IBM Developer
SKILLS NETWORK

IBM Data Science Capstone Project - SpaceX Rocket Launches

Gerardo Torres

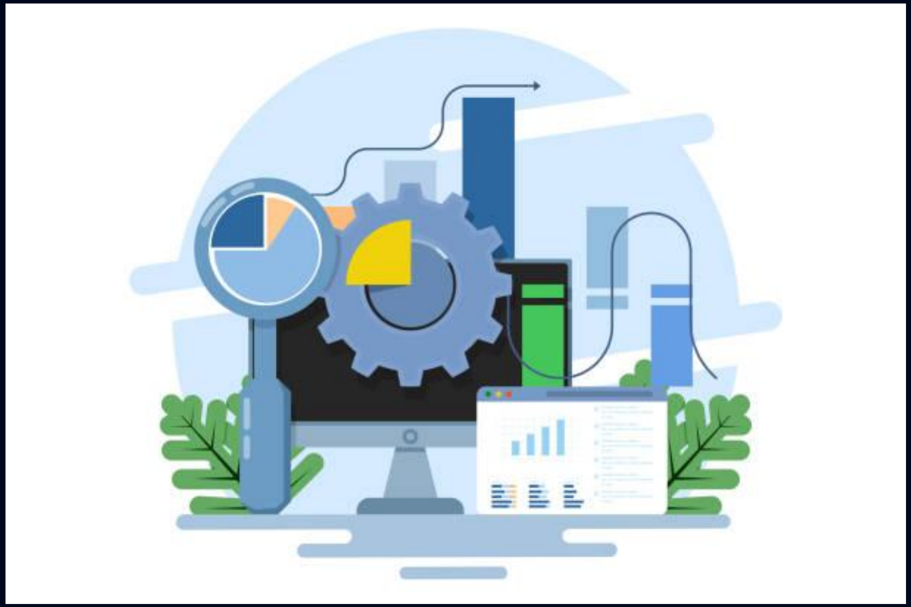
11 November 2024

Outline

- Executive Summary
- Introduction
- Methodology
- Results
- Conclusion

Executive Summary

- **Summary of Methodologies**
  - Data Collection
  - Data Wrangling
  - EDA with SQL
  - EDA with Data Visualization
  - Interactive Map using Folium
  - Interactive Dashboard using Plotly
  - Machine Learning Predictions
- **Summary of all results**
  - Exploratory Data Analysis results
  - Interactive Analytics results - screenshots
  - Predictive Analytics results



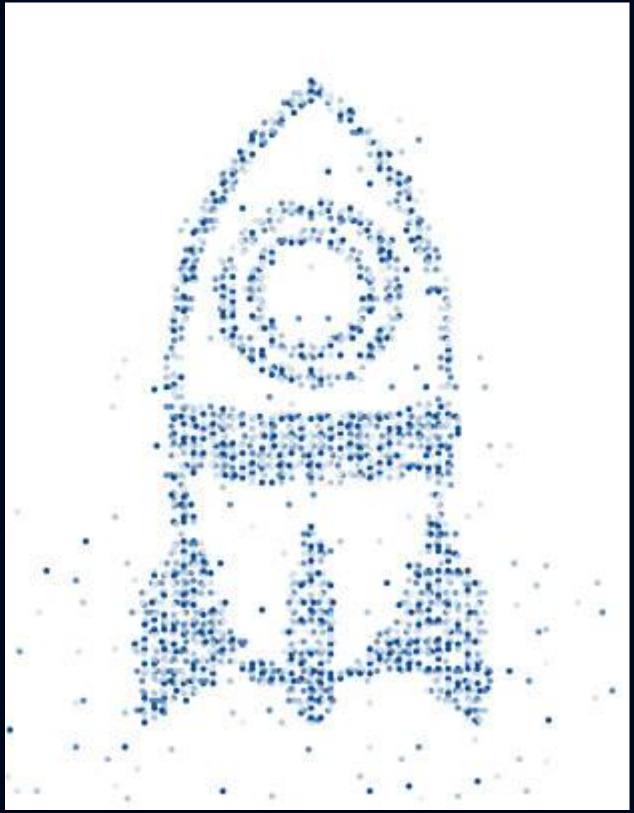
Introduction

- **Project background and context**

  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. The goal of this project is to create a machine learning pipeline to predict if the first stage will land successfully.

- **Problems to Answer**

  - Influences of a successful launch
  - The effects each rocket feature has on success rate of landing
  - Optimal conditions SpaceX must achieve best results and unsure highest successful rocket landing rate

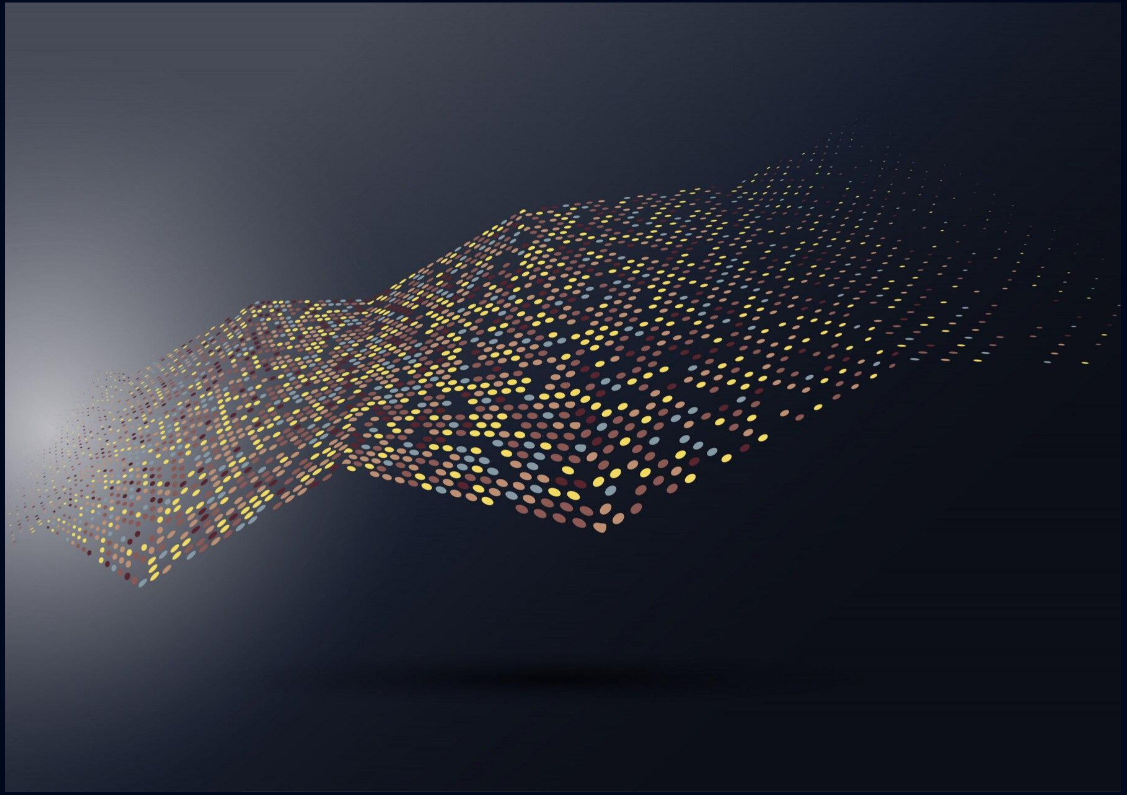


Section 1

Methodology

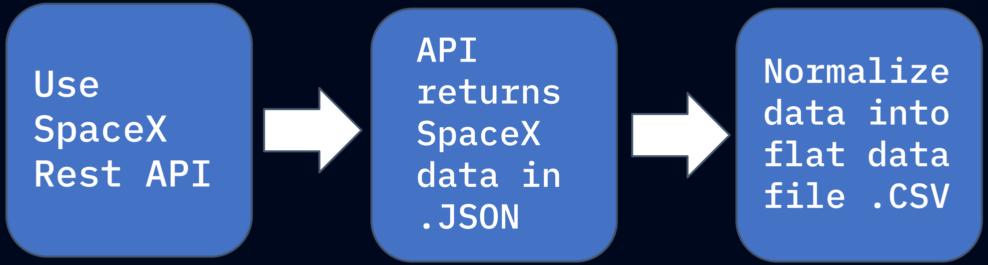
Executive Summary

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

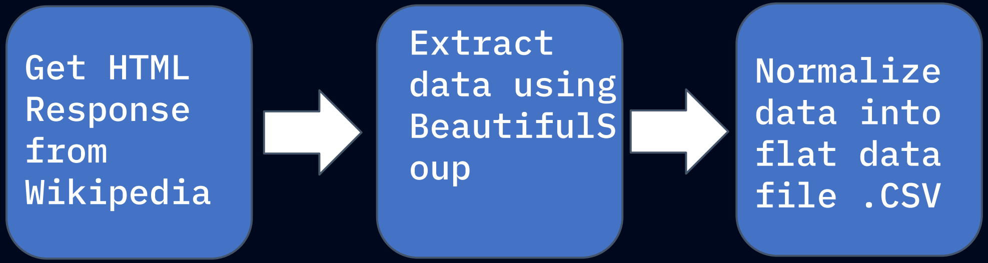


Data Collection

- SpaceX API
  - Request the SpaceX launch data using the GET request
  - Decode the response content as a Json using .json() and turn it into a Pandas dataframe using .json_normalize()
  - Cleaned data by checking for missing values and filling in values where necessary


Use SpaceX Rest API
API returns SpaceX data in .JSON
Normalize data into flat data file .CSV

- Web Scraping - Wikipedia
  - Web scraping from Wikipedia HTML tables using BeautifulSoup to obtain Falcon 9 Launch records
  - Parce the launch HTML tables and convert them to Pandas Dataframe
  - Save and export as .CSV

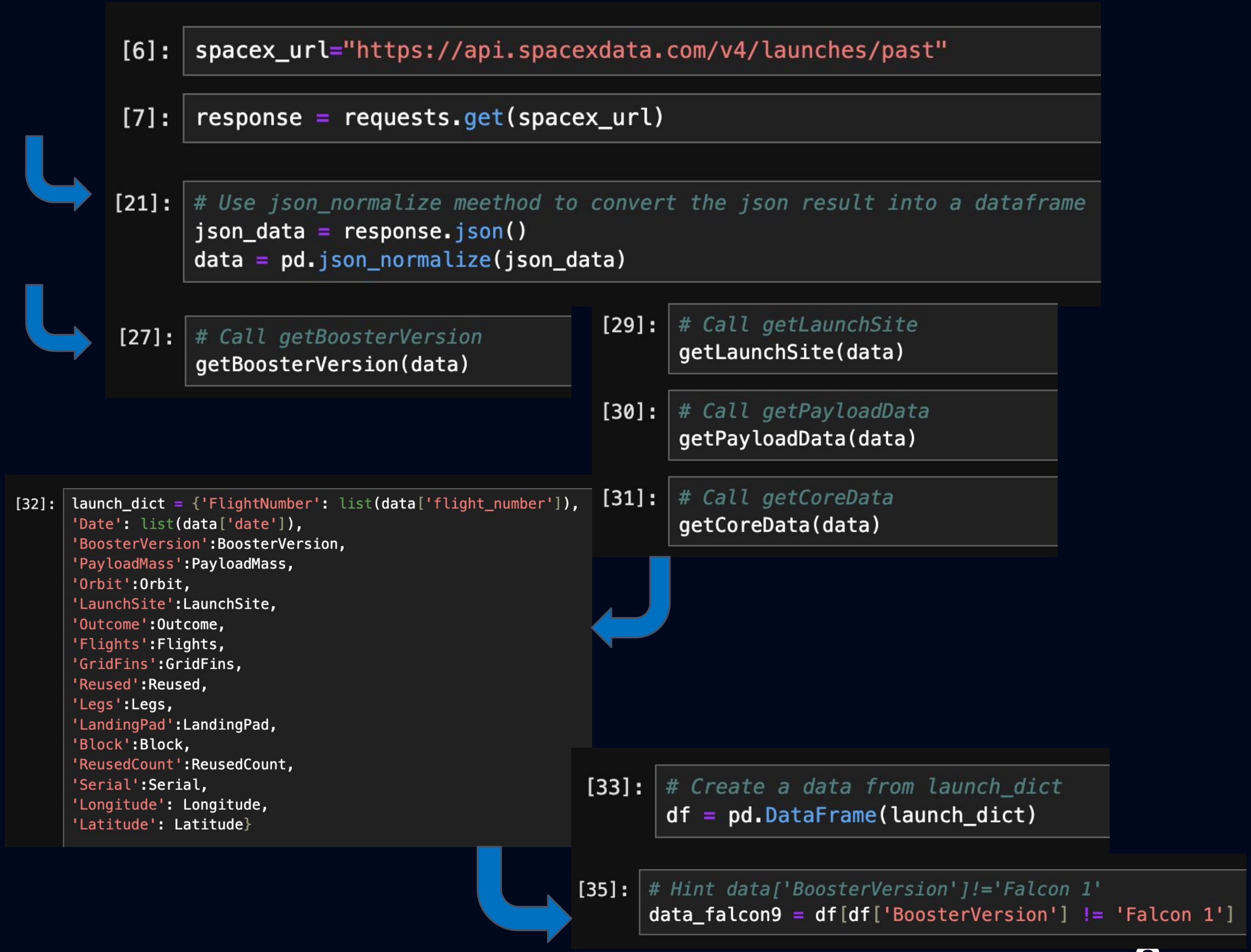

Get HTML Response from Wikipedia
Extract data using BeautifulSoup
Normalize data into flat data file .CSV

Data Collection – SpaceX API

1. Getting Response from API
2. Converting Response to .JSON file
3. Apply custom functions to clean data
4. Assign list functions to clean data
5. Filter dataframe and export to flat file (.CSV)

<u>Github Notebook URL:</u>

https://github.com/gn-wy/IBM-data-science-capstone/blob/main/jupyter-labs-spacex-data-collection-api_1-1.ipynb



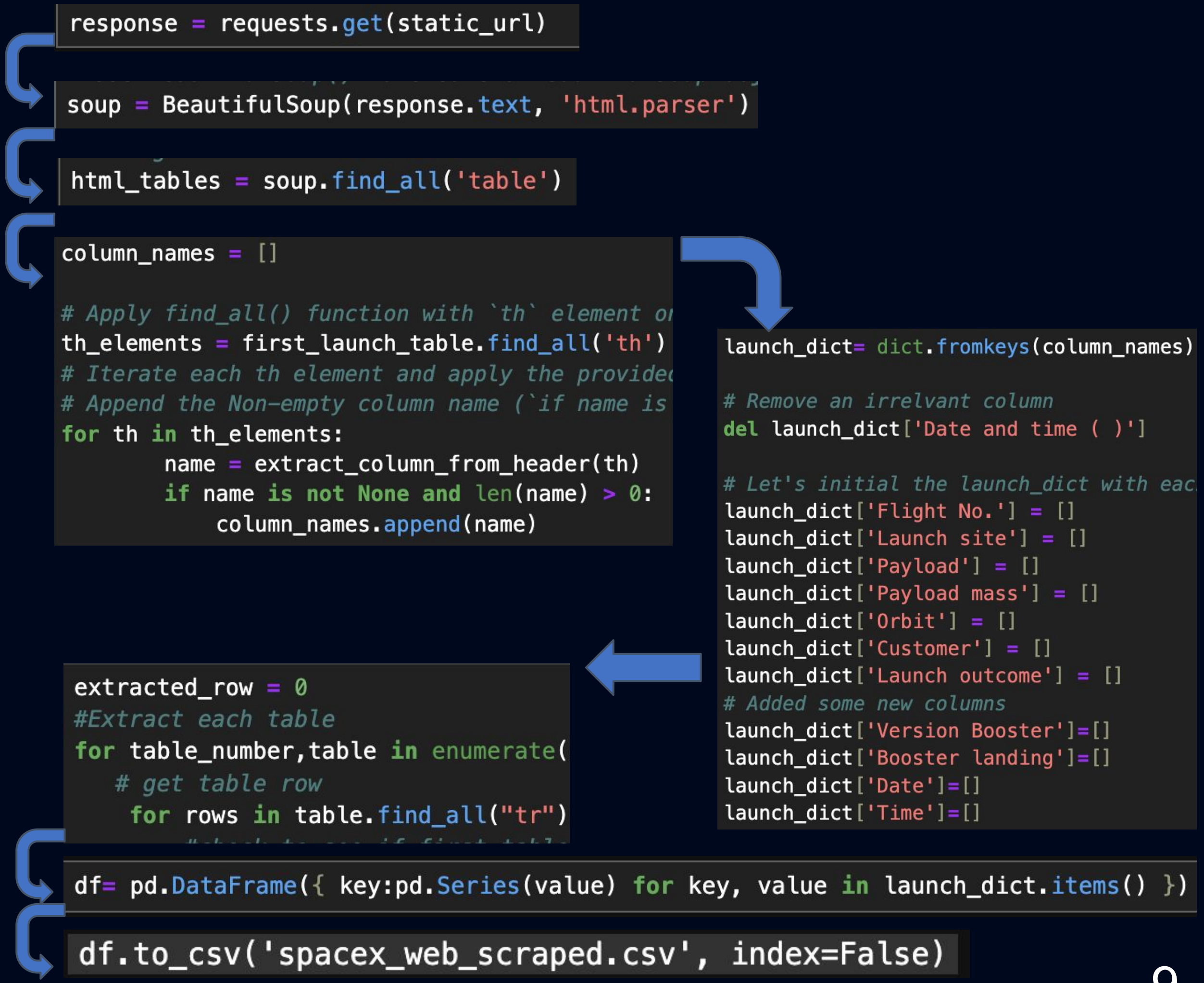
```
[6]: spacex_url="https://api.spacexdata.com/v4/launches/past"

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

[21]: # Use json_normalize meethod to convert the json result into a dataframe
      json_data = response.json()
      data = pd.json_normalize(json_data)

[27]: # Call getBoosterVersion
      getBoosterVersion(data)

[29]: # Call getLaunchSite
      getLaunchSite(data)

[30]: # Call getPayloadData
      getPayloadData(data)

[31]: # Call getCoreData
      getCoreData(data)

[32]: launch_dict = {'FlightNumber': list(data['flight_number']),
      'Date': list(data['date']),
      'BoosterVersion':BoosterVersion,
      'PayloadMass':PayloadMass,
      'Orbit':Orbit,
      'LaunchSite':LaunchSite,
      'Outcome':Outcome,
      'Flights':Flights,
      'GridFins':GridFins,
      'Reused':Reused,
      'Legs':Legs,
      'LandingPad':LandingPad,
      'Block':Block,
      'ReusedCount':ReusedCount,
      'Serial':Serial,
      'Longitude': Longitude,
      'Latitude': Latitude}

[33]: # Create a data from launch_dict
      df = pd.DataFrame(launch_dict)

[35]: # Hint data['BoosterVersion']!='Falcon 1'
      data_falcon9 = df[df['BoosterVersion'] != 'Falcon 1']
```

Data Collection - Scraping

1. Getting response from HTML
2. Creating BeautifulSoup Object
3. Finding tables
4. Getting Column names
5. Create dictionary
6. Append data to keys (refer to notebook cell [13]
7. Convert dictionary to Pandas dataframe
8. Dataframe to .CSV

Github Notebook URL:

https://github.com/gn-wy/IBM-data-science-capstone/blob/main/jupyter-labs-webscraping_1-2.ipynb



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

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

```
html_tables = soup.find_all('table')
```

```
column_names = []

# Apply find_all() function with `th` element o
th_elements = first_launch_table.find_all('th')
# Iterate each th element and apply the provide
# Append the Non-empty column name (`if name is
for th in th_elements:
        name = extract_column_from_header(th)
        if name is not None and len(name) > 0:
            column_names.append(name)
```

```
launch_dict= dict.fromkeys(column_names)

# Remove an irrelvant column
del launch_dict['Date and time ( )']

# Let's initial the launch_dict with eac
launch_dict['Flight No.'] = []
launch_dict['Launch site'] = []
launch_dict['Payload'] = []
launch_dict['Payload mass'] = []
launch_dict['Orbit'] = []
launch_dict['Customer'] = []
launch_dict['Launch outcome'] = []
# Added some new columns
launch_dict['Version Booster']=[]
launch_dict['Booster landing']=[]
launch_dict['Date']=[]
launch_dict['Time']=[]
```

```
extracted_row = 0
#Extract each table
for table_number,table in enumerate(
   # get table row
    for rows in table.find_all("tr")
```

```
df= pd.DataFrame({ key:pd.Series(value) for key, value in launch_dict.items() })
```

```
df.to_csv('spacex_web_scraped.csv', index=False)
```

Data Wrangling

Context

In the data set, there are several different cases where the booster did not land successfully. Sometimes a landing was attempted but failed due to an accident; for example, *True Ocean* means the mission outcome was successfully landed to a specific region of the ocean while *False Ocean* means the mission outcome was unsuccessfully landed to a specific region of the ocean. *True RTLS* means the mission outcome was successfully landed to a ground pad *False RTLS* means the mission outcome was unsuccessfully landed to a ground pad.*True ASDS* means the mission outcome was successfully landed on a drone ship *False ASDS* means the mission outcome was unsuccessfully landed on a drone ship.
We mainly converted those outcomes into Training Labels with **1** means the booster successfully landed **0** means it was unsuccessful.

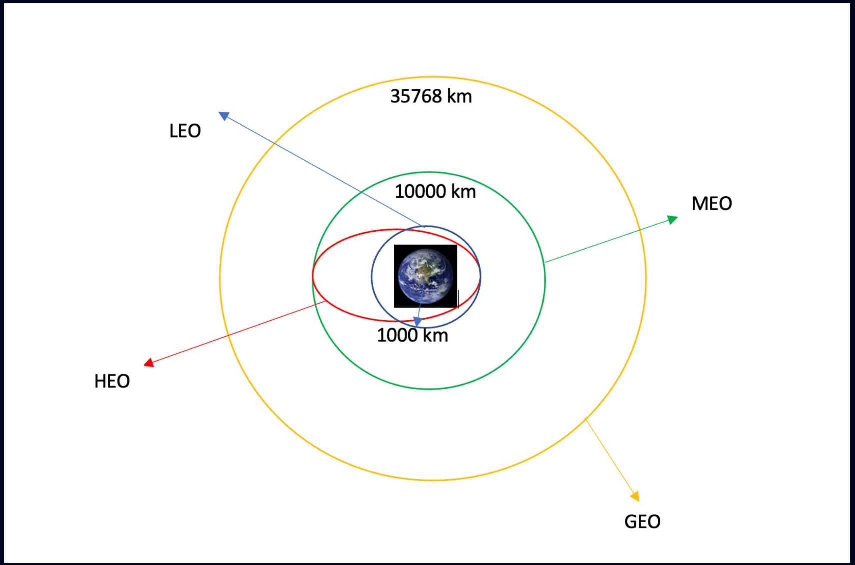
Process

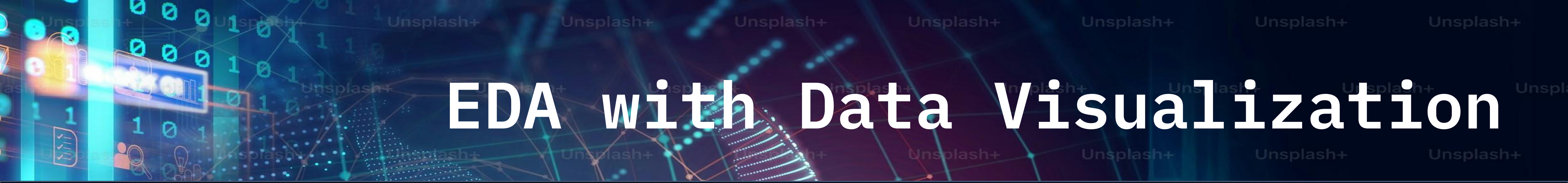
1. Perform Exploratory Data Analysis (EDA) on dataset
2. Calculate the number of launches per site
3. Calculate the number and occurrences of each orbit
4. Calculate the number and occurrences of mission outcomes per orbit type
5. Create a landing outcome label from *Outcome* column

Github Notebook URL:

https://github.com/gn-wy/IBM-data-science-capstone/blob/main/labs-jupyter-spacex-Data%20wrangling_2-1.ipynb

Each launch aims to a dedicated orbit, below are some common orbit types:





EDA with Data Visualization

Categorical Plots:
- Flight Number vs Payload Mass
- Flight Number vs Launch Site

Scatter Plots:
- Payload Mass vs Launch Site
- Flight Number vs Orbit Type
- Payload Mass vs Orbit Type

Scatter plots and Cat plots to show how one feature is affected by the other and their relationship is called their correlation. Scatter usually for 2 continuous variables, while cat plots are for categorical vs continuous data.

Bar Plots:
- Orbit Type vs Avg. Success Rate (Class)

Bar Plots compare different groups at a glance. Categories on 1 axis and discrete value in the other.

Line Plot:
- Date vs Class: Success rate by year

Line Graphs are useful in that they show data variables and trends clearly.

Github Notebook URL:

https://github.com/gn-wy/IBM-data-science-capstone/blob/main/edadataviz.ipynb

EDA with SQL

SQL Queries used to gather information about dataset:

• Displaying the names of the unique launch sites in the space mission
• Displaying 5 records where launch sites begin with the string 'KSC'
• Displaying the total payload mass carried by boosters launched by NASA (CRS)
• Displaying average payload mass carried by booster version F9 v1.1
• Listing the date where the successful landing outcome in drone ship was achieved.
• Listing the names of the boosters which have success in ground pad and have payload mass greater than 4000 but less than 6000
• Listing the total number of successful and failure mission outcomes
• Listing the names of the booster_versions which have carried the maximum payload mass.
• Listing the records which will display the month names, successful landing_outcomes in ground pad ,booster versions, launch_site for the months in year 2017
• Ranking the count of successful landing_outcomes between the date 2010-06-04 and 2017-03-20 in descending order.

Github Notebook URL:
https://github.com/gn-wy/IBM-data-science-capstone/blob/main/jupyter-labs-eda-sql-coursera_sqllite.ipynb

Build an Interactive Map with Folium

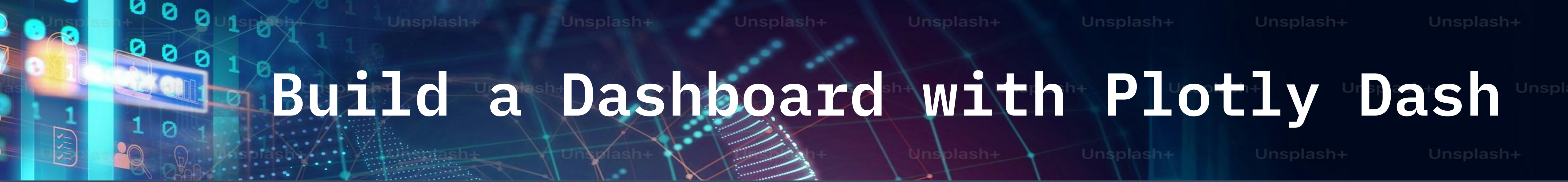
To visualize the launch data into an interactive map: we used the site's latitude and longitude for each launch site and added a *Circle Marker* and label of the name of each *launch site*

We assigned the dataframe *launch_outcomes(failures, successes)* to classes of *0* and *1* - with Green and Red markers on the map in a *Marker Cluster()*

Using Haversine's formula we calculated the distance from the *launch site* to various landmarks to find various trends about what is around the *launch sites* and measure patterns. Lines were drawn on the map to measure distance to landmarks.

Github Notebook URL:

https://github.com/gn-wy/IBM-data-science-capstone/blob/main/lab-jupyter-launch-site-location-v2_3-1.ipynb



Build a Dashboard with Plotly Dash

Built a Plotly Dash application for users to perform interactive visual analytics on SpaceX launch data in real-time. This dashboard application contains input components such as a dropdown list and a range slider to interact with a pie chart and a scatter point chart.

Pie Chart: showing the total launches by a certain site/all sites
- To display relative proportions of multiple classes of data
- Can select specific sites to see success rates

Scatter graph: showing the relationship between *Outcome* and *Payload Mass (Kg)* features for different *Booster Versions*
- Shows relationship between 2 variables
- The range of data flow, i.e. max and min values, can be determined
- Observations and interpretations are easy to understand

Github Source code URL:
https://github.com/gn-wy/IBM-data-science-capstone/blob/main/spacex_dash_app.py

Predictive Analysis (Classification)

Building the Model:
- Load datasets from previous labs, dataset_Part_2 and dataset_Part_3
- Create NumPy array from column “Class” (success/fail) in dataset_Part_2 and assign a variable, Y.
- Standardize dataset_Part_3 using Transform and assign it to variable, X.
- Split data into training and test datasets, using parameter_test_size to .2
- Decide which machine learning algorithms to use
- Set parameters and algorithms to GridSearchCV
- Fit the datasets into the GridsearchCV objects and train the datasets

Evaluating Models:
- Check accuracy for each model, using method **score**
- Get tuned hyperparameters for each algorithm
- Plot confusion matrix for each model

Improving Models:
- Feature Engineering
- Algorithm Tuning

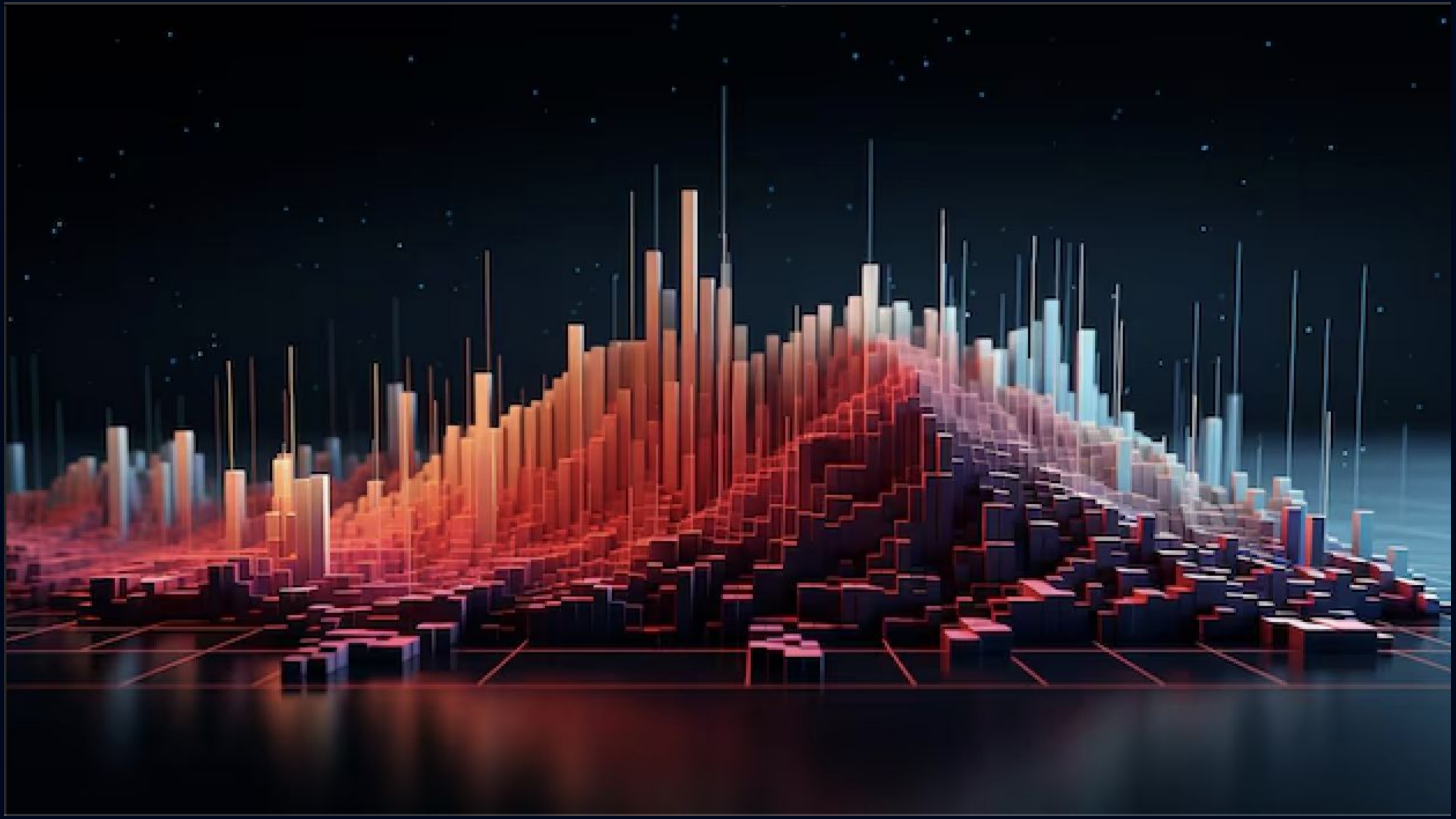
Best Classification Model:
- Model with best accuracy score wins

Github Notebook URL:
https://github.com/gn-wy/IBM-data-science-capstone/blob/main/SpaceX-Machine-Learning-Prediction-Part-5-v1.ipynb

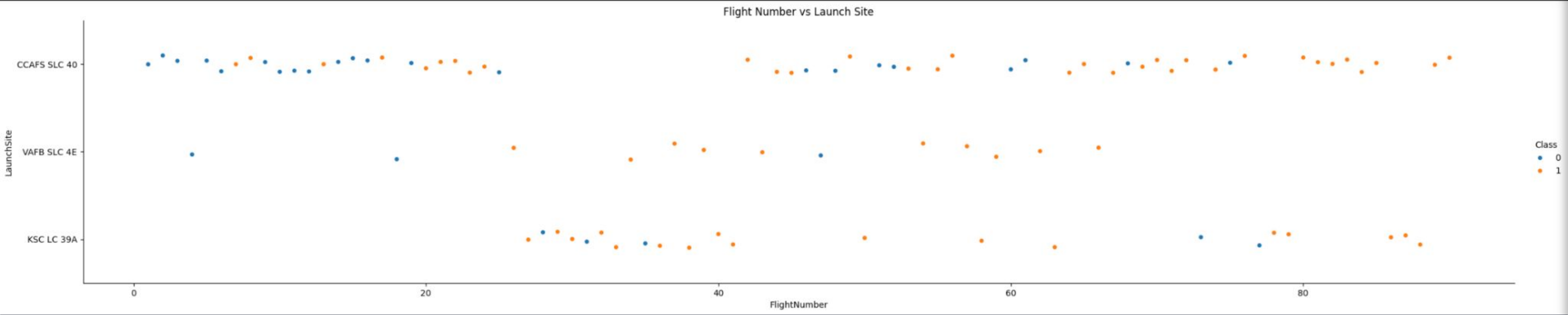
Results

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

Exploratory Data Analysis (EDA) with Visualization



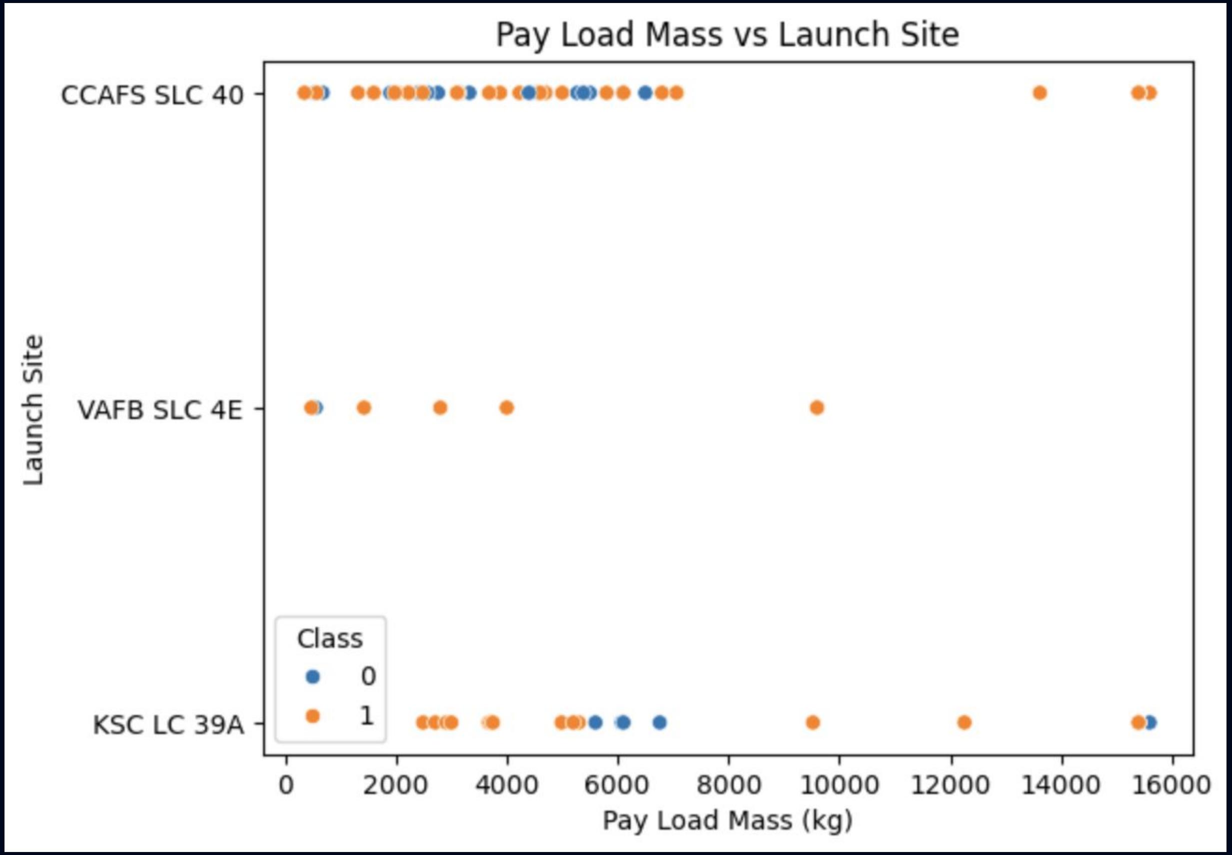
Flight Number vs. Launch Site



The higher the flight number for any launch site, the greater the success rate

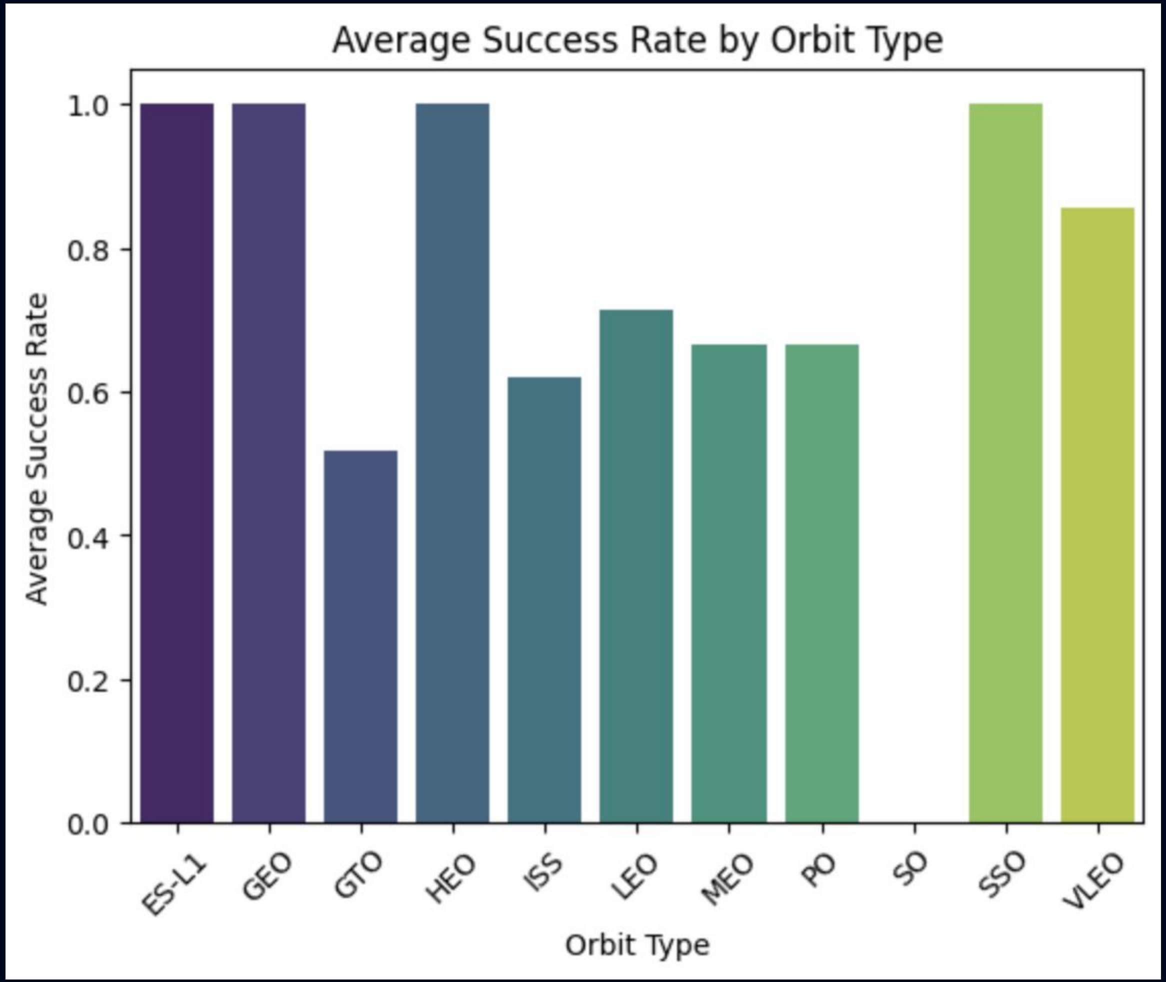
<u>**NOTE**</u>**: Class 1(blue)=Success, 0(orange)=Failure**

Payload vs. Launch Site



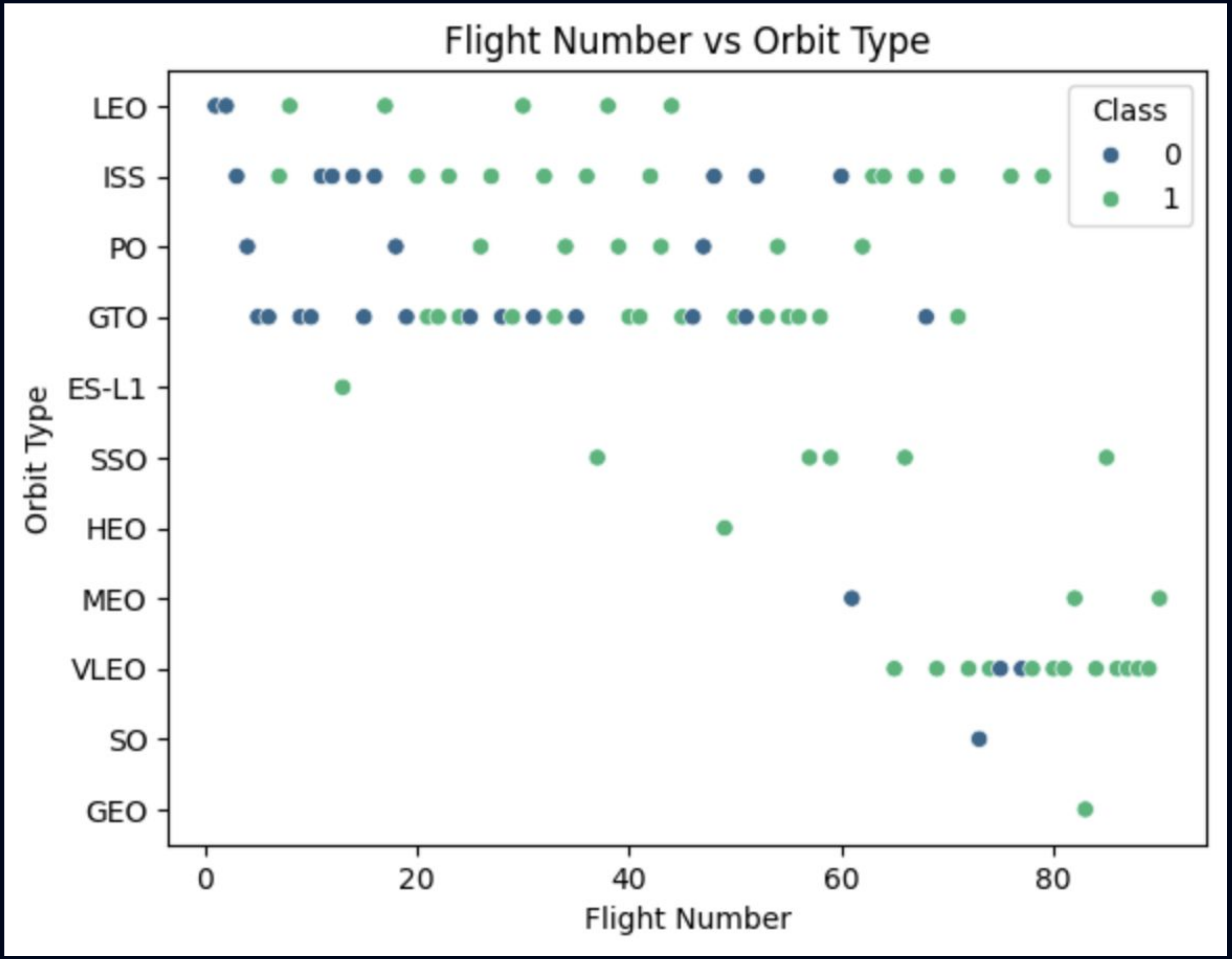
- Scatter point chart shows that for VAFB-SLC launchsite there are no rockets launched for heavy payload mass(greater than 10000)
- The higher the Payload mass for CCAFS SLC 40 the greater the success rate

Success Rate vs. Orbit Type



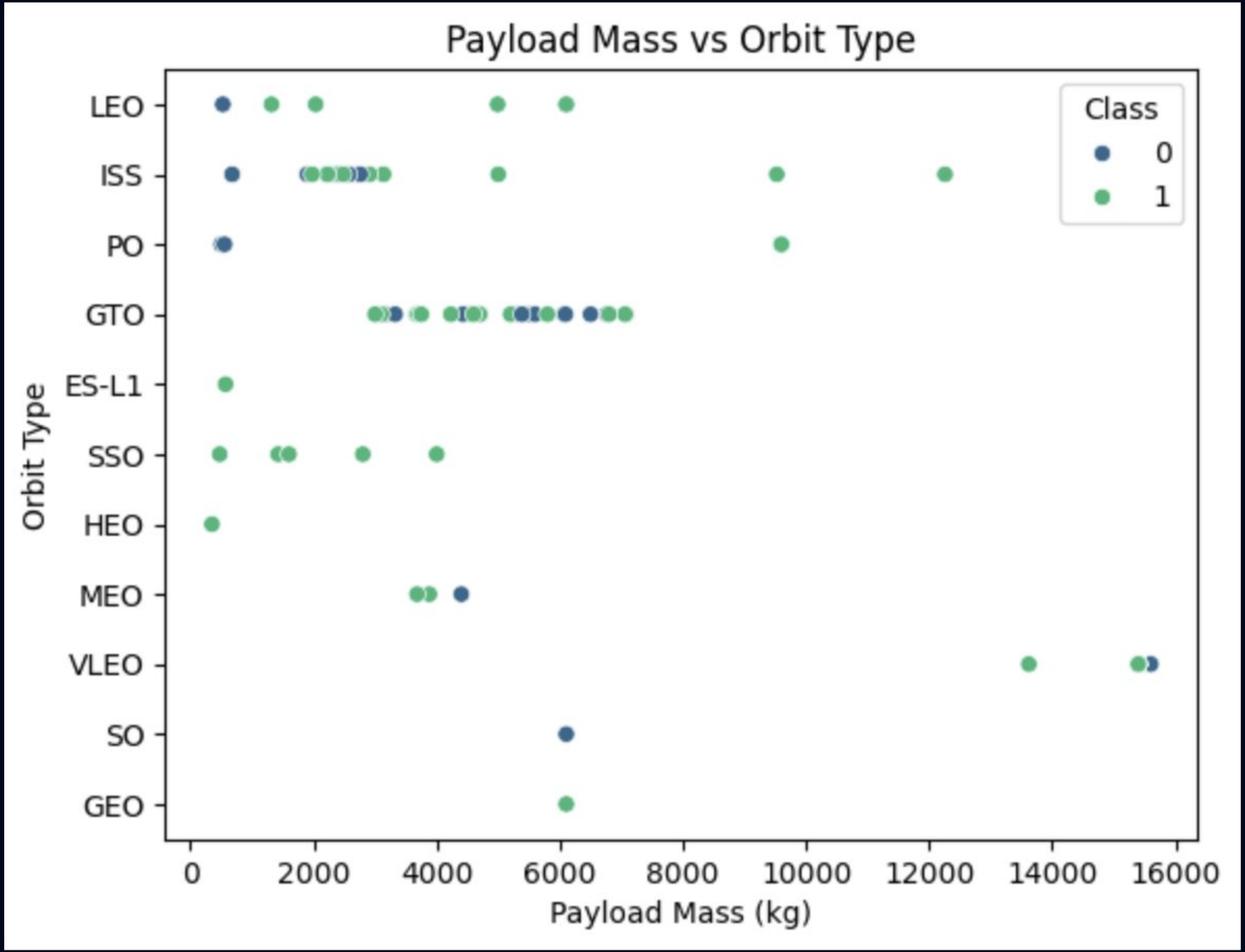
- Orbit types ES-L1, GEO, HEO, SSO had 100% average success rate
- Orbit type VLEO had an 80% success rate
- All others fell slightly below with GTO being the lowest

Flight Number vs. Orbit Type



- The LEO orbit, success seems to be related to the number of flights
- Conversely, in the GTO orbit, there appears to be no relationship between flight number and success rate

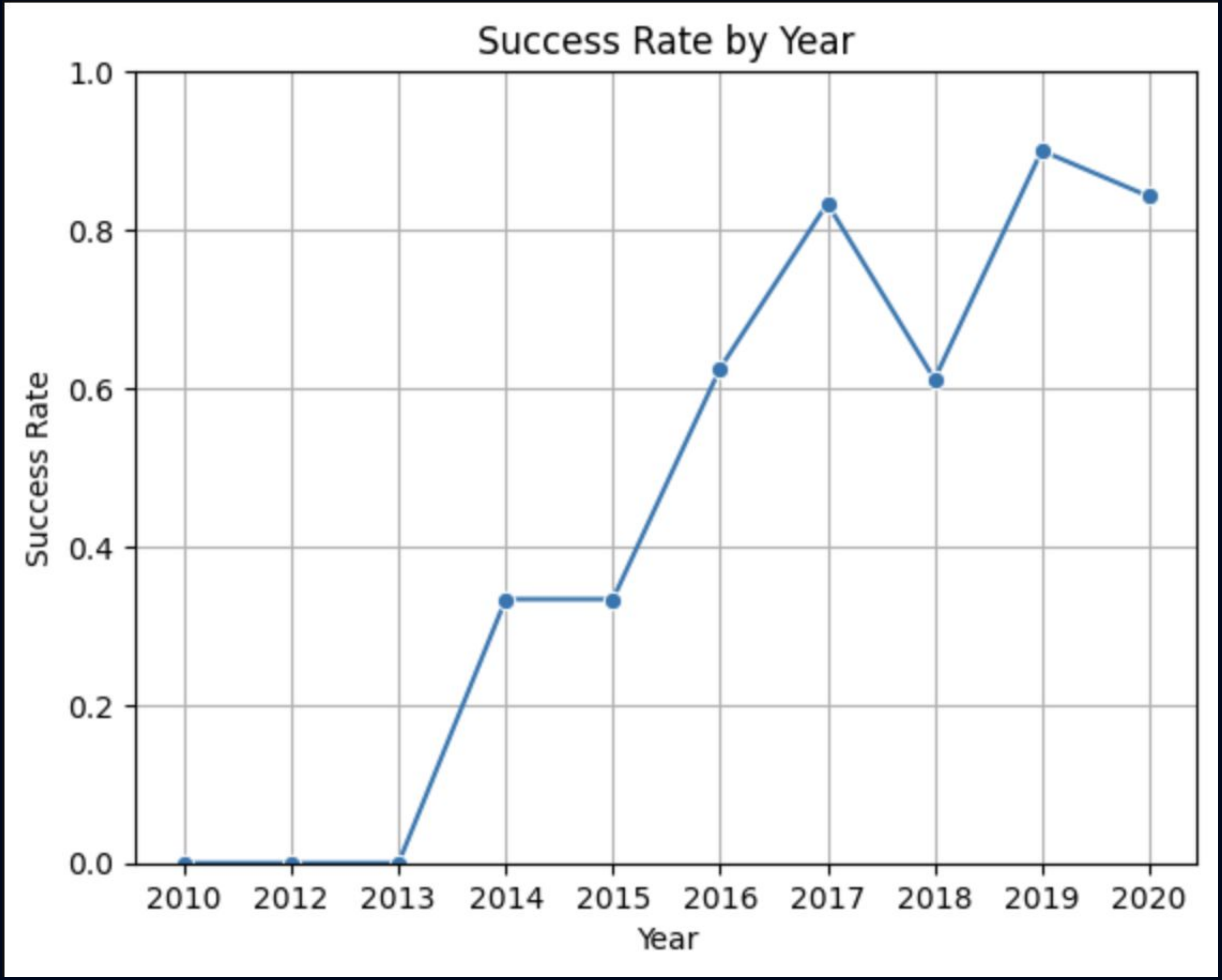
Payload vs. Orbit Type



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

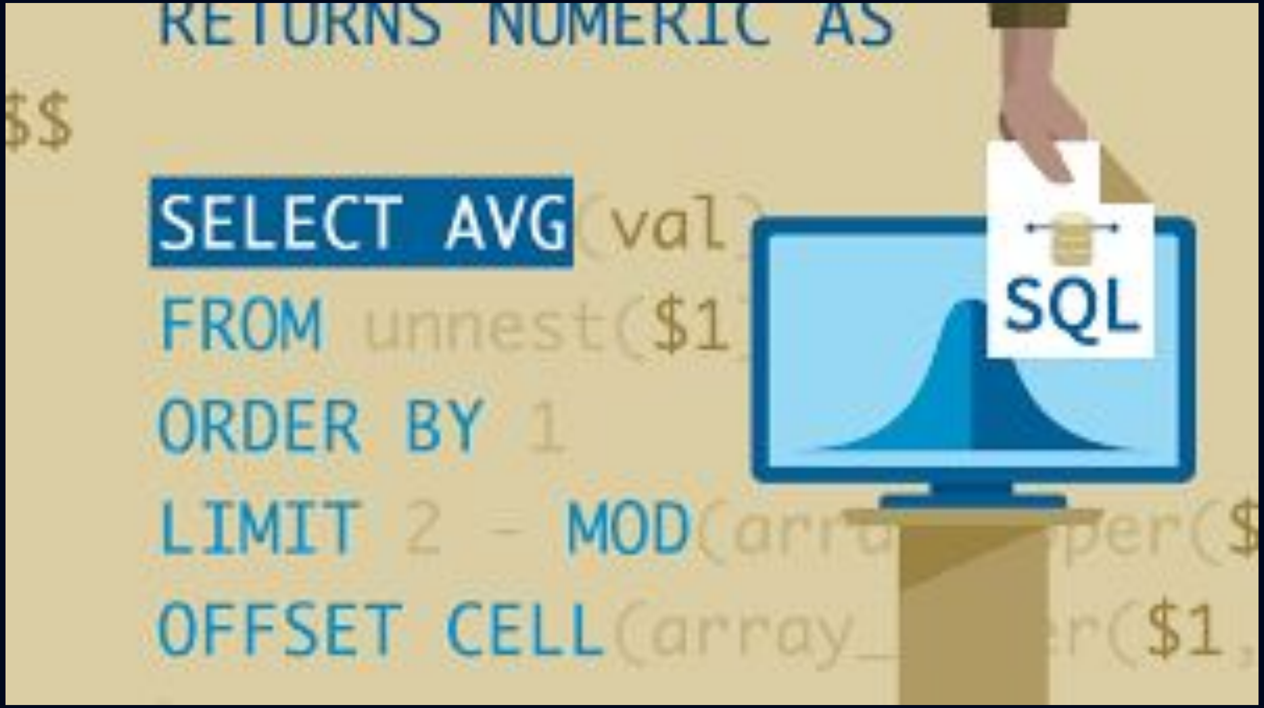
- However, for GTO, it's difficult to distinguish between successful and unsuccessful landings as both outcomes are present.

Launch Success Yearly Trend



- The success rate increased steadily from 2013 through 2020

Exploratory Data Analysis (EDA) with SQL



```
RETURNS NUMERIC AS
$$
SELECT AVG(val)
FROM unnest($1)
ORDER BY 1
LIMIT 2 - MOD(arr      per($
OFFSET CELL(array_     r($1,
```

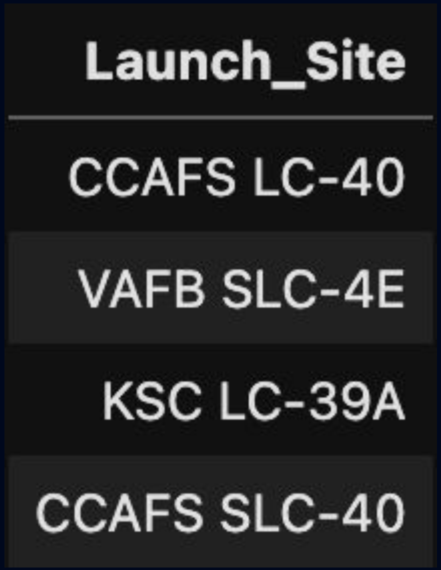
All Launch Site Names

SQL QUERY

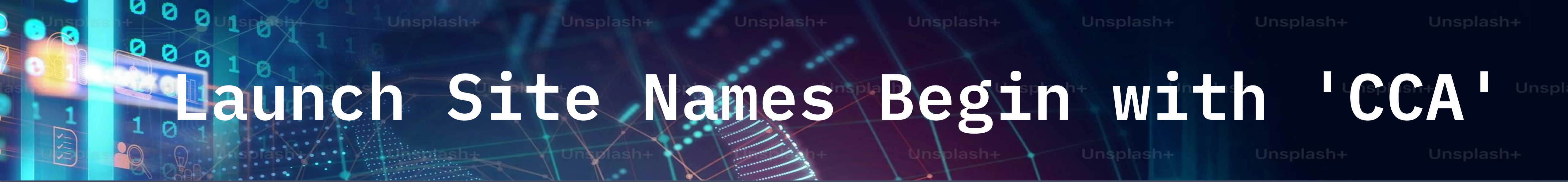
```
SELECT DISTINCT "Launch_Site" FROM
SPACEXTABLE;
```

Explanation:

DISTINCT in the query is important to only show unique values in the Launch Site column of SPACEXTABLE



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



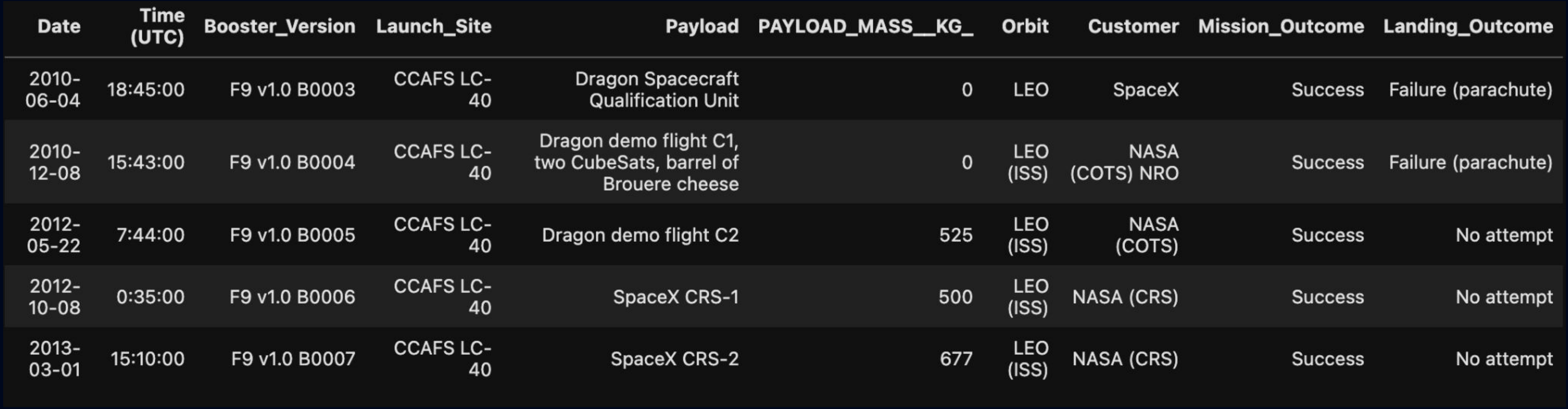
Launch Site Names Begin with 'CCA'

SQL QUERY

```
SELECT * FROM SPACEXTABLE

WHERE "Launch_Site" LIKE 'CCA%' LIMIT 5;
```

Explanation:

The LIKE keyword is used for pattern matching, allowing flexible filtering of results. The LIMIT 5 clause restricts the output to the first 5 matching records to reduce the dataset size for quick review or analysis.

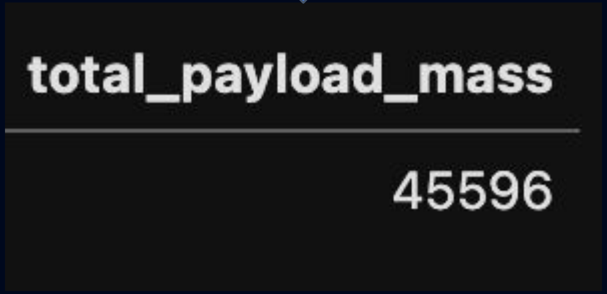


Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight 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

Total Payload Mass

SQL QUERY

```
SELECT SUM("PAYLOAD_MASS__KG_") AS
total_payload_mass FROM SPACEXTABLE

WHERE "Customer" = 'NASA (CRS)';
```



```
total_payload_mass
             45596
```

Explanation:

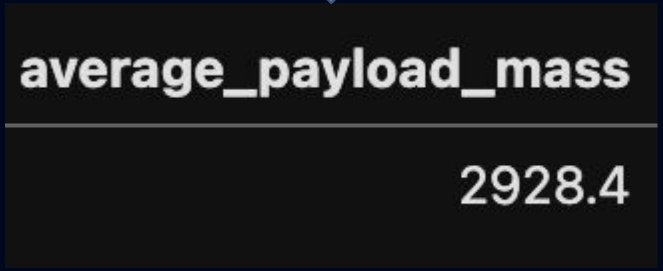
Using the function **SUM** summates the total in column **PAYLOAD_MASS_KG_**

The **WHERE** clause filters the dataset to only perform calculations on column **CUSTOMER** being **NASA (CRS)**

Average Payload Mass by F9 v1.1

SQL QUERY

```
SELECT AVG("PAYLOAD_MASS__KG_") AS
average_payload_mass FROM SPACEXTABLE

WHERE "Booster_Version" = 'F9 v1.1';
```



average_payload_mass
2928.4

Explanation:

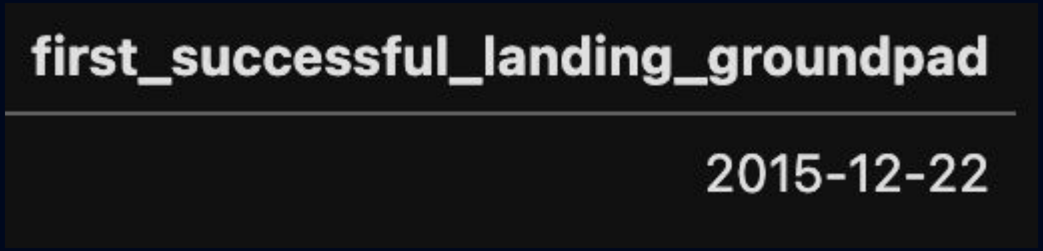
Using the function **AVG** gets the average in column **PAYLOAD_MASS_KG_**

The **WHERE** clause filters the dataset to only perform calculations on **Booster Version F9 v1.1**

First Successful Ground Landing Date

SQL QUERY

```
SELECT MIN("Date") AS
first_successful_landing_groundpad FROM SPACEXTABLE

WHERE "Landing_Outcome" LIKE '%ground pad%';
```



first_successful_landing_groundpad
2015-12-22

Explanation:

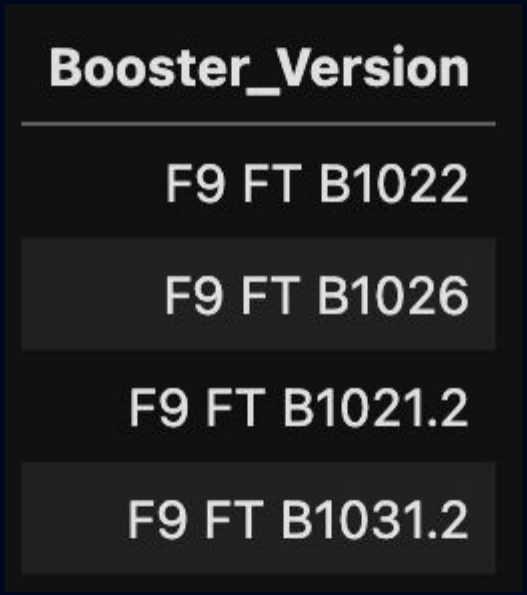
Using the function **MIN** gets the minimum date in column **DATE**

The **WHERE** clause filters the dataset to only perform calculations on **Landing_Outcome ground pad**

Successful Drone Ship Landing with Payload between 4000 and 6000

SQL QUERY

```
SELECT DISTINCT "Booster_Version" FROM
SPACEXTABLE

WHERE "Landing_Outcome" LIKE '%Success (drone
ship)%' AND "PAYLOAD_MASS__KG_" > 4000 AND
"PAYLOAD_MASS__KG_" < 6000;
```



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

Explanation:

Using the function **DISTINCT** gets only unique **Booster_Version**

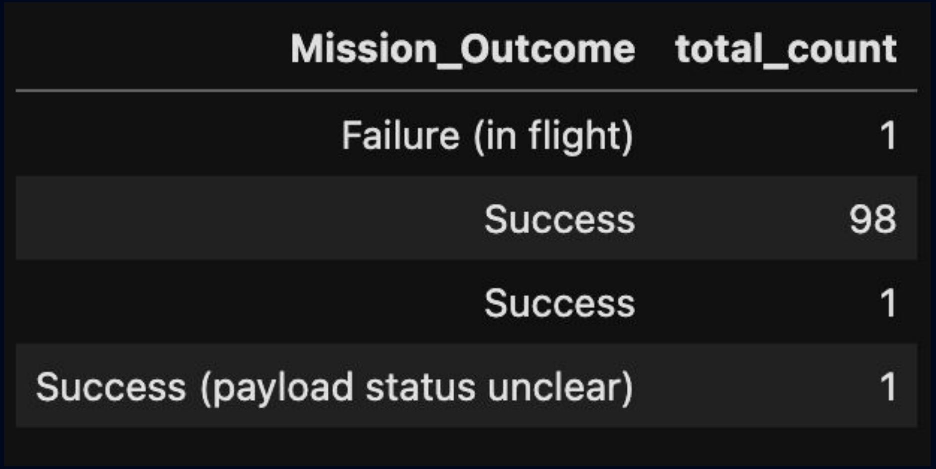
The **WHERE** clause filters the dataset to **Landing_Outcome = Success**

The **AND** clause specifies additional filter conditions such that Payload_MASS be between 4000 and 6000

Total Number of Successful and Failure Mission Outcomes

SQL QUERY

```
SELECT "Mission_outcome", COUNT(*)
AS total_count FROM SPACEXTABLE

GROUP BY "mission_outcome";
```



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

Explanation:

The **SELECT** statement retrieves the column **Mission_outcome** and counts the occurrences of each unique value, naming this count **total_count**.

The **GROUP BY** clause ensures that the results are aggregated by each distinct **Mission_outcome**, providing a breakdown of how many missions were successful or failed.

Boosters Carried Maximum Payload

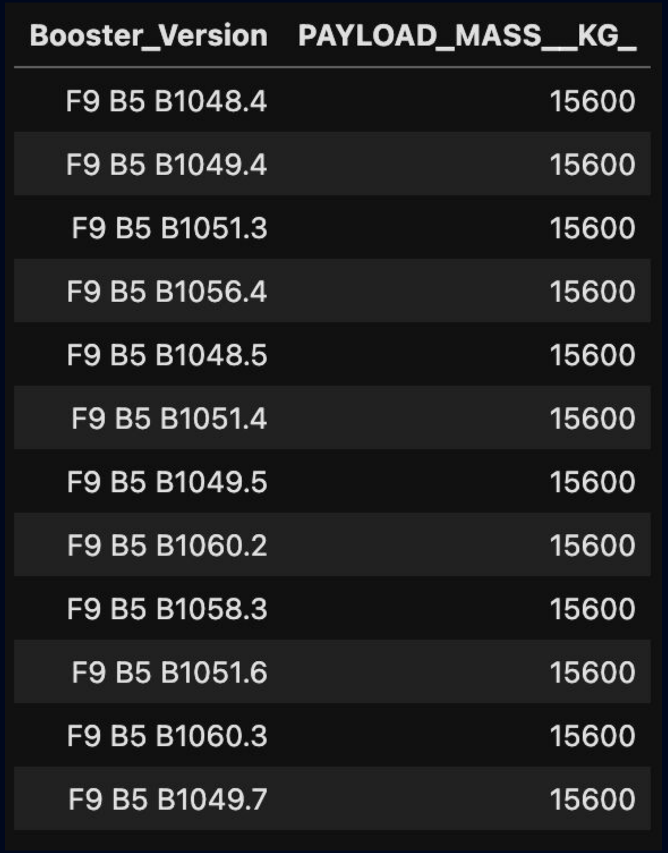
SQL QUERY

```
SELECT "Booster_Version",
"Payload_Mass__kg_" FROM SPACEXTABLE

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

Explanation:

Booster_Version and **Payload_Mass__kg_** from the SPACEXTABLE table where the PAYLOAD_MASS__KG_ matches the maximum payload mass found in the table.

The WHERE clause uses a subquery (SELECT MAX("PAYLOAD_MASS__KG_") FROM SPACEXTABLE) to identify the record with the highest payload mass, ensuring that only the row(s) with this maximum value are included in the result.

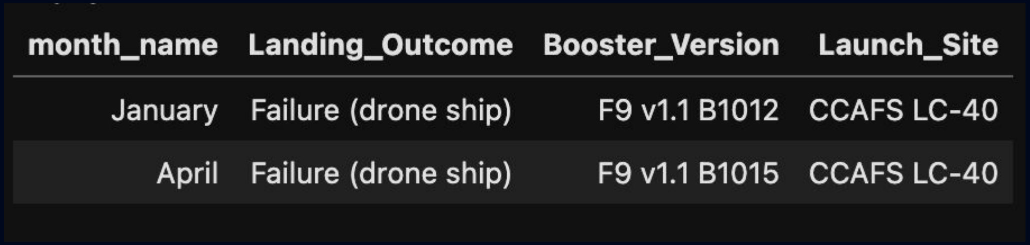


Booster_Version	PAYLOAD_MASS__KG_
F9 B5 B1048.4	15600
F9 B5 B1049.4	15600
F9 B5 B1051.3	15600
F9 B5 B1056.4	15600
F9 B5 B1048.5	15600
F9 B5 B1051.4	15600
F9 B5 B1049.5	15600
F9 B5 B1060.2	15600
F9 B5 B1058.3	15600
F9 B5 B1051.6	15600
F9 B5 B1060.3	15600
F9 B5 B1049.7	15600

2015 Launch Records

SQL QUERY

```
SELECT CASE substr("Date", 6, 2)
      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 substr("Date", 1, 4) = '2015';
```



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

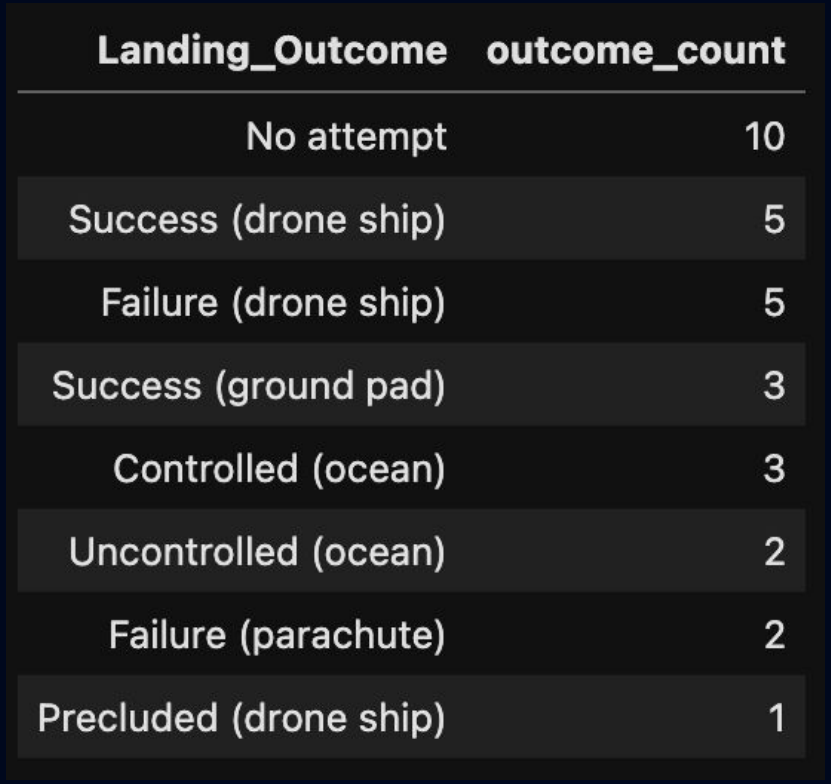
Explanation:

The **substr("Date", 6, 2)** function extracts the month part of the **Date** as a two-character string, which is then converted to its corresponding month name using **TO_CHAR(TO_DATE(..., 'MM'), 'Month')**
The **Booster_Version** and **Launch_Site** columns in the output to provide detailed information about each record matching these criteria

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

SQL QUERY

```
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;
```

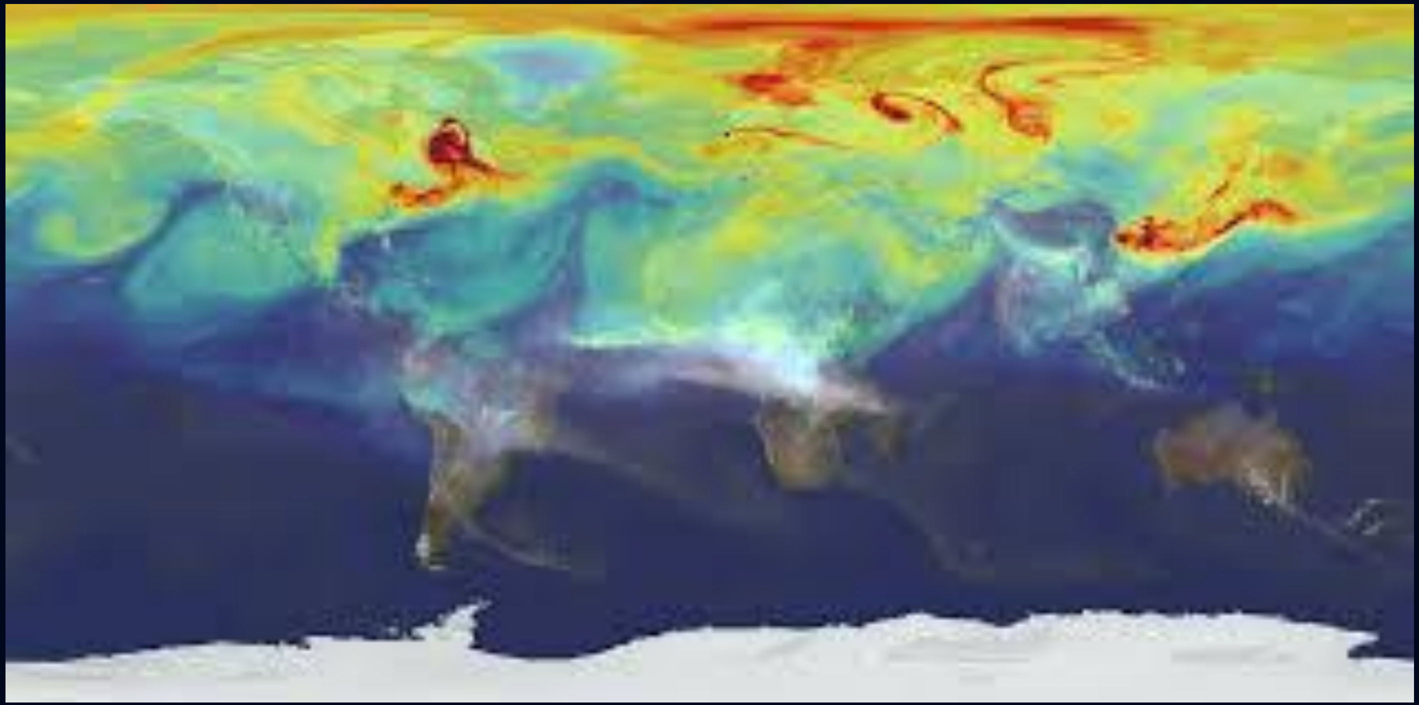


Landing_Outcome	outcome_count
No attempt	10
Success (drone ship)	5
Failure (drone ship)	5
Success (ground pad)	3
Controlled (ocean)	3
Uncontrolled (ocean)	2
Failure (parachute)	2
Precluded (drone ship)	1

Explanation:

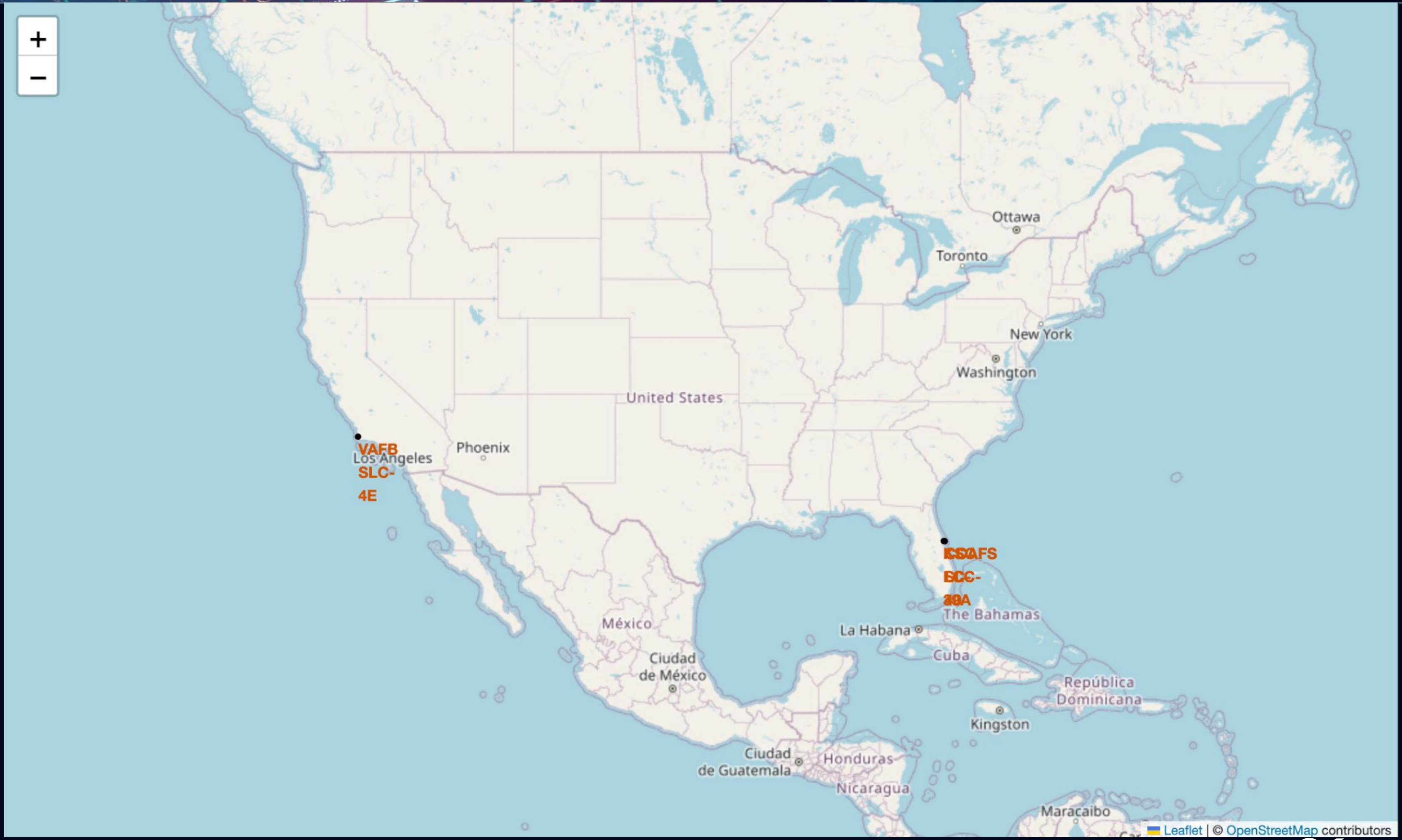
The **COUNT(*)** function tallies the number of rows for each **Landing_Outcome**, and the **GROUP BY** clause ensures that the count is grouped by each unique **Landing_Outcome**.
The **ORDER BY** outcome_count **DESC** arranges the results in descending order based on the count, showing the most common outcomes first.

Interactive Map with Folium

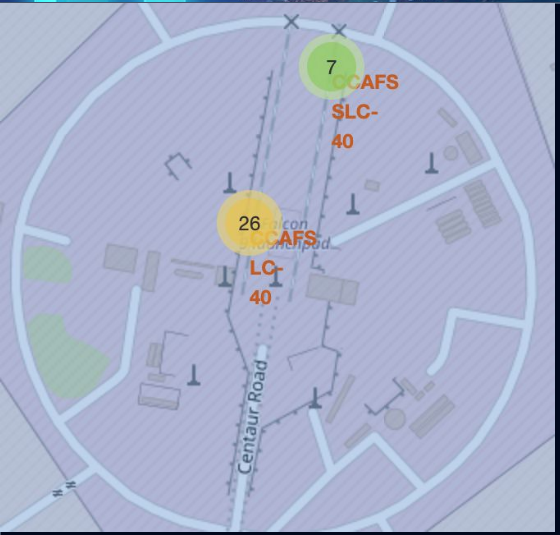


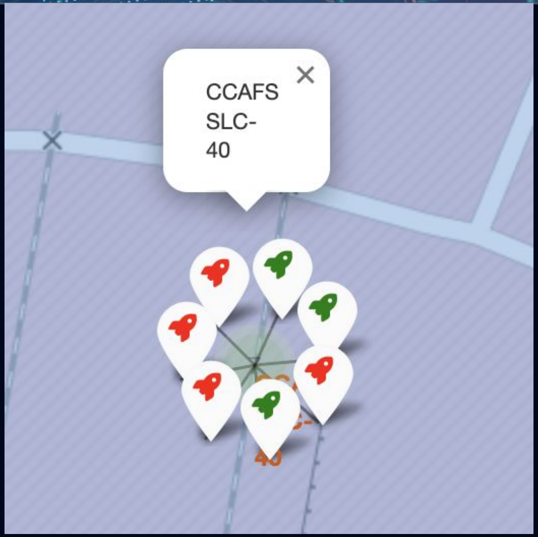
All Launch Sites Marked Map

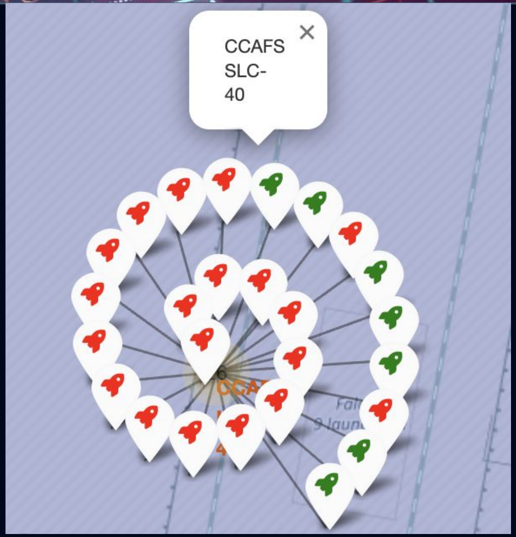
All SpaceX launch sites (labeled in red) are in the United States coasts - California and Florida

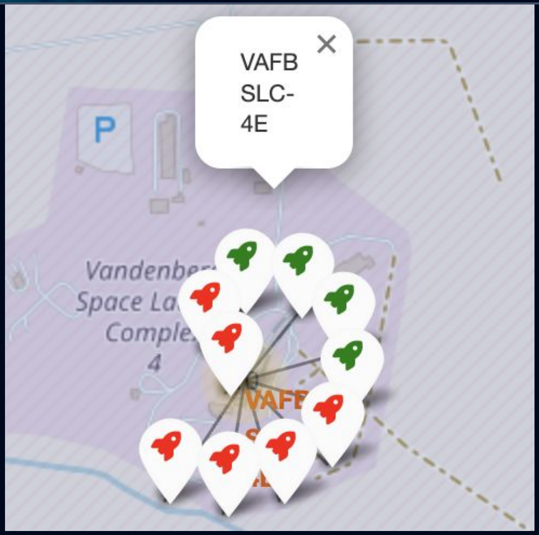


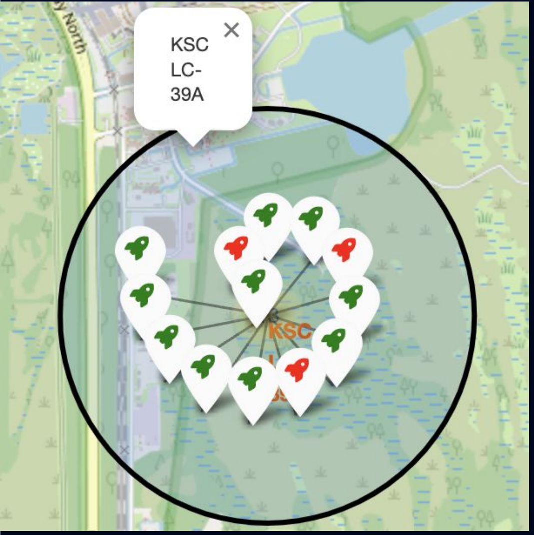
Launch Sites - Color Markers











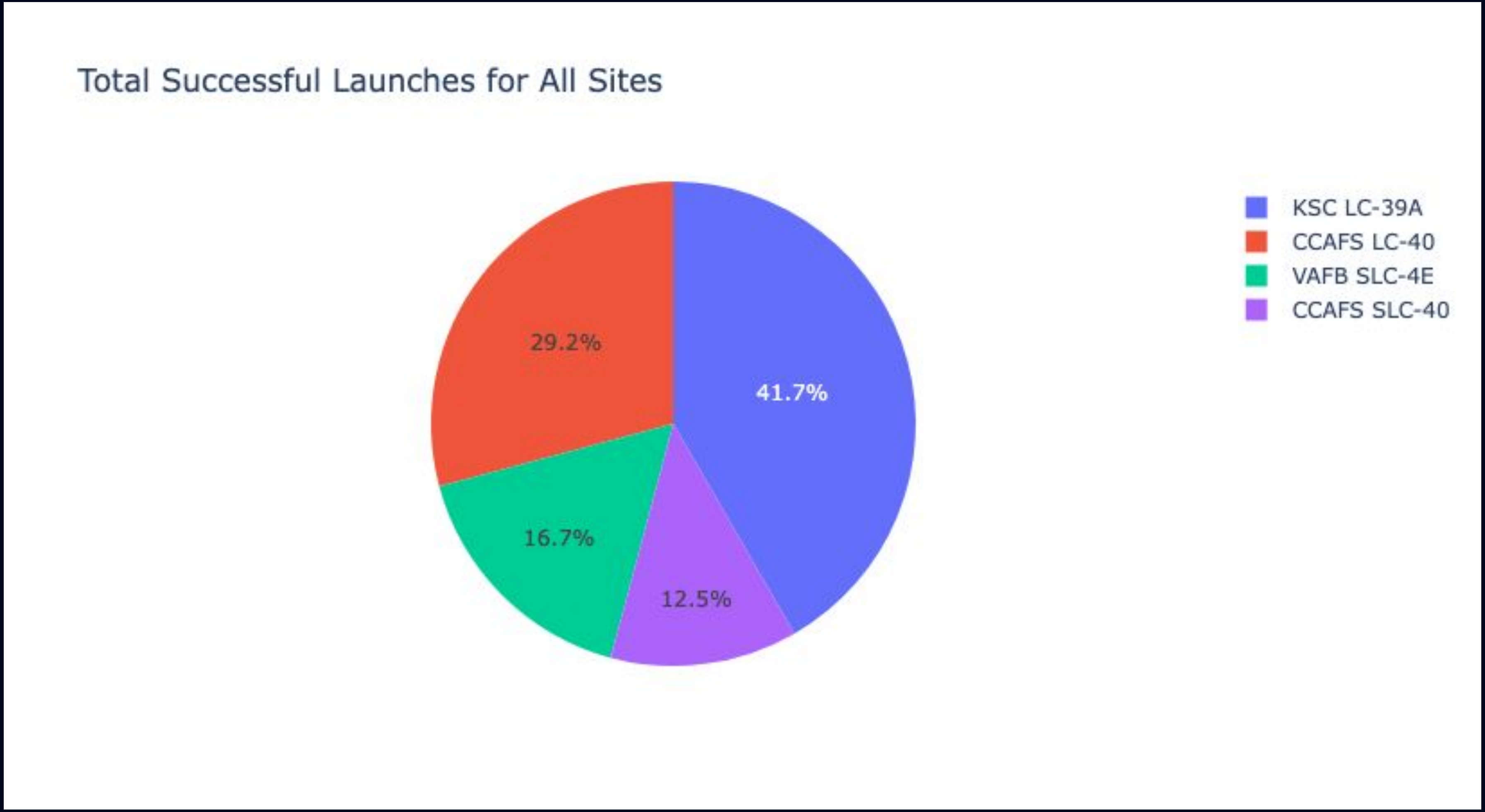
- Florida Launch Sites
- California Launch Site

Green Markers represent *successful* launches
and Red Markers represent *failed* launches

Dashboard with Plotly Dash

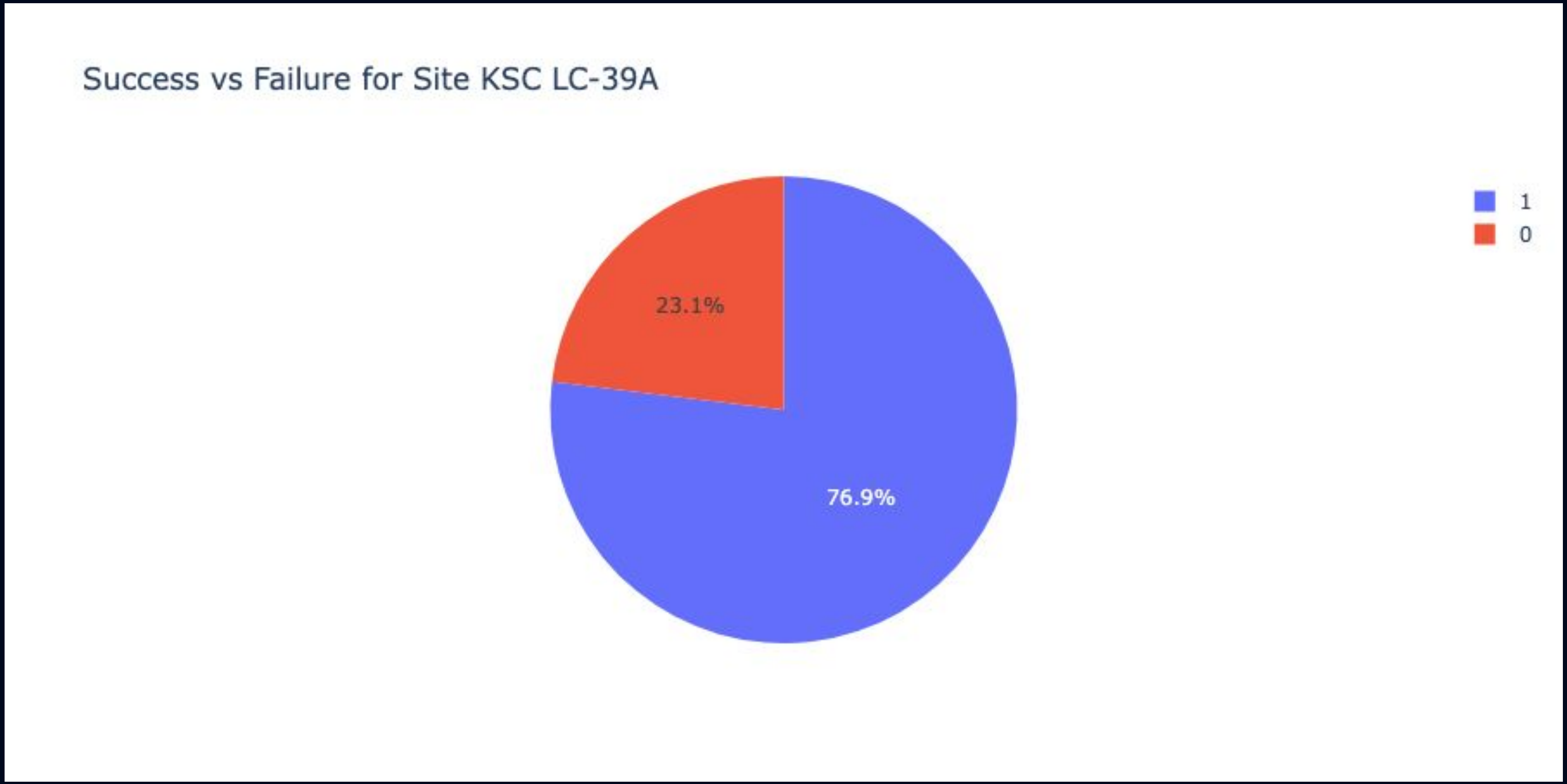
Pie Chart: Success Rates for all Launch Sites

Chart shows **KSC LC-39A** had the most successful launches from all sites



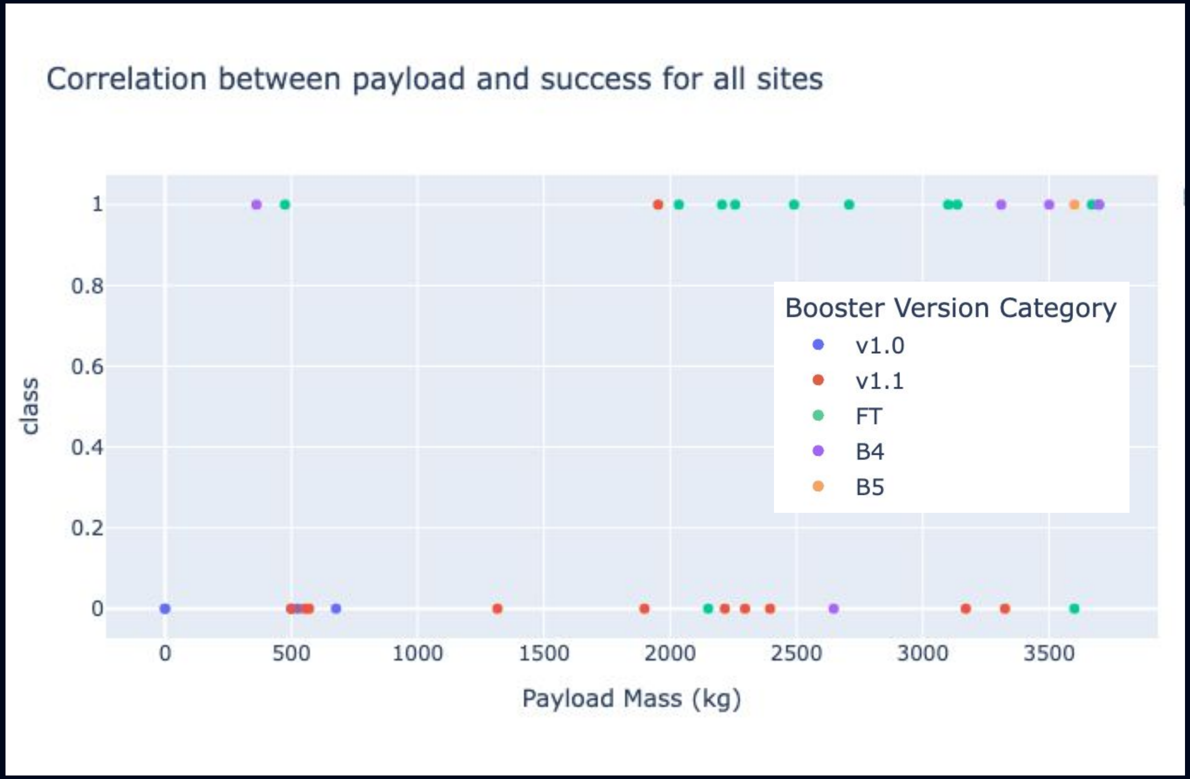
Pie Chart: Highest Launch Success Rate Site

Pie Chart shows **KSC LC 39A** had a **76.9%** success rate and **23.1%** failure rate

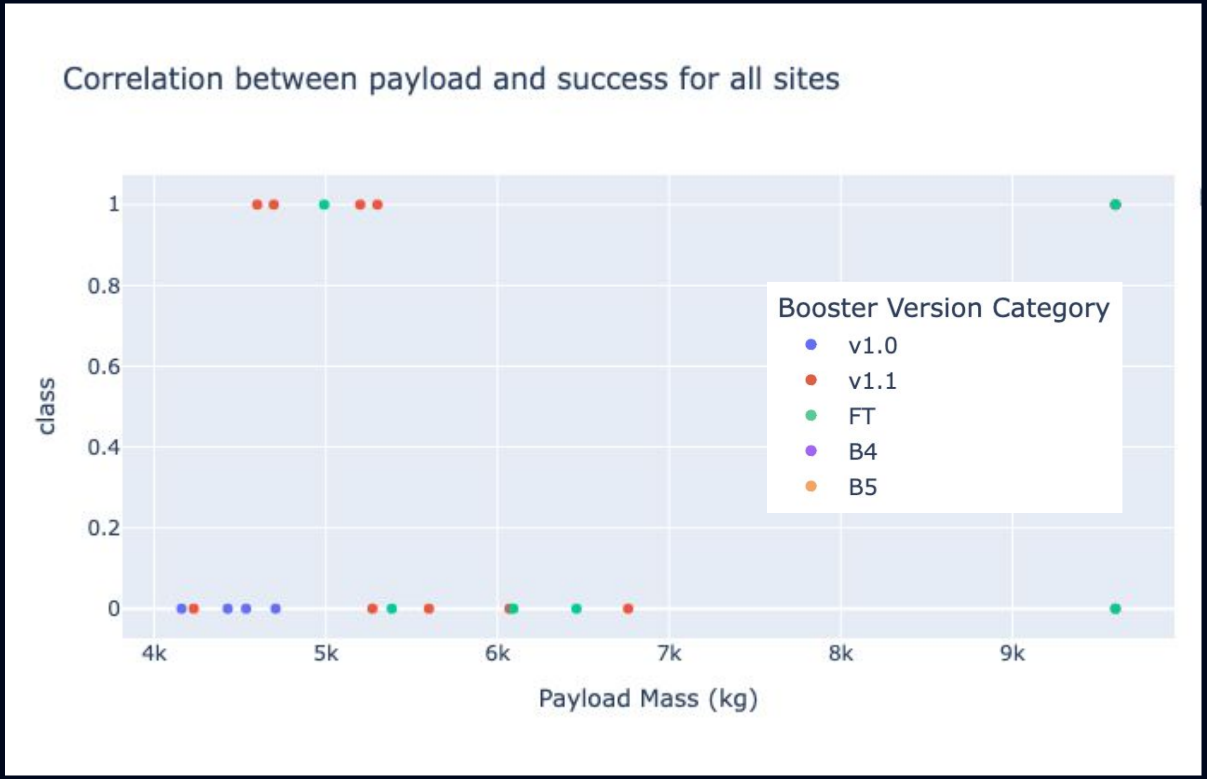


Scatter Plot: Payload vs Outcome for all sites-range slider comparison

Payload: 0kg-4000kg

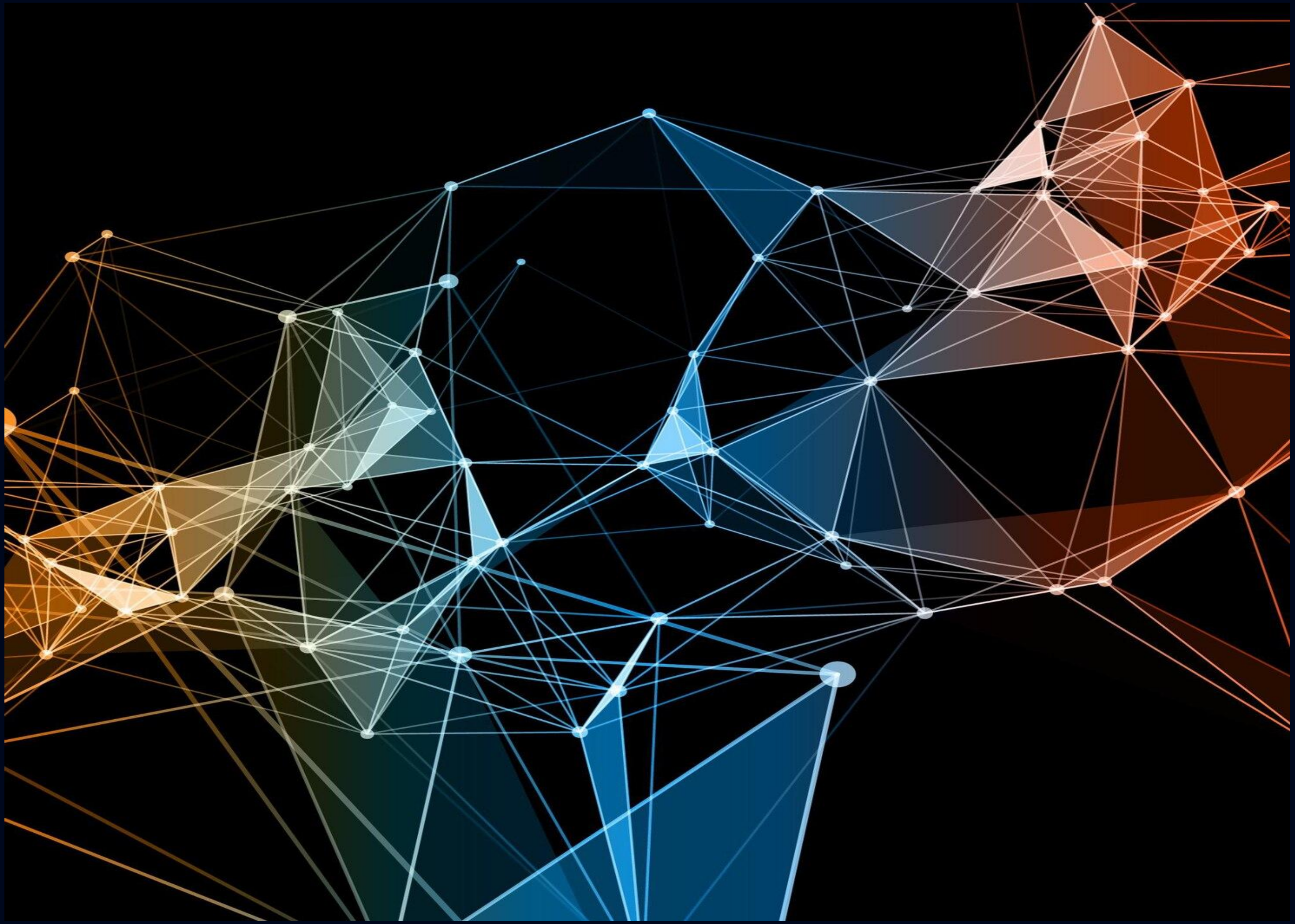


Payload: 4000kg-10000kg



Scatter plot shows higher success rates for low weight payloads than high weight payloads

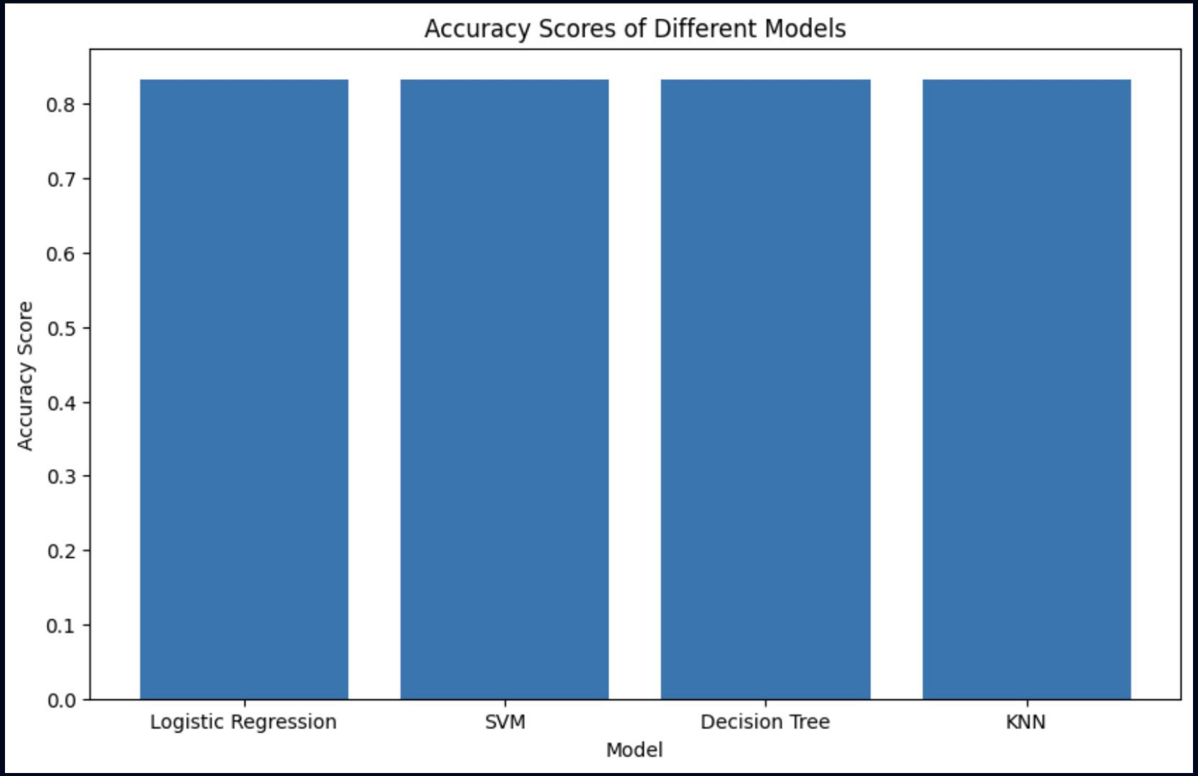
Predictive Analysis-Classification



Classification Accuracy

Calculating the accuracy of every classifications model using the Score method showed equal scores as shown below:

```
Logistic Regression Accuracy: 0.83
SVM Accuracy: 0.83
Decision Tree Accuracy: 0.83
K-Nearest Neighbors Accuracy: 0.83
Best Model: Logistic Regression with Accuracy: 0.83
```



However after selecting the optimal hyperparameters for each we see that the ***Decision Tree*** achieved the highest accuracy on the test data with **0.875**

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 8, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples
_split': 10, 'splitter': 'random'}
accuracy : 0.875
```

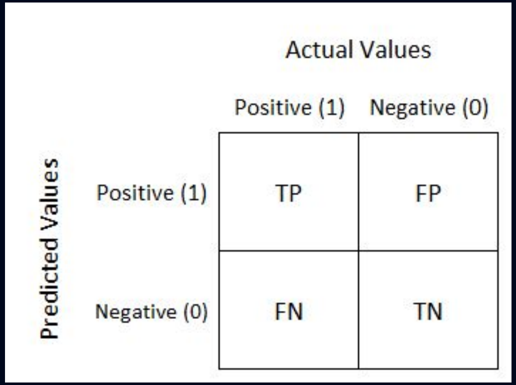
Confusion Matrix-Decision Tree

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

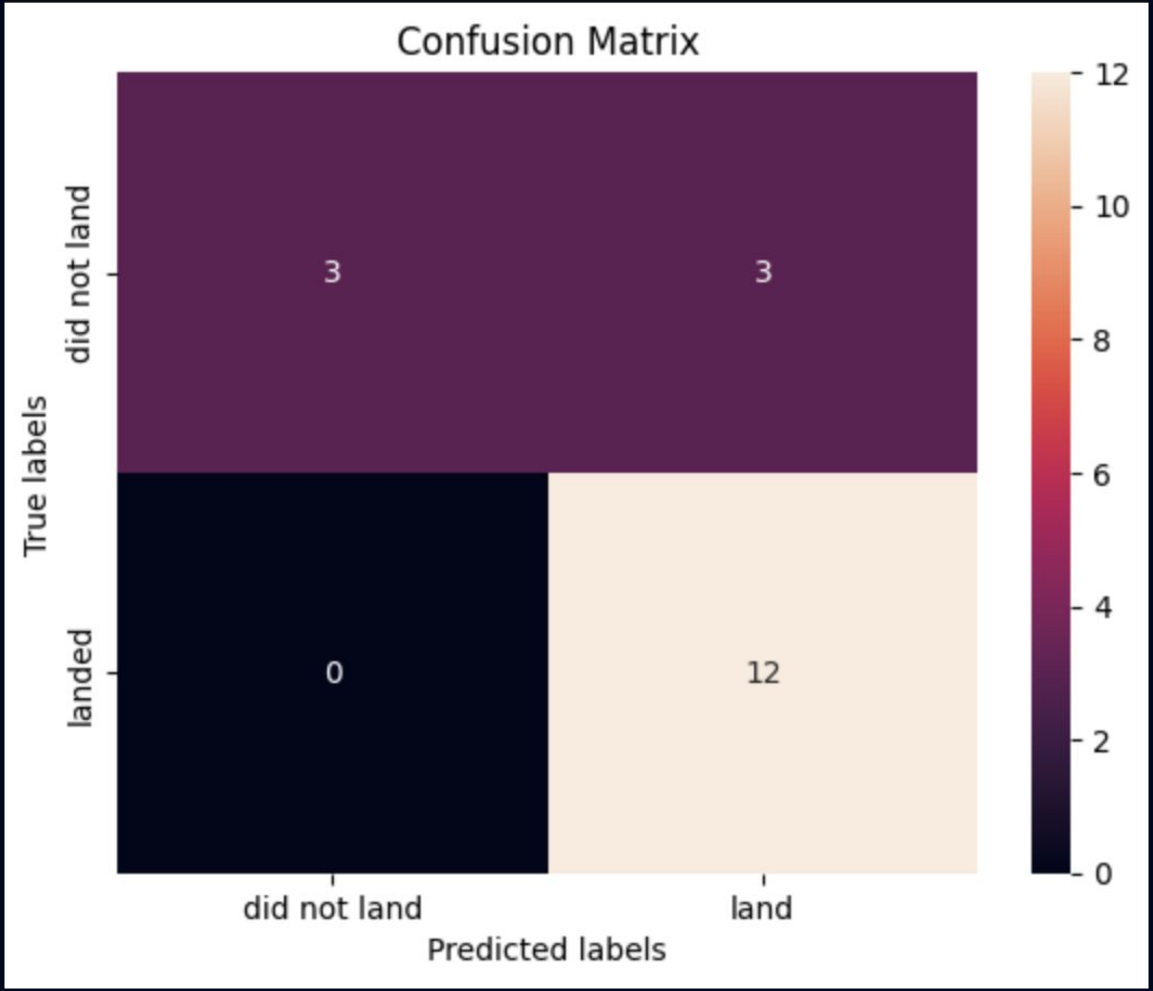
Overview:

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

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



		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN



Conclusions

- The Decision Tree Classifier Algorithm is the best for Machine Learning for this dataset
- Low weight payloads perform better than the heavier payloads
- The success rates for SpaceX launches is directly proportional to time in years signifying they will eventually perfect the launches
- We can see that KSC LC-39A had the most successful launches from all the sites
- Orbits GEO,HEO,SSO,ES-L1 have the best Success Rate