IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Salvador Francisco
Orjuela Echandía
January, 21, 2024

Salvadororjuela@gmail.com

Outline

- Executive Summary
- Introduction
- Methodology
- Results
- Conclusion

Executive Summary

- Summary of methodologies
  - Data Collection through API and Web Scrapping
  - Data Wrangling
  - Exploratory Data Analysis with SQL
  - Exploratory Data Analysis with Data Visualization
  - Interactive Visual Analytics with Folium
  - Machine Learning Evaluation and Prediction
- Summary of all results
  - Exploratory Data Analysis result
  - Interactive analytics in screenshots
  - Predictive Analytics result

Introduction

- **Project background and context**

- Space X advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is because Space X can reuse the first stage. Therefore, if we can determine if the first stage will land, we can determine the cost of a launch. This information can be used if an alternate company wants to bid against space X for a rocket launch.

- **Problems you want to find answers**

- Can we create a machine learning pipeline to predict if the first stage will land given the data from SpaceX API, Wikipedia Web scraping, and other data sets?

Section 1

Methodology

Methodology

Executive Summary

- **Data collection methodology:**
    - Data for the present analysis was collected via SpaceX API, and web scrapping from the list of Falcon 9 and Falcon Heavy launches Wikipage updated on 9th June 2021.
- **Perform data wrangling**
    - To determine what would be the label for training supervised models, some exploratory data analysis was done on the data. This include:
        - Calculation of percentage of missing values in each attribute
        - Determine which columns were numerical and categorical
        - Finding the number of launches on each site
        - Calculate the number and occurrence of each orbit
        - Calculate the number and occurrence of mission outcome of the orbits
        - Create a landing outcome label from the outcome column

Methodology

Executive Summary

- **Perform exploratory data analysis (EDA) using visualization and SQL**
    - Using SQL queries, it was possible extract from the data set information such as:
        - Names of unique launch sites in the space mission
        - Classify launch sites by the payload mass and the boosters launched by a specific customer
        - Determine the average payload mass carried by a specific booster version
        - List dates where successful landing outcomes were reached, and filter those results by the type of platform where the landing has place
        - Filter by booster version and landing outcome those launches between payload mass ranges (ie. Between 4000 and 6000 kg)
        - Determine the total number of successful and failure mission outcomes
        - Determine the names of booster versions which have carried the maximum payload mass.
        - Determine successful landing outcomes dates displaying month and day, which took place during 2017
        - Arrange in descending order the result of the landing outcomes

Methodology

Executive Summary

- **Perform interactive visual analytics using Folium and Plotly Dash**
  - In this section, different type of visualizations where used to determine factors such as:
    - Categorical plot applied to determine the overall outcome of the launch comparing the flight number vs the payload mass
    - Categorical plot to determine outcomes of the launch comparing flight number and launch site
    - Find if there is a relationship between the launch site and the payload mass of the rockets using a scatter plot
    - Visually check if there is a relationship between success rate and orbit type using a bar chart
    - Determine the possible relationship between the flight number and the orbit type using a scatter plot
    - Visualize the relationship between the payload mass and orbit type using a scatter point chart
    - Visualize the variation of the success rate in the period spanned between 2010 and 2020

Methodology

Executive Summary

- **Perform predictive analysis using classification models**
  - The classification models used in the present analysis were Logistic Regression, Support Vector Machine, Decision Tree, and K Nearest Neighbors.
  - All of them were built using the specific hyperparameters according to the classification model. To tune these models hyperparameters the .best_params_ method was used.
  - All the methods accuracy was evaluated using both, the .score and .best_score_ in order to determine the best classification model to predict the desired information.

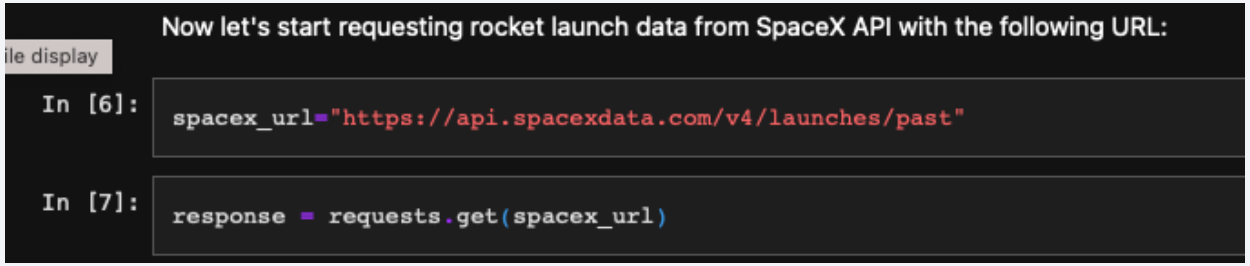
Data Collection

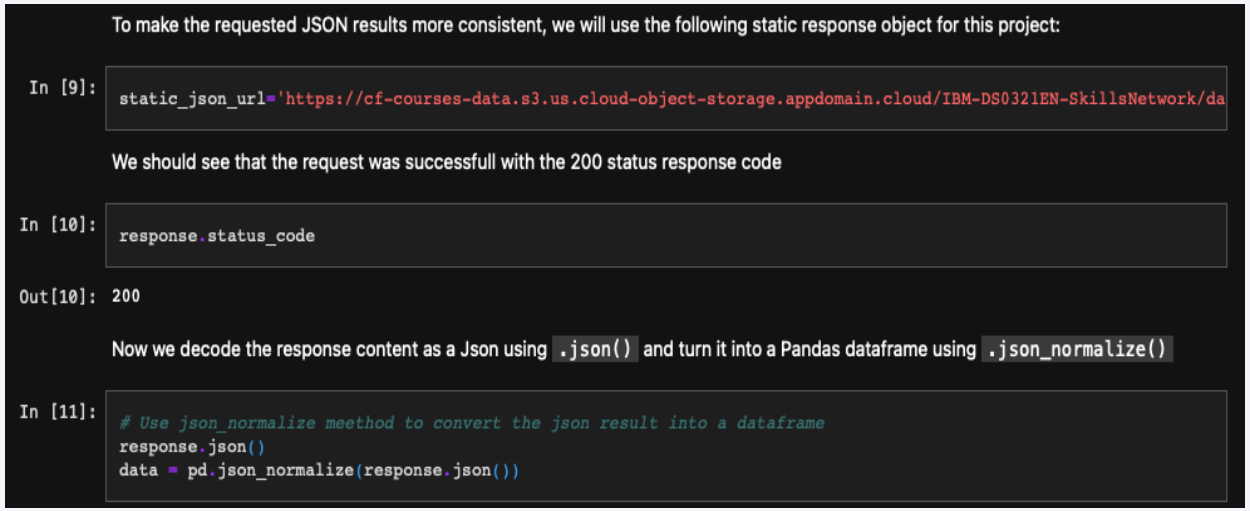
- **The process for collecting the data is described as follows:**
  - Request data using the get() method to the SpaceX API
  - Decode the response content as a JSON file using .json() and turn it into a Pandas data frame using .json_normalize()
  - Clean data, check missing values, complete the missing values when required.
  - Web scrapping from Wikipedia to complete the information
  - Add all the information into a Pandas data frame for further analysis.

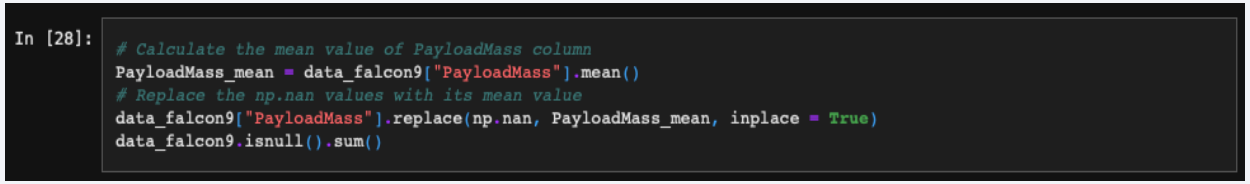
Data Collection – SpaceX API

- We used the request .get() method to obtain the data from the SpaceX API. Then we cleaned the data before wrangling the data. Finally, we format the data.

- URL to the notebook: https://github.com/salvadorojuela/Data-Science-and-Machine-Learning-Capstone-Project/blob/3550679b664640631d43cda49f6bba1df24eb0fb/lab1-spacex-data-collection-api.ipynb







Data Collection - Scraping

- We perform an HTTP GET method to request the Falcon9 Launch HTML page, as an HTTP response. We then Created a BeautifulSoup object from the HTML response. From the response we obtain the tables information and then create a data frame by parsing the launch HTML tables.

- URL to the notebook: https://github.com/salvadororjuela/Data-Science-and-Machine-Learning-Capstone-Project/blob/3550679b664640631d43cda49f6bba1df24eb0fb/lab2-spacex-webscraping.ipynb

In [4]:

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

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

In [12]:

```
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=r File display ng.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)
            #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)
            launch_dict["Version Booster"].append(bv)


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

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

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

Data Collection - Scraping

- We perform an HTTP GET method to request the Falcon9 Launch HTML page, as an HTTP response. We then Created a BeautifulSoup object from the HTML response. From the response we obtain the tables information and then create a data frame by parsing the launch HTML tables.

- URL to the notebook: https://github.com/salvadororjuela/Data-Science-and-Machine-Learning-Capstone-Project/blob/3550679b664640631d43cda49f6bba1df24eb0fb/lab2-spacex-webscraping.ipynb

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

# Customer
# TODO: Append the customer into launch_dict with key `Customer`
# Replace a tag with and empty string if its value is type None
customer = '' if row[6].a is None else row[6].a.strings
#print(customer)
launch_dict["Customer"].append(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)
launch_dict["Launch outcome"].append(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)
launch_dict["Booster landing"].append(booster_landing)
```

After you have fill in the parsed launch record values into `launch_dict`, you can create a dataframe from it.

In [14]:
```
df= pd.DataFrame({ key:pd.Series(value) for key, value in launch_dict.items() })
df
```

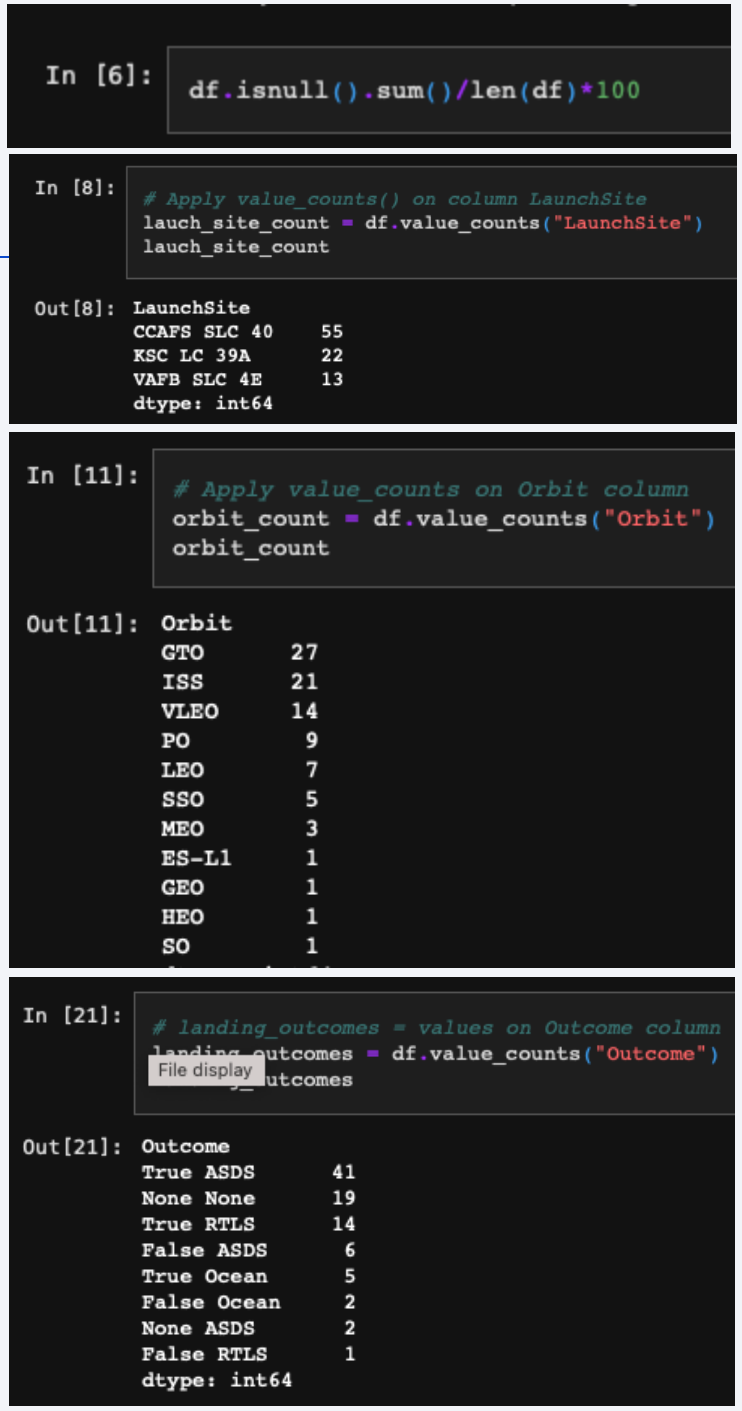
Out[14]:

	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	<generator object Tag._all_strings at 0x7fd477...	Success\n	F9 v1.0B0003.1	Failure	4 June 2010	18:45
1	2	CCAFS	Dragon	0	LEO	<generator object Tag._all_strings at 0x7fd477...	Success	F9 v1.0B0004.1	Failure	8 December 2010	15:43
2	3	CCAFS	Dragon	525 kg	LEO	<generator object Tag._all_strings at 0x7fd477...	Success	F9 v1.0B0005.1	Not attempted\n	22 May 2012	07:44
3	4	CCAFS	SpaceX CRS-1	4,700 kg	LEO	<generator object Tag._all_strings at 0x7fd477...	Success\n	F9 v1.0B0006.1	No attempt	8 October 2012	00:35
4	5	CCAFS	SpaceX CRS-2	4,877 kg	LEO	<generator object Tag._all_strings at 0x7fd477...	Success\n	F9 v1.0B0007.1	Not attempted\n	1 March 2013	15:10

Data Wrangling

- With data wrangling we look to find patterns in the data and determine training labels for the supervised model. The first step was calculate the percentage of the missing values in each attribute and identify the categorical and numerical columns. We then calculate some specific data such as the number of launches for each site, the number and occurrence of each orbit, the number and occurrence of mission outcome of the orbits, Create a landing outcome label from Outcome column, and finally get the average rate of success by using the .mean() method on the "Class" column.

- URL to the notebook: https://github.com/salvadororjuela/Data-Science-and-Machine-Learning-Capstone-Project/blob/3550679b664640631d43cda49f6bba1df24eb0fb/lab3-spacex-Data%20wrangling.ipynb



```
In [6]: df.isnull().sum()/len(df)*100
```

```
In [8]: # Apply value_counts() on column LaunchSite
        lauch_site_count = df.value_counts("LaunchSite")
        lauch_site_count

Out[8]: LaunchSite
        CCAFS SLC 40    55
        KSC LC 39A      22
        VAFB SLC 4E     13
        dtype: int64
```

```
In [11]: # Apply value_counts on Orbit column
         orbit_count = df.value_counts("Orbit")
         orbit_count

Out[11]: Orbit
         GTO      27
         ISS      21
         VLEO     14
         PO        9
         LEO       7
         SSO       5
         MEO       3
         ES-L1     1
         GEO       1
         HEO       1
         SO        1
```

```
In [21]: # landing_outcomes = values on Outcome column
         landing_outcomes = df.value_counts("Outcome")
         File display
         _outcomes

Out[21]: 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
         dtype: int64
```

Data Wrangling

- With data wrangling we look to find patterns in the data and determine training labels for the supervised model. The first step was calculate the percentage of the missing values in each attribute and identify the categorical and numerical columns. We then calculate some specific data such as the number of launches for each site, the number and occurrence of each orbit, the number and occurrence of mission outcome of the orbits, Create a landing outcome label from Outcome column, and finally get the average rate of success by using the .mean() method on the "Class" column.

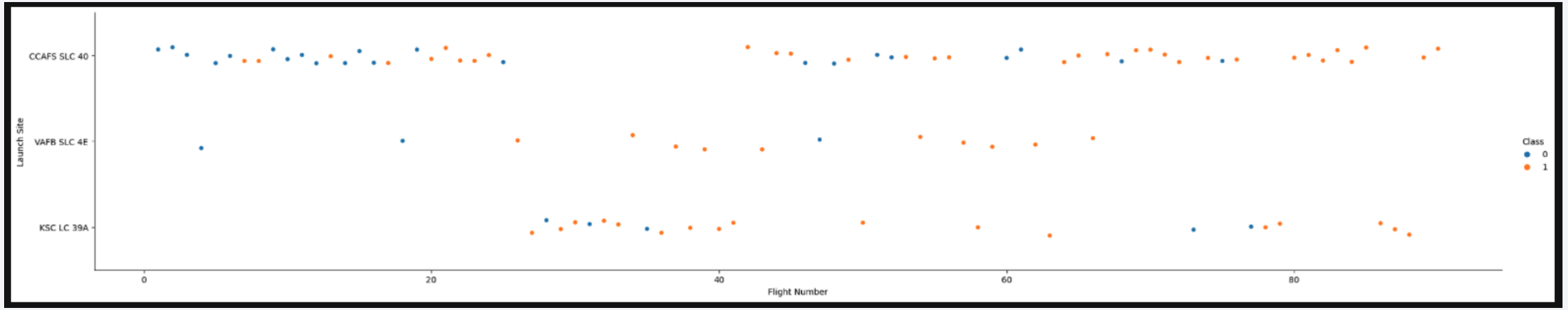
- URL to the notebook: https://github.com/salvadororjuela/Data-Science-and-Machine-Learning-Capstone-Project/blob/3550679b664640631d43cda49f6bba1df24eb0fb/lab3-spacex-Data%20wrangling.ipynb

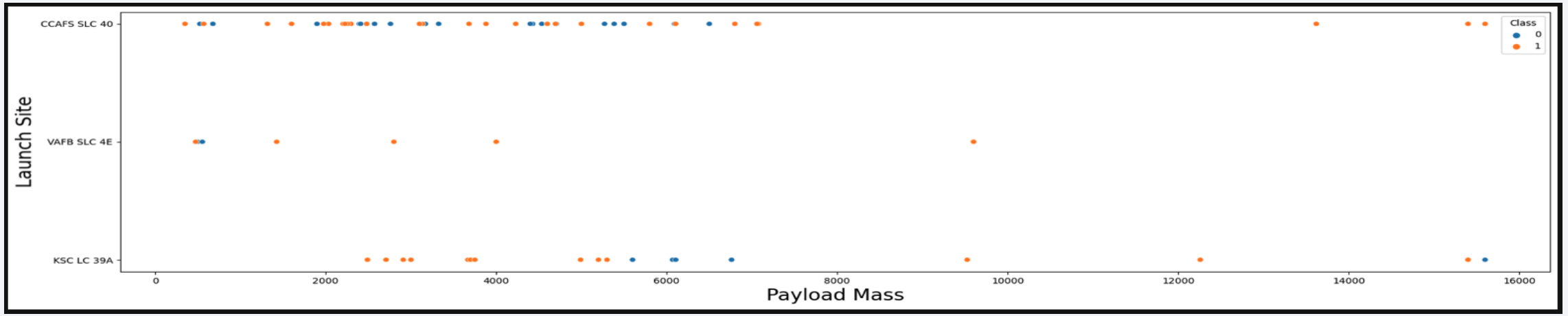
```
In [30]: # landing_class = 0 if bad_outcome
         # landing_class = 1 otherwise
         outcomes = df["Outcome"]

         bad_outcome = ['False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None']

         landing_class = list()

         for outcome in outcomes:
             if outcome in bad_outcome:
                 landing_class.append(0)
             else:
                 landing_class.append(1)

         landing_class

Out[30]: [0,
          0,
          0,
          0,
          0,
          0,
          1,
          1,
          0,
          0,
          0,
```

```
In [33]: df["Class"].mean()

Out[33]: 0.6666666666666666

In [35]: df.value_counts("Class")

Out[35]: Class
         1    60
         0    30
         dtype: int64
```

EDA with Data Visualization

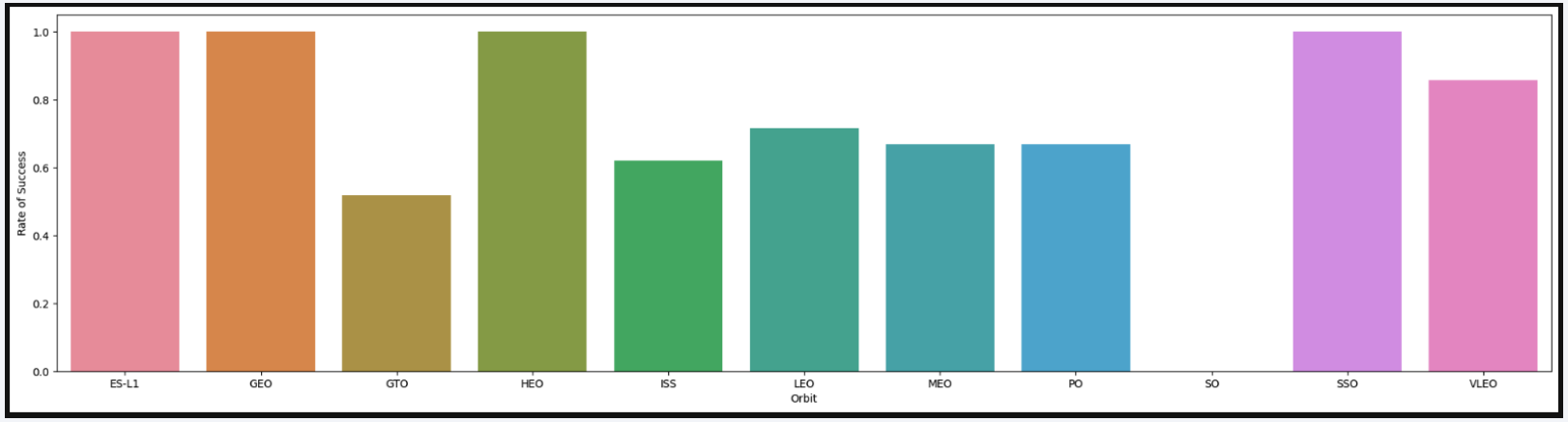
- We visualize the data for the relation between flight number and launch site, payload and launch site, success rate of each orbit type, flight number and orbit type, payload and orbit type, and the launch success yearly trend.
- URL to the notebook: https://github.com/salvadororjuela/Data-Science-and-Machine-Learning-Capstone-Project/blob/3550679b664640631d43cda49f6bba1df24eb0fb/lab5-eda-dataviz.ipynb.jupyterlite.ipynb

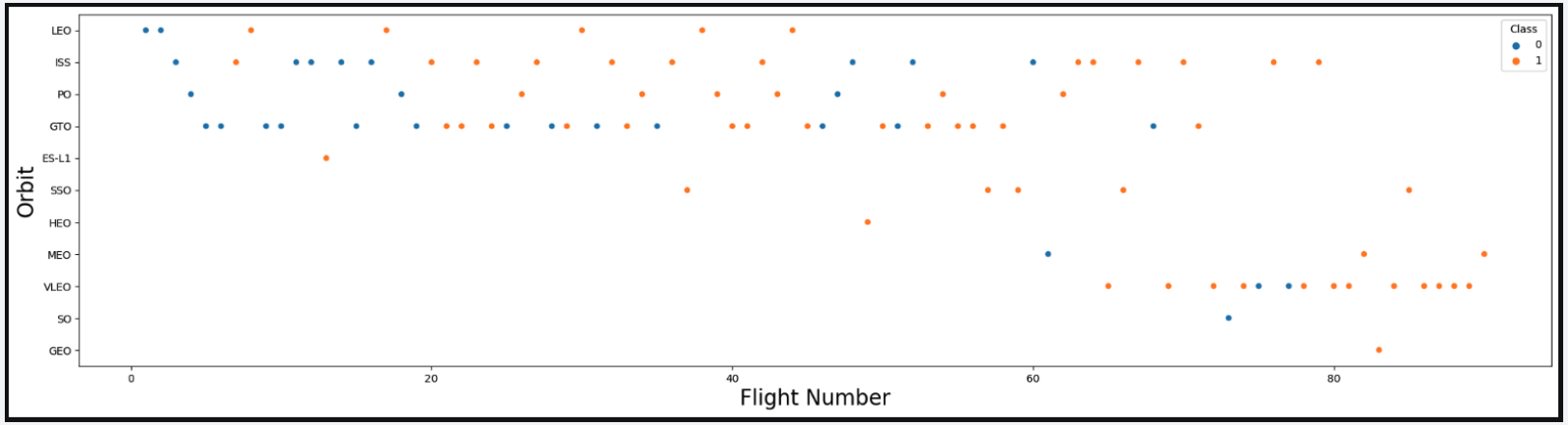




EDA with Data Visualization

- We visualize the data for the relation between flight number and launch site, payload and launch site, success rate of each orbit type, flight number and orbit type, payload and orbit type, and the launch success yearly trend.
- URL to the notebook: https://github.com/salvadororjuela/Data-Science-and-Machine-Learning-Capstone-Project/blob/3550679b664640631d43cda49f6bba1df24eb0fb/lab5-eda-dataviz.ipynb.jupyterlite.ipynb

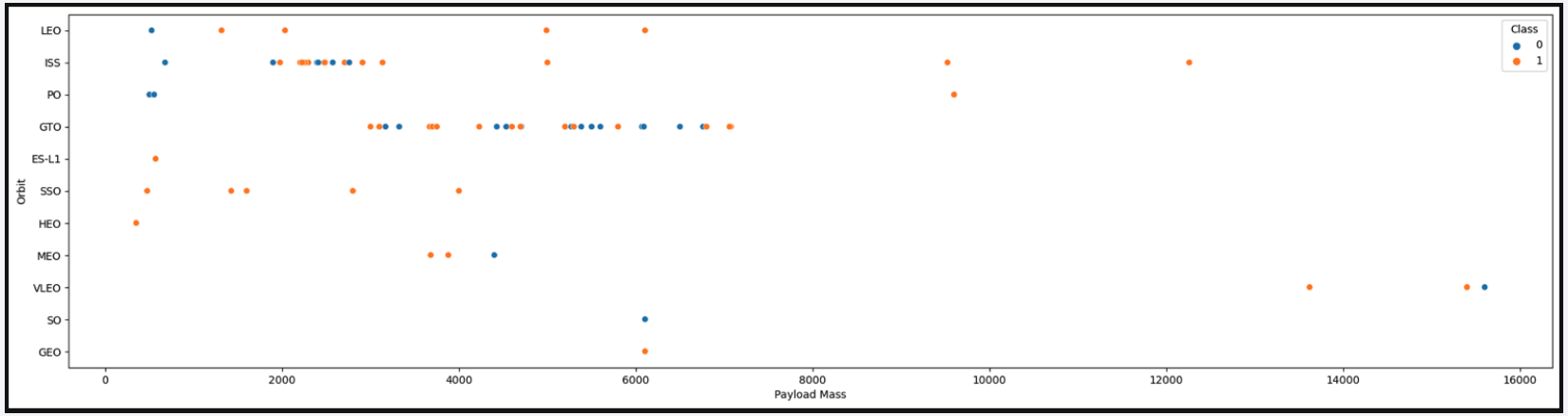


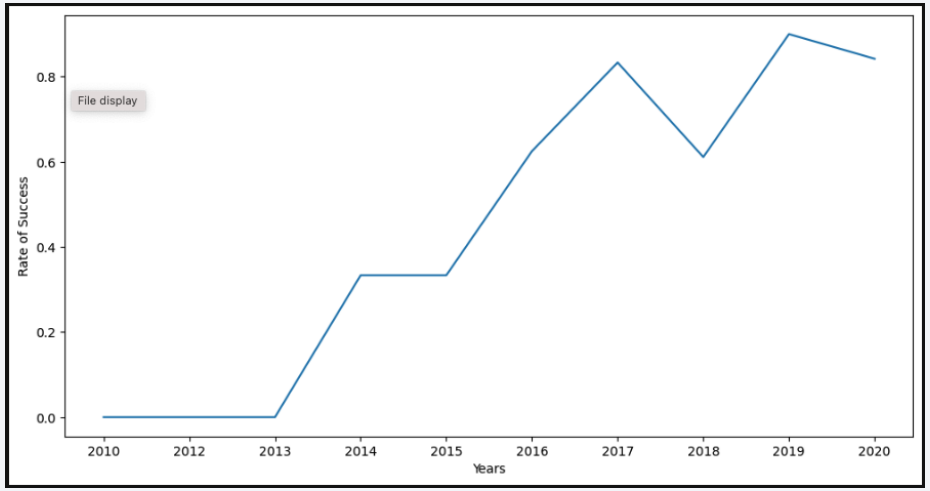


EDA with Data Visualization

- We visualize the data for the relation between flight number and launch site, payload and launch site, success rate of each orbit type, flight number and orbit type, payload and orbit type, and the launch success yearly trend.

- URL to the notebook: https://github.com/salvadororjuela/Data-Science-and-Machine-Learning-Capstone-Project/blob/3550679b664640631d43cda49f6bba1df24eb0fb/lab5-eda-dataviz.ipynb.jupyterlite.ipynb





EDA with SQL

- The queries we wrote were used to:
  - Get the names of the unique SpaceX launch sites
  - Get the records of launch sites that include certain string pattern in their names
  - Display the total payload mass carried by a specific client (NASA (CRS))
  - Show the average payload mass carried by the booster version No. "F9 v1.1"
  - Display the date where the successful landing outcome in drone ship was achieved
  - Display the names of the boosters which have success in ground pad and have a payload mass greater than 4000 and less than 6000 kg.
  - List the total number of successful and failure mission outcomes
  - Display the names of the booster versions which have carried the maximum payload mass
  - List the records which will display the month names, succesful landing_outcomes in ground pad ,booster versions, launch_site for the months in year 2017
  - 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

- URL to the notebook: https://github.com/salvadororjuela/Data-Science-and-Machine-Learning-Capstone-Project/blob/16d907b67f4804f613173216d4ebf9fd40fd7b3d/lab4-eda-sql-edx_sqllite.ipynb

Build an Interactive Map with Folium

- Markers were added to the map to identify if launches were successful or not. If success was achieved, a green marker is displayed. Otherwise, a red marker is displayed.
- Circles were used identify launch sites
- Marker clusters were used to group several markers that are located on the same spot or in close distance from one another
- Lines were used to display distances between launch sites and other landmarks.
- URL to the notebook: https://github.com/salvadororjuela/Data-Science-and-Machine-Learning-Capstone-Project/blob/16d907b67f4804f613173216d4ebf9fd40fd7b3d/lab6_launch_site_location.jupyterlite.ipynb

Build a Dashboard with Plotly Dash

- The dashboard contains two graphs: one pie chart which displays the rate of success of all launch sites or of a single one chosen by the user from a dropdown list. The second graph is a scatter plot which displays the landing success rate according to the payload mass that the user can select by using the range slider. Also, in the second graph the version booster is displayed
- URL to the notebook: https://github.com/salvadororjuela/Data-Science-and-Machine-Learning-Capstone-Project/blob/72d5ccb97734e2836fbe3204871e026414fcd3bb/lab7_spacex_dash_app.py

Predictive Analysis (Classification)

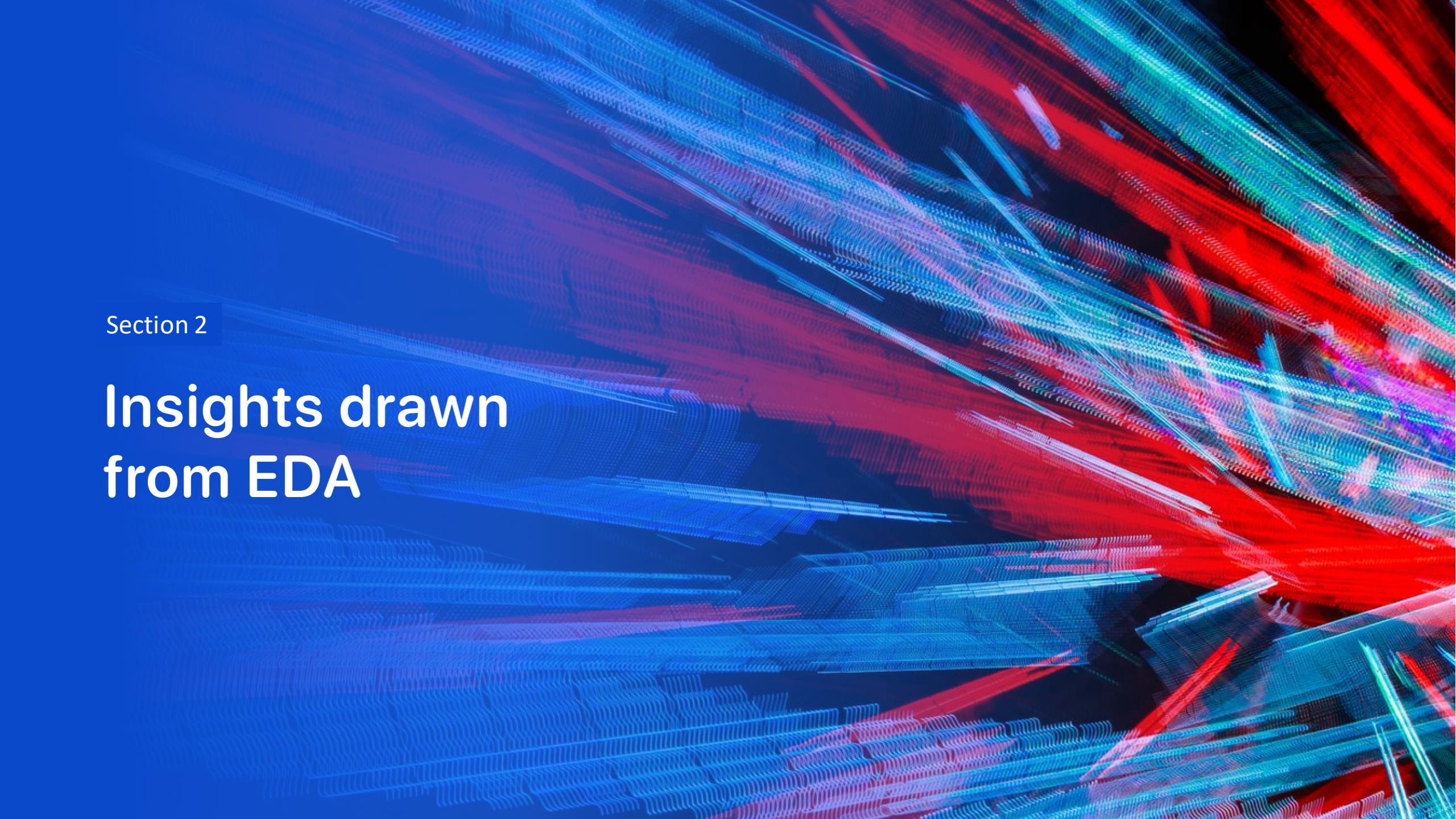
- We loaded the data using Numpy and Pandas. Split the data into training and test sets.
- Different machine learning models were built and the best hyperparameters were tuned by using GridSearchCV.
- The best parameters were determined by using best_parameters_. The .score() function was used to determine the accuracy for each model, as well as best_score.
- URL to the notebook: https://github.com/salvadororjuela/Data-Science-and-Machine-Learning-Capstone-Project/blob/72d5ccb97734e2836fbe3204871e026414fcd3bb/lab8_SpaceX_Machine_Learning_Prediction_Part_5.jupyterlite.ipynb

Results

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

Section 2

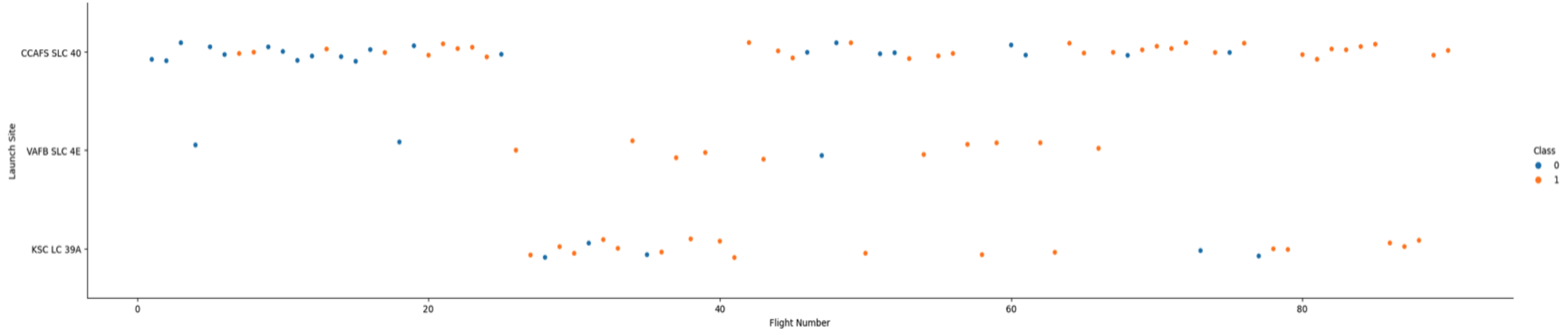
Insights drawn from EDA



Flight Number vs. Launch Site

- CCAFS SLC 40 has the highest rate of successful launches. Also, it is the site that has launched the most flights of all launch sites.

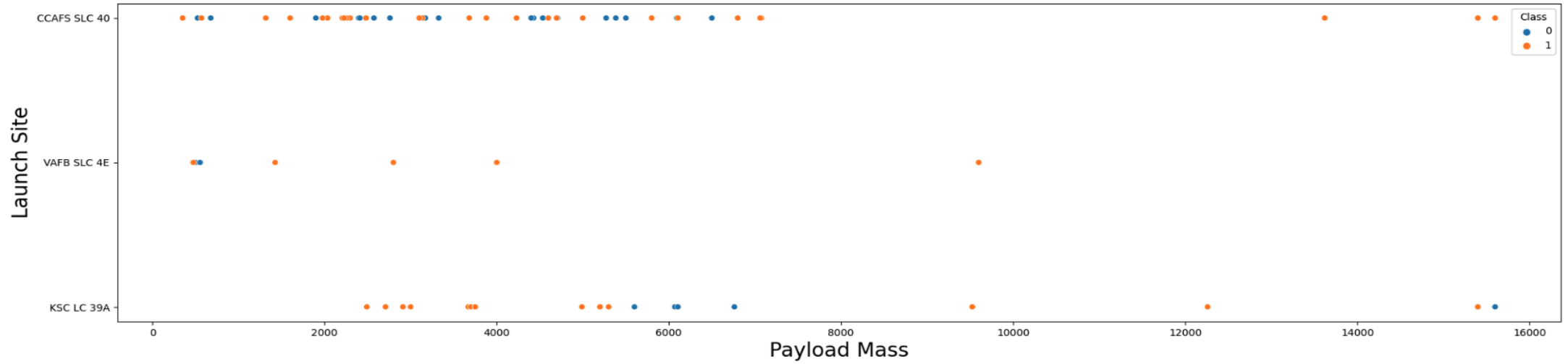
```
[7]: # Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to be the class value
sns.catplot(data=df, y="LaunchSite", x="FlightNumber", hue="Class", aspect = 5)
plt.xlabel("Flight Number")
plt.ylabel("Launch Site")
plt.show()
```



Payload vs. Launch Site

- VAFB SLC 4E has not launched rockets with payload masses greater than 10000 kg.
- KSC LC 39A has not launched rockets with payload masses smaller than 3000 kg.
- The greatest concentration of failed launches is found between a payload mas of 5000 kg to 7000 kg, and it occurred in CCAFS SLC 40, AND KSC LC 39A launch sites

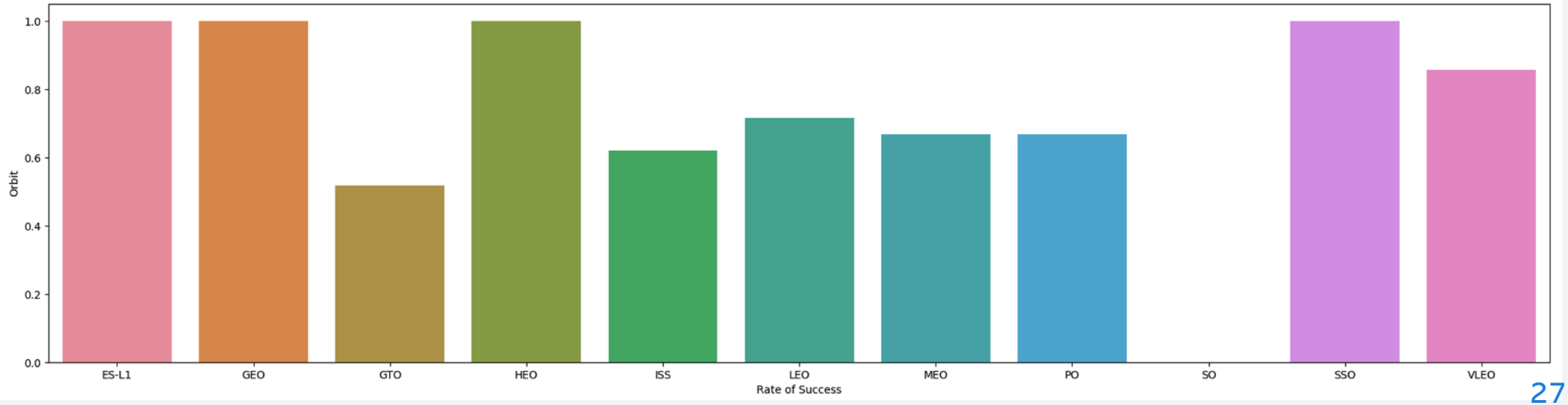
```
[9]: # Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch site, and hue to be the class value
     # Convert the type from str to int to be able to plot with seaborn
     # df["FlightNumber"] = df["FlightNumber"].astype(int)
     plt.figure(figsize=(25,6))
     sns.scatterplot(data=df, y="LaunchSite", x="PayloadMass", hue="Class")
     plt.xlabel("Payload Mass", fontsize=20)
     plt.ylabel("Launch Site", fontsize=20)
     plt.show()
```



Success Rate vs. Orbit Type

- ES-L1, GEO, HEO, and SSO are orbits with almost a 100% of success rate.
- SO is the orbit with the lowest launch success rate. The orbit with the second lowest launch success is GTO with About 50%.

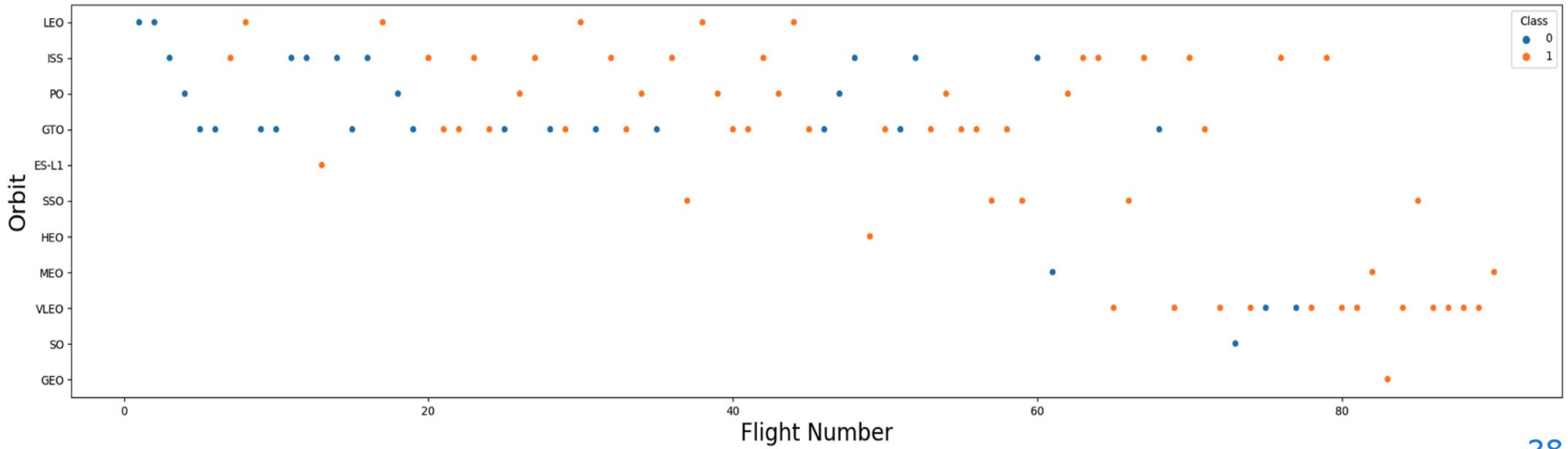
[11]:
```
# HINT use groupby method on Orbit column and get the mean of Class column
mean_success_serie = df.groupby(["Orbit"])["Class"].mean()
# Convert the series into a data frame
mean_success_df = pd.DataFrame(mean_success_serie)
# Convert the orbit into the index
mean_success_df.reset_index(inplace=True)
mean_success_df
plt.figure(figsize=(25,6))
sns.barplot(data=mean_success_df, y="Class", x="Orbit")
plt.xlabel("Rate of Success")
plt.ylabel("Orbit")
plt.show()
```



Flight Number vs. Orbit Type

- LEO orbit has a success related to the number of flights.
- GTO orbit does not show a relationship between flight number when in this orbit.
- From the 65th flight onwards, the orbit with the greatest number of flights is VLEO, followed by ISS orbits.

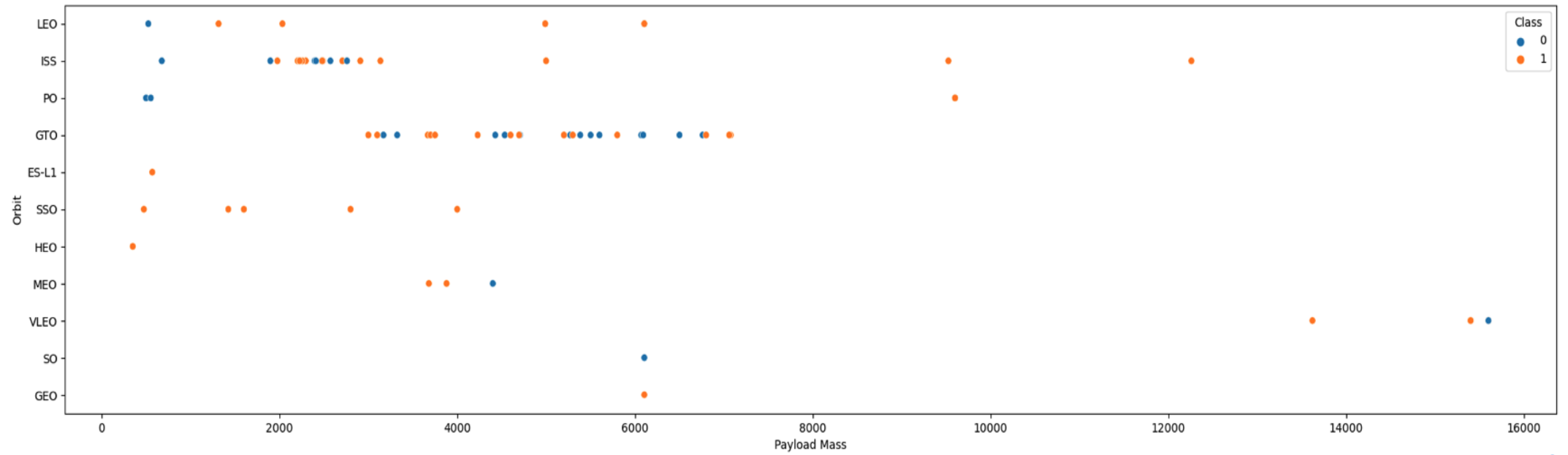
```
[14]: # Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class value
plt.figure(figsize=(25,6))
sns.scatterplot(data=df, y="Orbit", x="FlightNumber", hue="Class")
plt.xlabel("Flight Number", fontsize=20)
plt.ylabel("Orbit", fontsize=20)
plt.show()
```



Payload vs. Orbit Type

- For GTO orbit the relation is not well distinguishable as both failed and successful missions are present here.
- With heavy payloads the successful landing rate are more for PO, ISS, and VLEO.
- With payloads from 350 to 4000 kg, the successful landing rate is for SSO

```
[15]: # Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, and hue to be the class value
plt.figure(figsize=(25,6))
sns.scatterplot(data=df, x="PayloadMass", y="Orbit", hue="Class")
plt.xlabel("Payload Mass")
plt.ylabel("Orbit")
plt.show()
```

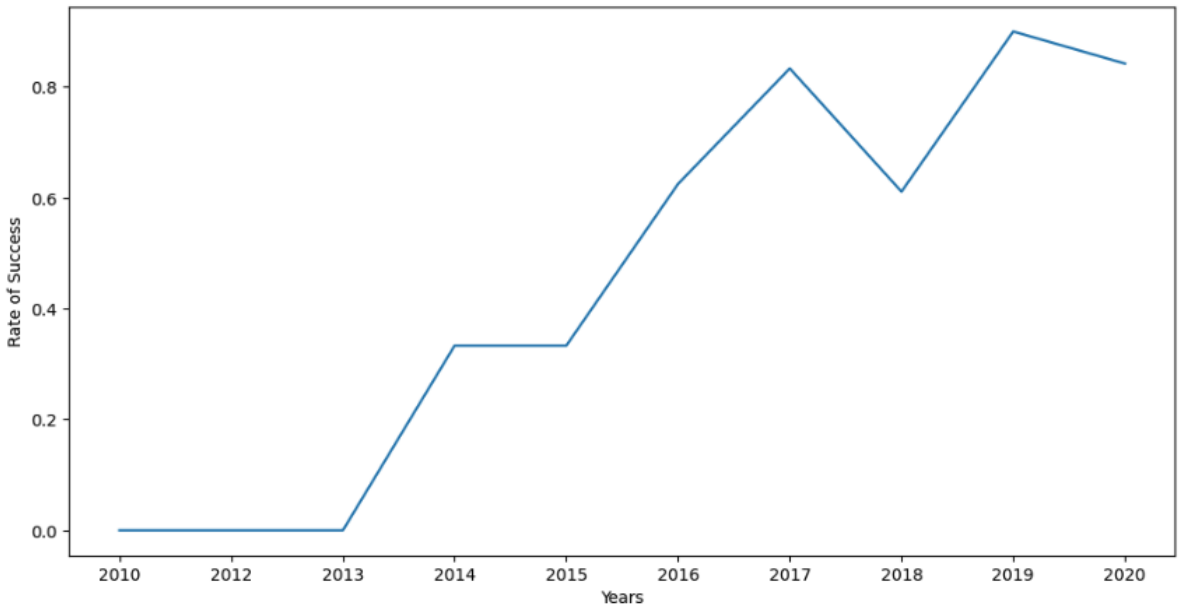


Launch Success Yearly Trend

- The launch success yearly trend started increasing since 2013 until 2017.
- There is a decrease of almost 20% in the yearly trend from 2017 to 2018.
- From 2018 until 2020 the launch success yearly trend increased again.

```
[18]: # Plot a line chart with x axis to be the extracted year and y axis to be the success rate
      # Group by year and get the average rate of success each year
      avg_success = pd.DataFrame(df.groupby(["Date"])["Class"].mean())
      print(avg_success)
      plt.figure(figsize=(12,6))
      sns.lineplot(data=avg_success, x="Date", y="Class")
      plt.xlabel("Years")
      plt.ylabel("Rate of Success")
      plt.show()
```

```
          Class
Date
2010  0.000000
2012  0.000000
2013  0.000000
2014  0.333333
2015  0.333333
2016  0.625000
2017  0.833333
2018  0.611111
2019  0.900000
2020  0.842105
```



All Launch Site Names

- We used the "DISTINCT" keyword on the column "Launch_Site" to display only unique names of SpaceX launch sites.

Task 1

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

[7]:
```
%sql SELECT DISTINCT "Launch_Site" FROM SPACEXTBL
```

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

[7]:

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

Launch Site Names Begin with 'KSC'

- We used the "LIKE" operator in combination with the "%" wildcard after the string "KSC" on the column "Launch_Site" to display the SpaceX launch sites which name begins with "KSC"

Task 2

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

[8]:
```
%sql SELECT * FROM SPACEXTBL WHERE "Launch_Site" LIKE "KSC%" LIMIT 5;
```

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

[8]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	M
2017-02-19	14:39:00	F9 FT B1031.1	KSC LC-39A	SpaceX CRS-10	2490	LEO (ISS)	NASA (CRS)	
2017-03-16	6:00:00	F9 FT B1030	KSC LC-39A	EchoStar 23	5600	GTO	EchoStar	
2017-03-30	22:27:00	F9 FT B1021.2	KSC LC-39A	SES-10	5300	GTO	SES	
2017-05-01	11:15:00	F9 FT B1032.1	KSC LC-39A	NROL-76	5300	LEO	NRO	
2017-05-15	23:21:00	F9 FT B1034	KSC LC-39A	Inmarsat-5 F4	6070	GTO	Inmarsat	

Total Payload Mass

- We used the "SUM" function on the "PAYLOAD_MASS_KG_" column and then filtered by "Customer" column where the "LIKE" operator was equal to "NASA (CRS)" to get the total payload mass carried by boosters launched by NASA (CRS)

Task 3

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

```
[10]: %sql SELECT SUM("PAYLOAD_MASS__KG_"), "Customer" FROM SPACEXTBL WHERE "Customer" LIKE "NASA (CRS)";
```

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

[10]:

SUM("PAYLOAD_MASS__KG_")	Customer
45596	NASA (CRS)

Task 3

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

```
[13]: %sql SELECT "PAYLOAD_MASS__KG_", "Customer" FROM SPACEXTBL WHERE "Customer" LIKE "NASA (CRS)";
```

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

[13]:

PAYLOAD_MASS__KG_	Customer
500	NASA (CRS)
677	NASA (CRS)
2296	NASA (CRS)
2216	NASA (CRS)
2395	NASA (CRS)
1898	NASA (CRS)
1952	NASA (CRS)
3136	NASA (CRS)
2257	NASA (CRS)
2490	NASA (CRS)
2708	NASA (CRS)
3310	NASA (CRS)
2205	NASA (CRS)
2647	NASA (CRS)
2697	NASA (CRS)
2500	NASA (CRS)
2495	NASA (CRS)
2268	NASA (CRS)
1977	NASA (CRS)
2972	NASA (CRS)

Average Payload Mass by F9 v1.1

- The "AVG" function returns the average payload mass carried by the booster version F9 v1.1 with a value of 2928.4 kg.

Task 4

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

```
[10]: %sql SELECT AVG("PAYLOAD_MASS__KG_") FROM SPACEXTBL WHERE "Booster_Version" == "F9 v1.1";
```

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

[10]:

AVG("PAYLOAD_MASS__KG_")
2928.4

First Successful Ground Landing Date

- We filtered the "Landing_Outcome" column display only those rows which content was equal to "Success (drone ship)" The "MIN" function displays that the date of the first successful landing outcome in a drone ship was the 8th April 2016.

Task 5

List the date where the succesful landing outcome in drone ship was acheived.

Hint:Use min function

[11]:
```
%sql SELECT MIN("Date"), "Landing_Outcome" FROM SPACEXTBL WHERE "Landing_Outcome" == "Success (drone ship)";
```

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

[11]:

MIN("Date")	Landing_Outcome
2016-04-08	Success (drone ship)

Successful Drone Ship Landing with Payload between 4000 and 6000

- To display the list of successful drone ship landings with payloads between 4000 and 6000 kg, we first filtered the column "Landing_Outcome" to match rows that contained the string "Success (ground pad). Then using the ">" and "<" operator we determined those "PAYLOAD_MASS_KG_" greater than 4000 and smaller than 6000 kg respectively.

Task 6

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

[12]:
```
%sql SELECT "Booster_Version", "Landing_Outcome", "PAYLOAD_MASS__KG_" FROM SPACEXTBL WHERE "Landing_Outcome" == "Success (ground pad)" AND "PAYLOAD_MASS__KG_" > 4000 AND "PAYLOAD_MASS__KG_" < 6000;
```

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

[12]:

Booster_Version	Landing_Outcome	PAYLOAD_MASS__KG_
F9 FT B1032.1	Success (ground pad)	5300
F9 B4 B1040.1	Success (ground pad)	4990
F9 B4 B1043.1	Success (ground pad)	5000

Total Number of Successful and Failure Mission Outcomes

- We used the "COUNT" function on the "Mission_Outcome" column, and then the "GROUP BY" on the same column to display the total number of successful and failure mission outcomes.

Task 7

List the total number of successful and failure mission outcomes

[13]:
```
%sql SELECT "Mission_Outcome", COUNT("Mission_Outcome") FROM SPACEXTBL GROUP BY "Mission_Outcome";
```

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

[13]:

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

Boosters Carried Maximum Payload

- The subquery applied on the "PAYLOAD_MASS_KG_" column filters and display only the values that correspond to the "MAX" value for "PAYLOAD_MASS_KG_". In other words, the maiximum payload mass.

Task 8

List the names of the booster_versions which have carried the maximum payload mass. Use a subquery

```
[14]: %sql SELECT "Booster_Version", "PAYLOAD_MASS__KG_"FROM SPACEXTBL WHERE "PAYLOAD_MASS__KG_" = (SELECT MAX("PAYLOAD_MASS__KG_") FROM SPACEXTBL);
```

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

[14]:

Booster_Version	PAYLOAD_MASS__KG_
F9 B5 B1048.4	15600
F9 B5 B1049.4	15600
F9 B5 B1051.3	15600
F9 B5 B1056.4	15600
F9 B5 B1048.5	15600
F9 B5 B1051.4	15600
F9 B5 B1049.5	15600
F9 B5 B1060.2	15600
F9 B5 B1058.3	15600
F9 B5 B1051.6	15600
F9 B5 B1060.3	15600
F9 B5 B1049.7	15600

2015 Launch Records

- The months, days, and years are obtained by using the substr() method to create the corresponding columns displayed in the outcome of the query. The result was obtained by filtering the "Landing_Outcome" column with values equal to "Success (ground pad)" and the year corresponding to 2017 (substr(date, 0, 5)="2017").

Task 9

List the records which will display the month names, succesful landing_outcomes in ground pad ,booster versions, launch_site for the months in year 2017

Note: SQLLite does not support monthnames. So you need to use substr(Date,6,2) for month, substr(Date,9,2) for date, substr(Date,0,5),='2017' for year.

[15]:
```
%sql SELECT substr(Date, 6, 2) as Month, substr(Date, 9, 2) as Day, "Landing_Outcome", substr(Date, 0, 5) AS Year FROM SPACEXTBL WHERE "Landing_Outcome" == "Success (ground pad)" AND substr(Date, 0, 5)="2017";
```

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

[15]:

Month	Day	Landing_Outcome	Year
02	19	Success (ground pad)	2017
05	01	Success (ground pad)	2017
06	03	Success (ground pad)	2017
08	14	Success (ground pad)	2017
09	07	Success (ground pad)	2017
12	15	Success (ground pad)	2017

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

- Here we filtered the occurrences of the landing outcomes using the "COUNT()" function on the "Landing_Outcome" column. The second filter is "WHERE", in which we determined to include only those rows where the date of the launch was between 2010-06-04 "AND" 2017-03-20, all of them grouped according to the "Landing_Outcome" values and arranged in descending order using the "DESC" keyword at the end of the query.

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

[16]:
```
%sql SELECT "Landing_Outcome", COUNT("Landing_Outcome") AS Total FROM SPACEXTBL WHERE "Date" >= "2010-06-04" AND "Date" <= "2017-03-20" GROUP BY "Landing_Outcome" ORDER BY Total DESC;
```

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

[16]:

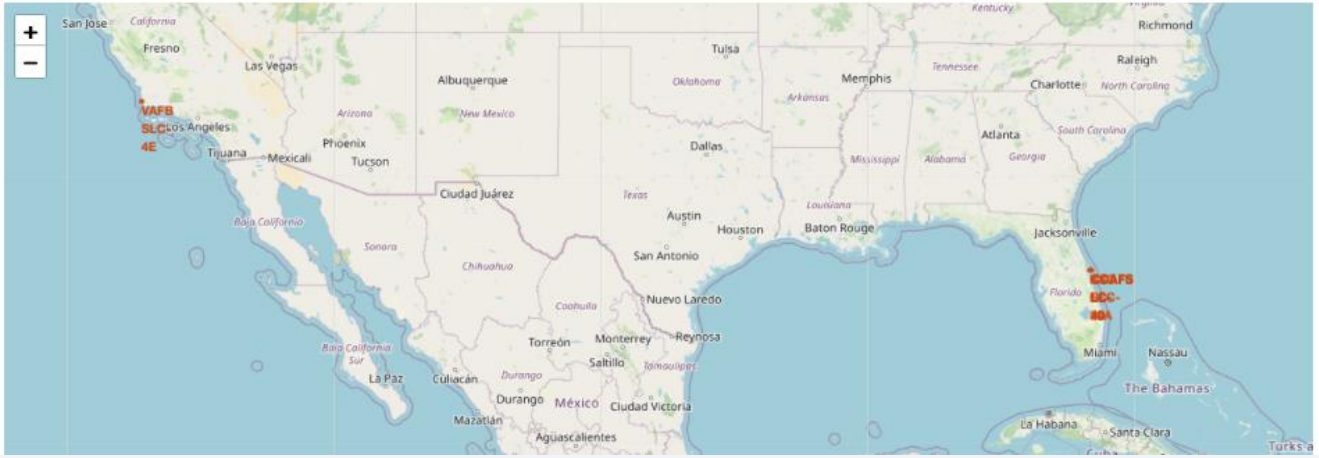
Landing_Outcome	Total
No attempt	10
Success (drone ship)	5
Failure (drone ship)	5
Success (ground pad)	3
Controlled (ocean)	3
Uncontrolled (ocean)	2
Failure (parachute)	2
Precluded (drone ship)	1


Section 3
Launch Sites
Proximities Analysis

SpaceX Launch Sites

The map displays the SpaceX launch sites located in the United States. Three of them are located in the eastern and one on the western coasts of the country.

[9]:
```
# Initial the map
site_map = folium.Map(location=nasa_coordinate, zoom_start=4)

# For each launch site, add a Circle object based on its coordinate (Lat, Long) values. In addition, add Launch site name as a popup label
# Create a list with the coordinates only
for site, lat, long in zip(launch_sites_df["Launch Site"], launch_sites_df["Lat"], launch_sites_df["Long"]):
    site_coord = [lat, long]
    site_name= site

    # Add each folium.Circle
    circle_site = folium.Circle(site_coord,
                                radius=1000, color='#d35400',
                                fill=True).add_child(folium.Popup(site_name))
    marker_launch_site = folium.map.Marker(site_coord,
                                           icon=DivIcon(icon_size=(20,20),icon_anchor=(0,0),
                                                        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % site_name, ))

    site_map.add_child(circle_site)
    site_map.add_child(marker_launch_site)

# Show the map
site_map
```

[9]:


Launch Outcome For Each Site

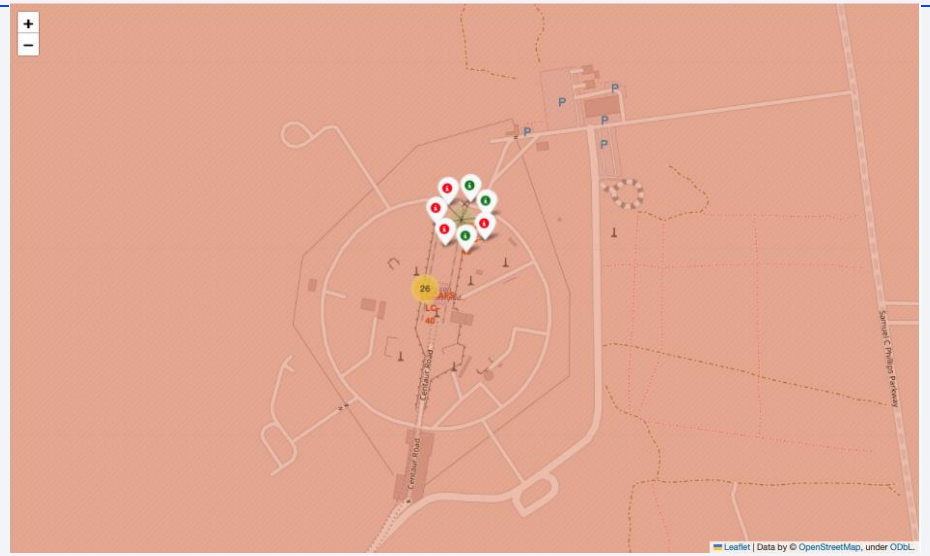
The following two screenshots display all the launches in each SpaceX launch site. The clusters include markers in red representing failing launches and green markers indicate a successful launch.

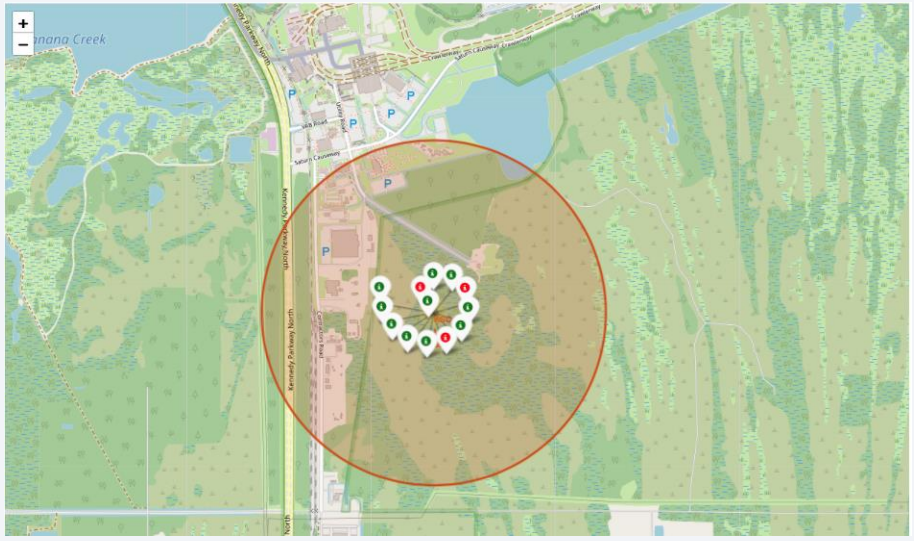
[14]:
```
# Add marker_cluster to current site_map
site_map.add_child(marker_cluster)

# 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 index, record in spacex_df.iterrows():
    # TODO: Create and add a Marker cluster to the site map
    marker = folium.Marker(location=[record.Lat, record.Long],
                           icon=folium.Icon(color="white", icon_color=record.marker_color),
                           popup=record["Launch Site"])
    marker_cluster.add_child(marker)

site_map
```

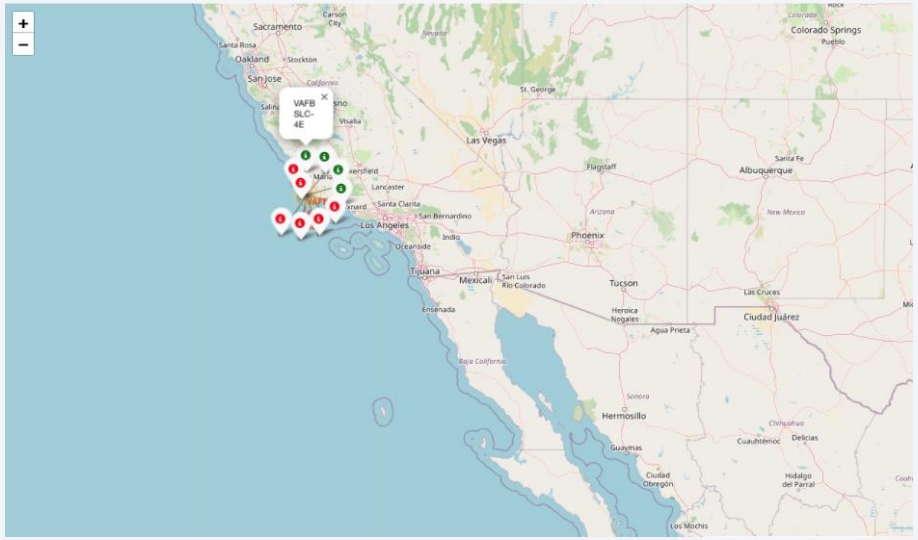
[14]:

Launch Outcome For Each Site

The screenshots display all the launches in each SpaceX launch site. The clusters include markers in red representing failing launches and green markers indicate a successful launch.







CCAF5 PROXIMITY TO THE NEAREST CITY, RAILROAD, HIGHWAY, AND COASTLINE

- The screenshot shows CCAF SLC-40 launch site and its proximities to the Titan III railway, Orlando, Samuel C. Phillips Parkway, and the eastern coastline, with distance calculated and displayed

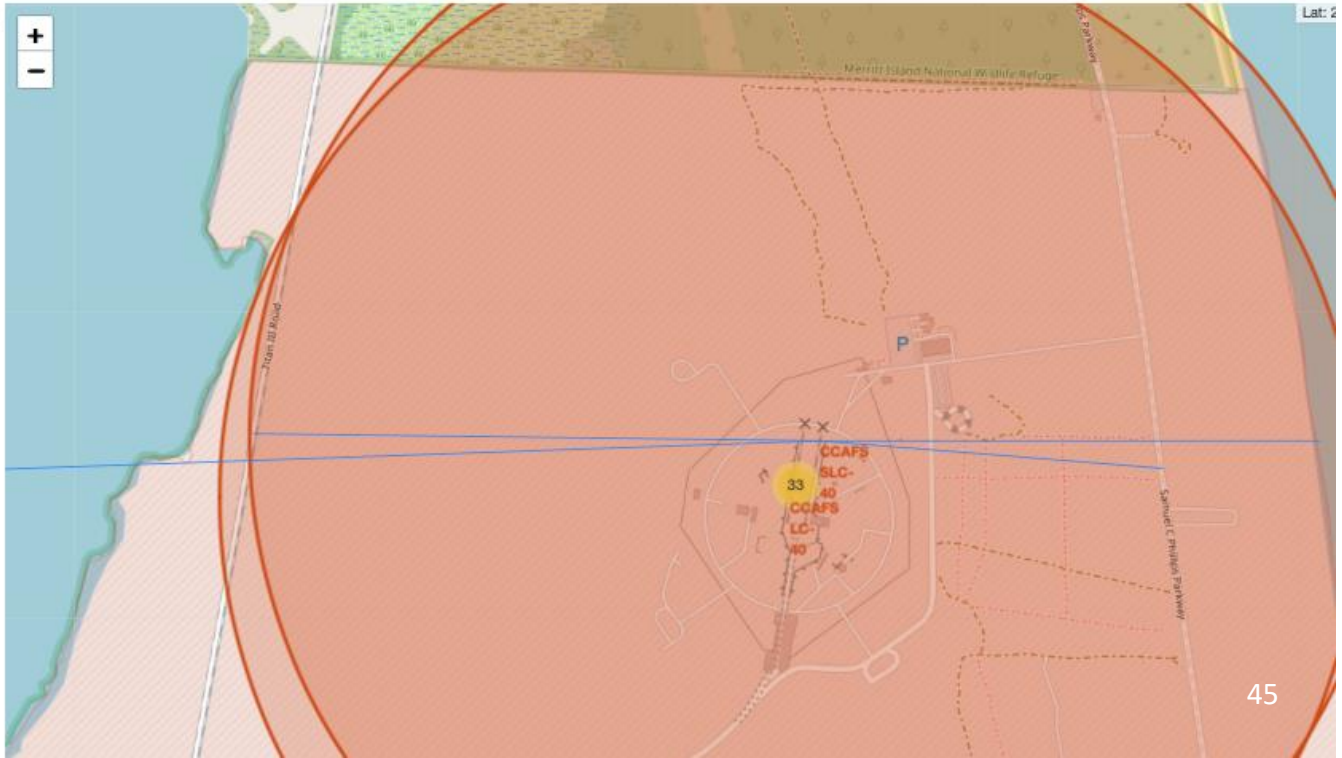
```
The distance to Orlando is 78.41446868981808 miles
The distance to Titan III railway is 0.9935940274296251 miles
The distance to the closest highway is 0.6038349365327684 miles
```

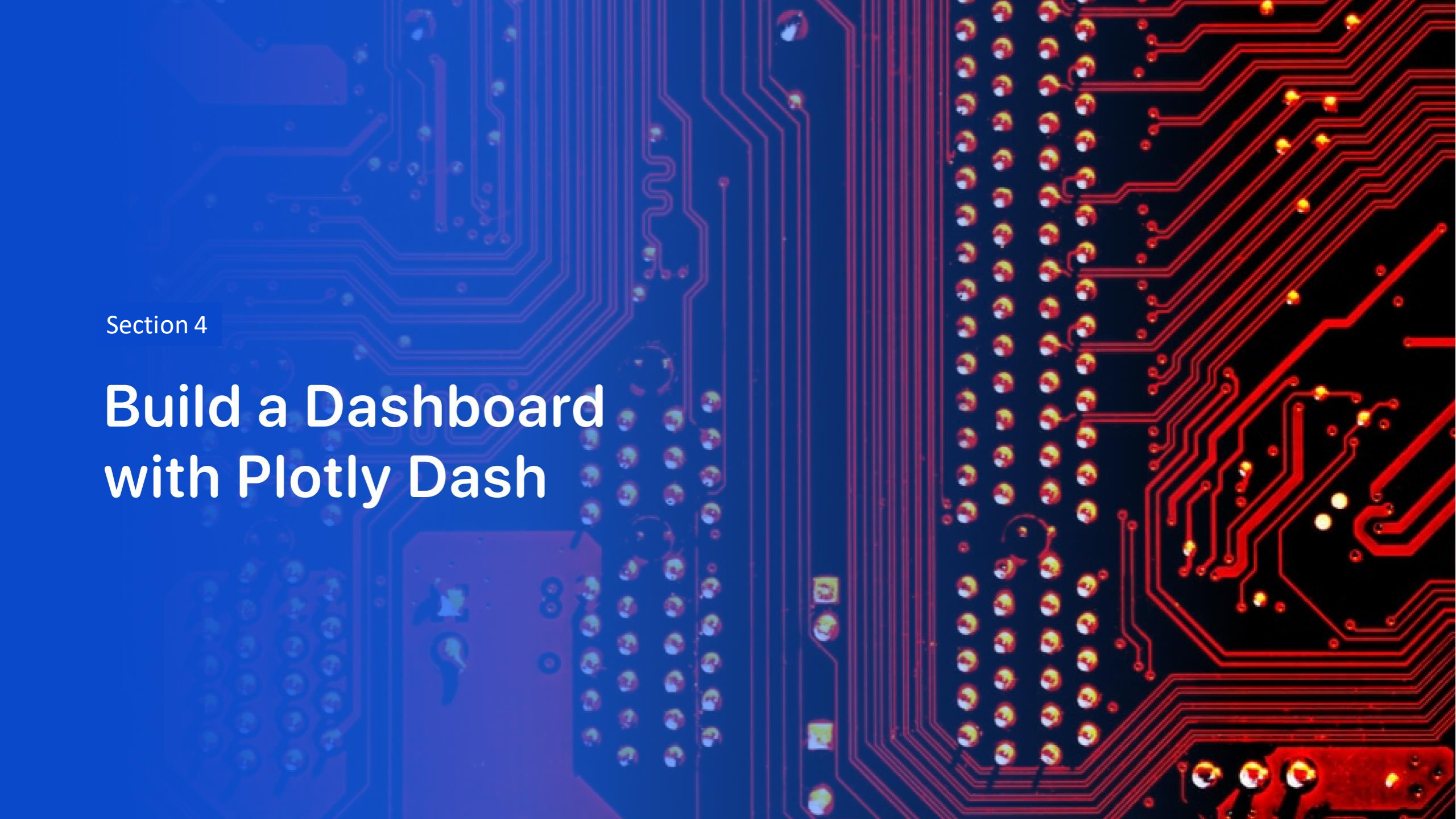
[22]:
```
# Use a for loop to create all the markers for Orlando, the railway, and the highway
# Create a list of lists with the latitude and longitude of the markers
markers = [[orlando_lat, orlando_lon], [railway_lat, railway_lon], [highway_lat, highway_lon]]
# List of distances
distances = [orlando_distance, railway_distance, higway_distance]

for coord, dist in zip(markers, distances):
    print(coord, dist)
    distance_marker = folium.Marker(
    location=[coord[0], coord[1]],
    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(dist),
        )
    )
```

```
[28.5384, -81.3789] 78.41446868981808
[28.5633, -80.58699] 0.9935940274296251
[28.56275, -80.57066] 0.6038349365327684
```

[23]:
```
# For loop to create the lines between the launch site and the markers, and add it to the site_map
for coord in markers:
    lines = folium.PolyLine(locations=[[launch_site_lat, launch_site_lon],[coord[0], coord[1]]], weight=1)
    site_map.add_child(lines)

# Display the map
site_map
```

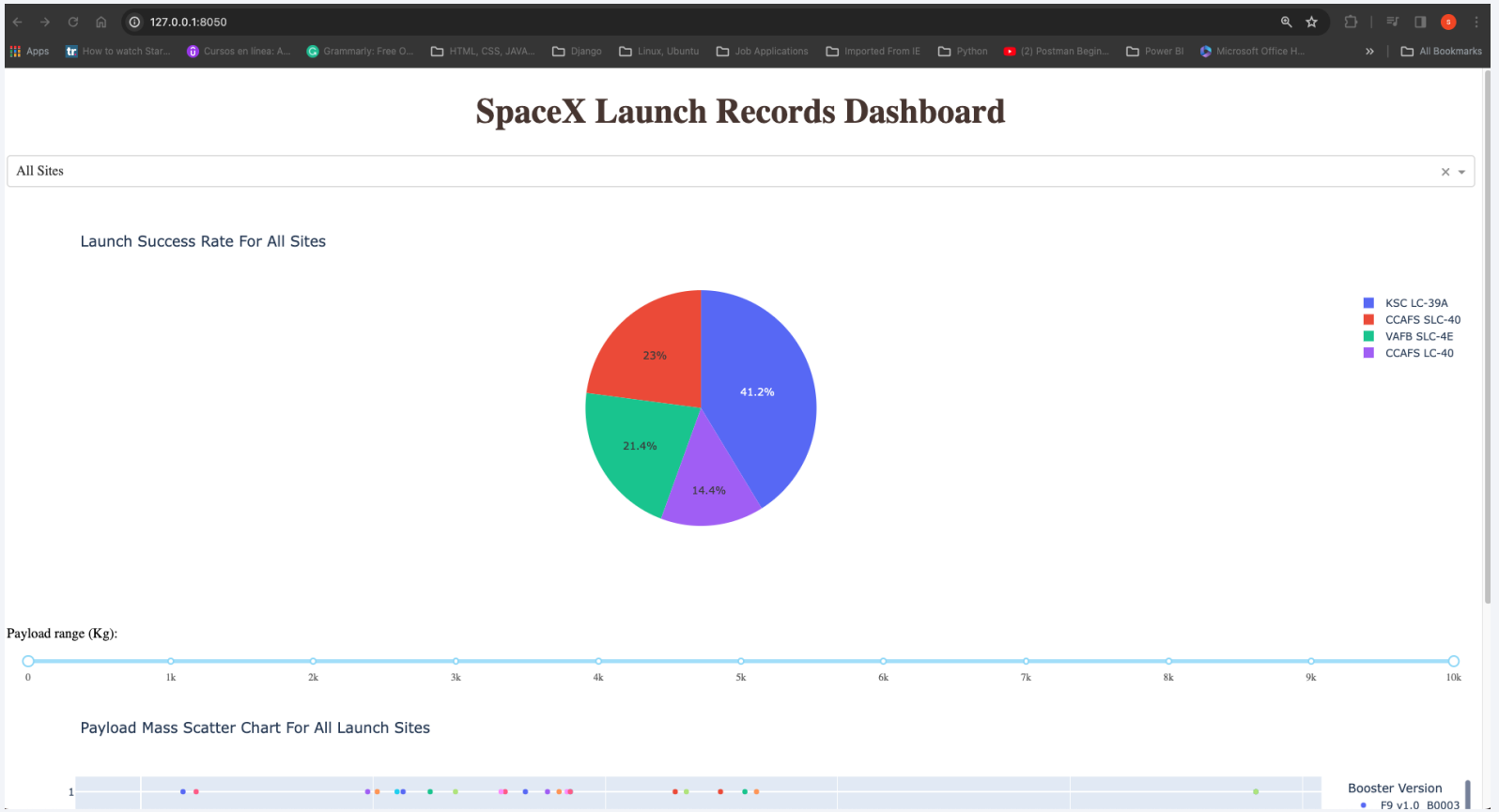
[23]:




Section 4
Build a Dashboard with Plotly Dash

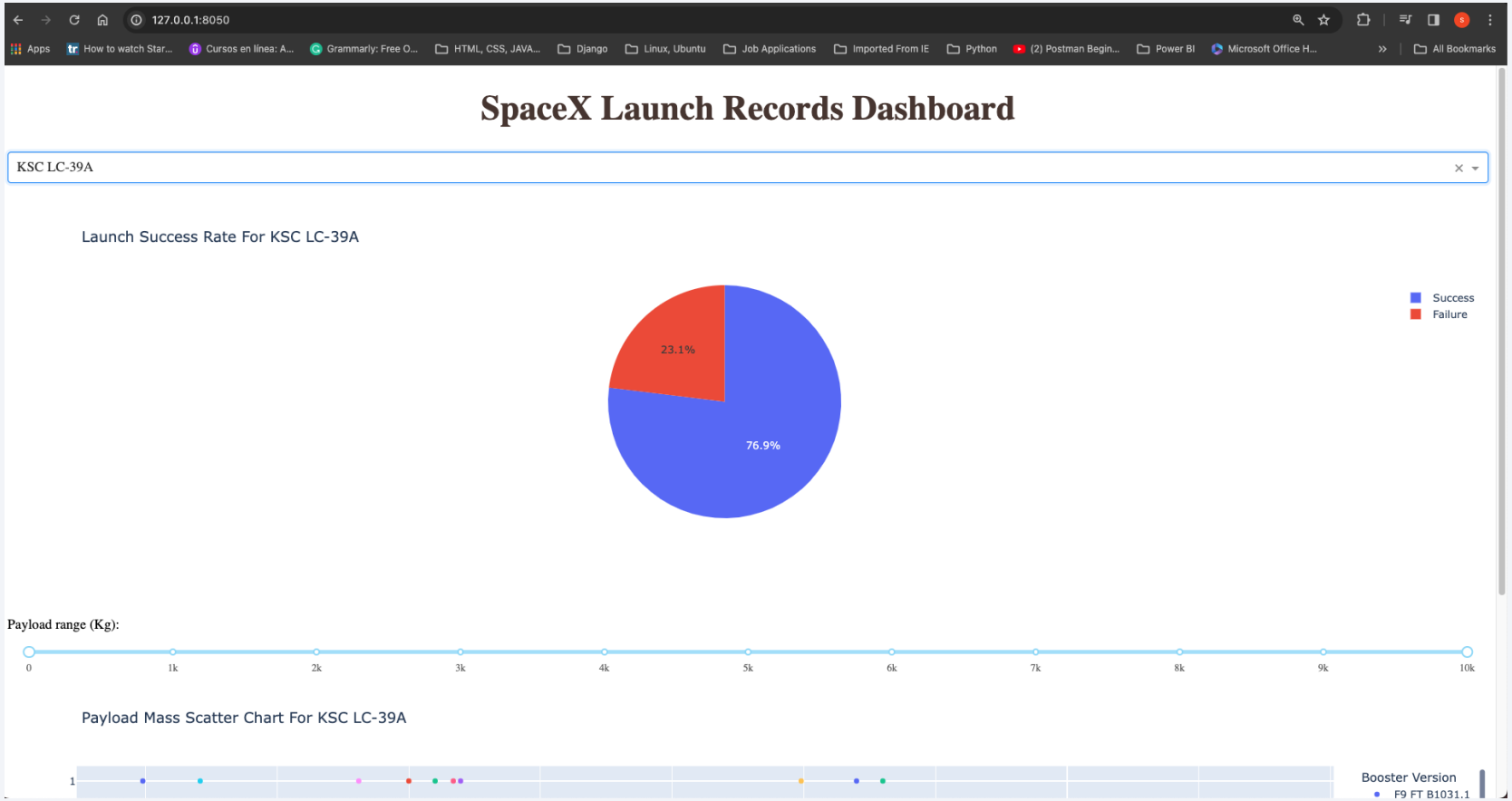
Launch Success Rate For All SpaceX Launch Sites

The pie chart shows the launch success rate for all SpaceX launch sites. The highest success rate is for KSC LC 39A, and the lowest rate corresponds to CCAFSLC-40.



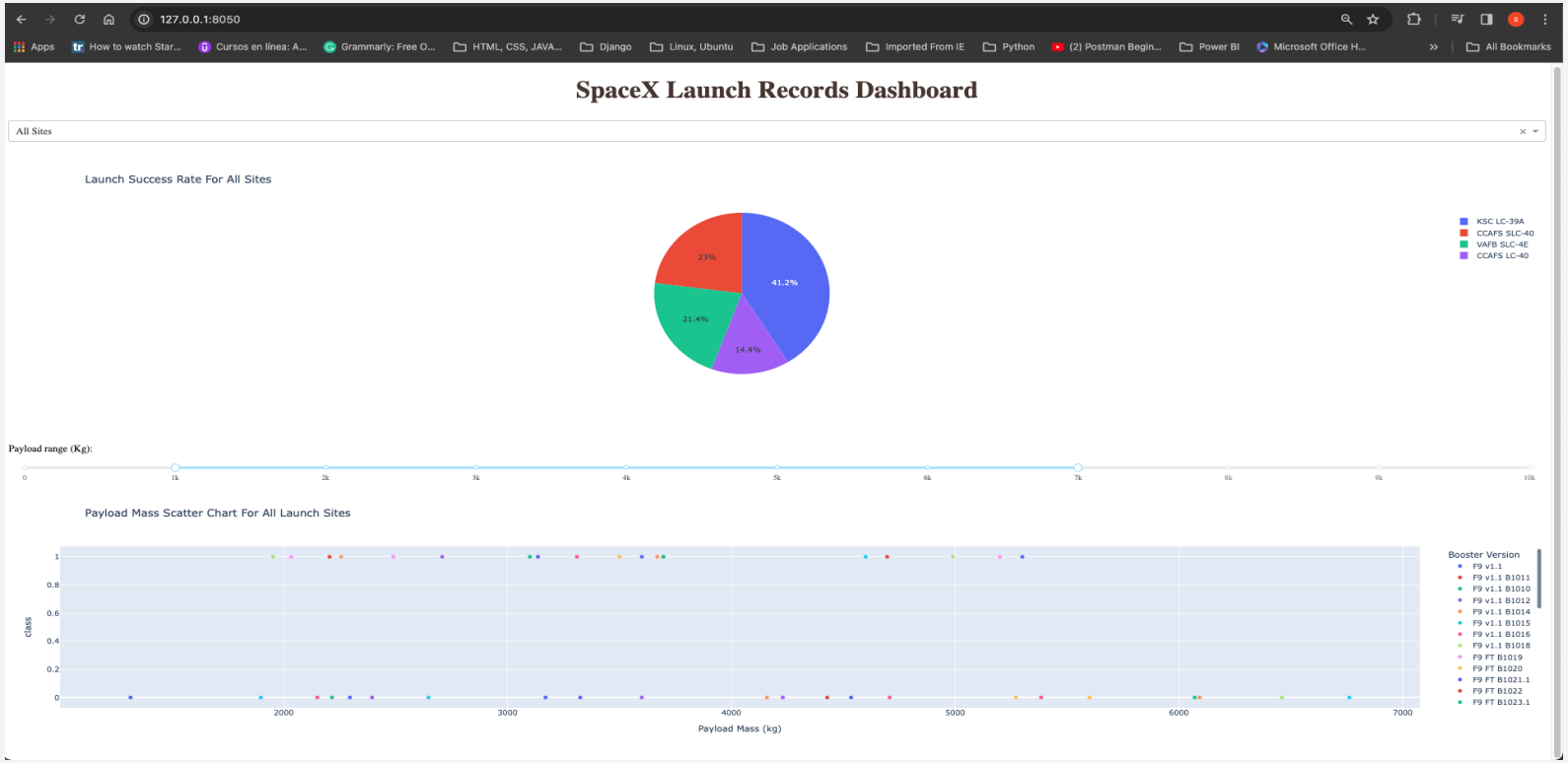
SpaceX Launch Site With The Highest Launch Success Rate

- KSC LC-39A launch site has the highest launch success rate of all SpaceX launch sites, with a result of 76.9% successful launches and only 23.1% failures.



Launch Outcome For Payload Masses Between 1000 and 7000 Kg For All Sites

- Replace <Dashboard screenshot 3> title with an appropriate title
- Show screenshots of Payload vs. Launch Outcome scatter plot for all sites, with different payload selected in the range slider
- Explain the important elements and findings on the screenshot, such as which payload range or booster version have the largest success rate, etc.



Section 5

Predictive Analysis (Classification)

Classification Accuracy

- On this slide we see that the accuracy obtain using the .score method shows that the tree classifier has the highest accuracy with 0.9444, while the other three methods return 0.8333

TASK 12

Find the method performs best:

•[37]:
```
# Create data frames
evaluation = pd.DataFrame({"Method": ["Accuracy"]})
evaluation_best_score = pd.DataFrame({"Metod": ["Best Score"]})

# Accuracy
logreg_accuracy = logreg_cv.score(X_test, Y_test)
svm_accuracy = svm_cv.score(X_test, Y_test)
tree_accuracy = tree_cv.score(X_test, Y_test)
knn_accuracy = knn_cv.score(X_test, Y_test)

# Key value pairs for the dictionary
methods_accuracy = [logreg_accuracy, svm_accuracy, tree_accuracy, knn_accuracy]
methods_list = ["Logistic Regression", "Support Vector Machine (SVM)", "Tree Classifier", "KNN Neighbors"]
initial_value = 0
best_method = ""

# Create the dataframe with the test accuracy
for key, value in zip(methods_list, methods_accuracy):
    if value > initial_value:
        initial_value = value
        best_method = key
    evaluation[key] = value

evaluation = evaluation.set_index("Method").transpose()

print(f"\nThe method that performs best is {best_method} with an accuracy of = {initial_value}\n")
evaluation
```

```
The method that performs best is Tree Classifier with an accuracy of = 0.9444444444444444
```

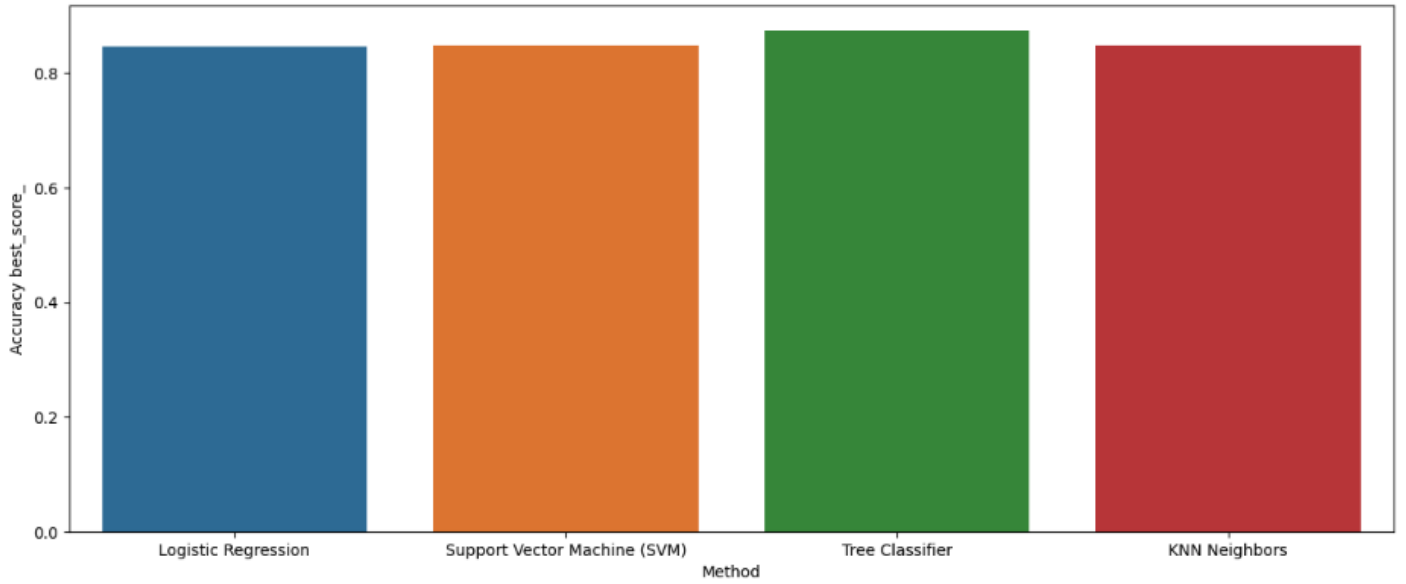
[37]:

Method	Accuracy
Logistic Regression	0.833333
Support Vector Machine (SVM)	0.833333
Tree Classifier	0.944444
KNN Neighbors	0.833333

Classification Accuracy

- We can observe that using the best_score() method, the Tree Classifier obtains the higher score with 0.875.

```
[65]: # best_score_ accuracy
logreg_best_score = logreg_cv.best_score_
svm_best_score = svm_cv.best_score_
tree_best_score = tree_cv.best_score_
knn_best_score = knn_cv.best_score_
# Dictionary to convert into a Pandas data frame
best_score = {
    "Method": methods_list,
    "Best Score": [logreg_best_score, svm_best_score, tree_best_score, knn_best_score]
}
# Create the data frame
bestScore_df = pd.DataFrame(best_score)
bestScore_df
# bar chart
plt.figure(figsize = (15, 6))
sns.barplot(bestScore_df, x="Method", y="Best Score", hue="Method")
plt.xlabel("Method")
plt.ylabel("Accuracy best_score_")
plt.show()
# Get the key value pair for the best_score_ with the highest value
bestScore_df = best_score
init_score = 0
high_best_score = ""
for k, v in zip(methods_list, best_score["Best Score"]):
    if v > init_score:
        init_score = v
        high_best_score = k
print(f"The hightest value of best_score is = {init_score}. It corresponds to the method = {high_best_score}")
```




```
The hightest value of best_score is = 0.875. It corresponds to the method = Tree Classifier
```

Confusion Matrix

- The tree classifier returns the highest false negatives of all four methods. In this case, it returns 5 false negative, while the other methods return 3. This means that the failing landings were effectively marked as unsuccessful by the classifier.
- The main concern is related to the high number of false positives or unsuccessful landings marked as positive.

TASK 9

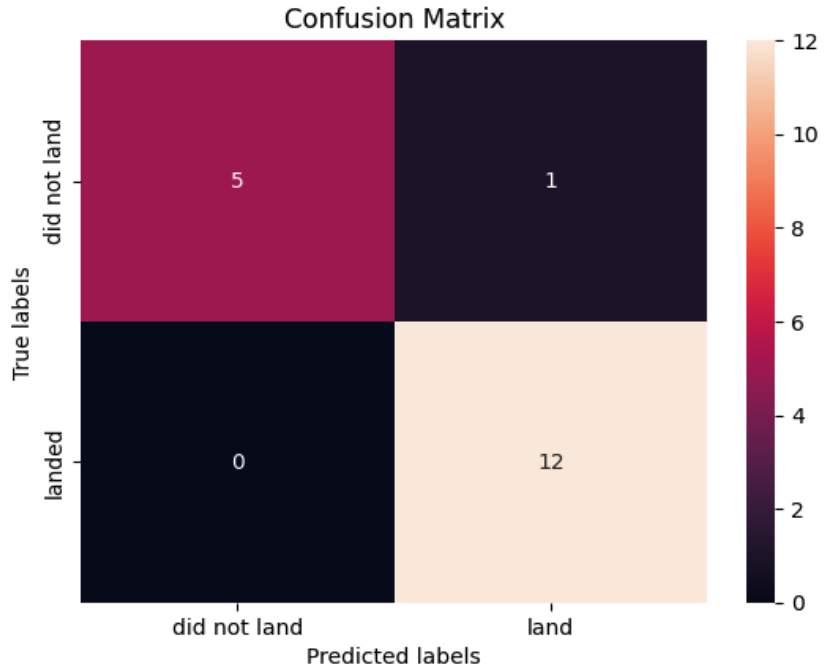
Calculate the accuracy of tree_cv on the test data using the method `score`:

[25]:
```
print(f"Decission tree test accuracy: {tree_cv.score(X_test, Y_test)}")
print(f"Decission tree train accuracy: {tree_cv.score(X_train, Y_train)}")
```

```
Decission tree test accuracy: 0.9444444444444444
Decission tree train accuracy: 0.7638888888888888
```

We can plot the confusion matrix

[26]:
```
yhat = tree_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



Conclusions

- Point 1
- Point 2
- Point 3
- Point 4
- …

Thank you!