Lucas Parreiras de Sousa

IBM Data Science Capstone Project

Outline

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
  - Machine Learning Prediction
- Summary of all results
  - Exploratory Data Analysis result
  - Interactive analytics in screenshots
  - Predictive Analytics result

Introduction

Project background and context:

Falcon 9 are SpaceX’s rockets that launches with a cost of U$62 million; other providers can cost up to U$65 million. Much of the savings is because Space X can reuse the first stage. So this project objective is to predict if the first stage will land successfully (to be reused) and this allows us to calculate the cost of a launch.

Problems to be answer:

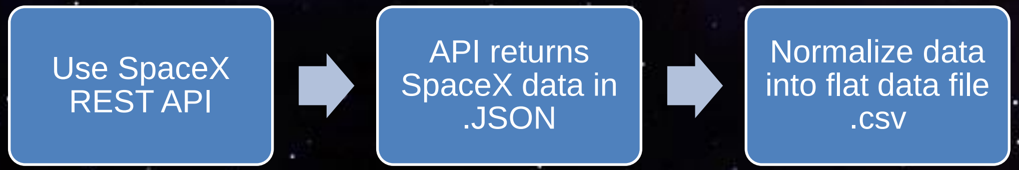
- Which features influence the succes of a rocket landing?
- The interaction amongst various features that determine the success rate of a successful landing.
- What are the best conditions to get the best results and ensure the best rocket success landind rate

Methodology

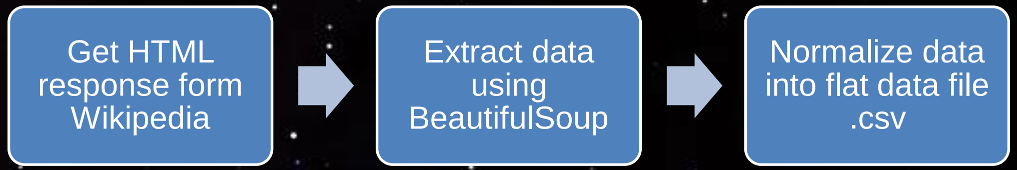
1. Data collection:
   - SpaceX API
   - web scraping from Wikipedia.
2. Data wrangling
   - One-hot encoding was applied to categorical features
   - Drop irrelevants columns
3. Perform exploratory data analysis (EDA) using visualization and SQL
4. Perform interactive visual analytics using Folium and Plotly Dash
5. Perform predictive analysis using classification models
   - How to build, tune, evaluate classification models

1. Data collection

- SpaceX API
  - Information about SpaceX launches in JSON format

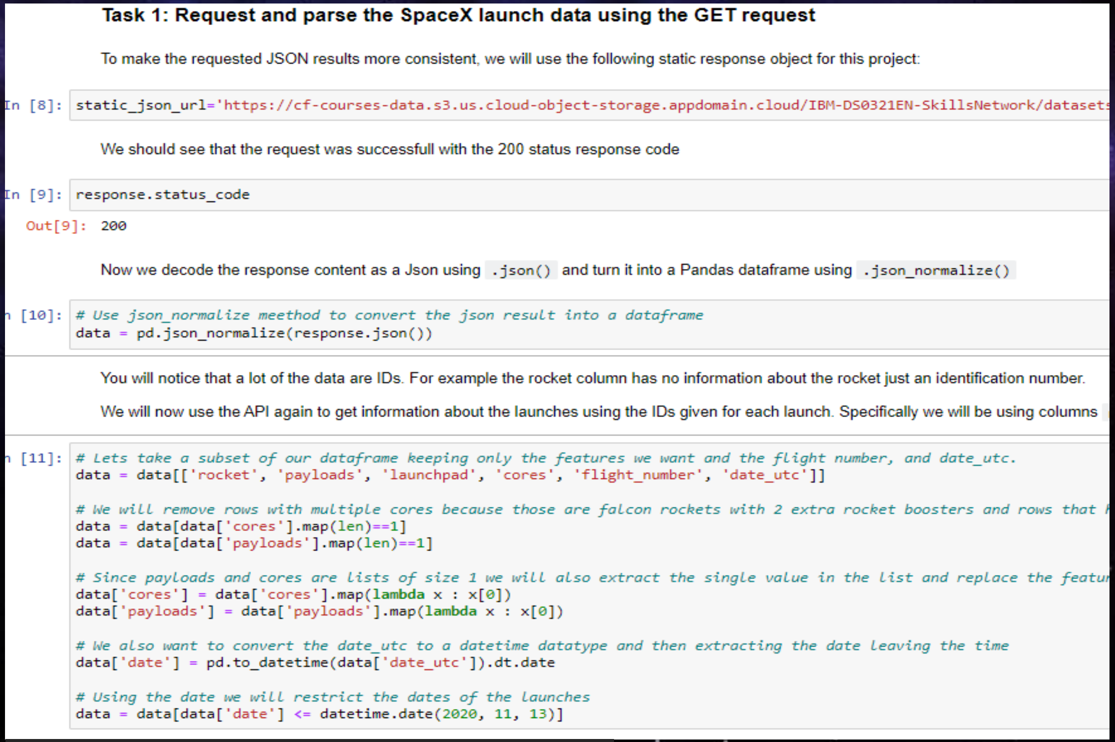

Use SpaceX REST API
API returns SpaceX data in .JSON
Normalize data into flat data file .csv

- Web Scrapping from Wikipedia
  - Information about SpaceX launches in HTML format


Get HTML response form Wikipedia
Extract data using BeautifulSoup
Normalize data into flat data file .csv

SpaceX API

Link para notebook



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 [8]: 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 [9]: response.status_code
Out[9]: 200
```

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

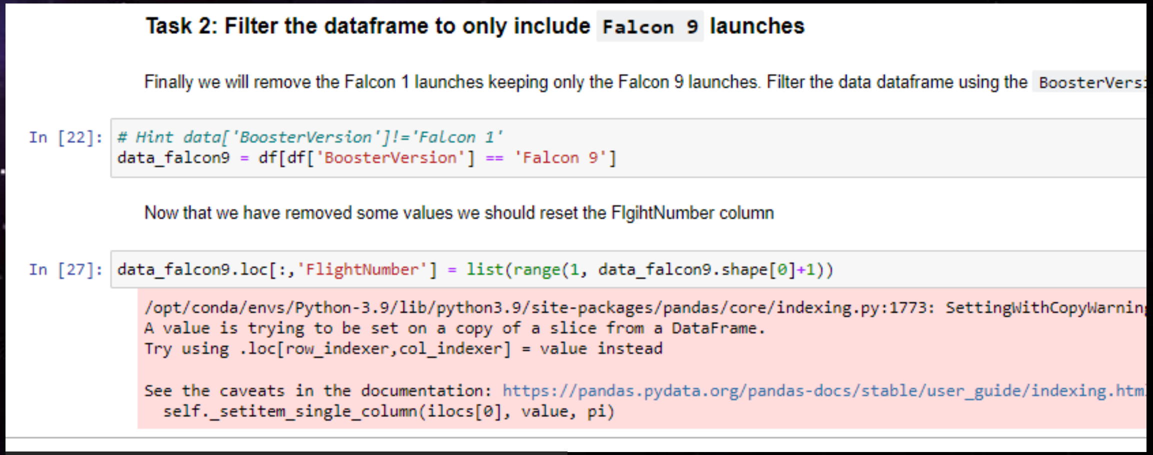
```
n [10]: # Use json_normalize meethod to convert the json result into a dataframe
        data = pd.json_normalize(response.json())
```

You will notice that a lot of the data are IDs. For example the rocket column has no information about the rocket just an identification number.

We will now use the API again to get information about the launches using the IDs given for each launch. Specifically we will be using columns

```
n [11]: # 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 that h
        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 featur
        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)]
```

1. Requesting JSON and normalizing it



Task 2: Filter the dataframe to only include `Falcon 9` launches

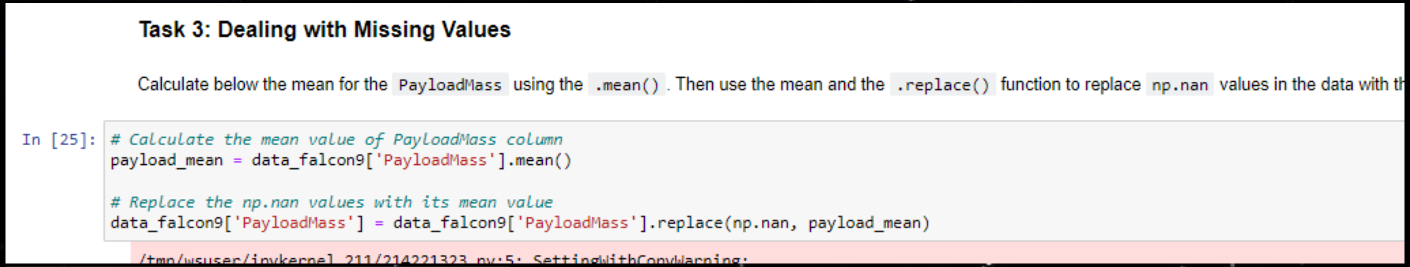
Finally we will remove the Falcon 1 launches keeping only the Falcon 9 launches. Filter the data dataframe using the `BoosterVers`

```
In [22]: # Hint data['BoosterVersion']!='Falcon 1'
         data_falcon9 = df[df['BoosterVersion'] == 'Falcon 9']
```

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

```
In [27]: data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))

         /opt/conda/envs/Python-3.9/lib/python3.9/site-packages/pandas/core/indexing.py:1773: SettingWithCopyWarnin
         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.htm
           self._setitem_single_column(ilocs[0], value, pi)
```

2. Filtering just Falcon 9 launches



Task 3: Dealing with Missing Values

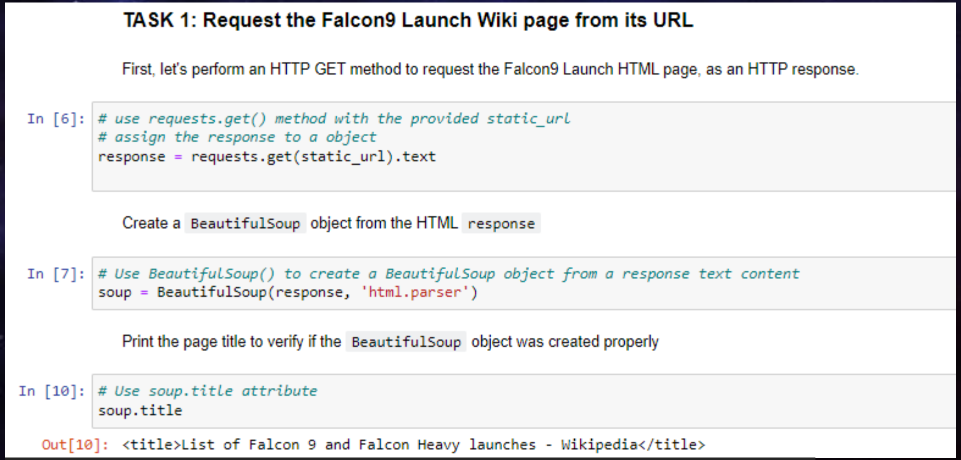
Calculate below the mean for the `PayloadMass` using the `.mean()`. Then use the mean and the `.replace()` function to replace `np.nan` values in the data with th

```
In [25]: # Calculate the mean value of PayloadMass column
         payload_mean = data_falcon9['PayloadMass'].mean()

         # Replace the np.nan values with its mean value
         data_falcon9['PayloadMass'] = data_falcon9['PayloadMass'].replace(np.nan, payload_mean)

         /tmp/wsuser/ipykernel_211/214221323.py:5: SettingWithCopyWarning:
```

3. Dealing with missing values

Scrapping from Wikipedia

Link para notebook



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

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

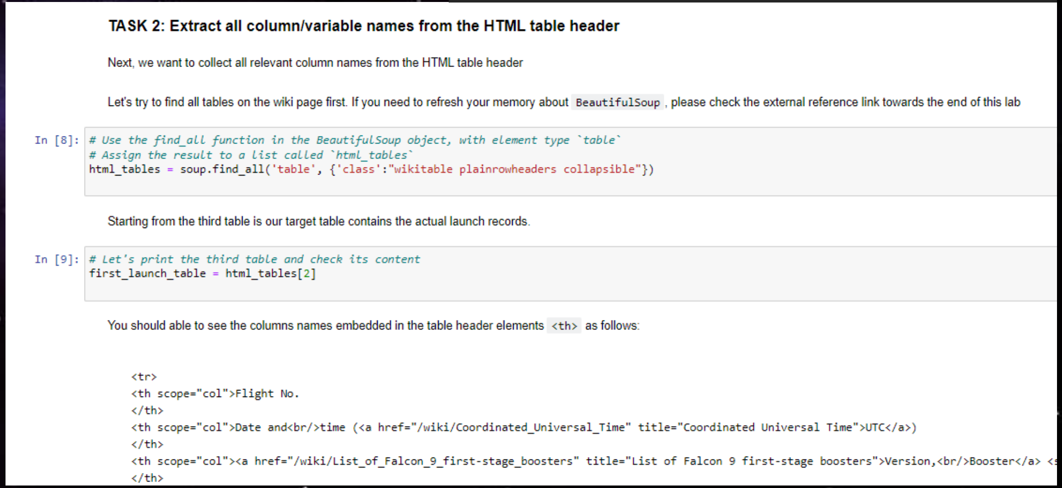
```
In [7]: # Use BeautifulSoup() to create a BeautifulSoup object from a response text content
        soup = BeautifulSoup(response, 'html.parser')
```

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

```
In [10]: # Use soup.title attribute
         soup.title

Out[10]: <title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

1. Creating BeautifulSoup object from response of url

2. Extracting the name of columns



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

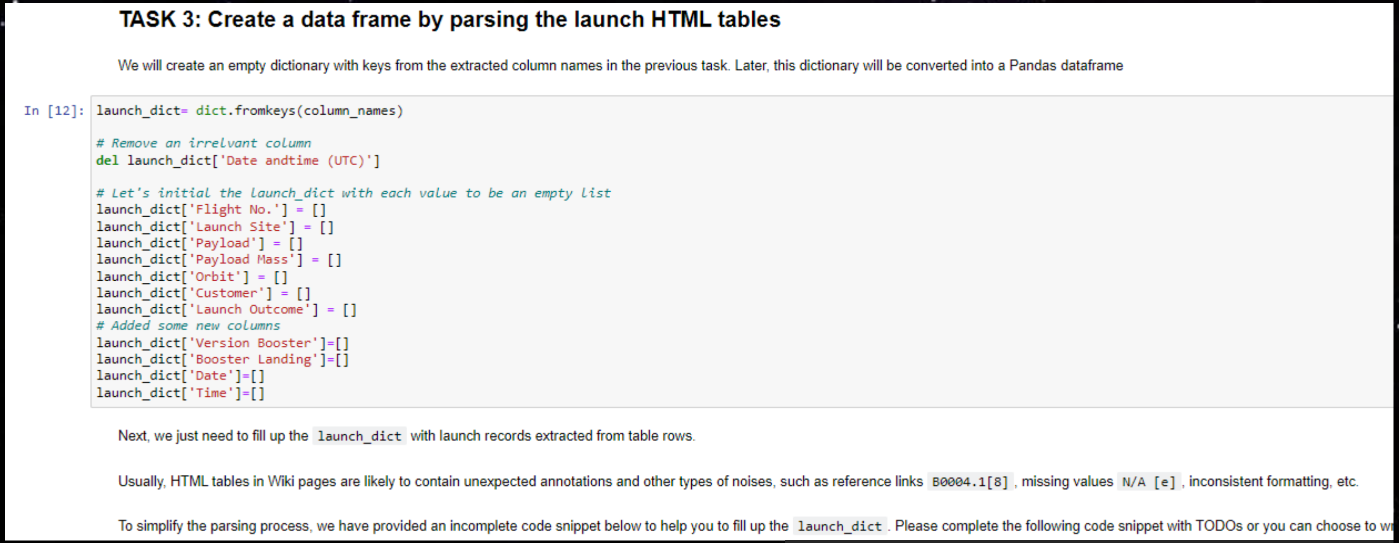
```
In [8]: # Use the find_all function in the BeautifulSoup object, with element type `table`
        # Assign the result to a list called `html_tables`
        html_tables = soup.find_all('table', {'class':"wikitable plainrowheaders collapsible"})
```

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

```
In [9]: # Let's print the third table and check its content
        first_launch_table = html_tables[2]
```

You should able to see the columns names embedded in the table header elements `<th>` as follows:

```
<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">Version,<br/>Booster</a> <
</th>
```



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 [12]: launch_dict= dict.fromkeys(column_names)

         # Remove an irrelvant column
         del launch_dict['Date andtime (UTC)']

         # 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.

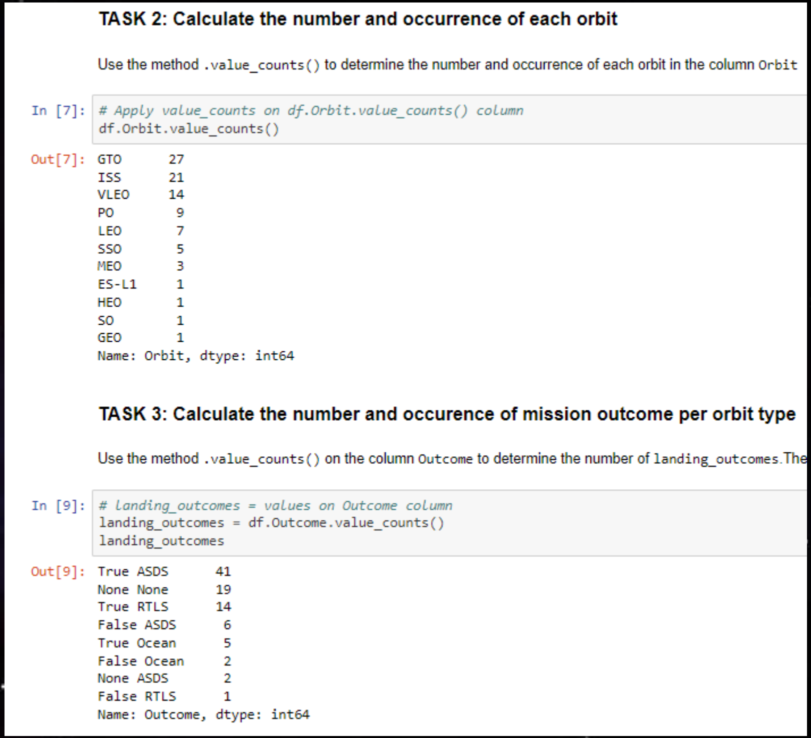
To simplify the parsing process, we have provided an incomplete code snippet below to help you to fill up the `launch_dict` . Please complete the following code snippet with TODOs or you can choose to w

3. Creating dataframe from a dictionary

2. Data wrangling

Link para notebook

- We performed exploratory data analysis and determined the training labels.
- We calculated the number of launches at each site, and the number and occurrence of each orbits
- We created landing outcome label from outcome column and exported the results to csv.



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 [7]:
```
# Apply value_counts on df.Orbit.value_counts() column
df.Orbit.value_counts()
```

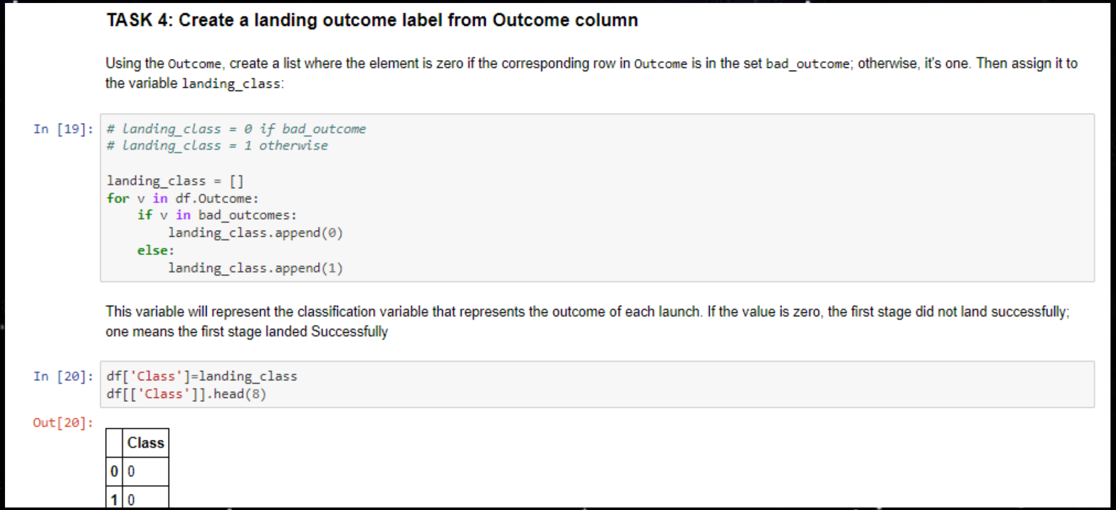
Out[7]:
```
GTO      27
ISS      21
VLEO     14
PO        9
LEO       7
SSO       5
MEO       3
ES-L1     1
HEO       1
SO        1
GEO       1
Name: Orbit, dtype: int64
```

TASK 3: Calculate the number and occurence of mission outcome per orbit type

Use the method `.value_counts()` on the column `Outcome` to determine the number of `landing_outcomes`.The

In [9]:
```
# landing_outcomes = values on Outcome column
landing_outcomes = df.Outcome.value_counts()
landing_outcomes
```

Out[9]:
```
True ASDS      41
None None      19
True RTLS      14
False ASDS      6
True Ocean      5
False Ocean     2
None ASDS       2
False RTLS      1
Name: Outcome, dtype: int64
```



TASK 4: Create a landing outcome label from Outcome column

Using the `Outcome`, create a list where the element is zero if the corresponding row in `Outcome` is in the set `bad_outcome`; otherwise, it's one. Then assign it to the variable `landing_class`:

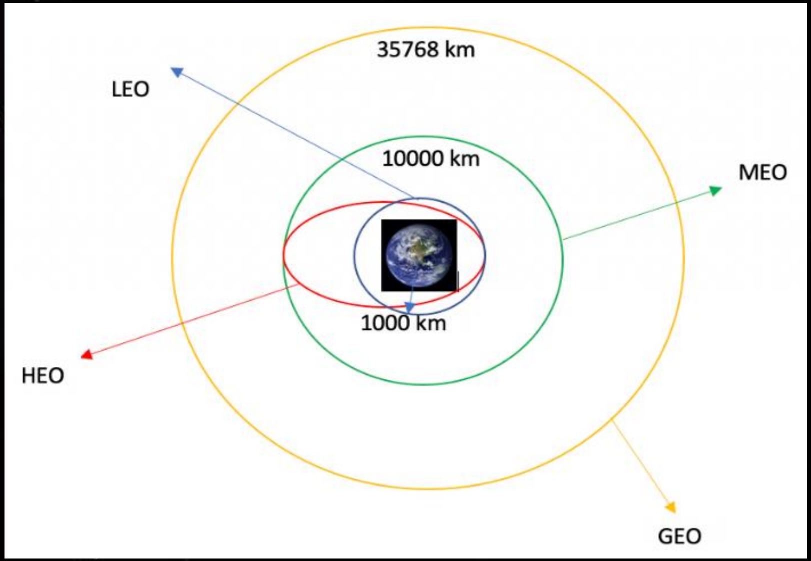
In [19]:
```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise

landing_class = []
for v in df.Outcome:
    if v in bad_outcomes:
        landing_class.append(0)
    else:
        landing_class.append(1)
```

This variable will represent the classification variable that represents the outcome of each launch. If the value is zero, the first stage did not land successfully; one means the first stage landed Successfully

In [20]:
```
df['Class']=landing_class
df[['Class']].head(8)
```

Out[20]:

	Class
0	0
1	0



3.1. EDA with SQL

Link para notebook

We applied EDA with SQL to get insight from the data. We wrote queries to find out for instance:

- Display the names of the unique launch sites in the space mission
- Display 5 records where launch sites begin with the string 'CCA'
- Display the total payload mass carried by boosters launched by NASA (CRS)
- Display average payload mass carried by booster version F9 v1.1
- List the date when the first successful landing outcome in ground pad was acheived.
- List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000
- 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 failed landing_outcomes in drone ship, their booster versions, and launch site names for 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

3.2. EDA with Data Visualization

Link para notebook

We explored the data by visualizing the relationship between features and plotting the graphs.

Scatter Graph

1. Flight Number Vs. Launch Site
2. Payload Vs. Launch Site
3. Flight Number Vs. Orbit Type
4. Flight Number Vs. Payload
5. Payload Vs. Orbit Type

Bar Graph

1. Success Rate Vs. Orbit Type

Line Graph

1. The Launch Success Vs. Year

4.1. Build interactive map with Folium

Link para notebook

- **Visualize launch data in na interactive map**: using Latitude and Longitude of each launch site, add circle marker around the launch site and label it with the name.
- Assign launch_outcomes to classes 1 and 0 with green and red markers
- Calculate distances between launch site to its proximities. We answered some question:
  - Are launch sites near railways, highways and coastlines.
  - Do launch sites keep certain distance away from cities.

4.2. Build a dashboard with Plotly Dash

- We built an interactive dashboard using Plotly dash
- We plotted pie charts showing:
  - Total launches by launch sites and the success rate of each
  - Allow us to identify best sites to launch
- We plotted scatter graph showing:
  - Relationship between Outcome and Payload Mass (Kg)
  - Allow us to identify best payload range(s) to launch succesfully

5. Predictive Analysis - Classification

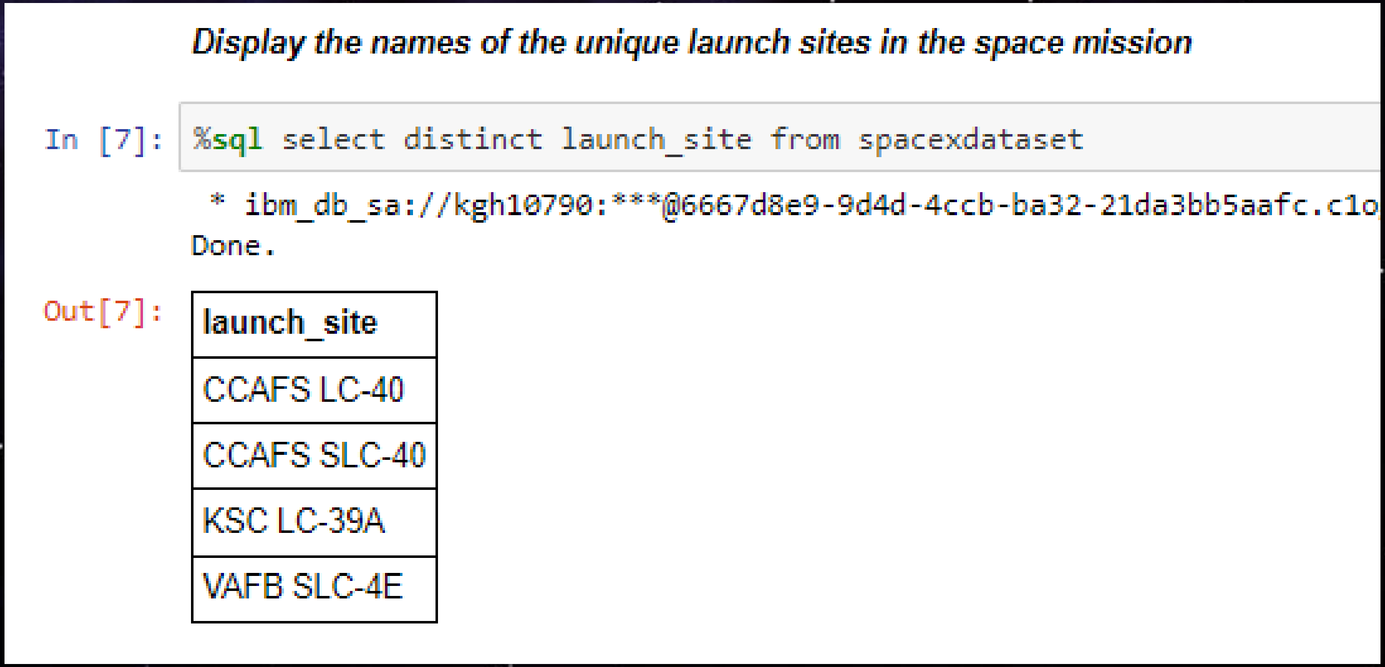
- Building the model
  - After loading and transforming the data, we split it into training and test datasets
  - Decide which techniques of machine learning we want to use
  - Set our parameter grid to find the best estimator using GridSearchCV
  - Fit datasets into GridSearchCV object and train the model
- Evaluating
  - Check accuracy for each model
  - Tune hyperparameters
  - Plot Confusion Matrix
- Identify the best model
  - Model with best accuracy

Results

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

EDA with SQL

Link para notebook



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

```
In [7]: %sql select distinct launch_site from spacexdataset

 * ibm_db_sa://kgh10790:***@6667d8e9-9d4d-4ccb-ba32-21da3bb5aafc.c1o
Done.
```

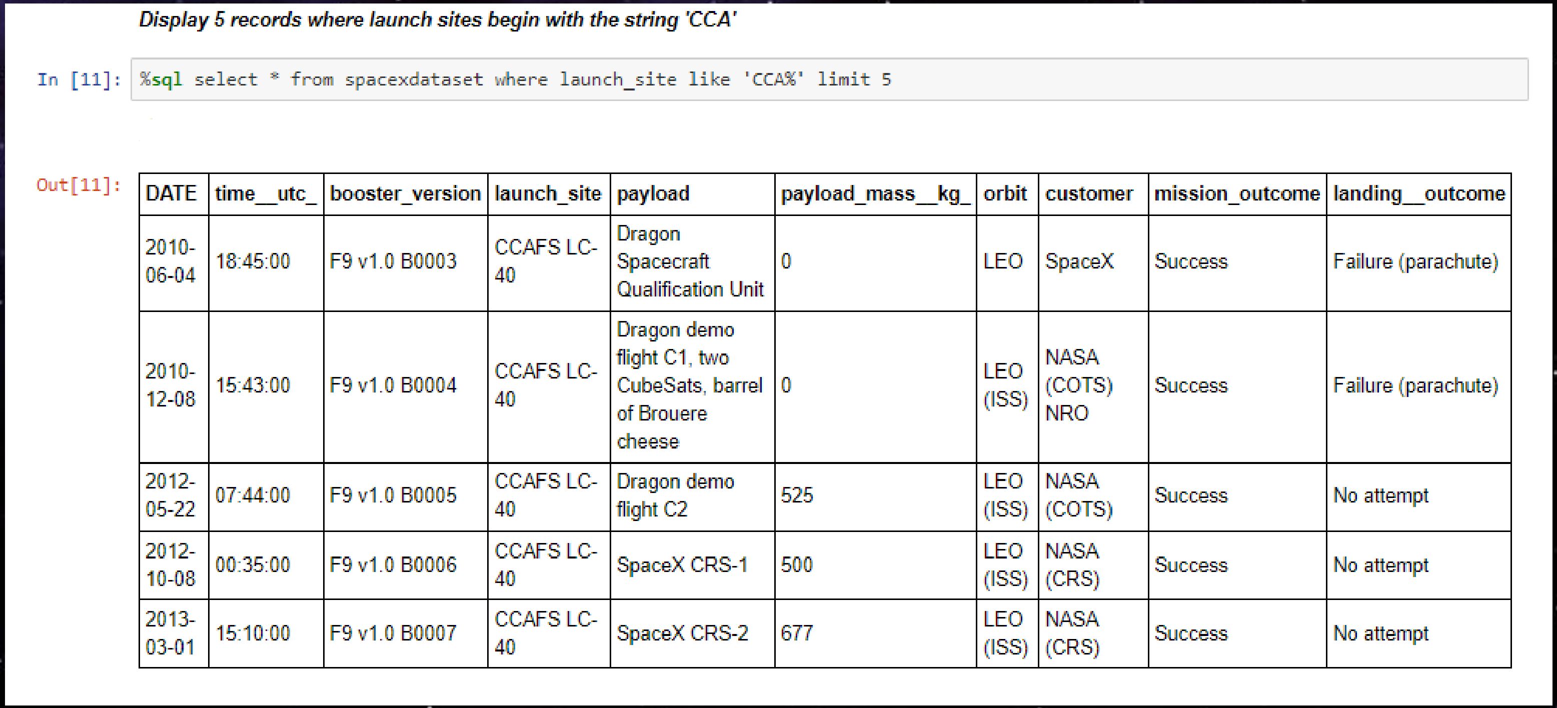
Out[7]:

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

We used the key word **DISTINCT** to show only unique values for launch_site from the SpaceX data.

EDA with SQL

Link para notebook



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

```
In [11]: %sql select * from spacexdataset where launch_site like 'CCA%' limit 5
```

Out[11]:

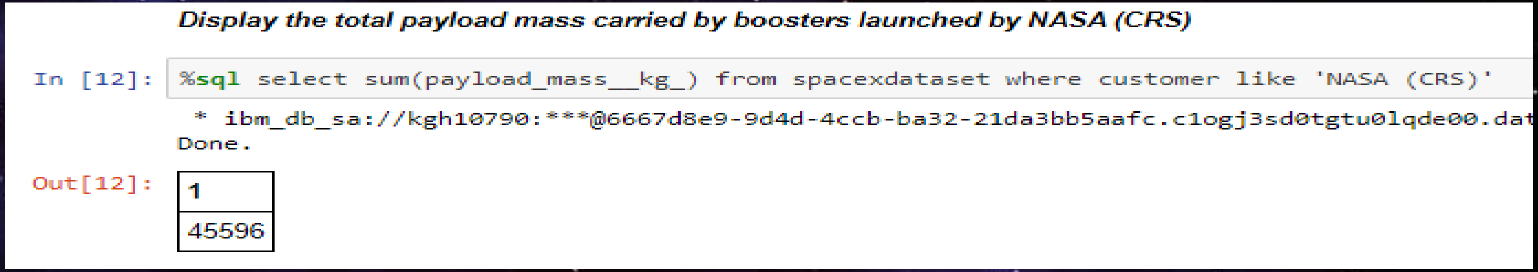
DATE	time__utc_	booster_version	launch_site	payload	payload_mass__kg_	orbit	customer	mission_outcome	landing__outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2012-05-22	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

We use LIMIT to display only 5 records

LIKE will look for records that begin with `CCA`

EDA with SQL

Link para notebook

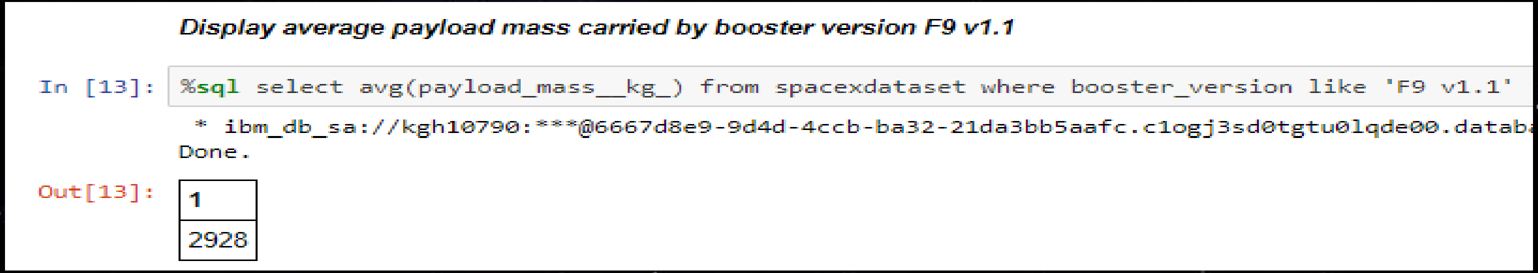


Display the total payload mass carried by boosters launched by NASA (CRS)

```
In [12]: %sql select sum(payload_mass__kg_) from spacexdataset where customer like 'NASA (CRS)'
          * ibm_db_sa://kgh10790:***@6667d8e9-9d4d-4ccb-ba32-21da3bb5aafc.c1ogj3sd0tgtu0lqde00.dat
         Done.
Out[12]:
```

1
45596

We calculated the total payload carried by boosters from NASA as **45596** using the query above



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

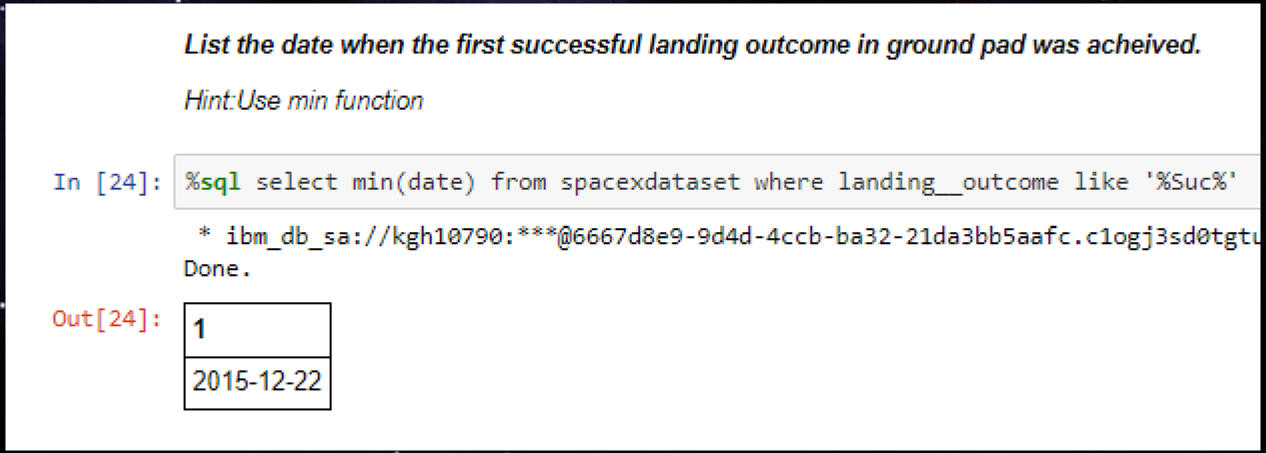
```
In [13]: %sql select avg(payload_mass__kg_) from spacexdataset where booster_version like 'F9 v1.1'
          * ibm_db_sa://kgh10790:***@6667d8e9-9d4d-4ccb-ba32-21da3bb5aafc.c1ogj3sd0tgtu0lqde00.databa
         Done.
Out[13]:
```

1
2928

We calculated the average (AVG) payload mass carried by booster version with name (LIKE) F9 v1.1 as **2928.4**

EDA with SQL

Link para notebook



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

Hint:Use min function

```
In [24]: %sql select min(date) from spacexdataset where landing__outcome like '%Suc%'

 * ibm_db_sa://kgh10790:***@6667d8e9-9d4d-4ccb-ba32-21da3bb5aafc.c1ogj3sd0tgtu
Done.
```

Out[24]:

1
2015-12-22

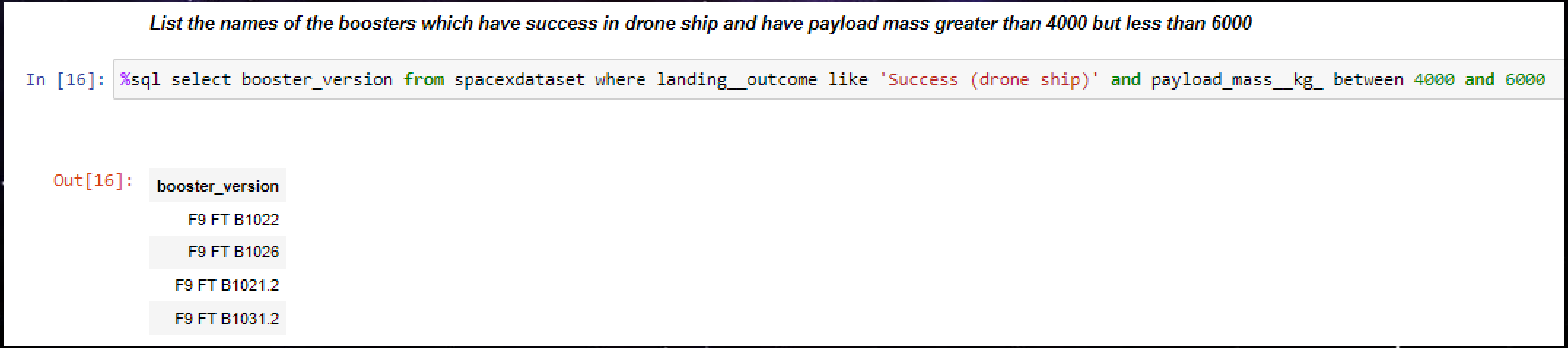
We observed that the dates of the first successful landing outcome on ground pad was 22nd December 2015

We used MIN to get the oldest date

LIKE will search for success launches

EDA with SQL

Link para notebook



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

```
In [16]: %sql select booster_version from spacexdataset where landing__outcome like 'Success (drone ship)' and payload_mass__kg_ between 4000 and 6000
```

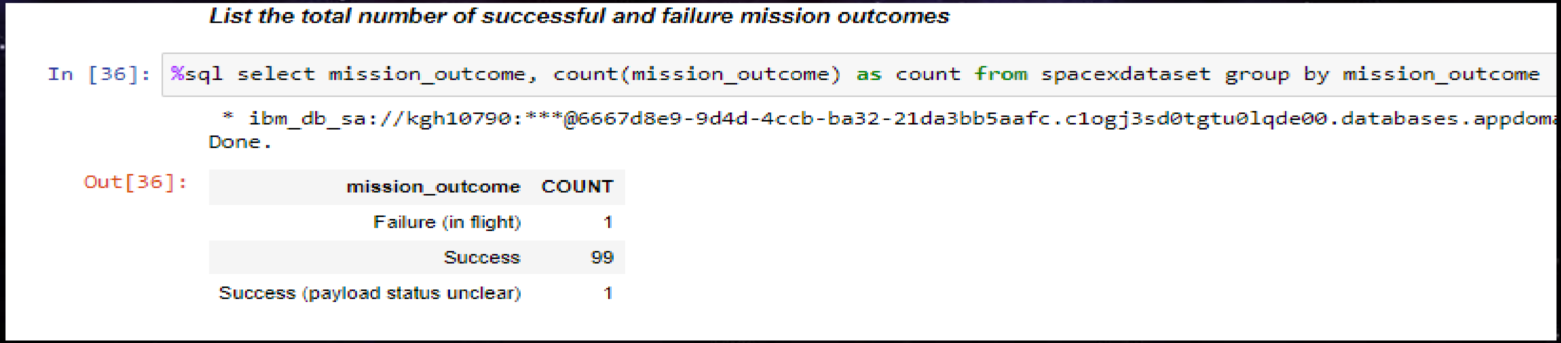
```
Out[16]:
booster_version
   F9 FT B1022
   F9 FT B1026
 F9 FT B1021.2
 F9 FT B1031.2
```

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

`

EDA with SQL

Link para notebook



List the total number of successful and failure mission outcomes

```
In [36]: %sql select mission_outcome, count(mission_outcome) as count from spacexdataset group by mission_outcome

          * ibm_db_sa://kgh10790:***@6667d8e9-9d4d-4ccb-ba32-21da3bb5aafc.c1ogj3sd0tgtu0lqde00.databases.appdoma
         Done.
```

Out[36]:

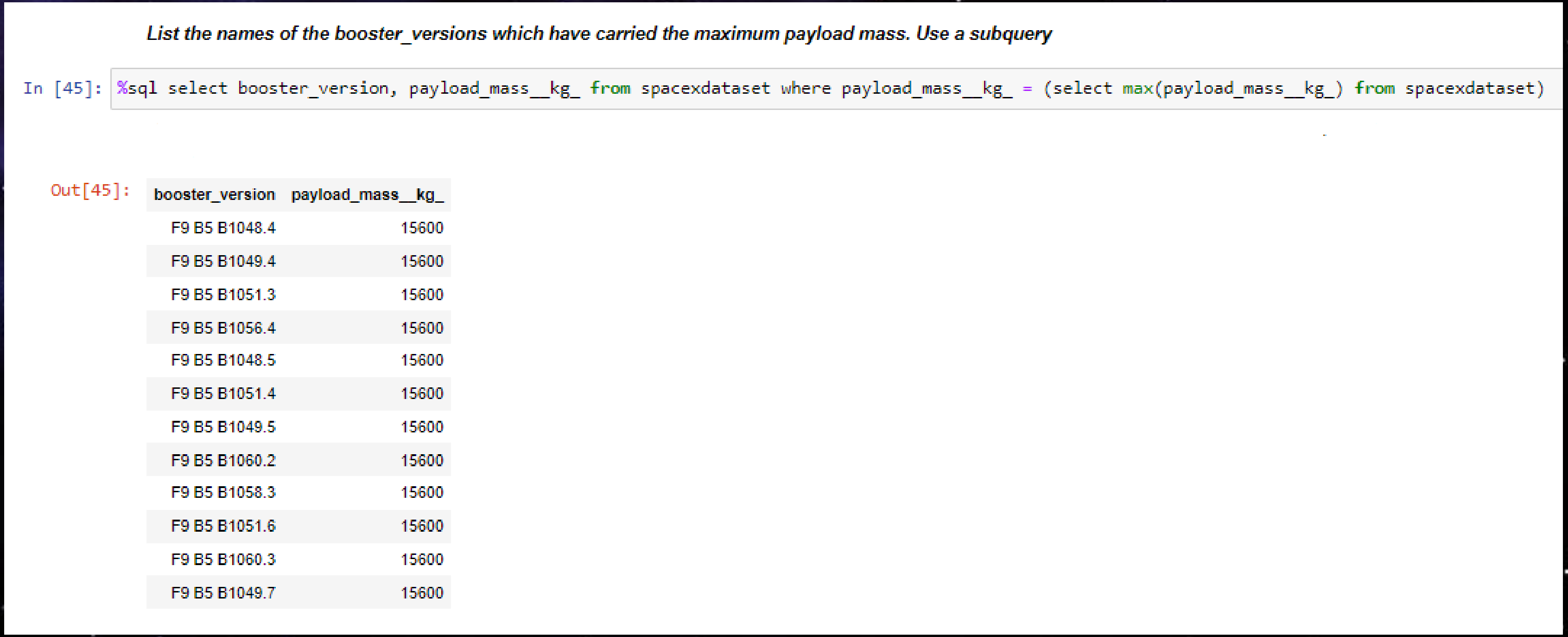
mission_outcome	COUNT
Failure (in flight)	1
Success	99
Success (payload status unclear)	1

We use COUNT to get number of each mission__outcome values

GROUP BY show the count for the column we want

EDA with SQL

Link para notebook



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

```
In [45]: %sql select booster_version, payload_mass__kg_ from spacexdataset where payload_mass__kg_ = (select max(payload_mass__kg_) from spacexdataset)
```

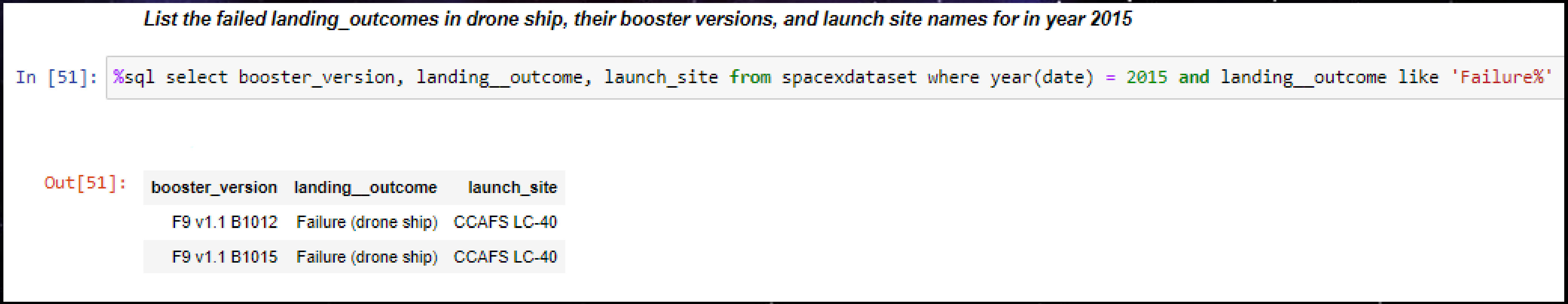
Out[45]:

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

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

EDA with SQL

Link para notebook



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

```
In [51]: %sql select booster_version, landing__outcome, launch_site from spacexdataset where year(date) = 2015 and landing__outcome like 'Failure%'
```

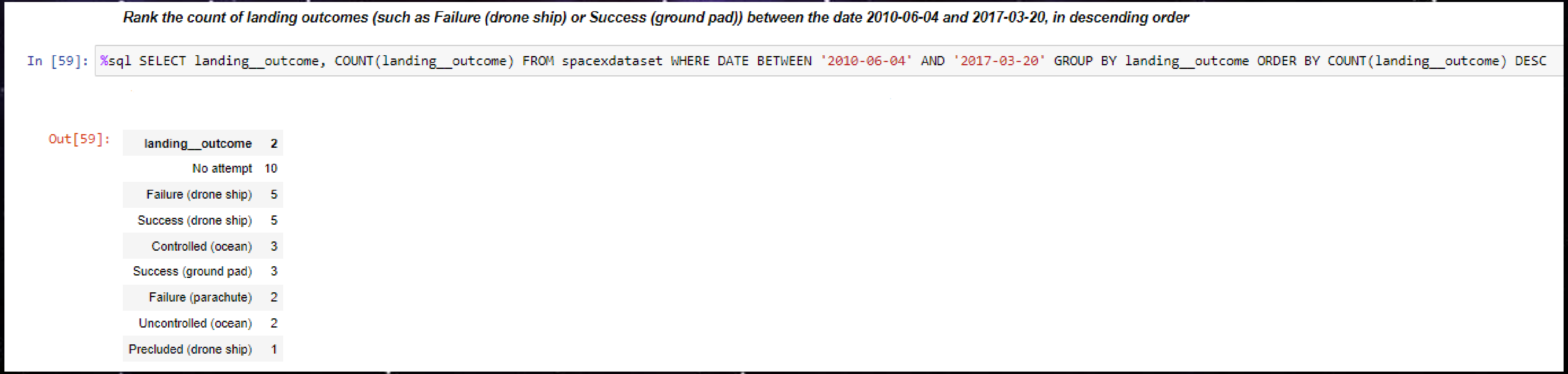
Out[51]:

booster_version	landing__outcome	launch_site
F9 v1.1 B1012	Failure (drone ship)	CCAFS LC-40
F9 v1.1 B1015	Failure (drone ship)	CCAFS LC-40

We found the boosters that have failed in drone ship in 2015 and it’s launch site using WHERE to filter the year AND failure outcome using LIKE

EDA with SQL

Link para notebook



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 [59]: %sql SELECT landing__outcome, COUNT(landing__outcome) FROM spacexdataset WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20' GROUP BY landing__outcome ORDER BY COUNT(landing__outcome) DESC
```

```
Out[59]:
      landing__outcome   2
            No attempt  10
   Failure (drone ship)  5
   Success (drone ship)  5
     Controlled (ocean)  3
   Success (ground pad)  3
    Failure (parachute)  2
   Uncontrolled (ocean)  2
  Precluded (drone ship)  1
```

We selected Landing outcomes and the COUNT of landing outcomes from the data and used the WHERE clause to filter for landing outcomes BETWEEN 2010-06-04 to 2010-03-20.

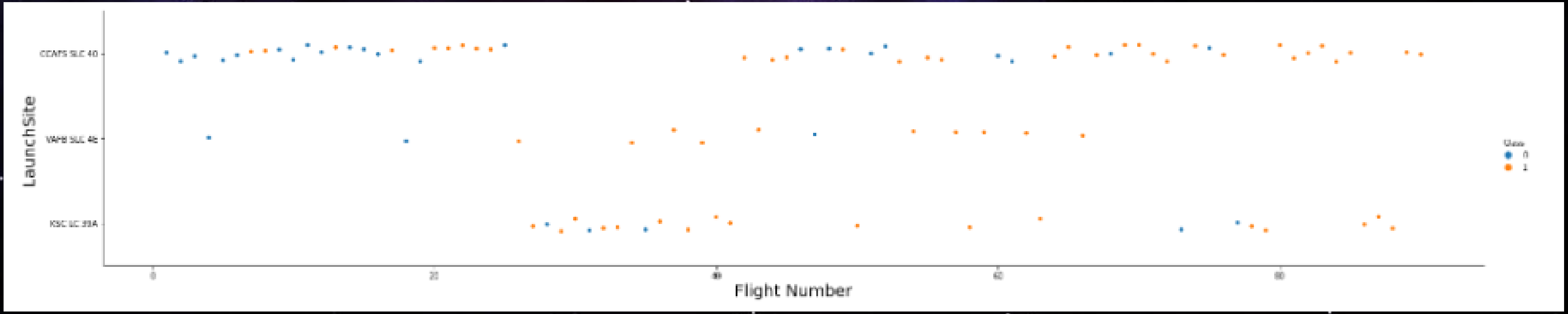
We applied the GROUP BY clause to group the landing outcomes and the ORDER BY clause to order the grouped landing outcome in descending order.

EDA with Data Visualization

Link para notebook

Scatter plots

Flight Number Vs. Launch Site



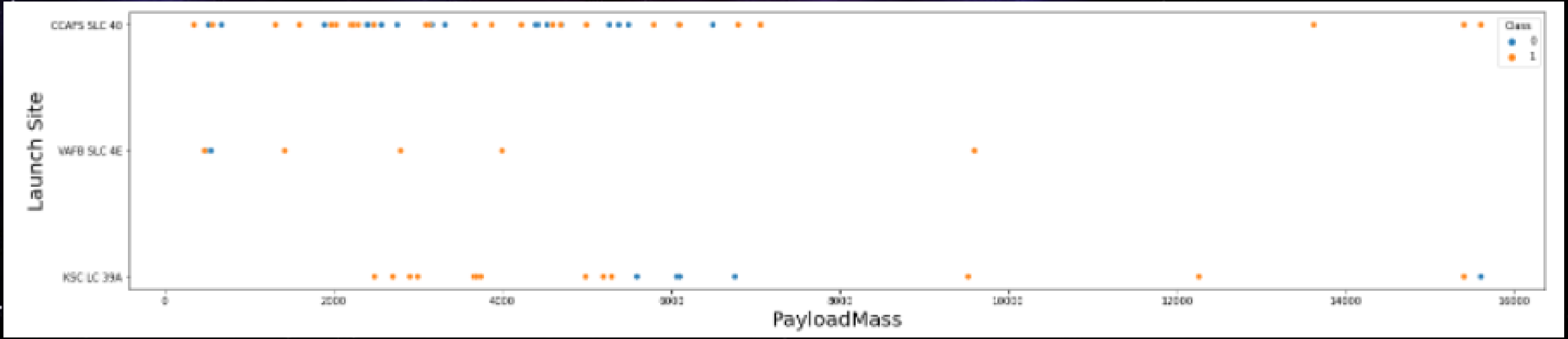
The higher amount of flight number carry out a higher succes rate for the launch

EDA with Data Visualization

Link para notebook

Scatter plots

Payload Vs. Launch Site



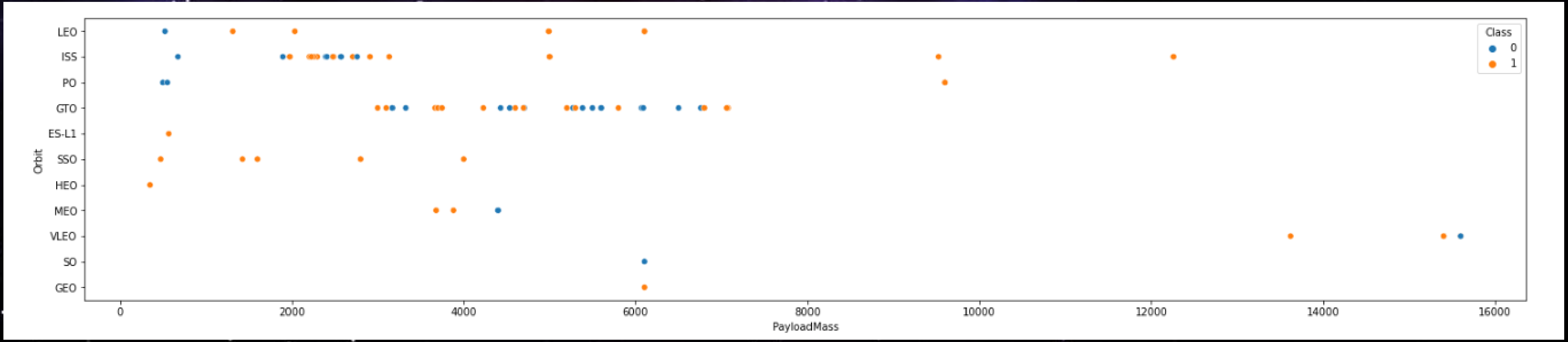
The greater the playload mass the higher the success rate for the rocket.

EDA with Data Visualization

Link para notebook

Scatter plots

Payload Vs. Orbit



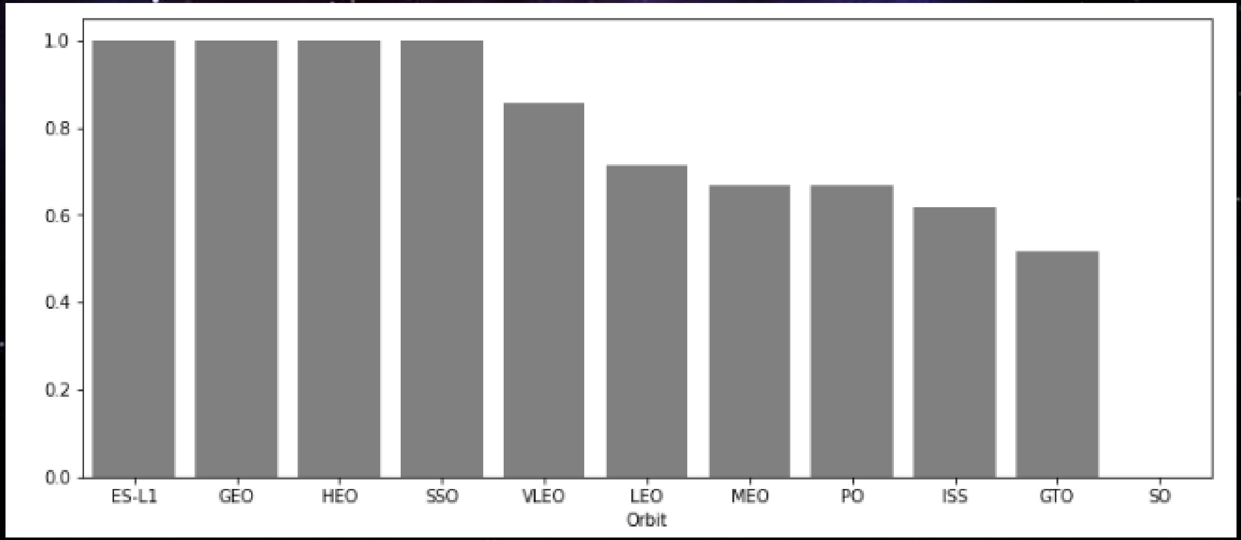
With heavy payloads, the successful landing are more for PO, LEO and ISS orbits.

EDA with Data Visualization

Link para notebook

Bar plot

Success Rate Vs. Orbit Type



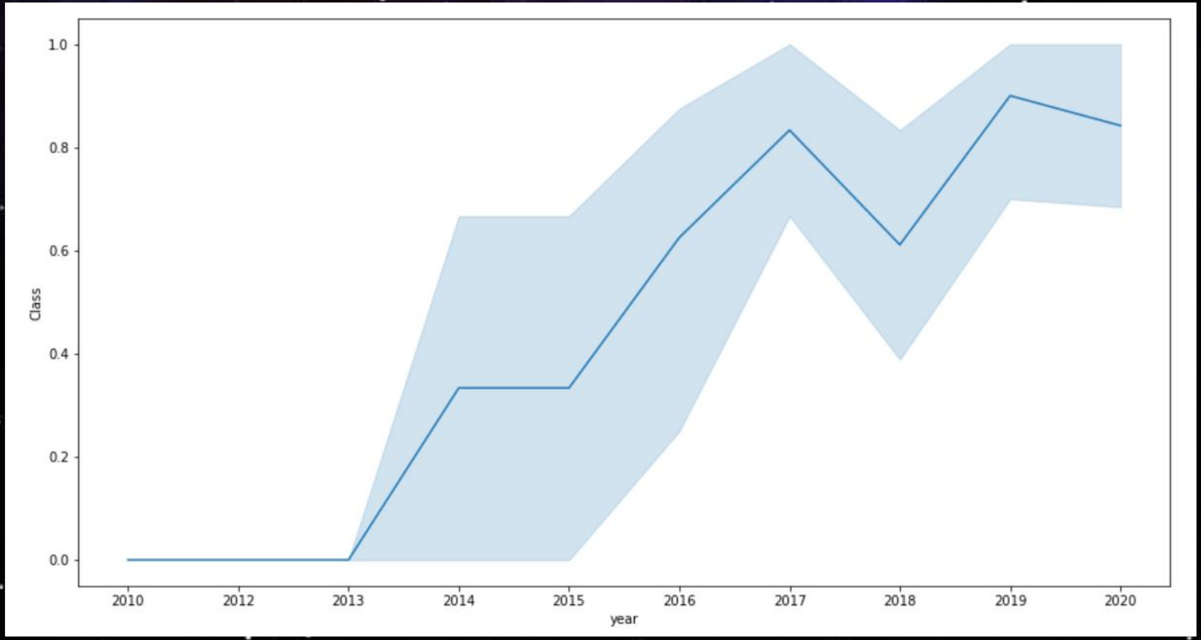
ES-L1, GEO, HEO, SSO, VLEO had the best success rate.

EDA with Data Visualization

Link para notebook

Line plot

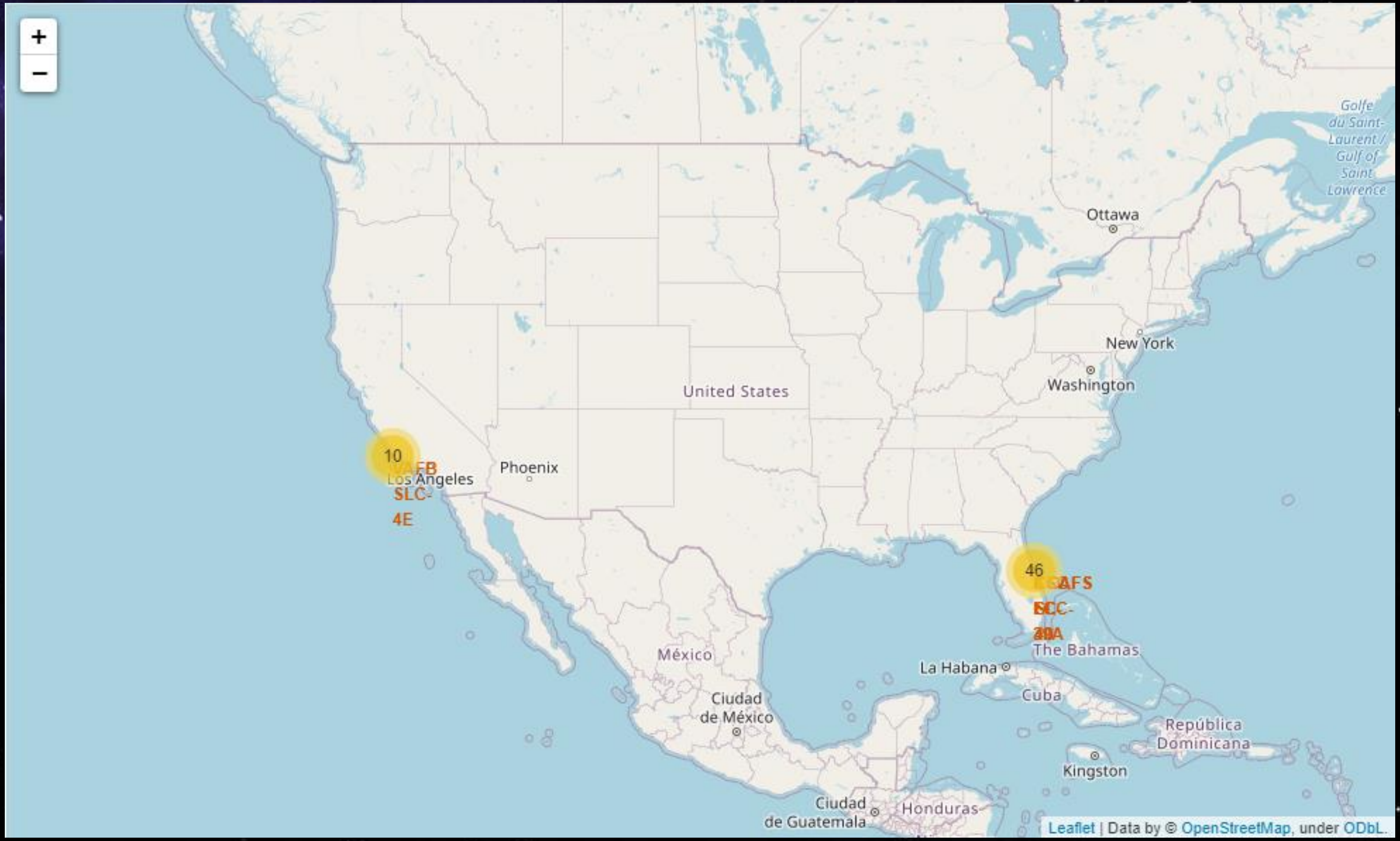
The Launch Success Vs. Year



we can observe that success rate since 2013 kept on increasing till 2020.

Interactive map with Folium

Link para notebook



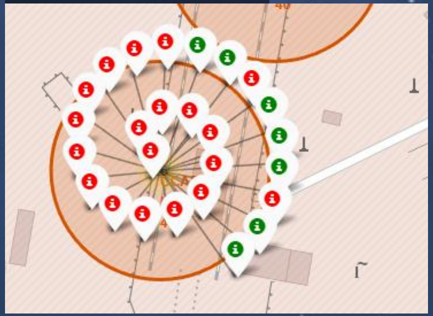
We can observe that all launch sites are located in coastal regions of the United States and thee East Side had more launches

Interactive map with Folium

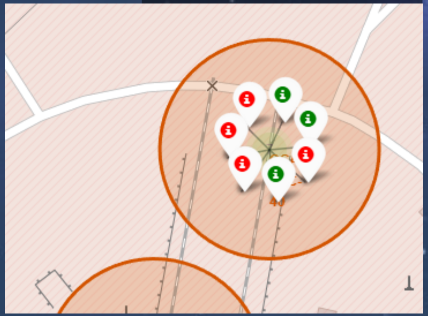
Link para notebook

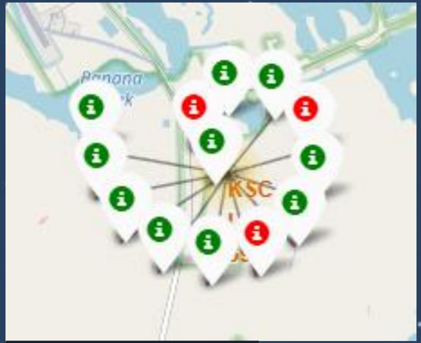
FLORIDA

CCAFS LC-40



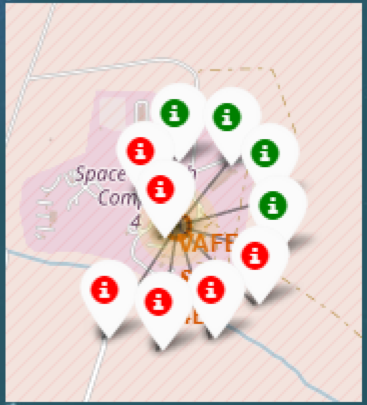
CCAFS SLC-40





KSC LC-39-A

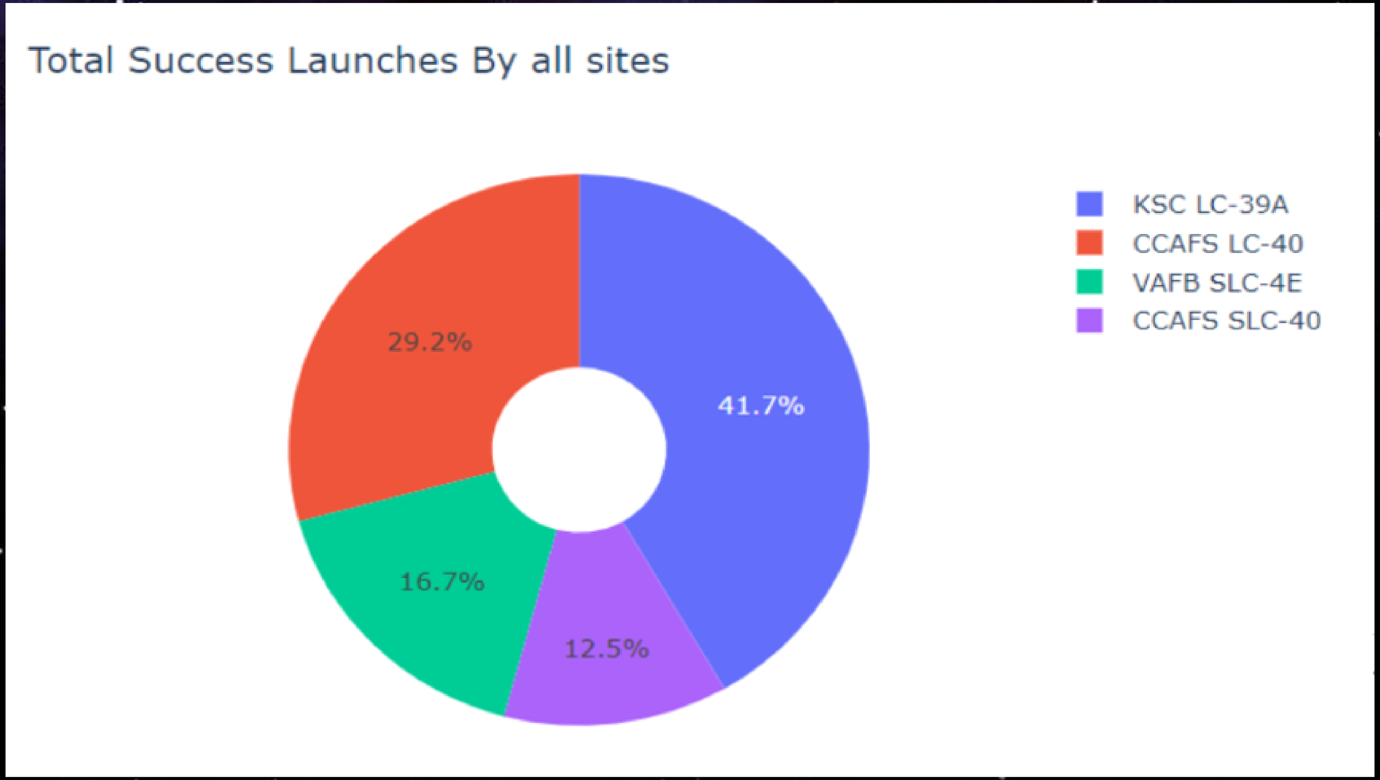
CALIFORNIA



GREEN markers shows succesful launches
RED markers show failures

Dashboard with Plotly Dash

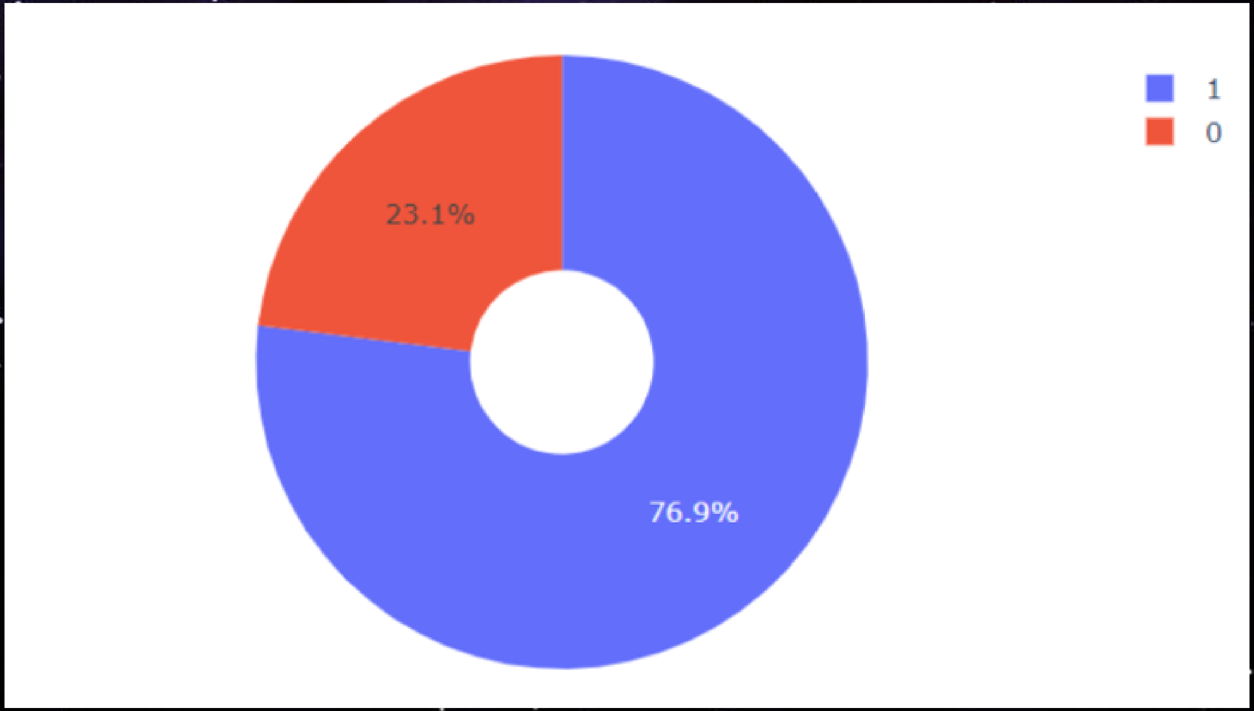
Pie chart with success rate achieved by each launch site



We can see KSC LC-39A had more successful launches than others launch sites

Dashboard with Plotly Dash

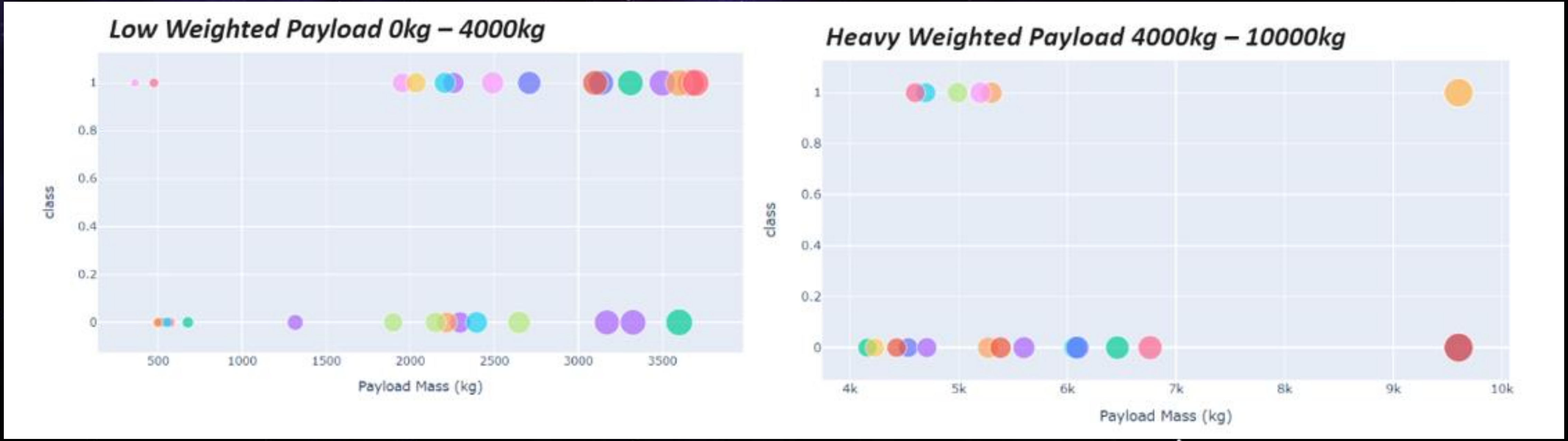
Pie chart showing KSC LC-39A launches success ratio



We can see KSC LC-39A achieve 76,9% of success rate in lauches

Dashboard with Plotly Dash

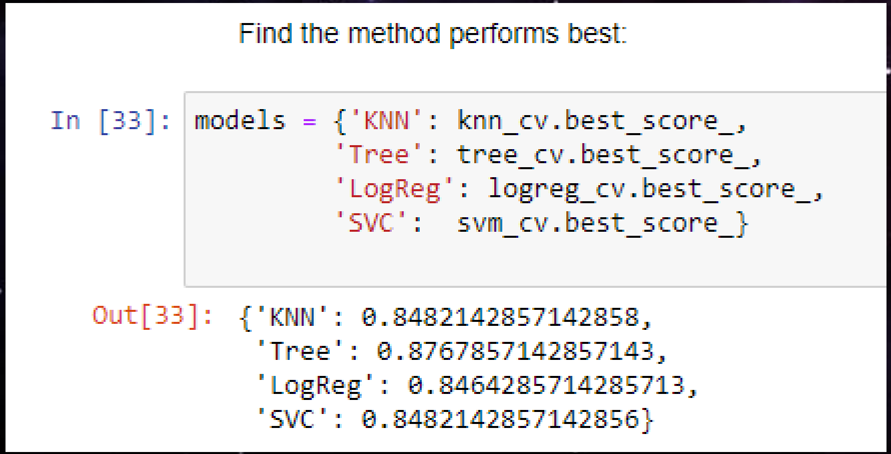
Scatter plots of Payload Vs Launch for all launch sites



Success rate for low weighted payloads is higher than heavy weighted

Classification Predict

Best method:



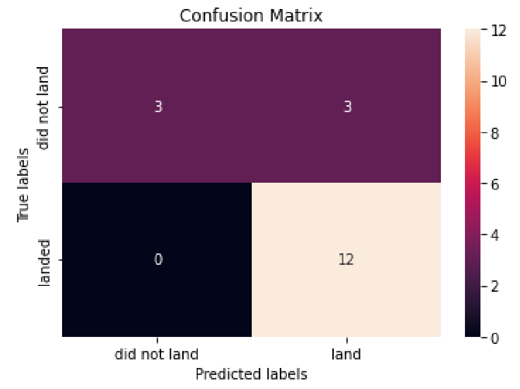
Find the method performs best:

```
In [33]: models = {'KNN': knn_cv.best_score_,
                   'Tree': tree_cv.best_score_,
                   'LogReg': logreg_cv.best_score_,
                   'SVC':  svm_cv.best_score_}

Out[33]: {'KNN': 0.8482142857142858,
          'Tree': 0.8767857142857143,
          'LogReg': 0.8464285714285713,
          'SVC': 0.8482142857142856}
```

As seen in the code above, the best technique of classification was Decision Tree, with 0.877 accuracy score

Confusion Matrix:

```
In [25]: yhat = tree_cv.predict(X_test)
         plot_confusion_matrix(Y_test,yhat)
```



The confusion matrix for decision tree shows that the classifier can distinguish betweeen different classes. The problem encountered was the difficulty of identifying false positives

Conclusion

We can conclude that:

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