IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Evangelia Kiosidou
10/23/2022

Outline

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

Executive Summary

- Summary of methodologies
  - Data Collection through API
  - Data Collection with Web Scraping
  - Data Wrangling
  - Exploratory Data Analysis with SQL
  - Exploratory Data Analysis with Data Visualization
  - interactive visual analytics using Folium and Plotly Dash
  - Machine Learning Prediction

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

Introduction

- Project background and context

Background: SpaceX advertises **Falcon 9** rocket launches on its website with a cost of 62 million dollars; other providers cost upwards of 165 million dollars each, much of the savings is because SpaceX can reuse the first stage. Therefore, if we can determine if the first stage will land, we can determine the cost of a launch.

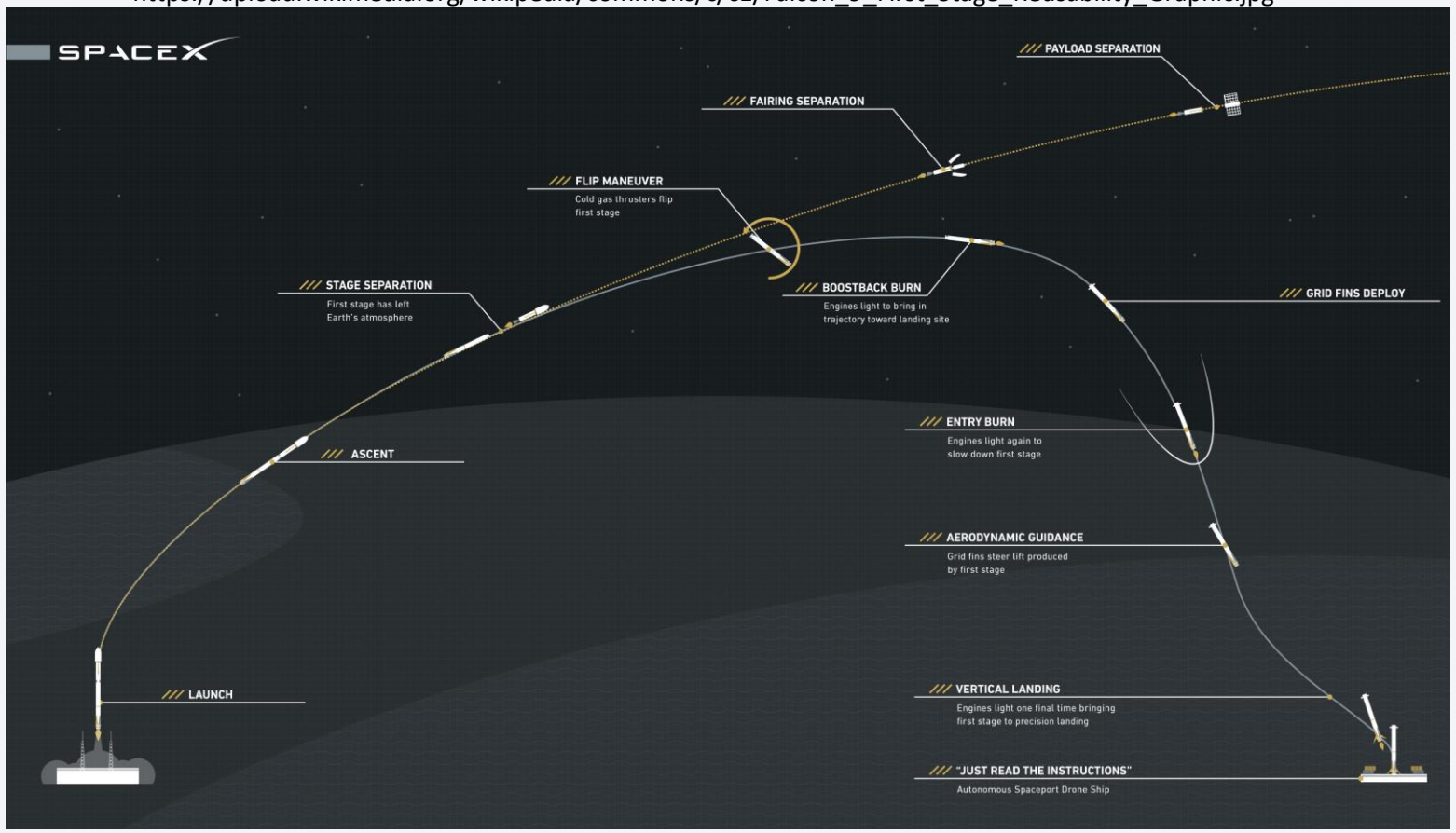
Context/goal: We want to determine the price of each launch. We will do this by gathering information about Space X and creating dashboards. We will also determine if SpaceX will reuse the first stage. We will train a machine learning model and use public information to predict if SpaceX will reuse the first stage, through prediction of successful landings.

- Problems you want to find answers

- Determination of the crucial parameters dictating whether a landing will be successful or not.
- Relationship between the most crucial parameters and their effect on the landing outcome.

Introduction

https://upload.wikimedia.org/wikipedia/commons/c/c1/Falcon_9_First_Stage_Reusability_Graphic.jpg



Graphic of the Falcon 9 reusability process

Section 1

Methodology

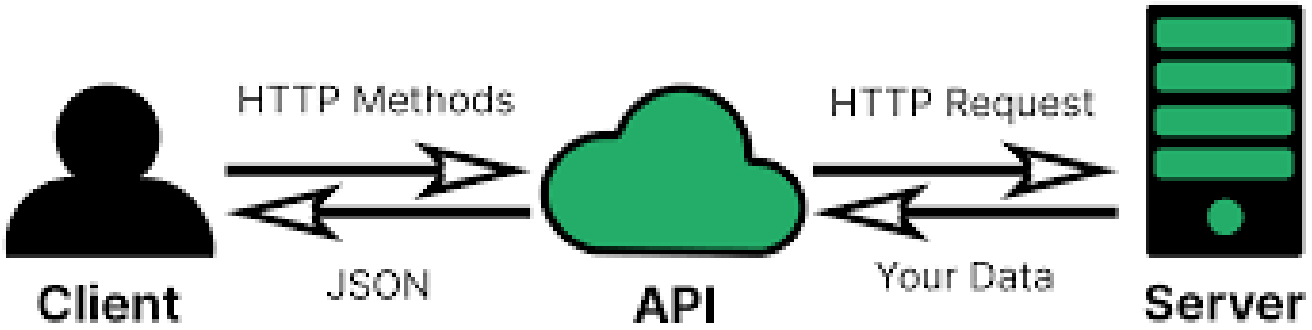
Methodology

Executive Summary

- Falcon 9 launch data collection methodology:
  - SpaceX REST API
  - Web scraping from related Wiki pages, using Python BeautifulSoup
- Perform data wrangling
    - One hot encoding was applied for “binarization” of the categorical variables and data cleaning of null values, in order to use them for model training.
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - LR, KNN, SVM, DT models were applied, and the best classifier was selected.

Data Collection – SpaceX REST API

https://help.klaviyo.com/hc/en-us/articles/360045726811-Getting-Started-with-Klaviyo-APIs



- Space X REST API
- API data: [launches, rockets, payload, landing specification, landing outcome]
- REST API URL: https://api.spacexdata.com/v4/launches/past
- HTTP Request to server: get(url)
- Response: response=requests.get(url)
- JSON file as a response: response.json()
- Transformation to a dataframe: data=pd.jason_normalize(response.json())

Data Collection – SpaceX API

1. Request data and get response from API:

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

In [42]:
```
spacex_url="https://api.spacexdata.com/v4/launches/past"
```

In [43]:
```
response = requests.get(spacex_url)
```

2. Decode response as a .json() file and turn it into a Pandas dataframe using .json_normalize()

In [45]:
```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_call_spacex_api.json'
```

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

In [46]:
```
response.status_code
```

Out[46]: 200

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

In [47]:
```
# Use json_normalize meethod to convert the json result into a dataframe
data_response=response.json()
data=pd.json_normalize(data_response)
```

[Continues to next slide]

Data Collection – SpaceX API

3. Drop Falcon 1 booster data and keep only Falcon 9:

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

In [27]:
```
df.drop(df[(df['BoosterVersion']=='Falcon 1')].index, inplace=True)
data_falcon9=df
data_falcon9
```

Out[27]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Lo
4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0003	-8(
5	8	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0005	-8(
6	10	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0007	-8(
7	11	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None	1.0	0	B1003	-12(
8	12	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B1004	-8(
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
89	102	2020-09-03	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	2	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	12	B1060	-8(
90	103	2020-10-06	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	3	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	13	B1058	-8(
91	104	2020-10-18	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	6	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	12	B1051	-8(
92	105	2020-10-24	Falcon 9	15600.0	VLEO	CCSFS SLC 40	True ASDS	3	True	True	True	5e9e3033383ecbb9e534e7cc	5.0	12	B1060	-8(
93	106	2020-11-05	Falcon 9	3681.0	MEO	CCSFS SLC 40	True ASDS	1	True	False	True	5e9e3032383ecb6bb234e7ca	5.0	8	B1062	-8(

90 rows × 17 columns

[Continues to next slide]

Data Collection – SpaceX API

4. Data Wrangling: Cleaning data/dealing with missing values using the mean:

Task 3: Dealing with Missing Values

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

In [32]:
```
# Calculate the mean value of PayloadMass column
mean_payload=data_falcon9['PayloadMass'].mean()
mean_payload
```

Out[32]:
```
6123.547647058824
```

In [60]:
```
# Replace the np.nan values with its mean value
data_falcon9.PayloadMass=data_falcon9.PayloadMass.replace(np.nan, mean_payload ,regex=True)
```

You should see the number of missing values of the `PayLoadMass` change to zero.

In [61]:
```
data_falcon9.isnull().sum()
```

Out[61]:
```
FlightNumber      0
Date              0
BoosterVersion    0
PayloadMass       0
Orbit             0
LaunchSite        0
Outcome           0
Flights           0
GridFins          0
Reused            0
Legs              0
LandingPad       26
Block             0
ReusedCount       0
Serial            0
Longitude         0
Latitude          0
dtype: int64
```

Now we should have no missing values in our dataset except for in `LandingPad`.

5. Export Clean data to CSV:

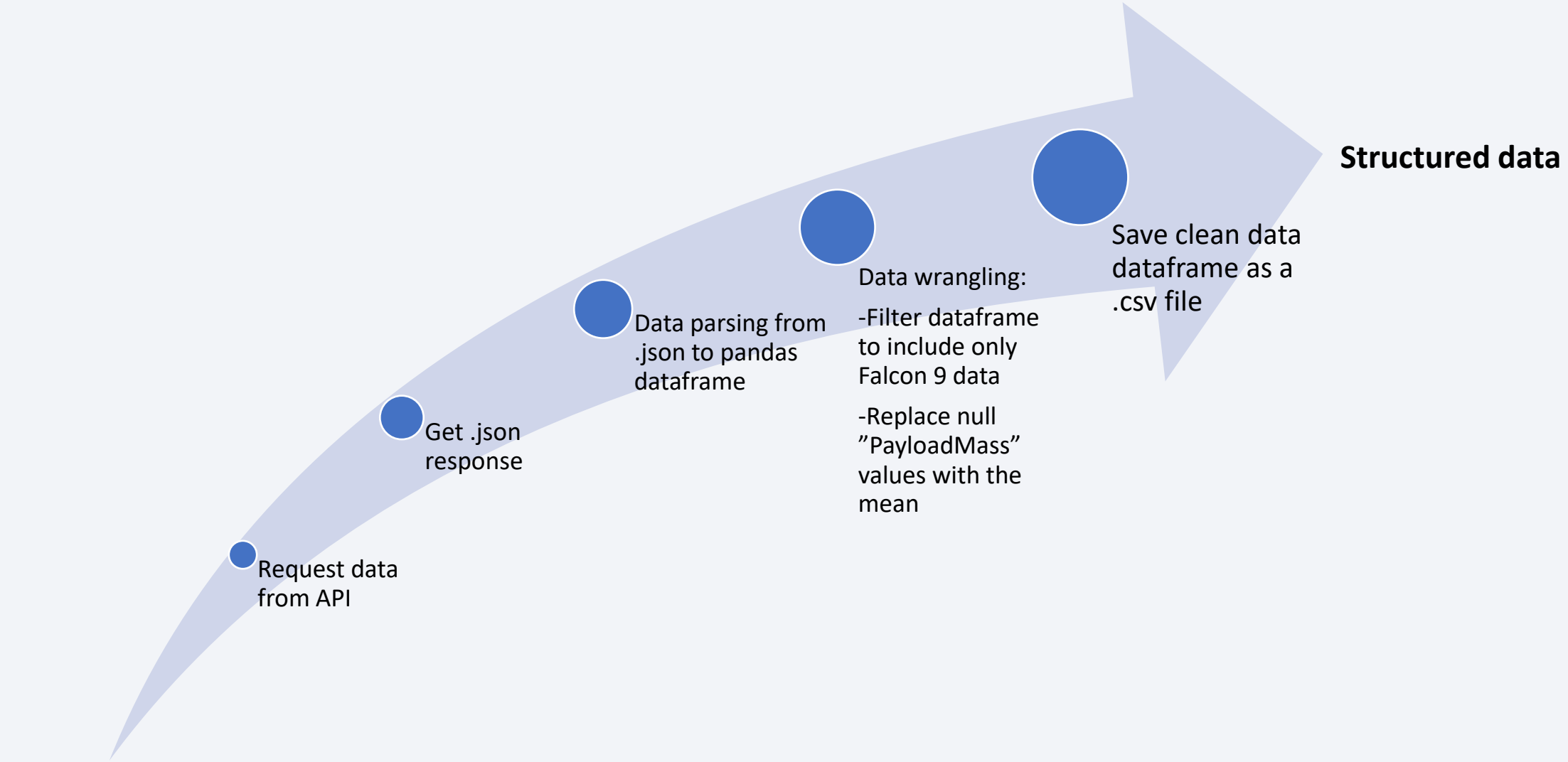
We can now export it to a **CSV** for the next section,but to make the answers consistent, in the next lab we will provide data in a pre-selected date range.

`data_falcon9.to_csv('dataset_part\_1.csv', index=False)`

In [62]:
```
data_falcon9.to_csv('dataset_part_1.csv', index=False)
```

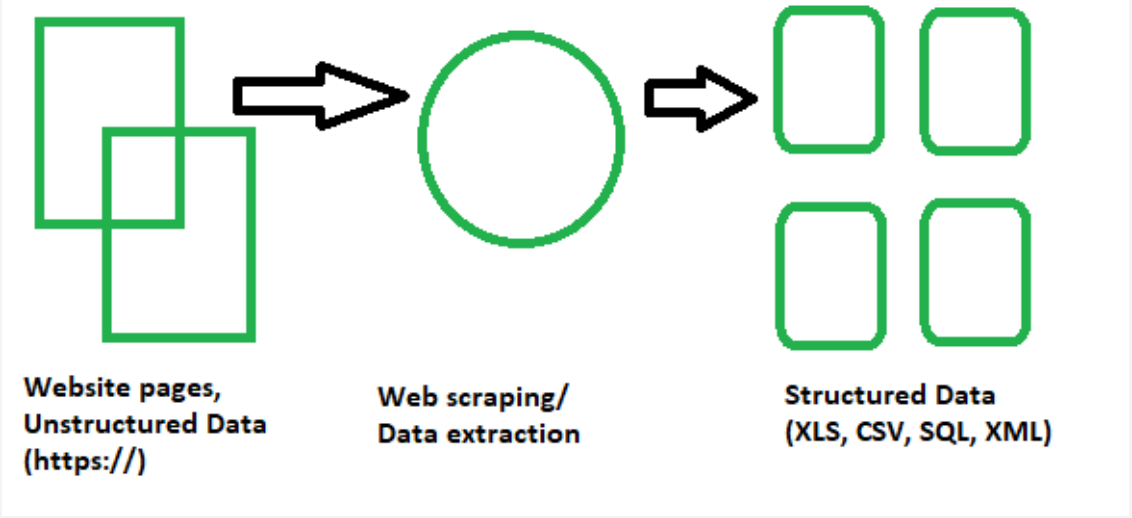
6. GitHub: https://github.com/lilaki34/testrepo/blob/main/jupyter-labs-spacex-data-collection-api-complete.ipynb

Data Collection – SpaceX API – Flowchart Summary



Data Collection – Web Scraping

https://www.geeksforgeeks.org/web-scraping-from-wikipedia-using-python-a-complete-guide/



- BeautifulSoup for webscraping through Wiki pages
- Webscraping of HTML tables
- Data parsing from HTML table to a dataframe

Data Collection – Web Scraping

1. Request Falcon9 Launch Wiki page

Python: *get(url)*

2. Parse the HTML data

Python: *soup=BeautifulSoup()*

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 [48]:
```
# use requests.get() method with the provided static_url
r = requests.get(static_url)
# assign the response to a object
print(r)
```

```
<Response [200]>
```

In [49]:
```
#r.content
```

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

In [50]:
```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(r.content, 'html.parser')
```

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

In [51]:
```
# Use soup.title attribute
#print(soup.prettify())
```

In [52]:
```
title = soup.find('title')
print(title)
```

```
<title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```


[Continues to next slide]

Data Collection – Web Scraping

3. Find all tables in the BeautifulSoup object

Python: *soup.find_all('table')*

4. Extract column names from selected table

Python: *extract_column_from_header()*

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

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

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

In [53]:
```
# Use the find_all function in the BeautifulSoup object, with element type `table`

table=soup.find_all('table')

# Assign the result to a list called `html_tables`
html_tables=table
```

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

In [54]:
```
# Let's print the third table and check its content
first_launch_table = html_tables[2]
print(first_launch_table)
```

```
<table class="wikitable plainrowheaders collapsible" style="width: 100%;">
<tbody><tr>
<th scope="col">Flight No.
</th>
<th scope="col">Date and<br/>time (<a href="/wiki/Coordinated_Universal_Time" title="Coordinated Universal Time">UTC</a>)
</th>
<th scope="col"><a href="/wiki/List_of_Falcon_9_first-stage_boosters" title="List of Falcon 9 first-stage boosters">Version,<br/>Booster</a> <sup class="reference" id="cite_ref-booster_11-0"><a href="#cite_note-booster-11">[b]</a></sup>
</th>
<th scope="col">Launch site
</th>
<th scope="col">Payload<sup class="reference" id="cite_ref-Dragon_12-0"><a href="#cite_note-Dragon-12">[c]</a></sup>
</th>
<th scope="col">Payload mass
</th>
<th scope="col">Orbit
</th>
<th scope="col">Customer
</th>
<th scope="col">Launch<br/>outcome
</th>
<th scope="col"><a href="/wiki/Falcon_9_first-stage_landing_tests" title="Falcon 9 first-stage landing tests">Booster<br/>landing</a>
</th></tr>
<tr>
<th rowspan="2" scope="row" style="text-align:center;">1
</th>
```

Next, we just need to iterate through the `<th>` elements and apply the provided `extract_column_from_header()` to extract column name one by one

In [55]:
```
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 row in first_launch_table.find_all('th'):
    name = extract_column_from_header(row)
    if (name != None and len(name) > 0):
        column_names.append(name)
```

Check the extracted column names

In [56]:
```
print(column_names)
```

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

Data Collection – Web Scraping

5. Create empty dictionary using as keys the extracted column names

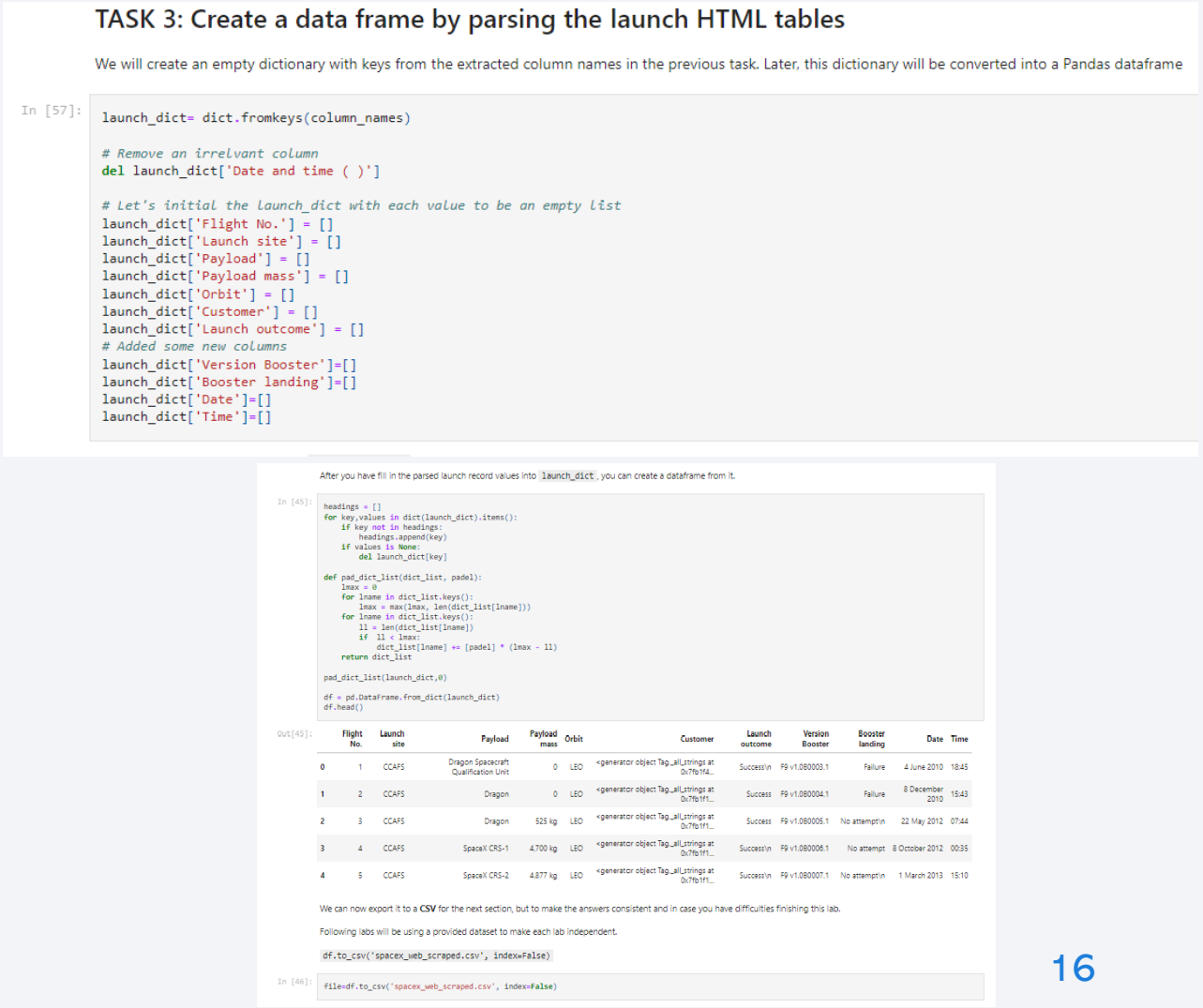
Python:
launch_dict=dict.fromkeys(column_names)

6. Create a dataframe from the formed dictionary

Python: *df=pd.DataFrame(launch_dict)*

7. Export dataframe into .csv

Python: *df.to_csv()*



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 [57]:

```
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']=[]
```

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

In [45]:

```
headings = []
for key,values in dict(launch_dict).items():
    if key not in headings:
        headings.append(key)
    if values is None:
        del launch_dict[key]

def pad_dict_list(dict_list, padel):
    lmax = 0
    for lname in dict_list.keys():
        lmax = max(lmax, len(dict_list[lname]))
    for lname in dict_list.keys():
        ll = len(dict_list[lname])
        if ll < lmax:
            dict_list[lname] += [padel] * (lmax - ll)
    return dict_list

pad_dict_list(launch_dict,0)

df = pd.DataFrame.from_dict(launch_dict)
df.head()
```

Out[45]:

	Flight No.	Launch site	Payload	Payload mass	Orbit	Customer	Launch outcome	Version Booster	Booster landing	Date	Time
0	1	CCAFS	Dragon Spacecraft Qualification Unit	0	LEO	<generator object Tag._all_strings at 0x7fb1f4...	Success\n	F9 v1.0B0003.1	Failure	4 June 2010	18:45
1	2	CCAFS	Dragon	0	LEO	<generator object Tag._all_strings at 0x7fb1f1...	Success	F9 v1.0B0004.1	Failure	8 December 2010	15:43
2	3	CCAFS	Dragon	525 kg	LEO	<generator object Tag._all_strings at 0x7fb1f1...	Success	F9 v1.0B0005.1	No attempt\n	22 May 2012	07:44
3	4	CCAFS	SpaceX CRS-1	4,700 kg	LEO	<generator object Tag._all_strings at 0x7fb1f1...	Success\n	F9 v1.0B0006.1	No attempt	8 October 2012	00:35
4	5	CCAFS	SpaceX CRS-2	4,877 kg	LEO	<generator object Tag._all_strings at 0x7fb1f1...	Success\n	F9 v1.0B0007.1	No attempt\n	1 March 2013	15:10

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.

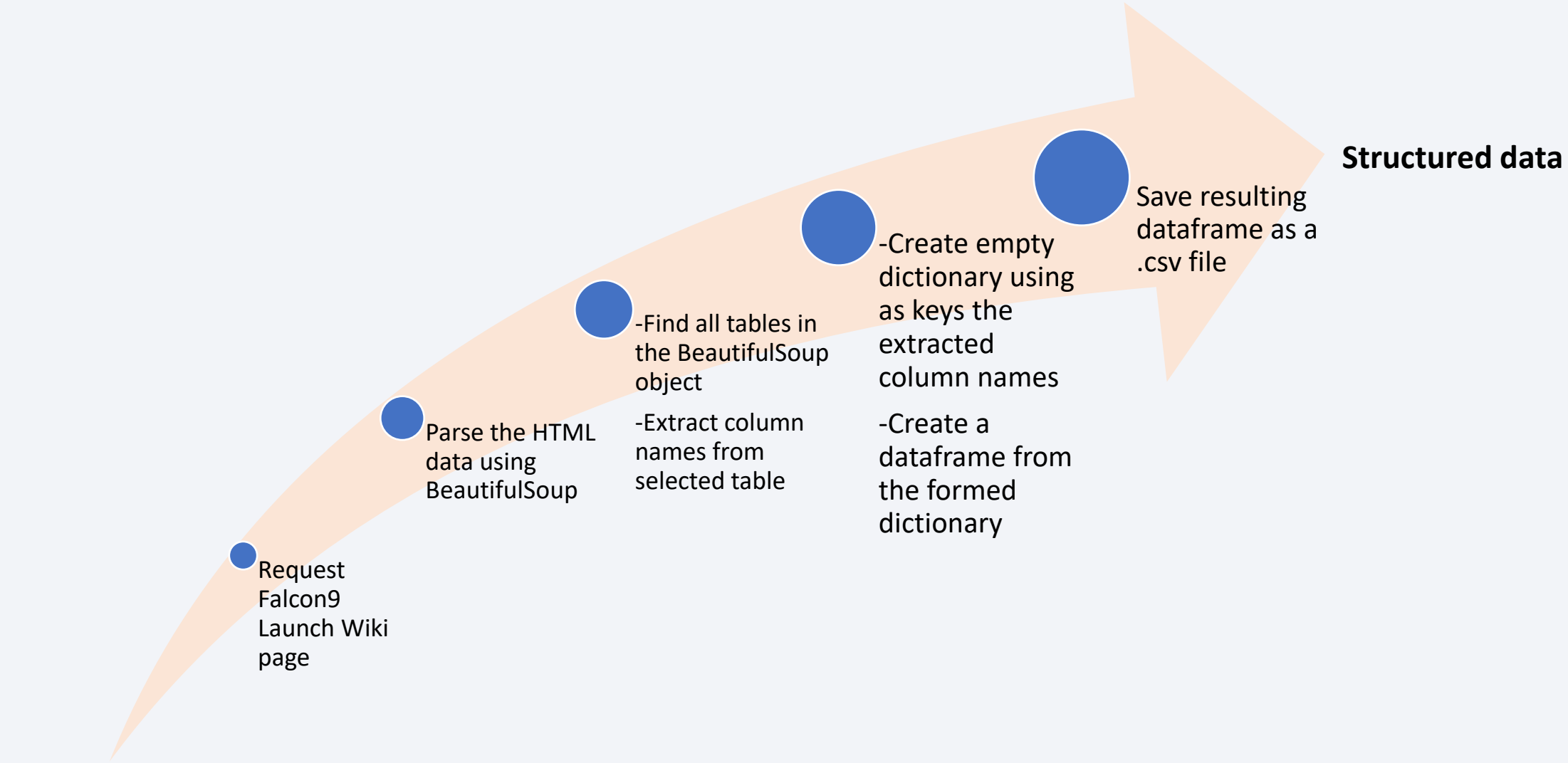
```
df.to_csv('spacex_web_scraped.csv', index=False)
```

In [46]:

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


8. GitHub: https://github.com/lilaki34/testrepo/blob/main/jupyter-labs-webscraping.ipynb

Data Collection – Web Scraping– Flowchart Summary



Data Wrangling

Purpose: Exploratory Data Analysis (EDA) to find some patterns in the data and determine what would be the labels for training Supervised Models.

Data: The dataframe including the Falcon 9 data.

Columns: [Flight Number, Date, Booster Version=Falcon 9, Payload Mass, Orbit, Launch Site, Outcome, Flights, GridFins, Reused, Legs, Landing Pad, Block, Reused Count, Serial, Longitude, Latitude]

[Continues to next slide]

Data Wrangling – Flow Chart

Identify and calculate the percentage of the missing values in each attribute

Identify and calculate the percentage of the missing values in each attribute

```
In [3]: df.isnull().sum()/df.count()*100

Out[3]: FlightNumber      0.000
        Date              0.000
        BoosterVersion    0.000
        PayloadMass       0.000
        Orbit             0.000
        LaunchSite        0.000
        Outcome           0.000
        Flights           0.000
        GridFins          0.000
        Reused            0.000
        Legs              0.000
        LandingPad       40.625
        Block             0.000
        ReusedCount       0.000
        Serial            0.000
        Longitude         0.000
        Latitude          0.000
        dtype: float64
```

Identify which columns are numerical and categorical

Identify which columns are numerical and categorical:

```
In [4]: df.dtypes

Out[4]: FlightNumber      int64
        Date             object
        BoosterVersion   object
        PayloadMass     float64
        Orbit            object
        LaunchSite       object
        Outcome          object
        Flights           int64
        GridFins           bool
        Reused             bool
        Legs               bool
        LandingPad       object
        Block           float64
        ReusedCount       int64
        Serial           object
        Longitude       float64
        Latitude        float64
        dtype: object
```

Number of launches on each launching site

```
In [7]: # Apply value_counts() on column LaunchSite
        df['LaunchSite'].value_counts()

Out[7]: CCAFS SLC 40    55
        KSC LC 39A      22
        VAFB SLC 4E     13
        Name: LaunchSite, dtype: int64
```

"Launch Site" is a categorical value

Number and occurrence of each orbit

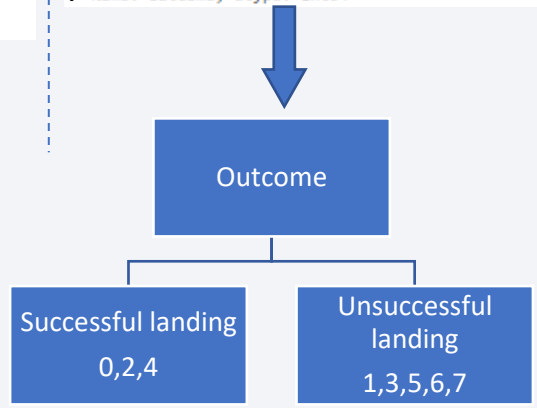
```
In [8]: # Apply value_counts on Orbit column
        df['Orbit'].value_counts()

Out[8]: 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
```

Number and occurence of mission outcome per orbit type

```
# Landing_outcomes = values on Outcome column
landing_outcomes= df['Outcome'].value_counts()
landing_outcomes

0  True ASDS      41
1  None None      19
2  True RTLS      14
3  False ASDS      6
4  True Ocean      5
5  False Ocean     2
6  None ASDS       2
7  False RTLS      1
   Name: Outcome, dtype: int64
```



[Continues to next slide]

Data Wrangling – Flow Chart

Landing outcome label from “Outcome” column

```
In [14]:
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise

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

This variable will represent the classification first stage landed Successfully

```
In [15]:
df['Class']=landing_class
df[['Class']].head(8)
```

Out[15]:

	Class
0	0
1	0
2	0
3	0
4	0
5	0
6	1
7	1

Average success rate:

```
[28]: df["Class"].mean()

[28]: 0.6666666666666666
```

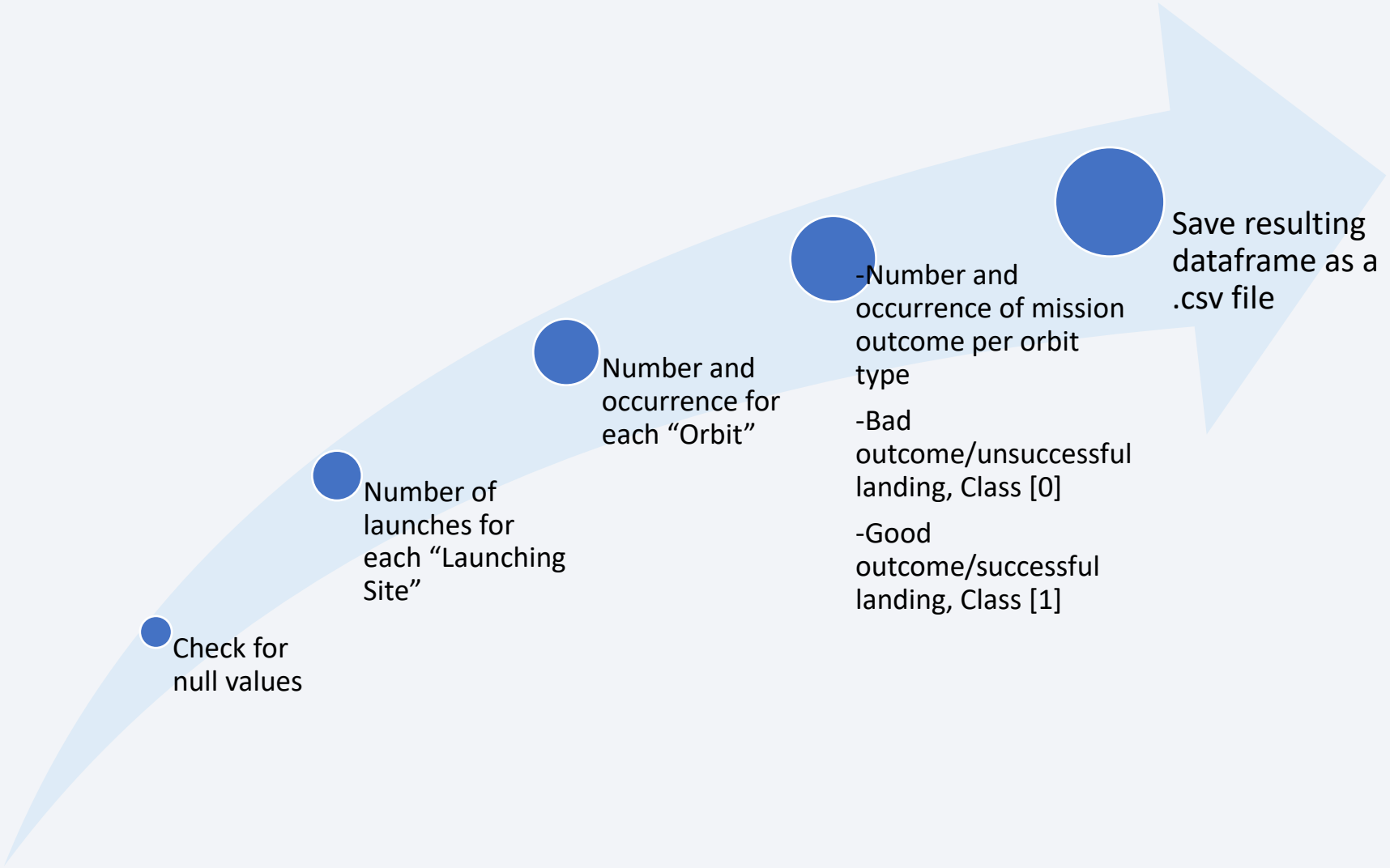
Export dataframe to csv

```
[29]: df.to_csv("dataset_part_2.csv", index=False)
```

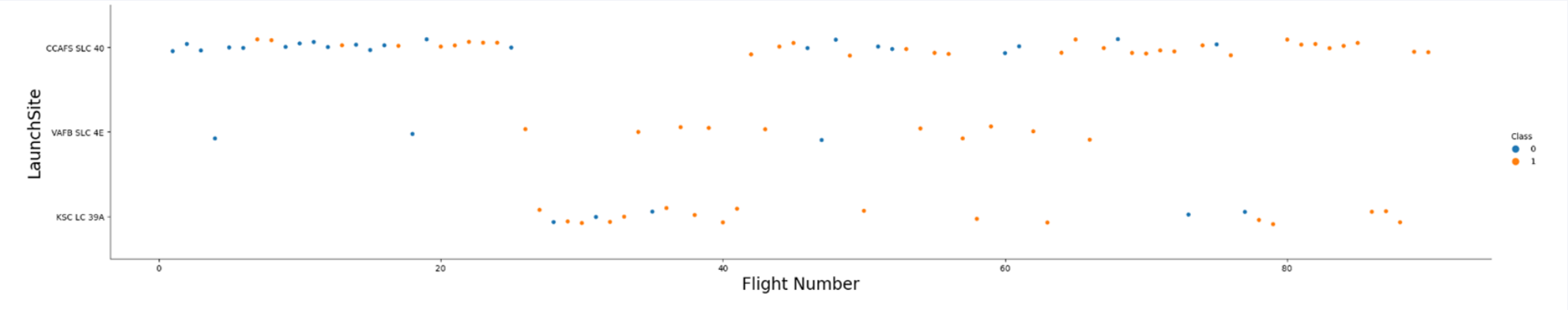
We can now export it to a CSV for the next sectio

GitHub: https://github.com/lilaki34/testrepo/blob/main/labs-jupyter-spacex-Data%20wrangling_2.ipynb

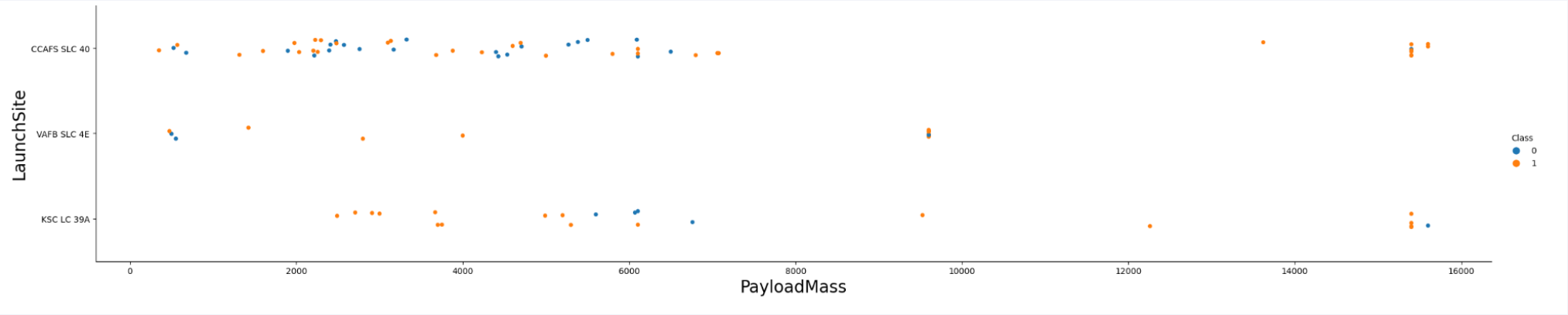
Data Wrangling– Flowchart Summary



EDA with Data Visualization



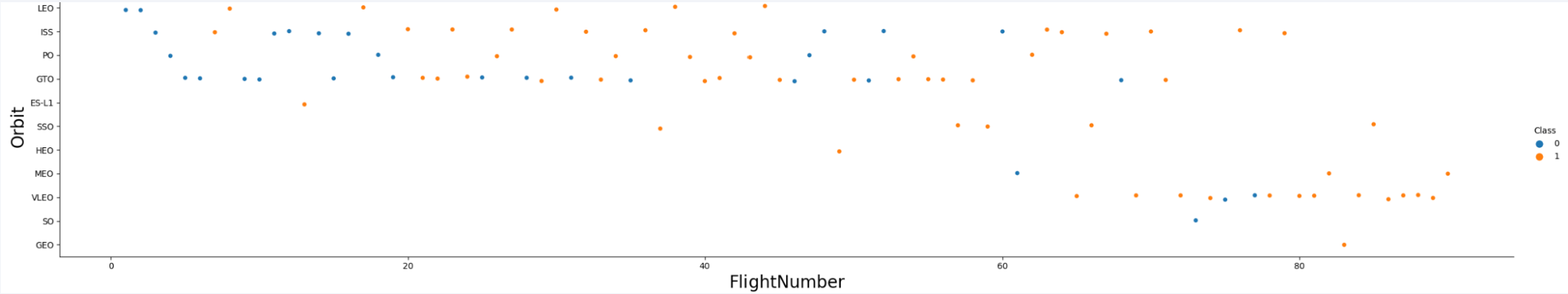
Flight Number vs Launch Site



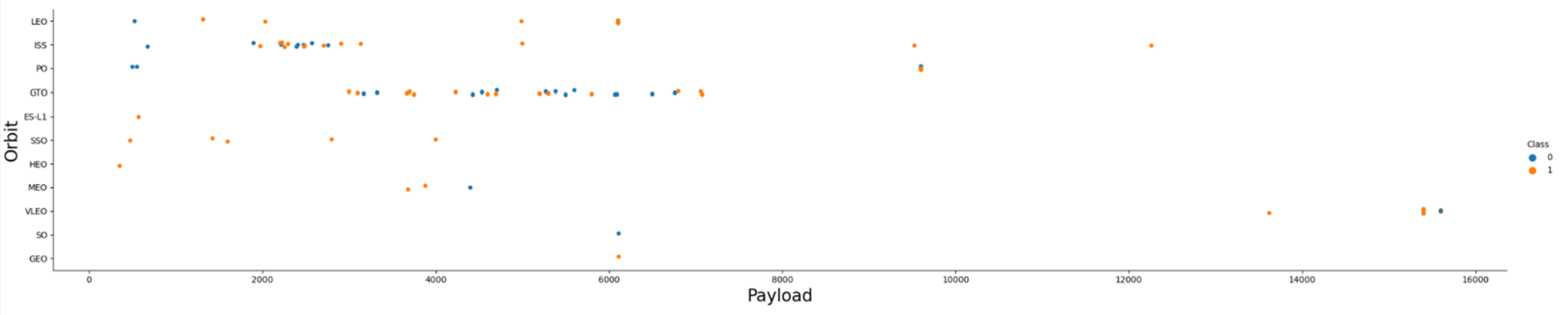
Payload Mass vs Launch Site

https://github.com/lilaki34/testrepo/blob/main/jupyter-labs-eda-dataviz_final.ipynb

EDA with Data Visualization



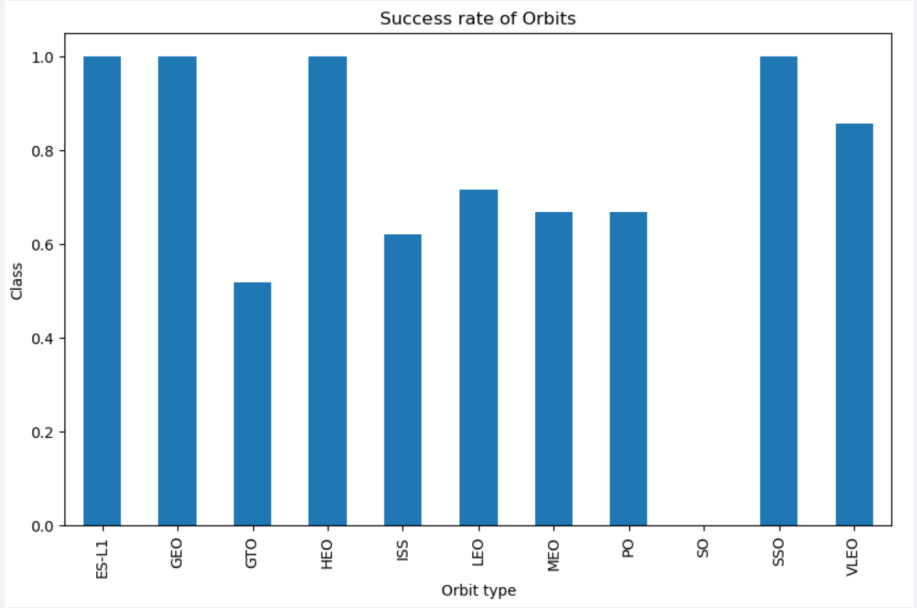
Flight Number vs Orbit Type



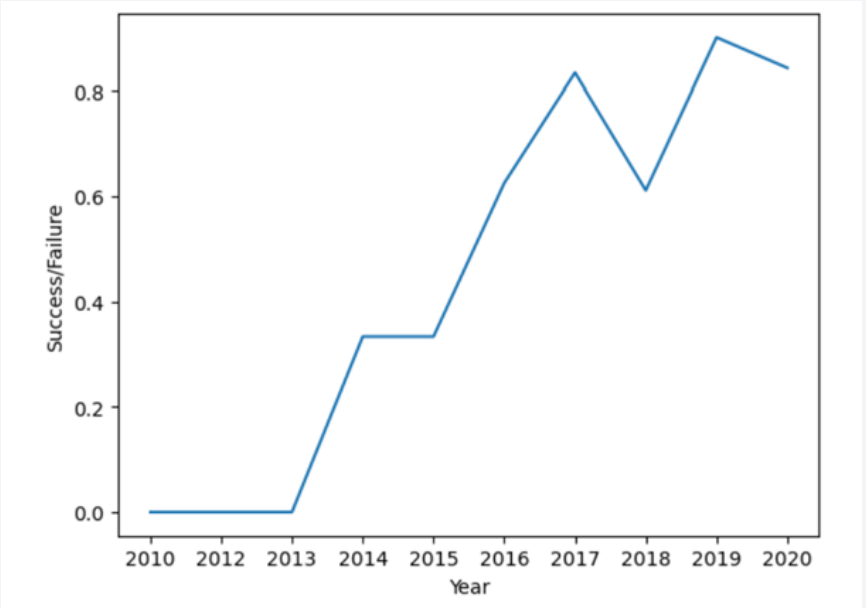
Payload Mass vs Orbit Type

https://github.com/lilaki34/testrepo/blob/main/jupyter-labs-eda-dataviz_final.ipynb

EDA with Data Visualization



Success rate of each orbit type



Success rate per launching number per year

Flight number vs Launch Site: CCAFS SLC 40 site has the highest number of flights, which contribute to a positive outcome.
Payload Mass vs Launch Site: CCAFS SLC 40 and KSC LC 39A seem to operate generally well for payload > 8000 kg.
Flight Number vs Orbit Type: **LEO**-Success related to the number of flights, **VLEO-** Mostly successful outcome for flight number >80
Payload Mass vs Orbit Type: With heavy payloads the successful landing or positive landing rate are more for SO,LEO and ISS.
Success rate of each orbit type: ES-L1, GEO, HEO and SSO have the highest success rate.
Success rate per launching number per year: Overall, there is an increasing trend both in launches and successful recoveries over the years.

https://github.com/lilaki34/testrepo/blob/main/jupyter-labs-eda-dataviz_final.ipynb

EDA with SQL

- We first uploaded the SpaceX dataset.
- SELECT DISTINCT (Launch_Site): display the names of the unique launch sites in the space mission.
- SELECT * from SPACEXTBL where (LAUNCH_SITE) LIKE 'CCA%' LIMIT 5: Display 5 records where launch sites begin with the string 'CCA'
- SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE CUSTOMER='NASA (CRS)': display the total payload mass carried by boosters launched by NASA (CRS)
- SELECT AVG(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE BOOSTER_VERSION='F9 v1.1': display average payload mass carried by booster version F9 v1.1
- SELECT min(DATE) FROM SPACEXTBL WHERE LANDING_OUTCOME='Success (ground pad)': list the date when the first successful landing outcome in ground pad was achieved.
- SELECT BOOSTER_VERSION FROM SPACEXTBL WHERE PAYLOAD_MASS__KG_ between 4000 and 6000 AND LANDING_OUTCOME='Success (drone ship)' : List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000.
- SELECT COUNT(*) FROM SPACEXTBL WHERE MISSION_OUTCOME LIKE '%Success%' OR MISSION_OUTCOME LIKE '%Failure%' : List the total number of successful and failure mission outcomes.

https://github.com/lilaki34/testrepo/blob/main/jupyter-labs-eda-sql-coursera_sqllite%20(1).ipynb

EDA with SQL

- SELECT BOOSTER_VERSION FROM SPACEXTBL WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTBL): List the names of the booster_versions which have carried the maximum payload mass. Use a subquery
- SELECT substr(Date, 6, 2),MISSION_OUTCOME,BOOSTER_VERSION,LAUNCH_SITE FROM SPACEXTBL where LANDING_OUTCOME = 'Failure (drone ship)' AND substr(Date,1,4)='2015': 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.
- SELECT LANDING_OUTCOME,COUNT(LANDING_OUTCOME) FROM SPACEXTBL WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20' AND LANDING_OUTCOME LIKE '%Success%'
  GROUP BY LANDING_OUTCOME
  ORDER BY COUNT(LANDING_OUTCOME) DESC
  Rank the count of successful landing_outcomes between the date 04-06-2010 and 20-03-2017 in descending order.

https://github.com/lilaki34/testrepo/blob/main/jupyter-labs-eda-sql-coursera_sqllite%20(1).ipynb

Build an Interactive Map with Folium

- We marked all launch sites on the Earth map and we saw that there are 3 sites in Florida and 1 in California.
- We marked the success/failed launches for each site on the map. We used markers of different colors, green for success and red for failed outcome.
- We calculated the distances between a launch site and its proximities. Such as the coast, the city and the railway.
- The launch sites are close to the Equator line and the coast. When the launch site is near the equator, it can take advantage of Earth's rotational speed, saving fuel. Also, the position of the launch sites serves in favor of launching towards particular orbits that need to be in specific angles compared to the equator, e.g. ISS or GEO. Regarding the site being nearby the coast, this makes sense in terms of safety and nearby railroads/highways, this serves for transportation purposes.

https://github.com/lilaki34/Coursera-Applied-Data-Science-Capstone-Project/blob/main/lab_jupyter_launch_site_location_Folium.ipynb

- The interactive maps might not appear on Chrome, please try with Firefox or another browser. I saw that many people have experienced similar issues with the loading of the maps on GitHub. The maps appear as they should on SkillsNetwork. I tried to save my notebook from SkillsNetwork as pdf but the maps still did not appear.
- https://nbviewer.org/github/lilaki34/Coursera-Applied-Data-Science-Capstone-Project/blob/main/lab_jupyter_launch_site_location_Folium.ipynb --> Try this instead, nbviewer can show all maps.
- Otherwise copy/paste the GitHub URL provided above the first bullet to jupyter nbviewer https://nbviewer.org/ → I tried it, everything appears correctly.

Build a Dashboard with Plotly Dash

- We built an interactive dashboard with Plotly dash
- We plotted pie charts showing success counts for all launching sites.
- We selected the site with the highest launch success ratio.
- We plotted the payload range with the highest launch success rate.
- We plotted the payload range with the lowest launch success rate.
- We selected the booster version with the highest launch success rate.

GitHub code: https://github.com/lilaki34/Coursera-Applied-Data-Science-Capstone-Project/blob/main/Dash%20interactivity-E-Kiosidou.txt

GitHub answers to the questions about the dashboard: https://github.com/lilaki34/Coursera-Applied-Data-Science-Capstone-Project/blob/main/Dash%20interactivity-Questions.txt

Predictive Analysis (Classification)

- We loaded the data and transformed them into a Pandas dataframe.
- We used NumPy to create a separate array from the column “Class” of the dataframe.
- We standardized all rest variables of the dataframe using StandardScaler.
- We used train/test/split to create a training set with 80% of the original data and a test set with the remaining 20%.
- We tried different classifiers: a) logistic regression, b) support vector machine, KNN, decision tree.
- We used accuracy as the metric for our models.
- We found the best performing classification model.

There are some warnings appearing in my notebook, related to some depreciated version that is included in the provided commands, but I could not resolve the issue. All the tasks are completed, it is just that this warning makes it hard to open on GitHub. Please maybe try downloading it instead. The GitHub URL is the following: https://github.com/lilaki34/Coursera-Applied-Data-Science-Capstone-Project/blob/main/SpaceX_Machine%20Learning%20Prediction_Part_5%20(1).ipynb

Also, another way to view my Notebook is through NB viewer. The URL is provided here: https://nbviewer.org/github/lilaki34/Coursera-Applied-Data-Science-CapstoneProject/blob/main/SpaceX_Machine%20Learning%20Prediction_Part_5%20%281%29.ipynb

Predictive Analysis (Classification)

In order to make it easier, I provide screenshots from all tasks:

TASK 1

Create a NumPy array from the column `Class` in `data`, by applying the method `to_nump` series (only one bracket df['name of column']).

```
Y = data['Class'].to_numpy()
Y
```

```
array([0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1,
       1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1])
```

TASK 2

Standardize the data in `X` then reassign it to the variable `X` using the transform provided below.

```
# students get this
transform = preprocessing.StandardScaler()
```

```
X= preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

```
array([[-1.71291154e+00, -1.94814463e-16, -6.53912840e-01,
        -1.57589457e+00, -9.73440458e-01, -1.05999788e-01,
        -1.05999788e-01, -6.54653671e-01, -1.05999788e-01,
        -5.51677284e-01,  3.44342023e+00, -1.85695338e-01,
        -3.33333333e-01, -1.05999788e-01, -2.42535625e-01,
        -4.29197538e-01,  7.97724035e-01, -5.68796459e-01,
        -4.10890702e-01, -4.10890702e-01, -1.50755672e-01,
        -7.97724035e-01, -1.50755672e-01, -3.92232270e-01,
         9.43398113e+00, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
        -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
        -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
```

TASK 3

Use the function train_test_split to split the data X and Y into training and test data. Set the parameter test_siz test data should be assigned to the following labels.

`X_train, X_test, Y_train, Y_test`

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (72, 83) (72,)
Test set: (18, 83) (18,)
```

we can see we only have 18 test samples.

```
Y_test.shape
```

```
(18,)
```

Predictive Analysis (Classification)

In order to make it easier, I provide screenshots from all tasks:

TASK 4

Create a logistic regression object then create a GridSearchCV object `logreg_cv` with cv = 10. Fit the objec
`parameters`.

```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

```
#parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()
lr.fit(X_train, Y_train)
y_pred = lr.predict(X_test)
print(confusion_matrix(Y_test, y_pred))
print(classification_report(Y_test, y_pred))
```

```
[[ 5  1]
 [ 0 12]]
              precision    recall  f1-score   support

           0       1.00      0.83      0.91         6
           1       0.92      1.00      0.96        12

    micro avg       0.94      0.94      0.94        18
    macro avg       0.96      0.92      0.93        18
weighted avg       0.95      0.94      0.94        18
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/linear_model/logi
  FutureWarning)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/linear_model/base
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/relea
  indices = (scores > 0).astype(np.int)
```

```
cv = 10
logreg_cv = GridSearchCV(lr, parameters,cv=cv)
grid_result = logreg_cv.fit(X_train, Y_train)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.847222 using {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
0.847222 (0.099047) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
0.819444 (0.122213) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'}
0.805556 (0.122213) with: {'C': 1, 'penalty': 'l2', 'solver': 'lbfgs'}
```

We output the `GridSearchCV` object for logistic regression. We display the best parameters using the data attribu
validation data using the data attribute `best_score\_`.

```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8472222222222222
```

TASK 5

Calculate the accuracy on the test data using the method `score`:

```
print("test set accuracy :",logreg_cv.score(X_test, Y_test))
```

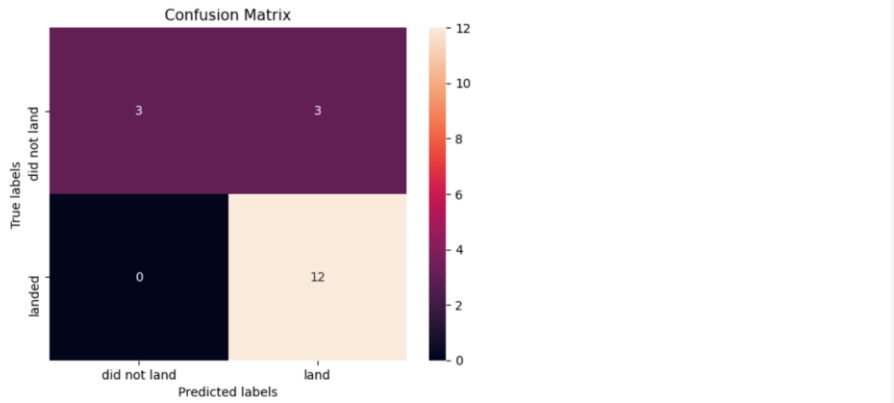
```
test set accuracy : 0.8333333333333334
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/linear_model/base.py:283: DeprecationWarning: `np.int`
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  indices = (scores > 0).astype(np.int)
```

Lets look at the confusion matrix:

```
yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test, yhat)
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/linear_model/base.py:283: DeprecationWarning: `np.int`
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  indices = (scores > 0).astype(np.int)
```



Examining the confusion matrix, we see that logistic regression can distinguish between the different classes. We see that the major problem is false positives.

TASK 6

Create a support vector machine object then create a `GridSearchCV` object `svm_c`
`parameters`.

```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

```
cv=10
svm_cv = GridSearchCV(svm, parameters,cv=cv)
grid_result = svm_cv.fit(X_train, Y_train)
```

```
print("tuned hpyerparameters :(best parameters) ",svm_cv.best_params_)
print("accuracy :",svm_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
accuracy : 0.8472222222222222
```

Predictive Analysis (Classification)

In order to make it easier, I provide screenshots from all tasks:

TASK 7

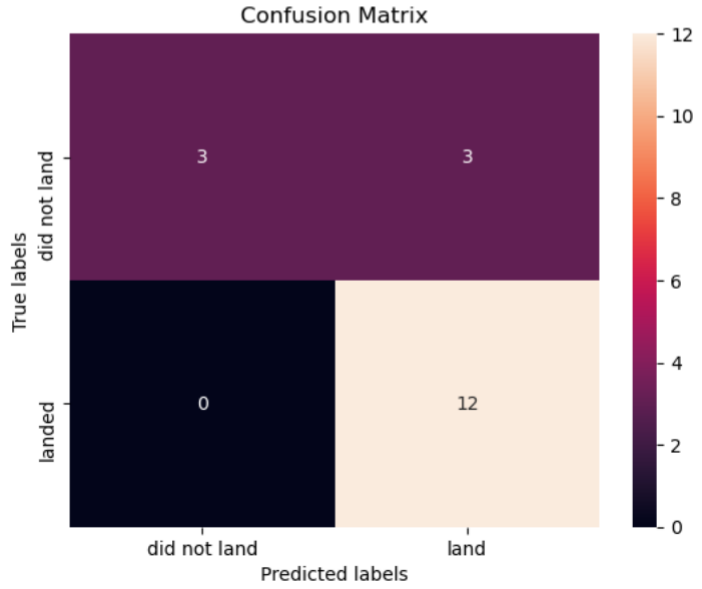
Calculate the accuracy on the test data using the method `score` :

```
print("test set accuracy :",svm_cv.score(X_test, Y_test))
```

```
test set accuracy : 0.8333333333333334
```

We can plot the confusion matrix

```
yhat=svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



TASK 8

Create a decision tree classifier object then create a `GridSearchCV` object `tree_cv` with cross-validation = 10. You can assign the arguments to the predictor you've created in the previous task with its parameters. `parameters` .

```
parameters = {'criterion': ['gini', 'entropy'],
     'splitter': ['best', 'random'],
     'max_depth': [2*n for n in range(1,10)],
     'max_features': ['auto', 'sqrt'],
     'min_samples_leaf': [1, 2, 4],
     'min_samples_split': [2, 5, 10]}

tree = DecisionTreeClassifier()
```

```
cv=10
tree_cv = GridSearchCV(tree, parameters,cv=cv)
grid_result = tree_cv.fit(X_train, Y_train)
```

```
print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 12, 'max_features': 'sqrt', 'min_samples_leaf':
accuracy : 0.8888888888888888
```

TASK 9

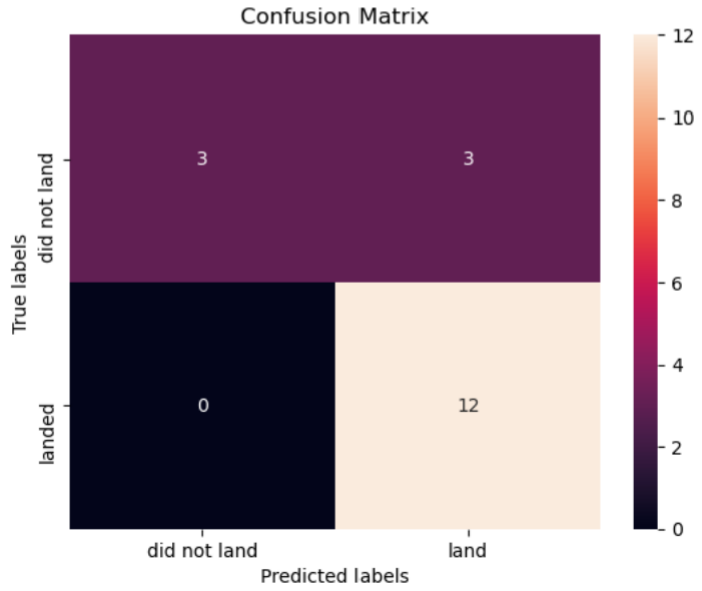
Calculate the accuracy of tree_cv on the test data using the method `score` :

```
print("test set accuracy :",tree_cv.score(X_test, Y_test))
```

```
test set accuracy : 0.8333333333333334
```

We can plot the confusion matrix

```
yhat = svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



Predictive Analysis (Classification)

In order to make it easier, I provide screenshots from all tasks:

TASK 10

Create a k nearest neighbors object then create a `GridSearchCV` object `knn_cv` with c
`parameters`.

```
parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'p': [1,2]}

KNN = KNeighborsClassifier()
```

```
cv=10
knn_cv = GridSearchCV(KNN,parameters,cv=10)
knn_cv.fit(X_train, Y_train)
```

```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform'),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'], 'p
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

```
print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
print("accuracy :",knn_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 9, 'p': 1}
accuracy : 0.8472222222222222
```

TASK 11

Calculate the accuracy of tree_cv on the test data using the method `score`:

```
print("test set accuracy :",knn_cv.score(X_test, Y_test))
```

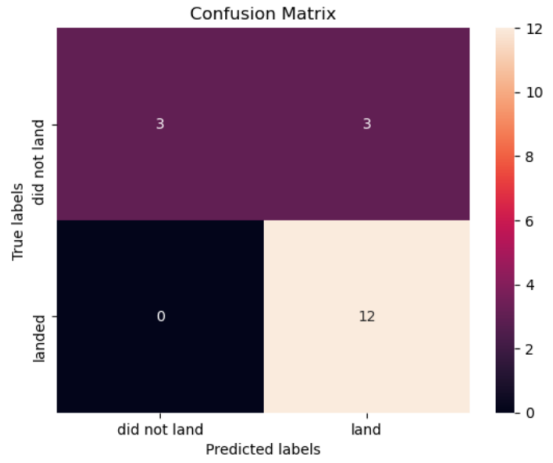
```
test set accuracy : 0.8333333333333334
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/neighbors/base.py
  old_joblib = LooseVersion(joblib_version) < LooseVersion('0.12')
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/neighbors/base.py
  old_joblib = LooseVersion(joblib_version) < LooseVersion('0.12')
```

We can plot the confusion matrix

```
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/neighbors/base.py
  old_joblib = LooseVersion(joblib_version) < LooseVersion('0.12')
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/neighbors/base.py
  old_joblib = LooseVersion(joblib_version) < LooseVersion('0.12')
```



TASK 12

Find the method performs best:

```
print("test set accuracy :",knn_cv.score(X_test, Y_test))
```

```
test set accuracy : 0.8333333333333334
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklear
  old_joblib = LooseVersion(joblib_version) < LooseVersion('0.12')
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklear
  old_joblib = LooseVersion(joblib_version) < LooseVersion('0.12')
```

```
print("test set accuracy :",tree_cv.score(X_test, Y_test))
```

```
test set accuracy : 0.8333333333333334
```

```
print("test set accuracy :",svm_cv.score(X_test, Y_test))
```

```
test set accuracy : 0.8333333333333334
```

```
print("test set accuracy :",logreg_cv.score(X_test, Y_test))
```

```
test set accuracy : 0.8333333333333334
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklear
Deprecated in NumPy 1.20; for more details and guidance: https://nump
  indices = (scores > 0).astype(np.int)
```

```
print("accuracy :",logreg_cv.best_score_)
```

```
accuracy : 0.8472222222222222
```

```
print("accuracy :",svm_cv.best_score_)
```

```
accuracy : 0.8472222222222222
```

```
print("accuracy :",tree_cv.best_score_)
```

```
accuracy : 0.8888888888888888
```

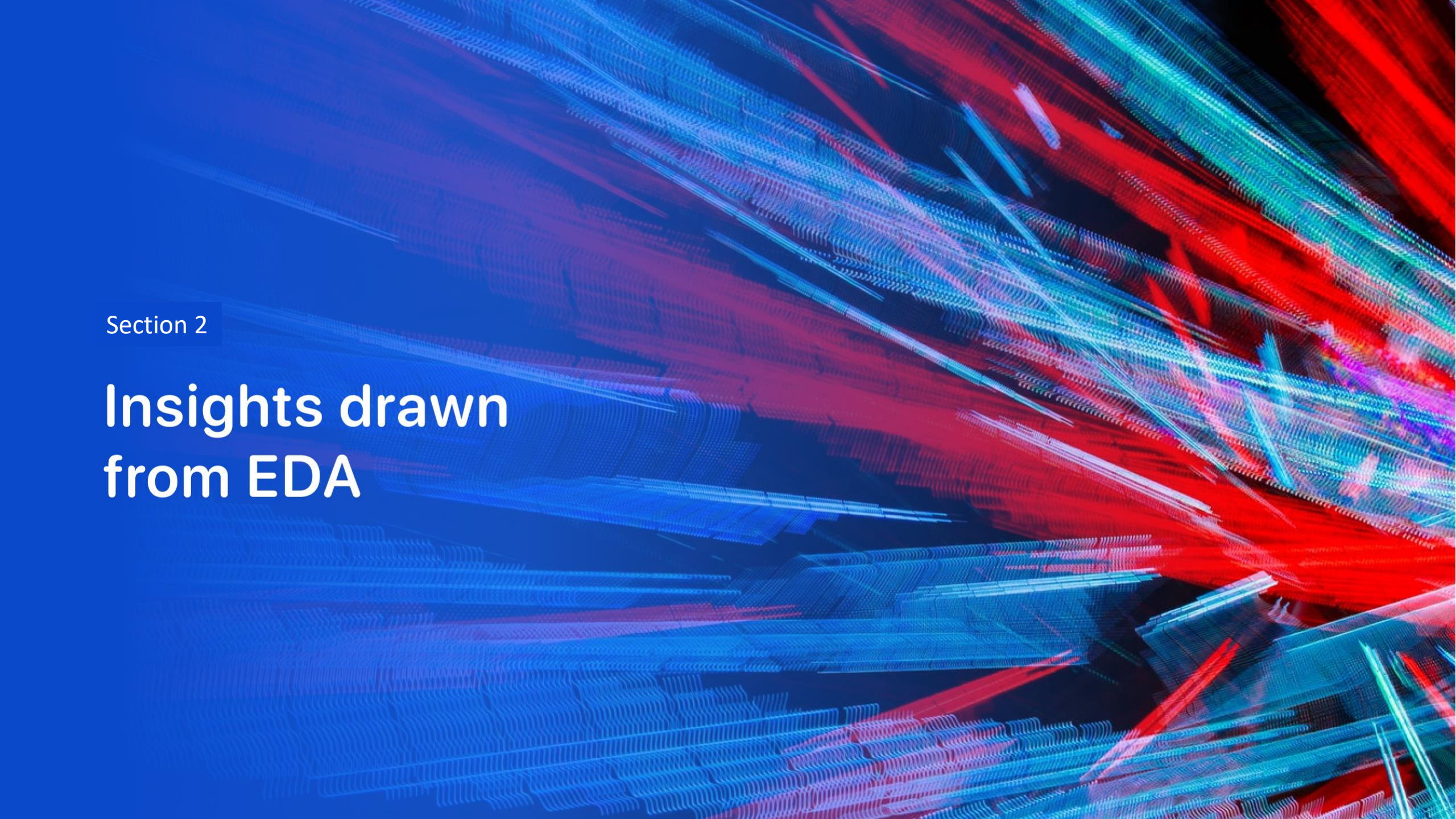
```
print("accuracy :",knn_cv.best_score_)
```

```
accuracy : 0.8472222222222222
```

All methods worked similarly, the tree decision method worked slightly better.

Results

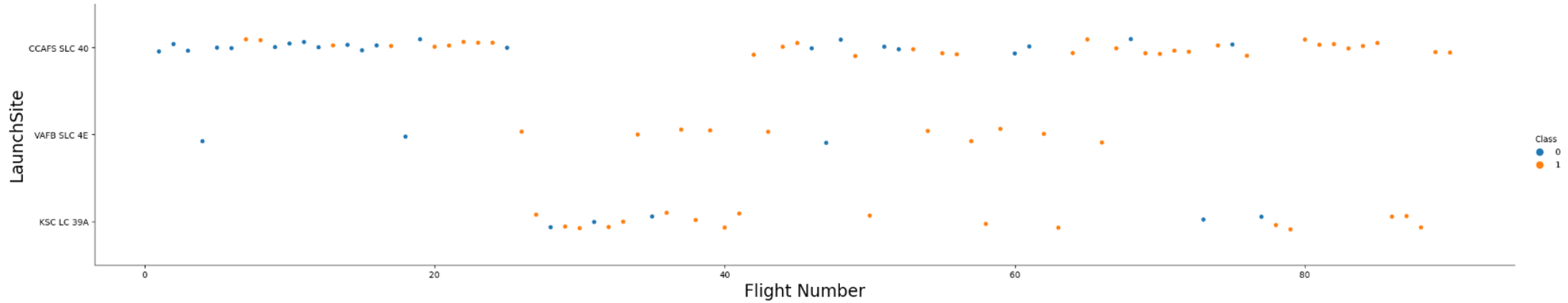
- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results


Section 2
Insights drawn from EDA

Flight Number vs. Launch Site

In [5]:

```
# Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to be the class value
sns.catplot(y="LaunchSite", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("LaunchSite",fontsize=20)
plt.show()
```



1.CCAFS SLC 40: The percentage of failed attempts is 40%, while the percentage of successful landings is 60%. The flight number seems to act in favor of a positive outcome, especially for number of launching attempts > 60 .

2.VAFB SLC 4E: The percentage of failed attempts is 23% and of the successful ones 77%. This site leads mostly to successful landings, even though the number of launches is smaller overall compared to the other two launching sites.

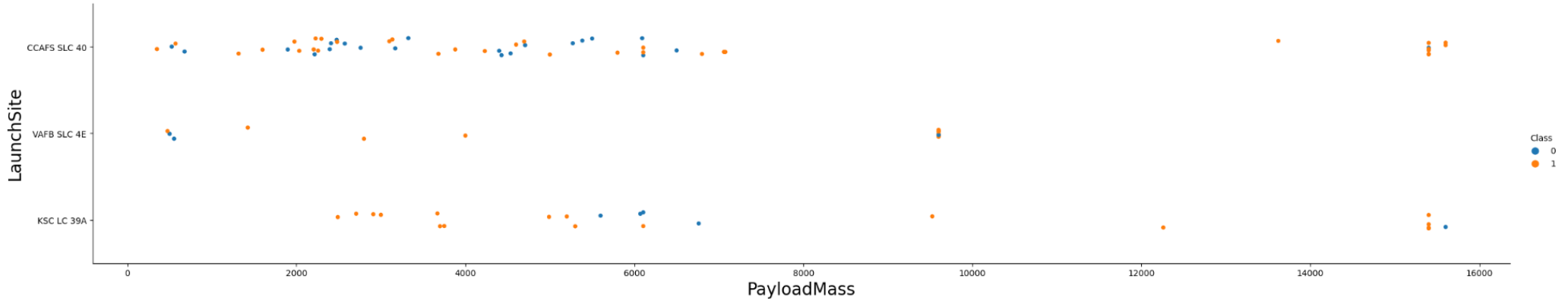
3.KSC LC 39A: Failed-22.7%, Successful-77.3%. Even though the amount of launching attempts is again large, it seems that they do not affect the success outcome. However, compared to site CCAFS SLC 40, there is a lack of data for number of flights < 20, so we cannot extract a conclusion as per whether the increased amount of launch attempts increases the chances for a positive outcome.

Payload vs. Launch Site

In [6]:

```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch site, and hue to be the class value

sns.catplot(y="LaunchSite", x="PayloadMass", hue="Class", data=df, aspect = 5)
plt.xlabel("PayloadMass",fontsize=20)
plt.ylabel("LaunchSite",fontsize=20)
plt.show()
```



Generally: the launching attempts with higher payloads lead to a positive outcome when it comes to the success of landing.

1.CCAFS SLC 40: The number of successful landings over non-successful ones is almost divided for payloads < 8000 kg. However, as soon as the payload increases over 10000 kg, the success rate becomes 6:1.

2.VAFB SLC 4E: for this site, there are not many attempts. For the presented attempts, the successful landings is double the non-successful ones. Hence, this site leads to generally successful landings, even at smaller payloads.

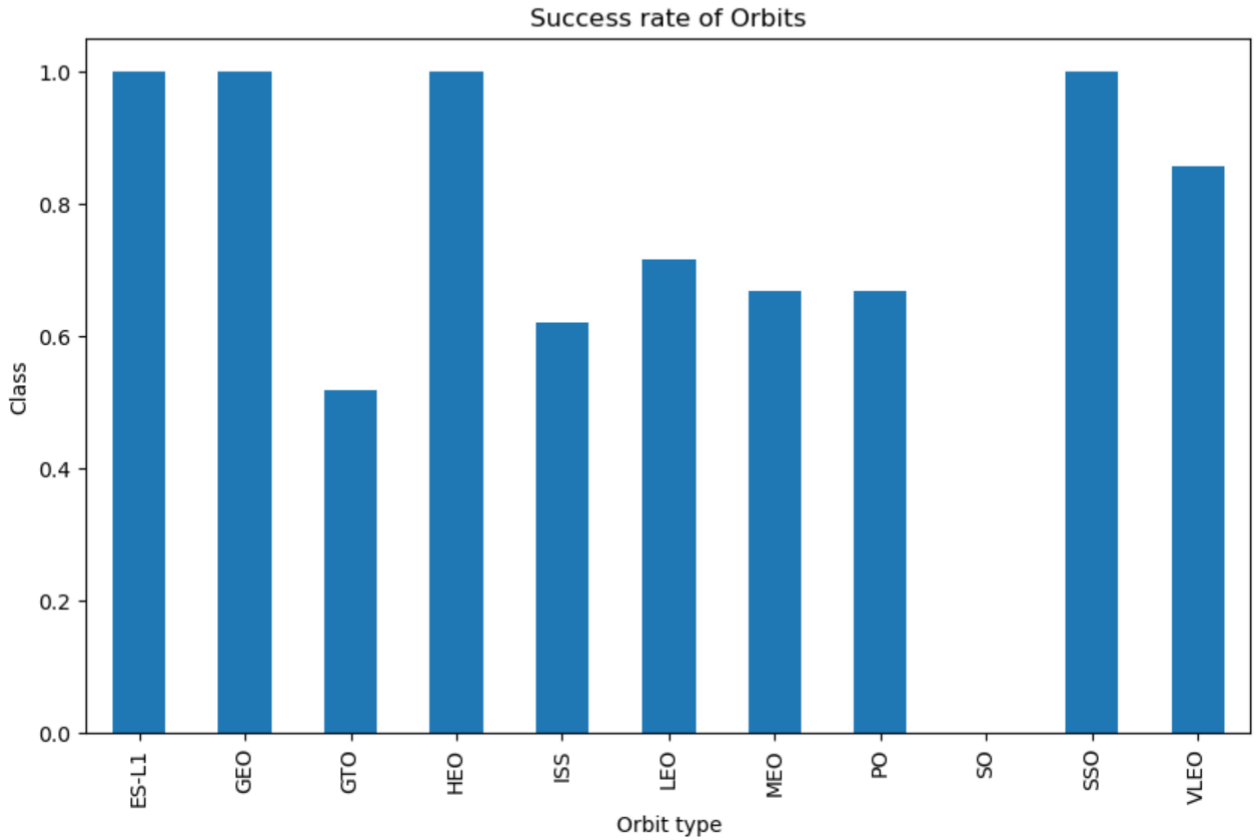
3.KSC LC 39A: The success rate for this site is overall 76.2%. It seems that this site operates very well for payloads < 5500 kg and for payloads > 8000 kg, even though the data are scarce.

Success Rate vs. Orbit Type

The orbit types ES-L1, GEO, HEO and SSO have the highest success rate.

```
[23]: df_group['Class'].plot(kind="bar", title='Success rate of Orbits',
                            ylabel='Class', xlabel='Orbit type', figsize=(10, 6))
```

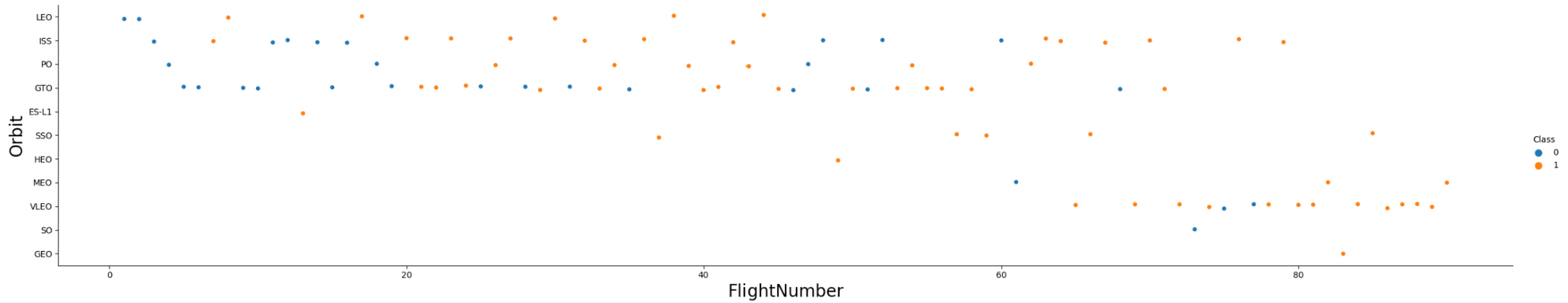
```
[23]: <AxesSubplot:title={'center':'Success rate of Orbits'}, xlabel='Orbit type', ylabel='Class'>
```



Flight Number vs. Orbit Type

In [9]:

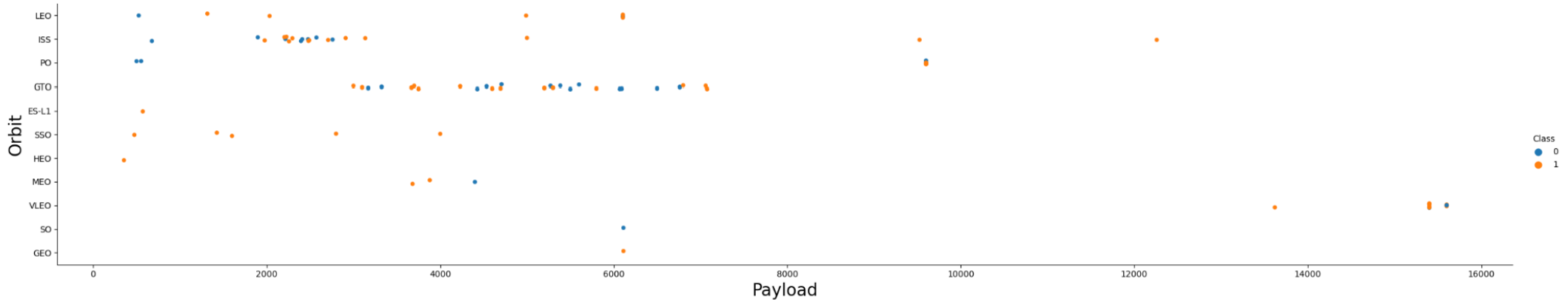
```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class value
sns.catplot(y="Orbit", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("FlightNumber",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```



LEO: Success appears related to the number of flights.
GTO: No relationship between flight number and success.
VLEO: Mostly successful outcome as the flight number increases above 80, which is one of the highest flight numbers overall.
SSO: Completely successful outcome for all flight numbers.
Rest orbits: Mixed outcome (successful/unsuccessful) independent of the number of flights.

Payload vs. Orbit Type

```
[10]: # Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, and hue to be the class value
sns.catplot(y="Orbit", x="PayloadMass", hue="Class", data=df, aspect = 5)
plt.xlabel("Payload",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```

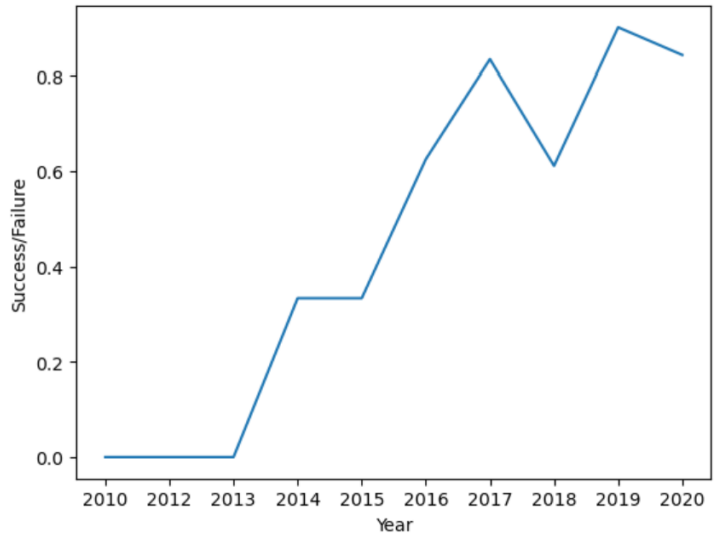


- With heavy payloads the successful landing or positive landing rate are more for SO,LEO and ISS.
- However, for GTO we cannot distinguish this well as both positive landing rate and negative landing(unsuccessful mission) are both there.
- For SSO, all landings were again successful, even through no payload higher than 4000 kg was launched.

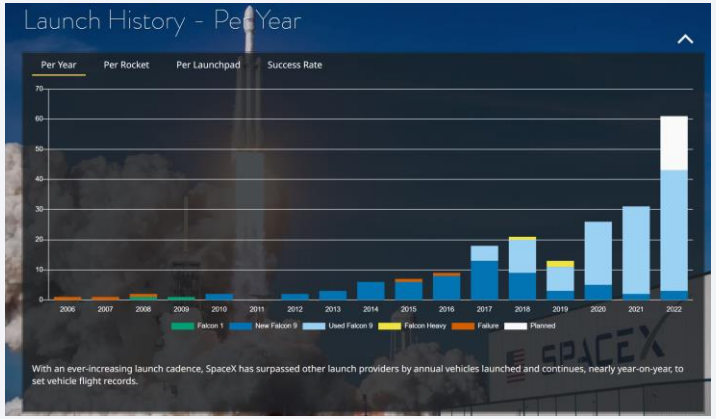
Launch Success Yearly Trend

```
[64]: # A function to Extract years from the date
year=[]
def Extract_year(date):
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year
Extract_year(1)
df["Year"]=year
average_by_year = df.groupby(by="Year").mean()
average_by_year.reset_index(inplace=True)
```

```
[65]: # Plot a line chart with x axis to be the extracted year and y axis to be the success rate
plt.plot(average_by_year["Year"],average_by_year["Class"])
plt.xlabel("Year")
plt.ylabel("Success/Failure")
plt.show()
```



- The “success” metric always refers to the success of landing the booster after launch. For comparison, the statistical data from Space X website are provided.
- Overall, there is an increasing trend both in launches and successful recoveries, as can be seen from the graph on the left and verified from the Space X statistical data.



https://www.spacexstats.xyz/#launchhistory-per-year

https://www.spacexstats.xyz/#recovery-landing-history

All Launch Site Names

DISTINCT: To show the unique values

Task 1

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

```
[133]: %sql select DISTINCT (Launch_Site) from SPACEXTBL
```

```
   sqlite:///my_data1.csv.db
   sqlite:///my_data1.db
 * sqlite:///my_data2.db
Done.
```

[133]:

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

https://github.com/lilaki34/testrepo/blob/main/jupyter-labs-eda-sql-coursera_sqllite%20(1).ipynb

Launch Site Names Begin with 'CCA'

Task 2

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

In [134…

```
%sql SELECT * from SPACEXTBL where (LAUNCH_SITE) LIKE 'CCA%' LIMIT 5
```

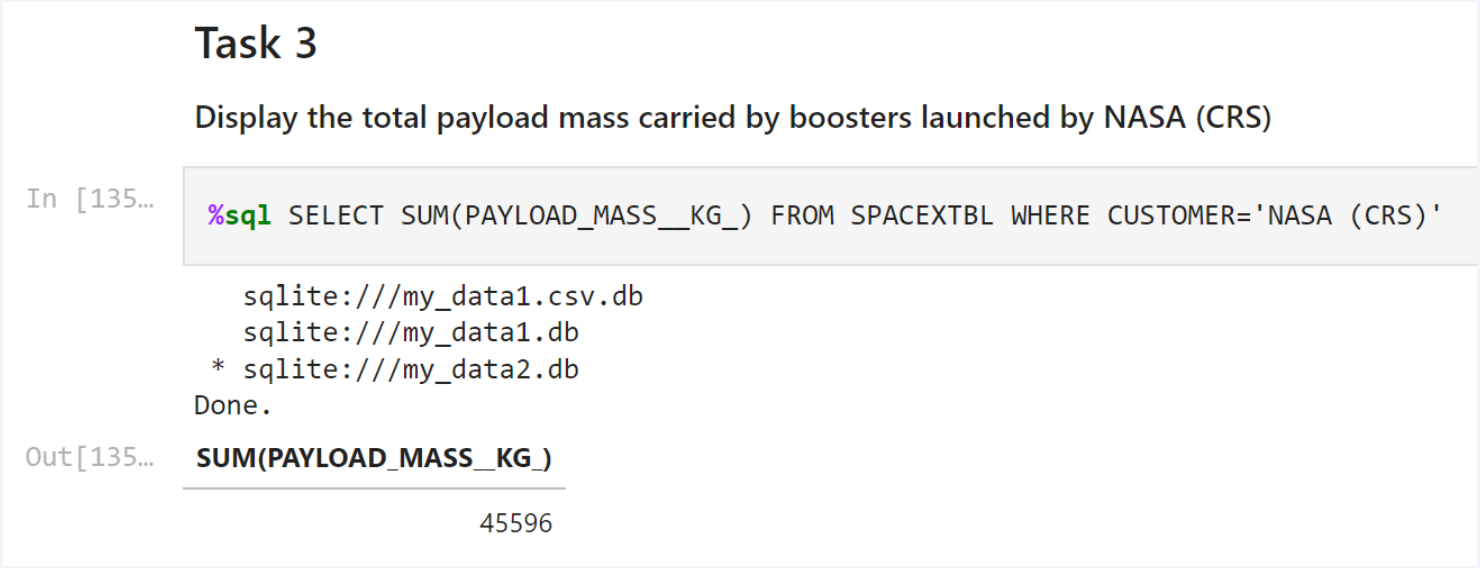
```
   sqlite:///my_data1.csv.db
   sqlite:///my_data1.db
 * sqlite:///my_data2.db
Done.
```

Out[134…

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

LIKE ‘CCA%’: To show only the launch sites that start with ‘CCA’.
LIMIT 5: To display only 5 records as requested.

Total Payload Mass



Task 3

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

In [135…

```
%sql SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE CUSTOMER='NASA (CRS)'
```

```
   sqlite:///my_data1.csv.db
   sqlite:///my_data1.db
 * sqlite:///my_data2.db
Done.
```

Out[135…

SUM(PAYLOAD_MASS__KG_)
45596

SUM: To calculate the summation of the payload mass column
WHERE CUSTOMER ='NASA (CRS)': To calculate the total payload mass only for the case where NASA is the customer.

Average Payload Mass by F9 v1.1

Task 4

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

```
[136]: %sql SELECT AVG(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE BOOSTER_VERSION='F9 v1.1'
   sqlite:///my_data1.csv.db
   sqlite:///my_data1.db
 * sqlite:///my_data2.db
Done.
```

[136]:

AVG(PAYLOAD_MASS__KG_)
2928.4

AVG: To calculate the average of the payload mass column

WHERE BOOSTER_VERSION ='F9 v1.1': To calculate the average payload mass only for the case where the booster version is the F9 v1.1.

First Successful Ground Landing Date

Task 5

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

Hint:Use min function

```
[137]: %sql SELECT min(DATE) FROM SPACEXTBL WHERE LANDING_OUTCOME='Success (ground pad)'
   sqlite:///my_data1.csv.db
   sqlite:///my_data1.db
 * sqlite:///my_data2.db
Done.
[137]: min(DATE)
       2015-12-22
```

MIN: To calculate the minimum date.
WHERE LANDING_OUTCOME='Success (ground pad)': To calculate the minimum date for the case where there was a successful landing on the ground pad.

Successful Drone Ship Landing with Payload between 4000 and 6000

Task 6

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

[138]:
```
%sql SELECT BOOSTER_VERSION FROM SPACEXTBL WHERE PAYLOAD_MASS__KG_ between 4000 and 6000 AND LANDING_OUTCOME='Success (drone ship)'
```

```
   sqlite:///my_data1.csv.db
   sqlite:///my_data1.db
  * sqlite:///my_data2.db
Done.
```

[138]:

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

In this case we have 2 conditions:

WHERE PAYLOAD_MASS_KG between 4000 and 6000: To include the payload in the requested range.

AND LANDING_OUTCOME='Success (drone ship)': To display the boosters that carried this payload mass range and had success with landing on a drone ship.

Total Number of Successful and Failure Mission Outcomes

Task 7

List the total number of successful and failure mission outcomes

```
[13]: %sql SELECT COUNT(*) FROM SPACEXTBL WHERE MISSION_OUTCOME LIKE '%Success%'
 * sqlite:///my_data2.db
Done.
[13]: COUNT(*)
      100
```

```
[17]: %sql SELECT COUNT(*) FROM SPACEXTBL WHERE MISSION_OUTCOME LIKE '%Failure%'
 * sqlite:///my_data2.db
Done.
[17]: COUNT(*)
      1
```

SELECT COUNT(*): To measure the values we are looking for.
WHERE MISSION_OUTCOME LIKE '%SUCCESS%'/'%FAILURE%': The condition to count only for the times each of these strings appear in the table.

Boosters Carried Maximum Payload

Task 8

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

```
[14]: %sql SELECT BOOSTER_VERSION FROM SPACEXTBL WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTBL)
 * sqlite:///my_data2.db
Done.
```

[14]:

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

WHERE PAYLOAD_MASS_KG = (SELECT MAX(PAYLOAD_MASS_KG) FROM SPACEXTBL): In order to calculate the maximum value in a column, the syntax includes a subquery.
We SELECT the booster from the BOOSTER_VERSION that carried the maximum payload.

2015 Launch Records

Task 9

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

Note: SQLLite does not support monthnames. So you need to use substr(Date, 4, 2) as month to get the months and substr(Date,7,4)='2015' for year.

In [150…

```
%sql SELECT substr(Date, 6, 2),MISSION_OUTCOME,BOOSTER_VERSION,LAUNCH_SITE FROM SPACEXTBL where LANDING_OUTCOME = 'Failure (drone ship)' AND substr(Da
```

```
   sqlite:///my_data1.csv.db
   sqlite:///my_data1.db
 * sqlite:///my_data2.db
Done.
```

Out[150…

substr(Date, 6, 2)	Mission_Outcome	Booster_Version	Launch_Site
01	Success	F9 v1.1 B1012	CCAFS LC-40
04	Success	F9 v1.1 B1015	CCAFS LC-40

SELECT substr(Date, 6, 2),MISSION_OUTCOME,BOOSTER_VERSION,LAUNCH_SITE FROM SPACEXTBL where LANDING_OUTCOME = 'Failure (drone ship)' AND substr(Date,1,4)='2015’

substr(Date,6,2) correponds to the month, 1- January, 4-April

From the actual SpaceX table we can see that the following lines correspond to the solution of Task 9, which agrees with what the code gives. A note: the code gives “mission outcome” as requested, which is “Success” both times, even though the “landing outcome” is “failure (drone ship)”.

10/**1**/2015 9:47:00 F9 v1.1 B1012 CCAFS LC-40 SpaceX CRS-5 2395 LEO (ISS) NASA (CRS) Success **Failure (drone ship)**

14/**4**/2015 20:10:00 F9 v1.1 B1015 CCAFS LC-40 SpaceX CRS-6 1898 LEO (ISS) NASA (CRS) Success **Failure (drone ship)**

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

I had some issues with the datafile, regarding “Landing_Outcome” syntax and the Watson subscription, so I downloaded SQLite, uploaded the SPaceXTBL there and ran the correct SQL code in the SQL environment.

```
SELECT LANDING_OUTCOME,COUNT(LANDING_OUTCOME) FROM SPACEXTBL WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20' AND LANDING_OUTCOME LIKE '%Success%'
    GROUP BY LANDING_OUTCOME
    ORDER BY COUNT(LANDING_OUTCOME) DESC
```

Landing_Outcome	UNT(LANDING_OUTCOM
Success (drone ship)	5
Success (ground pad)	3

SELECT LANDING_OUTCOME: To find the values of interest in this column.
SELECT COUNT(LANDING_OUTCOME): To measure the values of the same group.
WHERE DATE BETWEEN 2010-06-04 AND 2017-03-20: Condition that the value count applies to.
GROUP BY LANDING_OUTCOME: Group the results based on the different types of outcomes
ORDER BY COUNT(LANDING OUTCOME) DESC: To present the results in descending order starting with the group that has the most counts.

Section 3

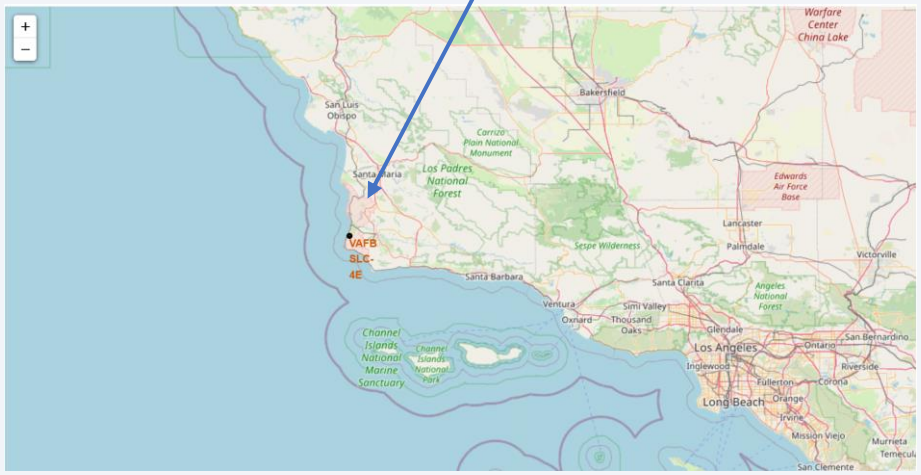
Launch Sites
Proximities Analysis

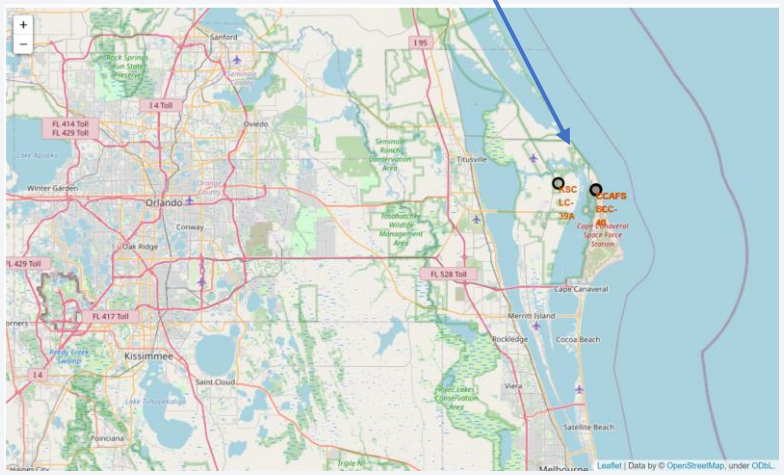
All launch sites on the global map

- ✓ "Launches from Vandenberg fly southward, allowing payloads to be placed in high-inclination orbits such as polar or Sun-synchronous orbit, which allow full global coverage on a regular basis and are often used for weather, Earth observation, and reconnaissance satellites"

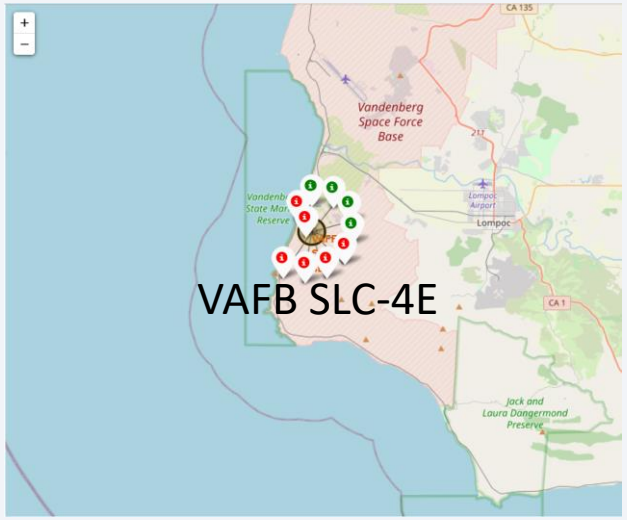
https://en.wikipedia.org/wiki/Vandenberg_Space_Launch_Complex_6

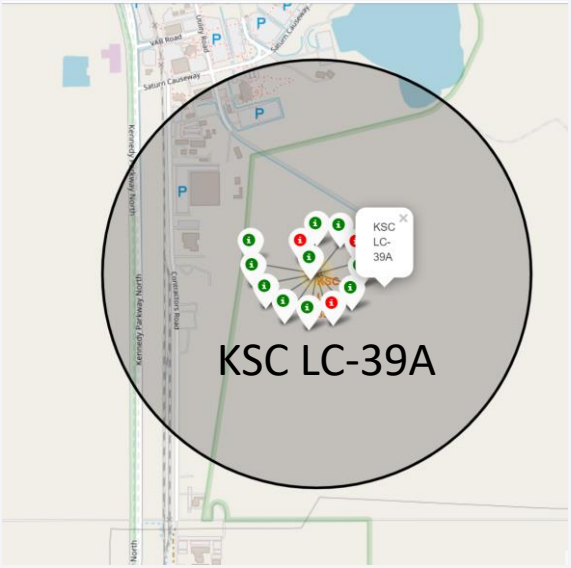
- ✓ 3 launching sites are in Florida, taking advantage of the proximity to the equator where the kinetic energy is greater, allowing the use of less energy during launching.

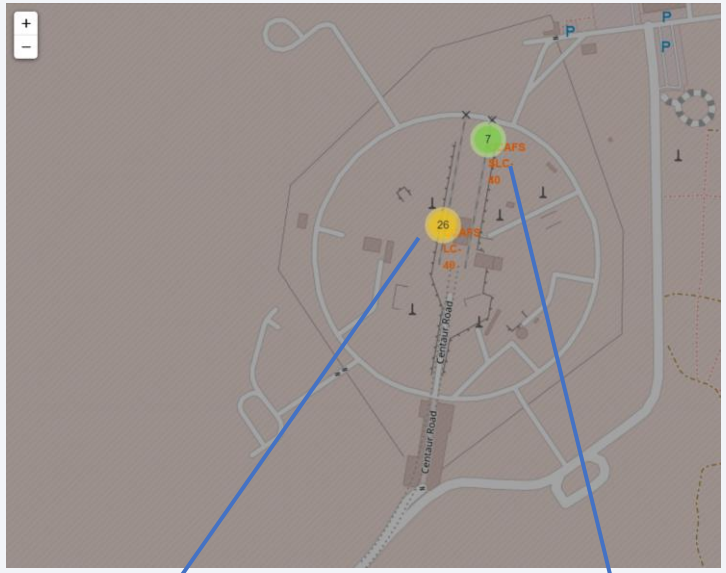




Markers showing success/failed launches for each site on the map



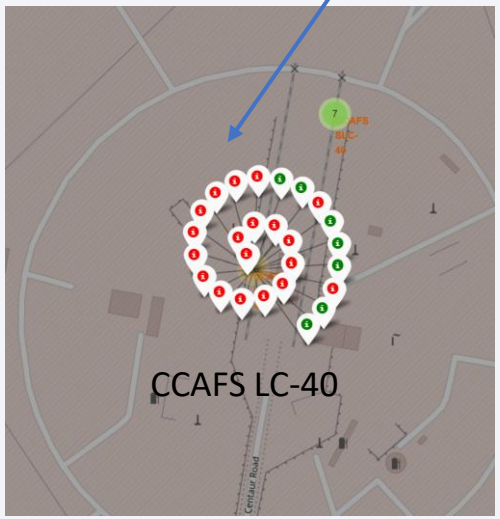


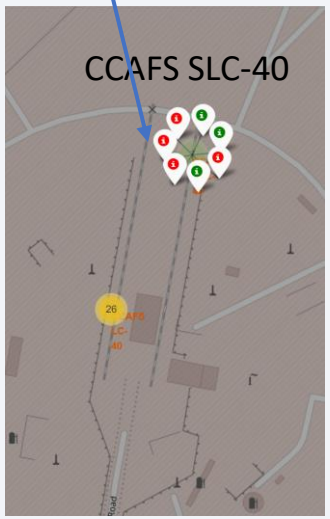


Green marker: successful launches

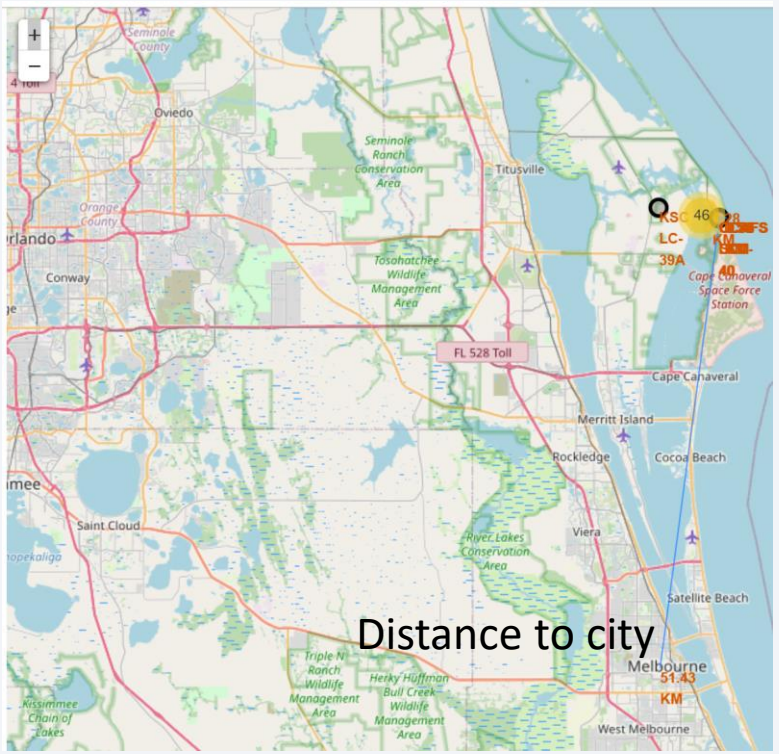
Red marker: failed launches

✓ KSC LC-39A seems to have the highest number of successful launches



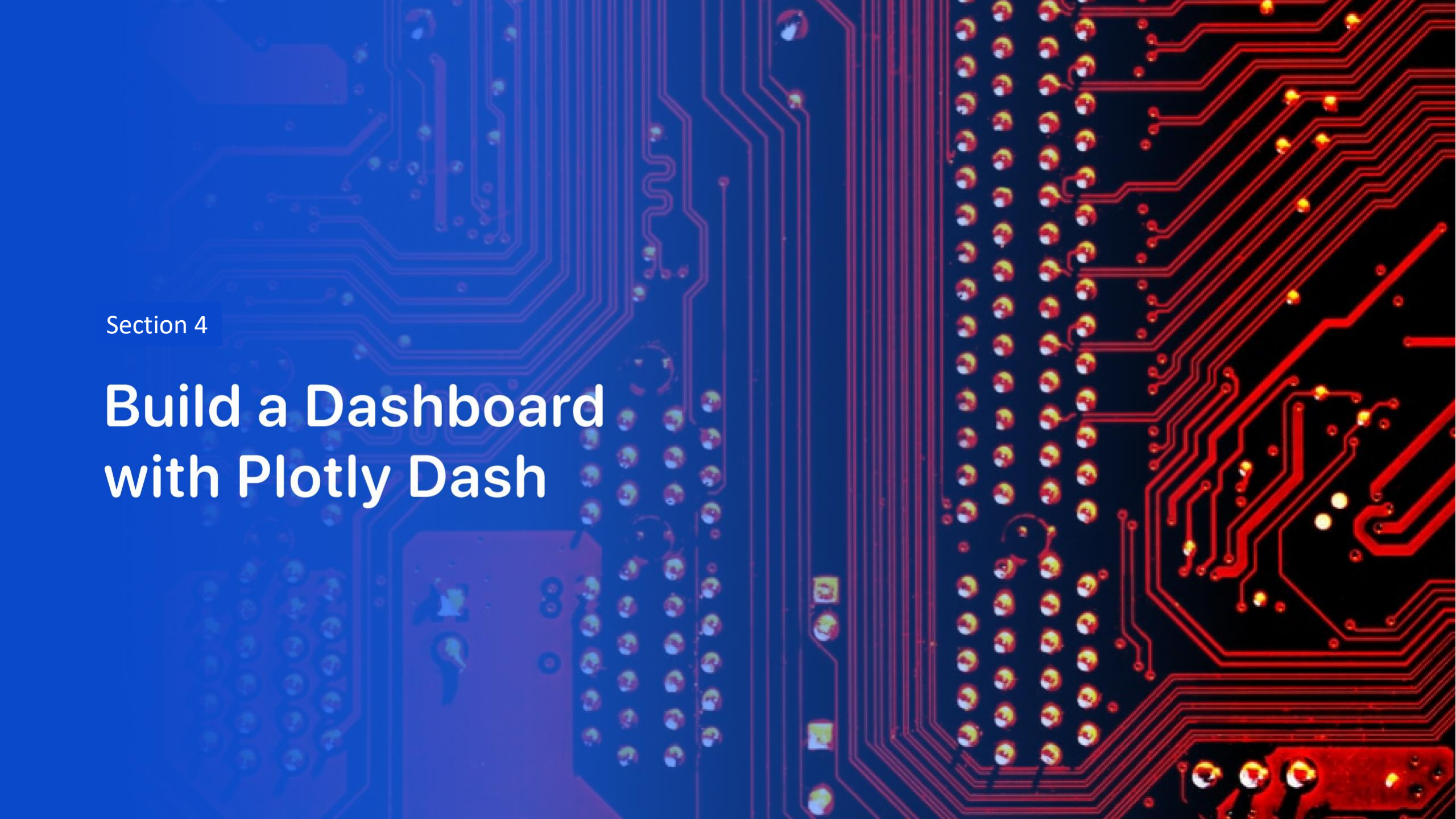


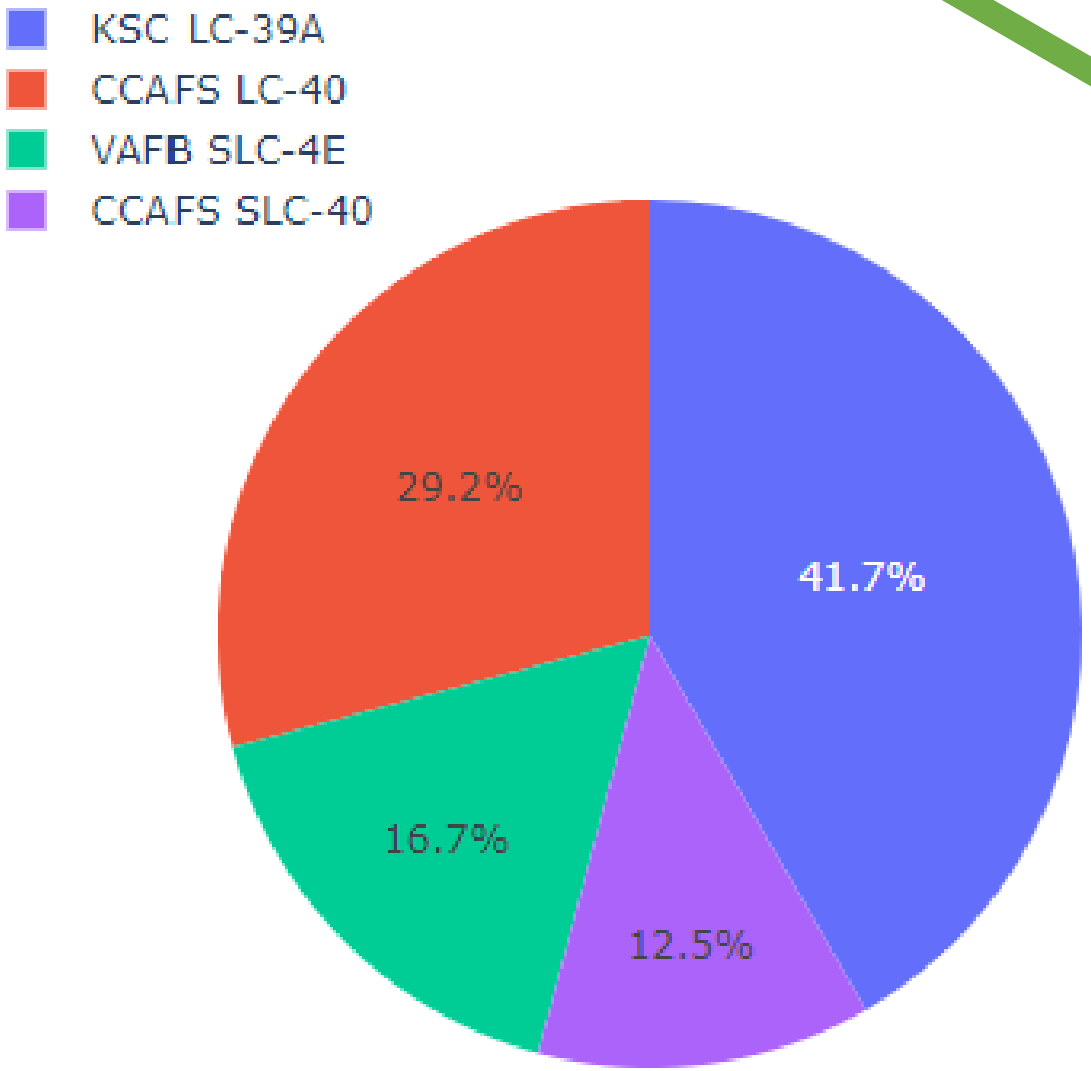
Proximities to the launch sites



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

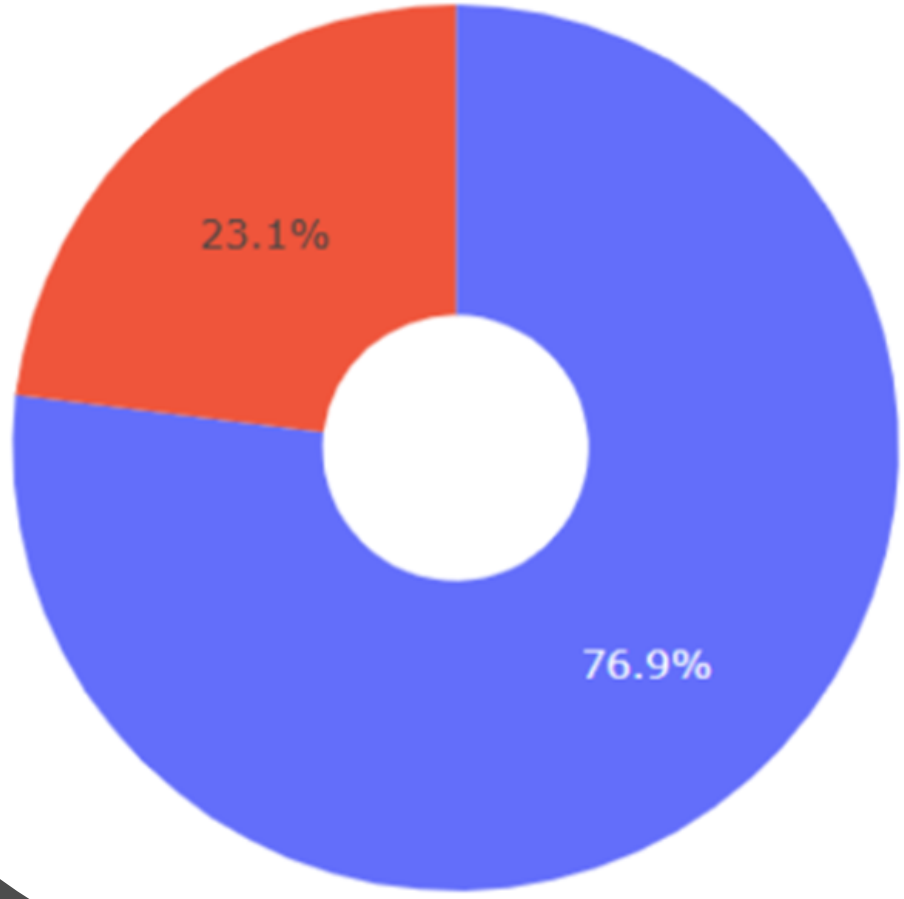
1. Launch sites are close to highways and railways, for easy transportation of people and required cargo, spare parts, heavy equipment and heavy parts etc.
2. Launch sites are in proximity to the coastline, mainly for safety purposes. For example, if the crew needs to abort during launching or in the unlikely event of a failed launch, the public needs to be protected from debris.
3. For the reasons mentioned above, the launch sites are also kept away from the cities, isolated by still accessible.


Section 4
Build a Dashboard with Plotly Dash



Success count for all launch sites

KSC LC-39A is the most successful site



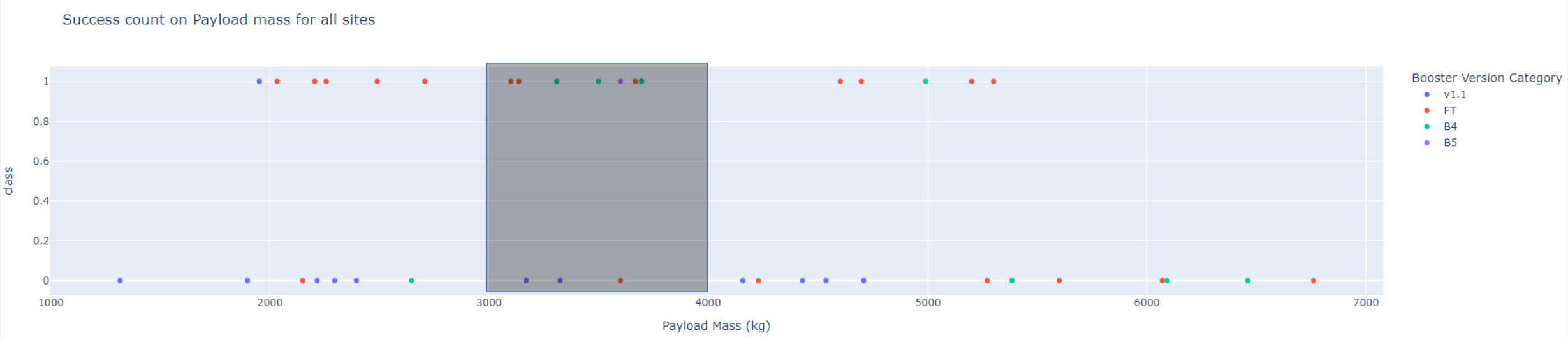
Launch site with highest launch success ratio

KSC LC-39A has the highest launch success ratio: 76.9% over 23.1%

Payload range with the highest launch success rate

Which payload range(s) has the highest launch success rate?

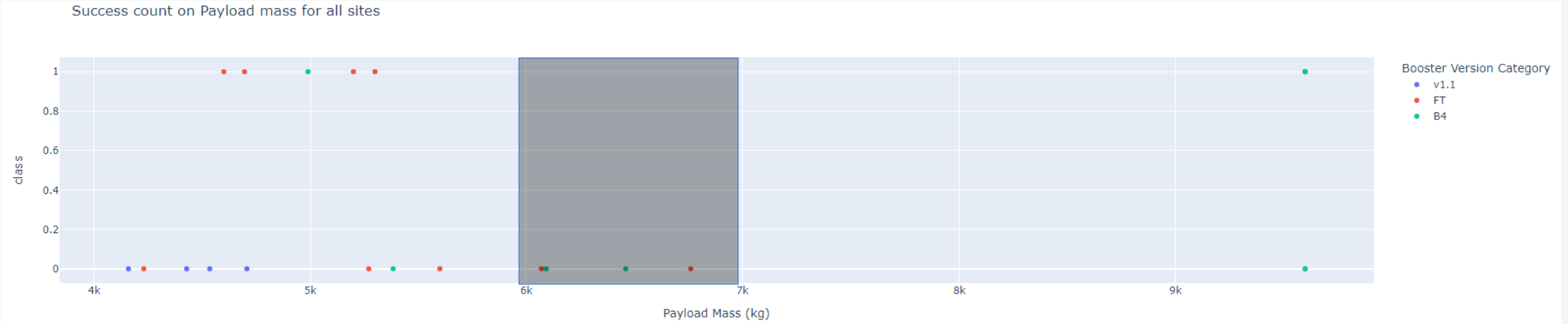
The payload range 3k-4k kg. The success rate is 7:3



Payload range with the lowest launch success rate

Which payload range(s) has the lowest launch success rate?

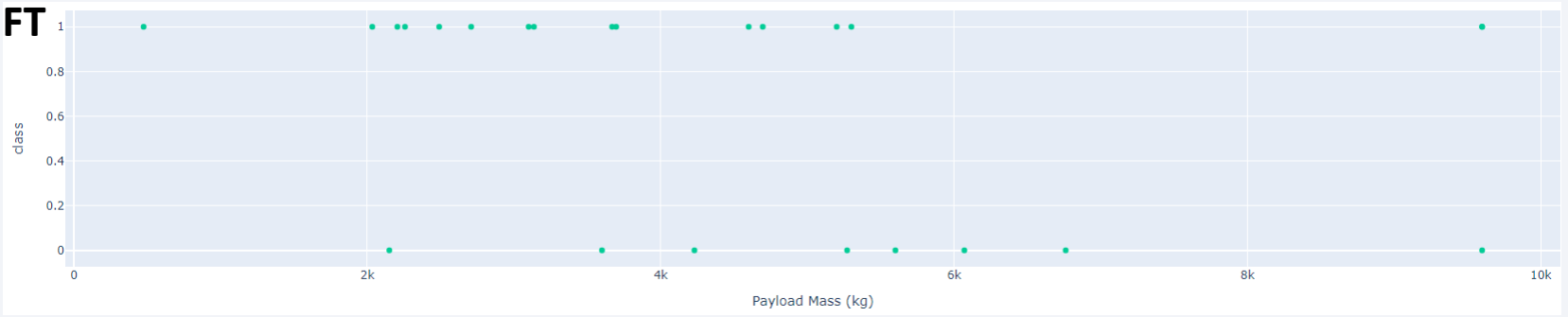
The payload range 6k-7k kg. All attempts were unsuccessful.



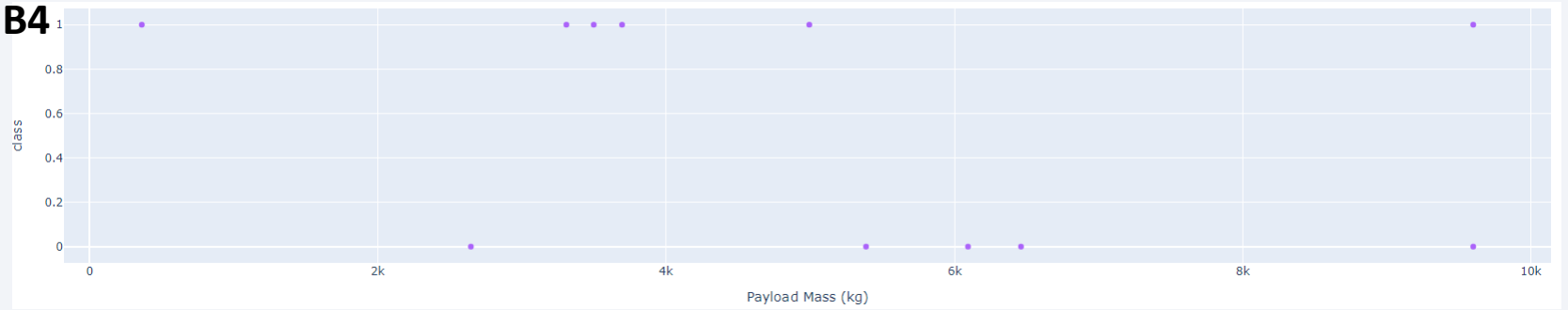
Booster version with the highest launch success rate

The FT Booster exhibited the highest launch success rate.

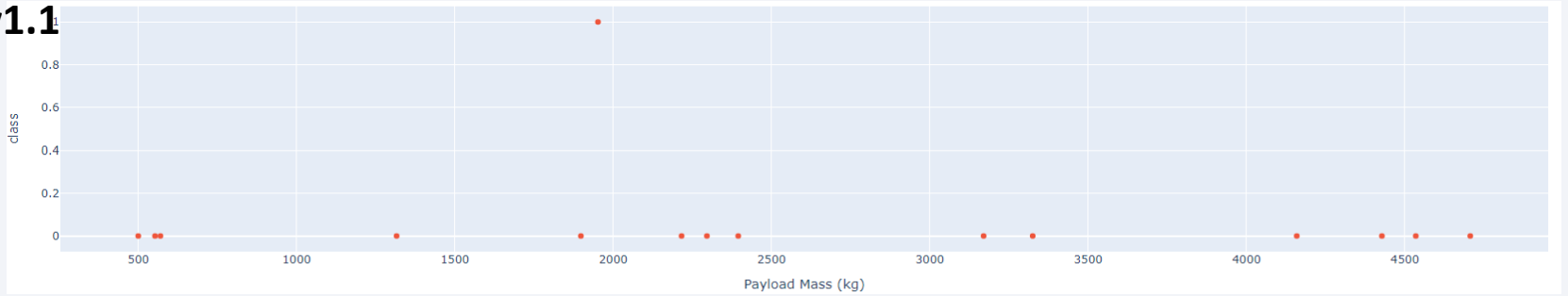
FT



B4



v1.1



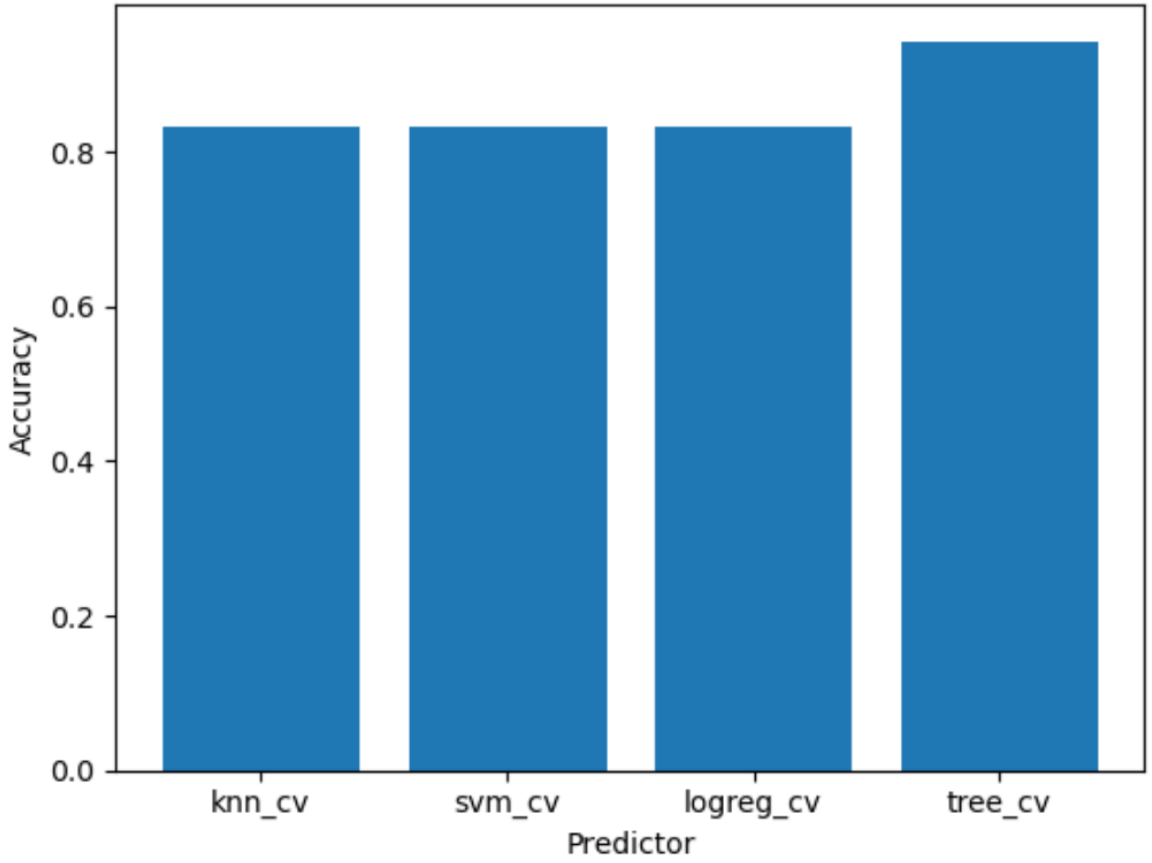
- ✓ The FT booster was used in a very wide range of payloads and had a success rate of 15:8 overall.
- ✓ B4 was also used in the same payload range but had an overall success rate of 6:5.
- ✓ v1.1 was used in smaller range of payloads, up to 5000 kg, and it only had one success over 14 fails.
- ✓ The rest boosters are not depicted because they exhibited even worse behaviors or were not tested in a sufficiently wide payload range.

Section 5

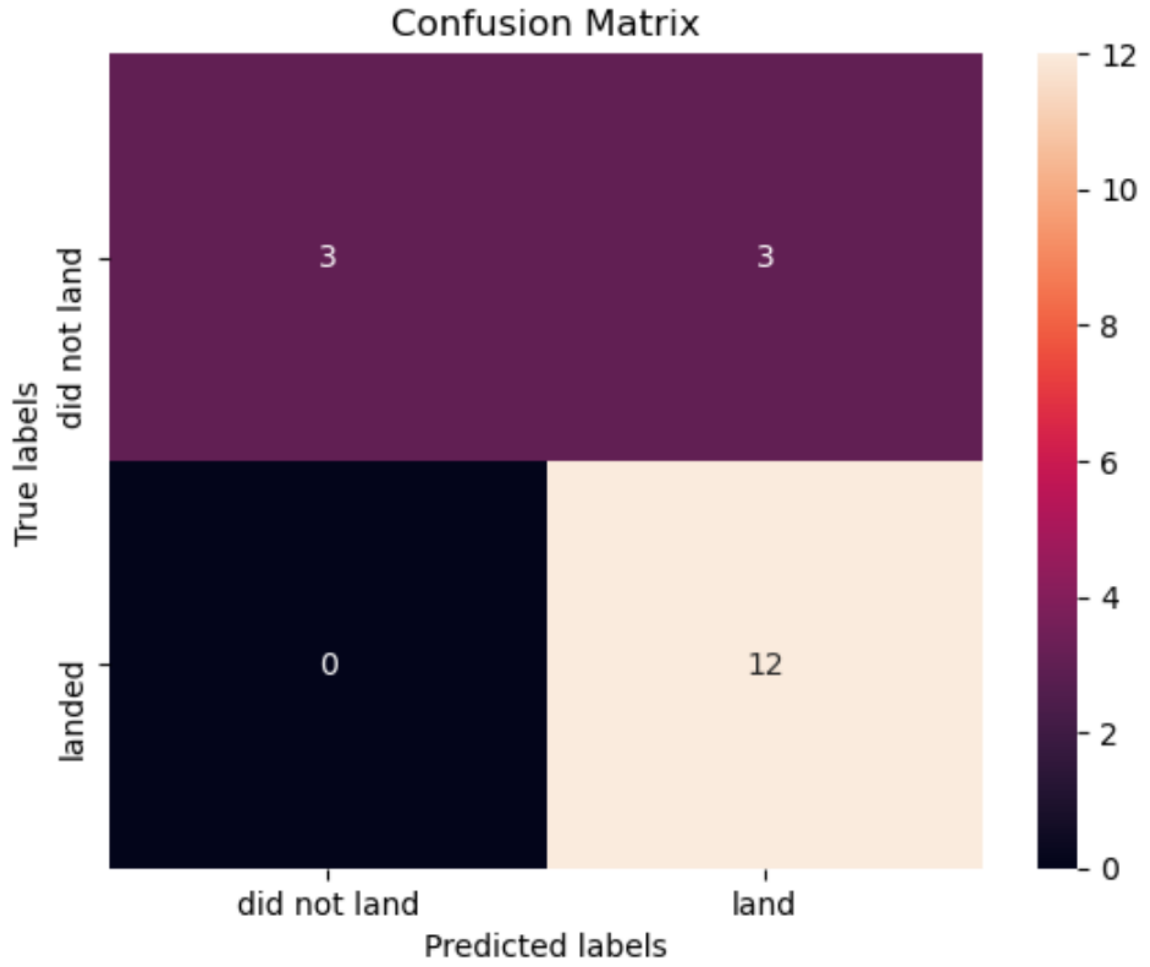
Predictive Analysis (Classification)

The decision tree classifier exhibited the highest accuracy overall, equal to 0.9244.

The rest classifiers exhibited similar accuracy, equal to 0.833.



Examining the confusion matrix, we see that the decision tree classifier can distinguish between the different classes. We see that the major problem is the false positives.



Conclusions

- The larger the flight amount at a launch site, the greater the success rate at a launch site.
- For heavy payloads the successful landing or positive landing rate are more for SO,LEO and ISS.
- There is an increasing trend both in launches and successful recoveries for years 2013 to 2020.
- Orbits ES-L1, GEO, HEO, SSO, VLEO had the highest success rates.
- Launching sites are located close to the equator and the ocean, in order to take advantage of Earth's rotational speed. They are safely away from populated areas but still accessible from highway and railroads.
- KSC LC-39A had the most successful launches of any sites.
- The payload range 3000-4000 kg is the most successful among all booster versions.
- The payload range 6000-7000 kg is the most unsuccessful among all booster versions.
- The FT Booster exhibited the highest launch success rate.
- The Decision tree classifier is the best machine learning algorithm for this task.

Thank you!