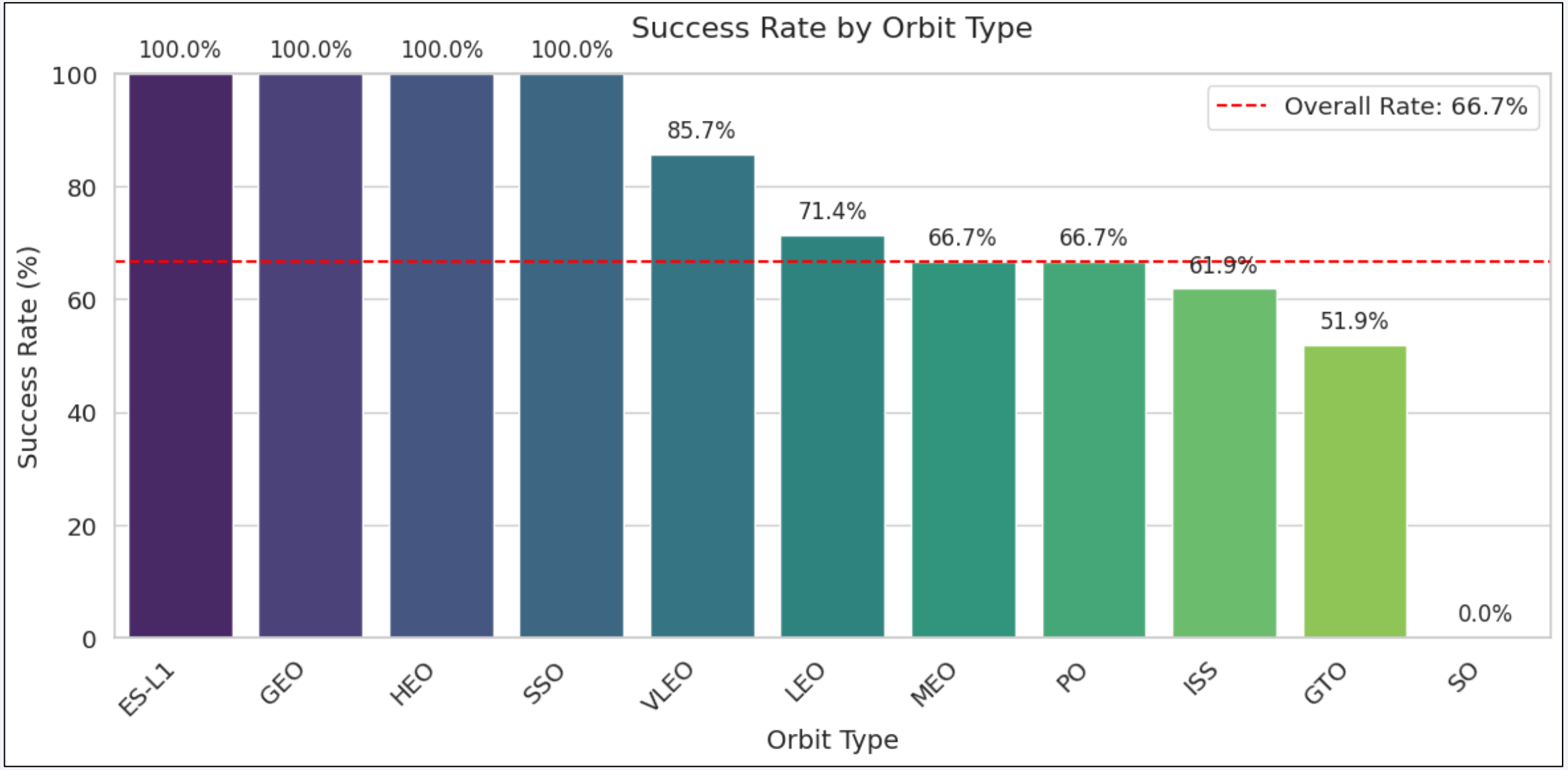
Payload vs. Launch Site

The **Payload Mass vs Launch Site** scatter plot shows that:

- **Heavier payloads** don't always mean failure—some sites still achieve successful landings with large payloads.
- **VAFB SLC-4E** has **no launches** with payloads over ~10,000 kg.
- Success (green) and failure (red) vary by site, but certain sites handle a broader payload range than others.

```
# Set plot style
sns.set(style="whitegrid", font_scale=1.2)

# Create the plot
plt.figure(figsize=(12, 6))
scatter = sns.scatterplot(
    x='PayloadMass',
    y='LaunchSite',
    hue='Class',
    data=df,
    palette={1: "green", 0: "red"},
    s=100,          # Point size
    alpha=0.7       # Transparency
)

# Customize the plot
plt.title("Payload Mass vs Launch Site (Success=Green, Failure=Red)", pad=20, fontsize=16)
plt.xlabel("Payload Mass (kg)", fontsize=14)
plt.ylabel("Launch Site", fontsize=14)
plt.axvline(x=df['PayloadMass'].median(), color='blue', linestyle='--', label='Median Payload')
plt.legend(title='Mission Outcome')

# Add payload mass annotations
for site in df['LaunchSite'].unique():
    median_mass = df[df['LaunchSite']==site]['PayloadMass'].median()
    plt.text(median_mass+200, site, f"Median: {median_mass:.0f} kg",
             va='center', color='blue')

plt.tight_layout()
plt.show()
```

Success Rate vs. Orbit Type



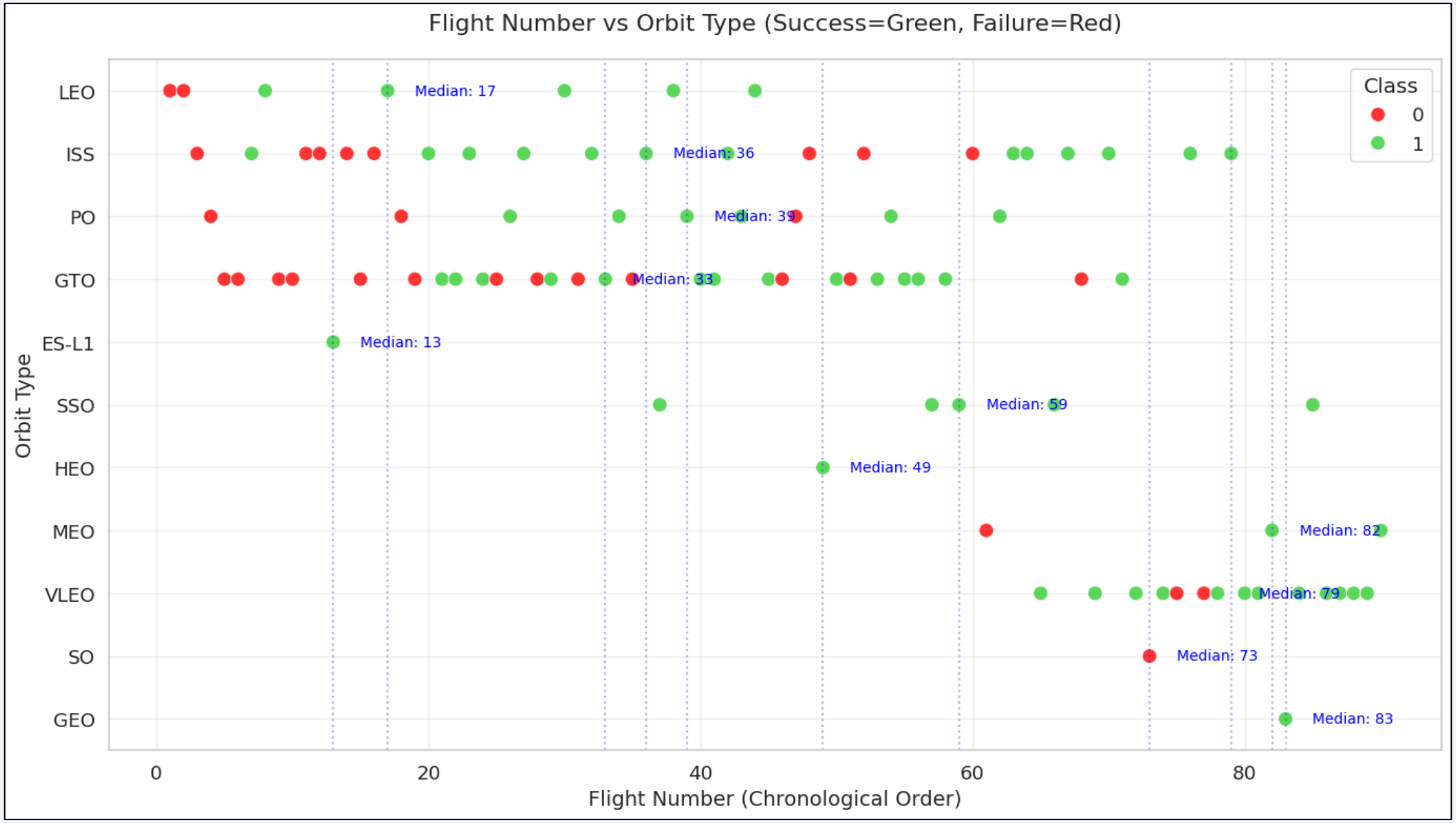
Success Rate vs. Orbit Type

The **Success Rate vs Orbit Type** bar chart shows that:

- Some orbits, like **LEO** and **ISS**, have the **highest success rates**, often near or above the overall average.
- **GTO** and a few other orbits have noticeably **lower success rates**.
- This suggests that the type of orbit targeted impacts landing success, likely due to differences in mission difficulty and payload requirements.

```
success_rates = df.groupby('Orbit')['Class'].mean().sort_values(ascending=False) * 100

# Create the plot
plt.figure(figsize=(12, 6))
barplot = sns.barplot(
    x=success_rates.index,
    y=success_rates.values,
    hue=success_rates.index,  # Explicit hue assignment
    palette='viridis',
    dodge=False,              # Prevent automatic dodging
    legend=False              # Turn off redundant legend
)

# Customize the plot (rest remains the same)
plt.title('Success Rate by Orbit Type', fontsize=16, pad=20)
plt.xlabel('Orbit Type', fontsize=14)
plt.ylabel('Success Rate (%)', fontsize=14)
plt.xticks(rotation=45, ha='right')
plt.ylim(0, 100)

# Add value labels
for i, rate in enumerate(success_rates):
    barplot.text(i, rate + 2, f'{rate:.1f}%',
                 ha='center', va='bottom', fontsize=12)

# Reference line
overall_rate = df['Class'].mean() * 100
plt.axhline(y=overall_rate, color='red', linestyle='--',
            label=f'Overall Rate: {overall_rate:.1f}%')
plt.legend()

plt.tight_layout()
plt.show()
```

Flight Number vs. Orbit Type



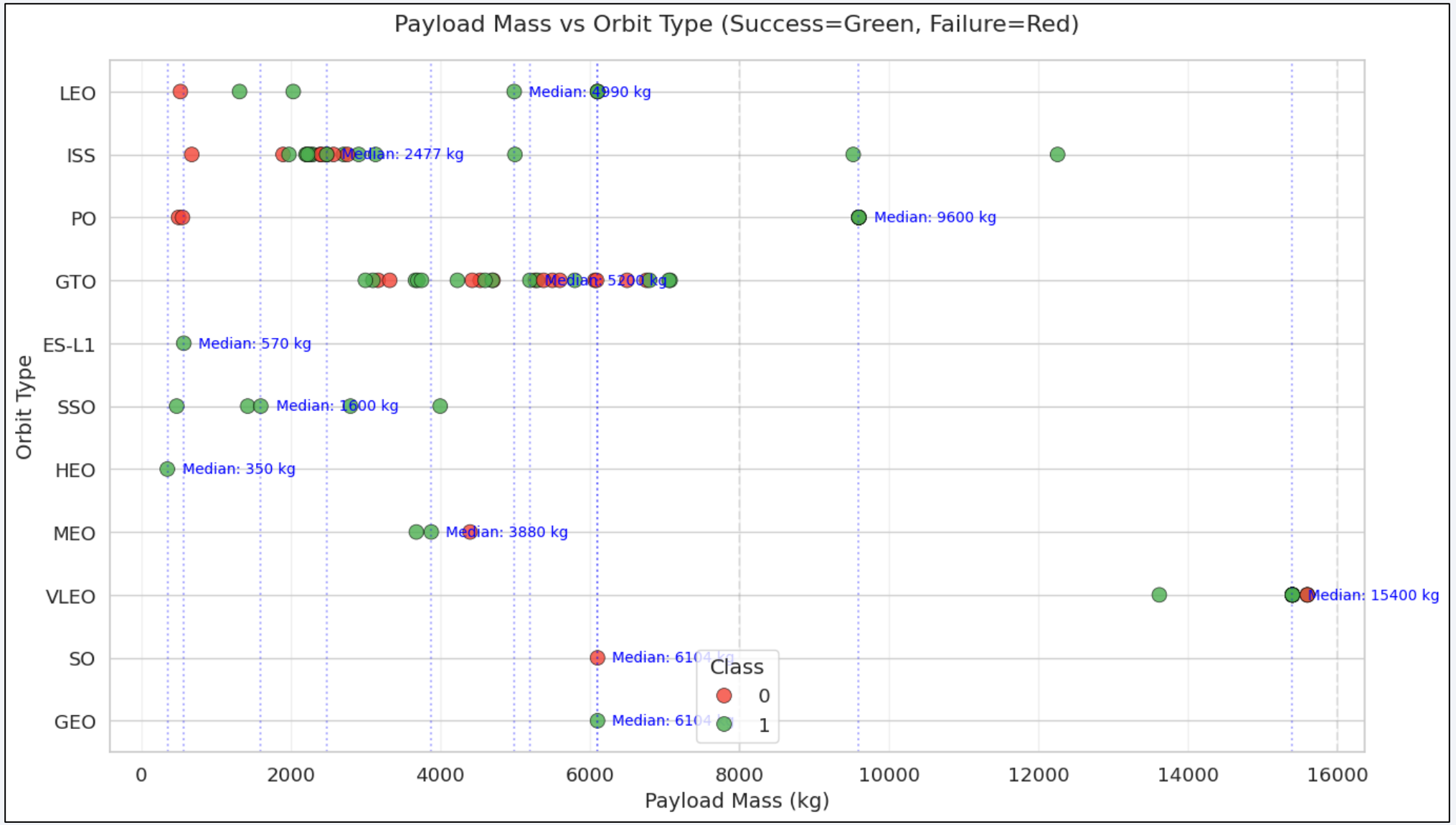
Flight Number vs. Orbit Type

The **Flight Number vs Orbit Type** scatter plot shows that:

- For **LEO** missions, success rates improve with **higher flight numbers** (more recent launches).
- For **GTO** missions, success appears **unrelated** to flight number—both successes and failures are spread across all flights.
- Other orbits have fewer data points, so trends are less distinct.

```
plt.figure(figsize=(14, 8))
scatter = sns.scatterplot(
    x='FlightNumber',
    y='Orbit',
    hue='Class',
    data=df,
    palette={1: "limegreen", 0: "red"},
    s=100,
    alpha=0.8
)

# Customize the plot
plt.title('Flight Number vs Orbit Type (Success=Green, Failure=Red)', fontsize=16, pad=20)
plt.xlabel('Flight Number (Chronological Order)', fontsize=14)
plt.ylabel('Orbit Type', fontsize=14)
plt.grid(True, alpha=0.3)

# Add median flight number markers for each orbit
for orbit in df['Orbit'].unique():
    median_flight = df[df['Orbit'] == orbit]['FlightNumber'].median()
    plt.axvline(x=median_flight, color='blue', linestyle=':', alpha=0.3)
    plt.text(median_flight+2, orbit, f'Median: {int(median_flight)}',
             va='center', color='blue', fontsize=10)

plt.tight_layout()
plt.show()
```

Payload vs. Orbit Type



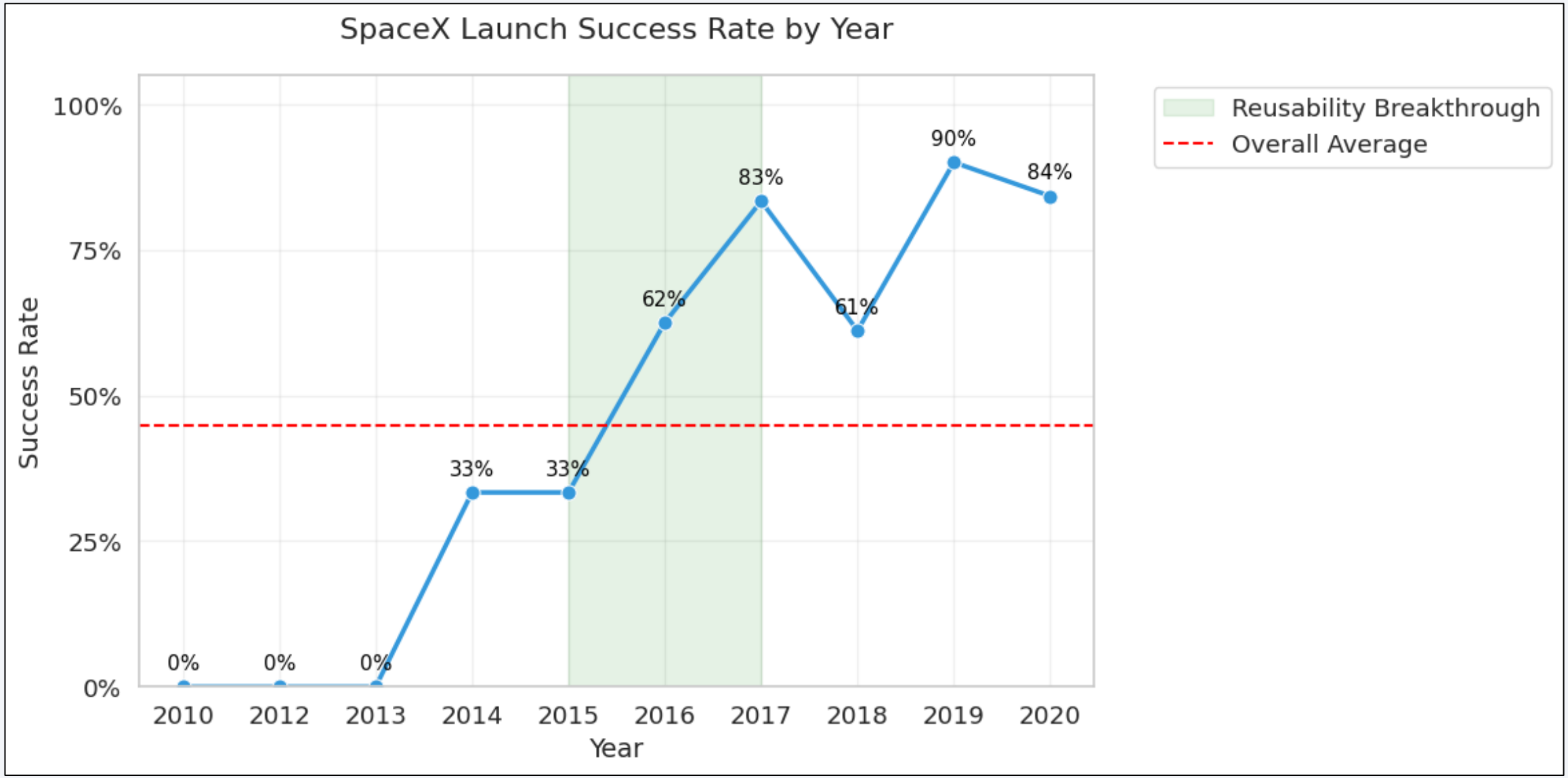
Payload vs. Orbit Type

The **Payload Mass vs Orbit Type** scatter plot shows that:

- **Polar, LEO, and ISS** orbits handle heavy payloads while still achieving many successful landings.
- **GTO** missions show both successes and failures across similar payload ranges, making patterns less clear.
- Median payload masses vary by orbit, indicating different mission profiles and requirements.

```
plt.figure(figsize=(14, 8))
scatter = sns.scatterplot(
    x='PayloadMass',
    y='Orbit',
    hue='Class',
    data=df,
    palette={1: "#4CAF50", 0: "#F44336"},  # Green=Success, Red=Failure
    s=100,
    alpha=0.8,
    edgecolor='black',
    linewidth=0.5
)

# Customize the plot
plt.title('Payload Mass vs Orbit Type (Success=Green, Failure=Red)', fontsize=16, pad=20)
plt.xlabel('Payload Mass (kg)', fontsize=14)
plt.ylabel('Orbit Type', fontsize=14)
plt.grid(True, axis='x', alpha=0.3)

# Add median payload lines for each orbit
for orbit in df['Orbit'].unique():
    median_mass = df[df['Orbit'] == orbit]['PayloadMass'].median()
    plt.axvline(x=median_mass, color='blue', linestyle=':', alpha=0.3)
    plt.text(median_mass + 200, orbit,
             f'Median: {int(median_mass)} kg',
             va='center', color='blue', fontsize=10)

# Add reference lines for payload ranges
plt.axvline(x=8000, color='gray', linestyle='--', alpha=0.2)
plt.axvline(x=16000, color='gray', linestyle='--', alpha=0.2)

plt.tight_layout()
plt.show()
```

Launch Success Yearly Trend



Launch Success Yearly Trend

The **Launch Success Yearly Trend** line chart shows that:

- Success rates were **low before 2013**, then began to **rise steadily**.
- From **2015 to 2017**, there's a marked improvement, coinciding with SpaceX's **reusability breakthroughs**.
- By **2020**, success rates approached **100%**, showing strong reliability gains over time.

```
year = []
for i in df["Date"]:
    year.append(i.split("-")[0])
df['Year'] = year

# Calculate yearly success rates
yearly_success = df.groupby('Year')['Class'].mean().reset_index()

# Create the line plot
plt.figure(figsize=(12, 6))
line = sns.lineplot(
    x='Year',
    y='Class',
    data=yearly_success,
    marker='o',
    markersize=8,
    linewidth=2.5,
    color='#3498db'
)

# Customize the plot
plt.title('SpaceX Launch Success Rate by Year', fontsize=16, pad=20)
plt.xlabel('Year', fontsize=14)
plt.ylabel('Success Rate', fontsize=14)
plt.ylim(0, 1.05)
plt.yticks([0, 0.25, 0.5, 0.75, 1.0], ['0%', '25%', '50%', '75%', '100%'])

# Add value annotations
for index, row in yearly_success.iterrows():
    line.text(row['Year'], row['Class'] + 0.03,
              f"{row['Class']:.0%}",
              ha='center', color='black', fontsize=11)

# Highlight key milestones
plt.axvspan('2015', '2017', alpha=0.1, color='green',
            label='Reusability Breakthrough')
plt.axhline(y=yearly_success['Class'].mean(), color='red',
            linestyle='--', label='Overall Average')

plt.legend(bbox_to_anchor=(1.05, 1))
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```

All Launch Site Names

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

```
[19] %sql SELECT DISTINCT "Launch_Site" FROM SPACEXTABLE;
```

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

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

Explanation :

Task 1 - Display Unique Launch Sites

- This task requires fetching and displaying all unique launch sites present in the SPACEXTABLE.
- The purpose is to understand the different locations from which SpaceX missions were launched.

Launch Site Names Begin with 'CCA'

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

Python

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

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

Explanation : **Task 2 - Display Records for 'CCA' Launch Sites**

- This task involves retrieving and displaying five records from the SPACEXTABLE where the "Launch_Site" column begins with the string ‘CCA’.
- This helps in filtering data based on specific launch site prefixes.

Total Payload Mass

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

```
[21] %sql SELECT SUM("PAYLOAD_MASS__KG_") AS TotalPayloadMass FROM SPACEXTABLE WHERE "Customer" LIKE '%NASA (CRS)%';
```

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

TotalPayloadMass
48213

Explanation : **Task 3 - Total Payload Mass for NASA (CRS)**

- The goal of this task is to calculate and display the total payload mass carried by boosters launched for customers whose names include 'NASA (CRS)'.
- This provides an aggregate sum of payload for a specific customer type.

Average Payload Mass by F9 v1.1

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

```
%sql SELECT AVG("PAYLOAD_MASS__KG_") AS AvgPayloadMass FROM SPACEXTABLE WHERE "Booster_Version" LIKE '%F9 v1.1%';
```

[22]

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

AvgPayloadMass
2534.6666666666665

Explanation : **Task 4 - Average Payload Mass for F9 v1.1**

- This task asks for the calculation and display of the average payload mass carried by booster versions identified as 'F9 v1.1'.
- This helps in understanding the typical payload capacity for a specific booster model.

First Successful Ground Landing Date

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

Hint:Use min function

```
%sql SELECT MIN("Date") AS FirstSuccessfulGroundLanding FROM SPACEXTABLE WHERE "Landing_Outcome" LIKE '%Success (ground pad)%';
```

[23]

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

FirstSuccessfulGroundLanding
2015-12-22

Explanation : **Task 5 - First Successful Ground Pad Landing Date**

- This task aims to identify and display the earliest date on which a successful landing outcome on a ground pad was achieved.
- It uses an aggregate function (MIN) to find the minimum date for a specific landing outcome.

Successful Drone Ship Landing with Payload between 4000 and 6000

```
%%sql
SELECT DISTINCT "Booster_Version"
FROM SPACEXTABLE
WHERE "Landing_Outcome" LIKE '%Success (drone ship)%'
AND "PAYLOAD_MASS__KG_" > 4000
AND "PAYLOAD_MASS__KG_" < 6000;
```

[24]

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

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

Explanation : **Task 6 - Boosters with Specific Success and Payload Range**

- This task requires listing the names of booster versions that had a successful landing on a drone ship and carried a payload mass greater than 4000 kg but less than 6000 kg.
- This combines filtering by landing outcome and payload mass range.

Total Number of Successful and Failure Mission Outcomes

List the total number of successful and failure mission outcomes

```
%sql SELECT "Mission_Outcome", COUNT(*) AS OutcomeCount FROM SPACEXTABLE GROUP BY "Mission_Outcome";
```

[25] Python

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

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

Explanation : **Task 7 - Total Count of Mission Outcomes**

- This task is about displaying the total number of successful and failure mission outcomes.
- It groups the data by "Mission_Outcome" and counts the occurrences of each type.

Boosters Carried Maximum Payload

```
%%sql
SELECT "Booster_Version"
FROM SPACEXTABLE
WHERE "PAYLOAD_MASS__KG_" = (
    SELECT MAX("PAYLOAD_MASS__KG_")
    FROM SPACEXTABLE
);
```

[26]

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

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

Explanation : **Task 8 - Boosters with Maximum Payload Mass**

- The objective here is to list all booster versions that have carried the maximum payload mass.
- This involves using a subquery with an aggregate function (MAX) to find the highest payload mass, then selecting the booster versions associated with that mass.

2015 Launch Records

```
%%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 (drone ship)%';
```

[27]

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

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)

Explanation :

Task 9 - Failure Landing Outcomes in 2015

- This task focuses on displaying the month names, booster versions, launch sites, and landing outcomes for failure landing outcomes on a drone ship in the year 2015.

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

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

[28]

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

Landing_Outcome	OutcomeCount
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

Explanation :

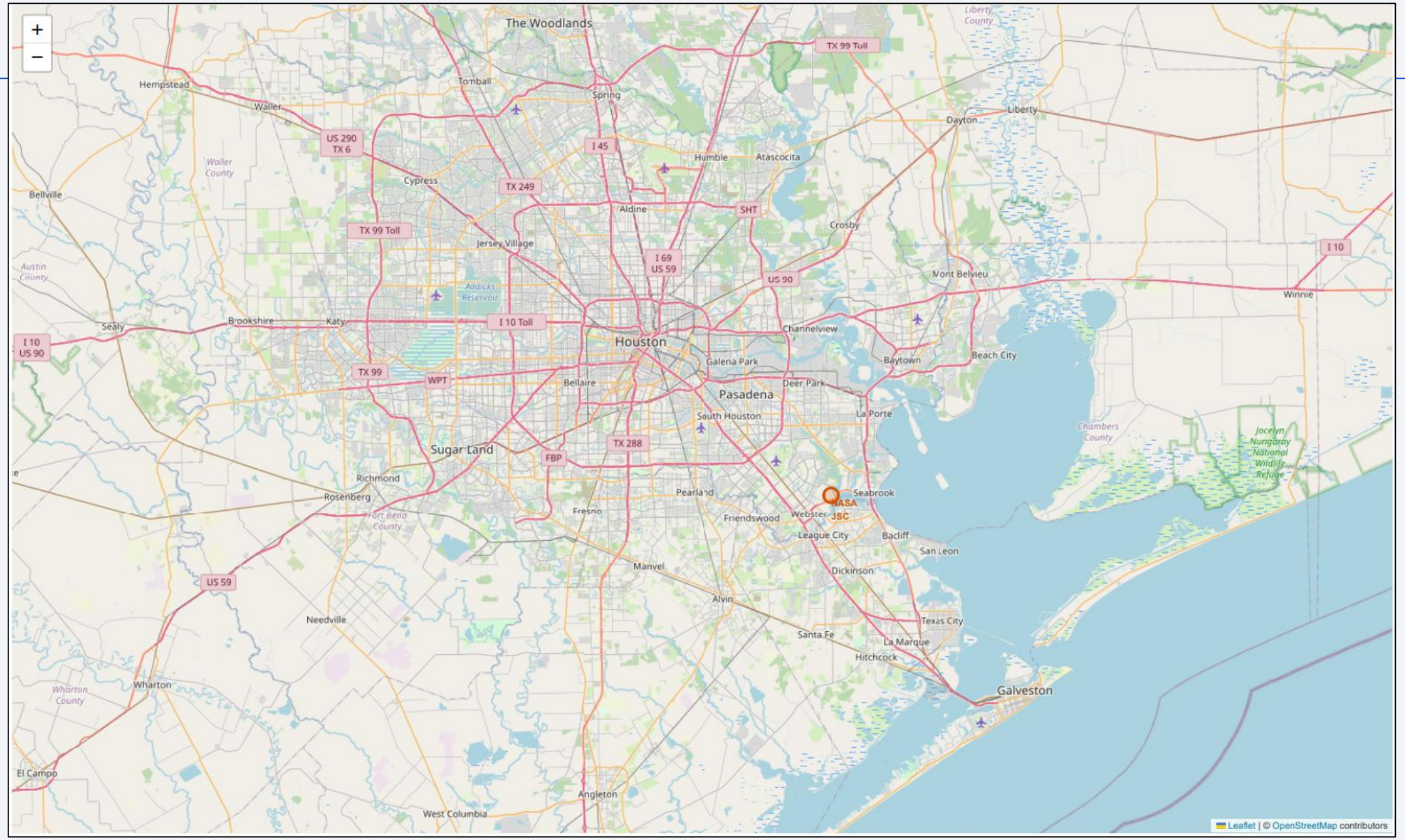
Task 10: Ranked Landing Outcome Counts (2010-2017)

- The final task is to rank the count of landing outcomes (e.g., 'Failure (drone ship)' or 'Success (ground pad)') between June 4, 2010, and March 20, 2017, in descending order.
- This provides an ordered view of the frequency of different landing outcomes within a specified date range.

Section 3

Launch Sites Proximities Analysis

Folium Map 1 - SpaceX Launch Sites Map



Folium Map 1 - SpaceX Launch Sites Map

Folium Map 1 - SpaceX Launch Sites Map

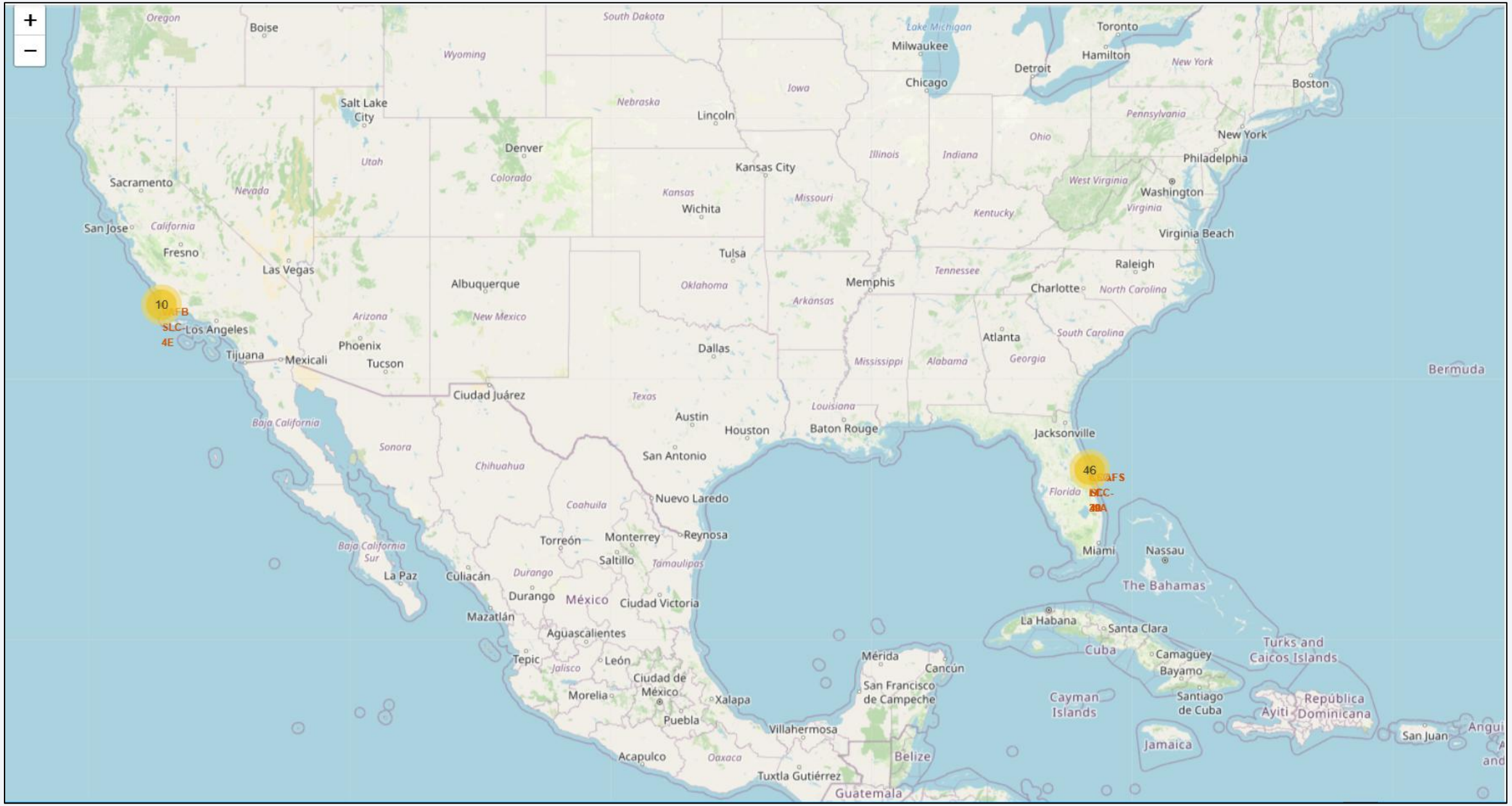
"**TASK 1: Mark all launch sites on a map**" involves using the **Folium** Python library to visualize SpaceX launch sites on an interactive map.

The process includes:

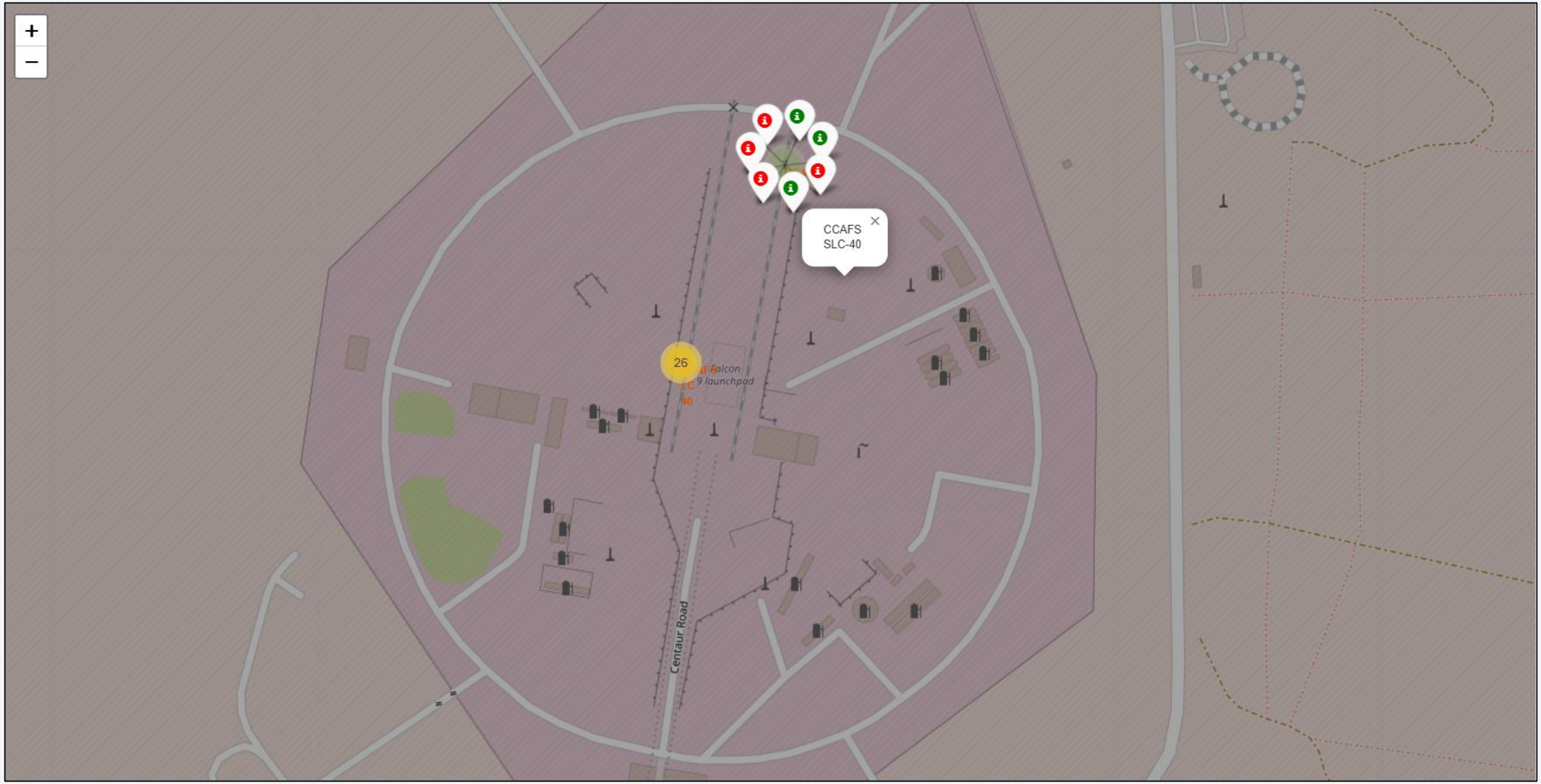
- Importing necessary libraries like folium, pandas, MarkerCluster, MousePosition, and DivIcon.
- Downloading and preparing the "spacex_launch_geo.csv" dataset to extract unique Launch Site, Lat, and Long coordinates.
- Initializing a Folium map object centered at the NASA Johnson Space Center in Houston, Texas, with a starting zoom level.
- Adding a marker for NASA JSC as a small yellow circle with a text label.
- Iterating through each launch site from the prepared data to add a black folium.Circle (1000m radius with a popup) and a folium.Marker (with the site name as a label) at its coordinates.
- Displaying the generated map, which allows users to zoom and explore.

The objective is to gain intuitive geographical insights into the launch site locations, helping to understand patterns such as their proximity to the Equator or the coast, which are not apparent from raw coordinates.

Folium Map 2 - Launch Success/Failure Visualization by Site



Folium Map 2 - Launch Success/Failure Visualization by Site



Folium Map 2 - Launch Success/Failure Visualization by Site

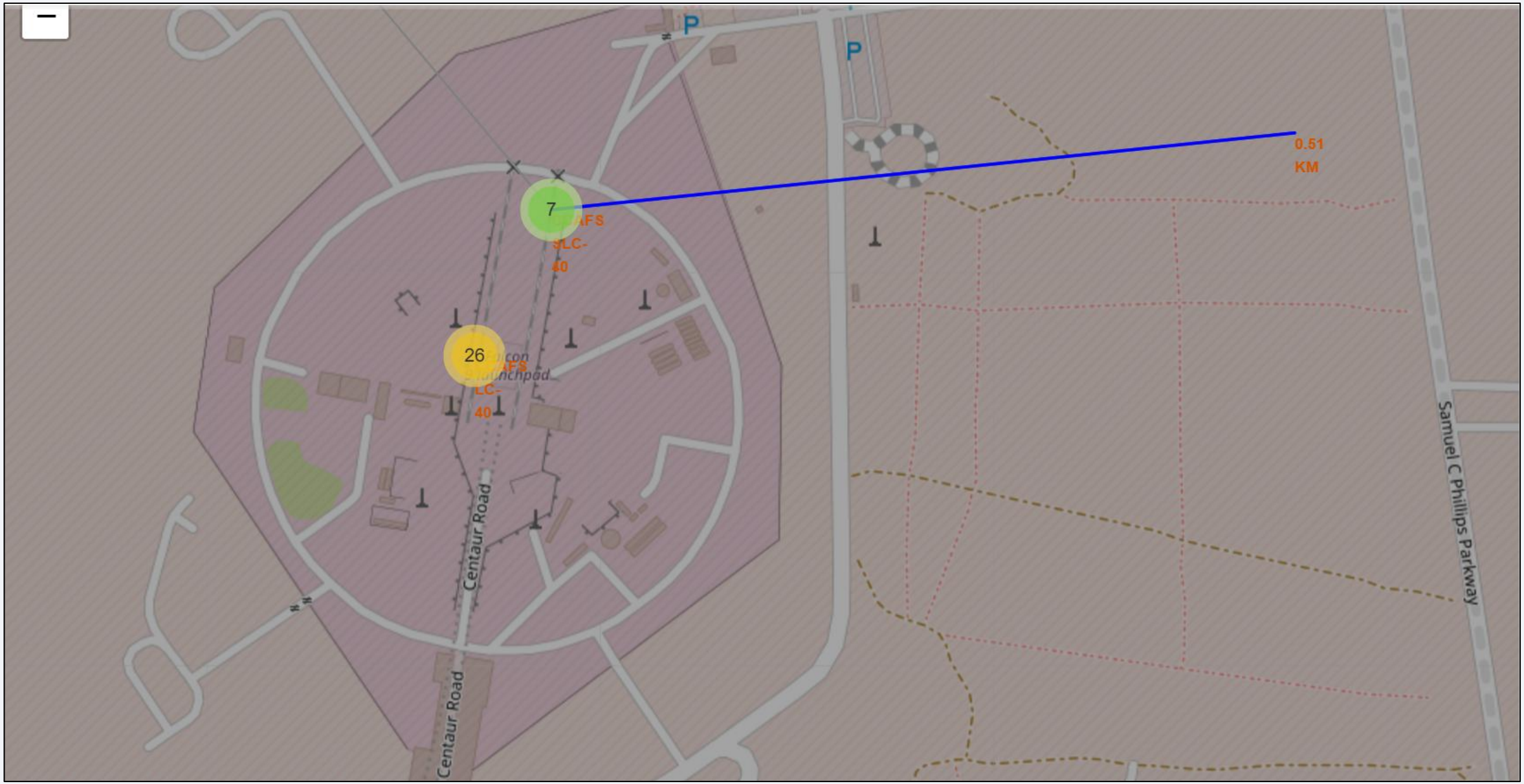
"**TASK 2: Mark the success/failed launches for each site on the map**" builds upon the previous map by visualizing the outcome of individual SpaceX launches at each site. The goal is to easily identify which launch sites demonstrate high success rates.

Key aspects of the "TASK 2" maps include:

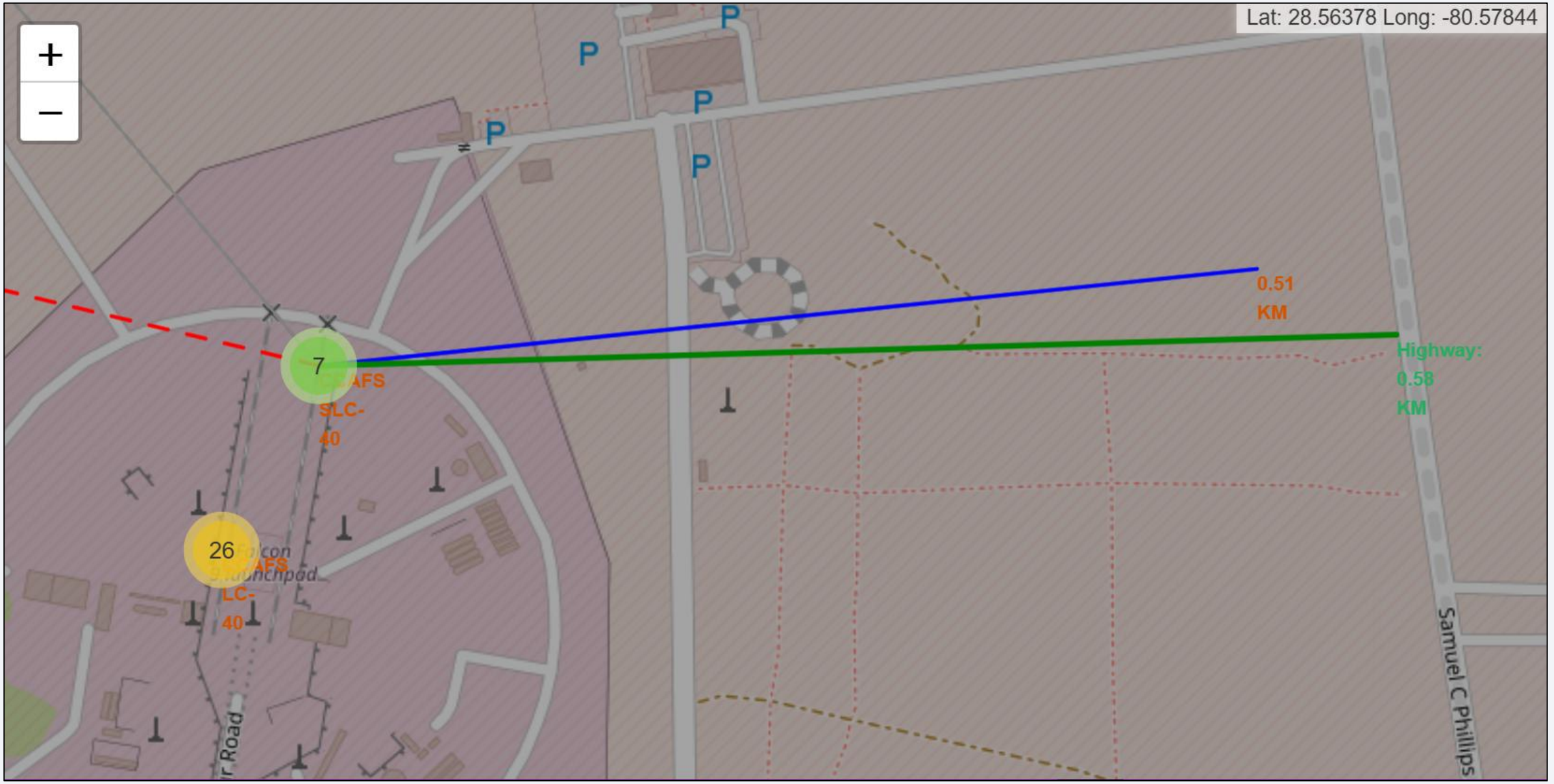
- **Data Source**: The spacex_df DataFrame is used, specifically its class column which indicates if a launch was successful (1) or failed (0).
- **Color-Coding**: A new marker_color column is added to the DataFrame. Successful launches (class=1) are marked green, while failed launches (class=0) are marked red.
- **Marker Clusters**: To simplify the map where many launches share the same coordinates, a MarkerCluster object is used. This groups overlapping markers, revealing individual outcomes upon zooming in.
- **Interactive Markers**: Each launch record is represented by a folium.Marker colored according to its outcome. These markers include a popup displaying "Outcome: Success" or "Outcome: Failure".

By adding these color-labeled markers within marker clusters, the maps allow for **intuitive identification of success rates** at different launch sites.

Folium Map 3 - Launch Site Proximity and Distance Analysis Map



Folium Map 3 - Launch Site Proximity and Distance Analysis Map



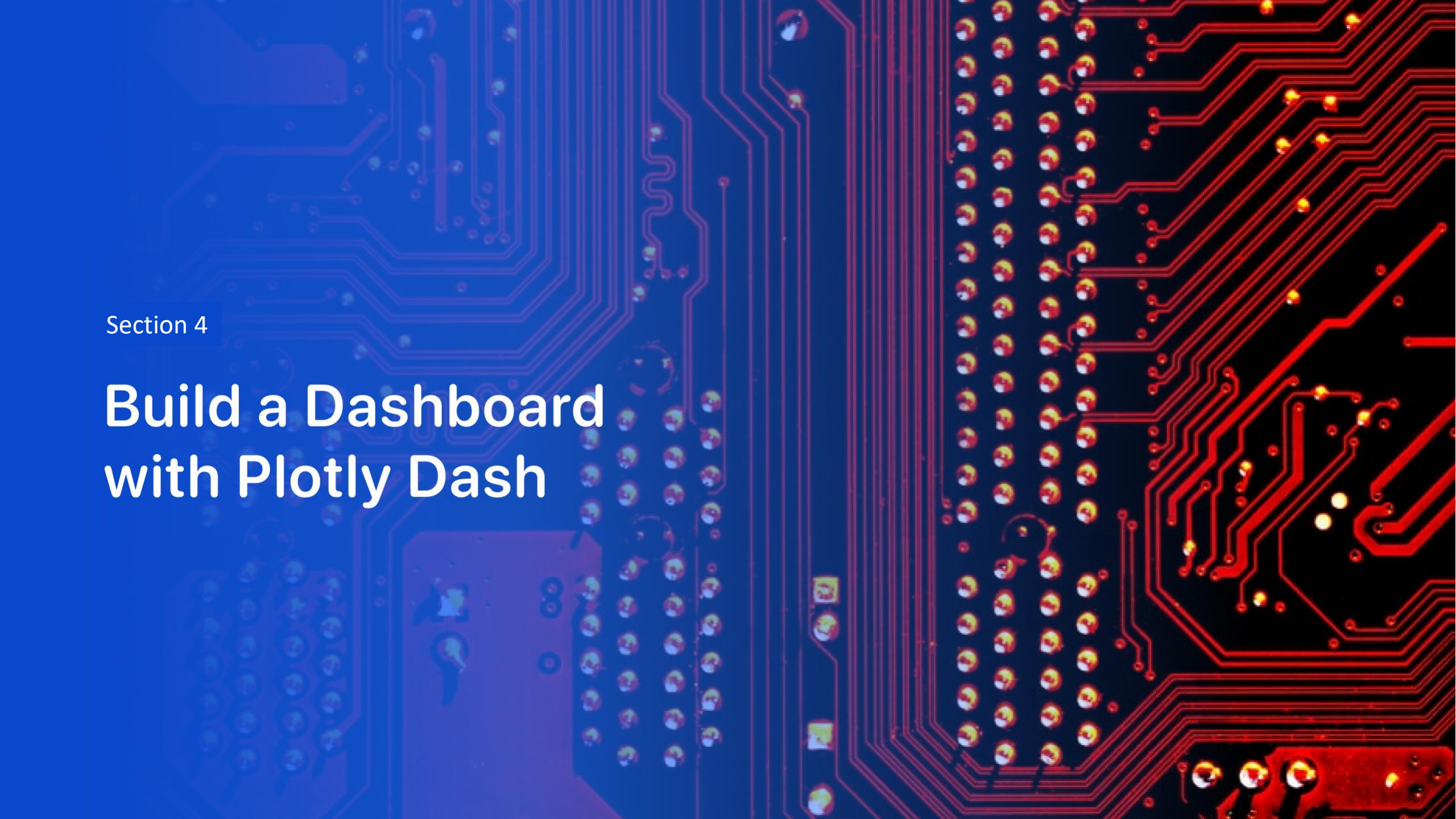
Folium Map 3 - Launch Site Proximity and Distance Analysis Map

"**TASK 3: Calculate the distances between a launch site to its proximities**" focuses on analyzing the geographical context of launch sites to discover patterns about their optimal locations.

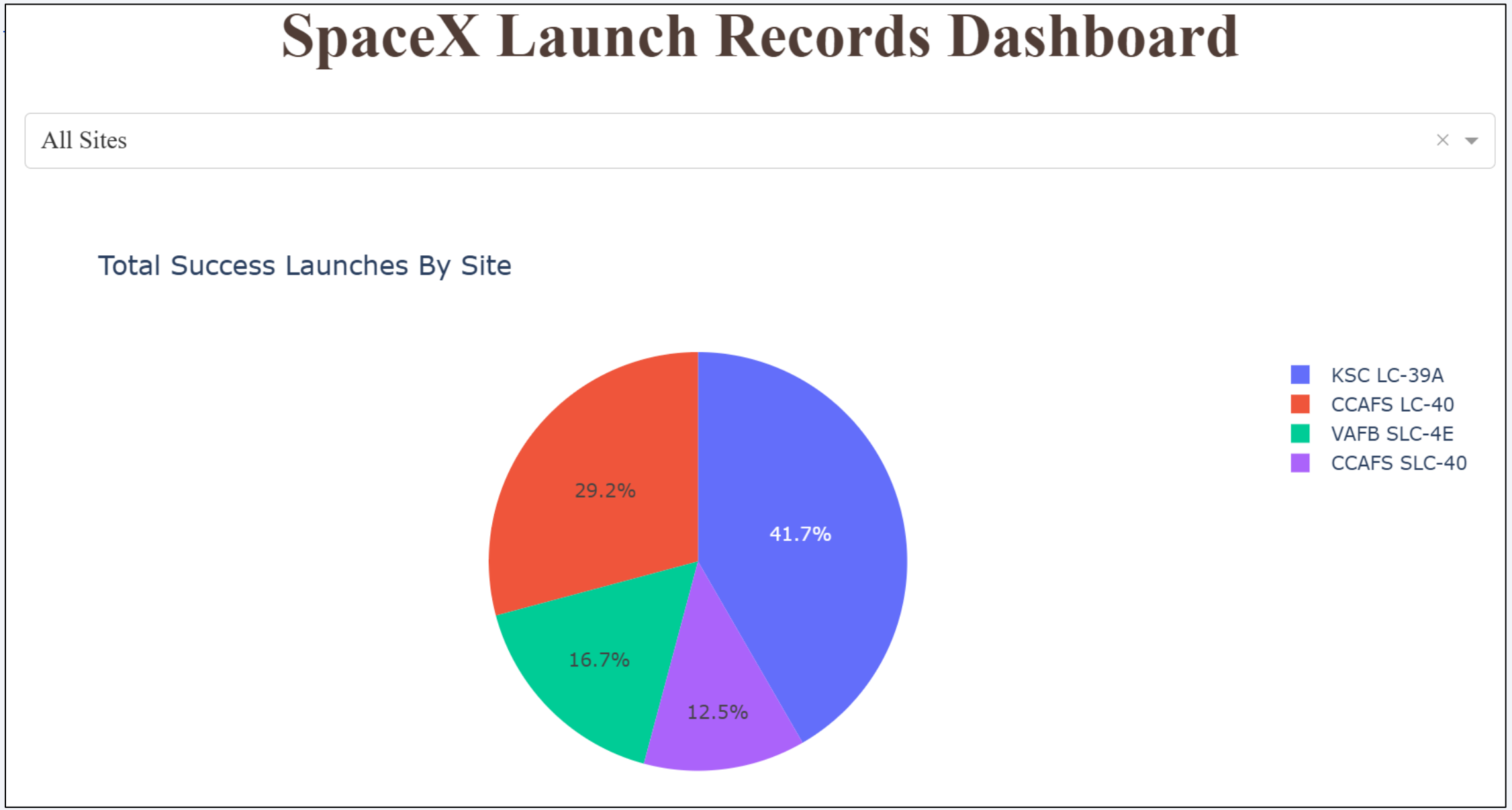
The maps in "TASK 3" are generated to:

- **Identify Coordinates**: A MousePosition plugin is added to the map, allowing users to find the latitude and longitude of any point of interest (like a coastline, railway, highway, or city) by hovering their mouse.
- **Calculate Distances**: A calculate_distance function is provided to compute the distance in kilometers between a launch site and these identified proximity points.
- **Visualize Proximities**:
  - folium.Marker objects are placed at the coordinates of the closest coastline, railway, highway, and city to a selected launch site.
  - Each marker displays the calculated distance from the launch site.
  - folium.PolyLine objects are drawn to visually connect the launch site to each of these proximity points, showing the measured distances on the map.

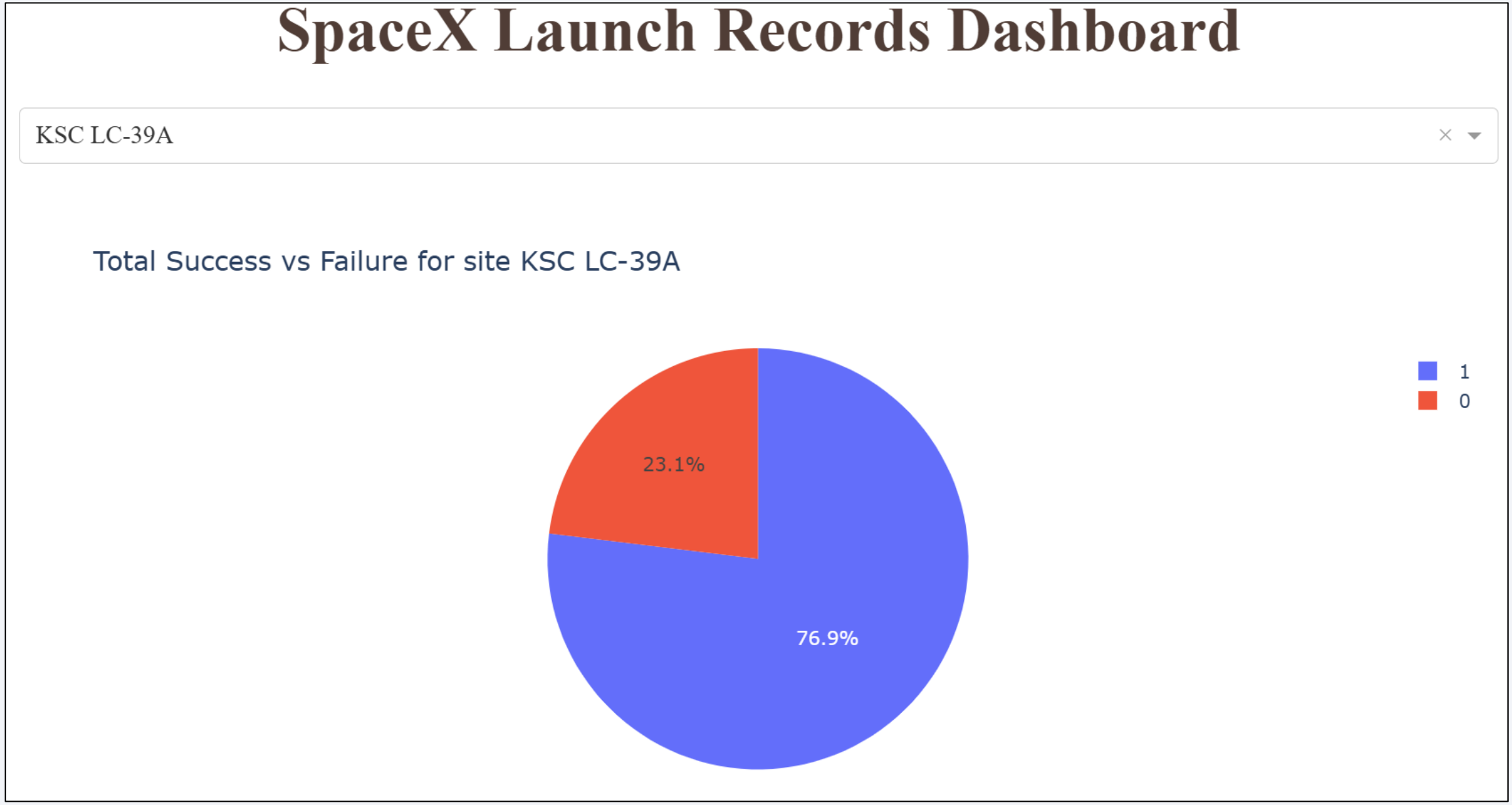
This task helps in **exploring and analyzing the proximities of launch sites** and finding **geographical patterns** that might influence launch site locations.


Section 4
Build a Dashboard with Plotly Dash

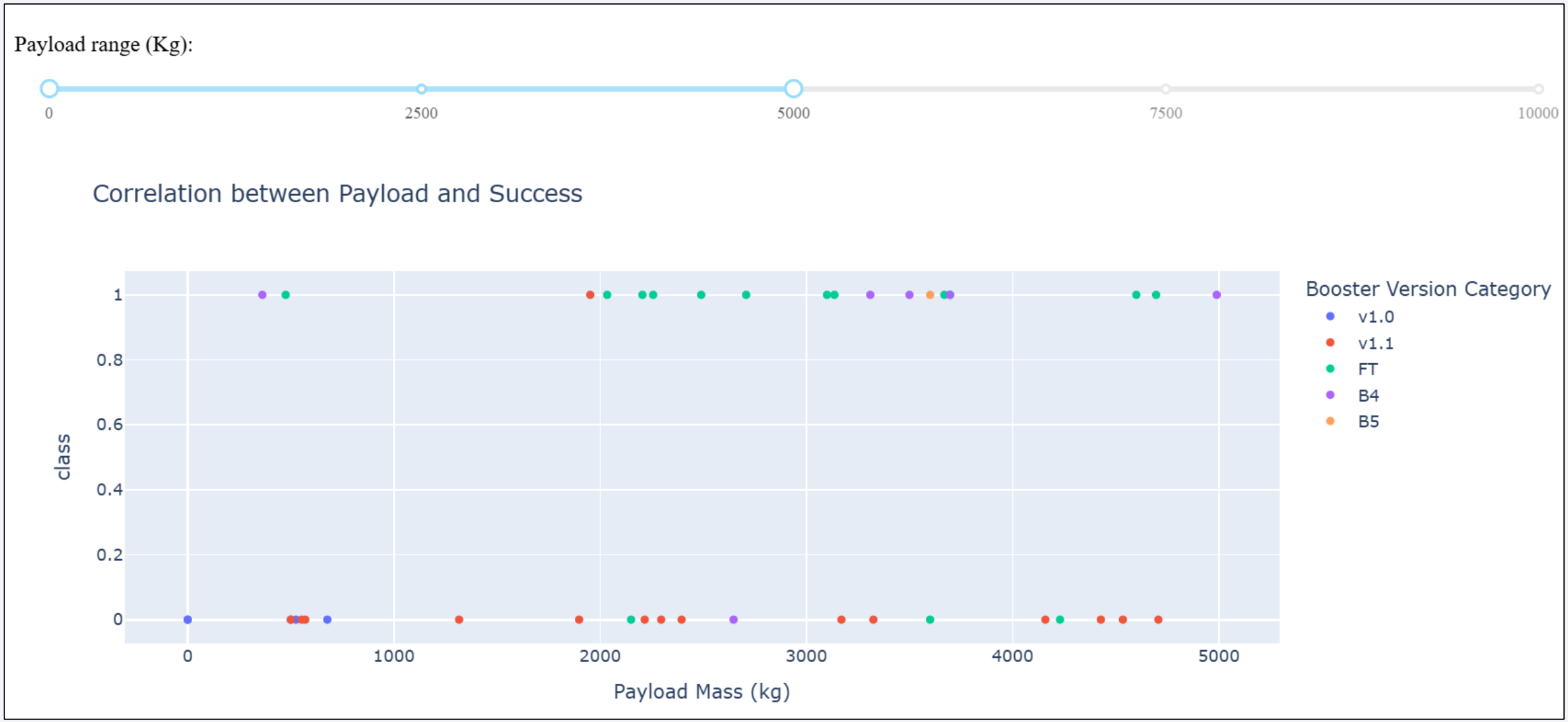
Launch success count for all sites, in a Piechart



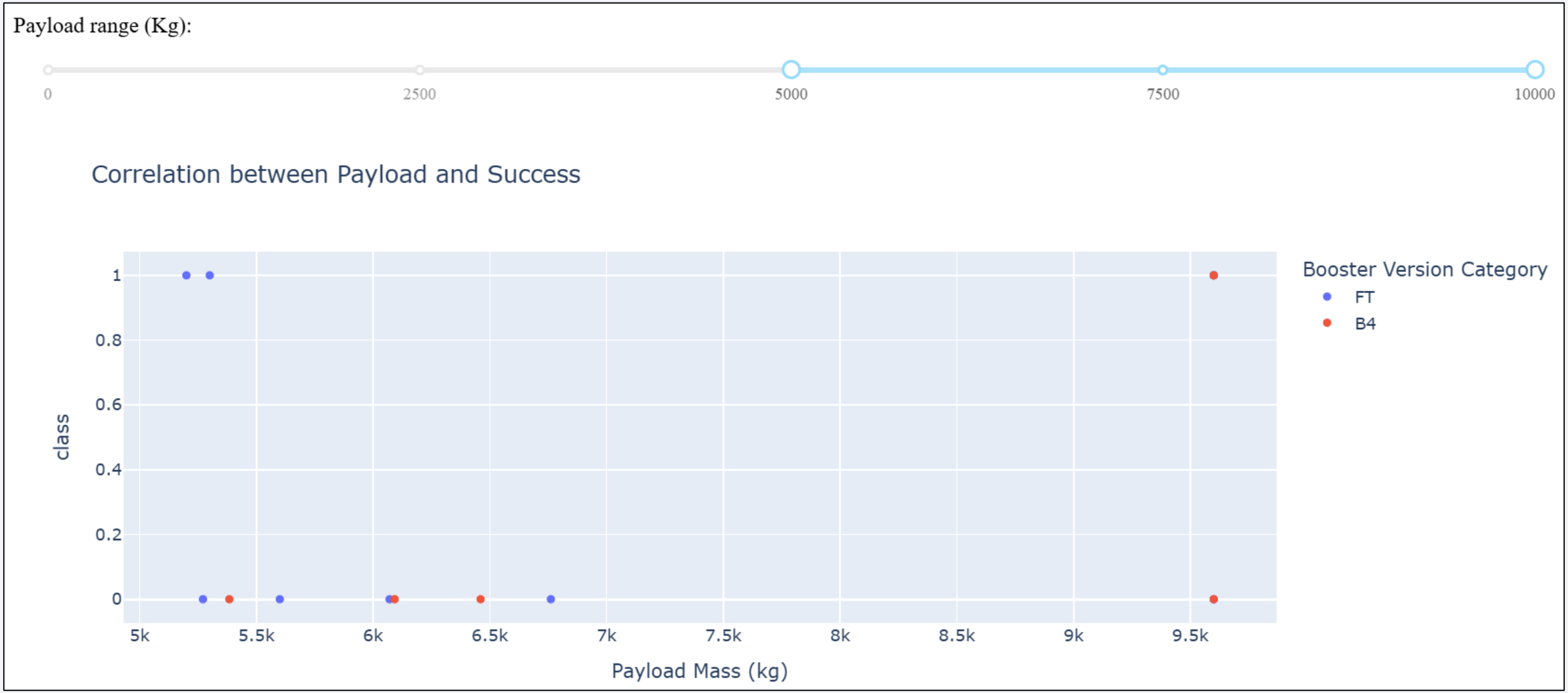
Piechart for the launch site with highest launch success ratio



Payload vs. Launch Outcome scatter plot for all sites, with different payload selected in the range slider



Payload vs. Launch Outcome scatter plot for all sites, with different payload selected in the range slider



Section 5

Predictive Analysis (Classification)

Classification Accuracy

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

# Parameters given in the task
parameters = {'C': [0.01, 0.1, 1],
              'penalty': ['l2'],
              'solver': ['lbfgs']}  # l1 = Lasso, l2 = Ridge

# Create logistic regression object
lr = LogisticRegression()

# Create GridSearchCV object with 10-fold cross-validation
logreg_cv = GridSearchCV(lr, parameters, cv=10)

# Fit the model
logreg_cv.fit(X_train, Y_train)
```

Logistic Regression Model

```
test_accuracy = logreg_cv.score(X_test, Y_test)
print("Test set accuracy:", test_accuracy)
```

```
Test set accuracy: 0.8333333333333334
```

```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

[15]

```
tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8464285714285713
```

Classification Accuracy

```python
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
import numpy as np

# Parameter grid for SVM
parameters = {
    'kernel': ('linear', 'rbf', 'poly', 'rbf', 'sigmoid'),
    'C': np.logspace(-3, 3, 5),
    'gamma': np.logspace(-3, 3, 5)
}

# Create SVM object
svm = SVC()

# Create GridSearchCV object with 10-fold cross-validation
svm_cv = GridSearchCV(svm, parameters, cv=10)

# Fit the model
svm_cv.fit(X_train, Y_train)
```

Support Vector Machine (SVM) Model

```python
# Calculate test set accuracy
test_accuracy_svm = svm_cv.score(X_test, Y_test)
print("Test set accuracy:", test_accuracy_svm)
```

[20]

```
Test set accuracy: 0.8333333333333334
```

```python
print("tuned hpyerparameters :(best parameters) ",svm_cv.best_params_)
print("accuracy :",svm_cv.best_score_)
```

[19] Python

```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
accuracy : 0.8482142857142856
```

Classification Accuracy

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

# Parameter grid for Decision Tree
parameters = {
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random'],
    'max_depth': [2 * n for n in range(1, 10)],
    'max_features': ['auto', 'sqrt'],
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 5, 10]
}

# Create Decision Tree object
tree = DecisionTreeClassifier()

# Create GridSearchCV object with 10-fold cross-validation
tree_cv = GridSearchCV(tree, parameters, cv=10)

# Fit the model
tree_cv.fit(X_train, Y_train)
```

Decision Tree Model

```python
accuracy = tree_cv.score(X_test, Y_test)
print("Test set accuracy:", accuracy)
```

[24]

```
Test set accuracy: 0.8333333333333334
```

```python
print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', 'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 1,
accuracy : 0.8625
```

Classification Accuracy

```
parameters = {
    'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'p': [1, 2]
}

KNN = KNeighborsClassifier()

knn_cv = GridSearchCV(KNN, parameters, cv=10)
knn_cv.fit(X_train, Y_train)
```

K-Nearest Neighbors (KNN) Model

```
test_accuracy = knn_cv.score(X_test, Y_test)
print("Test set accuracy:", test_accuracy)
```

[28]

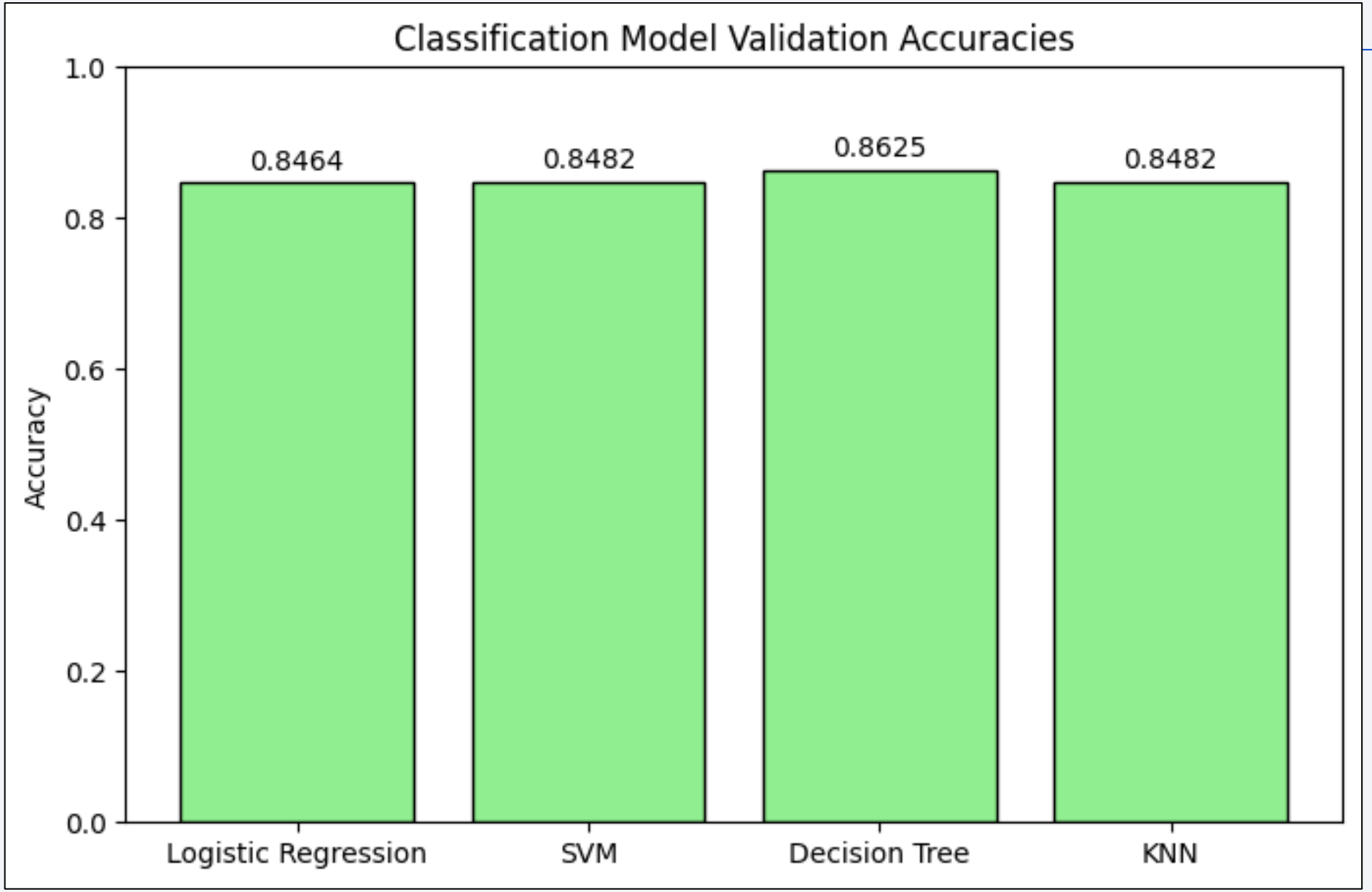
```
Test set accuracy: 0.8333333333333334
```

```
print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
print("accuracy :",knn_cv.best_score_)
```

[27]

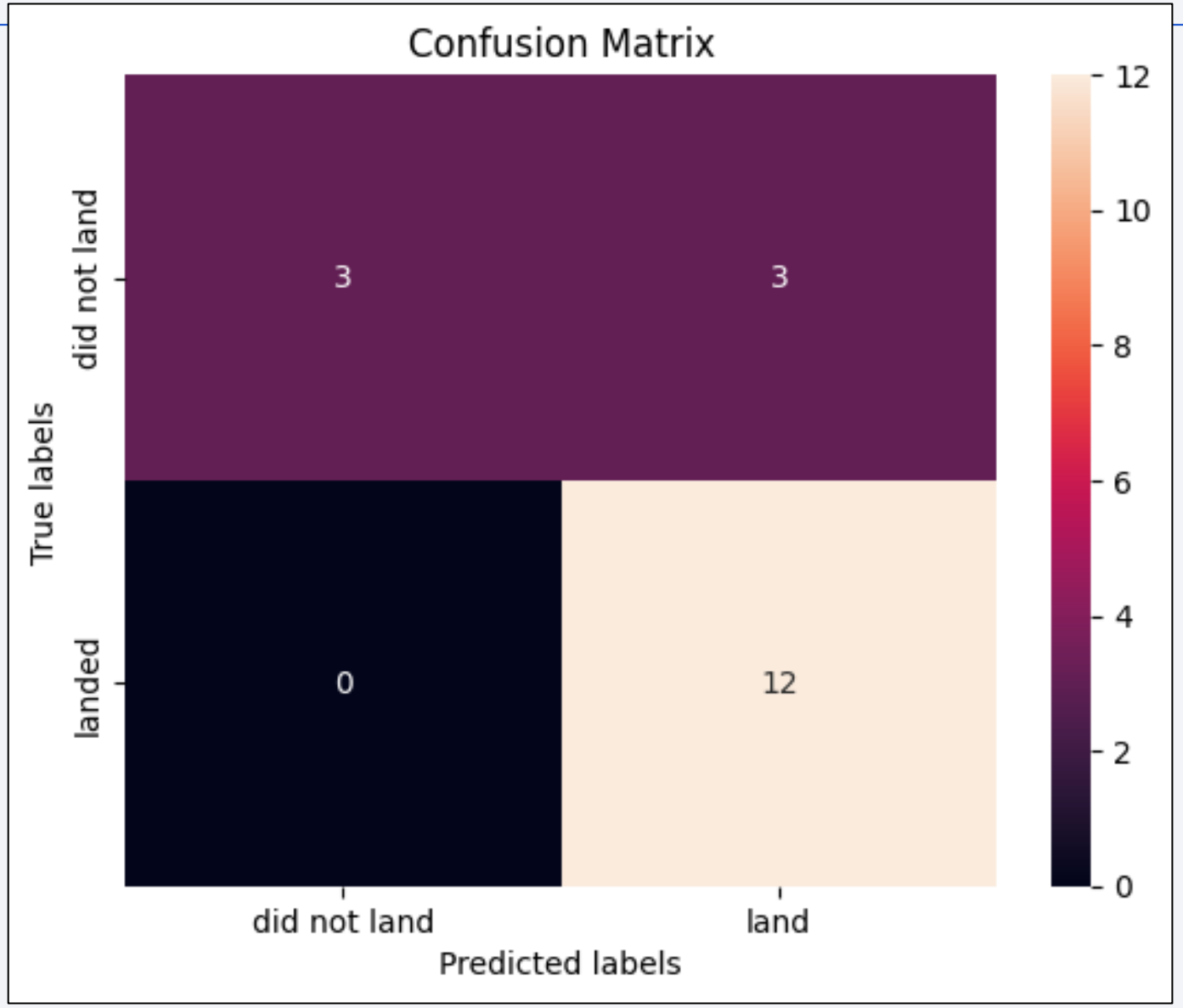
```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}
accuracy : 0.8482142857142858
```

Classification Accuracy



This is the bar chart of the Models' validation accuracies.
The **Decision Tree** model performed best, with an accuracy of **0.8625**.

Confusion Matrix



Confusion Matrix

The best performing model is **Decision Tree Classifier**, which achieved an accuracy of **0.8625**.

Confusion Matrix of the Decision Tree Classifier model represents the following :-

- The x-axis labels are **'did not land'** and **'land'**, representing the predicted outcomes.
- The y-axis labels are **'did not land'** and **'landed'**, representing the true labels.
- The matrix helps to visualize the number of **True Positives** (correctly predicted as landed), **False Positives** (incorrectly predicted as landed when the true label was not landed), **True Negatives** (correctly predicted as not landed), and **False Negatives** (incorrectly predicted as not landed when the true label was landed).

Conclusions

- **A comprehensive data pipeline was established**, encompassing data collection via SpaceX API and web scraping, rigorous data wrangling, and in-depth Exploratory Data Analysis (EDA) using both SQL queries and data visualization techniques.
- **Key insights into Falcon 9 landing success factors were uncovered through EDA**, demonstrating improved success rates with higher flight numbers, particularly at sites like CCAFS SLC-40, and a consistent increase in overall success rate from 2013 to 2020, approaching 100% by 2020. Specific orbits like LEO and ISS also showed higher success rates.
- **Interactive visual analytics tools were developed** to enhance data exploration, including Folium maps to visualize launch sites and success/failure outcomes and analyze proximity to key geographical features, and a Plotly Dash dashboard for interactive analysis of launch success rates by site and payload mass.
- **Multiple machine learning classification models were successfully built and evaluated** to predict Falcon 9 first stage landing success, including Logistic Regression, Support Vector Machine (SVM), Decision Tree Classifier, and K-Nearest Neighbors (KNN).
- **The Decision Tree Classifier emerged as the best performing model**, achieving a validation accuracy of **0.8625** and a test set accuracy of **0.8333** across all evaluated models, providing a robust prediction capability for Falcon 9 first stage landings which can inform launch cost determination.

Appendix

GitHub Link :
https://github.com/Avik-Das-567/IBM-Applied-Data-Science-Capstone

Thank you!