ANALYSING “SPACEY” AS COMPETITOR

Ali Mohammadi

05/02/2024

OUTLINE

- Executive Summary
- Introduction
- Methodology
- Results
  - Visualization – Charts
  - Dashboard
- Discussion
  - Findings & Implications
- Conclusion
- Appendix

EXECUTIVE SUMMARY

- SPACEY would like to compete with SpaceX Company
- Our goals are:
  - Determining price of each launch
  - predict if SpaceX will reuse the first stage
  - Determining the launch cost based on:
    - Determining if the first stage can be recovered

INTRODUCTION

- Data collection and Data Wrangling
- EDA and Interactive Visual Analytics
- Predictive Analysis
- EDA with Visualization
- EDA with SQL Database
- Interactive Map with Folium
- Plotly Dash Dashboard
- Predictive Analysis (Classification)
- Conclusion

METHODOLOGY

- use the spacex data to predict weather this company attempt to land a rocket or not
- perform get request using requests library to obtain launch data
- json_normalize() function: normalize json data to flat table
- web scrapping with Beautifulsoup
- wrangling data using an API
- Filtering/Sampling Data
- Dealing with Nulls

RESULTS

DataCollection and Wrangling

1- Data Collection API

Task 1

Task 1: Request and parse the SpaceX launch data using the GET request

To make the requested JSON results more consistent, we will use the following static response object for this project:

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

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

```
response.status_code
```

```
200
```

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

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

Task 1 - Continue

Using the dataframe `data` print the first 5 rows

[12]:
```python
# Get the head of the dataframe
data.head()
```

[12]:

	static_fire_date_utc	static_fire_date_unix	net	window	rocket	success	failures	details	crew	ships	capsules	
0	2006-03-17T00:00:00.000Z	1.142554e+09	False	0.0	5e9d0d95eda69955f709d1eb	False	[{'time': 33, 'altitude': None, 'reason': 'merlin engine failure'}]	Engine failure at 33 seconds and loss of vehicle	[]	[]	[]	[5eb0e4b5b6c
1	None	NaN	False	0.0	5e9d0d95eda69955f709d1eb	False	[{'time': 301, 'altitude': 289, 'reason': 'harmonic oscillation leading to premature engine shutdown'}]	Successful first stage burn and transition to second stage, maximum altitude 289 km, Premature engine shutdown at T+7 min 30 s, Failed to reach orbit, Failed to	[]	[]	[]	[5eb0e4b6b6c

Task 2

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

Finally we will remove the Falcon 1 launches keeping only the Falcon 9 launches. Filter the data dataframe using the `BoosterVersion` column to only keep the Falcon 9 launches. Save the filtered data to a new dataframe called `data_falcon9`.

```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = df[df['BoosterVersion'] == 'Falcon 9']
data_falcon9.head()
```

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Ser
4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B00
5	8	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B00
6	10	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B00
7	11	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None	1.0	0	B10
8	12	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B10

Task 2 - Continue

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

[26]:
```
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/pandas/core/indexing.py:1773: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
```

[26]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude
4	1	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0003	-80.577366	28.561857
5	2	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0005	-80.577366	28.561857
6	3	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0007	-80.577366	28.561857
7	4	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None	1.0	0	B1003	-120.610829	34.632093
8	5	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B1004	-80.577366	28.561857
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
89	86	2020-09-03	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	2	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	12	B1060	-80.603956	28.608058
90	87	2020-10-06	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	3	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	13	B1058	-80.603956	28.608058
91	88	2020-10-18	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	6	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	12	B1051	-80.603956	28.608058
92	89	2020-10-24	Falcon 9	15600.0	VLEO	CCSFS SLC 40	True ASDS	3	True	True	True	5e9e3033383ecbb9e534e7cc	5.0	12	B1060	-80.577366	28.561857
93	90	2020-11-05	Falcon 9	3681.0	MEO	CCSFS SLC 40	True ASDS	1	True	False	True	5e9e3032383ecb6bb234e7ca	5.0	8	B1062	-80.577366	28.561857

90 rows × 17 columns

Task 3

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.

```
# Calculate the mean value of PayloadMass column
meanval = data_falcon9['PayloadMass'].mean()
# Replace the np.nan values with its mean value
new1 = np.nan_to_num(data_falcon9['PayloadMass'], nan= meanval)
df1 = pd.DataFrame(new1)
data_falcon9['PayloadMass'] = df1
data_falcon9.head()
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/ipykernel_launcher.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude
4	1	2010-06-04	Falcon 9	3170.0	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0003	-80.577366	28.561857
5	2	2012-05-22	Falcon 9	3325.0	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0005	-80.577366	28.561857
6	3	2013-03-01	Falcon 9	2296.0	ISS	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0007	-80.577366	28.561857
7	4	2013-09-29	Falcon 9	1316.0	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None	1.0	0	B1003	-120.610829	34.632093
8	5	2013-12-03	Falcon 9	4535.0	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B1004	-80.577366	28.561857

RESULTS

DataCollection and Wrangling

2- Data Collection Web Scrapping

Task 1

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.

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

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

[6]:
```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content

soup = BeautifulSoup(html_spacex,"html.parser")
```

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

[7]:
```
# Use soup.title attribute
print(soup.title)
```

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

Task 2

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

```
[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.

```
[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>
<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-b
ref="#cite_note-booster-11">[b]</a></sup>
</th>
<th scope="col">Launch site
</th>
```

Task 2 - Continue

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

[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
t_c = first_launch_table.find_all('th')
for name in t_c:
    if name is not None and len(name) > 0:
        column_names.append((name.text)[:len(name.text)-1])
```

Check the extracted column names

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

```
['Flight No.', 'Date andtime (UTC)', 'Version,Booster [b]', 'Launch site', 'Payload[c]', 'Payload mass', 'Orbit', 'Customer', 'Launchoutcome', 'Boosterlanding', '1', '2', '3', '4', '5', '6', '7']
```

Task 3

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

[12]:
```
launch_dict= dict.fromkeys(column_names)
# launch_dict.keys()
# Remove an irrelvant column
del launch_dict['1']
del launch_dict['2']
del launch_dict['3']
del launch_dict['4']
del launch_dict['5']
del launch_dict['6']
del launch_dict['7']

# 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']=[]
```

Task 3 - Continue

```
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)
```

Task 3 - Continue

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

# Booster version
# TODO: Append the bv into launch_dict with key `Version Booster`
bv=booster_version(row[1])
if not(bv):
    bv=row[1].a.string
    launch_dict['Version Booster'].append(bv)
print(bv)

# Launch Site
# TODO: Append the bv into launch_dict with key `Launch Site`
launch_site = row[2].a.string
launch_dict['Launch site'].append(launch_site)
#print(launch_site)

# Payload
# TODO: Append the payload into launch_dict with key `Payload`
payload = row[3].a.string
launch_dict['Payload'].append(payload)
#print(payload)

# Payload Mass
# TODO: Append the payload_mass into launch_dict with key `Payload mass`
payload_mass = get_mass(row[4])
launch_dict['Payload mass'].append(payload_mass)
#print(payload)
```

Task 3 - Continue

```
            # Orbit
            # TODO: Append the orbit into launch_dict with key `Orbit`
            orbit = row[5].a.string
            launch_dict['Orbit'].append(orbit)
            #print(orbit)

            # Customer
            # TODO: Append the customer into launch_dict with key `Customer`

            if row[6].a is not None:
                customer = row[6].a.string
                launch_dict['Customer'].append(customer)
            #print(customer)

            # Launch outcome
            # TODO: Append the launch_outcome into launch_dict with key `Launch_outcome`
            launch_outcome = list(row[7].strings)[0]
            launch_dict['Launch outcome'].append(launch_outcome)
            #print(launch_outcome)

            # Booster landing
            # TODO: Append the launch_outcome into launch_dict with key `Booster_landing`
            booster_landing = landing_status(row[8])
            launch_dict['Booster landing'].append(booster_landing)
            #print(booster_landing)
```

```
F9 v1.0B0003.1
F9 v1.0B0004.1
F9 v1.0B0005.1
F9 v1.0B0006.1
F9 v1.0B0007.1
F9 v1.1B1003
F9 v1.1
F9 v1.1
F9 v1.1
F9 v1.1
F9 v1.1
F9 v1.1
F9 v1.1
```

RESULTS

DataCollection and Wrangling

3- Data Wrangling

Task 1

TASK 1: Calculate the number of launches on each site

The data contains several Space X launch facilities: Cape Canaveral Space Launch Complex 40 **VAFB SLC 4E** , Vandenberg Air Force Base Space Launch Complex 4E **(SLC-4E)**, Kennedy Space Center Launch Complex 39A **KSC LC 39A** .The location of each Launch Is placed in the column `LaunchSite`

Next, let's see the number of launches for each site.

Use the method `value_counts()` on the column `LaunchSite` to determine the number of launches on each site:

In [6]:
```
# Apply value_counts() on column LaunchSite
LaunchSite = df['LaunchSite']
LaunchSite.value_counts()
```

Out[6]:
```
CCAFS SLC 40    55
KSC LC 39A      22
VAFB SLC 4E     13
Name: LaunchSite, dtype: int64
```

Task 2

TASK 2: Calculate the number and occurrence of each orbit

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

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

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

Task 3

TASK 3: Calculate the number and occurence of mission outcome of the orbits

Use the method `.value_counts()` on the column `Outcome` to determine the number of `landing_outcomes` .Then assign it to a variable landing_outcomes.

In [11]:
```
# landing_outcomes = values on Outcome column
s = df['Outcome']
landing_outcomes = s.value_counts()
```

`True Ocean` means the mission outcome was successfully landed to a specific region of the ocean while `False Ocean` means the mission outcome was unsuccessfully landed to a specific region of the ocean. `True RTLS` means the mission outcome was successfully landed to a ground pad `False RTLS` means the mission outcome was unsuccessfully landed to a ground pad. `True ASDS` means the mission outcome was successfully landed to a drone ship `False ASDS` means the mission outcome was unsuccessfully landed to a drone ship. `None ASDS` and `None None` these represent a failure to land.

In [12]:
```
for i,outcome in enumerate(landing_outcomes.keys()):
    print(i,outcome)
```

```
0 True ASDS
1 None None
2 True RTLS
3 False ASDS
4 True Ocean
5 False Ocean
6 None ASDS
7 False RTLS
```

We create a set of outcomes where the second stage did not land successfully:

In [13]:
```
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
bad_outcomes
```

Out[13]:
```
{'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

Task 4

TASK 4: Create a landing outcome label from Outcome column

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

In [17]:
```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
landing_class = []
for i,out in enumerate(pd.Series(df['Outcome'])):
    if out in bad_outcomes:
        landing_class.append(0)
    else:
        landing_class.append(1)

landing_class
```

Out[17]:
```
[0,
 0,
 0,
 0,
 0,
 0,
 1,
 1,
 0,
 0
```

RESULTS

Exploratory Analysis Using SQL

1- EDA with SQL

Task 1

Task 1

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

[8]:
```
%sql select distinct Launch_Site from SPACEXTBL
```

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

[8]:

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

Task 2

Task 2

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

In [9]:
```
%sql select * from SPACEXTBL where Launch_Site like "CCA%" limit 5
#%sql select * from SPACEXTBL
```

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

Out[9]:

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

Task 3&4

Task 3

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

```
In [10]: %sql select sum(PAYLOAD_MASS__KG_) from SPACEXTBL where customer = 'NASA (CRS)'

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

Out[10]:

sum(PAYLOAD_MASS__KG_)
45596

Task 4

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

```
In [11]: %sql select sum(PAYLOAD_MASS__KG_) from SPACEXTBL where Booster_Version like 'F9 v1.1%'

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

Out[11]:

sum(PAYLOAD_MASS__KG_)
38020

Task 5

Task 5

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

Hint:Use min function

In [12]:
```
# Success (ground pad)
%sql select min(Date) from SPACEXTBL where Landing_Outcome = 'Success (ground pad)'
```

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

Out[12]:

min(Date)
2015-12-22

Task 6

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 [13]:
```
%sql select Booster_Version,Landing_Outcome from SPACEXTBL where Landing_Outcome = 'Success (drone ship)' and PAYLOAD_MASS__KG_>=
```

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

Out[13]:

Booster_Version	Landing_Outcome
F9 FT B1022	Success (drone ship)
F9 FT B1026	Success (drone ship)
F9 FT B1021.2	Success (drone ship)
F9 FT B1031.2	Success (drone ship)

Task 7

Task 7

List the total number of successful and failure mission outcomes

In [14]:
```
%sql select Landing_Outcome,count(Landing_Outcome) from SPACEXTBL group by Landing_Outcome
#%sql select count(*) from SPACEXTBL
#%sql select sum(total) from (select Landing_Outcome,count(Landing_Outcome) as total from SPACEXTBL group by Landing_Outcome) whe
```

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

Out[14]:

Landing_Outcome	count(Landing_Outcome)
Controlled (ocean)	5
Failure	3
Failure (drone ship)	5
Failure (parachute)	2
No attempt	21
No attempt	1
Precluded (drone ship)	1
Success	38
Success (drone ship)	14
Success (ground pad)	9
Uncontrolled (ocean)	2

Task 8

Task 8

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

In [21]:

```
#%sql select Booster_Version from SPACEXTBL where Booster_Version = (select Booster_Version,max(PAYLOAD_MASS__KG_) from SPACEXTBL
%sql select distinct Booster_Version, PAYLOAD_MASS__KG_ from SPACEXTBL where PAYLOAD_MASS__KG_ = (select max(PAYLOAD_MASS__KG_) f
# %sql select Booster_Version, count(Booster_Version) from SPACEXTBL group by Booster_Version
```

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

Out[21]:

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

Task 9

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.

In [26]:
```
%sql select substr(Date, 6,2) as Month,Landing_Outcome,Booster_Version,Launch_Site from SPACEXTBL where substr(Date,0,5)='2015' a
```

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

Out[26]:

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

Task 10

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 [29]:
```
%sql select Landing_Outcome,Date,count(Landing_Outcome) from SPACEXTBL group by Landing_Outcome having Date >= '2010-06-04' and D
```

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

Out[29]:

Landing_Outcome	Date	count(Landing_Outcome)
Success (drone ship)	2016-04-08	14
Success (ground pad)	2015-12-22	9
Precluded (drone ship)	2015-06-28	1
Failure (drone ship)	2015-01-10	5
Controlled (ocean)	2014-04-18	5
Uncontrolled (ocean)	2013-09-29	2
No attempt	2012-05-22	21
Failure (parachute)	2010-06-04	2

RESULTS

Exploratory Analysis Using SQL

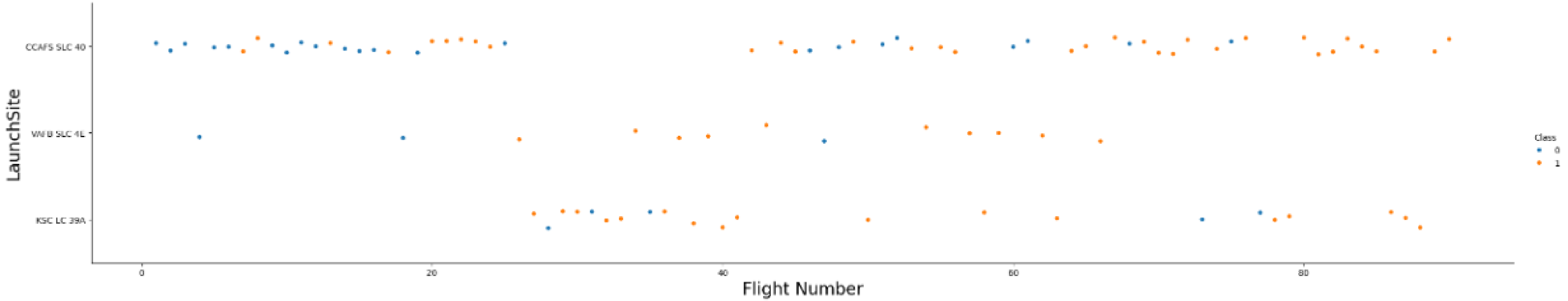
2- EDA with Visualization

Task 1

TASK 1: Visualize the relationship between Flight Number and Launch Site

Use the function `catplot` to plot `FlightNumber` vs `LaunchSite`, set the parameter `x` parameter to `FlightNumber`,set the `y` to `Launch Site` and set the parameter `hue` to `'class'`

In [6]:
```
# 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()
```

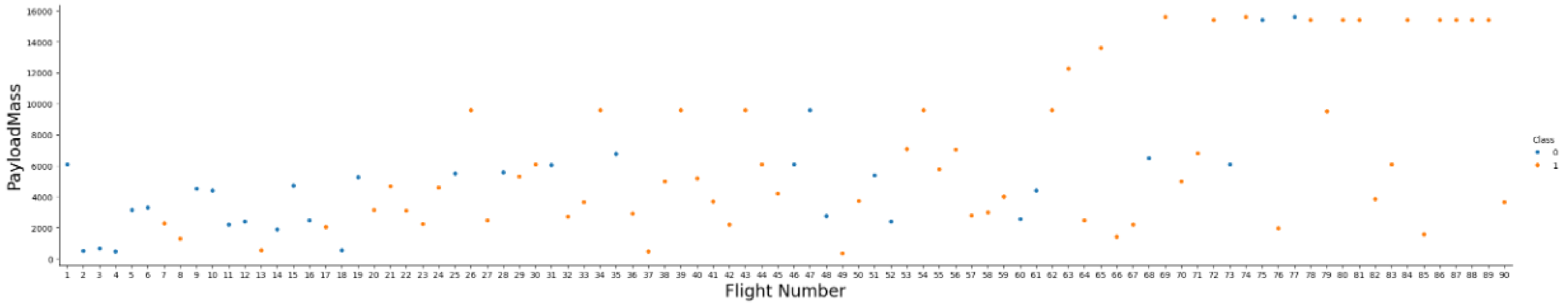


Task 2 - Part 1

TASK 2: Visualize the relationship between Payload and Launch Site

We also want to observe if there is any relationship between launch sites and their payload mass.

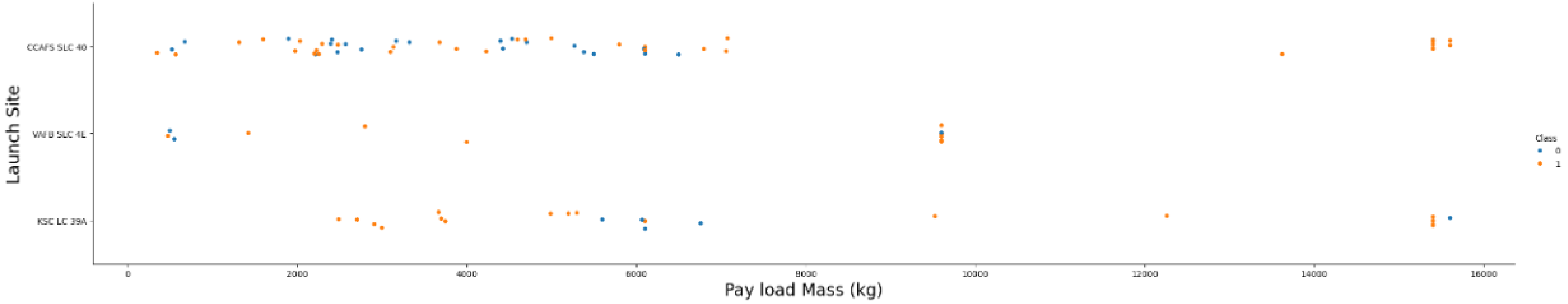
In [7]:
```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch site, and hue to be the class value
sns.catplot(y="PayloadMass", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("PayloadMass",fontsize=20)
plt.show()
```



Task 2 – Part 2

Now if you observe Payload Vs. Launch Site scatter point chart you will find for the VAFB-SLC launchsite there are no rockets launched for heavypayload mass(greater than 10000).

In [8]:
```
sns.catplot(y="LaunchSite", x="PayloadMass", hue="Class", data=df, aspect = 5)
plt.xlabel("Pay load Mass (kg)",fontsize=20)
plt.ylabel("Launch Site",fontsize=20)
plt.show()
```



Task 3

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

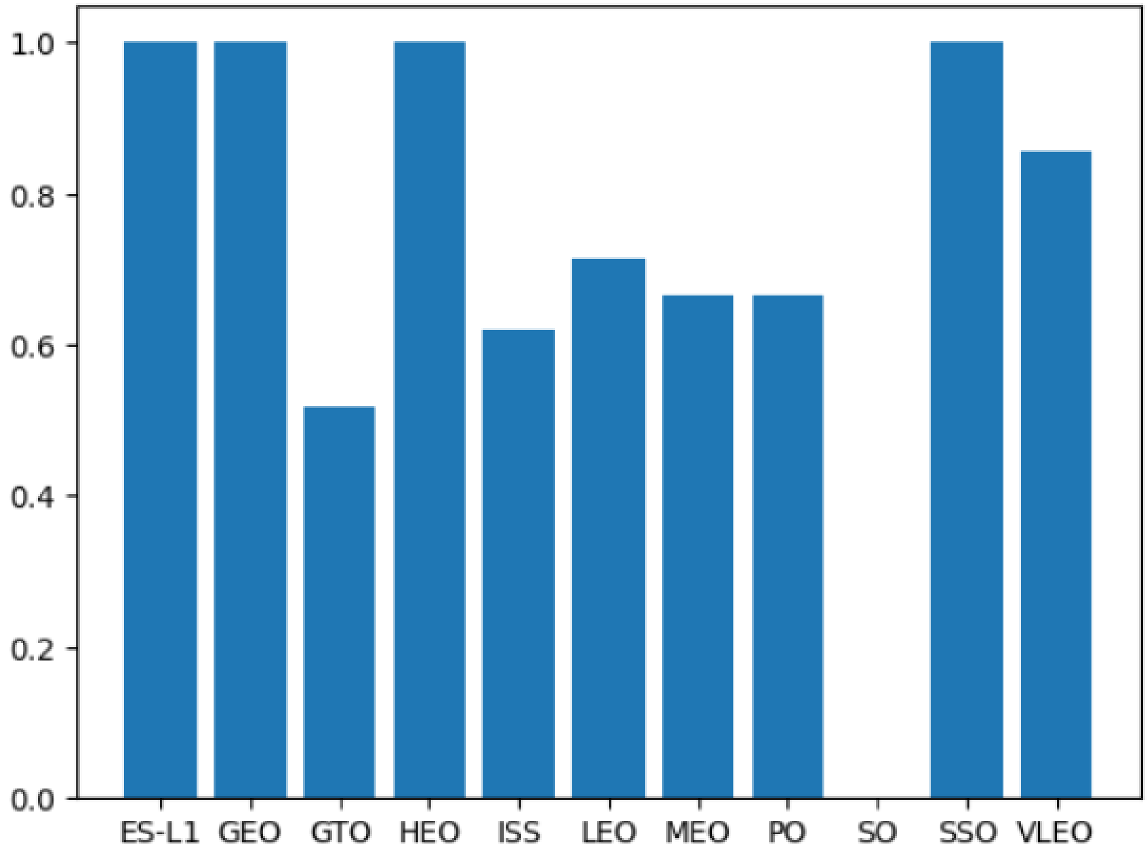
Next, we want to visually check if there are any relationship between success rate and orbit type.

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

In [9]:
```
# HINT use groupby method on Orbit column and get the mean of Class column

a = df[['Orbit','Class']].groupby(['Orbit']).mean()
a = a.reset_index()
plt.bar(a['Orbit'],a['Class'])
plt.show()
```

Task 3 - Continue

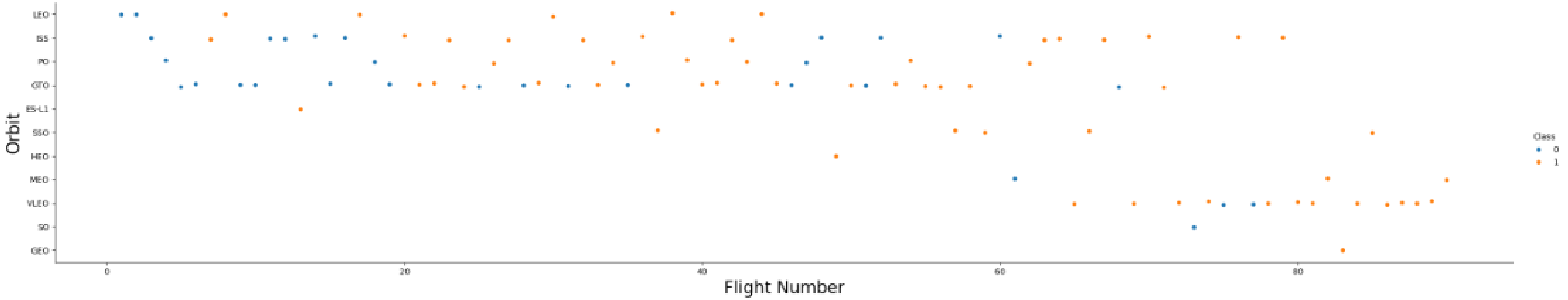


Task 4

TASK 4: Visualize the relationship between FlightNumber and Orbit type

For each orbit, we want to see if there is any relationship between FlightNumber and Orbit type.

In [10]:
```
# 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("Flight Number",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```

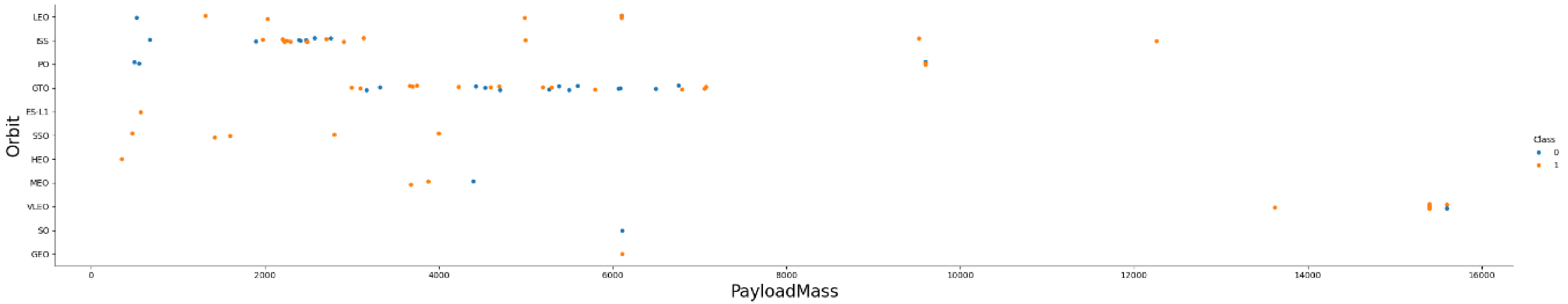


Task 5

TASK 5: Visualize the relationship between Payload and Orbit type

Similarly, we can plot the Payload vs. Orbit scatter point charts to reveal the relationship between Payload and Orbit type

In [12]:
```
# 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("PayloadMass",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```



IBM Developer

SKILLS NETWORK

Task 6

TASK 6: Visualize the launch success yearly trend

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

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

In [13]:
```
# A function to Extract years from the date
year=[]
def Extract_year():
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year
Extract_year()
df['Date'] = year
df.head()
```

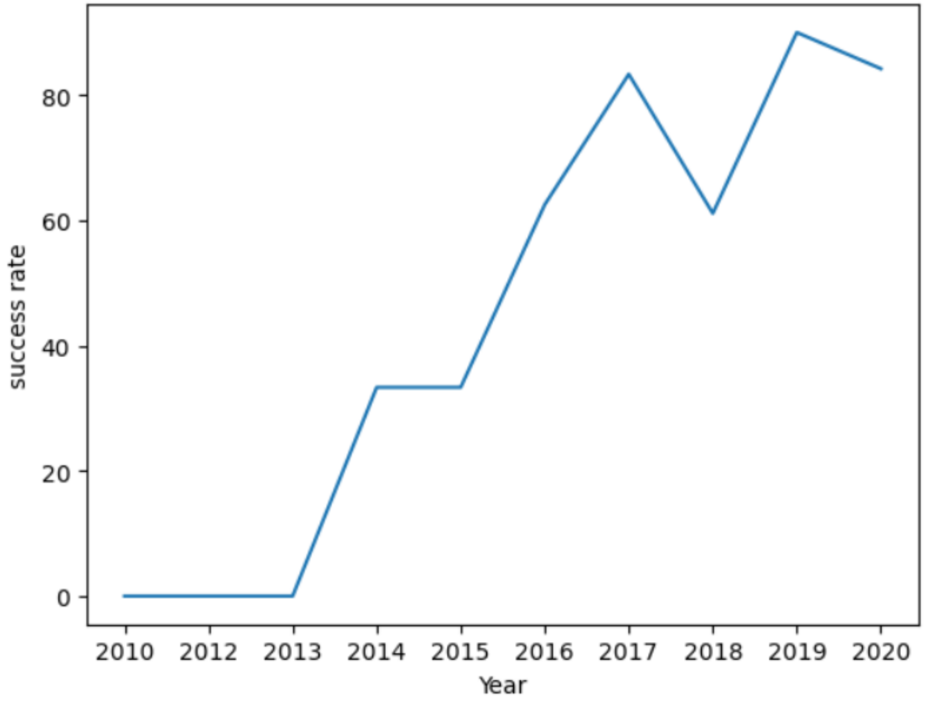
Out[13]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial
0	1	2010	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003
1	2	2012	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005
2	3	2013	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007
3	4	2013	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	0	B1003
4	5	2013	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004

Task 6 - Continue

In [19]:
```
# Plot a line chart with x axis to be the extracted year and y axis to be the success rate
b =  df[['Date','Class']].groupby(['Date']).mean()
b = b.reset_index()
plt.plot(b['Date'],b['Class']*100)
plt.xlabel('Year')
plt.ylabel('success rate')
```

Out[19]: `Text(0, 0.5, 'success rate')`



Task 7

TASK 7: Create dummy variables to categorical columns

Use the function `get_dummies` and `features` dataframe to apply OneHotEncoder to the column `Orbits`, `LaunchSite`, `LandingPad`, and `Serial`. Assign the value to the variable `features_one_hot`, display the results using the method head. Your result dataframe must include all features including the encoded ones.

In [26]:
```python
# HINT: Use get_dummies() function on the categorical columns
features_one_hot = pd.get_dummies(features[['Orbit','LaunchSite','LandingPad','Serial']])
features_one_hot.head()
```

Out[26]:

	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	Orbit_LEO	Orbit_MEO	Orbit_PO	Orbit_SO	Orbit_SSO	...	Serial_B1048	Serial_B1049	Serial_B105
0	0	0	0	0	0	1	0	0	0	0	...	0	0	
1	0	0	0	0	0	1	0	0	0	0	...	0	0	
2	0	0	0	0	1	0	0	0	0	0	...	0	0	
3	0	0	0	0	0	0	0	1	0	0	...	0	0	
4	0	0	1	0	0	0	0	0	0	0	...	0	0	

5 rows × 72 columns

Task 8

TASK 8: Cast all numeric columns to `float64`

Now that our `features_one_hot` dataframe only contains numbers cast the entire dataframe to variable type `float64`

In [31]:
```
# HINT: use astype function
features_one_hot = features_one_hot.astype('float64')
features_one_hot.dtypes
```

Out[31]:
```
Orbit_ES-L1     float64
Orbit_GEO       float64
Orbit_GTO       float64
Orbit_HEO       float64
Orbit_ISS       float64
                 ...
Serial_B1056    float64
Serial_B1058    float64
Serial_B1059    float64
Serial_B1060    float64
Serial_B1062    float64
Length: 72, dtype: object
```

RESULTS

Interactive Visual Analytics and Dashboard

1- Interactive Map with Folium

Task 1

TODO: Create and add `folium.Circle` and `folium.Marker` for each launch site on the site map

An example of folium.Circle:

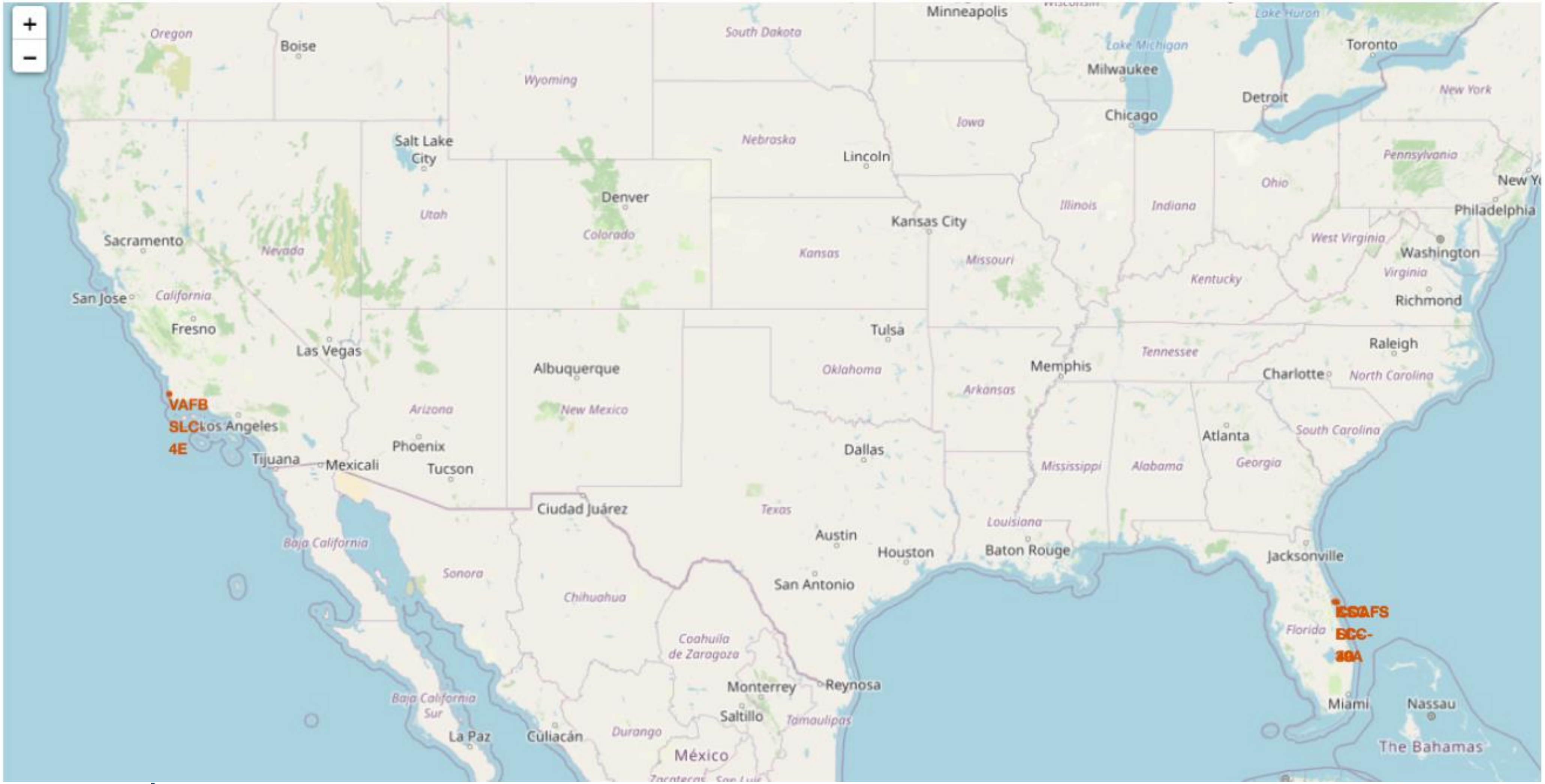
`folium.Circle(coordinate, radius=1000, color='#000000', fill=True).add_child(folium.Popup(...))`

An example of folium.Marker:

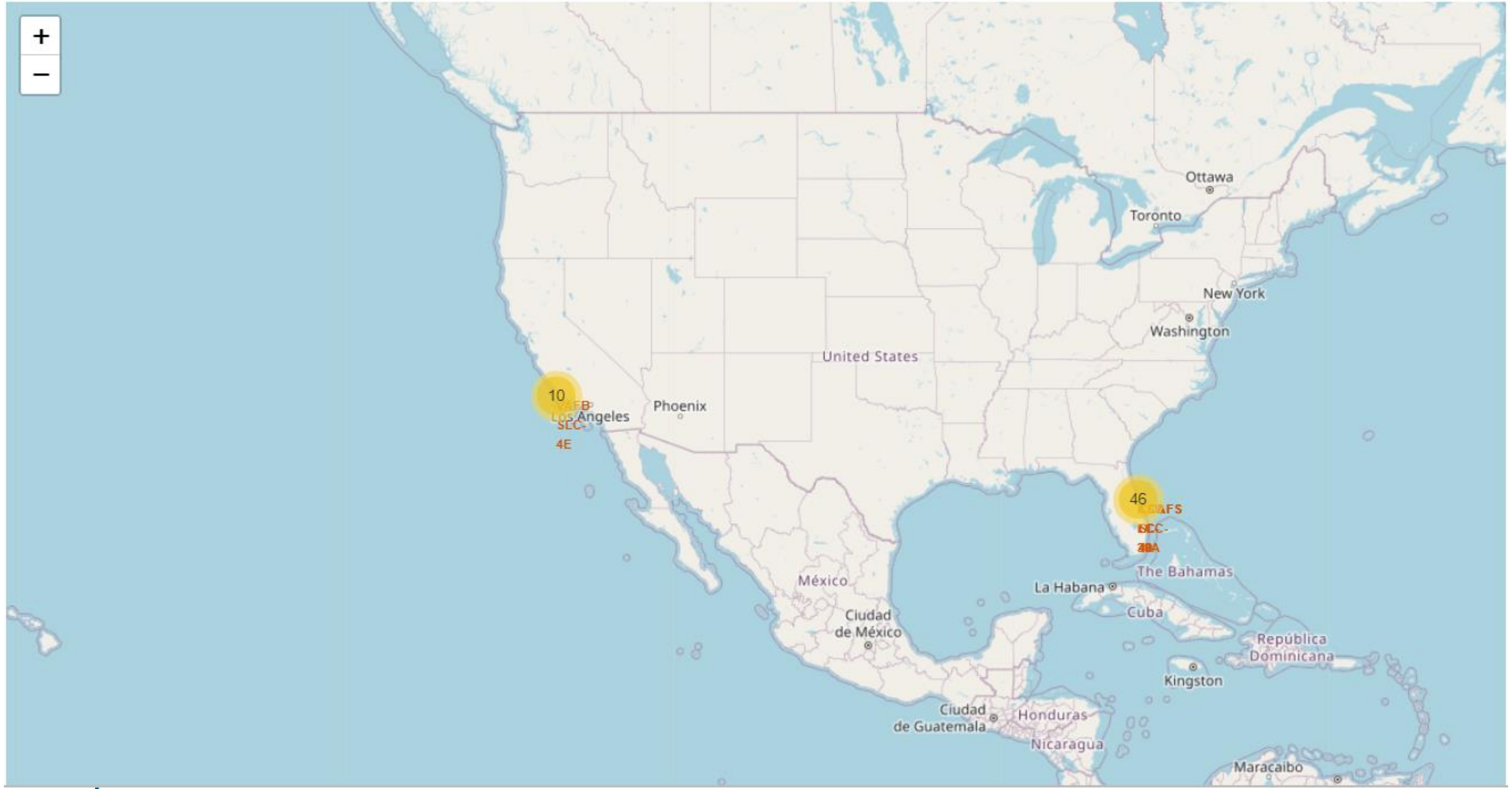
`folium.map.Marker(coordinate, icon=DivIcon(icon_size=(20,20),icon_anchor=(0,0), html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % 'label', ))`

```
# Initial the map
site_map = folium.Map(location=nasa_coordinate, zoom_start=5)
# For each launch site, add a Circle object based on its coordinate (Lat, Long) values. In addition, add Launch site name as a popup label
circle = folium.Circle(nasa_coordinate, radius=1000, color='red', fill=True).add_child(folium.Popup('NASA Johnson Space Center'))
for i in range(4):
    cord = launch_sites_df.loc[i][['Lat','Long']]
    site_name = launch_sites_df.loc[i][['Launch Site']]
    # print(site_name)
    circle = folium.Circle(cord, radius=50, color='yellow', fill=True).add_child(folium.Popup(launch_sites_df.loc[i][['Lat','Long']]))
    marker = folium.map.Marker(
        cord,
        # Create an icon as a text label
        icon=DivIcon(
            icon_size=(20,20),
            icon_anchor=(0,0),
            html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' %site_name[0] ,
            )
        )
    site_map.add_child(circle)
    site_map.add_child(marker)
site_map
```

Task 1 - Continue

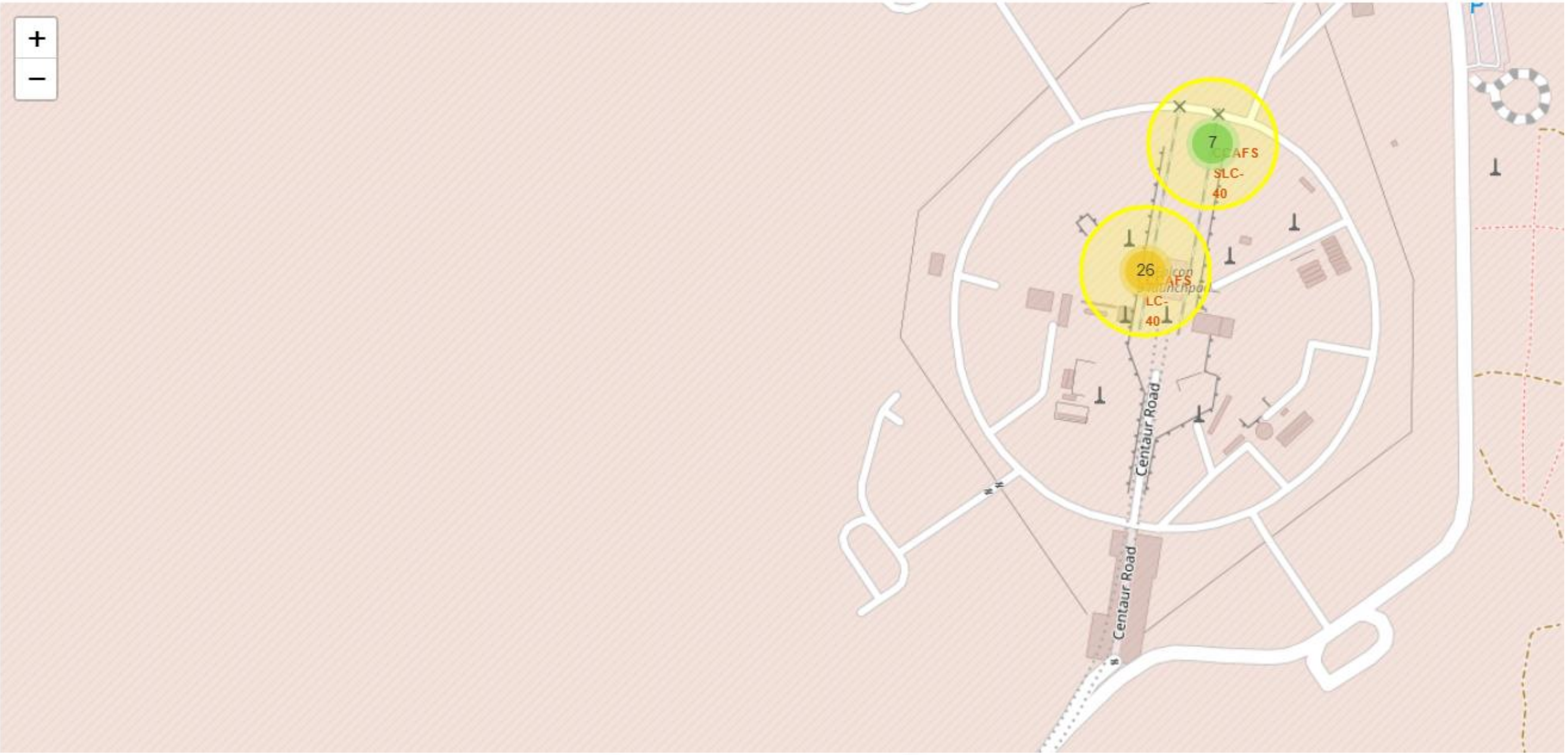


Task 2

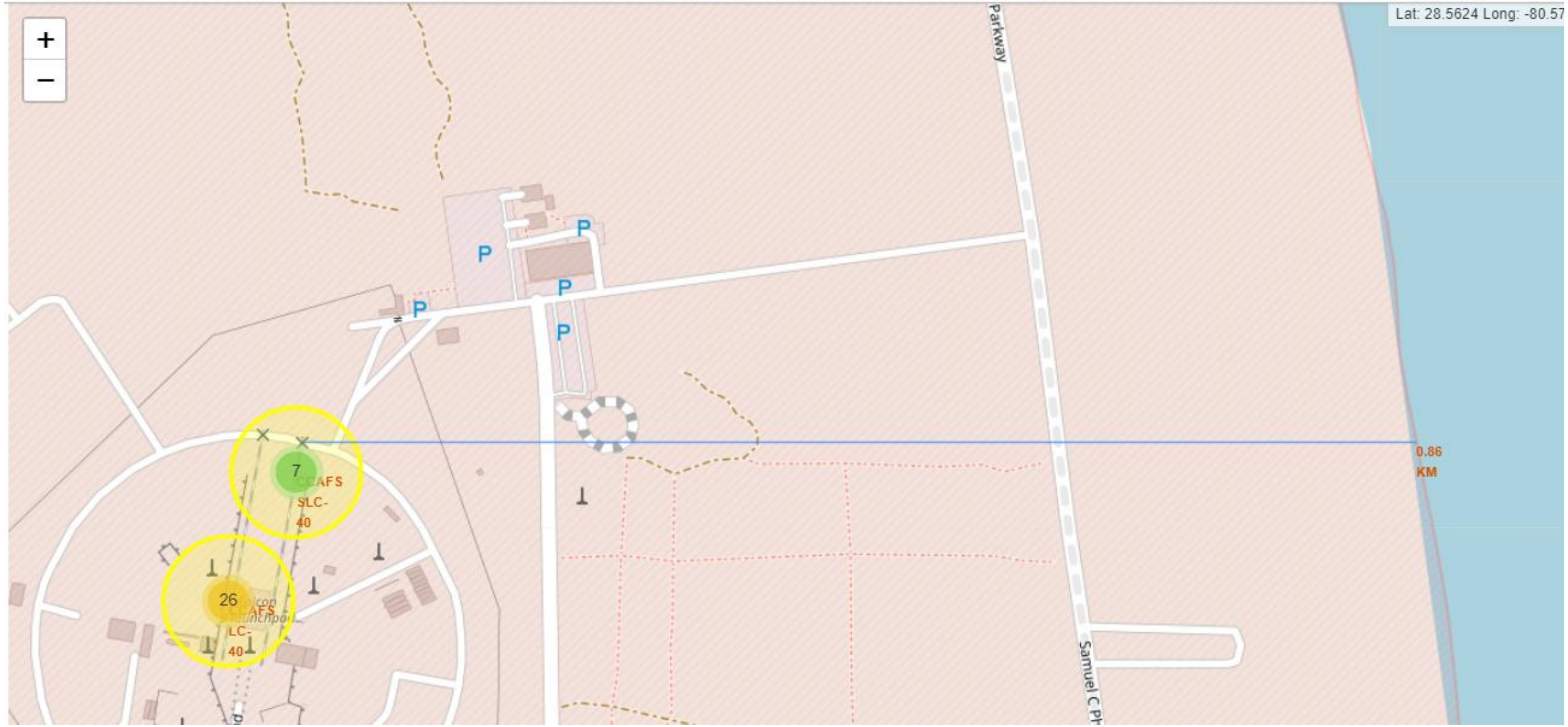


+
−
Ottawa
Toronto
New York
Washington
United States
Phoenix
10
Los Angeles
46
The Bahamas
La Habana
Cuba
México
Ciudad de México
República Dominicana
Kingston
Ciudad de Guatemala
Honduras
Nicaragua
Maracaibo

Task 2 - Continue



Task 3



IBM Developer

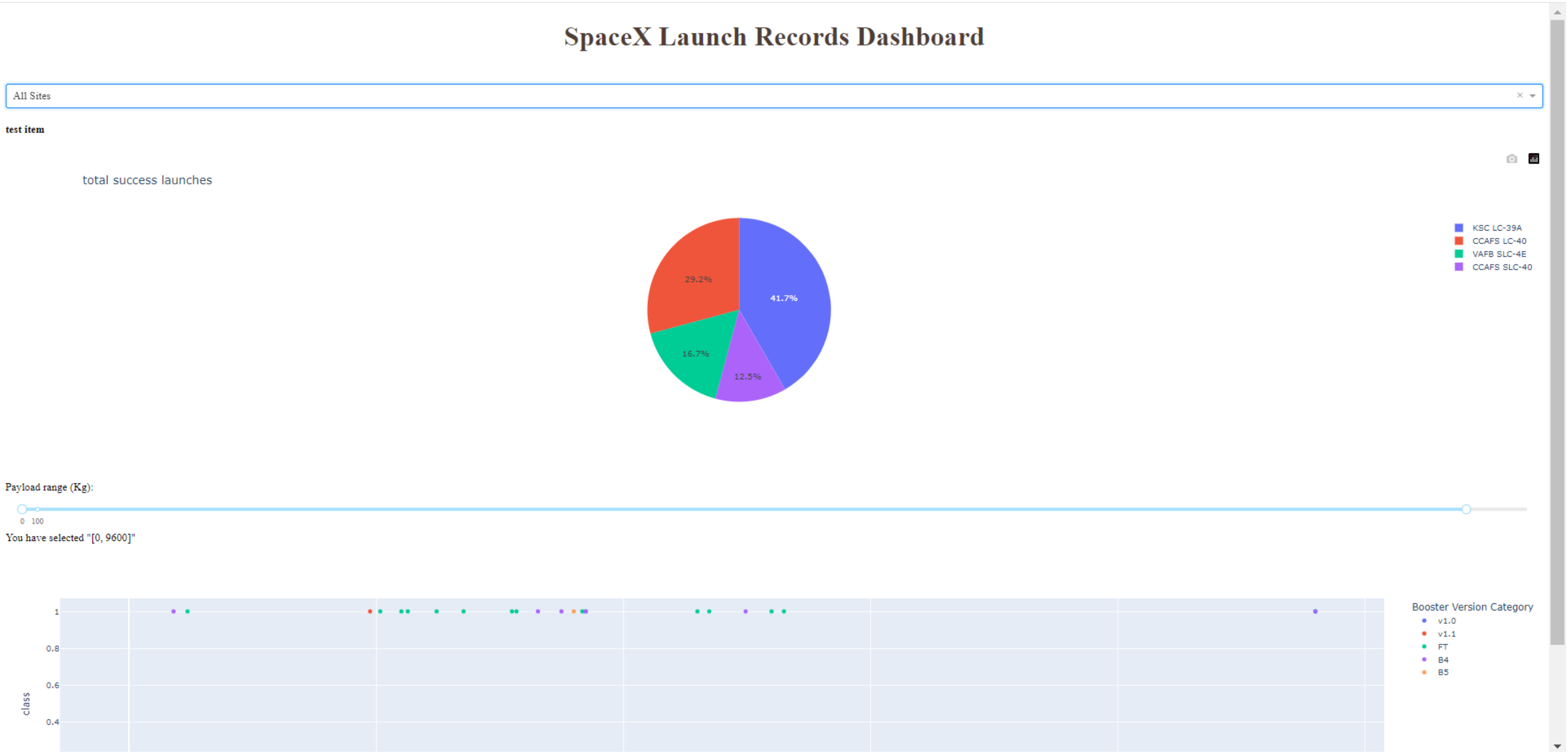
SKILLS NETWORK

RESULTS

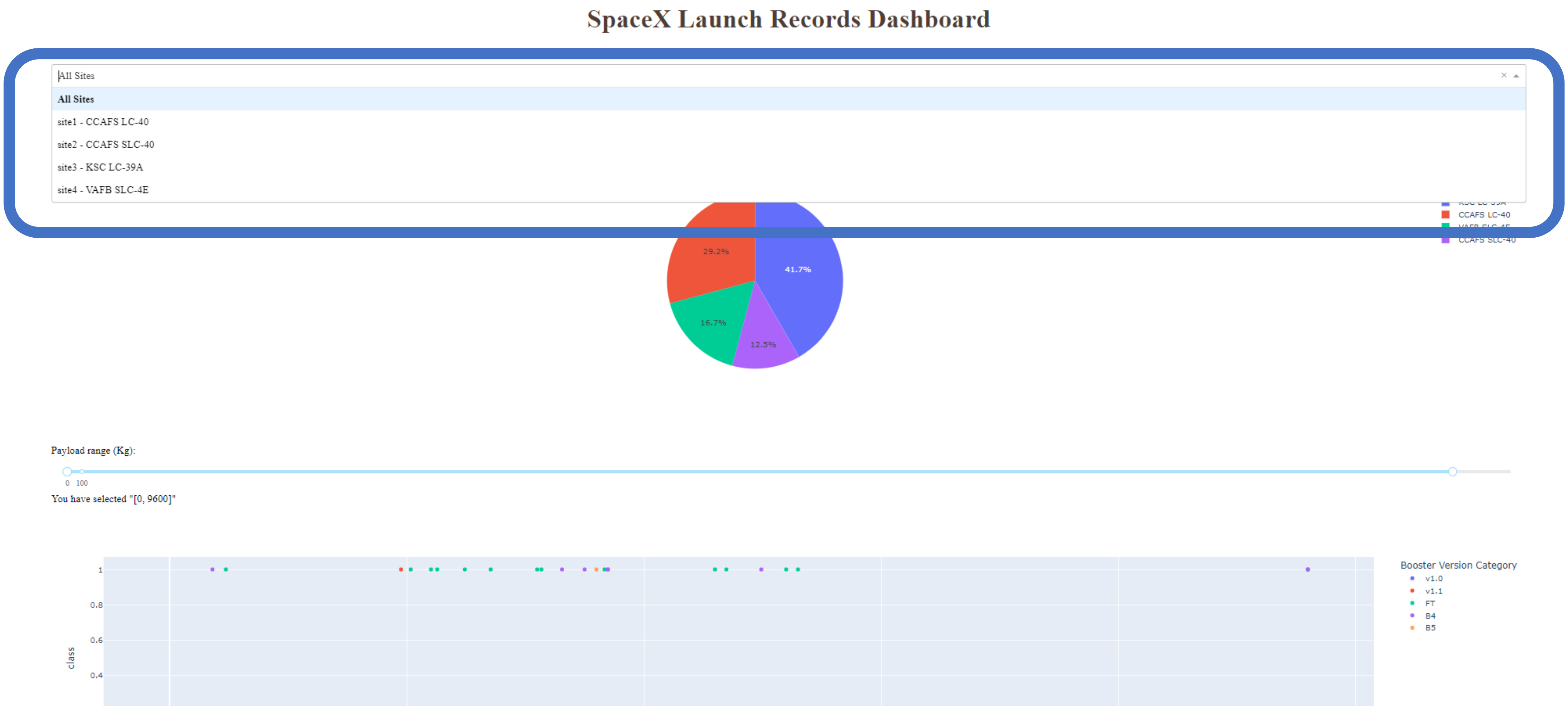
Interactive Visual Analytics and Dashboard

2- Interactive Dashboard with Ploty Dash

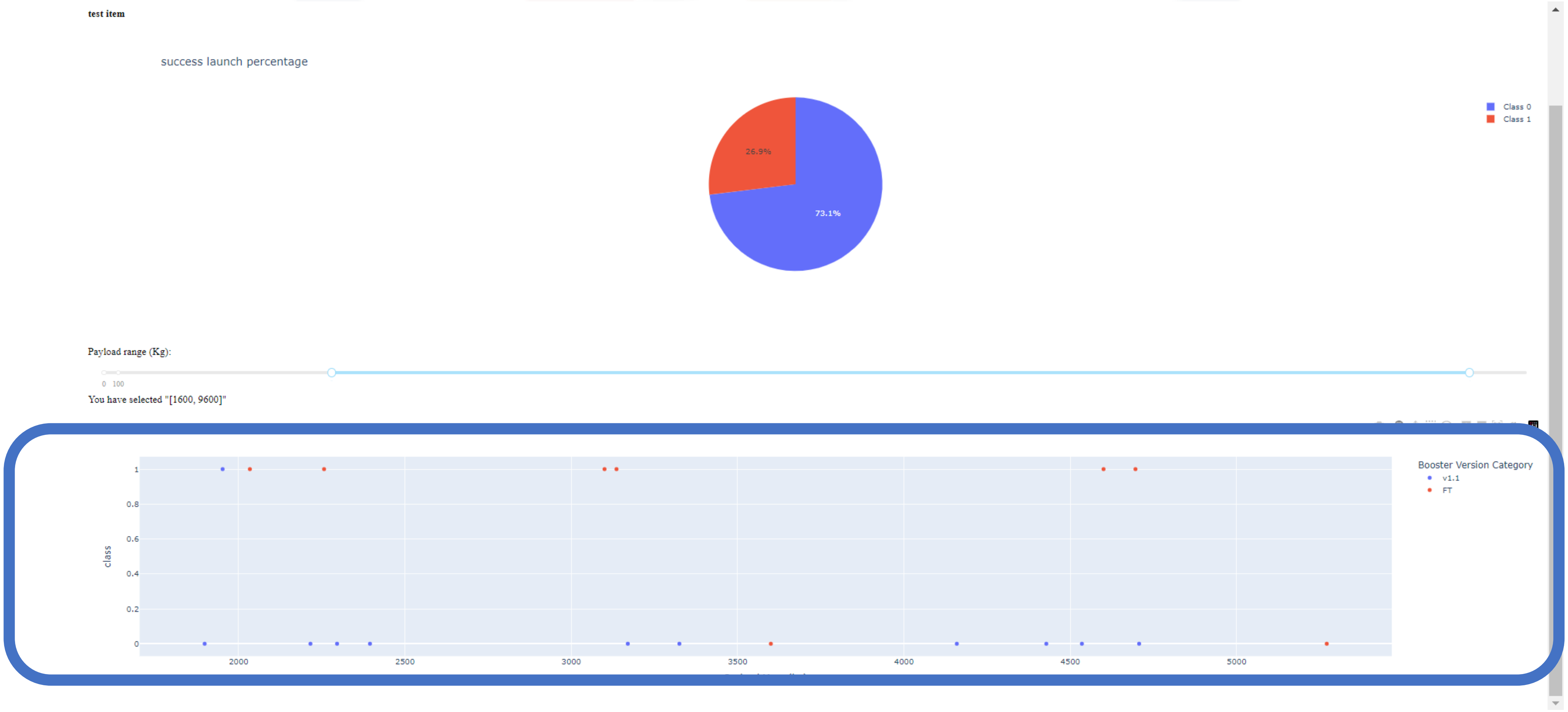
Complete Page – Select All



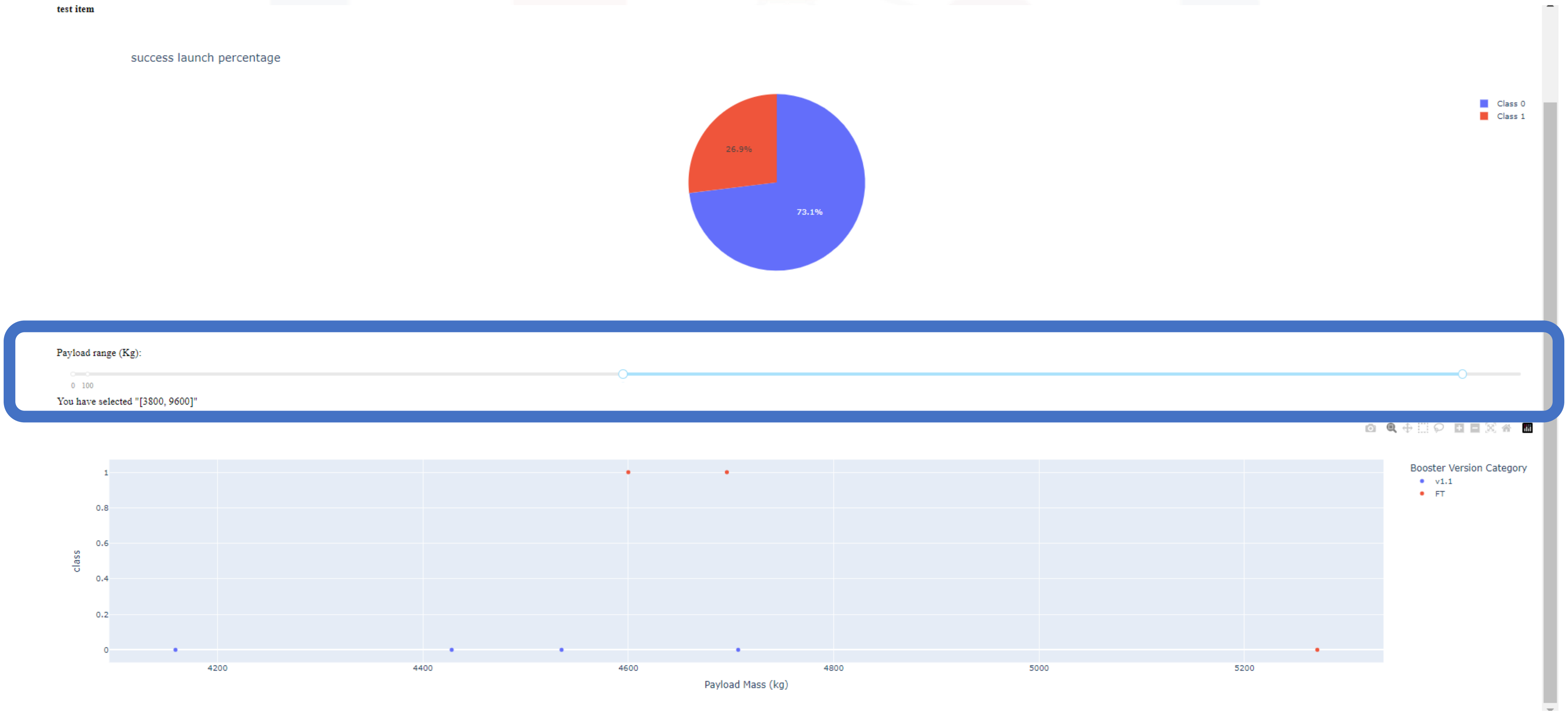
Dropdown Menu



Scatter

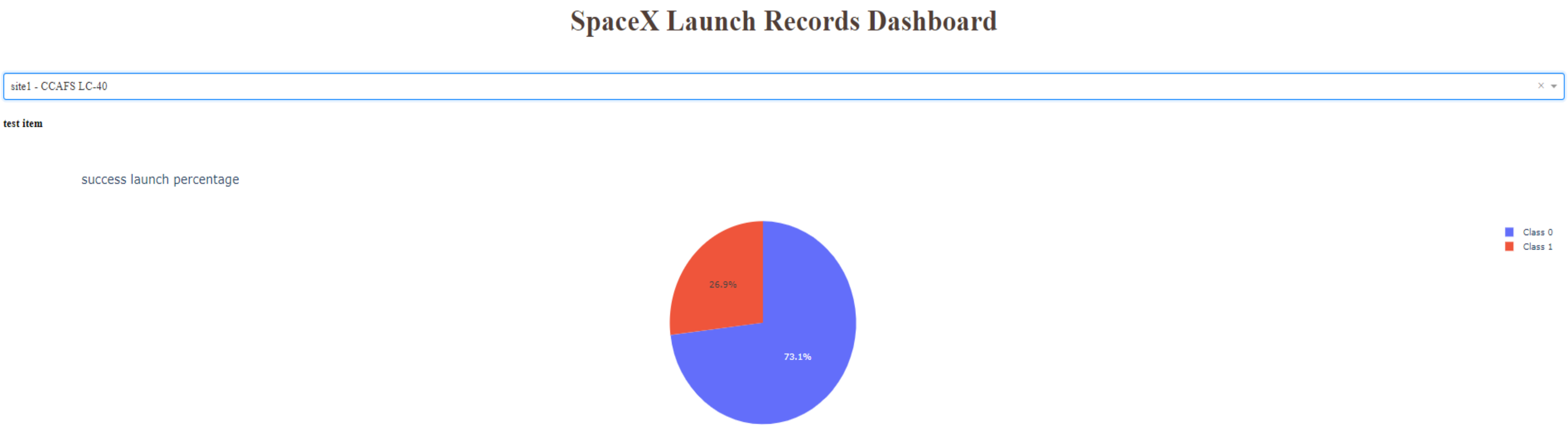

test item
success launch percentage
26.9%
73.1%
Class 0
Class 1
Payload range (Kg):
0 100
You have selected "[1600, 9600]"
class
1
0.8
0.6
0.4
0.2
0
2000
2500
3000
3500
4000
4500
5000
Booster Version Category
v1.1
FT

Slider



test item
success launch percentage
26.9%
73.1%
Class 0
Class 1
Payload range (Kg):
0
100
You have selected "[3800, 9600]"
Booster Version Category
v1.1
FT
class
Payload Mass (kg)

Specific Selection



RESULTS

Predictive Analysis

Prediction with Machine Learning

Task 1

TASK 1

Create a NumPy array from the column `Class` in `data`, by applying the method `to_numpy()` then assign it to the variable `Y`,make sure the output is a Pandas series (only one bracket df['name of column']).

[8]:
```
y=data['Class'].to_numpy()
y
```

[8]:
```
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], dtype=int64)
```

Task 2

TASK 2

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

[9]:
```
# students get this
transform = preprocessing.StandardScaler()
X = transform.fit_transform(X)
X
```

[9]:
```
array([[-1.71291154e+00, -1.94814463e-16, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       [-1.67441914e+00, -1.19523159e+00, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       [-1.63592675e+00, -1.16267307e+00, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       ...,
       [ 1.63592675e+00,  1.99100483e+00,  3.49060516e+00, ...,
         1.19684269e+00, -5.17306132e-01,  5.17306132e-01],
       [ 1.67441914e+00,  1.99100483e+00,  1.00389436e+00, ...,
         1.19684269e+00, -5.17306132e-01,  5.17306132e-01],
       [ 1.71291154e+00, -5.19213966e-01, -6.53912840e-01, ...,
        -8.35531692e-01, -5.17306132e-01,  5.17306132e-01]])
```

Task 3

TASK 3

Use the function train_test_split to split the data X and Y into training and test data. Set the parameter test_size to 0.2 and random_state to 2. The training data and test data should be assigned to the following labels.

`X_train, X_test, Y_train, Y_test`

```
[12]: X_train, X_test, Y_train, Y_test = train_test_split( X, y, test_size=0.2, random_state=2)
```

we can see we only have 18 test samples.

```
[13]: Y_test.shape
```

```
[13]: (18,)
```

Task 4

TASK 4

Create a logistic regression object then create a GridSearchCV object `logreg_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

[44]:
```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

[45]:
```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()
logreg_cv = GridSearchCV(lr, parameters,cv=10)
logreg_cv.fit(X_train,Y_train)
```

[45]:
```
▸ GridSearchCV
▸ estimator: LogisticRegression
  ▸ LogisticRegression
```

We output the `GridSearchCV` object for logistic regression. We display the best parameters using the data attribute `best_params_` and the accuracy on the validation data using the data attribute `best_score_`.

[46]:
```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

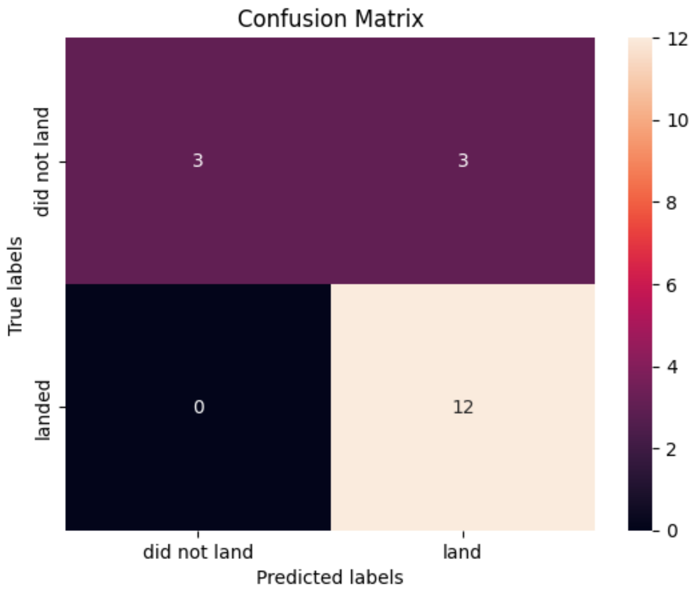
```
tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8464285714285713
```

Task 5

TASK 5

Lets look at the confusion matrix:

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



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

```
from sklearn.metrics import f1_score
f1_1 = f1_score(y_test, yhat, average='weighted')
f1_1
```

```
0.8148148148148149
```

Task 6

TASK 6

Create a support vector machine object then create a `GridSearchCV` object `svm_cv` with cv - 10. Fit the object to find the best parameters from the dictionary `parameters`.

[49]:
```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

[50]:
```
svm_cv = GridSearchCV(svm, parameters, cv  = 10)
svm_cv.fit(X_train,Y_train)
```

[50]:
```
▸ GridSearchCV
▸ estimator: SVC
    ▸ SVC
```

[51]:
```
print("tuned hpyerparameters :(best parameters) ",svm_cv.best_params_)
print("accuracy :",svm_cv.best_score_)
```

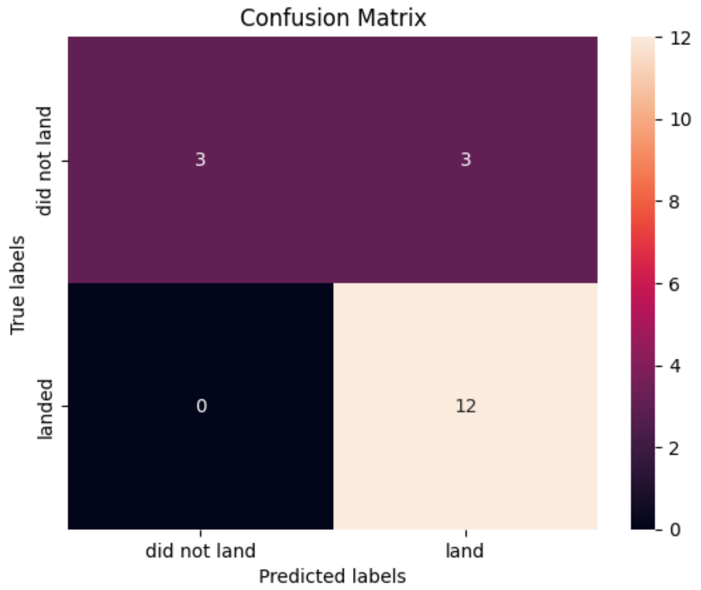
```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
accuracy : 0.8482142857142856
```

Task 7

TASK 7

We can plot the confusion matrix

[52]:
```
yhat=svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



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

[53]:
```
from sklearn.metrics import f1_score
f1_2 = f1_score(y_test, yhat, average='weighted')
f1_2
```

[53]: `0.8148148148148149`

Task 8

TASK 8

Create a decision tree classifier object then create a `GridSearchCV` object `tree_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

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

      tree = DecisionTreeClassifier()
```

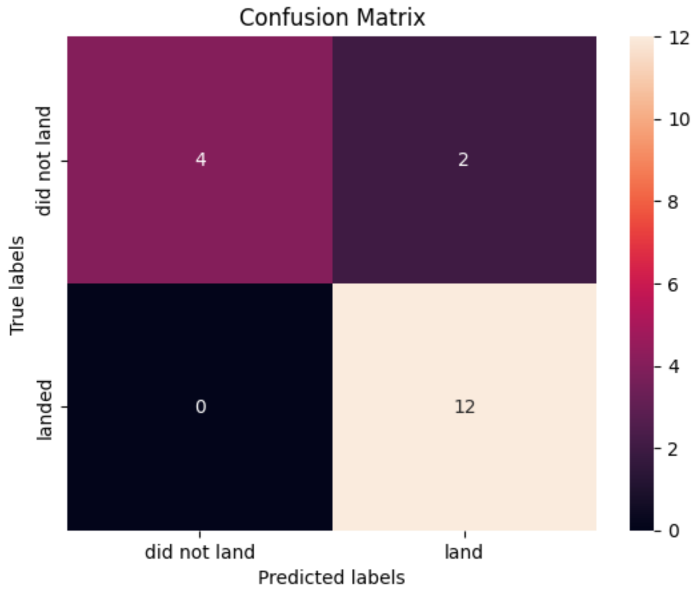
```
[55]: tree_cv = GridSearchCV(tree, parameters, cv  = 10)
      tree_cv.fit(X_train,Y_train)
```

Task 9

TASK 9

We can plot the confusion matrix

[57]:
```
yhat = tree_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



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

[58]:
```
from sklearn.metrics import f1_score
f1_3 = f1_score(y_test, yhat, average='weighted')
f1_3
```

[58]: 0.882051282051282

Task 10

TASK 10

Create a k nearest neighbors object then create a `GridSearchCV` object `knn_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

[59]:
```
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()
```

[60]:
```
knn_cv = GridSearchCV(KNN, parameters, cv  = 10)
knn_cv.fit(X_train,Y_train)
```

[60]:
```
▸ GridSearchCV
▸ estimator: KNeighborsClassifier
    ▸ KNeighborsClassifier
```

[61]:
```
print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
print("accuracy :",knn_cv.best_score_)
```

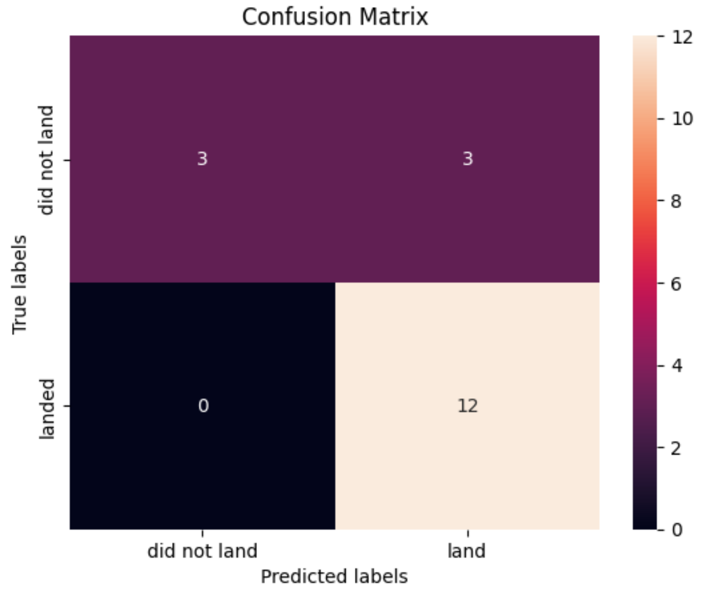
```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}
accuracy : 0.8482142857142858
```

Task 11

TASK 11

We can plot the confusion matrix

[62]:
```
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



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

[64]:
```
from sklearn.metrics import f1_score
f1_4 = f1_score(y_test, yhat, average='weighted')
f1_4
```

[64]: 0.8148148148148149

Task 12

TASK 12

Find the method performs best:

[74]:
```
meth = ['logreg','SVM','Decesion Tree','KNN']
meth_score = [f1_1,f1_2,f1_3,f1_4]
max_index = meth1.index(max(meth1))
print("the best method answer is: ",meth[max_index], "with F1 score: ",meth_score[max_index] )
```

```
the best method answer is:  Decesion Tree with F1 score:  0.882051282051282
```

CONCLUSION

- The best method for prediction is decision tree.
- It can be said with %88.2 accuracy that first stage will be usable
- The best orbits for using satellites would be ES-L1, GEO,HEO and SSO