IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

TETTEH CHINELO N. C.

11TH FEBRUARY, 2025

Outline

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

Executive Summary

- Summary of methodologies

  - Data Collection Through API
  - Data Collection with Web Scraping
  - Data Wrangling
  - Exploratory Data Analysis with SQL
  - Exploratory Data Analysis with Data Visualization
  - Interactive Visual Analytics with Folium
  - Building an Interactive Dashboard with Ploty Dash
  - Machine Learning Prediction -

- Summary of all results

  - Exploratory Data Analysis Results
  - Interactive Analytics Result
  - Predictive Analysis Result

Introduction

- Project background and context

SpaceX advertises Falcon 9 rocket launches on its website, with a cost of $62 million. Other providers cost above $165 million each. Much of the savings is because SpaceX 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 SpaceX for a rocket launch.

The goal of the project is to create a machine learning pipeline to predict if the first stage will land successfully.

- Problems you want to find answers

 - What factors determine if the rocket will land successfully?

 - The interaction amongst various features that determine the success rate of a successful landing.

 - What operating conditions need to be in place to ensure a successful landing program?

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - SpaceX Open Source Rest API
  - Web Scraping from Wikipedia page 'List of Falcon 9 and Falcon Heavy Launches'
- Perform data wrangling
  - One-hot encoding was applied to categorical features for machine learning algorithms; dataset was inspected and missing values were removed.
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - Logistic Regression, Support Vector Machine, Decision Tree and K-Nearest Neighbors models were developed to determine the most effective classification method.

Data Collection

The datasets were collected using 2 methods:

1. Request to the SpaceX API

   - Gathered SpaceX's past launch data via their open-source API.

   - Retrieved and processed this data using GET request.

   - Ensured the data included only Falcon 9 launches.

   - Filled in missing payload weights from secret missions with average values.

2. Web Scraping

   - Requested past Falcon 9 and Falcon Heavy launch data from Wikipedia's relevant page.

   - Accessed the Falcon 9 Launch page via its direct Wikipedia link.

   - Extracted all the column names from the HTML table.

   - Parsed and transformed the table into a Pandas dataframe for analysis.

Data Collection – SpaceX API

Requested and parsed the data from SpaceX API using GET request

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-SkillsNetw
```

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]: launch_data = response.json()

         # Convert the data to a DataFrame
         data = pd.json_normalize(launch_data)
```

Using the dataframe `data` print the first 5 rows

```
In [13]: data.head(5)
Out[13]: static_fire_date_utc  static_fire_date_unix  tbd  net  window  rocket  success  details  crew
```

Filtered the dataframe to include only Falcon 9 launches needed for the project analysis.

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 [32]: data_falcon9 = data_launch[data_launch['BoosterVersion'] != 'Falcon 1']
```

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

```
In [33]: data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
         data_falcon9

/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/pandas/core/indexing.py:1773: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
```

Out[33]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs
4	1	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False
5	2	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False	False	False
6	3	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False	False	False

Stored the data and constructed the dataset into a new dictionary with relative columns.

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

```
In [28]: launch_dict = {'FlightNumber': list(data['flight_number']),
         'Date': list(data['date']),
         'BoosterVersion':BoosterVersion,
         'PayloadMass':PayloadMass,
         'Orbit':Orbit,
         'LaunchSite':LaunchSite,
         'Outcome':Outcome,
         'Flights':Flights,
         'GridFins':GridFins,
         'Reused':Reused,
         'Legs':Legs,
         'LandingPad':LandingPad,
         'Block':Block,
         'ReusedCount':ReusedCount,
         'Serial':Serial,
         'Longitude': Longitude,
         'Latitude': Latitude}
```

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

```
In [23]: data_launch = pd.DataFrame.from_dict(launch_dict)
```

Show the summary of the dataframe

```
In [29]: # Show the head of the dataframe
         data_launch.head()
Out[29]: FlightNumber  Date  BoosterVersion  PayloadMass  Orbit  LaunchSite  Outcome  Flights  GridFins  Reused  Legs
```

Cleaned the dataf and removed missing values in PayLoadMass

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 [36]: # Calculate the mean value of PayloadMass column
         payload_mass_mean = data_falcon9['PayloadMass'].mean()
         # Replace the np.nan values with its mean value
         data_falcon9['PayloadMass'].replace(np.nan, payload_mass_mean, inplace = True)
         data_falcon9.isnull().sum()

/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/pandas/core/generic.py:6619: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return self._update_inplace(result)

Out[36]: FlightNumber      0
         Date              0
         BoosterVersion    0
         PayloadMass       0
         Orbit             0
         LaunchSite        0
         Outcome           0
         Flights           0
         GridFins          0
         Reused            0
         Legs              0
         LandingPad       26
         Block             0
         ReusedCount       0
         Serial            0
         Longitude         0
```

GitHub URL Reference:

: https://github.com/chinelotetteh/SpaceX-Falcon-9-Landing-Machine-Learning-Prediction.git

Data Collection – Web Scraping

Requested data from Wikipedia using HTTP GET request and BeautifulSoup

Preview | Code | Blame | 922 lines (922 loc) · 47.1 KB | Code 55% faster with GitHub Copilot | Raw

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 [11]:
```python
# 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[11]: 200

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

In [12]:
```python
# 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 [13]:
```python
# Use soup.title attribute
print("Page Title:", soup.title.text)
```

```
Page Title: List of Falcon 9 and Falcon Heavy launches - Wikipedia
```

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 [21]:
```python
# Use the find_all function in the BeautifulSoup object, with element type `table`
html_tables = soup.find_all("table")
```

Cleaned the column data, created an empty dictionary with extracted columns and appended the column names.

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 [25]:
```python
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']=[]
```

Next, we just need to fill up the `launch_dict` with launch records extracted from table rows.

Usually, HTML tables in Wiki pages are likely to contain unexpected annotations and other types of noises, such as reference links `B0004.1[8]`, missing values `N/A [e]`, inconsistent formatting, etc.

To simplify the parsing process, we have provided an incomplete code snippet below to help you to fill up the `launch_dict`. Please complete the following code snippet with TODOs or you can choose to write your own logic to parse all launch tables:

In [27]:
```python
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders collapsible")):
    # get table row
```

Extracted all column/variable names from the HTML table header.

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

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

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

In [21]:
```python
# Use the find_all function in the BeautifulSoup object, with element type `table`
html_tables = soup.find_all("table")
```

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

In [22]:
```python
# Let's print the third table and check its content
first_launch_table = html_tables[2]
print(first_launch_table)
```

```
<table class="wikitable plainrowheaders collapsible" style="width: 100%;">
<tbody><tr>
<th scope="col">Flight No.
</th>
<th scope="col">Date and<br/>time (<a href="/wiki/Coordinated_Universal_Time" title="Coordinated Universal Time">UTC</a>)
</th>
<th scope="col"><a href="/wiki/List_of_Falcon_9_first-stage_boosters" title="List of Falcon 9 first-stage boosters">Versi
on,<br/>Booster</a> <sup class="reference" id="cite_ref-booster_11-0"><a href="#cite_note-booster-11"><span class="cite-b
racket">[</span>b<span class="cite-bracket">]</span></a></sup>
</th>
<th scope="col">Launch site
</th>
<th scope="col">Payload<sup class="reference" id="cite_ref-Dragon_12-0"><a href="#cite_note-Dragon-12"><span class="cite-
bracket">[</span>c<span class="cite-bracket">]</span></a></sup>
</th>
<th scope="col">Payload mass
</th>
<th scope="col">Orbit
</th>
<th scope="col">Customer
</th>
```

Created a dataframe by parsing the launch HTML tables

Preview | Code | Blame | 922 lines (922 loc) · 47.1 KB | Code 55% faster with GitHub Copilot | Raw

```python
        launch_dict['Payload mass'].append(payload_mass)

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

        # Customer
        # TODO: Append the customer into launch_dict with key `Customer`
        try:
            customer = row[6].a.string
        except:
            customer = 'Various'
        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]
        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])
        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 [28]:
```python
df=pd.DataFrame(launch_dict)
```

We can now export it to a **CSV** for the next section, but to make the answers consistent and in case you have difficulties finishing this lab.

Following labs will be using a provided dataset to make each lab independent.

```python
df.to_csv('spacex_web_scraped.csv', index=False)
```

GitHub URL Reference:

: https://github.com/chinelotetteh/SpaceX-Falcon-9-Landing-Machine-Learning-Prediction.git

Data Wrangling

Perform exploratory data analysis to find patterns in the dataset and determine what would be the label for train supervised models.

1. Identify missing values
2. Identify numerical and categorical columns
3. Calculate the number of launches on each site
4. Calculate the number of occurrence of each orbit
5. Calculate the number of occurrence of mission outcome per orbit type
6. Create a landing outcome label from Outcome column

The variable represents the classification outcome of ecah launch. Zero means the first stage did not land successfully. One means the first stage landed successfully.

GitHub URL Reference: : https://github.com/chinelotetteh/SpaceX-Falcon-9-Landing-Machine-Learning-Prediction.git

EDA with Data Visualization

Summary of charts that were plotted:

- Catplot: To visualize the relationship between Flight Number and Payload.
- Catplot: To visualize the relationship between Flight Number and Launch Site.
- Catplot: To visualize the relationship between Payload and Launch Site.
- Bar Chart: To visualize the relationship between success rate of each Orbit type.
- Catplot: To visualize the relationship between Flight Number and Orbit type.
- Catplot: To visualize the relationship between Payload and Orbit type.
- Line Chart: To visualize the launch success yearly trend.

GitHub URL Reference: : https://github.com/chinelotetteh/SpaceX-Falcon-9-Landing-Machine-Learning-Prediction.git

EDA with SQL

SQL queries you performed:

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

```
query = "SELECT DISTINCT Launch_Site FROM SPACEXTBL;"
unique_launch_sites = pd.read_sql(query, conn)
unique_launch_sites
```

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

```
query = "SELECT * FROM SPACEXTBL WHERE Launch_Site LIKE 'CCA%' LIMIT 5;"
filtered_records = pd.read_sql(query, conn)
filtered_records
```

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

```
query = "SELECT SUM(PAYLOAD_MASS__KG_) AS Total_Payload_Mass FROM SPACEXTBL
WHERE Customer LIKE 'NASA (CRS)';"
```

EDA with SQL

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

```
query = "SELECT AVG(PAYLOAD_MASS__KG_) AS Average_Payload_Mass FROM SPACEXTBL WHERE Booster_Version = 'F9 v1.1';"
avg_payload = pd.read_sql(query, conn)
avg_payload
```

- List the date when the first successful landing outcome in ground pad was achieved.

```
query = "SELECT MIN(Date) AS First_Successful_Landing FROM SPACEXTBL WHERE Landing_Outcome = 'Success (ground pad)';"
successful_first_landing = pd.read_sql(query, conn)
successful_first_landing
```

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

```
query = """
SELECT DISTINCT Booster_Version
FROM SPACEXTBL
WHERE Landing_Outcome = 'Success (drone ship)'
AND PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000;
b    oosters = pd.read_sql(query, conn)
```

EDA with SQL

- List the total number of successful and failure mission outcomes.
- List the names of the booster_versions which have carried the maximum payload mass. Use a subquery.
- 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.
- 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.

Interactive Folium Map

- Map objects such as markers, circles and lines were added to the Folium map to mark the success or failure of launches for each site on the map.
- Feature launch outcomes (failure or success) were assigned to class 0 and 1, that is, 0 for failure and 1 for success.
- Using the colour-labelled marker clusters, launch sites with relatively high success rates were identified.
- The distance between a launch site to the coastline, rail line and to the perimeter road were calculated.

GitHub URL Reference: : https://github.com/chinelotetteh/SpaceX-Falcon-9-Landing-Machine-Learning-Prediction.git

Build a Dashboard with Plotly Dash

- The total launches by a specific site was plotted with a pie chart.
- A scatter graph was plotted to show the relationship with Outcome and Payload Mass (kg) for the different booster versions.

GitHub URL Reference:
https://github.com/chinelotetteh/SpaceX-Interactive-Dashboard-With-Plotly.git

Predictive Analysis (Classification)

- The dataset was split into training and testing sets.
- The following machine learning models were trained on the training data set:
  - Logistic Regression
  - Support Vector Machine
  - Decision Tree
  - K-Nearest Neighbours
- Hyperparameters were evaluated using GridSearchCV(). The best was selected using the best_params method.
- Using the best hyperparameters, each of the four models were scored on accuracy by using the testing dataset.

GitHub URL Reference: : https://github.com/chinelotetteh/SpaceX-Machine-Learning-Prediction.git

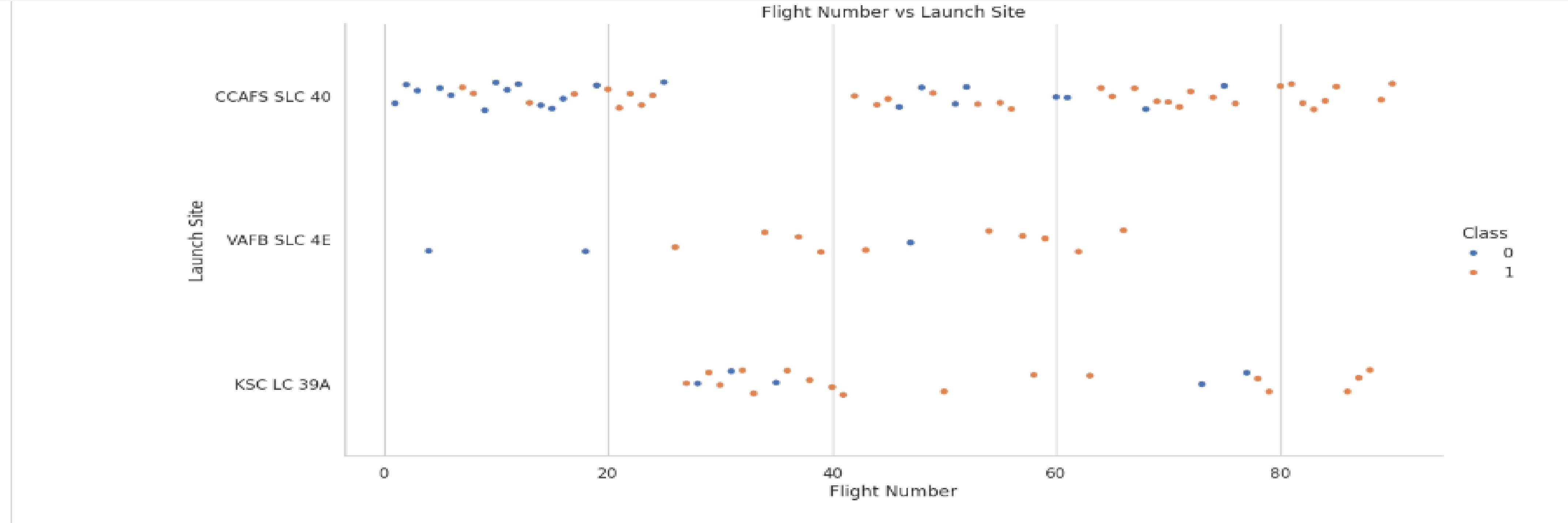
Results

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

Section 2

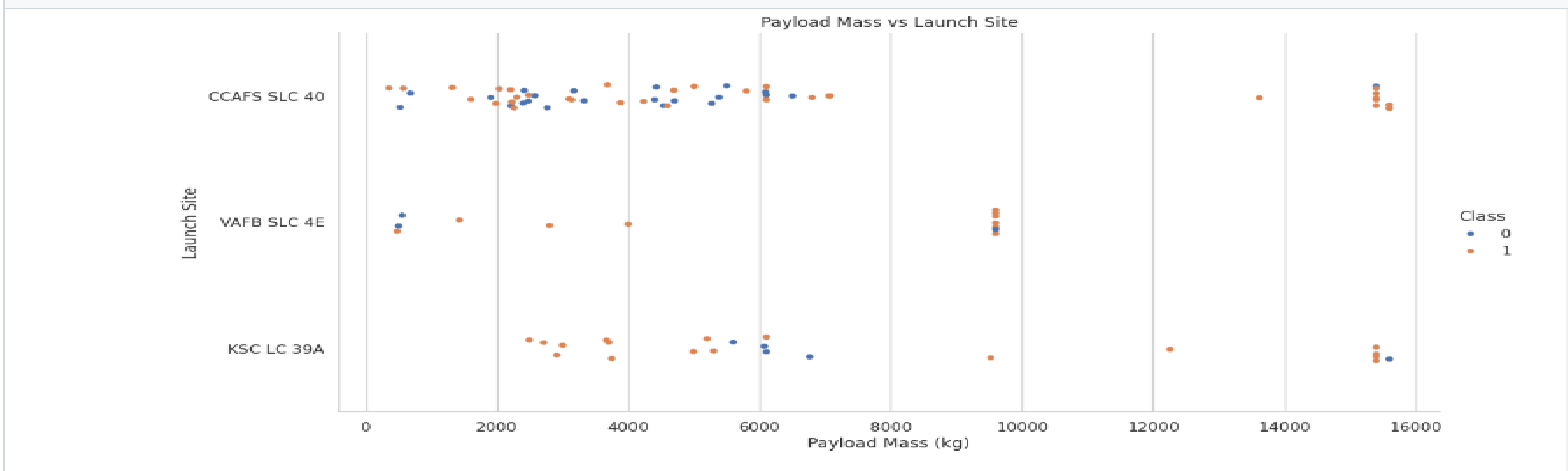
Insights drawn from EDA

Flight Number vs. Launch Site



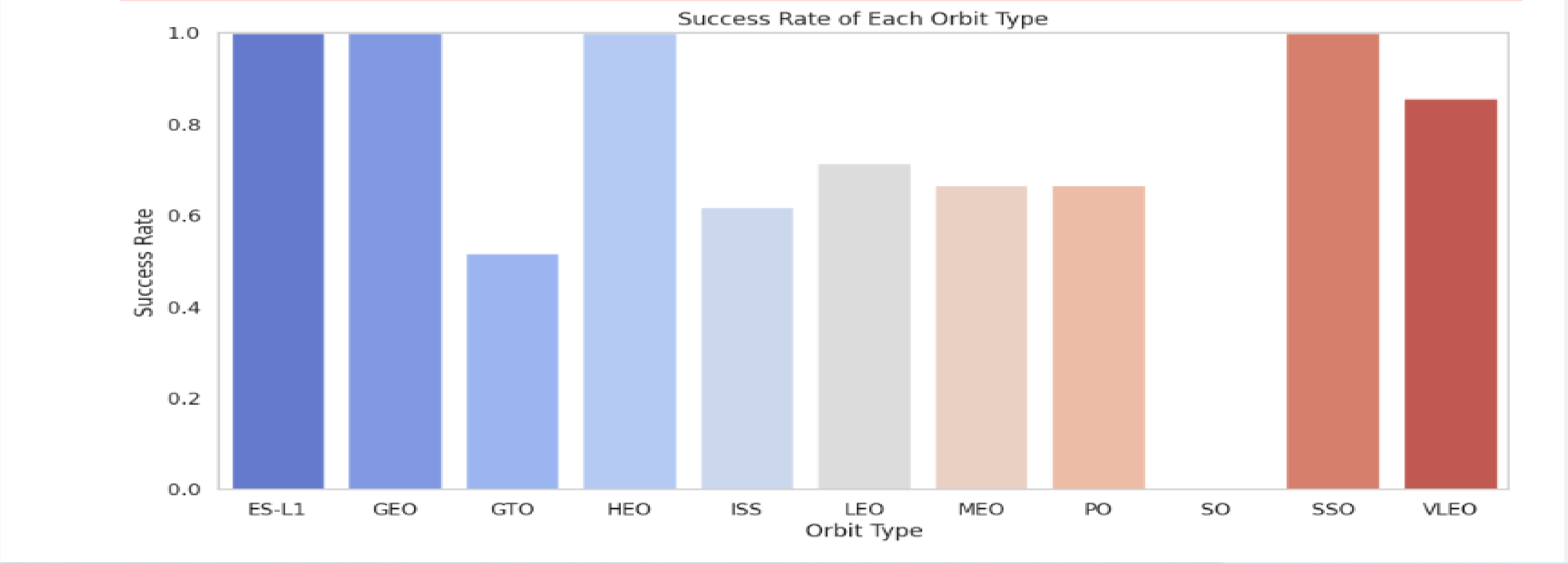
- **Total number of launches from launch site CCAFS SLC 40 are significantly higher than other launch sites.**

Payload Mass vs. Launch Site



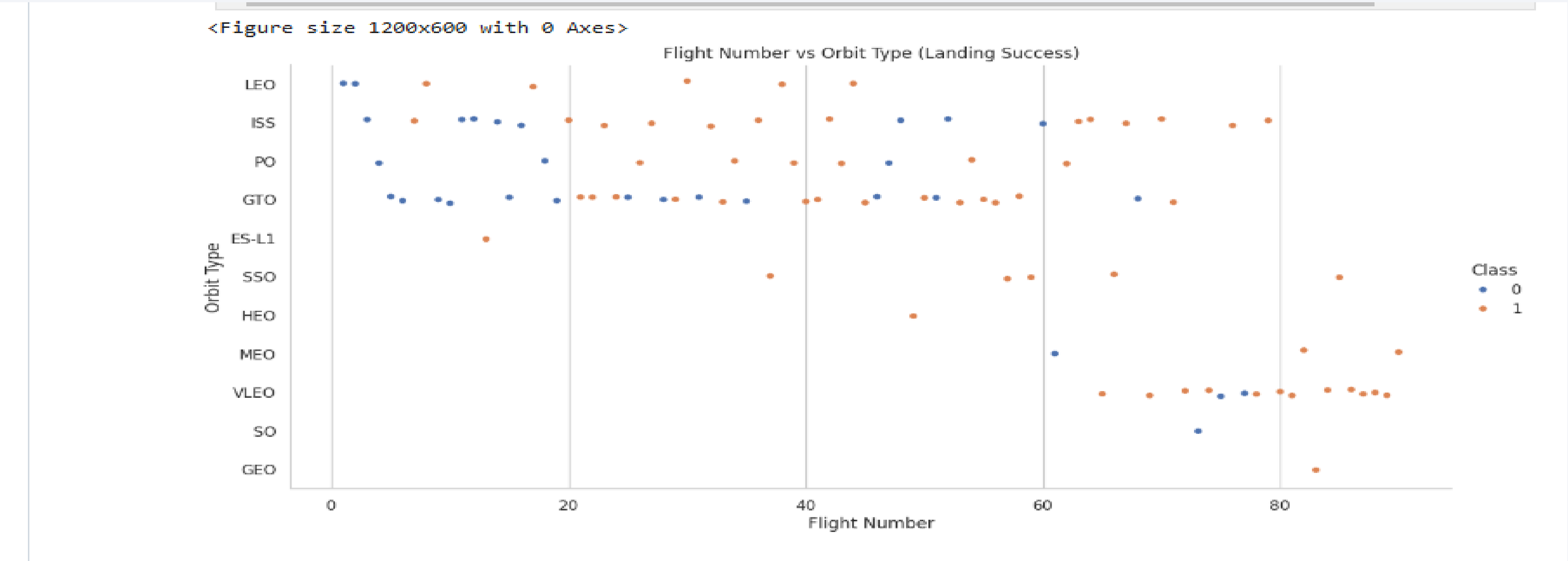
- **Payloads with lower masses have more launches compared to those with higher mass across all three launch sites.**

Success Rate vs. Orbit Type



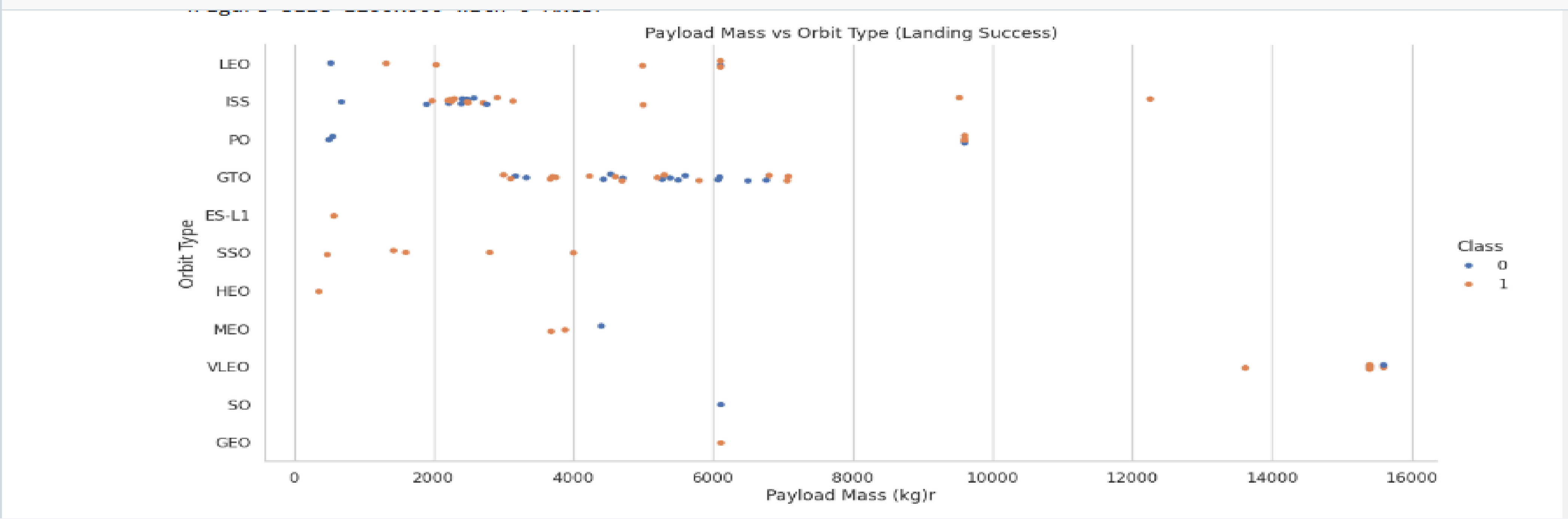
- **Certain orbits such as GEO and SSO tend to have high success rates. This suggests that SpaceX has optimized launches for these orbits, possibly due to consistent mission profiles.**
- **LEO and GTO show mid-range success rates.**
- **HEO or experimental ones (e.g., SSO in early flights) tend to have lower success rates.**

Flight Number vs. Orbit Type



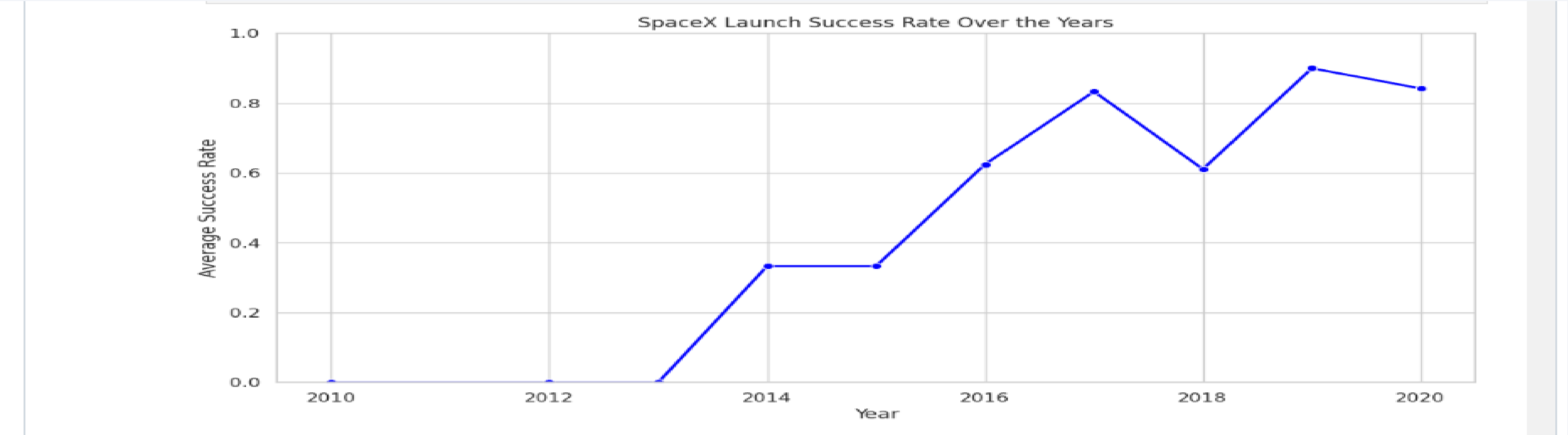
- **You can observe that in the LEO orbit, success seems to be related to the number of flights. Conversely, in the GTO orbit, there appears to be no relationship between flight number and orbit.**

Payload vs. Orbit Type



- **With heavy payloads the successful landing or positive landing rate are more for Polar, LEO and ISS.**
- **However, for GTO, it's difficult to distinguish between successful and unsuccessful landings as both outcomes are present.**

Launch Success Yearly Trend



- **The success rate since 2013 kept increasing till 2020 possibly due to technology advancement and experience.**

All Launch Site Names

Task 1

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

In [27]:

```
query = "SELECT DISTINCT Launch_Site FROM SPACEXTBL;"
unique_launch_sites = pd.read_sql(query, conn)

unique_launch_sites
```

Out[27]:

	Launch_Site
0	CCAFS LC-40
1	VAFB SLC-4E
2	KSC LC-39A
3	CCAFS SLC-40

- **The keyword DISTINCT was used to show only unique launch sites from SpaceX data.**

Launch Site Names Begin with 'CCA'

Task 2

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

In [28]:

```
# Query execution
query = "SELECT * FROM SPACEXTBL WHERE Launch_Site LIKE 'CCA%' LIMIT 5;"
filtered_records = pd.read_sql(query, conn)

filtered_records
```

Out[28]:

	Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcom
0	2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Succes
1	2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of...	0	LEO (ISS)	NASA (COTS) NRO	Succes
2	2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Succes
3	2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Succes
4	2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Succes

- **The query was used to display 5 records where launch sites begin with 'CCA'**

Total Payload Mass

Task 3

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

In [29]:
```
query = "SELECT SUM(PAYLOAD_MASS__KG_) AS Total_Payload_Mass FROM SPACEXTBL WHERE Customer LIKE 'NASA (CRS)
total_payload = pd.read_sql(query, conn)

total_payload
```

Out[29]:

	Total_Payload_Mass
0	45596

- **The total payload carried by boosters from NASA was calculated as 45596**

Average Payload Mass by F9 v1.1

Task 4

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

In [31]:
```
query = "SELECT AVG(PAYLOAD_MASS__KG_) AS Average_Payload_Mass FROM SPACEXTBL WHERE Booster_Version = 'F9 v
avg_payload = pd.read_sql(query, conn)

avg_payload
```

Out[31]:

	Average_Payload_Mass
0	2928.4

- **The average payload mass carried by booster version F9 v1.1 was calculated as 2928.4**

First Successful Ground Landing Date

Task 5

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

Hint:Use min function

In [32]:

```
query = "SELECT MIN(Date) AS First_Successful_Landing FROM SPACEXTBL WHERE Landing_Outcome = 'Success (grou
successful_first_landing = pd.read_sql(query, conn)

successful_first_landing
```

Out[32]:

	First_Successful_Landing
0	2015-12-22

- **The date when the first successful landing outcome in ground pad was achieved was 22nd December, 2015**

Successful Drone Ship Landing with Payload between 4000 and 6000

Task 6

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

In [33]:

```
query = """
SELECT DISTINCT Booster_Version
FROM SPACEXTBL
WHERE Landing_Outcome = 'Success (drone ship)'
AND PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000;
"""
boosters = pd.read_sql(query, conn)

boosters
```

Out[33]:

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

- The WHERE clause was used to filter the names of boosters which have successfully landed on drone ship.
- The AND condition was applied to determine successful landing with payload mass greater than 4000 but less than 6000.

Total Number of Successful and Failure Mission Outcomes

Task 7

List the total number of successful and failure mission outcomes

In [34]:
```
query_1 = '''
          SELECT COUNT(Mission_Outcome) AS Total_Successful_Outcome
          FROM SPACEXTBL
          WHERE Mission_Outcome LIKE 'Success%'
          '''

query_2 = '''
          SELECT COUNT(Mission_Outcome ) AS Total_Failure_Outcome
          FROM SPACEXTBL
          WHERE Mission_Outcome  LIKE 'Failure%'
          '''

successful_mission_outcomes = pd.read_sql(query_1, conn)
successful_mission_outcomes

failed_mission_outcomes = pd.read_sql(query_2, conn)
failed_mission_outcomes

print('The total number of successful mission outcomes is:',successful_mission_outcomes)
print('The total number of failed mission outcomes is:',failed_mission_outcomes)
```

```
The total number of successful mission outcomes is:    Total_Successful_Outcome
0                        100
The total number of failed mission outcomes is:    Total_Failure_Outcome
0                       1
```

- The wildcard like '%' was used to filter for WHERE mission outcomes was successful and a failure.

Boosters Carried Maximum Payload

Task 8

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

In [35]:
```
query = """
SELECT DISTINCT Booster_Version, PAYLOAD_MASS__KG_
FROM SPACEXTBL
WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTBL);
"""
max_boosters = pd.read_sql(query, conn)

max_boosters
```

Out[35]:

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

- **The names of the booster which have carried the maximum payload mass was retrieved using a subquery in the WHERE clause and the MAX() function.**

2015 Launch Records

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 [19]:

```
query = """
SELECT
    SUBSTR(Date, 6, 2) AS Month,
    Booster_Version,
    Launch_Site,
    Landing_Outcome
FROM SPACEXTBL
WHERE Landing_Outcome LIKE 'Failure (drone ship)'
AND SUBSTR(Date, 1, 4) = '2015';
"""
failure_records = pd.read_sql(query, conn)

failure_records
```

Out[19]:

	Month	Booster_Version	Launch_Site	Landing_Outcome
0	01	F9 v1.1 B1012	CCAFS LC-40	Failure (drone ship)
1	04	F9 v1.1 B1015	CCAFS LC-40	Failure (drone ship)

- **A combination of WHERE , LIKE, AND and BETWEEN conditions were used to retrieve the records which displayed the month names, failure landing outcomes in drone ship, booster versions, launch site for the months in year 2015.**

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

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 [20]:

```
query = """
SELECT Landing_Outcome, COUNT(*) AS Outcome_Count
FROM SPACEXTBL
WHERE Date BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY Landing_Outcome
ORDER BY Outcome_Count DESC;
"""
ranking = pd.read_sql(query, conn)

ranking
```

Out[20]:

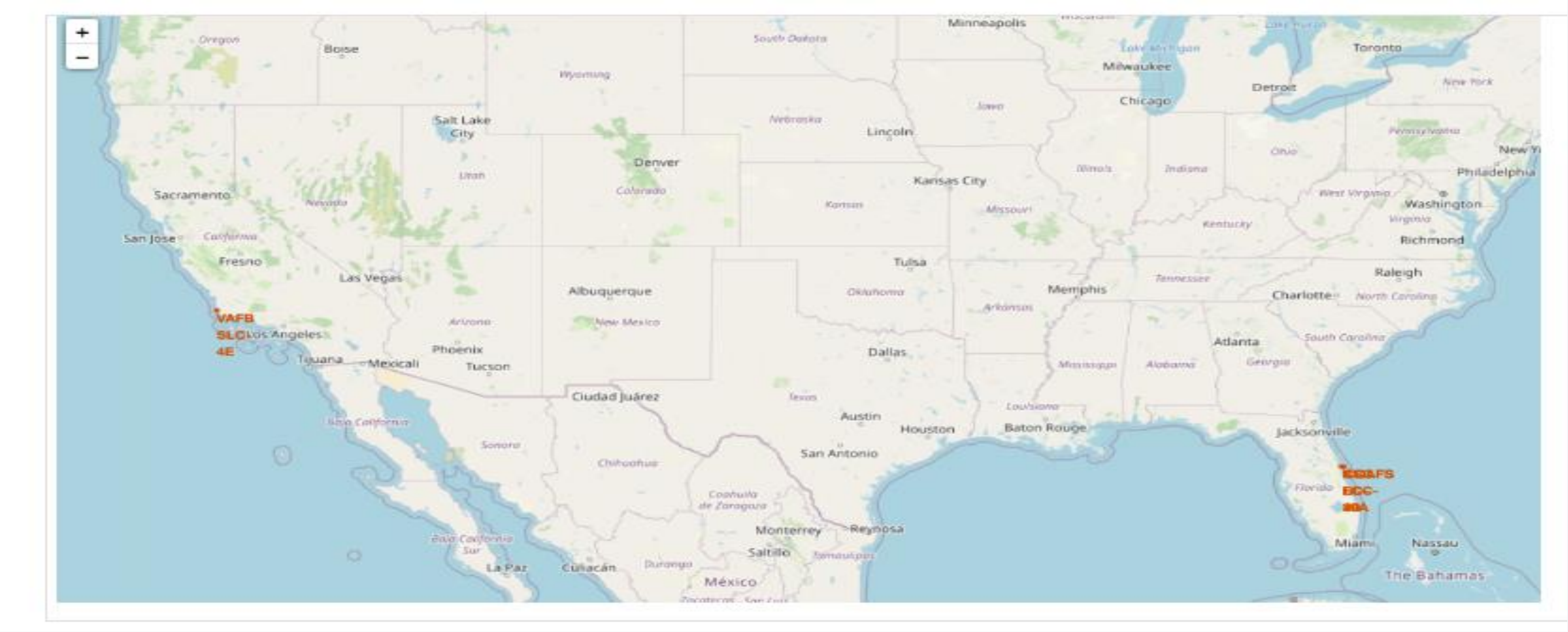
	Landing_Outcome	Outcome_Count
0	No attempt	10
1	Success (drone ship)	5
2	Failure (drone ship)	5
3	Success (ground pad)	3
4	Controlled (ocean)	3
5	Uncontrolled (ocean)	2
6	Failure (parachute)	2
7	Precluded (drone ship)	1

- COUNT, WHERE, BETWEEN, GROUP BY and ORDER BY were all applied to rank landing outcomes between 2010-06-04 and 2017-03-20.

Section 3

Launch Sites Proximities Analysis

Falcon 9 Launch site Locations

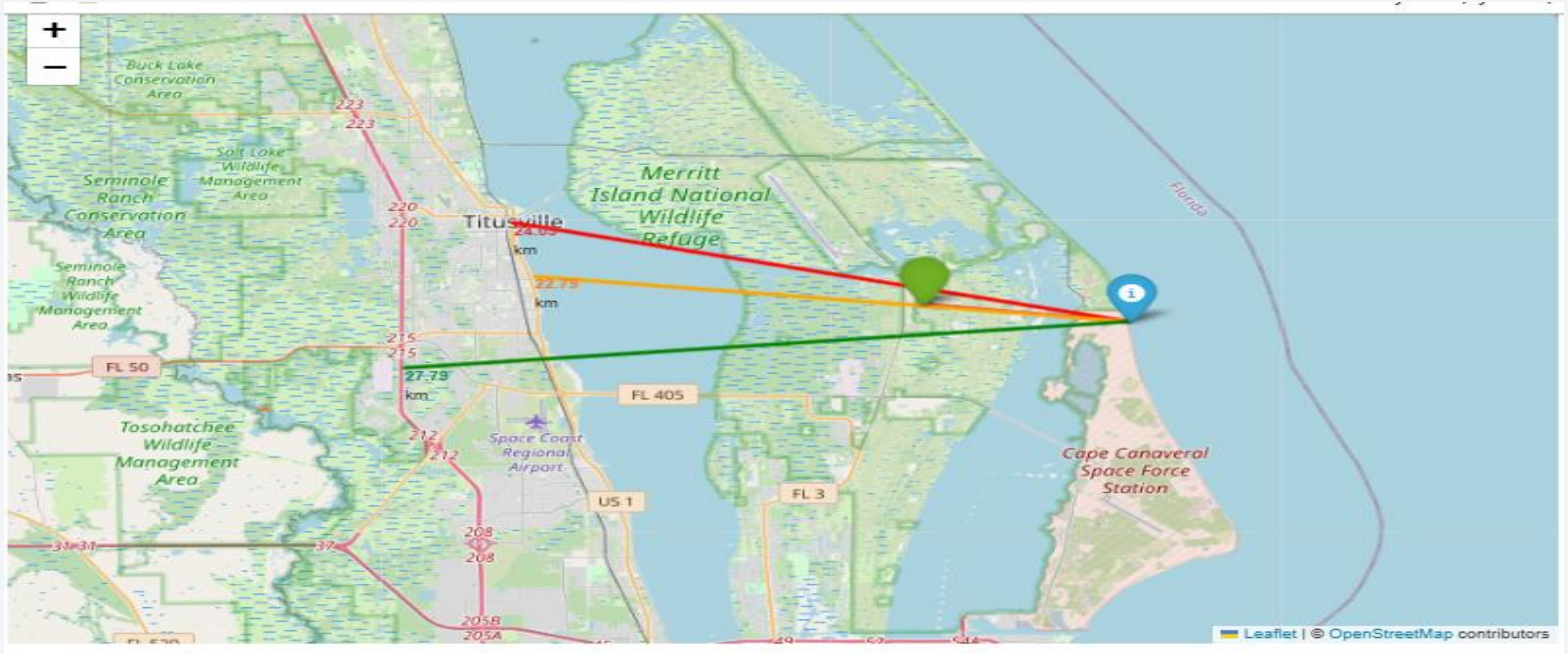


From the map, the launch sites are located in Florida and California.

Map Markers of Successful/Failed Launches

The green markers show successful launches while the red markers show unsuccessful launches.

Distance From Launch Site To Proximities

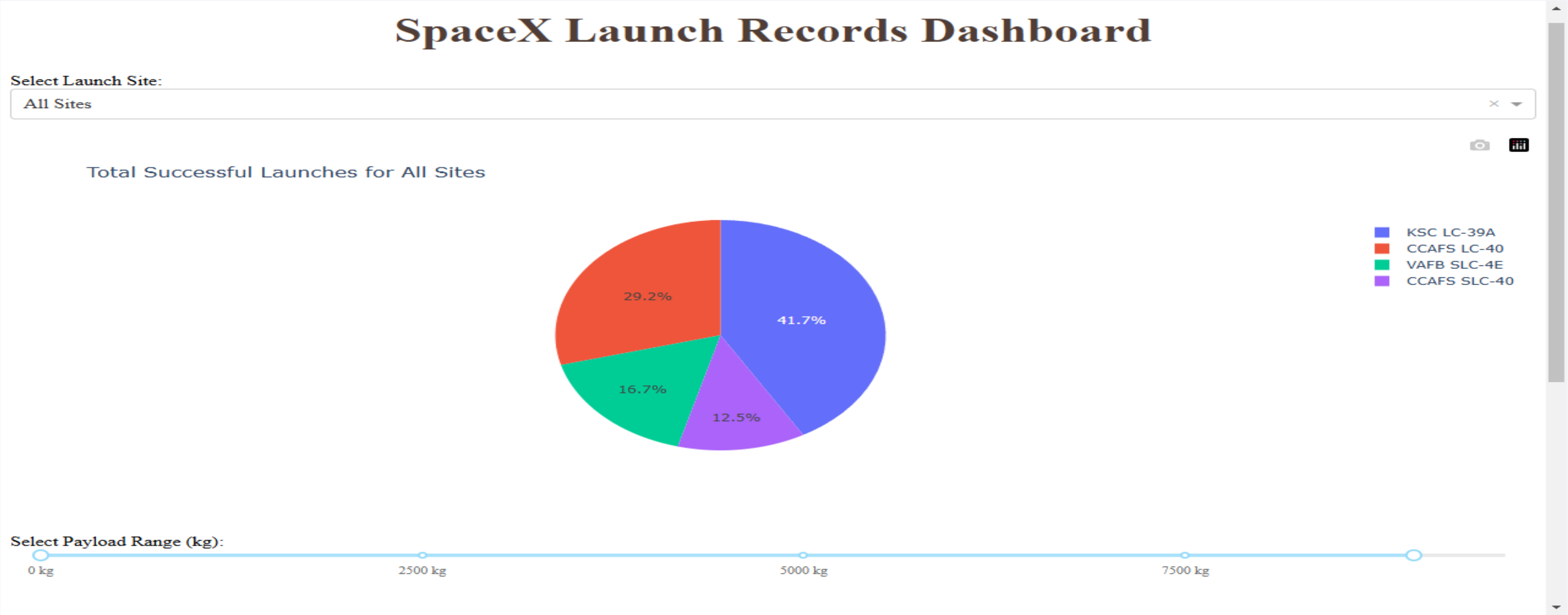


The distance from one of the launch sites to Titusville is 24.05km

Section 4

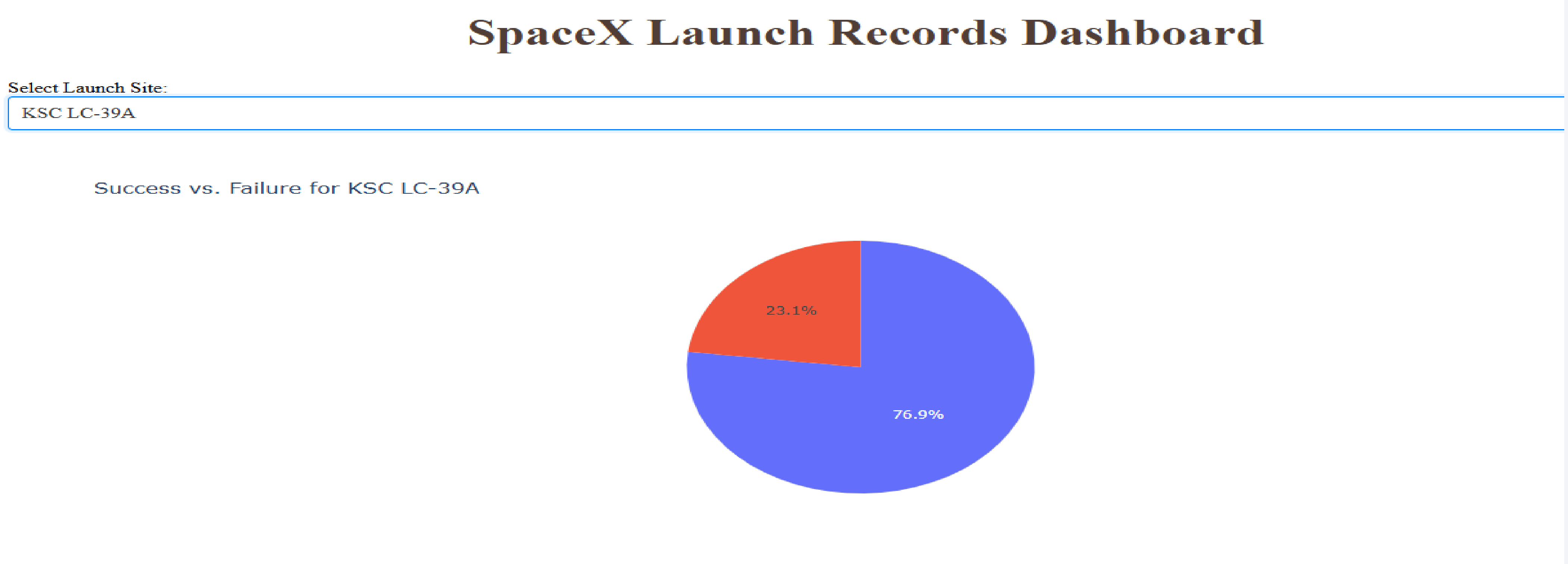
Build a Dashboard with Plotly Dash

Total Success Launches For All Sites



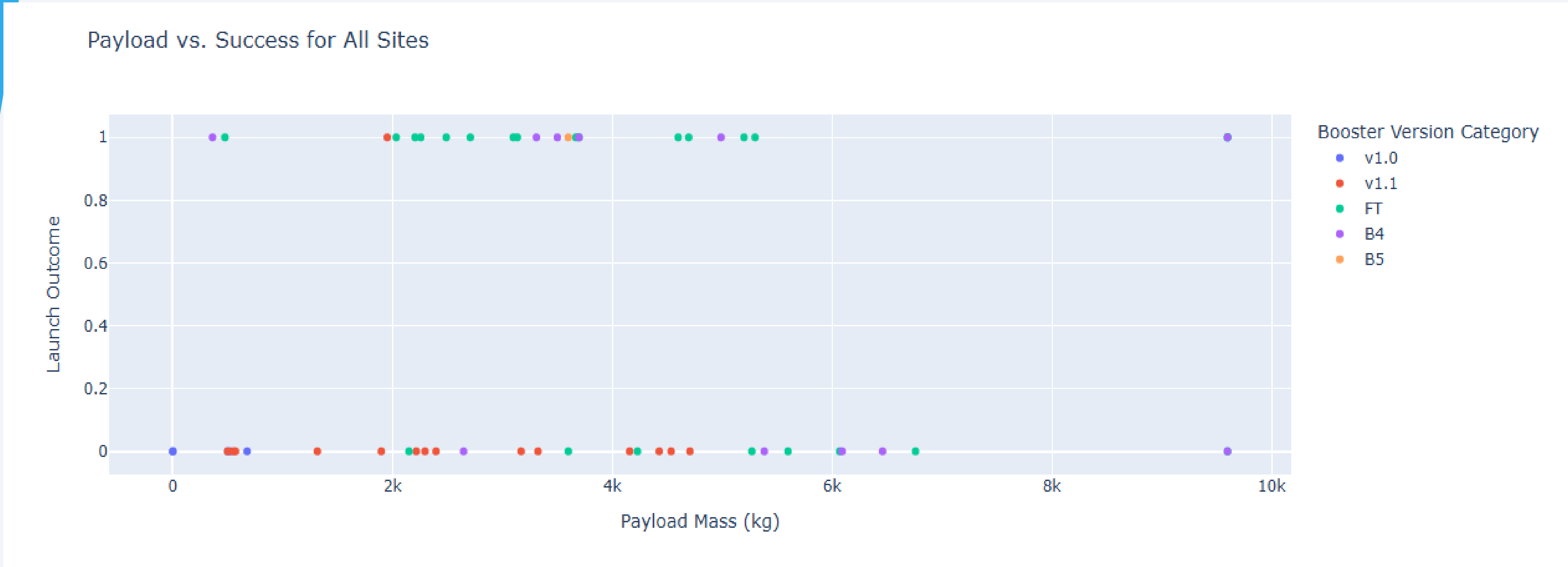
- **KSC LC-39A had the highest percentage of success launches (41.7%) whereas CCAFS SLC-40 has the lowest percentage of success launches (12.5%).**

Launch Site With Highest Launch Success Ratio



- **KSC LC-39A achieved 76.9% success rate and 23.1% failure rate.**

Payload VS Launch Outcome Scatter Plot



- **The success rates for the low-weighted payloads is higher than that of the heavy-weighted payloads.**

Section 5

Predictive Analysis (Classification)

Classification Accuracy

TASK 12

Find the method performs best:

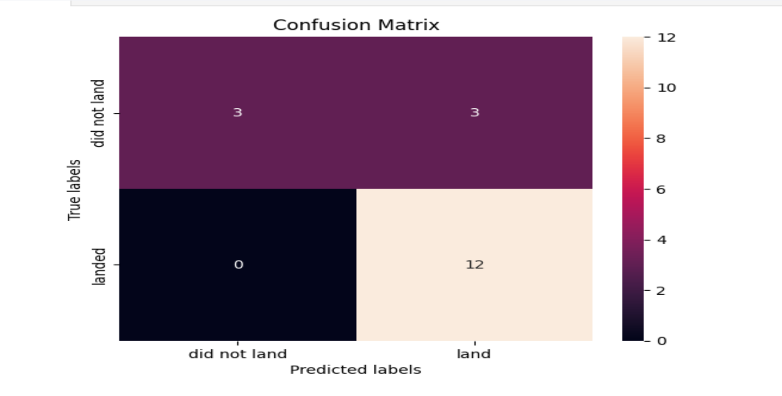
In [32]:
```
# Store model accuracies in a dictionary
model_scores = {
    "Logistic Regression": logreg_cv.score(X_test, Y_test),
    "Support Vector Machine": svm_cv.score(X_test, Y_test),
    "Decision Tree": tree_cv.score(X_test, Y_test),
    "K-Nearest Neighbors": knn_cv.score(X_test, Y_test)
}

# Find the best model
best_model = max(model_scores, key=model_scores.get)

# Print results
print("Model Performance:")
for model, score in model_scores.items():
    print(f"{model}: {score:.4f}")

print(f"\nThe best-performing model is: **{best_model}** with an accuracy of {model_scores[best_model]:.4f}")
```

```
Model Performance:
Logistic Regression: 0.8333
Support Vector Machine: 0.8333
Decision Tree: 0.8333
K-Nearest Neighbors: 0.8333
```

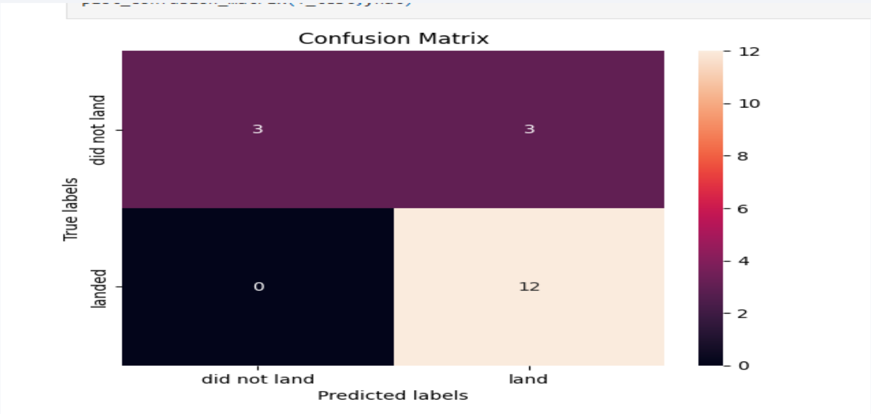
- **Logistic Regression, Support Vector Machine, Decision Tree and K-Nearest Neighbors all had an accuracy of 83.3%.**

Confusion Matrix

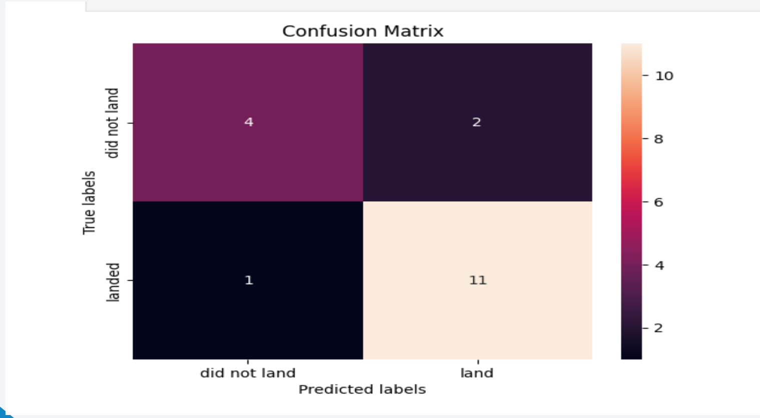
Logistic Regression Confusion Matrix



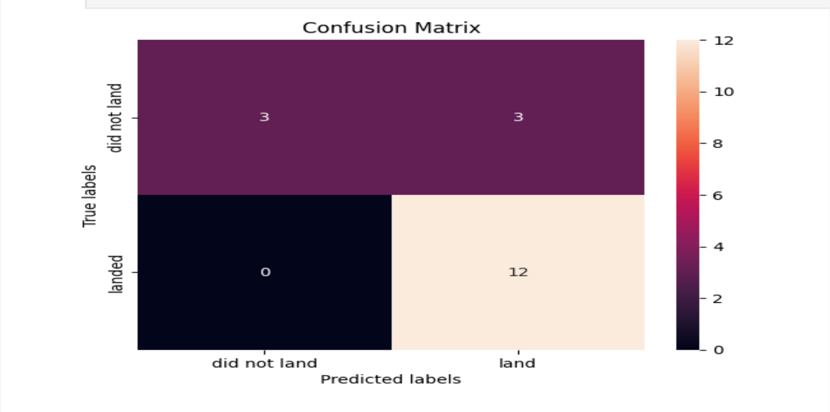
Support Vector Movement Confusion Matrix



Decision Tree Confusion Matrix



K-Nearest Neighbour Confusion Matrix

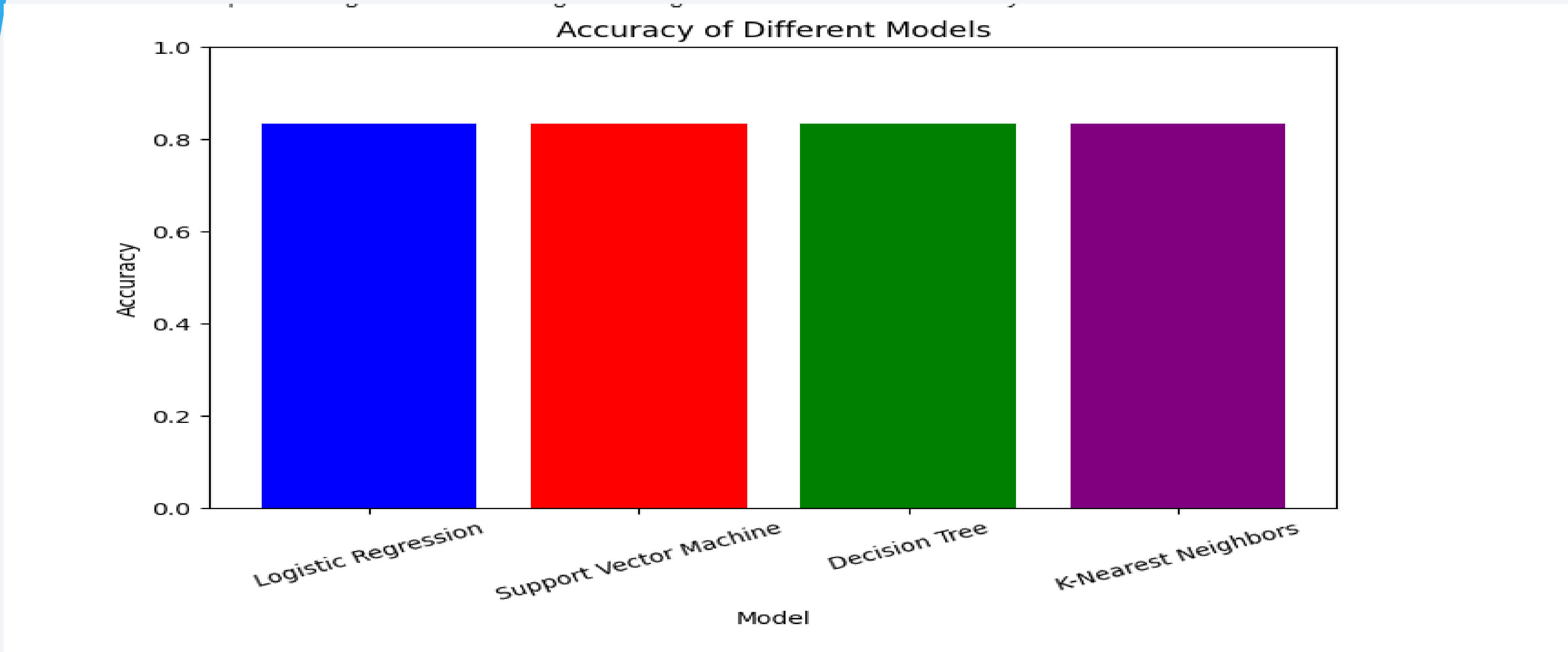


- All had more true positive predictions.

Conclusions

- The likelihood of a SpaceX launch to succeed increases with years. This suggests a trend towards flawless launches over time.
- Launch Complex 39A at Kennedy Space Center has the highest number of successful launches compared to other launch sites.
- Light-weighted payloads have a better performance compared to heavy-weighted payloads.
- GEO, HEO, SSO, ES L1 orbit types exhibit the highest rates of successful launches.
- Logistic regression, support vector movement, decision tree and k-nearest neighbour models all performed well in forecasting outcomes.

Appendix



Visualization of the Accuracy of all the Models Applied

Thank you!