
# Winning Space Race with Data Science

Nnanke Williams

9th November 2022

# Outline

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

# Executive Summary

- Summary of methodologies
  - Data Collection using SpaceX API
  - Data Collection via Web Scraping from Wikipedia
  - Data Wrangling
  - Exploratory Data Analysis using SQL
  - Exploratory Data Analysis with Visualization using Python
  - Interactive Visual Analytics with Folium
  - Machine Learning Prediction with Python
- Summary of all results
  - Findings from Exploratory Data Analysis
  - Findings from Interactive Analytics
  - Findings from Predictive Analytics

# Introduction

## Project background and context

- This project intends to determine whether the first stage of a SpaceX Falcon 9 rocket launch will land successfully.
- Falcon 9 rocket launches are much cheaper than their counterparts because of the savings made from reusing the first stage. Therefore, if we predict the successful landing of the first stage, we can determine the cost of the launch.
- This information can be used if an alternate company wants to bid against SpaceX for a rocket launch

## Problems you want to find answers

- We will determine the possibility of success of the first stage of a rocket launch in order to determine the cost of the launch.
- We will find which features determine a successful landing.
- We will select the machine learning model with the best prediction capability given the question.

Section 1

# Methodology

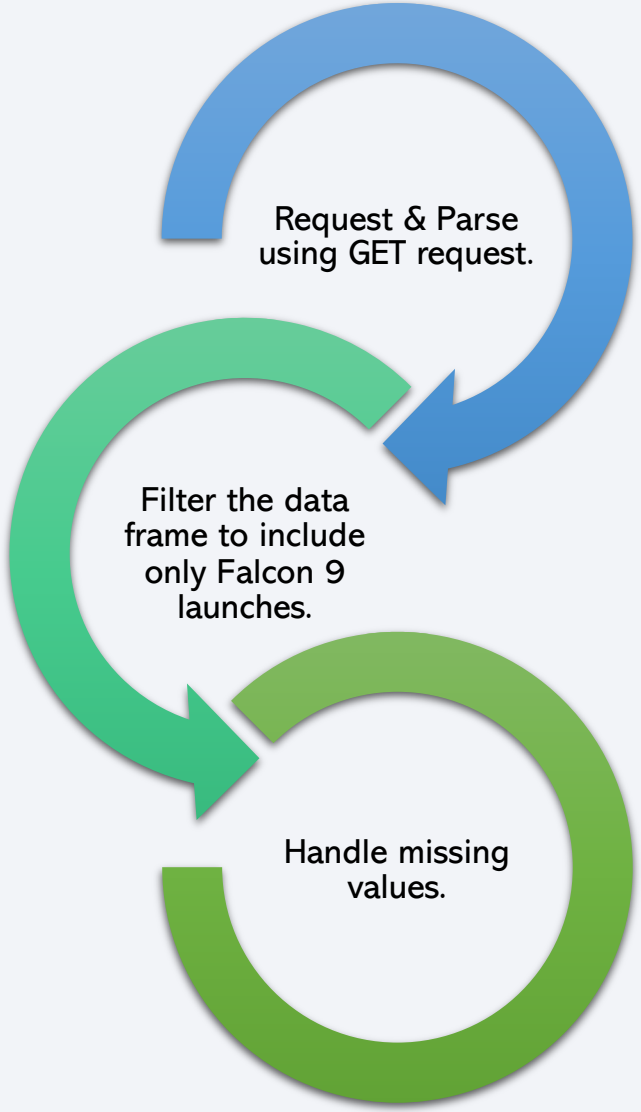
# Methodology

## Executive Summary

- Data collection methodology:
  - Describe how data was collected
- Perform data wrangling
  - Describe how data was processed
- 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

- The dataset was collected using SpaceX Rest API & web scraping from Wikipedia.
- After collection, the data was parsed, analyzed and converted into a data frame for cleaning and further analysis.



# Data Collection – SpaceX API

- We used the get request to the SpaceX API to collect data, clean the requested data and conducted feature selection.

- GitHub URL – SpaceX API Data Collection Notebook

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

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

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

```
# 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 payloa
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)]
```

```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = data[data['BoosterVersion'] != 'Falcon 1']
print(data_falcon9.shape)
data_falcon9.head()
```

```
# Calculate the mean value of PayloadMass column
pm_mean = data_falcon9['PayloadMass'].mean()

# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'].replace(np.nan, pm_mean, inplace=True)
data_falcon9.isnull().sum()
```

# Data Collection – SpaceX API – *Code Cells*

*1. Make GET Request on SpaceX API*

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

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

*2. Normalize the GET request json result into a data frame*

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

*3. Feature Selection & Cleaning*

```
# 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 payloa
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)]
```

*4. Filter data frame to only Falcon 9 launches*

```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = data[data['BoosterVersion'] != 'Falcon 1']
print(data_falcon9.shape)
data_falcon9.head()
```

*5. Replace missing values with Mean*

```
# Calculate the mean value of PayloadMass column
pm_mean = data_falcon9['PayloadMass'].mean()

# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'].replace(np.nan, pm_mean, inplace=True)
data_falcon9.isnull().sum()
```

# Data Collection - Scraping

- We scraped the Wikipedia page '*List of Falcon 9 and Falcon Heavy Launches*' using a GET request.
- We parsed the data using BeautifulSoup() and selected the relevant table using the find_all() method.
- We iterated through the soup object to retrieve all rows into a dictionary and converted the dictionary to a data frame.
- GitHub URL – Data Collection with Web Scraping Notebook

```
# use requests.get() method with the provided static_url
# assign the response to a object
url_get = requests.get(static_url)

# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(url_get.text, 'html.parser')

# Use the find_all function in the BeautifulSoup object, with element type `table`
# Assign the result to a list called `html_tables`
html_tables = soup.find_all('table')
```

```
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders collapsible")):
   # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number corresponding to launch a number
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
        else:
            flag=False
        #get table element
        row=rows.find_all('td')
        #if it is number save cells in a dictonary
        if flag:
            extracted_row += 1
            # Flight Number value
            # TODO: Append the flight_number into launch_dict with key `Flight No.`
            datatimelist=date_time(row[0])
            launch_dict['Flight No.'].append(flight_number)
            print(flight_number)

            # Date value
            # TODO: Append the date into launch_dict with key `Date`
            date = datatimelist[0].strip(',')
            launch_dict['Date'].append(date)
            print(date)

            # Time value
            # TODO: Append the time into launch_dict with key `Time`
            time = datatimelist[1]
            launch_dict['Time'].append(time)
            print(time)
```

```
df=pd.DataFrame(launch_dict)
df.head()
```

# Data Collection – Web Scraping – *Code Cells*

*1. Retrieve web page using GET request and parse page into a BeautifulSoup object.*

```
# use requests.get() method with the provided static_url
# assign the response to a object
url_get = requests.get(static_url)

# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(url_get.text, 'html.parser')
```

*2. Select the relevant table using the find_all() method*

```
# Use the find_all function in the BeautifulSoup object, with element type `table`
# Assign the result to a list called `html_tables`
html_tables = soup.find_all('table')

# Let's print the third table and check its content
first_launch_table = html_tables[2]
```

*3. Retrieve Column Headers from table object & Make Dictionary for table*

```
column_names = []

# Apply find_all() function with `th` element on first_launch_table
# Iterate each th element and apply the provided extract_column_from_header() to get a column name
# Append the Non-empty column name (`if name is not None and len(name) > 0`) into a list called column_names
for element in first_launch_table.find_all('th'):
    name = extract_column_from_header(element)
    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 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']=[]
```

*4. Extract rows from table object and append to dictionary*

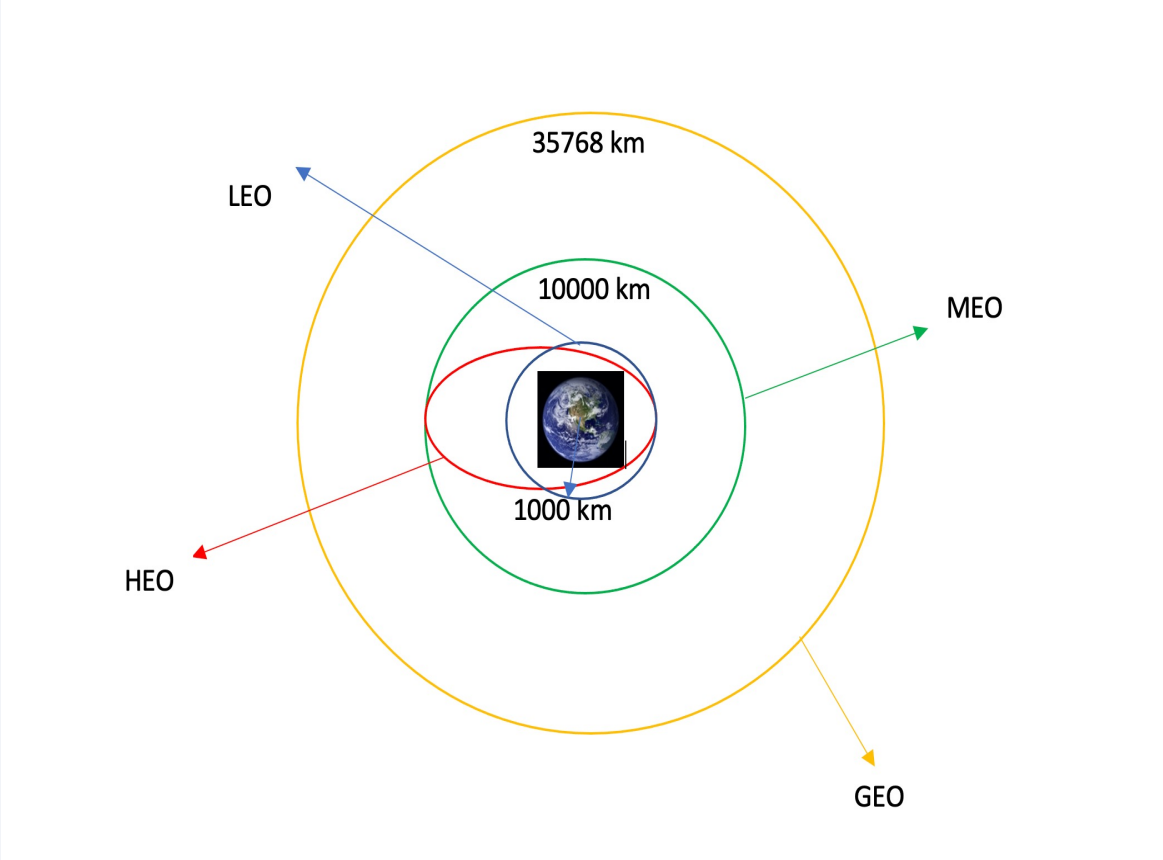
```
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders collapsible")):
   # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number corresponding to launch a number
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
        else:
            flag=False
        #get table element
        row=rows.find_all('td')
        #if it is number save cells in a dictonary
        if flag:
            extracted_row += 1
            # Flight Number value
            # TODO: Append the flight_number into launch_dict with key `Flight No.`
            datatimelist=date_time(row[0])
            launch_dict['Flight No.'].append(flight_number)
            print(flight_number)

            # Date value
            # TODO: Append the date into launch_dict with key `Date`
            date = datatimelist[0].strip(',')
            launch_dict['Date'].append(date)
            print(date)

            # Time value
            # TODO: Append the time into launch_dict with key `Time`
            time = datatimelist[1]
            launch_dict['Time'].append(time)
            print(time)
```

*5. Convert dictionary to data frame*

```
df=pd.DataFrame(launch_dict)
df.head()
```

# Data Wrangling

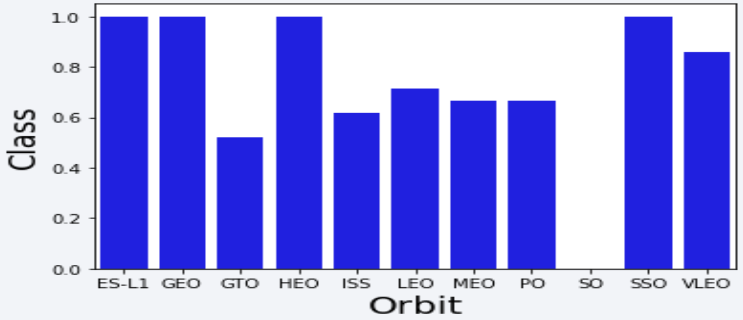
- We performed some data wrangling in order to convert launch outcomes into dummy training labels
- 1 means the booster landed successfully and 0 means it was unsuccessful.
- GitHub URL – EDA Data Wrangling Notebook



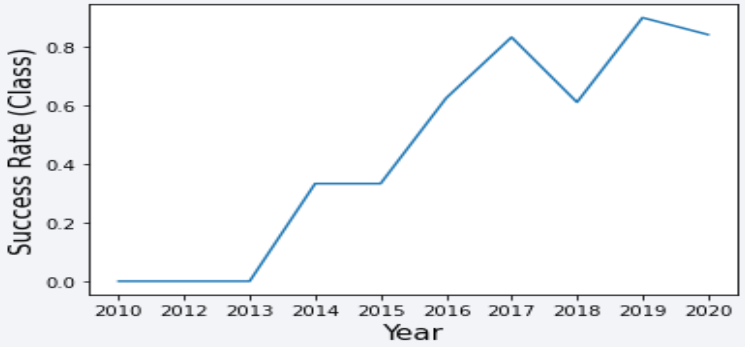
# EDA with Data Visualization

- We visualized the relationship between a few variables in the dataset in order to determine:
  - if any correlation exists between variables and
  - what strategy was employed by SpaceX in their Falcon 9 launches.
- We converted categorical features into dummy variables and converted all records into float data type.
- GitHub URL – EDA with Data Visualization Notebook

- We used a bar chart to visualize the average success rate per Orbit

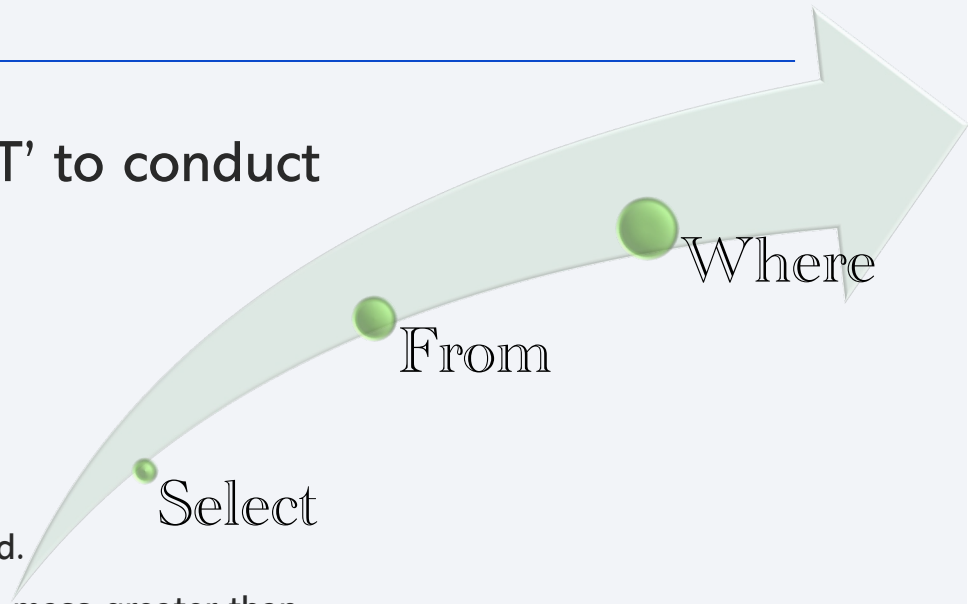



- We used a line chart to visualize the average success rate per year.



# EDA with SQL

- We used the SQL data manipulation statement 'SELECT' to conduct exploratory data analysis and determine:
  - The list of the names of the unique launch sites in the space mission.
  - The list of the 5 records where launch sites begin with the string 'CCA'.
  - The total payload mass carried by boosters launched by NASA (CRS).
  - The average payload mass carried by booster version F9 v1.1.
  - The date when the first successful landing outcome in ground pad was achieved.
  - The names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000.
  - The total number of successful and failure mission outcomes.
  - The names of the booster versions which have carried the maximum payload mass using a subquery.
  - A list of the failed landing outcomes in drone ship for the year 2015, along with their booster versions, and launch site names.
  - The number of landing outcomes between 4th June 2010 and 20th March 2017 ranked in descending order.




- GitHub URL – EDA with SQL Notebook

# Build an Interactive Map with Folium

- Using an Interactive Folium Map:
  - We marked all launch sites, and added map objects such as markers, circles, lines to mark the success or failure of launches for each site.
  - Using the color-labeled marker clusters, we identified which launch sites have relatively high success rate.
  - We calculated the distances between a launch site and some nearby coordinates.

| | Nearest Coordinates | Latitude | Longitude | Distance |
|---|---|---|---|---|
| 0 | Railway | 28.57374 | -80.58523 | 1.475210 |
| 1 | Samuel C. Phillips Pkwy | 28.56274 | -80.57068 | 0.655569 |
| 2 | Atlantic Coastline | 28.56270 | -80.56712 | 1.002846 |
| 3 | Melbourne City | 28.10711 | -80.62866 | 50.891185 |
| 4 | Titusville City | 28.61286 | -80.81200 | 23.592270 |

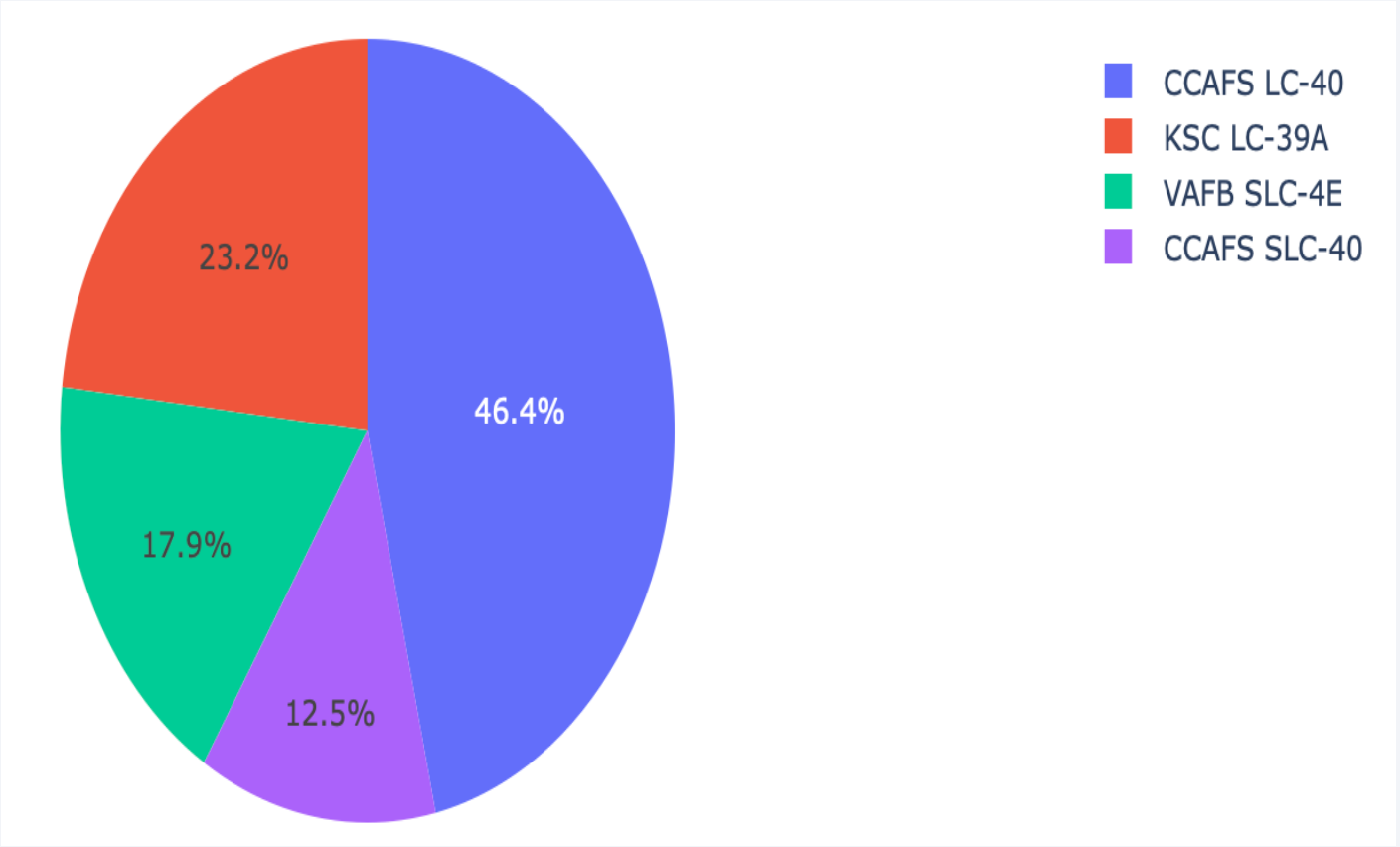
- GitHub URL – Interactive Visual Analytics with Folium Map Notebook

# Build a Dashboard with Plotly Dash

- We built an interactive dashboard with Plotly dash.
- We plotted pie charts showing the total launches by selected sites.
- We plotted a scatter chart showing the relationship between Outcome and Payload Mass for different booster versions.

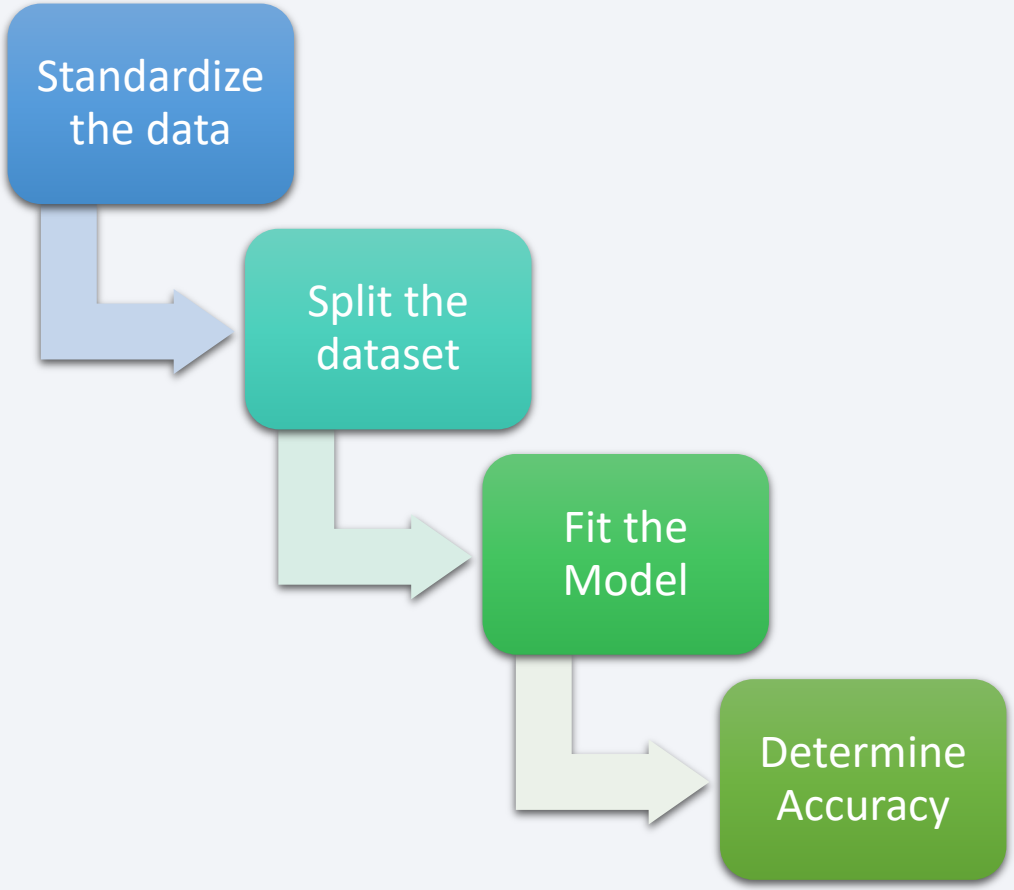
- GitHub URL – SpaceX Dash App Script

*All Launch Site Successes*



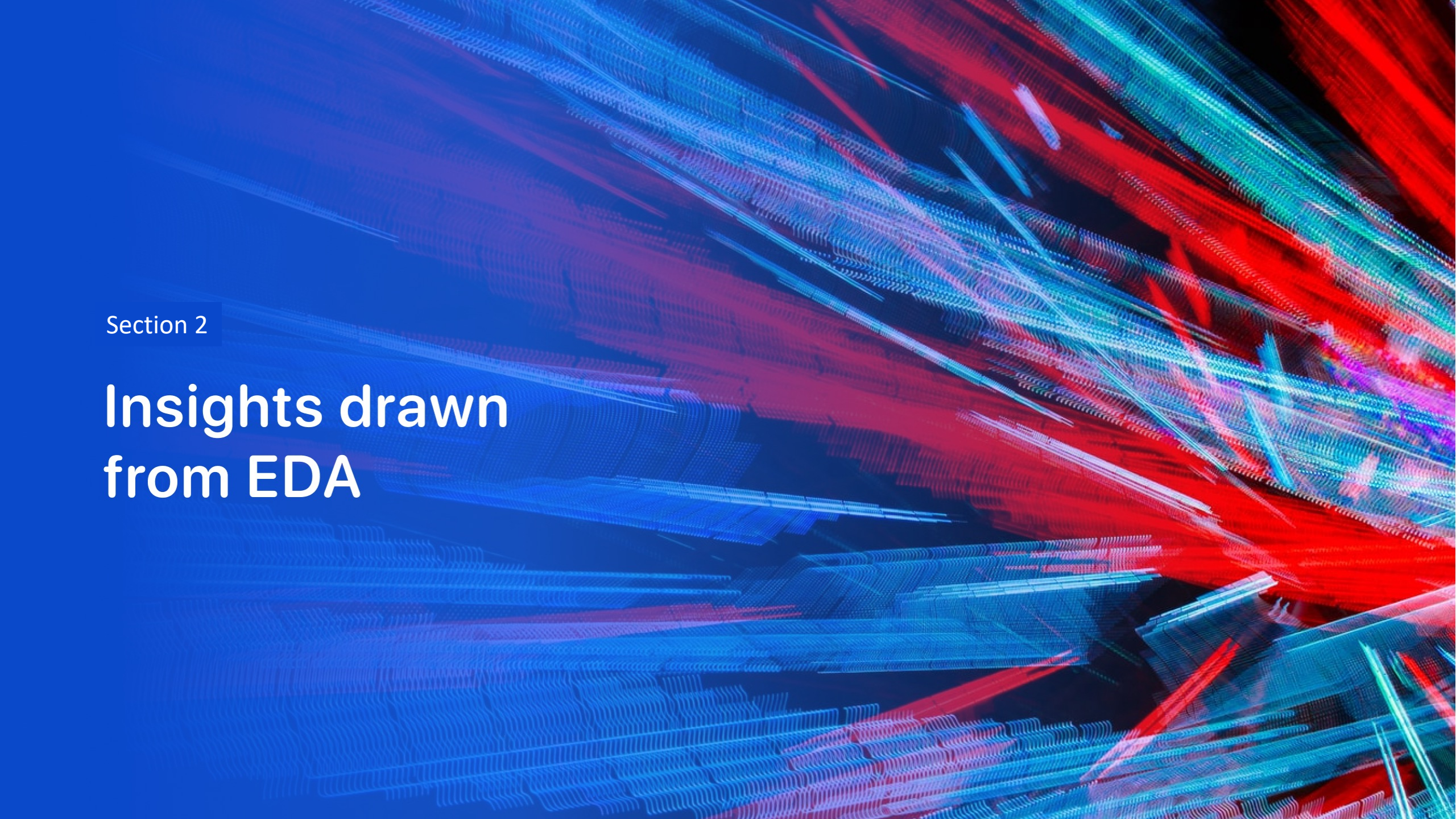
# Predictive Analysis (Classification)

- We conducted predictive analysis using the following process:
  - We loaded the data into a data frame, standardized the records, and split the dataset into train and test sets for predictive analysis.
  - We built different machine learning models and tuned different hyperparameters using GridSearchCV.
  - We fit the models using the training data and tested them using the test data.
  - We used the .score() method to determine how accurate the models were in predicting successful launches
  - We determined the best performing classification model.
- GitHub URL – Machine Learning Prediction Notebook



# Results

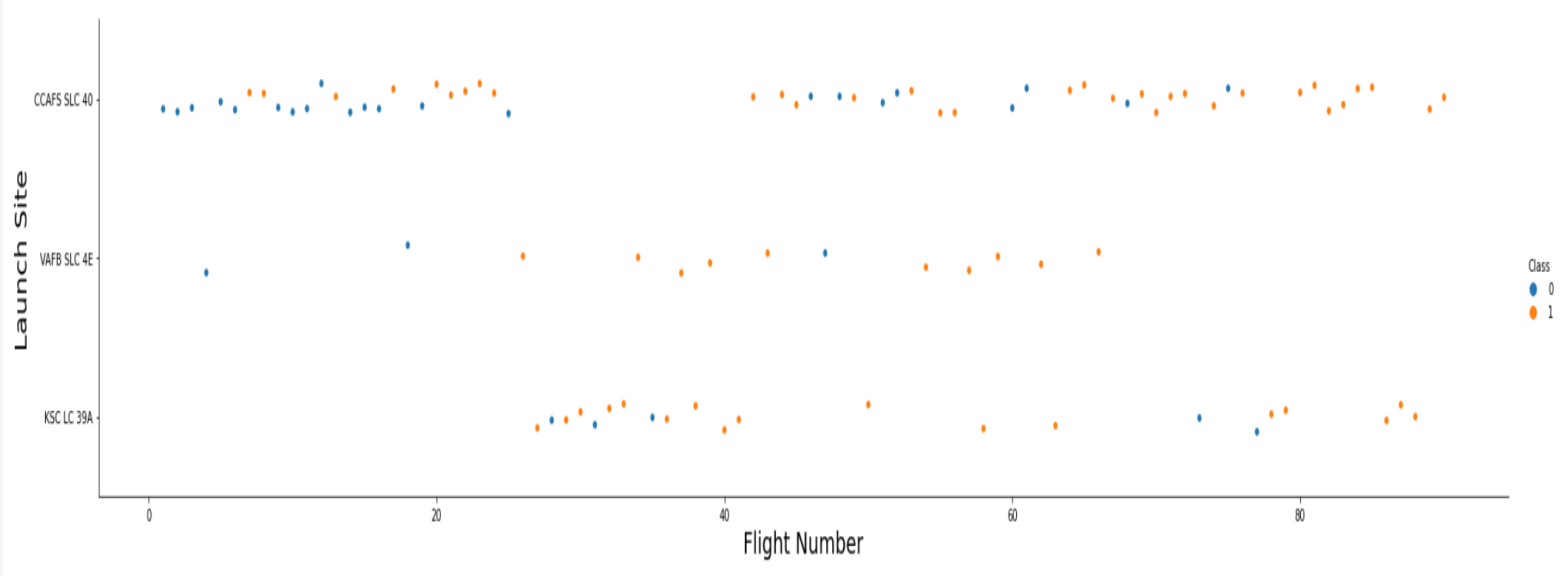
- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results


Section 2
Insights drawn from EDA

# Flight Number vs. Launch Site

- We see that the initial strategy put emphasis on the first site - 'CCAFS SLC 40' and saw more losses than successes.
- Then the second 2 sites were added with less focus on the first site for the launch attempts between 20 and 40 which recorded more successes than the initial 20 attempts.
- Emphasis returned to the 1st site after this detour and the success rate increased.
- Overall, the other sites recorded more successes than the first.

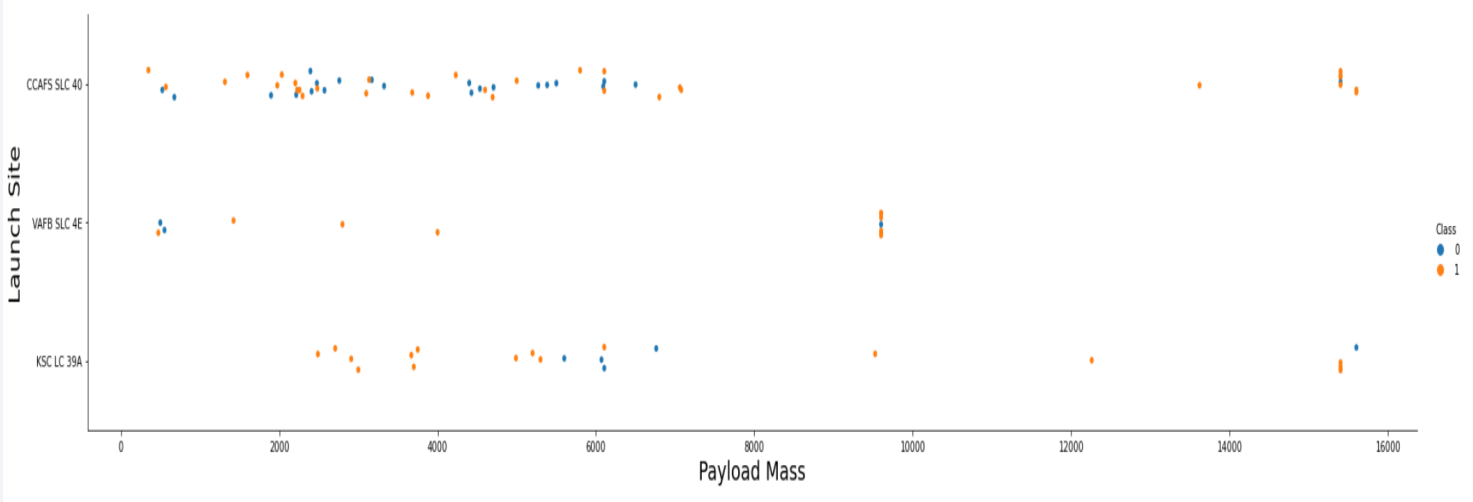
*Labels: Orange (Success), Blue (Failure)*



# Payload vs. Launch Site

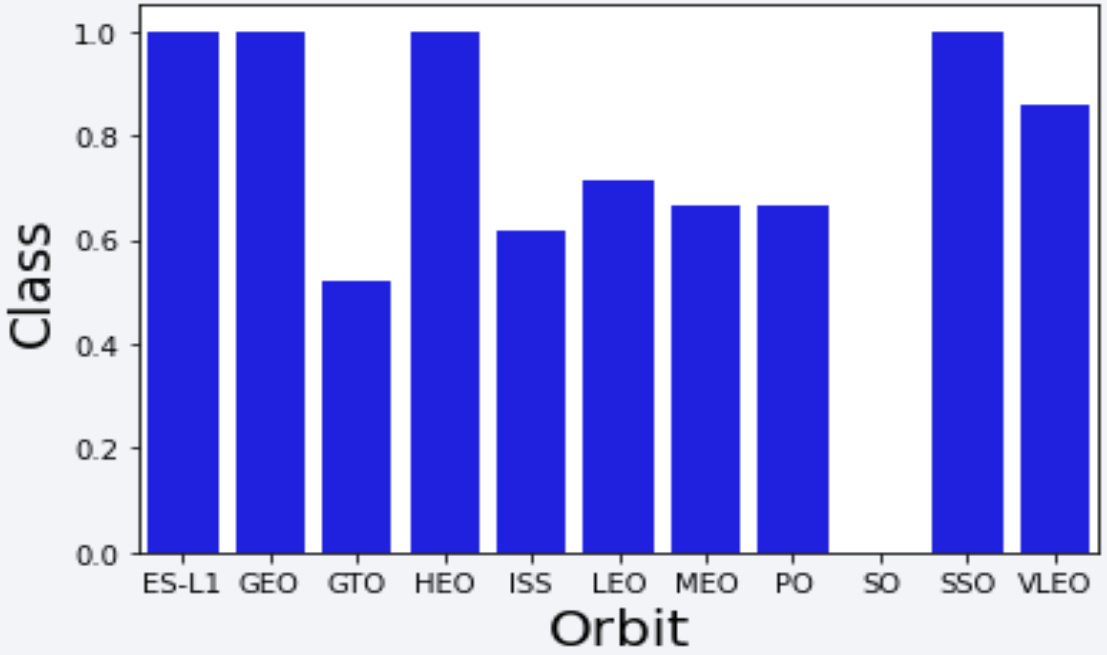
- SpaceX favours a lighter payload mass for its launches
- Heavier payloads are more successful.
- For the VAFB-SLC launch site there are no rockets launched for payload masses greater than 10000.

*Labels: Orange (Success), Blue (Failure)*



# Success Rate vs. Orbit Type

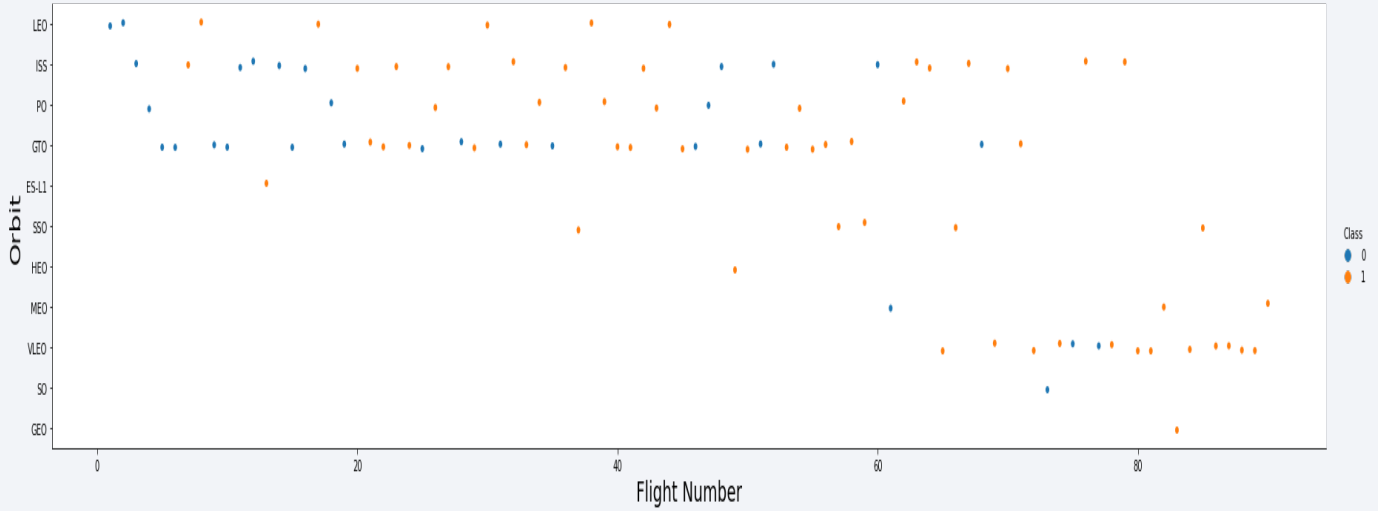
- The highest success rate is recorded in orbits ES-L1, GEO, HEO, and SSO. However, each of these orbits only record one attempt except SSO which has 5 attempts all of which are successful.
- The second most successful orbit is VLEO with 14 attempts and over 80% success rate
- Other orbits had successes averaging between the 40-60% range.
- There were no successes from the SO orbit.



# Flight Number vs. Orbit Type

- In the LEO orbit, success appears related to the number of flights.
- There seems to be no relationship between flight number when in GTO orbit.

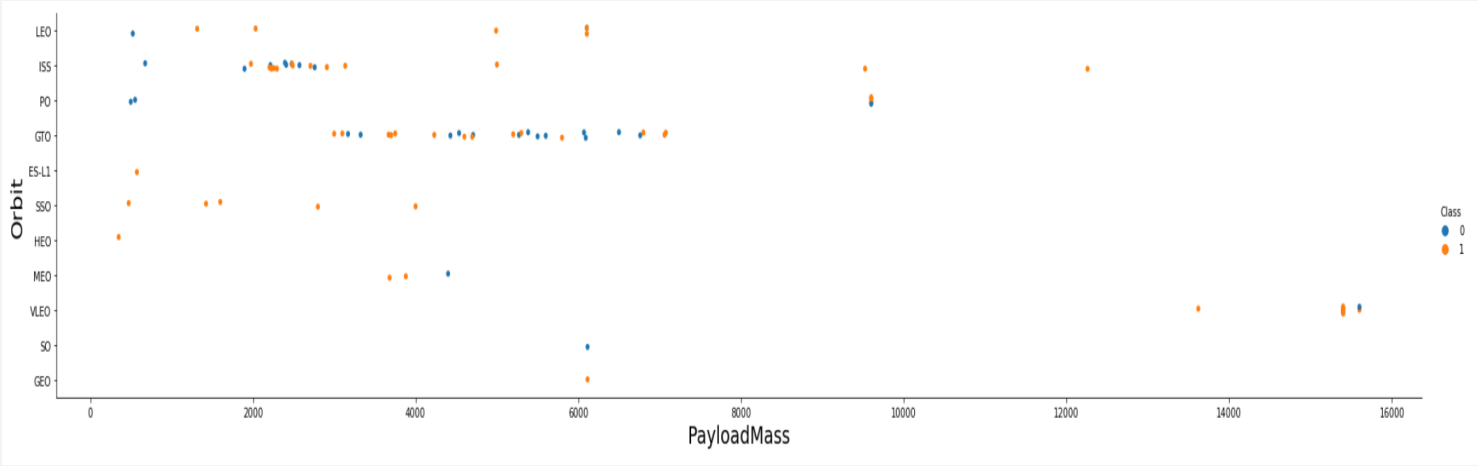
*Labels: Orange (Success), Blue (Failure)*



# Payload vs. Orbit Type

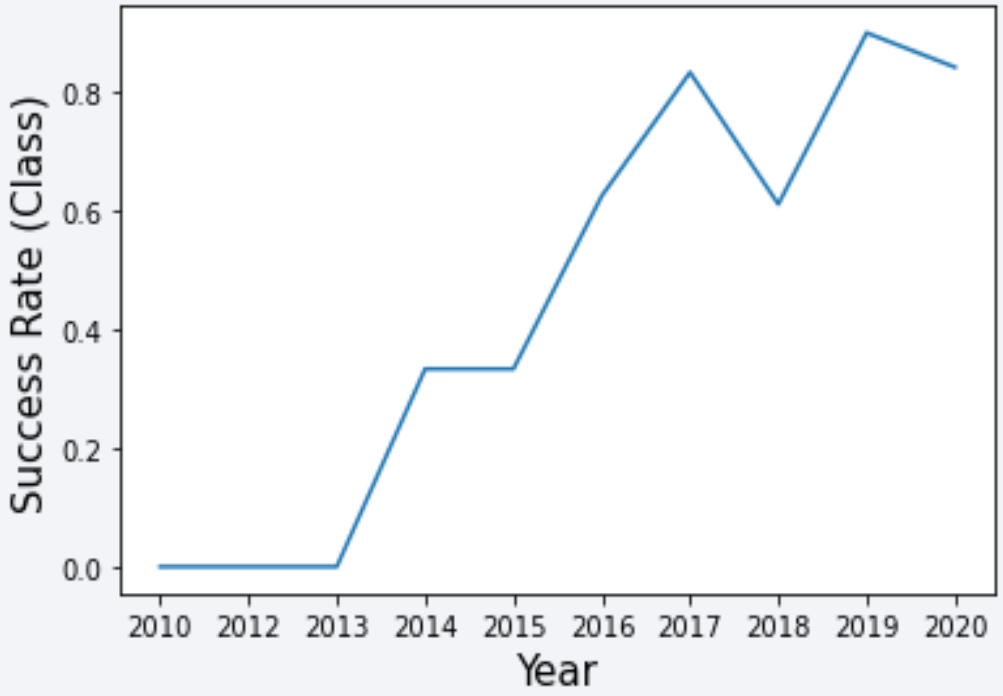
- The orbits PO, LEO and ISS have higher success rates with heavy payloads.
- There is no clear correlation between success rate and payload for the GTO orbit.

*Labels: Orange (Success), Blue (Failure)*



# Launch Success Yearly Trend

- There is a steady increase in success rate for launches from 2013.
- In the final year recorded in the dataset (2020) we see a slight dip in success rate.



# All Launch Site Names

- Finding the names of the unique launch sites
  - We used the select statement and the distinct clause to find the unique launch sites in the SpaceX table in SQL.

```
%sql select distinct Launch_Site from spacextbl;
```

```
 * ibm_db_sa://snn30638:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08
Done.
```

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

# Launch Site Names Begin with 'CCA'

- Finding 5 records where launch sites begin with `CCA`
  - We filtered the records in the table using the 'WHERE' and the 'LIKE' clause to produce records for launch sites beginning with 'CCA'.
  - We used the 'LIMIT' clause to restrict the number of results to 5 records.

```
%sql select *\
    from spacextbl\
    where launch_site like 'CCA%'\
    limit 5;
```

```
 * ibm_db_sa://snn30638:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu01qde00.databases.appdomain.cloud:31321/bludb
Done.
```

| DATE | time__utc_ | booster_version | launch_site | payload | payload_mass__kg_ | orbit | customer | mission_outcome | landing__outcome |
|---|---|---|---|---|---|---|---|---|---|
| 2010-04-06 | 18:45:00 | F9 v1.0 B0003 | CCAFS LC-40 | Dragon Spacecraft Qualification Unit | 0 | LEO | SpaceX | Success | Failure (parachute) |
| 2010-08-12 | 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 | 07:44:00 | F9 v1.0 B0005 | CCAFS LC-40 | Dragon demo flight C2 | 525 | LEO (ISS) | NASA (COTS) | Success | No attempt |
| 2012-08-10 | 00:35:00 | F9 v1.0 B0006 | CCAFS LC-40 | SpaceX CRS-1 | 500 | LEO (ISS) | NASA (CRS) | Success | No attempt |
| 2013-01-03 | 15:10:00 | F9 v1.0 B0007 | CCAFS LC-40 | SpaceX CRS-2 | 677 | LEO (ISS) | NASA (CRS) | Success | No attempt |

# Total Payload Mass

- Calculating the total payload carried by boosters from NASA
  - We used the SUM() function to determine the total payload and filtered results to produce only records where the customer was 'NASA (CRS)'.
  - Because we used an aggregate function (sum), we had to group the result by the unaggregated column (customer) using the GROUP BY() function.

```
%sql select sum(PAYLOAD_MASS__KG_) as total_payload_mass_KG, customer\
    from SPACEXTBL\
    where customer = 'NASA (CRS)'\
    group by customer;
```

```
 * ibm_db_sa://snn30638:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3s
Done.
```

| total_payload_mass_kg | customer |
|---|---|
| 45596 | NASA (CRS) |

# Average Payload Mass by F9 v1.1

- Calculating the average payload mass carried by booster version F9 v1.1
  - We used the AVG() function to determine the average payload and filtered results to produce only records where the booster version was ‘F9 v1.1’.
  - Because we used an aggregate function (average), we had to group the result by the unaggregated column (booster version) using the GROUP BY() function.

```
%sql select avg(PAYLOAD_MASS__KG_) as Avg_Payload_Mass_KG, booster_version\
    from SPACEXTBL\
    where booster_version = 'F9 v1.1'\
    group by booster_version;
```

```
 * ibm_db_sa://snn30638:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgt
Done.
```

| avg_payload_mass_kg | booster_version |
|---|---|
| 2928 | F9 v1.1 |

# First Successful Ground Landing Date

- Finding the dates of the first successful landing outcome on ground pad
  - We used the MIN() function to determine the earliest date and filtered results to produce only records where the landing outcome contained the phrase ‘ground pad’.
  - Because we used an aggregate function (min), we had to group the result by the unaggregated column (landing outcome) using the GROUP BY() function.

```
%sql select min(DATE) as Earliest_Date, landing__outcome\
    from SPACEXTBL\
    where landing__outcome like '%ground pad%'\
    group by landing__outcome;
```

```
 * ibm_db_sa://snn30638:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgt
Done.
```

| earliest_date | landing__outcome |
|---|---|
| 2015-12-22 | Success (ground pad) |

# Successful Drone Ship Landing with Payload between 4000 and 6000

- Finding the names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000
  - We selected the booster version, landing outcome and payload mass columns from the table and filtered the results to produce only records where the landing outcome was ‘Success (drone ship)’, and the payload weight was between 4000 and 6000.

```
%sql select booster_version, landing__outcome, PAYLOAD_MASS__KG_\
    from spacextbl\
    where landing__outcome = 'Success (drone ship)'\
    and PAYLOAD_MASS__KG_ between 4000 and 6000;
```

```
 * ibm_db_sa://snn30638:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgt
Done.
```

| booster_version | landing__outcome | payload_mass__kg_ |
|---|---|---|
| F9 FT B1022 | Success (drone ship) | 4696 |
| F9 FT B1026 | Success (drone ship) | 4600 |
| F9 FT B1021.2 | Success (drone ship) | 5300 |
| F9 FT B1031.2 | Success (drone ship) | 5200 |

# Total Number of Successful and Failure Mission Outcomes

- Calculating the total number of successful and failure mission outcomes
  - We used the COUNT() function to determine the total number of mission outcomes and used the GROUP BY() function to group the results by mission outcome designations.

```
%sql select mission_outcome, count(mission_outcome) as total\
    from spacextbl\
    group by mission_outcome;
```

```
 * ibm_db_sa://snn30638:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgt
Done.
```

| mission_outcome | total |
|---|---|
| Failure (in flight) | 1 |
| Success | 99 |
| Success (payload status unclear) | 1 |

# Boosters Carried Maximum Payload

- Finding the names of the booster which have carried the maximum payload mass
  - We used a subquery in the WHERE clause to highlight only booster versions where the payload weight was the the maximum payload weight.

```
%sql select booster_version, PAYLOAD_MASS__KG_\
     from spacextbl\
     where PAYLOAD_MASS__KG_ = (select max(PAYLOAD_MASS__KG_) from spacextbl);
```

```
 * ibm_db_sa://snn30638:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0l
Done.
```

| 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

- Finding the failed landing outcomes in drone ship, their booster versions, and launch site names for in year 2015
  - We used the MONTH() and YEAR() functions on the Date column to select the month and year.
  - We filtered results to include only failure landing outcomes from the year 2015.

```
%sql select month(Date) as Month, landing__outcome, Booster_Version, Launch_Site\
    from SPACEXTBL\
    where year(Date) = 2015 and  landing__outcome like '%Failure%';
```

```
 * ibm_db_sa://snn30638:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00
Done.
```

| MONTH | landing__outcome | booster_version | launch_site |
|---|---|---|---|
| 10 | Failure (drone ship) | F9 v1.1 B1012 | CCAFS LC-40 |
| 4 | Failure (drone ship) | F9 v1.1 B1015 | CCAFS LC-40 |

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

- Ranking 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
  - We used the COUNT() function to count the number of landing outcomes and grouped the results by landing outcome.
  - We filtered the result to the specified dates.
  - We sorted our result by num of outcomes in descending order using the ORDER BY() function.

```
%sql select landing__outcome, count(landing__outcome) as num_successful_outcomes\
    from SPACEXTBL\
    where landing__outcome = 'Success' or landing__outcome like '%Success%' and Date between '2010-06-04' and '2017-03-20'\
    group by landing__outcome\
    order by num_successful_outcomes desc;
```

```
 * ibm_db_sa://snn30638:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:31321/bludb
Done.
```

| landing__outcome | num_successful_outcomes |
|---|---|
| Success | 38 |
| Success (drone ship) | 5 |
| Success (ground pad) | 5 |

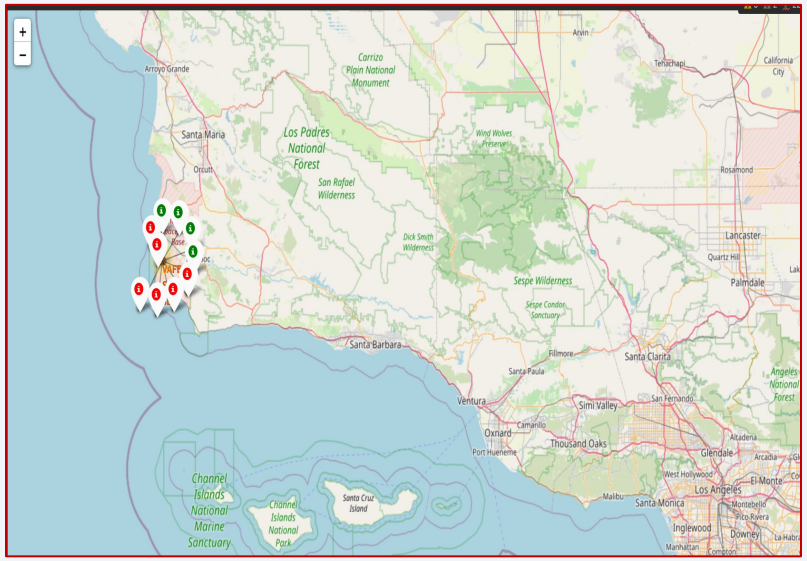

Section 3
Launch Sites
Proximities Analysis

# Launch Site Markers on Folium Map

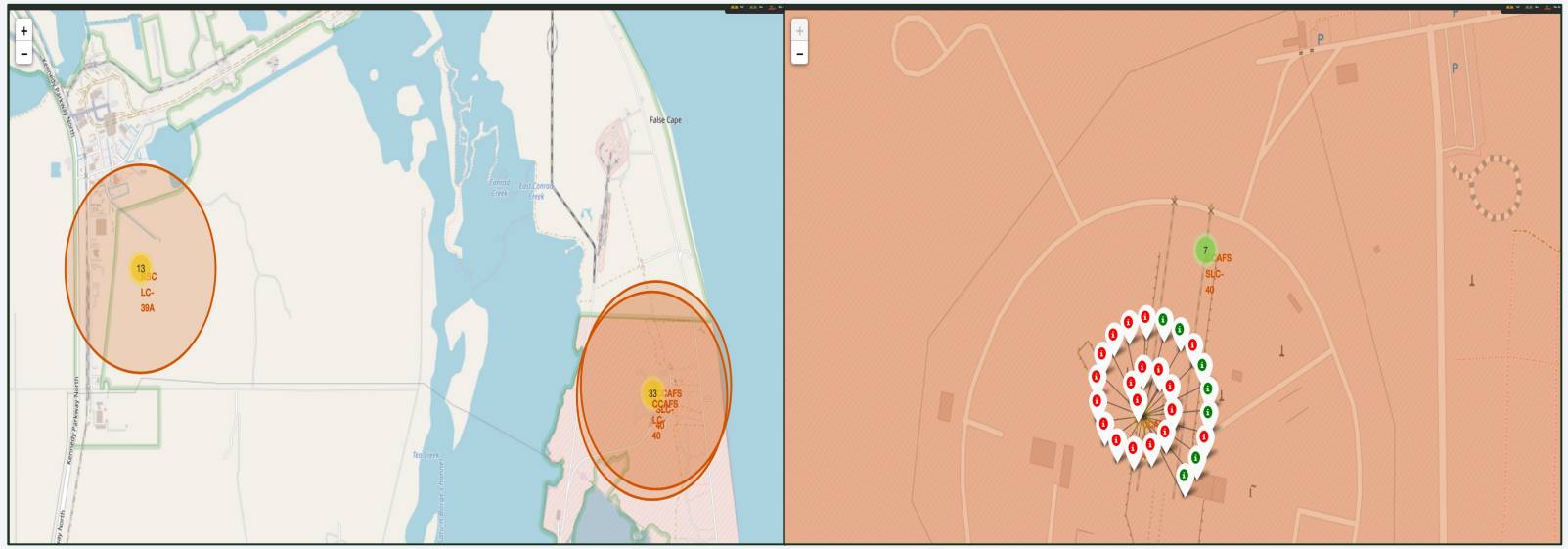
- SpaceX launch sites are located on the coast of the United States in Florida and California

# Markers Showing Launch Sites with Color Labels

*California Site*




*Florida Sites*

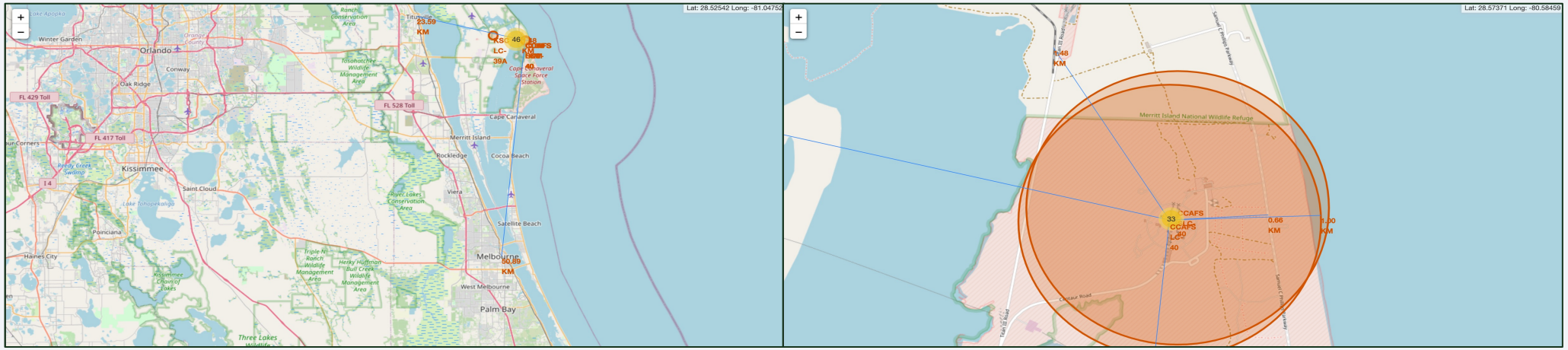



*The Markers designate outcomes*

Green = Success

Red = Failure

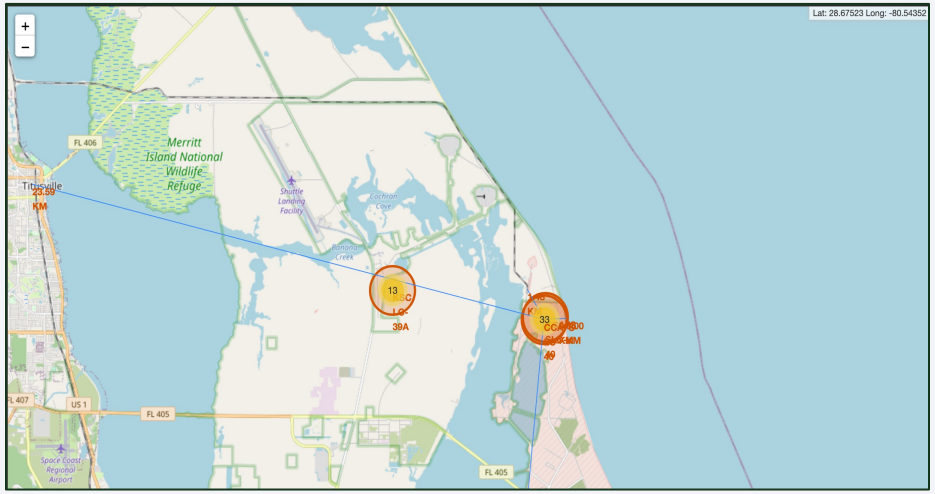
# Distance to Landmarks




In siting launch sites, proximity to the coastline is priority. All sites are very near the coast.

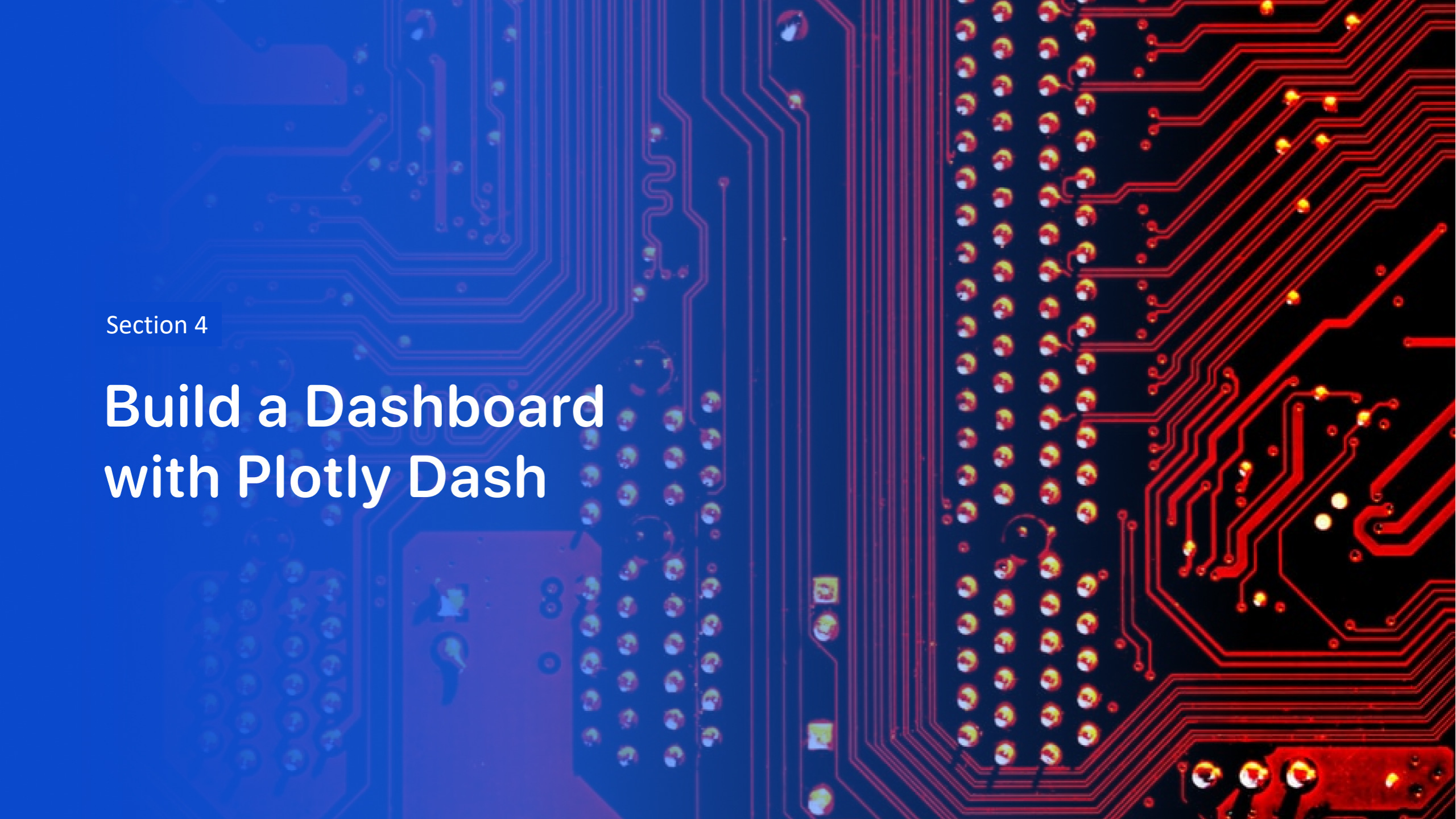
The closest coordinate to the CCAFS launch sites is the highway (Samuel C. Phillips Pkwy) (0.66Km). This may reflect transportation needs, especially of persons.

The next closest coordinate is the Atlantic coastline (1.0km). This is possibly because of SpaceX's landing guidelines which prioritize ocean landing.



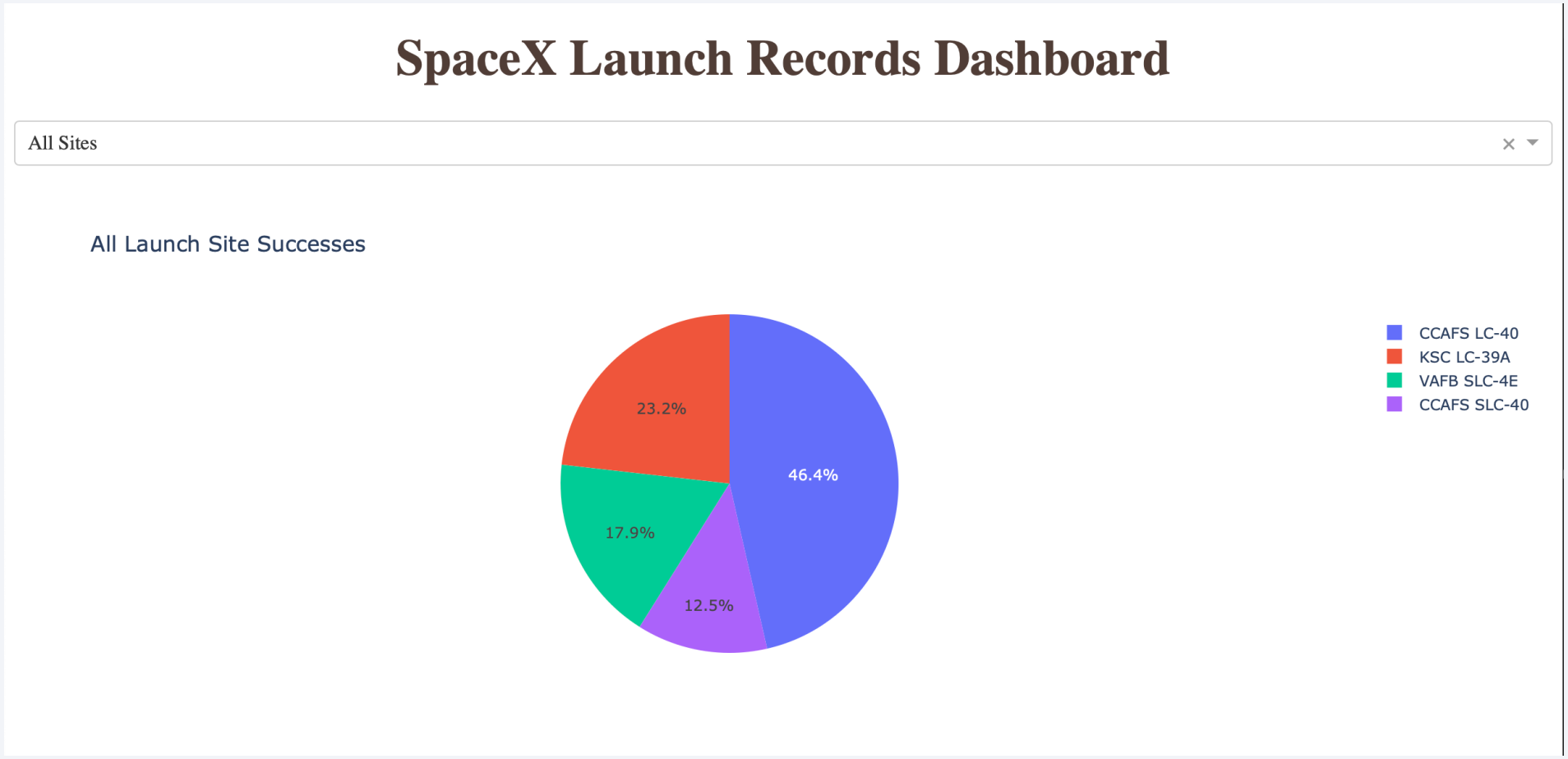

The third nearest distance is between the launch sites and the nearest railway (1.45km). This may reflect the need for quick and easy transport of supplies.

Finally, each launch site is a considerable distance away from city centers (Titusville - 23.59km and Melbourne - 50.89km). Due to the nature of the work being done, it is important to site launches away from cities and places where the population density is high.


Section 4
Build a Dashboard with Plotly Dash

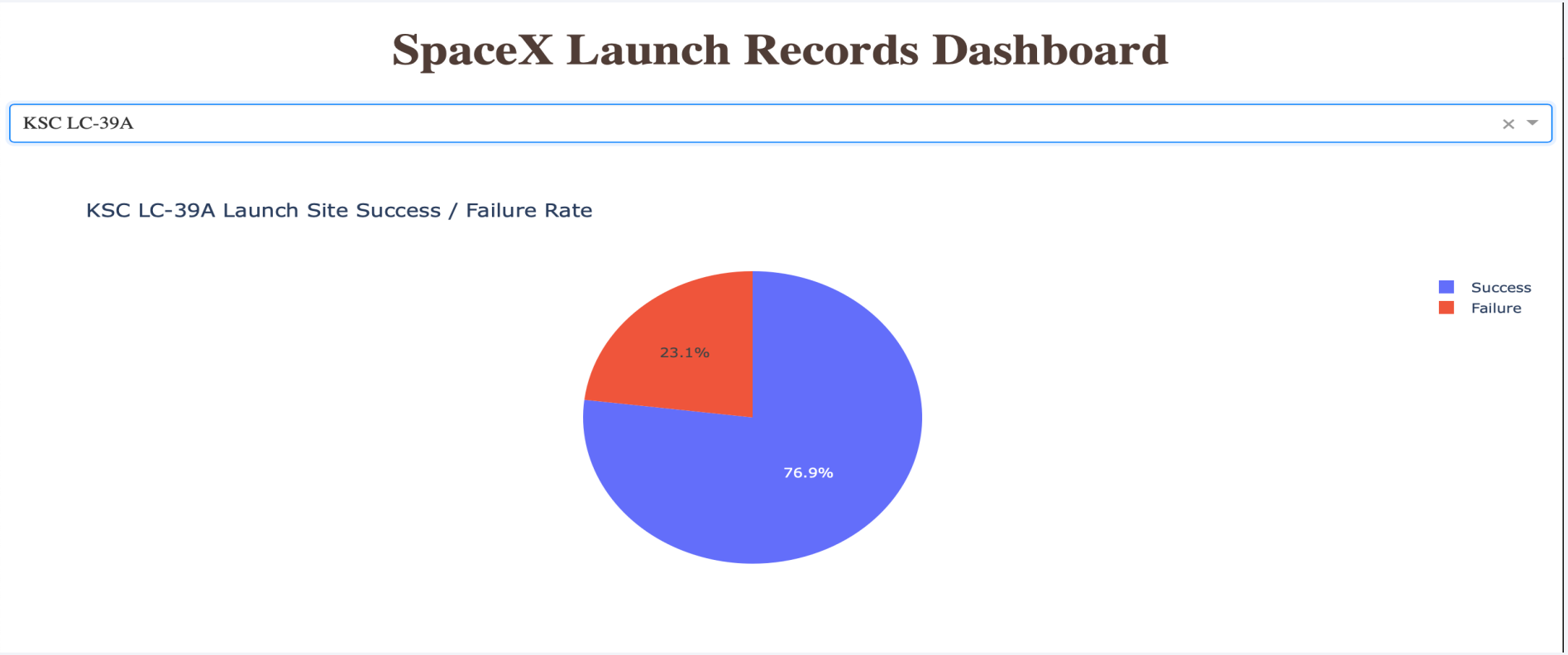
# Pie chart showing success rate for all Sites

- CCAFS LC-40 is the launch site with the most successful launches



# Pie chart showing Site with highest success ratio

- KSC LC-39A is the launch site with the highest success ratio

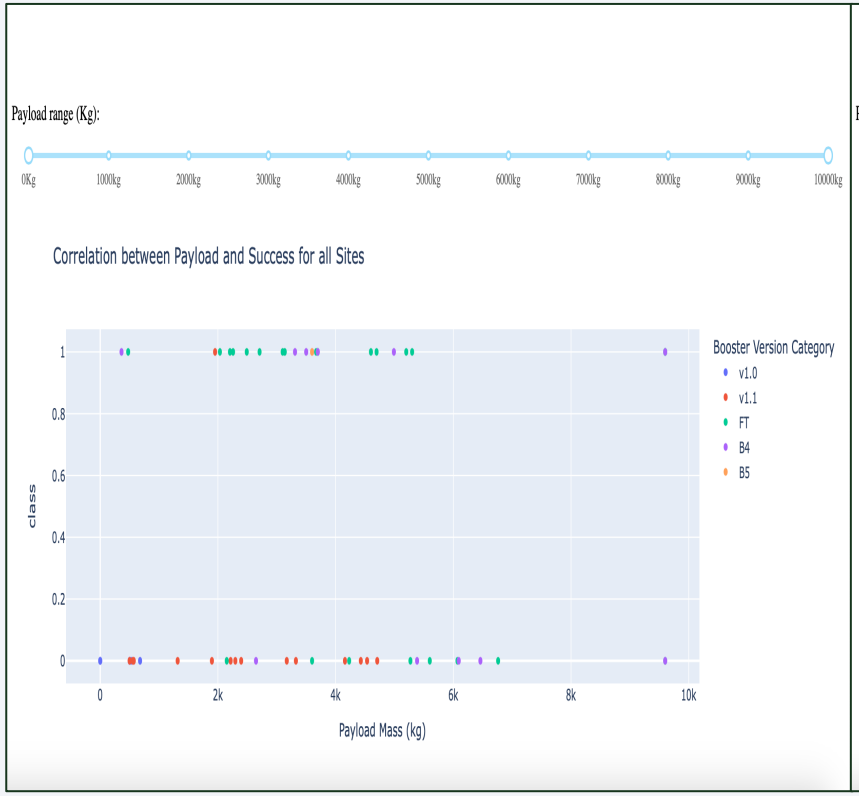


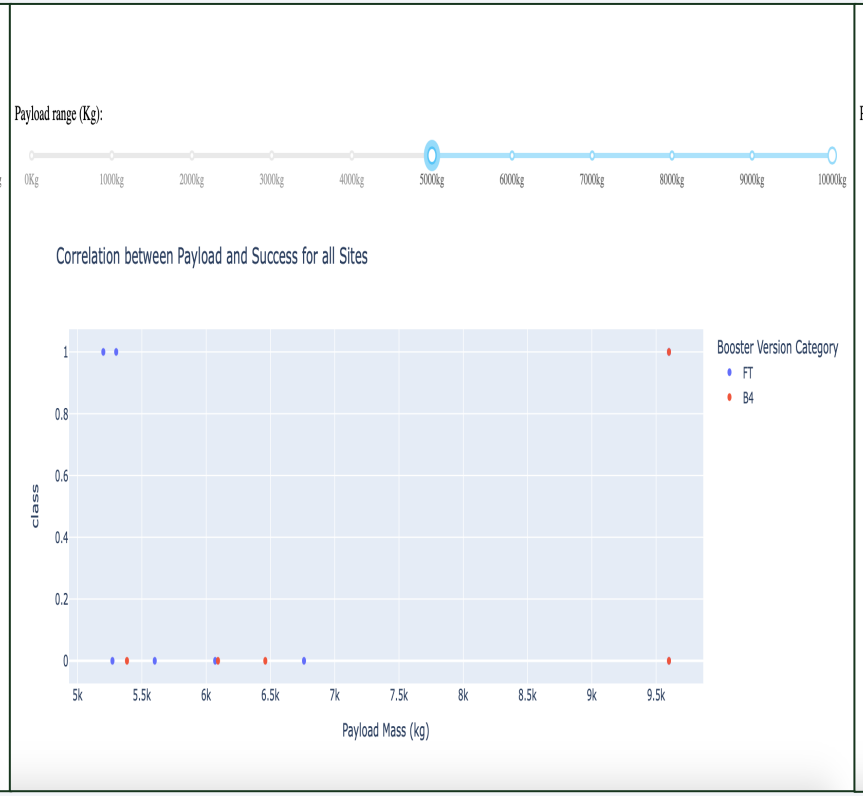
# Scatterplot showing Payload v. Launch Outcome

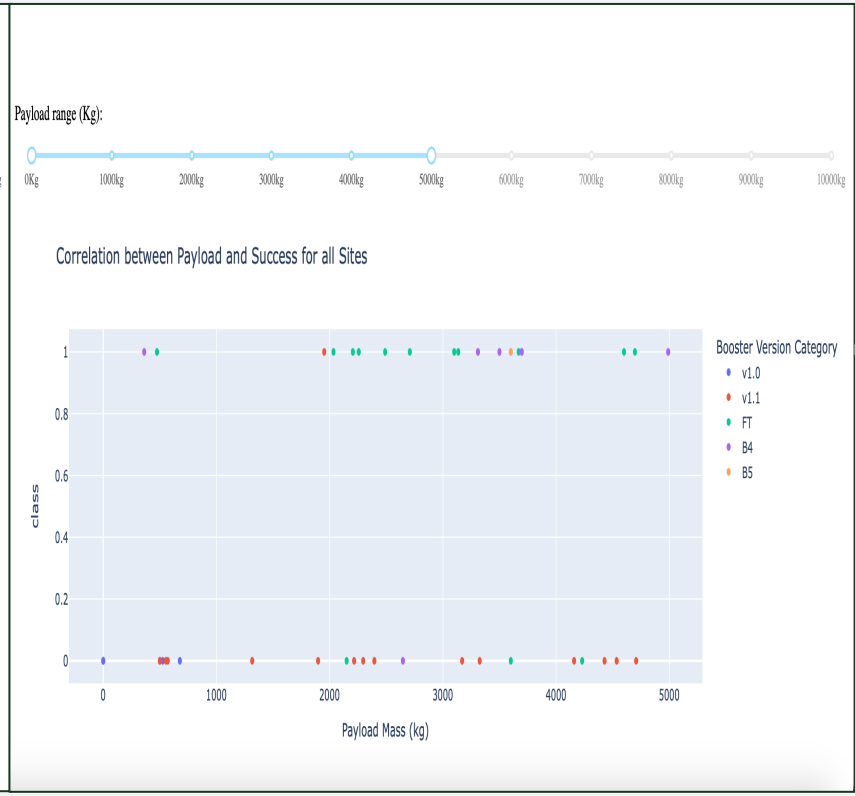
We can see success v. failure ratio based on payload range.

The range 5k – 10k has the lowest launch success rate.

The FT booster has the highest success rate and v1.1 has the lowest



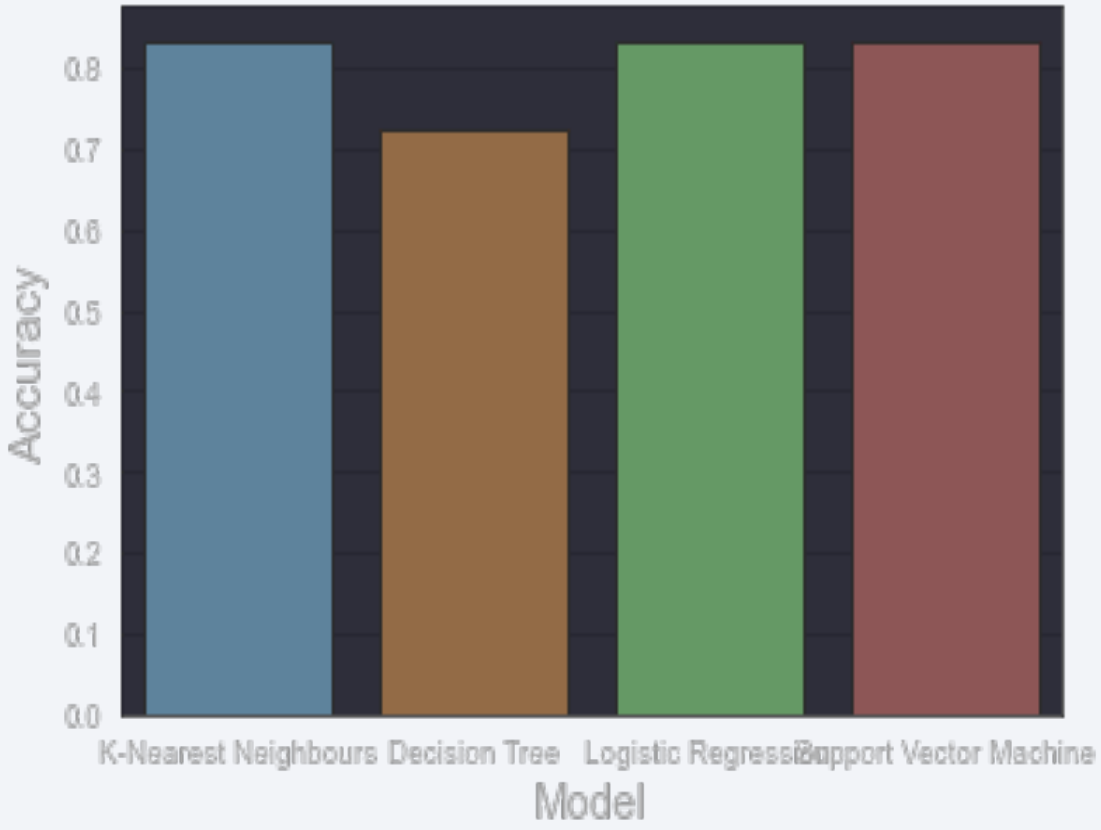
Section 5

# Predictive Analysis (Classification)

# Classification Accuracy

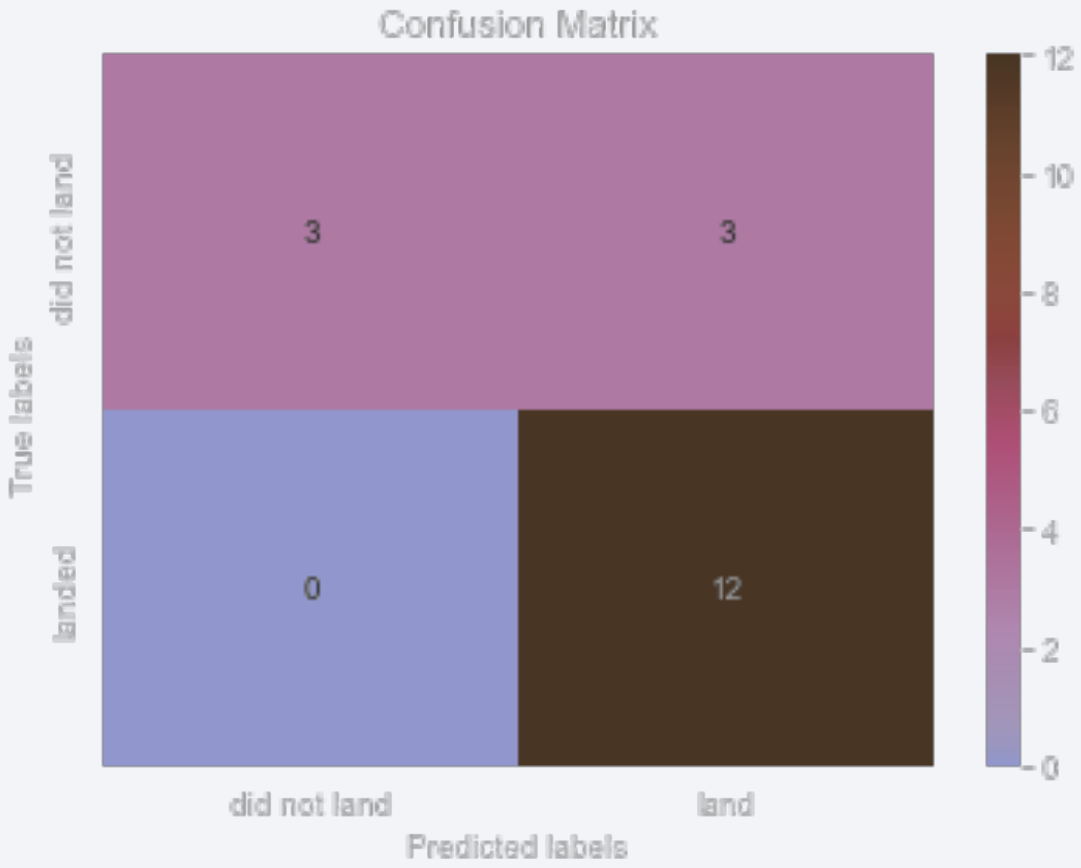
- The best model for the Training Data is Decision Tree with an accuracy score of 0.89
- For the test data, all models except Decision Tree have an accuracy of 83%

| | | |
|---|---|---|
| 0 | K-Nearest Neighbours | 0.833333 |
| 1 | Decision Tree | 0.722222 |
| 2 | Logistic Regression | 0.833333 |
| 3 | Support Vector Machine | 0.833333 |



# Confusion Matrix

- From the confusion matrix, we can see that the models are biased toward successful outcomes.
- Therefore, the model predicts landed when (positive outcomes) for did not land (negative outcomes) but does not predict the inverse case.



# Conclusions

- The larger the flight amount at a launch site, the greater the success rate at a launch site.
- Launch success rate started to increase in 2013 till 2020.
- Orbits ES-L1, GEO, HEO, SSO, VLEO had the most success rate.
- KSC LC-39A had the most successful launches of any sites.
- Depending on whether training or test data is used for accuracy, the Decision Tree model may be the most or least accurate model.

# Thank you!