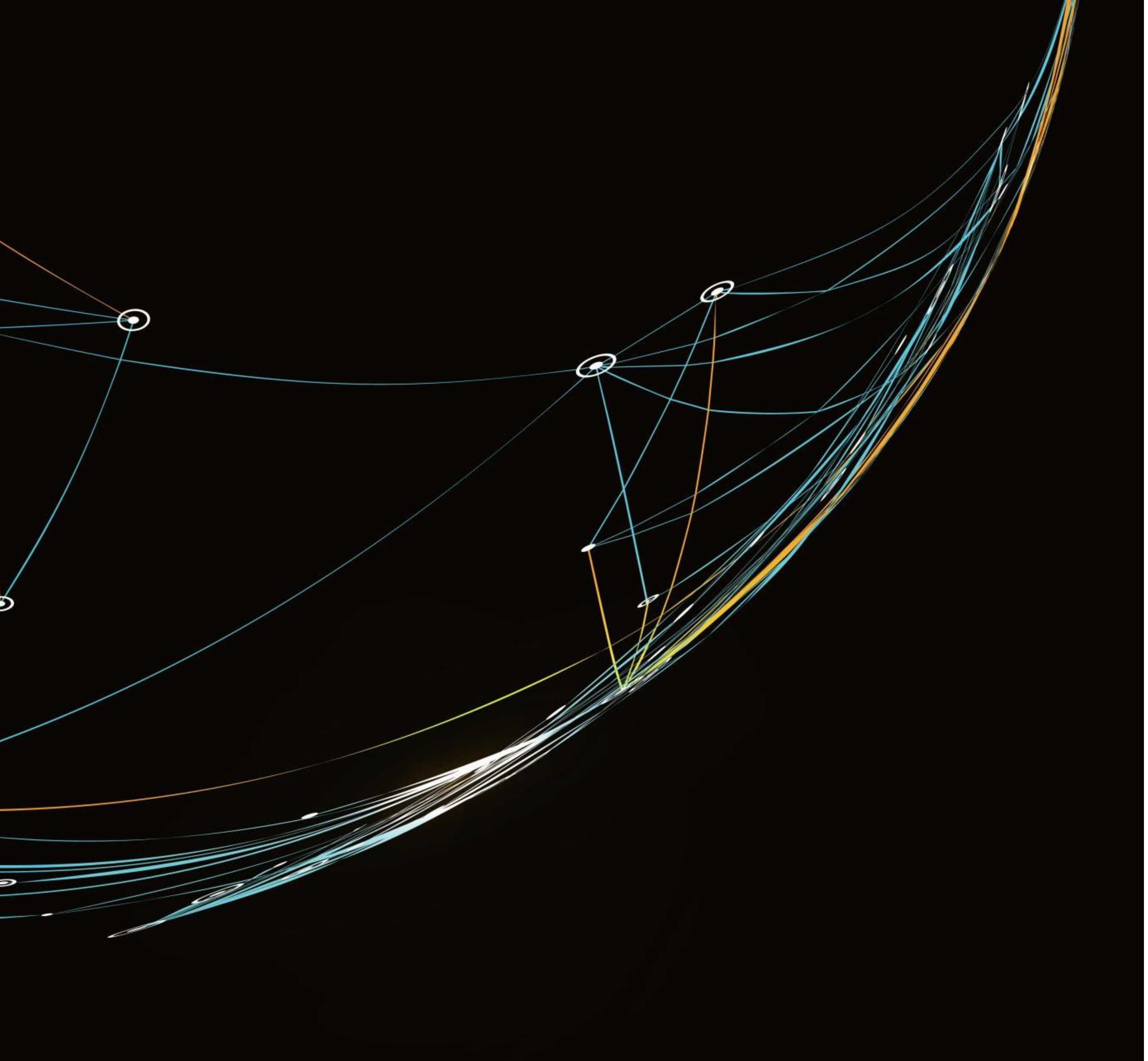
Predictive Analytics For SpaceX Falcon 9 First Stage Landings

Aditya Jarugumilli
07/07/2024



Outline

- Executive Summary
- Table of Contents
- Introduction
- Methodology
- Results
- Conclusion
- Appendix

Executive Summary

Summary of Methodologies:

- Data Collection API and Web Scraping
- Data Wrangling
- Exploratory Data Analysis (EDA) with SQL and Visualizations
- Interactive Visual Analytics with Folium
- Interactive Dashboard with Plotly Dash
- Machine Learning Predictive Analysis

Summary of Results:

- Determine Success Rate
- Exploratory Data Analysis (EDA) Results
- Interactive Geospatial Analytics Maps
- Interactive DashBoard Results
- Machine Learning Predictive Analysis for Classification Model Results

Table of Contents

- Introduction..5
- Methodology.. 6
- Data Collection..8
- Data Collection – SpaceX API..9
- Data Collection – Web Scraping..10
- Data Wrangling..1 1
- EDA with DATA Visualizations..13
- EDA with SQL..14
- Interactive Map with Folium..15
- Interactive DashBoard with Plotly Dash..16
- Predictive Analysis(Classification)..17
- Results..18
- Exploratory Data Analysis with Visualizations Results..19
- Exploratory Data Analysis with SQL Results..26
- Interactive Geospatial Analytics Maps..37
- Interactive DashBoard Results..41
- Machine Learning Predictive Analysis for Classification Model Results..45
- Conclusion..48
- Appendix..49

Introduction

Project background and Context:

SpaceX has been at the forefront of space exploration, achieving remarkable milestones with its Falcon 9 rockets which are known for its reliability, versatility and cost efficiency, primarily due to its reusability. A critical aspect of these missions is the successful landing of the Falcon 9 first stage, which significantly reduces costs by enabling the reuse of rockets. Understanding the factors that influence the success of these landings is crucial for enhancing mission reliability and efficiency.

Problem Statement:

The primary goal of this project is to analyze and predict the successful landing of the Falcon 9 first stage. By leveraging historical launch data, web scraped information and machine learning techniques, we determine what factors contribute to the successful landing of the Falcon 9 at first stage and how can we predict the outcome of future landings using data-driven models.

Methodology

- Data Collection – SpaceX API
- Data Collection - Web Scraping
- Data Wrangling
- Exploratory Data Analysis (EDA) with SQL
- Exploratory Data Analysis (EDA) with Data Visualizations
- Interactive Visual Analytics with Folium
- Interactive Dashboard with Plotly Dash
- Machine Learning Predictive Analysis

Methodology

Executive Summary

- Data collection methodology:

  Data was collected through the SpaceX API for detailed launch information, complemented by web scraping from Wikipedia.

- Perform data wrangling:

  Data wrangling involves cleaning the dataset, handling missing values and creating a target variable to indicate successful or unsuccessful landings.

- Perform exploratory data analysis (EDA) using visualization and SQL:

  Using SQL queries to extract and analyze unique launch sites, with visualizations created to present the findings clearly.

- Perform interactive visual analytics using Folium and Plotly Dash:

  Using Folium to create interactive maps to visualize launch site locations, while Plotly Dash created to build a detailed dashboard to visualize launch records interactively.

- Perform predictive analysis using classification models:

  Using Logistic Regression, Support Vector Machine, Decision Trees and K-Nearest Neighbors, then identify best performing model through cross validation and accuracy metrics.

Data Collection

API Data Collection:

Data on SpaceX launches was collected using the SpaceX REST API. This includes detailed information about launch dates, payloads, launch sites and landing outcomes. The API provided a structure and reliable source of historical launch data.

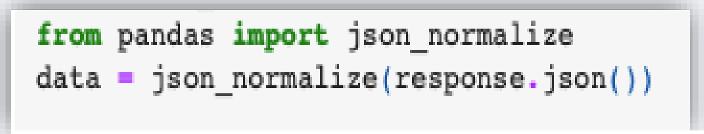
Web Scraping:

Additional data was scraped from Wikipedia to supplement the API data. This involves extracting information about historical launches, payload specifics and other relevant details.

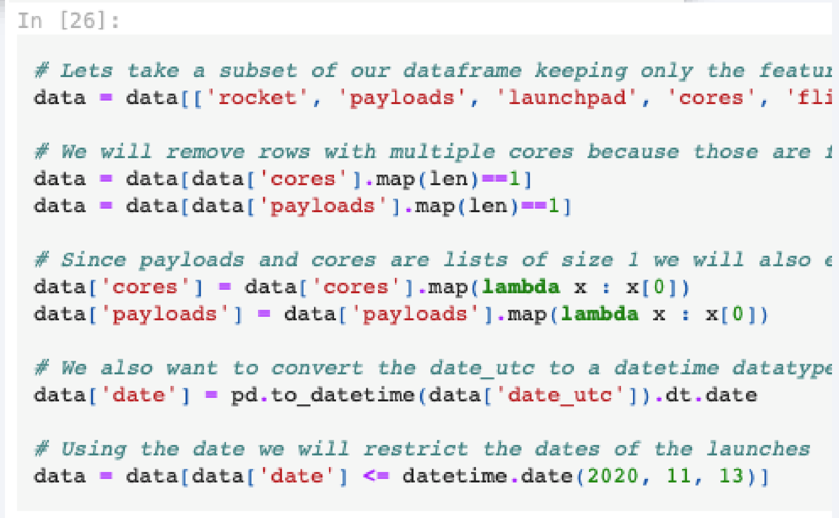
Data Collection – SpaceX API

- Making a GET request and parsing the SpaceX launch data from the API to successfully launching 200 status response code.
- Decoding the response content using .json() and turning it into pandas DataFrame using .json_normalize()
- Requesting custom functions to retrieve the data and store it in lists.
- Using these lists as values in dictionary and construct the dataset.
- Creating a Pandas DataFrame from the dictionary.
- Filter the DataFrame to only include Falcon 9 launches using BoosterVersion column.
- Reset the FlightNumber column.
- Replace missing values pf Payload Mass column with calculated .mean()
- Export the DataFrame to "csv" file.

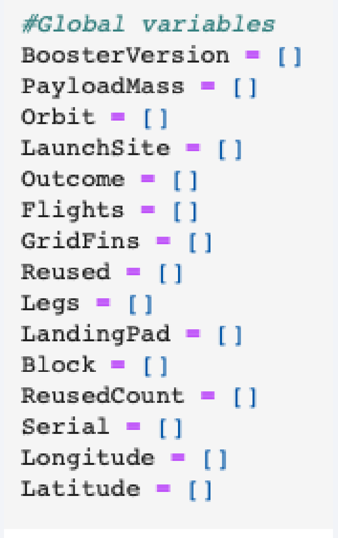
[GitHub URL: Data Collection API (Click Here)]



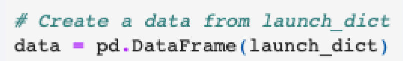
```
from pandas import json_normalize
data = json_normalize(response.json())
```



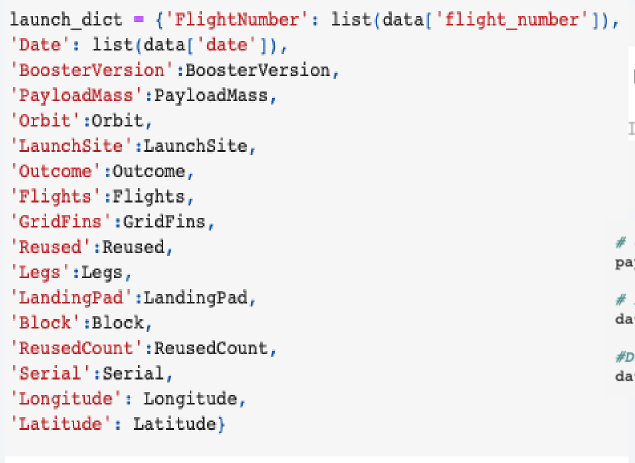
```
In [26]:

# Lets take a subset of our dataframe keeping only the featu
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'fli

# We will remove rows with multiple cores because those are 
data = data[data['cores'].map(len)==1]
data = data[data['payloads'].map(len)==1]

# Since payloads and cores are lists of size 1 we will also 
data['cores'] = data['cores'].map(lambda x : x[0])
data['payloads'] = data['payloads'].map(lambda x : x[0])

# We also want to convert the date_utc to a datetime datatype
data['date'] = pd.to_datetime(data['date_utc']).dt.date

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



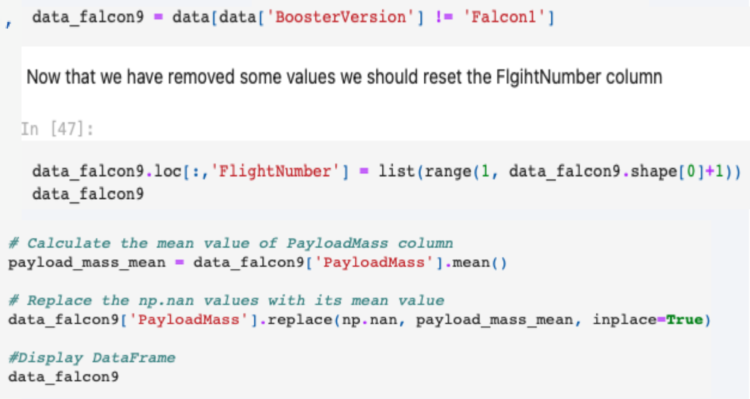
```
#Global variables
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
ReusedCount = []
Serial = []
Longitude = []
Latitude = []
```



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



```
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}
```



```
data_falcon9 = data[data['BoosterVersion'] != 'Falcon1']
```

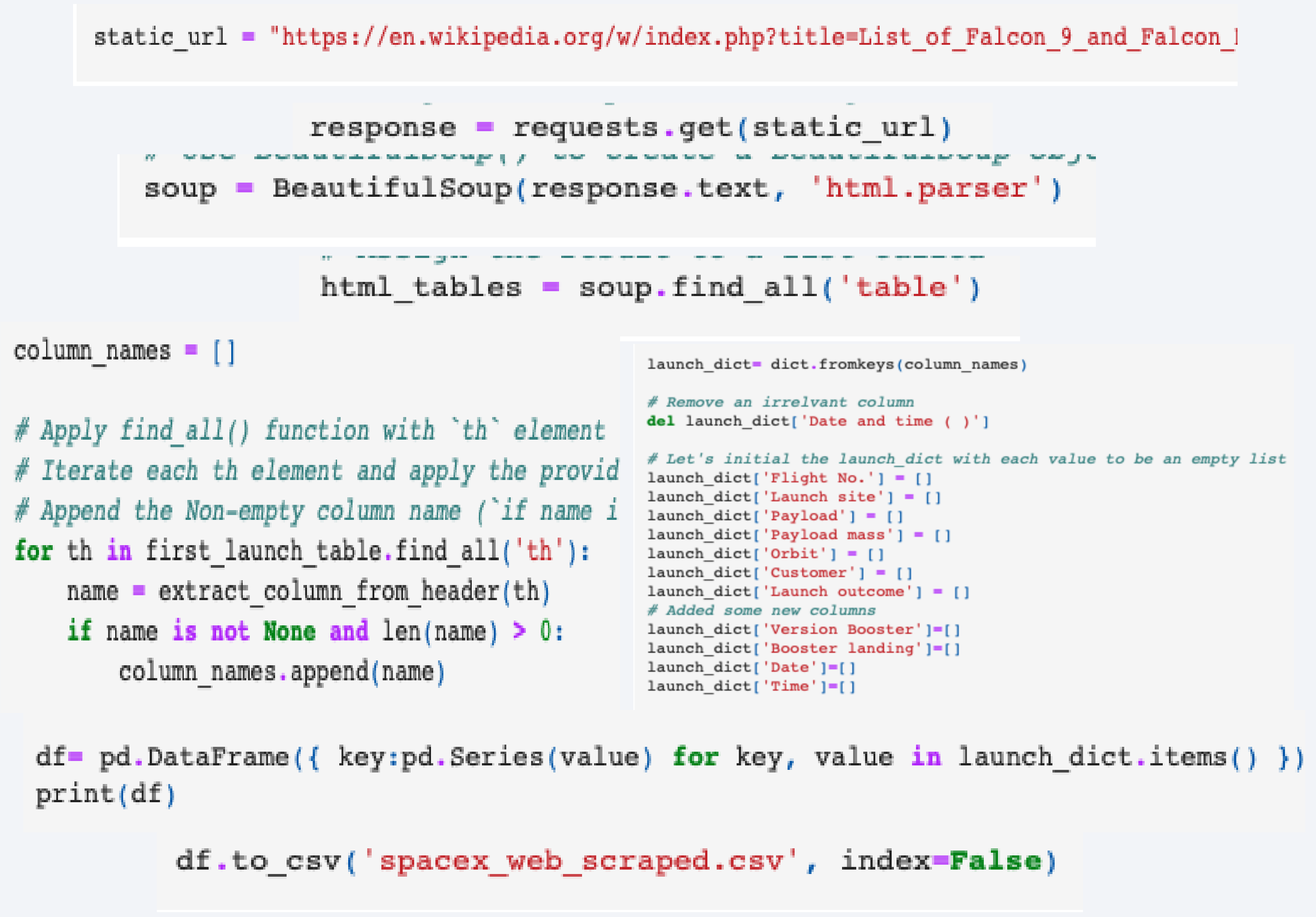
Now that we have removed some values we should reset the FlgihtNumber column

```
In [47]:

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

# 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)

#Display DataFrame
data_falcon9
```

Data Collection – Web Scraping

- Requesting the Falcon 9 Launch HTML page with provided static_url using requests.get() and assign the response to an object.
- Creating a BeautifulSoup object from HTML response.
- Extract all column names from the HTML table header and assign the result to list called html_tables.
- Creating a DataFrame by parsing the launch HTML Tables.
- Export the DataFrame to "csv" file.

GitHub URL: Data Collection Web Scraping (Click Here)



```
static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_
```

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

```
soup = BeautifulSoup(response.text, 'html.parser')
```

```
html_tables = soup.find_all('table')
```

```
column_names = []

# Apply find_all() function with `th` element
# Iterate each th element and apply the provid
# Append the Non-empty column name (`if name i
for th in first_launch_table.find_all('th'):
    name = extract_column_from_header(th)
    if name is not None and len(name) > 0:
        column_names.append(name)
```

```
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']=[]
```

```
df= pd.DataFrame({ key:pd.Series(value) for key, value in launch_dict.items() })
print(df)
```

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

Data Wrangling

- In the Data set, there are several different cases where the booster did not land successfully. The landing was attempted but outcome was failed due to several reasons such as:

  True Ocean – means the mission outcome was successfully landed to a specific region.

  False Ocean – means the mission outcome was unsuccessfully landed to a specific region.

  True RTLS - means the mission outcome was successfully landed to a ground pad.

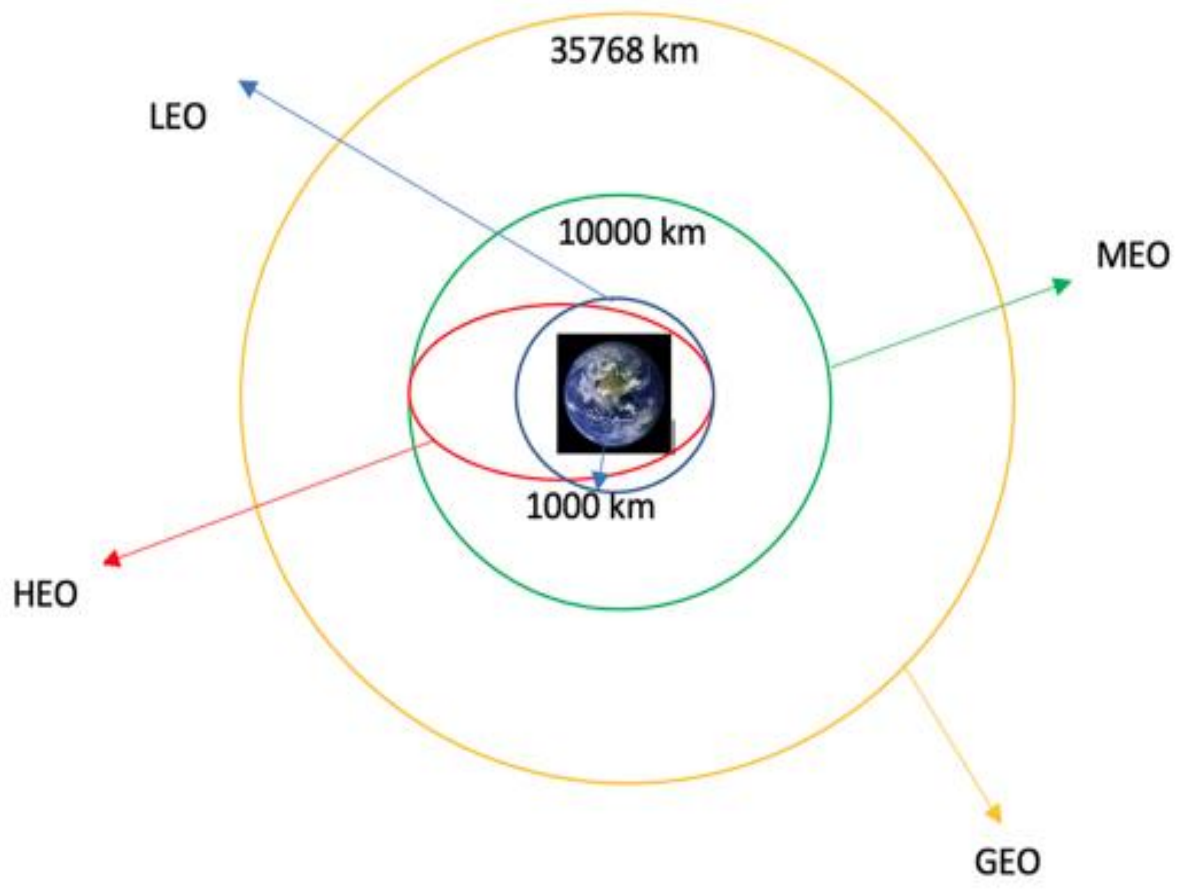
  False RTLS – means the mission outcome was unsuccessfully landed to a ground pad.

  True ASDS – means the mission outcome was successfully landed on a drone ship.

  False ASDS – means the mission outcome was unsuccessfully landed on a drone ship.

  None ASDS and None None – represent the failure to land.

- We mainly convert these outcomes into Training labels with "1" means the booster successfully landed and "0" means the booster was unsuccessful.



[GitHub URL: Data Wrangling (Click Here)]

Data Wrangling

- We first Perform the Exploratory Data Analysis (EDA) and determine the Training labels.
- We then Calculate the number of launches on each site, the number and occurrence of each orbit, the number and occurrence of mission outcome of the orbits using the method value_counts()
- Each launch aims to a dedicated orbit, and here are some orbit types shown on the image.
- Using the Outcome, create a list where element is "0" if corresponding row in Outcome is in the set bad_outcome, otherwise it's "1".
- Assigning to the variable "landing_class".
- Export the Data to "csv" file.

EDA with Data Visualization

Scatter Plots

Scatter Plots are used for visualizing the relationship between:

- Flight Number vs Launch Site
- Payload Mass vs Launch Site
- Flight Number vs Orbit Type
- Payload Mass vs Orbit Type

Scatter Plots are ideal for identifying correlations and patterns between two continuous variables.

Bar Charts

Bar Charts are used for visualizing the relationship between:

- Success rate vs Orbit Type

Bar Charts are effective for comparing the frequency of categorical data.

Line Charts

Line Charts are used for visualizing the Launch success yearly trend.

Line charts are for visualizing trends and changes over time, providing insights into how launch success rate have evolved.

GitHub URL: Exploratory Data Analysis with Visualizations (Click Here)

EDA with SQL

The SQL Queries performed on the dataset were used to:

- Displaying the names of the unique launch sites in space mission.
- Displaying 5 records where launch sites begin with the string “CCA”.
- Displaying the total payload mass carried by boosters launched by NASA (CRS).
- Displaying the average payload mass carried by booster version F9 v1.1
- Listing the date when the first successful landing outcomes in ground pad was achieved.
- Listing the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000.
- Listing the total number of successful and failure mission outcomes.
- Listing the names of the booster versions which have carried the maximum payload mass.
- Listing the failed landing outcomes on drone ship, booster versions and launch site for the months in year 2015.
- Ranking 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.

GitHub URL: Exploratory Data Analysis with SQL (Click Here)

Build an Interactive Map with Folium

For Building an Interactive Map with Folium to visualize the launch data the following tasks were performed:

- Marking all the launch sites on a map

  1. Creating a folium Map object, with an initial center location to be NASA Johnson Space Center.
  2. Creating and adding folium.Circle and folium.Marker for each launch site on the map.

- Marking the success/failed launches for each site on the map

  1. Creating markers for all launch records, assigning marker color for successful(class=1) as green marker, and failed(class=0) as red marker.
  2. For each launch, adding a folium.Marker to marker_cluster.

- Calculating the distances between a launch site to its proximities

  1. Adding a MousePosition on the map to get coordinate for a mouse over a point on a map.
  2. Marking down a point on the closest coastline using MousePosition to calculate the distance.
  3. Adding a folium.Marker to show the distance between the coastline and launch site.
  4. Creating a folium.Ployline to display the distance between the two points.

GitHub URL: Interactive Visual Analytics with Folium (Click Here)

Build a Dashboard with Plotly Dash

The following Tasks were performed to build a Dashboard:

- Added a Launch Site dropdown list to select specific launch site.
- Added a Pie Chart showing the total successful launches for all sites and the Success vs Failure ratio for each site.
- Added a Slider to Select the Payload range.
- Added a Scatter Plot to show the correlation between Payload and Outcome being successful or unsuccessful.

GitHub URL: Interactive Dashboard with Plotly Dash (Click Here)

Predictive Analysis (Classification)

Creating a NumPy array from the column "Class" in "data".

Standardize the data in "X" and reassigning it using "StandardScaler".

Using the "train_test_split" function to split the data "X" and "Y" into training and testing the data.

Creating "GridSearchCV" object to find the best "parameters".

Applying "GridSeachCV" on Logistic Regression, Support Vector Machine, Decision Tree Classifier, K Nearest Neighbor models.

Calculating the accuracy on the test data using the "score" method on all four models.

Reviewing the Confusion Matrix for all four models and finding the best performing model.

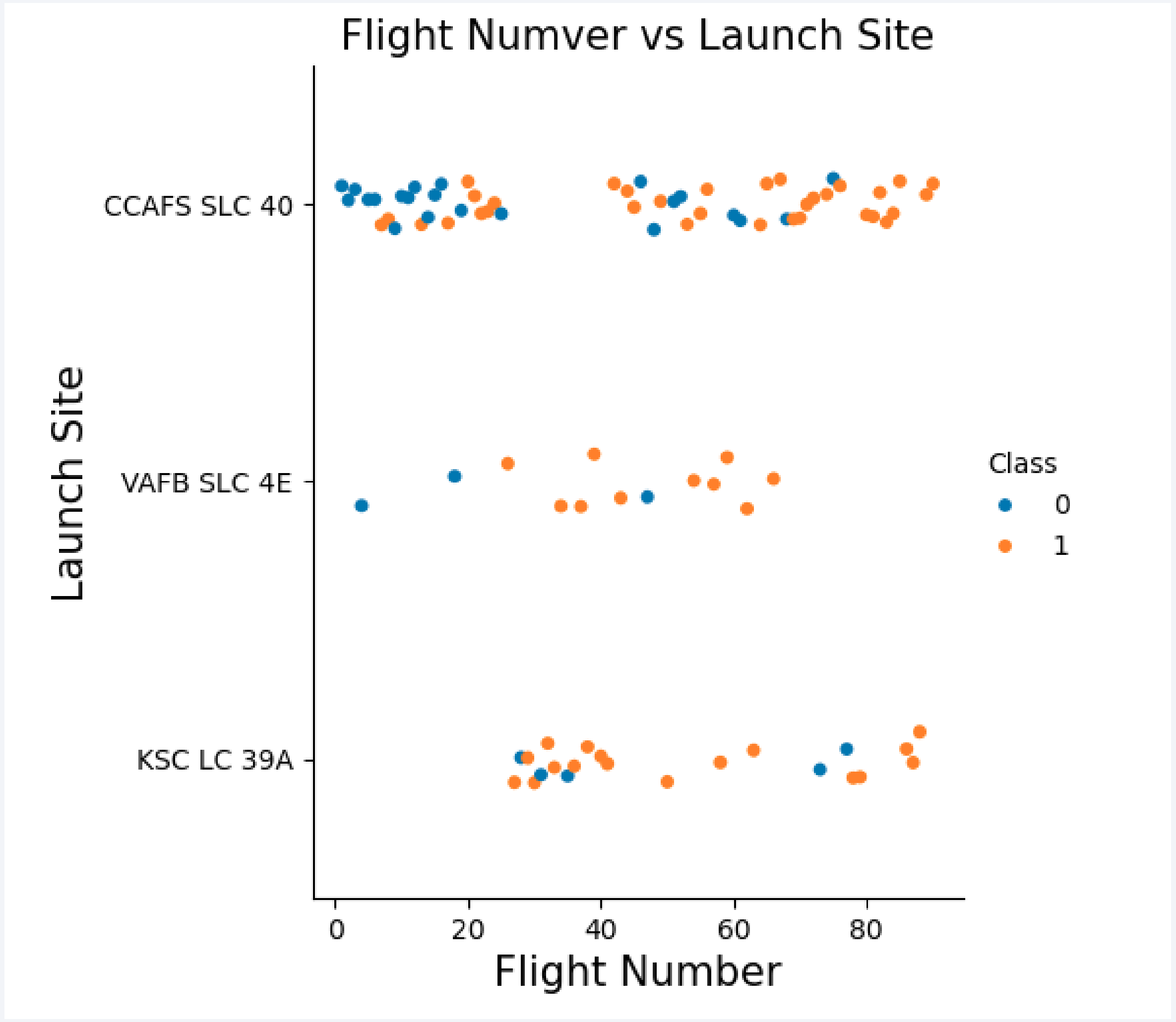
[GitHub URL: Machine Learning Predictive Analysis (Click Here)]

Results

- Exploratory Data Analysis (EDA) Results
- Interactive Geospatial Analytics Maps
- Interactive DashBoard Results
- Machine Learning Predictive Analysis for Classification Model Results

Exploratory Data Analysis (EDA) with Visualizations

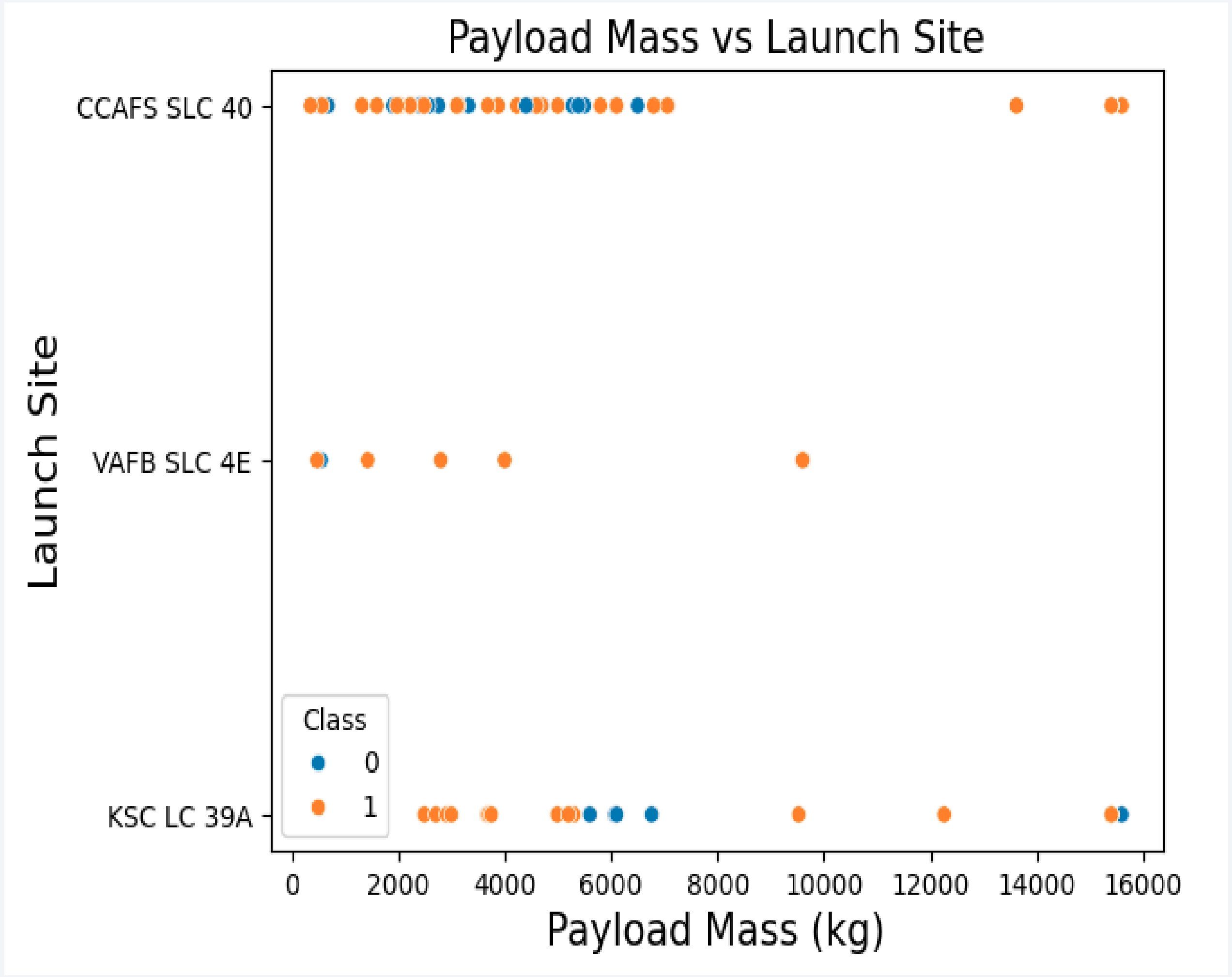
Flight Number vs. Launch Site



The following observations are made from the plot:

- The earliest flights mostly failed while the later flights all succeeded.
- The flights from “CCAFS SLC 40” site shows that most of the flights that were launched early(flight numbers before 20) were unsuccessful.
- The flights from “VAFB SLC 4E” site also shows that flights launch early were unsuccessful.
- The flights from “KSC LC 39A” site shows that there were no early launches, hence are more successful.
- From three sites it is evident that flight numbers > 20 had better successful.

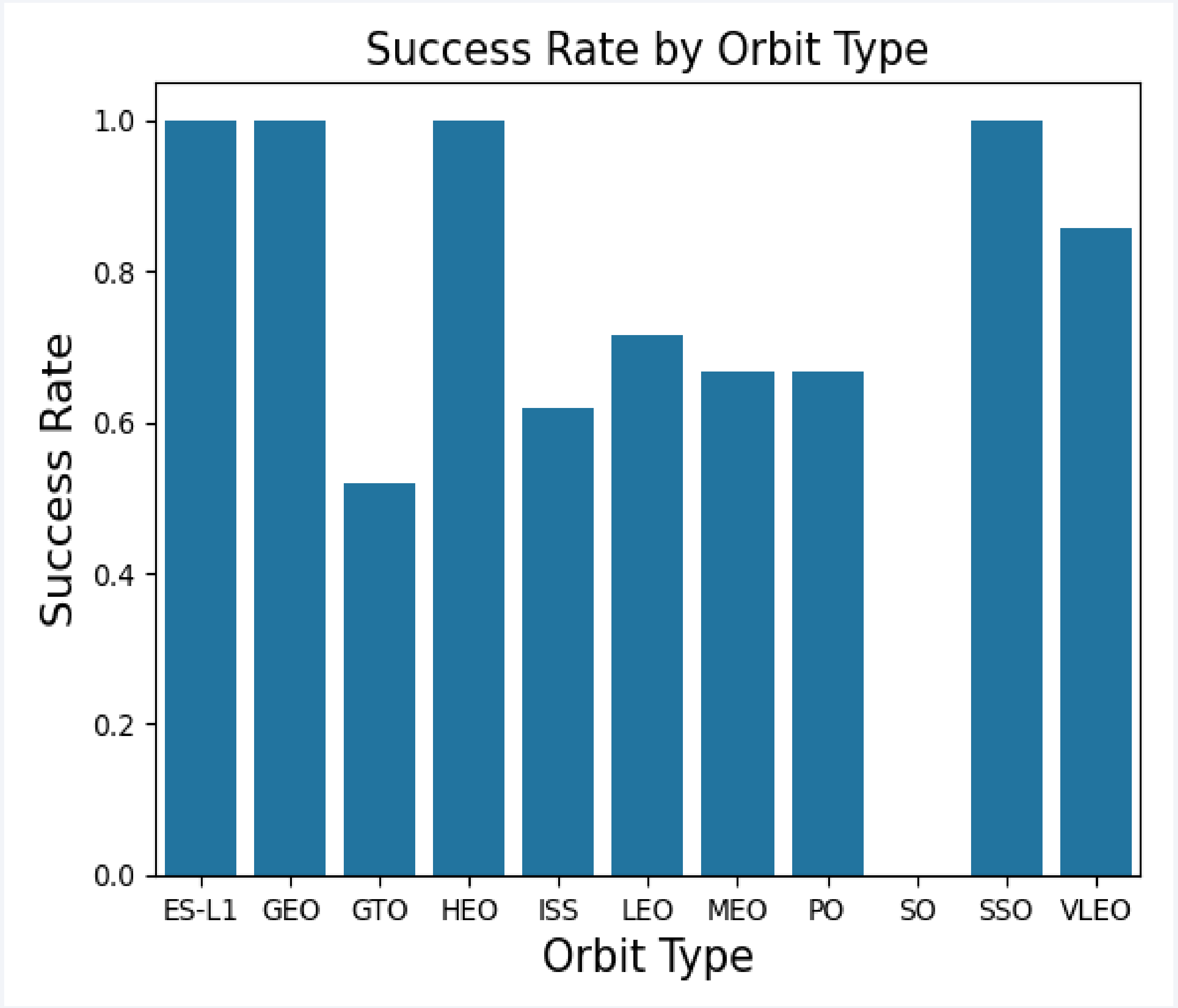
Payload Mass vs. Launch Site



The following observations are made from the plot:

- It is evident that for Payload Mass above 8000 kg, there are very less unsuccessful landings provided there is no much launches for the Payloads above 8000.
- All three sites had different payload masses with most of the launches from “CCAFS SLC 40” site being comparatively lesser payloads.

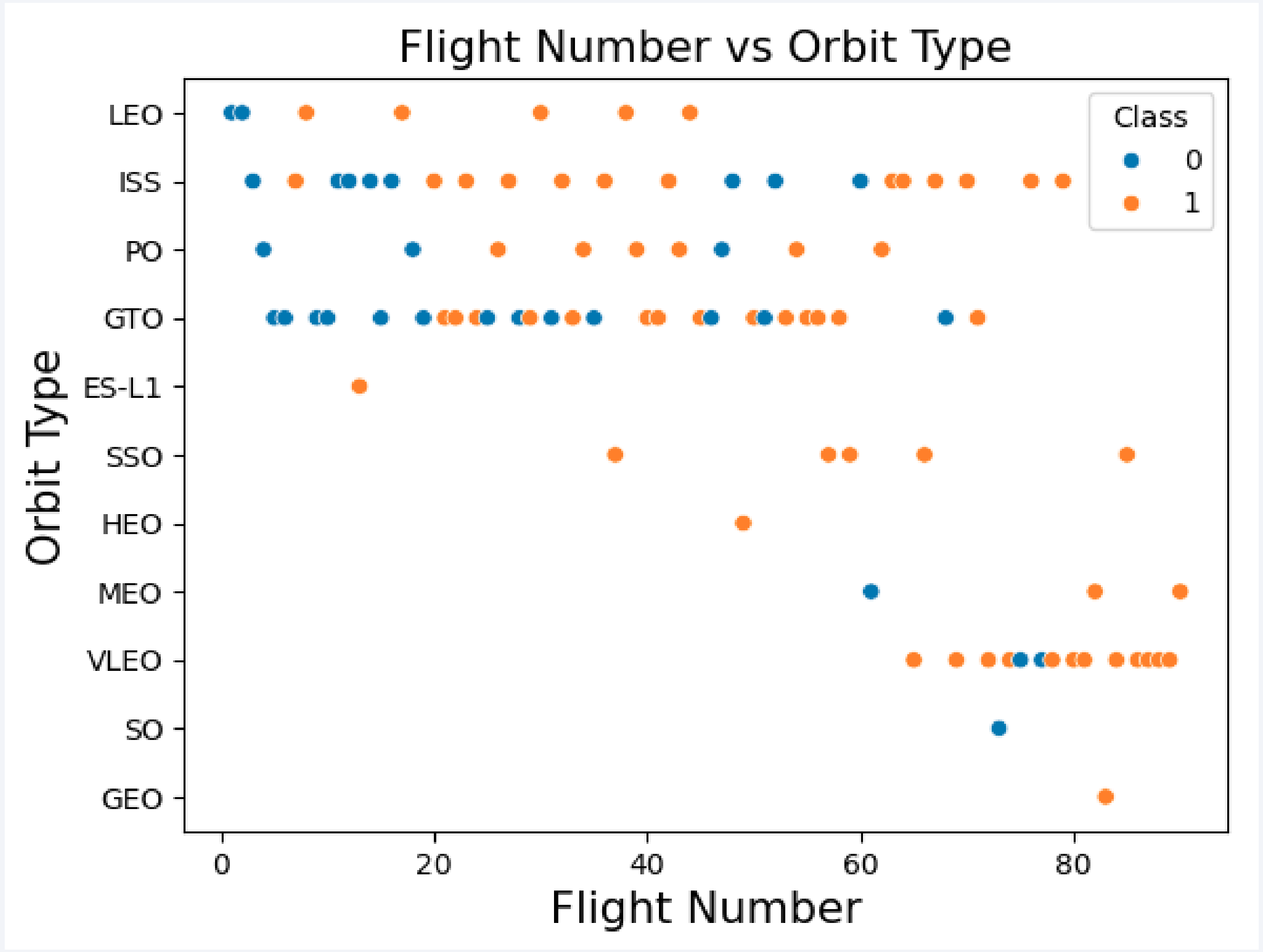
Success Rate vs. Orbit Type



The following observations are made from the Bar Chart:

- Orbits with 100% success rate:
  -> ES-L1, GEO, HEO, SSO
- Orbits with 45% to 90% success rate:
  -> GTO, ISS, LEO, MEO, PO, VLEO
- ORBITS with 0% success rate:
  -> SO

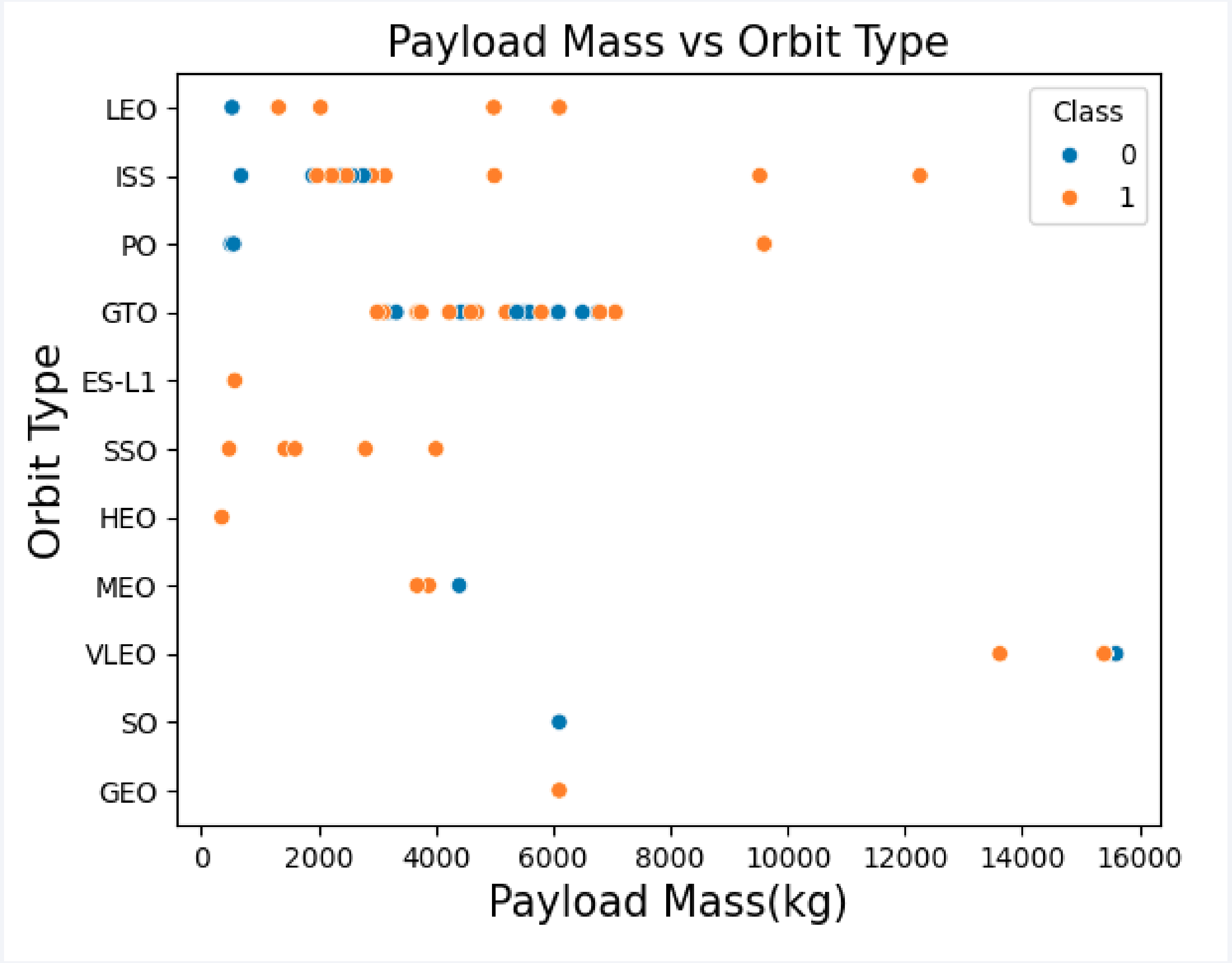
Flight Number vs. Orbit Type



The following observations are made from the plot:

- Orbit “SSO” has 100% success rate with 5 successful flights.
- Orbit “GEO, HEO, ES-L1” has 100% success rate with 1 successful flight.
- Orbit “SO” has 0% success rate with only 1 flight.
- For most orbits as the flight numbers kept increasing the success rate kept increasing.

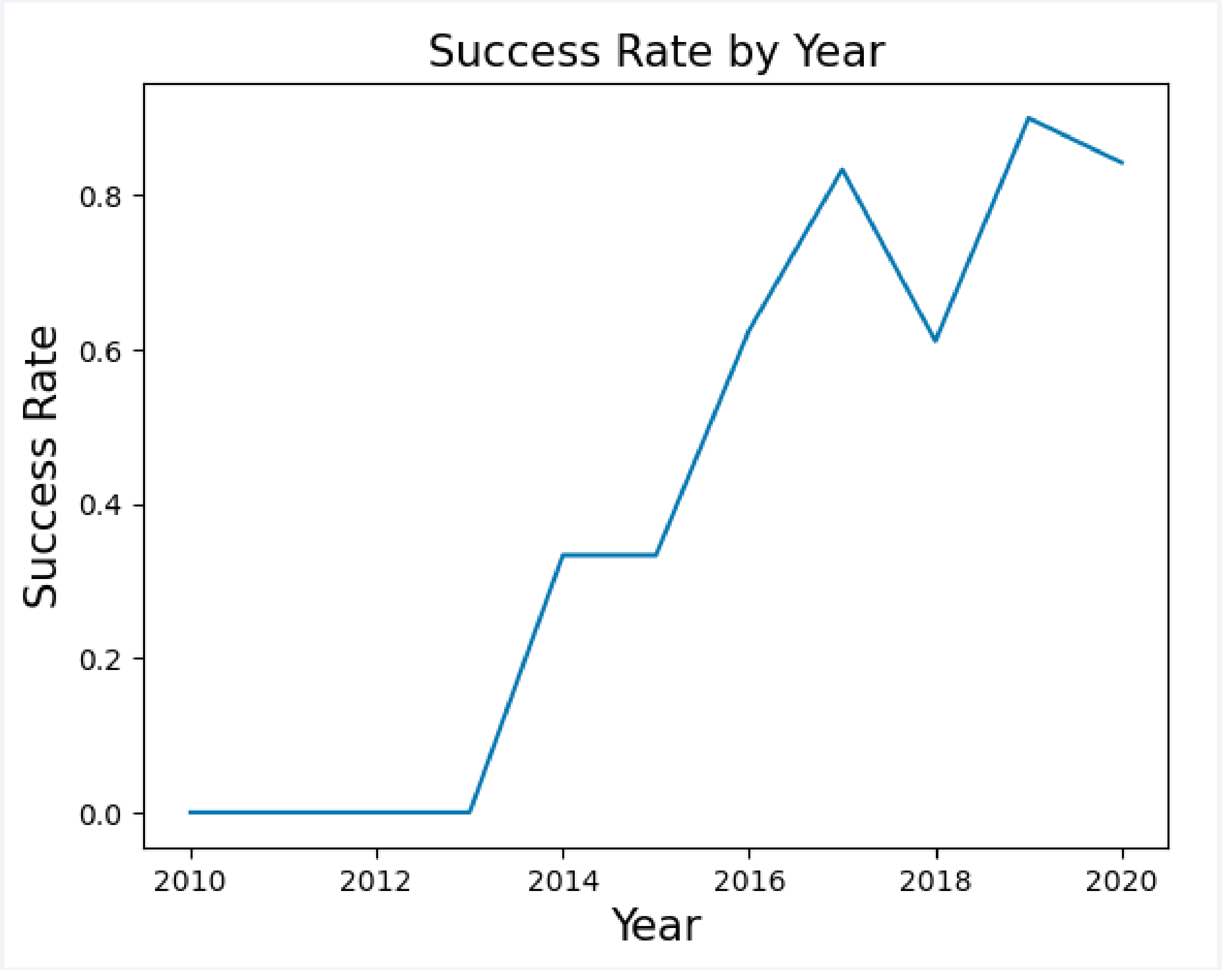
Payload vs. Orbit Type



The following observations are made from the plot:

- Orbit Type “SSO, HEO, ES-L1 and GEO” has 100% success rate with lesser Payload mass.
- Orbit Type having success rate with heavy Payload Mass are “ ISS, PO and VLEO”.
- Orbit Type “SO” has 0% success rate.
- For most Orbit Types it is evident that it was successful with lesser payload mass.

Launch Success Yearly Trend

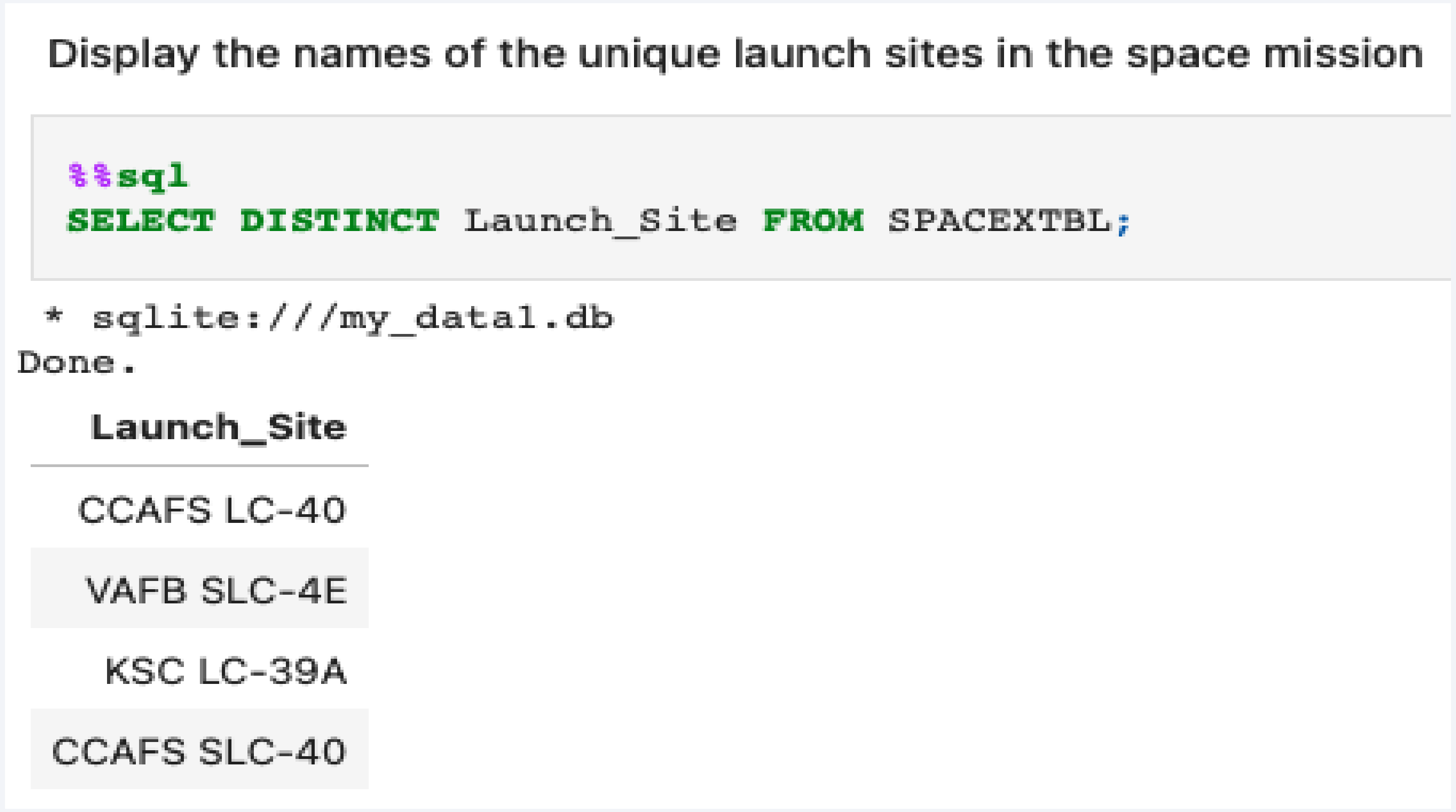


The following observations are made from the plot:

- Between years 2010 to 2013 all landings were unsuccessful.
- From 2013 the success rate kept increasing till 2020.

Exploratory Data Analysis (EDA) with SQL

All Launch Site Names


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

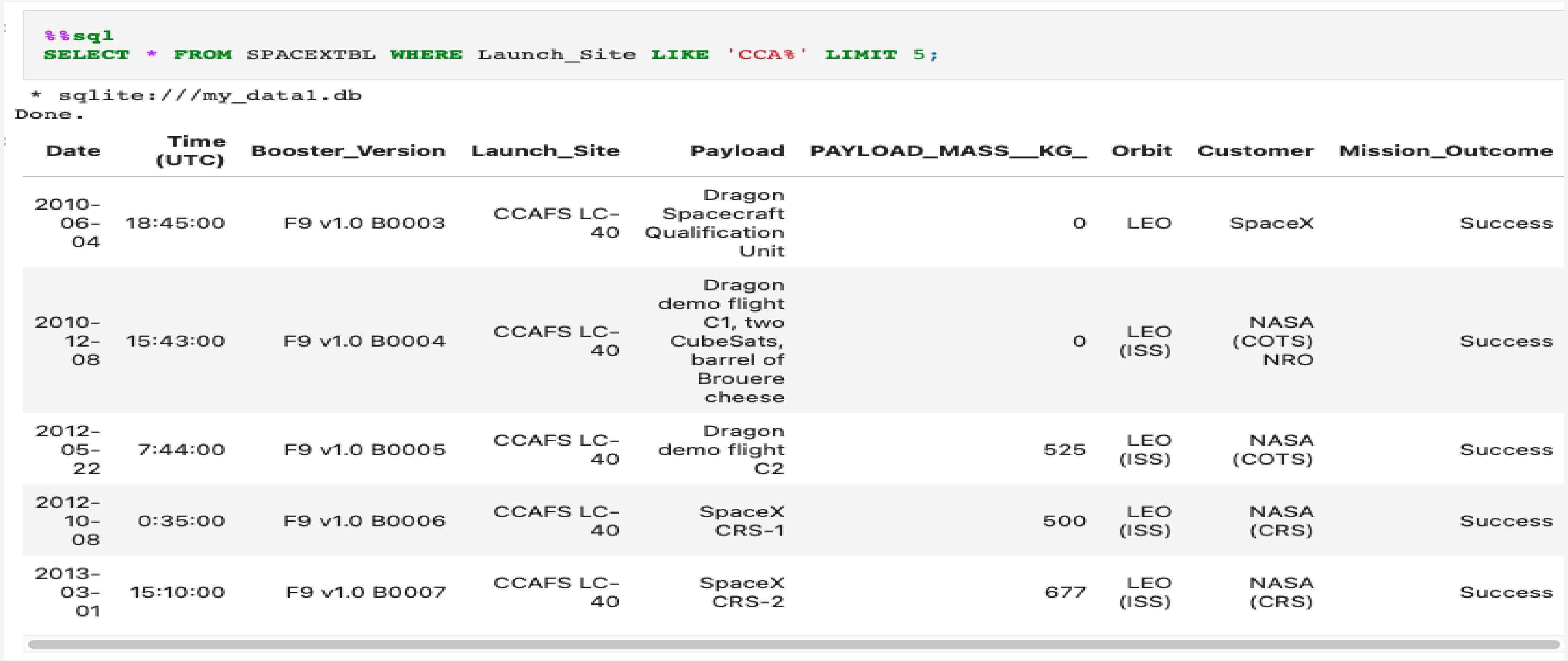
```
%%sql
SELECT DISTINCT Launch_Site FROM SPACEXTBL;
```

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

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

“DISTINCT” displays unique launch sites in SpaceX space mission from the “Launch_Site” from “SPACEXTBL”.

Launch Site Names Begin with 'CCA'



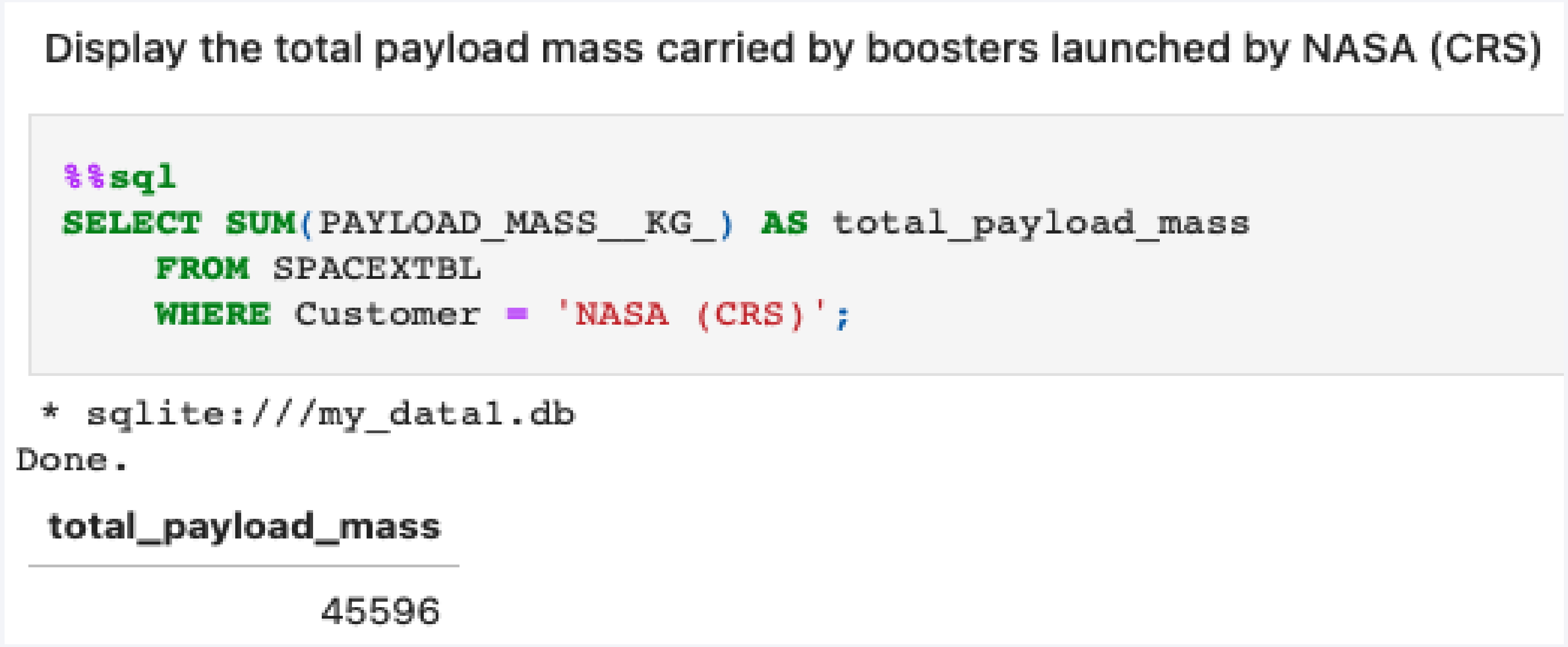
```
%%sql
SELECT * FROM SPACEXTBL WHERE Launch_Site LIKE 'CCA%' LIMIT 5;
```

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

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

“LIKE” retrieves string values beginning with ‘CCA’ and ‘LIMIT 5” provides only data for 5 records.

Total Payload Mass



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

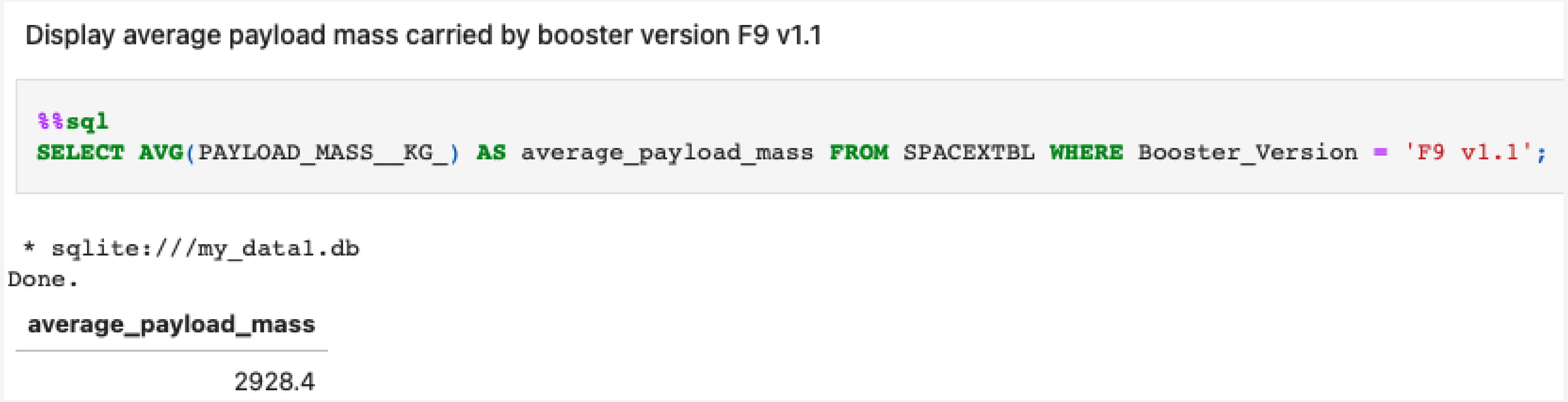
```
%%sql
SELECT SUM(PAYLOAD_MASS__KG_) AS total_payload_mass
    FROM SPACEXTBL
    WHERE Customer = 'NASA (CRS)';
```

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

total_payload_mass
45596

“SUM” is used to calculate the total payload mass carried by boosters launched by NASA (CRS).

Average Payload Mass by F9 v1.1



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

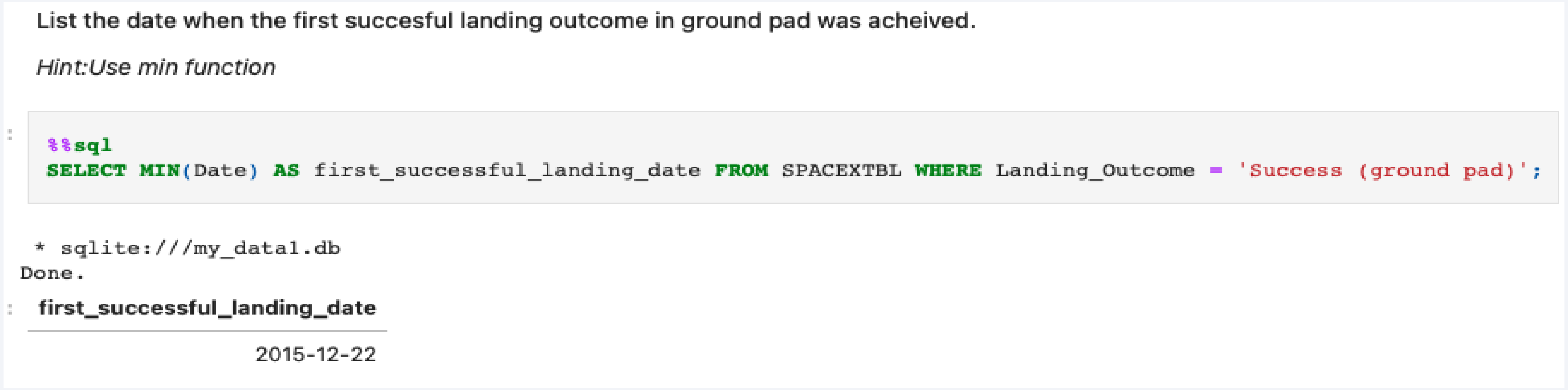
```
%%sql
SELECT AVG(PAYLOAD_MASS__KG_) AS average_payload_mass FROM SPACEXTBL WHERE Booster_Version = 'F9 v1.1';
```

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

average_payload_mass
2928.4

“AVG” is used to calculate the average of “Payload_Mass__kg_” and “WHERE” filters the results only for F9 v1.1

First Successful Ground Landing Date



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

Hint:Use min function

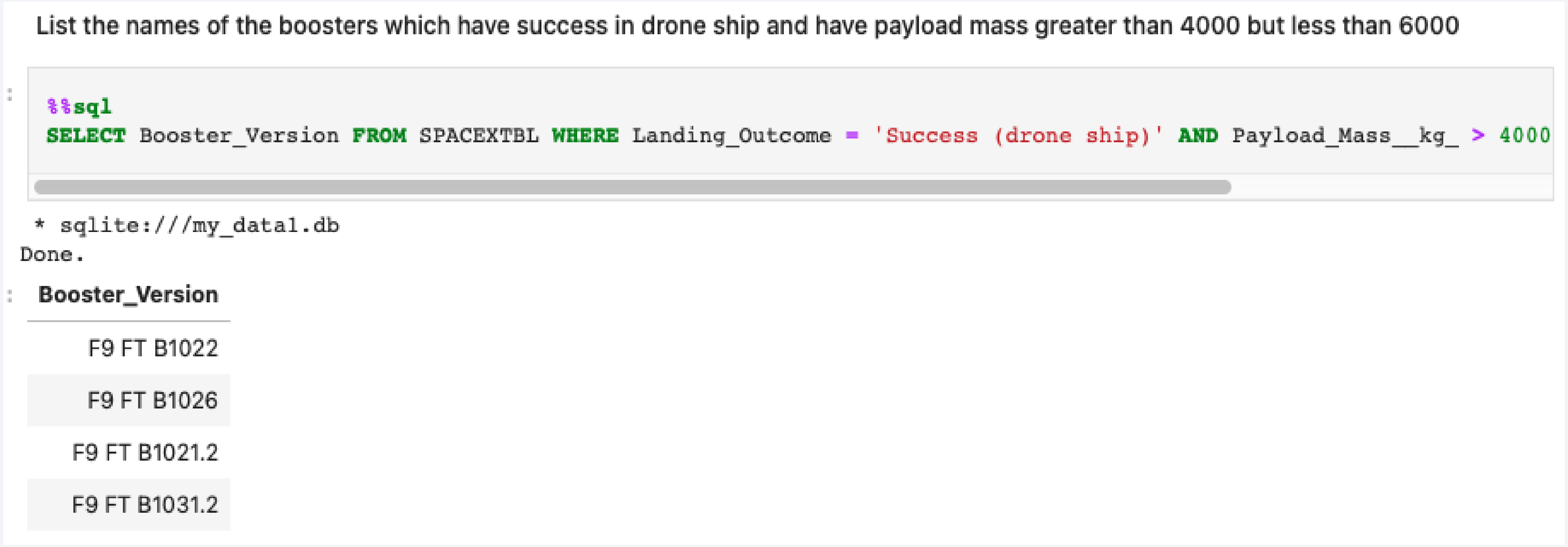
```
%%sql
SELECT MIN(Date) AS first_successful_landing_date FROM SPACEXTBL WHERE Landing_Outcome = 'Success (ground pad)';
```

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

first_successful_landing_date
2015-12-22

“MIN” is used to calculate the minimum of “Date” and “WHERE” filters the results only for successful ground pad .

Successful Drone Ship Landing with Payload between 4000 and 6000



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

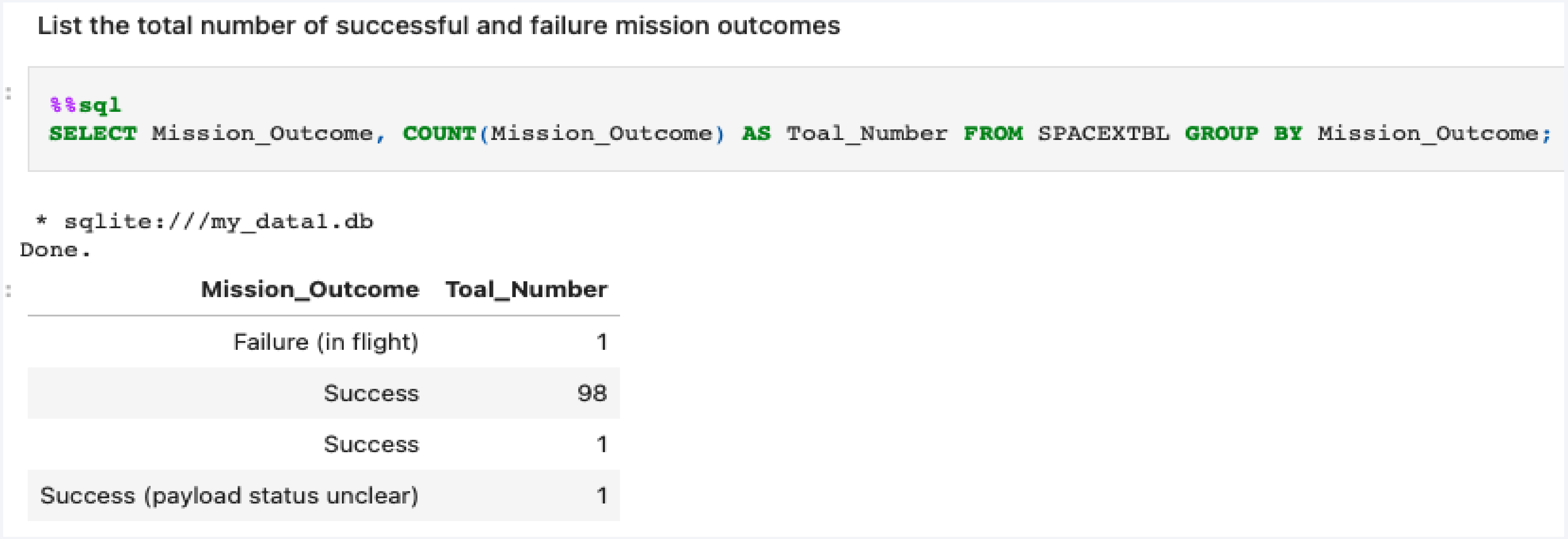
```
%%sql
SELECT Booster_Version FROM SPACEXTBL WHERE Landing_Outcome = 'Success (drone ship)' AND Payload_Mass__kg_ > 4000
```

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

Booster_Version
F9 FT B1022
F9 FT B1026
F9 FT B1021.2
F9 FT B1031.2

“WHERE” filters the results only for Successful drone ship for both conditions. “AND” is used to input the range of the values.

Total Number of Successful and Failure Mission Outcomes



List the total number of successful and failure mission outcomes

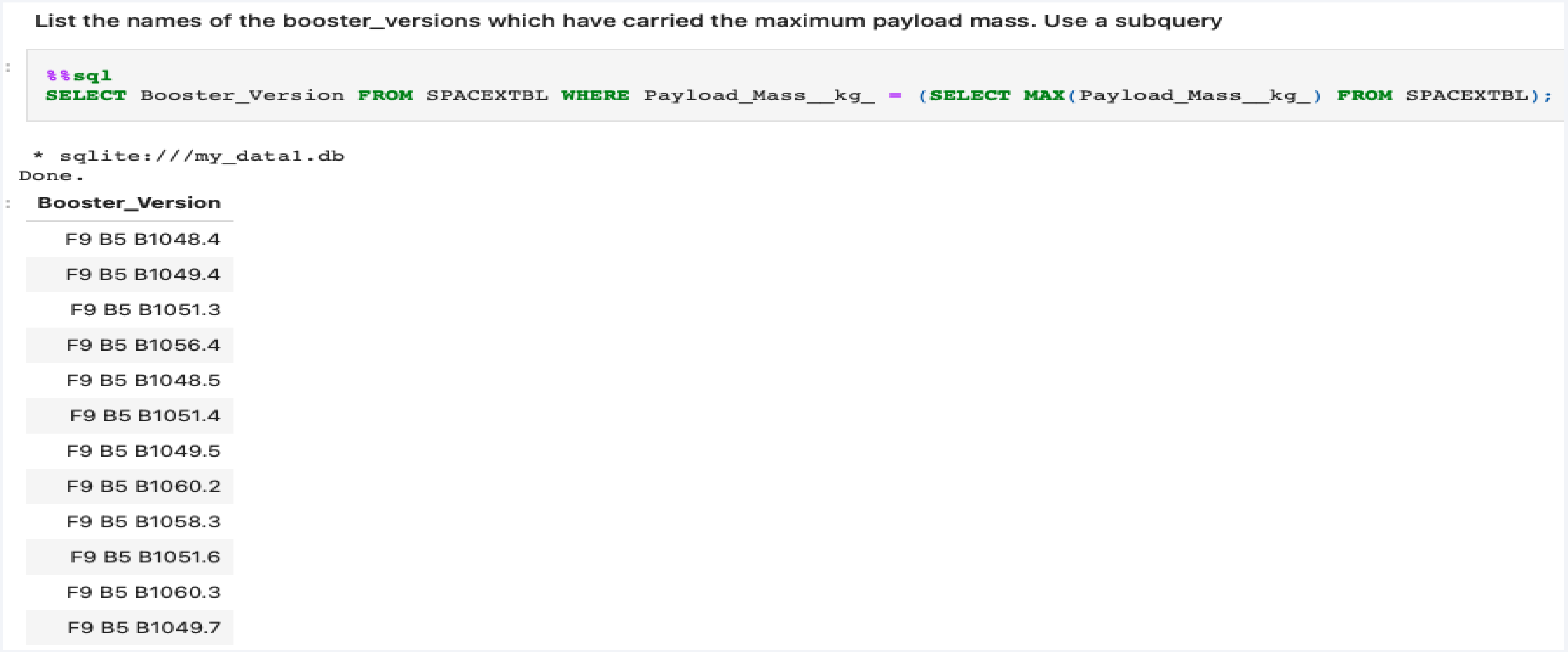
```
%%sql
SELECT Mission_Outcome, COUNT(Mission_Outcome) AS Toal_Number FROM SPACEXTBL GROUP BY Mission_Outcome;
```

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

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

“COUNT” is used to calculate the total number of successful and failure mission outcomes and “GROUP BY” is used to group the results by type of the outcome.

Boosters Carried Maximum Payload



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

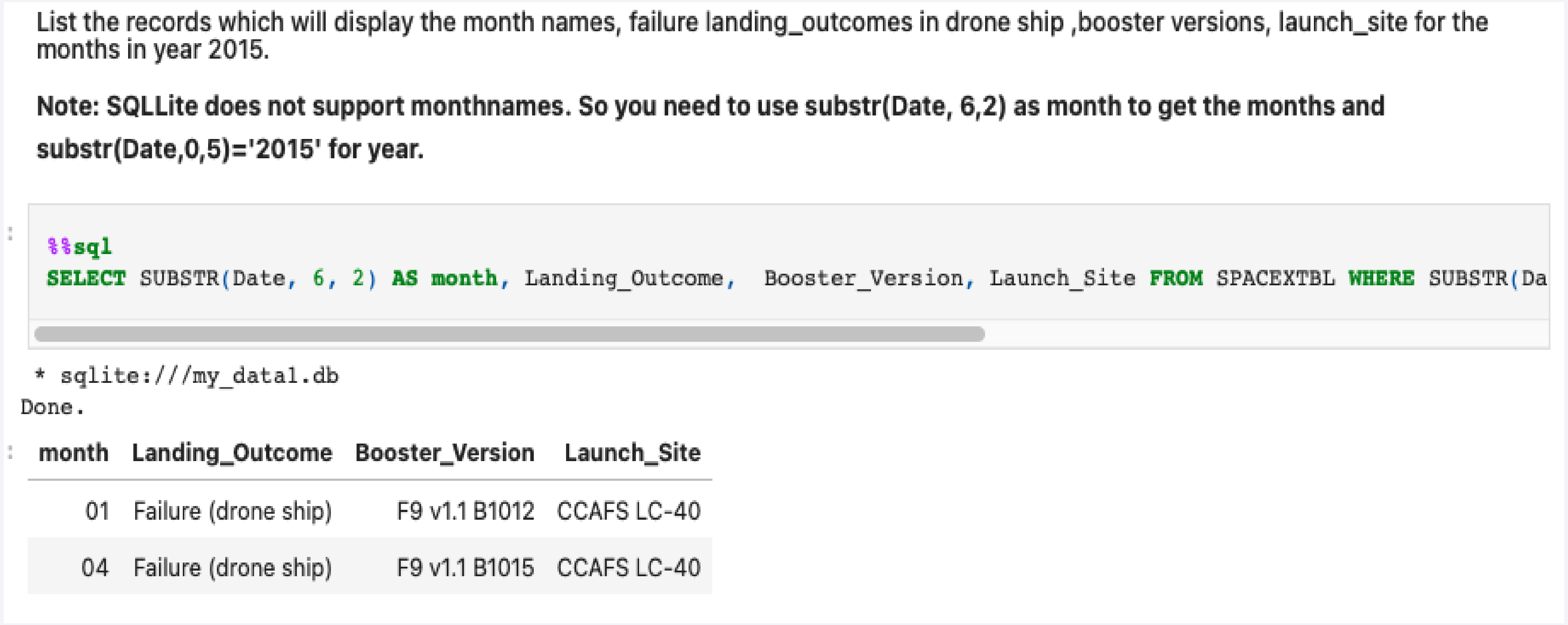
```
%%sql
SELECT Booster_Version FROM SPACEXTBL WHERE Payload_Mass__kg_ = (SELECT MAX(Payload_Mass__kg_) FROM SPACEXTBL);
```

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

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

“SELECT MAX” displays maximum payload mass carried by booster versions.

2015 Launch Records



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.

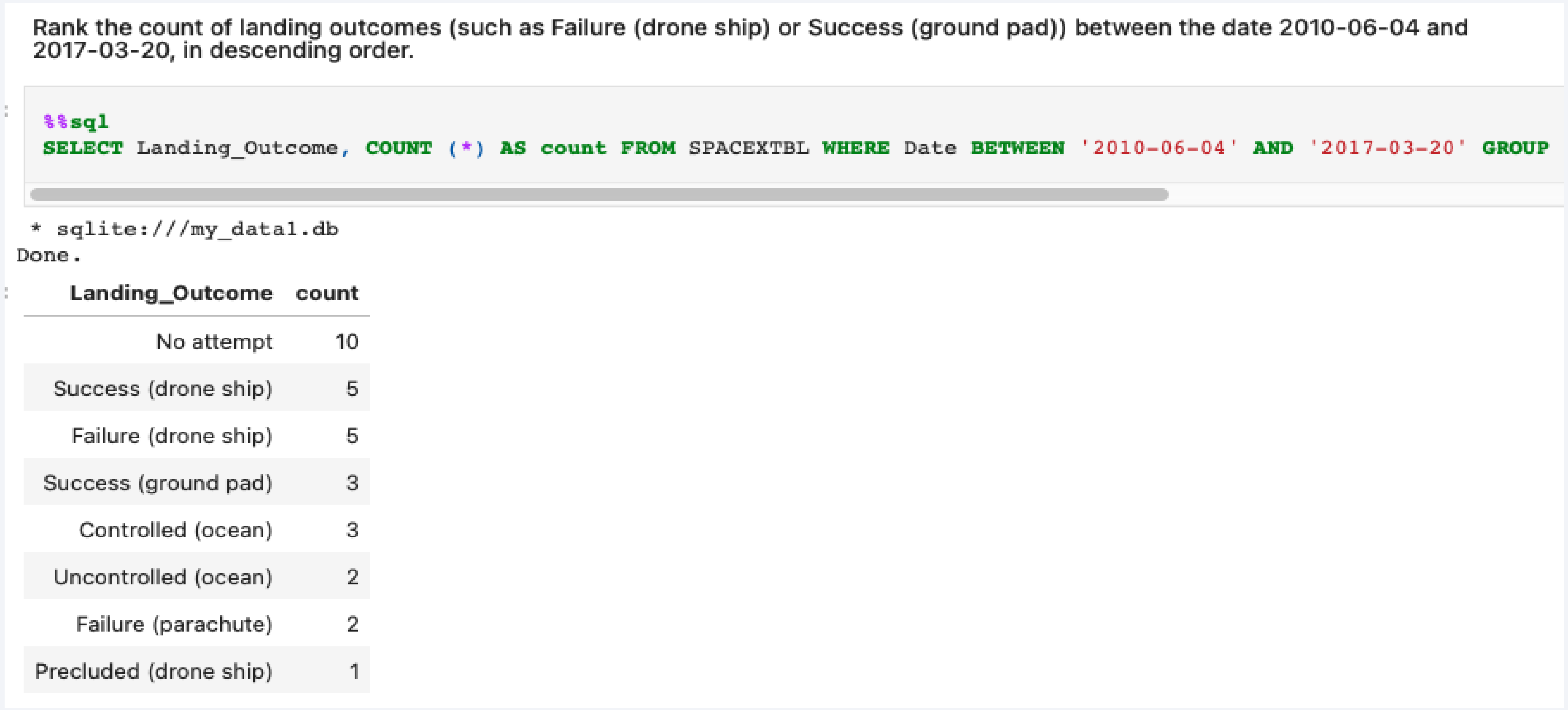
```
%%sql
SELECT SUBSTR(Date, 6, 2) AS month, Landing_Outcome,  Booster_Version, Launch_Site FROM SPACEXTBL WHERE SUBSTR(Da
```

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

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

Listing the failed landing outcomes in drone ship their launch site and booster versions for the year 2015.

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



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.

```
%%sql
SELECT Landing_Outcome, COUNT (*) AS count FROM SPACEXTBL WHERE Date BETWEEN '2010-06-04' AND '2017-03-20' GROUP
```

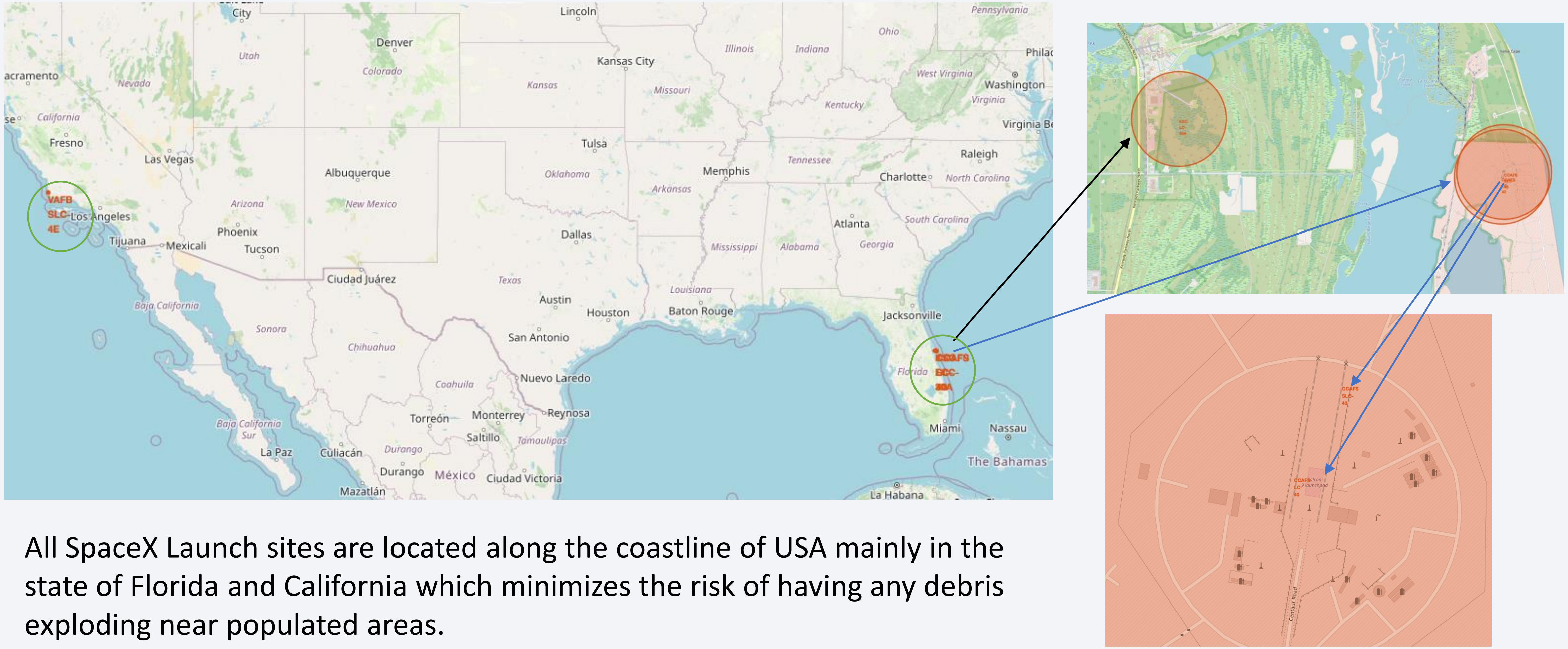
```
 * sqlite:///my_data1.db
Done.
```

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

“Count” displays number of successful and failure of landing outcomes, ”WHERE” and “BETWEEN” filters the duration between 2010-06-04 to 201-03-20 and “GROUP BY”, “ORDER BY” and “DESC” displays the results in grouped and ordered in descending order.

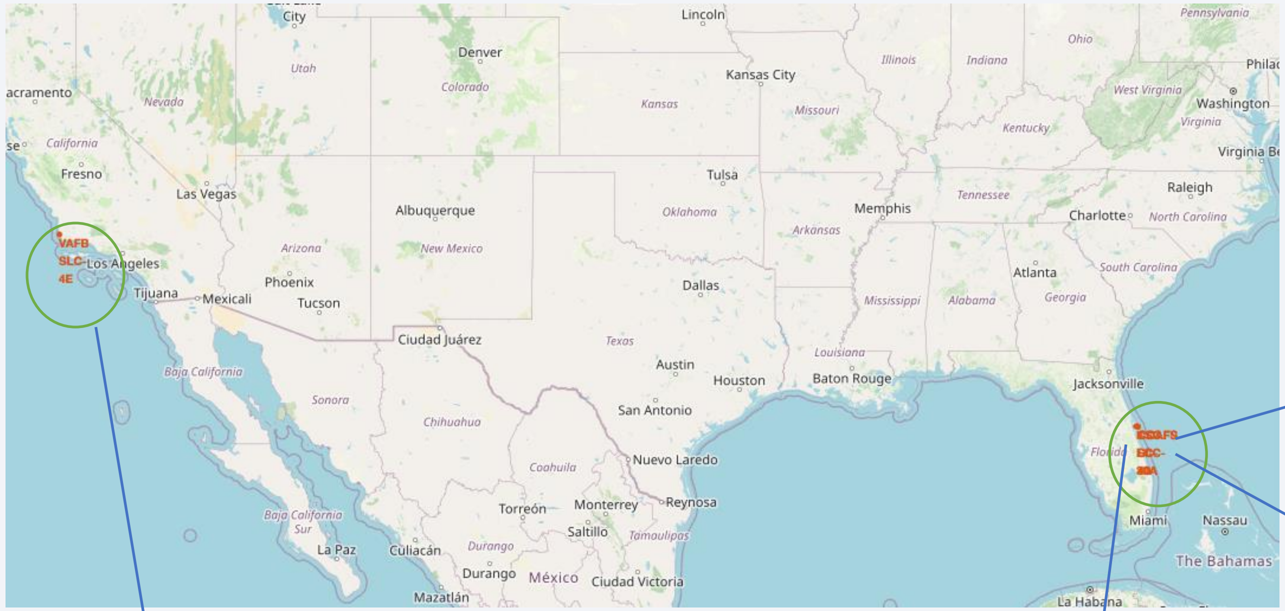
Interactive Geospatial Analytics Maps

All Launch Sites on a Map

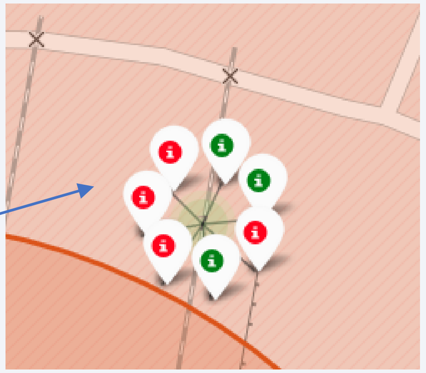


All SpaceX Launch sites are located along the coastline of USA mainly in the state of Florida and California which minimizes the risk of having any debris exploding near populated areas.

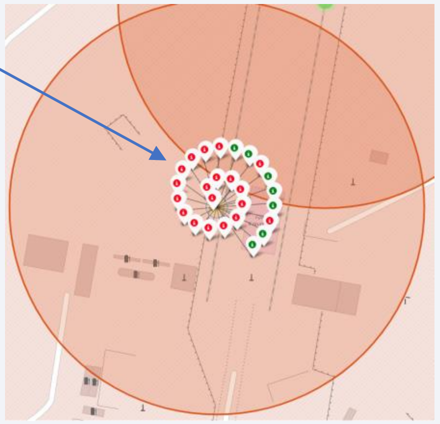
Successful & Unsuccessful Launches for Each Site



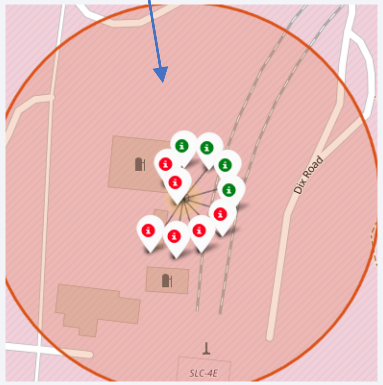
CCAFS SLC 40



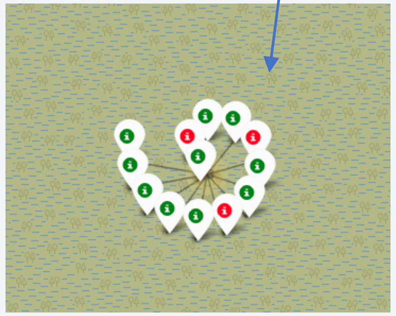
CCAFS LC 40



VAFB SLC 4E

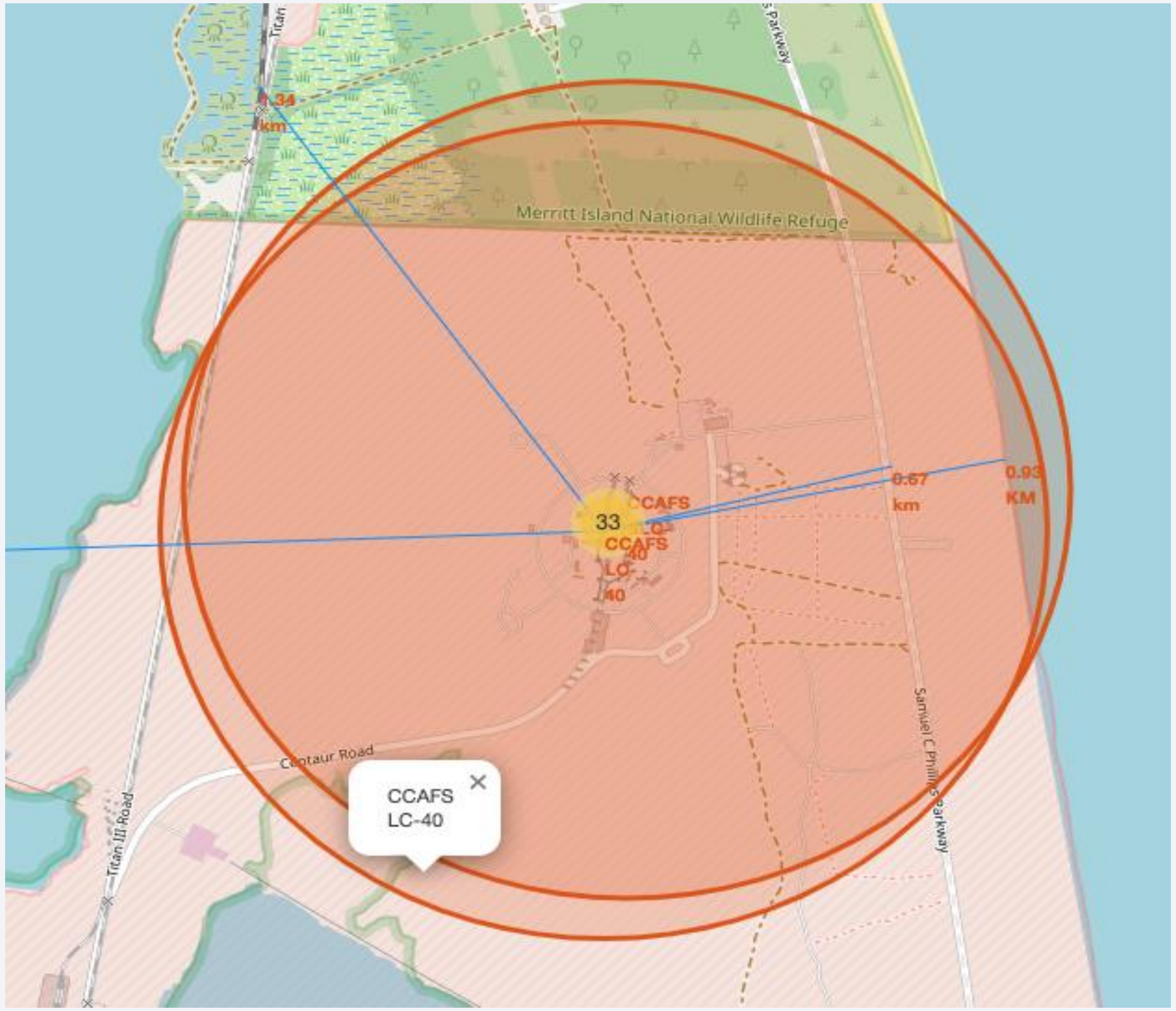


KSC LC-39A



- LAUNCHES HAVE BEEN COLOR LABELLED i.e.
  GREEN MARKER = SUCCESSFUL
  RED MARKER = UNSUCCESSFUL
- It is evident from the maps that launch site "KSC LC-39A has most successful launches.

Distance Between Launch Site to its Proximities



Let us consider launch site "CCAFS LC-40" to calculate the distance between the site to its proximities such as Coastline, Railways, Highway, Cities.

From the map we can observe the following:

1. Are the launch sites in close proximity to railways?

A. Yes, a distance of 1.34 km from launch site

2. Are the launch sites in close proximity to highways?

A. Yes, a distance of 0.67 Km from launch site

3. Are the launch sites in close proximity to coastline?

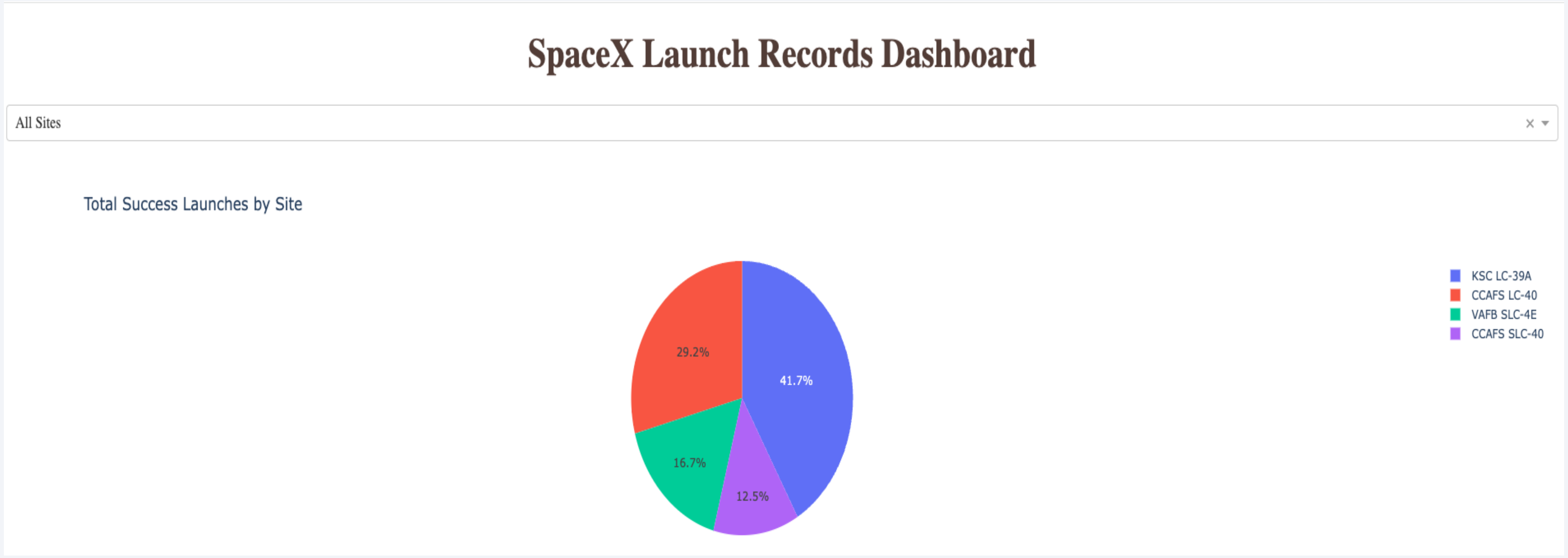
A. Yes, a distance of 0.93 km from launch site.

4. Do the launch sites keep certain distance away from cities?

A. Yes, a distance of 78.39 Km from the launch site.

Interactive DashBoard Results

Launch Success Count for all sites

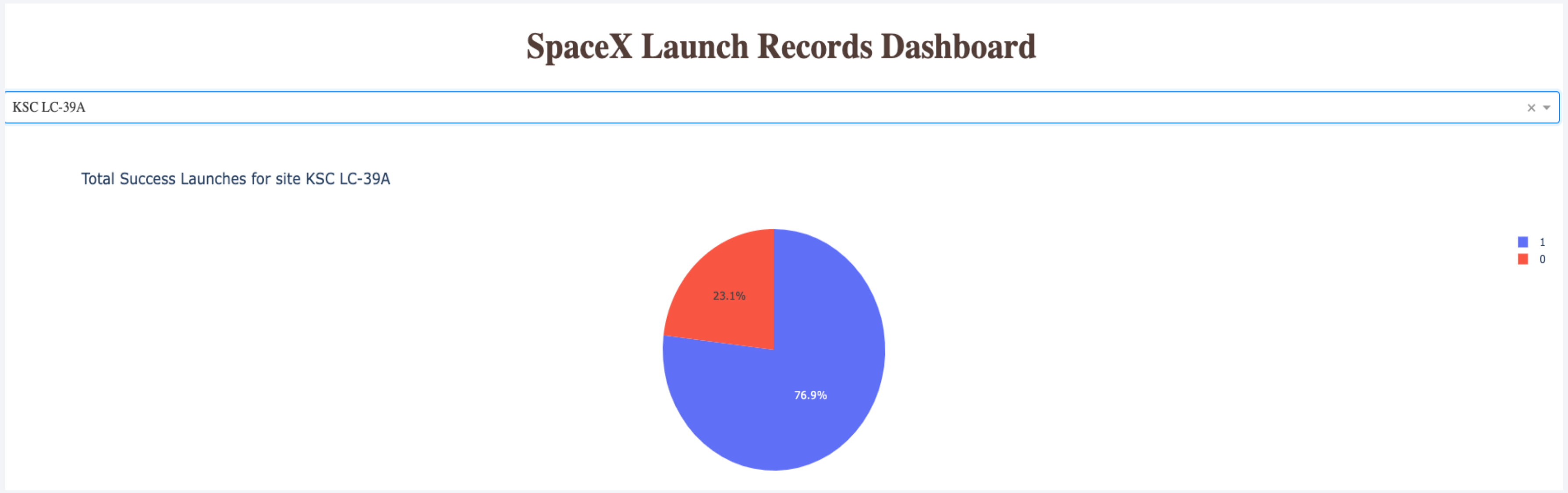

SpaceX Launch Records Dashboard

All Sites

Total Success Launches by Site

The launch site “KSC LC-39A” had the most successful launches with 41.7% of total successful launches.

Launch Site with Highest Launch Success Ratio

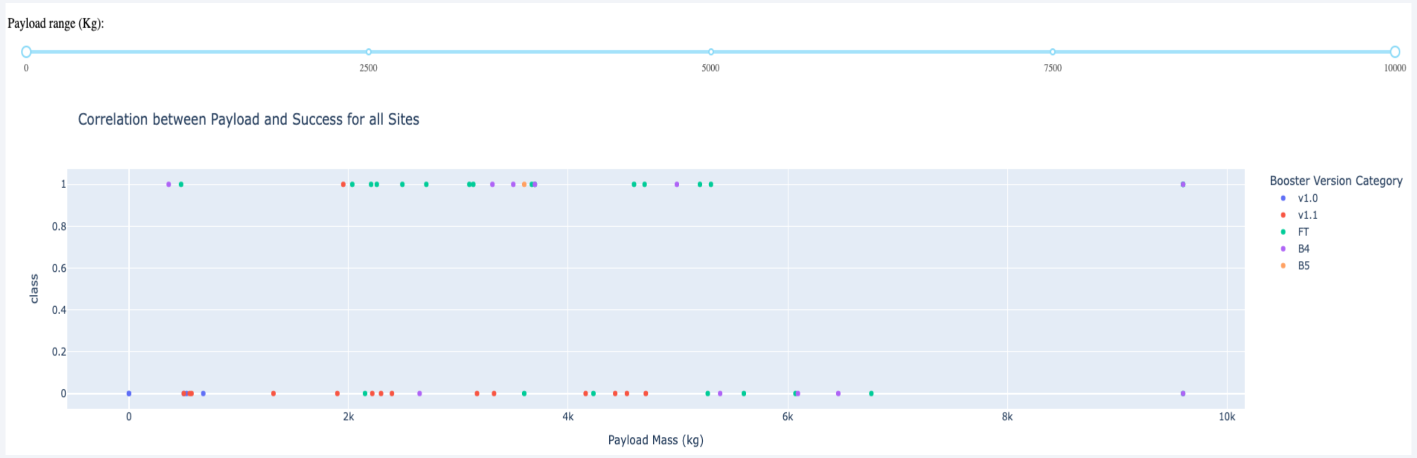

SpaceX Launch Records Dashboard

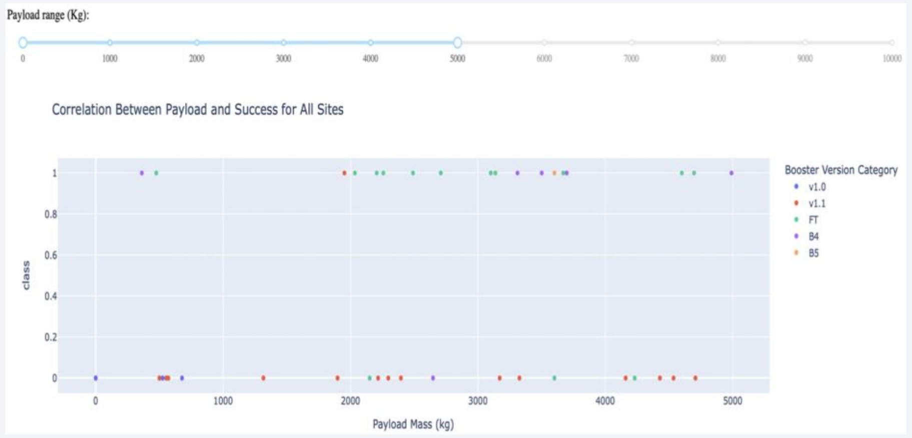
KSC LC-39A

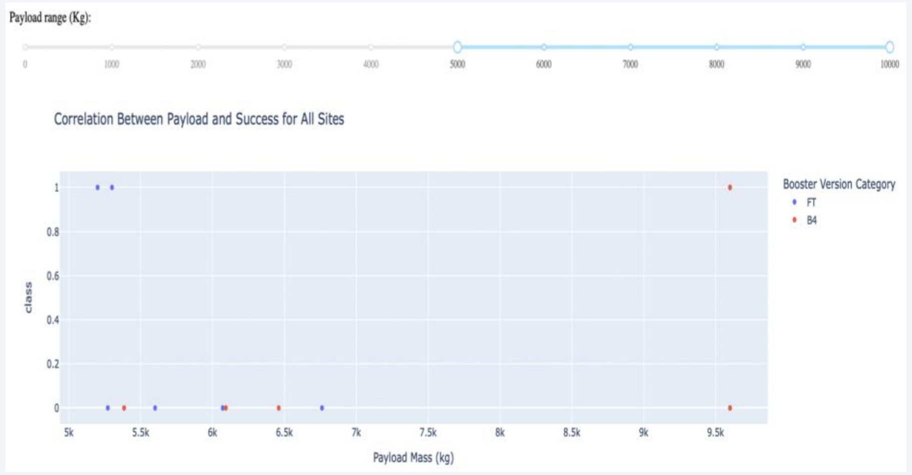
Total Success Launches for site KSC LC-39A

KSC LC-39A had the highest launch success rate with over 76.9% success rate.

Payload vs Launch Outcome for all site





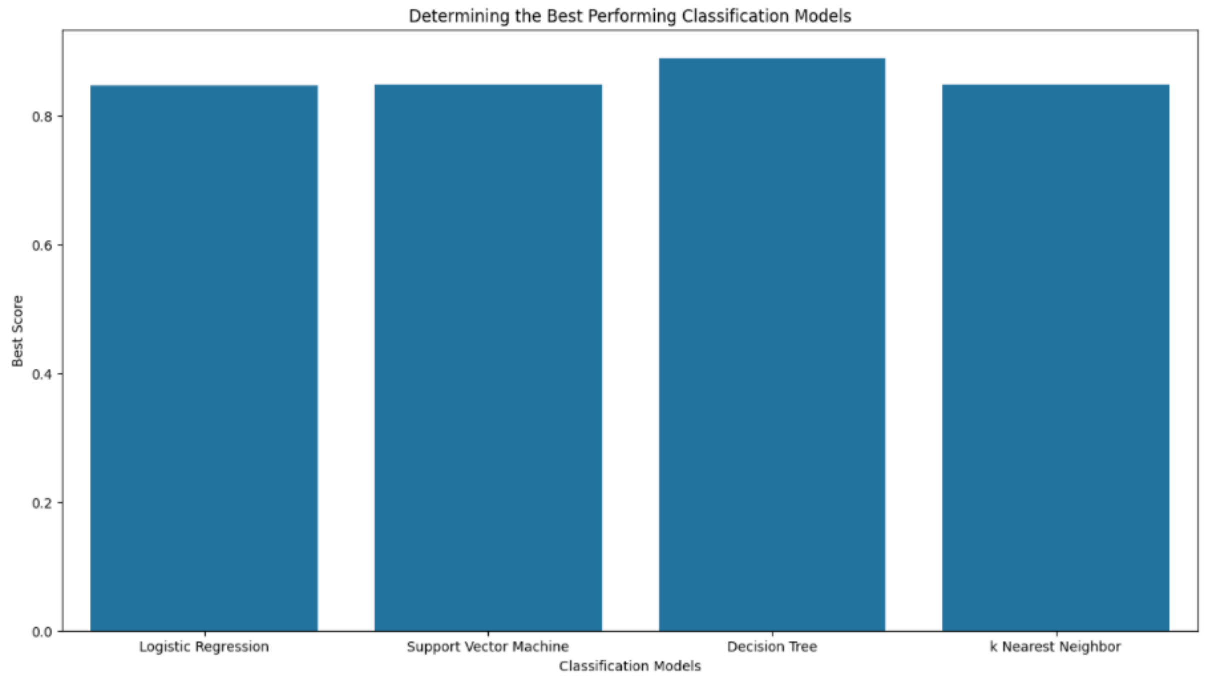


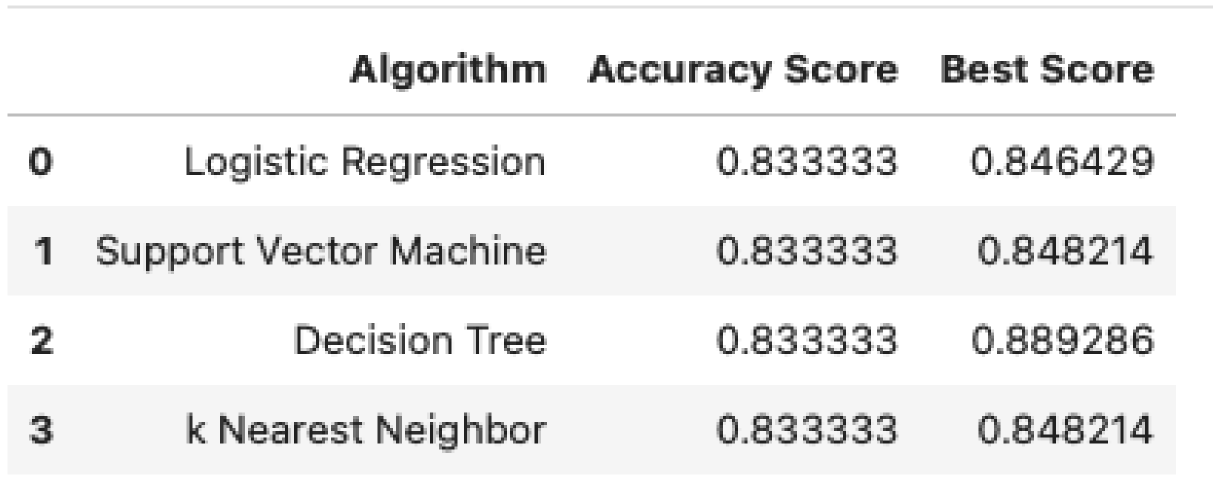
It is evident from the charts that payloads between 0 to 5000 kg have better success rate that payloads between 5000 to 10,000 kg.

Machine Learning Predictive Analysis for Classification Model Results

Classification Accuracy

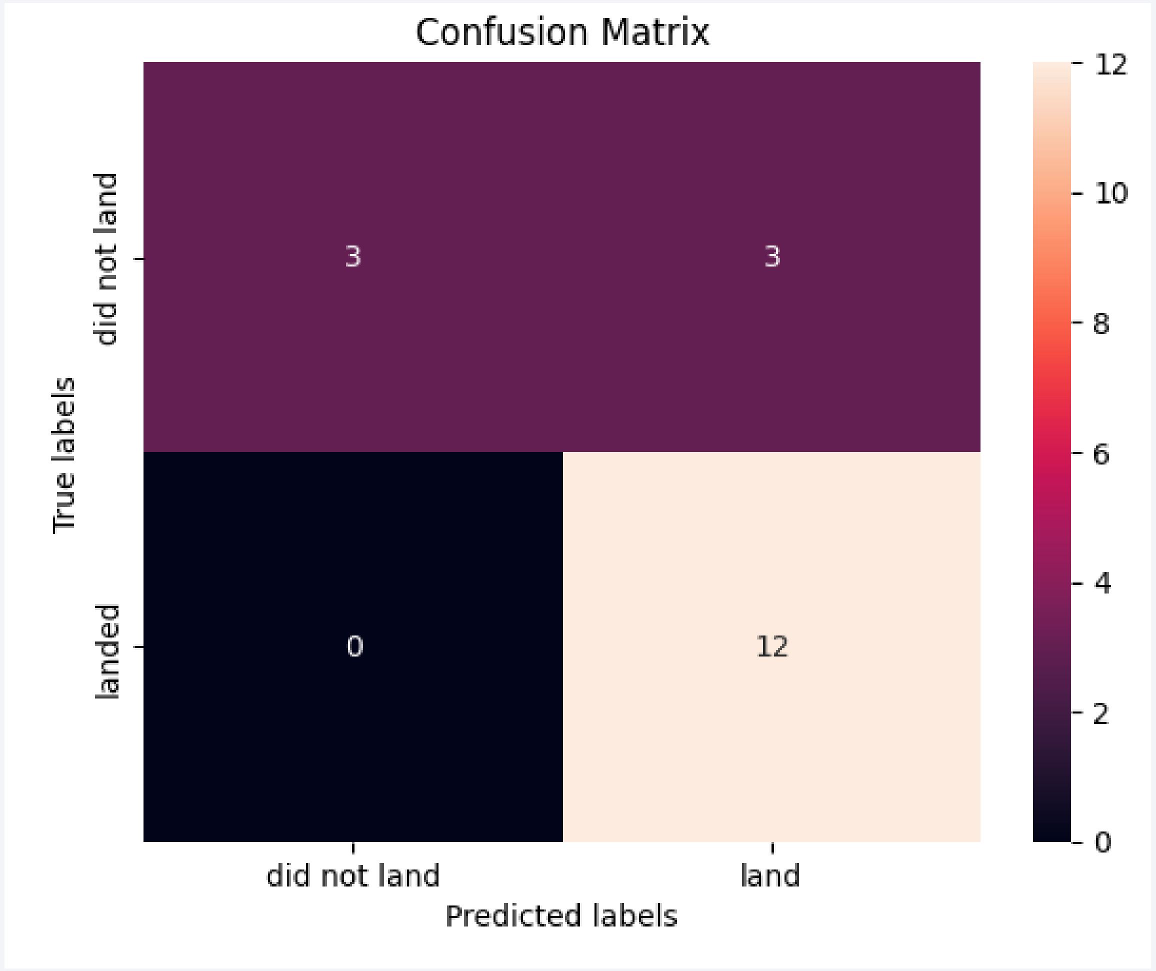
Based on the Accuracy Score and Best Score for each classification model with sample size "18" it is evident that "Decision Tree" has the highest classification score of 88.9%





	Algorithm	Accuracy Score	Best Score
0	Logistic Regression	0.833333	0.846429
1	Support Vector Machine	0.833333	0.848214
2	Decision Tree	0.833333	0.889286
3	k Nearest Neighbor	0.833333	0.848214

Confusion Matrix



- Using the best performing model the "Decision Tree" with accuracy of 88.9% , we notice that 3 of 18 total sample results classified incorrectly (False Positive).
- Whereas remaining 15 results are classified correctly (3 True Negative, 12 True Positive).

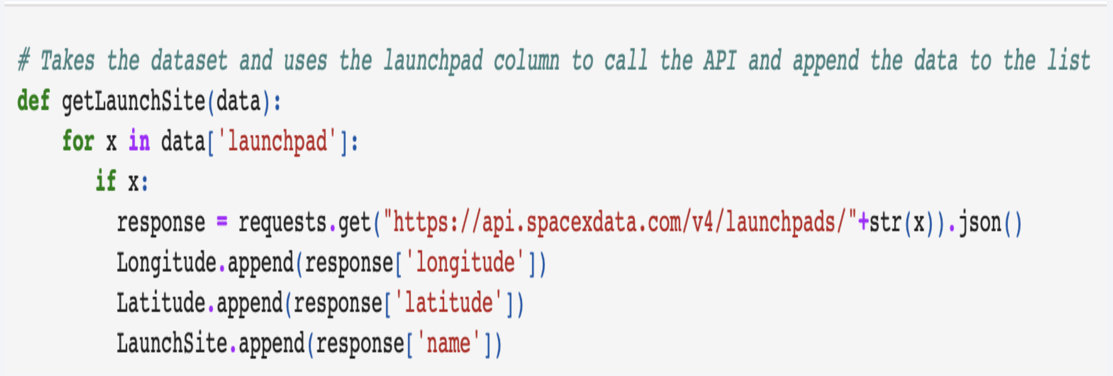
Conclusions

- As the number of flights increased, the rate of success increased, with early flights being unsuccessful.
- Orbit Type “GEO”, “ES-L1”, “HEO” and “SSO” had 100% success rate.
- Orbit Type “SSO”, “HEO”, “ES-L1” and “GEO” has 100% success rate with lesser Payload mass.
- Between 2010 to 2013 all landings were unsuccessful, whereas after 2013 to 2020 the success rate gradually increased.
- Launch site “KSC LC-39A” had the highest successful launches with 41.7% of total launches and the highest rate of successful launches with 76.9% success rate.
- Launches with lower Payload Mass showed better success rate than launches with higher Payload Mass.
- The best performing classification model is “Decision Tree” with 88.9% accuracy.

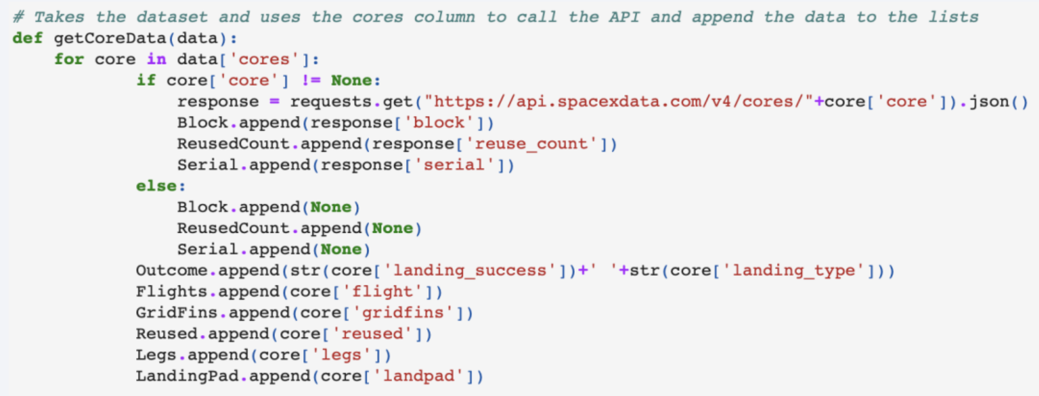
Appendix

Data Collection – SpaceX API

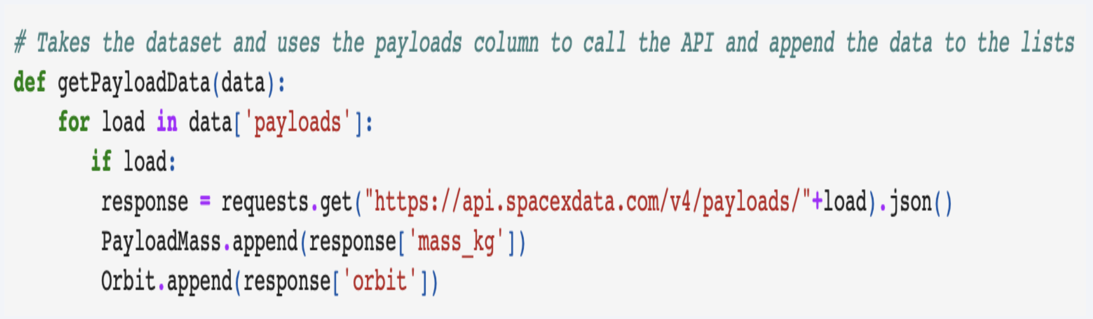
Custom functions used for retrieving and cleaning the data



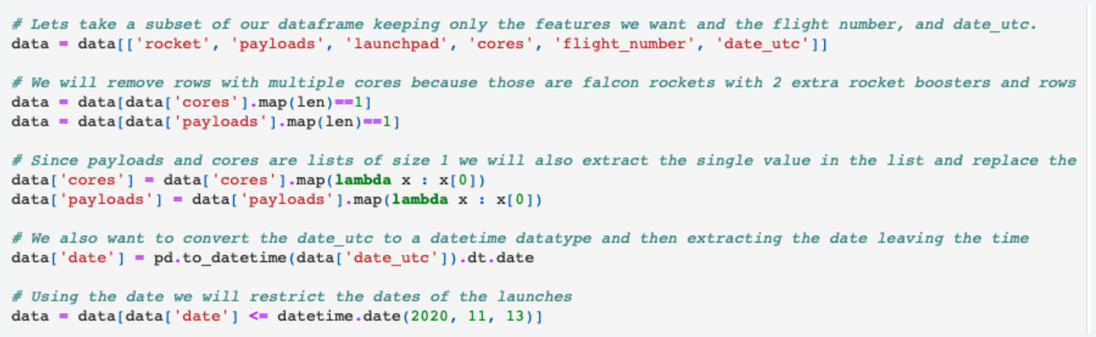
```python
# Takes the dataset and uses the launchpad column to call the API and append the data to the list
def getLaunchSite(data):
    for x in data['launchpad']:
       if x:
         response = requests.get("https://api.spacexdata.com/v4/launchpads/"+str(x)).json()
         Longitude.append(response['longitude'])
         Latitude.append(response['latitude'])
         LaunchSite.append(response['name'])
```



```python
# Takes the dataset and uses the cores column to call the API and append the data to the lists
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cores/"+core['core']).json()
                Block.append(response['block'])
                ReusedCount.append(response['reuse_count'])
                Serial.append(response['serial'])
            else:
                Block.append(None)
                ReusedCount.append(None)
                Serial.append(None)
            Outcome.append(str(core['landing_success'])+' '+str(core['landing_type']))
            Flights.append(core['flight'])
            GridFins.append(core['gridfins'])
            Reused.append(core['reused'])
            Legs.append(core['legs'])
            LandingPad.append(core['landpad'])
```



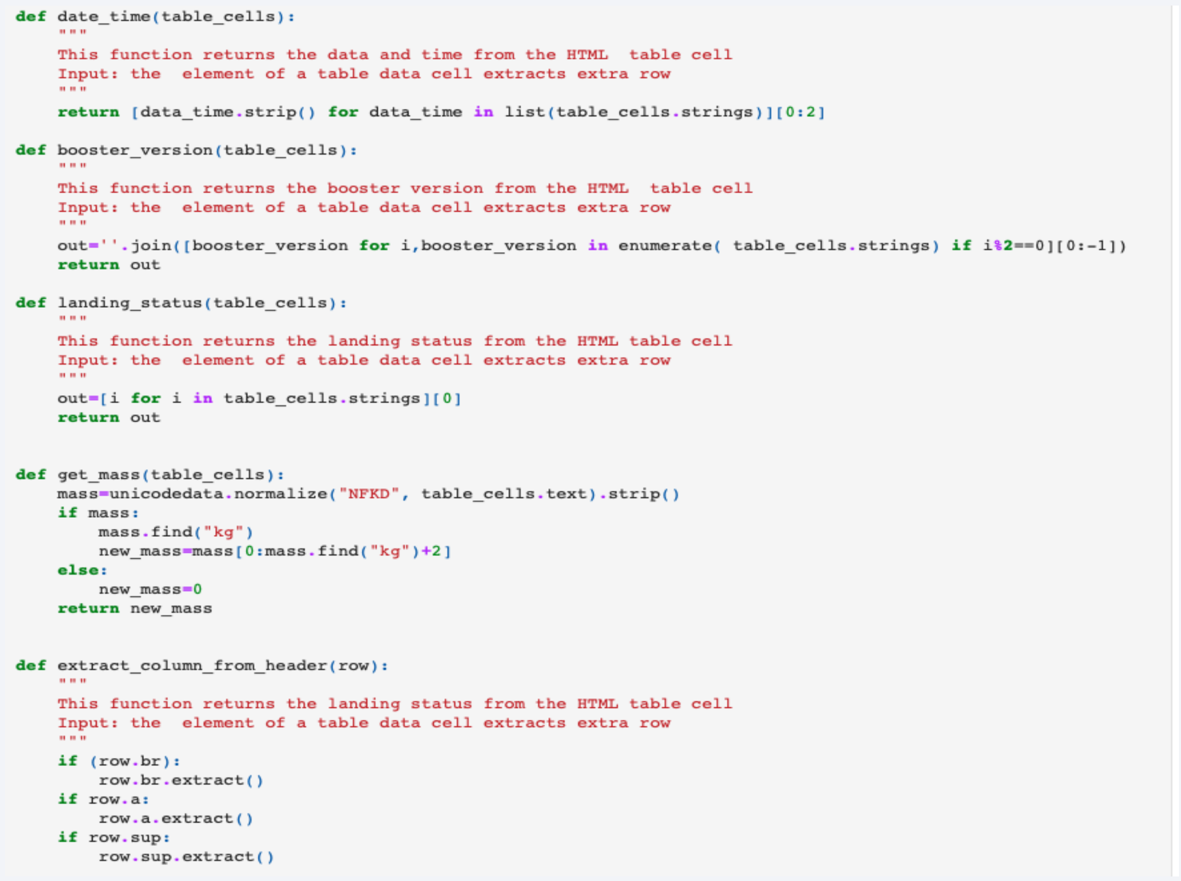
```python
# Takes the dataset and uses the payloads column to call the API and append the data to the lists
def getPayloadData(data):
    for load in data['payloads']:
       if load:
        response = requests.get("https://api.spacexdata.com/v4/payloads/"+load).json()
        PayloadMass.append(response['mass_kg'])
        Orbit.append(response['orbit'])
```



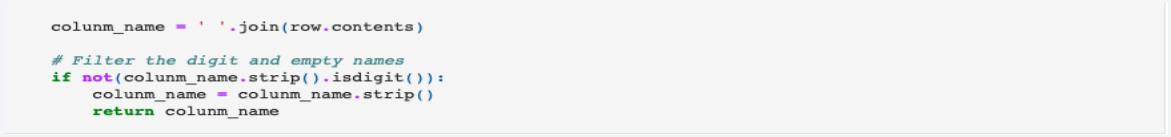
```python
# Lets take a subset of our dataframe keeping only the features we want and the flight number, and date_utc.
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]

# We will remove rows with multiple cores because those are falcon rockets with 2 extra rocket boosters and rows
data = data[data['cores'].map(len)==1]
data = data[data['payloads'].map(len)==1]

# Since payloads and cores are lists of size 1 we will also extract the single value in the list and replace the
data['cores'] = data['cores'].map(lambda x : x[0])
data['payloads'] = data['payloads'].map(lambda x : x[0])

# We also want to convert the date_utc to a datetime datatype and then extracting the date leaving the time
data['date'] = pd.to_datetime(data['date_utc']).dt.date

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

Appendix

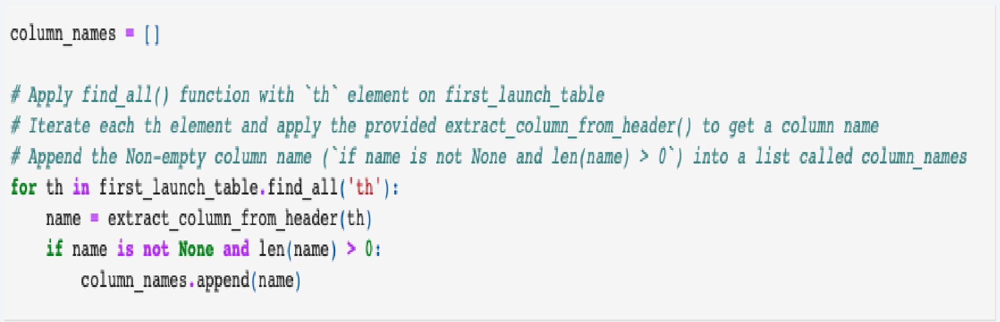
Data Collection – Web Scraping



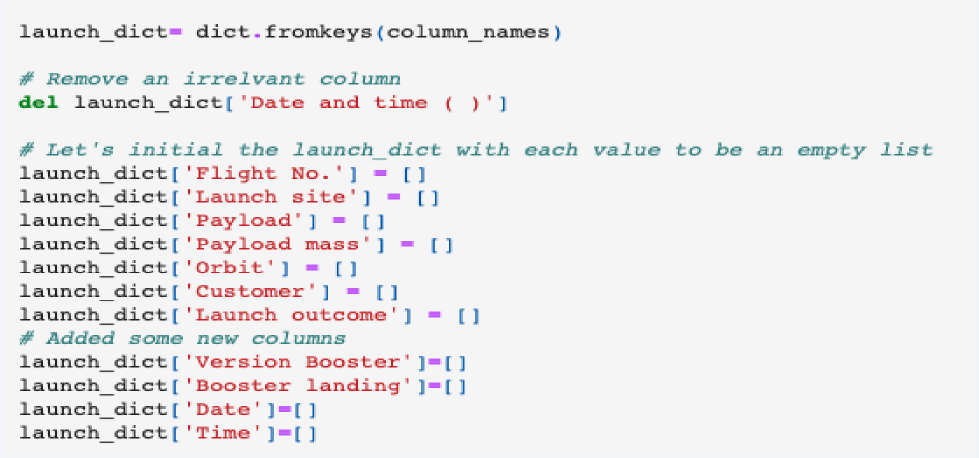
```python
def date_time(table_cells):
    """
    This function returns the data and time from the HTML  table cell
    Input: the  element of a table data cell extracts extra row
    """
    return [data_time.strip() for data_time in list(table_cells.strings)][0:2]

def booster_version(table_cells):
    """
    This function returns the booster version from the HTML  table cell
    Input: the  element of a table data cell extracts extra row
    """
    out=''.join([booster_version for i,booster_version in enumerate( table_cells.strings) if i%2==0][0:-1])
    return out

def landing_status(table_cells):
    """
    This function returns the landing status from the HTML table cell
    Input: the  element of a table data cell extracts extra row
    """
    out=[i for i in table_cells.strings][0]
    return out


def get_mass(table_cells):
    mass=unicodedata.normalize("NFKD", table_cells.text).strip()
    if mass:
        mass.find("kg")
        new_mass=mass[0:mass.find("kg")+2]
    else:
        new_mass=0
    return new_mass


def extract_column_from_header(row):
    """
    This function returns the landing status from the HTML table cell
    Input: the  element of a table data cell extracts extra row
    """
    if (row.br):
        row.br.extract()
    if row.a:
        row.a.extract()
    if row.sup:
        row.sup.extract()
```



```python
    colunm_name = ' '.join(row.contents)

    # Filter the digit and empty names
    if not(colunm_name.strip().isdigit()):
        colunm_name = colunm_name.strip()
        return colunm_name
```



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



```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']=[]
```

Thank you!