IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Carmen Sánchez Pérez
25th January 2025

Outline

Executive Summary

- **Summary of methodologies**

The Data Science methodology was used for this project: Data was first collected from various sources, cleaned and prepared for analysis to get rid of null values, filter useful data, etc. The exploratory analysis on the data was perform using all the tools learned throughout the course which gave us important insights about the data. We plotted the results to visually understand our findings. Finally, machine learning models were developed to make predictions based on our data and their effectiveness was assessed during the model evaluation part.

- **Summary of all results**

Our findings determined that:

- Most launches occur near the equator since it offers energy savings for certain orbits
- Close to the sea because the first stage is likely to land safely and can be reused
- The heavier the payload, the more energy it needs. Polar orbits require more energy so polar launches do not go over a certain payload limit
- Most successful orbits include: ES-L1, GEO, HEO. Least successful seems to be GTO.

Introduction

- **Project background and context**

Space tourism and transportation is right around the corner. Space X is a pioneer company in this field but also the most profitable thanks to their inexpensive rocket launches. Our company, Space Y, would like to compete with Space X. By determining how the different factors affect the launch, we can predict if it will be successful or not and its cost, so that we can bid against Space X for a rocket launch.

- **Problems you want to find answers**

With this research we want to discover which are the most profitable launches. This includes, best places for the launch, types of rockets, whether to reuse first stage or not, and other variables that might affect cost-effective launches so that we can compete with Space X in this uprising field.

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
    - The data was collected using the SpaceX API, web scraping and get requests. Then it was converted using JSON normalization into a flat table format for easy access and manipulation.
- Perform data wrangling
    - Data was processed by handling missing values, applying format standardization, feature engineering and structuring data
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
    - By splitting the data into a test/train data, applying GridSearch to find the best hyperparameters for different models such as KNN, Regression, SVM, etc, training them and evaluating its confusion matrix to determine the best

Data Collection

Data sets were collected using python requests library to perform a GET request to said API the Space X REST API (api.spacexdata.com/v4/launches/past) we gathered the information about past launches. The data was obtained in JSON format, specifically a JSON object list, each object representing one of the past Space X launches.

We also performed webscraping using the BeautifulSoup python package to scrape HTML tables from the Space X Falcon 9 Launches Wikipedia page (https://en.wikipedia.org/wiki/List_of_Falcon_9_and_Falcon_Heavy_launches) with relevant information. After that, the gathered data was parsed and converted into a Pandas dataframe.

Data Collection – SpaceX API

- My data collection notebook:

https://github.com/carmensanpe2/Capstone/blob/main/jupyter-labs-spacex-data-collection-api.ipynb

- Github link of the SpaceX API:

https://github.com/r-spacex/SpaceX-API

- Space X REST API
  api.spacexdata.com/v4/launches/past

1. Getting response from API

```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_call_spacex_api.json'
response=requests.get(static_json_url)
```

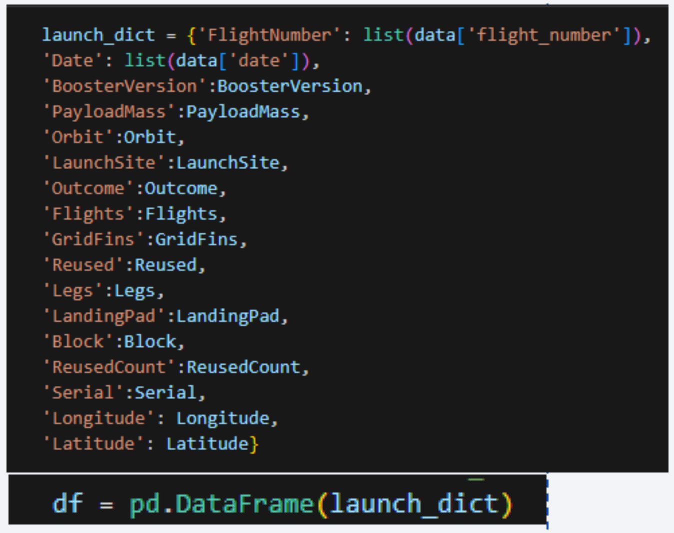
2. Converting response to .json

```
data = pd.json_normalize(response.json())
```

3. Use given functions to parse/clean data

```
getBoosterVersion(data)
getLaunchSite(data)
getPayloadData(data)
getCoreData(data)
```

4. Parsing to dictionary and dataframe



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

df = pd.DataFrame(launch_dict)
```

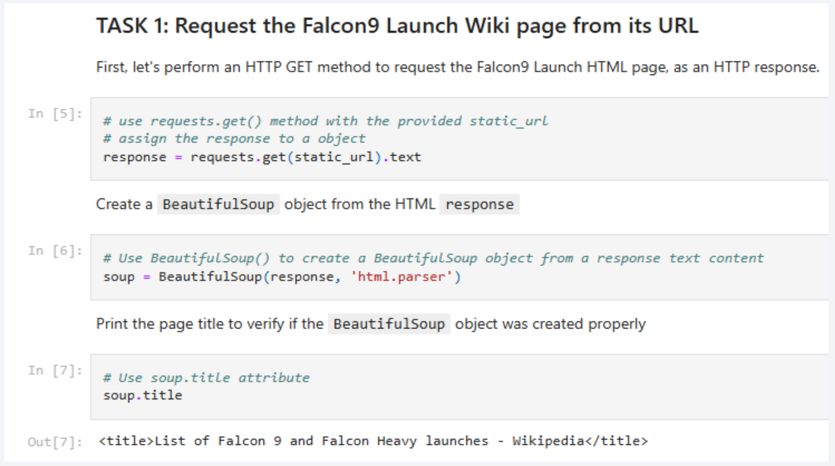
5. Filtering dataframe and exporting to flat format (.csv)

```
data_falcon9 = df[df['BoosterVersion']!='Falcon 1']
data_falcon9.to_csv('dataset_part_1.csv', index=False)
```

Data Collection - Scraping

- My webscraping notebook: https://github.com/carmensanpe2/Capstone/blob/main/jupyter-labs-webscraping.ipynb

1. Request from URL



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

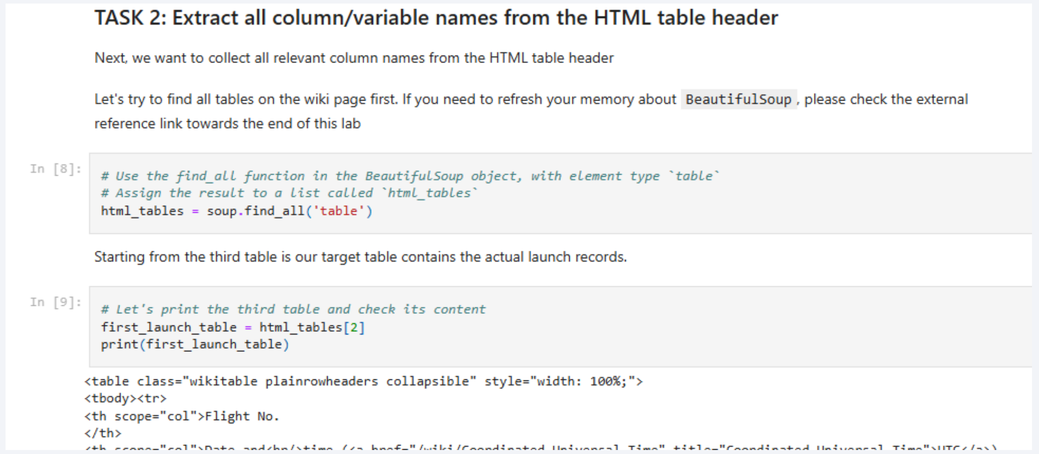
Create a `BeautifulSoup` object from the HTML `response`

```
In [6]:  # 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 [7]:  # Use soup.title attribute
         soup.title

Out[7]:  <title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

2. Extract column names from HTML table



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

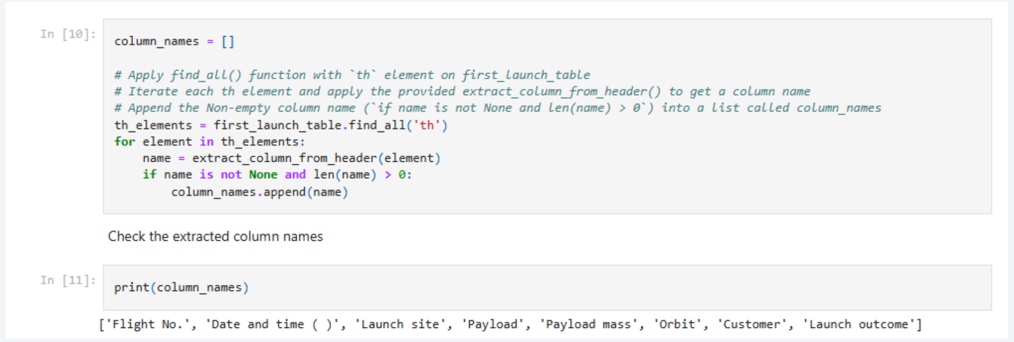
Next, we want to collect all relevant column names from the HTML table header

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')
```

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]
         print(first_launch_table)

<table class="wikitable plainrowheaders collapsible" style="width: 100%;">
<tbody><tr>
<th scope="col">Flight No.
</th>
```



```
In [10]:  column_names = []

          # Apply find_all() function with `th` element on first_launch_table
          # Iterate each th element and apply the provided extract_column_from_header() to get a column name
          # Append the Non-empty column name (`if name is not None and len(name) > 0`) into a list called column_names
          th_elements = first_launch_table.find_all('th')
          for element in th_elements:
              name = extract_column_from_header(element)
              if name is not None and len(name) > 0:
                  column_names.append(name)
```

Check the extracted column names

```
In [11]:  print(column_names)

['Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome']
```

3. Parse HTML tables and create a dataframe

(see appendix for step 3)

Data Wrangling

The data set includes several attributes such as flight N°, date, booster, payload mass, orbit, launch site, and outcome. The outcome field indicates if the first stage landed successfully and we focus on it during the data wrangling process as well as in standardizing the data:

1. Identifying missing values in each variable
2. Calculate the number of launches on each site
3. Calculate the number and occurrence of each orbit
4. Calculate the number and occurrence of mission outcomes of the orbits
5. Categorize outcome attribute by a binary classification with 1 indicating successful landing and 0 stating otherwise

My Data Wrangling notebook: https://github.com/carmensanpe2/Capstone/blob/main/labs-jupyter-spacex-Data%20wrangling.ipynb

EDA with Data Visualization

The charts plotted were:

- Flight Number and Launch Site: to see which launches have the most and the least flights
- Payload Mass and Launch Site: to see from which sites the heaviest rockets are launched and find the relationship between this variables
- Success rate of each orbit: to visualize the most and least successful ones
- Flight Number and orbit type: to see which orbits are more common and which ones are less common
- Payload Mass and Orbit type: to see if there is any weight limit for the orbits
- Yearly launch success: as to find the success trend

My EDA and Data Visualization notebooks:

https://github.com/carmensanpe2/Capstone/blob/main/edadataviz.ipynb

EDA with SQL

SQL queries performed include:

- Displaying the names of the unique launch sites in the 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 average payload mass carried by booster version F9 v1.1
- Listing the date when the first successful landing outcome 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 records which will display the month names, failure landing_outcomes in drone ship ,booster versions, launch_site for the months in year 2015.
- Ranking the count of landing outcomes between the date 2010-06-04 and 2017-03-20, in descending order.

My SQL notebook: https://github.com/carmensanpe2/Capstone/blob/main/jupyter-labs-eda-sql-coursera_sqllite.ipynb

Build an Interactive Map with Folium

The objects added to the Folium Map were:

- A marker for each launch sites in order to visualize their position on the map and check whether they are close to the equator or to the coast
- Cluster markers for each successful and failed launch to identify the sites with the highest and lowest success rate
- Lines to calculate distances between the proximities of a launch site to identify closest city, railway and highway in order as to gather some information about which is a safety distance from the launch site to this important spots.

My Interactive Map with Folium Notebook:
https://nbviewer.org/github/carmensanpe2/Capstone/blob/main/lab_jupyter_launch_site_location.ipynb

My GitHub repository (check the other link because this one does not display the interactive maps)

https://github.com/carmensanpe2/Capstone/blob/main/lab_jupyter_launch_site_location.ipynb

Build a Dashboard with Plotly Dash

- First a dropdown bar was added in order to see which launch site out the four ones obtained during the collection, wrangling and EDA process has the largest success count.
- After that we used a callback function that renders a dynamic pie chart that depends on what is selected in the previous dropdown bar, allowing us to visualize launch success counts per [selected] site
- Then a range slider to select payload was added in order to study the interaction between success rate and payload mass
- Finally another callback was added to render a scatter plot displaying the relationship between payload and success for all sites

My Plotly Dashboard code:
https://github.com/carmensanpe2/Capstone/blob/main/spacex_dash_app.py

Predictive Analysis (Classification)

1. Standardization: data was normalized to ensure equal contribution from all features
2. Train/Test split: data was randomly divided into a 80% training set and 20% testing set, which was later used to see how precise the model was for unseen data.
3. Different algorithms were train with the data, those included logistic regression, K-nearest neighbor (KNN), Decision Tree Classifiers and Support Vector Machines (SVM)
4. For each algorithm, GridSearch was used to find the best hyperparameters so that the model performed at its best.
5. After each algorithm training, their performance was evaluated using the test data to measure their accuracy
6. Finally, the confusion matrix was generated for each case which allowed to visualize the performance accuracy of each algorithm and the best option was found

My Predictive Analisys notebook:
https://github.com/carmensanpe2/Capstone/blob/main/SpaceX_Machine%20Learning%20Prediction_Part_5.ipynb

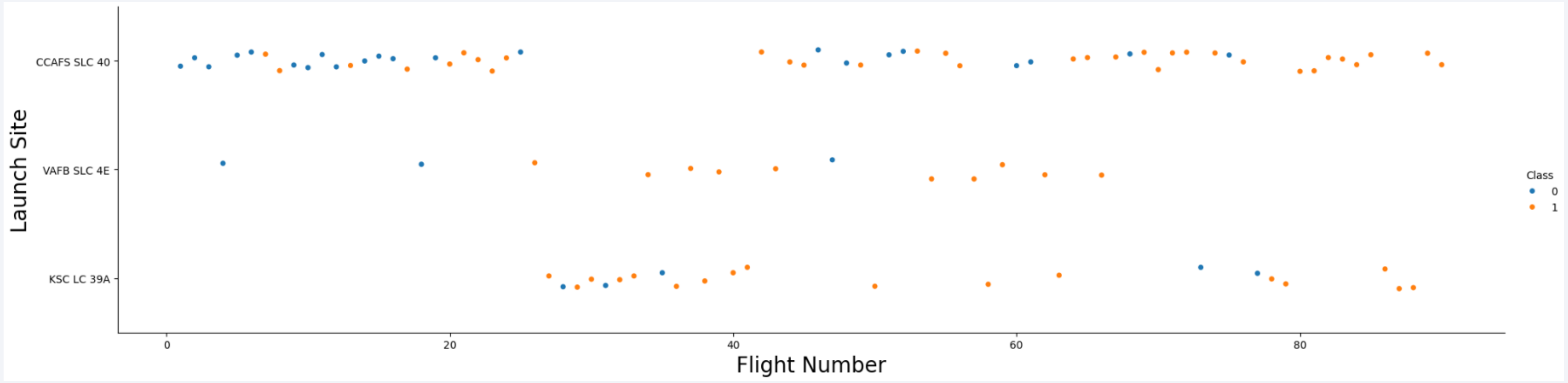
Results

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

Section 2

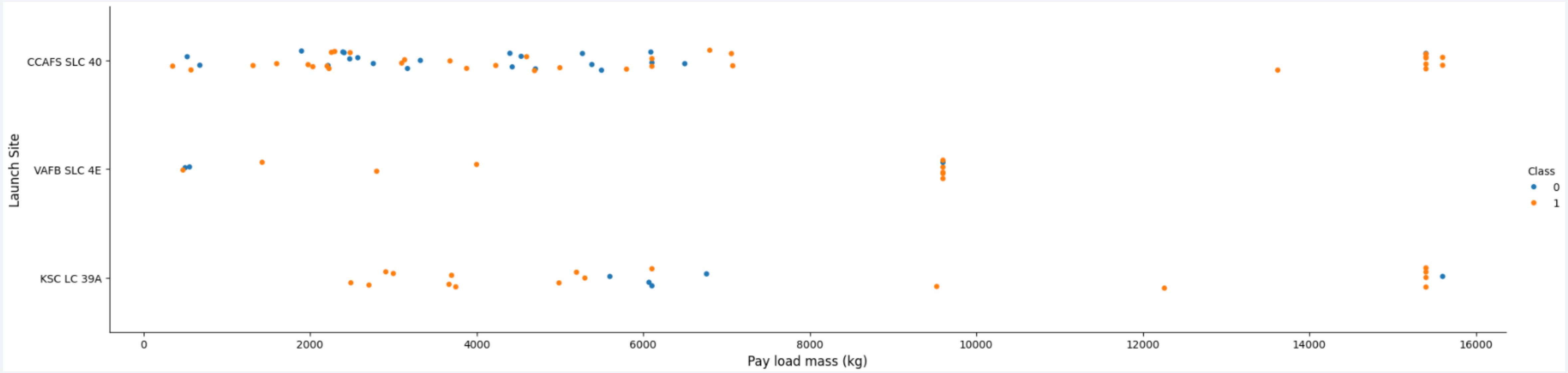
Insights drawn from EDA

Flight Number vs. Launch Site



- Most launches occur from CCAFS and KSC since they are closer the equator. This translates into energy savings for equatorial and geostationary orbits. The first flights were unsuccessful, but they say, there is no failure without learning a lesson, hence the more flights, the more is known and the more likely to be successful.
- The least number of launches occur from the VAFB launch site and seem to be more sporadic. However it also seen that the first flights for this site were unsuccessful and most later ones successful

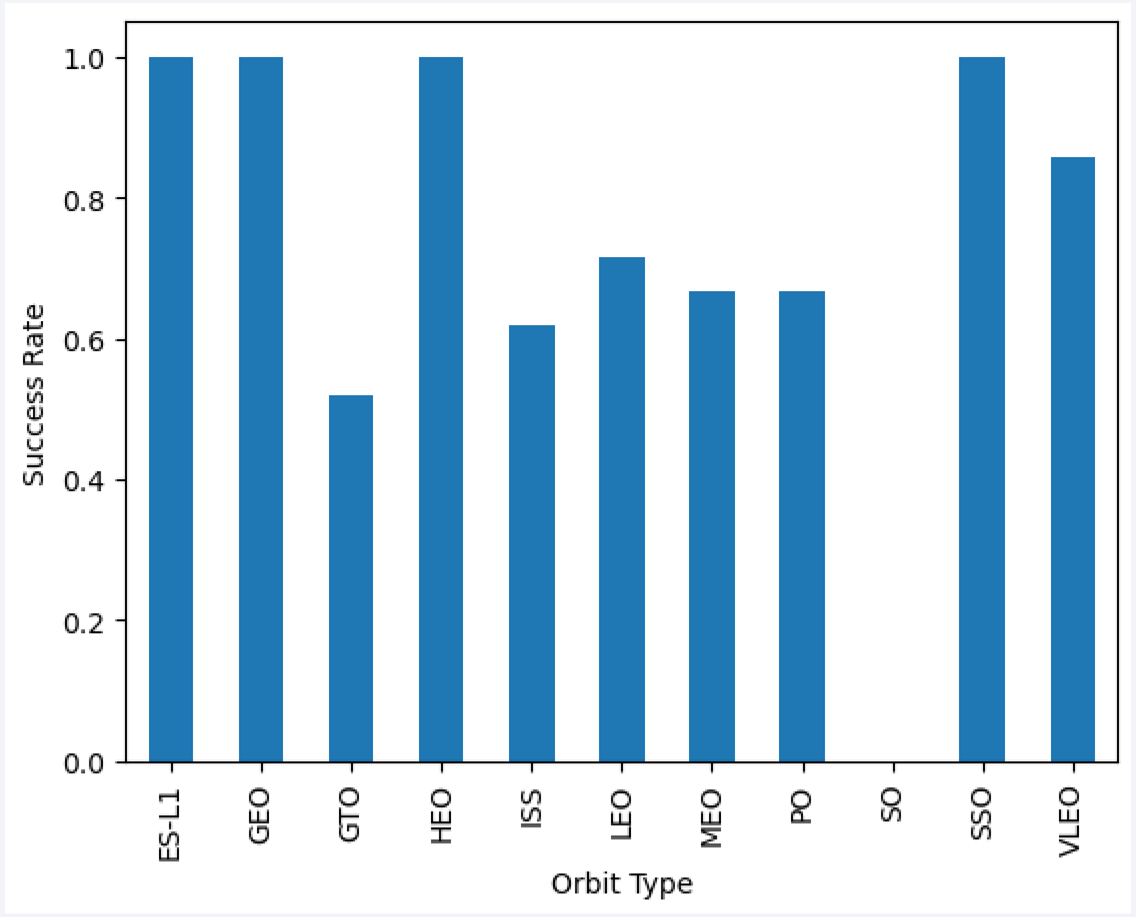
Payload vs. Launch Site



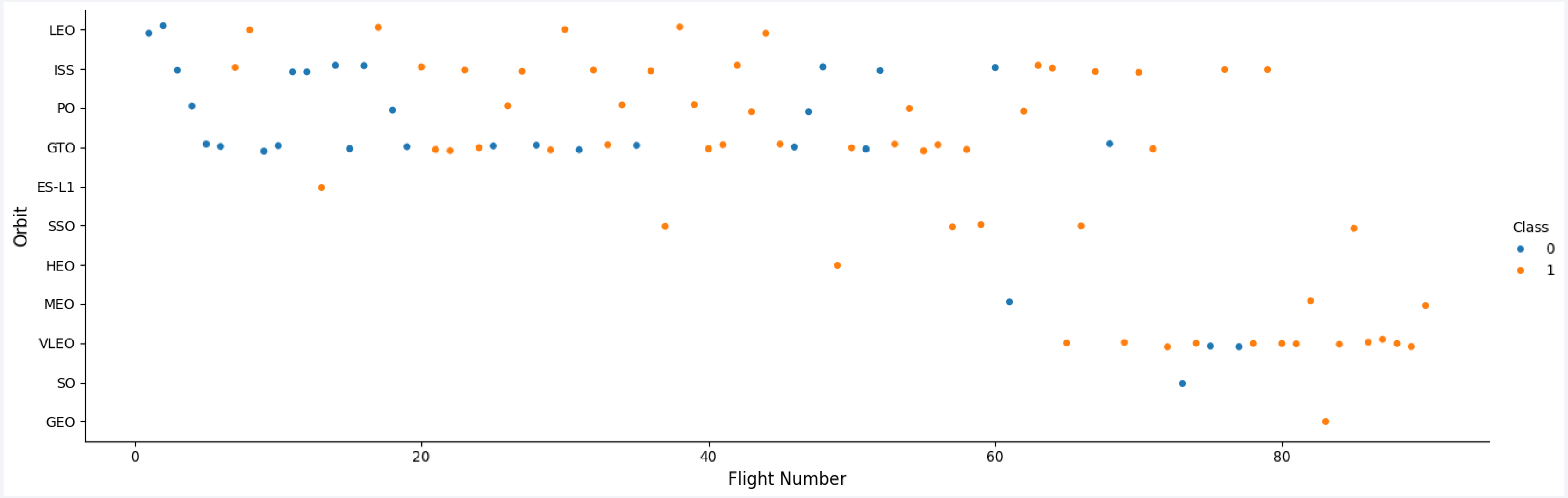
It can be observed that there are not launches from the VAFB site above 10k kg of payload mass. Most launches are between the1k-8k kg range and just a few over 14k for CCAFS and KSC launch sites.

Success Rate vs. Orbit Type

- ES-L1, GEO, HEO and SSO orbits have the highest success rate.
- Lowest success rate occurs for GTO orbit

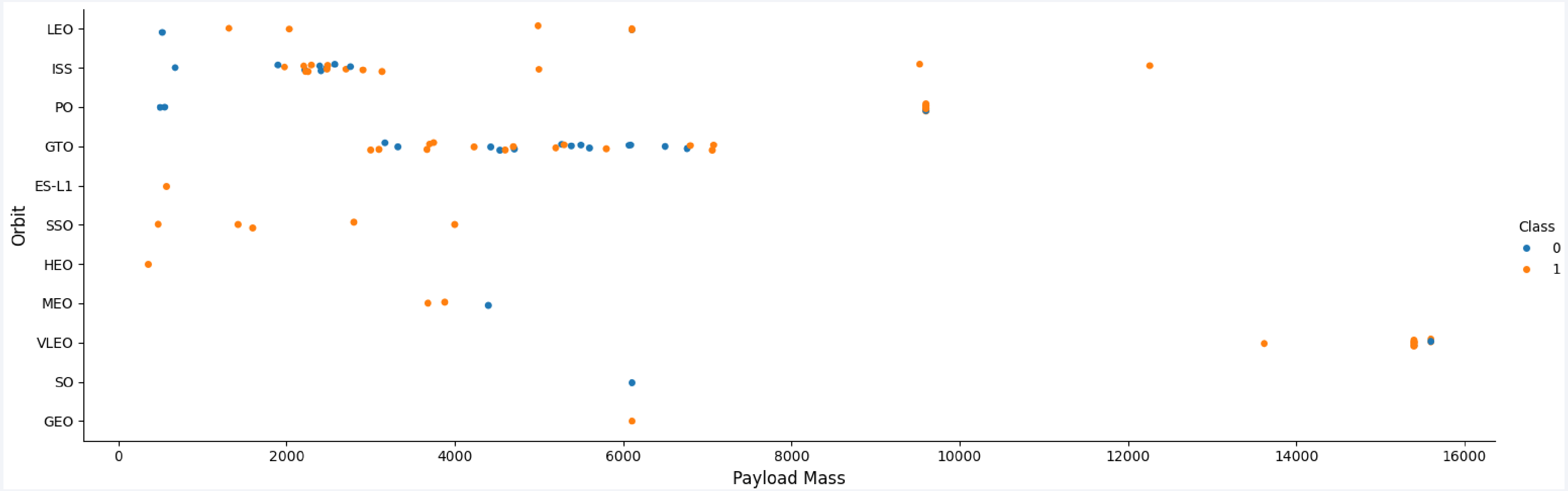


Flight Number vs. Orbit Type



For the LEO orbit we clearly see that the more the number of flights, the more possible it is to be successful. However for the rest of the orbits, such as the GTO orbit, the relationship between flight number and orbit type is not so clear and they are even likely to be unrelated.

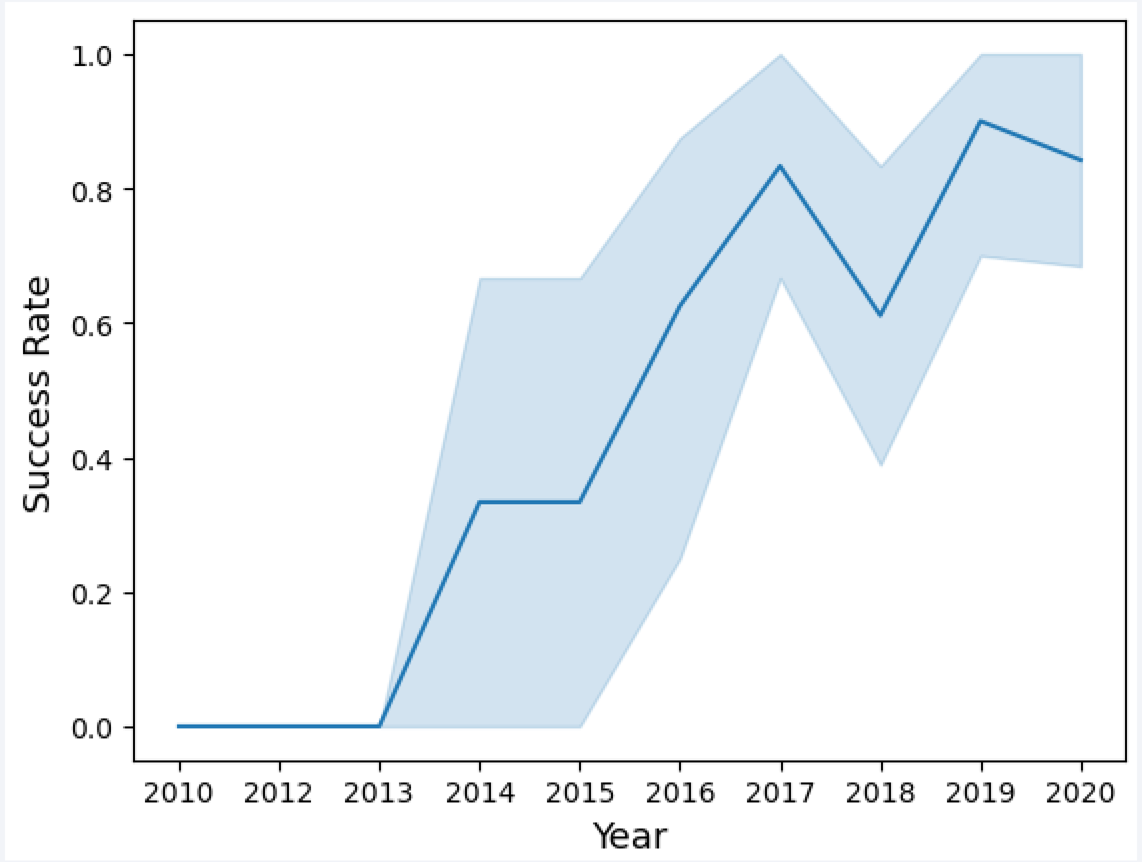
Payload vs. Orbit Type



For heavy payloads, successful landings are more common for Polar, VLEO and ISS orbits. For the GTO orbit, the relationship is unclear but we can see there are no payload mass heavier than 8k kg for this orbit. Similarly, SSO orbits are commonly used for lighter payload mass, up to 6k kg, as seen on the graph

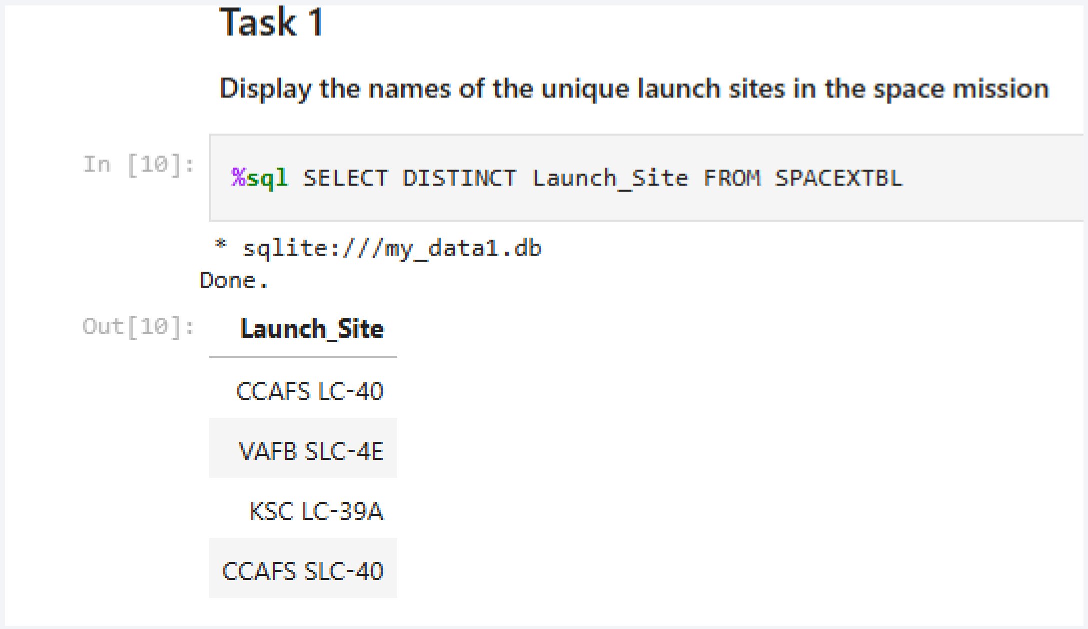
Launch Success Yearly Trend

- The success rate has increased every year since 2013 until 2020, except for 2018
- 2013 was the year that Space X started to massively grow and had over 50 programmed commercial launches, becoming a main competitor in the field by decreasing prices. That year same year the Falcon 9 first stage reuse became a thing and Elon Musk also announced their aim to reach the planet Mars
- At the end of 2018, one of the Falcon 9 launches was unsuccessful due to a malfunction of a booster. This occurred for the B5 booster version.



All Launch Site Names

For this and all the following presented queries, we use the %sql magic command for python which allows using SQL language directly in python. For this case, we found all the different launch sites using the SELECT statement and the DISCTINCT clause:



Task 1

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

```
In [10]: %sql SELECT DISTINCT Launch_Site FROM SPACEXTBL
```

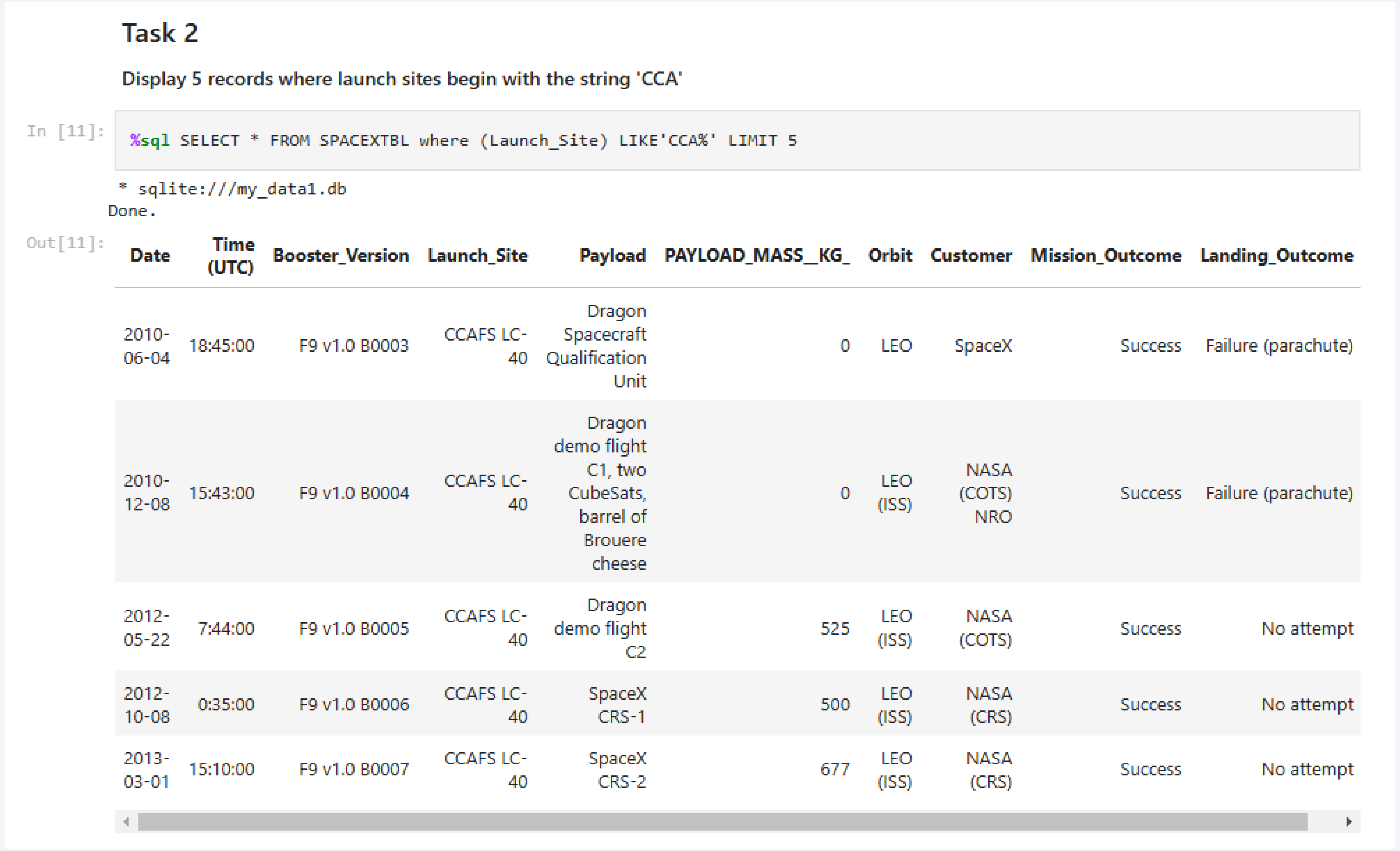
```
 * sqlite:///my_data1.db
Done.
```

Out[10]:

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

Launch Site Names Begin with 'CCA'

- With a SELECT statement and use the WHERE clause to filter by Launch Site, the CCA% indicates we consider all launches sites starting by CCA without caring about the rest of the name. LIMIT 5 limits our results to 5 results



Task 2

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

```
In [11]: %sql SELECT * FROM SPACEXTBL where (Launch_Site) LIKE'CCA%' LIMIT 5
```

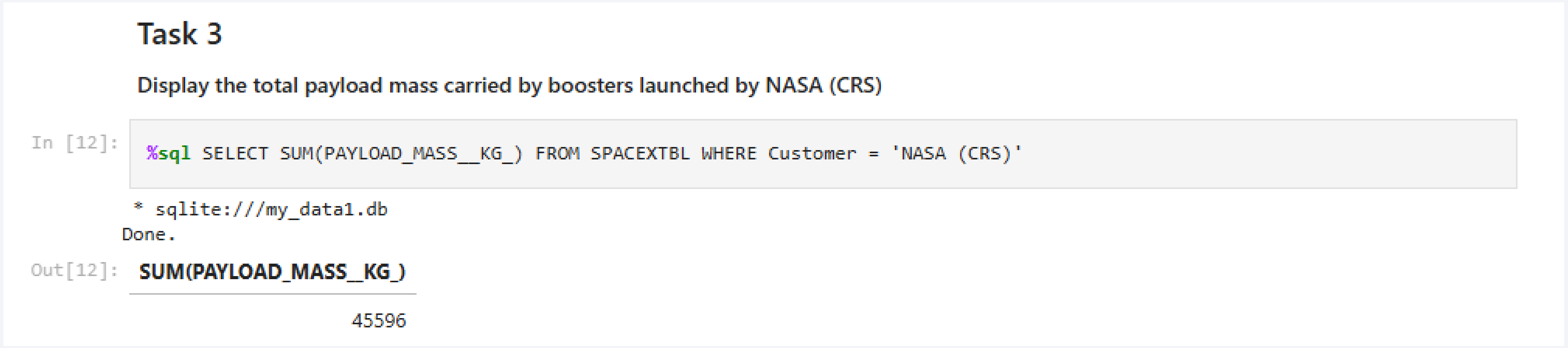
```
 * sqlite:///my_data1.db
Done.
```

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	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	0:35:00	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

Total Payload Mass by NASA (CRS) boosters

We use the SUM(PAYLOAD_MASS__KG_) to sum over all the payload mass carried by boosters launches by NASA (CRS)



Task 3

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

```
In [12]: %sql SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE Customer = 'NASA (CRS)'
```

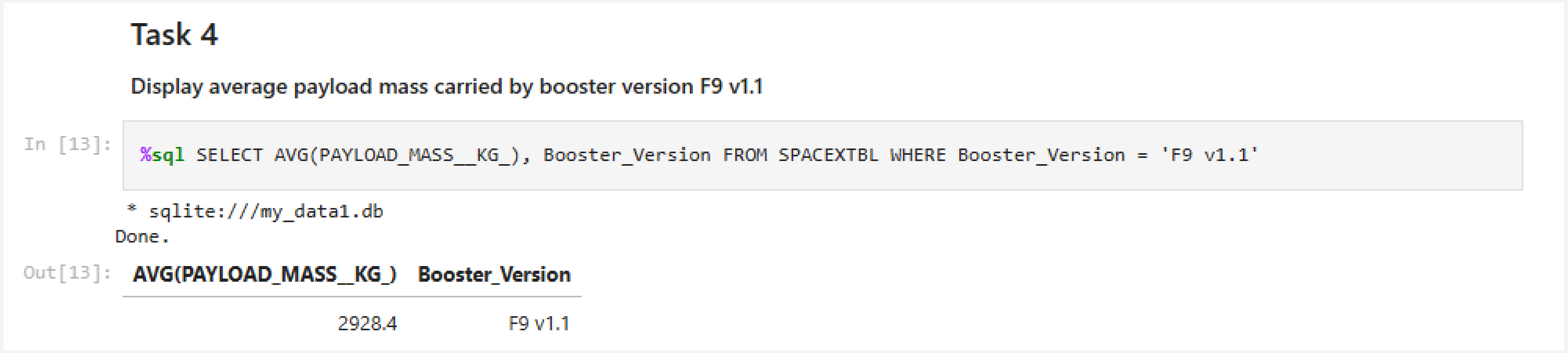
```
 * sqlite:///my_data1.db
Done.
```

Out[12]:

SUM(PAYLOAD_MASS__KG_)
45596

Average Payload Mass by F9 v1.1

Similarly, the average payload mass carried by booster version F9 v.1.1 is calculated by using the AVG function



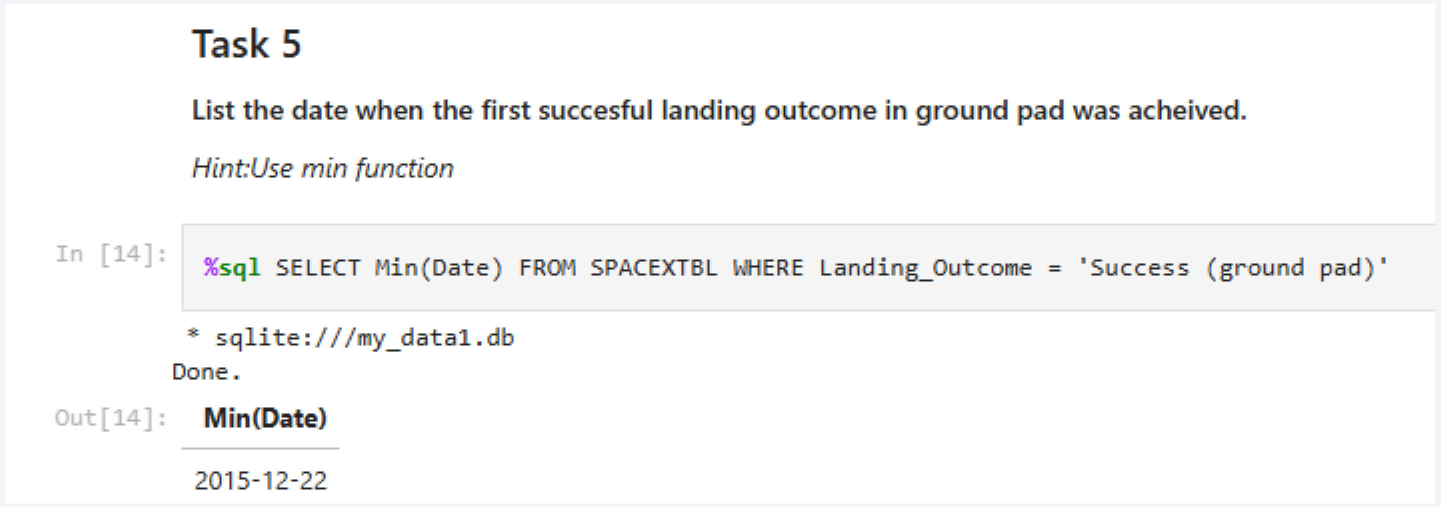
Task 4

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

```
In [13]: %sql SELECT AVG(PAYLOAD_MASS__KG_), Booster_Version FROM SPACEXTBL WHERE Booster_Version = 'F9 v1.1'

 * sqlite:///my_data1.db
Done.

Out[13]: AVG(PAYLOAD_MASS__KG_)  Booster_Version
                         2928.4          F9 v1.1
```

First Successful Ground Landing Date

We use the MIN function to select the oldest date with a Success (ground pad) landing outcome



Task 5

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

Hint:Use min function

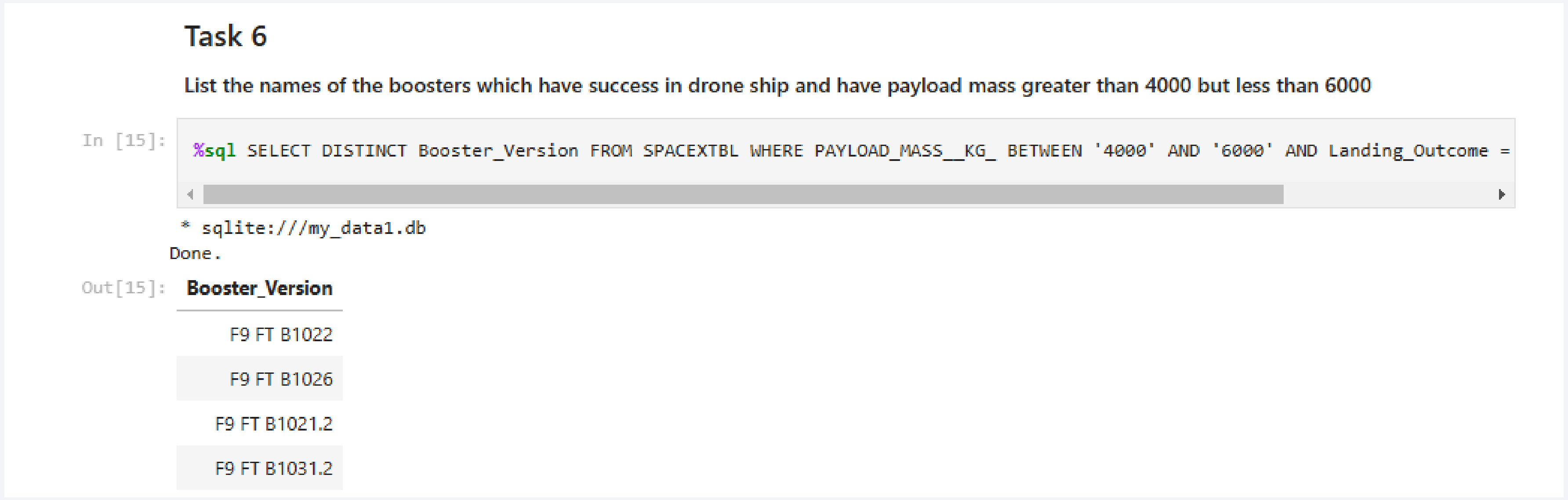
```
In [14]: %sql SELECT Min(Date) FROM SPACEXTBL WHERE Landing_Outcome = 'Success (ground pad)'

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

Out[14]:

Min(Date)
2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

We filter by 4000 kg and 6000 kg payloads using the BETWEEN clause and the AND to add an additional filter of Landing outcome = 'Success (drone ship)'


Task 6

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

```
In [15]: %sql SELECT DISTINCT Booster_Version FROM SPACEXTBL WHERE PAYLOAD_MASS__KG_ BETWEEN '4000' AND '6000' AND Landing_Outcome =
```

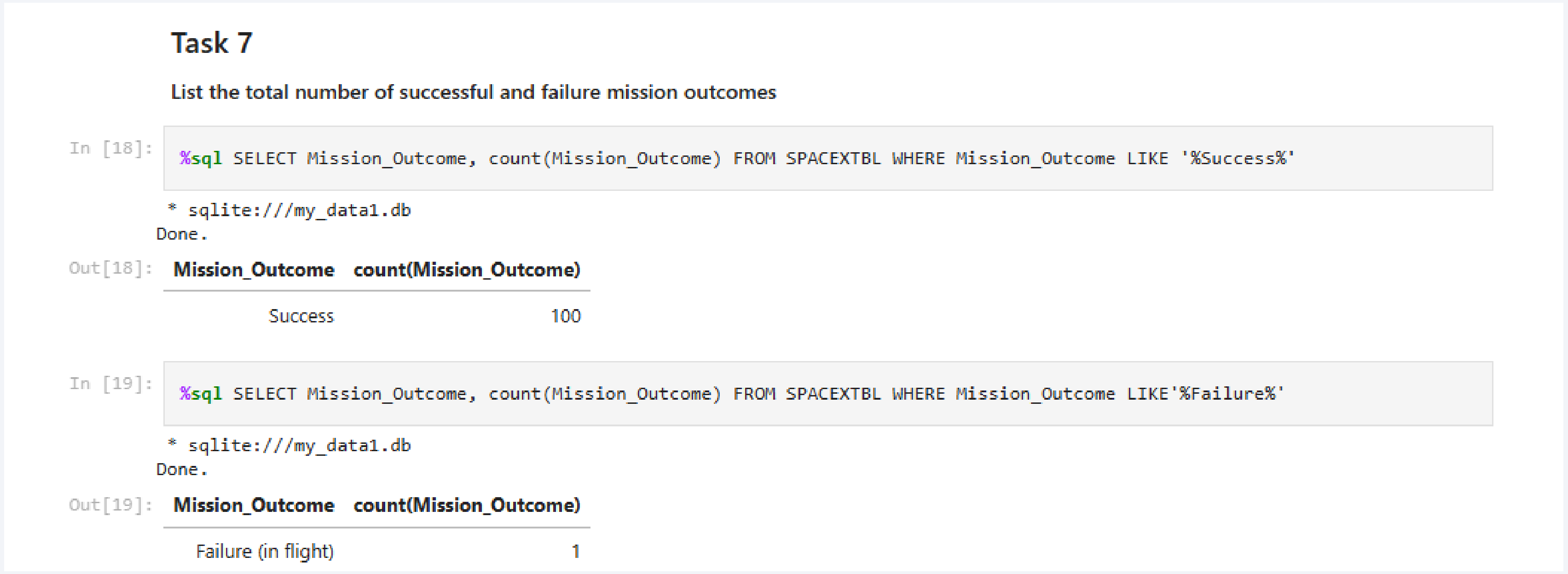
```
 * sqlite:///my_data1.db
Done.
```

Out[15]:

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

Total Number of Successful and Failure Mission Outcomes

The COUNT function is used, like its name states, to count the total number of Mission Outcomes, filtering with the WHERE and LIKE clauses by Successful Mission Outcome. Similarly, we do it for failed missions too:



Task 7

List the total number of successful and failure mission outcomes

```
In [18]: %sql SELECT Mission_Outcome, count(Mission_Outcome) FROM SPACEXTBL WHERE Mission_Outcome LIKE '%Success%'

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

Out[18]:

Mission_Outcome	count(Mission_Outcome)
Success	100

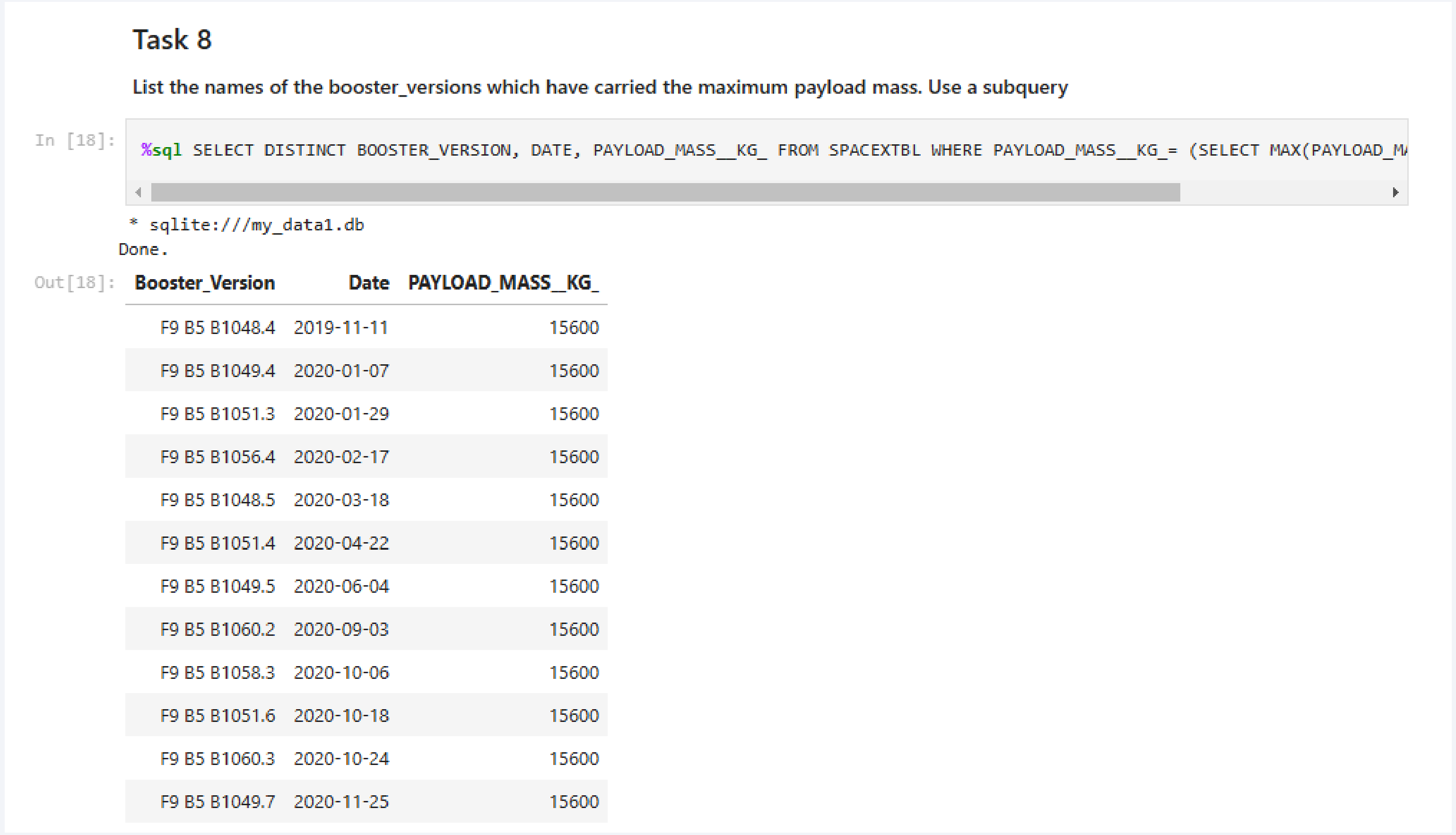
```
In [19]: %sql SELECT Mission_Outcome, count(Mission_Outcome) FROM SPACEXTBL WHERE Mission_Outcome LIKE'%Failure%'

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

Out[19]:

Mission_Outcome	count(Mission_Outcome)
Failure (in flight)	1

Boosters Carried Maximum Payload

We select different booster_versions with the DISCTINCT clause. Filtering is done by the WHERE clause, which uses a nested query where we are calling the MAX function to obtained the maximum payload



Task 8

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

```
In [18]: %sql SELECT DISTINCT BOOSTER_VERSION, DATE, PAYLOAD_MASS__KG_ FROM SPACEXTBL WHERE PAYLOAD_MASS__KG_= (SELECT MAX(PAYLOAD_M
```

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

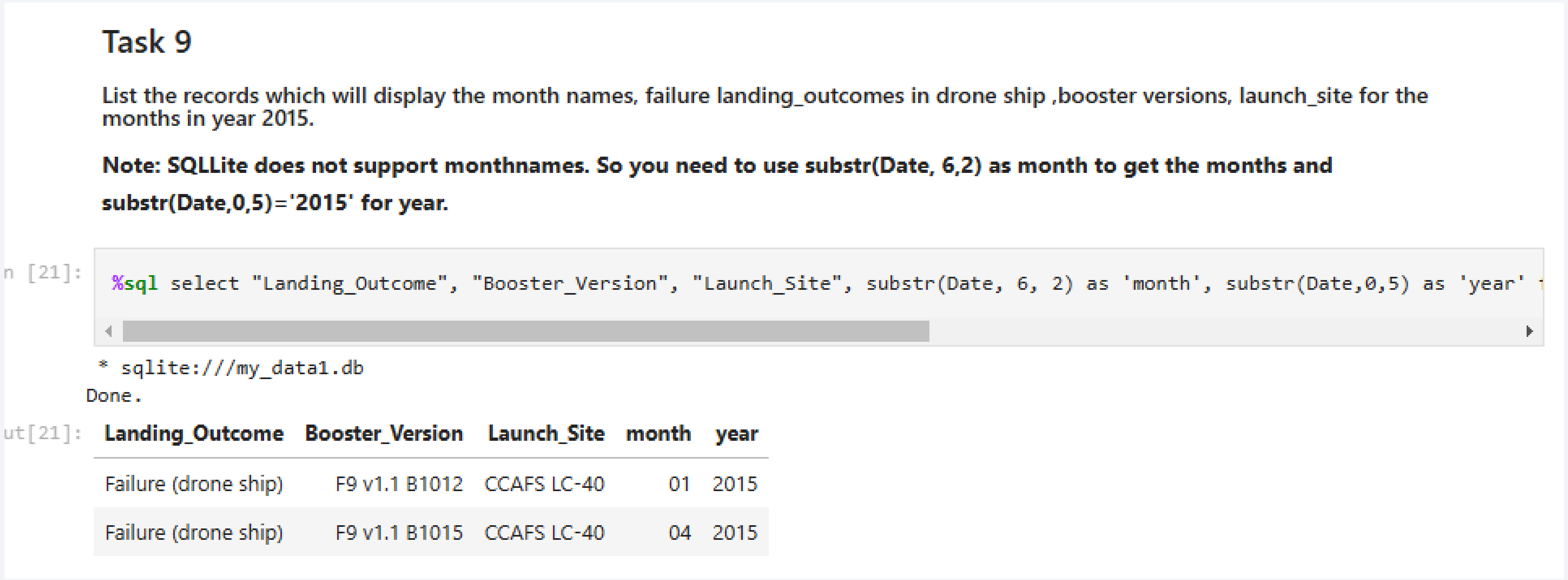
Out[18]:

Booster_Version	Date	PAYLOAD_MASS__KG_
F9 B5 B1048.4	2019-11-11	15600
F9 B5 B1049.4	2020-01-07	15600
F9 B5 B1051.3	2020-01-29	15600
F9 B5 B1056.4	2020-02-17	15600
F9 B5 B1048.5	2020-03-18	15600
F9 B5 B1051.4	2020-04-22	15600
F9 B5 B1049.5	2020-06-04	15600
F9 B5 B1060.2	2020-09-03	15600
F9 B5 B1058.3	2020-10-06	15600
F9 B5 B1051.6	2020-10-18	15600
F9 B5 B1060.3	2020-10-24	15600
F9 B5 B1049.7	2020-11-25	15600

```
%sql SELECT DISTINCT BOOSTER_VERSION,
DATE, PAYLOAD_MASS__KG_ FROM SPACEXTBL
WHERE PAYLOAD_MASS__KG_=
(SELECT      MAX(PAYLOAD_MASS__KG_)
FROM SPACEXTBL)
```

2015 Launch Records

The **substr** is used to extract the desired part of the Date column, by indicating the starting position and the length. We filter by year 2015 with the same reasoning and by landing outcome 'Failure (drone ship):



Task 9

List the records which will display the month names, failure landing_outcomes in drone ship ,booster versions, launch_site for the months in year 2015.

Note: SQLLite does not support monthnames. So you need to use substr(Date, 6,2) as month to get the months and substr(Date,0,5)='2015' for year.

```
n [21]:  %sql select "Landing_Outcome", "Booster_Version", "Launch_Site", substr(Date, 6, 2) as 'month', substr(Date,0,5) as 'year'
```

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

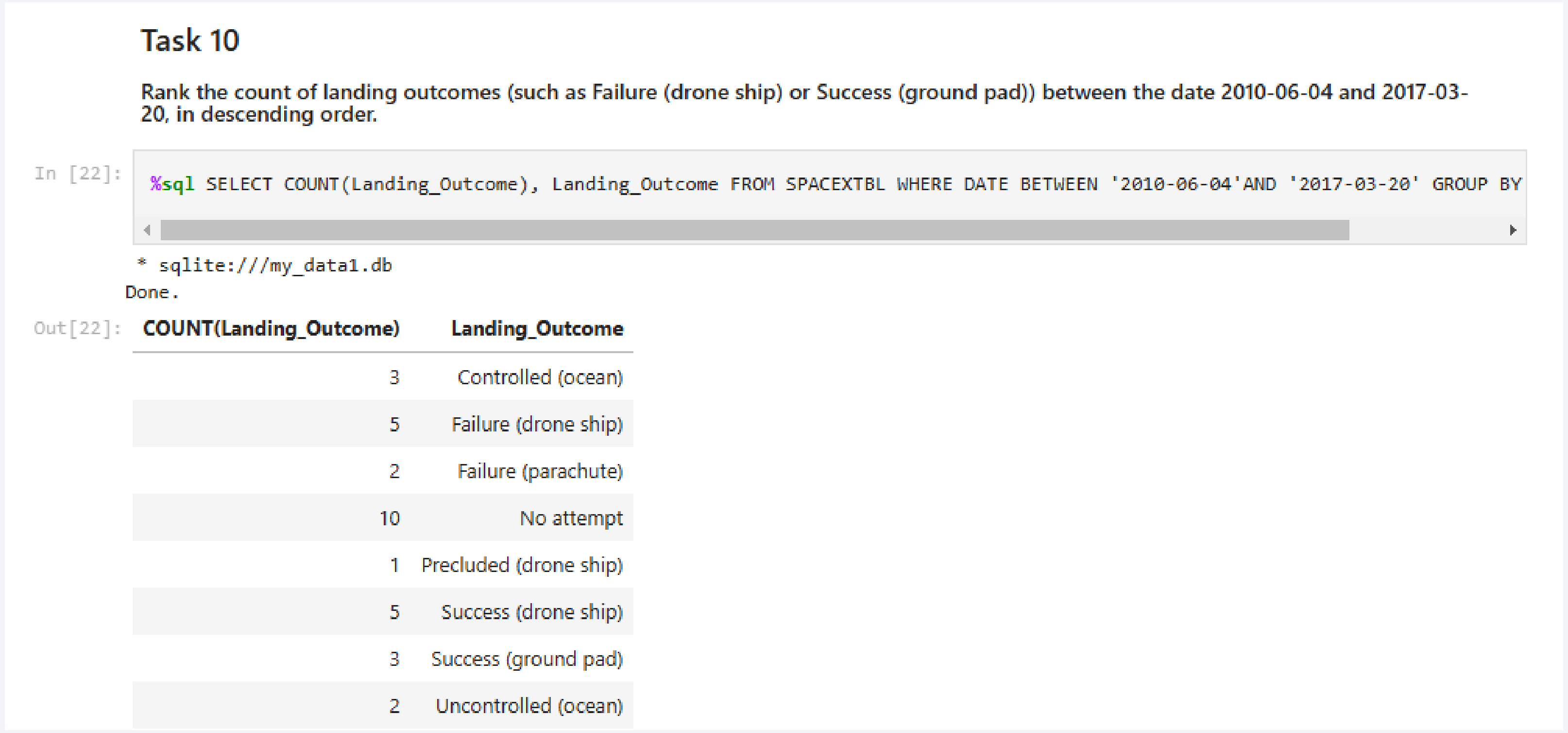
ut[21]:

Landing_Outcome	Booster_Version	Launch_Site	month	year
Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40	01	2015
Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40	04	2015

```
%sql SELECT "Landing_Outcome", "Booster_Version", "Launch_Site", substr(Date, 6, 2) as 'month', substr(Date,0,5) as 'year' FROM
SPACEXTABLE WHERE Landing_Outcome = "Failure (drone ship)" and substr(Date,0,5) = '2015'
```

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

For ranking 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 we use again the COUNT function and filter the dates accordingly using the BETWEEN clause and grouping by landing outcome using the GROUP BY clause



Task 10

Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order.

```
In [22]: %sql SELECT COUNT(Landing_Outcome), Landing_Outcome FROM SPACEXTBL WHERE DATE BETWEEN '2010-06-04'AND '2017-03-20' GROUP BY
```

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

Out[22]:

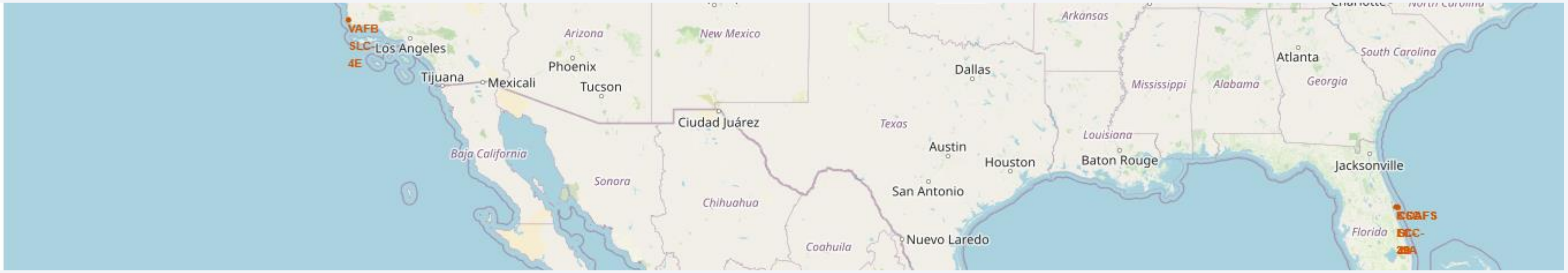
COUNT(Landing_Outcome)	Landing_Outcome
3	Controlled (ocean)
5	Failure (drone ship)
2	Failure (parachute)
10	No attempt
1	Precluded (drone ship)
5	Success (drone ship)
3	Success (ground pad)
2	Uncontrolled (ocean)

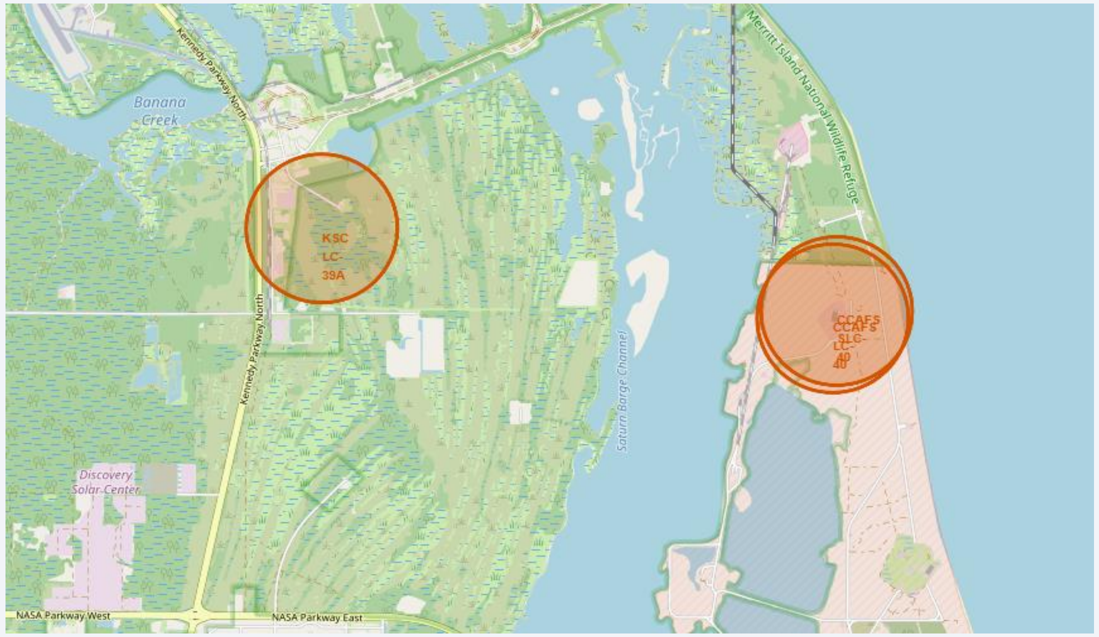
%sql SELECT COUNT(Landing_Outcome), Landing_Outcome FROM SPACEXTBL WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20' GROUP BY Landing_Outcome

Section 3

Launch Sites Proximities Analysis

Launch site locations

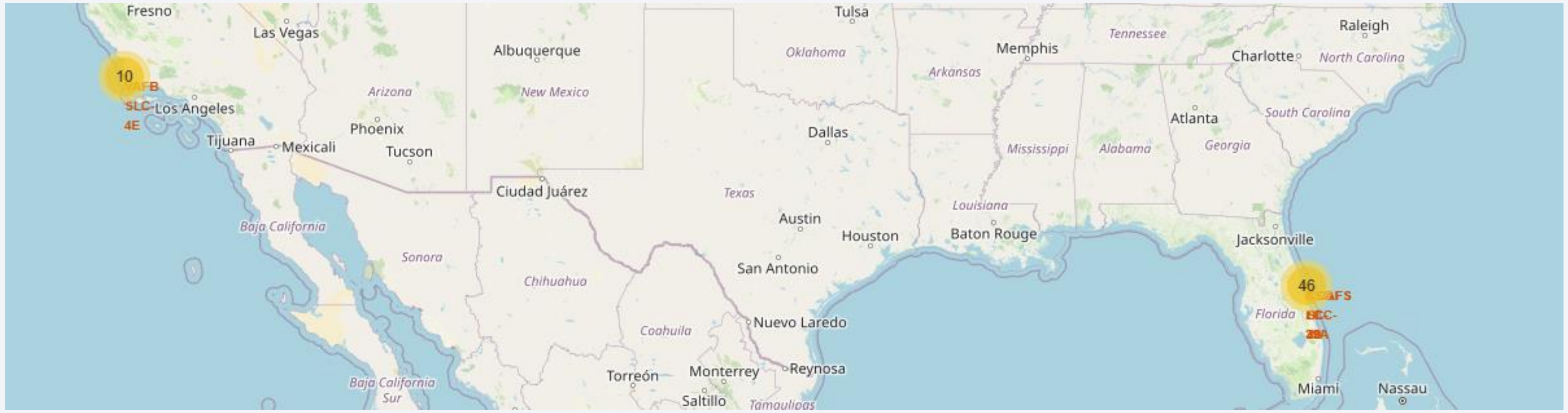


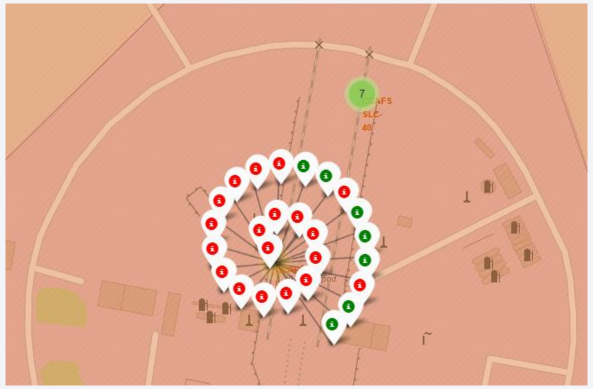


Markers were added to visually locate each launch site. When zooming in, each marker can be seen easily (left picture). Zoomed out, the markers are seen as the image above

We can see most of the launch shite are near the equator and close to the coast.

Color labelled launch outcomes



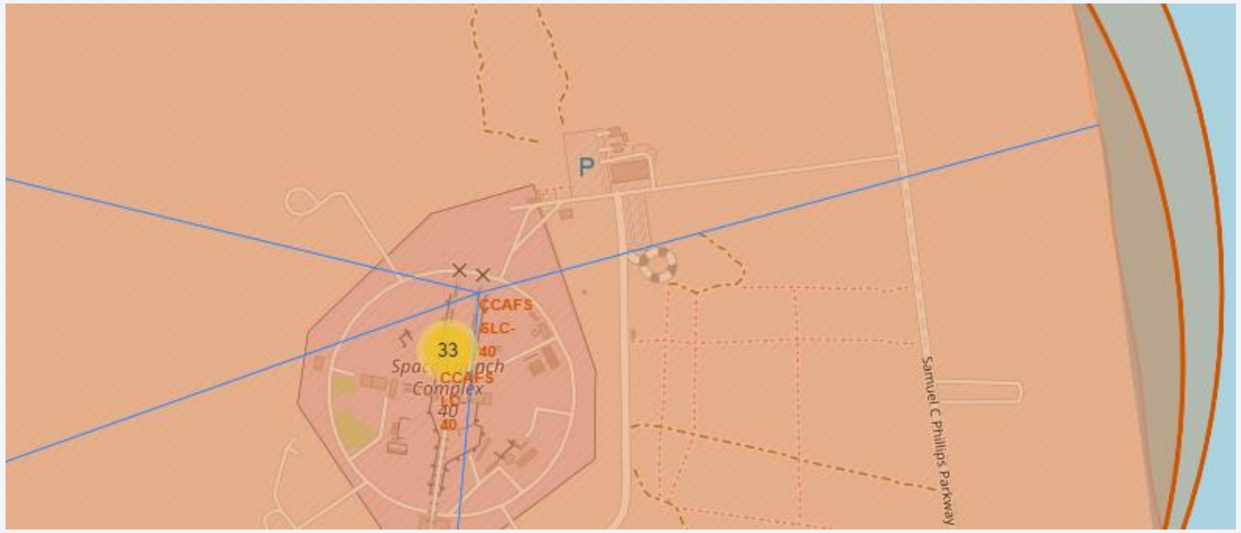


We use cluster markers to label each launch outcome for each site.

When zoomd out, you can see both cluster markers 10 and 46. When, zooming in, for example, the 10 cluster mark and cliking on it, the different outcomes can be seen, successful ones in green, unsuccessful ones in red.

From the map, we observe taht the launch site with the most successful outcomes is KSC

Launch site proximities

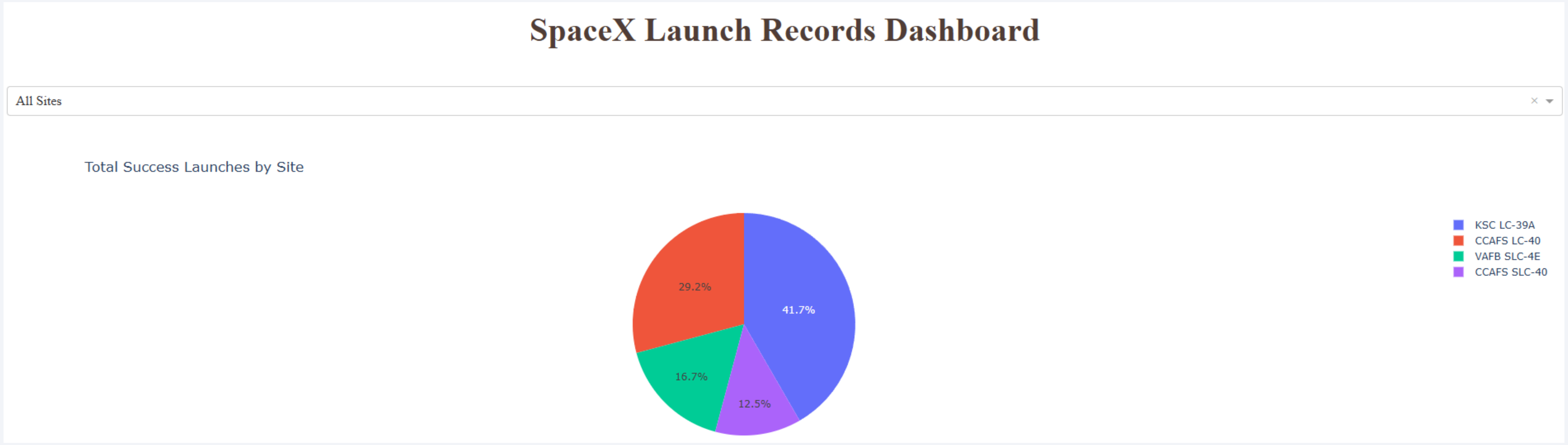


- We draw lines from, for example, the CCAFS launch site, to its closest coastal point, highway, major city, and railway.
- We found that most launch sites are usually located close enough to main highways and railways for logistical reason
- Near coastlines for safety.
- Nonetheless, they are generally located quite far from major cities as to ensure safety and minimize noise/inconveniences to the rest of the population.

Section 4

Build a Dashboard with Plotly Dash

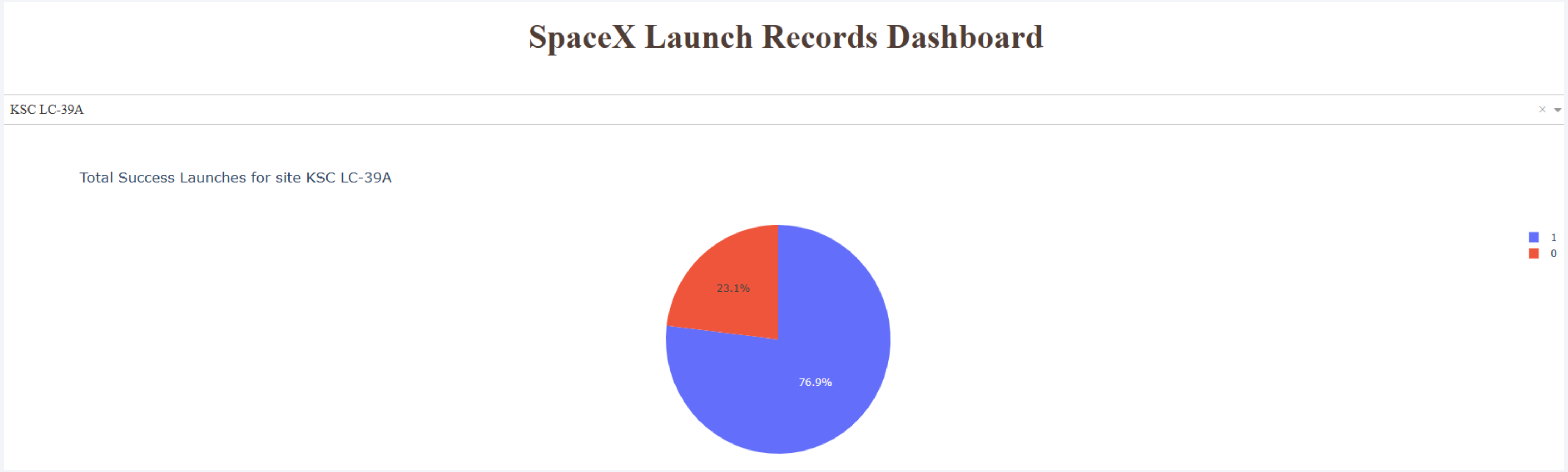
Launch success count per site


SpaceX Launch Records Dashboard

All Sites

On one hand, we found out that the highest number of success launches is for the KSC LC-39A site. On the other hand, the lowest one is CCAFS SLC-40

KSC LC-39A success ratio

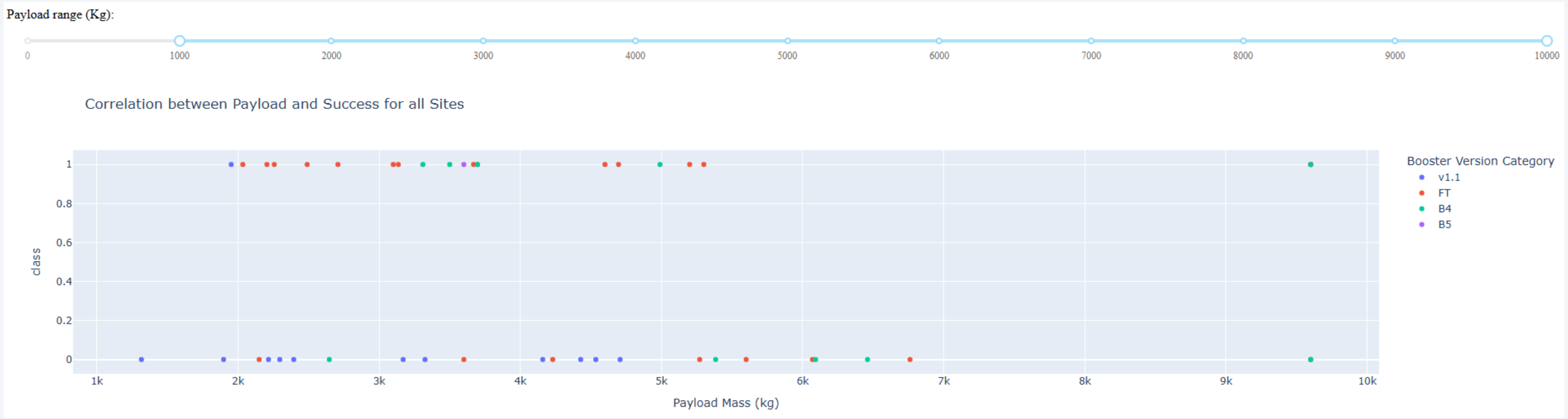

SpaceX Launch Records Dashboard

KSC LC-39A

Total Success Launches for site KSC LC-39A

For the launch site with the highest success count, the ratio is 76.9% of success launches against a 23.1% of unsuccessful ones

Payload vs. Launch Outcome



For a payload range between 2-5k, booster v1.1 has the least successful rate, whereas for that same payload range, booster FT has the highest successful rate.

B5 is one of the least common booster versions.

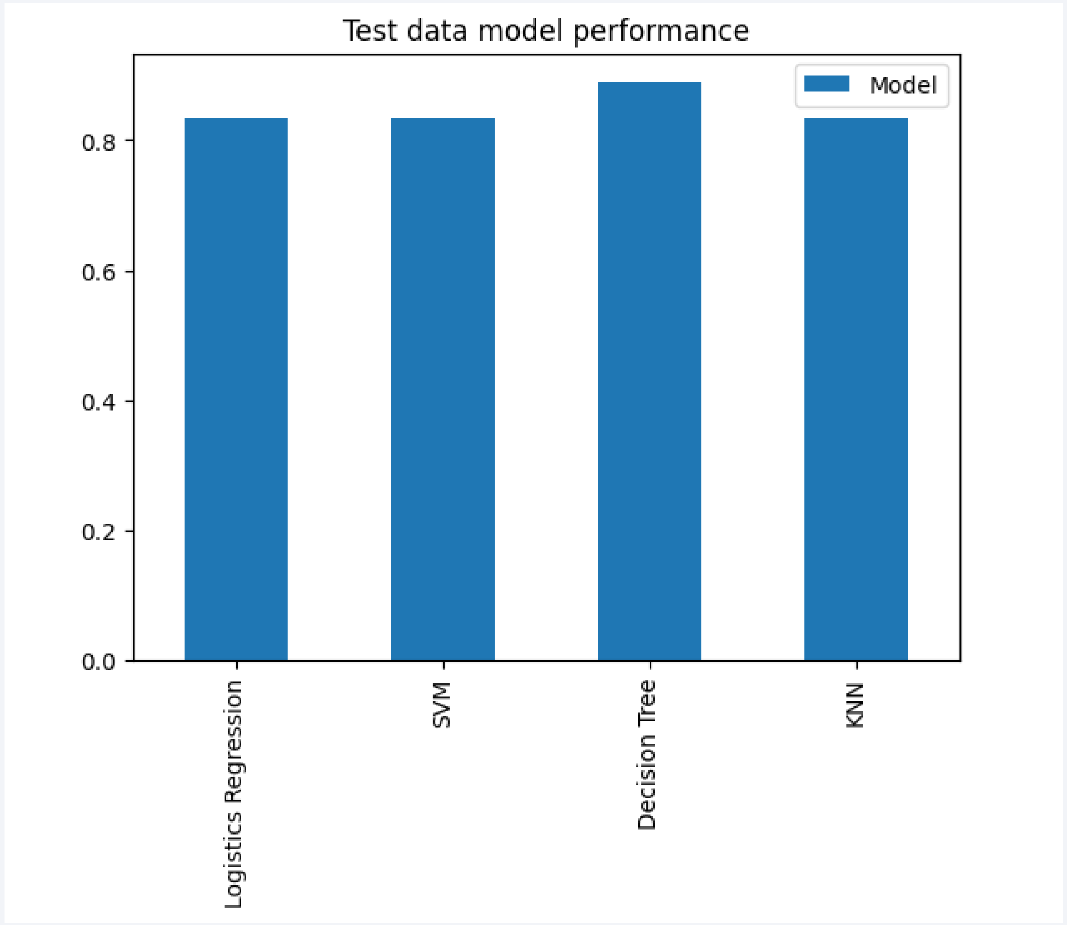
B4 is mostly successful between that same 2k-5k payload range, but for greater payload mass, it is mostly unsuccessful

Section 5

Predictive Analysis (Classification)

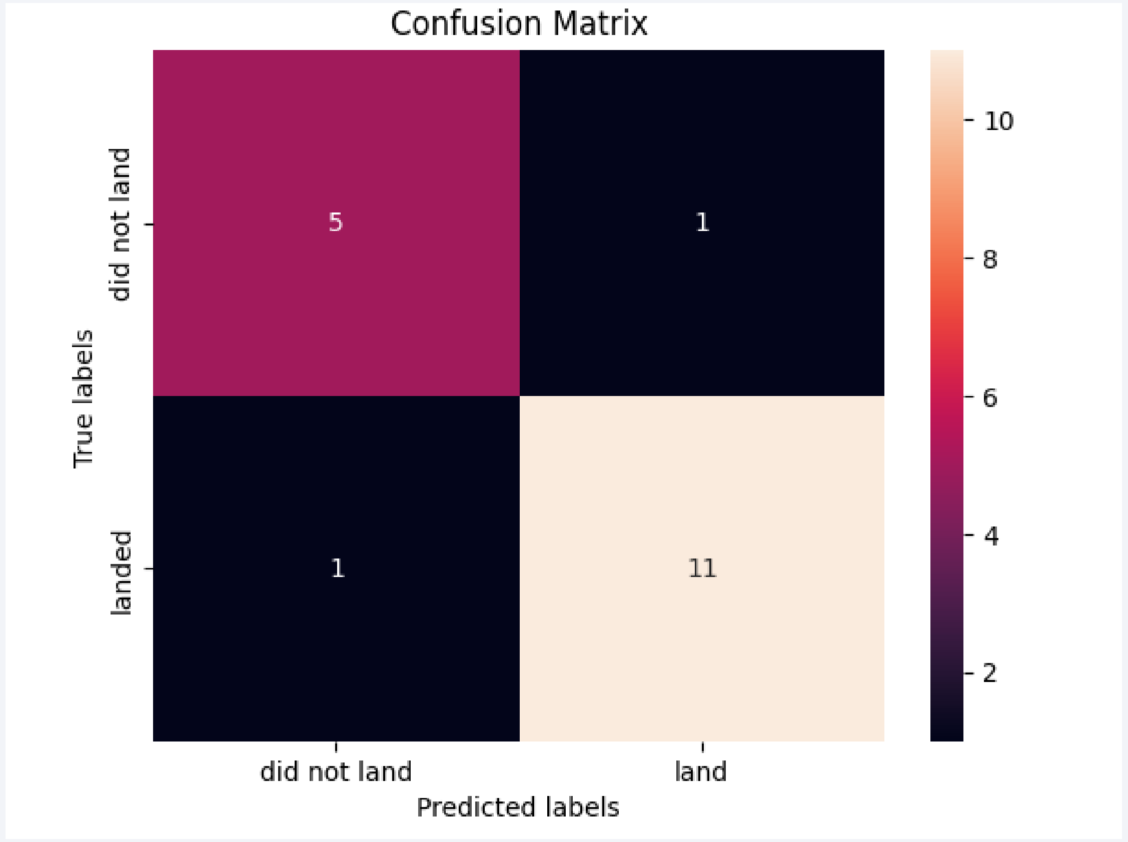
Classification Accuracy

- The model with the best performance is the Decision Tree model. However, if we run it a couple times, the score for this model gets lower and lower, whereas the rest of the methods (Logistic Regression, SVM and KNN) maintained a score of 0.83 out of 1.00 which quite good.
- All models have pretty much the same performance since our data is not quite large.



Confusion Matrix

In the first run, the best performance model was the **Decision Tree Classifier**, with only 1 false positive and 1 false negative, as seen in its confusion matrix

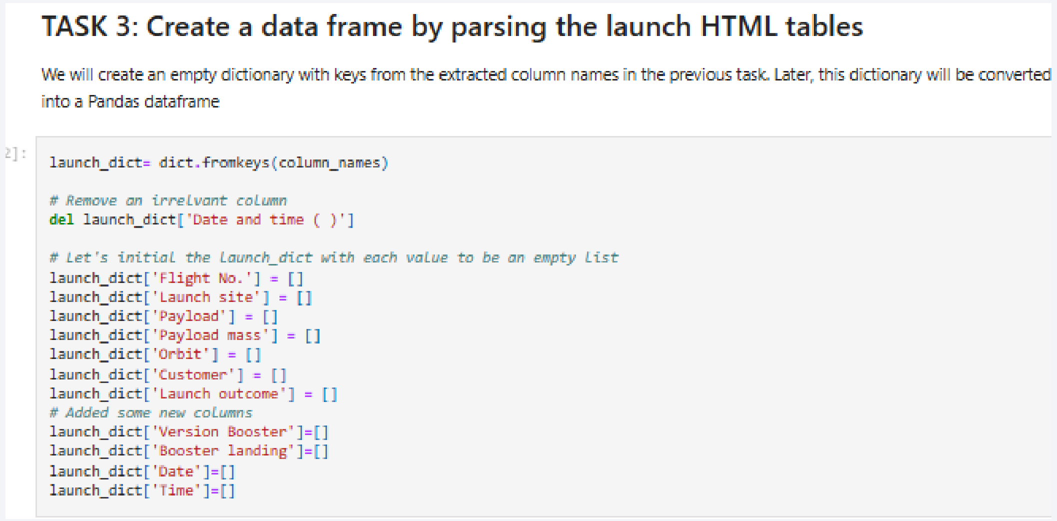


Conclusions

- Highest successful launch sites CCAFS and KSC are close to the equator and to the coast but away from major cities. Close to the equator allows energy saving in certain orbits. Also, being close enough to highways and railways allows easy transportation of employees needed to prepare the rocket launch
- Being close to the sea allows controlled landings, which translates into more likely reusing the first stage and saving money.
- The heavier the payload, the more energy is needed thus probably the most expensive the launch becomes.
- The more numerous flights the more likely they are to be successful, practice is key everywhere.
- Most launches do not attempt to land
- For a payload range between 2-5k, booster v1.1 has the least successful rate, whereas for that same payload range, booster FT has the highest successful rate. B4 is mostly successful between that same 2k-5k payload range, but for greater payload mass, it is mostly unsuccessful
- The model with the best performance is the Decision Tree model. However, if we run it a couple times, the score for this model gets lower and lower, whereas the rest of the methods (Logistic Regression, SVM and KNN) maintained a score of 0.83 out of 1.00 which quite good. All models have pretty much the same performance since our data is not quite large.

Appendix

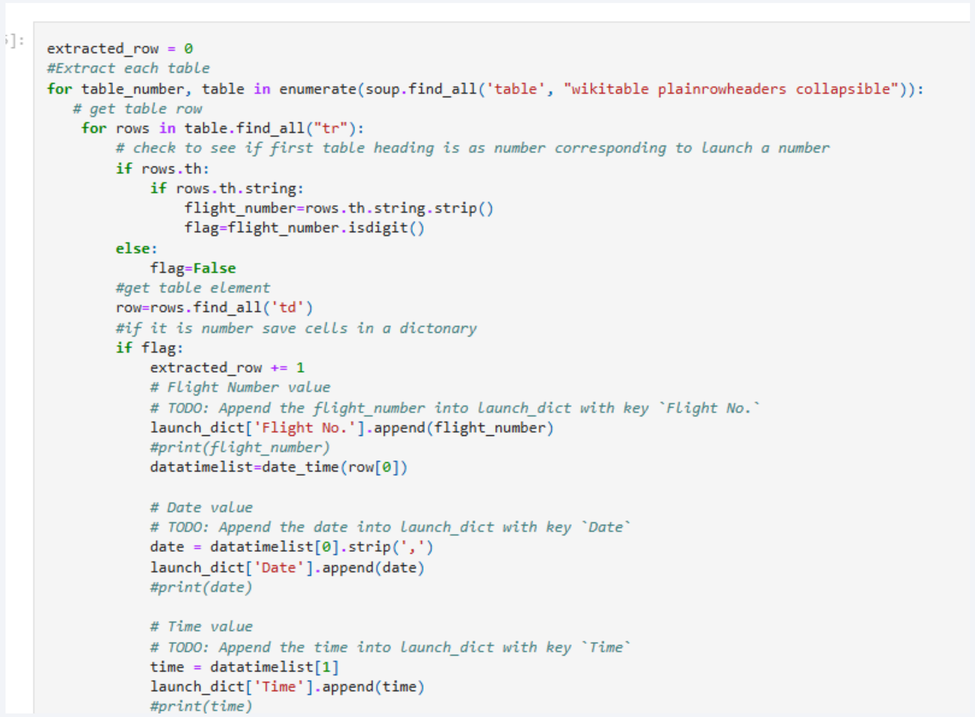
- Parse HTML tables and create a dataframe



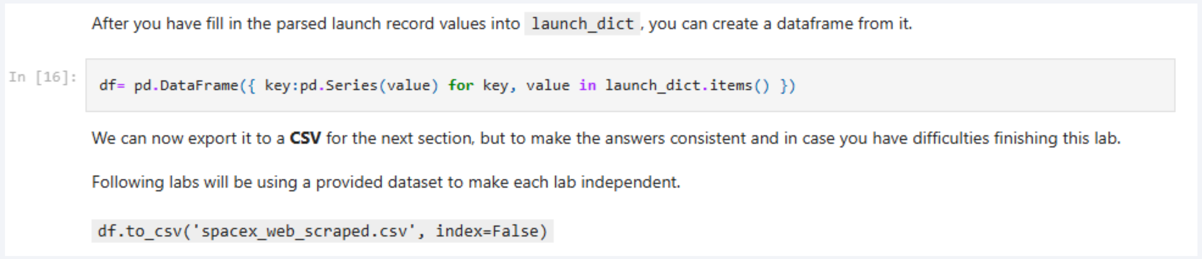
TASK 3: Create a data frame by parsing the launch HTML tables

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

```
2]:  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']=[]
```



```
]:  extracted_row = 0
    #Extract each table
    for table_number, table in enumerate(soup.find_all('table', "wikitable plainrowheaders collapsible")):
       # get table row
        for rows in table.find_all("tr"):
            # check to see if first table heading is as number corresponding to launch a number
            if rows.th:
                if rows.th.string:
                    flight_number=rows.th.string.strip()
                    flag=flight_number.isdigit()
            else:
                flag=False
            #get table element
            row=rows.find_all('td')
            #if it is number save cells in a dictonary
            if flag:
                extracted_row += 1
                # Flight Number value
                # TODO: Append the flight_number into launch_dict with key `Flight No.`
                launch_dict['Flight No.'].append(flight_number)
                #print(flight_number)
                datatimelist=date_time(row[0])

                # Date value
                # TODO: Append the date into launch_dict with key `Date`
                date = datatimelist[0].strip(',')
                launch_dict['Date'].append(date)
                #print(date)

                # Time value
                # TODO: Append the time into launch_dict with key `Time`
                time = datatimelist[1]
                launch_dict['Time'].append(time)
                #print(time)
```



After you have fill in the parsed launch record values into `launch_dict`, you can create a dataframe from it.

```
In [16]:  df= pd.DataFrame({ key:pd.Series(value) for key, value in launch_dict.items() })
```

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

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

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

Thank you!