IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Oswaldo Felizzola
10 April 2025

Outline

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

Introduction

- Project background and context
  - SpaceX markets its Falcon 9 rocket launches at a cost of $62 million. This is considerably lower than the $165 million and higher prices quoted by other providers, a difference primarily attributed to SpaceX's ability to reuse the first stage.
  - Consequently, the feasibility of the first stage landing directly influences the launch cost. This information is pertinent for any competing company seeking to bid against SpaceX for a rocket launch contract.
- Problems you want to find answers
  - The intent is to predict whether a Falcon 9 first stage will achieve a successful landing.
  - Find out who SpaceX's main customer is
  - What is the performance of the different rocket launchers?
  - Learn who SpaceX's main customer is.
  - How the different rocket launchers perform.
  - Learn about the evolution of SpaceX's experience.

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

- Describe how data sets were collected.
  - The initial phase of data acquisition utilized the SpaceX API through a get request. This was achieved by first establishing a set of helper functions designed to extract information based on identification numbers within the launch data, subsequently followed by a request for rocket launch data from the designated SpaceX API URL.
  - Additionally, we performed web scraping on the Wikipedia page "List of Falcon 9 and Falcon Heavy launches" to gather historical Falcon 9 launch data. We employed the requests library to retrieve the HTML content, and BeautifulSoup for parsing the relevant table, which was then converted into a Pandas DataFrame.
  - To achieve greater consistency in the requested JSON results, SpaceX launch data was retrieved and parsed using a GET request. Subsequently, the response content was decoded as JSON and then converted into a Pandas DataFrame.

Data Collection – SpaceX API

- In addition to the API data, we collected historical Falcon 9 launch records by web scraping the "List of Falcon 9 and Falcon Heavy launches" Wikipedia page. The launch records on this page are in HTML table format. We used the BeautifulSoup and requests libraries to extract and parse this table, ultimately converting it into a Pandas DataFrame.
- Data acquisition involved utilizing the SpaceX API, a RESTful interface, through a GET request. The SpaceX launch data obtained was then parsed, and the JSON response was decoded and structured as a Pandas DataFrame.
- GitHub URL of the completed notebook: (https://github.com/oswaldofelizzola/Coursera-IBM-Data-Science-Professional/blob/fdc3e5636e43081ca45e5065b47b2332f3a7eaab/Applied%20Data%20Science%20Capstone/1%20jupyter-labs-spacex-data-collection-api.ipynb)

Data Collection – SpaceX API

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:

In [9]:
```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-Sl
```

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

In [10]:
```
response=requests.get(static_json_url)
```

In [11]:
```
response.status_code
```

Out[11]:
```
200
```

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

In [12]:
```
# Use json_normalize meethod to convert the json result into a dataframe
data = pd.json_normalize(response.json())
```

Data Collection – SpaceX API

In [14]:

```
# Lets take a subset of our dataframe keeping only the features we want and the flight number, and
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 boo
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 a
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
data['date'] = pd.to_datetime(data['date_utc']).dt.date

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

Data Collection – SpaceX API

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`.

In [25]:
```
# Hint data['BoosterVersion']!='Falcon 1'
filt =  df['BoosterVersion']!=  'Falcon 1'
data_falcon9 = df.loc[filt]
data_falcon9.head()
```

Out[25]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reuse
4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	Fals
5	8	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False	Fals
6	10	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False	Fals
7	11	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False	Fals
8	12	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	False	Fals

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

In [26]:
```
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9
```

Data Collection – SpaceX API

Data Wrangling

We can see below that some of the rows are missing values in our dataset.

In [27]:
```
data_falcon9.isnull().sum()
```

Out[27]:
```
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
```

Before we can continue we must deal with these missing values. The `LandingPad` column will retain None values to represent when landing pads were not used.

Data Collection – SpaceX API

Task 3: Dealing with Missing Values

Calculate below the mean for the `PayloadMass` using the `.mean()`. Then use the mean and the `.replace()` function to replace `np.nan` values in the data with the mean you calculated.

In [28]:

```
# Calculate the mean value of PayloadMass column

plm_mean = data_falcon9['PayloadMass'].mean()

# Replace the np.nan values with its mean value

data_falcon9['PayloadMass'].replace(np.nan, plm_mean, inplace=True)
```

Data Collection - Scraping

- To gather historical Falcon 9 launch records, we employed the BeautifulSoup and requests libraries to scrape the relevant HTML table from Wikipedia and then parsed this data into a DataFrame.
- GitHub URL of the completed notebook: (https://github.com/oswaldofelizzola/Coursera-IBM-Data-Science-Professional/blob/fdc3e5636e43081ca45e5065b47b2332f3a7eaab/Applied%20Data%20Science%20Capstone/2%20jupyter-labs-webscraping.ipynb)

Data Collection - Scraping

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 [5]:
```
# use requests.get() method with the provided static_url
# assign the response to a object
response = requests.get(static_url)
```

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

In [6]:
```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(response.content, 'html.parser')
```

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

In [7]:
```
# Use soup.title attribute
soup.title
```

Out[7]:
```
<title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

Data Collection – Scraping (cont)

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

In [8]:
```
# 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.

In [9]:
```
# Let's print the third table and check its content
first_launch_table = html_tables[2]
print(first_launch_table)
```

In [10]:
```
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 co

table = first_launch_table.find_all('th')
for row in table:
    name = extract_column_from_header(row)
    if name is not None and len(name) > 0:
        column_names.append(name)
```

Check the extracted column names

In [11]:
```
print(column_names)
```

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

Data Collection – Scraping (cont)

TASK 3: Create a data frame by parsing the launch HTML tables

We will create an empty dictionary with keys from the extracted column names in the previous task. Later, this dictionary will be converted into a Pandas dataframe

In [12]:

```
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']=[]
```

Data Collection – Scraping (cont)

In [13]:

```
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.`
            #print(flight_number)
            datatimelist=date_time(row[0])

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

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

            # Booster version
            # TODO: Append the bv into launch_dict with key `Version Booster`
            bv=booster_version(row[1])
            if not(bv):
                bv=row[1].a.string
            print(bv)
```

Data Collection – Scraping (cont)

```
# Launch Site
# TODO: Append the bv into launch_dict with key `Launch Site`
launch_site = row[2].a.string
#print(launch_site)

# Payload
# TODO: Append the payload into launch_dict with key `Payload`
payload = row[3].a.string
#print(payload)

# Payload Mass
# TODO: Append the payload_mass into launch_dict with key `Payload mass`
payload_mass = get_mass(row[4])
#print(payload)

# Orbit
# TODO: Append the orbit into launch_dict with key `Orbit`
orbit = row[5].a.string
#print(orbit)

# Customer
# TODO: Append the customer into launch_dict with key `Customer`
customer = row[6].a.string
#print(customer)

# Launch outcome
# TODO: Append the launch_outcome into launch_dict with key `Launch outcome`
launch_outcome = list(row[7].strings)[0]
#print(launch_outcome)

# Booster landing
# TODO: Append the launch_outcome into launch_dict with key `Booster landing`
booster_landing = landing_status(row[8])
#print(booster_landing)
```

Data Wrangling

- Following the acquisition of data and its organization into a Pandas DataFrame, we filtered the dataset using the 'BoosterVersion' column to retain only records pertaining to Falcon 9 launches. Subsequently, we managed missing data in the 'LandingPad' and 'PayloadMass' columns, imputing the mean value for the latter.
- Furthermore, Exploratory Data Analysis (EDA) was performed to discern underlying patterns within the data and to establish the target variable for supervised model training.
- GitHub URL of the completed notebook: (https://github.com/oswaldofelizzola/Coursera-IBM-Data-Science-Professional/blob/fdc3e5636e43081ca45e5065b47b2332f3a7eaab/Applied%20Data%20Science%20Capstone/3%20labs-jupyter-spacex-Data%20wrangling.ipynb)

Data Wrangling

TASK 1: Calculate the number of launches on each site

The data contains several Space X launch facilities: Cape Canaveral Space Launch Complex 40 **VAFB SLC 4E** , Vandenberg Air Force Base Space Launch Complex 4E **(SLC-4E)**, Kennedy Space Center Launch Complex 39A **KSC LC 39A** .The location of each Launch Is placed in the column `LaunchSite`

Next, let's see the number of launches for each site.

Use the method `value_counts()` on the column `LaunchSite` to determine the number of launches on each site:

In [6]:
```
# Apply value_counts() on column LaunchSite
df['LaunchSite'].value_counts()
```

Out[6]:
```
LaunchSite
CCAFS SLC 40    55
KSC LC 39A      22
VAFB SLC 4E     13
Name: count, dtype: int64
```

Data Wrangling

TASK 2: Calculate the number and occurrence of each orbit

Use the method `.value_counts()` to determine the number and occurrence of each orbit in the column `Orbit`

In [7]:
```
# Apply value_counts on Orbit column
df['Orbit'].value_counts()
```

Out[7]:
```
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
```

Data Wrangling

TASK 3: Calculate the number and occurence of mission outcome of the orbits

Use the method `.value_counts()` on the column `Outcome` to determine the number of `landing_outcomes`.Then assign it to a variable landing_outcomes.

In [10]:
```
# landing_outcomes = values on Outcome column
landing_outcomes = df['Outcome'].value_counts()
landing_outcomes
```

Out[10]:
```
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
```

Data Wrangling

In [11]:
```
for i,outcome in enumerate(landing_outcomes.keys()):
    print(i,outcome)
```

```
0 True ASDS
1 None None
2 True RTLS
3 False ASDS
4 True Ocean
5 False Ocean
6 None ASDS
7 False RTLS
```

We create a set of outcomes where the second stage did not land successfully:

In [12]:
```
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
bad_outcomes
```

Out[12]:
```
{'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

Data Wrangling

TASK 4: Create a landing outcome label from Outcome column

Using the `Outcome`, create a list where the element is zero if the corresponding row in `Outcome` is in the set `bad_outcome`; otherwise, it's one. Then assign it to the variable `landing_class`:

In [15]:
```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
landing_class = df['Outcome'].apply(lambda x: 0 if x in bad_outcomes else 1)
landing_class.value_counts()
```

Out[15]:
```
Outcome
1    60
0    30
Name: count, dtype: int64
```

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

Data Wrangling

In [16]:
```
df['Class']=landing_class
df[['Class']].head(8)
```

Out[16]:

	Class
0	0
1	0
2	0
3	0
4	0
5	0
6	1
7	1

Data Wrangling

In [17]:
```
df.head(5)
```

Out[17]:

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

We can use the following line of code to determine the success rate:

In [18]:
```
df["Class"].mean()
```

Out[18]:
```
np.float64(0.6666666666666666)
```

EDA with Data Visualization

- The data was analyzed and features were engineered through the application of Pandas and Matplotlib.
  - Exploratory Data Analysis
  - Preparing Data Feature Engineering
- The relationships between Flight Number and Launch Site, Payload and Launch Site, Flight Number and Orbit type, and Payload and Orbit type were visualized through the creation of scatter plots.
- A bar chart was employed to illustrate the success rate associated with each orbit type.
- A line plot was employed to illustrate the yearly trend in launch success rates.

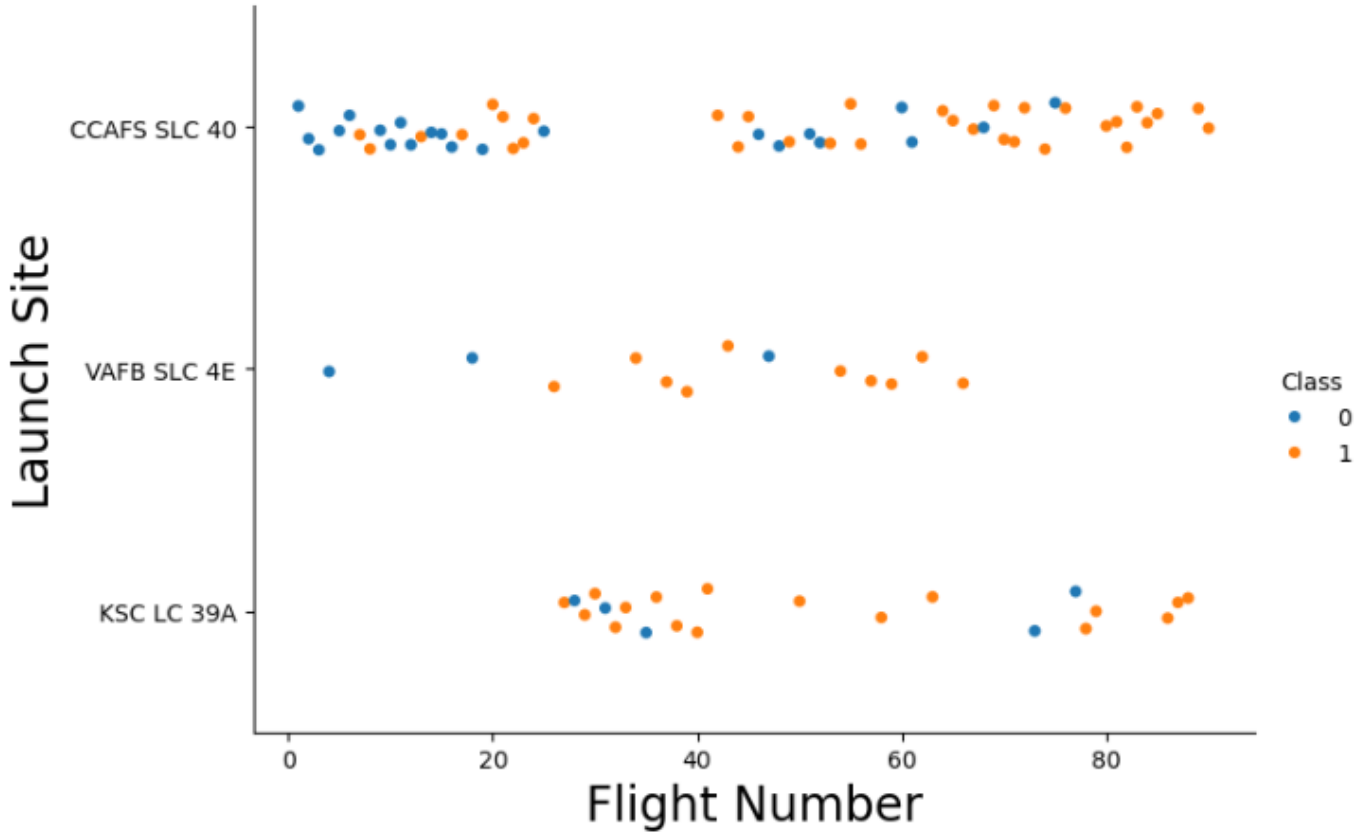
EDA with Data Visualization

- GitHub URL of the completed notebook: (https://github.com/oswaldofelizzola/Coursera-IBM-Data-Science-Professional/blob/fdc3e5636e43081ca45e5065b47b2332f3a7eaab/Applied%20Data%20Science%20Capstone/5%20edadataviz.ipynb)

EDA with Data Visualization

In [5]:

```
# Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and
sns.catplot(y="LaunchSite", x="FlightNumber", hue="Class", data=df, aspect = 1.5 ,height=5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Launch Site",fontsize=20)
plt.show()
```



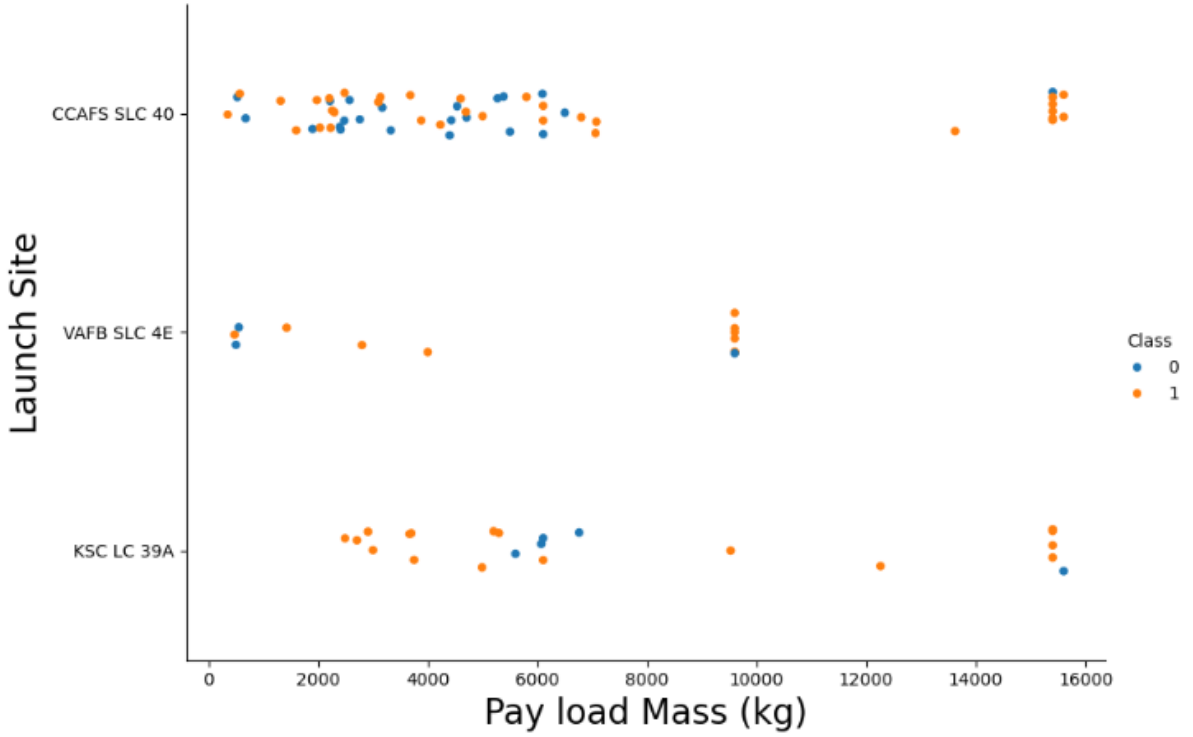
EDA with Data Visualization

TASK 2: Visualize the relationship between Payload Mass and Launch Site

We also want to observe if there is any relationship between launch sites and their payload mass.

In [6]:

```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch site,
sns.catplot(y="LaunchSite", x="PayloadMass", hue="Class", data=df, aspect = 1.5 ,height=6)
plt.xlabel("Pay load Mass (kg)",fontsize=20)
plt.ylabel("Launch Site",fontsize=20)
plt.show()
```



EDA with Data Visualization

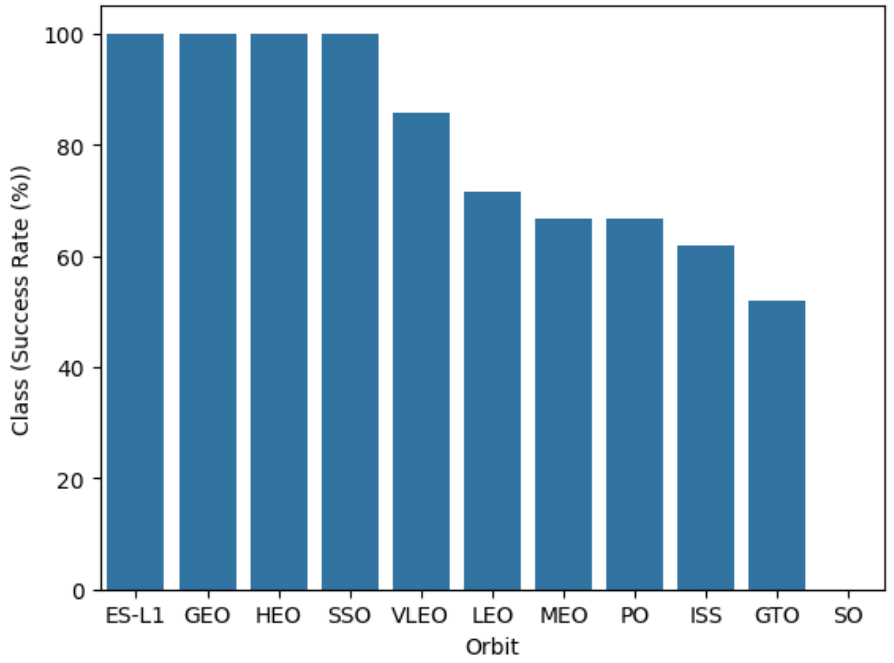
TASK 3: Visualize the relationship between success rate of each orbit type

Next, we want to visually check if there are any relationship between success rate and orbit type.

Let's create a `bar chart` for the sucess rate of each orbit

In [7]:

```
# HINT use groupby method on Orbit column and get the mean of Class column
sr_df = df.groupby('Orbit')['Class'].mean().reset_index().sort_values(by='Class', ascending=False)
sr_df['Class'] = sr_df['Class'] * 100


sns.barplot(data=sr_df, x='Orbit', y='Class')
plt.xlabel('Orbit')
plt.ylabel('Class (Success Rate (%))')
plt.show()
```



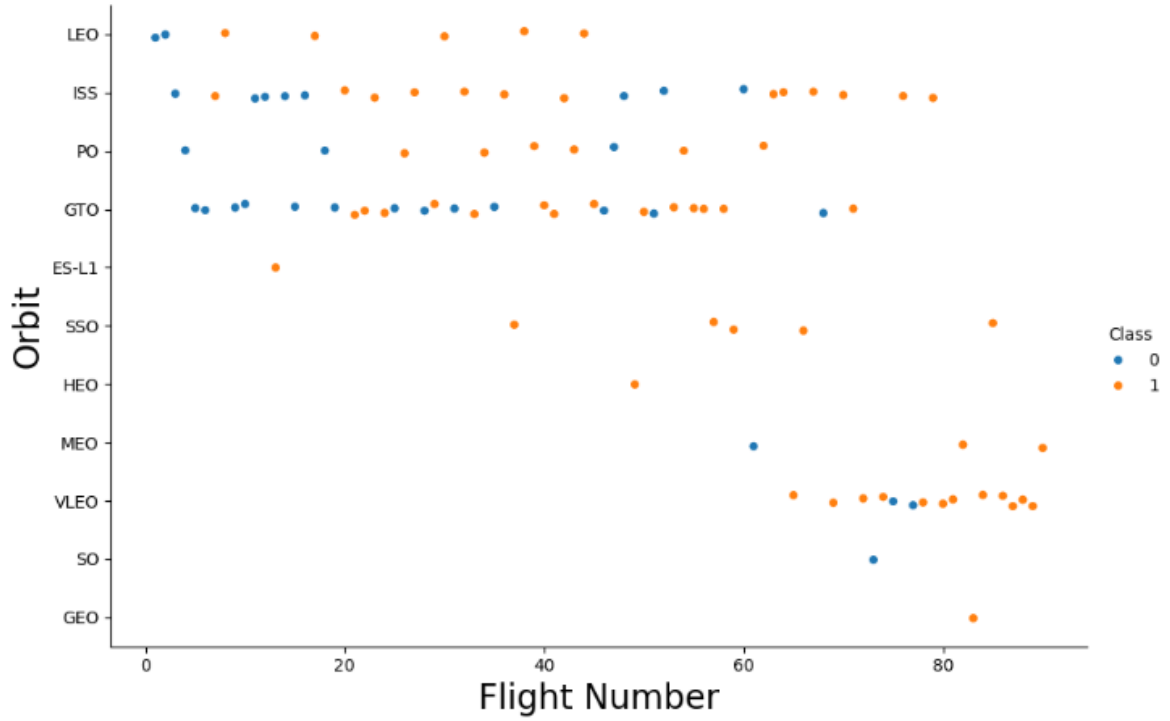
EDA with Data Visualization

TASK 4: Visualize the relationship between FlightNumber and Orbit type

For each orbit, we want to see if there is any relationship between FlightNumber and Orbit type.

In [8]:

```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to
sns.catplot(y="Orbit", x="FlightNumber", hue="Class", data=df, aspect = 1.5 ,height=6)

plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```



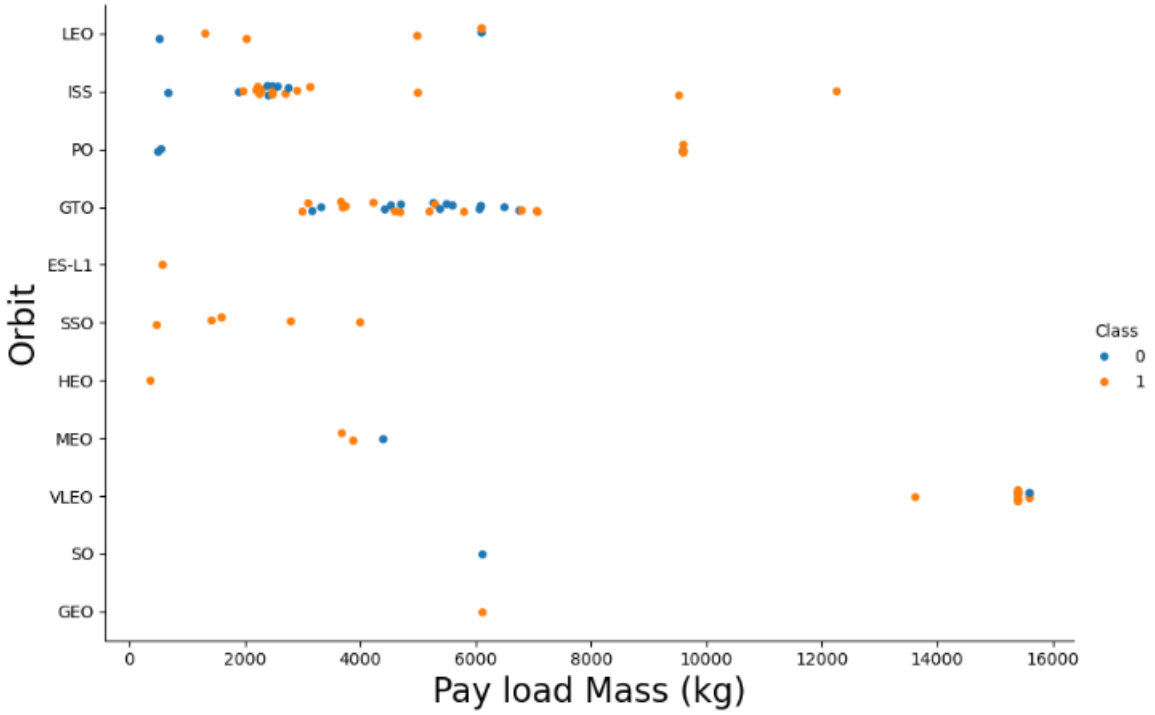
EDA with Data Visualization

TASK 5: Visualize the relationship between Payload Mass and Orbit type

Similarly, we can plot the Payload Mass vs. Orbit scatter point charts to reveal the relationship between Payload Mass and Orbit type

In [9]:

```
# Plot a scatter point chart with x axis to be Payload Mass and y axis to be the Orbit, and hue to
sns.catplot(y="Orbit", x="PayloadMass", hue="Class", data=df, aspect = 1.5 ,height=6)

plt.xlabel("Pay load Mass (kg)",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```



EDA with Data Visualization

TASK 6: Visualize the launch success yearly trend

You can plot a line chart with x axis to be `Year` and y axis to be average success rate, to get the average launch success trend.

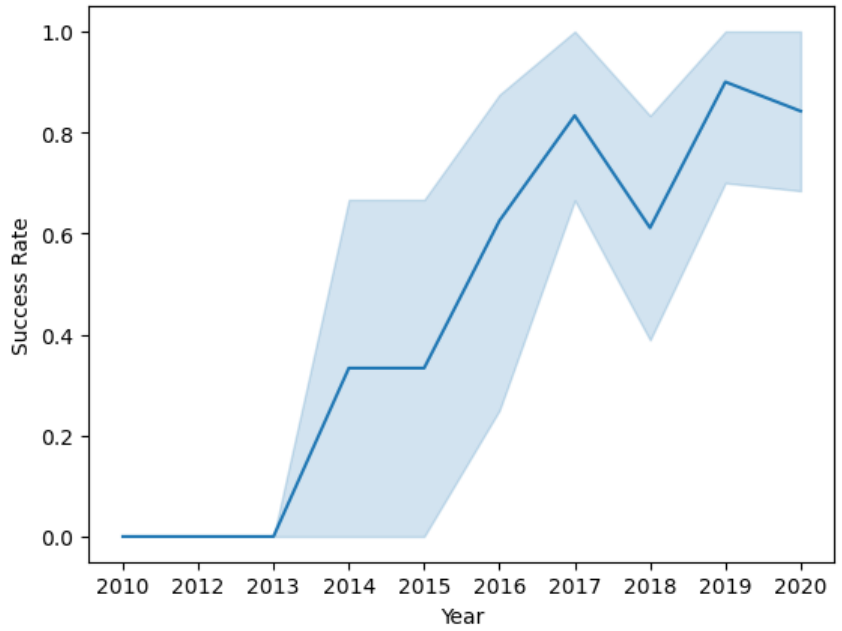
The function will help you get the year from the date:

In [10]:
```
# A function to Extract years from the date
year=[]
def Extract_year():
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year
Extract_year()
df['Date'] = year
df.head()
```

Out[10]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused
0	1	2010	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False
1	2	2012	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False
2	3	2013	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False
3	4	2013	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False
4	5	2013	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False

In [11]:
```
# Plot a line chart with x axis to be the extracted year and y axis to be the success rate
sns.lineplot(data=df, x="Date", y="Class")
plt.xlabel("Year")
plt.ylabel("Success Rate")
plt.show()
```



you can observe that the sucess rate since 2013 kept increasing till 2020

EDA with Data Visualization

TASK 7: Create dummy variables to categorical columns

Use the function `get_dummies` and `features` dataframe to apply OneHotEncoder to the column `Orbits`, `LaunchSite`, `LandingPad`, and `Serial`. Assign the value to the variable `features_one_hot`, display the results using the method head. Your result dataframe must include all features including the encoded ones.

In [13]:
```
# HINT: Use get_dummies() function on the categorical columns
features_one_hot = pd.get_dummies(features, columns=['Orbit', 'LaunchSite', 'LandingPad', 'Serial'
features_one_hot.head()
```

Out[13]:

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO
0	1	6104.959412	1	False	False	False	1.0	0	False	False
1	2	525.000000	1	False	False	False	1.0	0	False	False
2	3	677.000000	1	False	False	False	1.0	0	False	False
3	4	500.000000	1	False	False	False	1.0	0	False	False
4	5	3170.000000	1	False	False	False	1.0	0	False	False

5 rows × 80 columns

EDA with Data Visualization

TASK 8: Cast all numeric columns to `float64`

Now that our `features_one_hot` dataframe only contains numbers, cast the entire dataframe to variable type `float64`

In [14]:
```
# HINT: use astype function
features_one_hot.astype('float64')
```

Out[14]:

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO
0	1.0	6104.959412	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
1	2.0	525.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
2	3.0	677.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
3	4.0	500.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
4	5.0	3170.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...
85	86.0	15400.000000	2.0	1.0	1.0	1.0	5.0	2.0	0.0	0.0
86	87.0	15400.000000	3.0	1.0	1.0	1.0	5.0	2.0	0.0	0.0
87	88.0	15400.000000	6.0	1.0	1.0	1.0	5.0	5.0	0.0	0.0
88	89.0	15400.000000	3.0	1.0	1.0	1.0	5.0	2.0	0.0	0.0
89	90.0	3681.000000	1.0	1.0	0.0	1.0	5.0	0.0	0.0	0.0

90 rows × 80 columns

EDA with SQL

- As part of the Exploratory Data Analysis, the following SQL queries were executed:
  - Display the names of the unique launch sites in the space mission
  - Display 5 records where launch sites begin with the string 'CCA‘
  - Display the total payload mass carried by boosters launched by NASA (CRS)
  - Display average payload mass carried by booster version F9 v1.1
  - List the date when the first successful landing outcome in ground pad was achieved
  - List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000
  - List the total number of successful and failure mission outcomes

EDA with SQL

- GitHub URL of the completed notebook: (https://github.com/oswaldofelizzola/Coursera-IBM-Data-Science-Professional/blob/fdc3e5636e43081ca45e5065b47b2332f3a7eaab/Applied%20Data%20Science%20Capstone/4%20jupyter-labs-eda-sql-coursera_sqllite.ipynb)

EDA with SQL

Task 1

Display the names of the unique launch sites in the space mission

In [10]:
```
%sql SELECT DISTINCT LAUNCH_SITE as "Launch_Sites" FROM SPACEXTBL;
```

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

Out[10]:

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

EDA with SQL

Task 2

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

In [11]:
```
%sql SELECT * FROM 'SPACEXTBL' WHERE Launch_Site LIKE 'CCA%' LIMIT 5;
```

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

Out[11]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_C
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	

EDA with SQL

Task 3

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

In [12]:
```
%sql SELECT SUM(PAYLOAD_MASS__KG_) as "Total Payload Mass(Kgs)", Customer FROM 'SPACEXTBL' WHERE Cu
```

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

Out[12]:

Total Payload Mass(Kgs)	Customer
45596	NASA (CRS)

EDA with SQL

Task 4

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

In [13]:
```
%sql SELECT AVG(PAYLOAD_MASS__KG_) as "Payload Mass Kgs", Customer, Booster_Version FROM 'SPACEXTBL
```

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

Out[13]:

Payload Mass Kgs	Customer	Booster_Version
2534.6666666666665	MDA	F9 v1.1 B1003

Task 5

List the date when the first succesful landing outcome in ground pad was acheived.

Hint:Use min function

In [25]:
```
%sql SELECT MIN(DATE) FROM 'SPACEXTBL' WHERE "Landing_Outcome" = "Success (ground pad)";
```

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

Out[25]:

MIN(DATE)
2015-12-22

EDA with SQL

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

In [26]:
```
%sql SELECT DISTINCT Booster_Version, Payload FROM SPACEXTBL WHERE "Landing_Outcome" = "Success (d
```

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

Out[26]:

Booster_Version	Payload
F9 FT B1022	JCSAT-14
F9 FT B1026	JCSAT-16
F9 FT B1021.2	SES-10
F9 FT B1031.2	SES-11 / EchoStar 105

Task 7

List the total number of successful and failure mission outcomes

In [27]:
```
%sql SELECT "Mission_Outcome", COUNT("Mission_Outcome") as Total FROM SPACEXTBL GROUP BY "Mission_(
```

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

Out[27]:

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

EDA with SQL

Task 8

List all the booster_versions that have carried the maximum payload mass. Use a subquery.

In [28]:
```
%sql SELECT "Booster_Version",Payload, "PAYLOAD_MASS__KG_" FROM SPACEXTBL WHERE "PAYLOAD_MASS__KG_
```

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

Out[28]:

Booster_Version	Payload	PAYLOAD_MASS__KG_
F9 B5 B1048.4	Starlink 1 v1.0, SpaceX CRS-19	15600
F9 B5 B1049.4	Starlink 2 v1.0, Crew Dragon in-flight abort test	15600
F9 B5 B1051.3	Starlink 3 v1.0, Starlink 4 v1.0	15600
F9 B5 B1056.4	Starlink 4 v1.0, SpaceX CRS-20	15600
F9 B5 B1048.5	Starlink 5 v1.0, Starlink 6 v1.0	15600
F9 B5 B1051.4	Starlink 6 v1.0, Crew Dragon Demo-2	15600
F9 B5 B1049.5	Starlink 7 v1.0, Starlink 8 v1.0	15600
F9 B5 B1060.2	Starlink 11 v1.0, Starlink 12 v1.0	15600
F9 B5 B1058.3	Starlink 12 v1.0, Starlink 13 v1.0	15600
F9 B5 B1051.6	Starlink 13 v1.0, Starlink 14 v1.0	15600
F9 B5 B1060.3	Starlink 14 v1.0, GPS III-04	15600
F9 B5 B1049.7	Starlink 15 v1.0, SpaceX CRS-21	15600

EDA with SQL

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.

In [32]:
```
%sql SELECT substr(Date,6,2), substr(Date, 0, 5),"Booster_Version", "Launch_Site", Payload, "PAYLO
```

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

Out[32]:

substr(Date,6,2)	substr(Date, 0, 5)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Mission_Outcome
01	2015	F9 v1.1 B1012	CCAFS LC-40	SpaceX CRS-5	2395	Success
04	2015	F9 v1.1 B1015	CCAFS LC-40	SpaceX CRS-6	1898	Success

EDA with SQL

Task 10

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.

In [38]:
```
%sql SELECT * FROM SPACEXTBL WHERE "Landing_Outcome" LIKE 'Success%' AND (Date BETWEEN '2010-06-04
```

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

Out[38]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Missi
2017-02-19	14:39:00	F9 FT B1031.1	KSC LC-39A	SpaceX CRS-10	2490	LEO (ISS)	NASA (CRS)	
2017-01-14	17:54:00	F9 FT B1029.1	VAFB SLC-4E	Iridium NEXT 1	9600	Polar LEO	Iridium Communications	
2016-08-14	5:26:00	F9 FT B1026	CCAFS LC-40	JCSAT-16	4600	GTO	SKY Perfect JSAT Group	
2016-07-18	4:45:00	F9 FT B1025.1	CCAFS LC-40	SpaceX CRS-9	2257	LEO (ISS)	NASA (CRS)	
2016-05-27	21:39:00	F9 FT B1023.1	CCAFS LC-40	Thaicom 8	3100	GTO	Thaicom	
2016-05-06	5:21:00	F9 FT B1022	CCAFS LC-40	JCSAT-14	4696	GTO	SKY Perfect JSAT Group	
2016-04-08	20:43:00	F9 FT B1021.1	CCAFS LC-40	SpaceX CRS-8	3136	LEO (ISS)	NASA (CRS)	
2015-12-22	1:29:00	F9 FT B1019	CCAFS LC-40	OG2 Mission 2 11 Orbcomm-OG2 satellites	2034	LEO	Orbcomm	

Build an Interactive Map with Folium

- A Folium map was generated to visualize all launch sites. Markers, circles, and lines were implemented as map objects to denote the outcome of launches at each site.
- Additionally, a binary variable representing 'launch set outcomes' was created, with failure encoded as 0 and success as 1.
- GitHub URL of the completed notebook: (https://github.com/oswaldofelizzola/Coursera-IBM-Data-Science-Professional/blob/fdc3e5636e43081ca45e5065b47b2332f3a7eaab/Applied%20Data%20Science%20Capstone/6%20lab_jupyter_launch_site_location.ipynb)

Build an Interactive Map with Folium

[4]:
```
## Task 1: Mark all launch sites on a map
```

First, let's try to add each site's location on a map using site's latitude and longitude coordinates

The following dataset with the name `spacex_launch_geo.csv` is an augmented dataset with latitude and longitude added for each site.

[5]:
```
# Download and read the `spacex_launch_geo.csv`
from js import fetch
import io

URL = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/space
resp = await fetch(URL)
spacex_csv_file = io.BytesIO((await resp.arrayBuffer()).to_py())
spacex_df=pd.read_csv(spacex_csv_file)
```

Now, you can take a look at what are the coordinates for each site.

[6]:
```
# Select relevant sub-columns: `Launch Site`, `Lat(Latitude)`, `Long(Longitude)`, `class`
spacex_df = spacex_df[['Launch Site', 'Lat', 'Long', 'class']]
launch_sites_df = spacex_df.groupby(['Launch Site'], as_index=False).first()
launch_sites_df = launch_sites_df[['Launch Site', 'Lat', 'Long']]
launch_sites_df
```

[6]:

	Launch Site	Lat	Long
0	CCAFS LC-40	28.562302	-80.577356
1	CCAFS SLC-40	28.563197	-80.576820
2	KSC LC-39A	28.573255	-80.646895
3	VAFB SLC-4E	34.632834	-120.610745

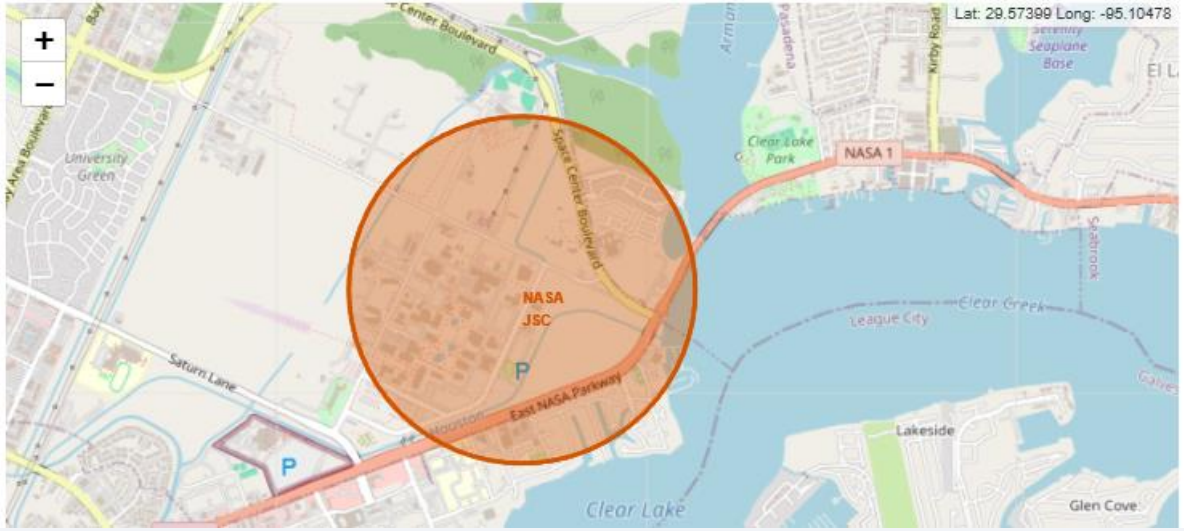
Build an Interactive Map with Folium

[7]:
```
# Start location is NASA Johnson Space Center
nasa_coordinate = [29.559684888503615, -95.0830971930759]
site_map = folium.Map(location=nasa_coordinate, zoom_start=10)
```

We could use `folium.Circle` to add a highlighted circle area with a text label on a specific coordinate. For example,

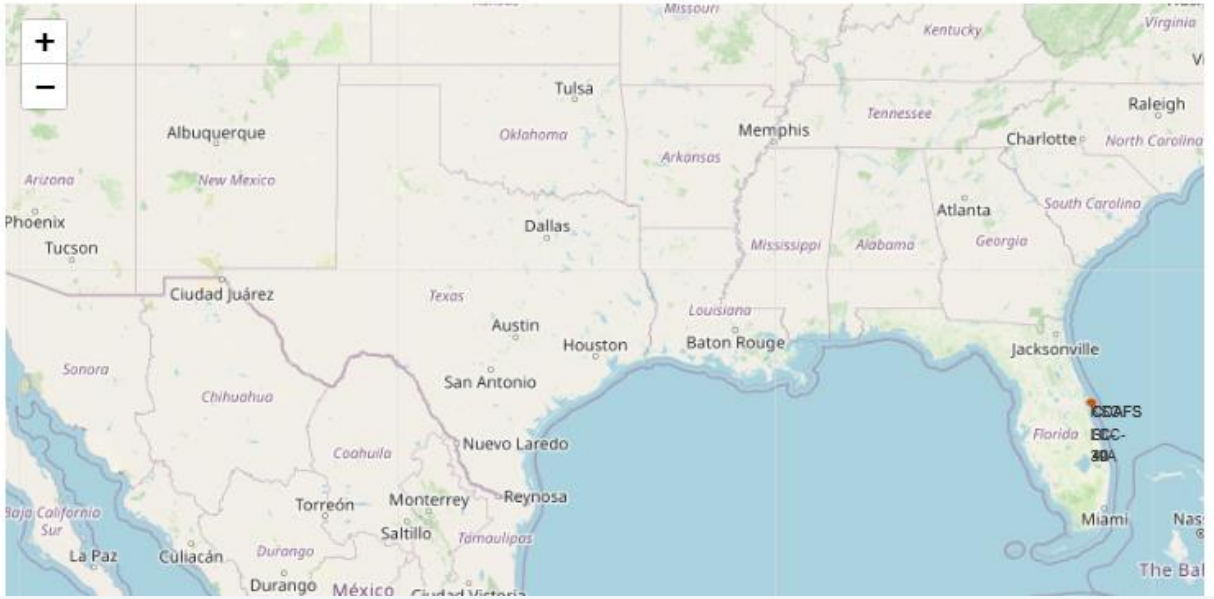
[25]:
```
# Create a blue circle at NASA Johnson Space Center's coordinate with a popup label showing its name
circle = folium.Circle(nasa_coordinate, radius=1000, color='#d35400', fill=True).add_child(folium.Popup('NASA Johr
# Create a blue circle at NASA Johnson Space Center's coordinate with a icon showing its name
marker = folium.map.Marker(
    nasa_coordinate,
    # Create an icon as a text label
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % 'NASA JSC',
        )
    )
site_map.add_child(circle)
site_map.add_child(marker)
```

[25]:



Build an Interactive Map with Folium

[9]:
```
# Initial the map
site_map = folium.Map(location=nasa_coordinate, zoom_start=5)
# For each launch site, add a Circle object based on its coordinate (Lat, Long) values. In addition, add Launch si
for lat, lng, label in zip(launch_sites_df['Lat'], launch_sites_df['Long'], launch_sites_df['Launch Site']):
    coordinate = [lat, lng]
    circle = folium.Circle(coordinate, radius=1000, color='#d35400', fill=True).add_child(folium.Popup(label))
    marker = folium.map.Marker(
        coordinate,
        icon=DivIcon(
            icon_size=(20,20),
            icon_anchor=(0,0),
            html='%s' % label,)
        )
    site_map.add_child(circle)
    site_map.add_child(marker)

site_map
```

[9]:


Build an Interactive Map with Folium

[10]:
```
# Task 2: Mark the success/failed launches for each site on the map
```

Next, let's try to enhance the map by adding the launch outcomes for each site, and see which sites have high success rates. Recall that data frame spacex_df has detailed launch records, and the `class` column indicates if this launch was successful or not

[11]:
```
spacex_df.tail(10)
```

[11]:

	Launch Site	Lat	Long	class
46	KSC LC-39A	28.573255	-80.646895	1
47	KSC LC-39A	28.573255	-80.646895	1
48	KSC LC-39A	28.573255	-80.646895	1
49	CCAFS SLC-40	28.563197	-80.576820	1
50	CCAFS SLC-40	28.563197	-80.576820	1
51	CCAFS SLC-40	28.563197	-80.576820	0
52	CCAFS SLC-40	28.563197	-80.576820	0
53	CCAFS SLC-40	28.563197	-80.576820	0
54	CCAFS SLC-40	28.563197	-80.576820	1
55	CCAFS SLC-40	28.563197	-80.576820	0

Next, let's create markers for all launch records. If a launch was successful `(class=1)`, then we use a green marker and if a launch was failed, we use a red marker `(class=0)`

Note that a launch only happens in one of the four launch sites, which means many launch records will have the exact same coordinate. Marker clusters can be a good way to simplify a map containing many markers having the same coordinate.

Let's first create a `MarkerCluster` object

[12]:
```
marker_cluster = MarkerCluster()
```

Build an Interactive Map with Folium

[13]:

```
# Apply a function to check the value of `class` column
# If class=1, marker_color value will be green
# If class=0, marker_color value will be red

def assign_marker_color(launch_outcome):
    if launch_outcome == 1:
        return 'green'
    else:
        return 'red'

spacex_df['marker_color'] = spacex_df['class'].apply(assign_marker_color)
spacex_df.tail(10)
```

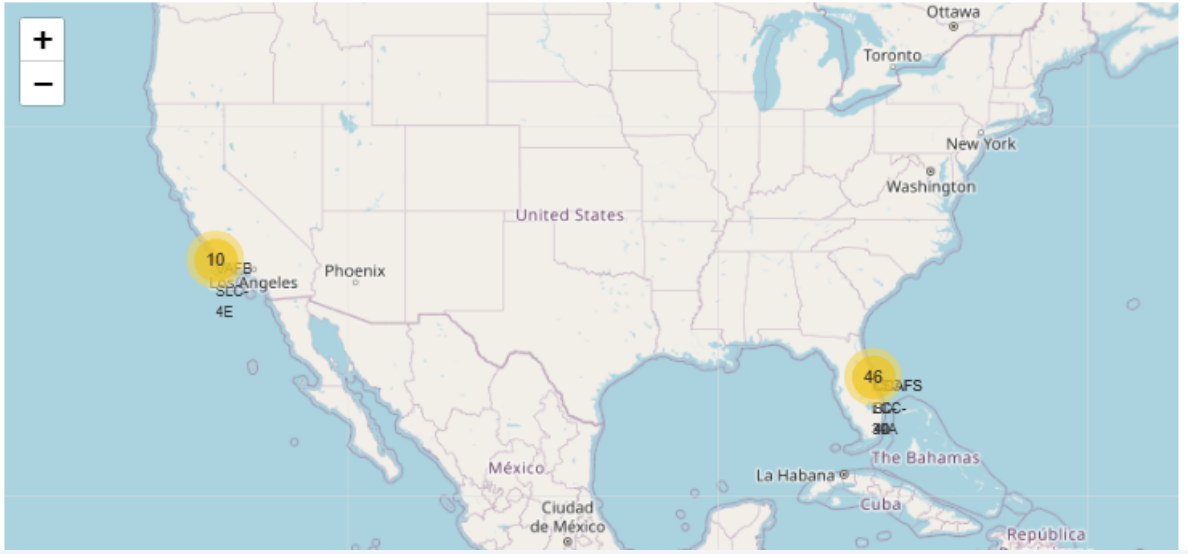
[13]:

	Launch Site	Lat	Long	class	marker_color
46	KSC LC-39A	28.573255	-80.646895	1	green
47	KSC LC-39A	28.573255	-80.646895	1	green
48	KSC LC-39A	28.573255	-80.646895	1	green
49	CCAFS SLC-40	28.563197	-80.576820	1	green
50	CCAFS SLC-40	28.563197	-80.576820	1	green
51	CCAFS SLC-40	28.563197	-80.576820	0	red
52	CCAFS SLC-40	28.563197	-80.576820	0	red
53	CCAFS SLC-40	28.563197	-80.576820	0	red
54	CCAFS SLC-40	28.563197	-80.576820	1	green
55	CCAFS SLC-40	28.563197	-80.576820	0	red

Build an Interactive Map with Folium

[16]:
```
# Add marker_cluster to current site_map
site_map.add_child(marker_cluster)

# for each row in spacex_df data frame
# create a Marker object with its coordinate
# and customize the Marker's icon property to indicate if this launch was successed or failed,
# e.g., icon=folium.Icon(color='white', icon_color=row['marker_color']
for lat, lng, label, color in zip(spacex_df['Lat'], spacex_df['Long'], spacex_df['Launch Site'], spacex_df['marker_
    # TODO: Create and add a Marker cluster to the site map
    # marker = folium.Marker(...)
    coordinate = [lat, lng]
    marker = folium.Marker(
        coordinate,
        icon=folium.Icon(color='white', icon_color=color),
        popup=label
    )
    marker_cluster.add_child(marker)

site_map
```

[16]:



Build an Interactive Map with Folium

Build an Interactive Map with Folium

[17]:
```
# TASK 3: Calculate the distances between a launch site to its proximities
```

Next, we need to explore and analyze the proximities of launch sites.

Let's first add a `MousePosition` on the map to get coordinate for a mouse over a point on the map. As such, while you are exploring the map, you can easily find the coordinates of any points of interests (such as railway)

[18]:
```
# Add Mouse Position to get the coordinate (Lat, Long) for a mouse over on the map
formatter = "function(num) {return L.Util.formatNum(num, 5);};"
mouse_position = MousePosition(
    position='topright',
    separator=' Long: ',
    empty_string='NaN',
    lng_first=False,
    num_digits=20,
    prefix='Lat:',
    lat_formatter=formatter,
    lng_formatter=formatter,
)

site_map.add_child(mouse_position)
site_map
```

[18]:


Build an Interactive Map with Folium

[19]:
```
from math import sin, cos, sqrt, atan2, radians

def calculate_distance(lat1, lon1, lat2, lon2):
    # approximate radius of earth in km
    R = 6373.0

    lat1 = radians(lat1)
    lon1 = radians(lon1)
    lat2 = radians(lat2)
    lon2 = radians(lon2)

    dlon = lon2 - lon1
    dlat = lat2 - lat1

    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))

    distance = R * c
    return distance
```

TODO: Mark down a point on the closest coastline using MousePosition and calculate the distance between the coastline point and the launch site.

[20]:
```
# find coordinate of the closet coastline
# e.g.,: Lat: 28.56367  Lon: -80.57163
# distance_coastline = calculate_distance(launch_site_lat, launch_site_lon, coastline_lat, coastline_lon)

coastline_lat = 28.56398
coastline_lon = -80.56809
launch_site_lat = 28.56321
launch_site_lon = -80.57683
distance_coastline = calculate_distance(launch_site_lat, launch_site_lon, coastline_lat, coastline_lon)
```

Build an Interactive Map with Folium

[21]:
```
# Create and add a folium.Marker on your selected closest coastline point on the map
# Display the distance between coastline point and launch site using the icon property
# for example
# distance_marker = folium.Marker(
#    coordinate,
#    icon=DivIcon(
#        icon_size=(20,20),
#        icon_anchor=(0,0),
#        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:10.2f} KM".format(distance),
#        )
#    )
coast_coordinates = [coastline_lat, coastline_lon]
distance_marker = folium.Marker(
    coast_coordinates,
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='%s' % "{:10.2f} KM".format(distance_coastline),)
    )
distance_marker.add_to(site_map)
site_map
```

[21]:

Build an Interactive Map with Folium

[22]:
```
# Create a `folium.PolyLine` object using the coastline coordinates and launch site coordinate
# lines=folium.PolyLine(locations=coordinates, weight=1)
launch_site_coordinates = [launch_site_lat, launch_site_lon]
lines=folium.PolyLine(locations=[coast_coordinates, launch_site_coordinates], weight=1)
site_map.add_child(lines)
```

[22]:

Build an Interactive Map with Folium

[23]:
```
# Create a marker with distance to a closest city, railway, highway, etc.
# Draw a line between the marker to the launch site

city_lat = 28.61208
city_lon = -80.80764
distance_city = calculate_distance(launch_site_lat, launch_site_lon, city_lat, city_lon)

city_coordinates = [city_lat, city_lon]
distance_marker = folium.Marker(
    city_coordinates,
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='%s' % "{:10.2f} KM".format(distance_city),
        )
    )
distance_marker.add_to(site_map)

launch_site_coordinates = [launch_site_lat, launch_site_lon]
lines=folium.PolyLine(locations=[city_coordinates, launch_site_coordinates], weight=1)
site_map.add_child(lines)
site_map
```

[23]:

Build a Dashboard with Plotly Dash

- The development of an interactive dashboard application was achieved using Plotly Dash by:
  - Adding a Launch Site Drop-down Input Component
  - Adding a callback function to render success-pie-chart based on selected site dropdown
  - Adding a Range Slider to Select Payload
  - Adding a callback function to render the success-payload-scatter-chart scatter plot
- GitHub URL of the completed notebook: (https://github.com/oswaldofelizzola/Coursera-IBM-Data-Science-Professional/blob/fdc3e5636e43081ca45e5065b47b2332f3a7eaab/Applied%20Data%20Science%20Capstone/7%20spacex-dash-app.py)

Build a Dashboard with Plotly Dash

```
# TASK 1: Add a dropdown list to enable Launch Site selection
# The default select value is for ALL sites
dcc.Dropdown(id='site-dropdown',
                options=[
                        {'label': 'All Sites', 'value': 'ALL'},
                        {'label': 'CCAFS LC-40', 'value': 'CCAFS LC-40'},
                        {'label': 'VAFB SLC-4E', 'value': 'VAFB SLC-4E'},
                        {'label': 'KSC LC-39A', 'value': 'KSC LC-39A'},
                        {'label': 'CCAFS SLC-40', 'value': 'CCAFS SLC-40'}
                        ],
                value='ALL',
                placeholder='Select a Launch Site here',
                searchable=True
                ),
html.Br(),
```

Build a Dashboard with Plotly Dash

```
# TASK 2: Add a pie chart to show the total successful launches count for all sites
# If a specific launch site was selected, show the Success vs. Failed counts for the site
html.Div(dcc.Graph(id='success-pie-chart')),
html.Br(),
```

```
# TASK 2:
# Add a callback function for `site-dropdown` as input, `success-pie-chart` as output
@app.callback(Output(component_id='success-pie-chart', component_property='figure'),
              Input(component_id='site-dropdown', component_property='value'))
def get_pie_chart(entered_site):
    filtered_df = spacex_df
    if entered_site == 'ALL':
        fig = px.pie(filtered_df, values='class',
        names='Launch Site',
        title='Success Count for all launch sites')
        return fig
    else:
        # return the outcomes piechart for a selected site
        filtered_df=spacex_df[spacex_df['Launch Site']== entered_site]
        filtered_df=filtered_df.groupby(['Launch Site','class']).size().reset_index(name='class count')
        fig=px.pie(filtered_df,values='class count',names='class',title=f"Total Success Launches for site {er
        return fig
```

Build a Dashboard with Plotly Dash

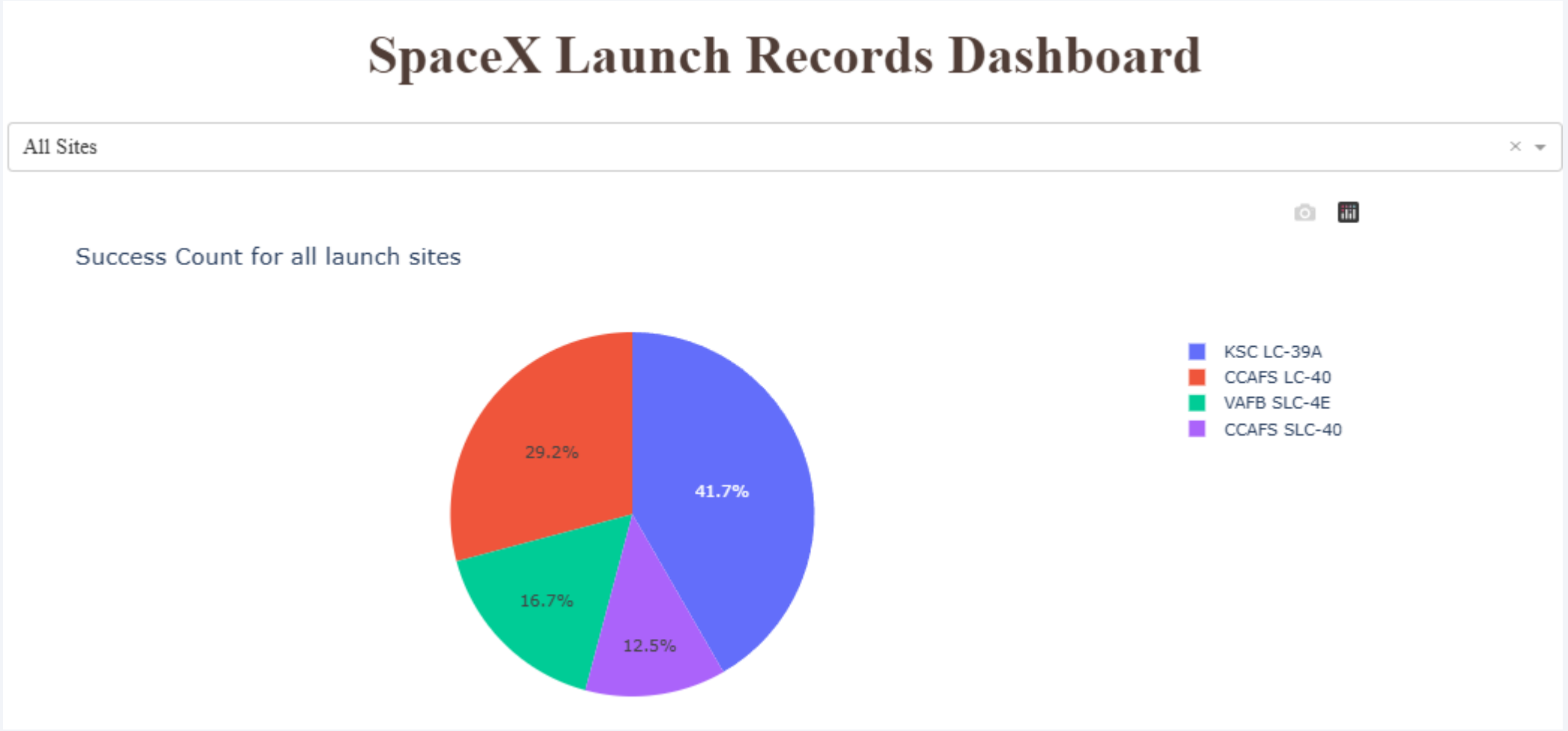
```
# TASK 3: Add a slider to select payload range
#dcc.RangeSlider(id='payload-slider',...)
dcc.RangeSlider(id='payload-slider',
                min=0,
                max=10000,
                step=1000,
                value=[min_payload, max_payload]
                ),
```

Build a Dashboard with Plotly Dash

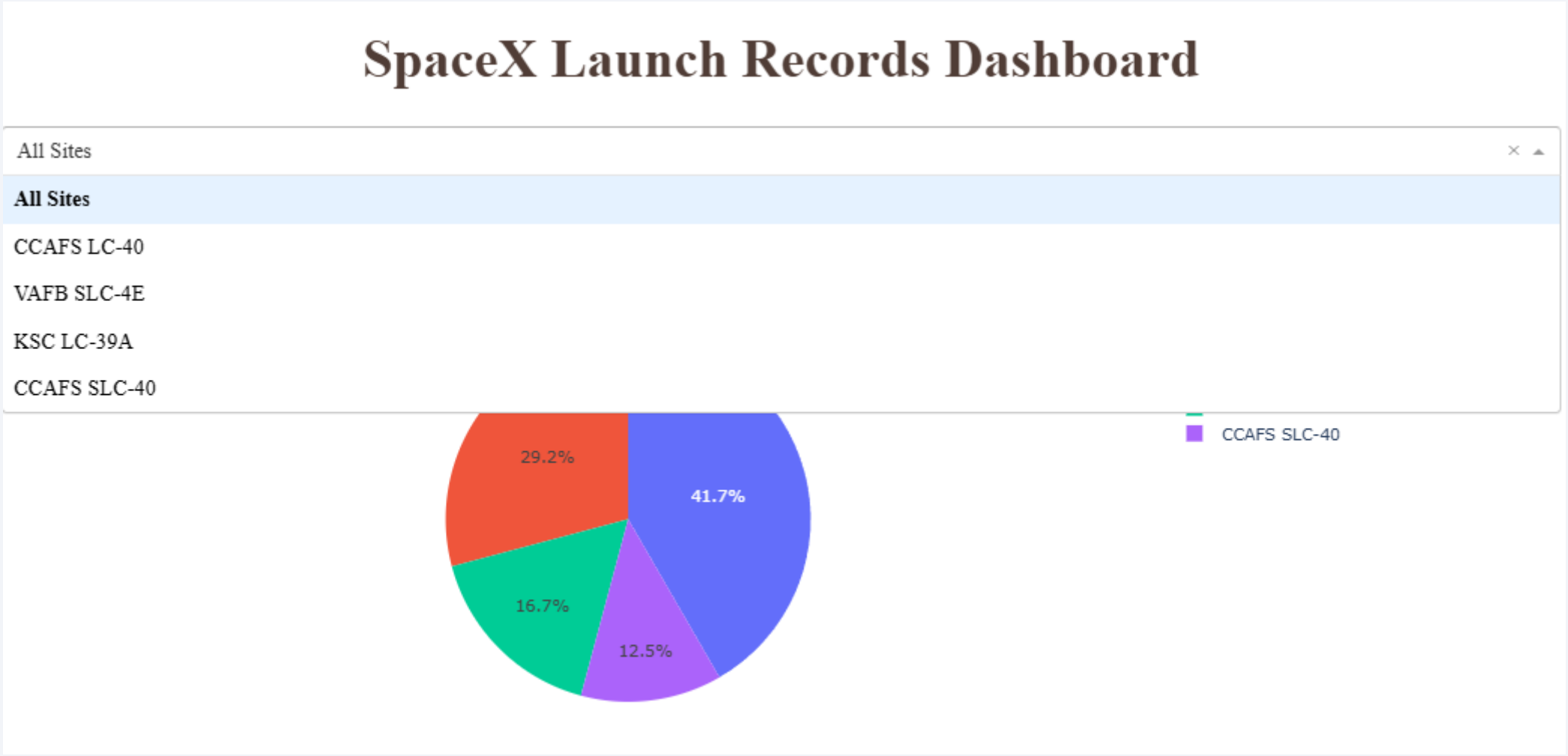
```
# TASK 4: Add a scatter chart to show the correlation between payload and launch success
html.Div(dcc.Graph(id='success-payload-scatter-chart')),
])
```

```
# TASK 4:
# Add a callback function for `site-dropdown` and `payload-slider` as inputs, `success-payload-scatter-chart`
@app.callback(Output(component_id='success-payload-scatter-chart',component_property='figure'),
              [Input(component_id='site-dropdown',component_property='value'),
              Input(component_id='payload-slider',component_property='value')])
def scatter(entered_site,payload):
    filtered_df = spacex_df[spacex_df['Payload Mass (kg)'].between(payload[0],payload[1])]
    # thought reusing filtered_df may cause issues, but tried it out of curiosity and it seems to be working

    if entered_site=='ALL':
        fig=px.scatter(filtered_df,x='Payload Mass (kg)',y='class',color='Booster Version Category',title='Su
        return fig
    else:
        fig=px.scatter(filtered_df[filtered_df['Launch Site']==entered_site],x='Payload Mass (kg)',y='class',
        return fig
```

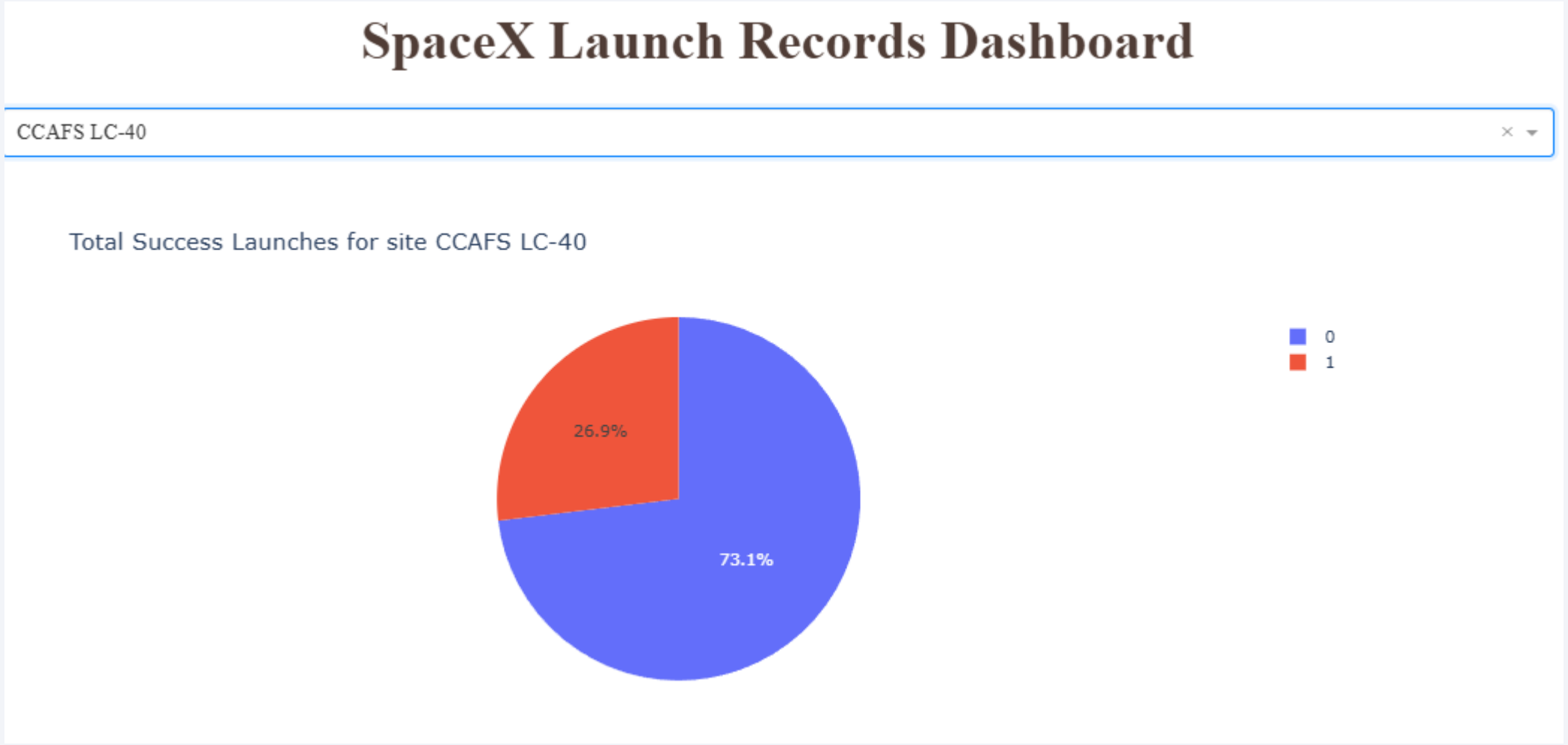
Build a Dashboard with Plotly Dash



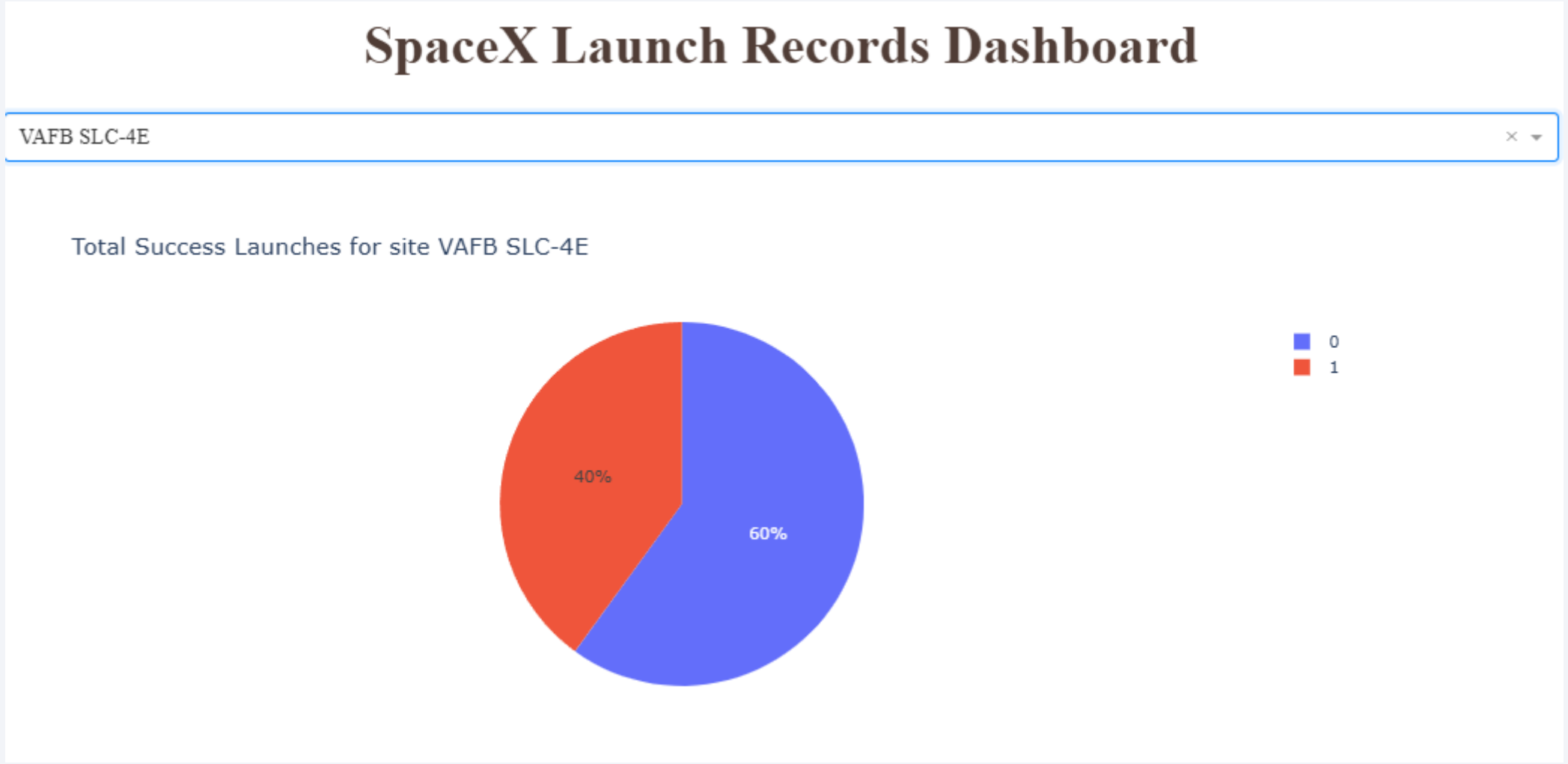
Build a Dashboard with Plotly Dash



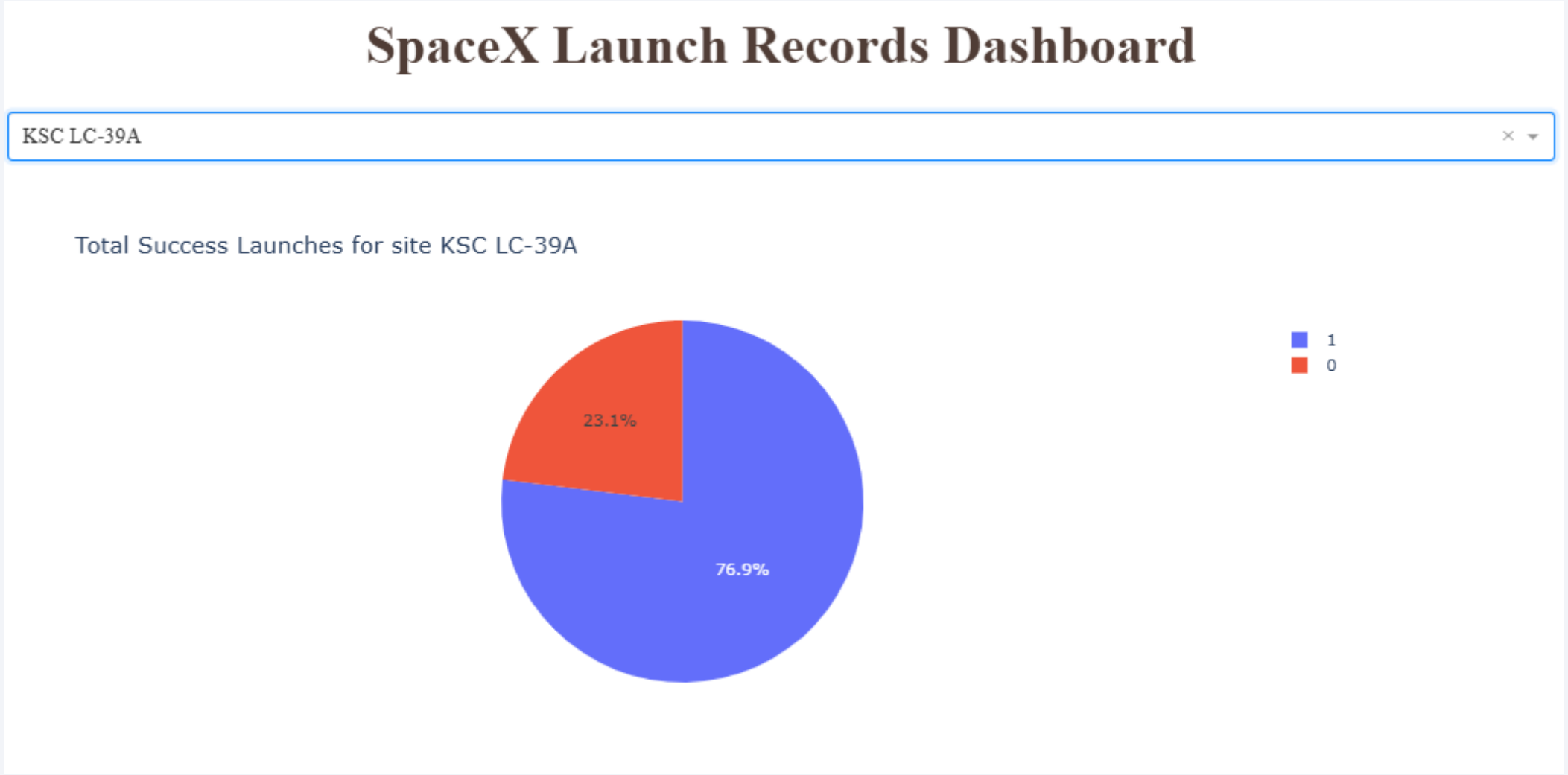
Build a Dashboard with Plotly Dash



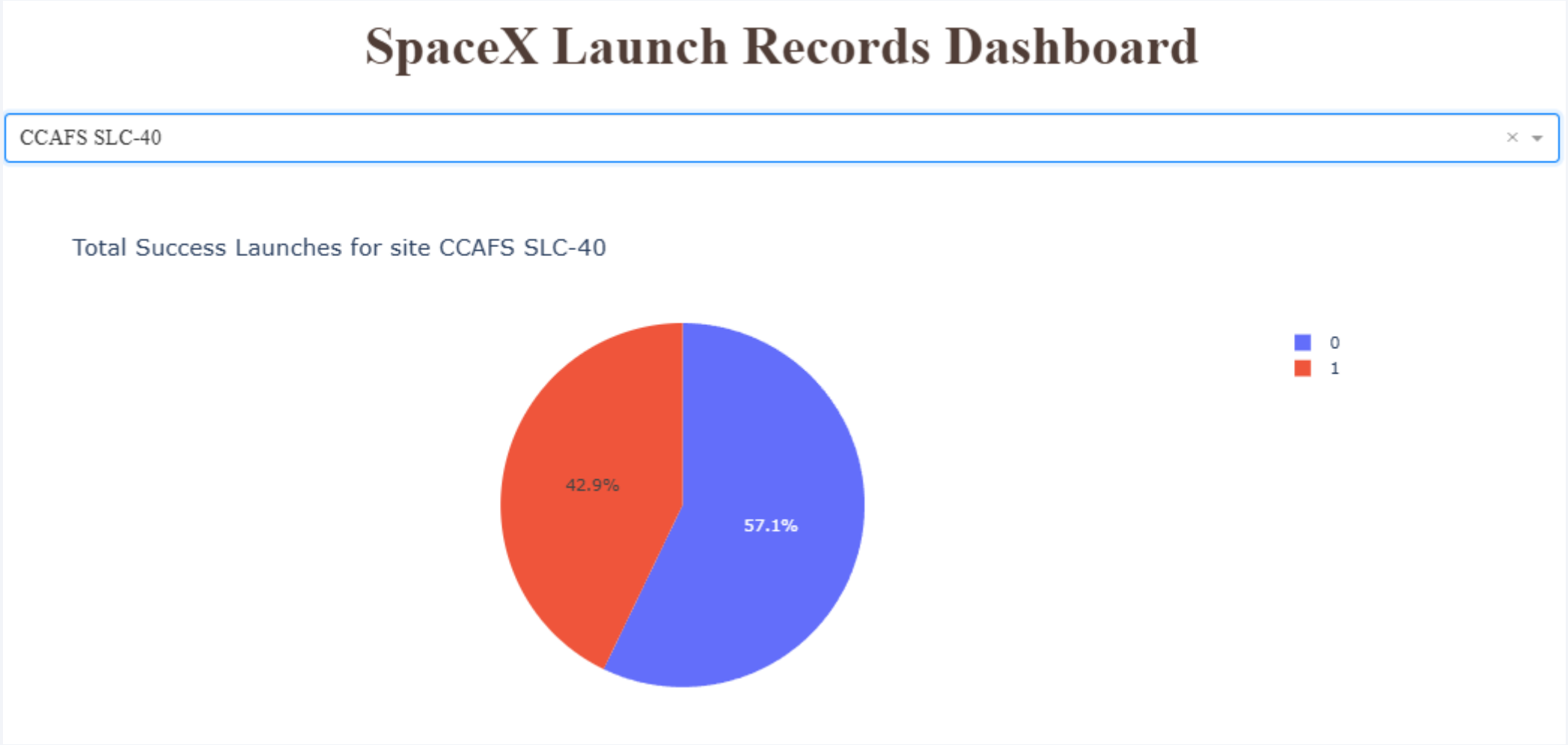
Build a Dashboard with Plotly Dash



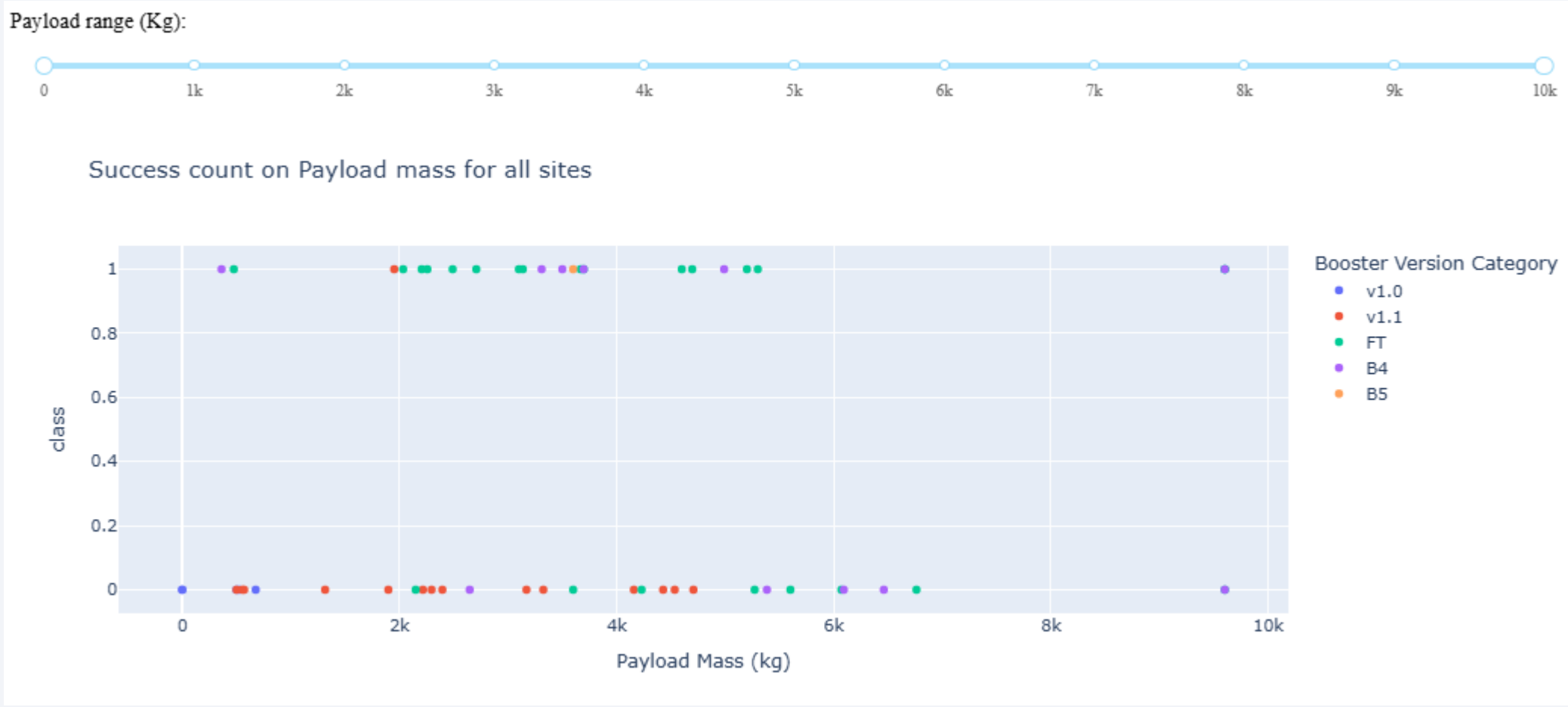
Build a Dashboard with Plotly Dash



Build a Dashboard with Plotly Dash



Build a Dashboard with Plotly Dash



Build a Dashboard with Plotly Dash

Predictive Analysis (Classification)

- Following the loading of the data into a Pandas DataFrame, I commenced Exploratory Data Analysis (EDA) and the process of determining appropriate training labels by:
  - A NumPy array was generated from the 'Class' column of the data by applying the to_numpy() method and assigned to the variable Y to represent the outcome variable.
  - Subsequently, the feature dataset (x) was standardized by applying the StandardScaler() function from the preprocessing module of Sklearn.
  - Following standardization, the dataset was divided into training and testing sets via the train_test_split function from Sklearn's model_selection module. The testing set comprised 20% of the data, and a random_state of 2 was used for reproducibility.

Predictive Analysis (Classification)

- We sought to identify the machine learning model (from the set of SVM, Classification Trees, k-Nearest Neighbors, and Logistic Regression) that yielded the highest performance when evaluated against the test data;
  - For each algorithm under consideration, an object was first created. Following this, a GridSearchCV object was instantiated and configured with a specific parameter space for each model.
  - For each model being evaluated, a GridSearchCV object was instantiated with a 10-fold cross-validation strategy. The training data was then fit to this object to identify the optimal hyperparameters.
  - Upon fitting the training set, the GridSearchCV object for each model was outputted. Subsequently, the optimal parameters were displayed using the best_params_ attribute, and the accuracy on the validation data was shown using the best_score_ attribute.
  - Finally, we assessed the performance of each model on the test data by calculating the accuracy using the score method. To further evaluate the predictions, a confusion matrix was generated for each model, comparing the test outcomes with the predicted outcomes.

Predictive Analysis (Classification)

- The subsequent tables presents the test data accuracy scores for each of the evaluated methods (SVM, Classification Trees, k-Nearest Neighbors, and Logistic Regression), facilitating a comparison of their performance on the test dataset to determine the optimal method.
- GitHub URL of the completed notebook: (https://github.com/oswaldofelizzola/Coursera-IBM-Data-Science-Professional/blob/fdc3e5636e43081ca45e5065b47b2332f3a7eaab/Applied%20Data%20Science%20Capstone/8%20SpaceX_Machine%20Learning%20Prediction_Part_5.ipynb)

Predictive Analysis (Classification)

TASK 1

Create a NumPy array from the column `Class` in `data`, by applying the method `to_numpy()` then assign it to the variable `Y`,make sure the output is a Pandas series (only one bracket df['name of column']).

In [8]:
```
Y = data['Class'].to_numpy()
Y.dtype
```

Out[8]:
```
dtype('int64')
```

TASK 2

Standardize the data in `X` then reassign it to the variable `X` using the transform provided below.

In [9]:
```
# students get this
transform = preprocessing.StandardScaler()
X = transform.fit_transform(X)
X
```

Out[9]:
```
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]])
```

Predictive Analysis (Classification)

TASK 3

Use the function train_test_split to split the data X and Y into training and test data. Set the parameter test_size to 0.2 and random_state to 2. The training data and test data should be assigned to the following labels.

`X_train, X_test, Y_train, Y_test`

In [10]:
```
X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size=0.2, random_state=2)
```

we can see we only have 18 test samples.

In [11]:
```
Y_test.shape
```

Out[11]:
```
(18,)
```

Predictive Analysis (Classification)

TASK 4

Create a logistic regression object then create a GridSearchCV object `logreg_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

In [12]:
```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

In [14]:
```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()

# Create a GridSearchCV object logreg_cv
logreg_cv = GridSearchCV(lr, parameters, cv=10)

#Fit the training data into the GridSearch object
logreg_cv.fit(X_train, Y_train)
```

Out[14]:
```
GridSearchCV(cv=10, estimator=LogisticRegression(),
             param_grid={'C': [0.01, 0.1, 1], 'penalty': ['l2'],
                         'solver': ['lbfgs']})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

We output the `GridSearchCV` object for logistic regression. We display the best parameters using the data attribute `best_params_` and the accuracy on the validation data using the data attribute `best_score_`.

In [15]:
```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8464285714285713
```

Predictive Analysis (Classification)

TASK 5

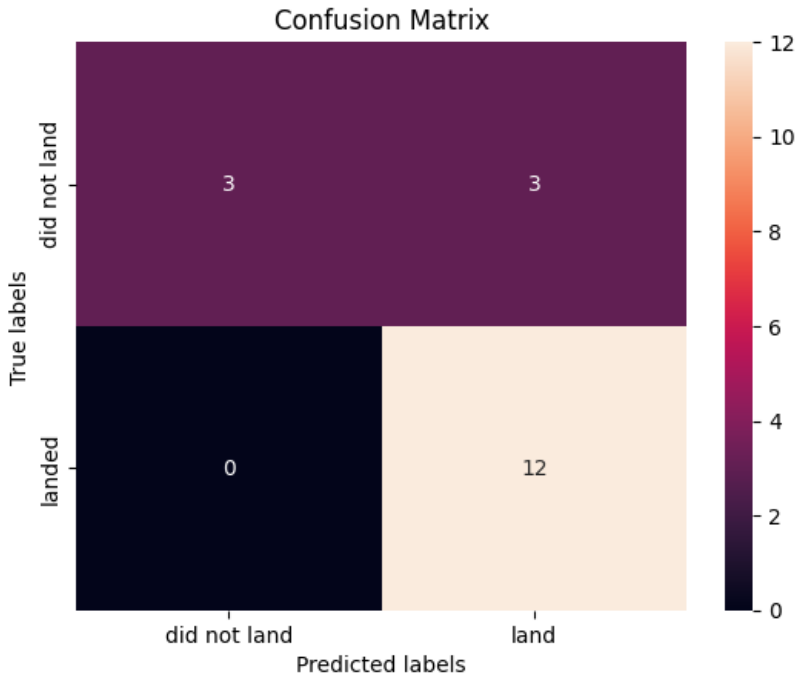
Calculate the accuracy on the test data using the method `score` :

In [16]:
```
print("Logistic Regression test data accuracy :",logreg_cv.score(X_test, Y_test))
```

```
Logistic Regression test data accuracy : 0.8333333333333334
```

Lets look at the confusion matrix:

In [17]:
```
yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



Predictive Analysis (Classification)

TASK 6

Create a support vector machine object then create a `GridSearchCV` object `svm_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

In [18]:
```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

In [19]:
```
# create a GridSearchCV object svm_cv with cv = 10
svm_cv = GridSearchCV(svm, parameters, cv= 10)

#Fit the training data into the GridSearch object
svm_cv.fit(X_train, Y_train)
```

Out[19]:
```
GridSearchCV(cv=10, estimator=SVC(),
             param_grid={'C': array([1.00000000e-03, 3.16227766e-02, 1.00000000e+00, 3.162
27766e+01,
       1.00000000e+03]),
                         'gamma': array([1.00000000e-03, 3.16227766e-02, 1.00000000e+00,
3.16227766e+01,
       1.00000000e+03]),
                         'kernel': ('linear', 'rbf', 'poly', 'rbf', 'sigmoid')})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [20]:
```
print("tuned hpyerparameters :(best parameters) ",svm_cv.best_params_)
print("accuracy :",svm_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoi
d'}
accuracy : 0.8482142857142856
```

Predictive Analysis (Classification)

TASK 7

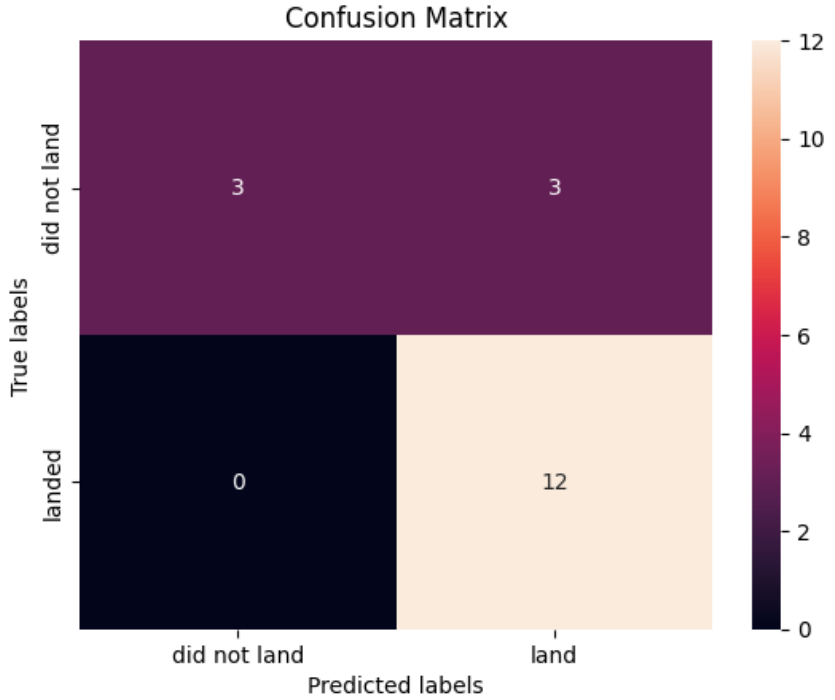
Calculate the accuracy on the test data using the method `score` :

In [21]:
```
print("SVM test data accuracy :",svm_cv.score(X_test, Y_test))
```

```
SVM test data accuracy : 0.8333333333333334
```

We can plot the confusion matrix

In [22]:
```
yhat=svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



Predictive Analysis (Classification)

TASK 8

Create a decision tree classifier object then create a `GridSearchCV` object `tree_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

In [23]:
```
parameters = {'criterion': ['gini', 'entropy'],
     'splitter': ['best', 'random'],
     'max_depth': [2*n for n in range(1,10)],
     'max_features': ['auto', 'sqrt'],
     'min_samples_leaf': [1, 2, 4],
     'min_samples_split': [2, 5, 10]}

tree = DecisionTreeClassifier()
```

In [24]:
```
# create a GridSearchCV object tree_cv with cv = 10
tree_cv = GridSearchCV(tree, parameters, cv= 10)

#Fit the training data into the GridSearch object
tree_cv.fit(X_train, Y_train)
```

Out[24]:
```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18],
                         'max_features': ['auto', 'sqrt'],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'splitter': ['best', 'random']})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [25]:
```
print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', 'max_depth': 8, 'max_features':
'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 5, 'splitter': 'best'}
accuracy : 0.8732142857142856
```

Predictive Analysis (Classification)

TASK 9

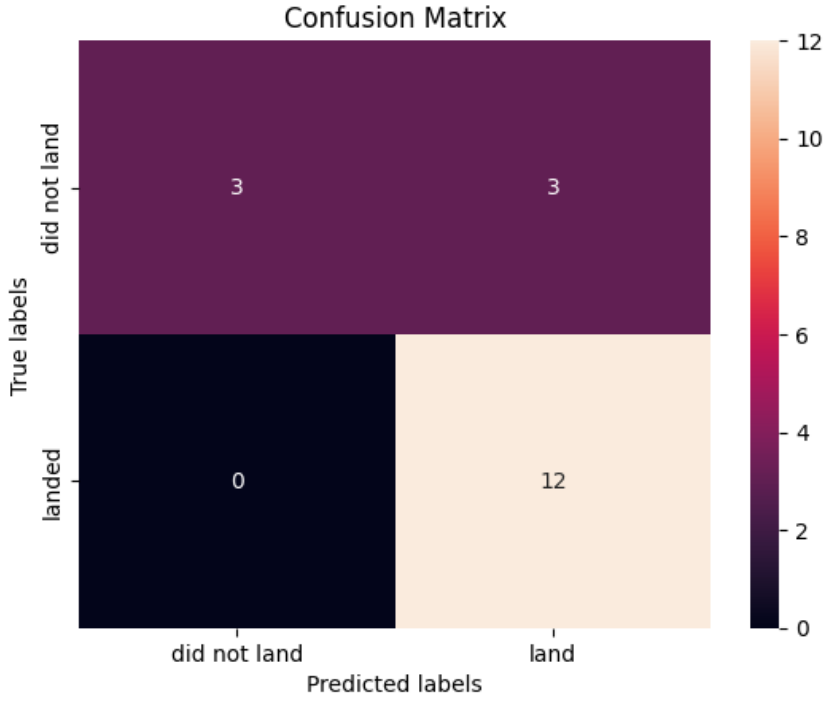
Calculate the accuracy of tree_cv on the test data using the method `score` :

In [26]:
```
print("Decision Tree accuracy on test set :",tree_cv.score(X_test, Y_test))
```

```
Decision Tree accuracy on test set : 0.8333333333333334
```

We can plot the confusion matrix

In [27]:
```
yhat = tree_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



Predictive Analysis (Classification)

TASK 10

Create a k nearest neighbors object then create a `GridSearchCV` object `knn_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

In [28]:
```
parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'p': [1,2]}

KNN = KNeighborsClassifier()
```

In [29]:
```
# create a GridSearchCV object knn_cv with cv = 10
knn_cv = GridSearchCV(KNN, parameters, cv= 10)

#Fit the training data into the GridSearch object
knn_cv.fit(X_train, Y_train)
```

Out[29]:
```
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                         'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'p': [1, 2]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [30]:
```
print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
print("accuracy :",knn_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}
accuracy : 0.8482142857142858
```

Predictive Analysis (Classification)

TASK 11

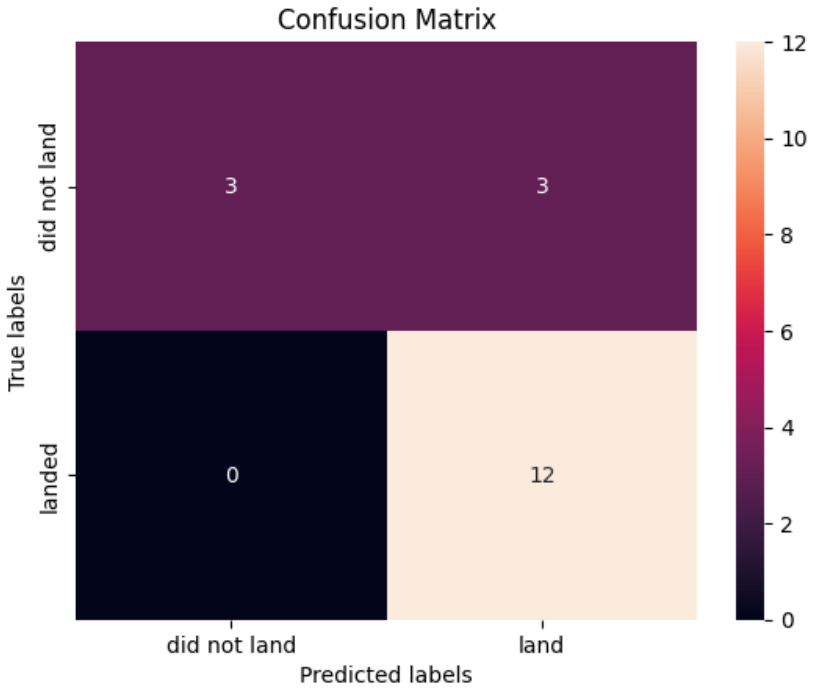
Calculate the accuracy of knn_cv on the test data using the method `score`:

In [31]:
```
knn_cv.score(X_test, Y_test)
```

Out[31]: 0.8333333333333334

We can plot the confusion matrix

In [32]:
```
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



Predictive Analysis (Classification)

TASK 12

Find the method performs best:

In [33]:

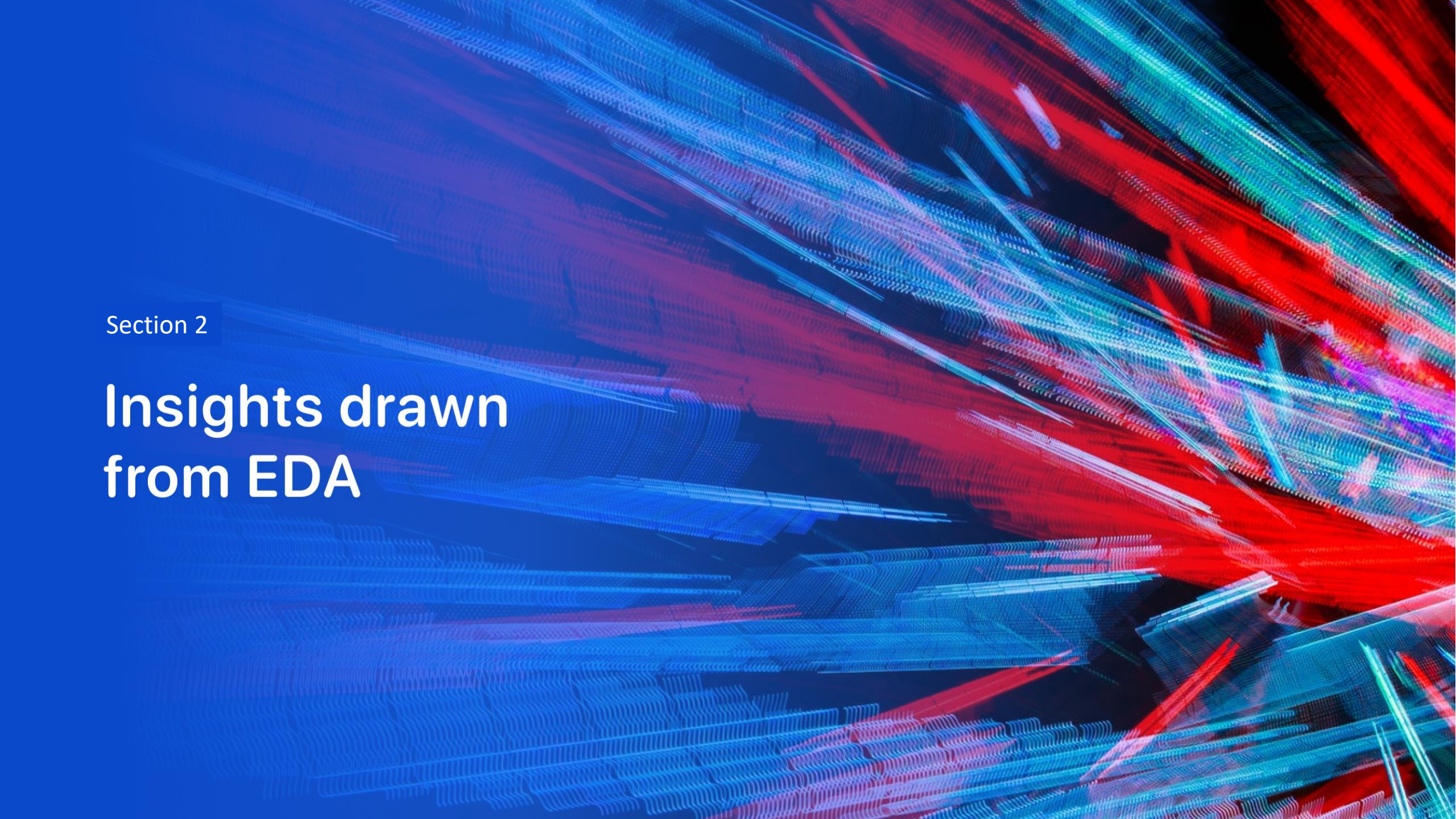
```
Report = pd.DataFrame({'Method' : ['Test Data Accuracy']})

knn_accuracy=knn_cv.score(X_test, Y_test)
Decision_tree_accuracy=tree_cv.score(X_test, Y_test)
SVM_accuracy=svm_cv.score(X_test, Y_test)
Logistic_Regression=logreg_cv.score(X_test, Y_test)

Report['Logistic_Reg'] = [Logistic_Regression]
Report['SVM'] = [SVM_accuracy]
Report['Decision Tree'] = [Decision_tree_accuracy]
Report['KNN'] = [knn_accuracy]




Report.transpose()
```

Out[33]:

	0
Method	Test Data Accuracy
Logistic_Reg	0.833333
SVM	0.833333
Decision Tree	0.833333
KNN	0.833333

Results

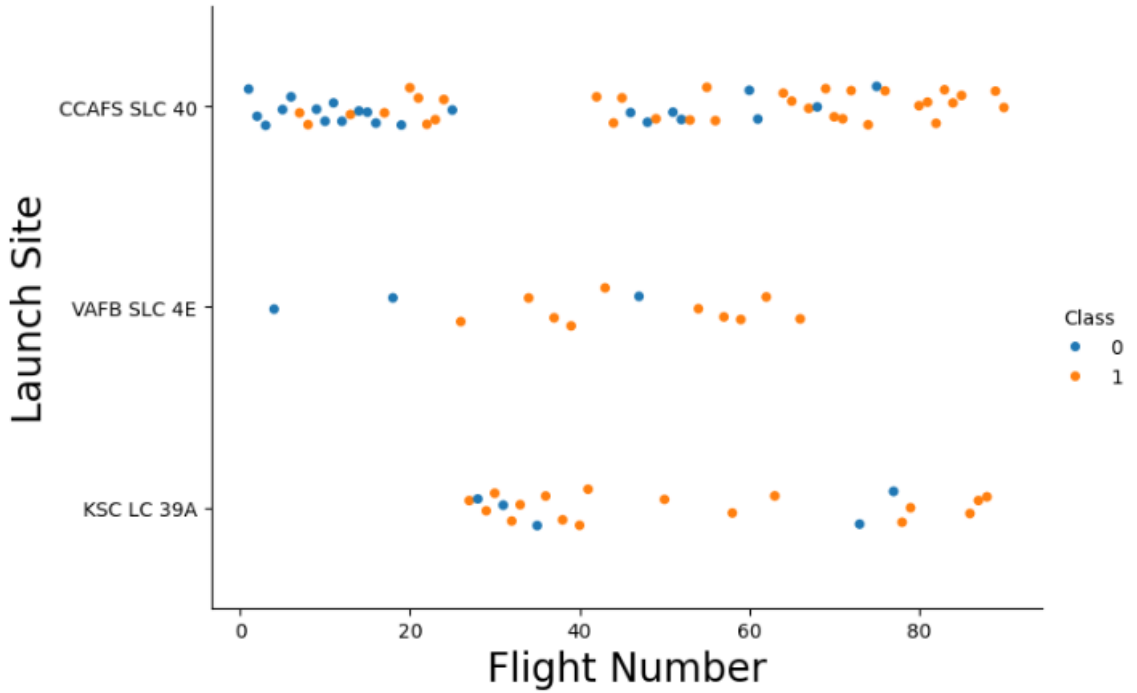
- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results


Section 2
Insights drawn from EDA

Flight Number vs. Launch Site

- It is observed that takeoff success exhibits a positive correlation with the number of flights at each of the three launch sites.
  - CCAFS SLC 40: This launch site shows the greatest dispersion of values, although a significant improvement is noted as the number of flights increases.
  - VAFB SLC 4E: Although it has few flights, it has been a very successful launch site , comparatively.
  - KSC LC 39A: This launch site has the highest number of successes per flight number.

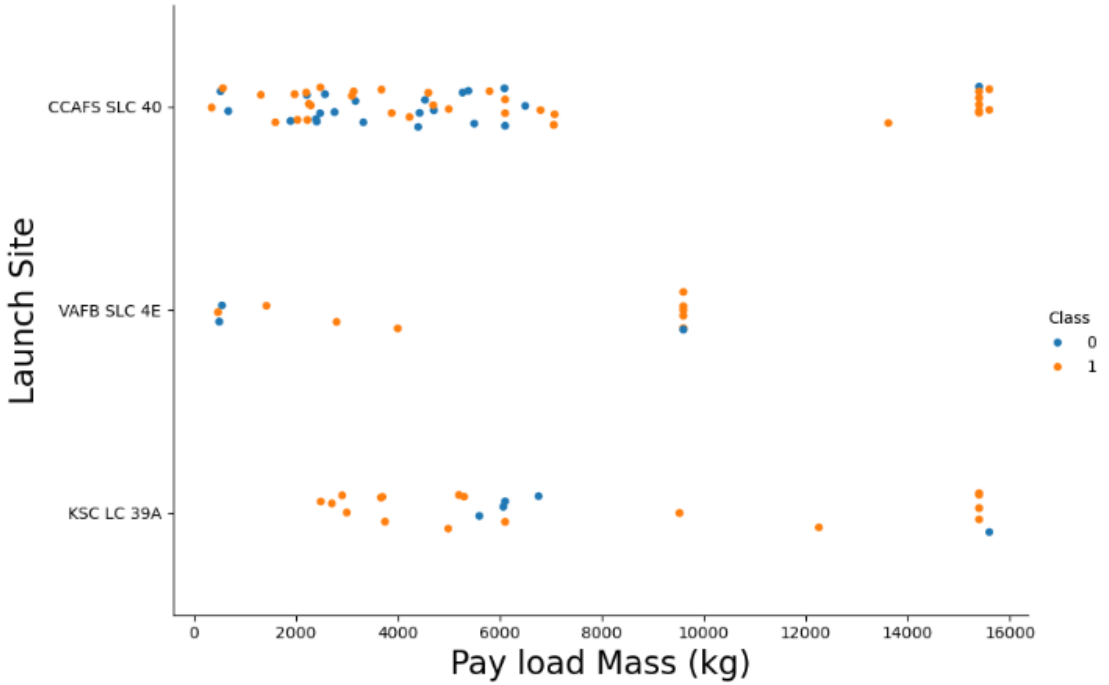
In [5]:
```
# Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, a
sns.catplot(y="LaunchSite", x="FlightNumber", hue="Class", data=df, aspect = 1.5 ,height=5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Launch Site",fontsize=20)
plt.show()
```



Payload vs. Launch Site

- As payload increases, port performance improves.
  - CCAFS SLC 40: Features many low-payload launches, but significantly improves when launching large payloads.
  - VAFB SLC 4E: Used less, and with lower payload launches than the others.
  - KSC LC 39A: Has the best distribution of launches per payload, and significant improvement can be seen with increasing payload.

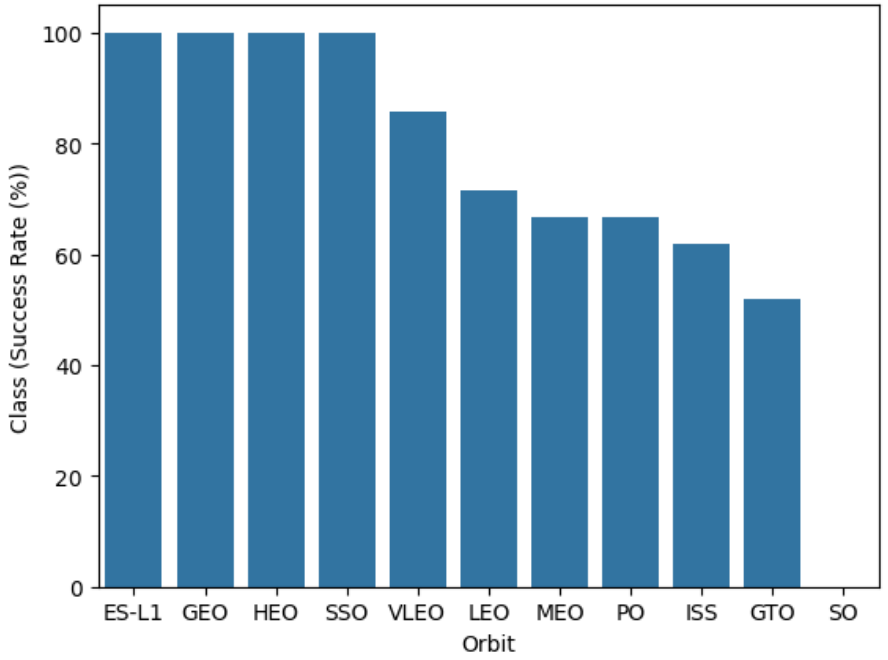
```
In [6]: # Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch si
sns.catplot(y="LaunchSite", x="PayloadMass", hue="Class", data=df, aspect = 1.5 ,height=6)
plt.xlabel("Pay load Mass (kg)",fontsize=20)
plt.ylabel("Launch Site",fontsize=20)
plt.show()
```



Success Rate vs. Orbit Type

- High Success Rates: Several orbit types demonstrate a 100% success rate, including ES-L1, GEO, HEO, and SSO. This suggests a high degree of reliability for launches targeting these orbits.
- Varied Success Rates: The success rates vary significantly across the different orbit types. VLEO shows a slightly lower success rate (around 86%), followed by LEO (approximately 72%).
- Lower Success Rates: MEO and PO have similar success rates, around 67%. ISS has a slightly lower rate at approximately 62%.
- Lowest Success Rates: GTO and SO exhibit the lowest success rates among the orbits presented, with GTO around 52% and SO slightly higher.
- The varying success rates might reflect the complexity and challenges associated with reaching and operating in different orbits. Orbits like GTO, which are transfer orbits to geostationary orbit, might involve more complex maneuvers and therefore have a higher chance of failure.

In [7]:
```
# HINT use groupby method on Orbit column and get the mean of Class column
sr_df = df.groupby('Orbit')['Class'].mean().reset_index().sort_values(by='Class', ascending=Fals
sr_df['Class'] = sr_df['Class'] * 100


sns.barplot(data=sr_df, x='Orbit', y='Class')
plt.xlabel('Orbit')
plt.ylabel('Class (Success Rate (%))')
plt.show()
```

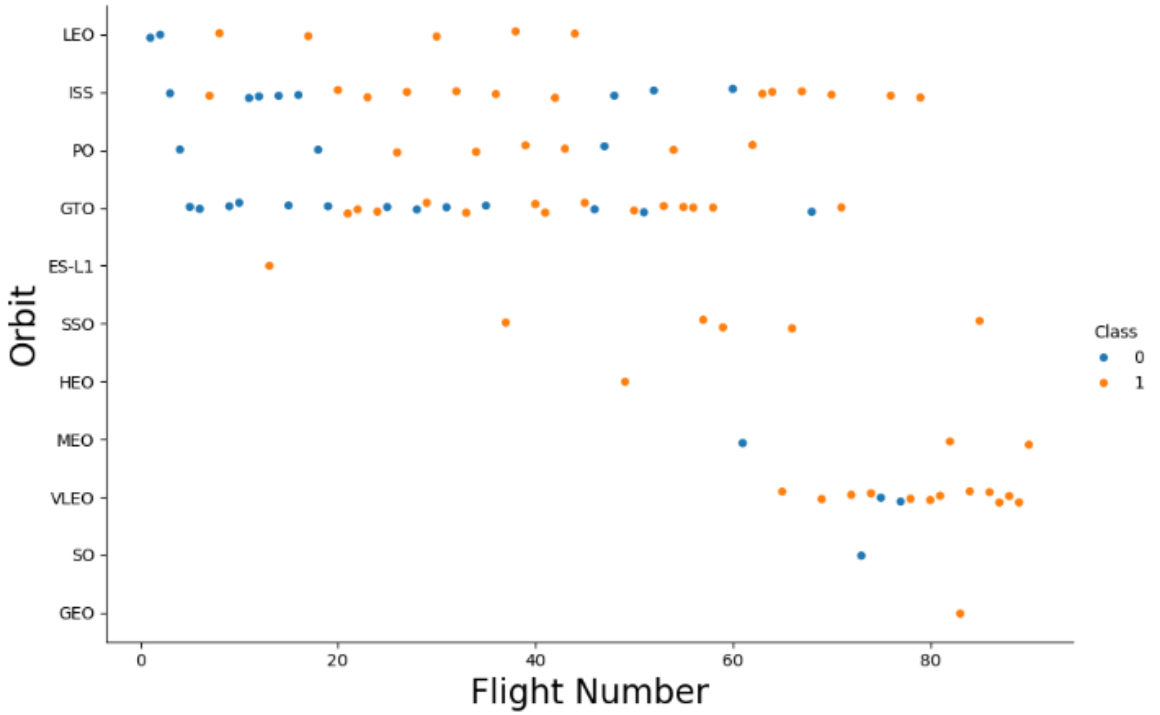


Flight Number vs. Orbit Type

- Lower orbits show more variable performance, demonstrating that the technology is improving.
- In the higher orbits, a significant improvement is seen compared to the others, demonstrating confidence in the technology.

In [8]:

```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue
sns.catplot(y="Orbit", x="FlightNumber", hue="Class", data=df, aspect = 1.5 ,height=6)

plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```

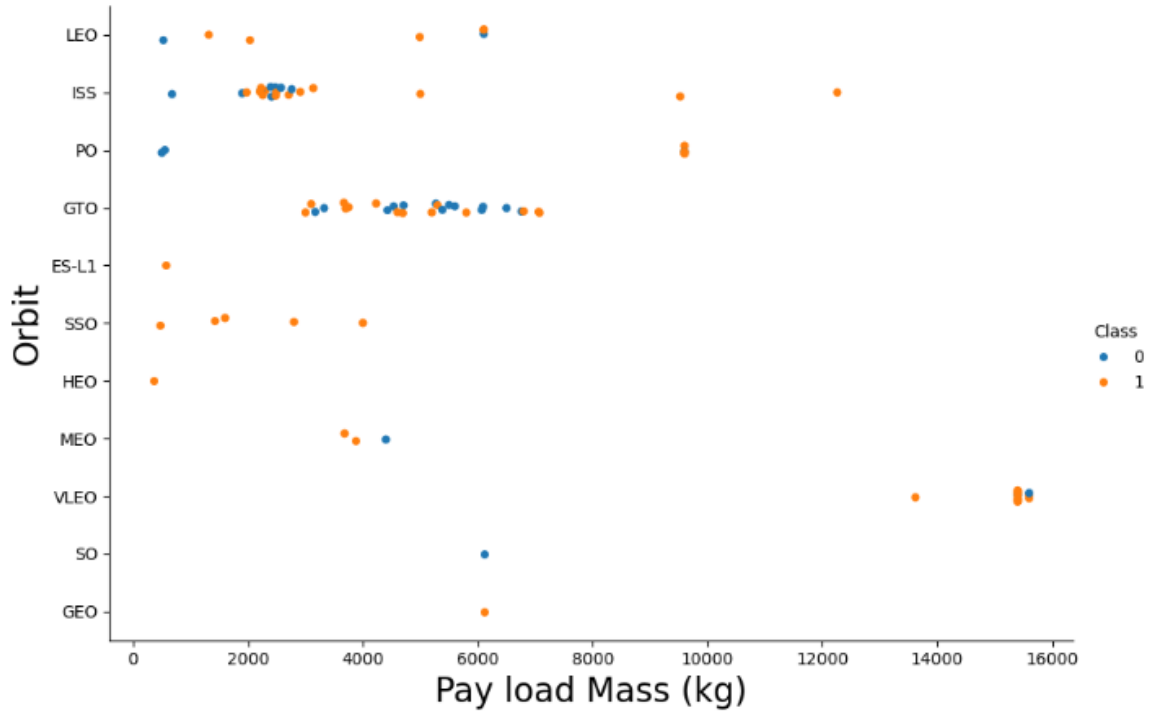


Payload vs. Orbit Type

- It can be seen how the payload varies significantly for the different orbits, especially since GEO only handles a single weight of 6,000 kg, indicating that it is a single type of satellite. Meanwhile, GTO displays greater weight variability, which indicates how complicated it can be to launch into that orbit due to the variability of satellite types.

In [9]:

```
# Plot a scatter point chart with x axis to be Payload Mass and y axis to be the Orbit, and hue
sns.catplot(y="Orbit", x="PayloadMass", hue="Class", data=df, aspect = 1.5 ,height=6)

plt.xlabel("Pay load Mass (kg)",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```

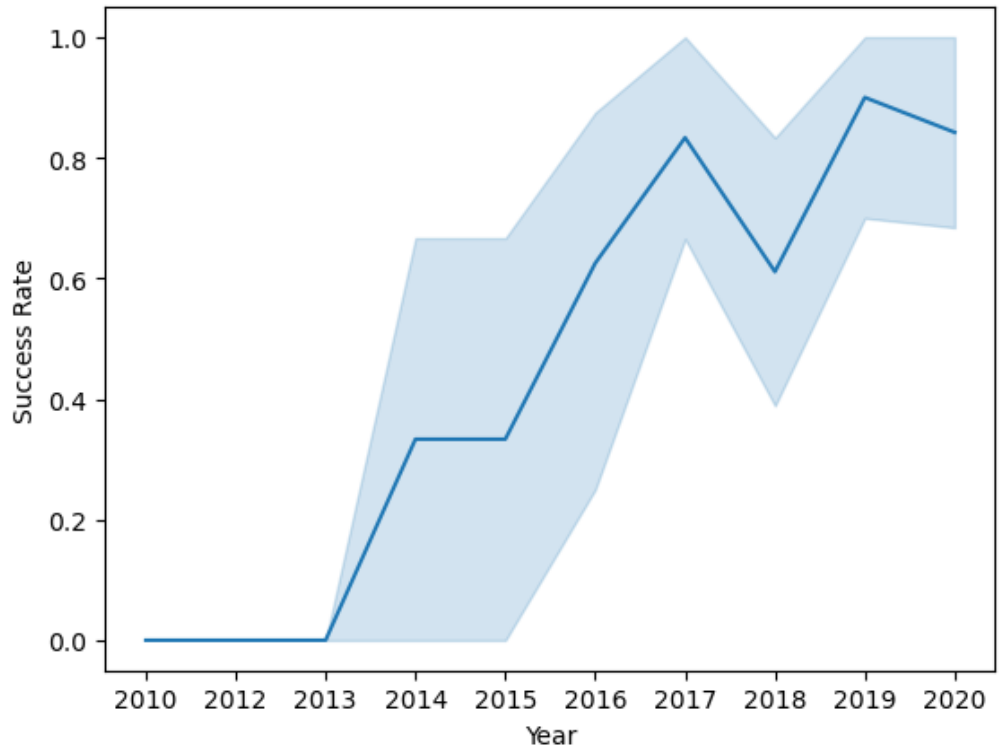


Launch Success Yearly Trend

- It's clear how launching success rate has been improving over time. Despite a decline in 2018 due to increased shipping, the organization was able to resume its path of improvement in subsequent years.

In [11]:

```
# Plot a line chart with x axis to be the extracted year and y axis to be the success rate
sns.lineplot(data=df, x="Date", y="Class")
plt.xlabel("Year")
plt.ylabel("Success Rate")
plt.show()
```



All Launch Site Names

- You can see how there are 4 launching sites, where two of them are very close CCAFS SLC-40 & CCAFS LC-40

In [10]:
```
%sql SELECT DISTINCT LAUNCH_SITE as "Launch_Sites" FROM SPACEXTBL;
```

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

Out[10]:

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

Launch Site Names Begin with 'CCA'

- It can be seen how the first launches occurred within a period of approximately three years, with NASA being the primary customer for all launches. Furthermore, all were launched from CCAFS LC-40.

In [11]:
```
%sql SELECT * FROM 'SPACEXTBL' WHERE Launch_Site LIKE 'CCA%' LIMIT 5;
```

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

Out[11]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Total Payload Mass

- It can be seen that the largest accumulated payload was made for NASA, indicating that the main client of SpaceX

```
In [12]: %sql SELECT SUM(PAYLOAD_MASS__KG_) as "Total Payload Mass(Kgs)", Customer FROM 'SPACEXTBL' WHERE Cu
 * sqlite:///my_data1.db
Done.
```

Out[12]:

Total Payload Mass(Kgs)	Customer
45596	NASA (CRS)

Average Payload Mass by F9 v1.1

- It is seen that the average load is 2534.6 kg, where the client is MDA, y en la version F9 v1.1 B1003

```
In [13]: %sql SELECT AVG(PAYLOAD_MASS__KG_) as "Payload Mass Kgs", Customer, Booster_Version FROM 'SPACEXTBI
 * sqlite:///my_data1.db
Done.
```

Out[13]:

Payload Mass Kgs	Customer	Booster_Version
2534.6666666666665	MDA	F9 v1.1 B1003

First Successful Ground Landing Date

- It can be seen how the first successful landing on the ground pad occurred almost 5 years after the first SpaceX launch, indicating how complicated the feat was.

In [25]:
```
%sql SELECT MIN(DATE) FROM 'SPACEXTBL' WHERE "Landing_Outcome" = "Success (ground pad)";
```

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

Out[25]:

MIN(DATE)
2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

- It can be seen how the payload of the JCSAT and SES (geostationary satellite) families are what justify the use of the F9 FT B102X boosters. This indicates the enormous opportunities to reduce costs by reusing the boosters.

In [26]:
```
%sql SELECT DISTINCT Booster_Version, Payload FROM SPACEXTBL WHERE "Landing_Outcome" = "Success (d
```

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

Out[26]:

Booster_Version	Payload
F9 FT B1022	JCSAT-14
F9 FT B1026	JCSAT-16
F9 FT B1021.2	SES-10
F9 FT B1031.2	SES-11 / EchoStar 105

Total Number of Successful and Failure Mission Outcomes

- It is appreciated how successful the missions have been, with success rates exceeding 98%.

```
In [27]: %sql SELECT "Mission_Outcome", COUNT("Mission_Outcome") as Total FROM SPACEXTBL GROUP BY "Mission_(
 * sqlite:///my_data1.db
Done.
```

Out[27]:

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

Boosters Carried Maximum Payload

- It can be seen how the maximum payload is from the Starlink (low orbit communications satellites), which indicates how profitable this type of payload can be for SpaceX, thanks to the recycling of the Boosters.

In [28]:
```
%sql SELECT "Booster_Version",Payload, "PAYLOAD_MASS__KG_" FROM SPACEXTBL WHERE "PAYLOAD_MASS__KG_"
```

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

Out[28]:

Booster_Version	Payload	PAYLOAD_MASS__KG_
F9 B5 B1048.4	Starlink 1 v1.0, SpaceX CRS-19	15600
F9 B5 B1049.4	Starlink 2 v1.0, Crew Dragon in-flight abort test	15600
F9 B5 B1051.3	Starlink 3 v1.0, Starlink 4 v1.0	15600
F9 B5 B1056.4	Starlink 4 v1.0, SpaceX CRS-20	15600
F9 B5 B1048.5	Starlink 5 v1.0, Starlink 6 v1.0	15600
F9 B5 B1051.4	Starlink 6 v1.0, Crew Dragon Demo-2	15600
F9 B5 B1049.5	Starlink 7 v1.0, Starlink 8 v1.0	15600
F9 B5 B1060.2	Starlink 11 v1.0, Starlink 12 v1.0	15600
F9 B5 B1058.3	Starlink 12 v1.0, Starlink 13 v1.0	15600
F9 B5 B1051.6	Starlink 13 v1.0, Starlink 14 v1.0	15600
F9 B5 B1060.3	Starlink 14 v1.0, GPS III-04	15600
F9 B5 B1049.7	Starlink 15 v1.0, SpaceX CRS-21	15600

2015 Launch Records

- It can be seen that the F9 v1.1 boosters B1012 and B1015 failed to land, although their mission was successful. All of them launched from CCAFS LC-40.

In [32]:
```
%sql SELECT substr(Date,6,2), substr(Date, 0, 5),"Booster_Version", "Launch_Site", Payload, "PAYLOAD_MASS__KG_", "Mission_O
```

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

Out[32]:

substr(Date,6,2)	substr(Date, 0, 5)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Mission_Outcome	Landing_Outcome
01	2015	F9 v1.1 B1012	CCAFS LC-40	SpaceX CRS-5	2395	Success	Failure (drone ship)
04	2015	F9 v1.1 B1015	CCAFS LC-40	SpaceX CRS-6	1898	Success	Failure (drone ship)

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

- It is appreciated that all the landings were successful, indicating how successful SpaceX has been in maintaining consistency, considering the great challenge that this technology represents.

In [38]:
```
%sql SELECT * FROM SPACEXTBL WHERE "Landing_Outcome" LIKE 'Success%' AND (Date BETWEEN '2010-06-04' AND '2017-03-20') ORDER
```

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

Out[38]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcc
2017-02-19	14:39:00	F9 FT B1031.1	KSC LC-39A	SpaceX CRS-10	2490	LEO (ISS)	NASA (CRS)	Success	Success (gro
2017-01-14	17:54:00	F9 FT B1029.1	VAFB SLC-4E	Iridium NEXT 1	9600	Polar LEO	Iridium Communications	Success	Success (dı
2016-08-14	5:26:00	F9 FT B1026	CCAFS LC-40	JCSAT-16	4600	GTO	SKY Perfect JSAT Group	Success	Success (dı
2016-07-18	4:45:00	F9 FT B1025.1	CCAFS LC-40	SpaceX CRS-9	2257	LEO (ISS)	NASA (CRS)	Success	Success (gro
2016-05-27	21:39:00	F9 FT B1023.1	CCAFS LC-40	Thaicom 8	3100	GTO	Thaicom	Success	Success (dı
2016-05-06	5:21:00	F9 FT B1022	CCAFS LC-40	JCSAT-14	4696	GTO	SKY Perfect JSAT Group	Success	Success (dı
2016-04-08	20:43:00	F9 FT B1021.1	CCAFS LC-40	SpaceX CRS-8	3136	LEO (ISS)	NASA (CRS)	Success	Success (dı
2015-12-22	1:29:00	F9 FT B1019	CCAFS LC-40	OG2 Mission 2 11 Orbcomm-OG2 satellites	2034	LEO	Orbcomm	Success	Success (gro


Section 3
Launch Sites
Proximities Analysis

All launch sites

[9]:

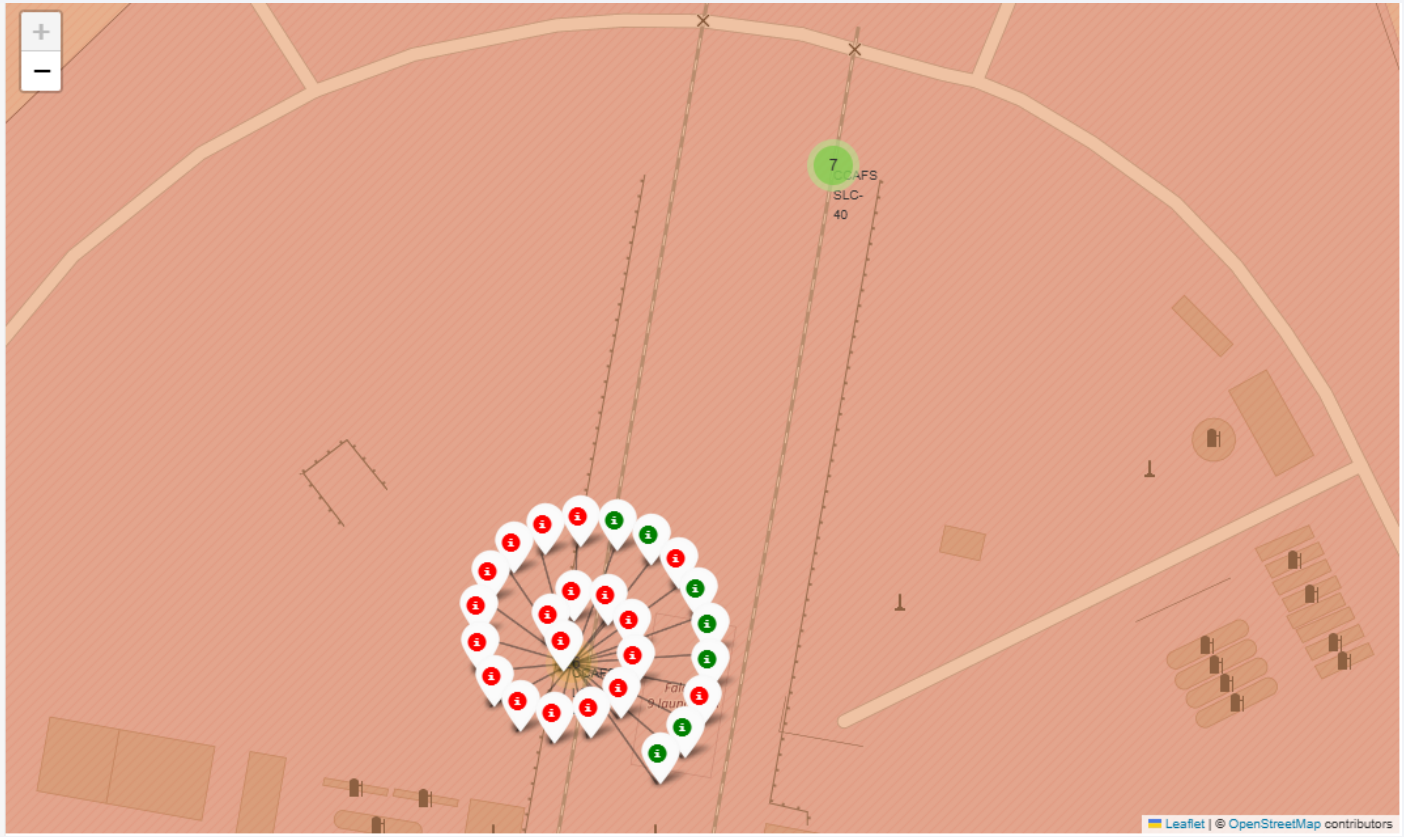
- It can be seen how all the launches occur near the equator, to take advantage of the Earth's rotation.

<Folium Map Screenshot 2>

- Replace <Folium map screenshot 2> title with an appropriate title
- Explore the folium map and make a proper screenshot to show the color-labeled launch outcomes on the map
- Explain the important elements and findings on the screenshot

Color-labeled launch outcomes

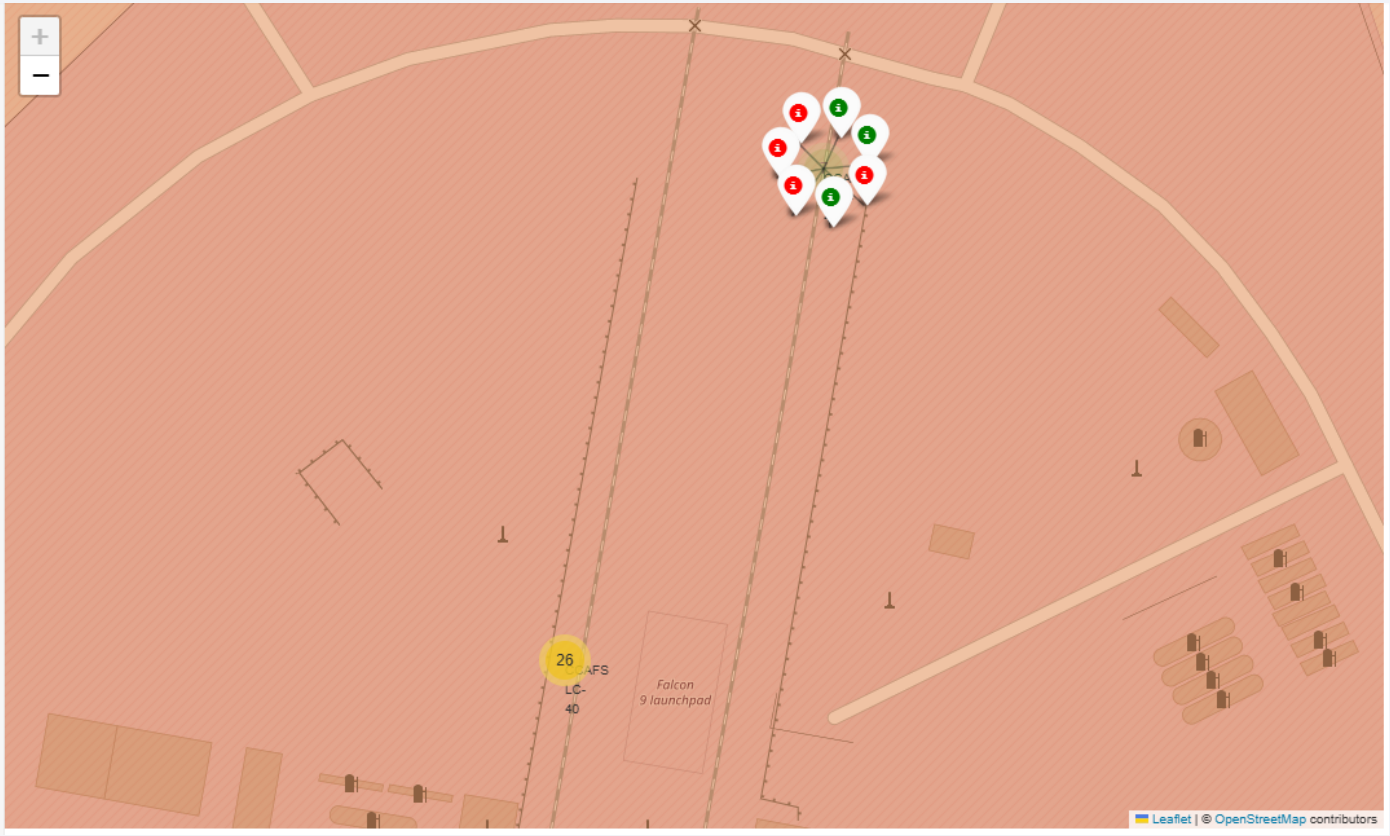
[16]:



- Florida launch pad, you can see how the first launches were unsuccessful, but as time goes by they become more successful
- It can be seen that it is the launch pad with the most launches

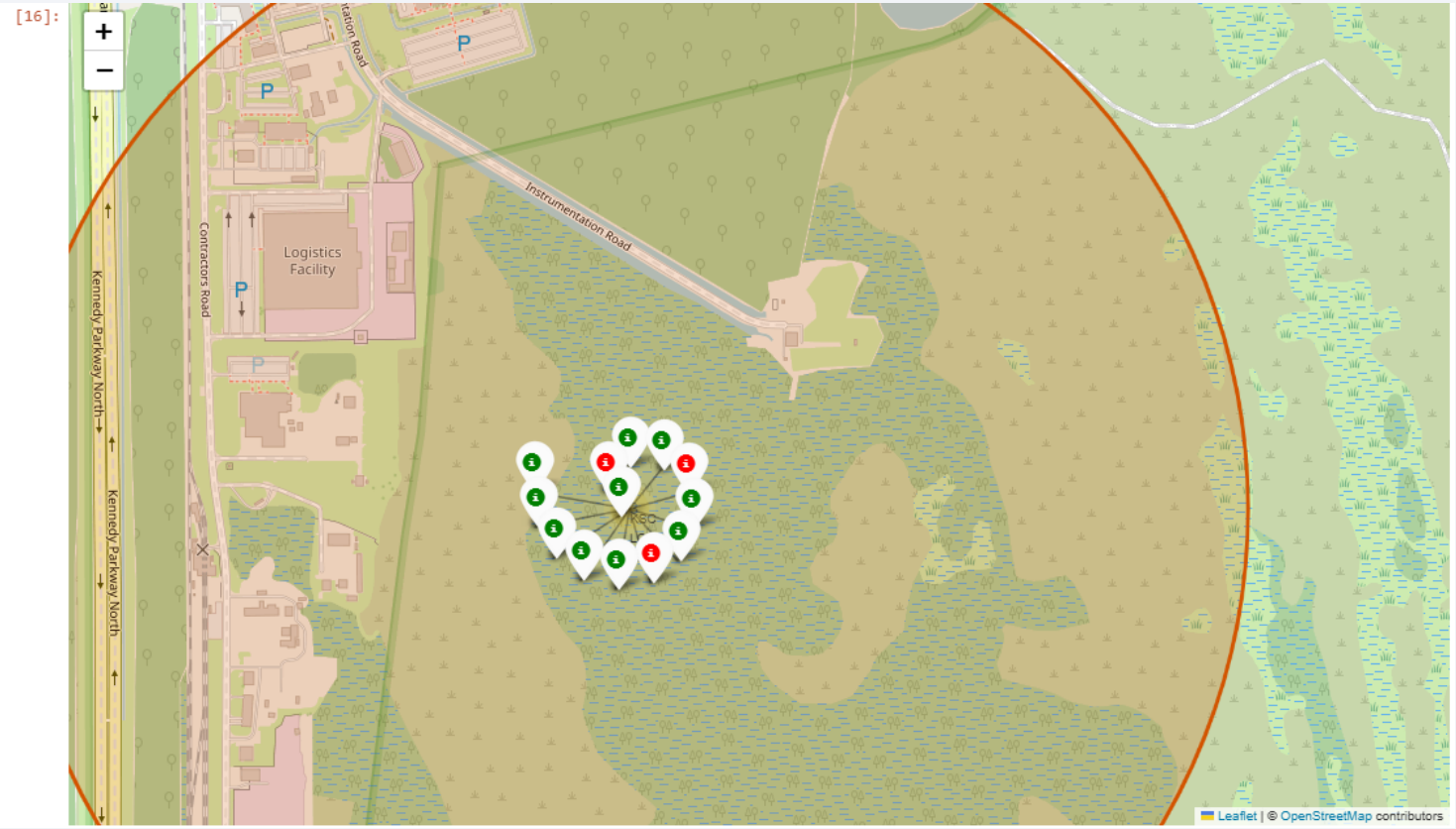
Color-labeled launch outcomes

[16]:



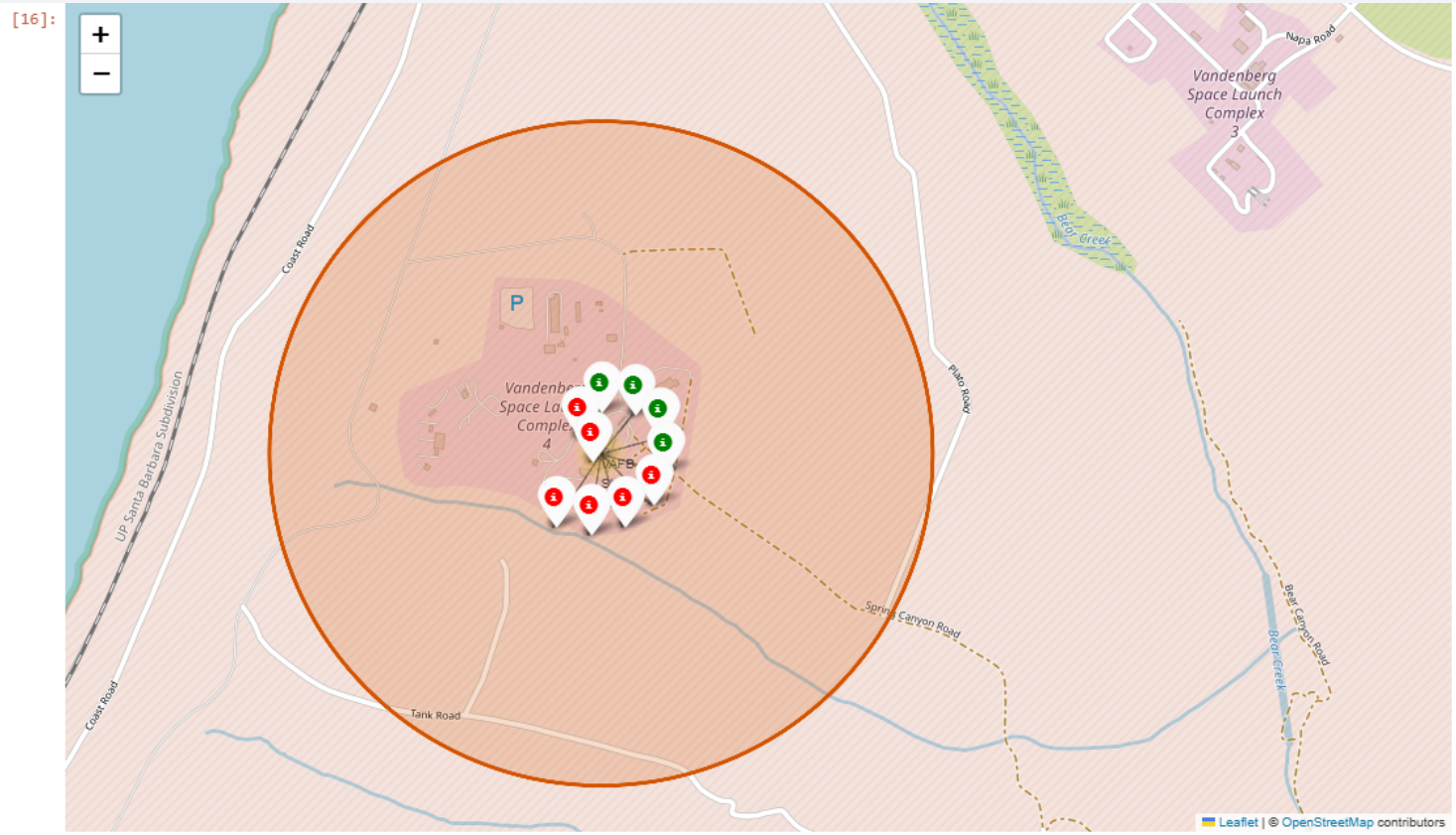
- Florida launch, you can see how the first launches were unsuccessful, but as time goes by they become more successful
- It is the one with the worst performance

Color-labeled launch outcomes



- Florida launch pad, you can see how almost all launches were successful, but as time goes by they become even more successful
- It is the best performing launch pad of all

Color-labeled launch outcomes



- California launch pad, you can see how the first launches were unsuccessful, but as time goes by they become more successful, but but in the end it presents the problems of failures again

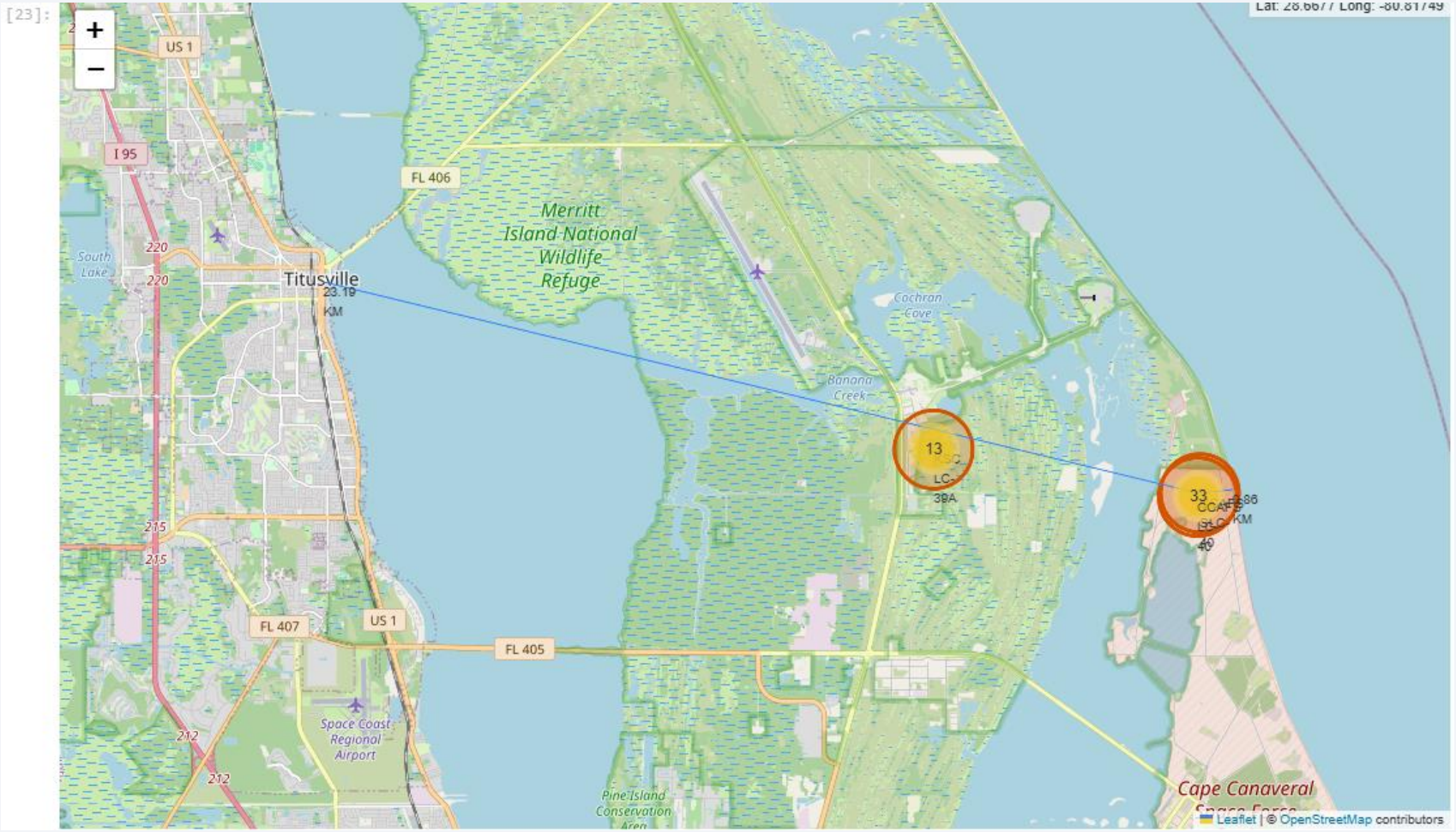
<Folium Map Screenshot 3>

- Replace <Folium map screenshot 3> title with an appropriate title
- Explore the generated folium map and show the screenshot of a selected launch site to its proximities such as railway, highway, coastline, with distance calculated and displayed
- Explain the important elements and findings on the screenshot

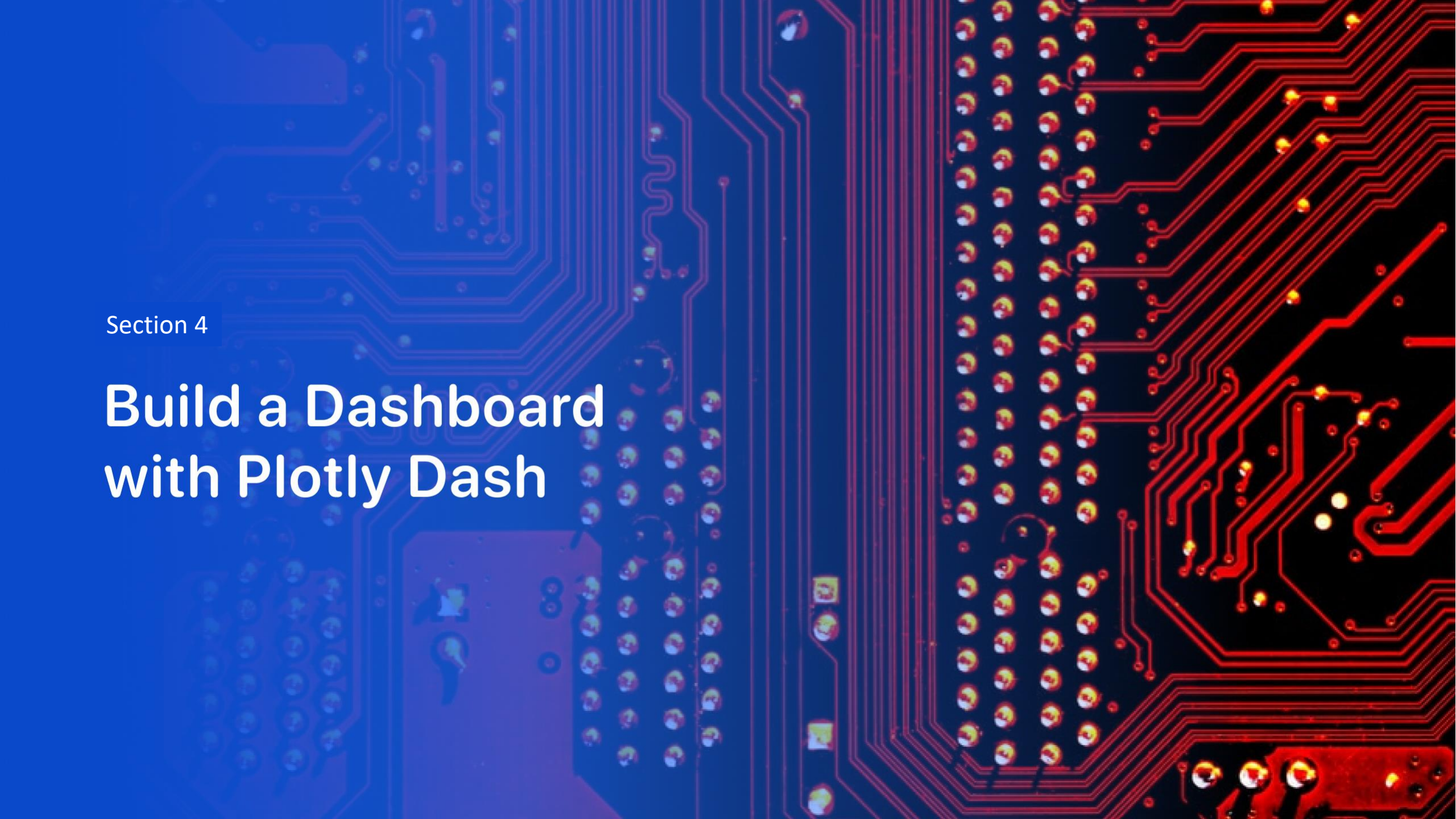
Selected launch site to its proximities

- It seemed that the distance to the sea was less than a kilometer, and that the only roads nearby were those used internally by NASA. This was evident for security reasons.

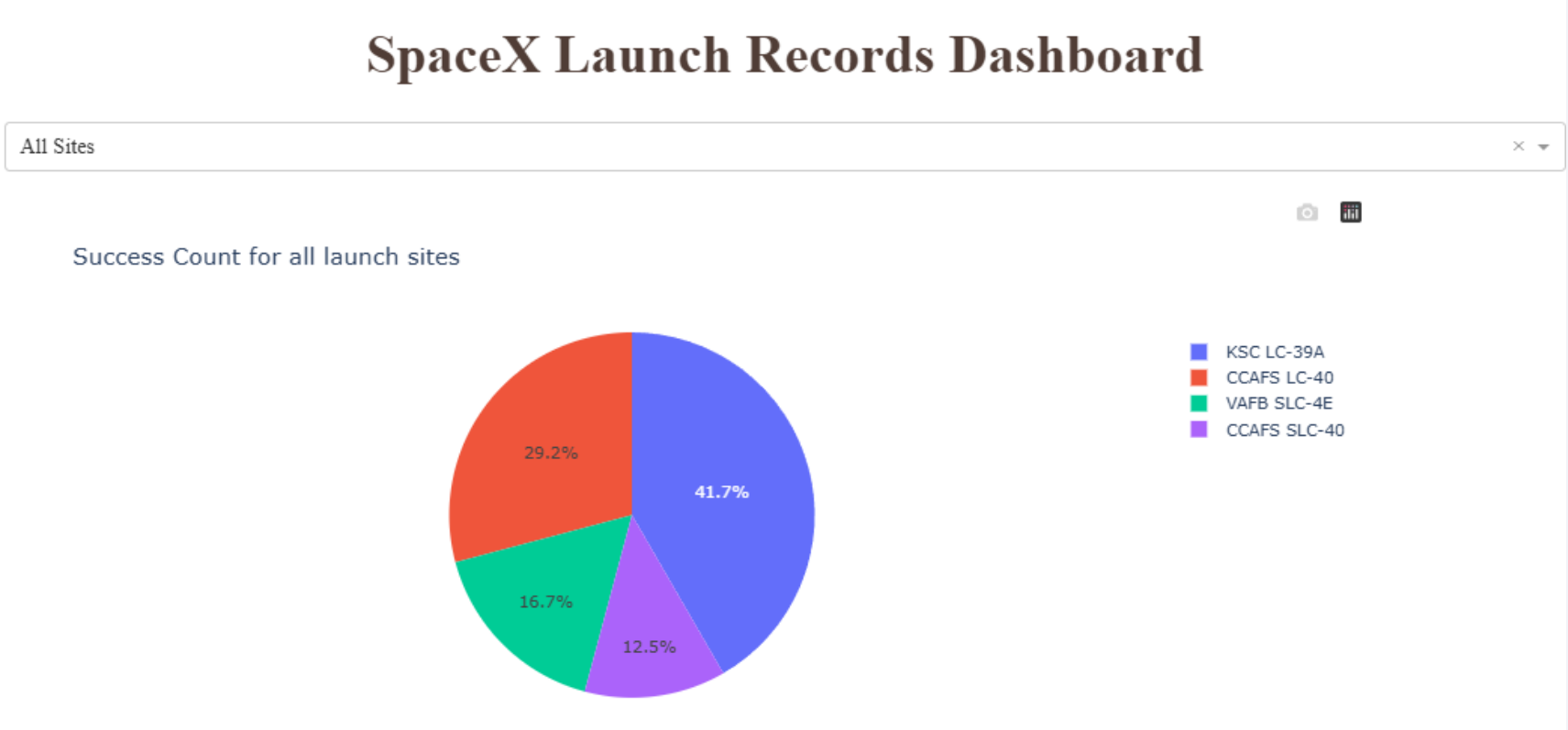
Selected launch site to its proximities



- It seemed that the distance to the nearest town was more than 20 kilometers, and that the only roads nearby were the one used internally by NASA. They were separated by water in all directions.

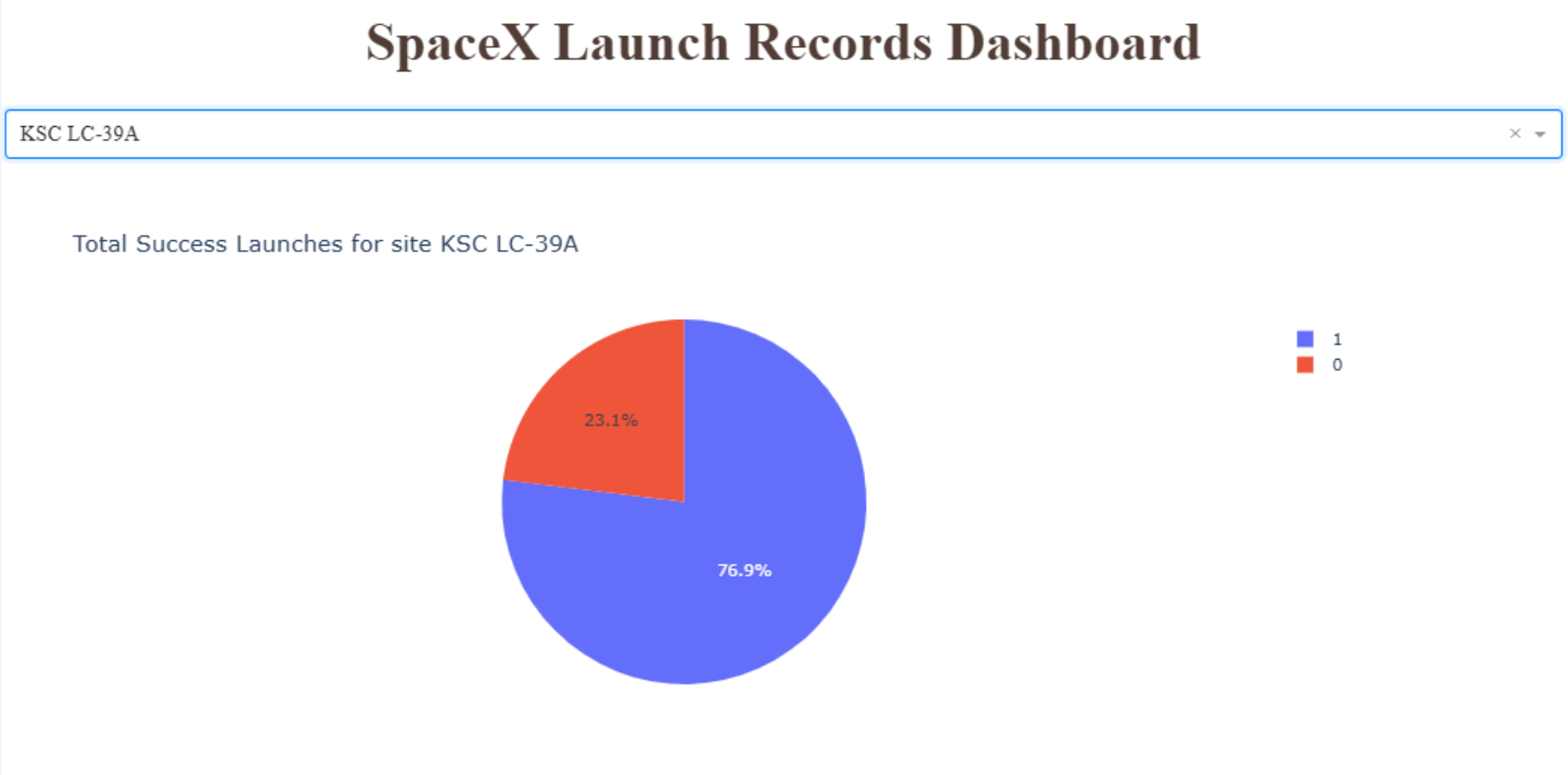

Section 4
Build a Dashboard with Plotly Dash

launch success count for all sites



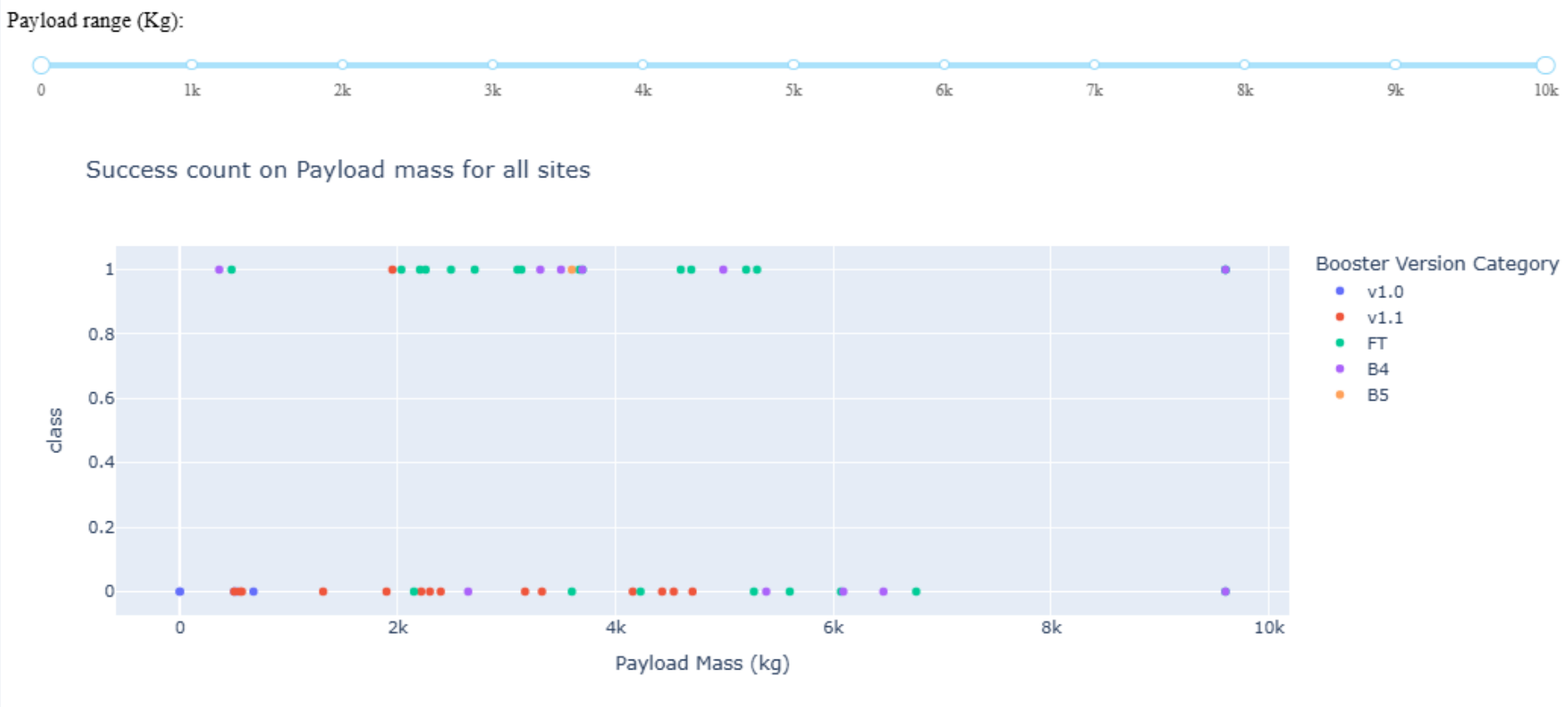
- It can be seen that the most used launch pad is KSC LC-39A, accounting for 41.7% of the launches, while CCAFS SLC-40 is the least used.

launch site with highest launch success ratio



- Not only is the KSC LC-39A the most used launch pad, but it's also the most successful, with a 76.9% success rate. This tells us why it's the most used.

Payload vs. Launch Outcome scatter plot



- It can be seen that the FT booster has the best performance in weights up to 6000 kg. The V1.1 booster has the worst performance, partly because it is the newest, so its technology is not yet mature.

Section 5

Predictive Analysis (Classification)

Classification Accuracy

- In reality, all models are equal, with a performance of 0.8333. Therefore, it cannot be said that one model is truly better than the others.

In [33]:

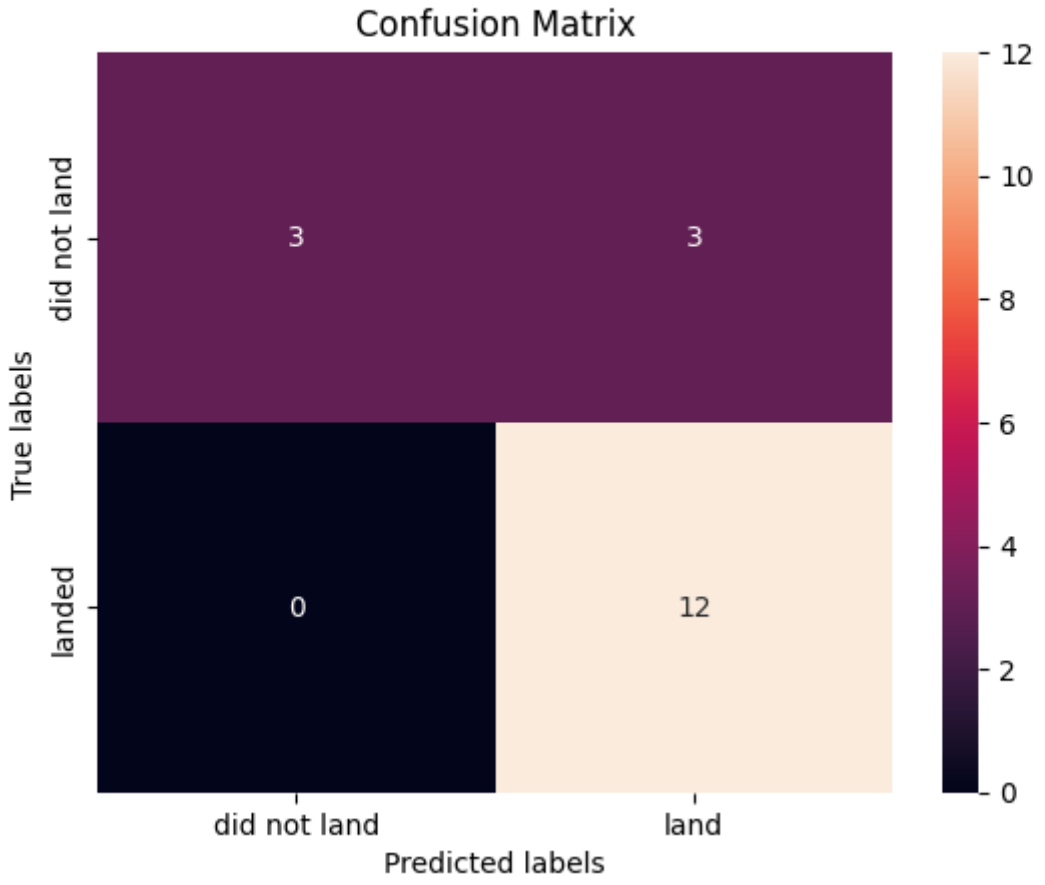
```
Report = pd.DataFrame({'Method' : ['Test Data Accuracy']})

knn_accuracy=knn_cv.score(X_test, Y_test)
Decision_tree_accuracy=tree_cv.score(X_test, Y_test)
SVM_accuracy=svm_cv.score(X_test, Y_test)
Logistic_Regression=logreg_cv.score(X_test, Y_test)

Report['Logistic_Reg'] = [Logistic_Regression]
Report['SVM'] = [SVM_accuracy]
Report['Decision Tree'] = [Decision_tree_accuracy]
Report['KNN'] = [knn_accuracy]



Report.transpose()
```

Out[33]:

	0
Method	Test Data Accuracy
Logistic_Reg	0.833333
SVM	0.833333
Decision Tree	0.833333
KNN	0.833333

Confusion Matrix

```
In [32]: yhat = knn_cv.predict(X_test)
         plot_confusion_matrix(Y_test,yhat)
```



- This confusion matrix reveals a model that has a high recall for the "landed" class, meaning it is very good at identifying actual landing events. The accuracy is also quite good at 83.3%.
- However, there are some false positives, indicating that the model occasionally predicts a landing when it doesn't happen. The zero false negatives for the "landed" class is a significant strength in scenarios where missing a positive event is costly.
- The performance on the "did not land" class seems less strong based on the lower true negatives and the presence of false positives.
- Overall, the model shows promising performance, particularly in its ability to correctly identify successful landings.

Conclusions

- The success rates of launches vary among the different launch sites. Specifically, CCAFS LC-40 has a success rate of 60%, whereas KSC LC-39A and VAFB SLC 4E each demonstrate a success rate of 77%.
- The data suggests a positive correlation between flight number and launch success rate at all three sites. Notably, VAFB SLC 4E achieves a 100% success rate subsequent to flight 50, while both KSC LC 39A and CCAFS SLC 40 attain this level after the 80th flight.
- Upon observation of the scatter plot depicting Payload against Launch Site, it is evident that the VAFB-SLC launch site has not accommodated launches with a payload mass of over 10,000.
- The analysis reveals that orbits ES-L1, GEO, HEO, and SSO achieve a 100% success rate, representing the highest performance. Conversely, the SO orbit records the lowest success rate at 0%.

Conclusions

- Analysis suggests a correlation between the number of flights and launch success for LEO orbit. However, this relationship does not appear to hold for launches into GTO orbit.
- When considering heavy payloads, Polar, LEO, and ISS orbits exhibit a higher rate of successful landings. However, for GTO orbit, a clear distinction is not readily apparent due to the presence of both positive and negative landing outcomes (unsuccessful missions).
- It is observed that the success rate demonstrated a continuous increase throughout the period spanning 2013 to 2020
- In summary, the predictive analysis models exhibits promising performance, with a notable strength in the accurate identification of successful landings.

Appendix

- Include any relevant assets like Python code snippets, SQL queries, charts, Notebook outputs, or data sets that you may have created during this project

Thank you!