IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Data Science Final Project

Dana Valeria Ponce Vera
January 11th of 2025

Outline

Executive Summary

Summary of methodologies

- Data collection through API
- Data collection with Web Scraping
- Data Wrangling
- EDA with SQL
- EDA with Visualization
- Interactive Visual Analytics with Folium
- Build an Interactive Dashboard with Plotly Dash
- Machine Learning Prediction

Summary of all results

- EDA results
- Interactive results
- Predictive Analytics results

Introduction

SpaceX is a company that has revolutioned the concept of space traveling or, perhaps, it has made the concept its own. A big part of SpaceX's success lies in correctly handling their launches by running different tests with said launches and creating data that the company can analyse and use to better handle their future launches.

The work done throughout the project aims to resolve different posible problems that may come up during the process of a launch through handling data, such as determining if the first stage of the rocket will be successful, if SpaceX will reuse the first stage or maybe determining the price of each launch.

Section 1

Methodology

Methodology

Executive Summary

- **Data collection methodology:**
  - ❖ The data used was collected through SpaceX REST API and through Web Scraping from the Wikipedia page “List of Falcon 9 and Falcon Heavy launches”.
- **Perform data wrangling**
  - ❖ Done by transforming categorical data through One Hot Encoding for machine learning and eliminating irrelevant information for the project.
- **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; resources like K-Nearest Neighbor, Decision Tree, Logistic Regression, and Support Vector Machine were used to determine the most effective model.

Data Collection

The data was collected through **SpaceX REST API** and by **Web Scraping** Wikipedia pages.

Which provided information such as:

- Rocket that was used.
- Payload delivered.
- Details about the launches.
- Details about the landings.

SpaceX REST API

Retrieve and process data with **GET request**

Data in **JSON format**

JSON to dataframe

WebScraping Falcon 9 data

BeautifulSoup to webscrape HTML tables

Parse data from tables

Tables to dataframe

Data Collection – SpaceX API

Requesting and **parsing** SpaceX launch data with **GET request**

Constructing dataset with data obtained

Task 1: Request and parse the SpaceX launch data using the GET request

To make the requested JSON results more consistent, we will use the following static response object for this project:

```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_
```

We should see that the request was successfull with the 200 status response code

```
response.status_code
```

```
200
```

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

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

Using the dataframe `data` print the first 5 rows

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

static_fire_date_utc	static_fire_date_unix	net	window	rocket	success	failures	details	crew	ships	cap

Finally lets construct our dataset using the data we have obtained. We we combine the columns into a dictionary.

```
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}
```

Then, we need to create a Pandas data frame from the dictionary launch_dict.

```
# Create a data from launch_dict
df = pd.DataFrame.from_dict(launch_dict)
```

Show the summary of the dataframe

```
# Show the head of the dataframe
df.head()
```


Github link: jupyter-labs-spacex-data-collection-api.ipynb

Data Collection – SpaceX API

Filtering data to include **only Falcon9** launches

Calculating mean of PayloadMass and replacing missing values

Task 2: Filter the dataframe to only include `Falcon 9` launches

Finally we will remove the Falcon 1 launches keeping only the Falcon 9 launches. Filter the data dataframe using the `BoosterVersion` column to only keep the Falcon 9 launches. Save the filtered data to a new dataframe called `data_falcon9`.

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

Now that we have removed some values we should reset the FlgihtNumber column

```
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9
```

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	La
4	1	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	
5	2	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False	False	False	
6	3	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False	False	False	
7	4	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False	False	False	
8	5	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	False	False	False	
...	...	...	...	...	...	...	...	...	...	...	...	

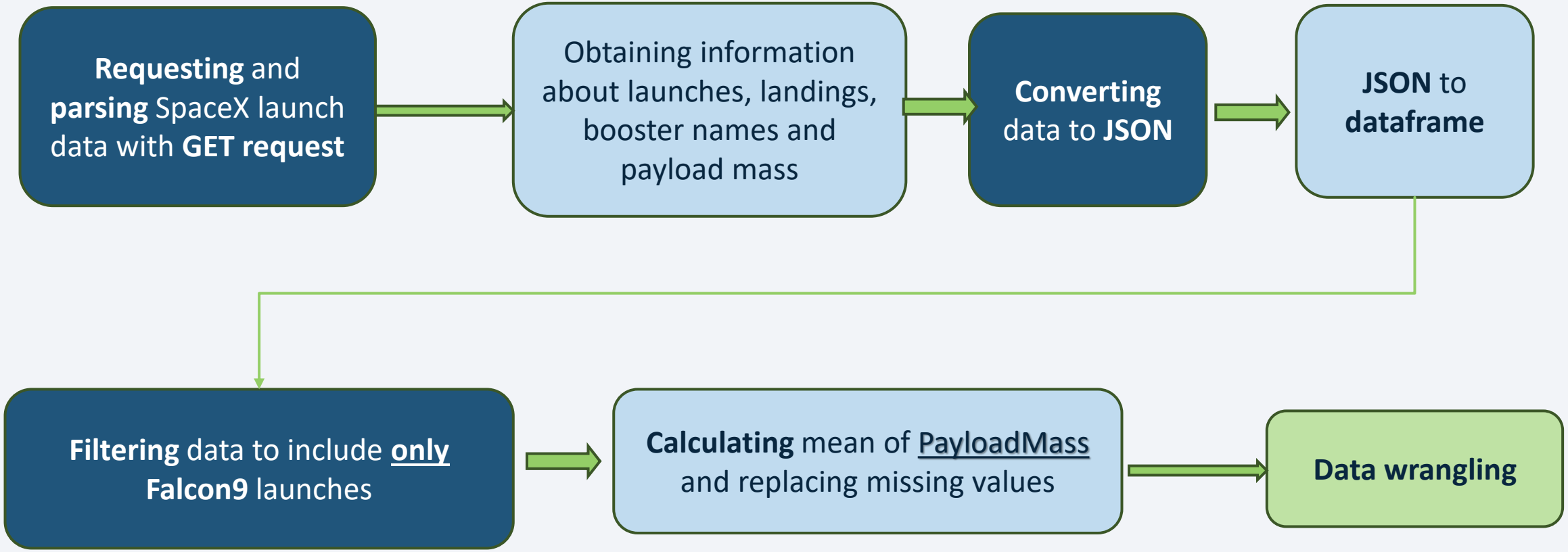
```
# Calculate the mean value of PayloadMass column
payloadmassavg = data_falcon9['PayloadMass'].mean()
# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'].replace(np.nan, payloadmassavg, inplace=True)
```

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

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

Github link: jupyter-labs-spacex-data-collection-api.ipynb

Data Collection – SpaceX API



Github link: jupyter-labs-spacex-data-collection-api.ipynb

Data Collection - Scraping

Requesting Falcon9 Wiki page from its URL

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.

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

```
<Response [200]>
```

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

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

```
<!DOCTYPE html>
```

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

```
# Use soup.title attribute
print(soup.title)
```

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

Github link: jupyter-labs-webscraping.ipynb

Extracting column/variable names from **HTML table header**

TASK 2: Extract all column/variable names from the HTML table header

Next, we want to collect all relevant column names from the HTML table header

Let's try to find all tables on the wiki page first. If you need to refresh your memory about `BeautifulSoup`, please check the external reference link towards the end of this lab

```
# 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')
```

Starting from the third table is our target table contains the actual launch records.

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

```
<table class="wikitable plainrowheaders collapsible" style="width: 100%;">
<tbody><tr>
<th scope="col">Flight No.
</th>
<th scope="col">Date and<br/>time (<a href="/wiki/Coordinated_Universal_Time" title="Coordinated Universal Time">UTC</a>)
</th>
<th scope="col"><a href="/wiki/List_of_Falcon_9_first-stage_boosters" title="List of Falcon 9 first-stage boosters">Version,<
br/>Booster</a> <sup class="reference" id="cite_ref-booster_11-0"><a href="#cite_note-booster-11"><span class="cite-bracket">
[</span>b<span class="cite-bracket">]</span></a></sup>
```

```
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 row in first_launch_table.find_all('th'):
    name = extract_column_from_header(row)
    if (name != None and len(name) > 0):
        column_names.append(name)
```

Check the extracted column names

```
print(column_names)
```

```
['Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome']
```

Data Collection - Scraping

Creating dataframe by parsing the launch HTML tables

```
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']=[]
```

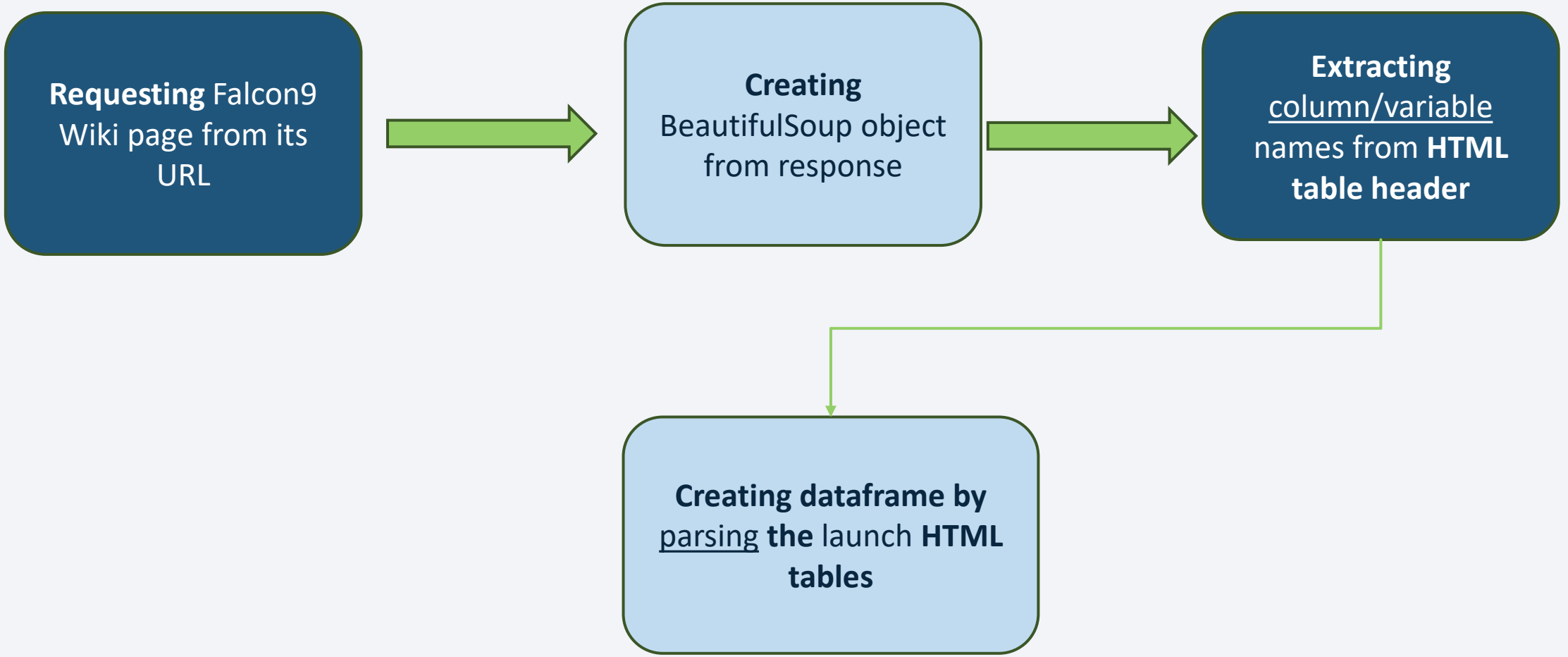
```
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders collapsible")):
   # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number corresponding to launch a number
        if 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.`
            launch_dict['Flight No.'].append(flight_number)
            #print(flight_number)
            datatimelist=date_time(row[0])

            # 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)
```

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

	Flight No.	Launch site	Payload	Payload mass	Orbit	Customer	Launch outcome	Version Booster	Booster landing	Date	Time
0	1	CCAFS	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success\n	F9 v1.07B0003.18	Failure	4 June 2010	18:45
1	2	CCAFS	Dragon	0	LEO	NASA	Success	F9 v1.07B0004.18	Failure	8 December 2010	15:43
2	3	CCAFS	Dragon	525 kg	LEO	NASA	Success	F9 v1.07B0005.18	No attempt\n	22 May 2012	07:44
3	4	CCAFS	SpaceX CRS-1	4,700 kg	LEO	NASA	Success\n	F9 v1.07B0006.18	No attempt	8 October 2012	00:35
4	5	CCAFS	SpaceX CRS-2	4,877 kg	LEO	NASA	Success\n	F9 v1.07B0007.18	No attempt\n	1 March 2013	15:10
...	...	...	...	...	...	...	...	...	...	...	...

Github link: jupyter-labs-webscraping.ipynb

Data Collection - Scraping



Github link: jupyter-labs-webscraping.ipynb

Github link: labs-jupyter-spacex-Data wrangling.ipynb

Data Wrangling

To process the data it was really important to first understand the information we were handling; for example, **True Ocean** refered to a succesful landing in the ocean, while **False Ocean** meant unsuccessful; **True RTLS** refered to a successful landing on the ground, while **False RTLS** was unsuccessful; **True ADSD** meant a succesful landing on a drone ship, but **False ASDS** was unsuccesful.

In this case, the mentioned outcomes were converted so that the number **1 means succesfull** and **0 unsuccessfull**

Perform **Exploratory Data Analysis (EDA)** to find patterns in the data and determine what would be the label for training supervised models.

Data processing

- Calculate number of launches for each site
- Calculate number and occurrence of each orbit
- Calculate the number and occurence of mission outcome of the orbits
- Create a landing outcome label from Outcome column

Github link: labs-jupyter-spacex-Data wrangling.ipynb

Data Wrangling

Calculate number of launches for each site

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

```
LaunchSite
CCAFS SLC 40    55
KSC LC 39A      22
VAFB SLC 4E     13
Name: count, dtype: int64
```

Calculate number and occurrence of each orbit

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

```
Orbit
GTO      27
ISS      21
VLEO     14
PO        9
LEO       7
SSO       5
MEO       3
HEO       1
ES-L1     1
SO        1
GEO       1
Name: count, dtype: int64
```

Calculate the number and occurence of mission outcome of the orbits

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

```
Outcome
True ASDS       41
None None       19
True RTLS       14
False ASDS       6
True Ocean       5
False Ocean      2
None ASDS        2
False RTLS       1
Name: count, dtype: int64
```

Create a landing outcome label from Outcome column

```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
landing_class = []
for outcome in df['Outcome']:
    if outcome in bad_outcomes:
        landing_class.append(0)
    else:
        landing_class.append(1)
```

This variable will represent the classification variable that represents the outcome of each launch. If the value is zero, the first stage did not land successfully; one means the first stage landed Successfully

```
df['Class']=landing_class
df[['Class']].head(8)
```

	Class
0	0
1	0
2	0
3	0

```
df.head(5)
```

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Bl
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	

EDA with Data Visualization

The charts plotted for this lab were:

- **Scatter plot** to visualize relationship between FlightNumber and PayloadMass.
- **Scatter plot** to visualize relationship between FlightNumber and LaunchSite.
- **Scatter plot** to visualize relationship between PayloadMass and LaunchSite.
- **Scatter plot** to visualize relationship between FlightNumber and Orbit type.
- **Scatter plot** to visualize relationship between PayloadMass and Orbit type.

→ Helpful to display how the relationship between the two variables changes when one is modified.

- **Bar chart** to visualize relationship between success rate of each orbit type.

→ Helpful to perform a comparison between different values.

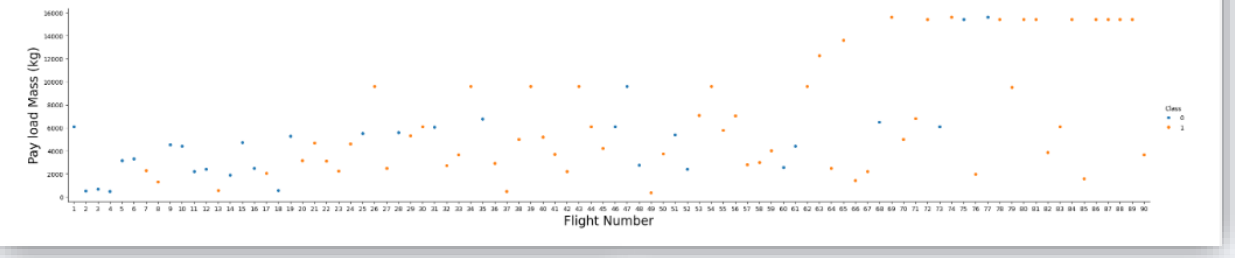
- **Line chart** to visualize Launch success yearly trend.

→ Helpful to show changes in data in a period of time.

Github link: jupyter-labs-eda-sql-coursera_sqllite.ipynb

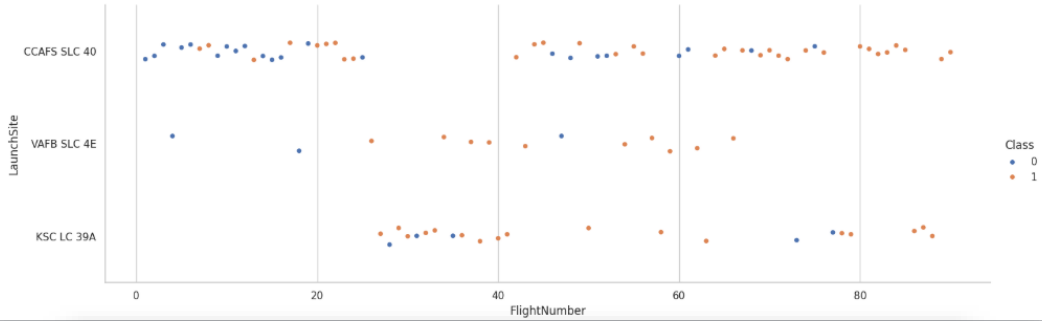
EDA with Data Visualization

```
sns.catplot(y="PayloadMass", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Pay load Mass (kg)",fontsize=20)
plt.show()
```



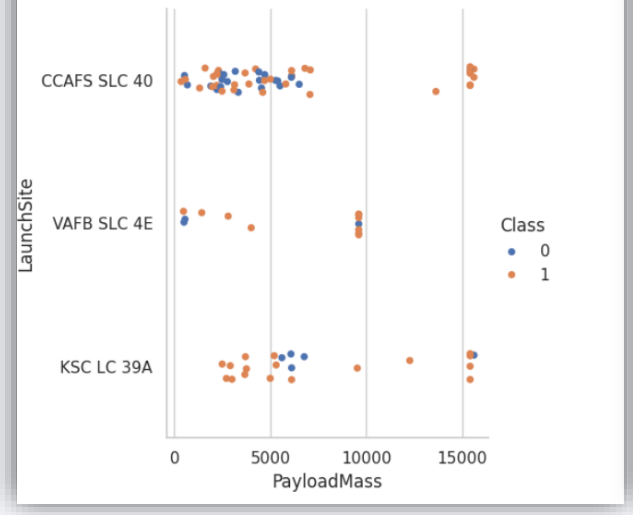

```
# Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to be the class va
sns.catplot(y="LaunchSite", x="FlightNumber",hue='Class',data=df,aspect=3)
```

```
<seaborn.axisgrid.FacetGrid at 0x6baa0d8>
```

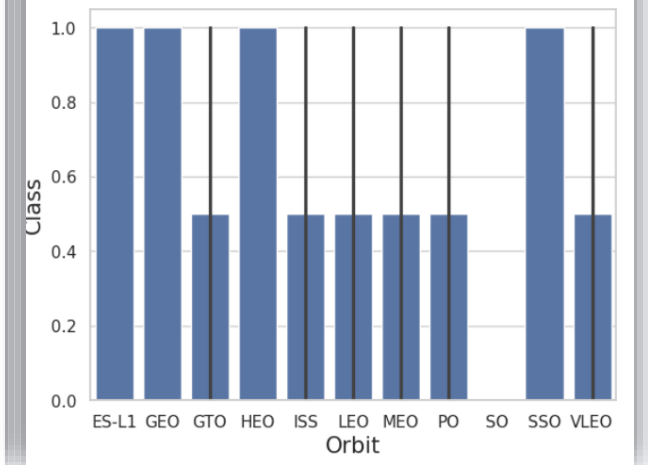



```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to
sns.catplot(y="LaunchSite",x="PayloadMass", hue="Class", data=df)
```

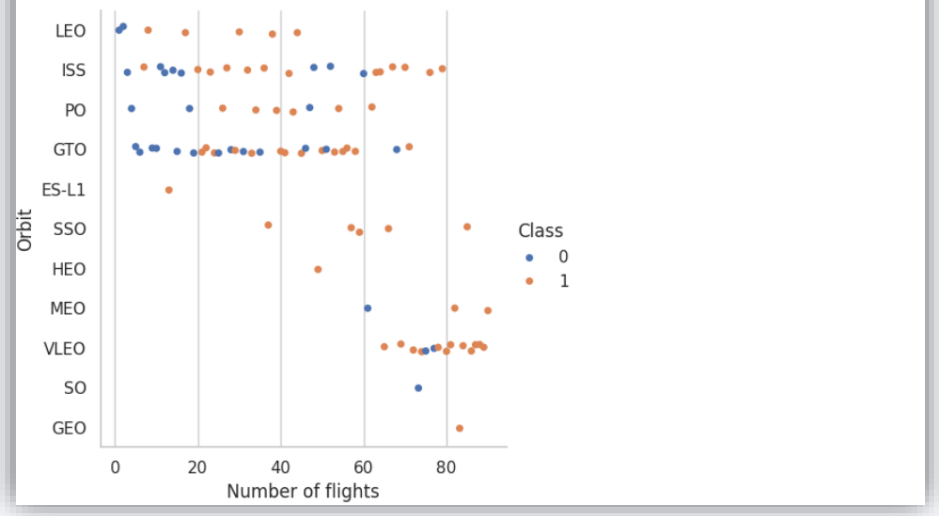
```
<seaborn.axisgrid.FacetGrid at 0xb87f9d0>
```




```
# HINT use groupby method on Orbit column and get the mean of Class column
orbit1 = df.groupby(['Orbit', 'Class'])['Class'].agg(['mean']).reset_index()
sns.barplot(y="Class", x="Orbit", data=orbit1)

plt.xlabel("Orbit",fontsize=15)
plt.ylabel("Class",fontsize=15)
plt.show()
```



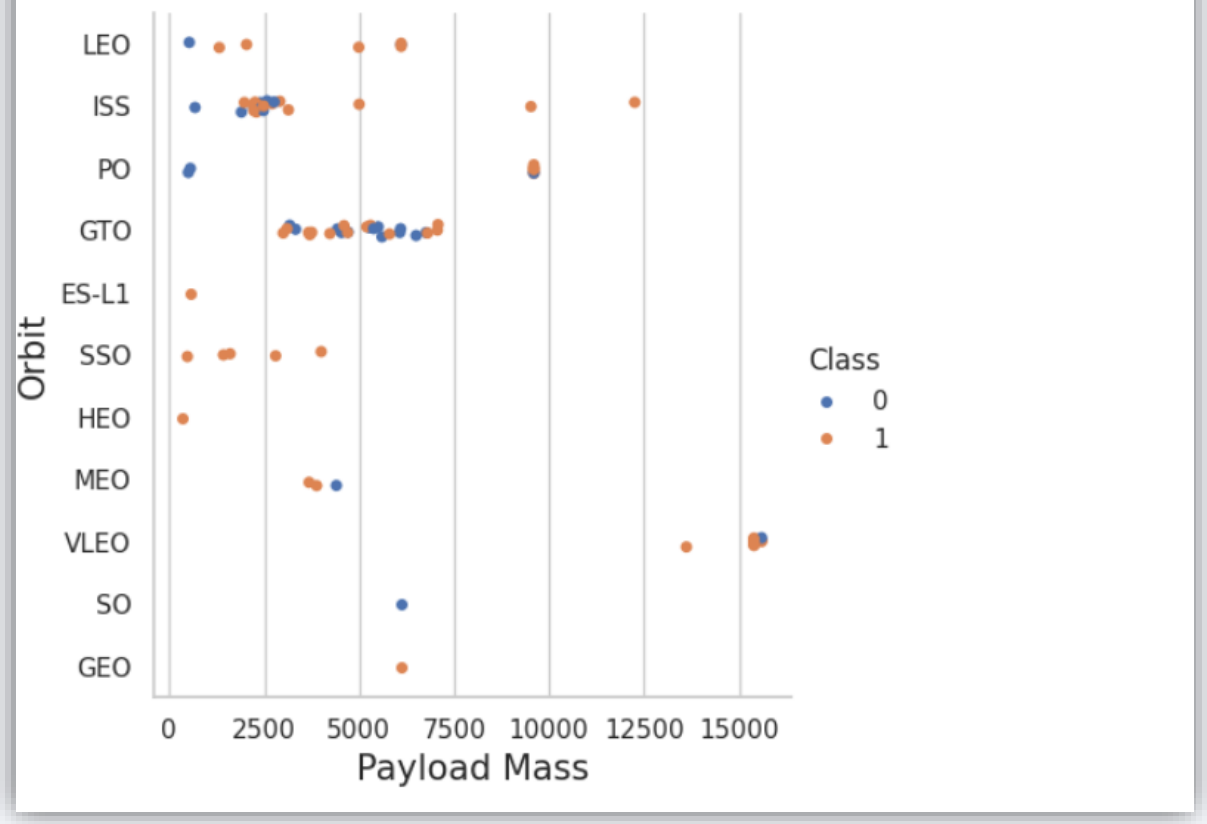

```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class value
sns.catplot(x='FlightNumber',y='Orbit',data=df,hue='Class')
plt.xlabel('Number of flights')
plt.ylabel('Orbit')
plt.show()
```



Github link: [jupyter-labs-eda-sql-coursera_sqllite.ipynb]

EDA with Data Visualization

```
# Plot a scatter point chart with x axis to be Payload Mass and y axis to be the Orbit, and hue t
sns.catplot(y="Orbit", x="PayloadMass", hue="Class", data=df)
plt.xlabel("Payload Mass",fontsize=15)
plt.ylabel("Orbit",fontsize=15)
plt.show()
```




```
df['Date'] = pd.to_datetime(df['Date'])

df['Year'] = df['Date'].dt.year
df1 = df.groupby('Year')['Class'].mean().reset_index()
sns.lineplot(data=df1, x='Year', y='Class')
plt.xlabel("Year", fontsize=15)
plt.ylabel("Average success", fontsize=15)
plt.title("Launch success yearly trend", fontsize=15)

plt.show()
```

Github link: jupyter-labs-eda-sql-coursera_sqllite.ipynb

EDA with SQL

The SQL queries performed were:

- **Displaying the names of the unique launch sites in the space mission:**

```
SELECT DISTINCT LAUNCH_SITE FROM SPACEXTBL;
```

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

```
SELECT LAUNCH_SITE FROM SPACEXTBL WHERE LAUNCH_SITE LIKE 'CCA%' LIMIT 5;
```

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

```
SELECT SUM(PAYLOAD_MASS__KG_) as total_payload_mass FROM SPACEXTBL WHERE Customer = 'NASA (CRS)';
```

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

```
SELECT AVG(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE booster_version = 'F9 v1.1';
```

- **List the date when the first succesful landing outcome in ground pad was achieved:**

```
SELECT MIN(Date) FROM SPACEXTBL WHERE landing_outcome = 'Success (ground pad)';
```

Github link: jupyter-labs-eda-sql-coursera_sqllite.ipynb

Github link: jupyter-labs-eda-sql-coursera_sqllite.ipynb

EDA with SQL

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

```
df_suc_drone_Mass4000_6000=df[(df['Landing_Outcome'] == 'Success (drone ship)') & (df['PAYLOAD_MASS__KG_'] < 6000) & (df['PAYLOAD_MASS__KG_'] > 4000)] booster=df_suc_drone_Mass4000_6000["Booster_Version"].to_list()
```

```
(f"Boosters such as {booster} have masses between 4000 and 6000 and succeded in drone ship ")
```

- **List the total number of successful and failure mission outcomes:**

```
SELECT MISSION_OUTCOME, COUNT(MISSION_OUTCOME) AS TOTAL_NUMBER FROM SPACEXTBL GROUP BY MISSION_OUTCOME;
```

- **List the names of the booster_versions which have carried the maximum payload mass:**

```
SELECT DISTINCT BOOSTER_VERSION FROM SPACEXTBL WHERE PAYLOAD_MASS__KG_ = ( SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTBL);
```

- **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:**

```
SELECT substr(Date, 6, 2) as Month, Mission_Outcome, Booster_Version, Launch_Site FROM SPACEXTBL WHERE substr(Date, 1, 4) = '2015';
```

- **Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order:**

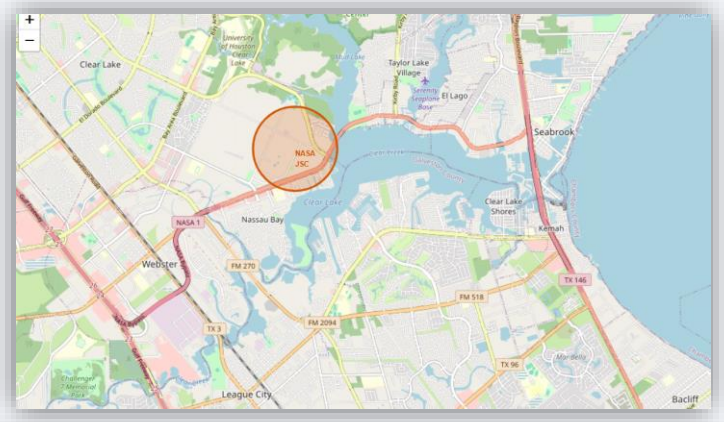
```
SELECT LANDING_OUTCOME, COUNT(LANDING_OUTCOME) AS TOTAL_NUMBER FROM SPACEXTBL WHERE
BETWEEN '2010-06-04' AND '2017-03-20' GROUP BY LANDING_OUTCOME ORDER BY TOTAL_NUMBER DESC
```

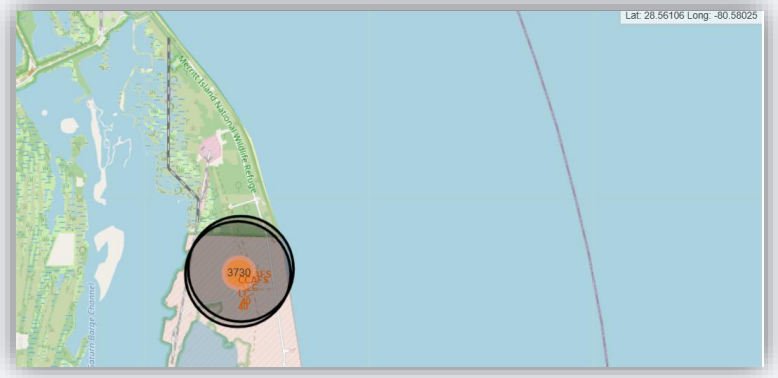
Github link: lab_jupyter_launch_site_location (2).ipynb

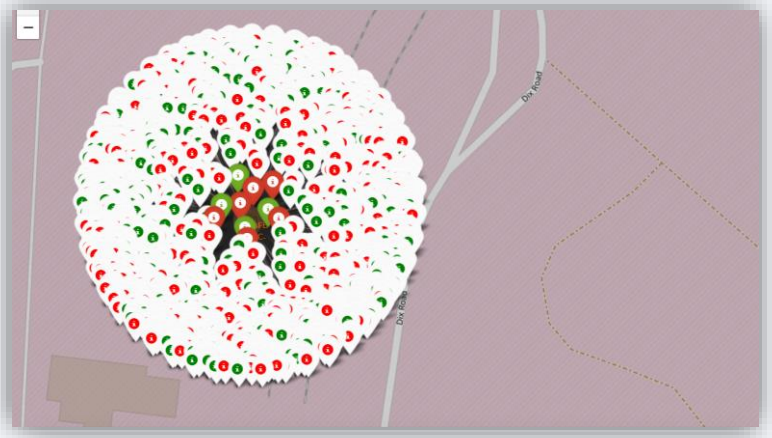
Build an Interactive Map with Folium

Map objects created and added to folium map:

- Used **folium.Circle** and **folium.Marker** to highlight an area and add a text label on a specific coordinate.
- Used **marker_cluster** to add multiple markers to the map at the same coordinate.
- Used **MousePosition** to get the coordinates of a point of the map where the mouse is.
- Used **folium.PolyLine** to create a line connecting to sites like the nearest railway, highway and coastline.





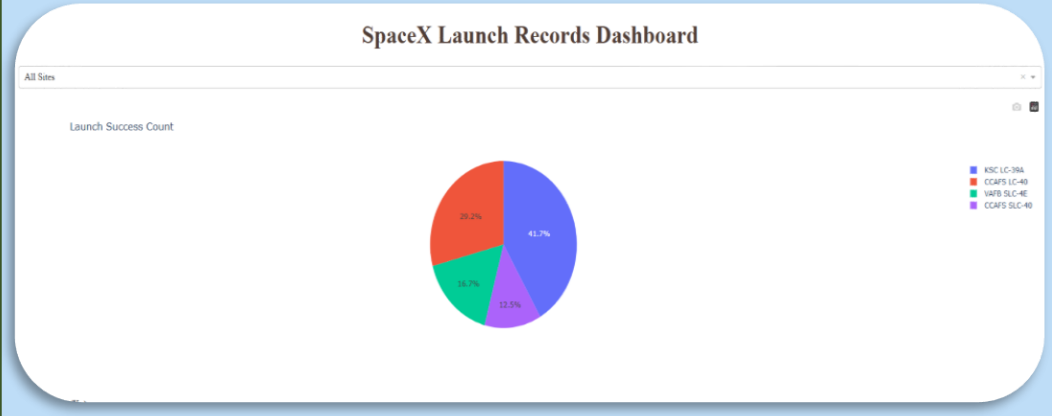


Github link: spacex_dash_app (2).py

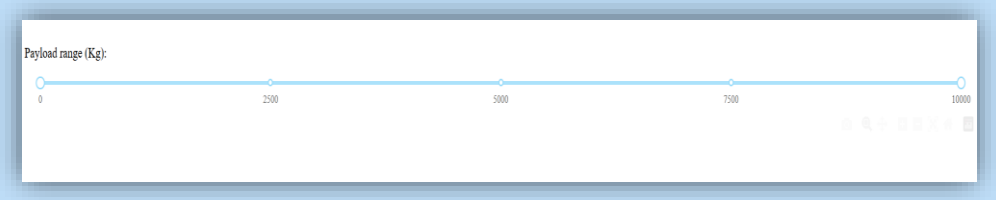
Build a Dashboard with Plotly Dash

The dashboard created for the project contains graphs and plots such as:

Dropdown Input component with the objective of allowing to select the desired launch site



Range slider to select the payload

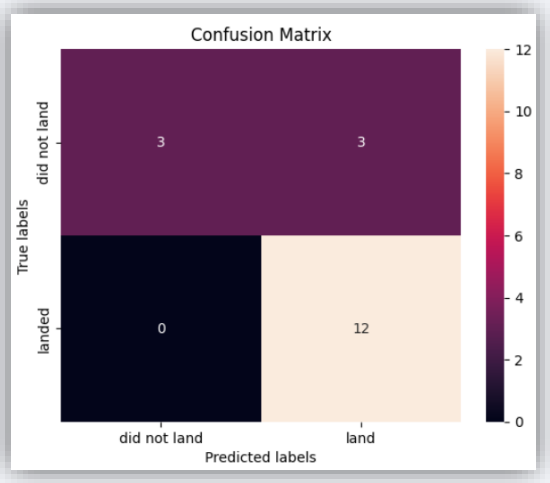


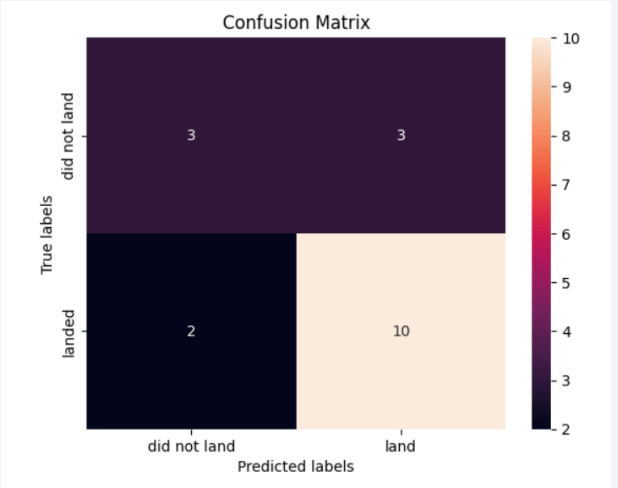
Callback function to generate a **Pie chart** based on the dropdown selection, this to visualize the launch success count

Callback function to show a **Scatter plot** to visualize how the payload connects with the mision outcome

Predictive Analysis (Classification)

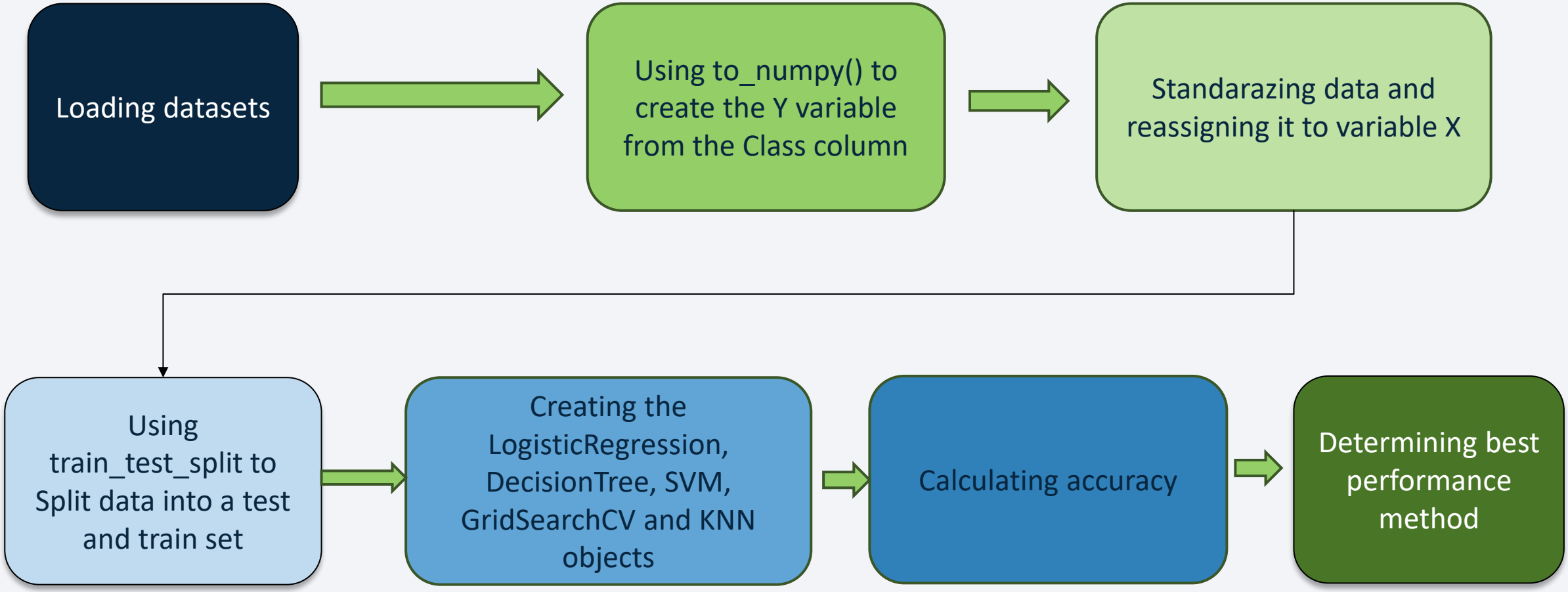
- Created a **NumPy array** from the column Class in data.
- **Standarized** the data.
- Using **train_test_split** to split the data X and Y into training and test data.
- Create a **logistic regression** object then create a **GridSearchCV** object.
- **Calculating accuracy** on the test set.
- Creating a **support vector machine** object then creating a **GridSearchCV** object.
- Creating a **decision tree classifier** object and a **GridSearchCV** object.
- Calculating the **accuracy of tree _cv** on the test data.
- Creating a **K nearest neighbors** object and a **GridSearchCV** object.
- Calculating the **accuracy of knn_cv** on the test data.
- **Finding the method that performs the best.**





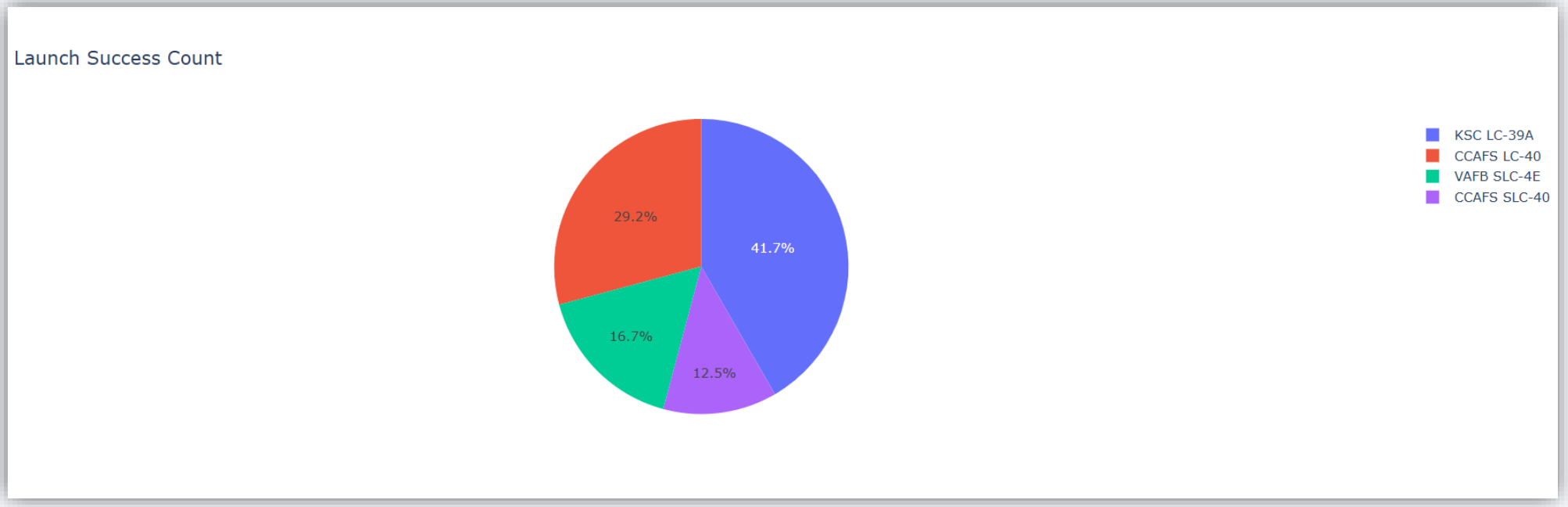
Github link: SpaceX_Machine Learning Prediction_Part_5.ipynb

Predictive Analysis (Classification)



Github link: SpaceX_Machine Learning Prediction_Part_5.ipynb

Results

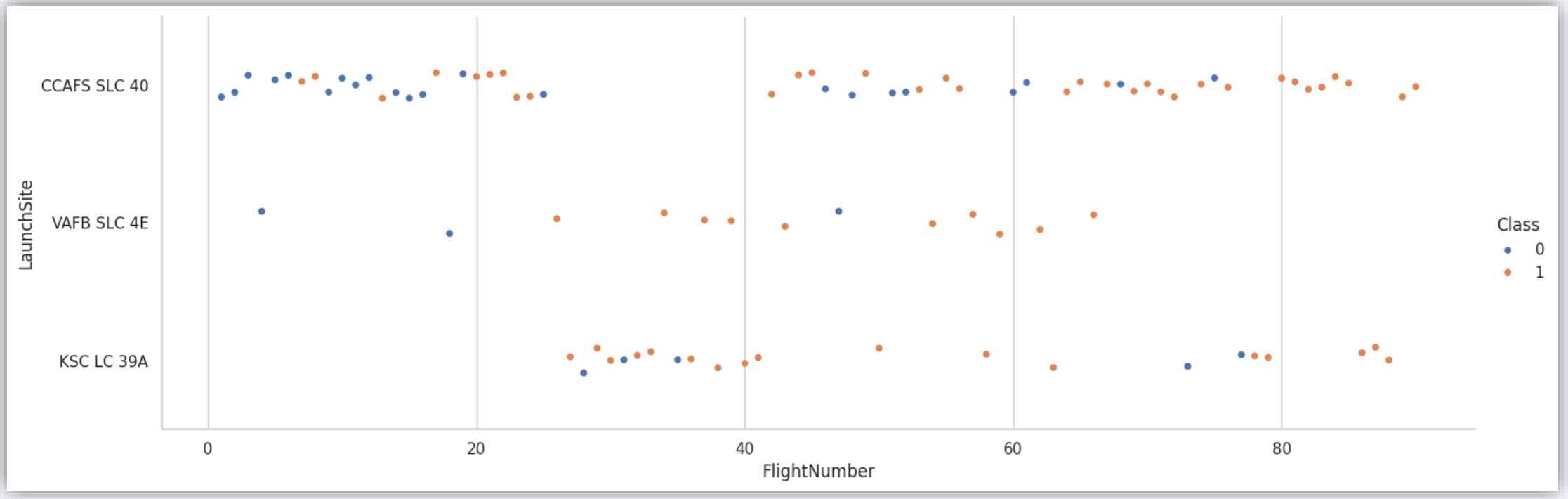


- The models with the highest success count were KSC LC-39A, CCAFS LC-40 and VAFB SLC 4-E, while the one with the lowest count was CCAFS SLC-40.

Section 2

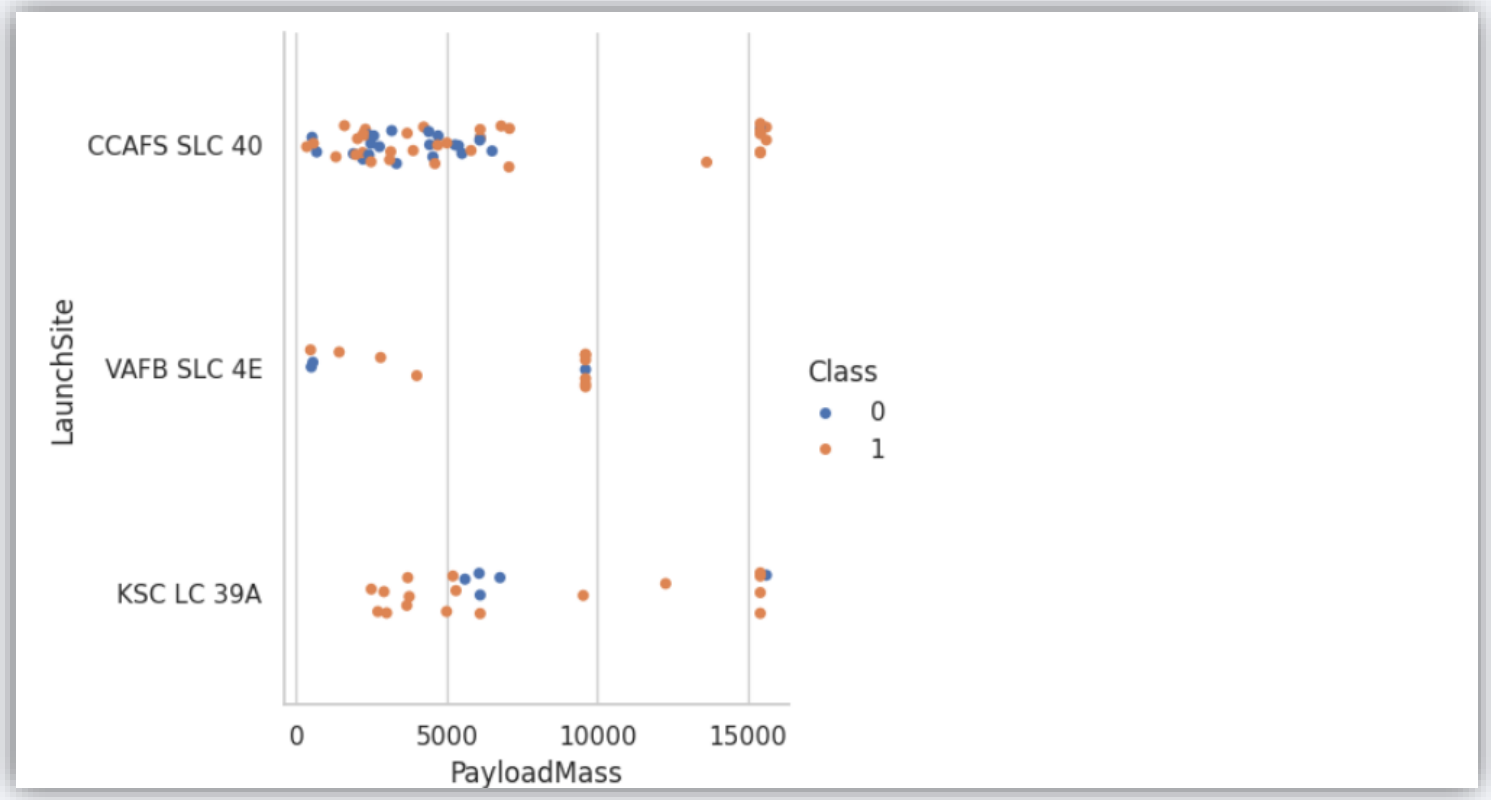
Insights drawn from EDA

Flight Number vs. Launch Site



We can observe that CCAFS SLC 40 was probably the principal launch site at the beginning, experiencing multiple fails. There is a clear gap in which we can observe a higher success around the KSC LC 39A region, and the VAFB SLC 4E; from this point there seems to be more success. So, compared to beginning of the plot, going back to CCAFS SLC 40 at the end shows a higher success.

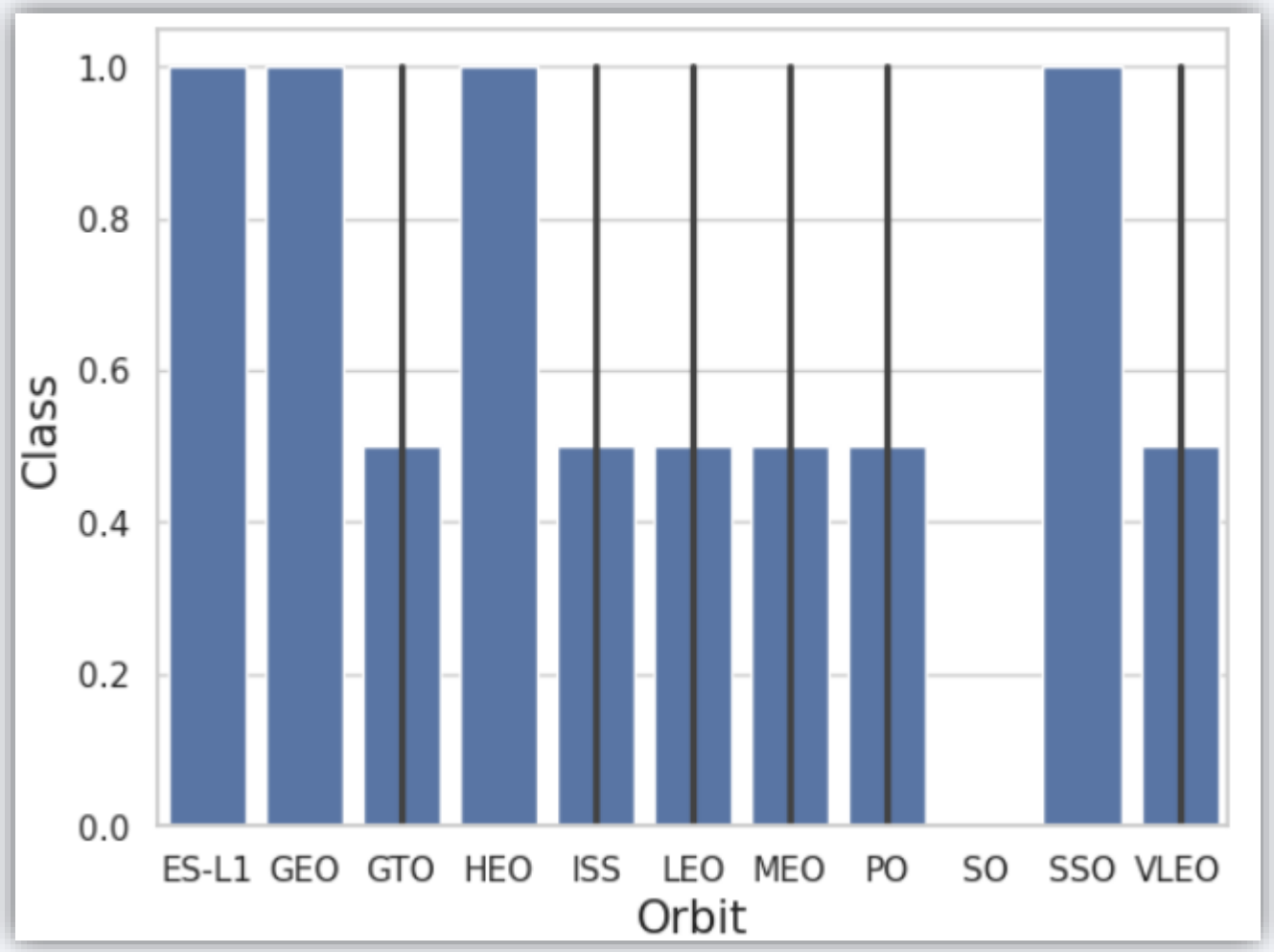
Payload vs. Launch Site



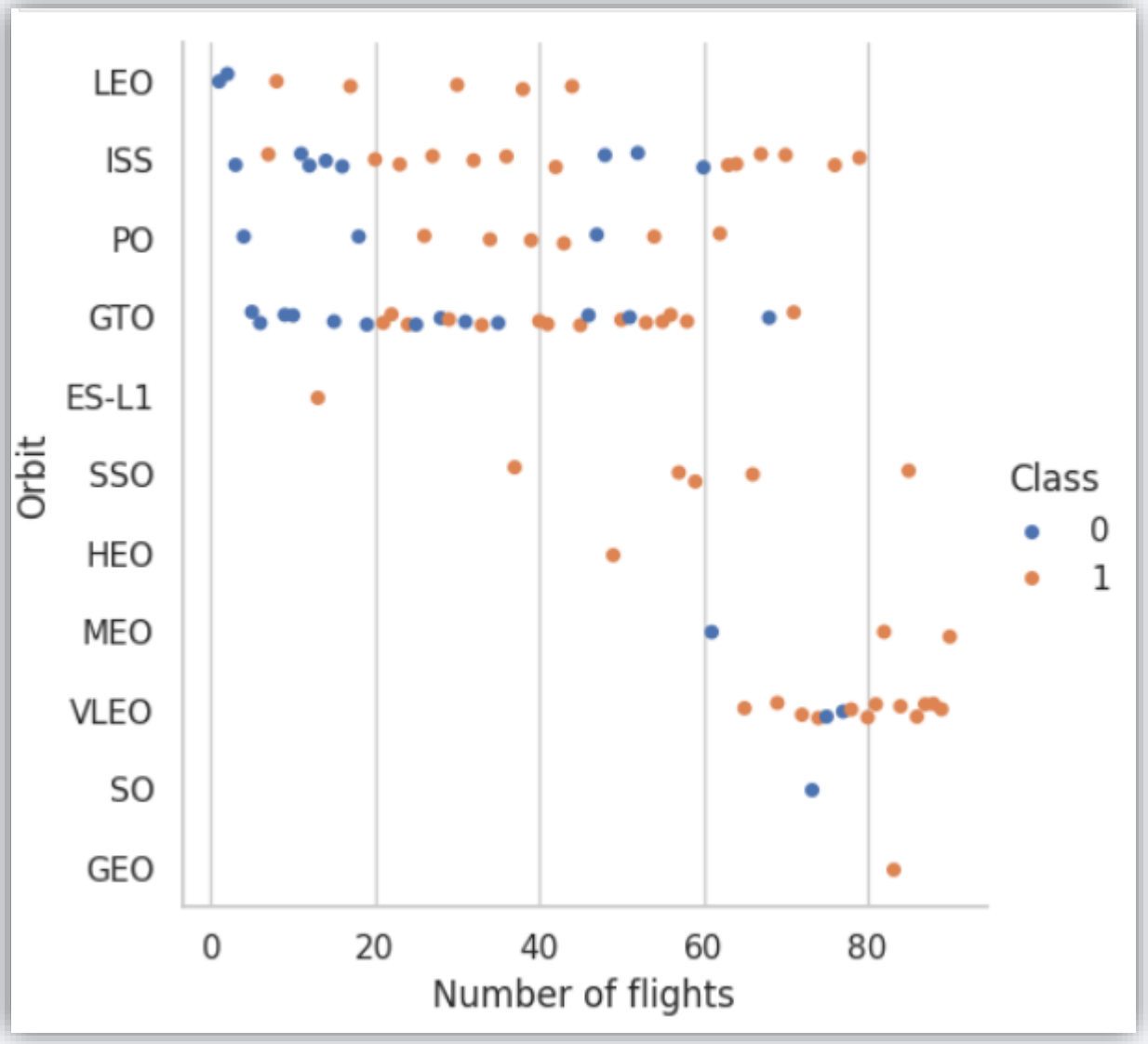
We can observe that there is relatively mixed outcome with the payload masses; visibly, we can observe a higher accumulation of launches on CCAFS SLC 40, and the second one being KSC LC 39A. The biggest concentration of launches sits between a payload of 0 to aproximately 7000 mainly forr CCAFS and KSC, while VAFB remains below 5000

Success Rate vs. Orbit Type

Notably we can observe that orbit types ES-L1, GEO, HEO and SSO have the highest success rates when compared to other orbit types; on the other hand, it seems that SO has no success.



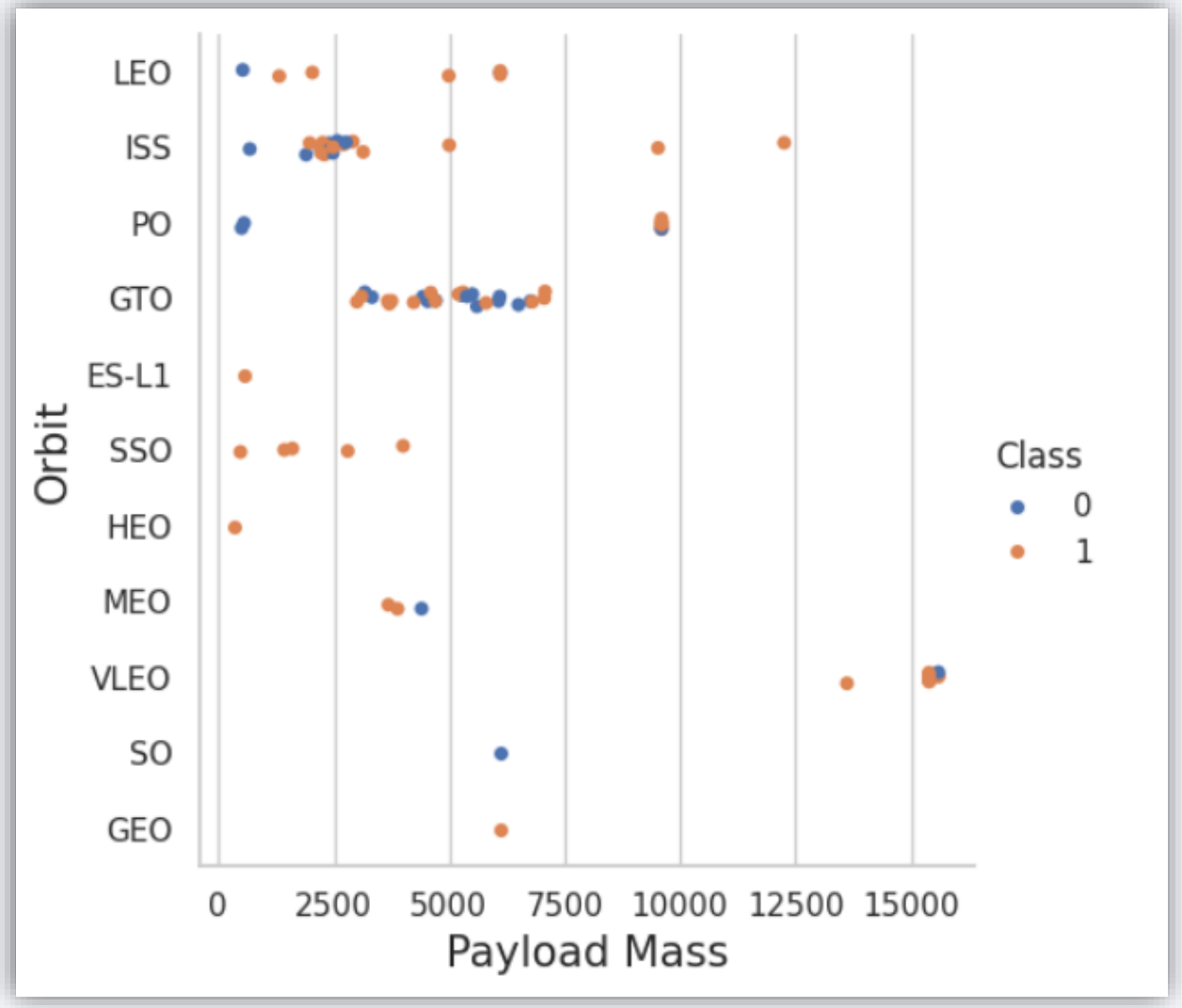
Flight Number vs. Orbit Type



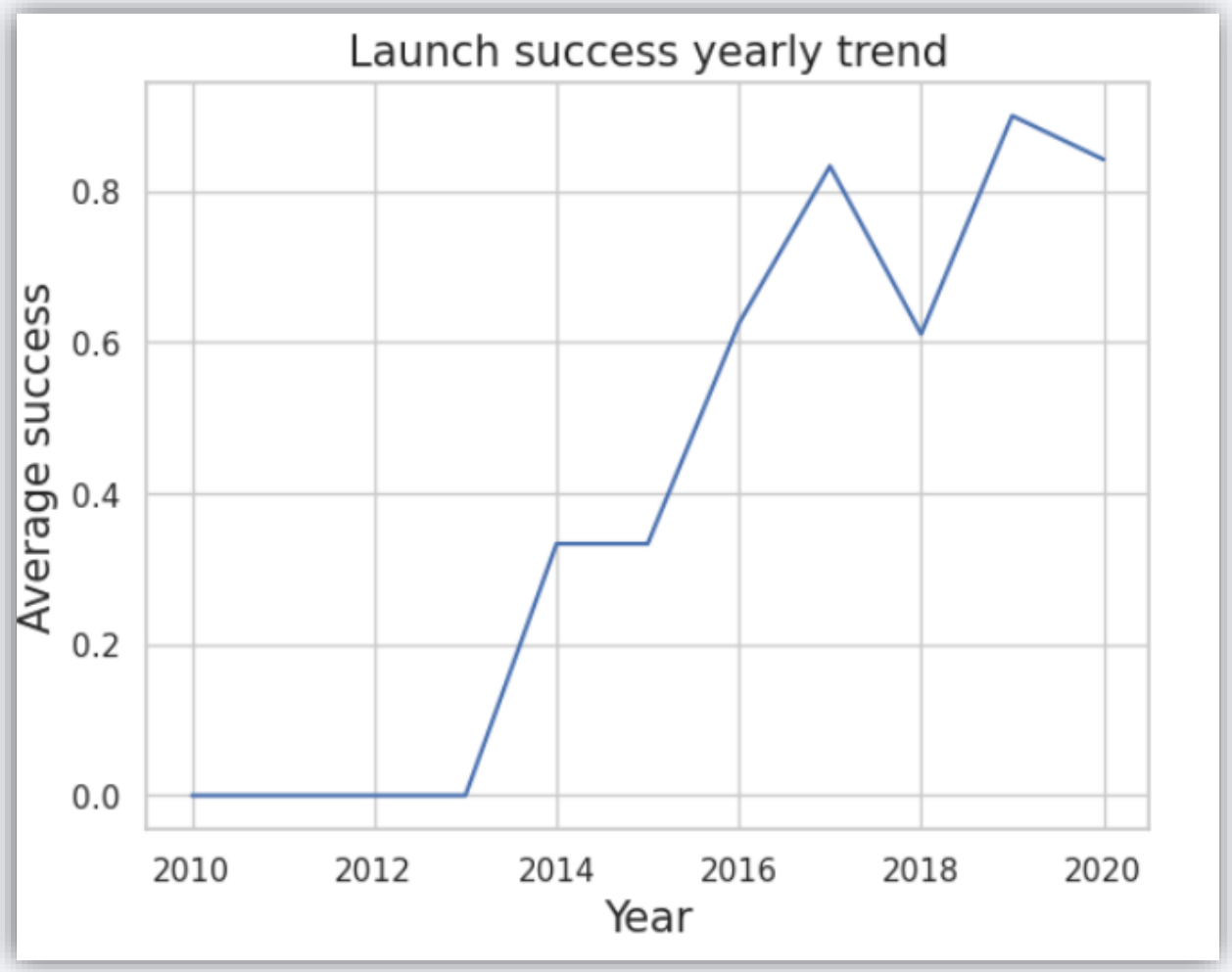
We can observe that at first LEO, ISS, PO and GTO possessed the most amount of flights reaching to a maximum of 80 aproximately; however, it seems that overtime VLEO was beeing used more, while the past popular ones were dropped. ES-L1, SSO, HEO, MEO, SO and GEO have a mínimum amount of launches.

Payload vs. Orbit Type

It looks like landings with heavy payloads were more successful with PO, LEO and ISS, being aproximately between 2500 and 7500.
For GTO its more difficult to determine, because both succesfull and unsuccesfull outcomes are very present.



Launch Success Yearly Trend



There is a visible success rate increase from the year 2013 to 2020 reaching beyond 0.8; during that period there was a slight decrease to 0.6 in 2018.

All Launch Site Names

```
%%sql
SELECT DISTINCT LAUNCH_SITE
FROM SPACEXTBL;
```

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

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

Used SQL query to find the distinct launchsite names.
Using the DISTINCT statement allowed to find said values.

Launch Site Names Begin with 'CCA'

Executed an SQL query to find the values; used the WHERE and LIKE statements to specify the conditions we were aiming for.

```
%%sql
SELECT LAUNCH_SITE
FROM SPACEXTBL
WHERE LAUNCH_SITE LIKE 'CCA%'
LIMIT 5;
```

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

Launch_Site
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40

Total Payload Mass

```
%%sql
SELECT SUM(PAYLOAD_MASS__KG_) as total_payload_mass
FROM SPACEXTBL
WHERE Customer = 'NASA (CRS)';
```

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

total_payload_mass
45596

Used an SQL query to find the value; using the SUM, AS and WHERE statements allowed to narrow the data and receive the wanted value.

Average Payload Mass by F9 v1.1

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

```
%%sql
SELECT AVG(PAYLOAD_MASS__KG_)
FROM SPACEXTBL
WHERE booster_version = 'F9 v1.1';
```

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

AVG(PAYLOAD_MASS__KG_)
2928.4

Performed an SQL query; using the AVG and WHERE statements allowed to get the average of payload mass from F9 v1.1

First Successful Ground Landing Date

```
%%sql
SELECT MIN(Date)
FROM SPACEXTBL
WHERE landing_outcome = 'Success (ground pad)';
```

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

MIN(Date)
2015-12-22

Executed an SQL query and used the MIN and WHERE statements to get the earliest date of a successfull landing.

Successful Drone Ship Landing with Payload between 4000 and 6000

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

```
df_suc_drone_Mass4000_6000=df[(df['Landing_Outcome'] == 'Success (drone ship)') & (df['PAYLOAD_MASS__KG_'] < 6000) & (df['PA
booster=df_suc_drone_Mass4000_6000["Booster_Version"].to_list()
print(f"Boosters such as {booster} have masses between 4000 and 6000 and succeded in drone ship ")
```

```
Boosters such as ['F9 FT B1022', 'F9 FT B1026', 'F9 FT  B1021.2', 'F9 FT  B1031.2'] have masses between 4000 and 6000 and suc
ceded in drone ship
```

Retrieved the names of the boosters with successfull landings in a drone ship and possessed payload mass greater tan 4000 but les tan 6000

Total Number of Successful and Failure Mission Outcomes

```
%%sql
SELECT MISSION_OUTCOME,
COUNT(MISSION_OUTCOME) AS TOTAL_NUMBER
FROM SPACEXTBL
GROUP BY MISSION_OUTCOME;
```

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

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

Used the COUNT, AS and GROUP BY statements to retrieve the total number of successful anf failure misión outcomes

Boosters Carried Maximum Payload

```
%%sql
SELECT DISTINCT BOOSTER_VERSION
FROM SPACEXTBL
WHERE PAYLOAD_MASS__KG_ = (
    SELECT MAX(PAYLOAD_MASS__KG_)
    FROM SPACEXTBL);
```

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

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

Performed an SQL query; used the DISTINCT, WHERE and SELECT MAX statements to retrieve the correct boosters

2015 Launch Records

Used an SQL query and executed the SELECT, AS MONTH and WHERE statements to get the 2015 launch records

```
%%sql
SELECT substr(Date, 6, 2) as Month,
       Mission_Outcome,
       Booster_Version,
       Launch_Site
FROM SPACEXTBL
WHERE substr(Date, 1, 4) = '2015';
```

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

Month	Mission_Outcome	Booster_Version	Launch_Site
01	Success	F9 v1.1 B1012	CCAFS LC-40
02	Success	F9 v1.1 B1013	CCAFS LC-40
03	Success	F9 v1.1 B1014	CCAFS LC-40
04	Success	F9 v1.1 B1015	CCAFS LC-40
04	Success	F9 v1.1 B1016	CCAFS LC-40
06	Failure (in flight)	F9 v1.1 B1018	CCAFS LC-40
12	Success	F9 FT B1019	CCAFS LC-40

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

```
%%sql
SELECT LANDING_OUTCOME, COUNT(LANDING_OUTCOME) AS TOTAL_NUMBER
FROM SPACEXTBL
WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY LANDING_OUTCOME
ORDER BY TOTAL_NUMBER DESC
```

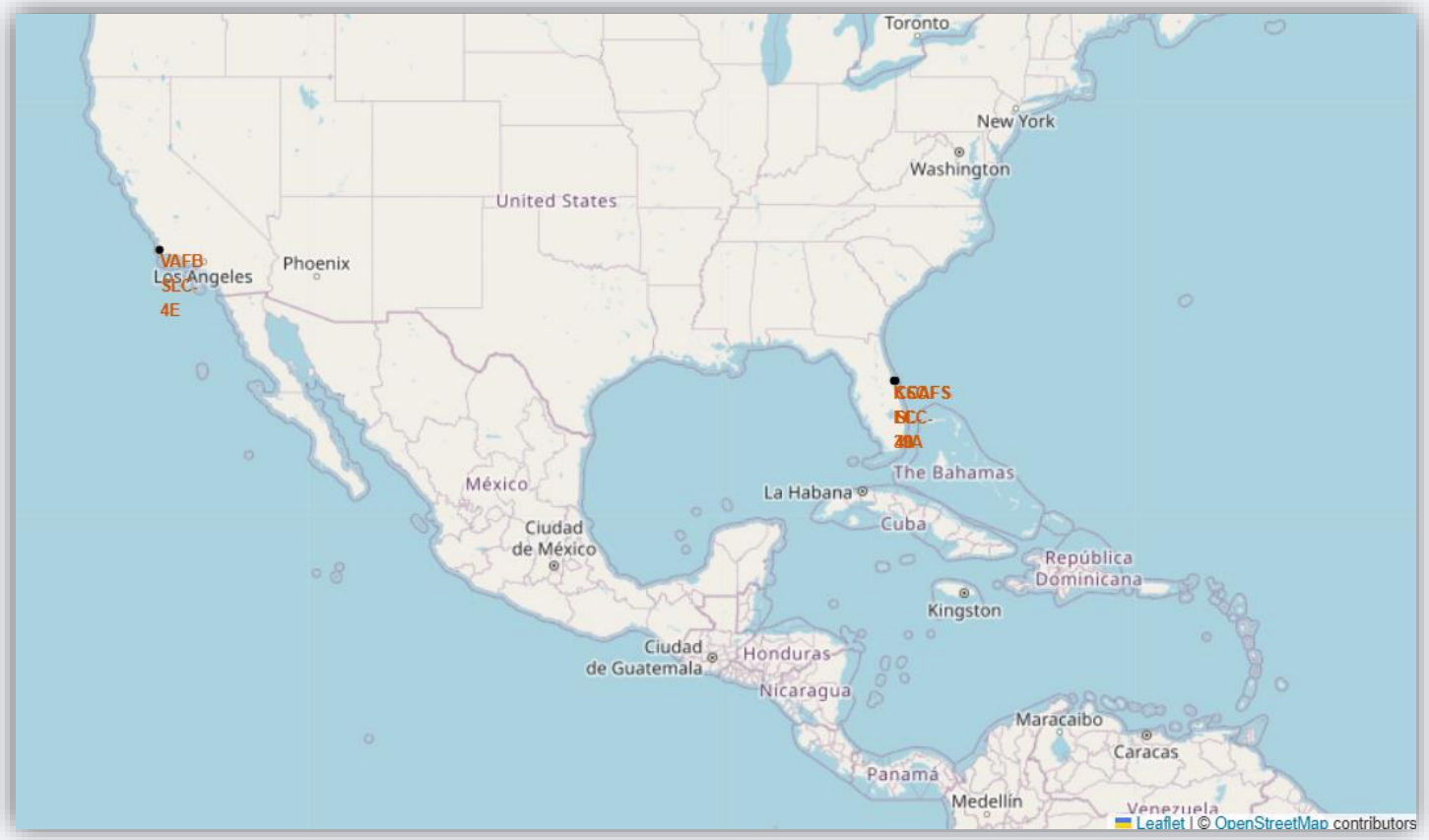
```
 * sqlite:///my_data1.db
Done.
```

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

Performed an SQL query and used the SELECT, COUNT, AS, WHERE, BETWEEN, GROUP BY, ORDER BY and DESC statements

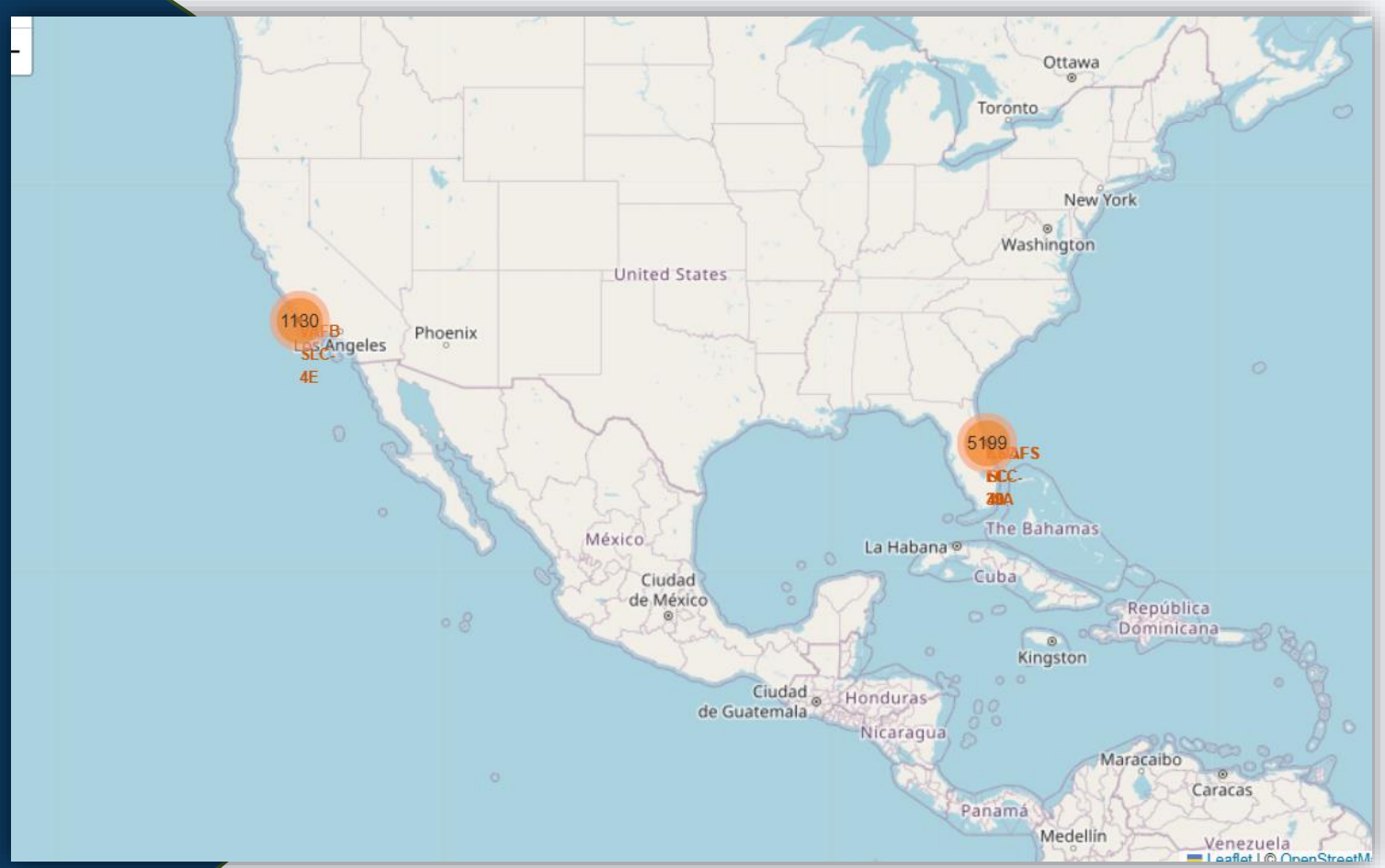

Section 3
Launch Sites
Proximities Analysis

Map of all launch sites

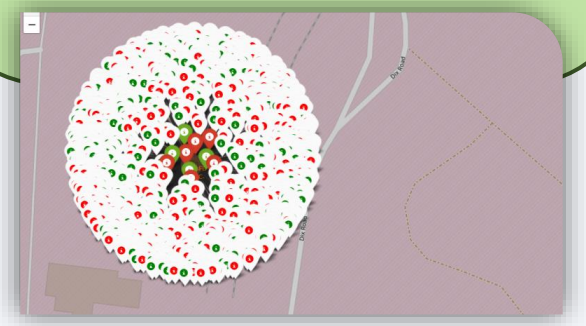


The launch sites are situated near coastlines and located in more secluded areas, far from big cities. Each launch site is marked with a text label

Map of successful and failed launches per site

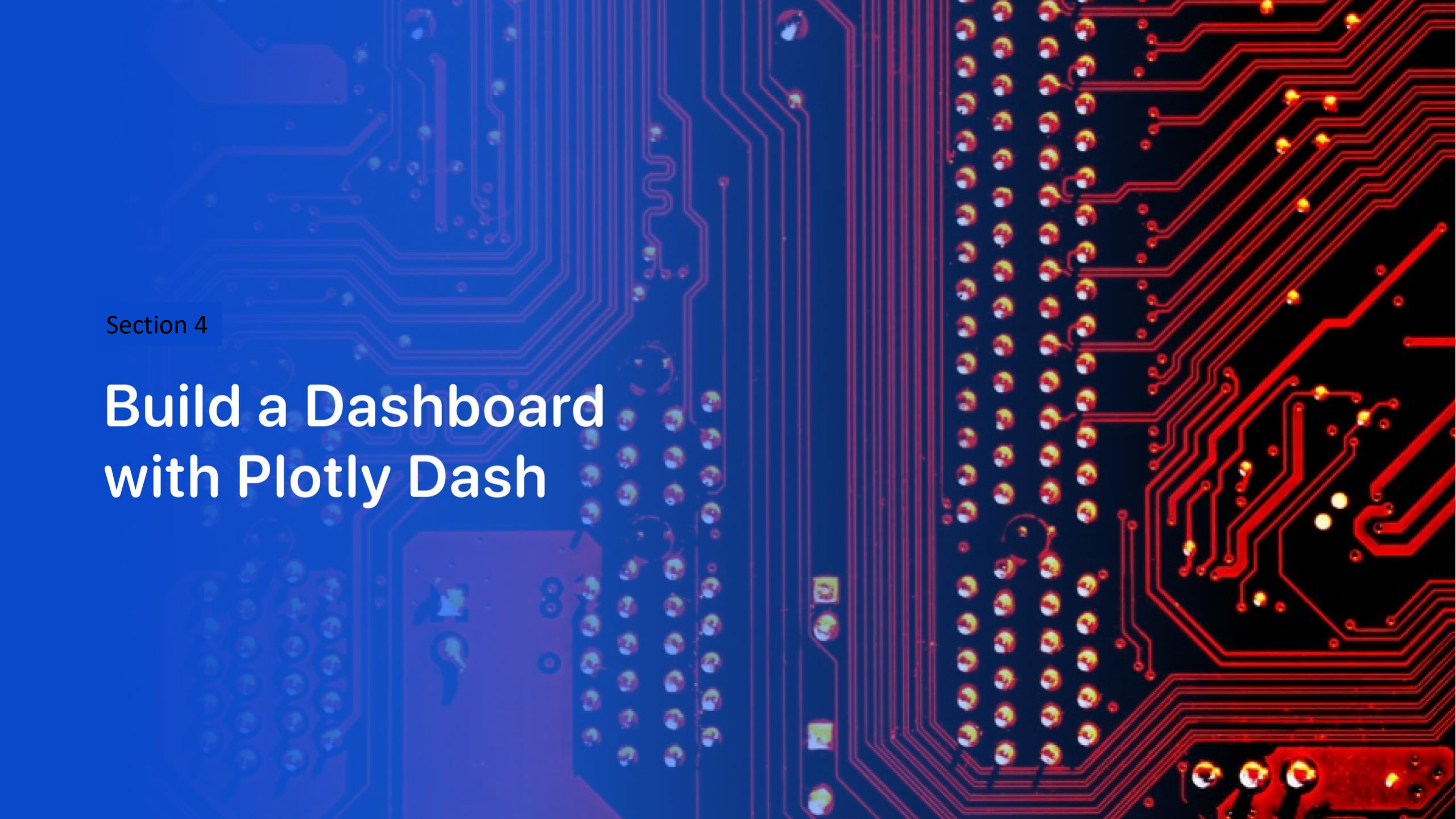


The launch sites are highlighted and possess red and green markers that indicate if the launch was successful or failed.

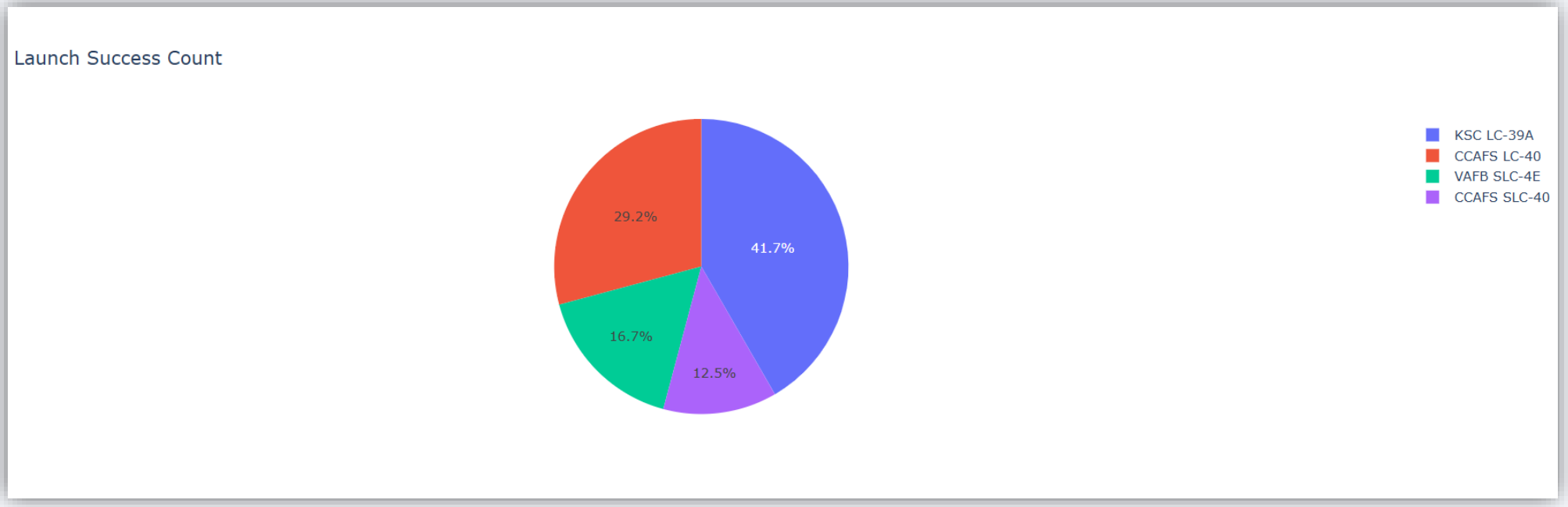


Map of areas near the launch sites

The map shows the distance between the nearest city, coastline and railway; PolyLines were used to visualize the distance, and said distance is included

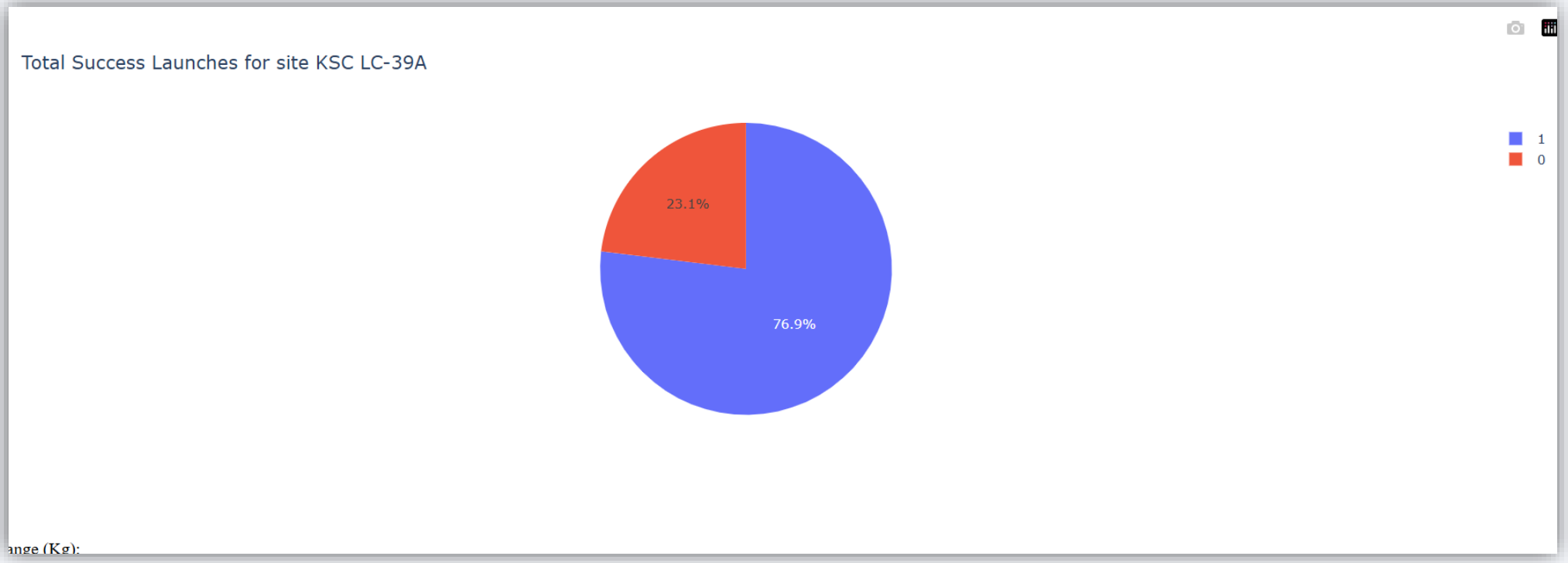

Section 4
Build a Dashboard with Plotly Dash

Pie chart of launch success for all sites



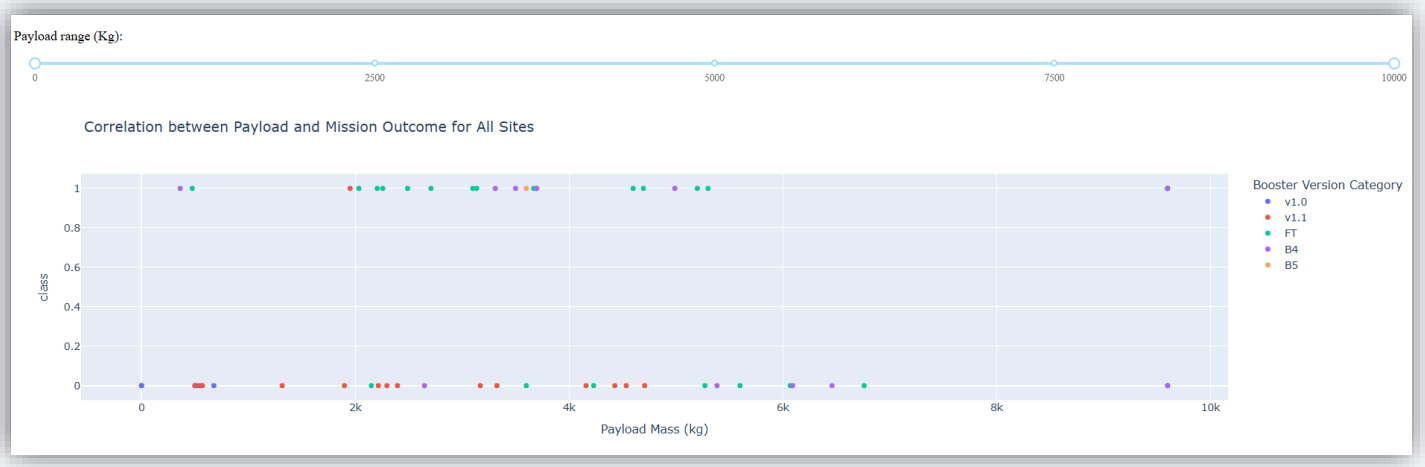
The most successful launch sites were KSC LC-39A, CCAFS LC-40 and VAFB SLC-4E; the model with the least success was CCAFS SLC-40.

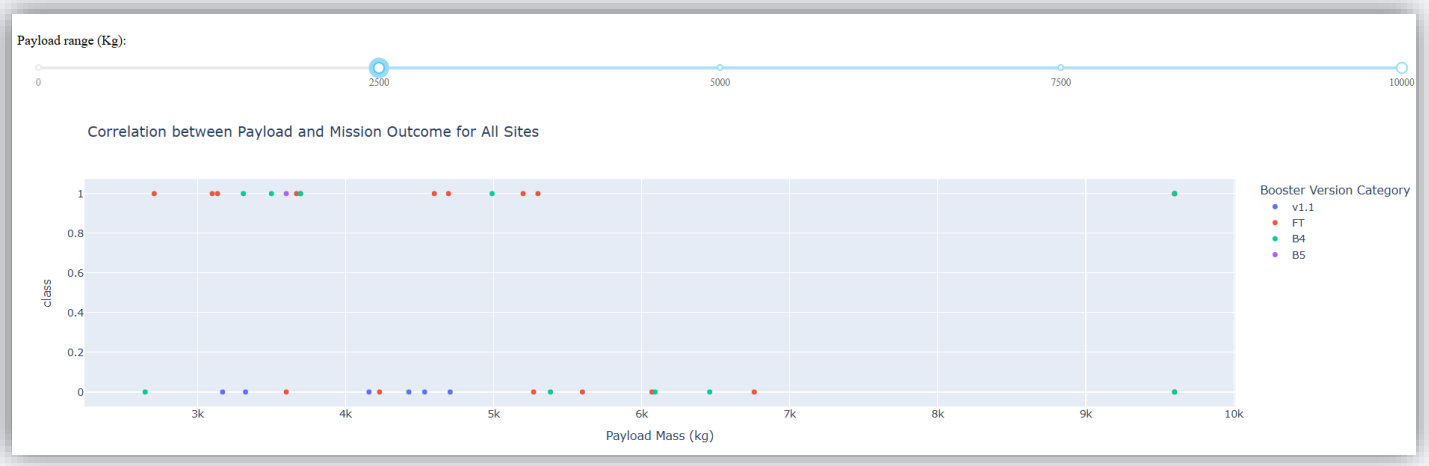
Pie chart of launch site with highest launch success ratio



KSC LC-39A was the launch site with the highest success ratio among all the other sites, having a 23.1% of failure an 76.9% success, and also a 41.7% of overall success when compared to other sites

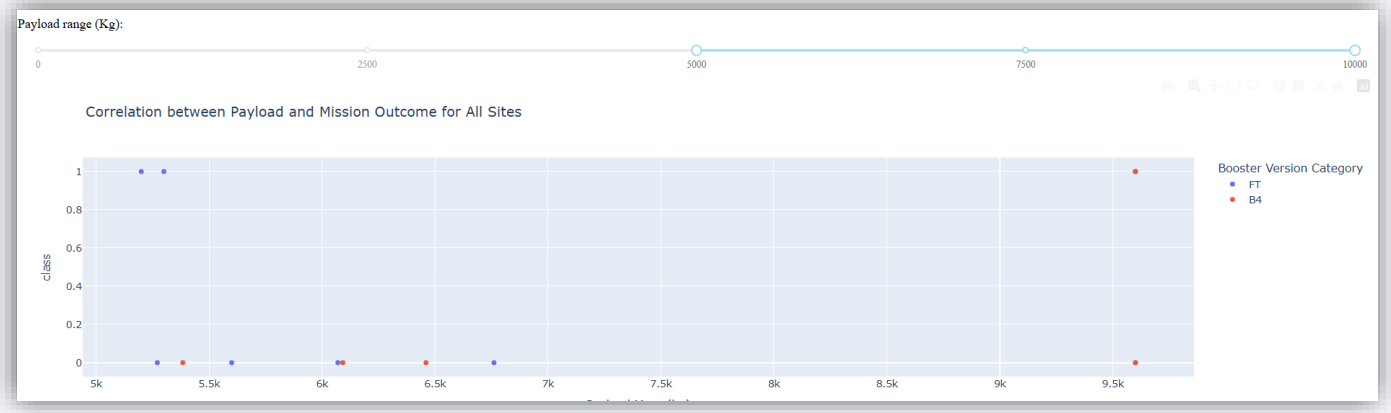
Scatter plot of Payload and Launch Outcome for all sites

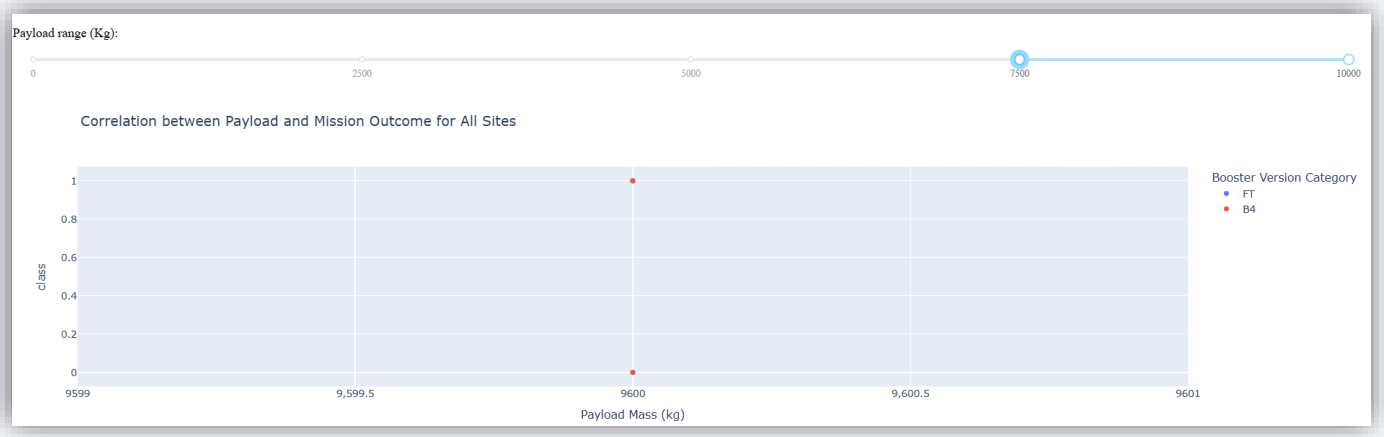




We can see that the payload range with the most successful launches is between 2000kg and 4000kg

Scatter plot of Payload and Launch Outcome for all sites



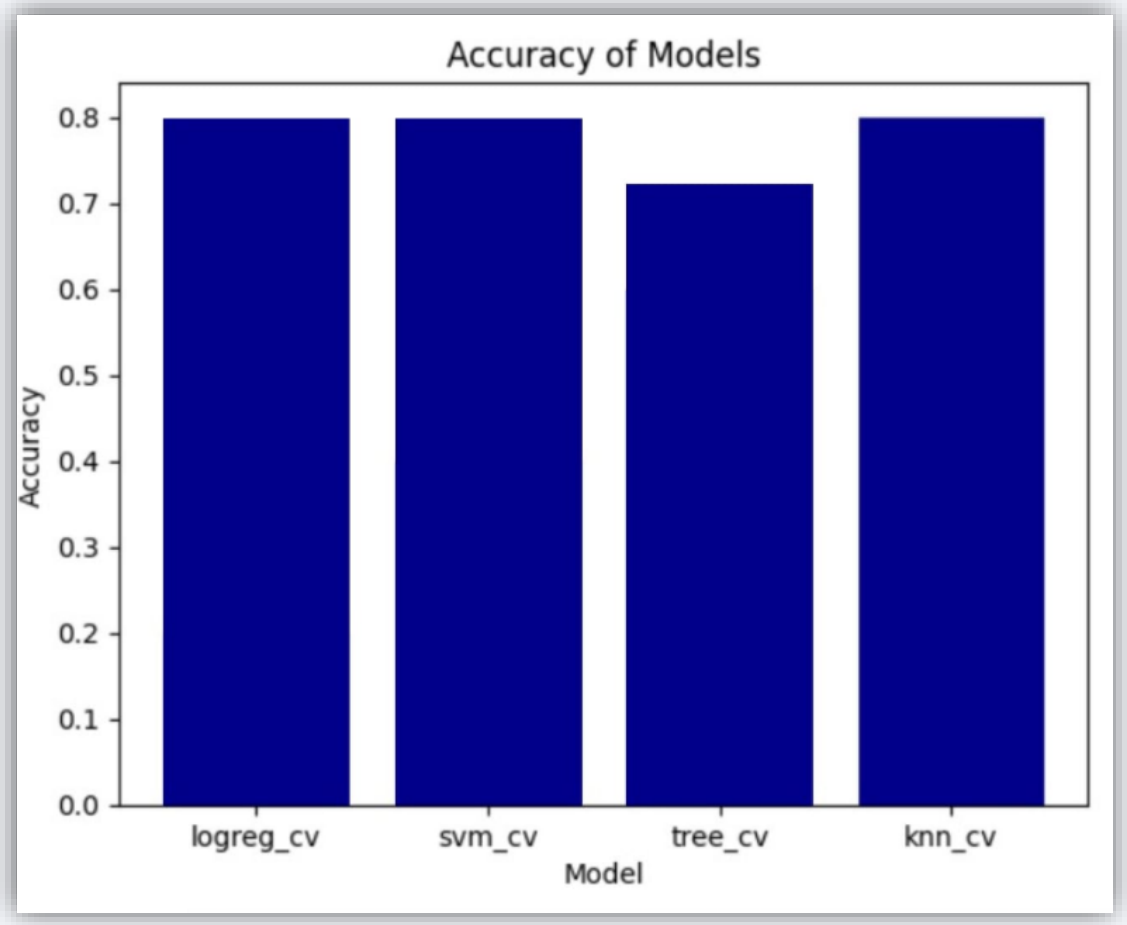


On the other hand, heavier payloads **don't** have a lot of successful launches.

Section 5

Predictive Analysis (Classification)

Classification Accuracy



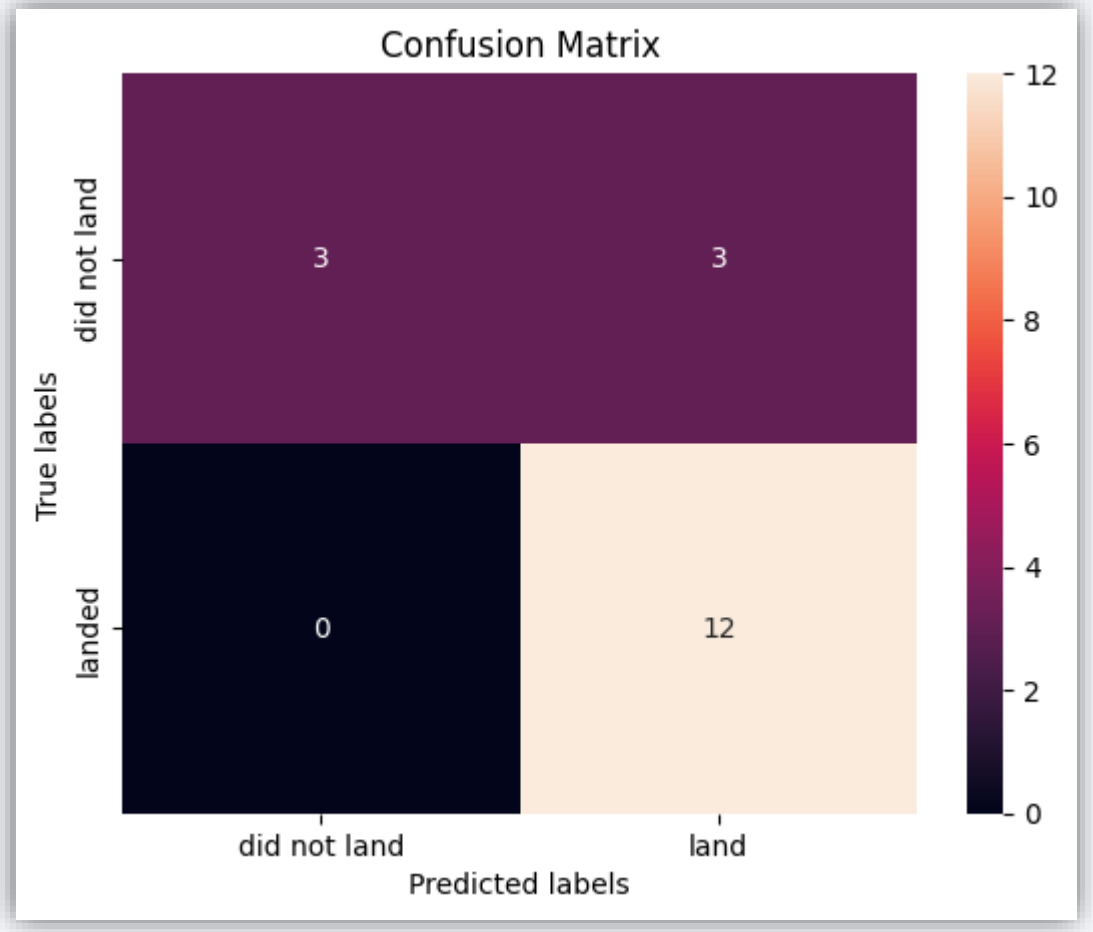
Based on the bar chart, almost all the models displayed the same accuracy of 0.83, except for the decision tree which had a lower accuracy of 0.72

```
print('Accuracy for Logistics Regression method:', logreg_cv.score(X_test, Y_test))
print( 'Accuracy for Support Vector Machine method:', svm_cv.score(X_test, Y_test))
print('Accuracy for Decision tree method:', tree_cv.score(X_test, Y_test))
print('Accuracy for K nearsdt neighbors method:', knn_cv.score(X_test, Y_test))
```

```
Accuracy for Logistics Regression method: 0.8333333333333334
Accuracy for Support Vector Machine method: 0.8333333333333334
Accuracy for Decision tree method: 0.7222222222222222
Accuracy for K nearsdt neighbors method: 0.8333333333333334
```

Confusion Matrix

Since almost all models displayed the same accuracy, the same confusion matrix was generated for logreg_cv, knn_cv and vsm_cv, said models displayed the highest accuracy.




```
print('Accuracy for Logistics Regression method:', logreg_cv.score(X_test, Y_test))
print( 'Accuracy for Support Vector Machine method:', svm_cv.score(X_test, Y_test))
print('Accuracy for Decision tree method:', tree_cv.score(X_test, Y_test))
print('Accuracy for K nearsdt neighbors method:', knn_cv.score(X_test, Y_test))
```

```
Accuracy for Logistics Regression method: 0.8333333333333334
Accuracy for Support Vector Machine method: 0.8333333333333334
Accuracy for Decision tree method: 0.7222222222222222
Accuracy for K nearsdt neighbors method: 0.8333333333333334
```

Conclusions

- The site KSC LC-39A has the highest launch success ratio among all the other sites.
- Launches that possess light payloads seem to perform better than launches with heavier payloads.
- Orbit types ES-L1, GEO, HEO and SSO have the highest success rates when compared to other orbit types.
- The models that had the better performance were logreg_cv, knn_cv and vsm_cv.

Appendix

Full notebooks can be found on GitHub:

❖ https://github.com/Ponce011/Data-Science.git

Thank you!