IBM Developer
SKILLS NETWORK

# Winning Space Race with Data Science

Elvin A. Morales
10/22/2021

# Outline

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

# Executive Summary

- Summary of methodologies
  - Data collection
  - Data wrangling
  - EDA with data visualization
  - EDA with SQL
  - Building an interactive map with Folium
  - Building a Dashboard with Plotly Dash
  - Predictive Analysis (Using Classification)
- Summary of all results
  - EDA results
  - Dashboard interaction demo (Using screenshots)
  - Predictive analysis results

# Introduction

- Project background and context
  - Space X advertises that their Falcon 9 rocket launches with a cost of 62 million dollars. Other providers cost upward of 165 million dollars for each launch. The reason for the savings is because Space X can reuse the first stage of the rocket. If we can determine if the first stage will land, we can determine the cost of a future launch and use this information to bid against Space X for a rocket launch.
- Problems you want to find answers
  - What are the features that influence if the first stage will land successfully?
    - What are the effects of each feature in determining the success rate of a successful landing?
    - What are the optimal conditions that Space X needs to maximize the success rate and achieve a successful landing?

Section 1

# Methodology

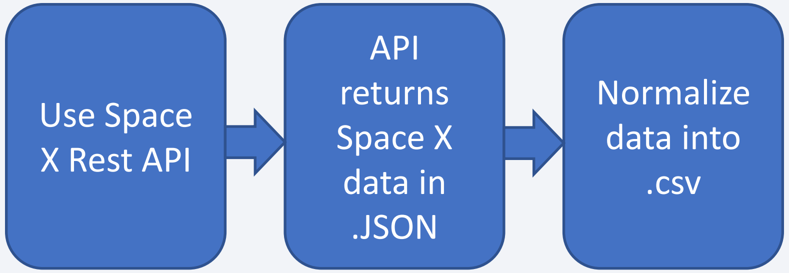
# Methodology

- Data collection methodology:
  - Space X Rest API
  - Web scrapping from Wikipedia
- Performed data wrangling
  - One Hot Enconding data fields for Machine Learning
  - Dropping irrelevant columns.
- Performed exploratory data analysis (EDA) using visualization and SQL
  - Scatter plots and bar graphs were used to show relationships between the features to show patterns of data.
- Performed interactive visual analytics using Folium and Plotly Dash
- Performed predictive analysis using classification models
  - How to build, tune, evaluate classification models

# Data Collection

- The launch data was collected in the following ways:
  - Space X REST API.
    - The API contained information about the rocket used, payload delivery, launch specifications, landing specifications and landing outcome.
    - The endpoints for the API start with api.spacexdata.com/v4/.
  - Web Scraping Wikipedia
    - Using beautiful soup, all Falcon 9 historical launch records were scraped from the Wikipedia page titled [“List of Falcon 9 and Falcon Heavy launches”]

Space X REST API

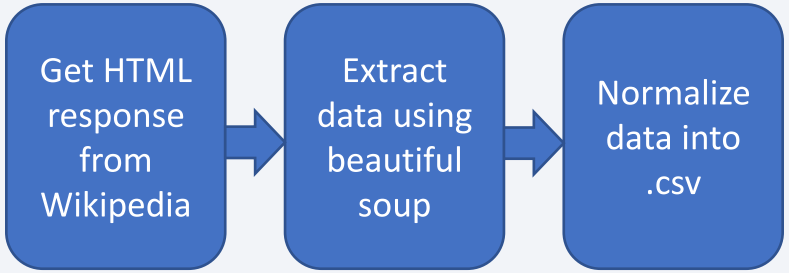


Use Space X Rest API

API returns Space X data in .JSON

Normalize data into .csv

Web Scrapping



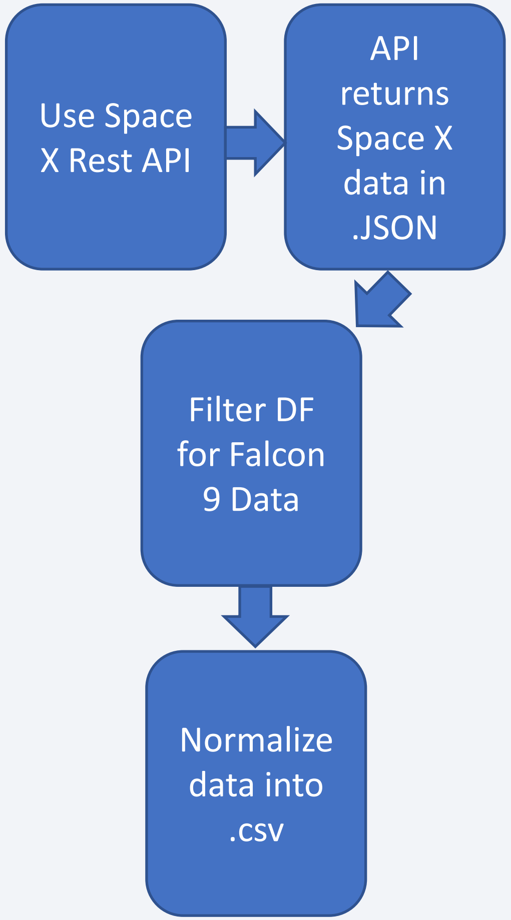
Get HTML response from Wikipedia

Extract data using beautiful soup

Normalize data into .csv

# Data Collection – SpaceX API

Space X REST API



Use Space X Rest API

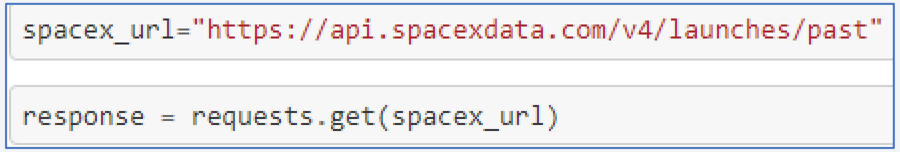
API returns Space X data in .JSON

Filter DF for Falcon 9 Data

Normalize data into .csv

GitHub URL to Notebook

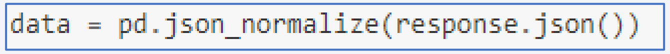
1. Getting response from API



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

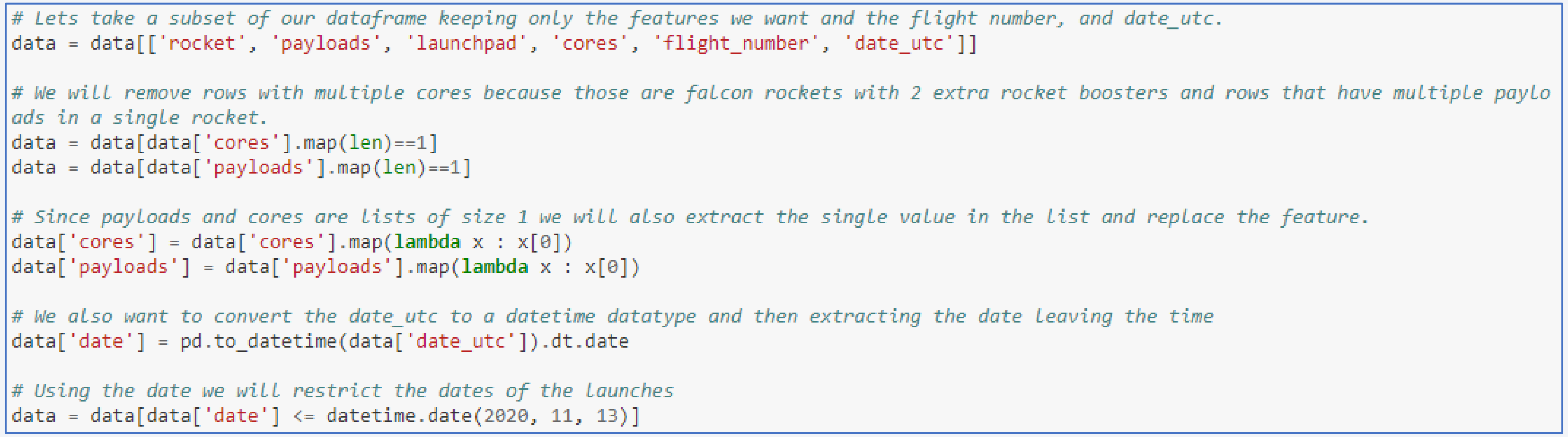
```
response = requests.get(spacex_url)
```

2. Converting response to a .json file



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

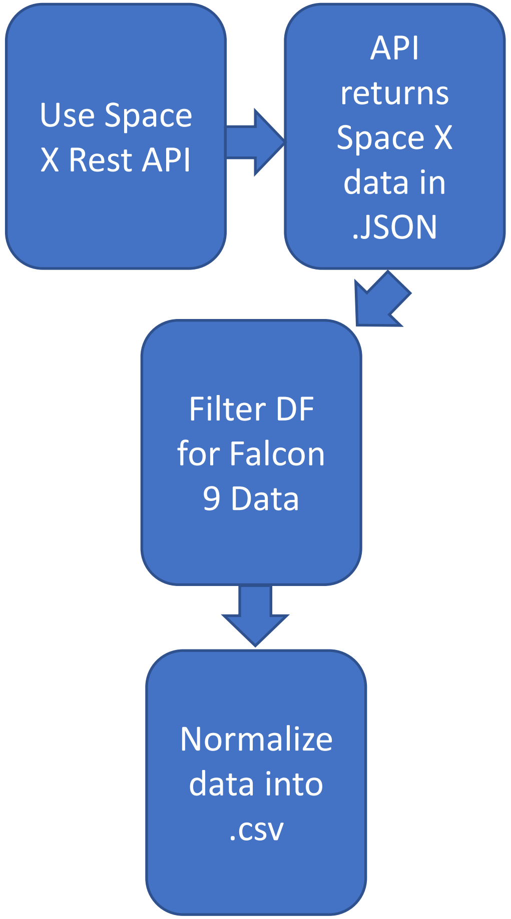
3. Clean data



```
# Lets take a subset of our dataframe keeping only the features we want and the flight number, and date_utc.
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]

# We will remove rows with multiple cores because those are falcon rockets with 2 extra rocket boosters and rows that have multiple paylo
ads in a single rocket.
data = data[data['cores'].map(len)==1]
data = data[data['payloads'].map(len)==1]

# Since payloads and cores are lists of size 1 we will also extract the single value in the list and replace the feature.
data['cores'] = data['cores'].map(lambda x : x[0])
data['payloads'] = data['payloads'].map(lambda x : x[0])

# We also want to convert the date_utc to a datetime datatype and then extracting the date leaving the time
data['date'] = pd.to_datetime(data['date_utc']).dt.date

# Using the date we will restrict the dates of the launches
data = data[data['date'] <= datetime.date(2020, 11, 13)]
```

# Data Collection – SpaceX API (continued…)

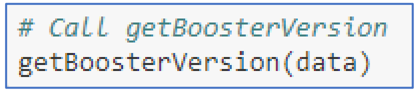
## Space X REST API



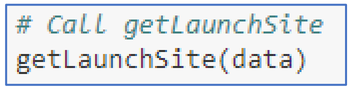
Use Space X Rest API
API returns Space X data in .JSON
Filter DF for Falcon 9 Data
Normalize data into .csv

[GitHub URL to Notebook]

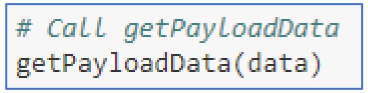
3. Clean data (Applying Custom Functions…)



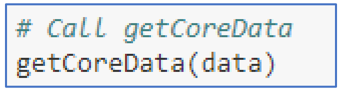
```
# Call getBoosterVersion
getBoosterVersion(data)
```



```
# Call getLaunchSite
getLaunchSite(data)
```

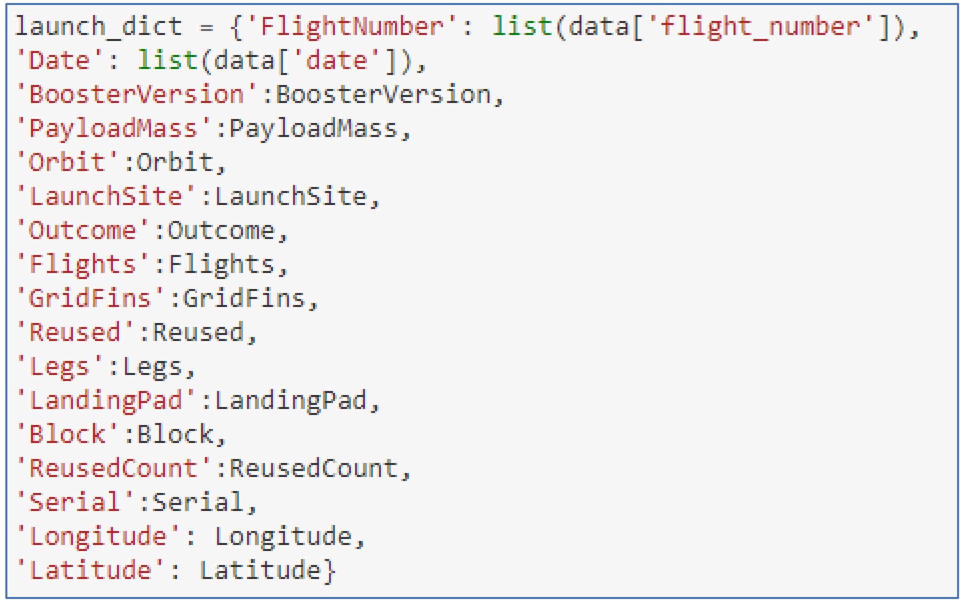


```
# Call getPayloadData
getPayloadData(data)
```

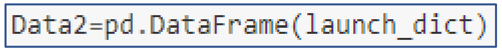


```
# Call getCoreData
getCoreData(data)
```

4. Assign list to a dictionary then dataframe

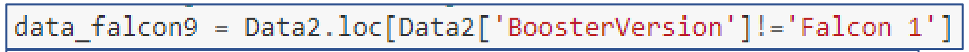


```
launch_dict = {'FlightNumber': list(data['flight_number']),
'Date': list(data['date']),
'BoosterVersion':BoosterVersion,
'PayloadMass':PayloadMass,
'Orbit':Orbit,
'LaunchSite':LaunchSite,
'Outcome':Outcome,
'Flights':Flights,
'GridFins':GridFins,
'Reused':Reused,
'Legs':Legs,
'LandingPad':LandingPad,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

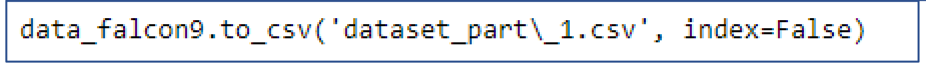


```
Data2=pd.DataFrame(launch_dict)
```

5. Filter and export into .csv



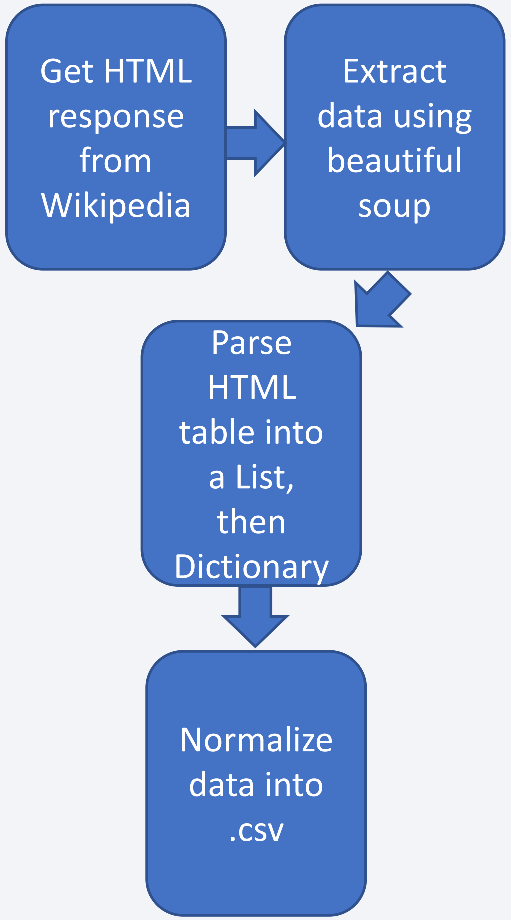
```
data_falcon9 = Data2.loc[Data2['BoosterVersion']!='Falcon 1']
```



```
data_falcon9.to_csv('dataset_part\_1.csv', index=False)
```

# Data Collection - Scraping

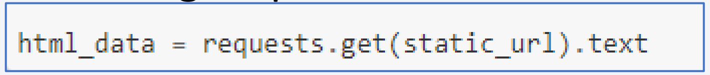
Space X REST API



Get HTML response from Wikipedia
Extract data using beautiful soup
Parse HTML table into a List, then Dictionary
Normalize data into .csv

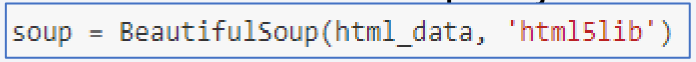
GitHub URL to Notebook

1. Getting response from HTML



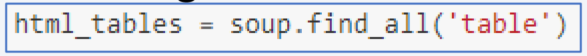
```
html_data = requests.get(static_url).text
```

2. Create BeautifulSoup Object



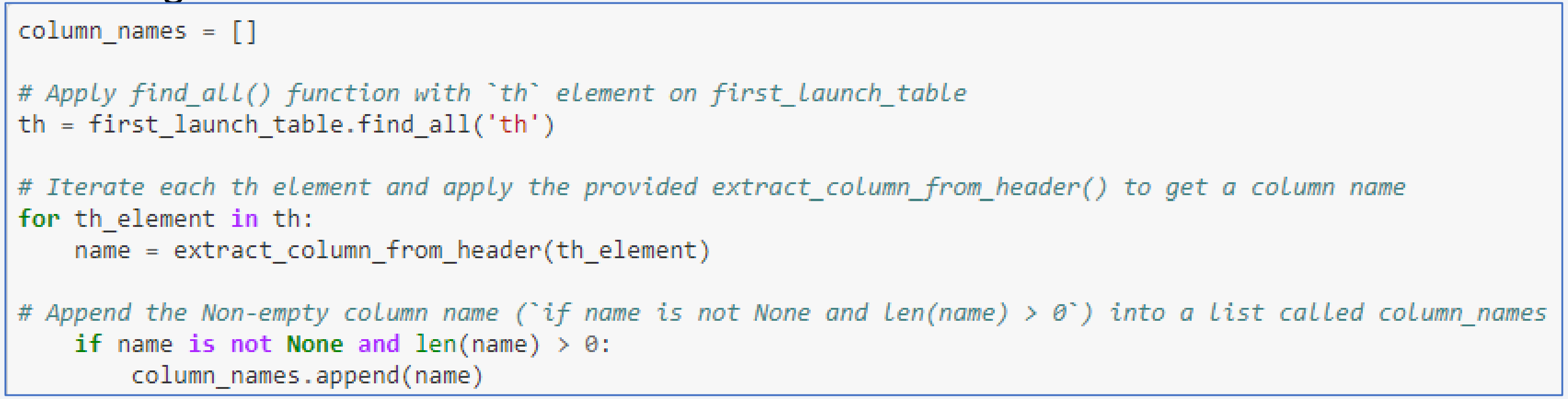
```
soup = BeautifulSoup(html_data, 'html5lib')
```

3. Finding tables



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

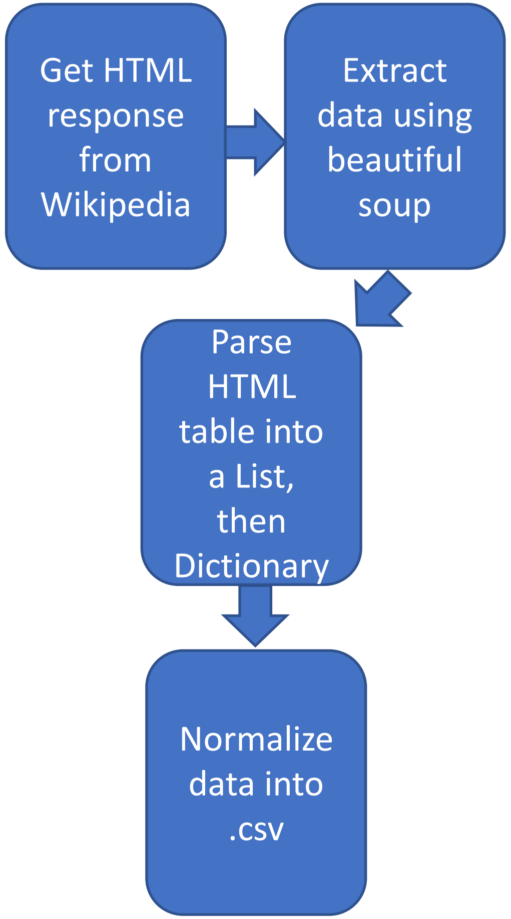
4. Getting column names



```
column_names = []

# Apply find_all() function with `th` element on first_launch_table
th = first_launch_table.find_all('th')

# Iterate each th element and apply the provided extract_column_from_header() to get a column name
for th_element in th:
    name = extract_column_from_header(th_element)

# Append the Non-empty column name (`if name is not None and len(name) > 0`) into a list called column_names
    if name is not None and len(name) > 0:
        column_names.append(name)
```

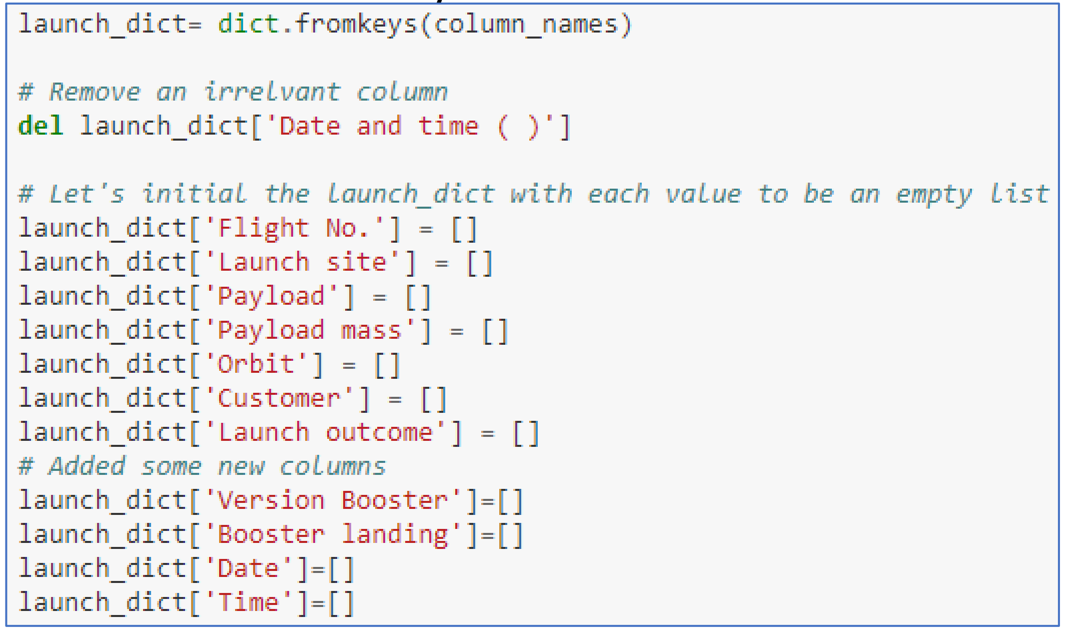
# Data Collection – Scraping (continued…)

Space X REST API


Get HTML response from Wikipedia
Extract data using beautiful soup
Parse HTML table into a List, then Dictionary
Normalize data into .csv

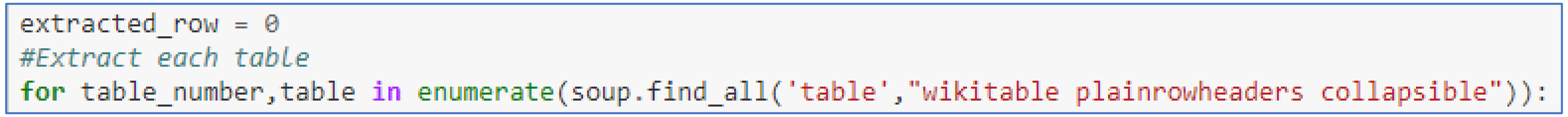
[GitHub URL to Notebook]()

5. Creation of dictionary



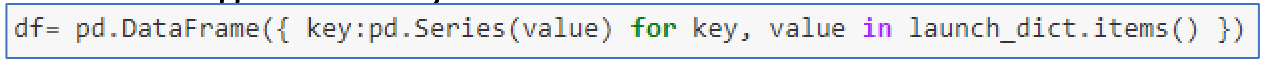
```
launch_dict= dict.fromkeys(column_names)

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

# Let's initial the launch_dict with each value to be an empty list
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']=[]
```

6. Appending data to keys(refer to block 21 in NB for the full loop)



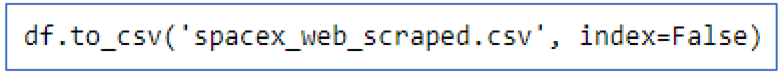
```
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders collapsible")):
```

7. Converting dictionary to dataframe



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

8. Dataframe to .csv

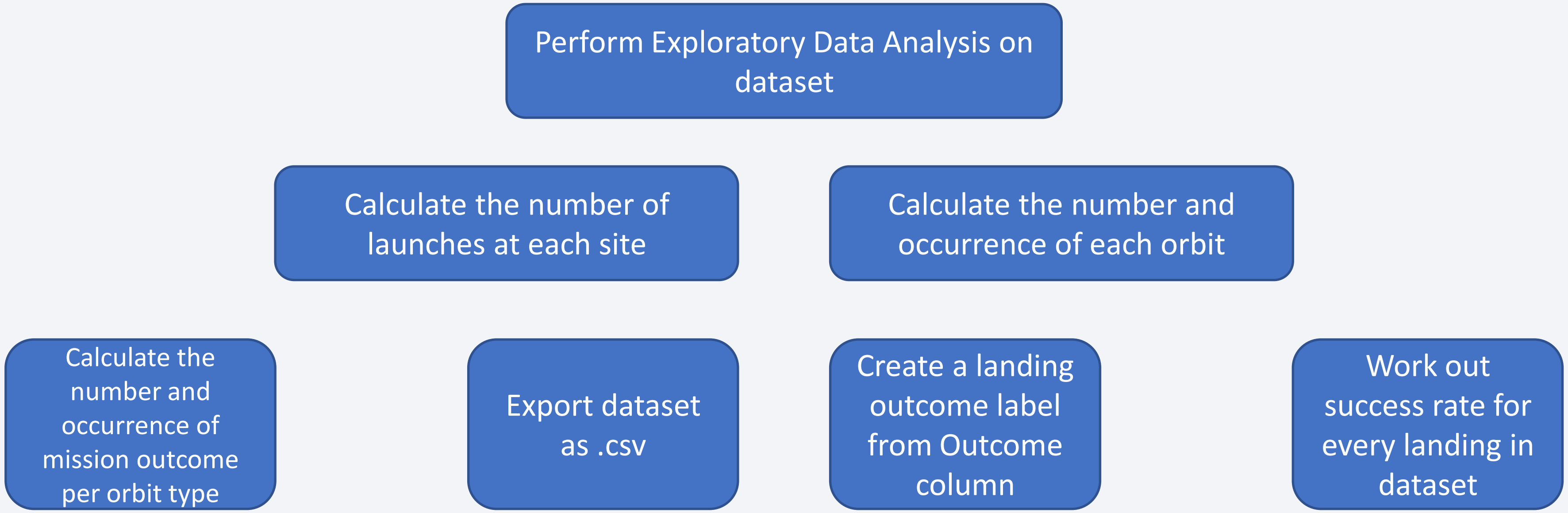


```
df.to_csv('spacex_web_scraped.csv', index=False)
```

# Data Wrangling

**Introduction:** 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 converted those outcomes into Training Labels with 1 meaning the booster successfully landed and 0, meaning it was unsuccessful.



Perform Exploratory Data Analysis on dataset

Calculate the number of launches at each site

Calculate the number and occurrence of each orbit

Calculate the number and occurrence of mission outcome per orbit type

Export dataset as .csv

Create a landing outcome label from Outcome column

Work out success rate for every landing in dataset

GitHub URL to Notebook

# EDA with Data Visualization

- Scatter graphs:
  - Payload Mass VS. Flight Number
  - Flight Number VS. Launch Site
  - Launch Site VS. Payload Mass
  - Orbit VS. Flight Number
  - Orbit VS. Payload Mass
- Bar graph:
  - Success Rate VS. Orbit Type
- Line graph:
  - Success Rate VS. Year

Scatter plots show how much one variable is affected by another. We used this to establish correlation between features.

A bar diagram makes it easy to compare sets of data between different groups at a glance. We used this to find out the success rate at each orbit type.

Line graphs are useful in that they show data variables and trends very clearly. We observed that the success rate kept increasing till 2020.

[GitHub URL to Notebook]

# EDA with SQL

[GitHub URL to Notebook]

- Performed SQL queries to gather information about dataset.
- The following queries were performed:
  - Displayed the names of the unique launch sites in the space mission
  - Displayed 5 records where launch sites begin with the string 'CCA’
  - Displayed the total payload mass carried by boosters launched by NASA (CRS)
  - Displayed average payload mass carried by booster version F9 v1.1
  - Listed the date when the first successful landing outcome in ground pad was acheived.
  - Listed the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000
  - Listed the total number of successful and failure mission outcomes
  - Listed the names of the booster_versions which have carried the maximum payload mass.
  - Listed the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015
  - Ranked 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

# Build an Interactive Map with Folium

- Visualized launch data into an interactive map:
  - Took Latitude and Longitude coordinates at each launch site and added *Circle Markers* around each launch site and labeled them
  - Assigned launch outcomes(failures, successes) to classes 0 and 1 with Green and Red markers on the map with MarkerCluster()
  - We calculated the distance from the Launch Site to various landmarks to find patterns on what's around the launch site. We used lines on the map to represent the calculated distance.
    - Examples of some trends found:
      - Close proximity to city? No
      - Close proximity to coastlines? Yes

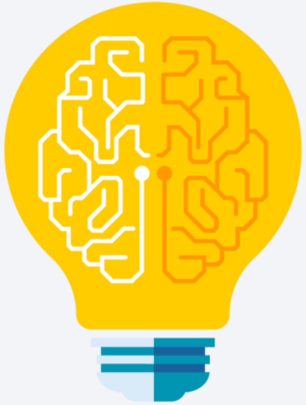
GitHub URL to Notebook

# Build a Dashboard with Plotly Dash

- A dashboard was built using Plotly dash:
  - Graphs:
    - Pie chart showing the total launches by a certain site/all sites.
      - It displays the relative proportions of success rates per site.
    - Scatter graphs showing the relationship with Outcome(success/failure) and Payload Mass for the different boosters.
      - Best method to show you the relationship between both features.
      - If the data has a non-linear patter it is the best choice to visualize it.
      - The range of data flow(max, min) can be determined easily.

GitHub URL to Notebook

# Predictive Analysis (Classification)



- Building Model
  - Load data into NumPy and Pandas
  - Transform data
  - Split data into training and test data sets
  - Decide which type of model to use
  - Set our parameters and models to GridSearchCV
  - Fit our datasets into the GridSearchCV objects and train the models.
- Evaluating Model
  - Check accuracy for each model
  - Get tuned hyperparameters for each type of model
  - Plot confusion Matrix

- Improving Model
  - Feature engineering
  - Model tuning
- Finding the best performing model
  - The model with the best accuracy score wins the best performing model.

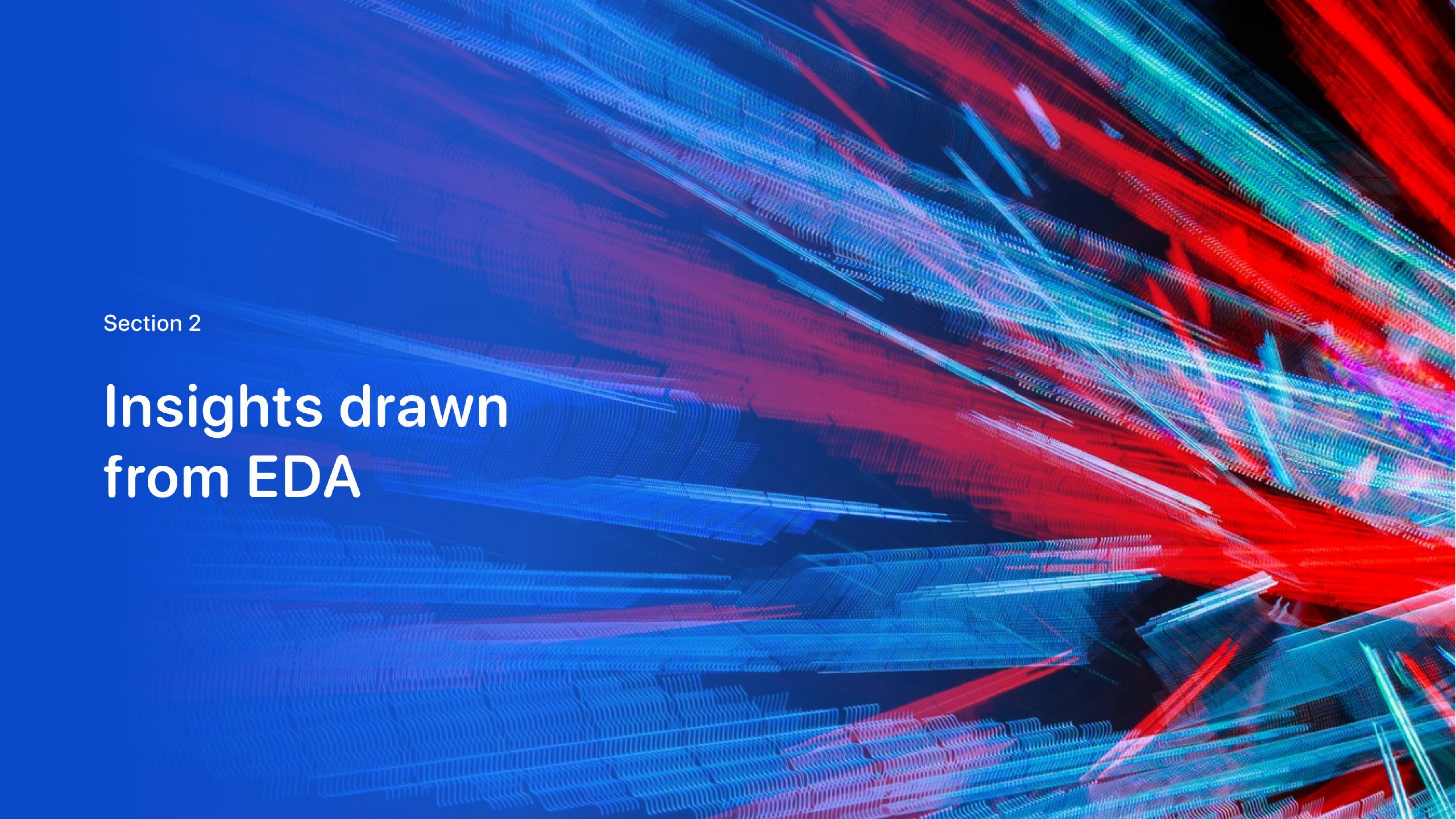
GitHub URL to Notebook

# Results

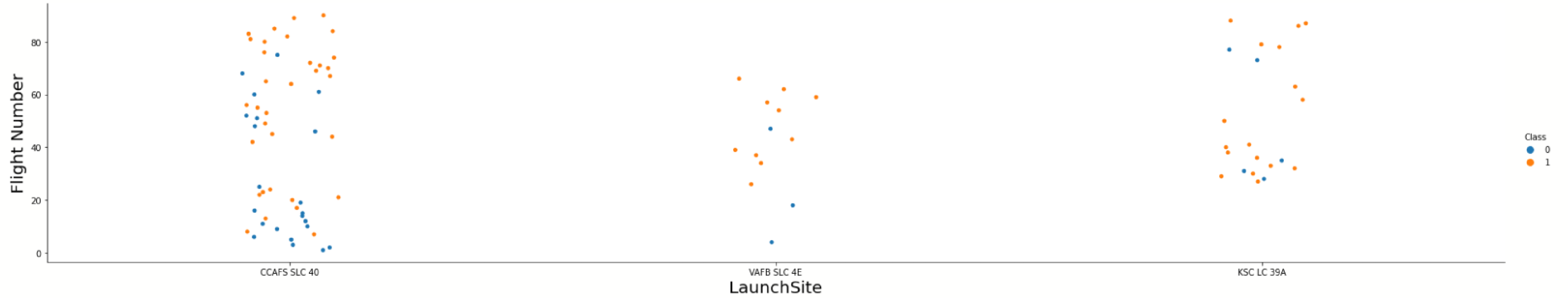
- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results

Section 2

# Insights drawn from EDA

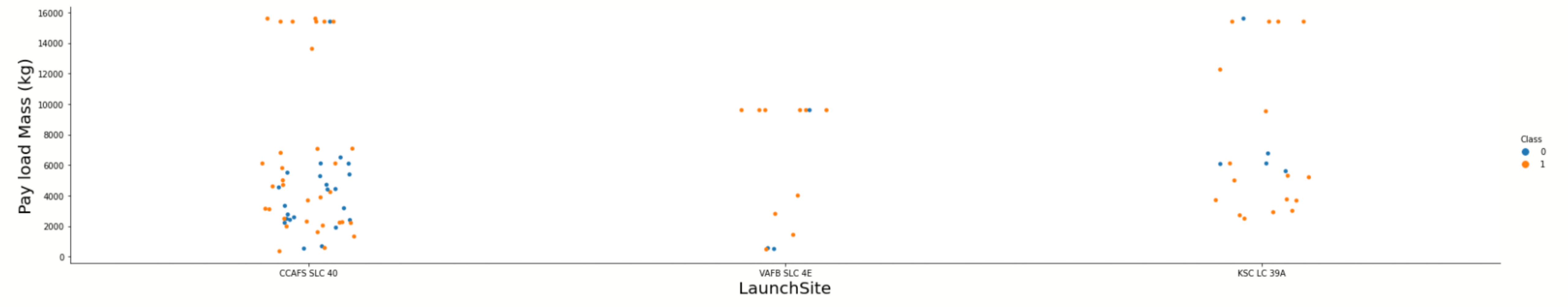


# Flight Number vs. Launch Site



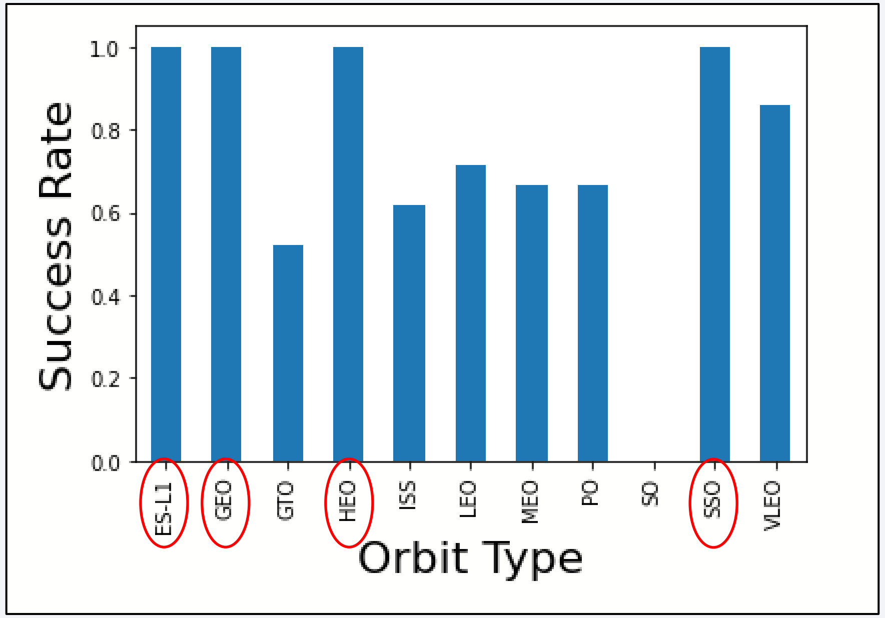
The greater number of flights at a launch site the greater the success rate at that launch site.

# Payload vs. Launch Site



The greater the payload mass for launch site CCAFS SLC 40 the higher the success rate for the booster. There is not a clear pattern to be found using this visualization to decide if the launch site is dependent on Payload Mass for a successful booster landing.

# Success Rate vs. Orbit Type



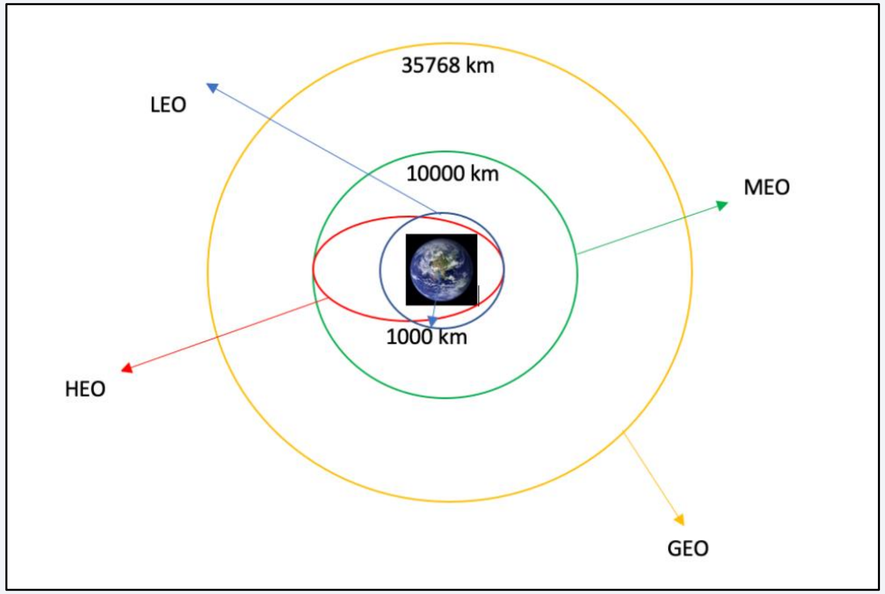
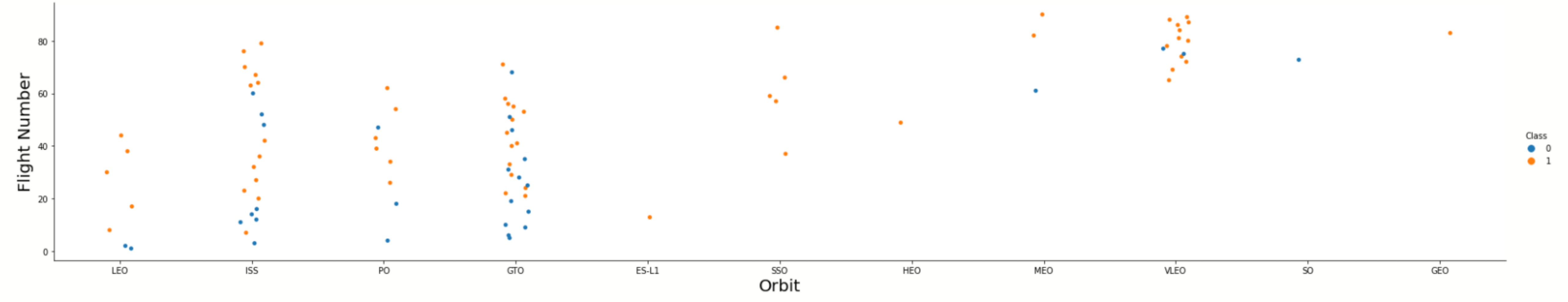


Diagram showing common orbit types Space X uses

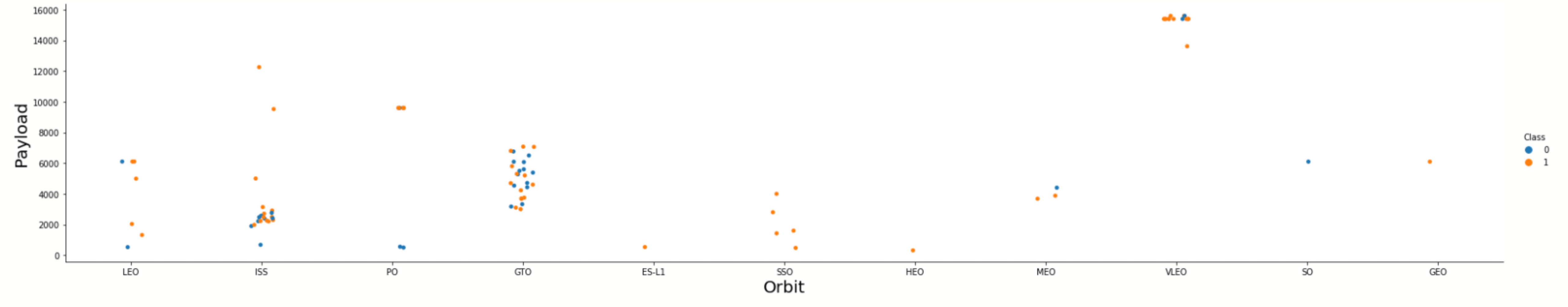
Orbit GEO, HEO, SSO, ES-L1 have the best success rate per the launch data.

# Flight Number vs. Orbit Type



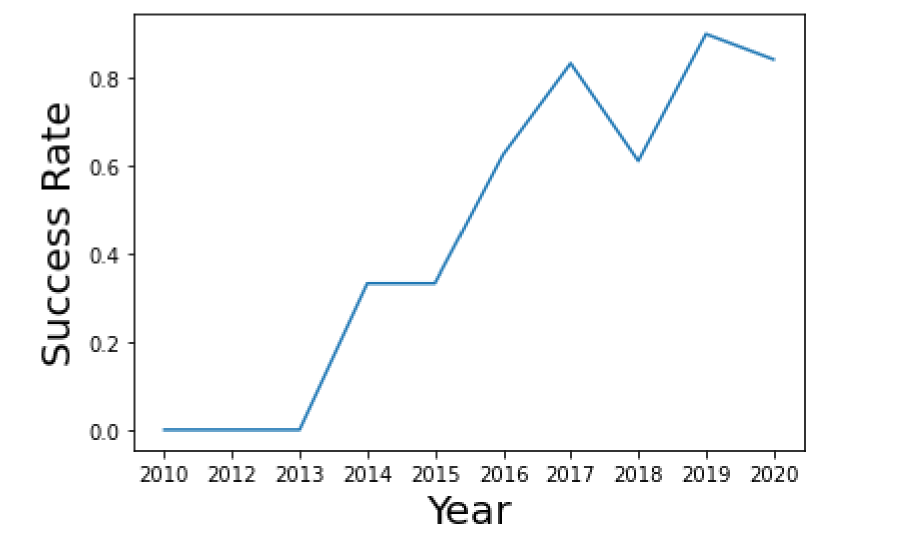
In the LEO orbit the success appears related to the number of flights; on the other hand, there seems to be no relationship with flight number when in GTO orbit.

# Payload vs. Orbit Type



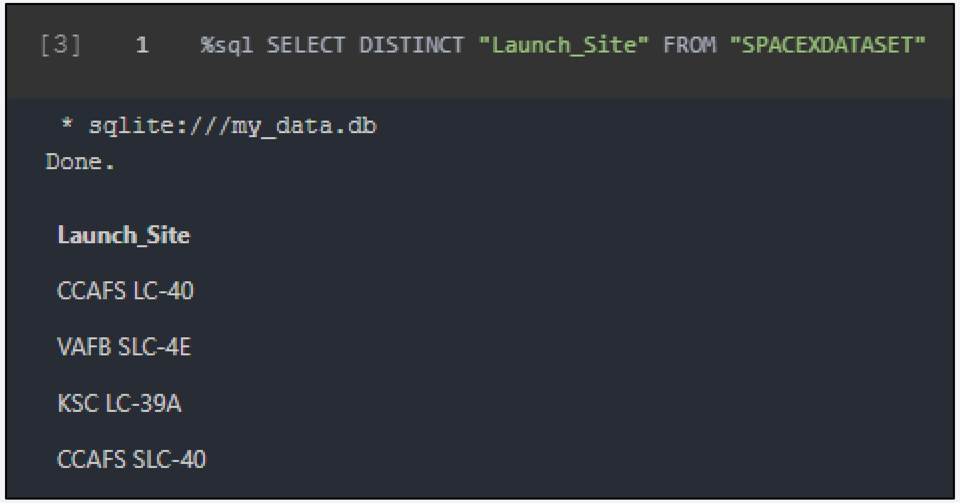
Heavy payloads have a negative influence on GTO orbits and positive on GTO and Polar LEO (ISS) orbits.

# Launch Success Yearly Trend



- You can observe that the success rate since 2013 kept increasing till 2020

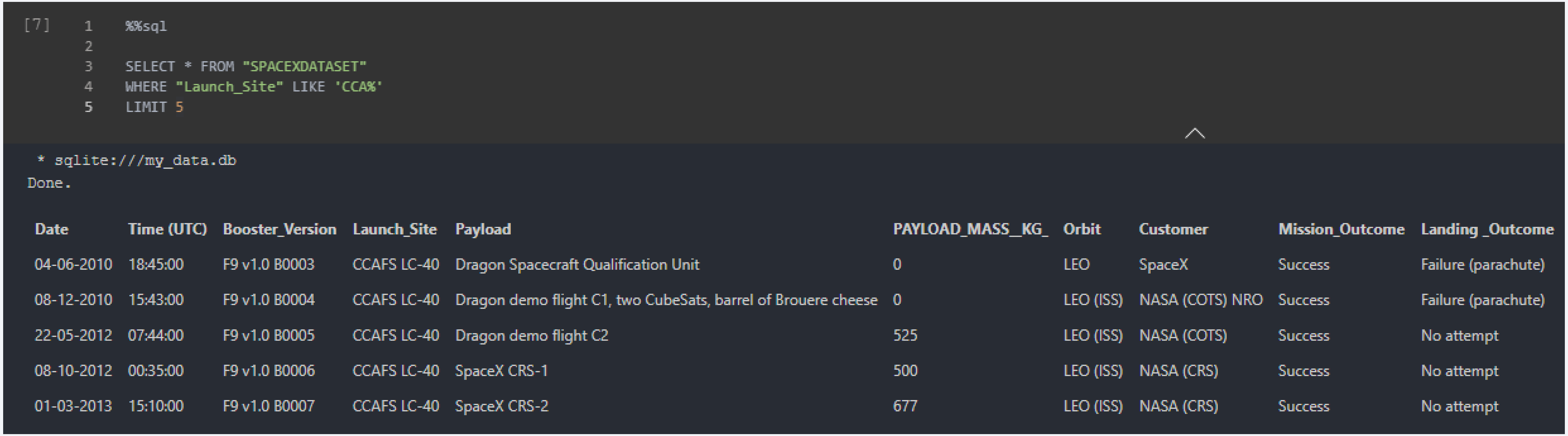
# All Launch Site Names



```
[3]  %sql SELECT DISTINCT "Launch_Site" FROM "SPACEXDATASET"

 * sqlite:///my_data.db
Done.

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

Using the word DISTINCT in the query means that it will only show the unique values in the Launch_Site column from the data set.

# Launch Site Names Begin with 'CCA'



[7]

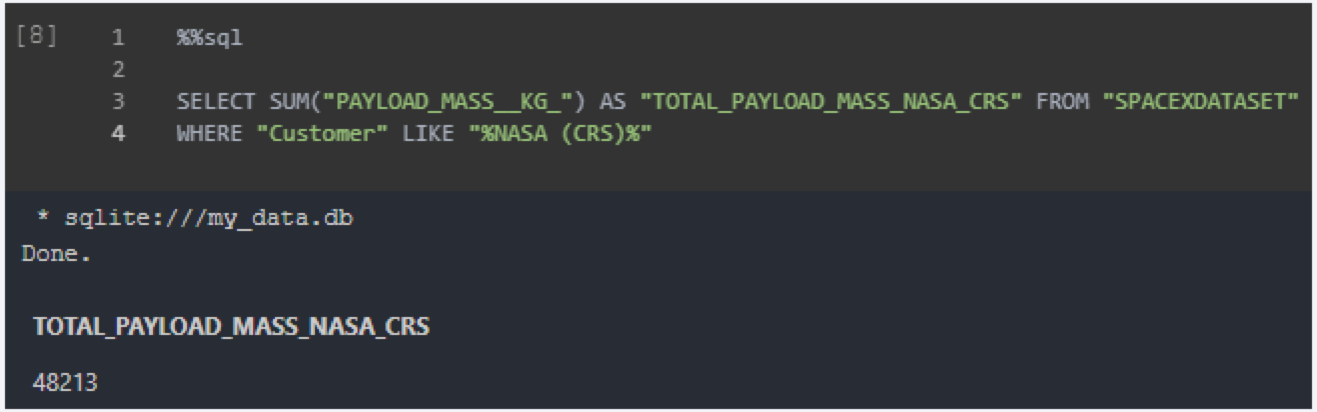
```
%%sql

SELECT * FROM "SPACEXDATASET"
WHERE "Launch_Site" LIKE 'CCA%'
LIMIT 5
```

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

| Date | Time (UTC) | Booster_Version | Launch_Site | Payload | PAYLOAD_MASS__KG_ | Orbit | Customer | Mission_Outcome | Landing _Outcome |
|---|---|---|---|---|---|---|---|---|---|
| 04-06-2010 | 18:45:00 | F9 v1.0 B0003 | CCAFS LC-40 | Dragon Spacecraft Qualification Unit | 0 | LEO | SpaceX | Success | Failure (parachute) |
| 08-12-2010 | 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) |
| 22-05-2012 | 07:44:00 | F9 v1.0 B0005 | CCAFS LC-40 | Dragon demo flight C2 | 525 | LEO (ISS) | NASA (COTS) | Success | No attempt |
| 08-10-2012 | 00:35:00 | F9 v1.0 B0006 | CCAFS LC-40 | SpaceX CRS-1 | 500 | LEO (ISS) | NASA (CRS) | Success | No attempt |
| 01-03-2013 | 15:10:00 | F9 v1.0 B0007 | CCAFS LC-40 | SpaceX CRS-2 | 677 | LEO (ISS) | NASA (CRS) | Success | No attempt |

Using the word LIMIT 5 in the query means that it will only show 5 records from the data set and LIKE keyword has a wild card with the words *'CCA%'* the percentage in the end suggest that the launch site name must start with CCA.

# Total Payload Mass by Customer NASA (CRS)



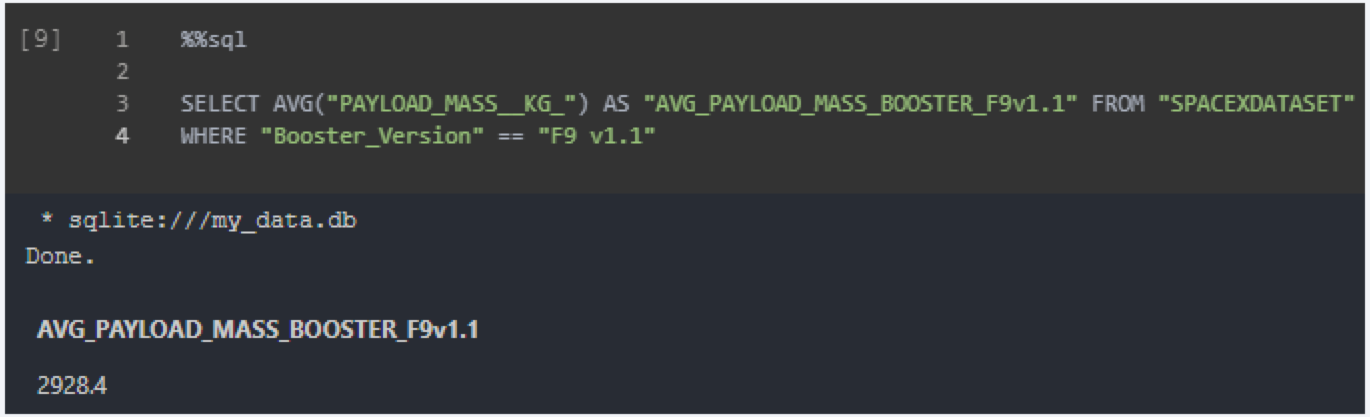
```
[8]
%%sql

SELECT SUM("PAYLOAD_MASS__KG_") AS "TOTAL_PAYLOAD_MASS_NASA_CRS" FROM "SPACEXDATASET"
WHERE "Customer" LIKE "%NASA (CRS)%"
```

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

| TOTAL_PAYLOAD_MASS_NASA_CRS |
|---|
| 48213 |

Using the SUM function we get the total of the sum in the Payload Mass column. The WHERE clause then filters the dataset to only perform the calculation on Nasa (CRS) costumers.

# Average Payload Mass by F9 v1.1

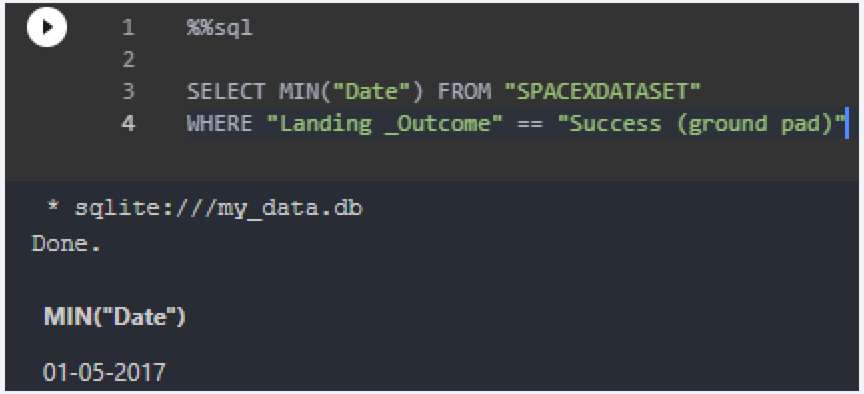


```
[9]
%%sql

SELECT AVG("PAYLOAD_MASS__KG_") AS "AVG_PAYLOAD_MASS_BOOSTER_F9v1.1" FROM "SPACEXDATASET"
WHERE "Booster_Version" == "F9 v1.1"
```

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

AVG_PAYLOAD_MASS_BOOSTER_F9v1.1

2928.4
```

Using the AVG function we get the average in the Payload Mass column. The WHERE clause then filters the dataset to only perform the calculation on the booster version F9 v1.1.

# First Successful Ground Landing Date

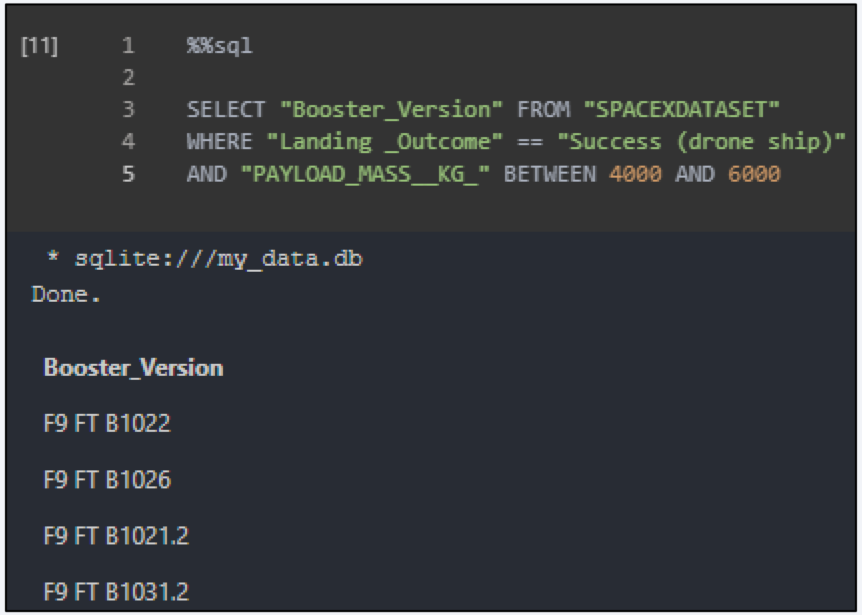


```
%%sql

SELECT MIN("Date") FROM "SPACEXDATASET"
WHERE "Landing _Outcome" == "Success (ground pad)"
```

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

 MIN("Date")

 01-05-2017
```

Using the MIN function we get the minimum date. The WHERE clause then filters the dataset to only get the minimum date for landing outcomes successes on a ground pad.

# Successful Drone Ship Landing with Payload between 4000 and 6000



```
[11]
%%sql

SELECT "Booster_Version" FROM "SPACEXDATASET"
WHERE "Landing _Outcome" == "Success (drone ship)"
AND "PAYLOAD_MASS__KG_" BETWEEN 4000 AND 6000
```

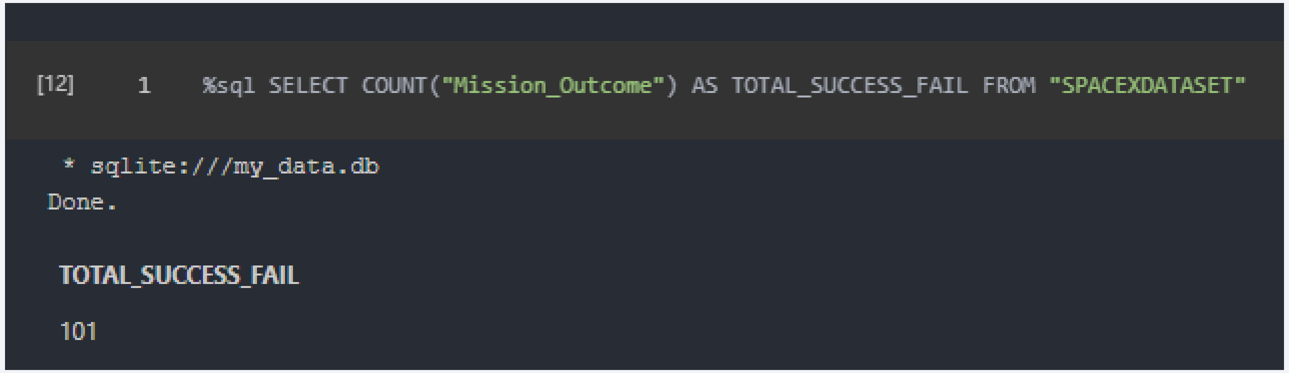
```
 * sqlite:///my_data.db
Done.

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

Selecting only Booster Versions.

The WHERE clause filters the dataset to Landing_Outcome = Success (drone ship).

The AND , BETWEEN clause specifies additional filter conditions to select the rows with payloads between 4000 and 6000 kg.

# Total Number of Successful and Failure Mission Outcomes

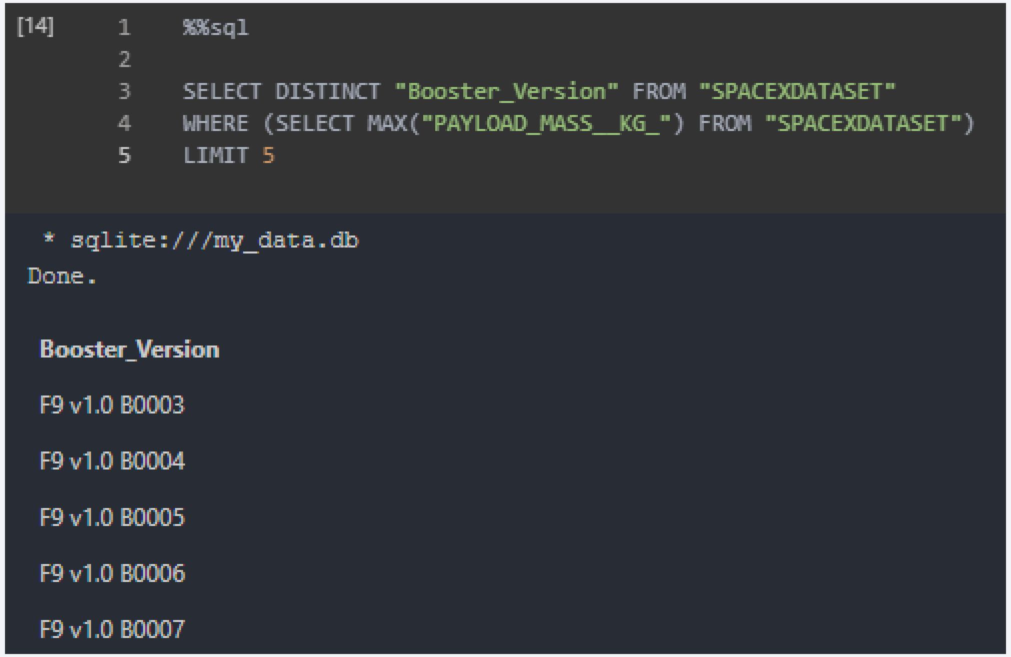


```
%sql SELECT COUNT("Mission_Outcome") AS TOTAL_SUCCESS_FAIL FROM "SPACEXDATASET"
```

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

TOTAL_SUCCESS_FAIL
101
```

Using the COUNT function we get the total amount of outcomes both of success and failure.

# Boosters Carried Maximum Payload

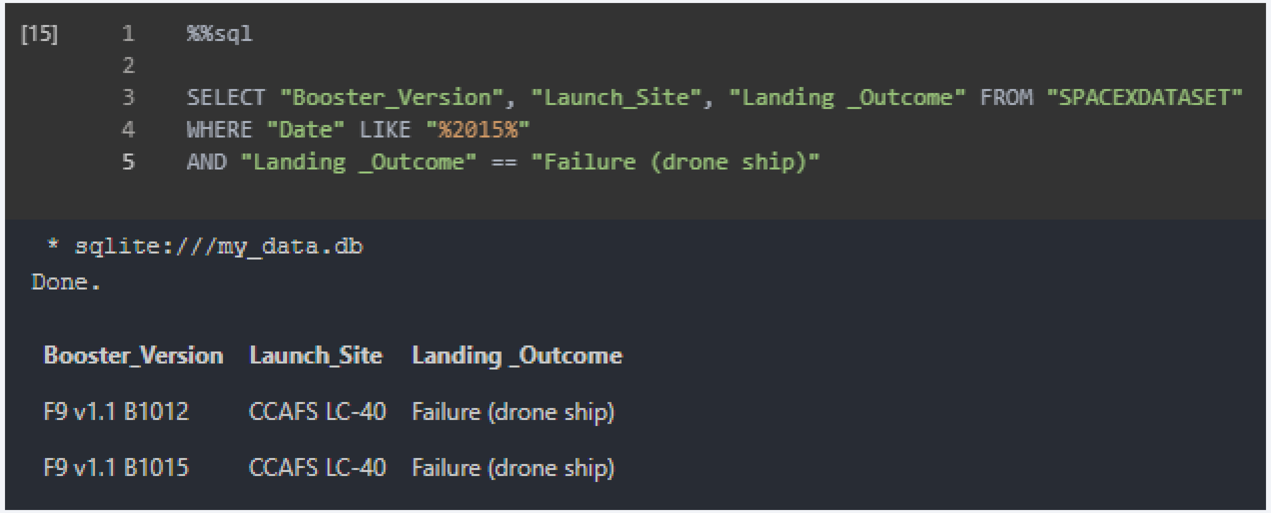


```
[14]
%%sql

SELECT DISTINCT "Booster_Version" FROM "SPACEXDATASET"
WHERE (SELECT MAX("PAYLOAD_MASS__KG_") FROM "SPACEXDATASET")
LIMIT 5
```

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

Booster_Version
F9 v1.0 B0003
F9 v1.0 B0004
F9 v1.0 B0005
F9 v1.0 B0006
F9 v1.0 B0007
```

Using the word DISTINCT in the query means that it will only show unique values in the booster version column. By using a sub query, we only select the boosters that carried the maximum payload using the MAX function. The query was limited to 5 in order to fit this slide.

# 2015 Launch Records



[15]

```
%%sql

SELECT "Booster_Version", "Launch_Site", "Landing _Outcome" FROM "SPACEXDATASET"
WHERE "Date" LIKE "%2015%"
AND "Landing _Outcome" == "Failure (drone ship)"
```

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

| Booster_Version | Launch_Site | Landing _Outcome |
|---|---|---|
| F9 v1.1 B1012 | CCAFS LC-40 | Failure (drone ship) |
| F9 v1.1 B1015 | CCAFS LC-40 | Failure (drone ship) |

Selected the booster version, launch site and landing outcome.

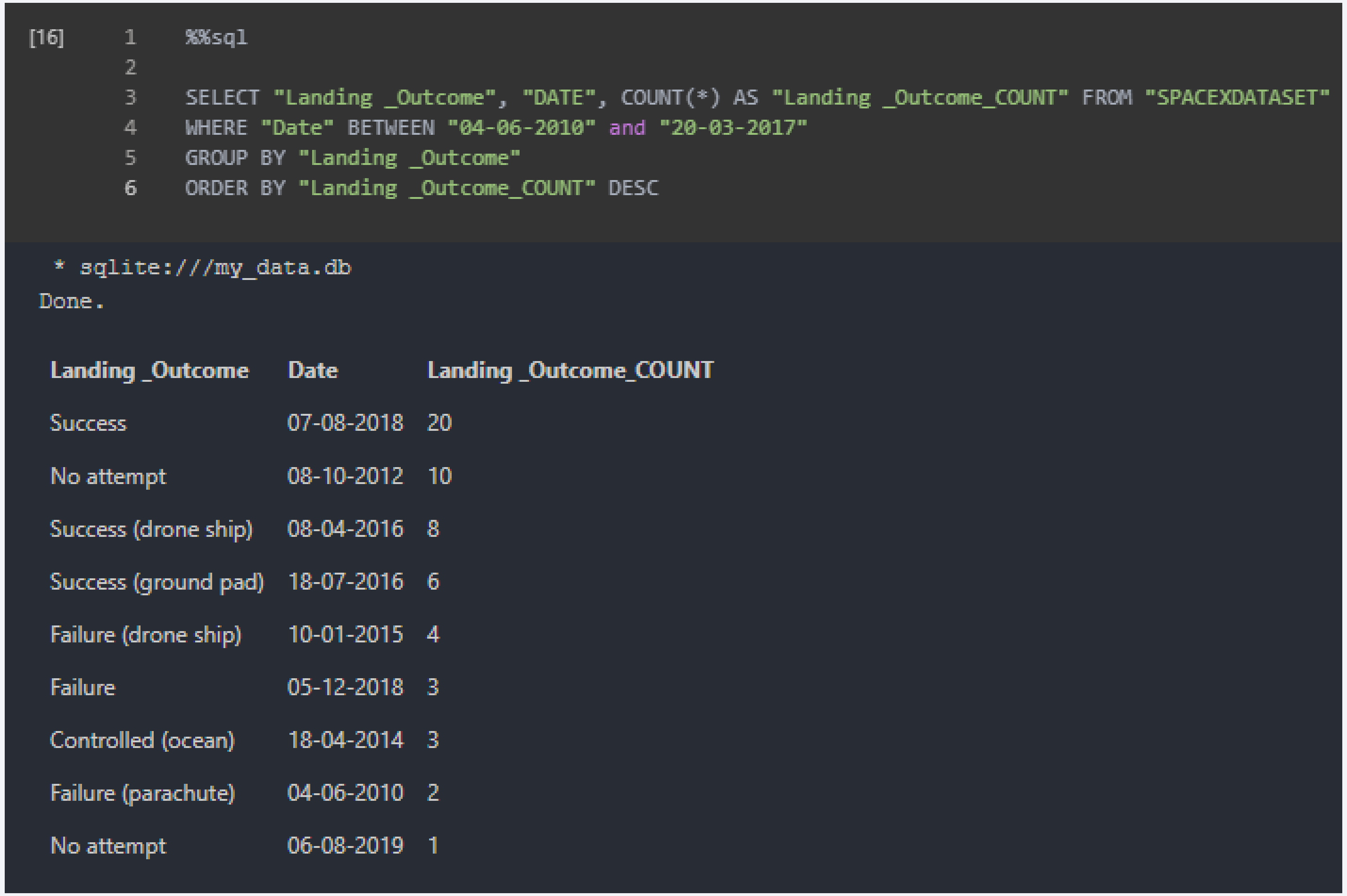
Used the WHERE, AND clause to specify the records for 2015 and the outcomes where a failure happened on a drone ship.

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

Selected the landing outcome, date and landing outcome count (using the COUNT function).

Used the WHERE, BETWEEN, AND clause to specify the records between 2010-06-04 and 2017-03-20.

Used GROUP BY and ORDER BY to rank the landing outcomes and put them in descending order of counts.



[16]

```
%%sql

SELECT "Landing _Outcome", "DATE", COUNT(*) AS "Landing _Outcome_COUNT" FROM "SPACEXDATASET"
WHERE "Date" BETWEEN "04-06-2010" and "20-03-2017"
GROUP BY "Landing _Outcome"
ORDER BY "Landing _Outcome_COUNT" DESC
```

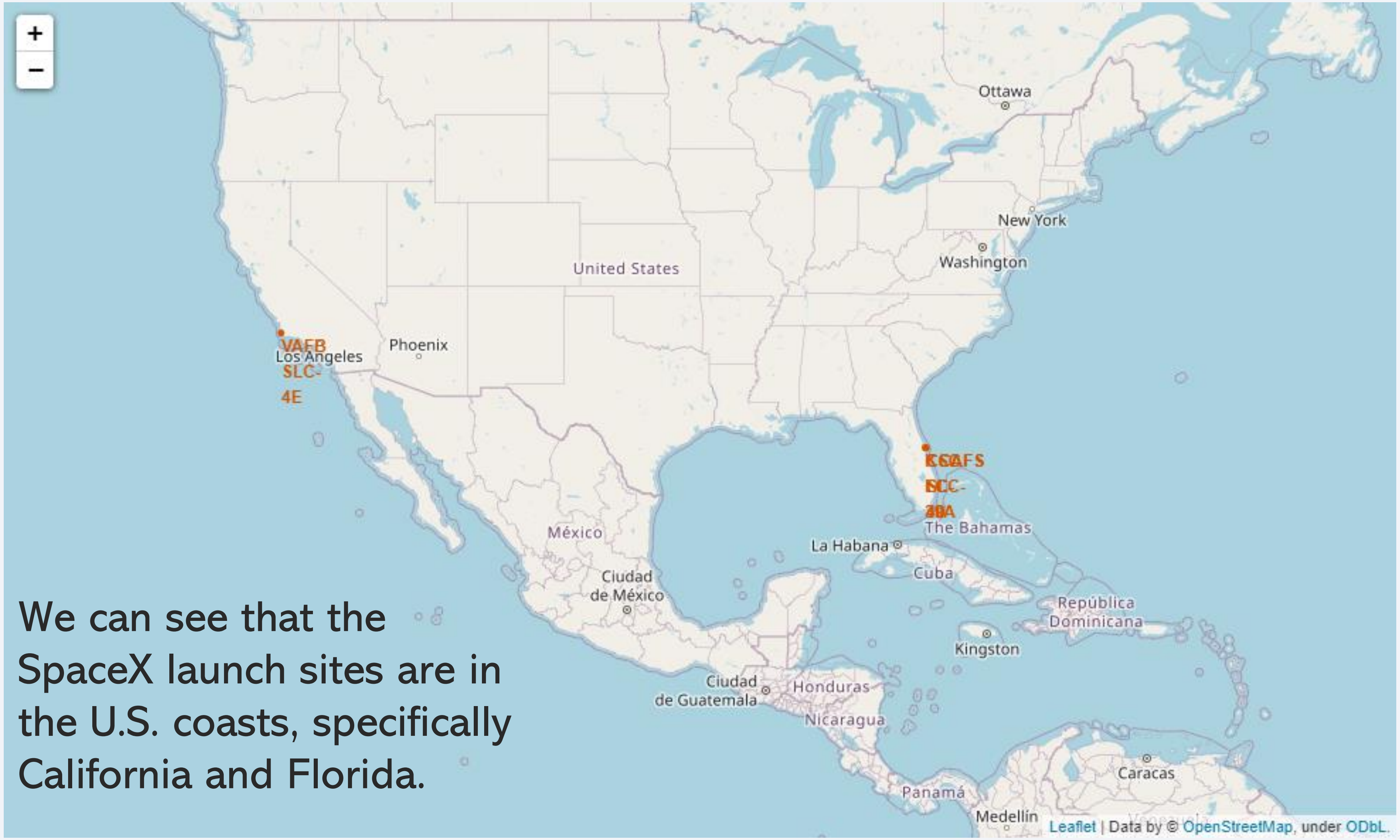
```
 * sqlite:///my_data.db
Done.
```

| Landing _Outcome | Date | Landing _Outcome_COUNT |
|---|---|---|
| Success | 07-08-2018 | 20 |
| No attempt | 08-10-2012 | 10 |
| Success (drone ship) | 08-04-2016 | 8 |
| Success (ground pad) | 18-07-2016 | 6 |
| Failure (drone ship) | 10-01-2015 | 4 |
| Failure | 05-12-2018 | 3 |
| Controlled (ocean) | 18-04-2014 | 3 |
| Failure (parachute) | 04-06-2010 | 2 |
| No attempt | 06-08-2019 | 1 |

Section 4

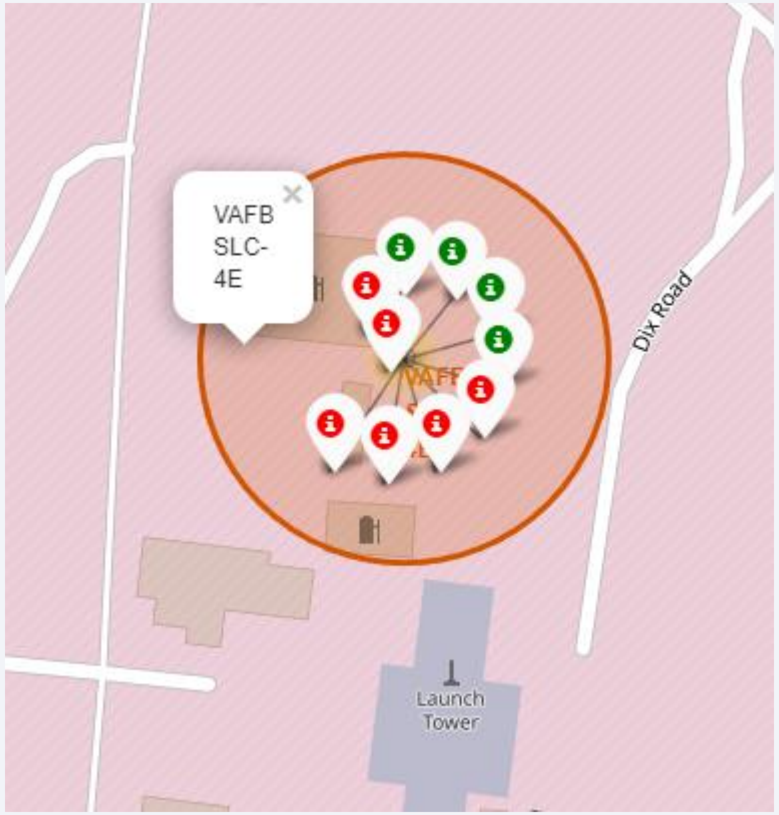
# Launch Sites Proximities Analysis

# All launch sites global map markers

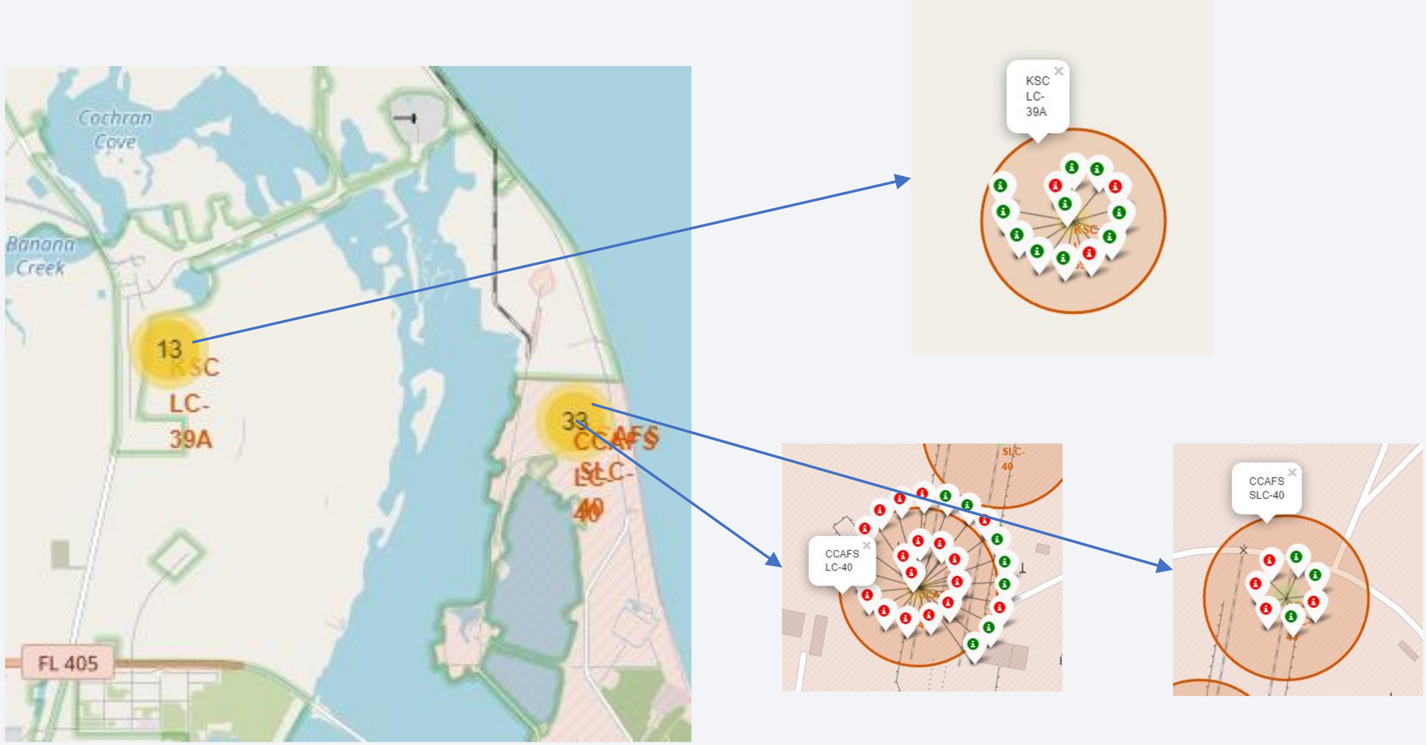


We can see that the SpaceX launch sites are in the U.S. coasts, specifically California and Florida.

# Color labeled markers of outcomes in Launch sites



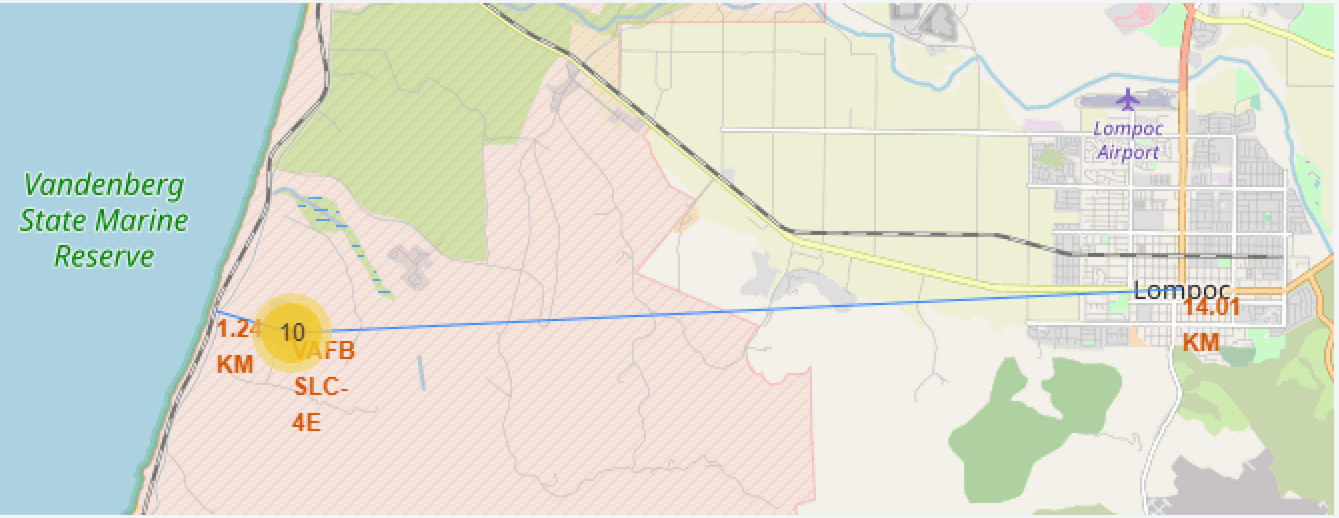
California Launch Site



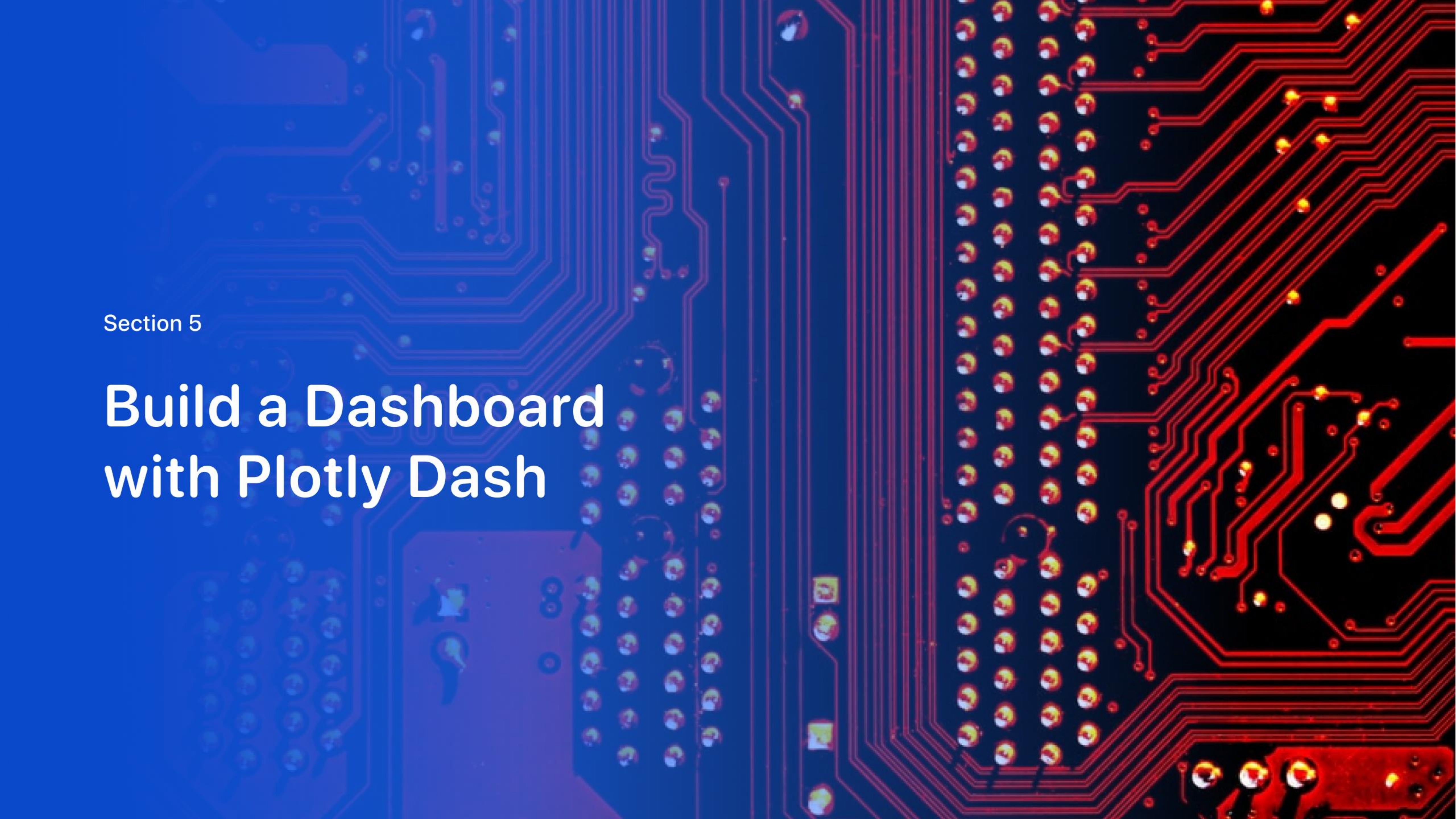
Florida Launch Sites

Green Markers show successful launches and red markers show failures.

# Launch Site distance to landmarks



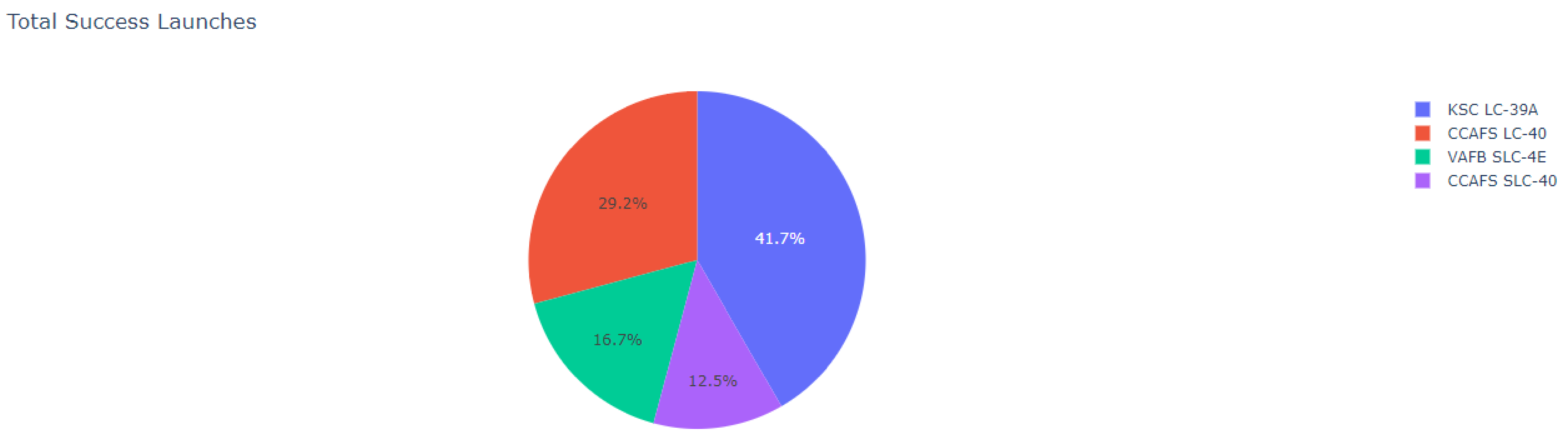
Close proximity to city? No
Close proximity to coastlines? Yes



Section 5

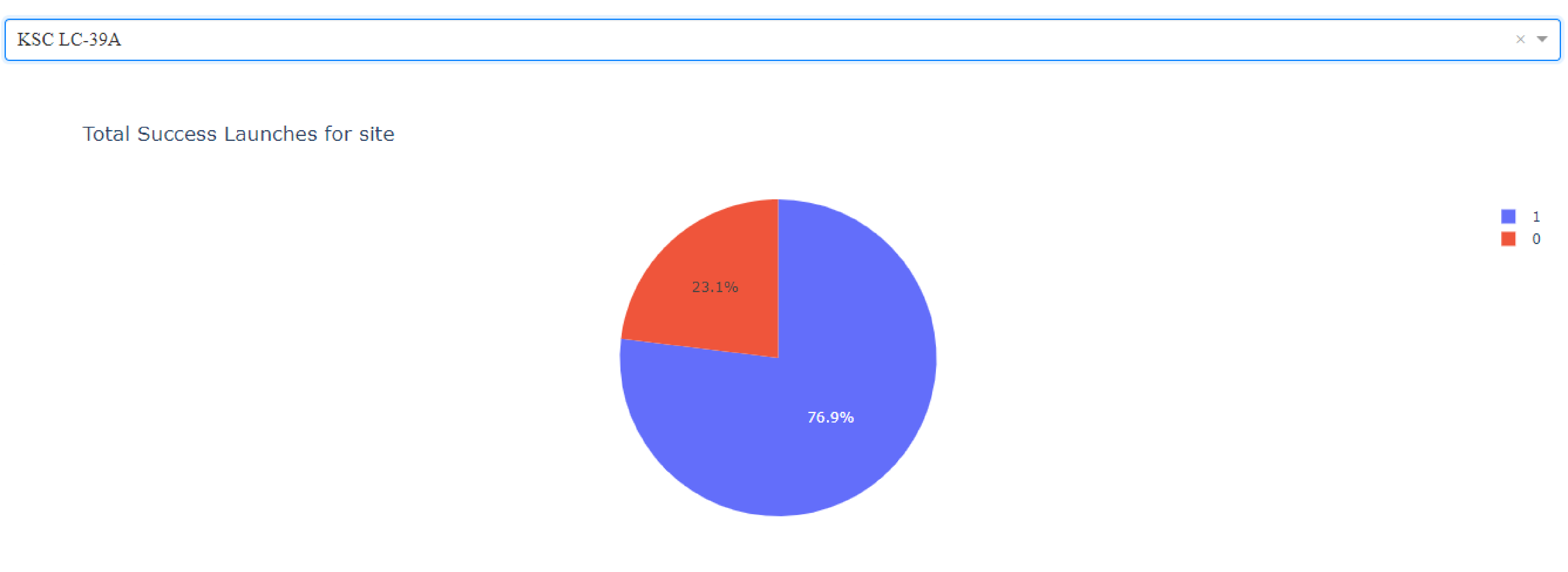
# Build a Dashboard with Plotly Dash

# Pie chart showing the success percentage by each launch site



We can see that KSC LC-39A had the most successful launches from all the sites.

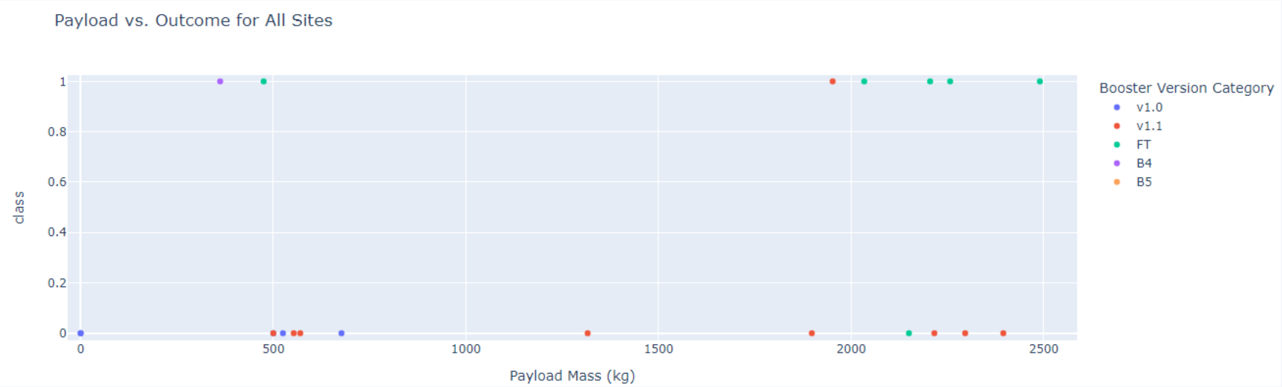
# Pie chart for the launch site with highest success ratio

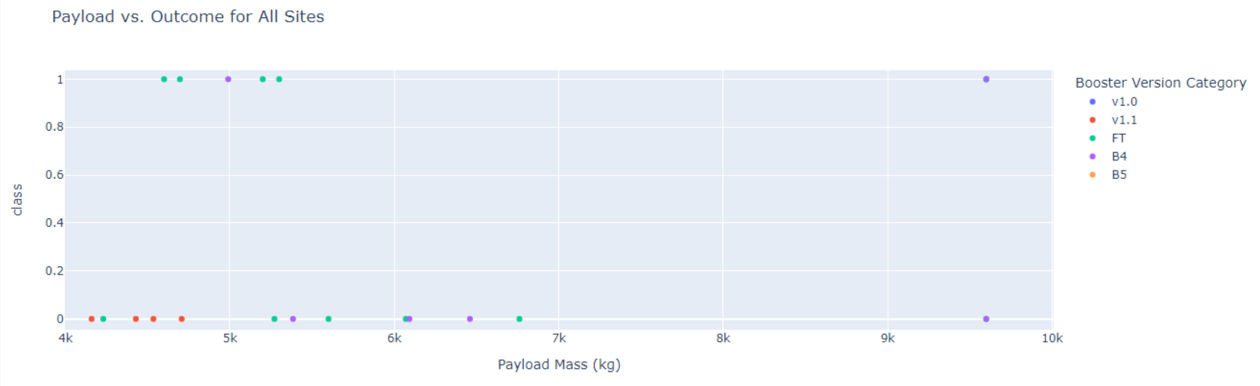


KSC LC-39A

KSC LC-39A achieved a 76.9% success rate while getting a 23.1 % failure rate.

# Launch Outcomes VS Payload Scatter plot for all sites.





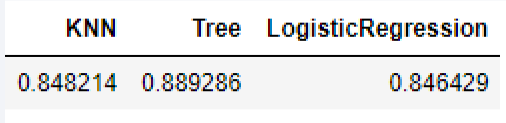
We can see the success rates for low weighted payloads is higher than the heavier ones.

Section 6

# Predictive Analysis (Classification)

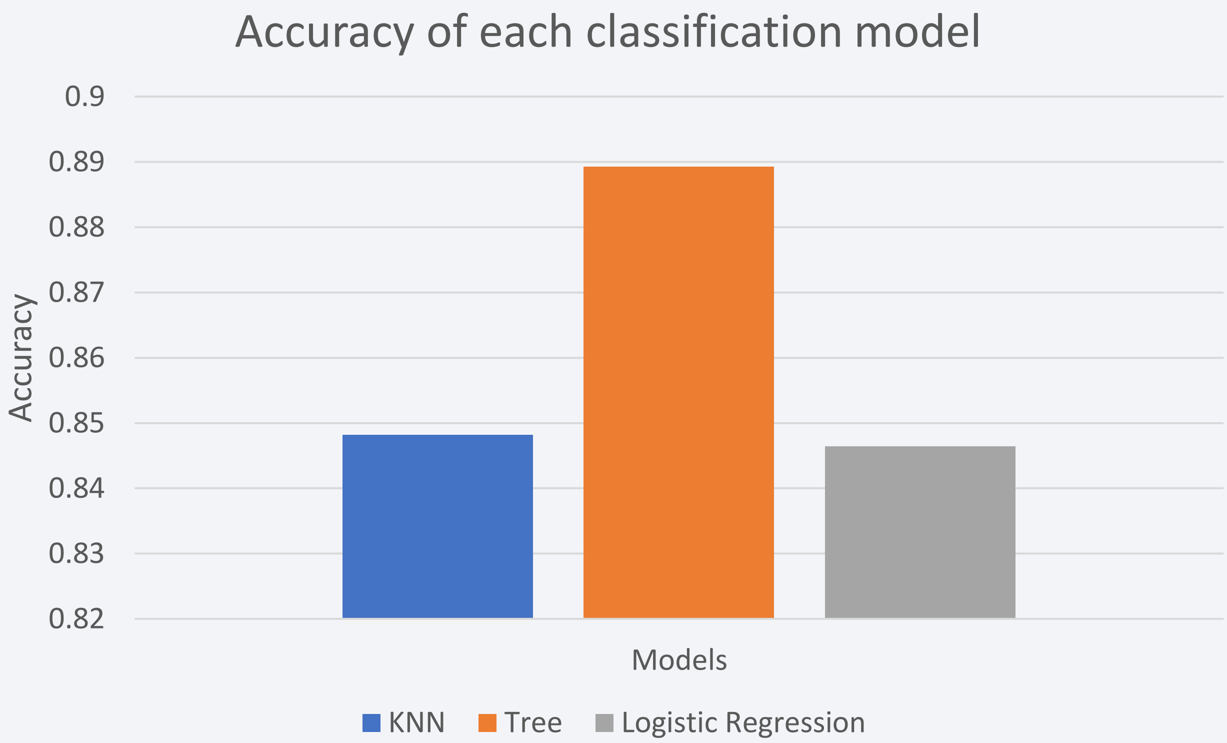
# Classification Accuracy

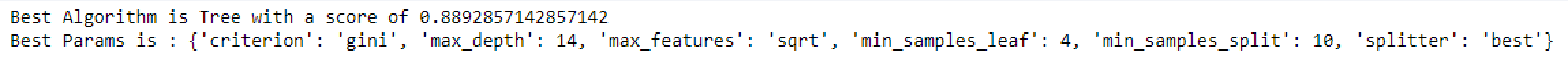
Several classification models were built, trained and tested with the data set. Here are their accuracies with predicting the test data:



| KNN | Tree | LogisticRegression |
|---|---|---|
| 0.848214 | 0.889286 | 0.846429 |

By comparing the accuracies of each model we can see that the Tree model performs the best using our test data.



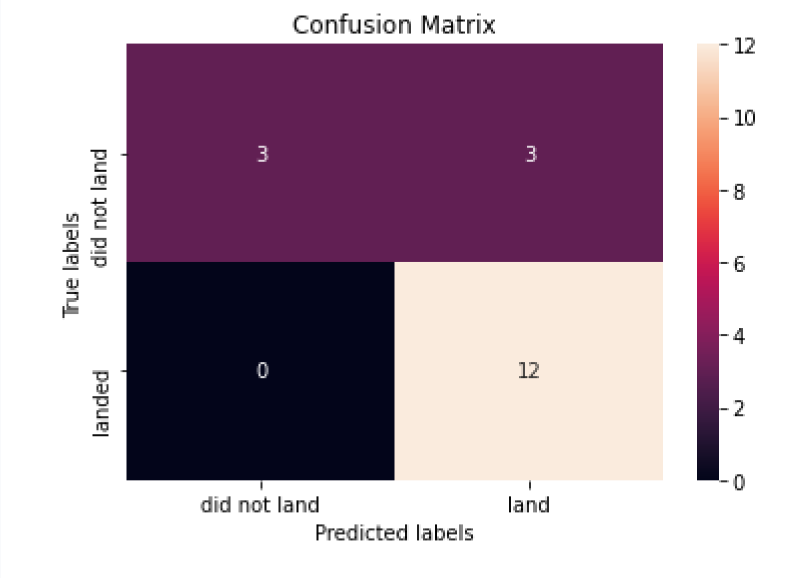


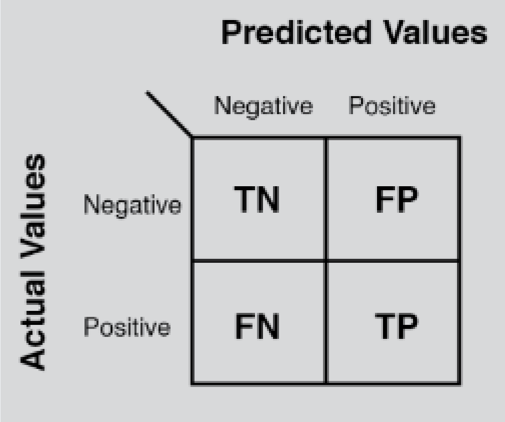
```
Best Algorithm is Tree with a score of 0.8892857142857142
Best Params is : {'criterion': 'gini', 'max_depth': 14, 'max_features': 'sqrt', 'min_samples_leaf': 4, 'min_samples_split': 10, 'splitter': 'best'}
```

After selecting the best hyperparameter for the decision tree classifier using the training data, we achieved a 88.93% accuracy on the test data.

# Confusion Matrix for the Tree model

By looking at the confusion matrix we see that the Tree model can distinguish between the different classes. We see that the problem is with false positives.





| | | Predicted Values | |
|---|---|---|---|
| | | Negative | Positive |
| Actual Values | Negative | TN | FP |
| | Positive | FN | TP |

# Conclusions

- The Tree classifier model is the best prediction model for this data set.
- Low weighted payloads perform better than heavier payloads.
- The success rate for Space X launches gets better as the years progress.
- KSC LC-39A had the most successful launches from all the sites.
- Orbit GEO, HEO, SSO, ES-L1 have the best success rate per the launch data.

# Appendix

- GitHub Repository of all Jupyter NBs used in the presentation.
- SQLite was used instead of IBM DB for the EDA with SQL phase of the project.
- Azure Data Studio was used for the Dasboard with Plotly Dash phase of the project. The code was ran locally and then accessed through a local browser to get the screenshots.

## Thank you!