IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

J Aquino
Nov 2023

Table of Contents

Executive Summary

- Summary of methodologies
  - Data collection using API call or webscraping
  - Data wrangling
  - EDA with SQL and visualization
  - Building an interactive map with Folium
  - Building a dashboard with Plotly Dash
  - Predictive analytics (classification)
- Summary of all results
  - Exploratory data analysis
  - Visual analysis
  - Predictive analysis
  - Conclusions at the end

Introduction

Project background and context

SpaceX advertises Falcon 9 rocket launches on its website, with a cost of $62million; other providers cost upward of $165million each. Much of the savings is because SpaceX can reuse the first stage.

SpaceY, has asked us to determine if SpaceX will be able to reuse the first stage. If we can determine if the first stage will land, we can estimate the cost of each launch and provide SpaceY with a competitive model.

Problems to answer

1. How do we determine if the first stage will land successfully?
2. Will SpaceX reuse the first stage?
3. What is the best machine learning model to use to predict this?

Section 1

Methodology

Methodology

Sections

- Data collection via API call
- Data collection via Data Wrangling
- Exploratory data analysis (EDA) using SQL and visualisation
- Interactive visual analytics using Folium and Plotly Dash
- Predictive analysis using classification models

Data Collection – SpaceX API

Data collection is the process of gathering and analysing data from various sources in order to find solutions to the problems you would like to answer.
Github notebook link: https://github.com/danyzephyr/IBM-Final-Capstone-Project/blob/main/Week%201.1%20-%20SpaceX%20Data%20Collection%20-%20API%20request_JA.ipynb

1. Call the API

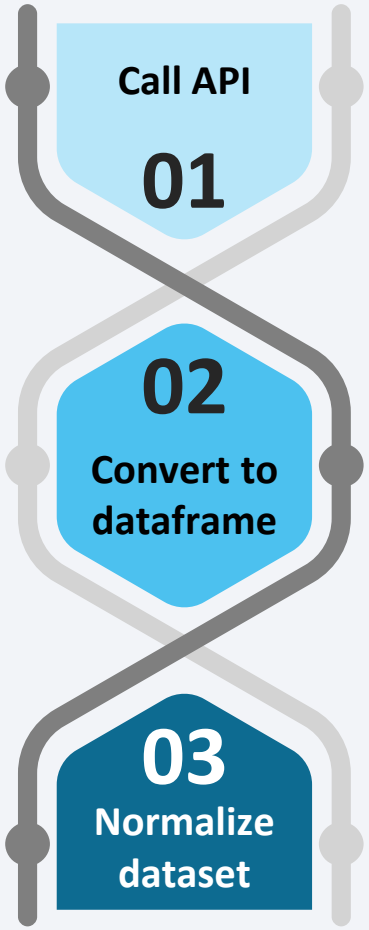
Request and parse the SpaceX launch data using the GET request. The file downloaded from the SpaceX API is a JSON file.

2. Convert to Pandas dataframe

Use .json_normalize() to decode the JSON file and turn it to a Pandas dataframe.

3. Normalize the dataset

Understand the downloaded dataset. Construct the required dataset by extracting only the required fields.
Apply normalization where needed, e.g. date formats, date range restriction, removal of redundant data.




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

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

```
In [28]:
# Use json_normalize meethod to convert the json result into a dataframe
data = pd.read_json(static_json_url)
pd.json_normalize(data)
```

```
In [30]:
# 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
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 – Webscraping

API is not the only method of data collection. Webscraping, or extracting data from a website is an alternative method.
Github notebook link: https://github.com/danyzephyr/IBM-Final-Capstone-Project/blob/main/Week%201.2%20-%20SpaceX%20Webscraping_JA.ipynb

1. Extract data from website

Falcon 9 launch records are available from an HTML table in Wikipedia. BeautifulSoup, a Python feature can be used to parse the data from the website.

2. Construct dataset

Find the 'th' (table headers) and tables from the website and assign these to a variable to start constructing the dataset.
The parsed HTML tables are used to create an empty dictionary with keys.

3. Convert to Pandas dataframe

Convert the empty dictionary created above to a Pandas dataframe.

TASK 1: Request the Falcon9 Launch Wiki page from its URL

First, let's perform an HTTP GET method to request the Falcon9 Launch HTML page, as an HTTP response.

In [8]:
```
# use requests.get() method with the provided static_url
# assign the response to a object
falcondata = requests.get(static_url).text
```

Create a `BeautifulSoup` object from the HTML `response`

In [9]:
```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
falconobj = BeautifulSoup(falcondata,"html5lib")
```

Print the page title to verify if the `BeautifulSoup` object was created properly

In [11]:
```
print(falconobj.prettify()) # extra code ran to display the HTML in a nested structure

# Use soup.title attribute
tag_object=falconobj.title
print(tag_object)
```

```
<title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

In [98]:
```
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'] = []
```

In [103…
```
df= pd.DataFrame({ key:pd.Series(value) for key, value in launch_dict.items() })
```

Data Wrangling

Data wrangling refers to the processes involved with transforming raw data into more readily used formats.
Github notebook link: https://github.com/danyzephyr/IBM-Final-Capstone-Project/blob/main/Week%201.3%20-%20SpaceX%20Data%20Wrangling_JA.ipynb

1. Identify any missing or NaN (Not a Number) values

In order to properly use the dataset, I understood the data types of each column and queried which of these columns have any missing or NaN values.

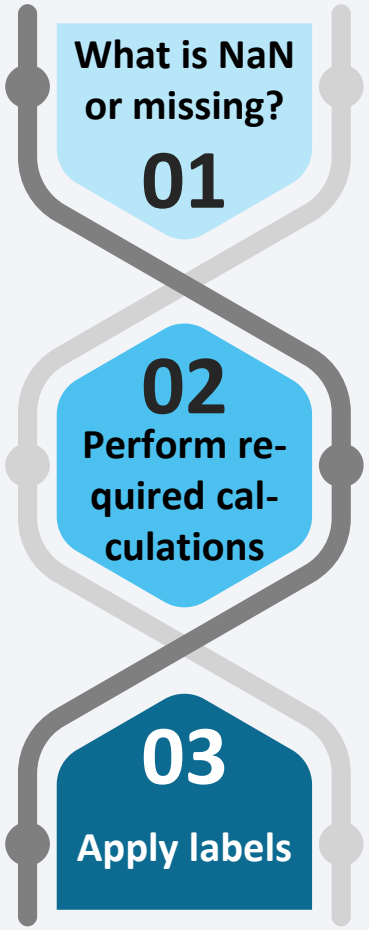
2. Perform required calculations

Some calculations were performed on the dataset:

- value_counts() for # of launches on each site
- value_counts() for # of occurrences per orbit
- value_counts() for # of different landing outcomes then differentiate between successful vs. bad landing outcomes

3. Add calculated variables to dataframe

Any calculated variables from #2 is added to the created dataframe for future querying.




```
In [3]: df.isnull().sum()/len(df)*100
```

```
In [4]: df.dtypes
```

```
# Apply value_counts() on column LaunchSite
launches = df['LaunchSite'].value_counts()
launches
```

```
# landing_outcomes = values on Outcome column
landing_outcomes = df['Outcome'].value_counts()
landing_outcomes
```

```
# Apply value_counts on Orbit column
orbits = df['Orbit'].value_counts()
orbits
```

```
In [15]: bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
         bad_outcomes
Out[15]: {'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise

landing_class = [0 if outcome in bad_outcomes else 1 for outcome in df['Outcome']]
landing_class
```

EDA with Data Visualization

One of SpaceY's key objectives is to be able to predict whether SpaceX will be able to land their rocket on its first stage successfully. If the landing is successful, this would cost only $62million, compared to unsuccessful landings where cost is estimated upward of $165million.

To help predict successful landings, I have created a couple of charts to compare the different launch parameters. You can see detailed explanations on Section 2 of this deck – Insights drawn from EDA. The overview of charts that were plotted are below:

1. Scatter chart on Flight Number vs. Payload
2. Scatter chart on Flight Number vs. Launch Site
3. Scatter chart on Flight Number vs. Orbit Type
4. Scatter chart on Payload vs. Orbit Type
5. Scatter chart on Payload vs. Launch Site
6. Bar chart to represent Success Rate per Orbit Type
7. Line chart to represent Yearly Success trend

Github notebook link: https://github.com/danyzephyr/IBM-Final-Capstone-Project/blob/main/Week%202.2%20-%20SpaceX%20EDA%20with%20Visualization_JA.ipynb

EDA with SQL

The SpaceX dataset can be loaded on to Jupyter Lab. I have written a couple of SQL queries to show how the variations on how this dataset can be queried.

The detailed explanations for a subset of the SQL queries are also in Section 2 – Insights drawn from the EDA. Here is a list of all the SQL queries I have written.

1. Identify all unique launch sites
2. Identify all launch sites beginning with 'CCA'
3. Calculate the total payload mass carried by boosters launched by NASA (CRS)
4. Calculate average payload mass carried by booster version F9 v1.1
5. Identify earliest date when first successful landing in ground pad was achieved
6. List all boosters which have success in drone ship with payload mass between 4000 to 6000
7. Calculate total number of successful vs. non-successful outcomes
8. List all boosters which have the max payload mass
9. Display all months where landing failed for drone ships in 2015
10. Rank the count of landing outcomes between 2010-06-04 and 2017-03-20

Github notebook link: https://github.com/danyzephyr/IBM-Final-Capstone-Project/blob/main/Week%202.1%20-%20SpaceX%20EDA%20with%20SQL_JA.pdf (a PDF file has been uploaded to GitHub as the Python file kept encountering error as Invalid Notebook. The screenshots prove that the SQL queries were running correctly in the kernel.)

Build an Interactive Map with Folium

The following objects have been added to the site map.

Markers

- A **blue** circle with the name for the NASA Johnson space centre using the NASA latitude and longitude coordinates
- **Red** circles for all launch sites with their names displayed as well, using each site's latitude and longitude coordinates
- **Green** markers to differentiate between successful landings and **red** markers for unsuccessful landings at each launch site

Plot lines to highlight proximity of the launch sites to some landmarks

- A **plot line** with the distance displayed from one of the launch sites to the coast line
- A **plot line** with the distance displayed from the same launch site to the nearest highway

Github notebook link: https://github.com/danyzephyr/IBM-Final-Capstone-Project/blob/main/Week%203.1%20-%20SpaceX%20Visual%20Analytics%20with%20Folium%20Lab_JA.ipynb (NB. Github can only show static data so the Folium maps are not shown properly. In order to view the Folium maps and interact with them, you have to type the notebook link and view using nbviewer.org.)

Build a Dashboard with Plotly Dash

I have created the following features and charts in the dashboard:

1. Dropdown list showing all unique launch site names + additional option to show ‘All sites’ to allow users to see an overview of everything or select specific launch sites
2. Pie chart showing the % rate of successful vs. non-successful launches of the user’s selected launch site
3. Range slider for payload mass range to allow users to select the payload range
4. Scatter chart to show the correlation between payload and success rate by booster version for each launch site

Github notebook link: https://github.com/danyzephyr/IBM-Final-Capstone-Project/blob/main/Week%203.2%20-%20SpaceX_Dash_App_JA.py

Predictive Analysis (Classification)

This stage of the process refers to building different classifier models to predict if the first stage of the launch will land successfully.
Github notebook link: https://github.com/danyzephyr/IBM-Final-Capstone-Project/blob/main/Week%204%20-%20SpaceX%20Machine%20Learning.ipynb

1. Preprocess the data

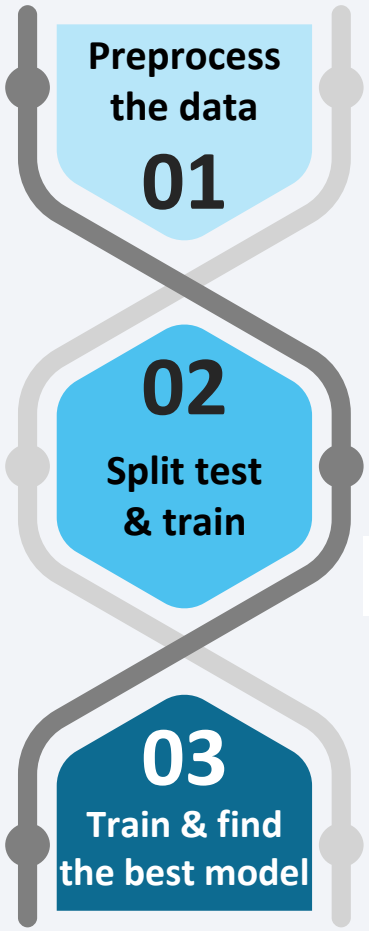
A NumPy array is created from the class column. The data is then standardized by using StandardScaler to fit and transform it.

2. Split test and train data

The dataset was split into test and train via train_test_split, with the overall parameter test size set to 20%.

3. Create, train and determine the best model

I created 4 different models: logistic regression, SVM, decision tree and KNN. All 4 models' accuracy scores were calculated using .score on the test data. A confusion matrix was also used to assess each mode.




```
Y = data['Class'].to_numpy()
Y
```

```
array([0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0,
       1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1,
       1, 1], dtype=int64)
```

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

```
array([[-1.71291154e+00, -1.94814463e-16, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       [-1.67441914e+00, -1.19523159e+00, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       [-1.63592675e+00, -1.16267307e+00, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       ...,
       [ 1.63592675e+00,  1.99100483e+00,  3.49060516e+00, ...,
         1.19684269e+00, -5.17306132e-01,  5.17306132e-01],
       [ 1.67441914e+00,  1.99100483e+00,  1.00389436e+00, ...,
         1.19684269e+00, -5.17306132e-01,  5.17306132e-01],
       [ 1.71291154e+00, -5.19213966e-01, -6.53912840e-01, ...,
        -8.35531692e-01, -5.17306132e-01,  5.17306132e-01]])
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
```

```
X = ['Logistic Regression', 'SVM', 'Decision Tree', 'KNN']
Y = [log_score, svm_score, tree_score, knn_score]
plt.bar(X,Y)
plt.xlabel('ML Model')
plt.ylabel('Accuracy')
plt.title('Accuracy Rating of the ML Models')
plt.show()
```

NB. These are selected code snippets. In order to view the extensive code, please access the notebook link provided above.

Key Findings Summary

Exploratory data analysis

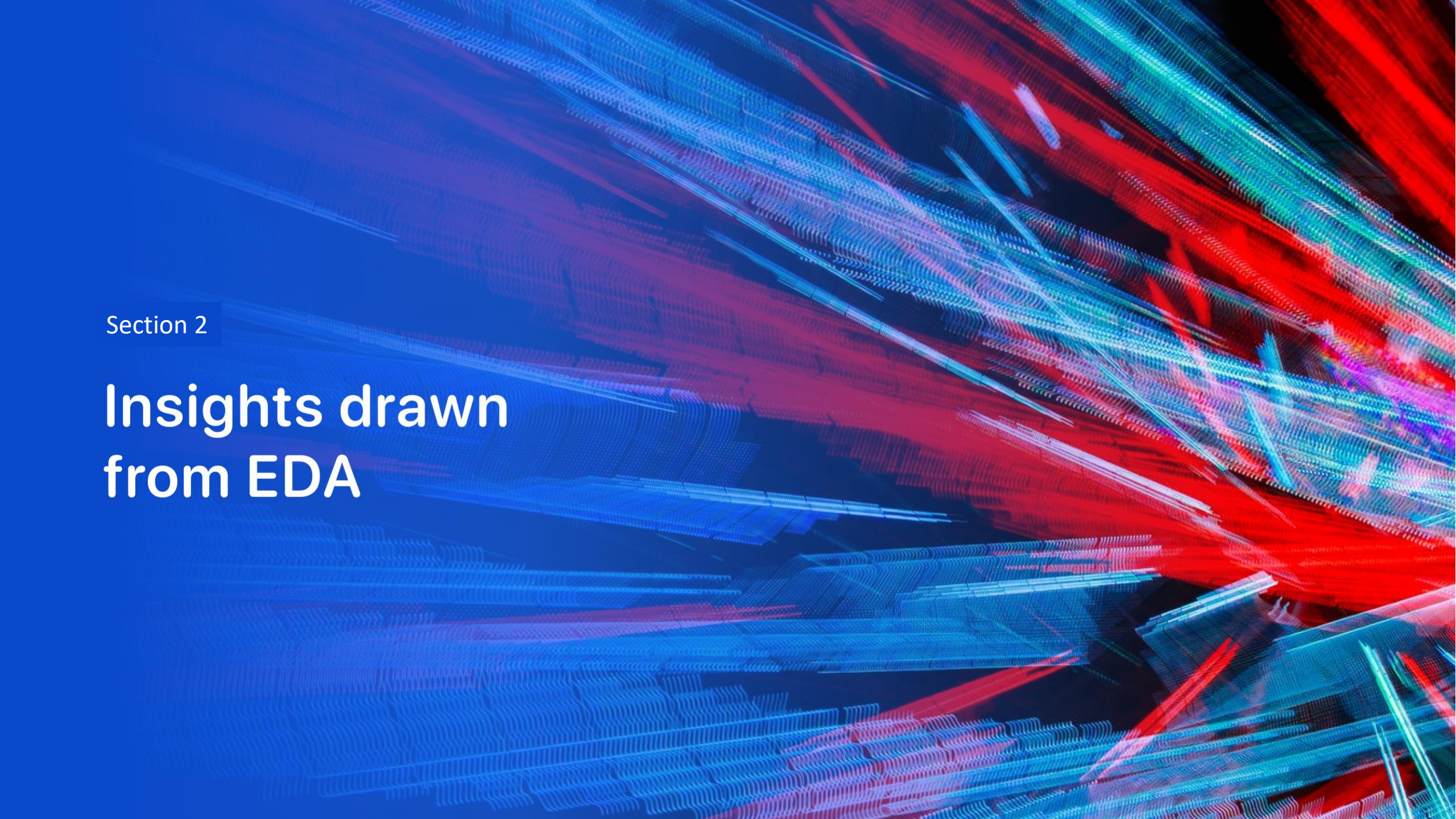
- As the flight number increases, the first stage landing is more likely to be successful.
- Orbits ES-L1, GEO, HEO and SSO have a 100% success rate.

Visual analytics

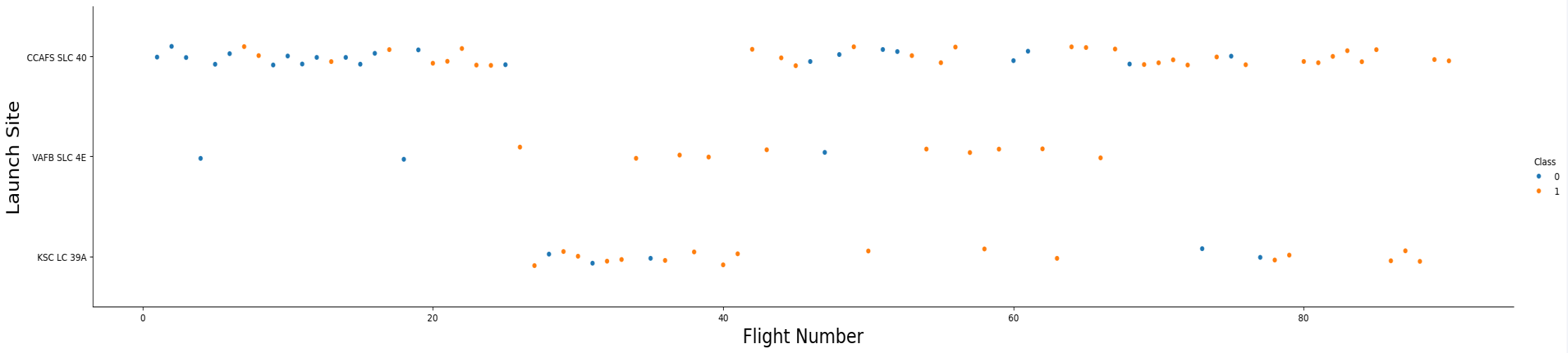
- KSC LC-39A has the highest success rate for launches at 41.7%.
- The proximity plot lines in the Folium site map shows that the launch sites are of sufficient distance from busy landmarks, like a coastline or highway.

Predictive analysis

- All 4 models have an 83% accuracy. I would like to caveat this by saying we need a bigger test data sample to produce a different outcome, which is currently not possible for the dataset we have been provided.

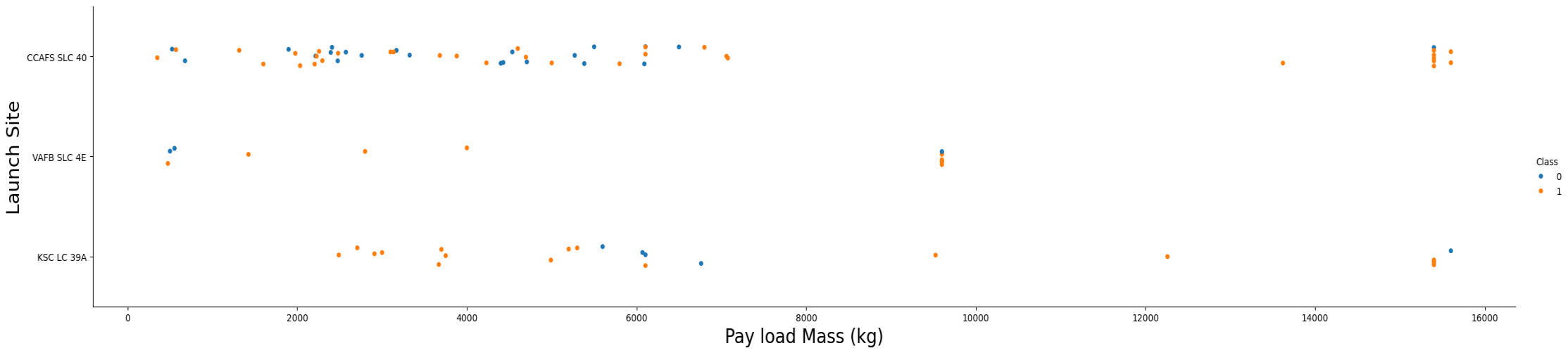

Section 2
Insights drawn from EDA

Flight Number vs. Launch Site



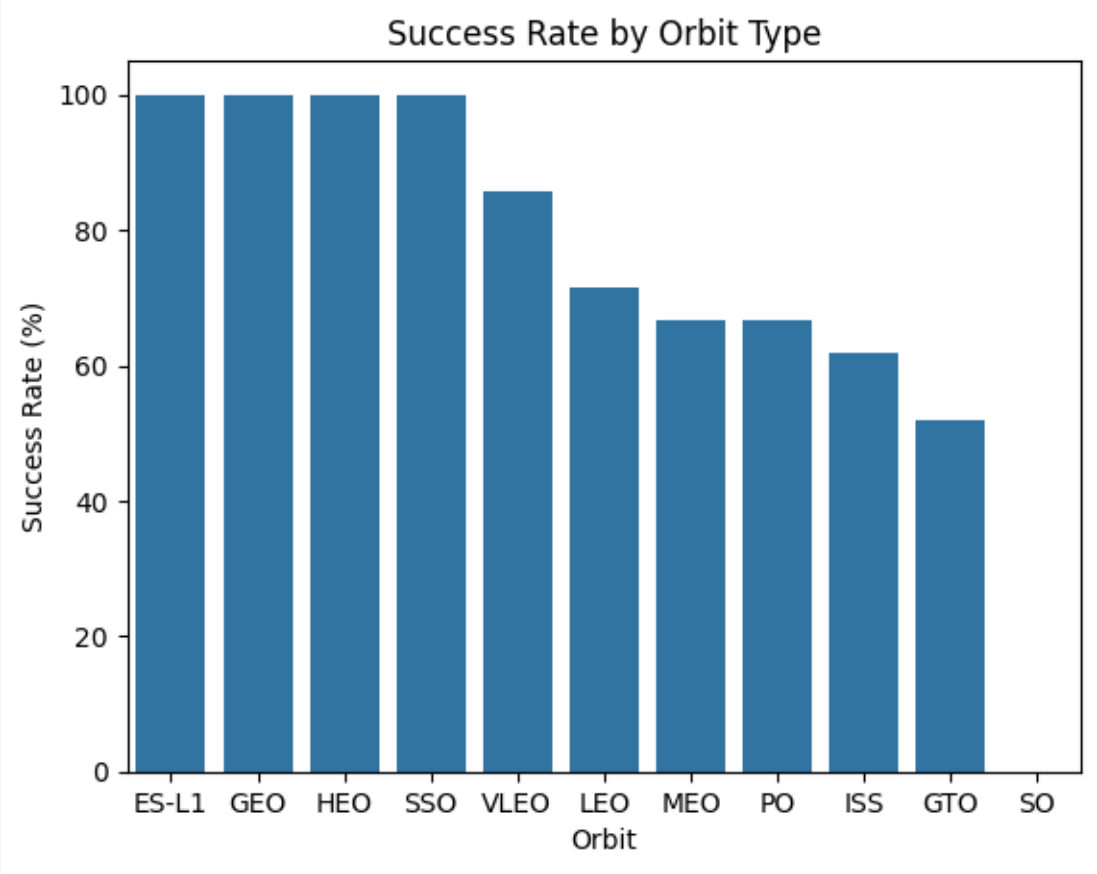
- There were more fails (blue/class 0) with earlier flights. Later flights were more successful (orange/class 1).
- Out of the 3 launch sites, the least successful was CCAFS SLC 40.
- We can infer that the new launches have a higher success rate.

Payload vs. Launch Site



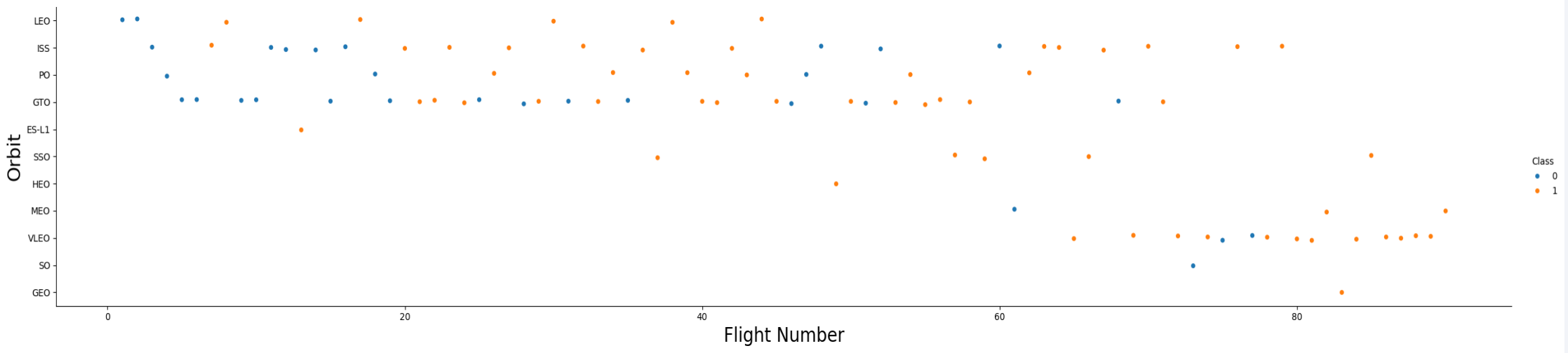
- There seems to be a higher success rate with payload masses greater than ~ 9000 kg.
- VAFB SLC-4E has not launched anything greater than 10000 kg.
- CCAFS SLC-40 has not launched anything with payload mass between 7000 to 13500 kg but seems to be quite successful in launching above 13500 kg.
- KSC LC-39A has a 100% success rate with launching payload masses below ~ 5500 kg.

Success Rate vs. Orbit Type



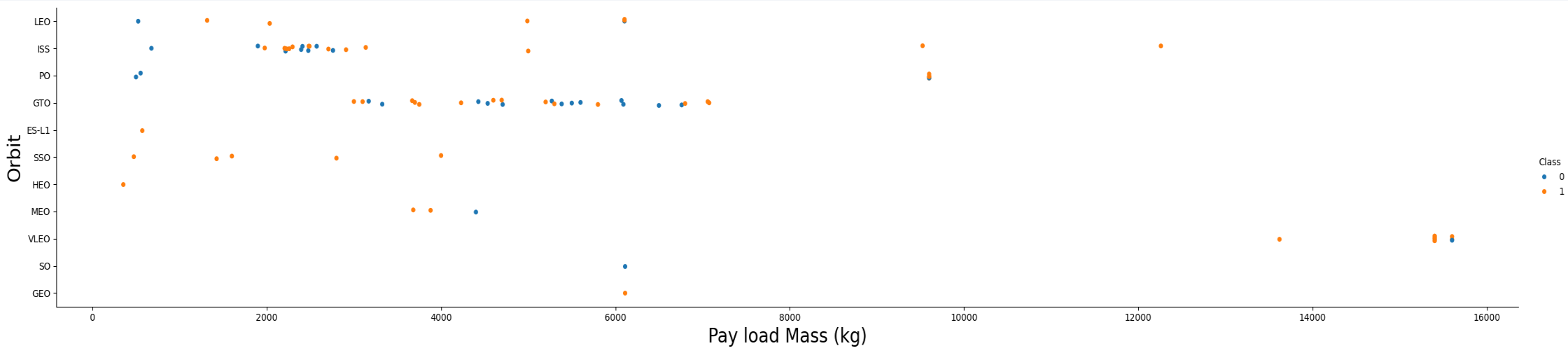
- As mentioned on the Key Findings slide, ES-L1, GEO, HEO and SSO have 100% success rates.
- This is followed by the 50% to 80% success rate of VLEO, LEO, MEO, PO, ISS and GTO orbits.
- SO has 0% success rate.

Flight Number vs. Orbit Type



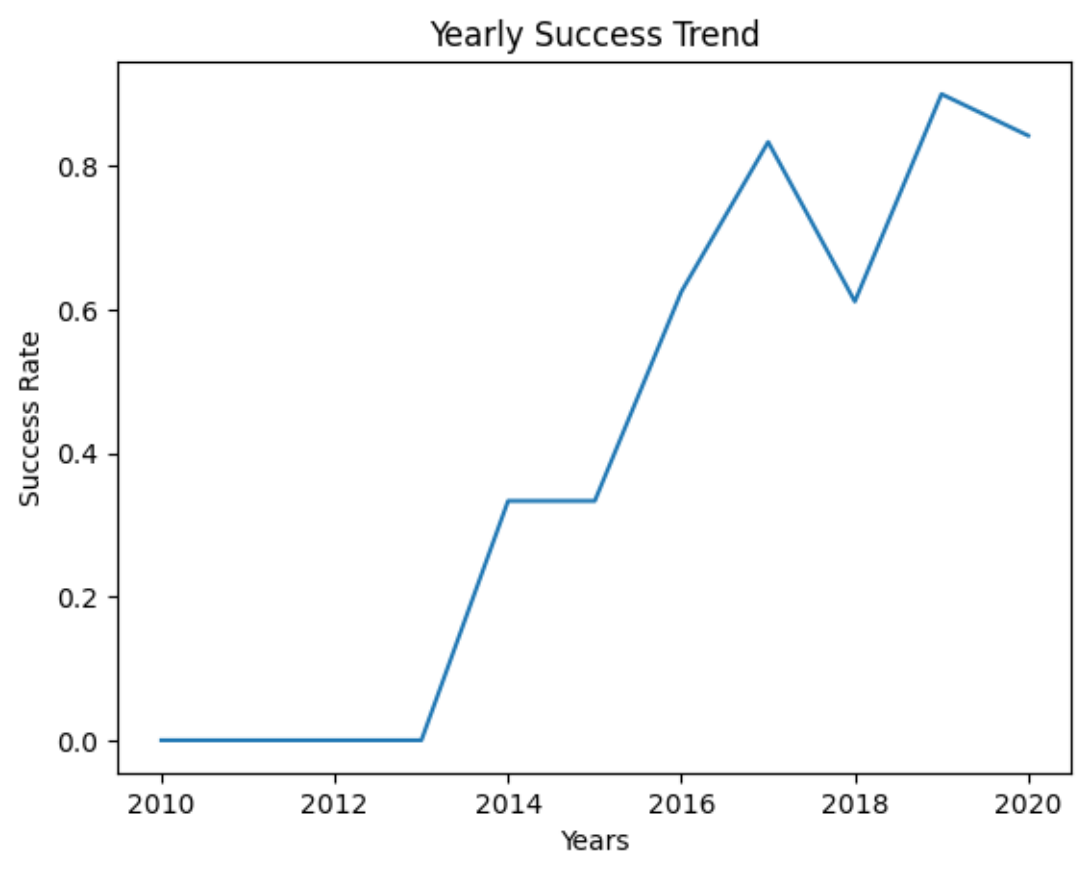
- It seems that the success rate increases when the flight numbers increase too, except for the GTO orbit which still presents mixed results.
- This is most noticeable in the LEO orbit.

Payload vs. Orbit Type



- GTO orbit has mixed success rate for payload masses.
- The lighter payload masses are the most successful with SSO orbit.
- The bigger payload masses seem to have the most success with ISS.

Launch Success Yearly Trend



- The success rate started to increase from 2013.
- The trend was increasing sharply apart from the steep downturn in 2017-2018.
- Overall, the success rate is marginally better compared to 2013.

SQL to display all unique launch sites

Task 1: Display the names of the unique launch sites in the space mission.

```
[9]: %%sql

     select distinct(Launch_Site) from SPACEXTABLE order by Launch_Site

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

[9]:

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

DISTINCT is used to display only unique launch site names.

SQL to display launch sites beginning with ‘CCA’

Task 2: Display 5 records where launch sites begin with the string ‘CCA’.

```
[11]: %%sql

      select * from SPACEXTABLE where Launch_Site like 'CCA%' limit 5
```

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

[11]:

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

LIKE is used to query wildcard values. ‘CCA%’ indicates we want to query for values beginning with CCA.

LIMIT 5 is used to only show 5 records.

SQL to calculate Payload Mass (total and avg)

Task 3: Display the total payload mass carried by boosters launched by NASA (CRS).

```
%%sql

select sum(PAYLOAD_MASS__KG_) as 'Total Payload Mass', Customer
from SPACEXTABLE
where Customer = 'NASA (CRS)'
```

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

Total Payload Mass	Customer
45596	NASA (CRS)

SUM is used to calculate the total of a column.

AS is used to provide a new name for the calculated column.

Task 4: Display average payload mass carried by booster version F9 v1.1.

```
%%sql

select avg(PAYLOAD_MASS__KG_) as 'Average Payload Mass', Booster_Version
from SPACEXTABLE
where Booster_Version like 'F9 v1.1%'
```

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

Average Payload Mass	Booster_Version
2534.6666666666665	F9 v1.1 B1003

AVG is used to calculate the average of a column. This query uses AS again to give a more meaningful column name and LIKE + % to search for values beginning with F9 v1.1.

SQL to display 1st successful ground pad landing

Task 5: List the date when the first successful landing outcome in ground pad was achieved.

Use the min function.

```
%%sql

select min(Date), Landing_Outcome from SPACEXTABLE
where Landing_Outcome = 'Success (ground pad)'
```

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

min(Date)	Landing_Outcome
2015-12-22	Success (ground pad)

MIN is used to provide the lowest value of a column. Here, I used MIN on Date to get the first date that meets the WHERE criteria.

SQL to display successful drone ship landing with payload between 4000 and 6000

Task 6: List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000.

```
%%sql

select Booster_Version, Landing_Outcome, PAYLOAD_MASS__KG_
from SPACEXTABLE
where Landing_Outcome = 'Success (drone ship)'
and PAYLOAD_MASS__KG_ between 4000 and 6000
order by PAYLOAD_MASS__KG_
```

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

Booster_Version	Landing_Outcome	PAYLOAD_MASS__KG_
F9 FT B1026	Success (drone ship)	4600
F9 FT B1022	Success (drone ship)	4696
F9 FT B1031.2	Success (drone ship)	5200
F9 FT B1021.2	Success (drone ship)	5300

The WHERE clauses have multiple conditions:

- Landing outcome → Success (drone ship)
- Payload mass must be between 4000 and 6000

I used an additional ORDER_BY clause to present the findings better as it sorts out Payload in ascending order.

SQL to display total of successful and non-successful outcomes

Task 7: List the total number of successful and failed mission outcomes.

Task 7

List the total number of successful and failure mission outcomes

[49]:
```
%%sql

select count(Mission_Outcome) as 'Count of Mission Outcomes', Mission_Outcome
from SPACEXTABLE
group by Mission_Outcome
```

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

[49]:

Count of Mission Outcomes	Mission_Outcome
1	Failure (in flight)
99	Success
1	Success (payload status unclear)

COUNT is used to count the number of records that meet the criteria.

GROUP BY “groups” records with the same values together. Above, the total number of records is counted for each distinct mission outcome value.

SQL to display booster versions which have max payload mass

Task 8: List the names of the booster versions which have carried the maximum payload mass. Use a subquery.

```
%%sql

select Booster_Version, PAYLOAD_MASS__KG_
from SPACEXTABLE
where PAYLOAD_MASS__KG_ in (select max(PAYLOAD_MASS__KG_)
                            from SPACEXTABLE)
```

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

The SUBQUERY contains the query to search for the MAX Payload Mass from the same database table.

The first query is saying to only produce the Booster Versions which have the maximum payload mass. As you can see, the query produced all booster versions with 15600 payload mass.

SQL to display 2015 launch records

Task 9: List the records which will display the month names, failed landings in drone ship, booster version and launch site for the months in year 2015.

Task 9

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

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

[24]:
```
%%sql

select substr(Date,6,2) as 'Month', substr(Date,0,5) as 'Year', Landing_Outcome, Booster_Version, Launch_Site
from SPACEXTABLE
where Landing_Outcome = 'Failure (drone ship)'
and substr(Date,0,5) = '2015'
```

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

[24]:

Month	Year	Landing_Outcome	Booster_Version	Launch_Site
10	2015	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
04	2015	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

In this query, SUBSTR extracts the month name and year from the Date field.

AS is used to provide meaningful names to the extracted columns.

SQL to rank landing outcomes between 2010-06-04 and 2017-03-20

Task 10: Rank the count of landing outcomes between the date 2010-06-04 and 2017-03-20 in descending order.

```
%%sql

select count(*) as 'Count', Landing_Outcome
from SPACEXTABLE
where Date between '2010-06-04' and '2017-03-20'
group by Landing_Outcome
order by count(*) desc
```

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

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

Here, COUNT counts the records per group – the distinct value of landing outcome.

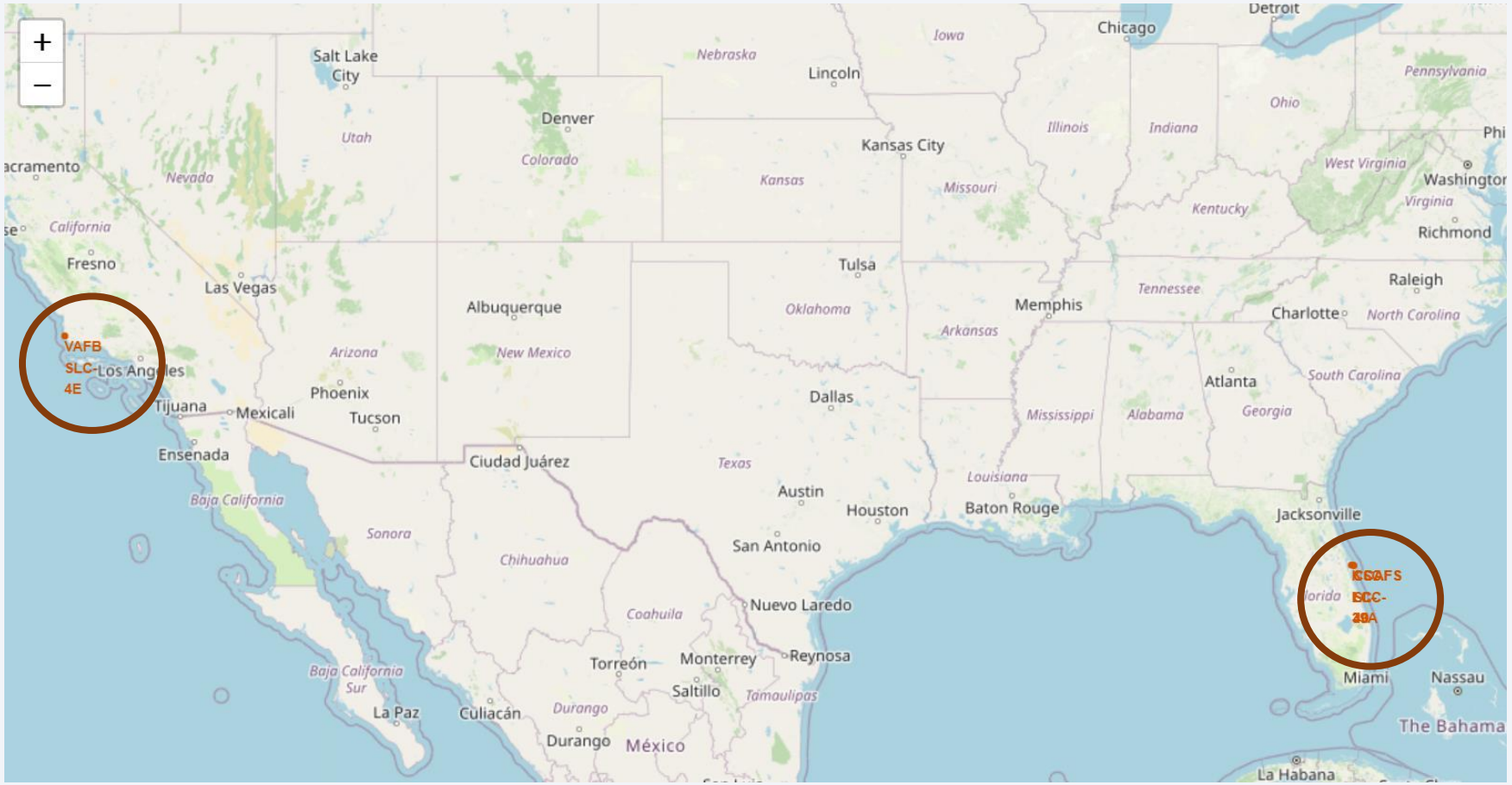
The WHERE clause restricts the records to the date range specified in the instructions.

The count of landing outcomes are sorted in descending order.

Section 3

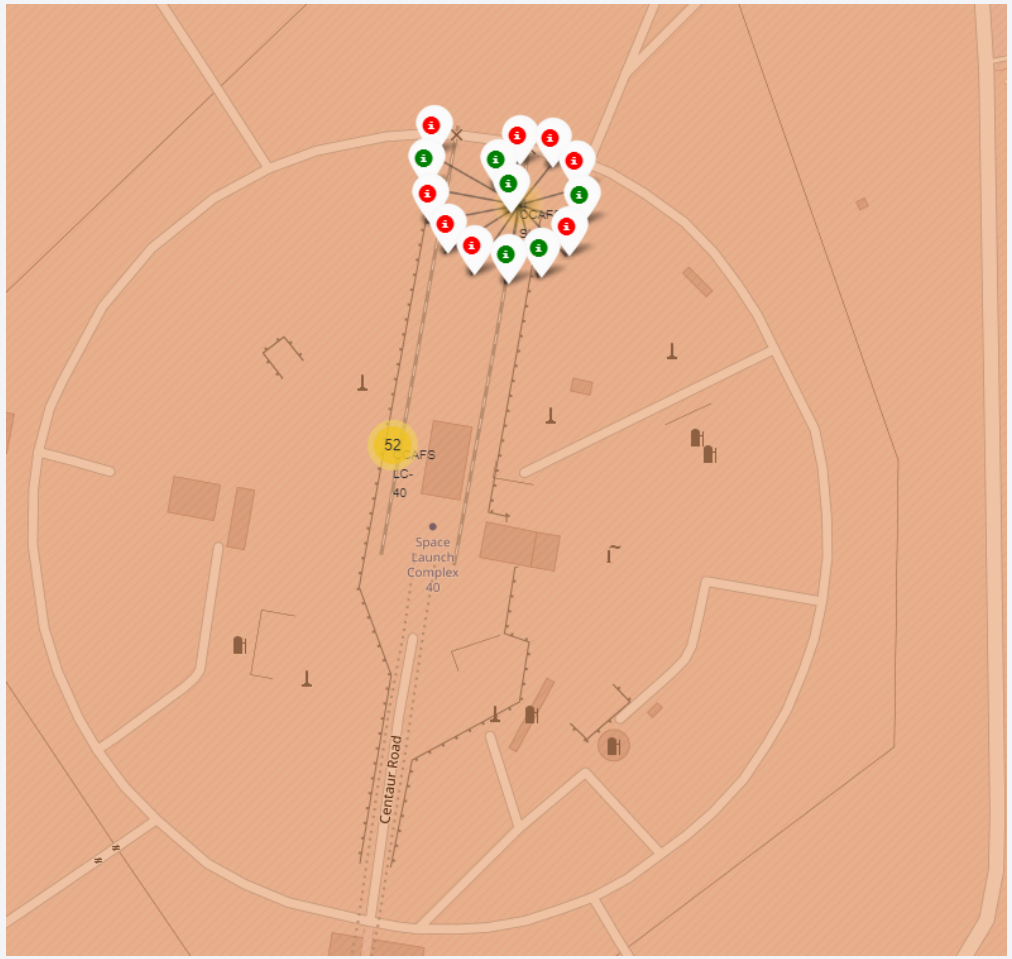
Launch Sites Proximities Analysis

All of SpaceX's Launch Sites



- There is a higher success rate with launch sites closer to the equator.
- More successful launches = less cost.

Successful vs. Non-Successful Launch Outcomes

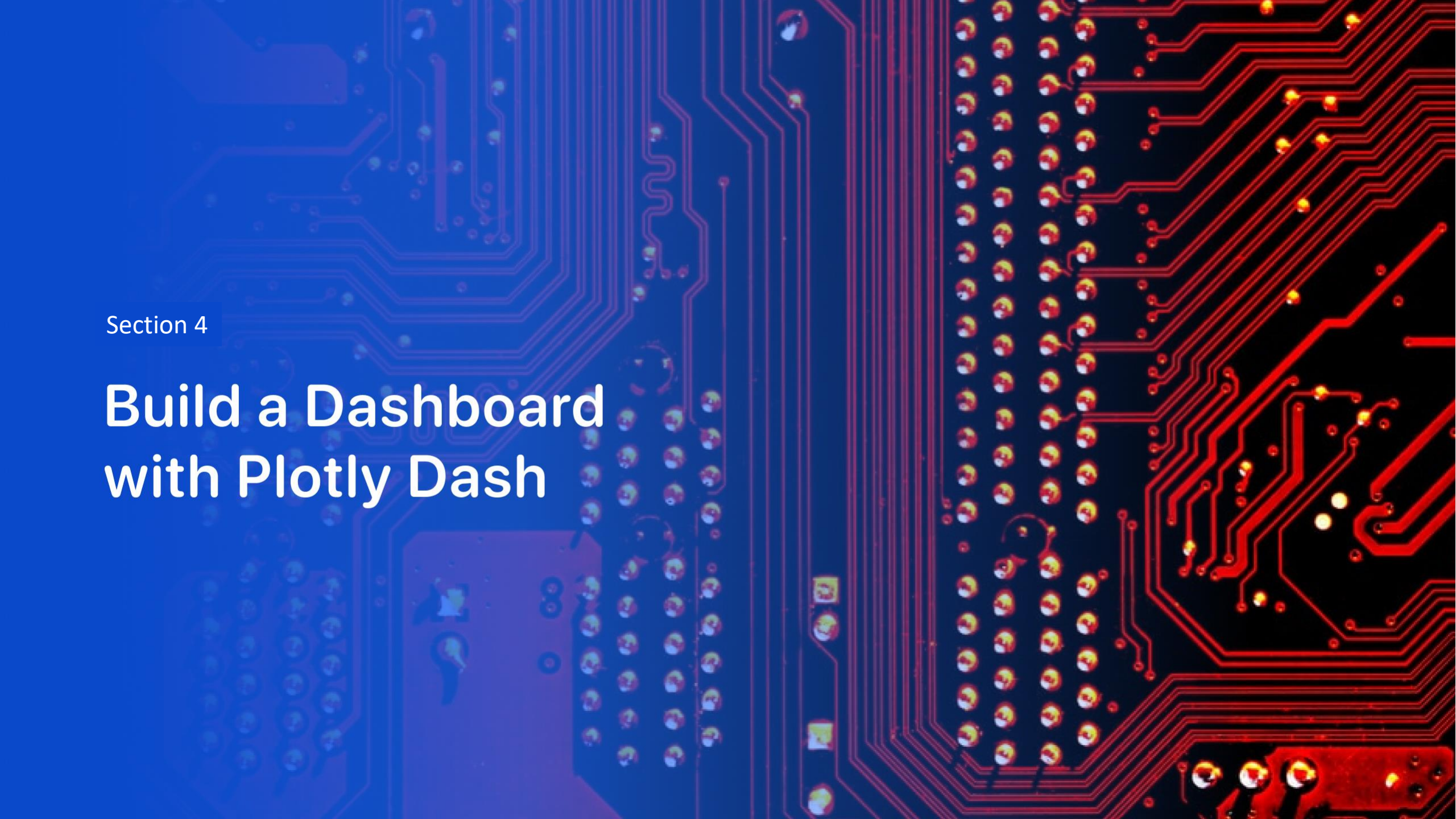


Green markers represent the successful outcomes while red ones represent the unsuccessful ones.

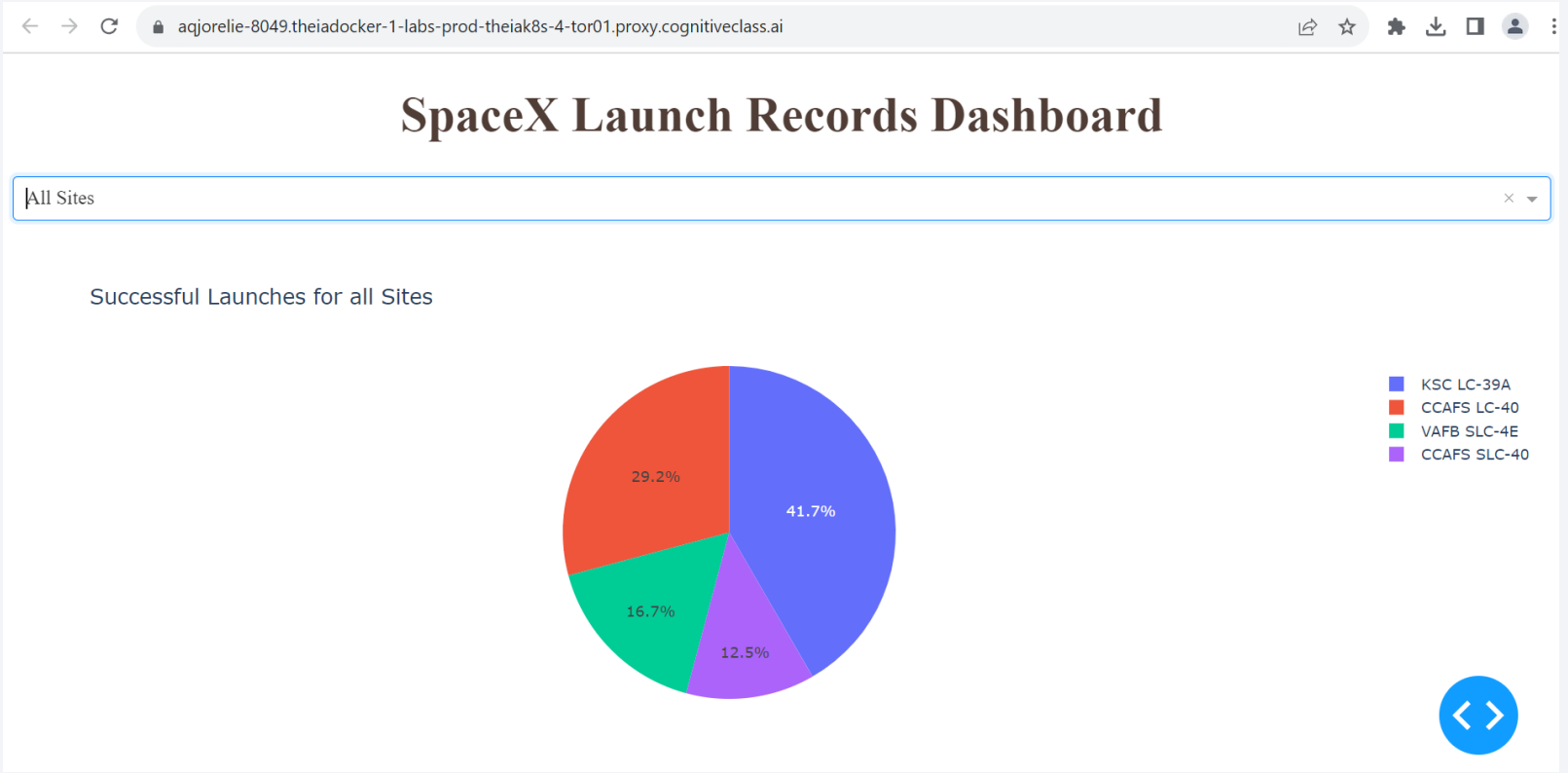
A Launch Site's Distance to the Coastline and the Highway

The launch sites are not in close proximity to railways, highways and coastlines. Above is just one screenshot showing a launch site's distance to a highway and the coastline.

The city is also of sufficient distance – probably just close enough for easier transport of supplies or get support when required.

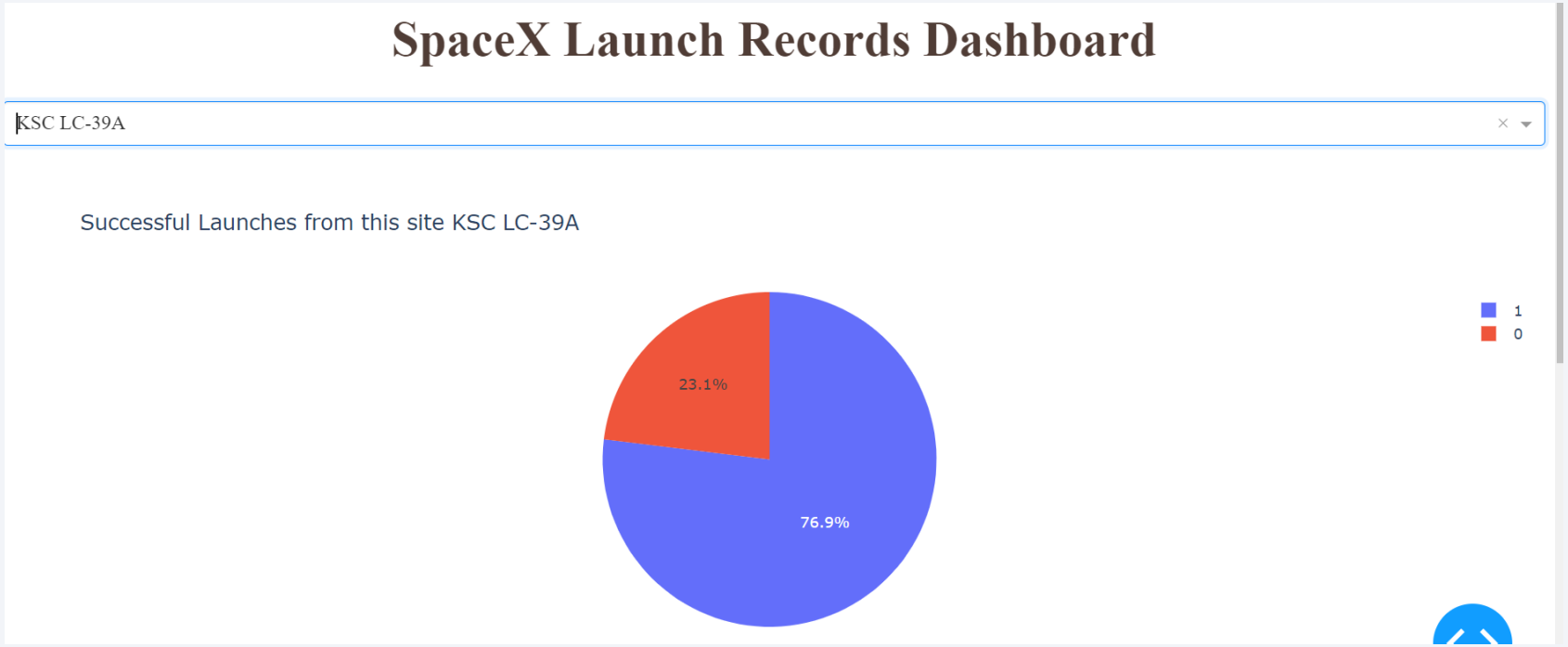

Section 4
Build a Dashboard with Plotly Dash

SpaceX Dashboard showing Total Success Count across all Launch Sites



- KSC LC-39A has the highest success rate out of all the sites at 41.7%.
- CCAFS SLC-40 has the lowest success rate at 12.5%.

Chart showing the Launch Site with the highest Launch Success Rate



- KSC LC-39A achieved a success rate of 76.9% against a failure rate of 23.1%.

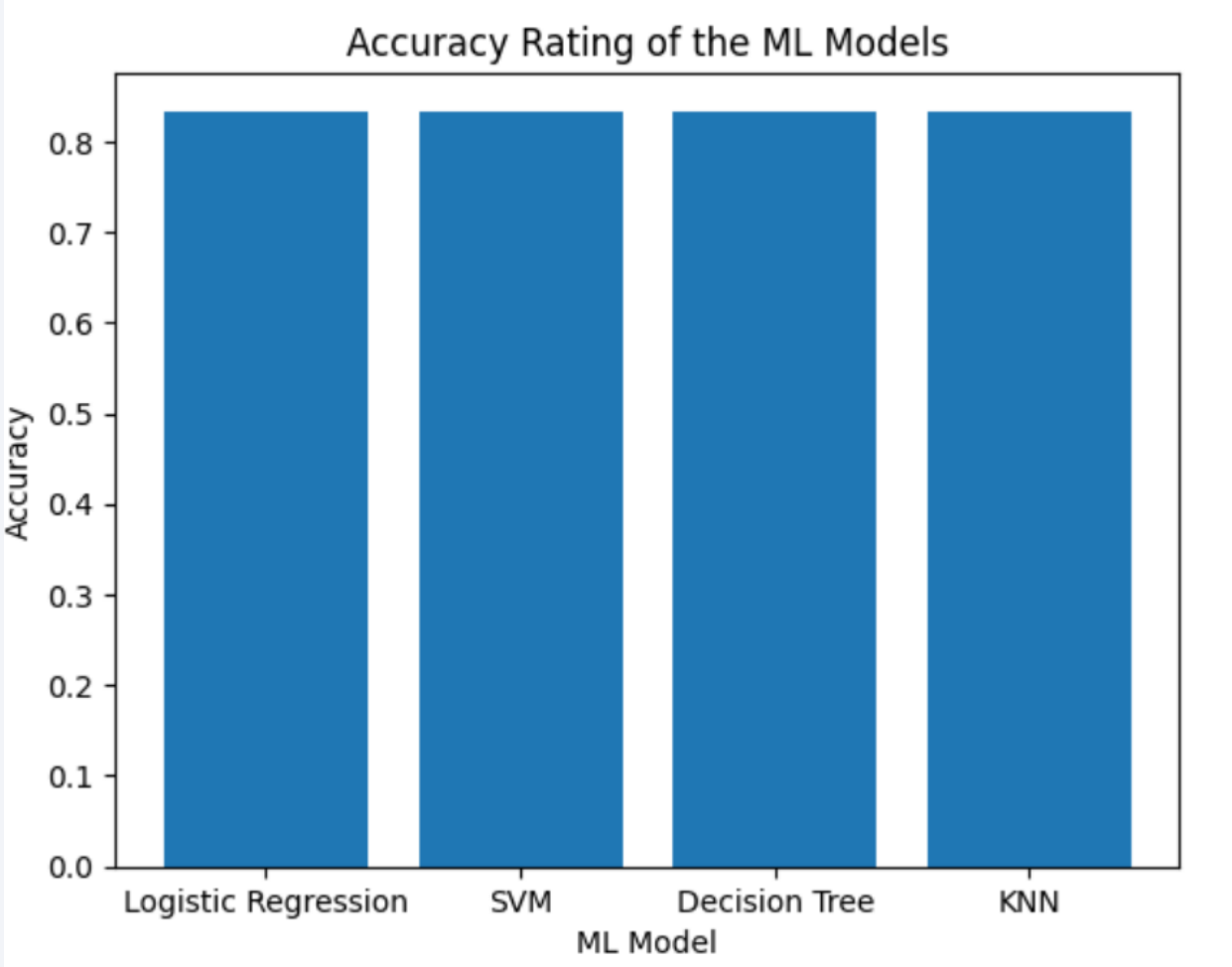
<Dashboard Screenshot 3>

- Payloads with mass between 2000 to 5000 kg have a higher success rate.
- V1.1 booster is the least successful booster version.

Section 5

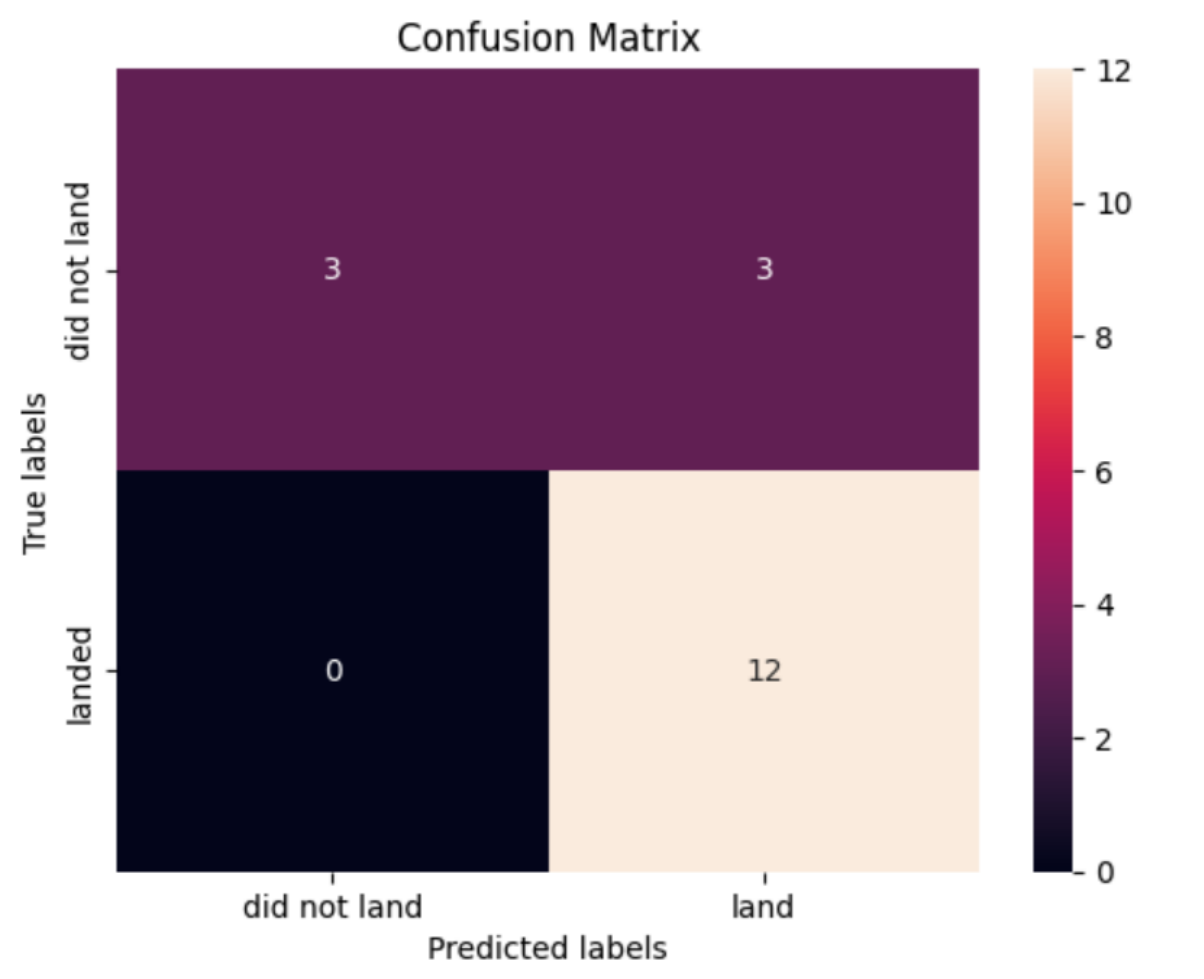
Predictive Analysis (Classification)

Classification Accuracy

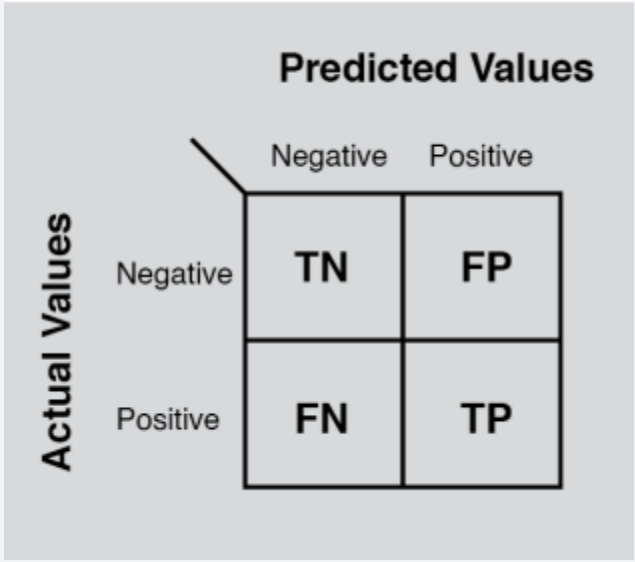


All 4 models have produced the same accuracy rating so there is no model that outperforms the other.

Confusion Matrix



Similar to the bar chart from the previous slide, all confusion matrices were also similar.



What do the values in the matrix mean?

- 12 True positives
- 3 True negatives
- 3 False positives
- 0 False negative

As all 4 produced the same confusion matrices, there is not one model that outperforms the other.

Conclusions

- Logistic regression, KNN, decision tree and SVM all produced the same accuracy rating and false positives, hence any of these models can be used to test and train the dataset. To reiterate, a bigger dataset/sample will be required to further test the accuracy of these models.
- Based on the analyses, the variables resulting to more successful launches are:
    - Sites closer to the equator and to the coastline
    - Usually have a higher payload mass
    - Occur in KSC LC-39A launch site, any payload above 5500 kg were 100% successful from this site
    - Probability of success seemed to increase with every passing year
    - Launches in orbits ES-L1, GEO, HEO and SSO

Thank you!