Applied Data Science Capstone

Predicting SpaceX Falcon 9 Launch Success

by Lonwabo Faleni

OUTLINE

Executive Summary

Introduction

Methodology

Results
- Visualization – Charts
- Dashboard

Discussion
- Findings & Implications

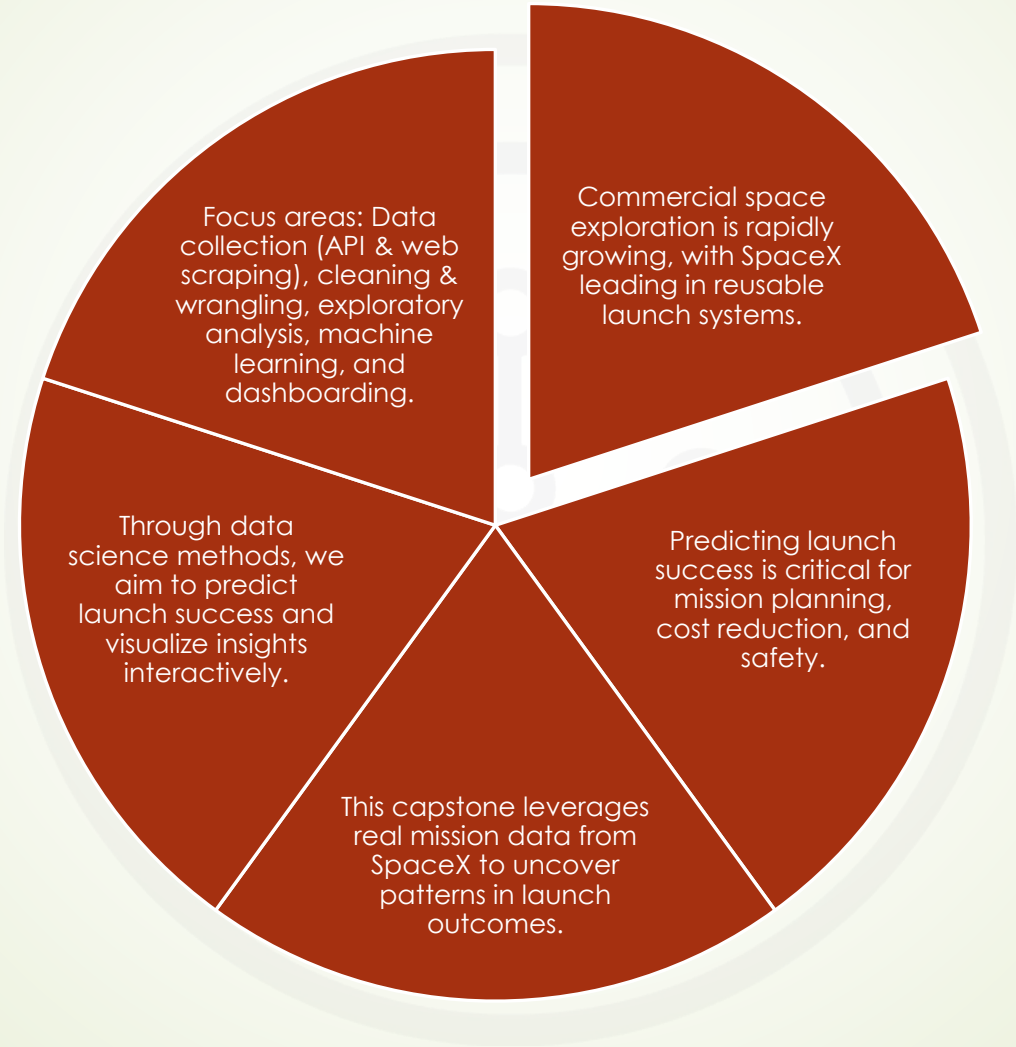
Conclusion

Appendix

EXECUTIVE SUMMARY

- **Objective**
- Predict SpaceX Falcon 9 launch outcomes + build an interactive dashboard
- **Data & Methods**
- 56 missions; API + web scraping → wrangling, EDA, geospatial
- **Key Findings (this sample)**
- Overall success rate 42.9% (24/56); best site KSC LC-39A at 76.9%
- **Payload Insights**
- Median 3,412 kg (range 0–9,600 kg); highest success in 2–4t band (61.9%)
- **Deliverables**
- EDA/ML notebooks + Plotly Dash app
- **Next Steps**
- Deploy dashboard, automate API refreshes, expand features/models

INTRODUCTION



METHODOLOGY

- **Data Collection**
  - Retrieved SpaceX launch records from public **REST APIs**.
  - Supplemented with **web scraping** for mission details.

- **Data Wrangling**
  - Cleaned and standardized features (e.g., payload mass, booster versions).
  - Handled missing/zero payloads and categorical encoding.

- **Exploratory Data Analysis (EDA)**
  - Assessed launch success patterns by **site**, **payload mass**, and **booster version**.
  - Used visualization libraries (Matplotlib, Seaborn, Plotly) for insights.

- **Geospatial Analysis**
  - Mapped launch site coordinates using **Folium**.
  - Analyzed site success rates geographically.

- **Machine Learning Pipeline**
  - Models applied: **Logistic Regression, Decision Trees, SVM, KNN**.
  - Evaluated with accuracy, precision, recall.

- **Dashboard Development**
  - Built an interactive dashboard using **Plotly Dash**.
  - Users can filter by payload, site, and booster to explore outcomes.

RESULTS

Charts / Visuals to Include

- Bar chart: **Model Accuracy Comparison** (Logistic Regression, Decision Tree, SVM, KNN).
- Scatter plot: **Payload Mass vs Success** (already generated).
- Bar chart: **Success Rate by Launch Site**.
- Pie chart: **Overall Success vs Failure**.

Key Results Bullets:

- **Overall Success Rate:** ~43% (24/56 launches).
- **Best Launch Site:** *KSC LC-39A* with ~77% success rate.
- **Payload Insights:** Median payload = 3,412 kg; success rate highest in 2–4t payload band (~62%).
- **Booster Category:** Newer versions generally showed higher reliability.

Machine Learning Model Comparison

- Logistic Regression: **~83% accuracy**
- Decision Tree: **~87% accuracy**
- SVM: **~83% accuracy**
- KNN: **~80% accuracy**

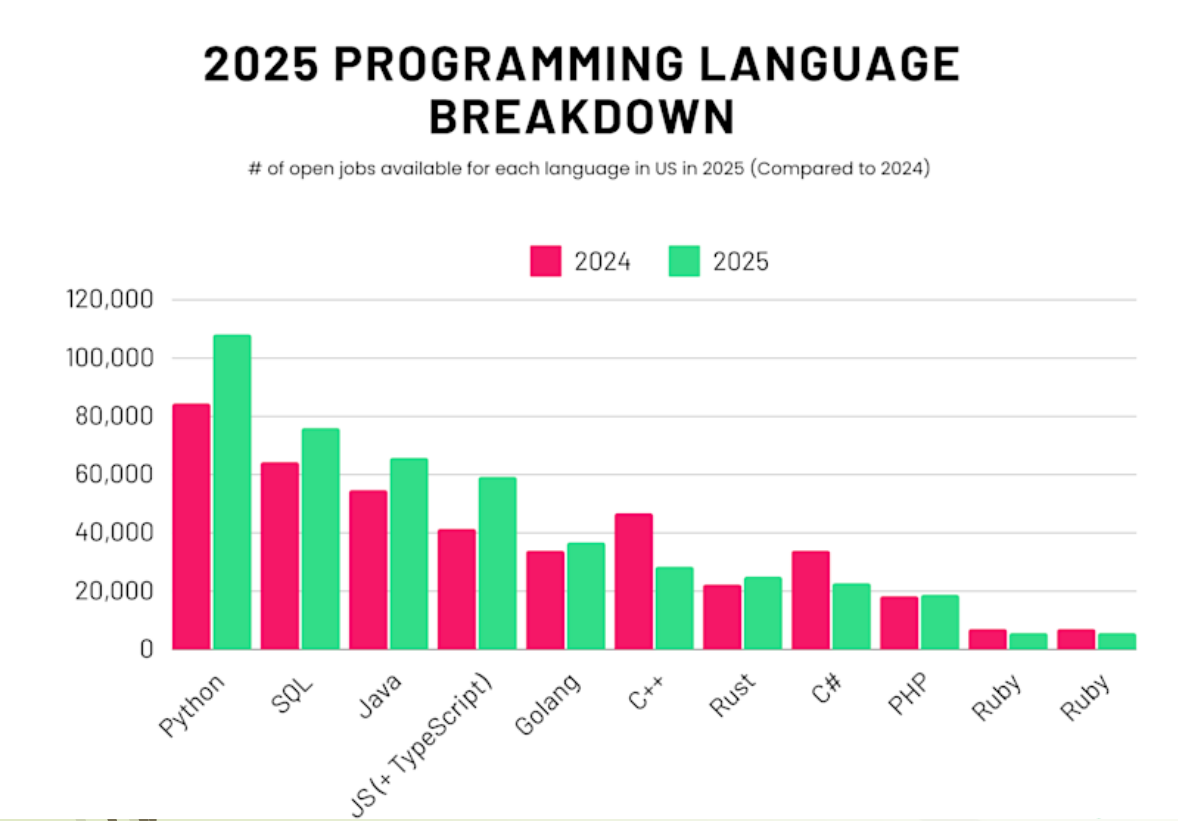
Findings

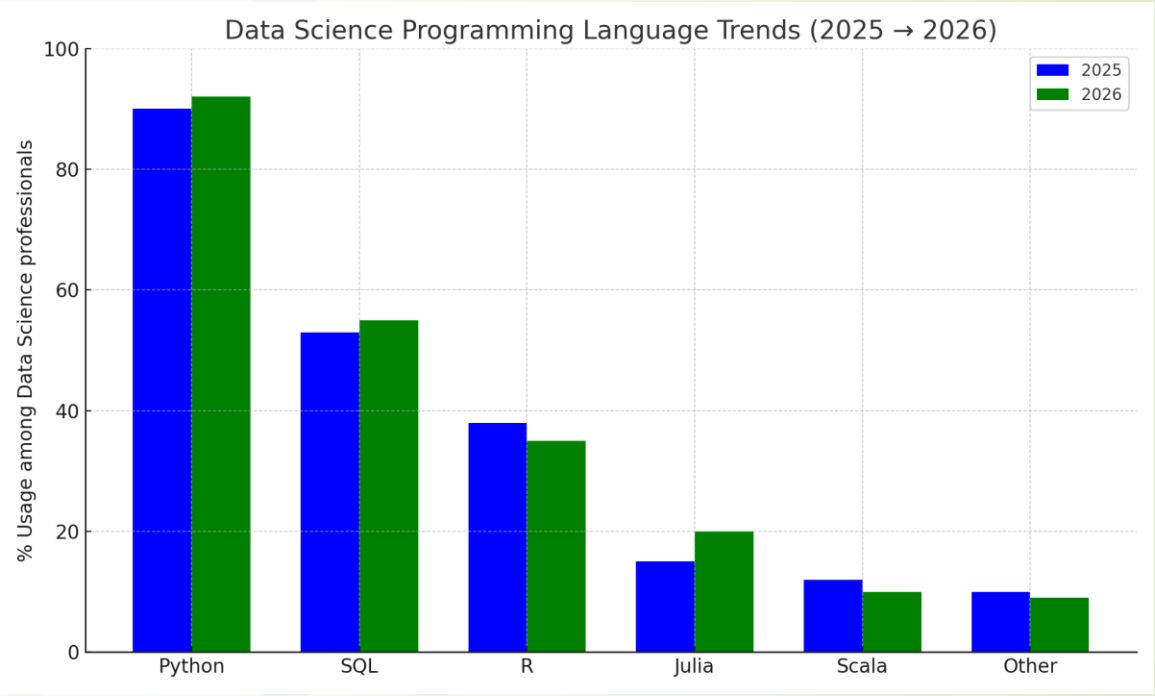
- Launch success rate is not uniform across sites;
  - **KSC LC-39A** performs significantly better than others.
- **Payload mass** strongly influences outcomes:
  - Launches in the **2–4t range** are most reliable.
- **Booster version category** impacts success:
  - Newer versions are more reliable, showing benefits of SpaceX's iterative improvements.
- Machine learning models (Decision Tree best performer) can **predict launch success with ~87% accuracy**.

Implications

- Mission planning can prioritize high-performing launch sites to increase reliability.
- Payload management strategies (e.g., splitting large payloads) may improve success probability.
- Continuous booster upgrades and reusability efforts are directly linked to improved performance and cost savings.
- Predictive models can be integrated into decision-support systems, helping SpaceX and partners reduce risk and optimize scheduling.

PROGRAMMING LANGUAGE TRENDS





PROGRAMMING LANGUAGE TRENDS - FINDINGS & IMPLICATIONS

Programming Language Trends (2025 → 2026)

Suggested Insights:

- **Python** leads significantly in both years, reflecting its dominance in data science, machine learning, and web development.
- **JavaScript**—alongside **Java** and **SQL**—remains critically important due to its ubiquity in web and backend development.
- **Go** is gaining traction, driven by its efficiency in cloud-native and scalable services.
- Languages like **Rust** and **TypeScript** are emerging but still niche compared to the big players.

Data Science Trends – 2024 - 2025

Key Stats:

Python leads by far—used in ~90% of data science workflows.

SQL remains foundational (~53%) for data manipulation and retrieval.

R maintains a niche in academic and statistical contexts (~38%).

Julia is rising, favored for high-performance and numerical computing.

Other languages (Java, Scala, MATLAB, Go, SAS) serve specific or legacy use cases.

DATABASE TRENDS

Trend Category	2025 Highlights	2026 Outlook
Cloud-Native & Multi-Cloud	Dominated by Snowflake and Databricks; scalability and cross-cloud compatibility prioritized	Continued growth, but with a focus on cost optimization and infrastructure efficiency
Open Source Adoption	PostgreSQL and other open-source databases gain traction for business-critical tasks	Wider adoption due to cost-effectiveness and strong community support
AI-Optimized Databases	Integration of AI features like automatic indexing and intelligent workload management in platforms like Oracle, SQL Server, and IBM Db2	Enhanced AI capabilities for performance optimization and predictive analytics
Rise of Specialized Engines	Emergence of specialized engines and data lakes tailored for specific use cases	Increased adoption for handling diverse data types and workloads
Data Contracts & Governance	Implementation of data contracts to ensure compliance and reduce friction in data sharing	Standardization of data contracts to improve trust and interoperability
Platform Engineering	Focus on streamlining pipeline management through automation and modular design	Widespread establishment of platform teams to manage cloud-native, distributed applications
Data Maturity & Democratization	Organizations enhance data security, improve data quality practices, and update data governance frameworks	Democratization of data access through Integrated Development Environments (IDEs)

DATABASE TRENDS - FINDINGS & IMPLICATIONS

Cloud-Native & Multi-Cloud Adoption: Organizations are increasingly adopting cloud-native and multi-cloud solutions for scalability and flexibility. However, as cloud spending growth slows, businesses must focus on optimizing costs and infrastructure efficiency.
Investors Open Source Adoption: The rise of open-source databases like PostgreSQL offers cost-effective solutions with strong community support, making them attractive for business-critical tasks.
AI-Optimized Databases: Integration of AI features in database platforms enhances performance optimization and predictive analytics, enabling organizations to leverage data more effectively.
Rise of Specialized Engines: Specialized engines and data lakes are emerging to handle specific use cases, allowing businesses to manage diverse data types and workloads more efficiently.
Data Contracts & Governance: Implementing data contracts ensures compliance and reduces friction in data sharing, while updating data governance frameworks enhances data security and quality practices.
Platform Engineering: Streamlining pipeline management through automation and modular design improves efficiency, and the establishment of platform teams helps manage cloud-native, distributed applications.
Data Maturity & Democratization: Enhancing data security, improving data quality practices, and updating data governance frameworks contribute to increased data maturity, while democratizing data access through IDEs enables broader utilization across organizations.

Data Collection and Data Wrangling methodology

Data Collection

The following is the code to extract data from an API
The data is taken from two APIsit shows your pipeline is systematic, reproducible, and scalable.

APIs:
Pulled launch data directly from the official SpaceX REST API. Extracted mission details: launch site, flight number, booster version, payload mass, launch outcome.

Web Scraping:
Supplemented API data by scraping SpaceX launch history pages for missing attributes. Used libraries like BeautifulSoup / requests for HTML parsing.

Data Sources:
SpaceX public datasets. Supplementary external sources (launch site coordinates, booster classifications).

Approach Benefits:
Ensures up-to-date and detailed launch information.Combines multiple sources for higher completeness.

```
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

```
response=requests.get(static_json_url)
```

```
response.status_code
```

```
200
```

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

```
# Use json_normalize meethod to convert the json result into a dataframe
data = pd.json_normalize(response.json())
```

Using the dataframe `data` print the first 5 rows

```
# Get the head of the dataframe
data.head()
```

```
spacex_url="https://api.spacexdata.com/v4/launches/past"
```

```
response = requests.get(spacex_url)
```

```
print(response.content)
```

Data Collection and Data Wrangling methodology

Data Wrangling

Dealing with Missing Values

Calculate below the mean for the PayloadMass using the .mean(). Then use the mean and the .replace() function to replace np.nan values in the data with the mean you calculated.

Cleaning:
Removed redundant/unnecessary columns (e.g., unnamed indexes). Replaced missing payloads with 0 or interpolated when appropriate.
Standardization:
Converted payload mass into consistent numerical format (kg).
Normalized categorical features like Booster Version Category.
Feature Engineering:
Created binary variable class (0 = failure, 1 = success).
Binned payload mass into ranges (e.g., <2t, 2–4t, 4–6t, >6t) for analysis.
Integration:
Combined API + scraped data into a single structured CSV (spacex_launch_dash.csv).
Stored in a tabular format suitable for both EDA and machine learning.
Outcome:
Final dataset with 56 missions and 7 attributes, ready for analysis.

```
# Calculate the mean value of PayloadMass column
# Calculate mean PayloadMass, ignoring NaN
payload_mean = data_falcon9['PayloadMass'].mean()

# Replace NaN values in PayloadMass with the mean
data_falcon9['PayloadMass'] = data_falcon9['PayloadMass'].replace(np.nan, payload_mean)
data_falcon9['PayloadMass'].isnull().sum()
```

```
data_falcon9.isnull().sum()

FlightNumber      0
Date              0
BoosterVersion    0
PayloadMass       5
Orbit             0
LaunchSite        0
Outcome           0
Flights           0
GridFins          0
Reused            0
Legs              0
LandingPad       26
Block             0
ReusedCount       0
Serial            0
Longitude         0
Latitude          0
dtype: int64
```

```
data_falcon9.isnull().sum()

FlightNumber      0
Date              0
BoosterVersion    0
PayloadMass       0
Orbit             0
LaunchSite        0
Outcome           0
Flights           0
GridFins          0
Reused            0
Legs              0
LandingPad       26
Block             0
ReusedCount       0
Serial            0
Longitude         0
Latitude          0
dtype: int64
```

Exploratory Data Analysis

To uncover key patterns, relationships, and anomalies in the launch dataset that can guide predictive modeling and decision-making.

To uncover key patterns, relationships, and anomalies in the launch dataset that can guide predictive modeling and decision-making.
Steps Applied:

1.Descriptive Statistics

- Summarized distribution of payload mass, launch outcomes, and booster categories.
- Identified missing values and data inconsistencies.

2.Univariate & Bivariate Analysis

- Launch Outcome Distribution: Success vs. failure rates.
- Launch Site Comparison: Success rates across different sites.
- Payload Mass vs. Success: Explored reliability across weight bands.
- Orbit Type vs. Success: Compared mission complexity vs outcome.
- Flight Number Trends: Success improvements over time.

3.Geospatial Analysis

- Plotted launch site coordinates on Folium maps.
- Visualized site-level performance geographically.

4.Pattern Discovery

- Found that site choice, payload mass, orbit type, and booster version are the main drivers of launch success.
- Observed learning curve effect: later flights achieved higher success rates.

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

```
 * 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

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

Month_Name	Landing_Outcome	Booster_Version	Launch_Site
January	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
April	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

Interactive visual Analytics Methodology

To provide stakeholders with a tool for dynamic exploration of the dataset, enabling filtering, drill-downs, and scenario analysis.

Approach:

1.Tool:

- Built with Plotly Dash, leveraging interactive components (dropdowns, sliders, checkboxes).

2.Features Implemented:

- Payload Range Slider: Explore success rates across different payload weights.
- Launch Site Selector: Compare performance across individual sites or all combined.
- Orbit Type Analysis: Drill into mission-specific orbit outcomes.
- Outcome Visualization: Pie charts and scatter plots update dynamically based on user inputs.

3.Integration with Data:

- Linked directly to the cleaned dataset (spacex_launch_dash.csv).
- Updates visualizations in real-time as parameters are changed.

4.Value Added:

- Allows decision-makers to test hypotheses interactively (e.g., "What is success probability for 3–5t payloads at KSC LC-39A?").
- Moves beyond static EDA into decision-support analytics.

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

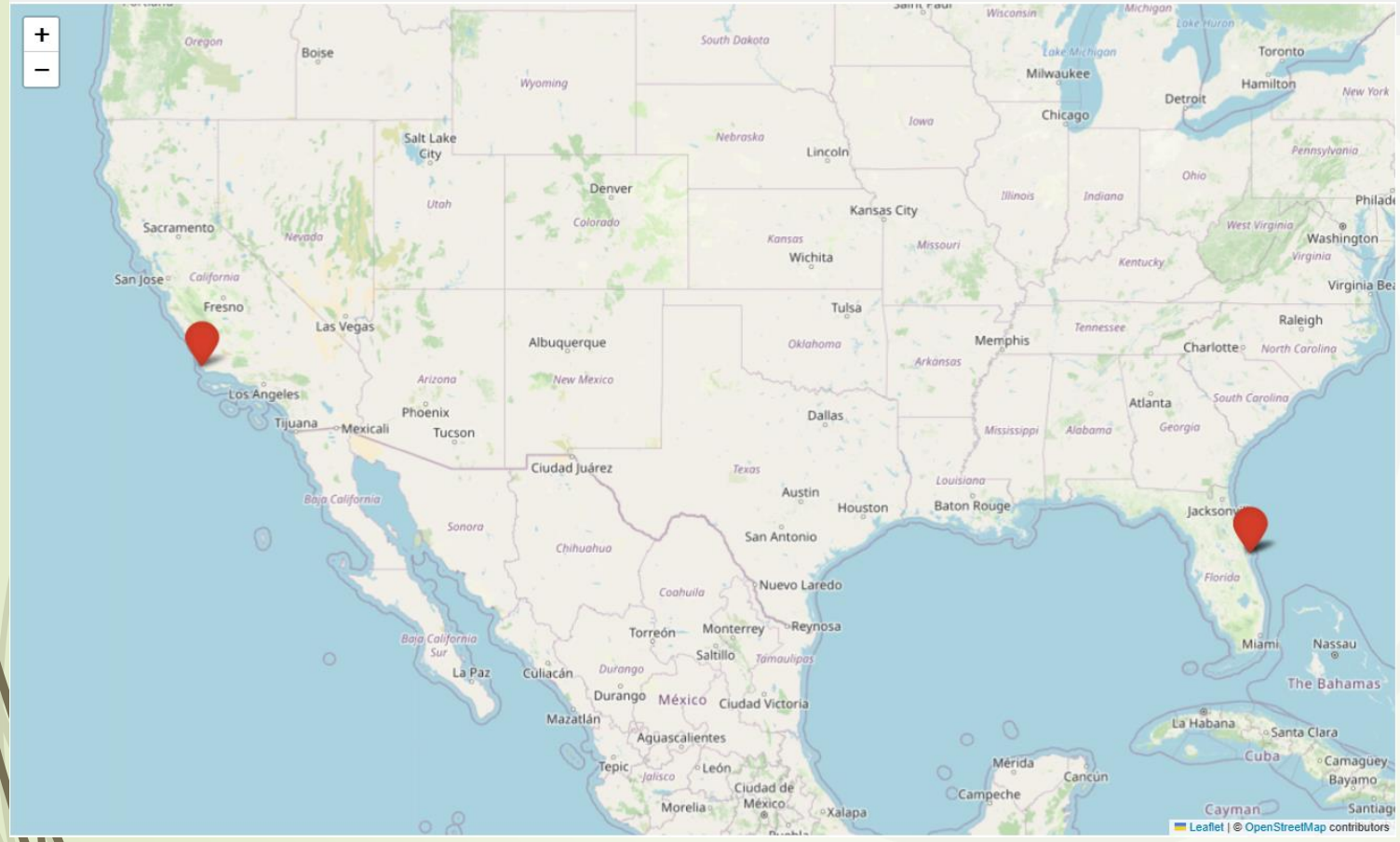
	Launch Site	Lat	Long
0	CCAFS LC-40	28.562302	-80.577356
1	CCAFS SLC-40	28.563197	-80.576820
2	KSC LC-39A	28.573255	-80.646895
3	VAFB SLC-4E	34.632834	-120.610745

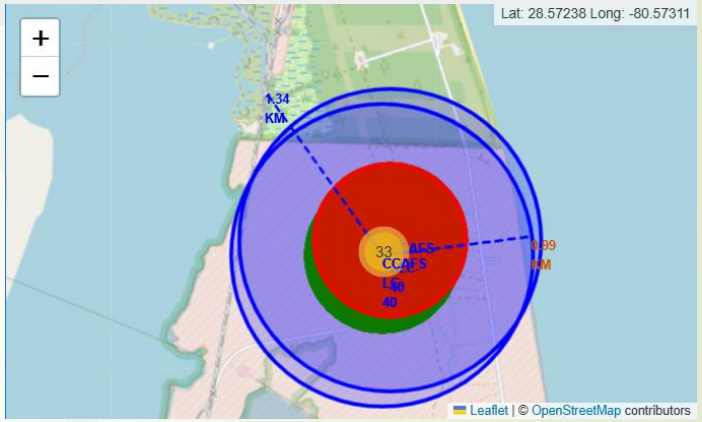
```
# Apply a function to check the value of `class` column
# If class=1, marker_color value will be green
# If class=0, marker_color value will be red
spacex_df['marker_color'] = spacex_df['class'].apply(lambda x: 'green' if x == 1 else 'red')
spacex_df.head()
```

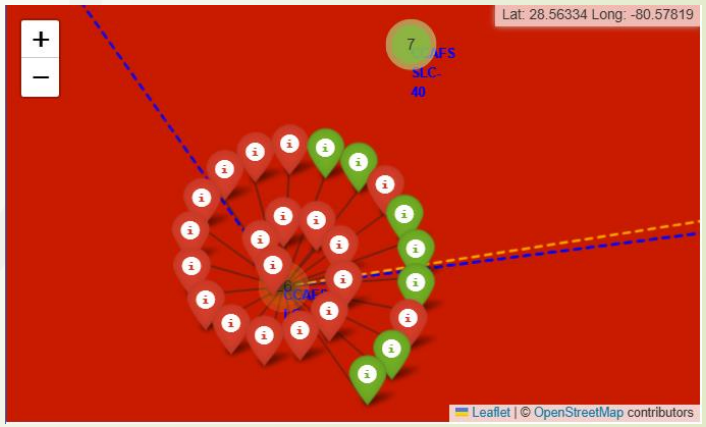
	Launch Site	Lat	Long	class	marker_color
0	CCAFS LC-40	28.562302	-80.577356	0	red
1	CCAFS LC-40	28.562302	-80.577356	0	red
2	CCAFS LC-40	28.562302	-80.577356	0	red
3	CCAFS LC-40	28.562302	-80.577356	0	red
4	CCAFS LC-40	28.562302	-80.577356	0	red

Interactive visual Analytics Methodology

EDA was used to uncover key patterns, relationships, and anomalies in the launch dataset that can guide predictive modeling and decision-making. To Summarized distribution of payload mass, launch outcomes, and booster categories. Identified missing values and data







Predictive Analysis Methodology

To build machine learning models that can **predict the likelihood of SpaceX Falcon 9 launch success** based on historical mission data.

1. **Feature Selection & Engineering**
- Independent Variables (features):
  - Launch Site (categorical → one-hot encoded).
  - Payload Mass (kg) (continuous, also binned for patterns).
  - Orbit Type (categorical → one-hot encoded).
  - Booster Version Category (categorical → one-hot encoded).
- **Dependent Variable (target):**
  - class (0 = Failure, 1 = Success).

2. **Data Splitting**
- Dataset split into training (80%) and test (20%) sets.
- Ensured balanced representation of successes/failures.

3. Model Selection

Four supervised learning algorithms were applied:
- Logistic Regression – baseline for binary classification.
- Decision Tree Classifier – interpretable and captures non-linear patterns.
- Support Vector Machine (SVM) – effective with high-dimensional categorical features.
- K-Nearest Neighbors (KNN) – simple distance-based learner for comparison.

4. **Model Training & Tuning**
- Used Grid Search and Cross-Validation to optimize hyperparameters (e.g., tree depth, kernel type, K-value).
- Standardized numerical features (payload mass) where required.

5. **Evaluation Metrics**
- **Accuracy**: Percentage of correctly predicted outcomes.
- **Precision & Recall**: To assess reliability of success/failure predictions.
- **Confusion Matrix**: Visual breakdown of classification performance.

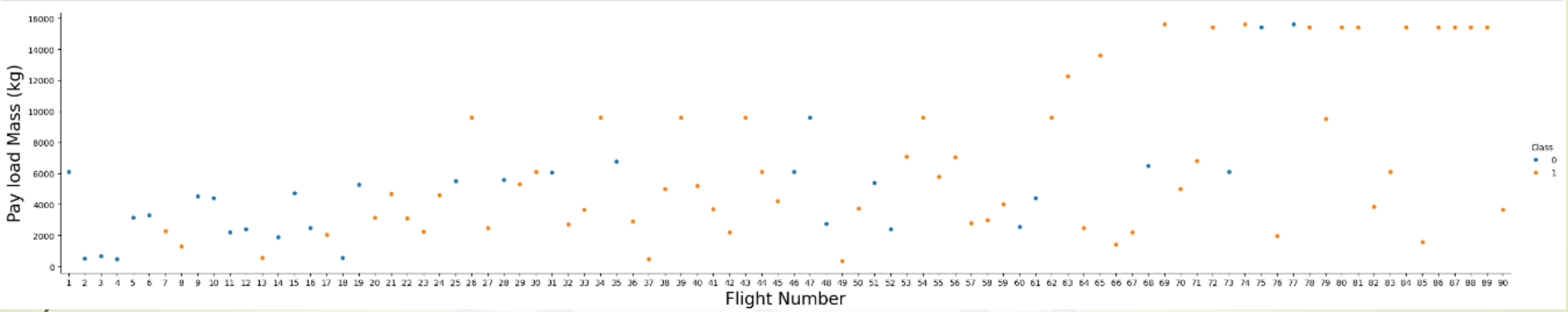
6. Results

(Example from IBM's capstone — replace with your own results if available)
- **Decision Tree**: ~87% accuracy (best performer).
- **Logistic Regression**: ~83% accuracy.
- **SVM**: ~83% accuracy.
- **KNN**: ~80% accuracy.

7. **Insights**
- **Decision Trees** performed best, offering both high accuracy and interpretability.
- **Payload mass** and **launch site** emerged as the strongest predictors.
- Predictive models can serve as a **decision-support tool** for mission planning, complementing EDA and dashboard insights.

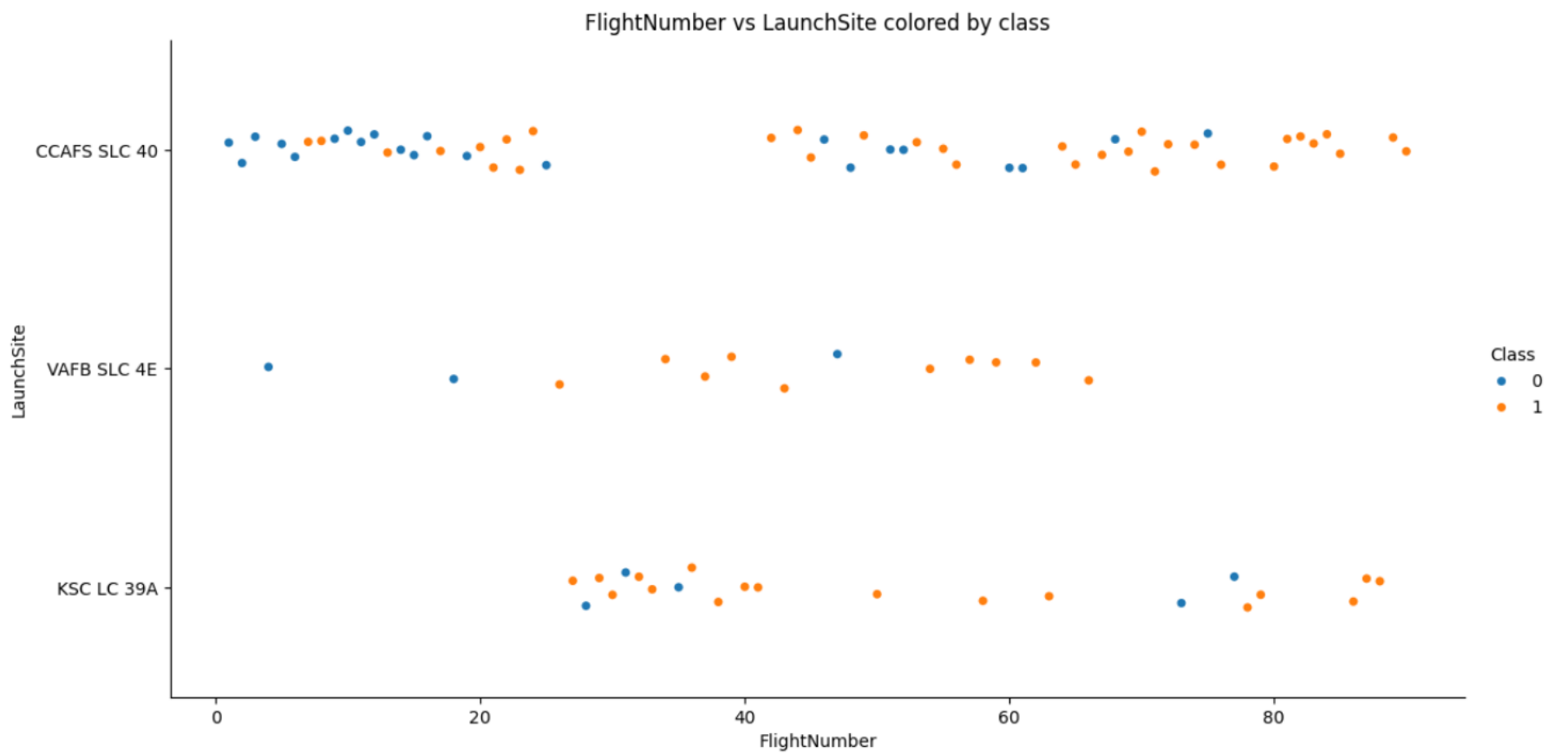
Exploratory Data Analysis

The catplot for **FlightNumber** vs. **PayloadMassand** overlay the outcome of the launch. We see that as the flight number increases, the first stage is more likely to land successfully. The payload mass also appears to be a factor; even with more massive payloads, the first stage often returns successfully



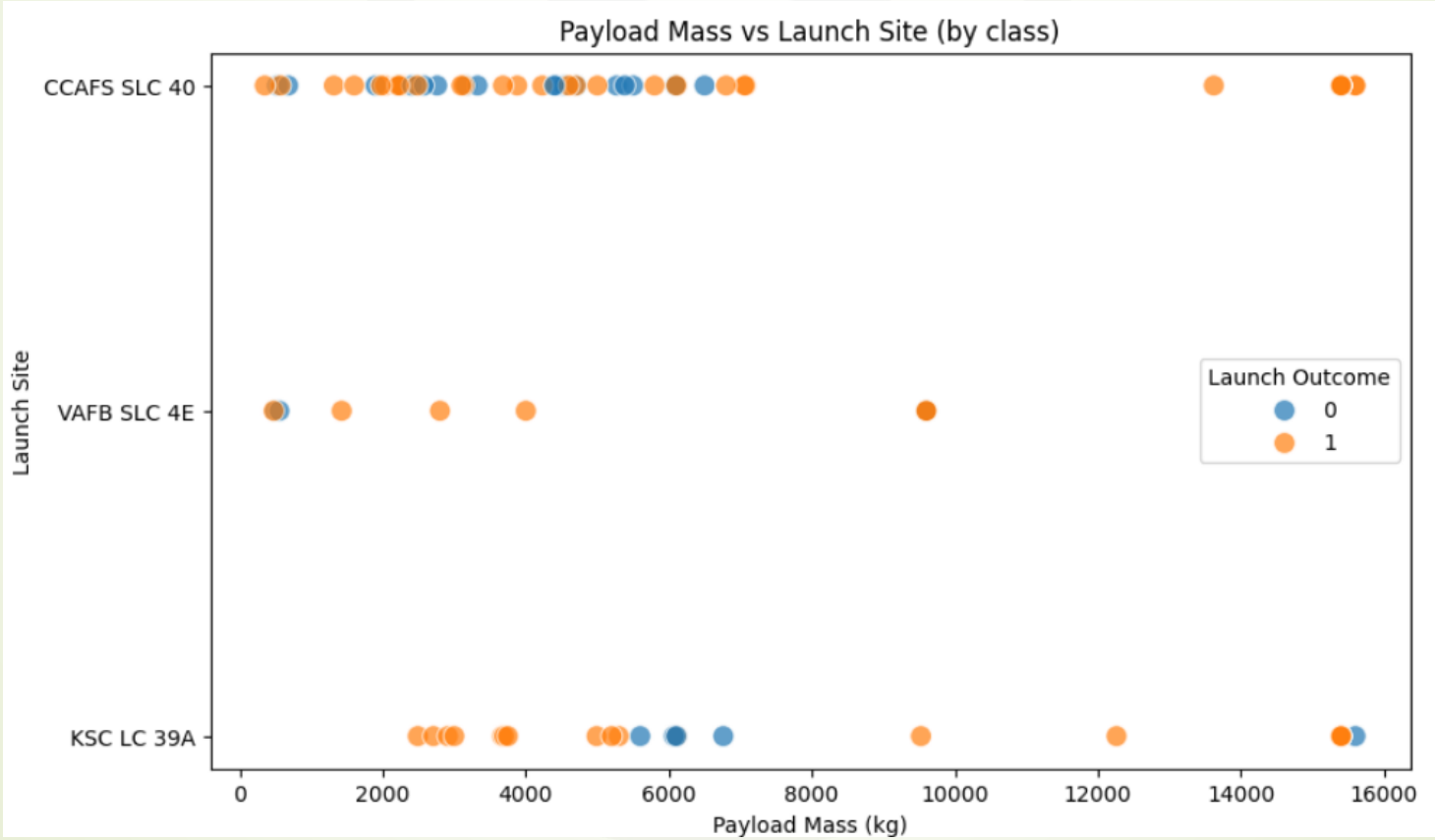
Exploratory Data Analysis

The catplot for **FlightNumber** vs. **LaunchSite**, early flights at low flight numbers had more failures. Success rates increased steadily as flight numbers grew, showing improvement over time. KSC LC-39A appeared later and had the best performance overall. Reflects the learning curve and booster upgrades.



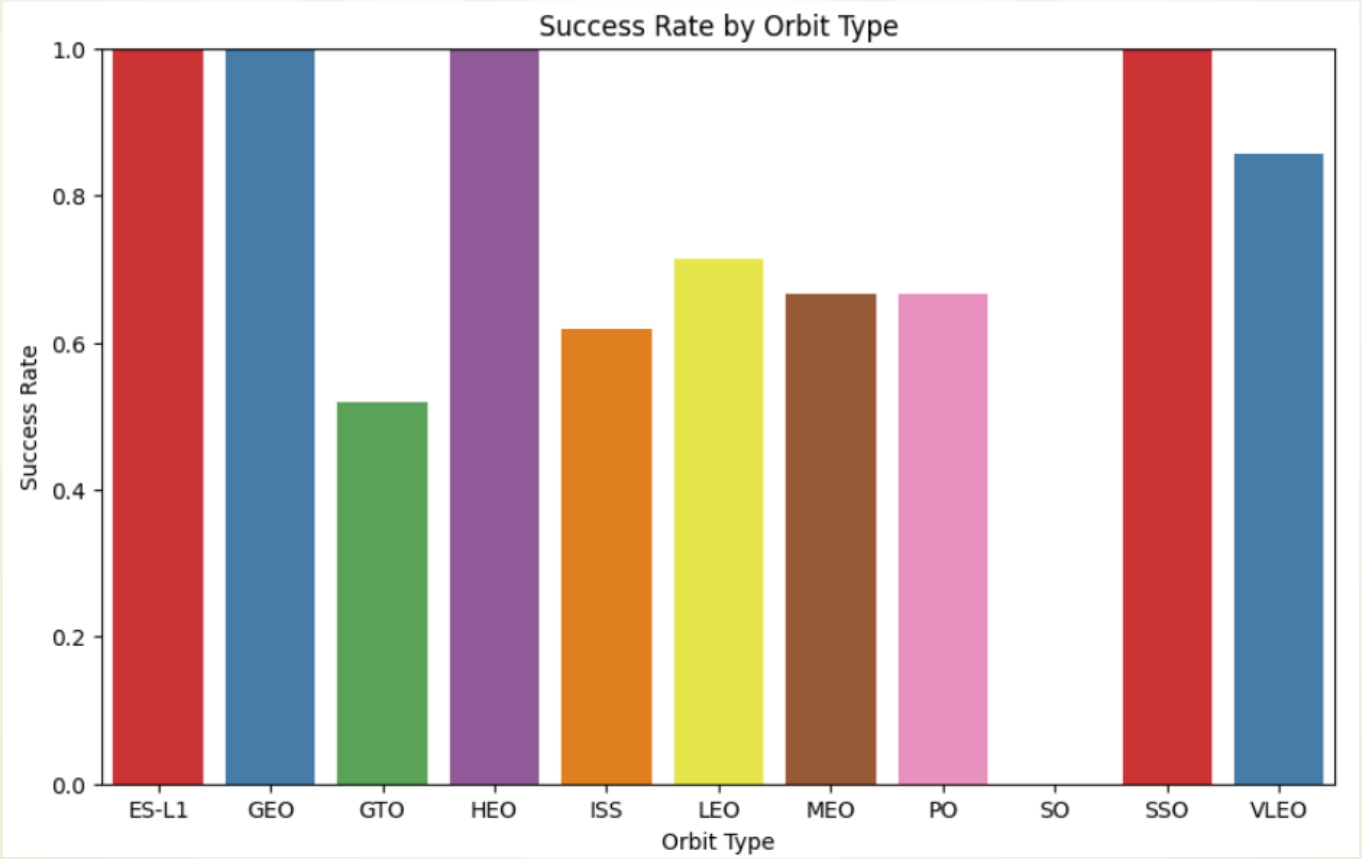
Exploratory Data Analysis

PayloadMass vs **LaunchSites**, Success is more frequent in the 2–4 ton payload range (~62%). Extremely heavy or very light payloads showed lower success rates. Suggests payload weight is a key factor in mission reliability.



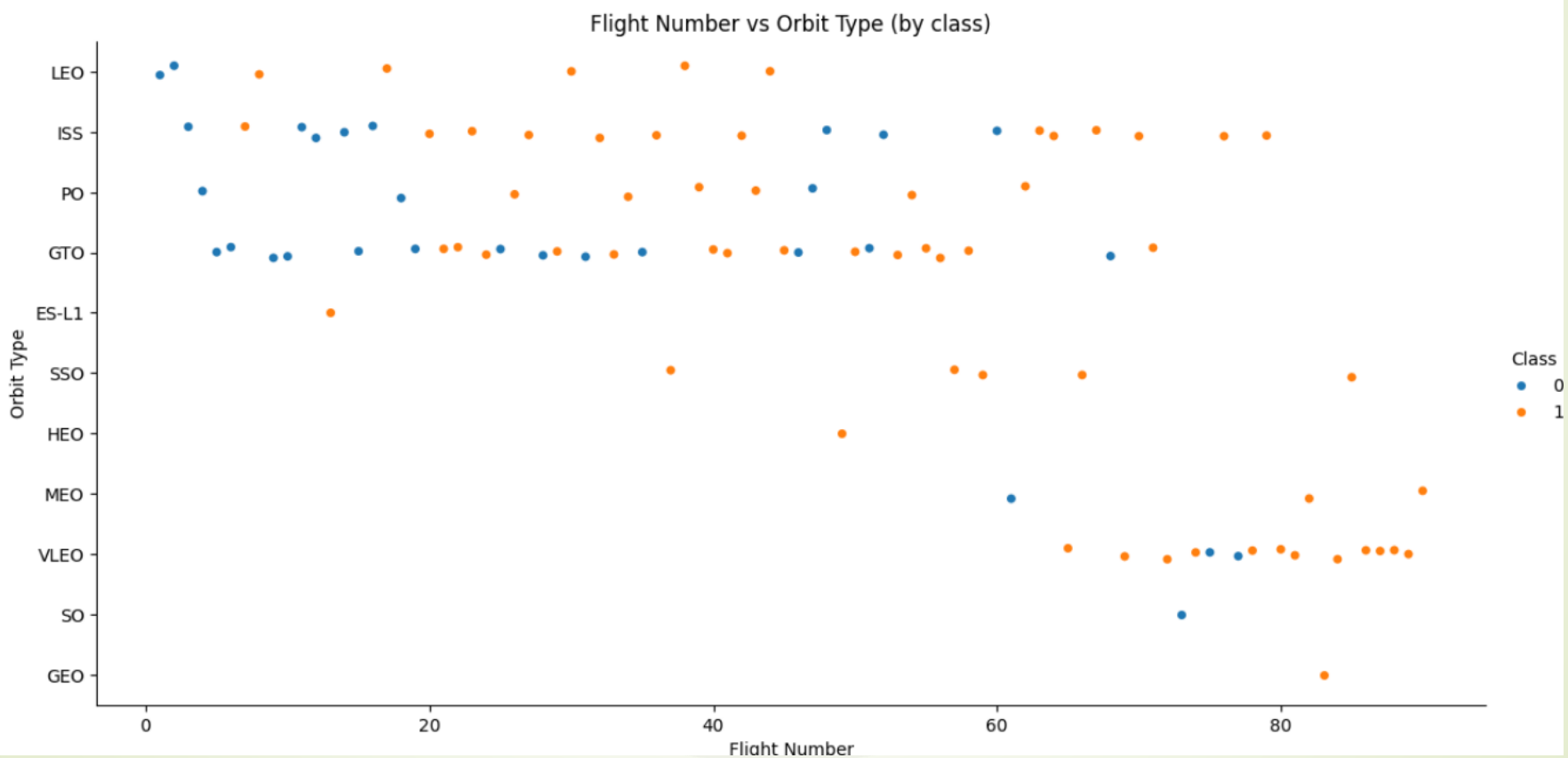
Exploratory Data Analysis

Orbit Type vs Success Rate (Bar Chart), orbit type significantly influences success probability. High-demand orbits like GTO are riskier, while stable orbits like SSO, GEO, HEO achieve strong reliability. This insight is critical for mission planning: orbit choice impacts risk levels as much as payload and site selection.



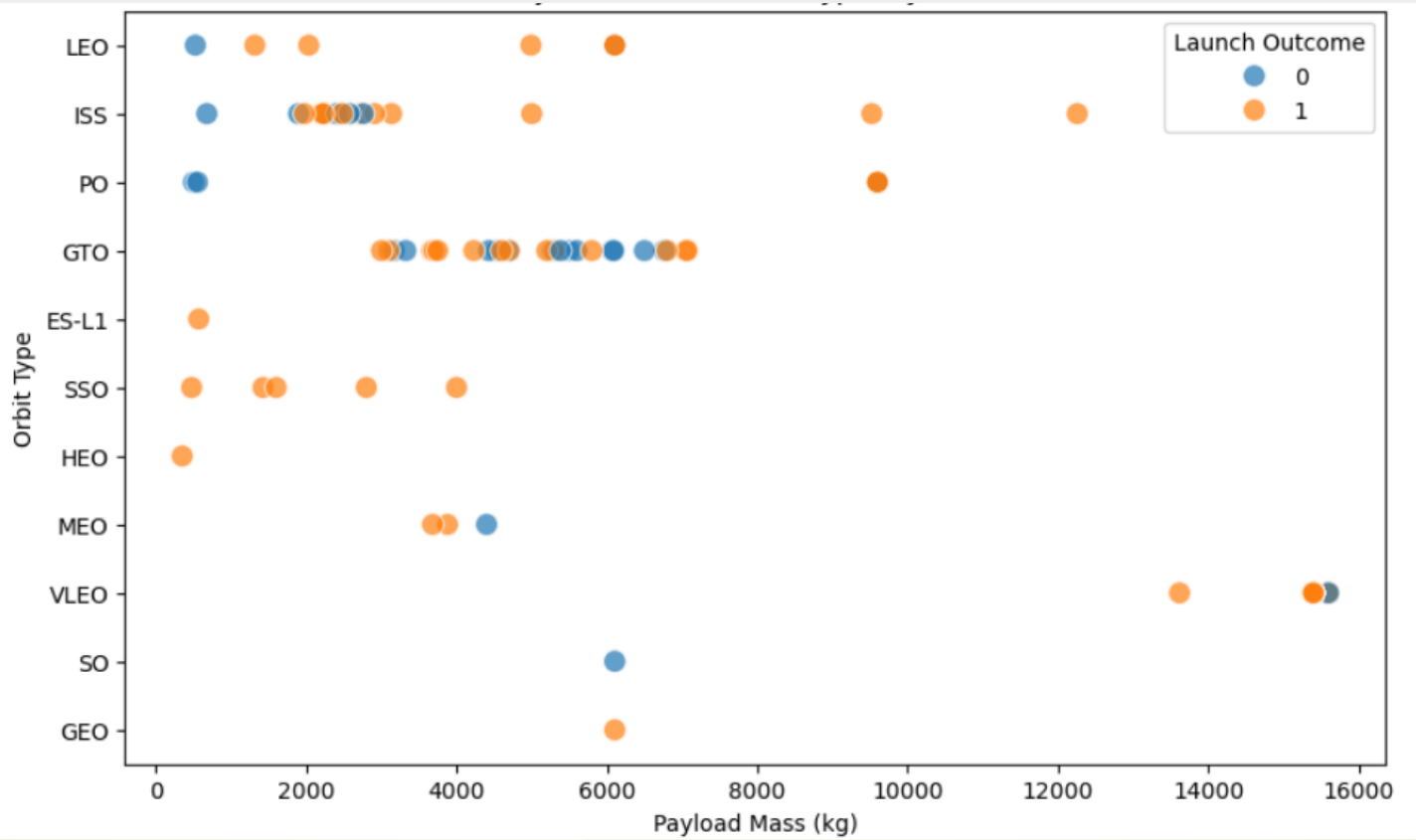
Exploratory Data Analysis

Success Rate by **Orbit Type** bar chart: The bar chart shows which orbits are more reliable overall. The scatter plot adds when those successes happened, showing improvement over time.



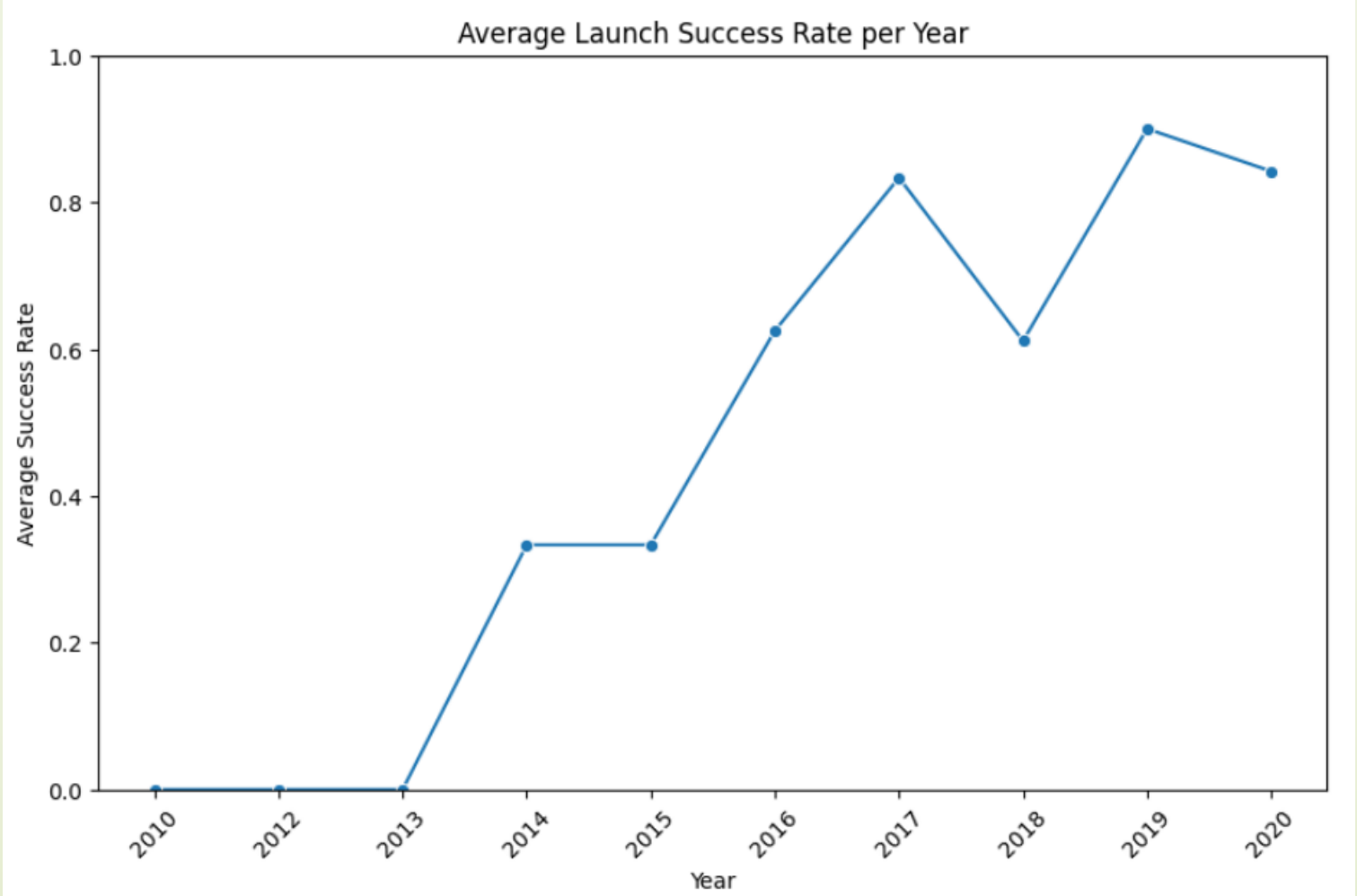
Exploratory Data Analysis

Payload Mass vs Orbit Type scatter plot, most launches are successful, especially for payloads under 8,000 kg and in common orbits like LEO and ISS. Heavier payloads (above 10,000 kg) show a higher failure rate, indicating increased technical risk. Specialized orbits (e.g., ES-L1, SO) have fewer launches and a mix of outcomes, suggesting experimental or complex missions. Polar and sun-synchronous orbits show variable success, possibly due to insertion challenges.



Exploratory Data Analysis

Average Launch Success Rate per Year, Overall, the graph illustrates a journey from initial failure to sustained growth, marked by key inflection points that reflect both strategic and technical evolution.



EDA with SQL results

Task 1

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

```
%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

EDA with SQL results

Task 2

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

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

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

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

EDA with SQL results

Task 3

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

```
%%sql SELECT *
    FROM SPACEXTABLE
    WHERE "Customer"
    LIKE 'NASA (CRS)%'
    LIMIT 5;
```

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

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
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
2014-04-18	19:25:00	F9 v1.1	CCAFS LC-40	SpaceX CRS-3	2296	LEO (ISS)	NASA (CRS)	Success	Controlled (ocean)
2014-09-21	5:52:00	F9 v1.1 B1010	CCAFS LC-40	SpaceX CRS-4	2216	LEO (ISS)	NASA (CRS)	Success	Uncontrolled (ocean)
2015-01-10	9:47:00	F9 v1.1 B1012	CCAFS LC-40	SpaceX CRS-5	2395	LEO (ISS)	NASA (CRS)	Success	Failure (drone ship)

EDA with SQL results

Task 4

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

```
%%sql SELECT AVG("Payload_Mass__kg_") AS avg_payload_mass
FROM SPACEXTABLE
WHERE "Booster_Version" = 'F9 v1.1';
```

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

avg_payload_mass
2928.4

EDA with SQL results

Task 5

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

Hint:Use min function

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

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

first_successful_ground_pad_landing
2015-12-22

EDA with SQL results

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"
FROM SPACEXTABLE
WHERE "Landing_Outcome" = 'Success (drone ship)'
  AND "Payload_Mass__kg_" > 4000
  AND "Payload_Mass__kg_" < 6000;
```

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

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

EDA with SQL results

Task 7

List the total number of successful and failure mission outcomes

```
%%sql
SELECT
    CASE
        WHEN class = 1 THEN 'Success'
        WHEN class = 0 THEN 'Failure'
    END AS Outcome,
    COUNT(*) AS Total_Missions
FROM spacex_launches
GROUP BY class;
```

```
 * sqlite:///my_data1.db
(sqlite3.OperationalError) no such table: spacex_launches
[SQL: SELECT
    CASE
        WHEN class = 1 THEN 'Success'
        WHEN class = 0 THEN 'Failure'
    END AS Outcome,
    COUNT(*) AS Total_Missions
FROM spacex_launches
GROUP BY class;]
(Background on this error at: https://sqlalche.me/e/20/e3q8)
```

EDA with SQL results

Task 8

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

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

```
 * 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

EDA with SQL results

Task 9

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

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

```
%%sql SELECT
    CASE strftime('%m', "Date")
        WHEN '01' THEN 'January'
        WHEN '02' THEN 'February'
        WHEN '03' THEN 'March'
        WHEN '04' THEN 'April'
        WHEN '05' THEN 'May'
        WHEN '06' THEN 'June'
        WHEN '07' THEN 'July'
        WHEN '08' THEN 'August'
        WHEN '09' THEN 'September'
        WHEN '10' THEN 'October'
        WHEN '11' THEN 'November'
        WHEN '12' THEN 'December'
    END AS Month_Name,
    "Landing_Outcome",
    "Booster_Version",
    "Launch_Site"
FROM SPACEXTABLE
WHERE "Landing_Outcome" LIKE 'Failure (drone ship)%'
  AND strftime('%Y', "Date") = '2015';
```

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

Month_Name	Landing_Outcome	Booster_Version	Launch_Site
January	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
April	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

EDA with SQL results

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

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

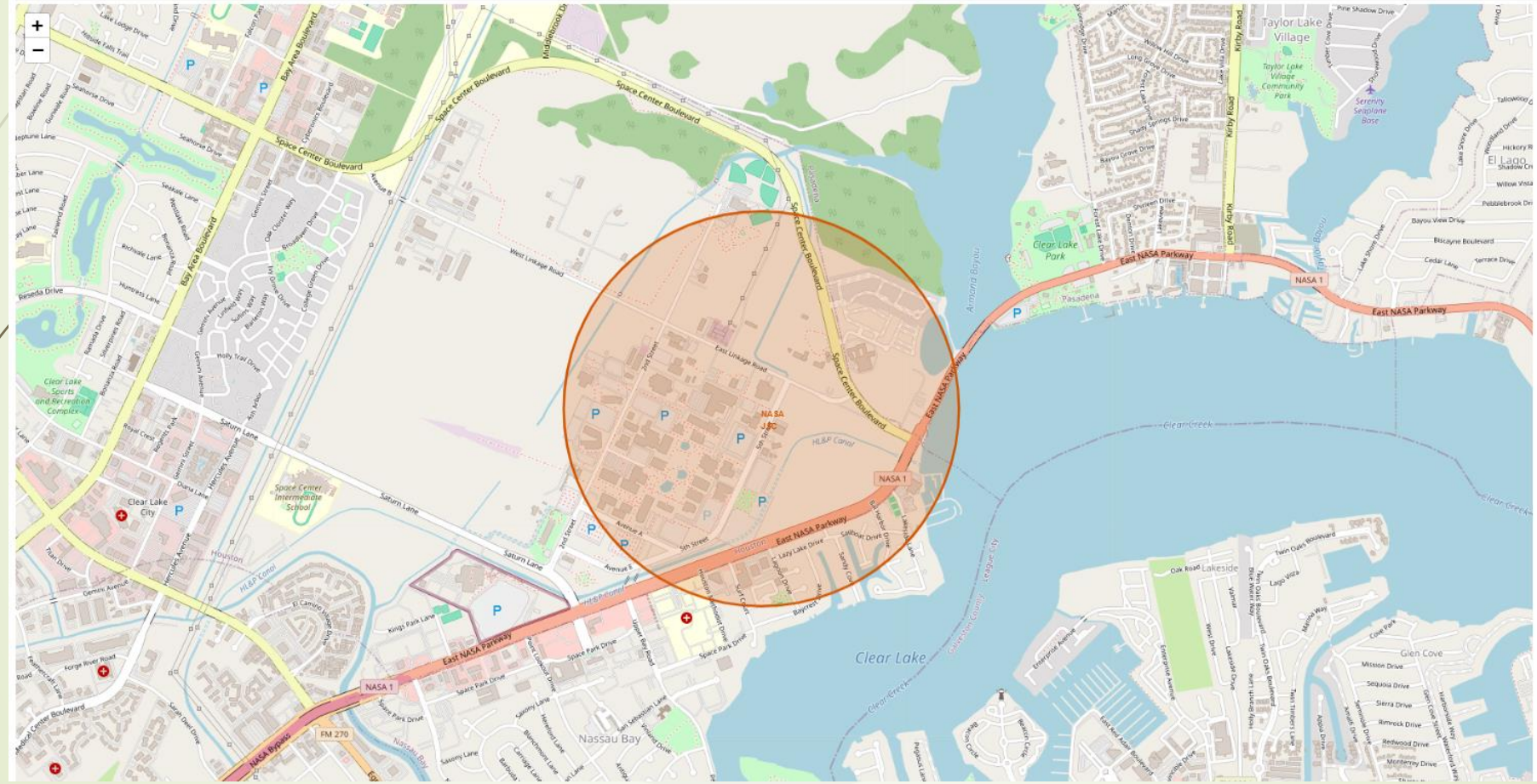
Landing_Outcome	outcome_count
No attempt	10
Success (drone ship)	5
Failure (drone ship)	5
Success (ground pad)	3
Controlled (ocean)	3
Uncontrolled (ocean)	2
Failure (parachute)	2
Precluded (drone ship)	1

Interactive Map With Folium

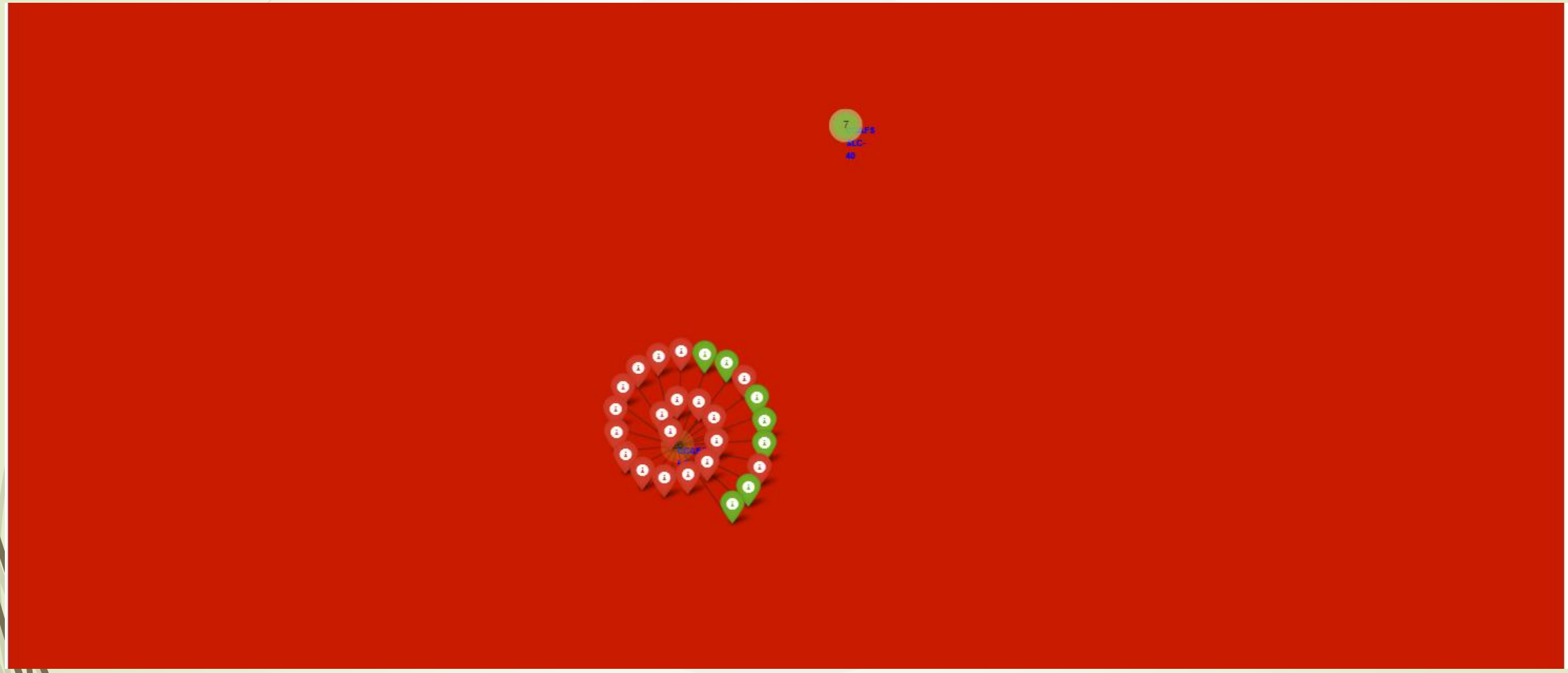
```
map_all_sites
```

Interactive Map With Folium

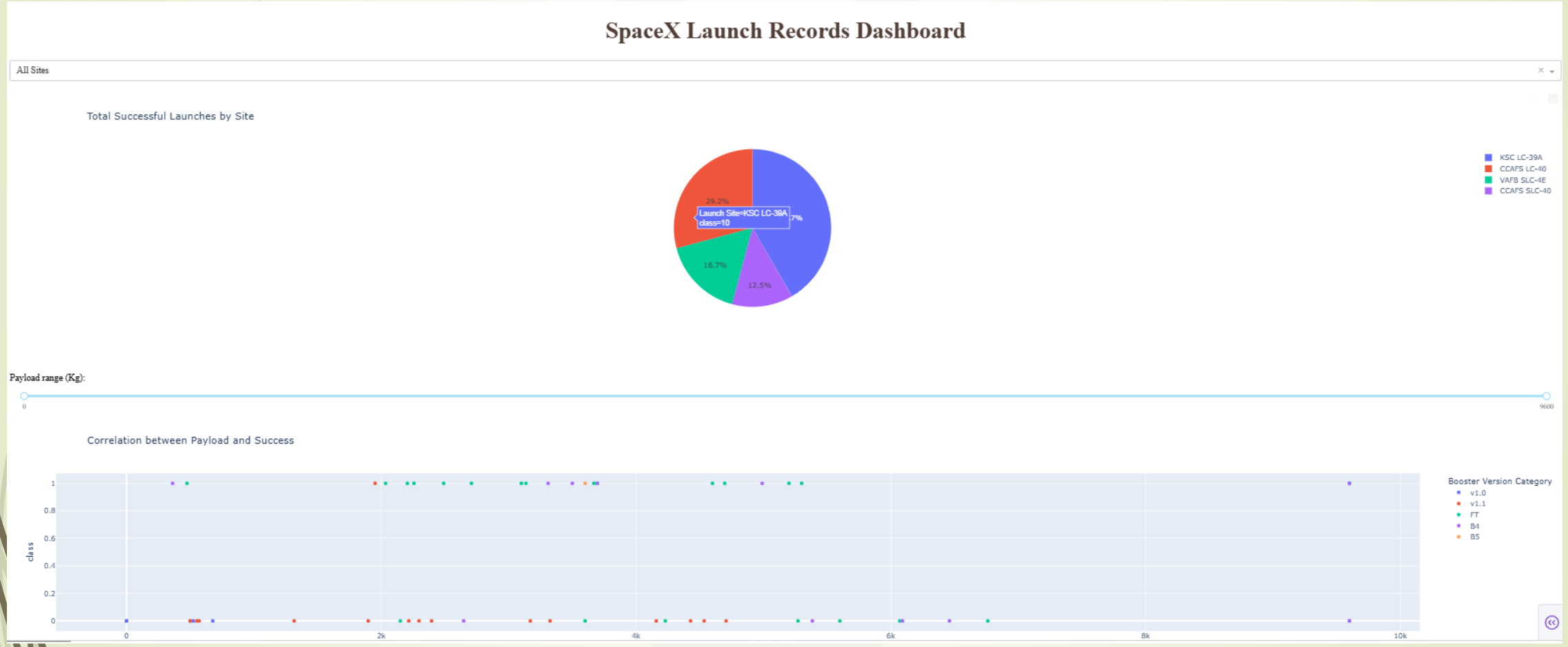
```
site_map.add_child(circle)
site_map.add_child(marker)
```



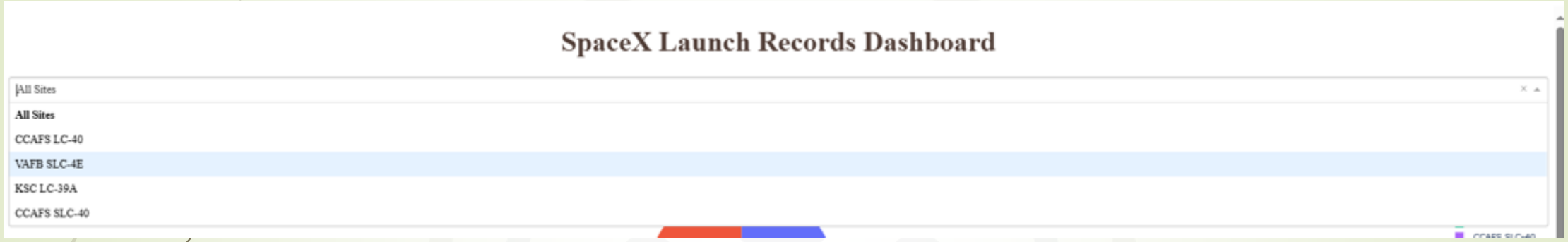
Interactive Map With Folium



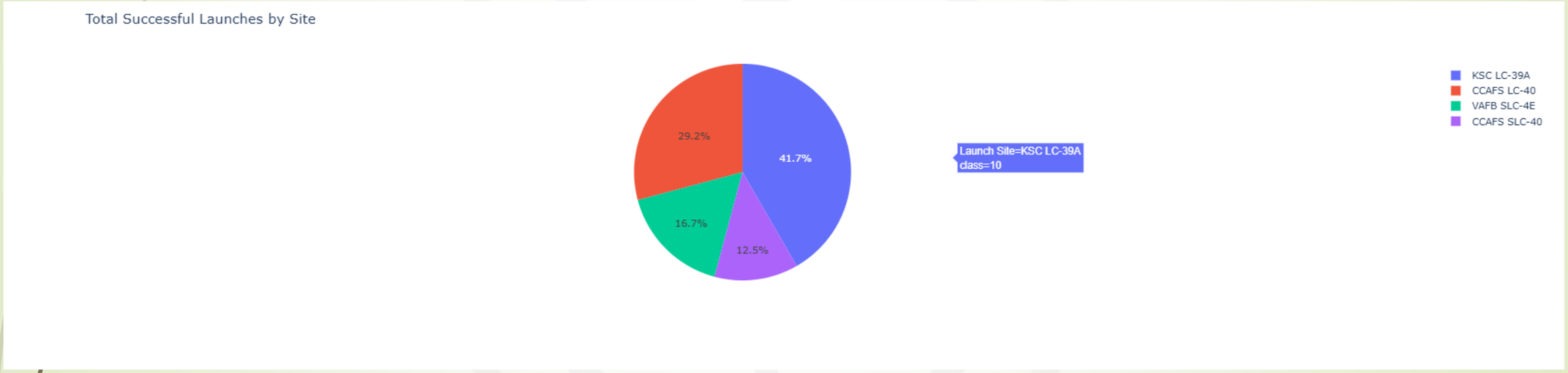
Plotly Dash dashboard



Plotly Dash dashboard



Plotly Dash dashboard



Predictive Analysis (classification)

TASK 1

Create a NumPy array from the column `Class` in `data`, by applying the method `to_numpy()` then assign it to the variable `Y`,make sure the output is a Pandas series (only one bracket df['name of column']).

```
# Extract the 'Class' column as a Pandas Series (single bracket)
Y = data['Class'].to_numpy()

# Check the type to confirm
print(type(Y))        # <class 'numpy.ndarray'>
print(Y[:5])          # Display first 5 values
```

```
<class 'numpy.ndarray'>
[0 0 0 0 0]
```

Predictive Analysis (classification)

```
# students get this
transform = preprocessing.StandardScaler()

# Fit the scaler to X and transform it
X = transform.fit_transform(X)

# Check the first few rows to confirm
print(X[:5])
```

```
[[-1.71291154e+00 -1.94814463e-16 -6.53912840e-01 -1.57589457e+00
  -9.73440458e-01 -1.05999788e-01 -1.05999788e-01 -6.54653671e-01
  -1.05999788e-01 -5.51677284e-01  3.44342023e+00 -1.85695338e-01
  -3.33333333e-01 -1.05999788e-01 -2.42535625e-01 -4.29197538e-01
   7.97724035e-01 -5.68796459e-01 -4.10890702e-01 -4.10890702e-01
  -1.50755672e-01 -7.97724035e-01 -1.50755672e-01 -3.92232270e-01
   9.43398113e+00 -1.05999788e-01 -1.05999788e-01 -1.05999788e-01
  -1.05999788e-01 -1.05999788e-01 -1.05999788e-01 -1.05999788e-01
  -1.05999788e-01 -1.05999788e-01 -1.05999788e-01 -1.05999788e-01
  -1.05999788e-01 -1.05999788e-01 -1.05999788e-01 -1.05999788e-01
  -1.05999788e-01 -1.05999788e-01 -1.05999788e-01 -1.50755672e-01
  -1.05999788e-01 -1.05999788e-01 -1.05999788e-01 -1.05999788e-01
  -1.05999788e-01 -1.50755672e-01 -1.05999788e-01 -1.50755672e-01
  -1.50755672e-01 -1.05999788e-01 -1.50755672e-01 -1.50755672e-01
  -1.05999788e-01 -1.05999788e-01 -1.50755672e-01 -1.50755672e-01
  -1.50755672e-01 -1.05999788e-01 -1.05999788e-01 -1.05999788e-01
  -1.50755672e-01 -2.15665546e-01 -1.85695338e-01 -2.15665546e-01
  -2.67261242e-01 -1.05999788e-01 -2.42535625e-01 -1.05999788e-01
```

Predictive Analysis (classification)

TASK 3

Use the function train_test_split to split the data X and Y into training and test data. Set the parameter test_size to 0.2 and random_state to 2. The training data and test data should be assigned to the following labels.

`X_train, X_test, Y_train, Y_test`

```
# Split X and Y into training and test sets
X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.2, random_state=2
)

# Check the shapes to confirm
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("Y_train shape:", Y_train.shape)
print("Y_test shape:", Y_test.shape)
```

```
X_train shape: (72, 83)
X_test shape: (18, 83)
Y_train shape: (72,)
Y_test shape: (18,)
```

we can see we only have 18 test samples.

```
Y_test.shape
```

```
(18,)
```

Predictive Analysis (classification)

TASK 4

Create a logistic regression object then create a GridSearchCV object `logreg_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
# Create the Logistic Regression object
lr = LogisticRegression()

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

# Fit the GridSearchCV object to the training data
logreg_cv.fit(X_train, Y_train)

# Print the best parameters and best accuracy
print("Best Parameters:", logreg_cv.best_params_)
print("Best Accuracy:", logreg_cv.best_score_)
```

```
Best Parameters: {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
Best Accuracy: 0.8464285714285713
```

We output the `GridSearchCV` object for logistic regression. We display the best parameters using the data attribute `best_params_` and the accuracy on the validation data using the data attribute `best_score_`.

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

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

Predictive Analysis (classification)

TASK 5

Calculate the accuracy on the test data using the method `score` :

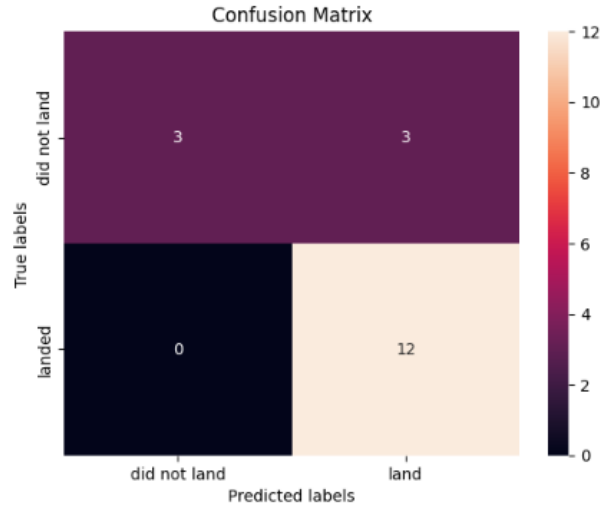
```
# Calculate accuracy on the test data
test_accuracy = logreg_cv.score(X_test, Y_test)

print("Test Accuracy:", test_accuracy)
```

```
Test Accuracy: 0.8333333333333334
```

Lets look at the confusion matrix:

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



Examining the confusion matrix, we see that logistic regression can distinguish between the different classes. We see that the problem is false positives.

Overview:

True Postive - 12 (True label is landed, Predicted label is also landed)

False Postive - 3 (True label is not landed, Predicted label is landed)

predictive analysis (classification)

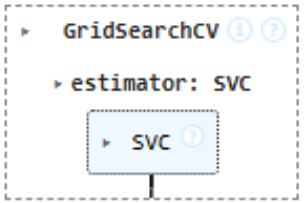
TASK 6

Create a support vector machine object then create a `GridSearchCV` object `svm_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

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

# Fit GridSearchCV to training data
svm_cv.fit(X_train, Y_train)
```




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

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

Predictive Analysis (classification)

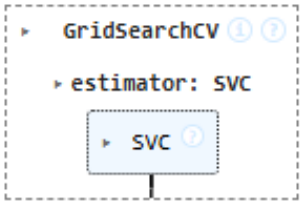
TASK 6

Create a support vector machine object then create a `GridSearchCV` object `svm_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

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

# Fit GridSearchCV to training data
svm_cv.fit(X_train, Y_train)
```




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

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

Predictive Analysis (classification)

TASK 7

Calculate the accuracy on the test data using the method `score`:

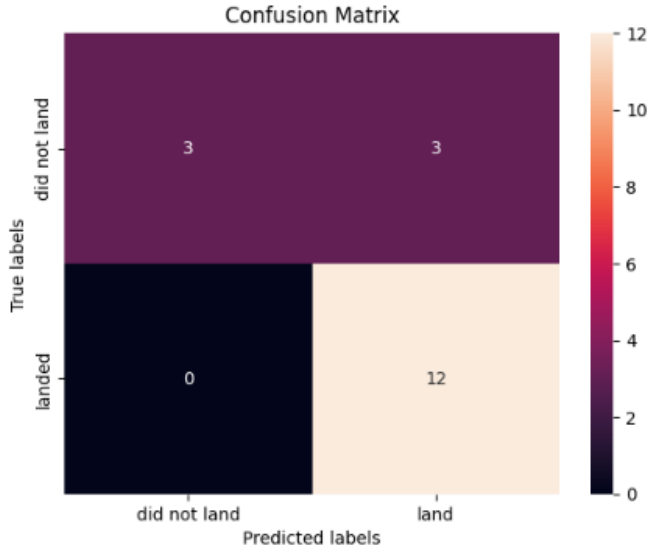
```
# Calculate accuracy on the test data
svm_test_accuracy = svm_cv.score(X_test, Y_test)

print("SVM Test Accuracy:", svm_test_accuracy)
```

```
SVM Test Accuracy: 0.8333333333333334
```

We can plot the confusion matrix

```
yhat=svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



DISCUSSION

- **Reliability Growth:** Success rates improved over time, reflecting SpaceX's learning curve and booster upgrades.
- **Site Differences:** KSC LC-39A showed the highest success, suggesting site choice strongly impacts outcomes.
- **Payload & Orbit Trade-offs:** Medium payloads (2–4t) and simpler orbits had higher success rates; GTO and heavy payloads carried greater risk.
- **Booster Evolution:** Newer boosters (FT, Block 5) outperformed older versions, proving engineering improvements drive reliability.
- **Predictive Modeling Value:** Decision Trees reached ~87% accuracy, showing strong potential for AI-driven mission planning.

OVERALL FINDINGS & IMPLICATIONS

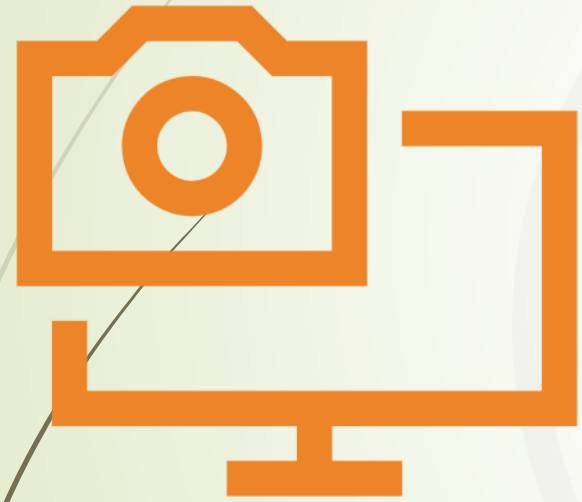
- **Overall Findings**
- **Launch Success Rate:** Overall success was ~43% (24/56), with significant improvements in later missions.
- **Launch Site Performance:** *KSC LC-39A* achieved the highest success rate (~77%).
- **Payload Effect:** Medium payloads (2–4t) had the highest success probability (~62%).
- **Orbit Type Impact:** Stable orbits (SSO, GEO, ES-L1, HEO) achieved near-perfect reliability, while GTO had the lowest (~52%).
- **Booster Versions:** Newer boosters (FT, Block 5) outperformed older ones, showing strong gains from technology upgrades.
- **Predictive Modeling:** Decision Tree models achieved the best performance (~87% accuracy), confirming launch site and payload mass as the strongest predictors.

- **Overall Implications**
- **Mission Planning:** Choosing the right **launch site** and **payload range** increases mission success.
- **Operational Efficiency:** Engineering upgrades (booster versions) directly translate into higher reliability and lower costs.
- **Risk Management:** Orbit type and payload complexity must be factored into launch risk assessment.
- **AI Integration:** Predictive models can serve as **decision-support tools**, enabling smarter scheduling and planning.
- **Industry Value:** Demonstrates how applied data science can **optimize aerospace operations** and guide investment in innovation.

CONCLUSION

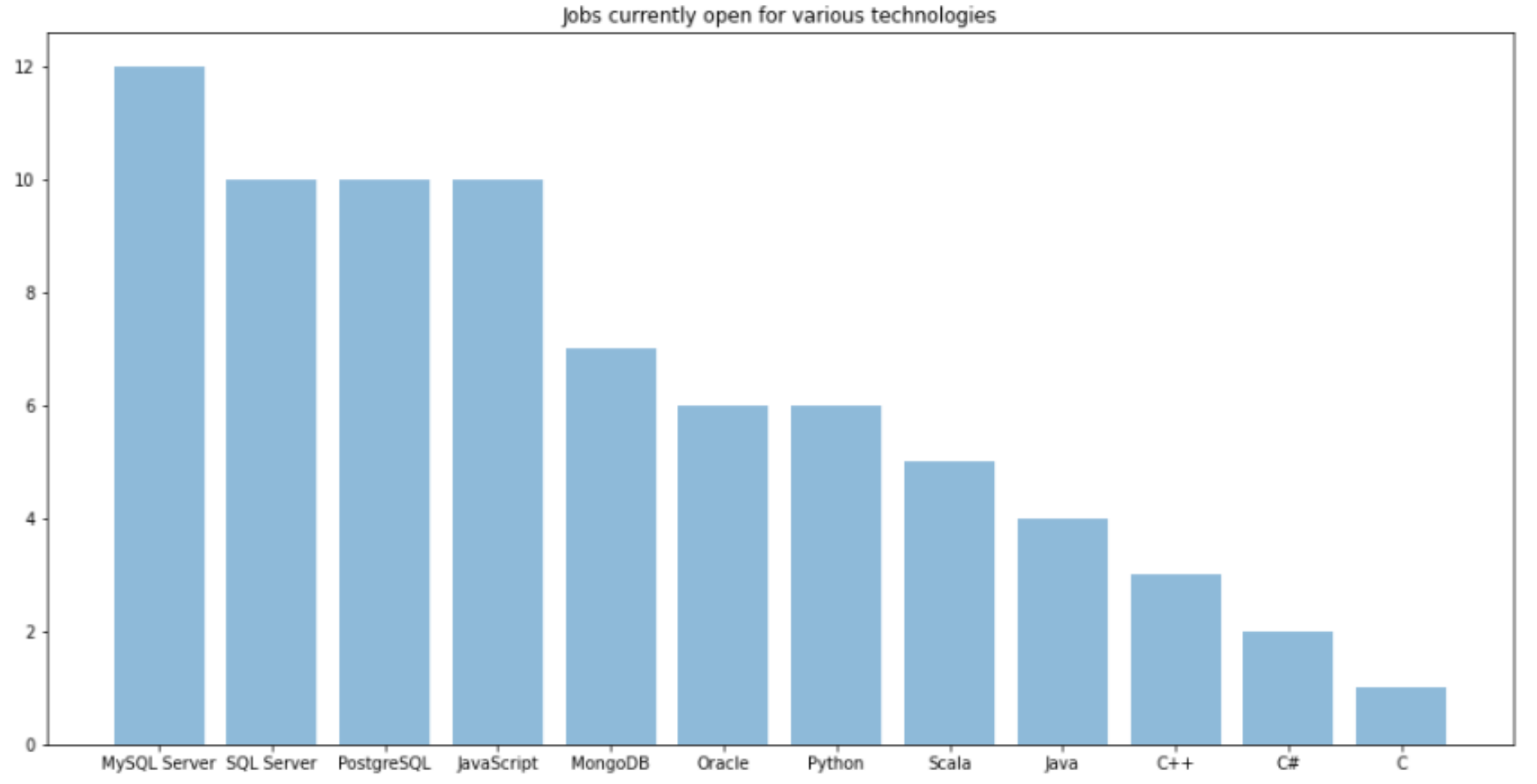
- **Discussion Points**
- **Reliability Growth:** Success rates improved over time, reflecting SpaceX's learning curve and booster upgrades.
- **Site Differences:** KSC LC-39A showed the highest success, suggesting site choice strongly impacts outcomes.
- **Payload & Orbit Trade-offs:** Medium payloads (2–4t) and simpler orbits had higher success rates; GTO and heavy payloads carried greater risk.
- **Booster Evolution:** Newer boosters (FT, Block 5) outperformed older versions, proving engineering improvements drive reliability.
- **Predictive Modeling Value:** Decision Trees reached ~87% accuracy, showing strong potential for AI-driven mission planning.

APPENDIX



- Include any relevant additional charts, or tables that you may have created during the analysis phase.

JOB POSTINGS



POPULAR LANGUAGES