Winning Space Race with Data Science

Ghazal Nazari
December 2023

Outline

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

Executive Summary

Summary of methodologies :

- Data Collection
- Data Wrangling
- EDA with Data Visualization
- EDA with SQL
- Building an interactive map with Folium
- Building a Dashboard with Plotly Dash
- Predictive Analysis (Classification)

Summary of all results :

- EDA results
- Interactive analytics
- Predictive analysis

Introduction

- Project background and context

The capstone project centers on predicting the success of Falcon 9 first stage landings in SpaceX rocket launches. SpaceX advertises cost savings due to the reusable first stage, offering launches at $62 million, significantly less than competitors. The aim is to develop a predictive model for landing success, providing valuable cost estimates for launches. This predictive capability is particularly crucial for companies bidding against SpaceX.

- Problems you want to find answers

a) **First Stage Landing Prediction:** The primary goal is to predict Falcon 9 first stage landing success, enabling accurate cost estimation.

b) **Cost Estimation:** Successful landings impact launch costs, making accurate predictions essential for budgeting and competitive bidding.

c) **Competitive Bidding Advantage:** Predictive insights empower companies bidding against SpaceX, aiding strategic decision-making during the bidding process.

d) **Data Collection and Formatting:** The lab involves collecting API data and ensuring proper formatting for effective model training and validation.

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - SpaceX Rest API
  - Web Scrapping from Wikipedia

  Perform data wrangling
  - One Hot Encoding data field for Machine Learning and data cleaning of null values and irrelevant columns

  Perform exploratory data analysis (EDA) using visualization and SQL

  Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - LR, KNN, SVM, and DT models have been built and evaluated for the best classifier

Data Collection

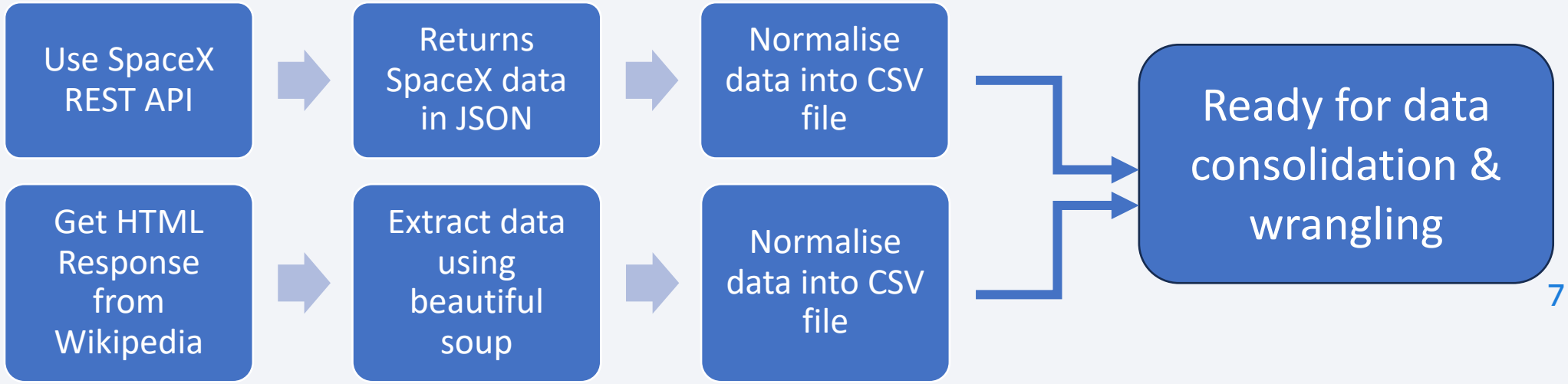
The following datasets were collected:

• SpaceX launch data that is gathered from the SpaceX REST API.

• This API will give us data about launches, including information about the rocket used, payload delivered, launch specifications, landing specifications, and landing outcome.

• The SpaceX REST API endpoints, or URL, starts with api.spacex.com/v4/.

• Another popular source for obtaining Falcon 9 Launch data is web scraping Wikipedia using BeautifulSoup .



Data Collection – SpaceX API

- Data collection with SpaceX REST calls
- https://github.com/ghazalna/Falcon9-Landing-Predictor/blob/main/Request%20to%20the%20SpaceX%20API.ipynb

Request and parse the SpaceX launch data using the GET request

1- Getting Response from API

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

Double-click (or enter) to edit

2 - Converting Response to a .jason file

```
[9]  response.status_code
```

```
200
```

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

3 - Apply custom functions to clean data

```
[15] # Call getBoosterVersion
     getBoosterVersion(data)
```

```
[16] BoosterVersion[0:5]
```

```
['Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 9']
```

```
[17] # Call getLaunchSite
     getLaunchSite(data)
```

```
[18] # Call getPayloadData
     getPayloadData(data)
```

```
[19] # Call getCoreData
     getCoreData(data)
```

4 - Assign list to dictionary then DataFrame

```
launch_dict = {'FlightNumber': list(data['flight_number']),
'Date': list(data['date']),
'BoosterVersion':BoosterVersion,
'PayloadMass':PayloadMass,
'Orbit':Orbit,
'LaunchSite':LaunchSite,
'Outcome':Outcome,
'Flights':Flights,
'GridFins':GridFins,
'Reused':Reused,
'Legs':Legs,
'LandingPad':LandingPad,
'Block':Block,
'ReusedCount':ReusedCount,
'Serial':Serial,
'Longitude': Longitude,
'Latitude': Latitude}
```

```
[21] # Create a data from launch_dict
     data = pd.DataFrame(launch_dict)
```

5 - Filter the dataframe to only include `Falcon 9` launches

```
[23] # Hint data['BoosterVersion']!='Falcon 1'
     data_falcon9=data[data['BoosterVersion']=='Falcon 9']
     data_falcon9.head()
```

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block
4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0
5	8	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0
6	10	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False	False	False	None	1.0
7	11	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None	1.0
8	12	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1.0

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

6 - Data Wrangling

```
[25] data_falcon9.isnull().sum()
```

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

7 - Dealing with Missing Values

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

     # Replace the np.nan values with its mean value
     data['PayloadMass'].replace(np.nan,mean,inplace=True)
     data.head(20)

     #you can see than payloadMass nan values has been replaced by mean value
```

8 - Export to flat files (.CSV)

```
[27] data_falcon9.to_csv('dataset_part\_1.csv', index=False)
```

Data Collection - Scraping

- We applied web scrapping to webscrap Falcon 9 launch records with BeautifulSoup
- We parsed the table and converted it into a pandas dataframe.
- The link to the notebook is :https://github.com/ghazalna/Falcon9-Landing-Predictor/blob/main/Data_Collection_with_Web_Scraping.ipynb

1. Apply HTTP Get method to request the Falcon 9 rocket launch page

```
In [4]: static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"
```

```
In [5]: # use requests.get() method with the provided static_url
        # assign the response to a object
        html_data = requests.get(static_url)
        html_data.status_code
Out[5]: 200
```

2. Create a BeautifulSoup object from the HTML response

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

3. Extract all column names from the HTML table header

```
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 column_names

         element = soup.find_all('th')
         for row in range(len(element)):
             try:
                 name = extract_column_from_header(element[row])
                 if (name is not None and len(name) > 0):
                     column_names.append(name)
             except:
                 pass
```

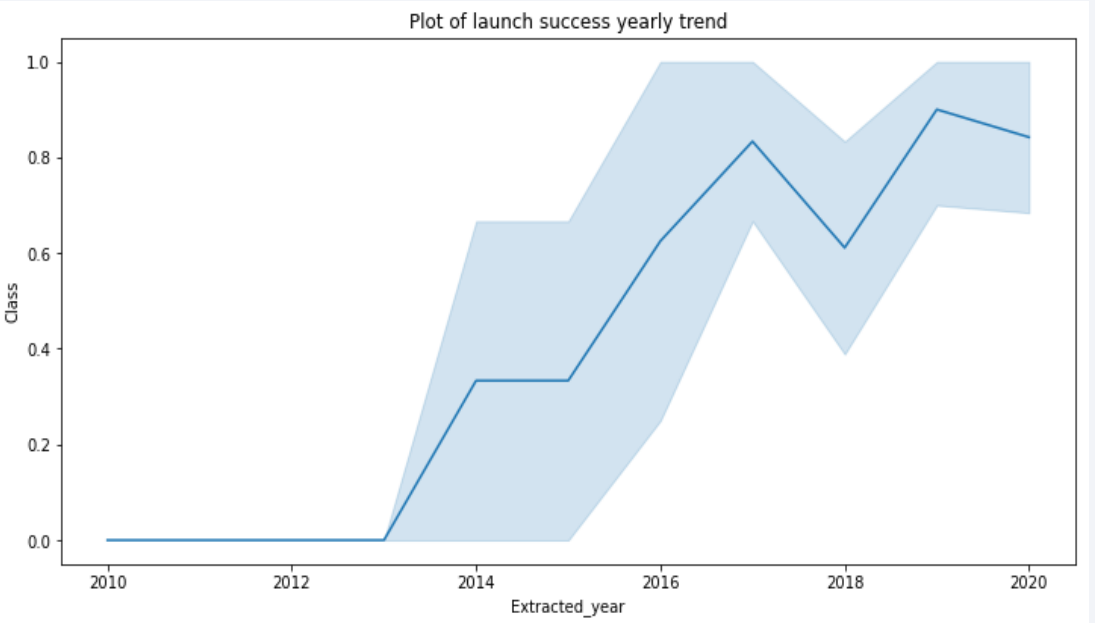
4. Create a dataframe by parsing the launch HTML tables
5. Export data to csv

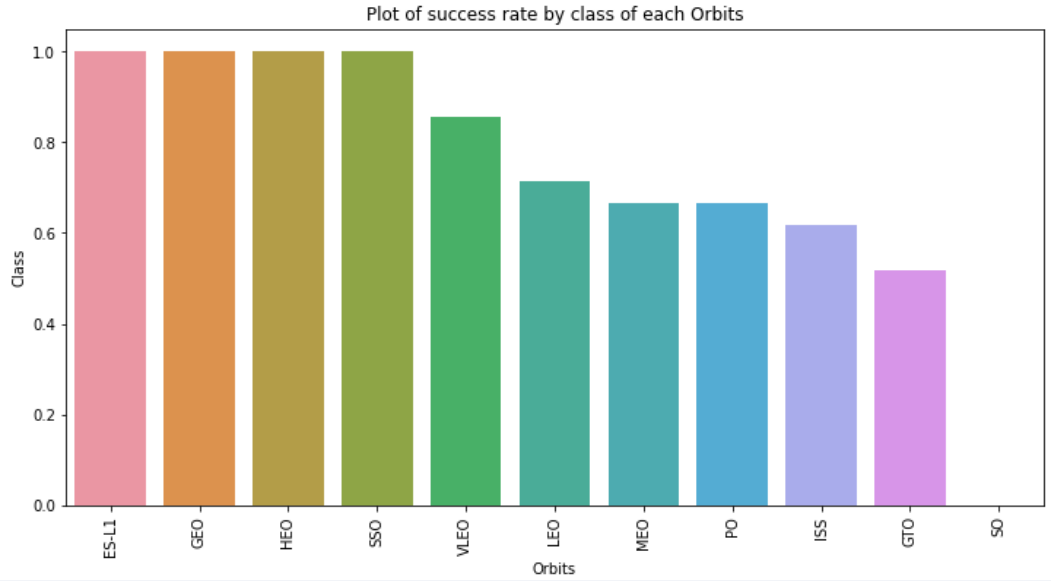
Data Wrangling

- We performed exploratory data analysis and determined the training labels.
- We calculated the number of launches at each site, and the number and occurrence of each orbits
- We created landing outcome label from outcome column and exported the results to csv.
- The link to the notebook is: https://github.com/ghazalna/Falcon9-Landing-Predictor/blob/main/Data_Wrangling.ipynb

EDA with Data Visualization

- We explored the data by visualizing the relationship between flight number and launch Site, payload and launch site, success rate of each orbit type, flight number and orbit type, the launch success yearly trend.





- The link to the notebook is https://github.com/ghazalna/Falcon9-Landing-Predictor/blob/main/EDA_with_Data_Visualization.ipynb

EDA with SQL

- We loaded the SpaceX dataset into a PostgreSQL database without leaving the jupyter notebook.
- We applied EDA with SQL to get insight from the data. We wrote queries to find out for instance:
  - The names of unique launch sites in the space mission.
  - The total payload mass carried by boosters launched by NASA (CRS)
  - The average payload mass carried by booster version F9 v1.1
  - The total number of successful and failure mission outcomes
  - The failed landing outcomes in drone ship, their booster version and launch site names.
- The link to the notebook is https://github.com/ghazalna/Falcon9-Landing-Predictor/blob/main/SQL_Notebook_.ipynb

EDA with SQL

- SQL queries

The Spacex DataSet

- !pip install sqlalchemy==1.3.9
- !pip install sqlalchemy==1.3.9
- !pip install -q pandas==1.1.5
- !pip install --upgrade pandas
- import pandas as pd

EDA with SQL

Connect to the database

- `%load_ext google.colab.data_table`
- `%load_ext sql`
- `import csv, sqlite3`

```
con = sqlite3.connect("my_data1.db")
cur = con.cursor()
```

- `%sql sqlite:///my_data1.db`
- `import pandas as pd`

```
df = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/labs/module_2/data/Spacex.csv")
df.to_sql("SPACEXTBL", con, if_exists='replace', index=False,method="multi")
```

- `%sql SELECT DISTINCT LaunchSite FROM SPACEXTABLE;`
- `cur.execute("PRAGMA table_info(SPACEXTABLE)")`

```
table_info = cur.fetchall()
print(table_info)
```

EDA with SQL

```
# Verify the existence of the table
```

- `tables = pd.read_sql_query("SELECT name FROM sqlite_master WHERE type='table';", con)`
- `print(tables)`
- `%sql create table SPACEXTABLE as select * from SPACEXTBL where Date is not null`

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

- `%sql SELECT DISTINCT LaunchSite FROM SPACEXTABLE;`

EDA with SQL

```
#Display the names of the unique launch sites in the space mission
%sql SELECT DISTINCT LaunchSite FROM SPACEXTABLE;
#Display 5 records where launch sites begin with the string 'CCA'
import sqlite3

# Connect to the SQLite database
con = sqlite3.connect("my_data1.db")

# Create a cursor object
cur = con.cursor()

# Fetch the table schema
cur.execute("PRAGMA table_info(SPACEXTBL)")
table_info = cur.fetchall()

# Display the table schema
print(table_info)
```

EDA with SQL

```
# Display total payload mass carried by boosters launched by NASA (CRS)
import sqlite3

import sqlite3

# Connect to the SQLite database
con = sqlite3.connect("my_data1.db")

# Create a cursor object
cur = con.cursor()

# Fetch the table schema
cur.execute("PRAGMA table_info(SPACEXTBL)")
table_info = cur.fetchall()

# Display the column names
column_names = [info[1] for info in table_info]
print(column_names)
```

EDA with SQL

```
# Display average payload mass carried by booster version F9 v1.1
import sqlite3

# Connect to the SQLite database
con = sqlite3.connect("my_data1.db")

# Create a cursor object
cur = con.cursor()

# Fetch the table schema
cur.execute("PRAGMA table_info(SPACEXTBL)")
table_info = cur.fetchall()

# Display the column names
column_names = [info[1] for info in table_info]
print(column_names)
```

EDA with SQL

```
# Display average payload mass carried by booster version F9 v1.1
import sqlite3

# Connect to the SQLite database
con = sqlite3.connect("my_data1.db")

# Create a cursor object
cur = con.cursor()

# Run the SQL query with the correct column name
cur.execute("SELECT AVG(PAYLOAD_MASS__KG_) AS AveragePayloadMass FROM SPACEXTBL WHERE Booster_Version = 'F9 v1.1'")

# Fetch and display the result
result = cur.fetchone()
print(result)
```

EDA with SQL

```
# List the date when the first successful landing outcome on the ground pad was achieved
import sqlite3

# Connect to the SQLite database
con = sqlite3.connect("my_data1.db")

# Create a cursor object
cur = con.cursor()

# Run the SQL query
cur.execute("SELECT MIN(Date) AS FirstSuccessfulLandingDate FROM SPACEXTBL WHERE Landing_Outcome = 'Success (ground pad)'")

# Fetch and display the result
result = cur.fetchone()
print(result)
```

EDA with SQL

```
# List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000
import sqlite3

# Connect to the SQLite database
con = sqlite3.connect("my_data1.db")

# Create a cursor object
cur = con.cursor()

# Run the SQL query with the correct column names
cur.execute("SELECT Booster_Version FROM SPACEXTBL WHERE Landing_Outcome = 'Success (drone ship)' AND PAYLOAD_MASS__KG_ > 4000 AND PAYLOAD_MASS__KG_ < 6000")

# Fetch and display the result
result = cur.fetchall()
print(result)
```

EDA with SQL

```
# List the total number of successful and failure mission outcomes
import sqlite3

# Connect to the SQLite database
con = sqlite3.connect("my_data1.db")

# Create a cursor object
cur = con.cursor()

# Run the SQL query
cur.execute("SELECT Mission_Outcome, COUNT(*) AS TotalCount FROM SPACEXTBL GROUP BY Mission_Outcome")

# Fetch and display the result
result = cur.fetchall()
print(result)
```

EDA with SQL

```
#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.
import sqlite3
# Connect to the SQLite database
con = sqlite3.connect("my_data1.db")
# Create a cursor object
cur = con.cursor()
# Run the corrected SQL query
cur.execute("""
SELECT substr(Date, 6, 2) AS Month, Landing_Outcome, Booster_Version, Launch_Site
FROM SPACEXTBL
WHERE substr(Date, 0, 5) = '2015' AND Landing_Outcome LIKE 'Failure%' AND Landing_Outcome IS NOT
NULL
""")
# Fetch and display the result
result = cur.fetchall()
print(result)
```

EDA with SQL

```
#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.
import sqlite3
# Connect to the SQLite database
con = sqlite3.connect("my_data1.db")
# Create a cursor object
cur = con.cursor()
# Run the SQL query to rank the count of landing outcomes
cur.execute("""
SELECT Landing_Outcome, COUNT(*) as Count
FROM SPACEXTBL
WHERE Date BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY Landing_Outcome
ORDER BY Count DESC
""")
# Fetch and display the result
result = cur.fetchall()
print(result)
```

Build an Interactive Map with Folium

- We marked all launch sites, and added map objects such as markers, circles, lines to mark the success or failure of launches for each site on the folium map.
- We assigned the feature launch outcomes (failure or success) to class 0 and 1.i.e., 0 for failure, and 1 for success.
- Using the color-labeled marker clusters, we identified which launch sites have relatively high success rate.
- We calculated the distances between a launch site to its proximities. We answered some question for instance:
  - Are launch sites near railways, highways and coastlines.
  - Do launch sites keep certain distance away from cities.

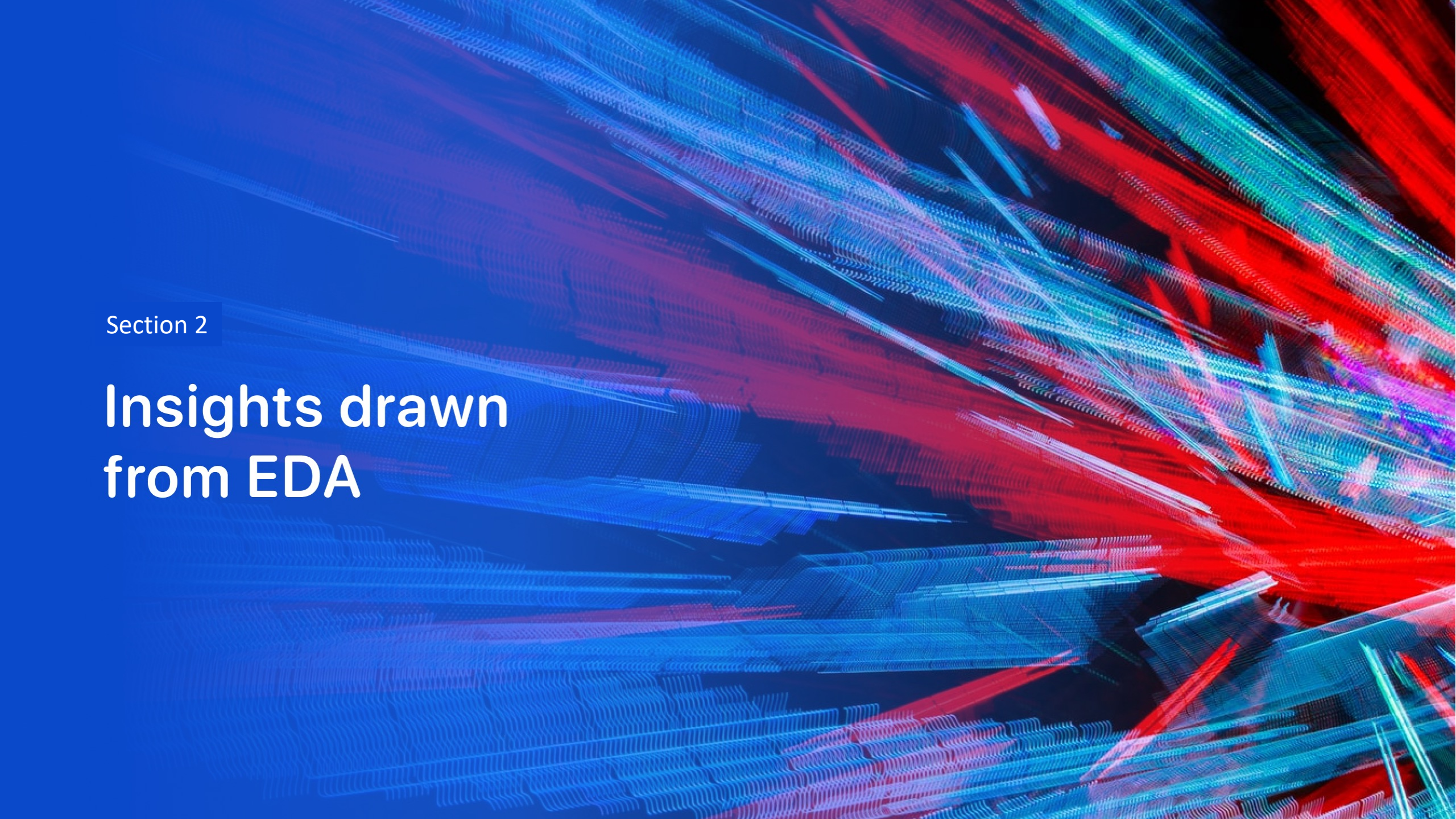
  The link to the notebook is:
  https://github.com/ghazalna/Falcon9-Landing-Predictor/blob/main/Launch_Sites_Locations_Analysis_with_Folium.ipynb

Build a Dashboard with Plotly Dash

- We built an interactive dashboard with Plotly dash
- We plotted pie charts showing the total launches by a certain sites
- We plotted scatter graph showing the relationship with Outcome and Payload Mass (Kg) for the different booster version.
- The link to the notebook is:
  https://github.com/ghazalna/Falcon9-Landing Predictor/blob/main/spacex_dash_app.py

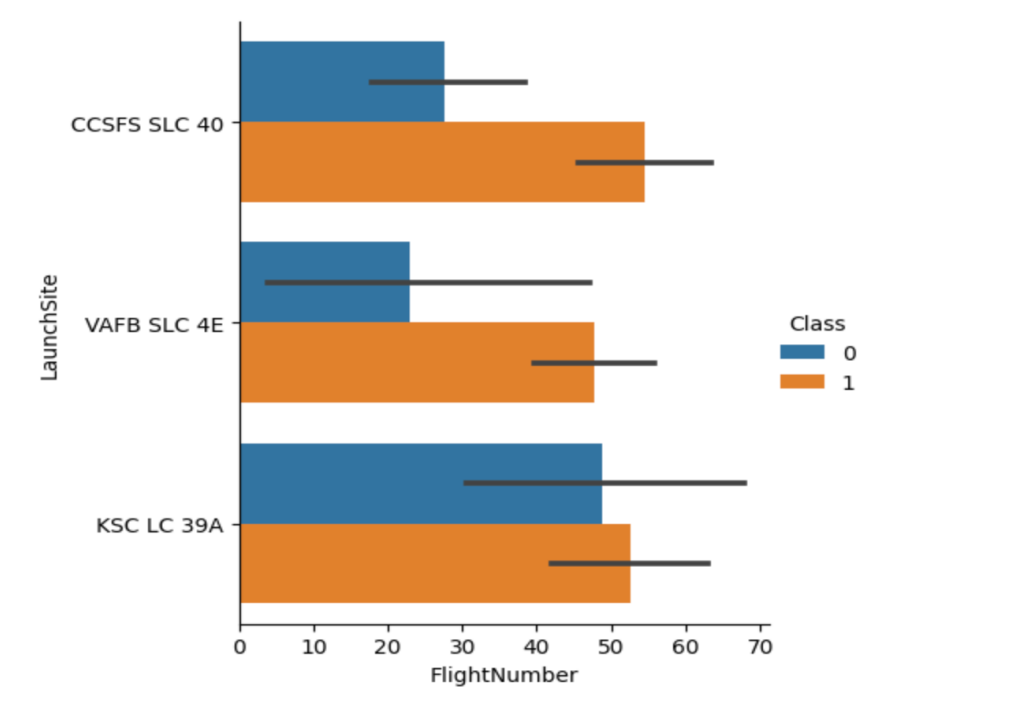
Predictive Analysis (Classification)

- We loaded the data using numpy and pandas, transformed the data, split our data into training and testing.
- We built different machine learning models and tune different hyperparameters using GridSearchCV.
- We used accuracy as the metric for our model, improved the model using feature engineering and algorithm tuning.
- We found the best performing classification model.
- The link to the notebook is https://github.com/ghazalna/Falcon9-Landing-Predictor/blob/main/Falcon9_SpaceX.ipynb

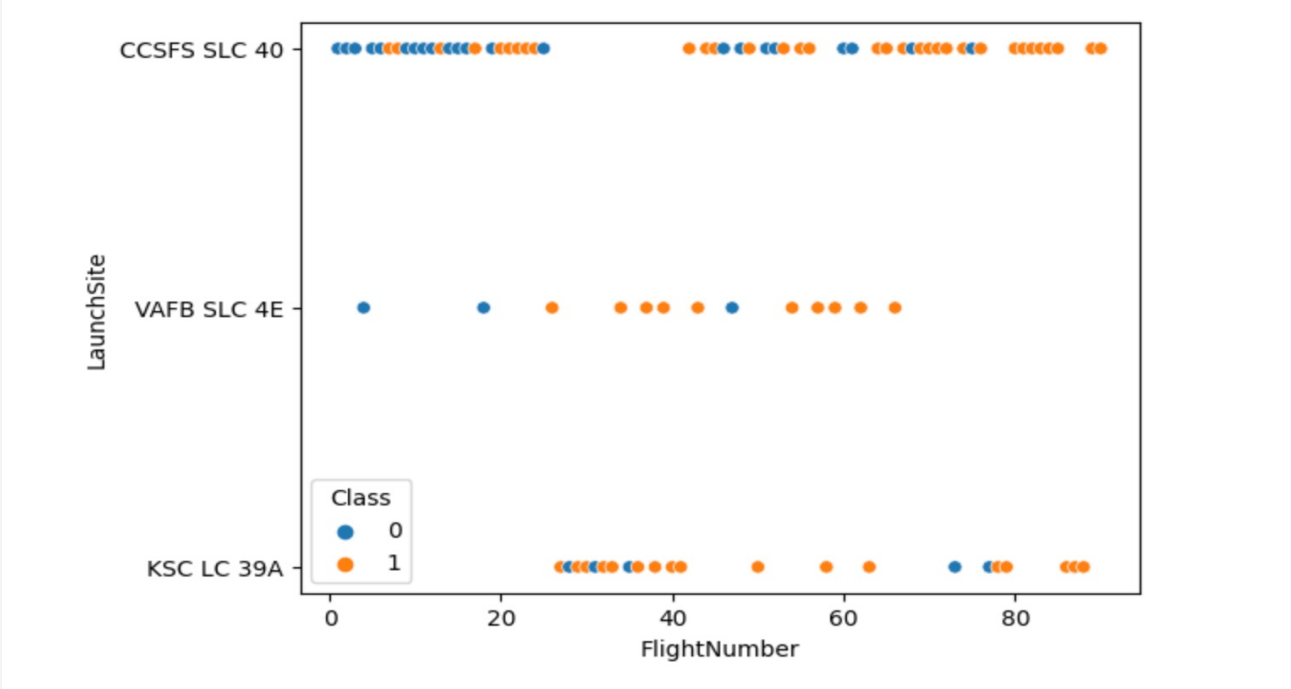

Section 2
Insights drawn from EDA

Flight Number vs. Launch Site

- `# The relationship between Flight Number and Launch Site`
- `sns.catplot(x='FlightNumber', y='LaunchSite', hue='Class', data=df, kind='bar')`



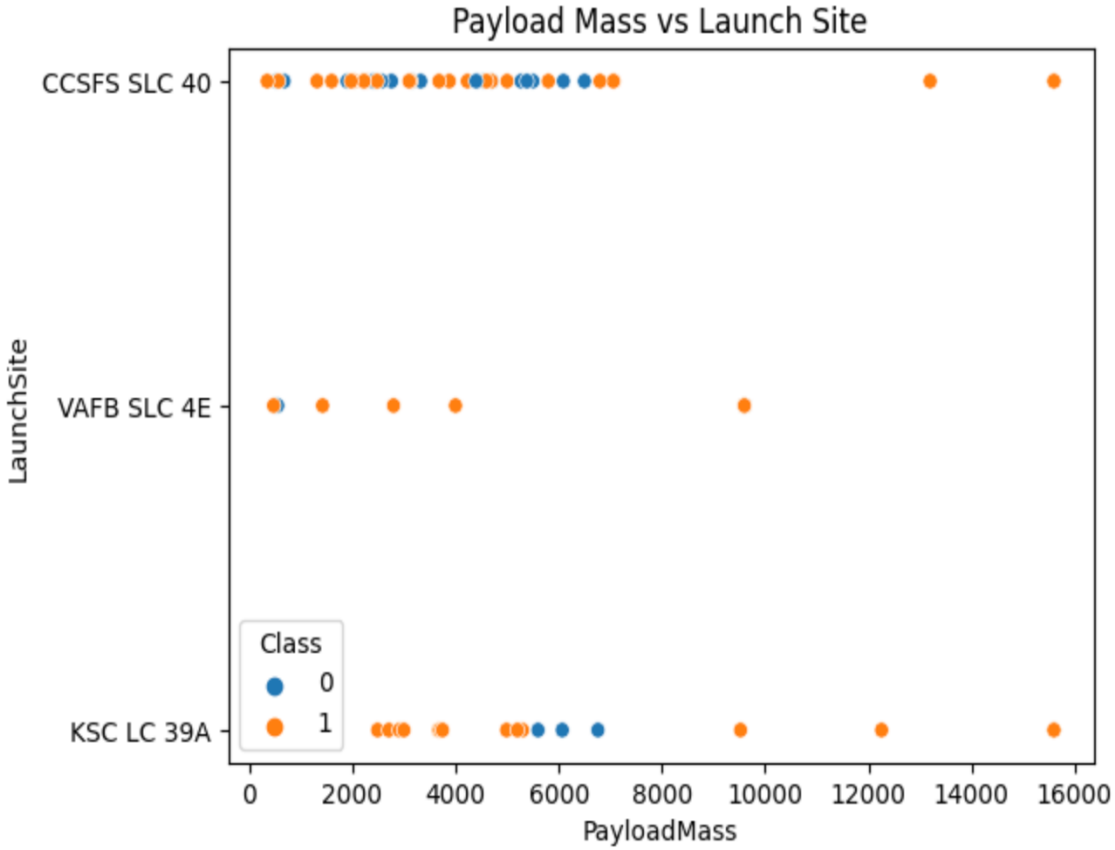
- `sns.scatterplot(x='FlightNumber', y='LaunchSite', hue='Class', data=df)`
- `plt.show()`



- From the plot, we found that the larger the flight amount at a launch site, the greater the success rate at a launch site.

Payload vs. Launch Site

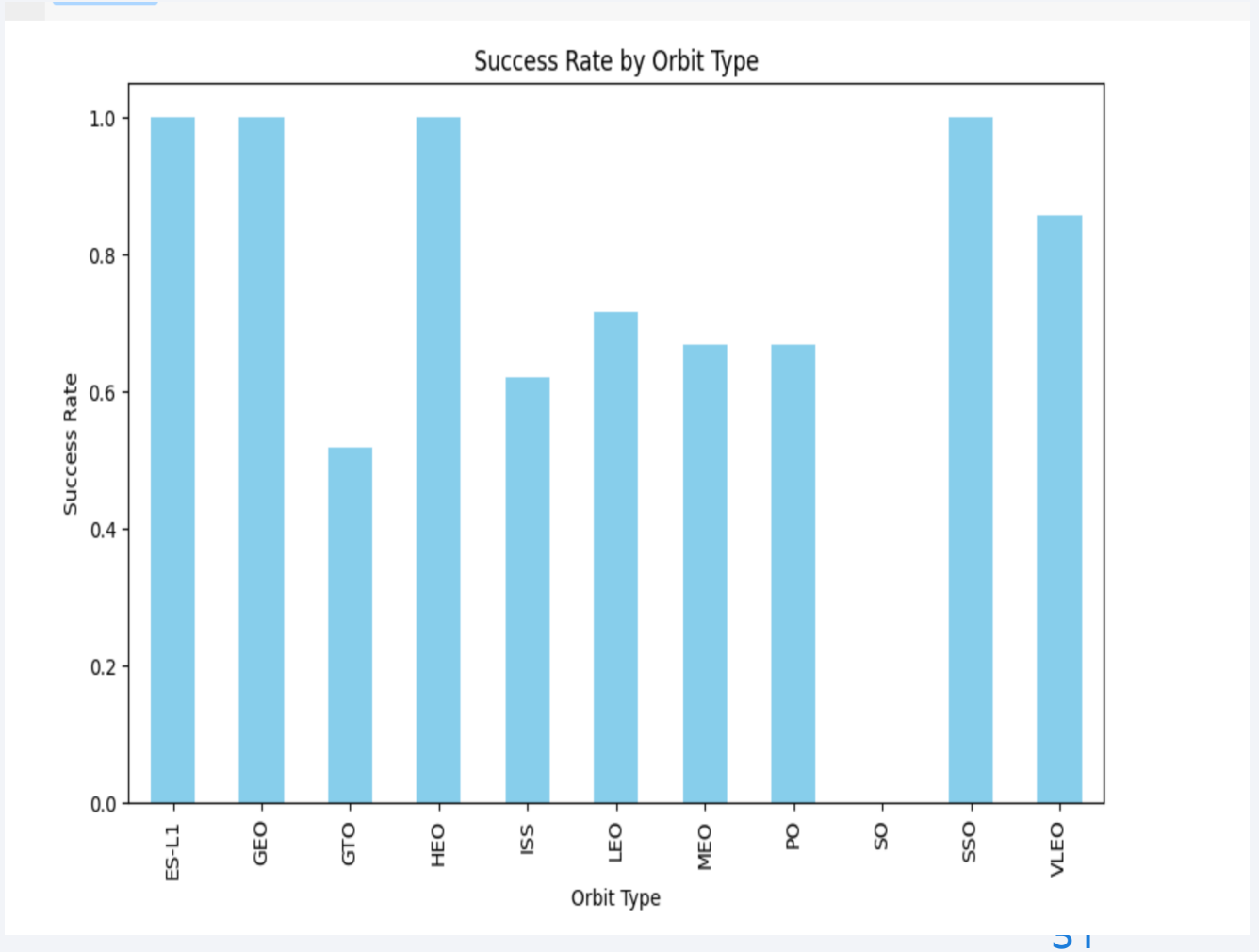
```
#The relationship between Payload and Launch Site
sns.scatterplot(x='PayloadMass', y='LaunchSite', hue='Class', data=df)

# Show the plot
plt.title('Payload Mass vs Launch Site')
plt.show()
```



- The greater the payload mass for launch site CCAFS SLC 40 the higher the success rate for the rocket.

Success Rate vs. Orbit Type

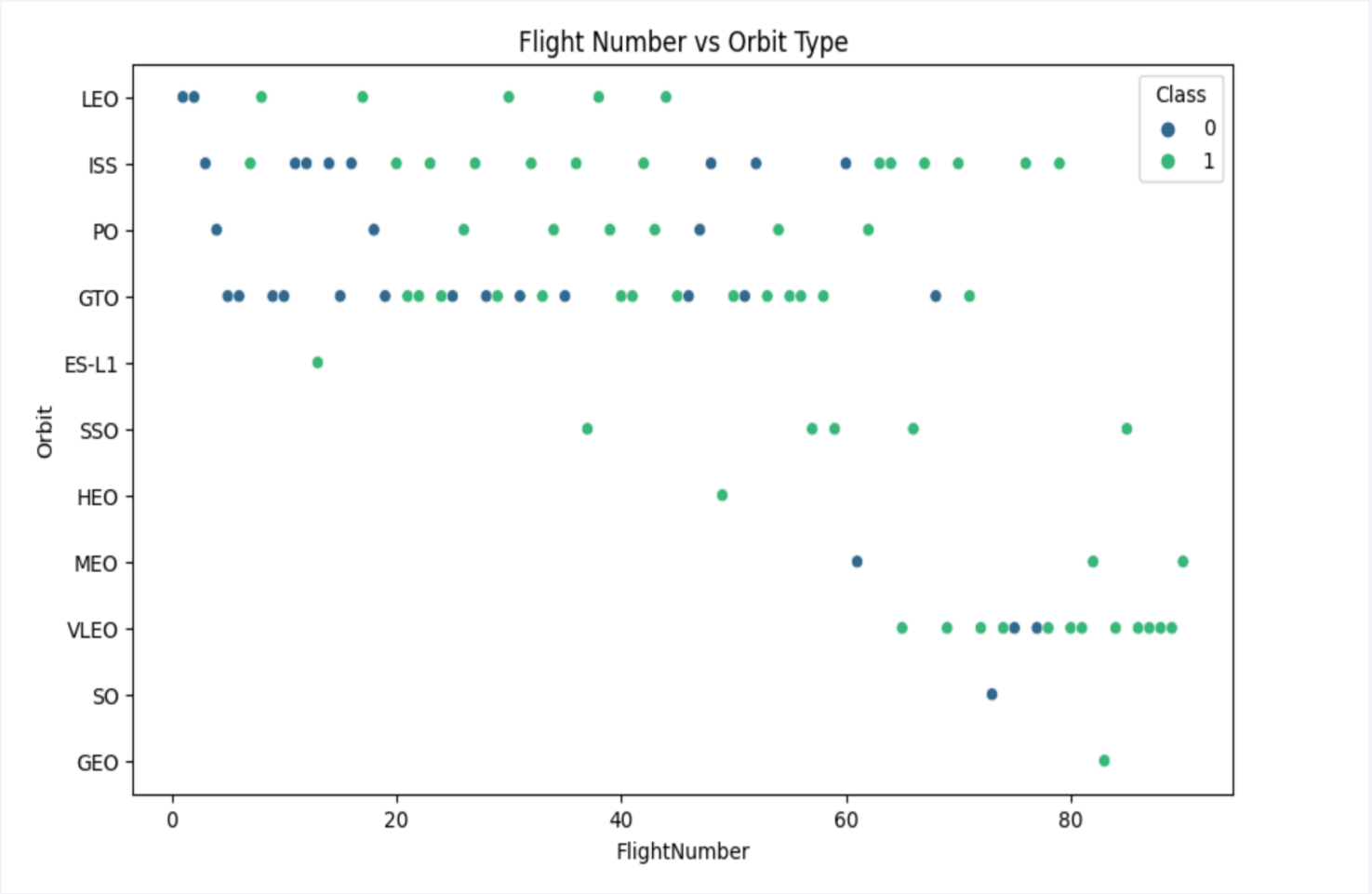
```
#The relationship between success rate of each orbit type
orbit_success_rate = df.groupby('Orbit')['Class'].mean()

# Plotting the bar chart
plt.figure(figsize=(10, 6))
orbit_success_rate.plot(kind='bar', color='skyblue')
plt.title('Success Rate by Orbit Type')
plt.xlabel('Orbit Type')
plt.ylabel('Success Rate')
plt.show()
```



- From the plot, we can see that ES-L1, GEO, HEO, SSO, VLEO had the most success rate.

Flight Number vs. Orbit Type

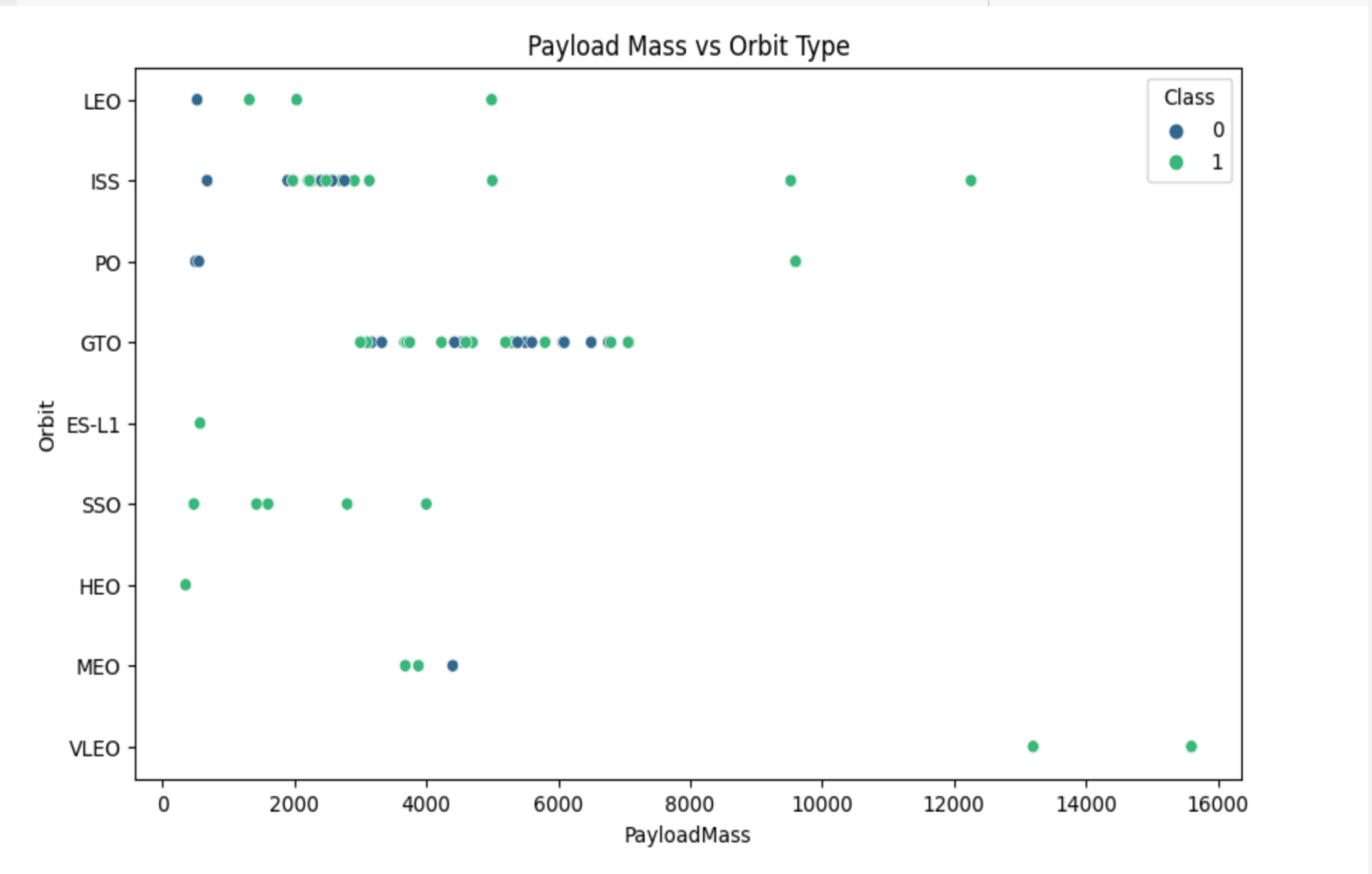
- `#The relationship between FlightNumber and Orbit type`
- `plt.figure(figsize=(10, 6))`
- `sns.scatterplot(x='FlightNumber', y='Orbit', hue='Class', data=df, palette='viridis')`
- `# Show the plot`
- `plt.title('Flight Number vs Orbit Type')`
- `plt.show()`
- The plot below shows the Flight Number vs. Orbit type. We observe that in the LEO orbit, success is related to the number of flights whereas in the GTO orbit, there is no relationship between flight number and the orbit.



Payload vs. Orbit Type

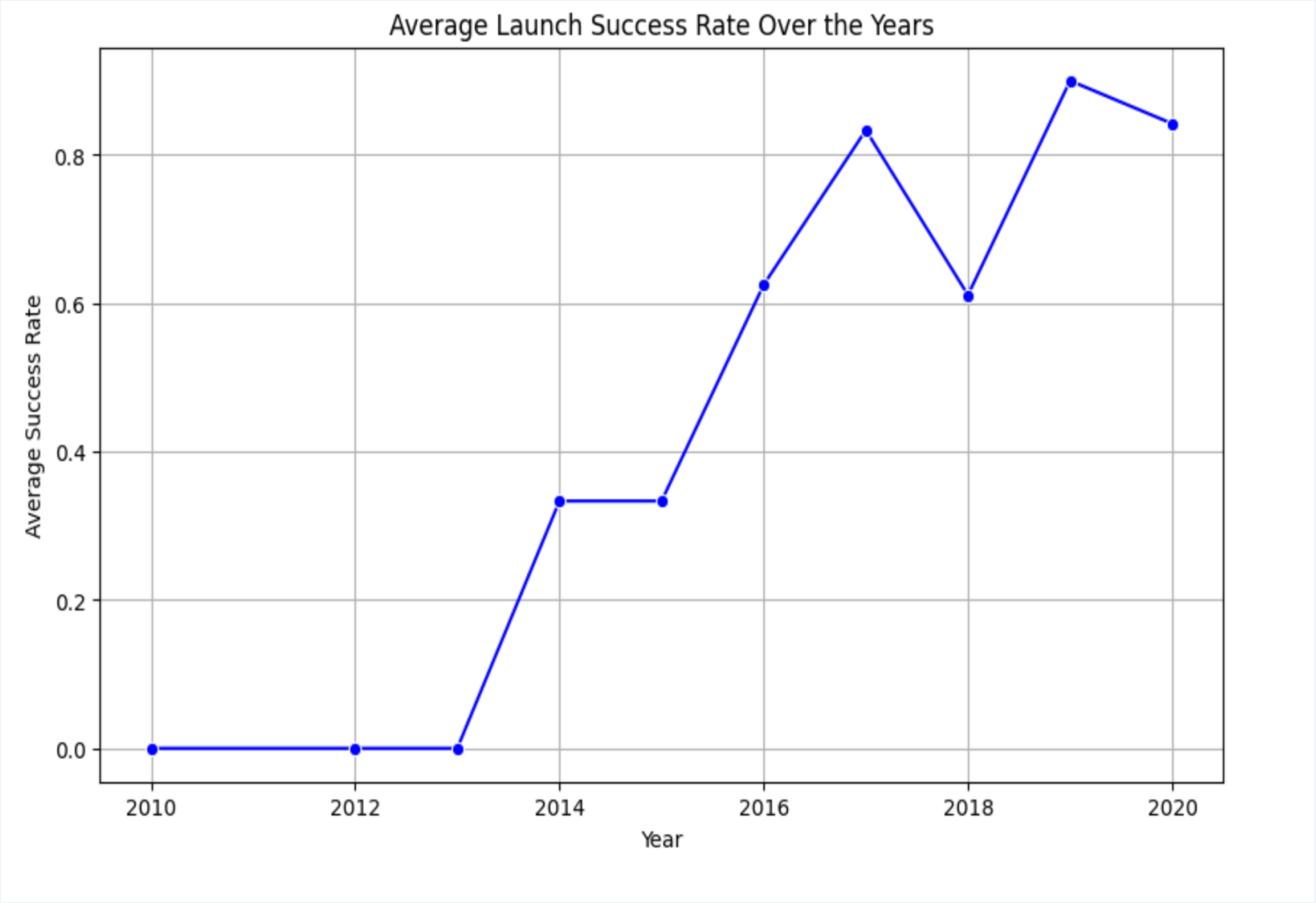
- `#The relationship between Payload and Orbit type`
- `plt.figure(figsize=(10, 6))`
- `sns.scatterplot(x='PayloadMass', y='Orbit', hue='Class', data=df, palette='viridis')`
- `# Show the plot`
- `plt.title('Payload Mass vs Orbit Type')`
- `plt.show()`

- We can observe that with heavy payloads, the successful landing are more for PO, LEO and ISS orbits.



Launch Success Yearly Trend

```
# Extract year from the 'Date' column and create a new column 'LaunchYear'
df['LaunchYear'] = pd.to_datetime(df['Date']).dt.year

# Calculate the average success rate per year
average_success_rate = df.groupby('LaunchYear')['Class'].mean()

# Plotting the line chart
plt.figure(figsize=(10, 6))
sns.lineplot(x=average_success_rate.index, y=average_success_rate.values, marker='o', color='b')
plt.title('Average Launch Success Rate Over the Years')
plt.xlabel('Year')
plt.ylabel('Average Success Rate')
plt.grid(True)
plt.show()
```



- From the plot, we can observe that success rate since 2013 kept on increasing till 2020.

All Launch Site Names

- We used the key word **DISTINCT** to show only unique launch sites from the SpaceX data.
- `df['LaunchSite'].value_counts()`

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

Launch Site Names Begin with 'CCA'

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

In [11]:
```
task_2 = '''
        SELECT *
        FROM SpaceX
        WHERE LaunchSite LIKE 'CCA%'
        LIMIT 5
        '''
create_pandas_df(task_2, database=conn)
```

Out[11]:

	date	time	boosterversion	launchsite	payload	payloadmasskg	orbit	customer	missionoutcome	landingoutcome
0	2010-04-06	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
1	2010-08-12	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of...	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2	2012-05-22	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
3	2012-08-10	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
4	2013-01-03	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

- We used the query above to display 5 records where launch sites begin with `CCA`

Total Payload Mass

- We calculated the total payload carried by boosters from NASA as 45596 using the query below

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

In [12]:
```
task_3 = '''
        SELECT SUM(PayloadMassKG) AS Total_PayloadMass
        FROM SpaceX
        WHERE Customer LIKE 'NASA (CRS)'
        '''
create_pandas_df(task_3, database=conn)
```

Out[12]:

	total_payloadmass
0	45596

Average Payload Mass by F9 v1.1

- We calculated the average payload mass carried by booster version F9 v1.1 as 2928.4

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

In [13]:
```
task_4 = '''
        SELECT AVG(PayloadMassKG) AS Avg_PayloadMass
        FROM SpaceX
        WHERE BoosterVersion = 'F9 v1.1'
        '''
create_pandas_df(task_4, database=conn)
```

Out[13]:

	avg_payloadmass
0	2928.4

First Successful Ground Landing Date

- We observed that the dates of the first successful landing outcome on ground pad was 22nd December 2015

In [14]:

```
task_5 = '''
        SELECT MIN(Date) AS FirstSuccessfull_landing_date
        FROM SpaceX
        WHERE LandingOutcome LIKE 'Success (ground pad)'
        '''
create_pandas_df(task_5, database=conn)
```

Out[14]:

	firstsuccessfull_landing_date
0	2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

- We used the **WHERE** clause to filter for boosters which have successfully landed on drone ship and applied the **AND** condition to determine successful landing with payload mass greater than 4000 but less than 6000

In [15]:

```
task_6 = '''
        SELECT BoosterVersion
        FROM SpaceX
        WHERE LandingOutcome = 'Success (drone ship)'
            AND PayloadMassKG > 4000
            AND PayloadMassKG < 6000
        '''
create_pandas_df(task_6, database=conn)
```

Out[15]:

	boosterversion
0	F9 FT B1022
1	F9 FT B1026
2	F9 FT B1021.2
3	F9 FT B1031.2

Total Number of Successful and Failure Mission Outcomes

- We used wildcard like '%' to filter for **WHERE** MissionOutcome was a success or a failure.

List the total number of successful and failure mission outcomes

In [16]:
```
task_7a = '''
        SELECT COUNT(MissionOutcome) AS SuccessOutcome
        FROM SpaceX
        WHERE MissionOutcome LIKE 'Success%'
        '''

task_7b = '''
        SELECT COUNT(MissionOutcome) AS FailureOutcome
        FROM SpaceX
        WHERE MissionOutcome LIKE 'Failure%'
        '''
print('The total number of successful mission outcome is:')
display(create_pandas_df(task_7a, database=conn))
print()
print('The total number of failed mission outcome is:')
create_pandas_df(task_7b, database=conn)
```

Out[16]:

The total number of successful mission outcome is:

	successoutcome
0	100

The total number of failed mission outcome is:

	failureoutcome
0	1

Boosters Carried Maximum Payload

- We determined the booster that have carried the maximum payload using a subquery in the **WHERE** clause and the **MAX()** function.

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

In [17]:
```
task_8 = '''
        SELECT BoosterVersion, PayloadMassKG
        FROM SpaceX
        WHERE PayloadMassKG = (
                                SELECT MAX(PayloadMassKG)
                                FROM SpaceX
                                )
        ORDER BY BoosterVersion
        '''
create_pandas_df(task_8, database=conn)
```

Out[17]:

	boosterversion	payloadmasskg
0	F9 B5 B1048.4	15600
1	F9 B5 B1048.5	15600
2	F9 B5 B1049.4	15600
3	F9 B5 B1049.5	15600
4	F9 B5 B1049.7	15600
5	F9 B5 B1051.3	15600
6	F9 B5 B1051.4	15600
7	F9 B5 B1051.6	15600
8	F9 B5 B1056.4	15600
9	F9 B5 B1058.3	15600
10	F9 B5 B1060.2	15600
11	F9 B5 B1060.3	15600

2015 Launch Records

- We used a combinations of the **WHERE** clause, **LIKE**, **AND**, and **BETWEEN** conditions to filter for failed landing outcomes in drone ship, their booster versions, and launch site names for year 2015

List the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015

In [18]:
```
task_9 = '''
        SELECT BoosterVersion, LaunchSite, LandingOutcome
        FROM SpaceX
        WHERE LandingOutcome LIKE 'Failure (drone ship)'
            AND Date BETWEEN '2015-01-01' AND '2015-12-31'
        '''
create_pandas_df(task_9, database=conn)
```

Out[18]:

	boosterversion	launchsite	landingoutcome
0	F9 v1.1 B1012	CCAFS LC-40	Failure (drone ship)
1	F9 v1.1 B1015	CCAFS LC-40	Failure (drone ship)

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

- We selected Landing outcomes and the **COUNT** of landing outcomes from the data and used the **WHERE** clause to filter for landing outcomes **BETWEEN** 2010-06-04 to 2010-03-20.
- We applied the **GROUP BY** clause to group the landing outcomes and the **ORDER BY** clause to order the grouped landing outcome in descending order.

Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad))

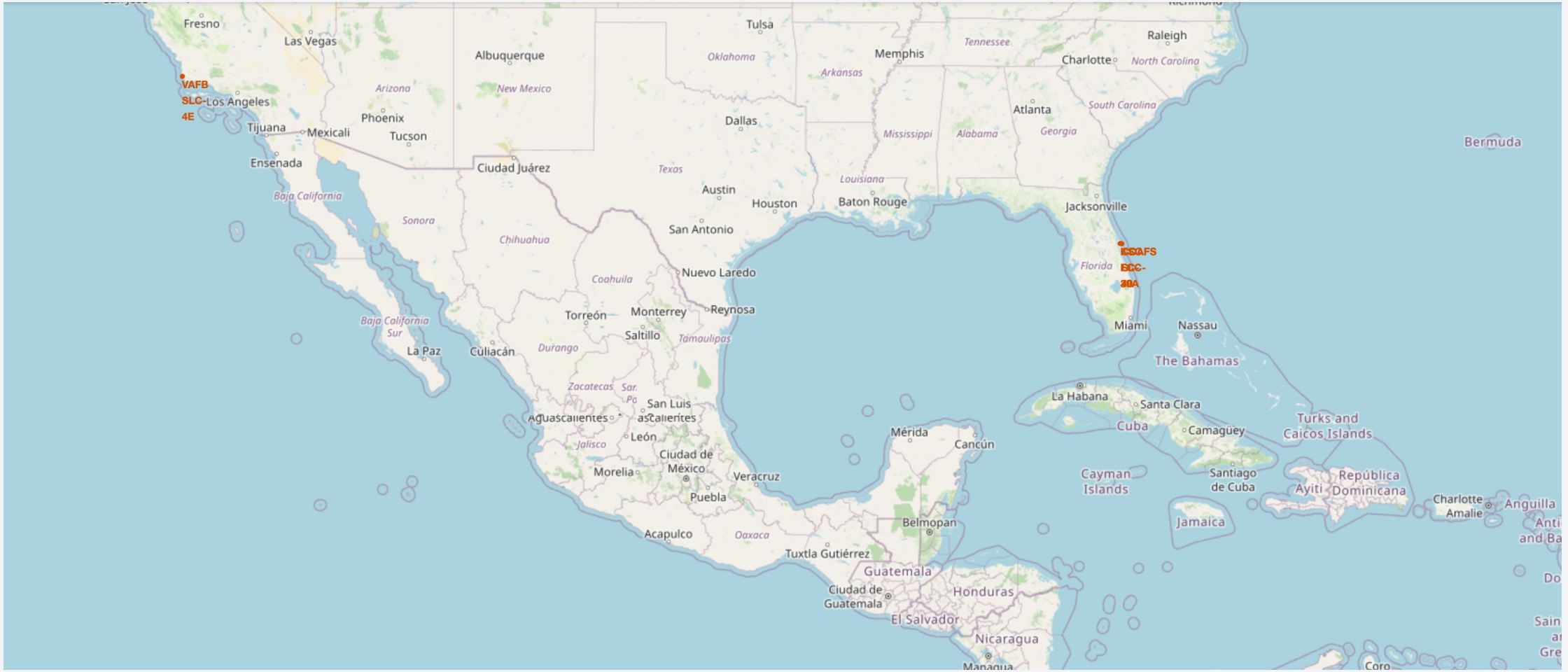
In [19]:
```
task_10 = '''
        SELECT LandingOutcome, COUNT(LandingOutcome)
        FROM SpaceX
        WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20'
        GROUP BY LandingOutcome
        ORDER BY COUNT(LandingOutcome) DESC
        '''
create_pandas_df(task_10, database=conn)
```

Out[19]:

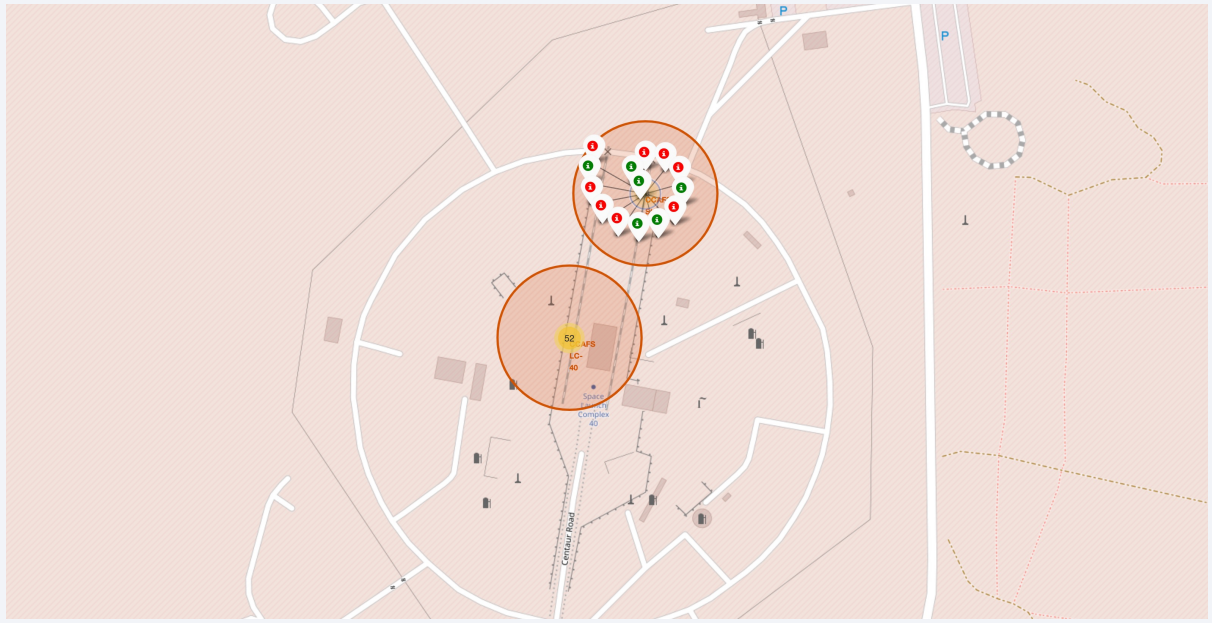
	landingoutcome	count
0	No attempt	10
1	Success (drone ship)	6
2	Failure (drone ship)	5
3	Success (ground pad)	5
4	Controlled (ocean)	3
5	Uncontrolled (ocean)	2
6	Precluded (drone ship)	1
7	Failure (parachute)	1

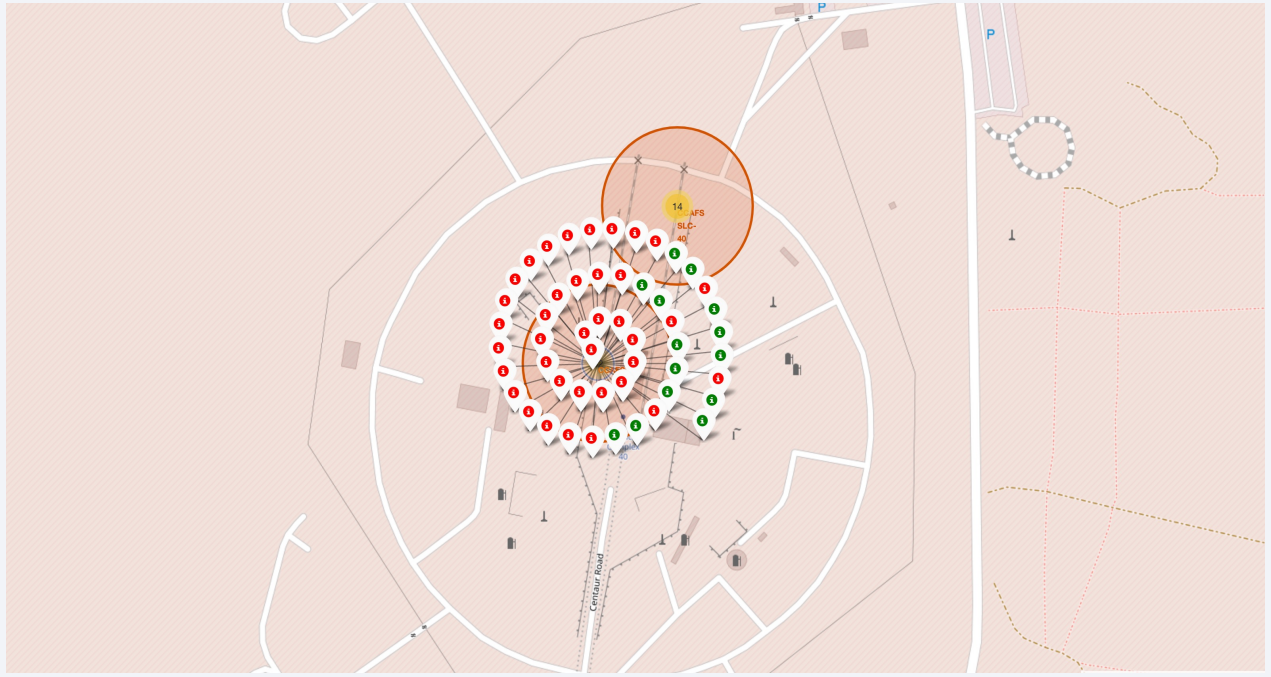

Section 3
Launch Sites
Proximities Analysis

Launch Sites Locations Analysis with Folium

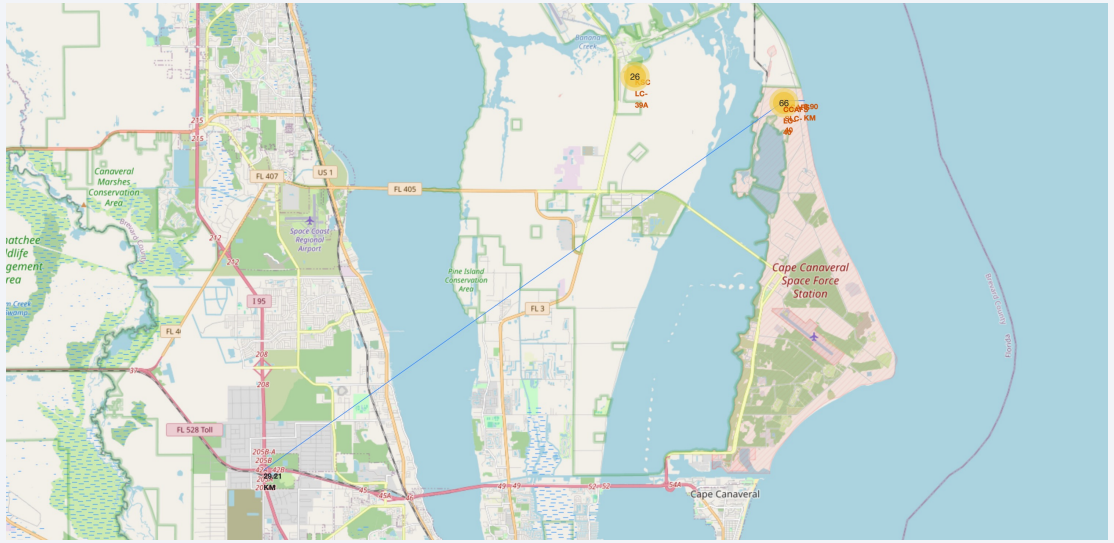


Mark the success/failed launches for each site on the map

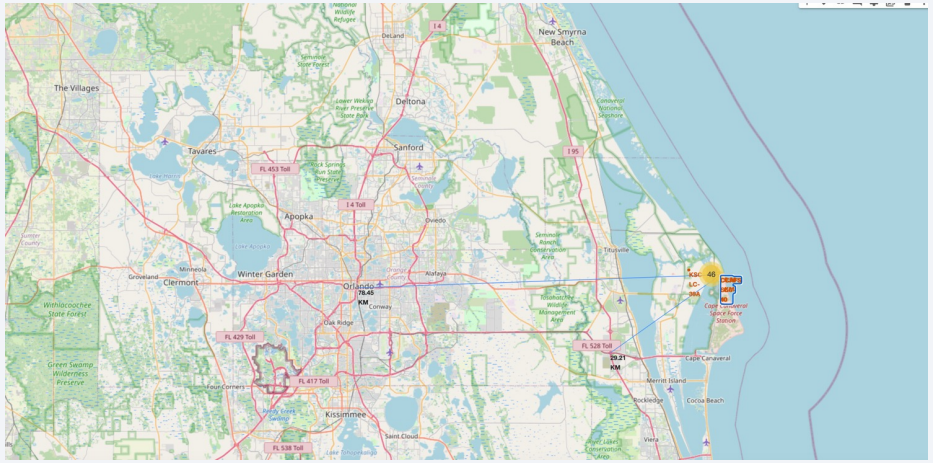




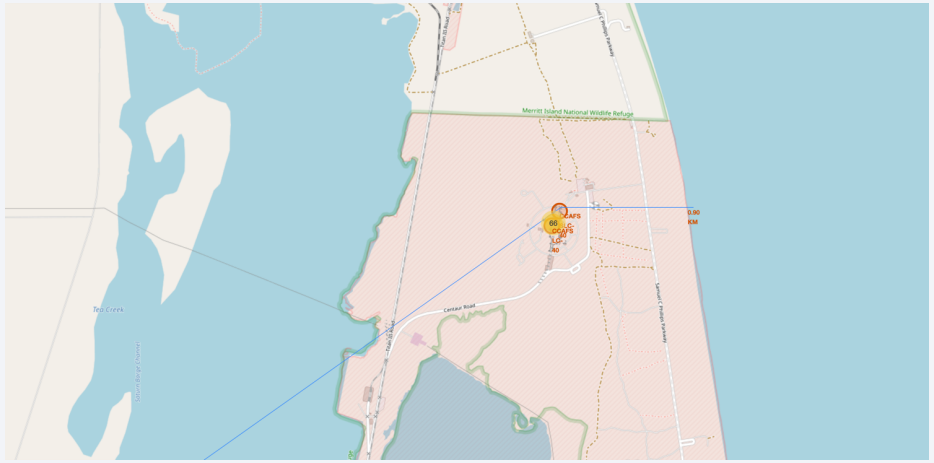
Calculate the distances between a launch site to its proximities



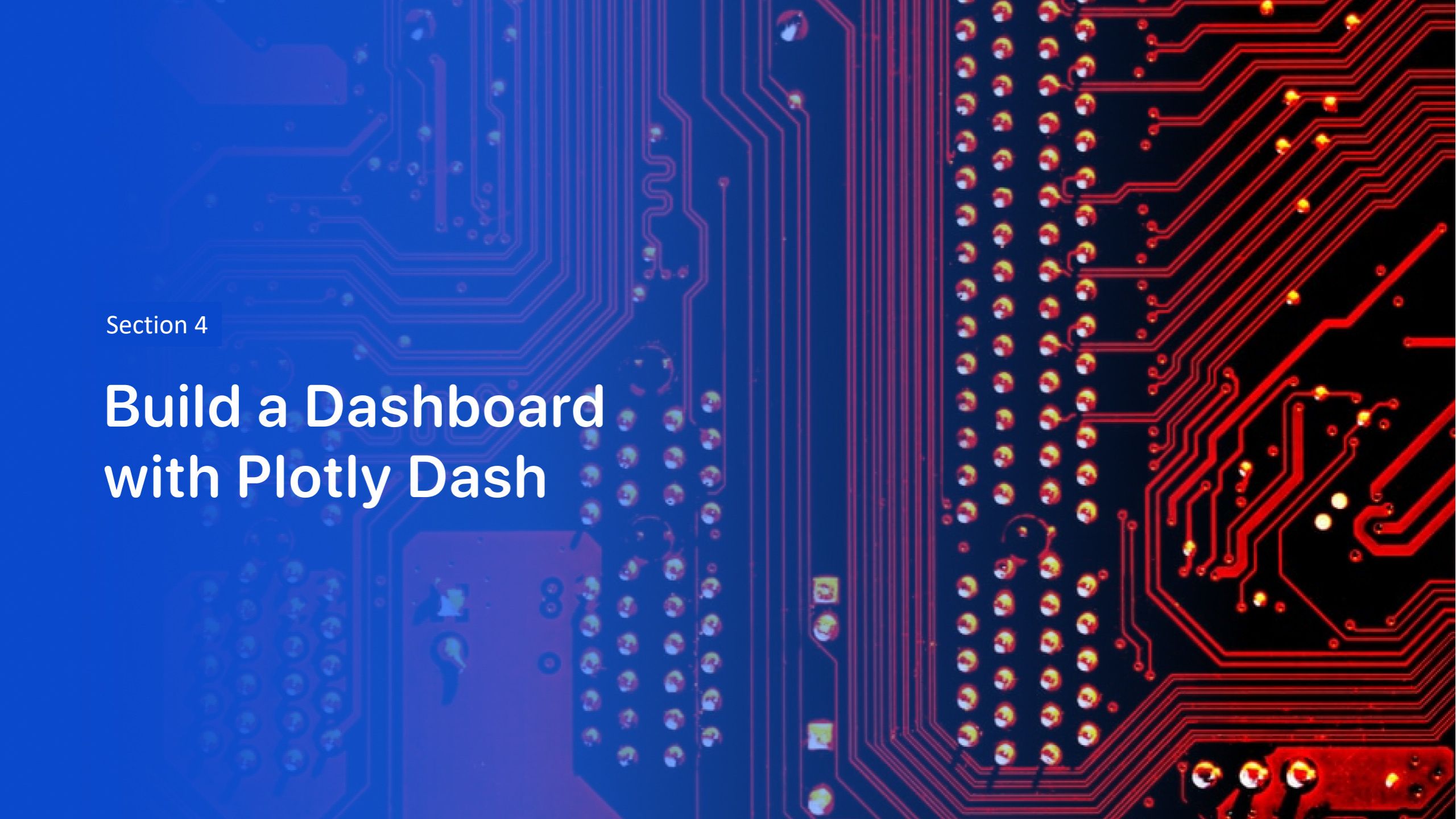
Distance to Highway



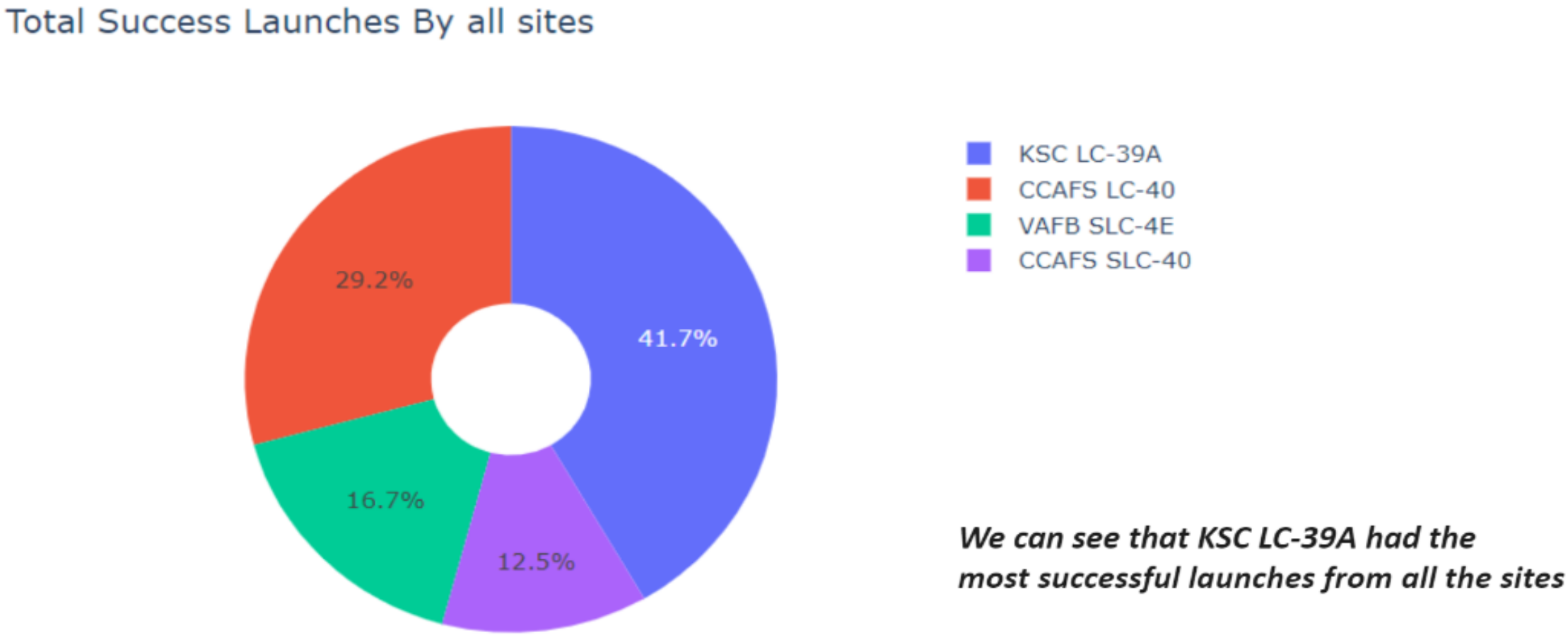
Distance to a closest city



closest coastline point on the map

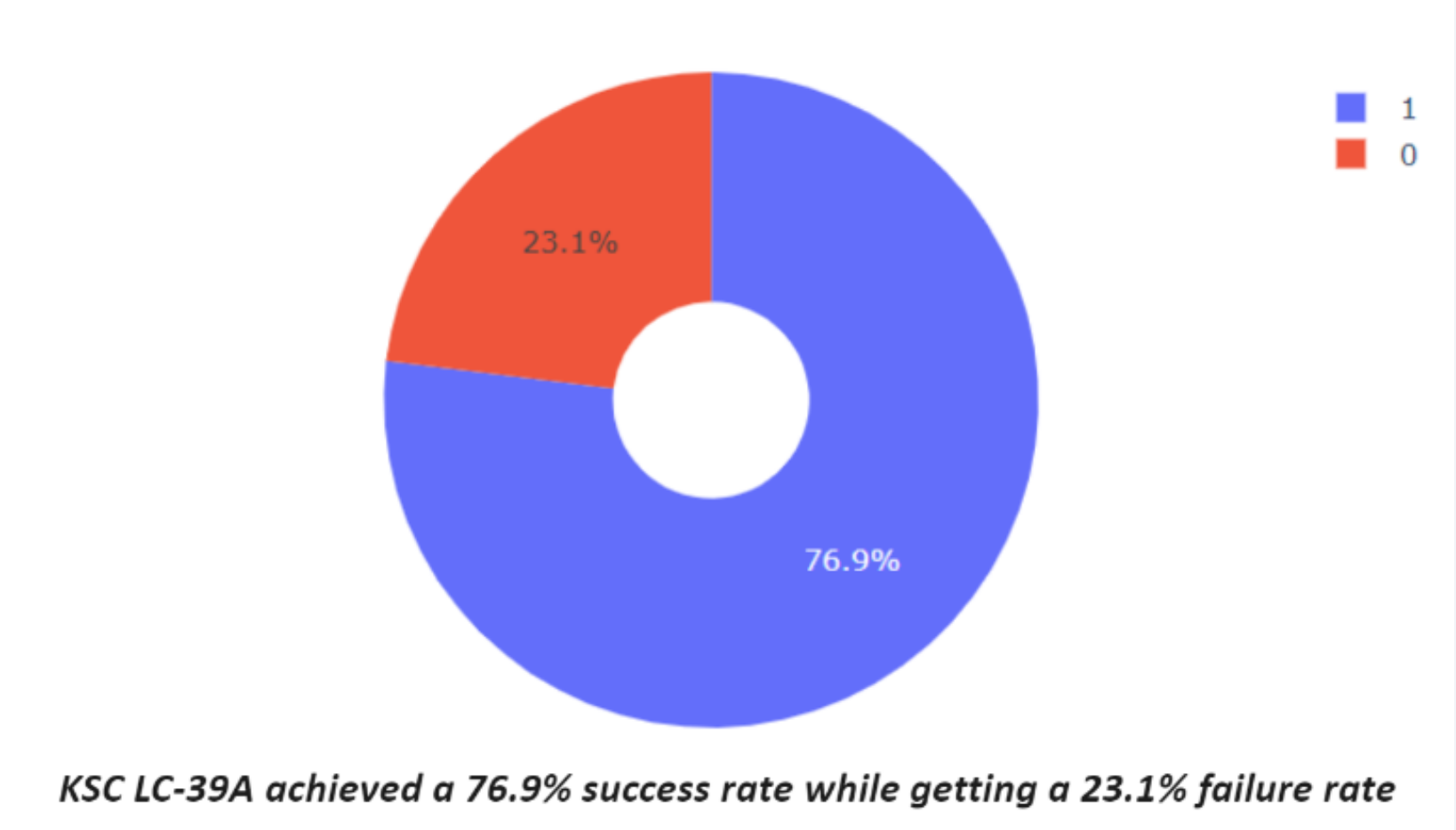

Section 4
Build a Dashboard with Plotly Dash

Pie chart showing the success percentage achieved by each launch site



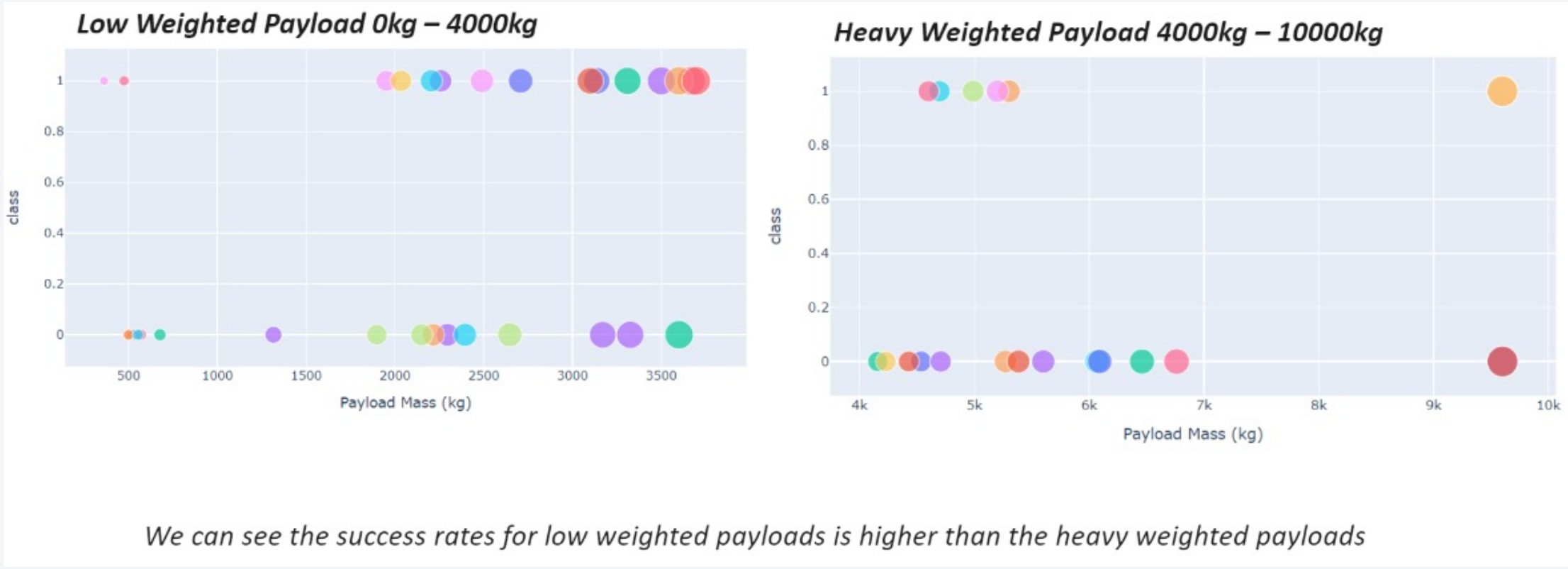
We can see that KSC LC-39A had the most successful launches from all the sites

Pie chart showing the Launch site with the highest launch success ratio



KSC LC-39A achieved a 76.9% success rate while getting a 23.1% failure rate

Scatter plot of Payload vs Launch Outcome for all sites, with different payload selected in the range slider



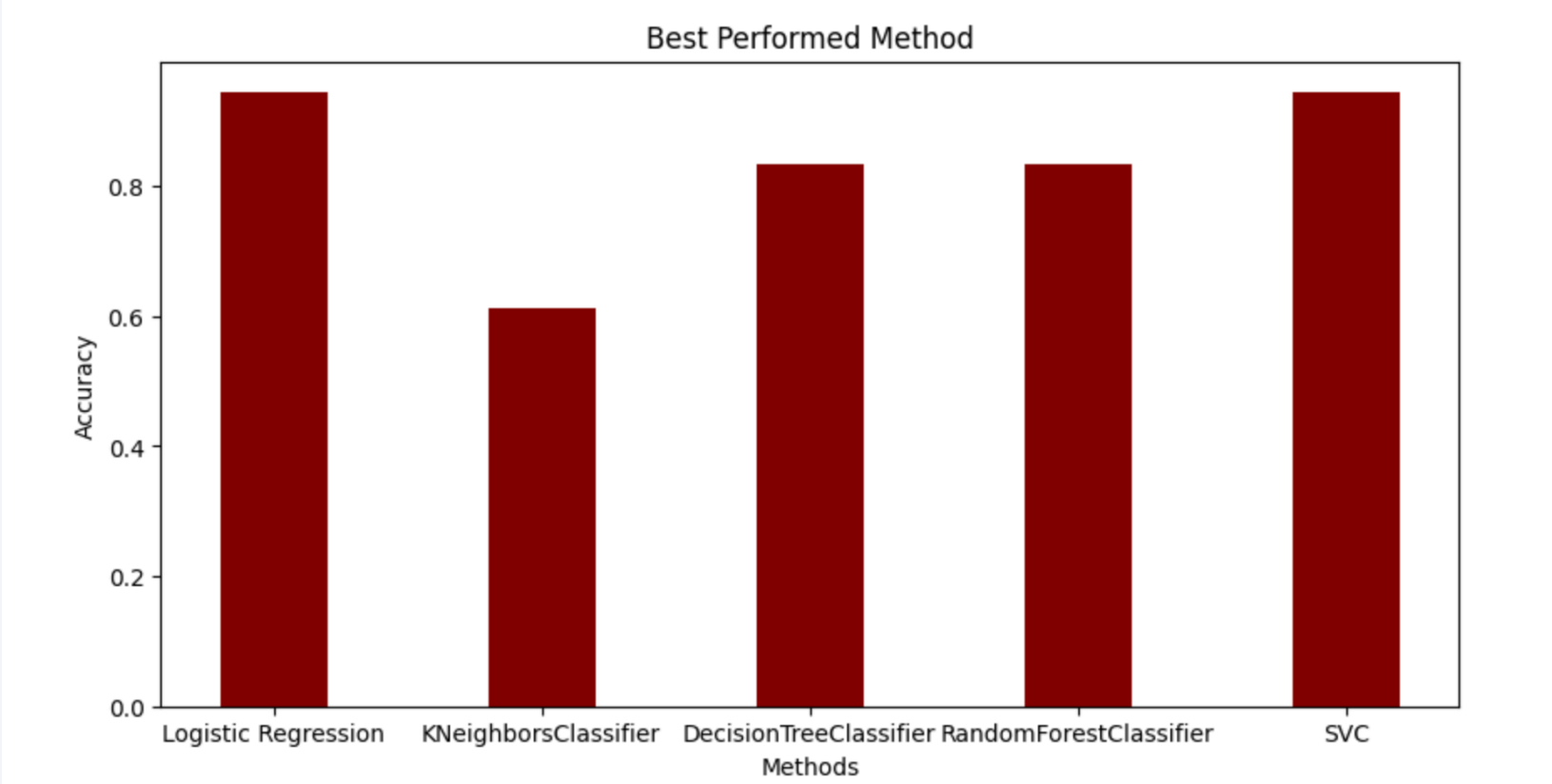
We can see the success rates for low weighted payloads is higher than the heavy weighted payloads

Section 5

Predictive Analysis (Classification)

Classification Accuracy

```
fig = plt.figure(figsize=(10, 5))
# Creating the bar plot
plt.bar(methods, accu, color='maroon', width=0.4)
plt.xlabel("Methods")
plt.ylabel("Accuracy") # Corrected from plt.vlabel to plt.ylabel
plt.title("Best Performed Method")
plt.show()
```

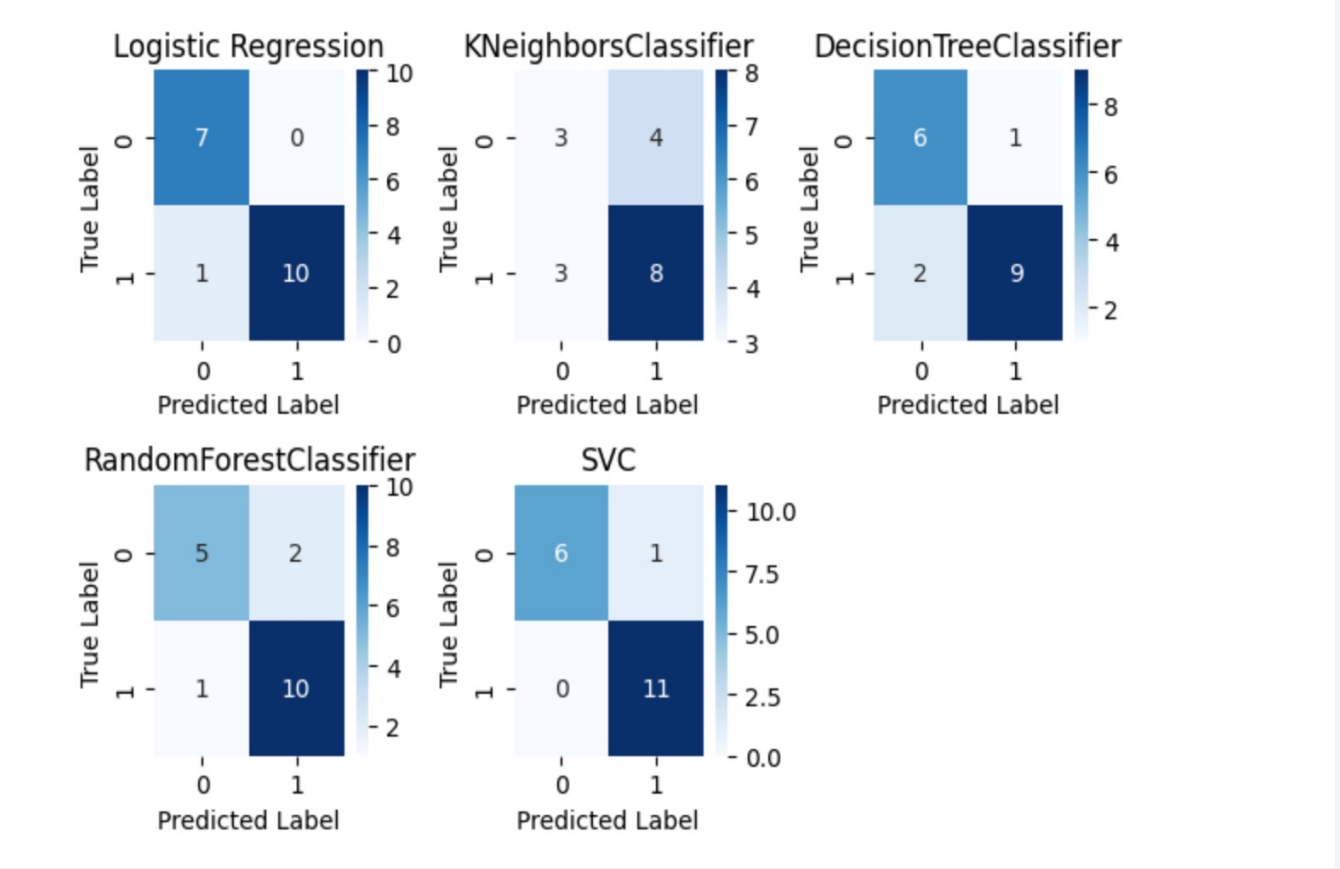


- The decision tree classifier is the model with the highest classification accuracy

Confusion Matrix

- The confusion matrix for the decision tree classifier shows that the classifier can distinguish between the different classes. The major problem is the false positives .i.e., unsuccessful landing marked as successful landing by the classifier.

```
import seaborn as sns
import matplotlib.pyplot as plt

models = ['Logistic Regression', 'KNeighborsClassifier', 'DecisionTreeClassifier', 'RandomForestClassifier', 'SVC']

for i, model in enumerate([logreg_cv, best_knn_model, tree_cv, rfc, svm_cv]):
plt.subplot(2, 3, i+1)
y_pred = model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['0', '1'], yticklabels=['0', '1'])
plt.title(models[i])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')

plt.tight_layout()
plt.show()
```



Conclusions

We can conclude that:

- The larger the flight amount at a launch site, the greater the success rate at a launch site.
- Launch success rate started to increase in 2013 till 2020.
- Orbits ES-L1, GEO, HEO, SSO, VLEO had the most success rate.
- KSC LC-39A had the most successful launches of any sites.
- The Decision tree classifier is the best machine learning algorithm for this task.

Thank you!