IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Luis Daniel Guevara Urbina
02-07-2023>

Outline

Executive Summary

- Summary of methodologies
  1. Data Collection via API
  2. Data Acquisition using Web Scraping
  3. Data Manipulation (Wrangling)
  4. Exploratory Data Analysis using SQL
  5. Exploratory Data Analysis through Data Visualization
  6. Interactive Visual Analytics using Folium
  7. Machine Learning Prediction
- Summary of all results
  1. Exploratory Data Analysis result
  2. Interactive analytics in screenshots
  3. Predictive Analytics result

Introduction

- Project background and context

  SpaceX promotes Falcon 9 rocket launches on its website at a price of 62 million dollars, while other providers charge upwards of 165 million dollars per launch. The significant cost savings for SpaceX is primarily due to their ability to reuse the first stage of the rocket. Consequently, if we can accurately predict whether the first stage will successfully land, we can determine the overall cost of a launch. This information can be valuable for other companies looking to compete with SpaceX in bidding for rocket launches. The objective of this project is to develop a machine learning pipeline that can predict the likelihood of a successful landing for the first stage.

- Problems you want to find answers

  1. What factors determine if the rocket will land successfully?
  2. The interaction amongst various features that determine the success rate of a successful landing.
  3. What operating conditions needs to be in place to ensure a successful landing program.

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology
    - Data was gathered through the use of SpaceX's API and by scraping information from Wikipedia.
- Perform data wrangling
    - Categorical features were encoded using one-hot encoding.
- Perform exploratory data analysis (EDA) using visualization and SQL
    - Generate an executive summary by performing exploratory data analysis (EDA) using visualization and SQL in Python with sqlite3.
- Perform interactive visual analytics using Folium and Plotly Dash
    - Perform interactive visual analytics using Folium and Plotly Dash to create dynamic and engaging data visualizations.
- Perform predictive analysis using classification models
    - How to build, tune, evaluate classification models

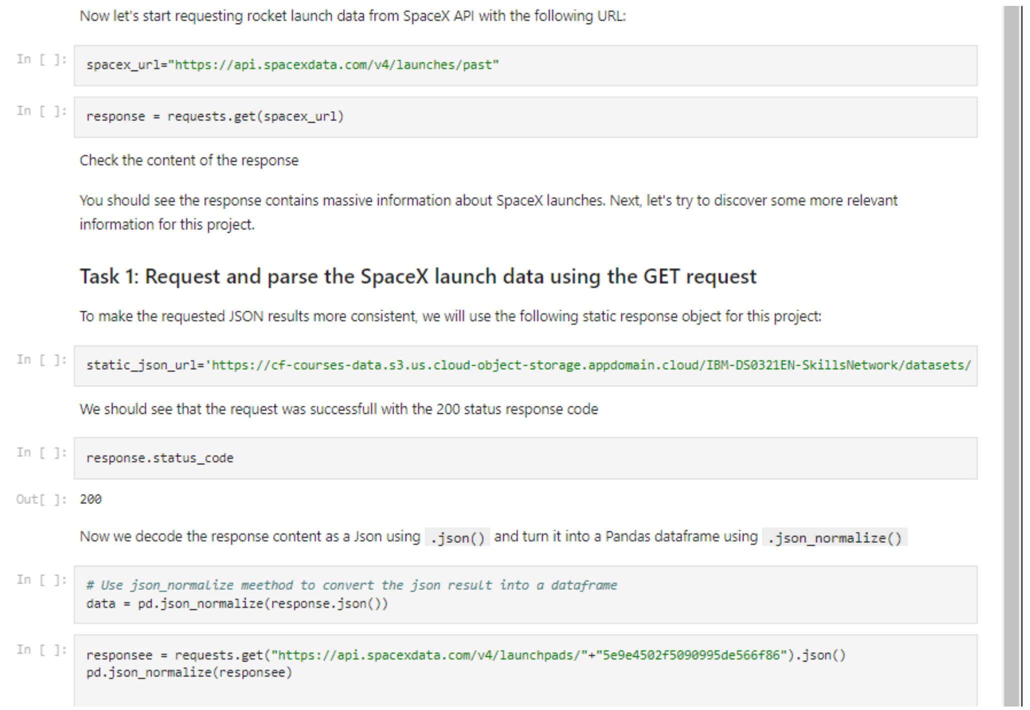
Data Collection

- The data was collected using multiple techniques:
    1. Data collection involved making GET requests to the SpaceX API.
    2. The response content was decoded as JSON using the .json() function and transformed into a pandas dataframe using .json_normalize().
    3. Data cleaning was performed, including checking for missing values and filling them in when needed.
    4. Furthermore, web scraping was conducted on Wikipedia to retrieve Falcon 9 launch records using BeautifulSoup.
    5. The goal was to extract the launch records from an HTML table, parse the table, and convert it into a pandas dataframe for further analysis.

Data Collection – SpaceX API

- We utilized the GET request to gather data from the SpaceX API, performed data cleaning and conducted basic data wrangling and formatting.

- The URL to access the notebook is
- https://github.com/FloxTBoTyy/Mi-primer-repository/blob/main/jupyter_labs_spacex_data_collection_api.ipynb



Now let's start requesting rocket launch data from SpaceX API with the following URL:

```
In [ ]: spacex_url="https://api.spacexdata.com/v4/launches/past"
```

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

Check the content of the response

You should see the response contains massive information about SpaceX launches. Next, let's try to discover some more relevant information for this project.

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

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

```
In [ ]: response.status_code

Out[ ]: 200
```

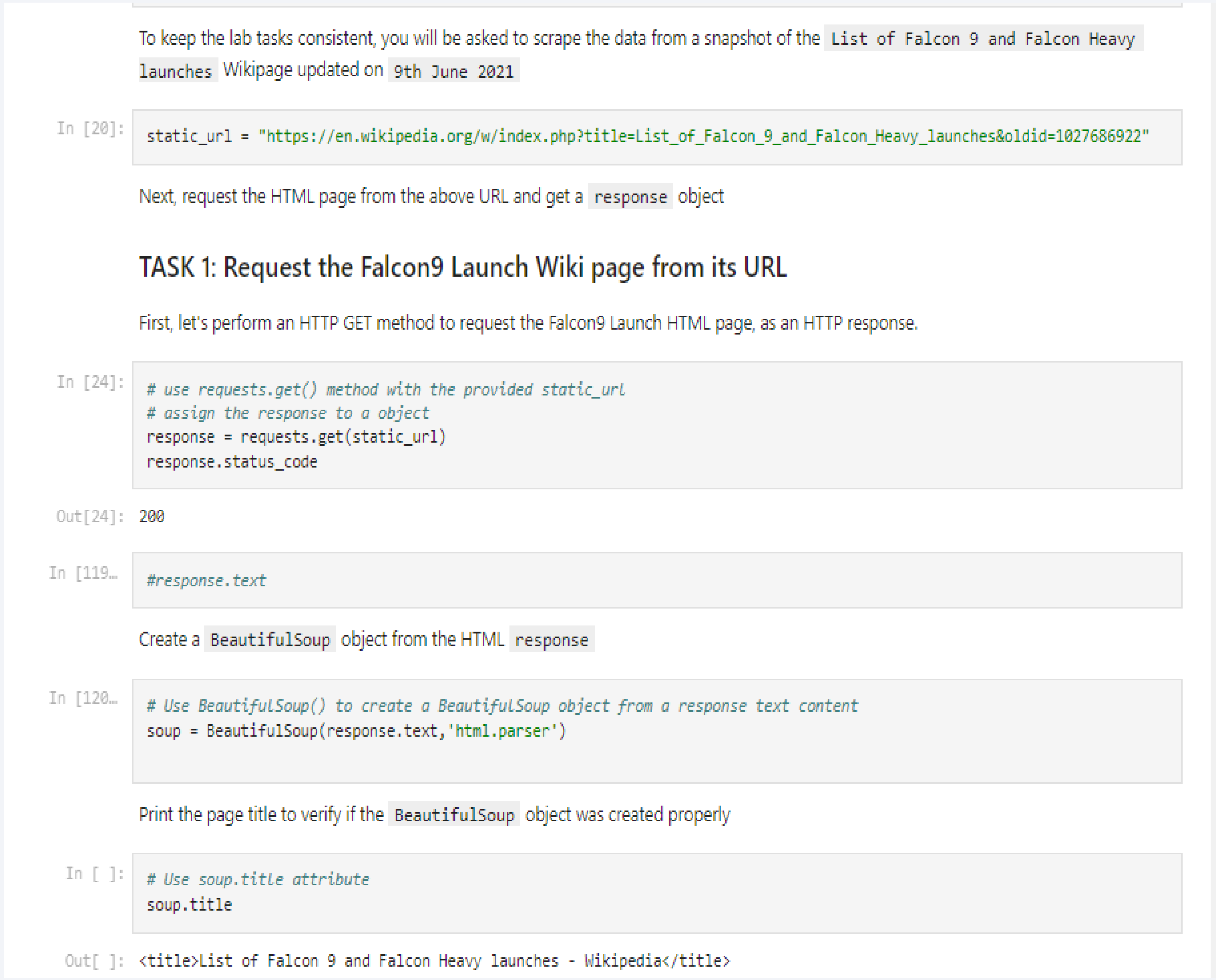
Now we decode the response content as a Json using `.json()` and turn it into a Pandas dataframe using `.json_normalize()`

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

```
In [ ]: responsee = requests.get("https://api.spacexdata.com/v4/launchpads/"+"5e9e4502f5090995de566f86").json()
        pd.json_normalize(responsee)
```

Data Collection - Scraping

- We utilized web scraping techniques using BeautifulSoup to extract Falcon 9 launch records from a web page.
- We then processed the scraped data, parsing the table and transforming it into a pandas dataframe.
- The URL to access the notebook is https://github.com/FloxTBoTyy/Mi-primer-repository/blob/main/jupyter_labs_webscraping.ipynb



To keep the lab tasks consistent, you will be asked to scrape the data from a snapshot of the `List of Falcon 9 and Falcon Heavy launches` Wikipage updated on `9th June 2021`

```
In [20]: static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"
```

Next, request the HTML page from the above URL and get a `response` object

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

Out[24]: 200

In [119… #response.text
```

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

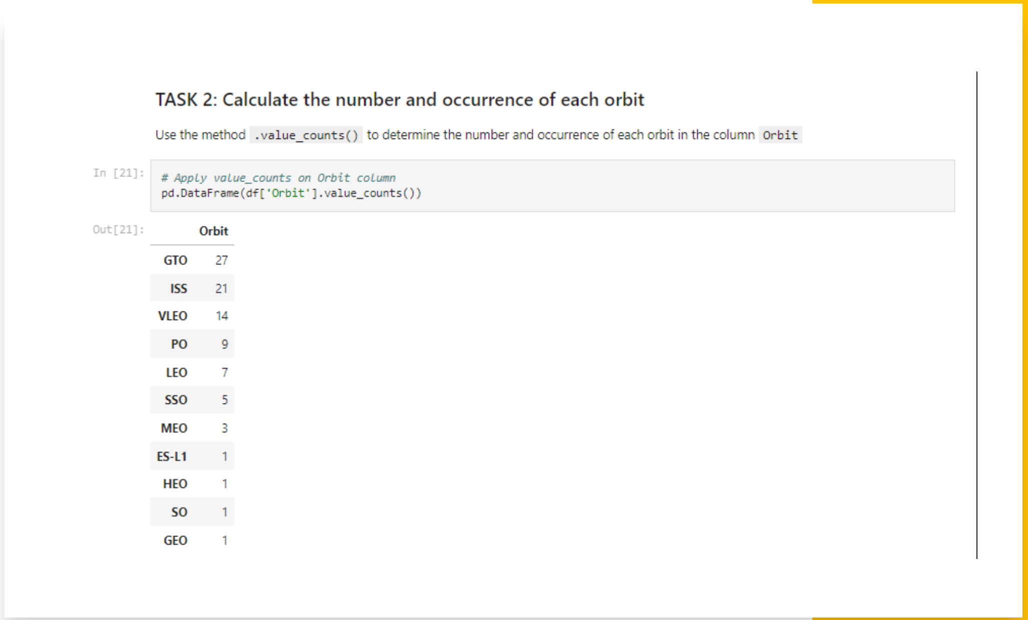
```
In [120… # Use BeautifulSoup() to create a BeautifulSoup object from a response text content
         soup = BeautifulSoup(response.text,'html.parser')
```

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

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

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

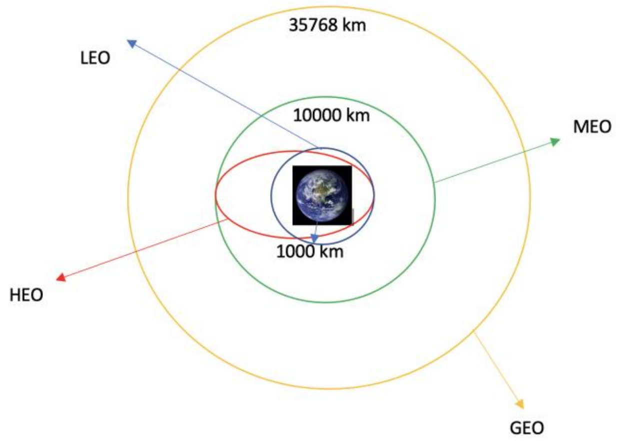
Data Wrangling

- Exploratory data analysis was performed to determine training labels.
- The number of launches at each site and the occurrence of each orbit were calculated.
- A landing outcome label was created from the outcome column and exported to a CSV file.
- The URL to access the notebook is https://github.com/FloxTBoTyy/Mi-primer-repository/blob/main/labs_jupyter_spacex_Data_wrangling.ipynb


TASK 2: Calculate the number and occurrence of each orbit

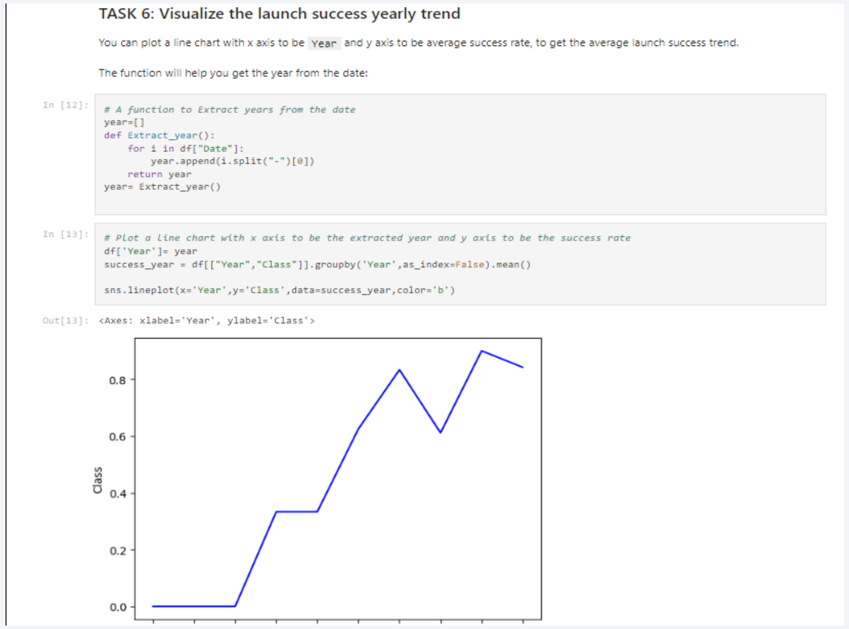
Use the method `.value_counts()` to determine the number and occurrence of each orbit in the column `Orbit`

```
In [21]: # Apply value_counts on Orbit column
         pd.DataFrame(df['Orbit'].value_counts())

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



EDA with Data Visualization

- We conducted data exploration by visualizing the relationships between flight number and launch site, payload and launch site, success rate of each orbit type, flight number and orbit type, and the yearly trend of launch success.



TASK 6: Visualize the launch success yearly trend

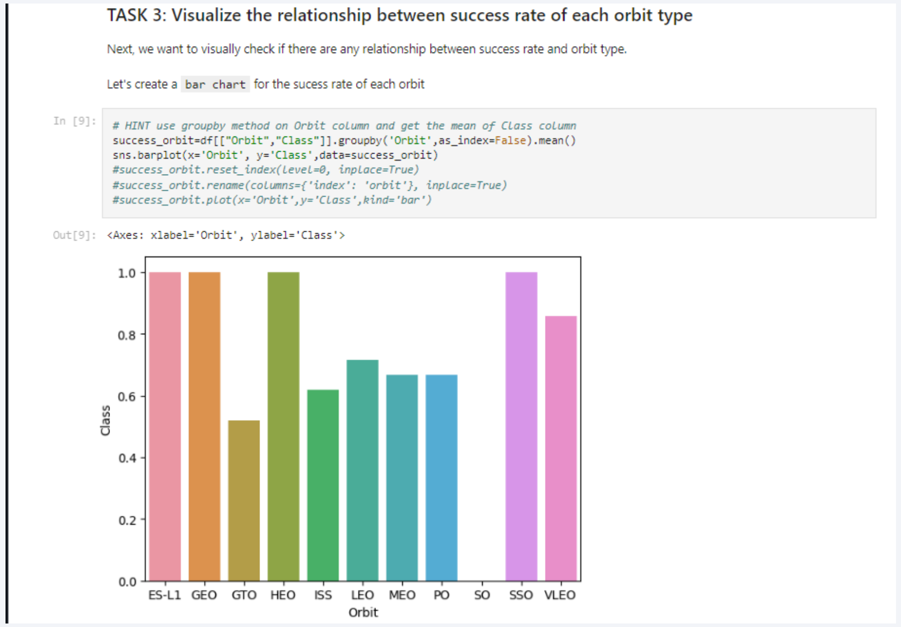
You can plot a line chart with x axis to be `Year` and y axis to be average success rate, to get the average launch success trend.

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

```
In [12]:
# A function to Extract years from the date
year=[]
def Extract_year():
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year
year= Extract_year()
```

```
In [13]:
# Plot a line chart with x axis to be the extracted year and y axis to be the success rate
df['Year']= year
success_year = df[["Year","Class"]].groupby('Year',as_index=False).mean()

sns.lineplot(x='Year',y='Class',data=success_year,color='b')
```

```
Out[13]: <Axes: xlabel='Year', ylabel='Class'>
```



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

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

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

```
In [9]:
# HINT use groupby method on Orbit column and get the mean of Class column
success_orbit=df[["Orbit","Class"]].groupby('Orbit',as_index=False).mean()
sns.barplot(x='Orbit', y='Class',data=success_orbit)
#success_orbit.reset_index(level=0, inplace=True)
#success_orbit.rename(columns={'index': 'orbit'}, inplace=True)
#success_orbit.plot(x='Orbit',y='Class',kind='bar')
```

```
Out[9]: <Axes: xlabel='Orbit', ylabel='Class'>
```

- The URL to access the notebook is https://github.com/FloxTBoTyy/Mi-primer-repository/blob/main/jupyter_labs_eda_dataviz.ipynb

EDA with SQL

- We imported the SpaceX dataset into a SQLite3 database directly from the Jupyter Notebook environment.
    1. We utilized SQL for exploratory data analysis to gain insights from the dataset. We formulated queries to retrieve information such as:
    2. The distinct names of launch sites involved in space missions.
    3. The total payload mass carried by boosters launched by NASA (CRS).
    4. The average payload mass carried by booster version F9 v1.1.
    5. The total count of successful and failed mission outcomes.
    6. The failed landing outcomes on drone ships, including the corresponding booster version and launch site names.
- The URL to the Notebook is https://github.com/FloxTBoTyy/Mi-primer-repository/blob/main/jupyter_labs_eda_sql_coursera_sqllite.ipynb

Build an Interactive Map with Folium

- We utilized Folium to visualize the launch sites on a map and added map elements such as markers, circles, and lines to indicate the success or failure of launches at each site.
- We assigned binary labels (0 for failure, 1 for success) to the launch outcomes feature.
- By analyzing the color-labeled marker clusters, we identified launch sites with relatively high success rates.
- We computed the distances between launch sites and their surrounding areas. This helped us answer questions such as:
    1. Are launch sites located near railways, highways, and coastlines?
    2. Do launch sites maintain a certain distance from cities?
- The URL to the notebook is https://github.com/FloxTBoTyy/Mi-primer-repository/blob/main/lab_jupyter_launch_site_location.ipynb

Build a Dashboard with Plotly Dash

- We developed an interactive dashboard using Plotly Dash.
- We created pie charts to visualize the total number of launches at each specific site.
- We generated scatter plots to examine the relationship between the launch outcome and payload mass (in kilograms) for different booster versions.
- The URL of the notebook is https://github.com/FloxTBoTyy/Mi-primer-repository/blob/main/dashboard_spacex.ipynb (in jupyter notebook the dashboard works very well)

Predictive Analysis (Classification)

- We imported the data using numpy and pandas, performed data transformation, and split the data into training and testing sets.
- We constructed various machine learning models and fine-tuned their hyperparameters using GridSearchCV.
- We evaluated the models based on their accuracy and iteratively enhanced them through feature engineering and algorithm tuning.
- We identified the classification model that achieved the highest performance.
- The URL for the notebook is https://github.com/FloxTBoTyy/Mi-primer-repository/blob/main/SpaceX_Machine_Learning_Prediction_Part_5_jupyterlite.ipynb

Results

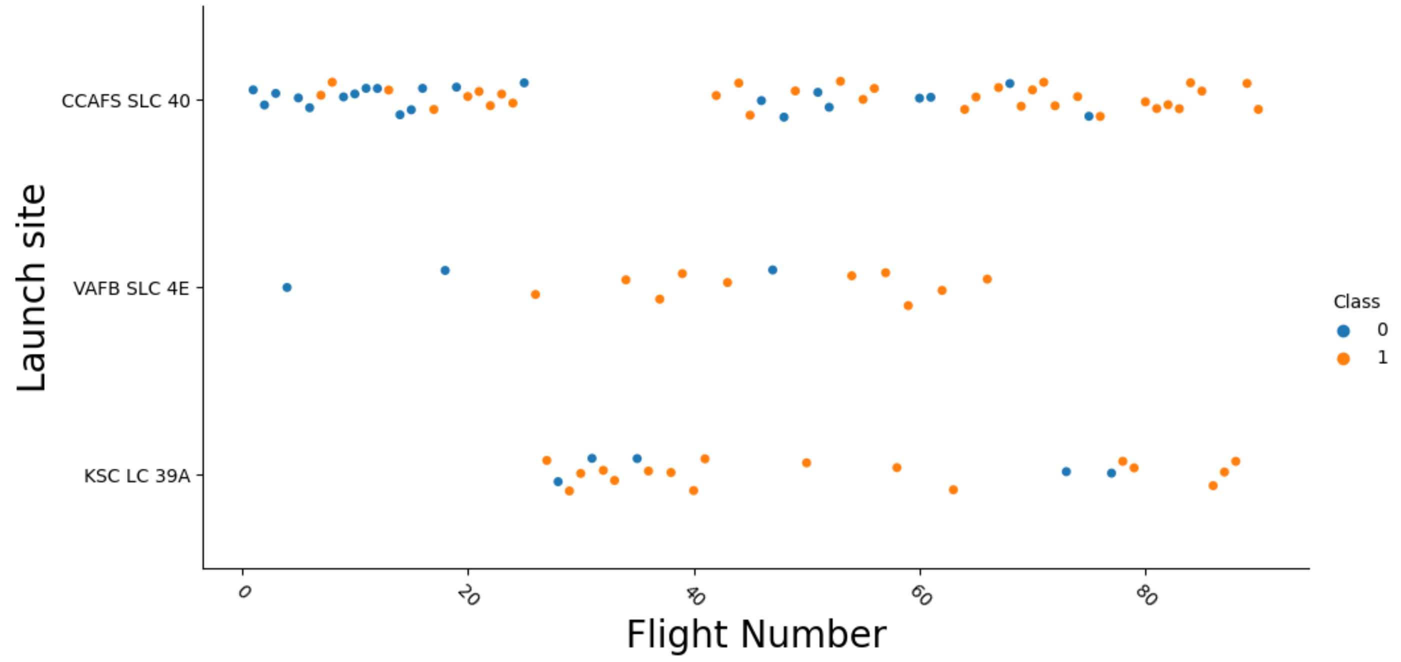
EXPLORATORY DATA ANALYSIS RESULTS

INTERACTIVE ANALYTICS DEMO IN SCREENSHOTS

PREDICTIVE ANALYSIS RESULTS

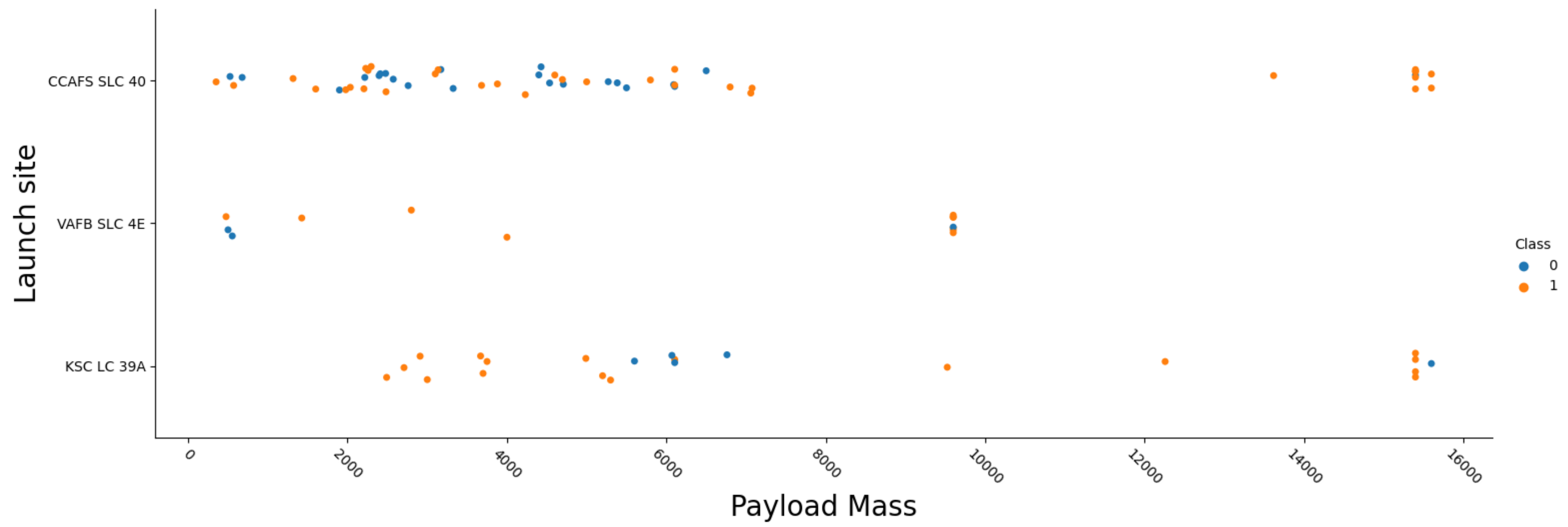
Section 2

Insights drawn from EDA



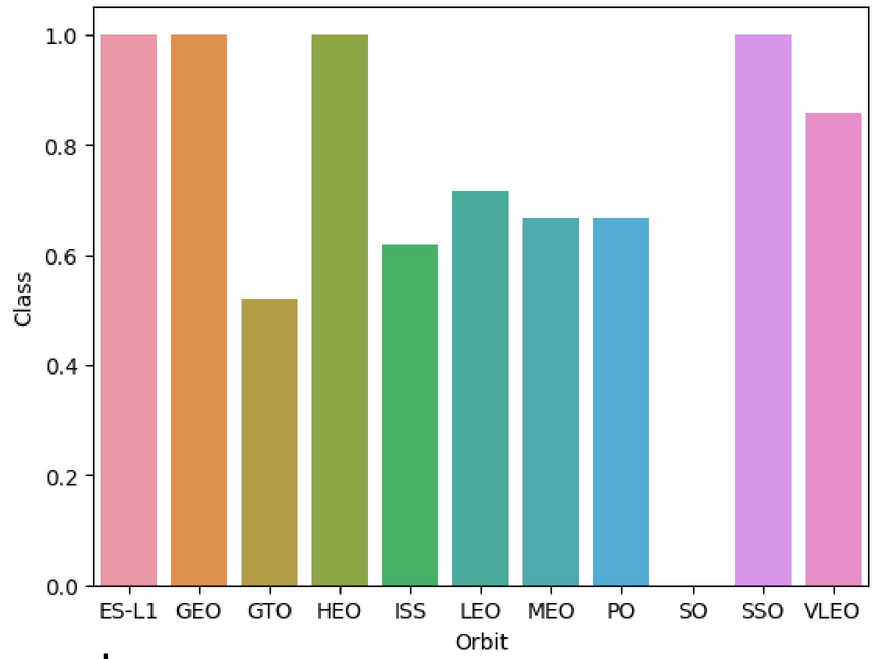
- We see that the higher success rate is from KSC LC 39 with 77.72%

Flight Number vs. Launch Site



- As we can see there are no rockets launched for heavypayload mass for VAFB-SLC

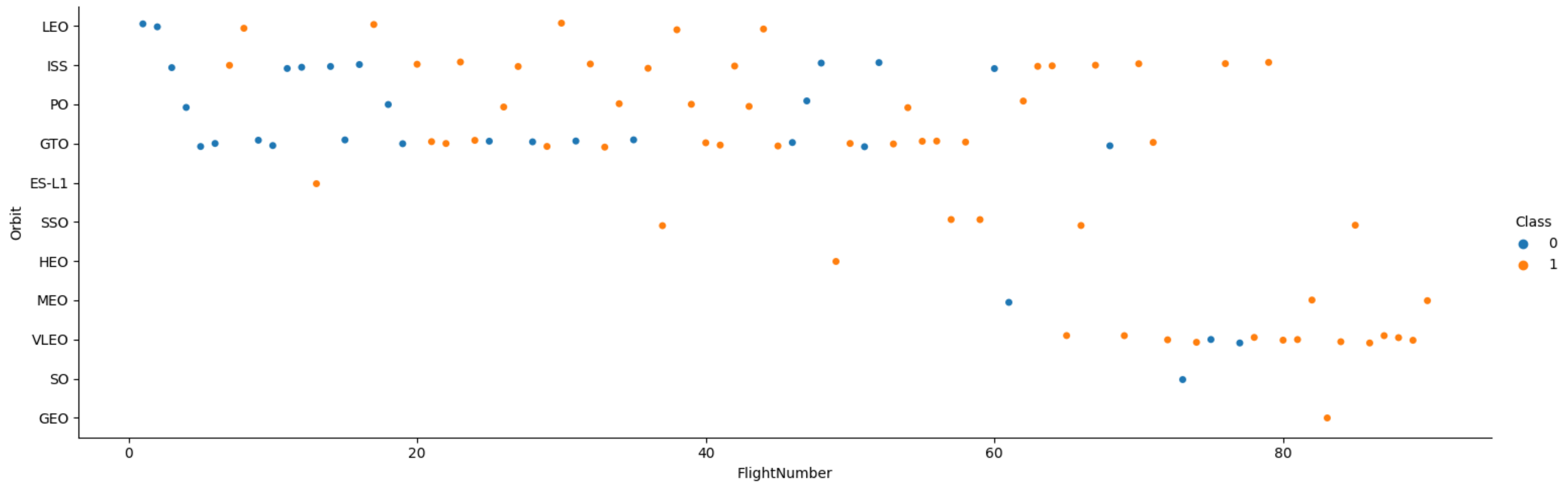
Payload vs. Launch Site



- The orbits with most success rate are:

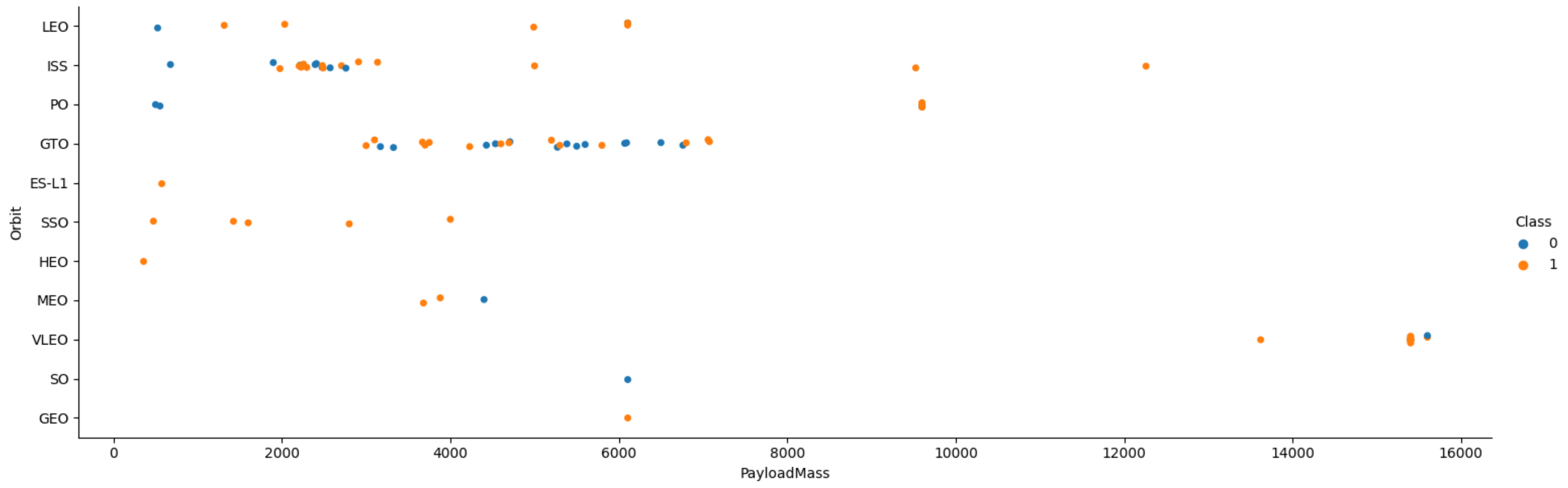
1. ES-L1
2. GEO
3. HEO
4. SSO

Success Rate vs. Orbit Type



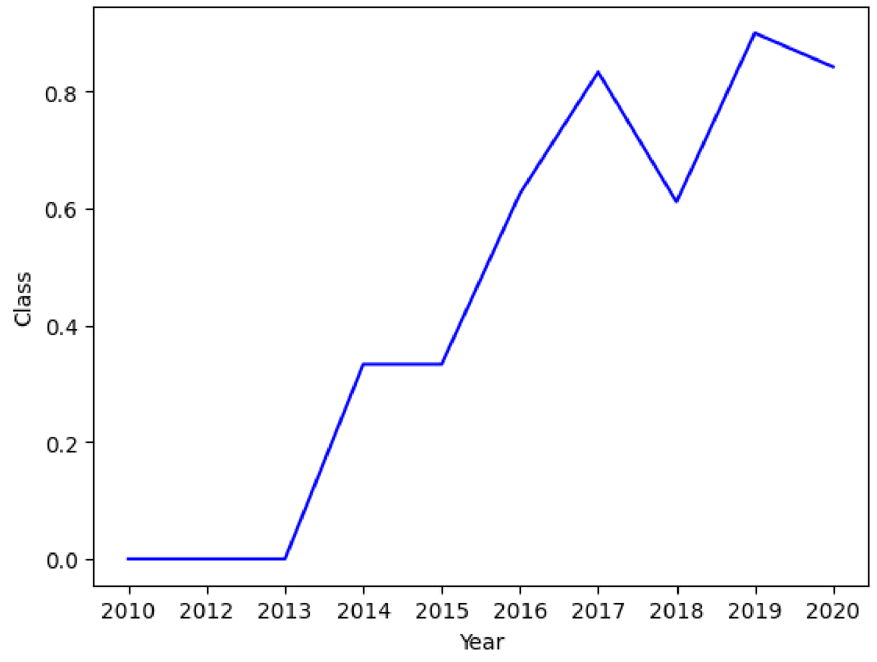
- In the Low Earth Orbit (LEO), we observe a correlation between the number of flights and the success rate. However, in the Geostationary Transfer Orbit (GTO), there doesn't seem to be a connection between the flight number and the success rate.

Flight Number vs. Orbit Type



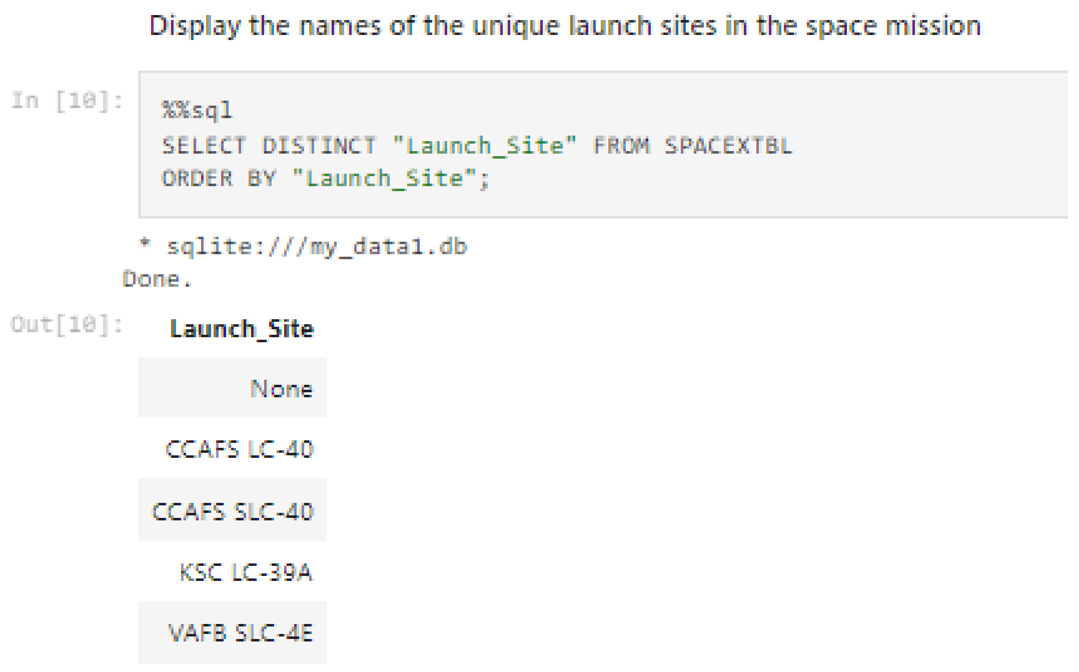
- For heavy payloads, we notice a higher success rate or positive landing rate in the Polar, LEO, and ISS orbits.
- However, in the GTO orbit, it is difficult to distinguish this pattern clearly as both the positive landing rate and the negative landing rate (unsuccessful missions) are present.

Payload vs. Orbit Type



- From 2013 to 2020, there is a noticeable upward trend in the success rate.

Launch Success Yearly Trend


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

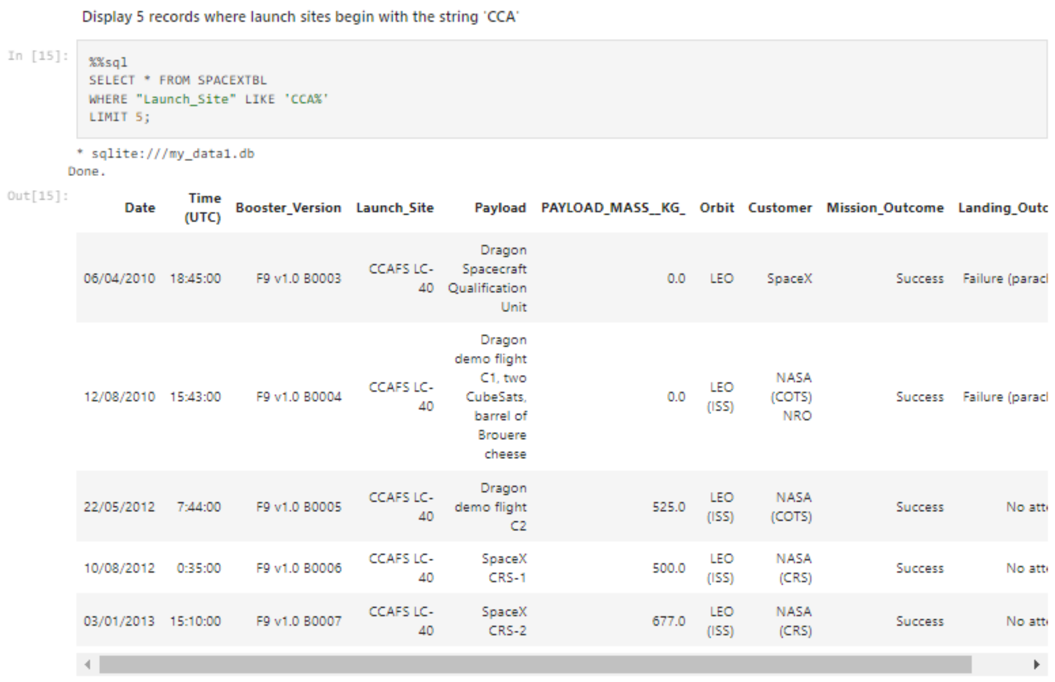
```
In [10]:  %%sql
          SELECT DISTINCT "Launch_Site" FROM SPACEXTBL
          ORDER BY "Launch_Site";

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

Out[10]:

Launch_Site
None
CCAFS LC-40
CCAFS SLC-40
KSC LC-39A
VAFB SLC-4E

- The first query uses the SELECT DISTINCT statement to retrieve unique launch sites from the SPACEXTBL table. The ORDER BY clause is used to sort the launch sites in alphabetical order.

All Launch Site Names


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

```
In [15]:
%%sql
SELECT * FROM SPACEXTBL
WHERE "Launch_Site" LIKE 'CCA%'
LIMIT 5;
```

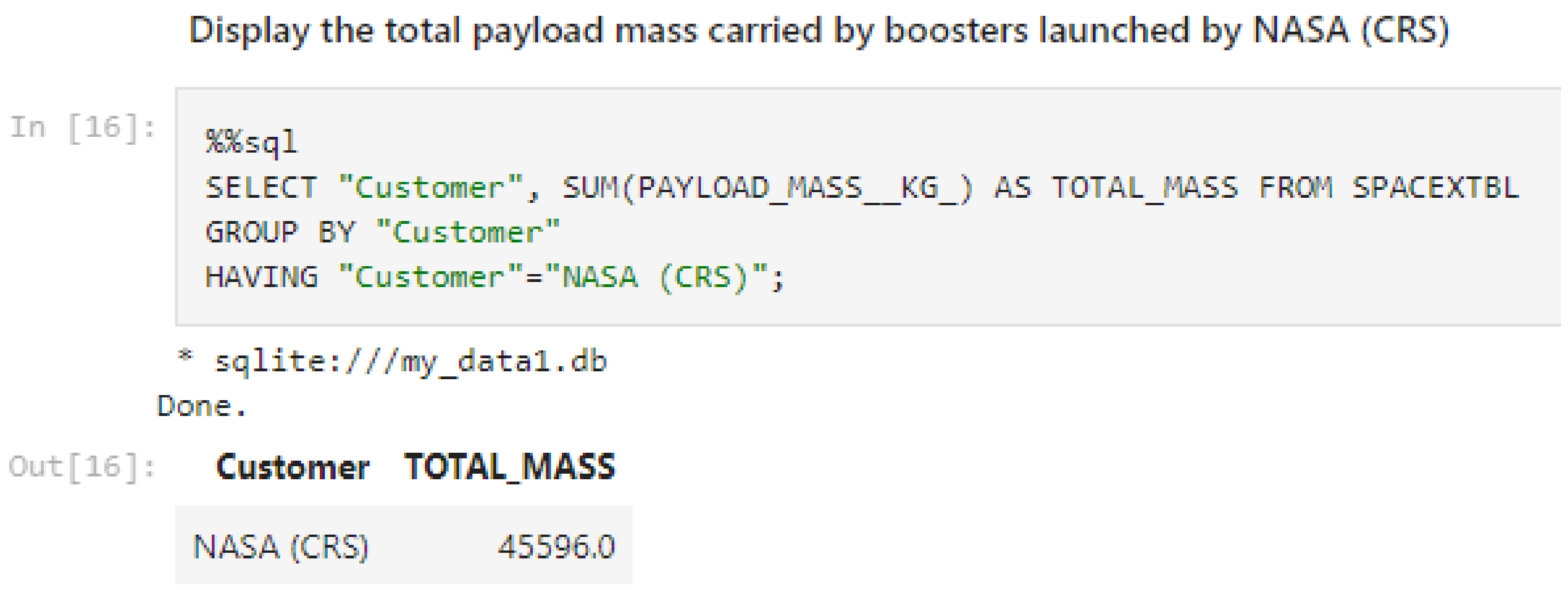
```
 * sqlite:///my_data1.db
Done.
```

Out[15]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outc
06/04/2010	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0.0	LEO	SpaceX	Success	Failure (paracl
12/08/2010	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0.0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (paracl
22/05/2012	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525.0	LEO (ISS)	NASA (COTS)	Success	No att
10/08/2012	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500.0	LEO (ISS)	NASA (CRS)	Success	No att
03/01/2013	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677.0	LEO (ISS)	NASA (CRS)	Success	No att

- The second query uses the LIKE operator with a wildcard '%' to find rows from the SPACEXTBL table where the Launch_Site starts with 'CCA'. The LIMIT clause limits the result to the first 5 rows.

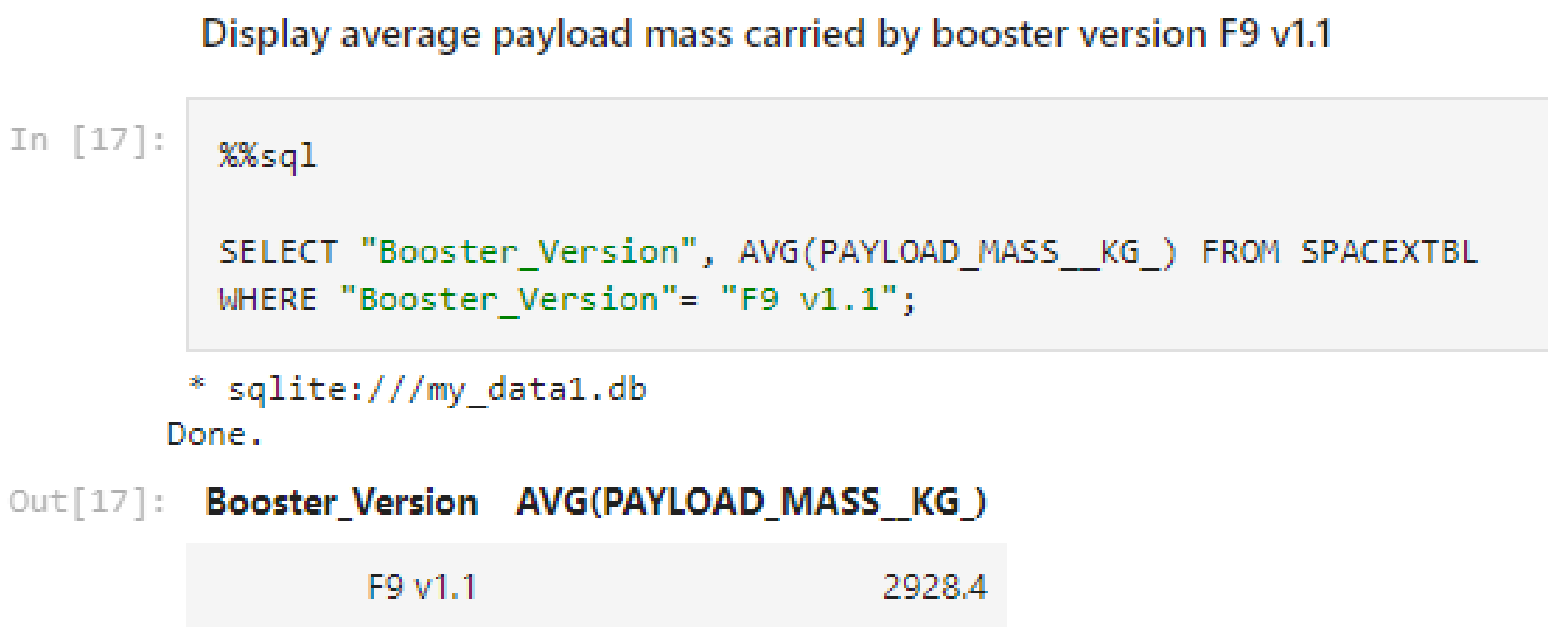
Launch Site Names Begin with 'CCA'


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

```
In [16]:  %%sql
          SELECT "Customer", SUM(PAYLOAD_MASS__KG_) AS TOTAL_MASS FROM SPACEXTBL
          GROUP BY "Customer"
          HAVING "Customer"="NASA (CRS)";

           * sqlite:///my_data1.db
          Done.

Out[16]:    Customer    TOTAL_MASS
          NASA (CRS)       45596.0
```

- The third query uses the GROUP BY clause to group the data by the "Customer" column. The HAVING clause filters the groups and selects only the group where the "Customer" is equal to "NASA (CRS)". The SUM function is used to calculate the total payload mass for each customer.

Total Payload Mass


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

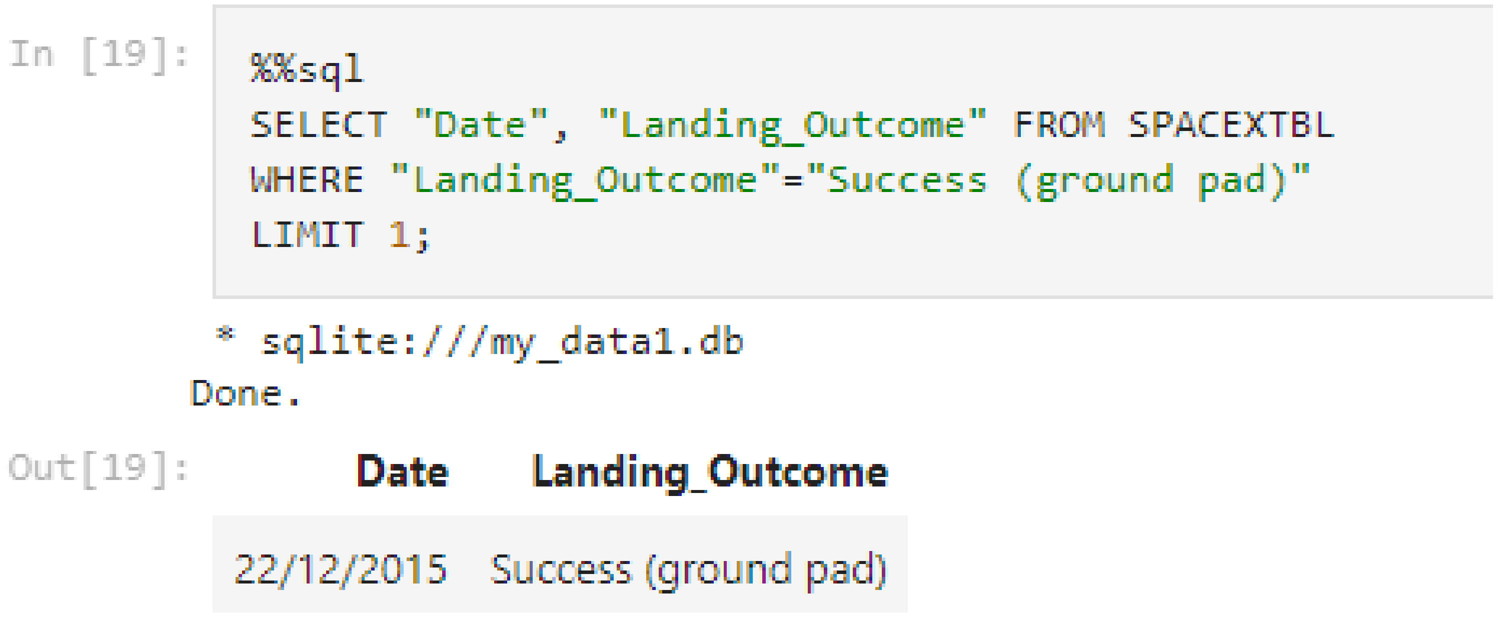
```
In [17]: %%sql

         SELECT "Booster_Version", AVG(PAYLOAD_MASS__KG_) FROM SPACEXTBL
         WHERE "Booster_Version"= "F9 v1.1";

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

Out[17]:

Booster_Version	AVG(PAYLOAD_MASS__KG_)
F9 v1.1	2928.4

- The fourth query uses the WHERE clause to filter rows where the "Booster_Version" is equal to "F9 v1.1". The AVG function is used to calculate the average payload mass for this specific booster version.

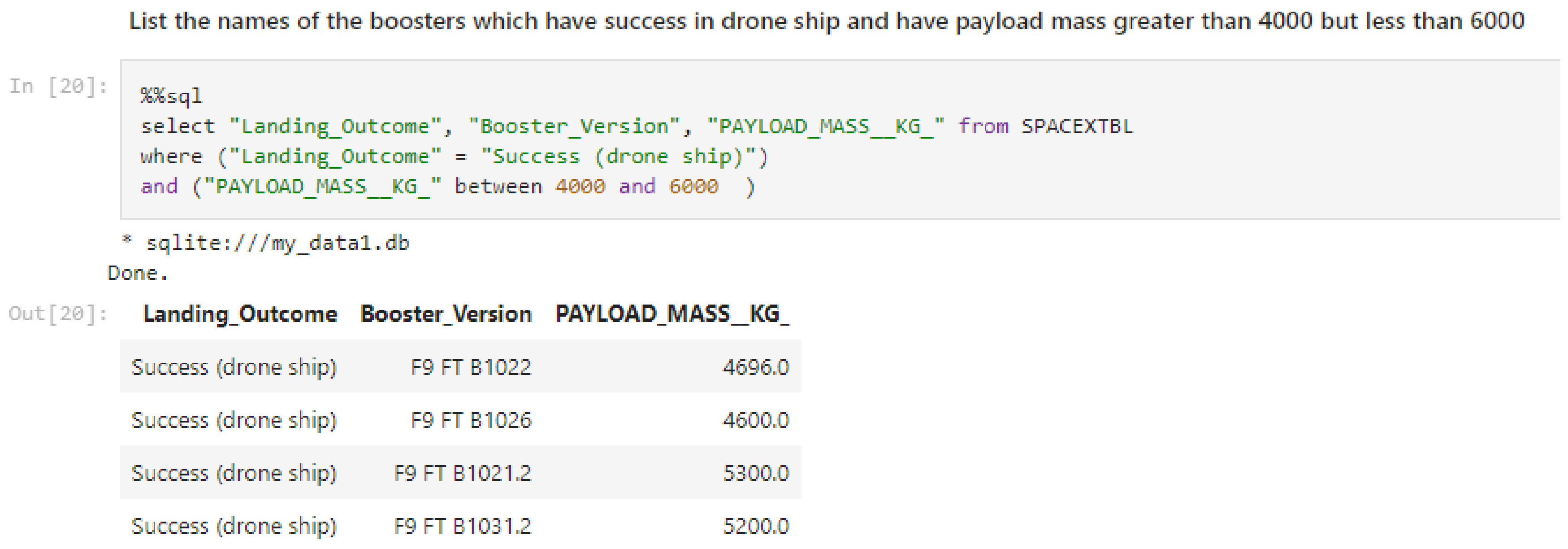
Average Payload Mass by F9 v1.1



```
In [19]: %%sql
         SELECT "Date", "Landing_Outcome" FROM SPACEXTBL
         WHERE "Landing_Outcome"="Success (ground pad)"
         LIMIT 1;

 * sqlite:///my_data1.db
Done.

Out[19]:     Date        Landing_Outcome
         22/12/2015   Success (ground pad)
```

- The fifth query uses the WHERE clause to select rows where the "Landing_Outcome" is equal to "Success (ground pad)". The LIMIT clause limits the result to 1 row.

First Successful Ground Landing Date


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

In [20]:
```
%%sql
select "Landing_Outcome", "Booster_Version", "PAYLOAD_MASS__KG_" from SPACEXTBL
where ("Landing_Outcome" = "Success (drone ship)")
and ("PAYLOAD_MASS__KG_" between 4000 and 6000  )
```

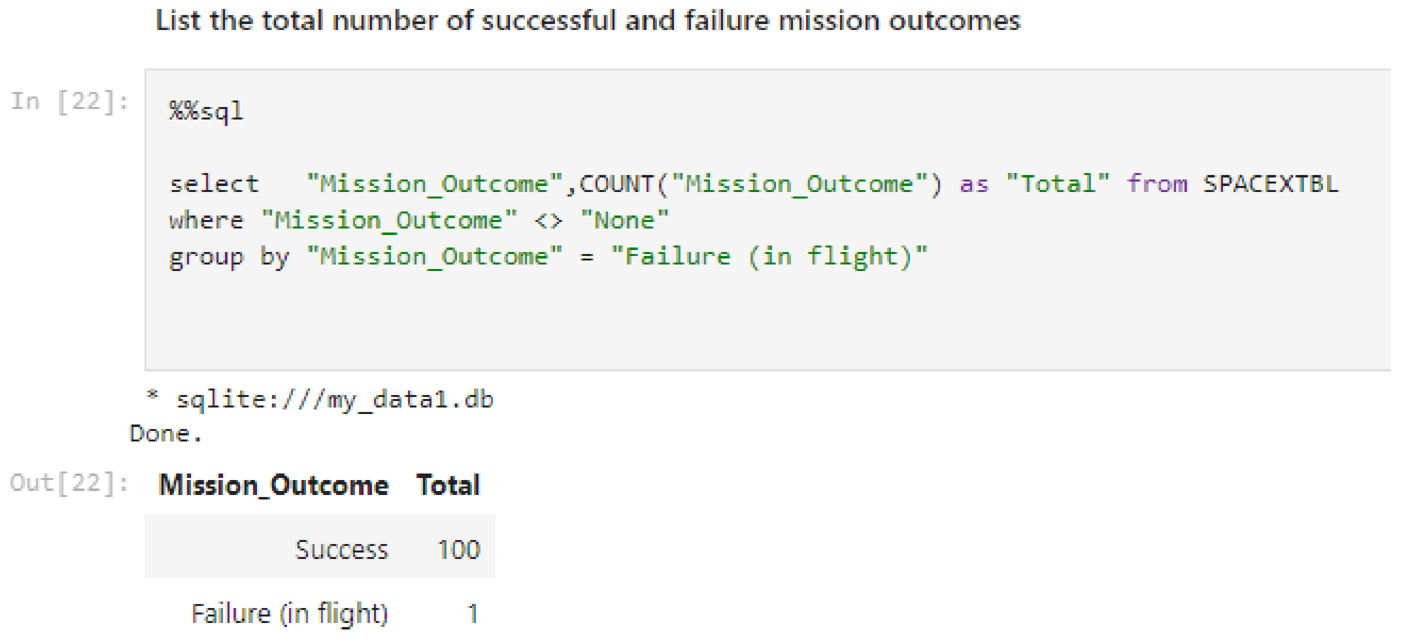
```
 * sqlite:///my_data1.db
Done.
```

Out[20]:

Landing_Outcome	Booster_Version	PAYLOAD_MASS__KG_
Success (drone ship)	F9 FT B1022	4696.0
Success (drone ship)	F9 FT B1026	4600.0
Success (drone ship)	F9 FT B1021.2	5300.0
Success (drone ship)	F9 FT B1031.2	5200.0

- The sixth query uses the WHERE clause with multiple conditions to select rows where the "Landing_Outcome" is equal to "Success (drone ship)" and the "PAYLOAD_MASS__KG_" is between 4000 and 6000.

Successful Drone Ship Landing with Payload between 4000 and 6000


List the total number of successful and failure mission outcomes

In [22]:
```
%%sql

select   "Mission_Outcome",COUNT("Mission_Outcome") as "Total" from SPACEXTBL
where "Mission_Outcome" <> "None"
group by "Mission_Outcome" = "Failure (in flight)"
```

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

Out[22]:

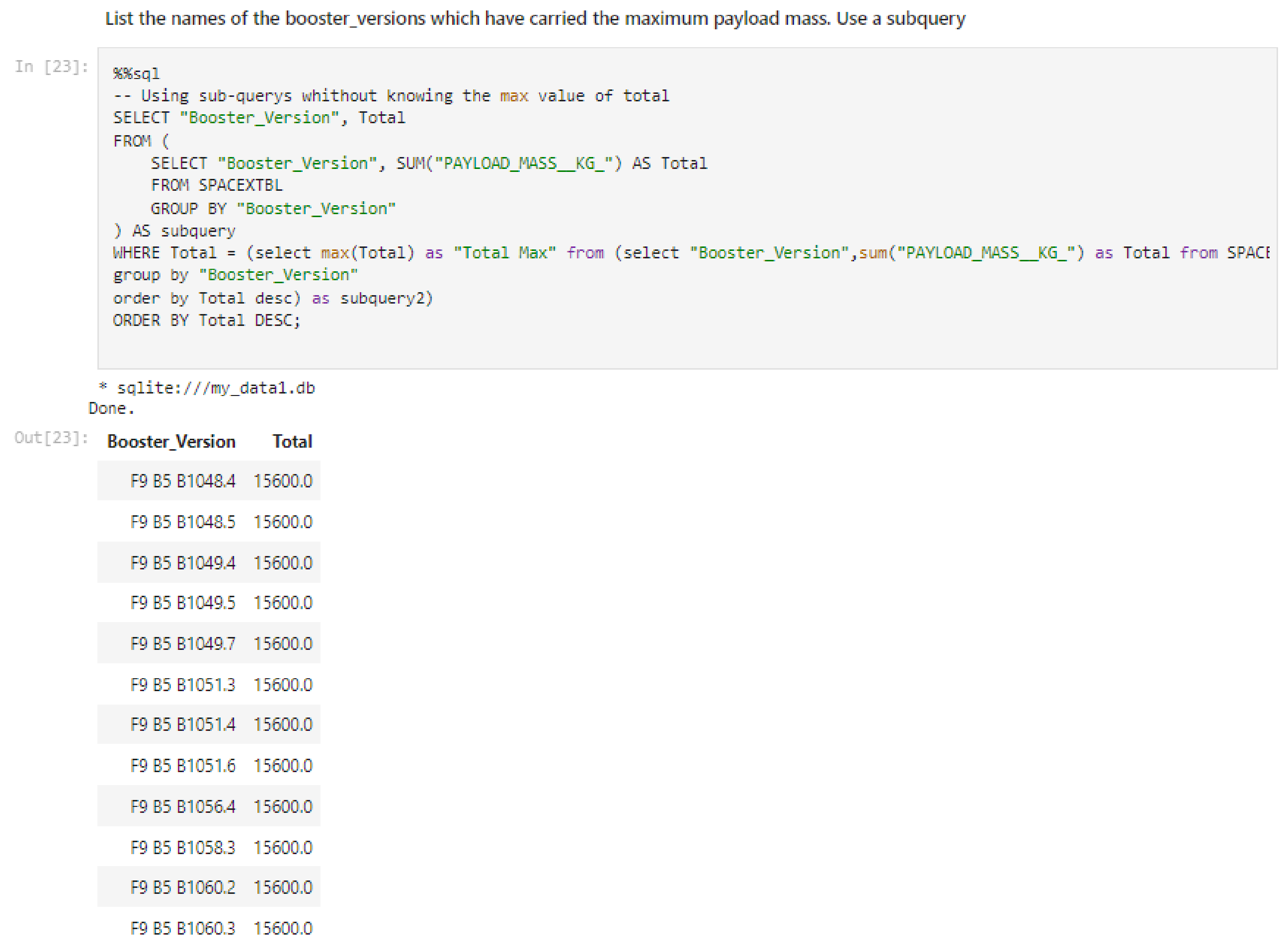
Mission_Outcome	Total
Success	100
Failure (in flight)	1

- The seventh query uses the WHERE clause to filter rows where the "Mission_Outcome" is not equal to "None". The GROUP BY clause groups the data by the "Mission_Outcome" column, and the COUNT function is used to count the occurrences of each "Mission_Outcome" value. The result is further filtered to include only the "Failure (in flight)" outcome.

Total Number of Successful and Failure Mission Outcomes

Boosters Carried Maximum Payload

- The eighth query uses a subquery to calculate the total payload mass for each booster version. The outer query then retrieves the booster version and total payload mass where the total payload mass is equal to the maximum value obtained from the subquery. The result is ordered by the total payload mass in descending order.



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

In [23]:
```
%%sql
-- Using sub-querys whithout knowing the max value of total
SELECT "Booster_Version", Total
FROM (
    SELECT "Booster_Version", SUM("PAYLOAD_MASS__KG_") AS Total
    FROM SPACEXTBL
    GROUP BY "Booster_Version"
) AS subquery
WHERE Total = (select max(Total) as "Total Max" from (select "Booster_Version",sum("PAYLOAD_MASS__KG_") as Total from SPACE
group by "Booster_Version"
order by Total desc) as subquery2)
ORDER BY Total DESC;
```

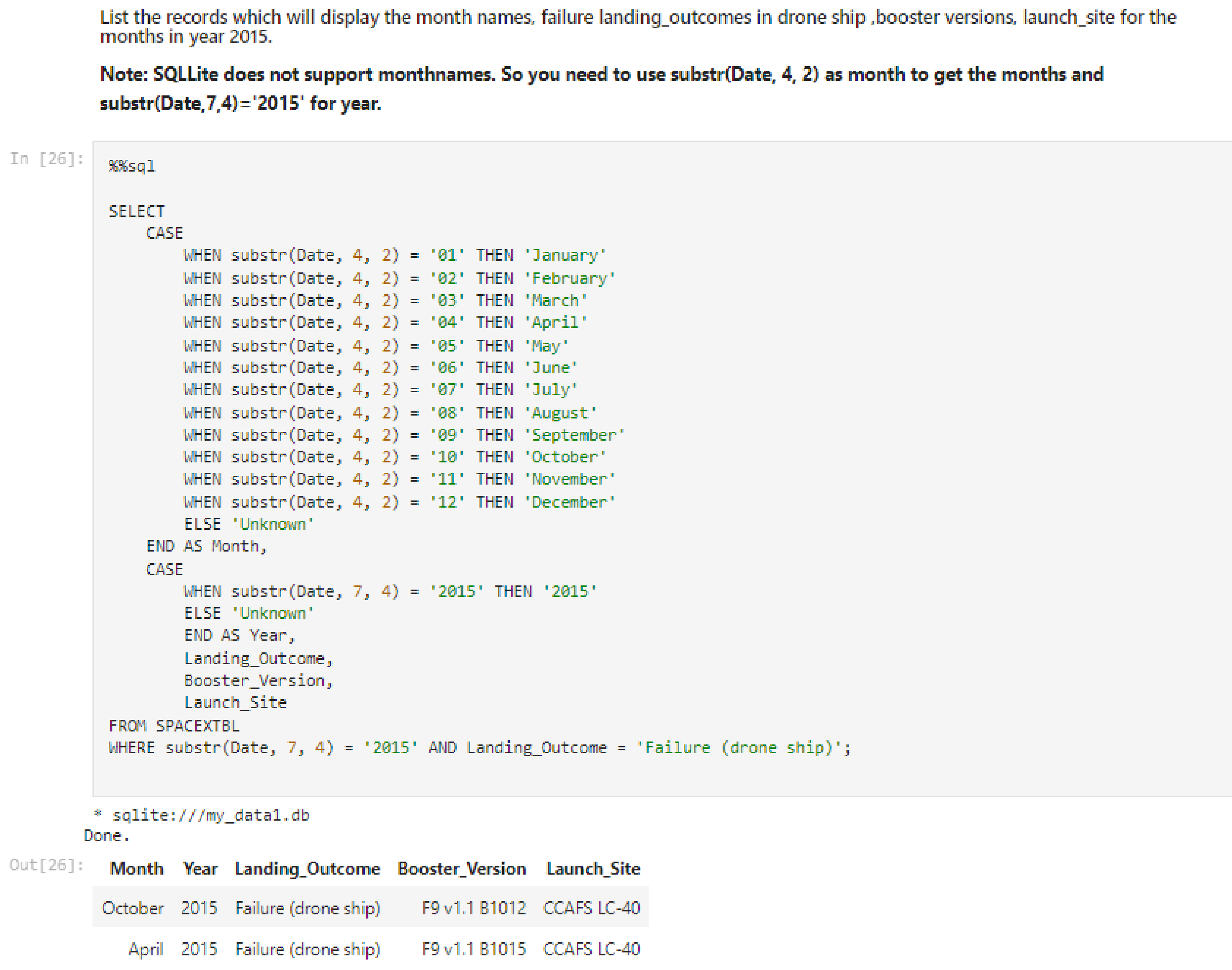
```
 * sqlite:///my_data1.db
Done.
```

Out[23]:

Booster_Version	Total
F9 B5 B1048.4	15600.0
F9 B5 B1048.5	15600.0
F9 B5 B1049.4	15600.0
F9 B5 B1049.5	15600.0
F9 B5 B1049.7	15600.0
F9 B5 B1051.3	15600.0
F9 B5 B1051.4	15600.0
F9 B5 B1051.6	15600.0
F9 B5 B1056.4	15600.0
F9 B5 B1058.3	15600.0
F9 B5 B1060.2	15600.0
F9 B5 B1060.3	15600.0

2015 Launch Records

- The ninth query uses the CASE statement to convert the month and year values in the "Date" column into readable formats. The WHERE clause filters rows where the year is equal to '2015' and the landing outcome is 'Failure (drone ship)'.



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, 4, 2) as month to get the months and substr(Date,7,4)='2015' for year.

In [26]:

```
%%sql

SELECT
    CASE
        WHEN substr(Date, 4, 2) = '01' THEN 'January'
        WHEN substr(Date, 4, 2) = '02' THEN 'February'
        WHEN substr(Date, 4, 2) = '03' THEN 'March'
        WHEN substr(Date, 4, 2) = '04' THEN 'April'
        WHEN substr(Date, 4, 2) = '05' THEN 'May'
        WHEN substr(Date, 4, 2) = '06' THEN 'June'
        WHEN substr(Date, 4, 2) = '07' THEN 'July'
        WHEN substr(Date, 4, 2) = '08' THEN 'August'
        WHEN substr(Date, 4, 2) = '09' THEN 'September'
        WHEN substr(Date, 4, 2) = '10' THEN 'October'
        WHEN substr(Date, 4, 2) = '11' THEN 'November'
        WHEN substr(Date, 4, 2) = '12' THEN 'December'
        ELSE 'Unknown'
    END AS Month,
    CASE
        WHEN substr(Date, 7, 4) = '2015' THEN '2015'
        ELSE 'Unknown'
        END AS Year,
        Landing_Outcome,
        Booster_Version,
        Launch_Site
FROM SPACEXTBL
WHERE substr(Date, 7, 4) = '2015' AND Landing_Outcome = 'Failure (drone ship)';
```

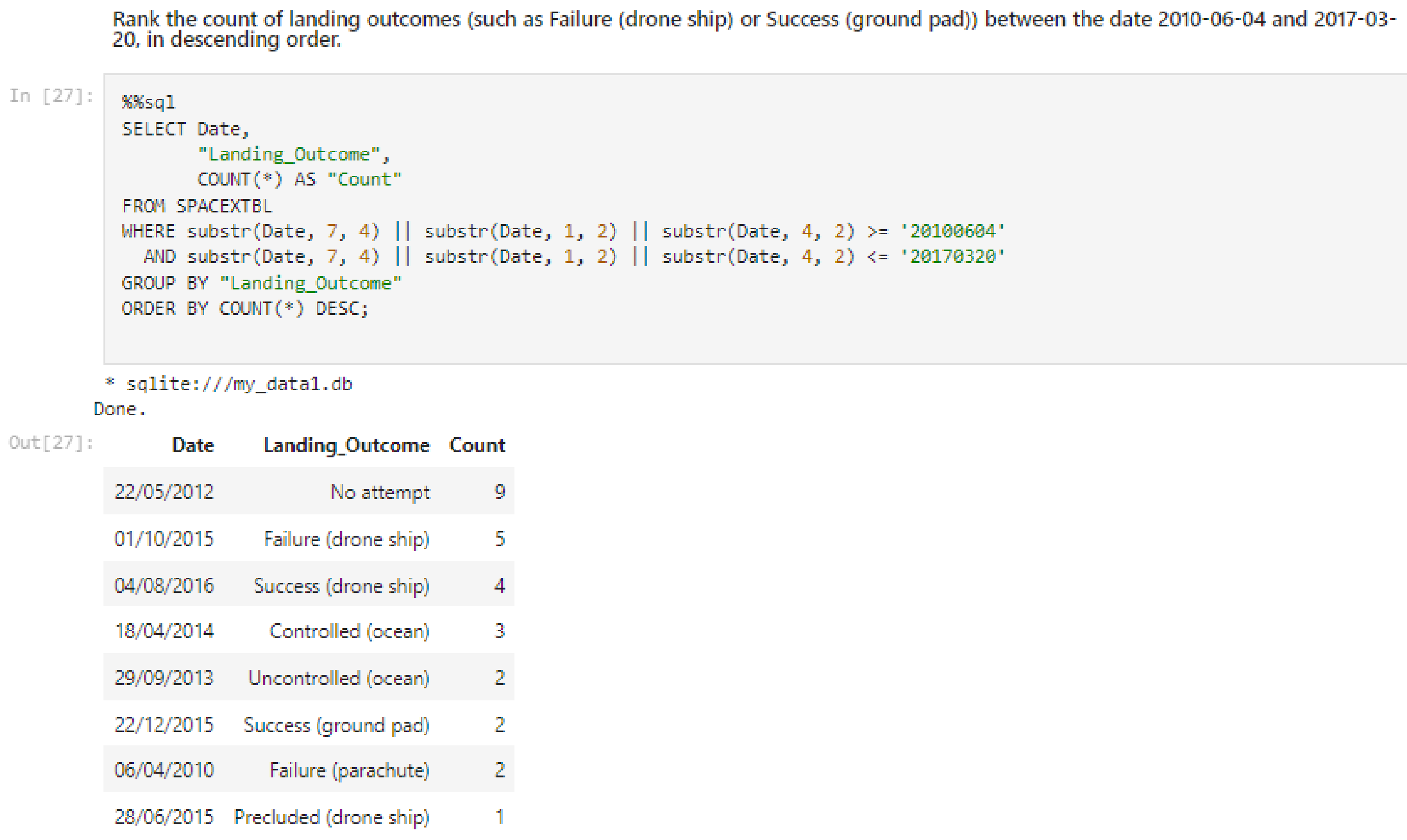
```
 * sqlite:///my_data1.db
Done.
```

Out[26]:

Month	Year	Landing_Outcome	Booster_Version	Launch_Site
October	2015	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
April	2015	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

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

- The tenth query uses the WHERE clause with the BETWEEN operator to select rows where the date falls within a specified range. The GROUP BY clause groups the data by the "Landing_Outcome" column, and the COUNT function is used to count the occurrences of each landing outcome. The result is ordered by the count in descending order.



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 [27]:
%%sql
SELECT Date,
       "Landing_Outcome",
       COUNT(*) AS "Count"
FROM SPACEXTBL
WHERE substr(Date, 7, 4) || substr(Date, 1, 2) || substr(Date, 4, 2) >= '20100604'
  AND substr(Date, 7, 4) || substr(Date, 1, 2) || substr(Date, 4, 2) <= '20170320'
GROUP BY "Landing_Outcome"
ORDER BY COUNT(*) DESC;

 * sqlite:///my_data1.db
Done.
Out[27]:
```

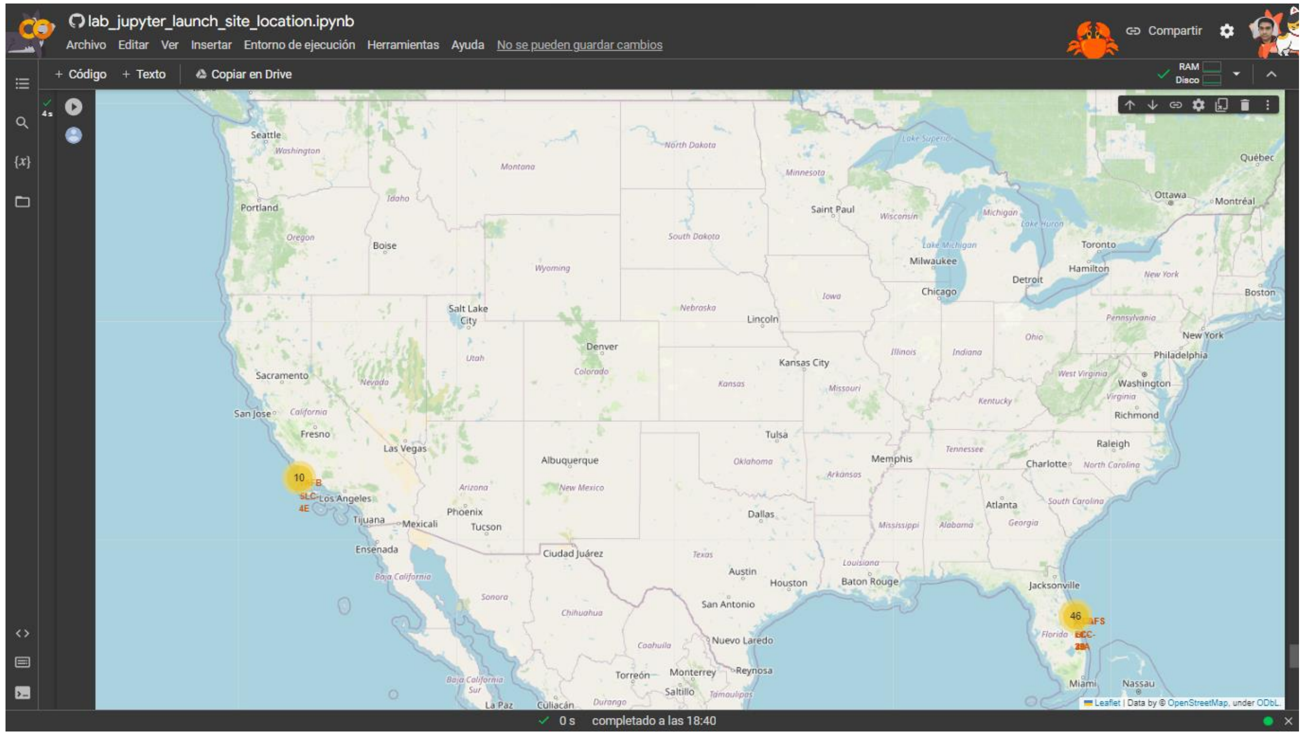
Date	Landing_Outcome	Count
22/05/2012	No attempt	9
01/10/2015	Failure (drone ship)	5
04/08/2016	Success (drone ship)	4
18/04/2014	Controlled (ocean)	3
29/09/2013	Uncontrolled (ocean)	2
22/12/2015	Success (ground pad)	2
06/04/2010	Failure (parachute)	2
28/06/2015	Precluded (drone ship)	1

Section 3

Launch Sites Proximities Analysis

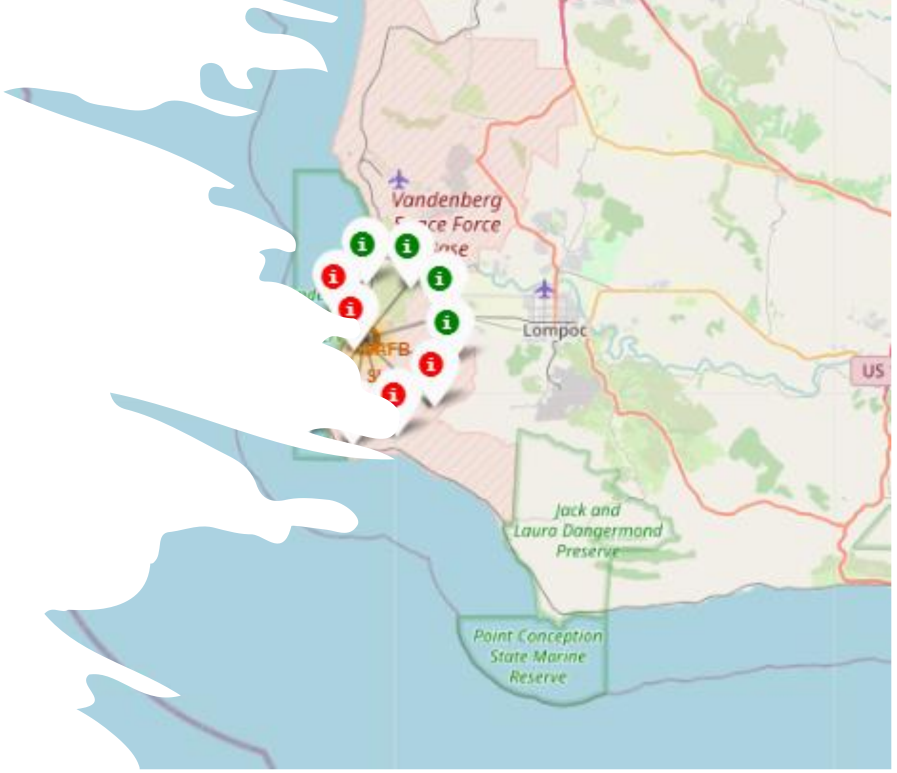
Markers of launch sites exclusively within the United States on a map.

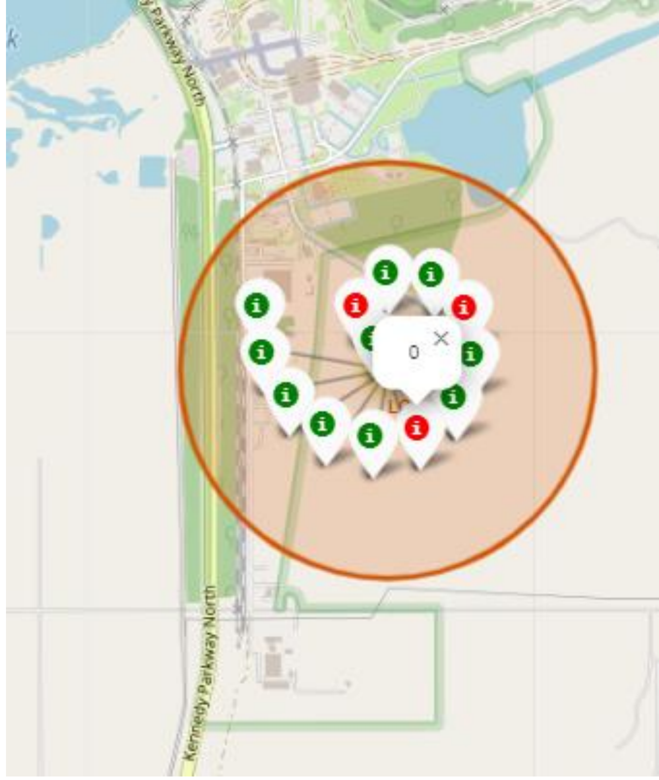
- As we can see the launch sites are in Florida and California.

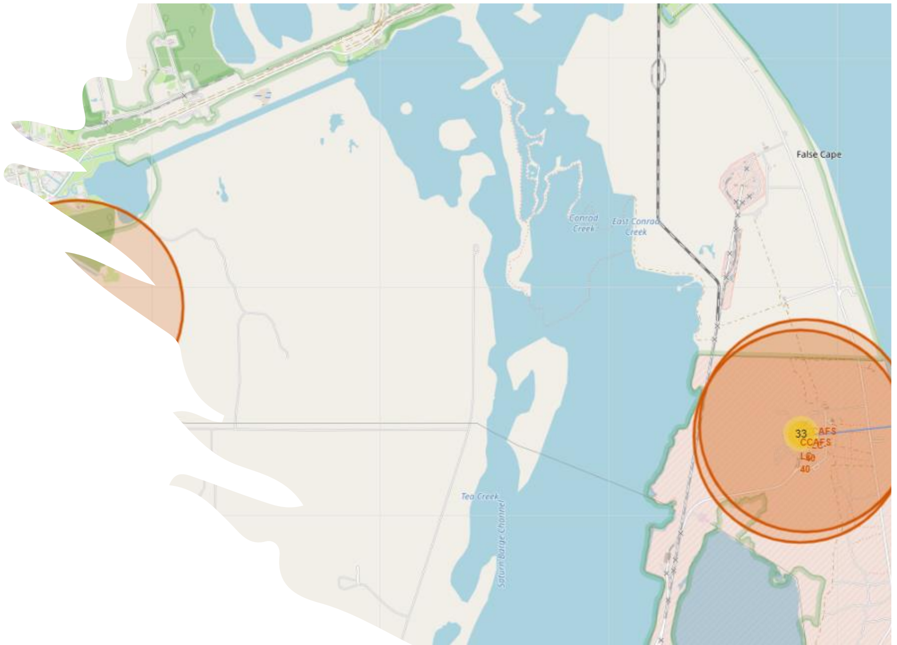

lab_jupyter_launch_site_location.ipynb
Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda No se pueden guardar cambios
+ Código + Texto Copiar en Drive
Compartir
10
46
0 s completado a las 18:40

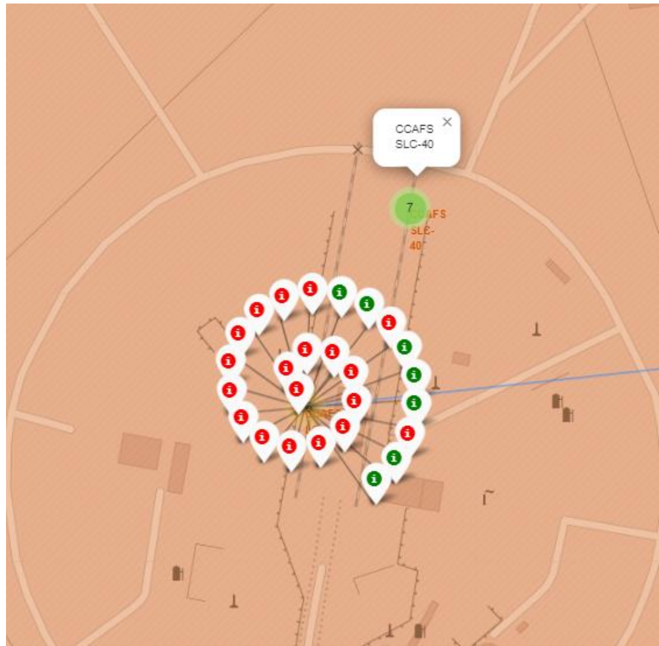
Markers displayed on a map indicating launch sites with color-coded labels.

- The first photo correspond to California
- The rest correspond to Florida
- Green markers means a success launch, while red markers means unsuccess launch


Vandenberg Space Force Base
Lompoc
VAFB
US
Jack and Laura Dangermond Preserve
Point Conception State Marine Reserve

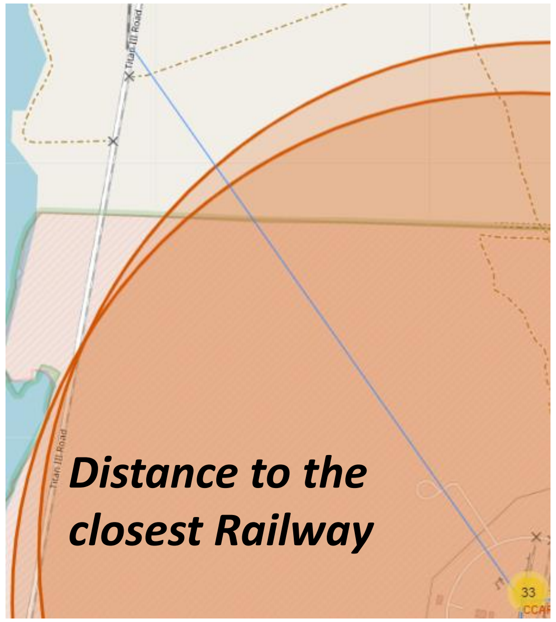


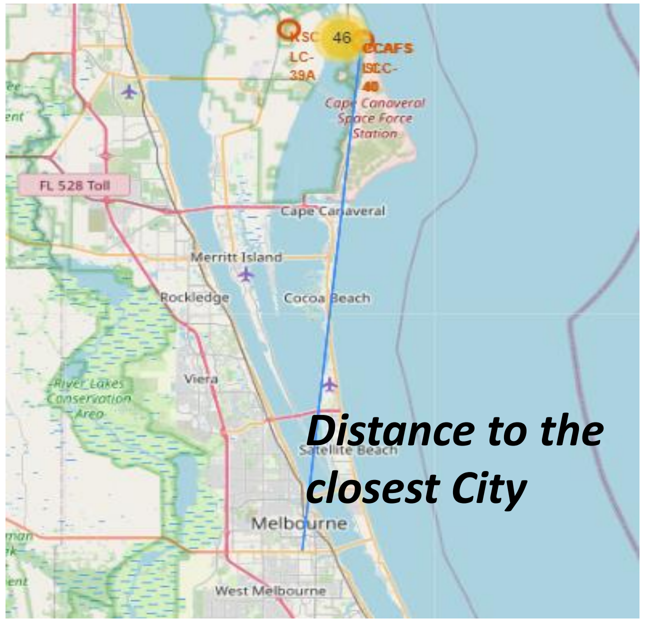


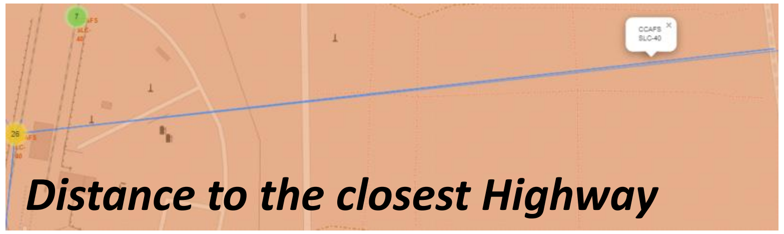


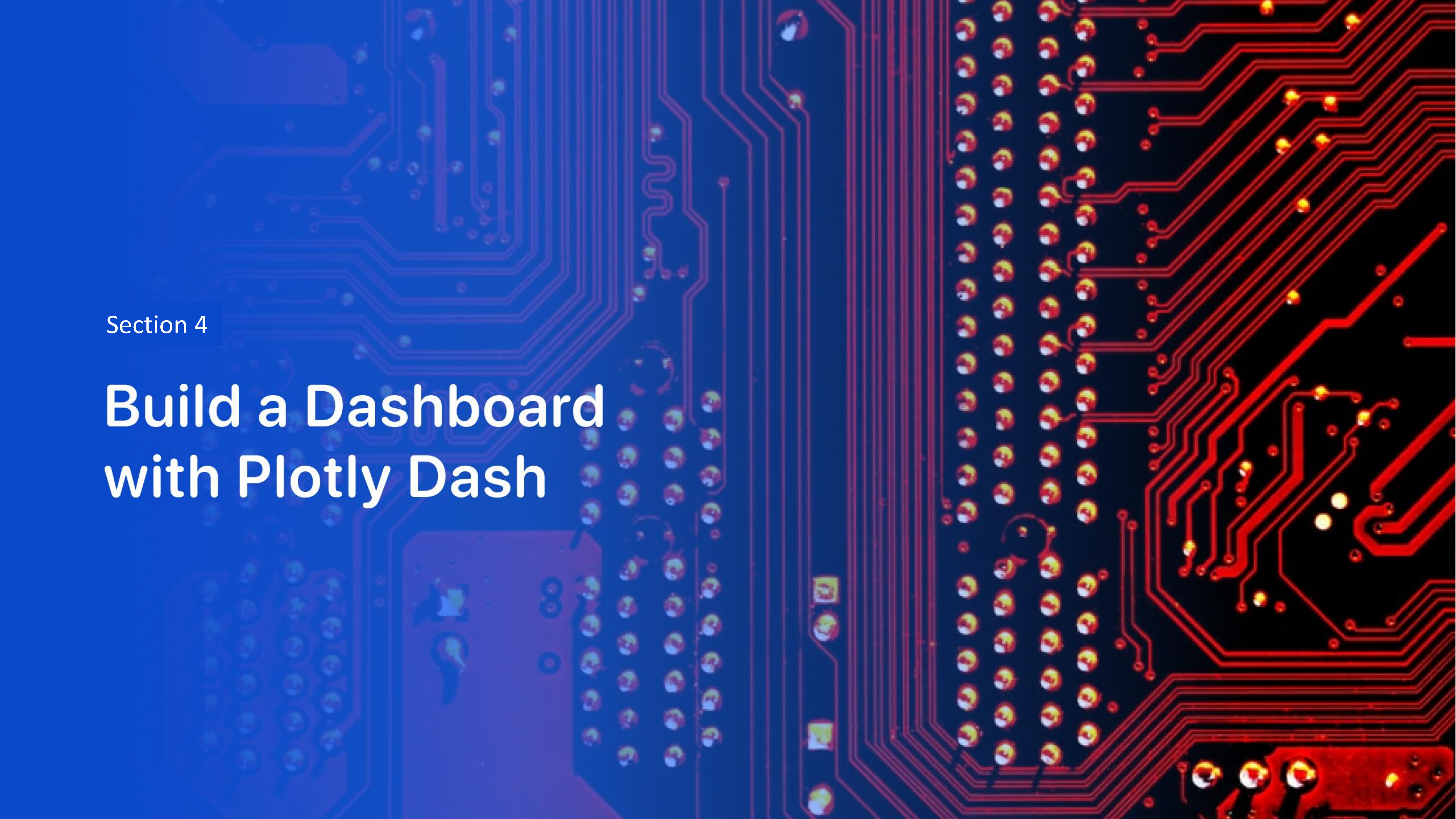
Launch Site distance to landmarks

- Are launch sites in close proximity to railways? Yes
- Are launch sites in close proximity to highways? Yes
- Are launch sites in close proximity to coastline? Yes
- Do launch sites keep certain distance away from cities? Yes, considerable distance







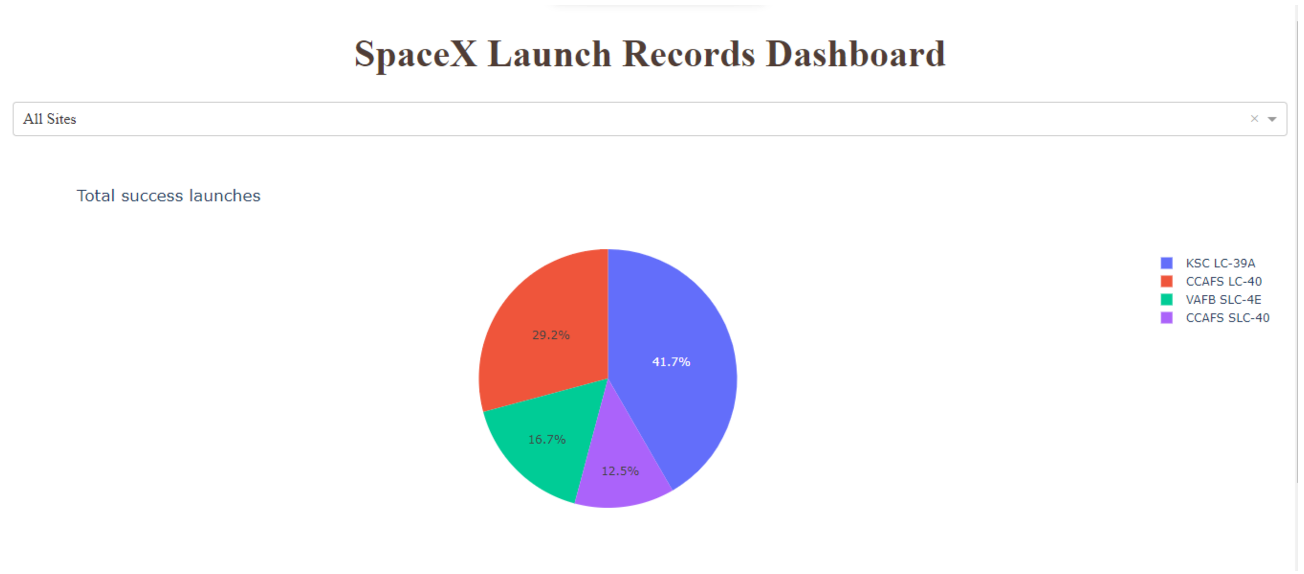


Section 4

Build a Dashboard with Plotly Dash

Total success rate

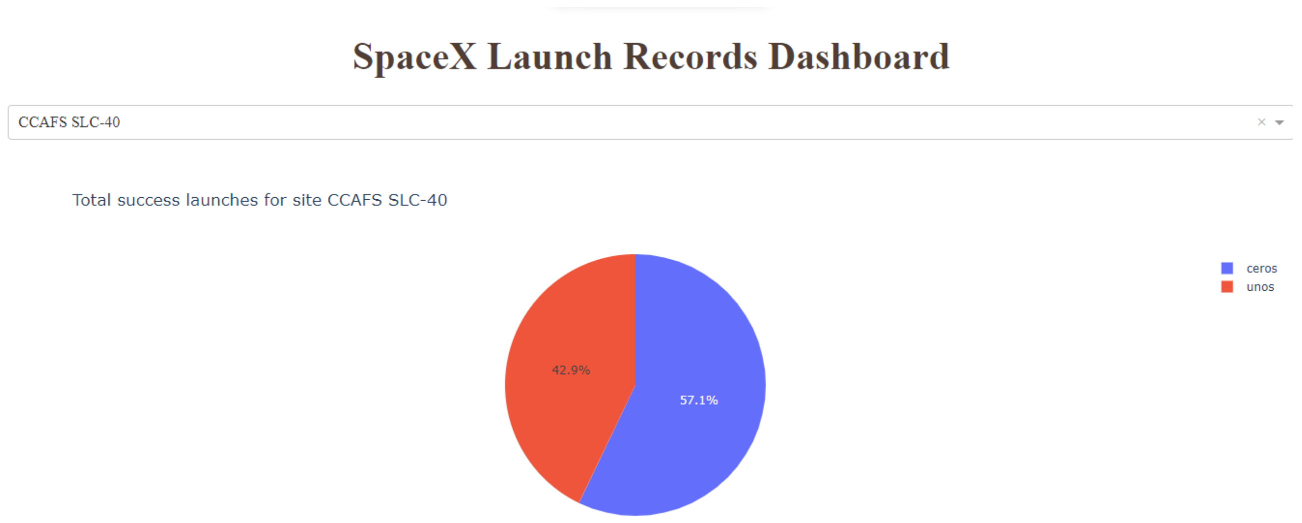
- As we can see KSC LC-39A had the largest success rate.
- And the smallest success rate is from CCAFS SLC-40


SpaceX Launch Records Dashboard

All Sites

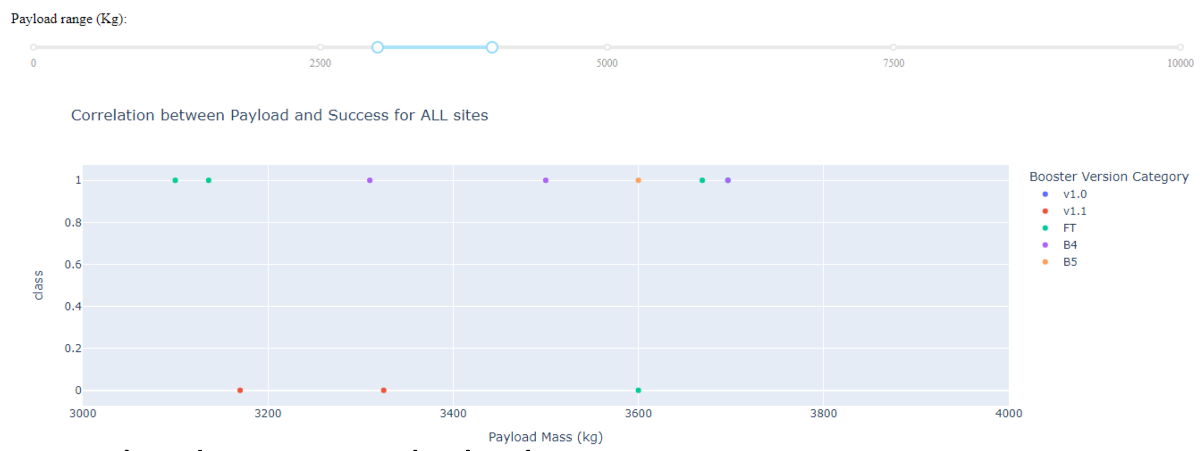
Total success launches for site CCAFS SLC-40

- We can see the CCAFS SLC-40 had the largest success rate .
- The success rate is 42.9%
- I strongly recommend that more missions be conducted at this launch site.

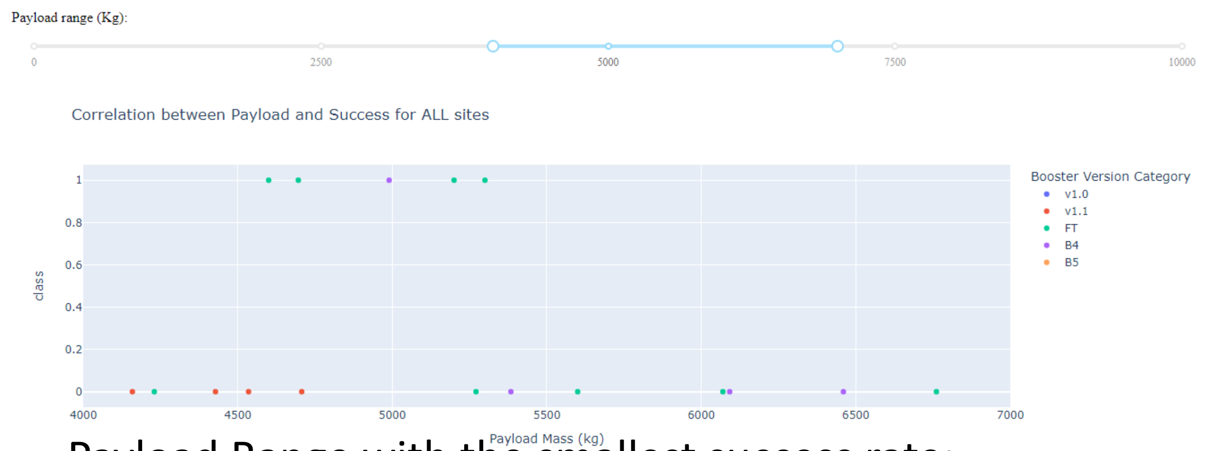

SpaceX Launch Records Dashboard

CCAFS SLC-40

Total success launches for site CCAFS SLC-40



Payload Range with the largest success rate: (3000-4000)



Payload Range with the smallest success rate: (4000-7000)

Success vs Unsuccess rate

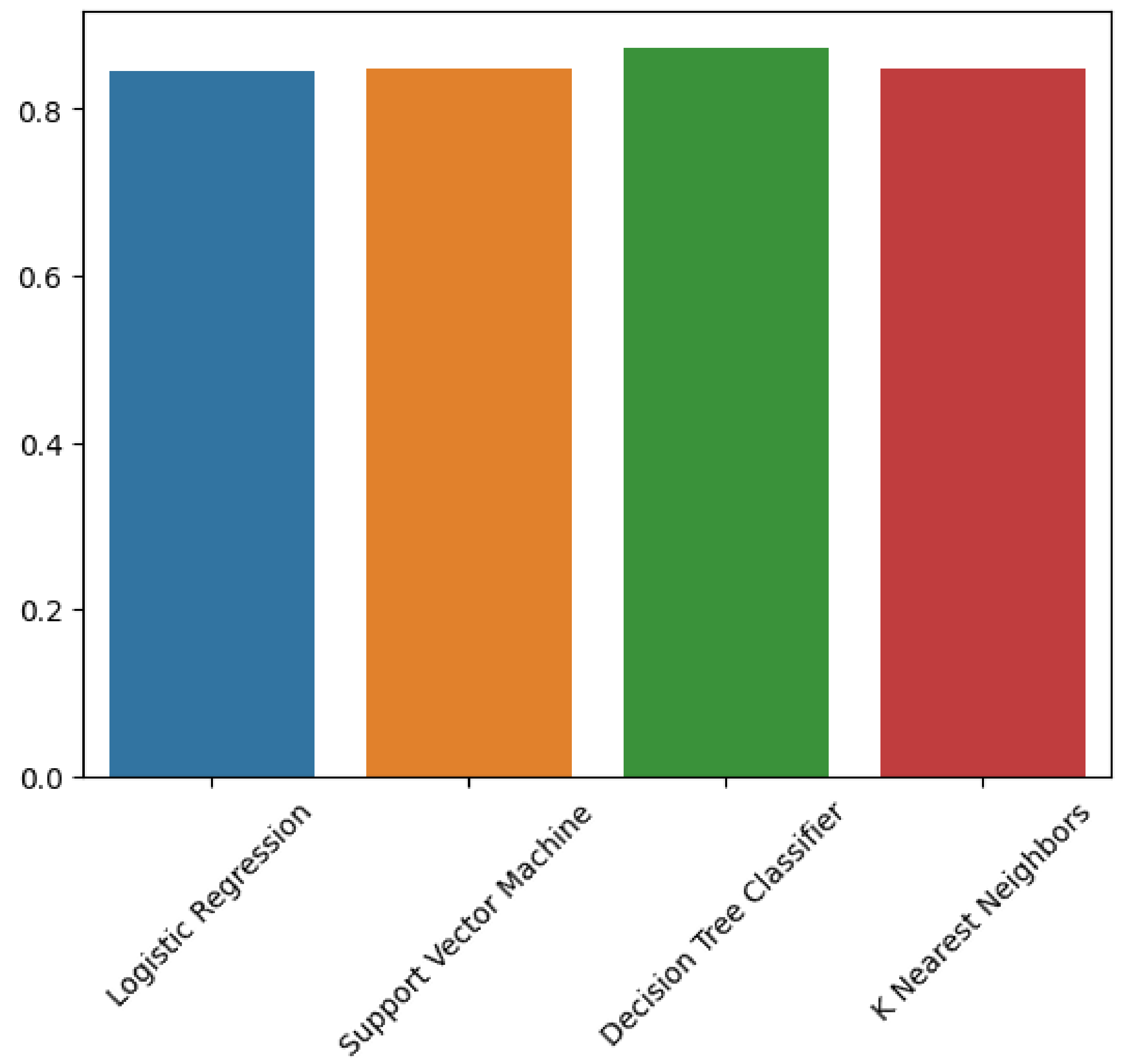
- As we can see the range with the largest success rate is between 3000 to 4000 Kg , with success rate of 70% .
- And the range with the smallest success rate is between 4000 to 7000 Kg, success rate of 29.4% .

Section 5

Predictive Analysis (Classification)

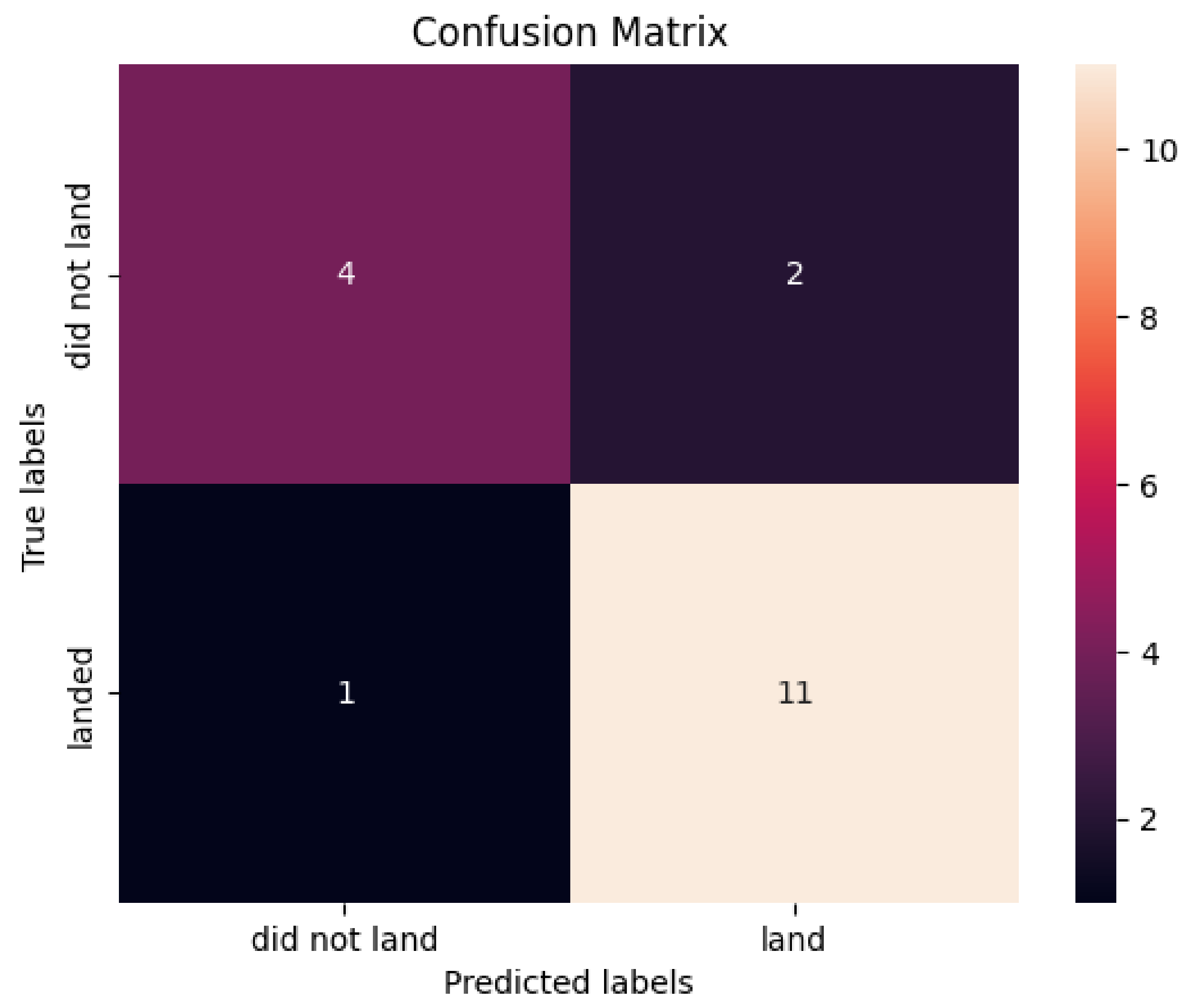
Classification Accuracy

- As we can see Decision Tree Classifier has the most accuracy model for predict in the cross validation.



Confusion Matrix

- As we can see the Decision Tree Classifier model hasta the most accuracy and F1 score.
- Predicted 4 correctly from the total (6) , and predicted 4 correctly from the total 5 predicted for the model. (class 1)
- Same case for the class 0.

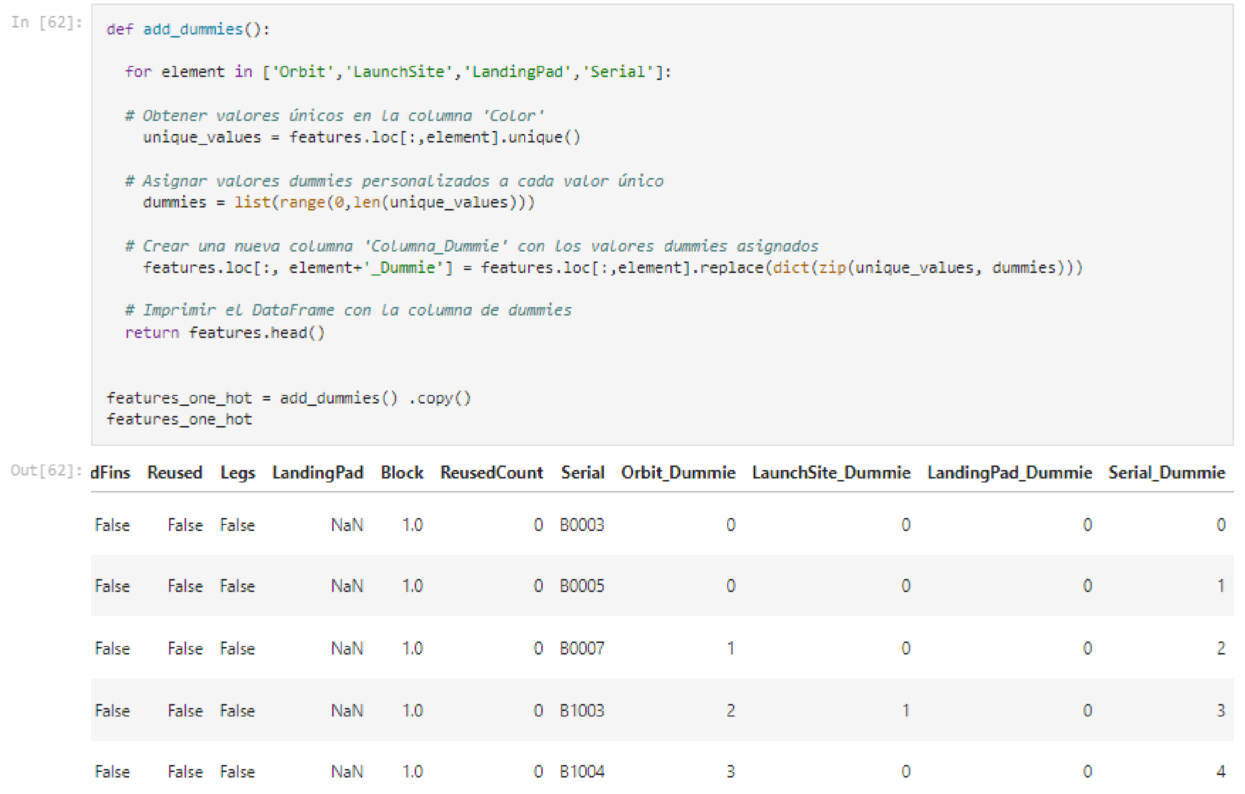


Conclusions

- Based on our analysis, we can draw the following conclusions:
  1. There is a positive correlation between the number of flights at a launch site and its success rate.
  2. The success rate of launches has shown a steady increase from 2013 to 2020.
  3. Orbits ES-L1, GEO, HEO, SSO, and VLEO have demonstrated the highest success rates.
  4. KSC LC-39A has the highest number of successful launches among all the sites.
  5. The Decision Tree Classifier proves to be the most effective machine learning algorithm for this particular task.

Appendix

I developed a function called "add_dummies()" which takes a categorical column as input. Let's assume there are 4 distinct values in the column. This function creates another column and adds it to the original one, assigning class labels 0, 1, 2, 3 to identify each unique value in the categorical column. It does this for any categorical columns that may exist. I am very proud of this code because it took me a long time to accomplish.



```
In [62]: def add_dummies():

           for element in ['Orbit','LaunchSite','LandingPad','Serial']:

           # Obtener valores únicos en la columna 'Color'
             unique_values = features.loc[:,element].unique()

           # Asignar valores dummies personalizados a cada valor único
             dummies = list(range(0,len(unique_values)))

           # Crear una nueva columna 'Columna_Dummie' con los valores dummies asignados
             features.loc[:, element+'_Dummie'] = features.loc[:,element].replace(dict(zip(unique_values, dummies)))

           # Imprimir el DataFrame con la columna de dummies
           return features.head()


         features_one_hot = add_dummies() .copy()
         features_one_hot
```

```
Out[62]: dFins  Reused  Legs   LandingPad  Block  ReusedCount  Serial  Orbit_Dummie  LaunchSite_Dummie  LandingPad_Dummie  Serial_Dummie
         False   False  False        NaN    1.0            0  B0003              0                  0                  0              0
         False   False  False        NaN    1.0            0  B0005              0                  0                  0              1
         False   False  False        NaN    1.0            0  B0007              1                  0                  0              2
         False   False  False        NaN    1.0            0  B1003              2                  1                  0              3
         False   False  False        NaN    1.0            0  B1004              3                  0                  0              4
```

Thank you!