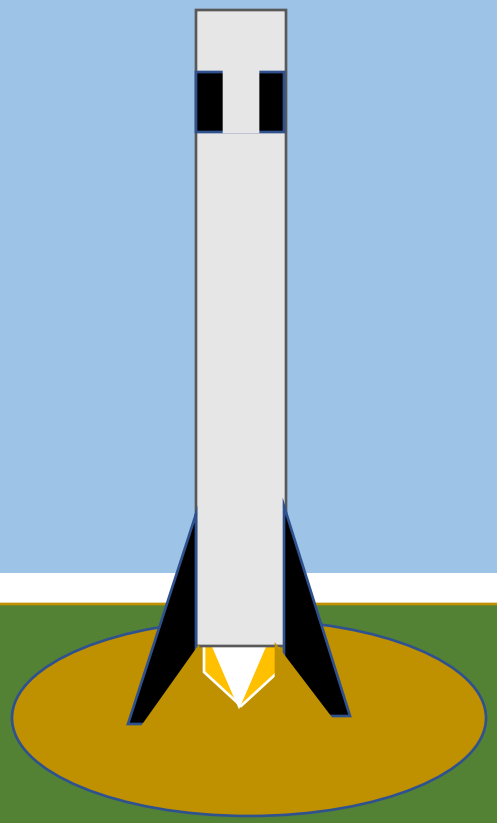


IBM Data Science Capstone Project for Studying

Feasibility of the Reusable Launch System

Author William Leung B. Eng

Date 9th October 2022

Contents

Executive summary

Introduction

SpaceX is the first commercial space company to use the partial usable launch system which can reduce costs significantly. The affordable launch cost is key for future space technology developments. The data between the publicly announced date and the transition to routine flight have been analyzed by using data science methods. The purpose of this project is to use data science methods to predict the possibility of successful reusablelaunch systems.

Methodology

In this data science project, I used the historical data from 2010 to 2020 from SpaceX to predict the feasibility of ReusableLaunch System. The SpaceX REST API gives us data about the date, orbit, launches, rocket used, payload delivered, launchSpecifications, launch sites and landing outcome. By using the URL to target a specific endpoint of the API to get past launch data, the request "GET request", normalize " json_normalize" functions to transform the JSON file into a flat table form. The completion of the required data science skills of EDA (exploring data analysis) included machine learning, visualizations to illustrate outcomes, and predictive data analysis of launch success rates to determine the feasibility of the Reusable Launch System. The launch rate is key for SpaceX's success in future developments.

Results

After completing the analysis, the Falcon 9 B5 is a reliable booster. Falcon 9 has the highest success rate from the Kennedy Space Center Launch Complex 39A (KSC LC39A), which is the best platform, and the Reusable Launch System is feasible.

Discussion

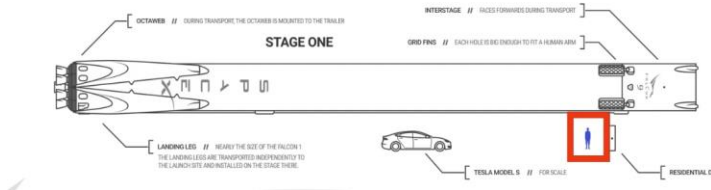
The modern space technology and engineering turn science fiction into science facts. The successfulpartially reusable launch system reduces the cost of each launch from 165 millions to 65 million. Because of this affordable reusable launch system, the space race between countries would accelerate at a tremendous speed. The international space treaties also become a challenge.

Conclusion

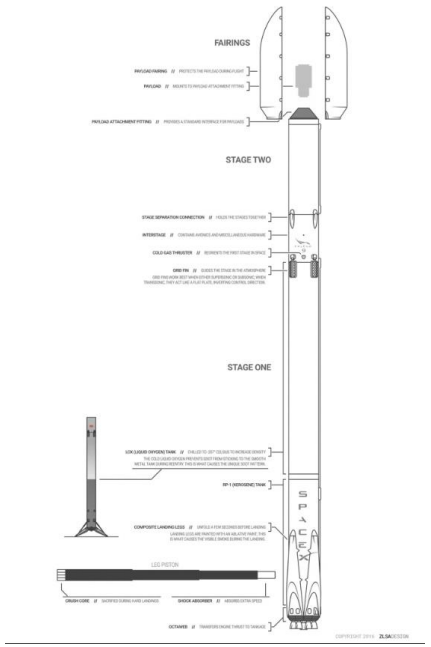
In this project, the prediction for the possibility of successful reusable launch system by using data science methods has been proven reliable and viable.

drone ship landing

ground pad landing



Falcon9 first stage



Falcon 9

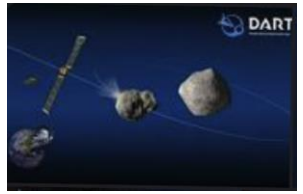
The first reflight of a Falcon 9, in March 2017

Falcon 9 1st stage takes 8:32 seconds to return landing after disengagement from Falcon 9

Introduction

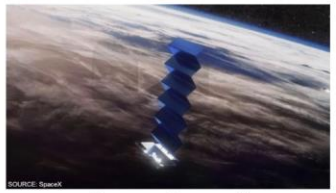
SpaceX , is an American commercial aerospace company founded in 2002 . It was the first private company to successfully launch and return a spacecraft from Earth Orbit to International Space Station (ISS). Headquarters are in Hawthorne, California. The Reusable Launch System Program was publicly announced in 2011. Four years later a 'Falcon9 first stage' successfully returned to its launch site. Then, the 'Falcon9 first stage' landed on drone ship landing in 2016. The reusable stage one allows SpaceX to launch the Falcon 9 at an affordable cost about 65 millions. The first reused 1st stage of Falcon 9 launched in March 2017.

21 stcentech.com



livemint.com

www.flickr.com



Source Space X

Source NASA

Space.com

cnet.com

	Spacecraft	Craft type	Engine	Application	Launch system	Launch type	Length (m)	Launch mass (kg)	Payload (kg)	Returned Payload (kg)	Range	Crew Size	Colony location	Status	Maiden flight
Cargon transport vehicles	Dragon 2				Falcon 9 B5	partially reusable	8.1	10,200	6,000	3,000				Operational	11 May 2018
Crew transport vehicles	Dragon 2				Falcon 9 B5	partially reusable				3,000	ISS LEO	5 7		Operational	crewed: 5 October 2022 crewed: 30 May 2020 (Uncrewed: 2 March 2019)
	Starship				Super Heavy	partially reusable	50	1,335,000	99,790		Mars	<= 100		Development	Planned 2022 without crew
Launch vehicle manufacturer					Falcon 9 B5	partially reusable		548,847	22,680					Operational	2018
					Falcon Heavy	partially reusable		1,420,788	63,800					Operational	2018
	Starship				Super Heavy	partially reusable	50	1,335,000	99,790					Development	2022 (planned)
Lander, rover, orbiter	Starship	crewed mars lander												Development	
	Starship HLS	crewed lunar lander												Development	
Propulsion manufacturers			Merlin	Falcon 1, Falcon 9, Falcon Heavy first stage/boosters										Operational	
			Merlin Vacuum	Falcon 9, Falcon Heavy second stage										Operational	
			Raptor	SpaceX Mars transportation Starship, Starship HLS										Testing	
Satellite launchers				SpaceX Starlink broadband internet service & satellite commercial contractor	Falcon 9 B5	partially reusable								Operational	
					Falcon Heavy	partially reusable									
Space settlement													Mars	Development	
Asteroid impactor transport vehicles	DART inpactor			NASA DART Planetery defence (DART inpactor by Applied Physics Lab)	Falcon 9 B5	partially resuable								Operational	1st Launch: 24 November 2021 Impact date: 26 September 2022

Present SpaceX commercial activities in space exploration and technology

Methodology

Collection and Data Wrangling

- By using the historical data from 2010 to 2020 from SpaceX to predict the feasibility of Reusable Launch System, the SpaceX REST API gives us data about the date, orbit, launches, rocket used, payload delivered, launch Specifications, launch sites and landing outcome.

Exploring Data Analysis (EDA)

- Using python/data base to determine if the Falcon 9's first stage will land and can be reused, I determined which launch site has the highest successful rate in landing, and what is an ideal payload mass which delivers 100% success rate.
- Using structed Query Language (SQL) to determine the details of launch sit KSC LC-39A for Falcon 9 booster,
- date and successful landings on drone boat and ground landing pad and total payload mass carried by boosters launched by NASA (CRS).
- Using visualization tools such as the scatter plot, bar plot and line plot to demonstrate the relationship between different variables.

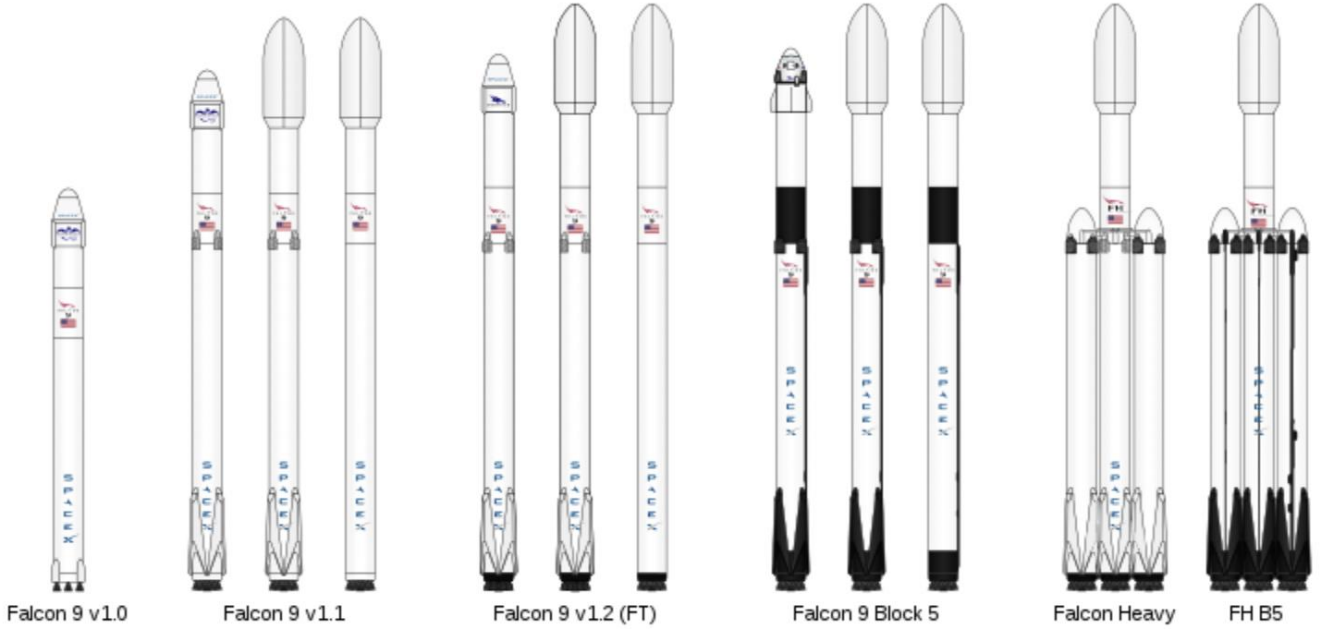
Interactive Visualization

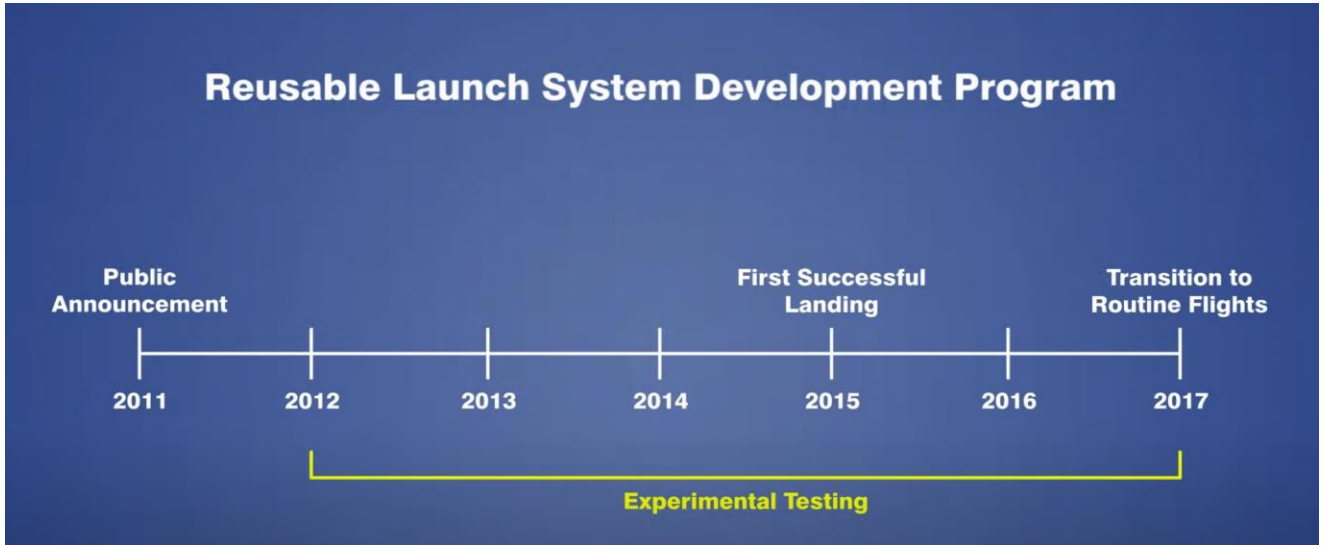
- Using Interactive Visual Analytics with Folium to analyze the launch site geo and proximities, the marker and marker cluster functions are used to discover any patterns from the SpaceX dataset.
- Building Plotly Dash Dashboard with input components such as a dropdown list and range slider to interact with a pie chart and a scatter plot chart, the patterns from the SpaceX dataset will be visualized.

Predictive Analysis (classification)

- Building a machine learning pipeline to predict if the first stage of the Falcon 9 lands successfully, train_test_split function to train and test the SpaceX dataset to search for the best hyperparameter values.
- By using the accuracy the following models: Logistic Regression, Support Vector Machine, Decision Tree Classifier and the K-nearest neighbors to generate the confusion matrix.

Collecting the Data
SpaceX Falcon 9 first stage Landing Prediction





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:

[10]:
```
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

[11]:
```
response.status_code
```

[11]:
```
200
```

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

[12]:
```
from pandas import json_normalize
```

[13]:
```
# Use json_normalize meethod to convert the json result into a dataframe
data = pd.read_json(static_json_url)
# Flattening JSON data
pd.json_normalize(data)
```

Data collected from 2010-06-04 to 2020-11-05

Using the dataframe `data` print the first 5 rows

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

[14]:

	fairings	links	static_fire_date_utc	static_fire_date_unix	tbd	net	window	rocket	success	details	crew	ships	capsules	payloads	
0	{'reused': False, 'recovery_attempt': False, 'recovered': False, 'ships': []}	{'patch': {'small': 'https://images2.imgbox.com/3c/0e/T8iJcSN3_o.png', 'large': 'https://images2.imgbox.com/40/e3/GypSkayF_o.png'}, 'reddit': {'campaign': None, 'launch': None, 'media': None, 'recovery': None}, 'flickr': {'small': [], 'original': []}, 'presskit': None, 'webcast': 'https://www.youtube.com/watch?v=0a_00nJ_Y88', 'youtube_id': '0a_00nJ_Y88', 'article': 'https://www.space.com/2196-spacex-inaugural-falcon-1-rocket-lost-launch.html', 'wikipedia': 'https://en.wikipedia.org/wiki/DemoSat'}	2006-03-17T00:00:00.000Z	1.142554e+09	False	False	0.0	5e9d0d95eda69955f709d1eb	False	Engine failure at 33 seconds and loss of vehicle	[]	[]	[]	[5eb0e4b5b6c3bb0006eeb1e1]	5e9e4502f5090
1	{'reused': False, 'recovery_attempt': False, 'recovered': False, 'ships': []}	{'patch': {'small': 'https://images2.imgbox.com/4f/e3/I0lkuJ2e_o.png', 'large': 'https://images2.imgbox.com/be/e7/iNqsqVYM_o.png'}, 'reddit': {'campaign': None, 'launch': None, 'media': None, 'recovery': None}, 'flickr': {'small': [], 'original': []}, 'presskit': None, 'webcast': 'https://www.youtube.com/watch?v=Lk4zQ2wP-Nc', 'youtube_id': 'Lk4zQ2wP-Nc', 'article': 'https://www.space.com/3590-spacex-falcon-1-rocket-fails-reach-orbit.html', 'wikipedia': 'https://en.wikipedia.org/wiki/DemoSat'}	None	NaN	False	False	0.0	5e9d0d95eda69955f709d1eb	False	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 recover first stage	[]	[]	[]	[5eb0e4b6b6c3bb0006eeb1e2]	5e9e4502f5090
3	{'reused': False, 'recovery_attempt': False, 'recovered': False, 'ships': []}	{'patch': {'small': 'https://images2.imgbox.com/e9/c9/T8CfiSYb_o.png', 'large': 'https://images2.imgbox.com/e0/a7/FNjvKlXW_o.png'}, 'reddit': {'campaign': None, 'launch': None, 'media': None, 'recovery': None}, 'flickr': {'small': [], 'original': []}, 'presskit': None, 'webcast': 'https://www.youtube.com/watch?v=dLQ2tZEH6G0', 'youtube_id': 'dLQ2tZEH6G0', 'article': 'https://en.wikipedia.org/wiki/Ratsat', 'wikipedia': 'https://en.wikipedia.org/wiki/Ratsat'}	2008-09-20T00:00:00.000Z	1.221869e+09	False	False	0.0	5e9d0d95eda69955f709d1eb	True	Ratsat was carried to orbit on the first successful orbital launch of any privately funded and developed, liquid-propelled carrier rocket, the SpaceX Falcon 1	[]	[]	[]	[5eb0e4b7b6c3bb0006eeb1e5]	5e9e4502f509
2	{'reused': False, 'recovery_attempt': False, 'recovered': False, 'ships': []}	{'patch': {'small': 'https://images2.imgbox.com/3d/86/cnu0pan8_o.png', 'large': 'https://images2.imgbox.com/4b/bd/d8UxLh4q_o.png'}, 'reddit': {'campaign': None, 'launch': None, 'media': None, 'recovery': None}, 'flickr': {'small': [], 'original': []}, 'presskit': None, 'webcast': 'https://www.youtube.com/watch?v=v0w9p3U8860', 'youtube_id': 'v0w9p3U8860', 'article': 'http://www.spacex.com/news/2013/02/11/falcon-1-flight-3-mission-summary', 'wikipedia': 'https://en.wikipedia.org/wiki/Trailblazer_(satellite)'}	None	NaN	False	False	0.0	5e9d0d95eda69955f709d1eb	False	Residual stage 1 thrust led to collision between stage 1 and stage 2	[]	[]	[]	[5eb0e4b6b6c3bb0006eeb1e3, 5eb0e4b6b6c3bb0006eeb1e4]	5e9e4502f509
4	{'reused': False, 'recovery_attempt': False, 'recovered': False, 'ships': []}	{'patch': {'small': 'https://images2.imgbox.com/a7/ba/NBZSw3Ho_o.png', 'large': 'https://images2.imgbox.com/8d/fc/0qdZMWWx_o.png'}, 'reddit': {'campaign': None, 'launch': None, 'media': None, 'recovery': None}, 'flickr': {'small': [], 'original': []}, 'presskit': 'http://www.spacex.com/press/2012/12/19/spacexs-falcon-1-successfully-delivers-razaksat-satellite-orbit', 'webcast': 'https://www.youtube.com/watch?v=yTaIDooc8Og', 'youtube_id': 'yTaIDooc8Og', 'article': 'http://www.spacex.com/news/2013/02/12/falcon-1-flight-5', 'wikipedia': 'https://en.wikipedia.org/wiki/RazakSAT'}	None	NaN	False	False	0.0	5e9d0d95eda69955f709d1eb	True	None	[]	[]	[]	[5eb0e4b7b6c3bb0006eeb1e6]	5e9e4502f5090

Using the requests library to obtain the launch data from API(.json() method), the response will be in the form of a JSON, a list of JSON objects.

I converted this JSON to a dataframe by using the json_normalize function.

It "normalized" the structured json data into a flat table.

Rearranging the data frame to Falcon 9 booster data frame

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`.

[27]:
```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = launch[launch['BoosterVersion'] == 'Falcon 9']
```

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

[28]:
```
data_falcon9.head()
```

[28]:

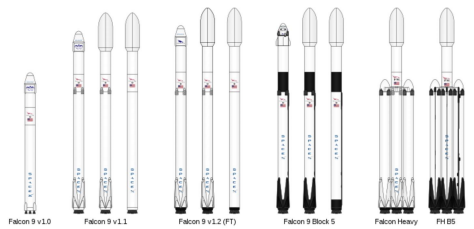
	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude
4	6	2010-06-04	Falcon 9	None	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1	0	B0003	-80.577366	28.561857
5	8	2012-05-22	Falcon 9	525	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1	0	B0005	-80.577366	28.561857
6	10	2013-03-01	Falcon 9	677	ISS	CCSFS SLC 40	None None	1	False	False	False	None	1	0	B0007	-80.577366	28.561857
7	11	2013-09-29	Falcon 9	500	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None	1	0	B1003	-120.610829	34.632093
8	12	2013-12-03	Falcon 9	3170	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1	0	B1004	-80.577366	28.561857

[34]:
```
data_falcon9.loc[:,"FlightNumber"] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9
```

[34]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude
4	1	2010-06-04	Falcon 9	None	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1	0	B0003	-80.577366	28.561857
5	2	2012-05-22	Falcon 9	525	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1	0	B0005	-80.577366	28.561857
6	3	2013-03-01	Falcon 9	677	ISS	CCSFS SLC 40	None None	1	False	False	False	None	1	0	B0007	-80.577366	28.561857
7	4	2013-09-29	Falcon 9	500	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None	1	0	B1003	-120.610829	34.632093
8	5	2013-12-03	Falcon 9	3170	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1	0	B1004	-80.577366	28.561857
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
89	86	2020-09-03	Falcon 9	15600	VLEO	KSC LC 39A	True ASDS	2	True	True	True	5e9e3032383ecb6bb234e7ca	5	12	B1060	-80.603956	28.608058
90	87	2020-10-06	Falcon 9	15600	VLEO	KSC LC 39A	True ASDS	3	True	True	True	5e9e3032383ecb6bb234e7ca	5	11	B1058	-80.603956	28.608058
91	88	2020-10-18	Falcon 9	15600	VLEO	KSC LC 39A	True ASDS	6	True	True	True	5e9e3032383ecb6bb234e7ca	5	11	B1051	-80.603956	28.608058
92	89	2020-10-24	Falcon 9	15600	VLEO	CCSFS SLC 40	True ASDS	3	True	True	True	5e9e3033383ecbb9e534e7cc	5	12	B1060	-80.577366	28.561857
93	90	2020-11-05	Falcon 9	3681	MEO	CCSFS SLC 40	True ASDS	1	True	False	True	5e9e3032383ecb6bb234e7ca	5	6	B1062	-80.577366	28.561857

90 rows × 17 columns



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.

[30]:
```
# Calculate the mean value of PayloadMass column
mean = data_falcon9["PayloadMass"].mean()

# Replace the np.nan values with its mean value
data_falcon9["PayloadMass"].replace(np.nan, mean)
```

[30]:
```
4      6123.547647
5       525.000000
6       677.000000
7       500.000000
8      3170.000000
          ...
89    15600.000000
90    15600.000000
91    15600.000000
92    15600.000000
93     3681.000000
Name: PayloadMass, Length: 90, dtype: float64
```

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

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

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.

[31]:
```
print(mean)
```

```
6123.547647058824
```

`data_falcon9.to_csv('dataset_part_1.csv', index=False)`

Replacing the missing values with the average falcon 9 Payload Mass 6,123.35 Kg

Data Wrangling

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:

[5]:
```
# Apply value_counts() on column LaunchSite
Num_launches=df['LaunchSite'].value_counts()
Num_launches
```

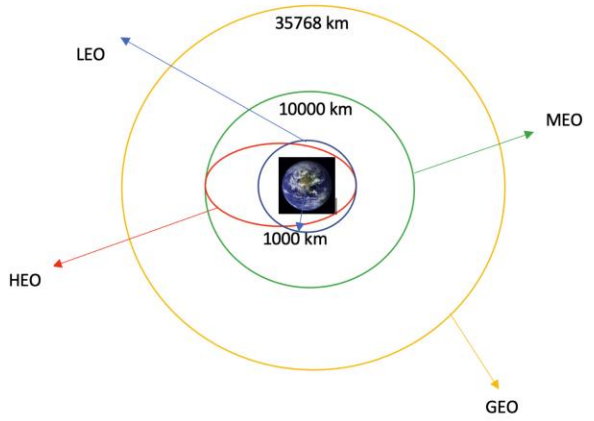
[5]:
```
CCAFS SLC 40    55
KSC LC 39A      22
VAFB SLC 4E     13
Name: LaunchSite, dtype: int64
```

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`

[9]:
```
# Apply value_counts on Orbit column
Num_occur_orbit = df['Orbit'].value_counts()
Num_occur_orbit
```

[9]:
```
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
```



The Number of Occurrence of Mission Outcome

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

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

[10]:
```
# landing_outcomes = values on Outcome column
landing_outcomes = df['Outcome'].value_counts()
landing_outcomes
```

[10]:
```
True ASDS      41
None None      19
True RTLS      14
False ASDS      6
True Ocean      5
False Ocean     2
None ASDS       2
False RTLS      1
Name: Outcome, dtype: int64
```

`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.

[11]:
```
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:

[12]:
```
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
bad_outcomes
```

[12]:
```
{'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

Overall Launch Success Rate is 66.67 %

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`:

[10]:
```
# 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 variable that represents the outcome of each launch. If the value is zero, the first stage did not land successfully; one means the first stage landed Successfully

[11]:
```
df['Class']=landing_class
df.head(10)
```

[11]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577366	28.561857	0
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366	28.561857	0
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.577366	28.561857	0
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	0	B1003	-120.610829	34.632093	0
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561857	0
5	6	2014-01-06	Falcon 9	3325.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1005	-80.577366	28.561857	0
6	7	2014-04-18	Falcon 9	2296.000000	ISS	CCAFS SLC 40	True Ocean	1	False	False	True	NaN	1.0	0	B1006	-80.577366	28.561857	1
7	8	2014-07-14	Falcon 9	1316.000000	LEO	CCAFS SLC 40	True Ocean	1	False	False	True	NaN	1.0	0	B1007	-80.577366	28.561857	1
8	9	2014-08-05	Falcon 9	4535.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1008	-80.577366	28.561857	0
9	10	2014-09-07	Falcon 9	4428.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1011	-80.577366	28.561857	0

We can use the following line of code to determine the success rate:

[12]:
```
df["Class"].mean()
```

[12]:
```
0.6666666666666666
```

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.

```
df.to_csv("dataset_part_2.csv", index=False)
```

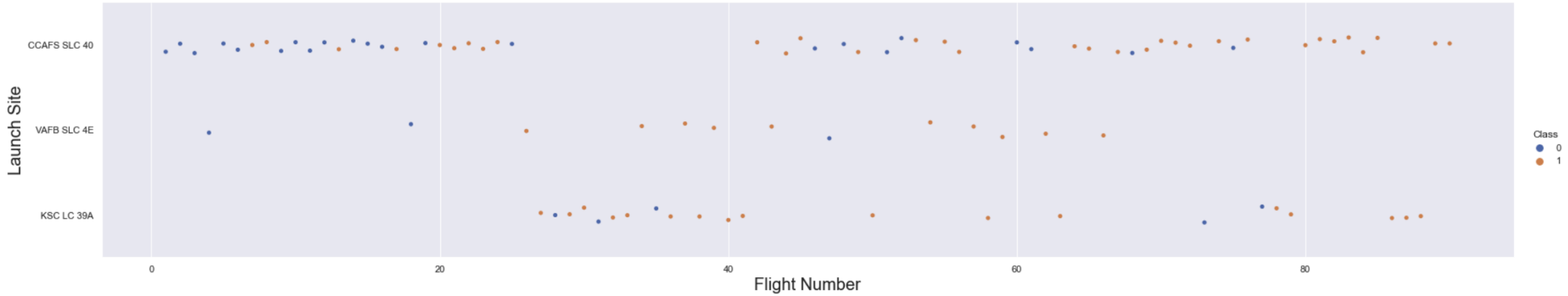
Exploring Data Analysis with Visualization

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'`

[20]:
```
# 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("Launch Site", fontsize=20)
plt.show
```

[20]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Now try to explain the patterns you found in the Flight Number vs. Launch Site scatter point plots.

Launch Site " CCAFS SLC 40" accommodates more flights than the other sites

Lauch Site "CCAFS SKC 40" has Largest Number of Launches

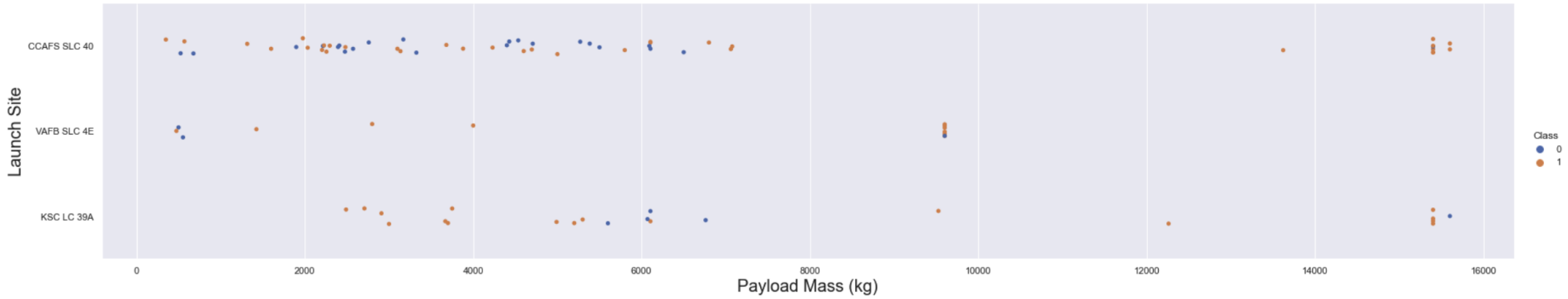
TASK 2: Visualize the relationship between Payload and Launch Site

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

[21]:
```
# 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("Payload Mass (kg)", fontsize=20)
plt.ylabel("Launch Site", fontsize=20)
plt.show
```

[21]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```

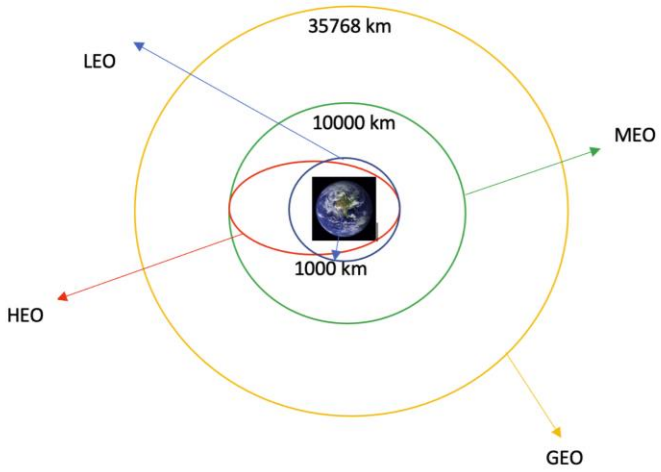


Now if you observe Payload Vs. Launch Site scatter point chart you will find for the VAFB-SLC launchsite there are no rockets launched for heavypayload mass(greater than 10000).

CCAFS LC-40, has a success rate of 60%, but the mass above 10,000 kg WITH THE SUCCESS RATE OF 100%

100 % SUCCESS RATE

ORBITS:
ES-L1
GEO
HEO
SSO



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

[6]:
```
# HINT use groupby method on Orbit column and get the mean of Class column
mean =df.groupby(['Orbit']).mean()
mean
```

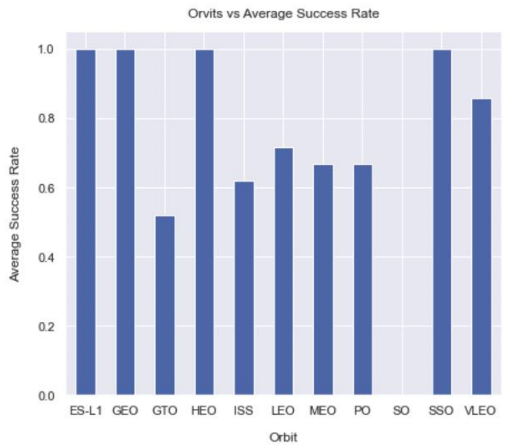
[6]:

Orbit	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Longitude	Latitude	Class
ES-L1	13.000000	570.000000	1.000000	1.000000	0.000000	1.000000	1.000000	0.000000	-80.577366	28.561857	1.000000
GEO	83.000000	6104.959412	2.000000	1.000000	1.000000	1.000000	5.000000	2.000000	-80.577366	28.561857	1.000000
GTO	35.037037	5011.994444	1.407407	0.629630	0.333333	0.629630	3.037037	0.962963	-80.586229	28.577258	0.518519
HEO	49.000000	350.000000	1.000000	1.000000	0.000000	1.000000	4.000000	1.000000	-80.577366	28.561857	1.000000
ISS	39.142857	3279.938095	1.238095	0.809524	0.238095	0.857143	3.142857	1.285714	-80.583697	28.572857	0.619048
LEO	20.000000	3882.839748	1.000000	0.571429	0.000000	0.714286	2.142857	0.428571	-80.584963	28.575058	0.714286
MEO	77.666667	3987.000000	1.000000	0.666667	0.000000	0.666667	5.000000	0.666667	-80.577366	28.561857	0.666667
PO	36.333333	7583.666667	1.333333	0.888889	0.333333	0.777778	3.222222	1.555556	-120.610829	34.632093	0.666667
SO	73.000000	6104.959412	4.000000	0.000000	1.000000	0.000000	5.000000	3.000000	-80.603956	28.608058	0.000000
SSO	60.800000	2060.000000	2.400000	1.000000	0.800000	1.000000	4.600000	3.200000	-112.604136	33.418046	1.000000
VLEO	78.928571	15315.714286	3.928571	1.000000	1.000000	1.000000	5.000000	3.928571	-80.586862	28.578358	0.857143

Analyze the ploted bar chart try to find which orbits have high sucess rate.

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

[7]:
```
sns.set(font_scale=1.0)
mean['Class'].plot(kind='bar', figsize=(7, 6), rot=0)
plt.xlabel("Orbit", labelpad=14)
plt.ylabel("Average Success Rate", labelpad=14)
plt.title("Orvits vs Average Success Rate", y=1.02);
```



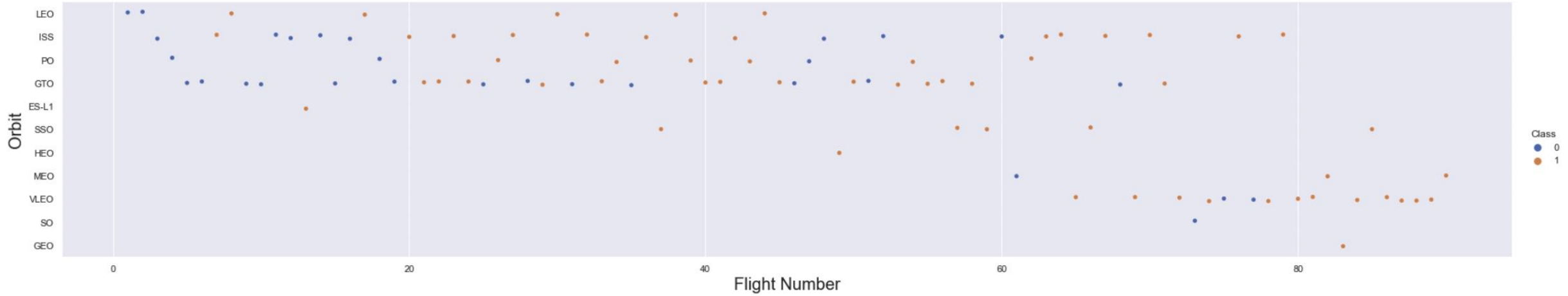
For LEO orbit the success appears related to the number of flights There is no relationship between success and flight in GTO orbit 100% success in ES-L1, SSO, HEO and GEO orbits

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.

[24]:
```
# 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
```

[24]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



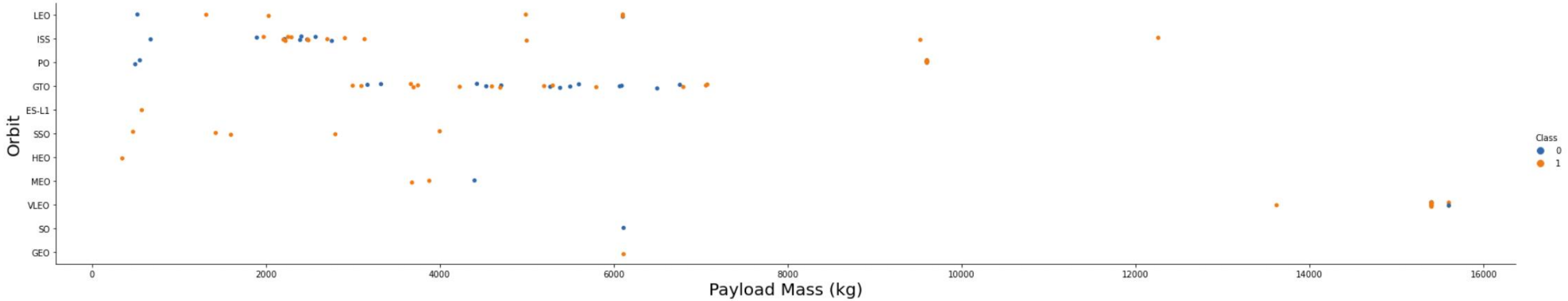
The orbit GTO has largest mix of successful and unsuccessful landings
100% success rate landing for ES-L1, SSO, HEO and MEO orbits with payload less than 4, 000 kg
The successful landing rates are higher for Polar, LEO and ISS orbits with heavy payload mass

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

[27]:
```
# 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 Mass (kg)", fontsize=20)
plt.ylabel("Orbit", fontsize=20)
plt.show
```

[27]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



The success rate continued to increase from 2013 to 2020

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:

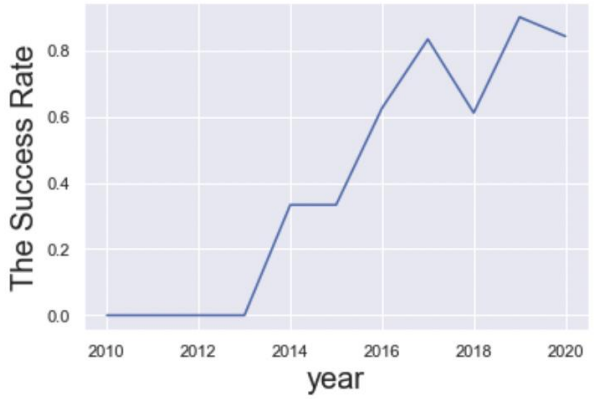
[11]:
```
df['year'] = pd.DatetimeIndex(df['Date']).year
mean_success =df.groupby(['year']).mean()
mean_success.reset_index()
mean_success.head()
```

[11]:

year	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Longitude	Latitude	Class
2010	1.0	6104.959412	1.0	0.000000	0.0	0.000000	1.0	0.0	-80.577366	28.561857	0.000000
2012	2.0	525.000000	1.0	0.000000	0.0	0.000000	1.0	0.0	-80.577366	28.561857	0.000000
2013	4.0	1449.000000	1.0	0.000000	0.0	0.000000	1.0	0.0	-93.921854	30.585269	0.000000
2014	8.5	3019.333333	1.0	0.000000	0.0	0.333333	1.0	0.0	-80.577366	28.561857	0.333333
2015	14.5	2346.833333	1.0	0.833333	0.0	0.833333	1.0	0.0	-80.577366	28.561857	0.333333

[13]:
```
# Plot a line chart with x axis to be the extracted year and y axis to be the success rate
sns.lineplot(y="Class", x="year",data=mean_success)
plt.xlabel("year", fontsize=20)
plt.ylabel("The Success Rate", fontsize=20)
plt.show
```

[13]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



you can observe that the sucess rate since 2013 kept increasing till 2020

Therefore, we will combine multiple features.
These features are correlated with successful landings.

The categorical variables will be converted using one hot encoding, preparing the data **for a machine learning model that will predict if the first stage will successfully land.**

Features Engineering

By now, you should obtain some preliminary insights about how each important variable would affect the success rate, we will select the features that will be used in success prediction in the future module.

[5]:
```
features = df[['FlightNumber', 'PayloadMass', 'Orbit', 'LaunchSite', 'Flights', 'GridFins', 'Reused', 'Legs', 'LandingPad', 'Block', 'ReusedCount', 'Serial']]
features.head()
```

[5]:

	FlightNumber	PayloadMass	Orbit	LaunchSite	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial
0	1	6104.959412	LEO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B0003
1	2	525.000000	LEO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B0005
2	3	677.000000	ISS	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B0007
3	4	500.000000	PO	VAFB SLC 4E	1	False	False	False	NaN	1.0	0	B1003
4	5	3170.000000	GTO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B1004

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.

[14]:
```
# HINT: Use get_dummies() function on the categorical columns

features_one_hot=pd.get_dummies(features)
features_one_hot.head()
```

[14]:

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	...	Serial_B1048	Serial_B1049	Serial_B1050	Serial_B1051	Serial_B1054	Serial_B1056	Serial_B1058	Serial_B1059	Serial_B1060	Serial_B1062
0	1	6104.959412	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	2	525.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	3	677.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	4	500.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	5	3170.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 80 columns

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`

[19]:
```
# HINT: use astype function

features_one_hot=features_one_hot.astype(float)
```

[20]:
```
features_one_hot.head()
```

[20]:

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	...	Serial_B1048	Serial_B1049	Serial_B1050	Serial_B1051	Serial_B1054	Serial_B1056	Serial_B1058	Serial_B1059	Serial_B1060	Serial_B1062
0	1.0	6104.959412	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	2.0	525.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	3.0	677.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	4.0	500.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	5.0	3170.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 80 columns

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.

```
features_one_hot.to_csv('dataset_part_3.csv', index=False)
```

Task 1 Display the names of the unique launch sites in the space mission

[72]:
```
%sql SELECT DISTINCT Launch_Site FROM SPACEXTBL;
```

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

[72]:

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

Task 2 Display 5 records where launch sites begin with the string 'KSC'

[73]:
```
%%sql SELECT * FROM SPACEXTBL
WHERE Launch_Site LIKE 'KSC%' LIMIT 5;
```

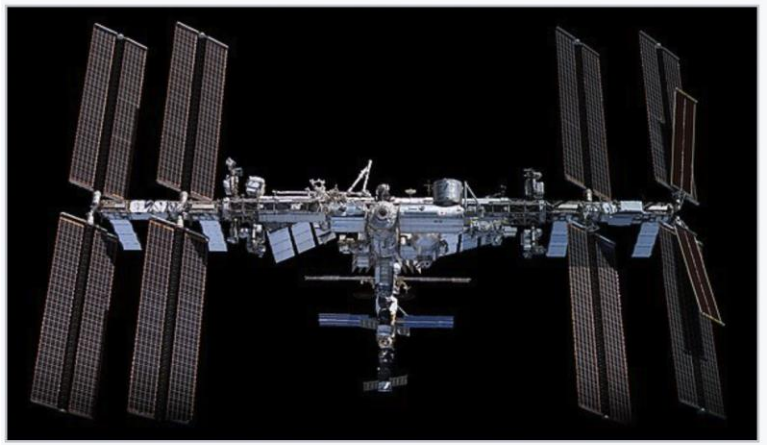
```
 * sqlite:///my_data1.db
Done.
```

[73]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2017-02-19	14:39:00	F9 FT B1031.1	KSC LC-39A	SpaceX CRS-10	2490	LEO (ISS)	NASA (CRS)	Success	Success (ground pad)
2017-03-16	06:00:00	F9 FT B1030	KSC LC-39A	EchoStar 23	5600	GTO	EchoStar	Success	No attempt
2017-03-30	22:27:00	F9 FT B1021.2	KSC LC-39A	SES-10	5300	GTO	SES	Success	Success (drone ship)
2017-05-01	11:15:00	F9 FT B1032.1	KSC LC-39A	NROL-76	5300	LEO	NRO	Success	Success (ground pad)
2017-05-15	23:21:00	F9 FT B1034	KSC LC-39A	Inmarsat-5 F4	6070	GTO	Inmarsat	Success	No attempt

Explore Data Analysis With SQL

Commercial Resupply Services (CRS) are a series of commercially contracted spacecraft for delivering of cargo and supplies to International Space Station (ISS)



Commercial Resupply Services missions approaching International Space Station

Task 3 Display the total payload mass carried by boosters launched by NASA (CRS)

[15]:
```
%%sql SELECT * FROM SPACEXTBL
WHERE Customer IN ('NASA (CRS)') LIMIT 5;
```

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

[15]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
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
2014-04-18	19:25:00	F9 v1.1	CCAFS LC-40	SpaceX CRS-3	2296	LEO (ISS)	NASA (CRS)	Success	Controlled (ocean)
2014-09-21	05:52:00	F9 v1.1 B1010	CCAFS LC-40	SpaceX CRS-4	2216	LEO (ISS)	NASA (CRS)	Success	Uncontrolled (ocean)
2015-01-10	09:47:00	F9 v1.1 B1012	CCAFS LC-40	SpaceX CRS-5	2395	LEO (ISS)	NASA (CRS)	Success	Failure (drone ship)

[16]:
```
%%sql SELECT SUM(PAYLOAD_MASS__KG_) AS 'Total payload mass in Kg carried by boosters for NASA (CRS)' FROM SPACEXTBL
WHERE Customer IN ('NASA (CRS)');
```

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

[16]:

Total payload mass in Kg carried by boosters for NASA (CRS)
45596

Task 4

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

[79]:
```
%%sql SELECT AVG(PAYLOAD_MASS__KG_) FROM SPACEXTBL
WHERE Booster_Version LIKE 'F9 v1.1';
```

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

[79]:

AVG(PAYLOAD_MASS__KG_)
2928.4

Task 5 List the date where the succesful landing outcome in drone ship was acheived. Hint:Use min function

[81]:
```
%%sql SELECT min(Date), Landing_Outcome FROM SPACEXTBL
WHERE Landing_Outcome='Success (drone ship)';
```

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

[81]:

min(Date)	Landing_Outcome
2016-04-08	Success (drone ship)

Task 6 List the names of the boosters which have success in ground pad and have payload mass greater than 4000 but less than 6000

[105]:
```
%%sql SELECT Booster_Version, Landing_Outcome  FROM SPACEXTBL
WHERE Landing_Outcome LIKE 'Success (ground pad)' AND PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000;
```

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

[105]:

Booster_Version	Landing_Outcome
F9 FT B1032.1	Success (ground pad)
F9 B4 B1040.1	Success (ground pad)
F9 B4 B1043.1	Success (ground pad)

Task 7 List the total number of successful and failure mission outcomes

[84]:
```
%%sql SELECT COUNT(Mission_Outcome) as Successful_Outcome FROM SPACEXTBL
WHERE Mission_Outcome LIke 'Succ%';
```

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

[84]:

Successful_Outcome
100

[85]:
```
%%sql SELECT COUNT(Mission_Outcome) as Failure_Outcome FROM SPACEXTBL
WHERE Mission_Outcome like 'Fail%';
```

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

[85]:

Failure_Outcome
1

Task 8

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

[7]:
```
%%sql SELECT Booster_version FROM SPACEXTBL
WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTBL)
```

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

[7]:

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

Falcon 9 Block 5

Task 9

List the records which will display the month names, succesful landing_outcomes in ground pad ,booster versions, launch_site for the months in year 2017¶ 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)='2017' for year.

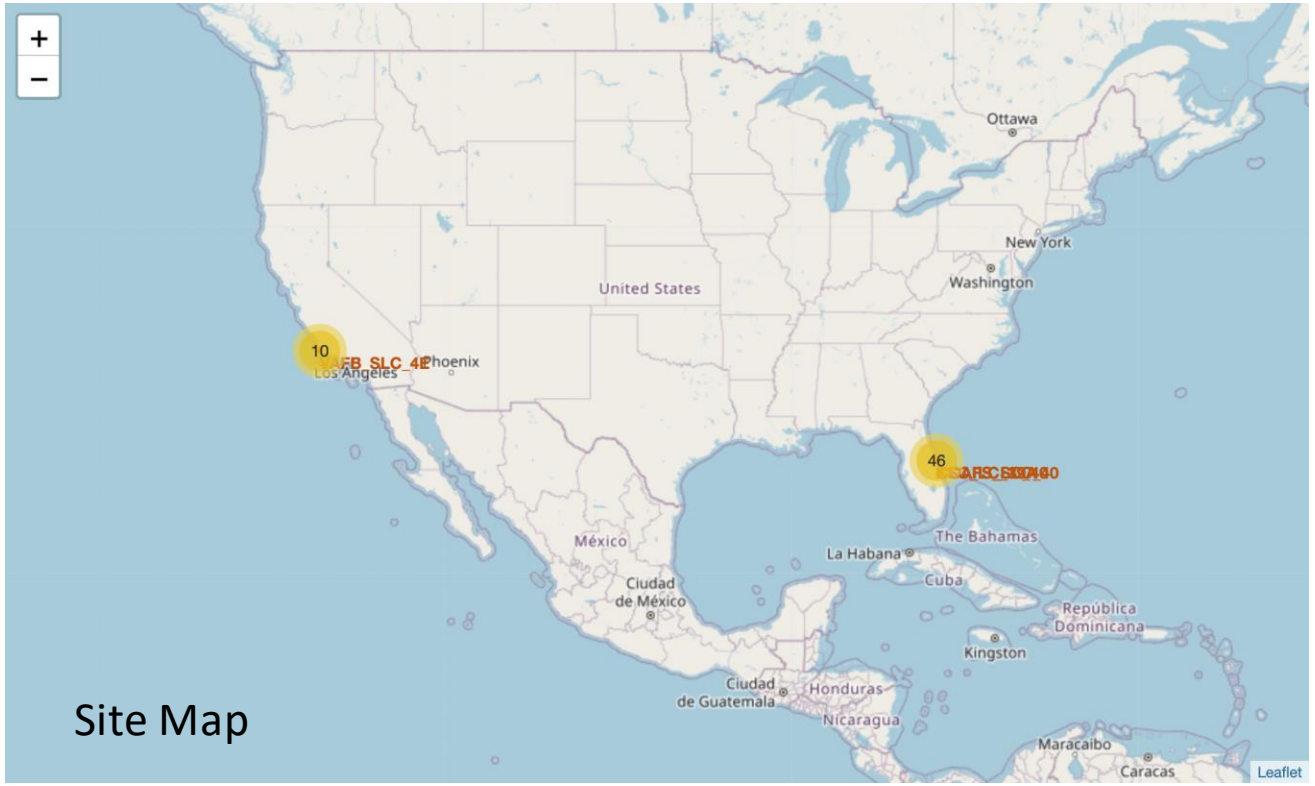
[56]:
```
%%sql SELECT
CASE
  WHEN SUBSTR(Date, 6,2)='01' THEN 'January'
  WHEN SUBSTR(Date, 6,2)='02' THEN 'February'
  WHEN SUBSTR(Date, 6,2)='03' THEN 'March'
  WHEN SUBSTR(Date, 6,2)='04' THEN 'April'
  WHEN SUBSTR(Date, 6,2)='05' THEN 'May'
  WHEN SUBSTR(Date, 6,2)='06' THEN 'June'
  WHEN SUBSTR(Date, 6,2)='07' THEN 'July'
  WHEN SUBSTR(Date, 6,2)='08' THEN 'August'
  WHEN SUBSTR(Date, 6,2)='09' THEN 'September'
  WHEN SUBSTR(Date, 6,2)='10' THEN 'October'
  WHEN SUBSTR(Date, 6,2)='11' THEN 'November'
  WHEN SUBSTR(Date, 6,2)='12' THEN 'December'
END AS 'Month Name (2017)', Landing_Outcome, Booster_version, Launch_Site
FROM SPACEXTBL
WHERE SUBSTR(Date,1,4)='2017' AND Landing_Outcome LIKE 'Success (ground pad)'
```

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

[56]:

Month Name (2017)	Landing_Outcome	Booster_Version	Launch_Site
February	Success (ground pad)	F9 FT B1031.1	KSC LC-39A
May	Success (ground pad)	F9 FT B1032.1	KSC LC-39A
June	Success (ground pad)	F9 FT B1035.1	KSC LC-39A
August	Success (ground pad)	F9 B4 B1039.1	KSC LC-39A
September	Success (ground pad)	F9 B4 B1040.1	KSC LC-39A
December	Success (ground pad)	F9 FT B1035.2	CCAFS SLC-40

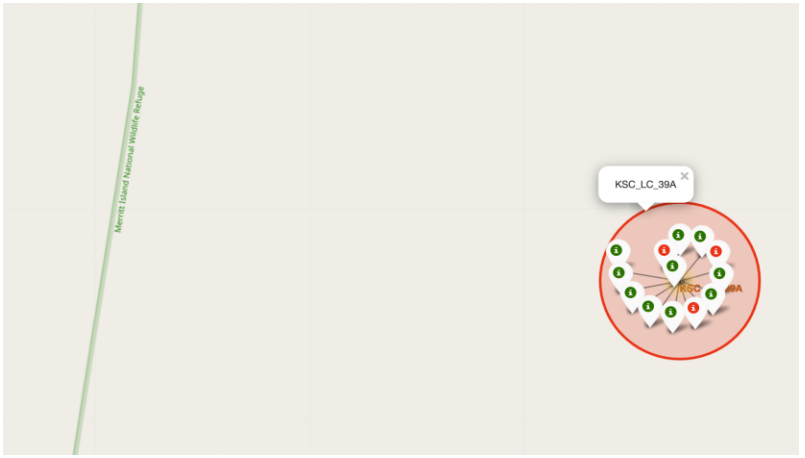
Converted the month number to month name

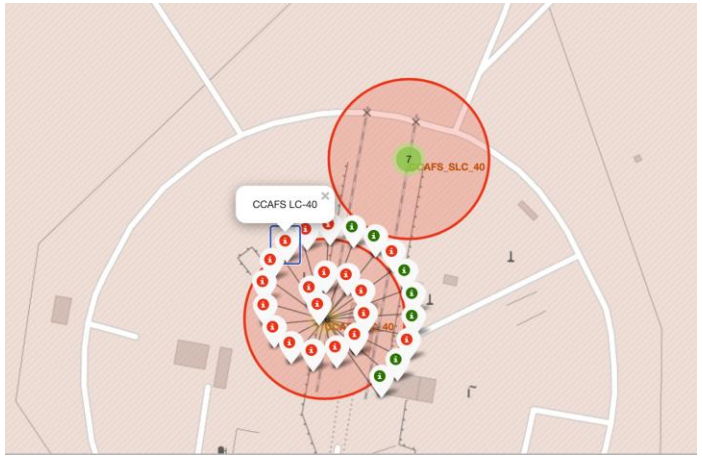


Site Map

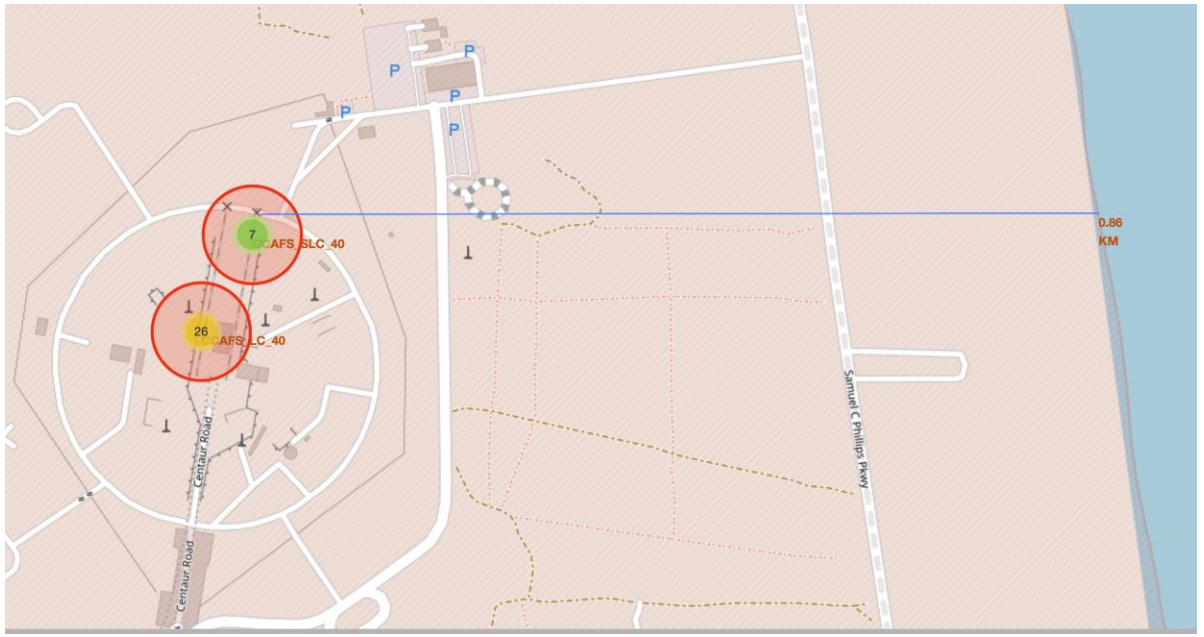
LAUNCH
SUCCESS RATE
RANKING
AS FOLLOW:

1ST KSC LC-39A
2ND CCAFS LC-40
3RD WAFB SLC-4E
4TH CCAFS SLC-4

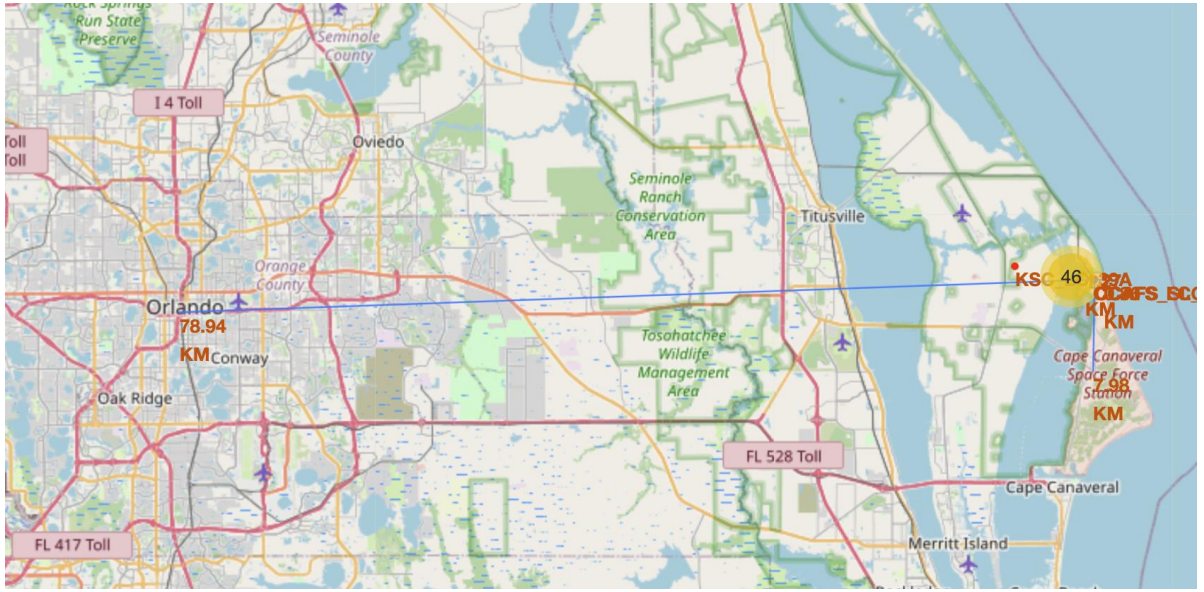




From the color-labeled markers in marker clusters, launch site KSC_LC_39A has highest high success rates.



Closest Coastline

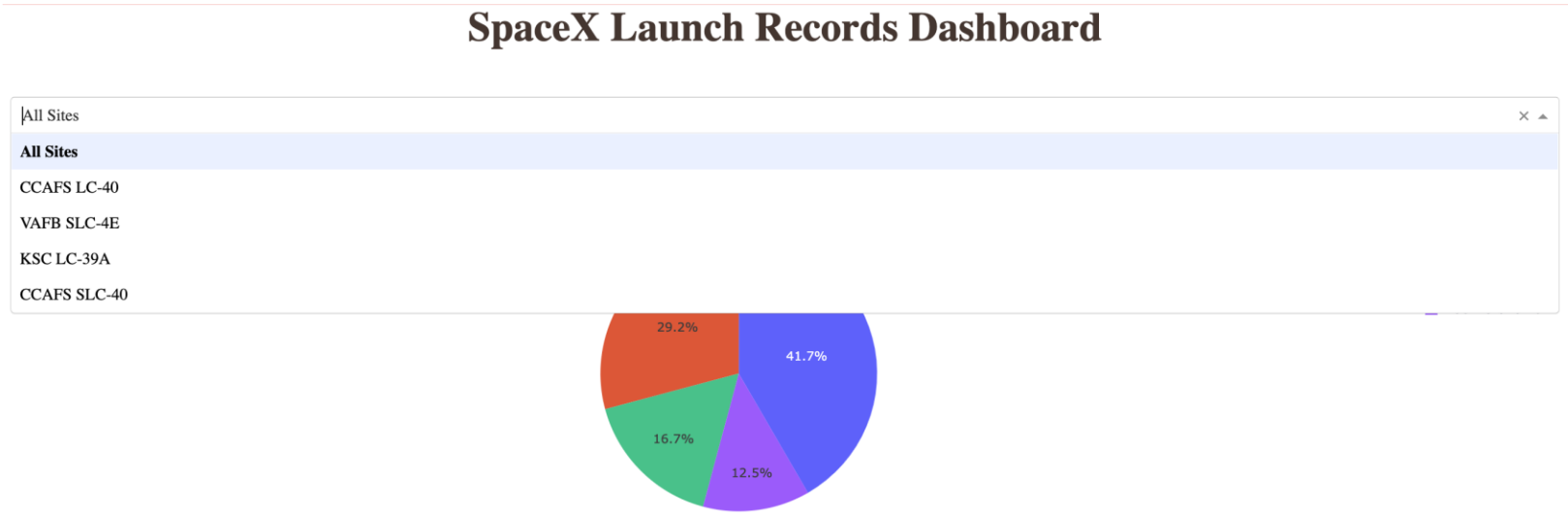


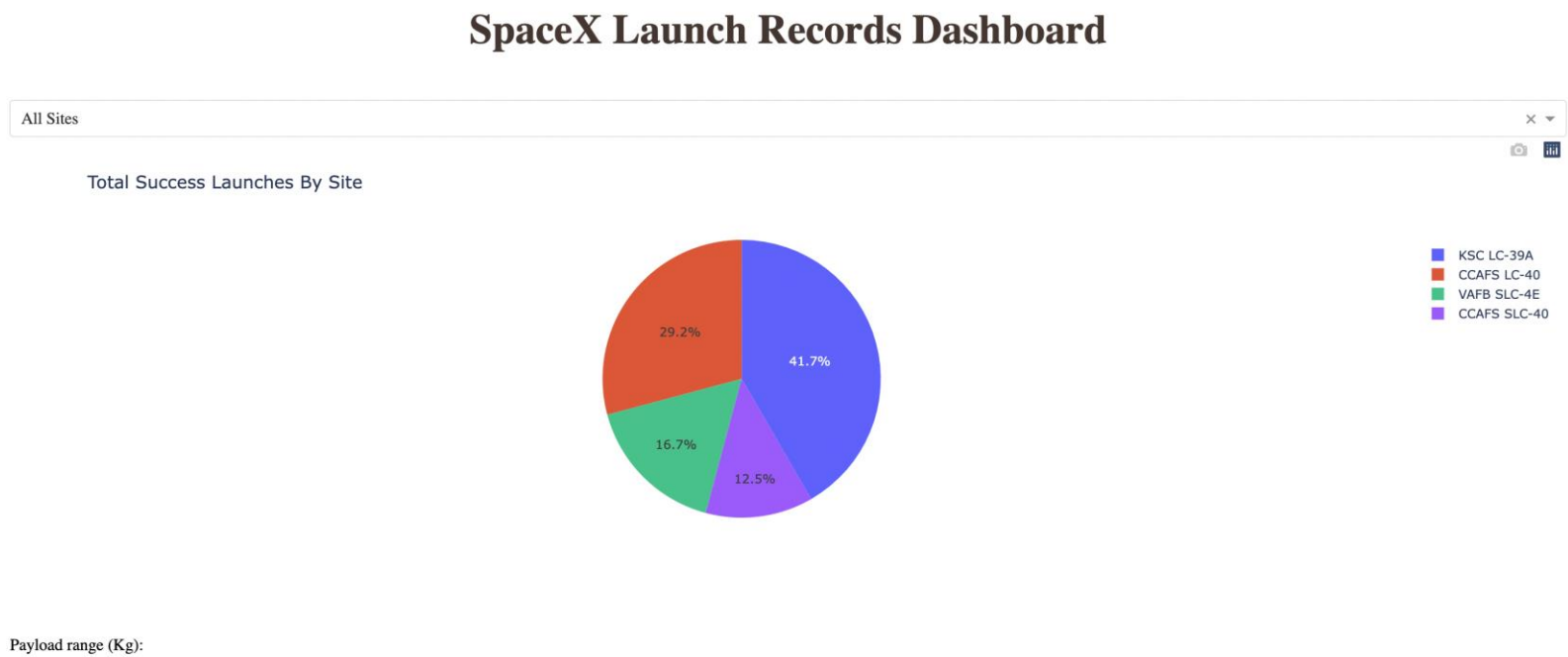
Closest City

Closest Highway and Railway

DROPDOWN MANUAL FOR CHOICES OF SITES ON COMMAND

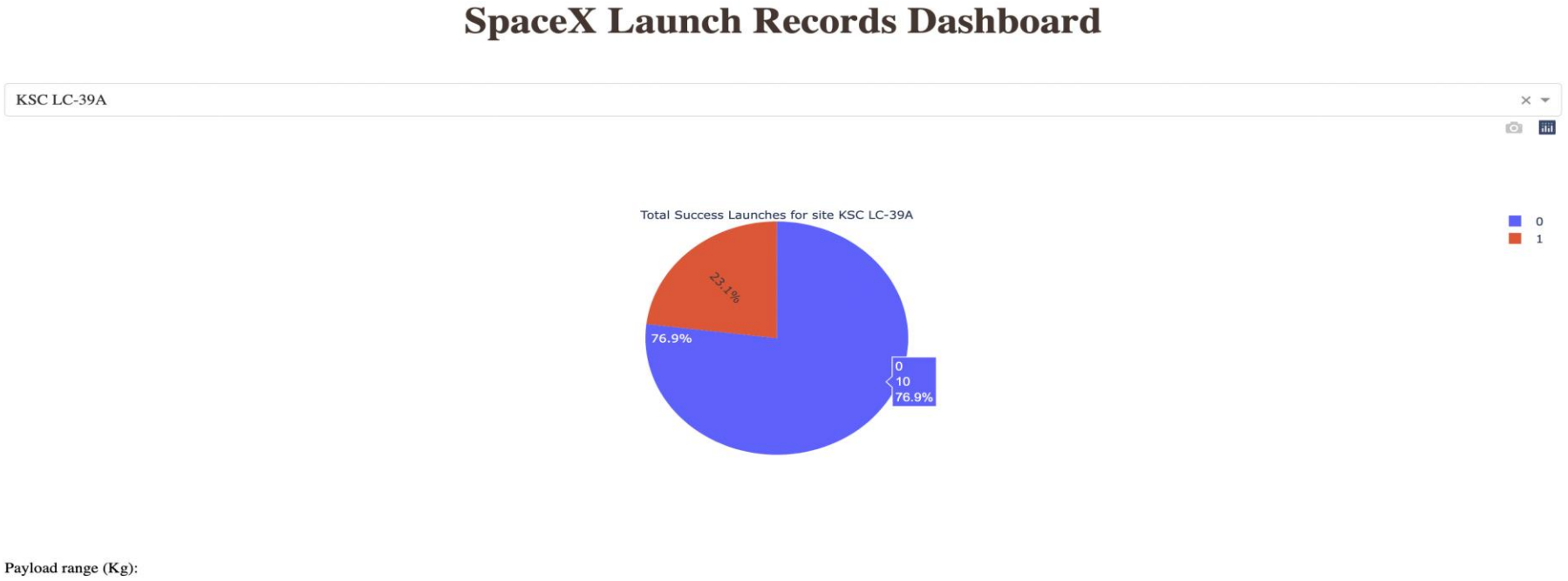
KSC LC-39A HAS THE HIGHEST SUCCESS RATE OF 41.7% AMONG ALL SUCCESSFUL LAUNCHES OF ALL SITES

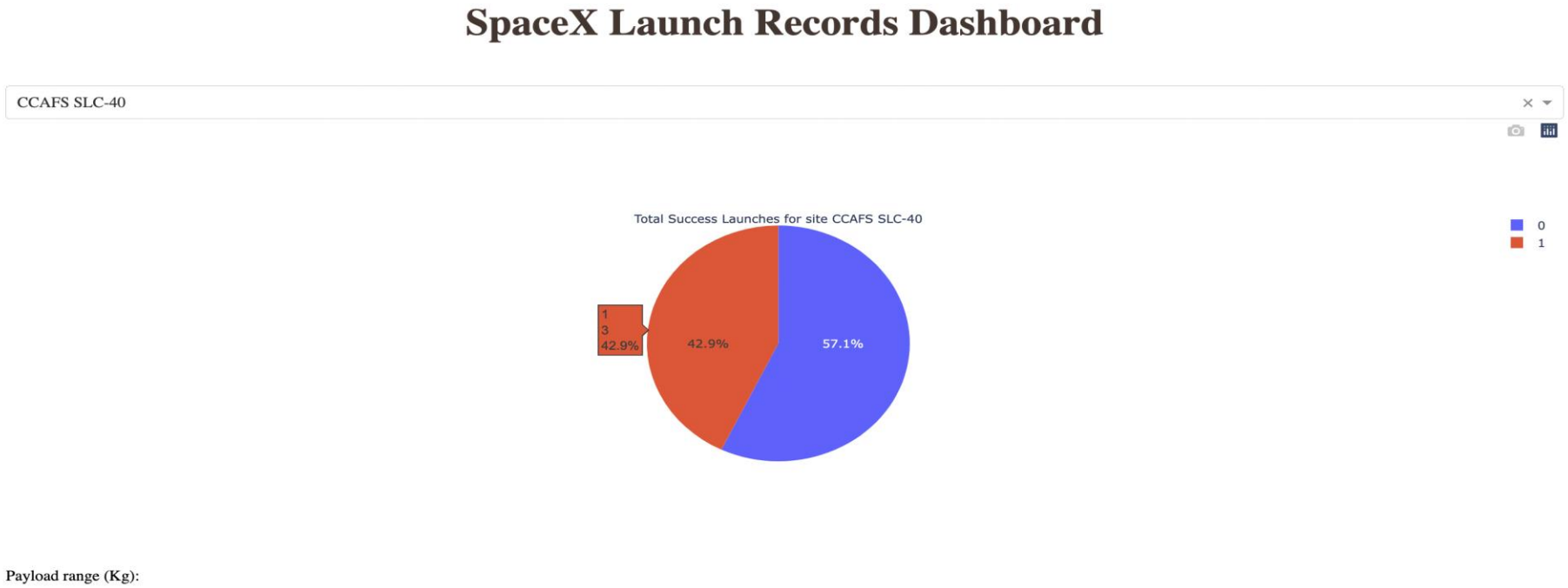




KSC LC-39A HAS THE HIGHEST SUCCESS RATE OF 76.9%

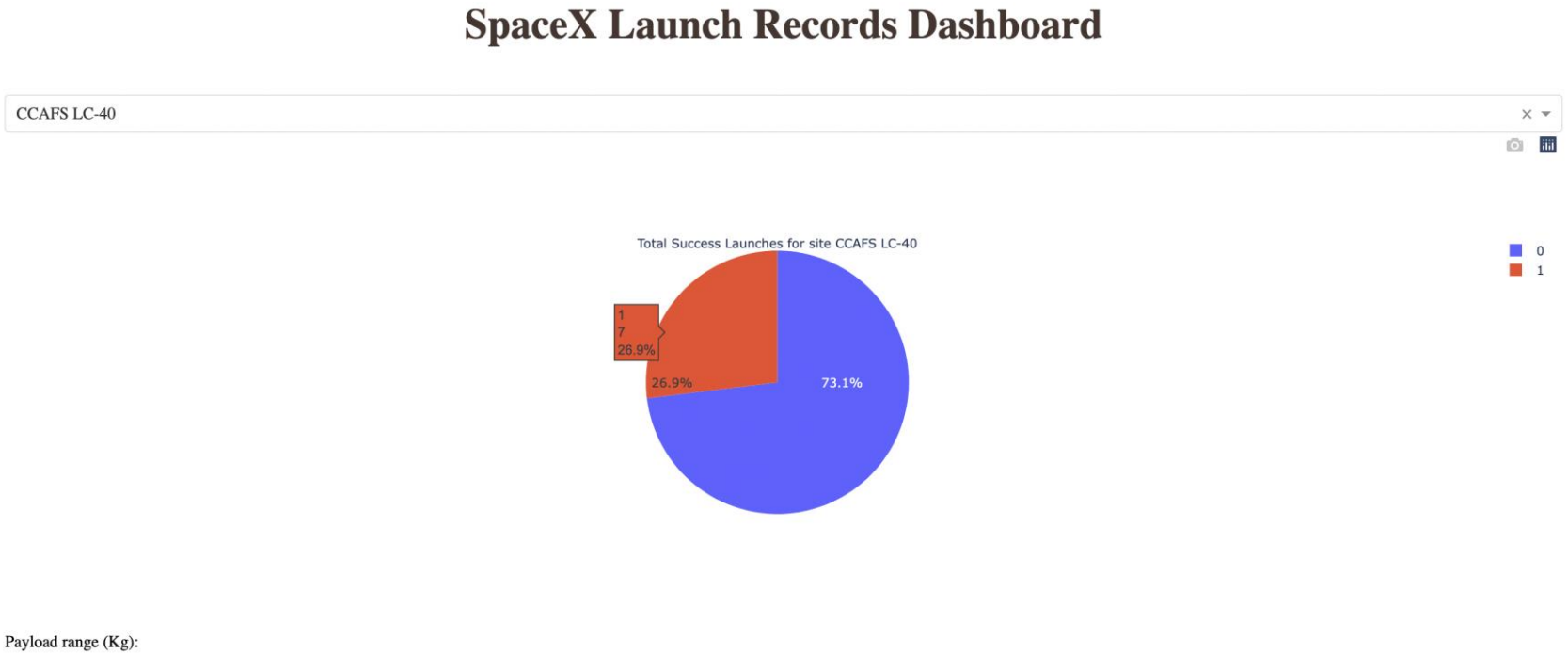
CCAFS SLC-40 HAS THE LOWEST SUCCESS RATE OF 57.1%

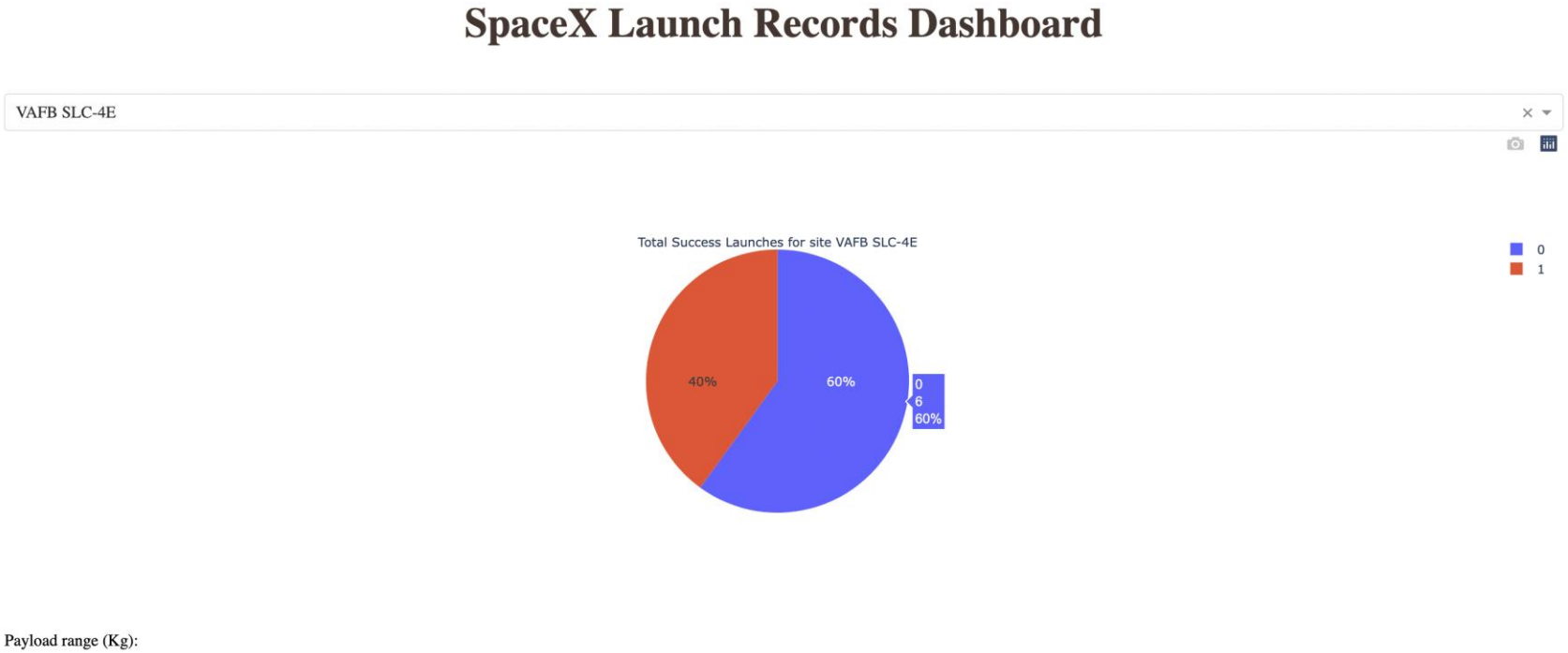




LAUNCH SUCCESS RATE RANKING AS FOLLOWS:

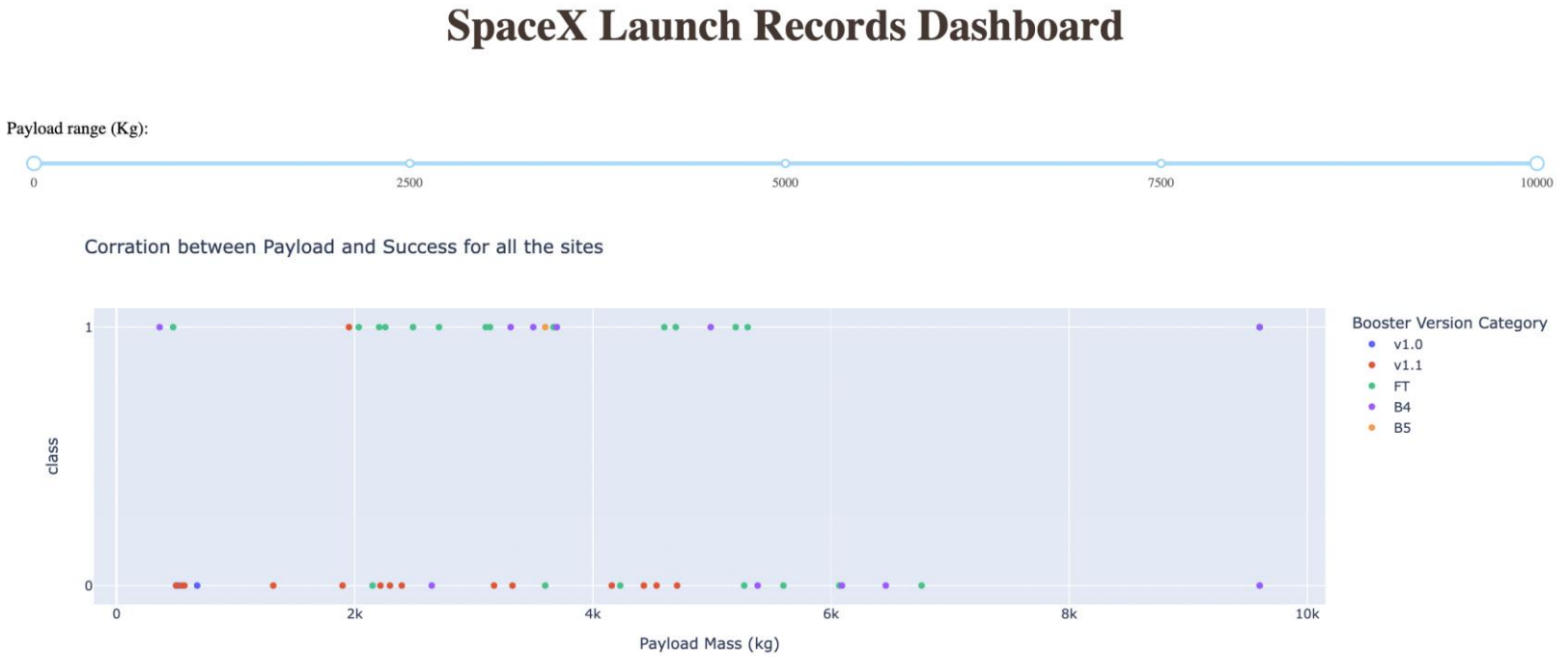
1ST	KSC LC-39A	76.9%
2ND	CCAFS LC-40	73.1%
3RD	VAFB SLC-4E	60.0%
4TH	CCAFS SLC-40	57.1%

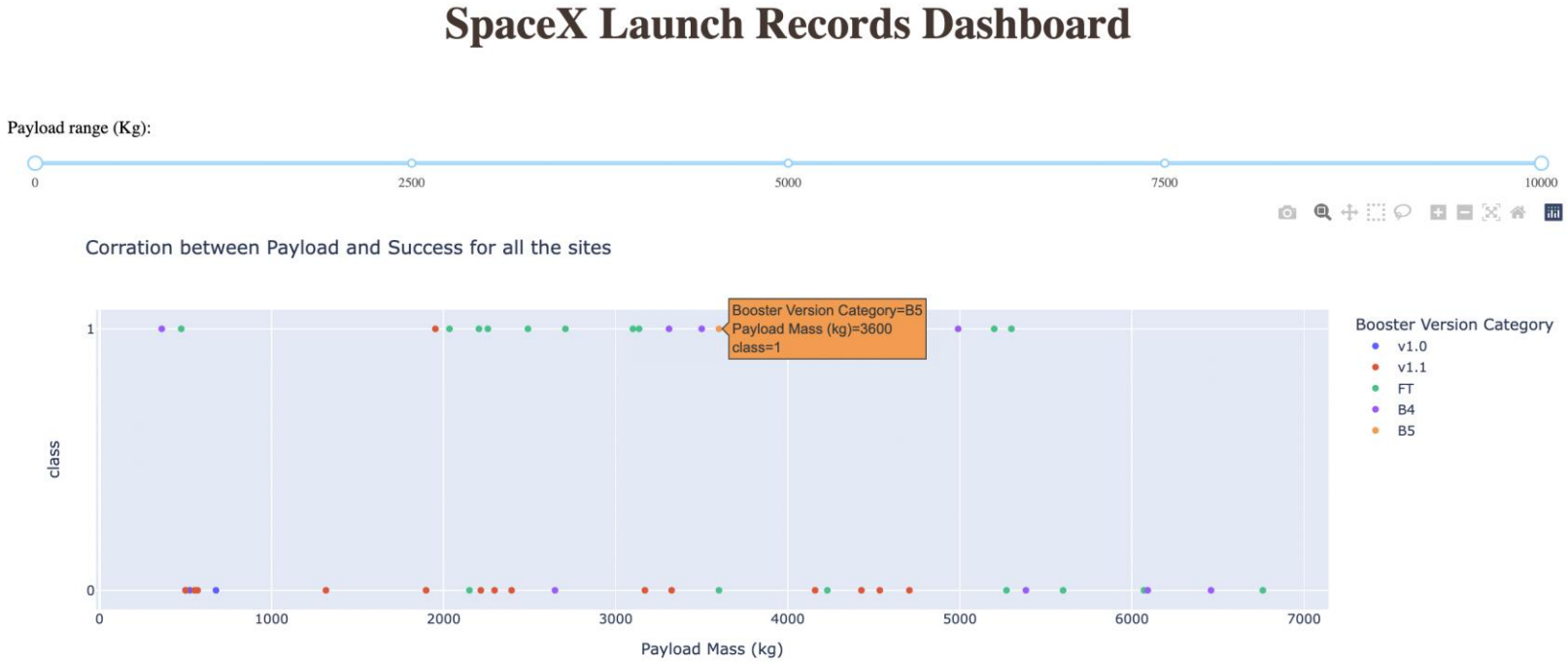




Booster	Launch Rate
V1.0	0.0%
V1.1	7.1%
FT	65%
B4	46%
B5	100%

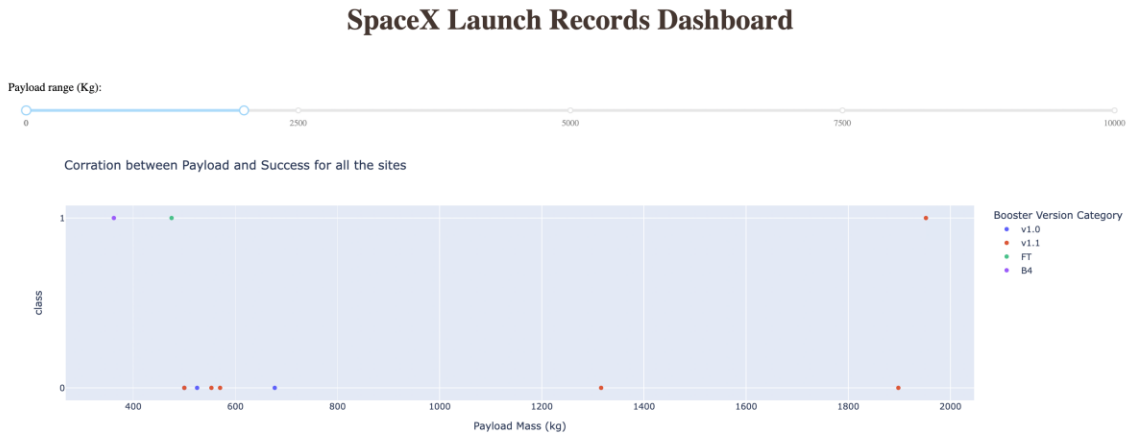
Booster FT and B5 have a promising launching rate for further development

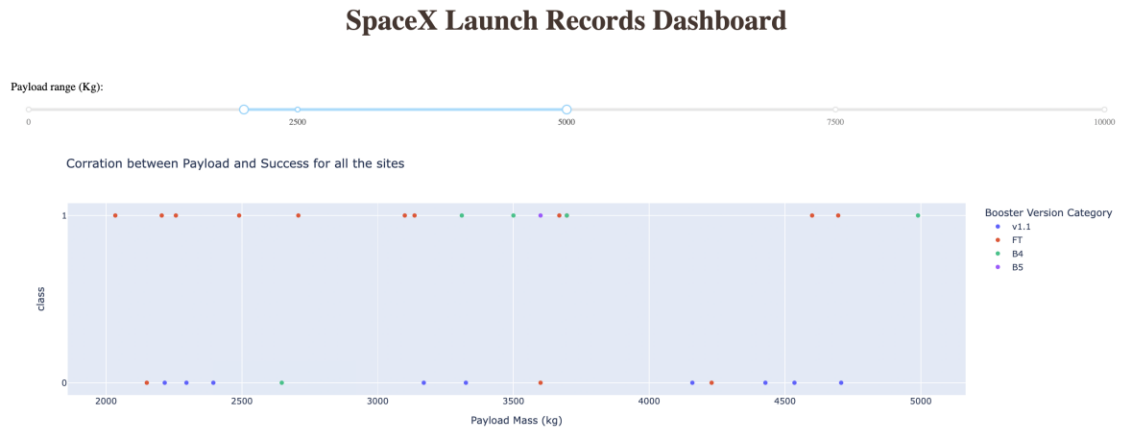


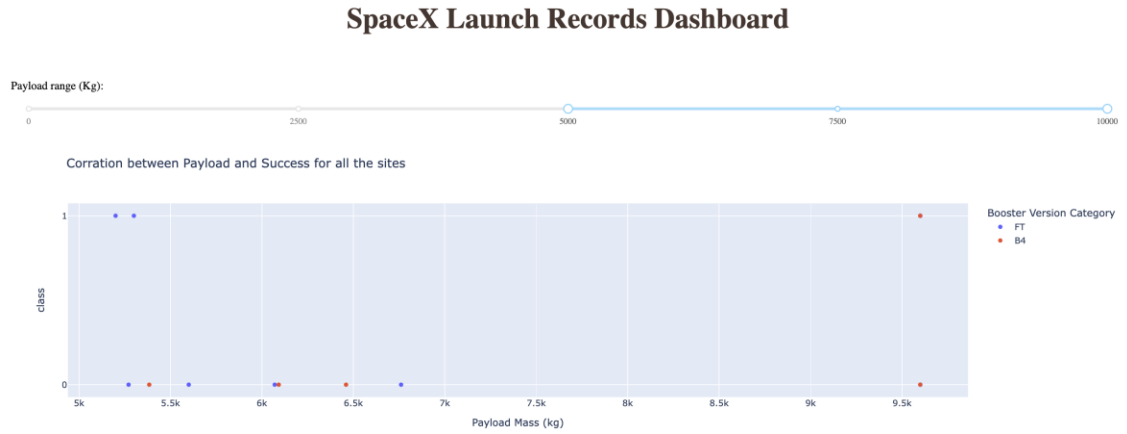


Payload Mass (kg) range	Launch Success Rate
0 to 2000	36.4%
2000 to 5000	53.6%
5000 to 10000	27.3%

Payload Mass range between 2000 to 5000 kg has the highest launch success rate







Predictive Analysis (Classification)

Load the dataframe

Load the data

[3]:
```
data = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_2.csv")

# If you were unable to complete the previous lab correctly you can uncomment and load this csv

# data = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DS0701EN-SkillsNetwork/api/dataset_part_2.csv')

data.head()
```

[3]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577366	28.561857	0
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366	28.561857	0
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.577366	28.561857	0
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	0	B1003	-120.610829	34.632093	0
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561857	0

[4]:
```
X = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_3.csv')

# If you were unable to complete the previous lab correctly you can uncomment and load this csv

# X = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DS0701EN-SkillsNetwork/api/dataset_part_3.csv')

X.head(100)
```

[4]:

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	...	Serial_B1058	Serial_B1059	Serial_B1060	Serial_B1062	GridFins_False	GridFins_True	Reused_False	Reused_True	Legs_False	Legs_True
0	1.0	6104.959412	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
1	2.0	525.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
2	3.0	677.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
3	4.0	500.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
4	5.0	3170.000000	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
85	86.0	15400.000000	2.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0
86	87.0	15400.000000	3.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0
87	88.0	15400.000000	6.0	5.0	5.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0
88	89.0	15400.000000	3.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0
89	90.0	3681.000000	1.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0

90 rows × 83 columns

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']).

[5]:
```
Y = data['Class'].values
Y[0:5]
```

[5]:
```
array([0, 0, 0, 0, 0])
```

TASK 2

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

[6]:
```
# students get this
transform = preprocessing.StandardScaler()
```

[7]:
```
X= preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

[7]:
```
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,
        -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
        -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.50755672e-01,
        -1.50755672e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -2.15665546e-01,
        -1.85695338e-01, -2.15665546e-01, -2.67261242e-01,
        -1.05999788e-01, -2.42535625e-01, -1.05999788e-01,
```

```
        -7.97724035e-01, -1.50755672e-01, -3.92232270e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-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, -1.50755672e-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,
        -1.05999788e-01, -1.50755672e-01, -1.50755672e-01,
        -1.50755672e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -2.15665546e-01,
        -1.85695338e-01, -2.15665546e-01, -2.67261242e-01,
        -1.05999788e-01, -2.42535625e-01, -1.05999788e-01,
        -2.15665546e-01, -1.85695338e-01, -2.15665546e-01,
        -1.85695338e-01, -1.05999788e-01,  1.87082869e+00,
        -1.87082869e+00,  8.35531692e-01, -8.35531692e-01,
         1.93309133e+00, -1.93309133e+00]])
```

Transforming data into an array format for further analysis

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`

[9]:
```
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 ('Validation set:', X_test.shape,  Y_test.shape)
```

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

we can see we only have 18 test samples.

[10]:
```
Y_test.shape
```

[10]:
```
(18,)
```

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`.

[11]:
```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

[12]:
```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=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_`.

[13]:
```
logreg_cv = GridSearchCV(lr, parameters, cv=10, verbose=3, n_jobs=-1).fit(X_train,Y_train)
```

```
Fitting 10 folds for each of 3 candidates, totalling 30 fits
```

[14]:
```
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
```

Hyperparameter are selected by using GridSearchCV function

TASK 5

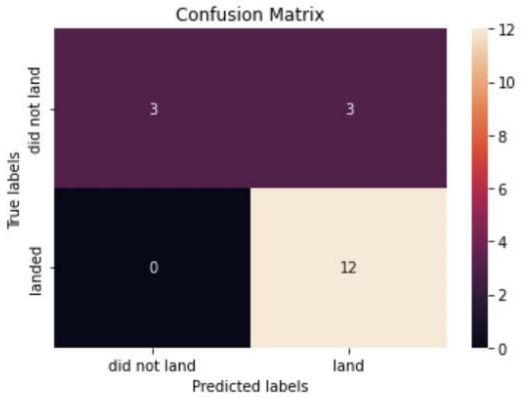
Calculate the accuracy on the test data using the method `score` :

[15]:
```
from sklearn.metrics import jaccard_score, f1_score
yhat1=logreg_cv.predict(X_test)
print("Avg F1-score:%.4f" % f1_score(Y_test, yhat1, average='weighted'))
print("Jaccard score: %.4f" % jaccard_score(Y_test, yhat1))
```

```
Avg F1-score:0.8148
Jaccard score: 0.8000
```

Lets look at the confusion matrix:

[16]:
```
plot_confusion_matrix(Y_test,yhat1)
```



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

[17]:
```
from sklearn.metrics import classification_report, confusion_matrix
import itertools
```

[18]:
```
print (classification_report(Y_test, yhat1))
```

```
              precision    recall  f1-score   support

           0       1.00      0.50      0.67         6
           1       0.80      1.00      0.89        12

    accuracy                           0.83        18
   macro avg       0.90      0.75      0.78        18
weighted avg       0.87      0.83      0.81        18
```

Logistic regression

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`.

[19]:
```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

[20]:
```
svm_cv = GridSearchCV(SVC(), parameters, cv=10, refit = True, verbose =3, n_jobs=-1).fit(X_train, Y_train)
```

```
Fitting 10 folds for each of 125 candidates, totalling 1250 fits
```

[21]:
```
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

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

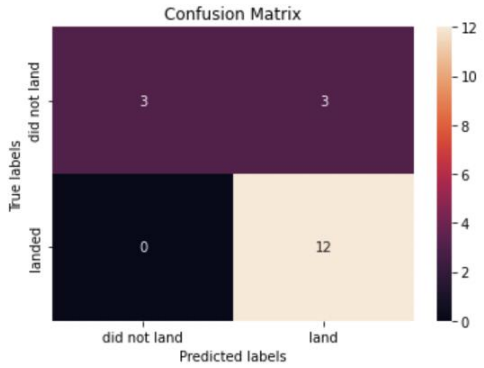
[22]:
```
from sklearn.metrics import jaccard_score, f1_score
yhat2=svm_cv.predict(X_test)

print("Avg F1-score:%.4f" % f1_score(Y_test, yhat2, average='weighted'))
print("Jaccard score: %.4f" % jaccard_score(Y_test, yhat2))
```

```
Avg F1-score:0.8148
Jaccard score: 0.8000
```

We can plot the confusion matrix

[23]:
```
yhat2=svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat2)
```



Support Vector Machine

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`.

[24]:
```
from sklearn.tree import DecisionTreeClassifier
import sklearn.tree as tree
```

[25]:
```
parameters = {'criterion': ['gini', 'entropy'],
     'splitter': ['best', 'random'],
     'max_depth': [2*n for n in range(1,10)],
     'max_features': ['sqrt','auto'],
     'min_samples_leaf': [1, 2, 4],
     'min_samples_split': [2, 5, 10]}
```

[26]:
```
tree_cv = GridSearchCV( DecisionTreeClassifier(), parameters, cv=10, refit = True, verbose =3).fit(X_train,Y_train)
```

```
Fitting 10 folds for each of 648 candidates, totalling 6480 fits
[CV 1/10] END criterion=gini, max_depth=2, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, splitter=best;, score=0.625 total time=   0.0s
[CV 2/10] END criterion=gini, max_depth=2, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, splitter=best;, score=0.625 total time=   0.0s
[CV 3/10] END criterion=gini, max_depth=2, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, splitter=best;, score=0.857 total time=   0.0s
[CV 4/10] END criterion=gini, max_depth=2, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, splitter=best;, score=0.714 total time=   0.0s
[CV 5/10] END criterion=gini, max_depth=2, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, splitter=best;, score=1.000 total time=   0.0s
[CV 6/10] END criterion=gini, max_depth=2, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, splitter=best;, score=0.857 total time=   0.0s
[CV 7/10] END criterion=gini, max_depth=2, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, splitter=best;, score=0.857 total time=   0.0s
[CV 8/10] END criterion=gini, max_depth=2, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, splitter=best;, score=0.857 total time=   0.0s
[CV 9/10] END criterion=gini, max_depth=2, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, splitter=best;, score=0.714 total time=   0.0s
[CV 1/10] END criterion=entropy, max_depth=18, max_features=auto, min_samples_leaf=4, min_samples_split=10, splitter=random;, score=1.000 total time=   0.0s
[CV 2/10] END criterion=entropy, max_depth=18, max_features=auto, min_samples_leaf=4, min_samples_split=10, splitter=random;, score=0.750 total time=   0.0s
[CV 3/10] END criterion=entropy, max_depth=18, max_features=auto, min_samples_leaf=4, min_samples_split=10, splitter=random;, score=0.714 total time=   0.0s
[CV 4/10] END criterion=entropy, max_depth=18, max_features=auto, min_samples_leaf=4, min_samples_split=10, splitter=random;, score=0.714 total time=   0.0s
[CV 5/10] END criterion=entropy, max_depth=18, max_features=auto, min_samples_leaf=4, min_samples_split=10, splitter=random;, score=1.000 total time=   0.0s
[CV 6/10] END criterion=entropy, max_depth=18, max_features=auto, min_samples_leaf=4, min_samples_split=10, splitter=random;, score=0.714 total time=   0.0s
[CV 7/10] END criterion=entropy, max_depth=18, max_features=auto, min_samples_leaf=4, min_samples_split=10, splitter=random;, score=0.857 total time=   0.0s
[CV 8/10] END criterion=entropy, max_depth=18, max_features=auto, min_samples_leaf=4, min_samples_split=10, splitter=random;, score=0.857 total time=   0.0s
[CV 9/10] END criterion=entropy, max_depth=18, max_features=auto, min_samples_leaf=4, min_samples_split=10, splitter=random;, score=0.857 total time=   0.0s
[CV 10/10] END criterion=entropy, max_depth=18, max_features=auto, min_samples_leaf=4, min_samples_split=10, splitter=random;, score=0.714 total time=   0.0s
```

[27]:
```
print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 8, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_samples_split': 2, 'splitter': 'random'}
accuracy : 0.9035714285714287
```

Decision Tree

TASK 9

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

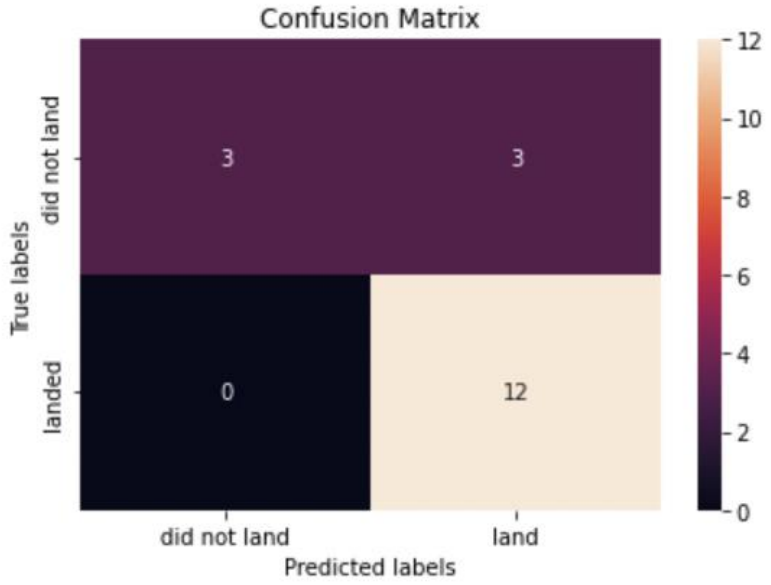
[32]:
```
from sklearn import metrics
from sklearn.metrics import accuracy_score
```

[33]:
```
predTree =tree_cv.predict(X_test)
print("DecisionTree's Accuracy: %.4f" % metrics.accuracy_score(Y_test, predTree))
```

```
DecisionTree's Accuracy: 0.8333
```

We can plot the confusion matrix

[34]:
```
yhat2 = svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat2)
```



Decision Tree

The "accuracy score" is 0.833

The "Best Score" of best fit parameter is 0.889

KNN k-nearest Neighbours Classification

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`.

[36]:
```
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()
```

[39]:
```
knn_cv = GridSearchCV(estimator=KNN, param_grid=parameters, cv=10, verbose=1).fit(X_train, Y_train)
```

```
Fitting 10 folds for each of 80 candidates, totalling 800 fits
```

[40]:
```
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
```

[41]:
```
from sklearn.metrics import jaccard_score, f1_score
yhat4 = knn_cv.predict(X_test)
print("Avg F1-score:%.4f" % f1_score(Y_test, yhat4, average='weighted'))
print("Jaccard score: %.4f" % jaccard_score(Y_test, yhat4))
```

```
Avg F1-score:0.8148
Jaccard score: 0.8000
```

TASK 11

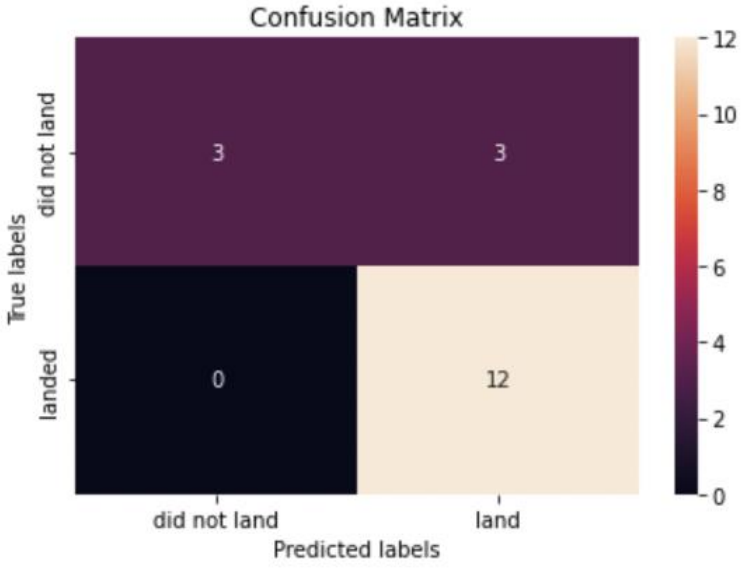
Calculate the accuracy of tree_cv on the test data using the method `score`:

[42]:
```
from sklearn.metrics import jaccard_score, f1_score
yhat3 = tree_cv.predict(X_test)
print("Avg F1-score:%.4f" % f1_score(Y_test, yhat3, average='weighted'))
print("Jaccard score: %.4f" % jaccard_score(Y_test, yhat3))
```

```
Avg F1-score:0.8148
Jaccard score: 0.8000
```

We can plot the confusion matrix

[53]:
```
yhat4 = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test, yhat4)
```



All three classification models, Logistic Progress, Decision tree, and KNN have the same evaluating performance , but the decision tree has a better fit over the specified parameters. Therefore the Decision Tree Model is the best among other models.

The decision tree (classification) model has the best accuracy

TASK 12

Find the method performs best:

BEST SCORE

```
[36]: B1 = logreg_cv.best_score_
      B2 = svm_cv.best_score_
      B3 = tree_cv.best_score_
      B4 = knn_cv.best_score_

      Best_Score = [B1, B2, B3, B4]
      Best_Score
```

```
[36]: [0.8464285714285713,
       0.8482142857142856,
       0.8892857142857142,
       0.8482142857142858]
```

ACCURACY

```
[37]: A1 = metrics.accuracy_score(Y_test, yhat1)
      A2 = metrics.accuracy_score(Y_test, yhat2)
      A3 = metrics.accuracy_score(Y_test, yhat3)
      A4 = metrics.accuracy_score(Y_test, yhat4)

      Accuracy = [A1, A2, A3, A4]
      Accuracy
```

```
[37]: [0.8333333333333334,
       0.8333333333333334,
       0.8333333333333334,
       0.8333333333333334]
```

JACCARDS

```
[38]: J1 = jaccard_score(Y_test, yhat1)
      J2 = jaccard_score(Y_test, yhat2)
      J3 = jaccard_score(Y_test, yhat3)
      J4 = jaccard_score(Y_test, yhat4)

      Jaccards = [J1, J2, J3, J4]
      Jaccards
```

```
[38]: [0.8, 0.8, 0.8, 0.8]
```

F1 SCORES

```
[43]: F1 = f1_score(Y_test, yhat1, average='weighted')
      F2 = f1_score(Y_test, yhat2, average='weighted')
      F3 = f1_score(Y_test, yhat3, average='weighted')
      F4 = f1_score(Y_test, yhat4, average='weighted')

      F1_scores = [F1, F2, F3, F4]
      F1_scores
```

```
[43]: [0.8148148148148149,
       0.8148148148148149,
       0.8148148148148149,
       0.8148148148148149]
```

```
[42]: from sklearn.metrics import log_loss
      LR_pred =logreg_cv.predict_proba(X_test)
      L = log_loss(Y_test, LR_pred)
      Log_loss = [L, 'NA', 'NA', 'NA']
      Log_loss
```

```
[42]: [0.4786666968559154, 'NA', 'NA', 'NA']
```

```
[44]: import pandas as pd

      data = [Best_Score, Accuracy, Jaccards, F1_scores, Log_loss]
      data = np.array(data).T

      model_df = pd.DataFrame(data, index = ["Logistic Regression", "SVM", "Decision tree", "KNN"],
                                    columns = ["Best_Score","Accuracy","Jaccards","F1-scores","Log_Loss"])

      print(model_df)
```

```
                     Best_Score           Accuracy Jaccards  \
Logistic Regression  0.8464285714285713  0.8333333333333334      0.8
SVM                  0.8482142857142856  0.8333333333333334      0.8
Decision tree        0.8892857142857142  0.8333333333333334      0.8
KNN                  0.8482142857142858  0.8333333333333334      0.8

                     F1-scores           Log_Loss
Logistic Regression  0.8148148148148149  0.4786666968559154
SVM                  0.8148148148148149                  NA
Decision tree        0.8148148148148149                  NA
KNN                  0.8148148148148149                  NA
```

Discussion and Conclusion

In this project, the prediction for the possibility of successful reusable launch system by using data science methods has been proven reliable and viable. The high accuracy from the predictive analysis (classification) is completely agreed with the present SpaceX's space development. After all analyses, the Falcon 9 B5 is a reliable booster, Falcon 9 has the highest success rate from the Kennedy Space Center Launch Complex 39A (KSC LC39A), which is the best platform, and the Reusable Launch System is feasible.

The modern space technology and engineering turns science fiction into science facts. The successful partially reusable launch system reduces the cost of each launch from 165 millions to 65 millions. Because of this affordable reusable launch system , SpaceX only charges 50 million USD per seat based on a five seat package deal, which is more less expensive than Boeing's 90 million USD per seat for transportation from earth orbits to International Space Station (ISS). SpaceX's software is also superior to Boeing's software, which had a miscalculation of orbit. SpaceX won the contract to send the crew to the Internation Space Station (ISS) under the NASA(CRS) program over Boeing . Therefore NASA does not need to depend on Russian rockets to transport crews to the International Space Station (ISS). A more affordable launching cost would encourage a lot of space ambitions in the future. The human race has a chance to become a multiplanetary species. Satellite projects from SpaceX, Blue Region, China National Space Administration and the Australian Space Agency will over congest the VLEO (very low earth orbit). The space race for military satellite, space mining, land settlements, and the international space treaties also become a challenge.

Appendix

ORBITS

ES-L1	European Space Largrangian 1
GEO	Geostationary Orbit
GSO	Geosynchronous Orbits
GTO	Geosynchronous Transfer Orbit
HEO	Highly Elliptical Orbit
ISS	International Space Station
LEO	Low Earth Orbit
MEO	Medium Earth Orbit
PO	Polar
SSO	Sun-Synchronous Orbit (SSO)
VLEO	Very Low Earth Orbit

SPACE LAUNCH COMPLEX

CCAFS LC-40	Cape Canaveral Air Force Space Launch Complex 40
CCAFS SLC-40	Cape Canaveral Air Force Space SpaceX Launch Complex 40
KSC LC-39A	Kennedy Space Center Launch Complex 39A
VAFB SLC-4E	Vandenberg AFB Space Launch Complex 4E

COMPUTER JARGON

API	Application Programming Interface
KNN	K-Nearest Neighbours
REST	Representational State Transfer
SVM	Support Vector Machine