SPACE-X PROJECT REPORT

SUCCESS RATE PREDICTION OF FIRST STAGE LANDING

OUTLINE

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

EXECUTIVE SUMMARY

- Summary of methodologies
  - Data Collection through API
  - Data Collection with Web Scraping
  - Data Wrangling
  - Exploratory Data Analysis with SQL
  - Exploratory Data Analysis with Data Visualization
  - Interactive Visual Analytics with Folium
  - Machine Learning Prediction
- Summary of all results
  - Exploratory Data Analysis result
  - Interactive analytics in screenshots
  - Predictive Analytics result

INTRODUCTION

- Project background:

  Space X advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is because Space X can reuse the first stage. Therefore, if we can determine if the first stage will land, we can determine the cost of a launch. This information can be used if an alternate company wants to bid against space X for a rocket launch. This goal of the project is to create a machine learning pipeline to predict if the first stage will land successfully.

- Problems you want to find answers:

  1) What factors determine if the rocket will land successfully?

  2) The interaction amongst various features that determine the success rate of a successful landing.

  3) What operating conditions needs to be in place to ensure a successful landing program

METHODOLOGY

METHODOLOGY

- Data collection methodology:
  - Data was collected using SpaceX API and web scraping from Wikipedia.
- Perform data wrangling:
  - One-hot encoding was applied to categorical features
  - Cleared the data set by removing NaN values and changed all object format into numerical format.
- Perform exploratory data analysis (EDA) using SQL and retrieved the meaningful data
- Performed Geo Visualization's tools using Folium and Plotly Dash for better understanding of distance from proximities like highways, railway tracks, cities and sea.
- Perform predictive analysis using classification models:
  - How to build, train, fit and predict classification models like (KNN, SVM, Decision Tree and Logistic regression) by using GridSearchCV method.

DATA COLLECTION

- The data was collected using various methods
    - Data collection was done using get request to the SpaceX API.
    - Next, we decoded the response content as a Json using .json() function call and turn it into a pandas dataframe using .json_normalize().
    - We then cleaned the data, checked for missing values and fill in missing values where necessary.
    - In addition, we performed web scraping from Wikipedia for Falcon 9 launch records with BeautifulSoup.
    - The objective was to extract the launch records as HTML table, parse the table and convert it to a pandas dataframe for future analysis.

DATA COLLECTION – SPACEX API

1) We used the get request to the SpaceX API to collect data.

2) Cleaned the requested data and did some basic data wrangling and formatting.

3) The link to the notebook is:

https://github.com/saketh0408/Data-Sciene-Capston-Project/blob/97db6d11daecb8a0f35b1a625127e5262b562e2d/SpaceX_jupyter_data_collection_api.ipynb.

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

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

In [9]:
```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_c
```

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

In [10]:
```
response.status_code
```

Out[10]: 200

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

In [14]:
```
# Use json_normalize meethod to convert the json result into a dataframe
data=pd.json_normalize(response.json())
```

Using the dataframe `data` print the first 5 rows

In [15]:
```
# Get the head of the dataframe
data.head()
```

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`.

In [60]:
```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9=data[data.BoosterVersion=='Falcon 9']
data_falcon9.head()
```

Out[60]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitu
4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0003	-80.577366	28.561
5	8	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0005	-80.577366	28.561
6	10	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0007	-80.577366	28.561
7	11	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None	1.0	0	B1003	-120.610829	34.632
8	12	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B1004	-80.577366	28.561

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

Task 3: Dealing with Missing Values

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

In [74]:
```
# Calculate the mean value of PayloadMass column
payloadmass_mean=data_falcon9['PayloadMass'].mean()
# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'].replace(np.nan,payloadmass_mean,inplace=True)
data_falcon9.isnull().sum()
data_falcon9.to_csv('dataset_part_1.csv', index=False)
```

Out[73]:
```
FlightNumber       0
Date               0
BoosterVersion     0
PayloadMass        0
Orbit              0
LaunchSite         0
Outcome            0
Flights            0
GridFins           0
Reused             0
Legs               0
LandingPad        26
Block              0
ReusedCount        0
Serial             0
Longitude          0
Latitude           0
dtype: int64
```

DATA COLLECTION - SCRAPING

1) We applied web scrapping to webscrap information about Falcon 9 launch records from HTML pages with BeautifulSoup library.

2) We parsed the HTML table and converted it into a pandas dataframe, which will be useful for our analysis.

3) The link to the notebook is:

https://github.com/saketh0408/Data-Sciene-Capston-Project/blob/d7faf4ed754aca346aee28dc1f59e0e360669175/SpaceX_jupyter_webscraping.ipynb

TASK 1: Request the Falcon9 Launch Wiki page from its URL

First, let's perform an HTTP GET method to request the Falcon9 Launch HTML page, as an HTTP response.

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

Out[5]: 200
```

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

```
In [6]:
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup=BeautifulSoup(response.text,'html5lib')
```

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

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

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

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

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

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

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

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

```
In [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
element = soup.find_all('th')
for row in range(len(element)):
    try:
        name = extract_column_from_header(element[row])
        if (name is not None and len(name) > 0):
            column_names.append(name)
    except:
        pass
```

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

We will create an empty dictionary with keys from the extracted column names in the previous task. Later, this dictionary will be converted into a Pandas dataframe

```
In [20]:
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"):
        print("################################################################################")
        #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)
            print(flight_number)
            datatimelist=date_time(row[0])

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

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

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

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

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

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

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

# Customer
# TODO: Append the customer into launch_dict with key `Customer`
launch_dict['Customer'].append(customer)
customer = row[6].a
#print(customer)
print(customer)


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

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

DATA WRANGLING

1)We performed exploratory data analysis and determined the training labels.

2)We calculated the number of launches at each site, and the number and occurrence of each orbits.

3)We created landing outcome label from outcome column and exported the results to csv.

4)The link to the notebook is:
https://github.com/saketh0408/Data-Sciene-Capston-Project/blob/d7faf4ed754aca346aee28dc1f59e0e360669175/SpaceX_jupyter_Data%20wrangling.ipynb

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

Out[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`

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

Out[6]:
```
GTO      27
ISS      21
VLEO     14
PO        9
LEO       7
SSO       5
MEO       3
ES-L1     1
HEO       1
SO        1
GEO       1
Name: Orbit, dtype: int64
```

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

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

In [7]:
```
# landing_outcomes = values on Outcome column
landing_outcomes=df['Outcome'].value_counts()
landing_outcomes
```

Out[7]:
```
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.

In [8]:
```
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 [9]:
```
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
bad_outcomes
```

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 [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)
```

In [13]:
```
df["Class"].mean()
```

Out[13]:
```
0.6666666666666666
```

EDA WITH DATA VISUALIZATION

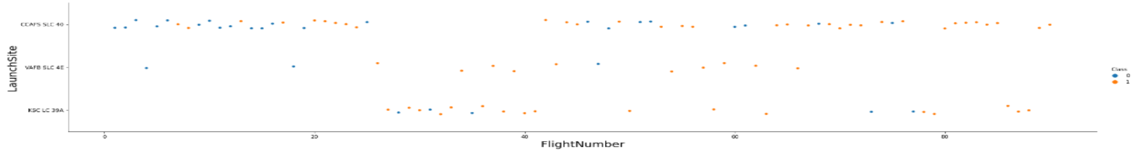
1)We explored the data by visualizing the relationship between flight number and launch Site, payload and launch site, success rate of each orbit type, flight number and orbit type, the launch success yearly.

2)The link to the notebook is:
https://github.com/saketh0408/Data-Sciene-Capston-Project/blob/d7faf4ed754aca346aee28dc1f59e0e360669175/SpaceX_jupyter_dataviz.ipynb

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'`

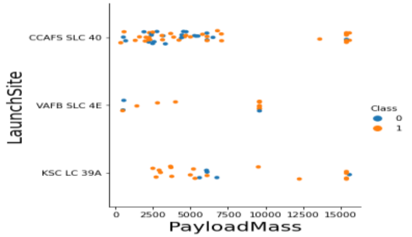
In [12]:
```
# Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to be the class value
sns.catplot(y="LaunchSite",x="FlightNumber",hue="Class",data=df,aspect=5)
plt.xlabel("FlightNumber",fontsize=20)
plt.ylabel("LaunchSite",fontsize=20)
plt.show()
```



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 [15]:
```
# 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(x="PayloadMass",y="LaunchSite",hue="Class",data=df)
plt.xlabel("PayloadMass",fontsize=20)
plt.ylabel("LaunchSite",fontsize=20)
plt.show()
```



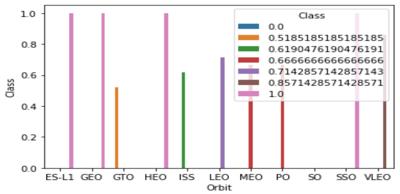
TASK 3: Visualize the relationship between success rate of each orbit type

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 [24]:
```
# HINT use groupby method on Orbit column and get the mean of Class column
orbit_s=df.groupby('Orbit').mean()
orbit_s.reset_index(inplace=True)
sns.barplot(x="Orbit",y="Class",data=orbit_s,hue='Class')
```

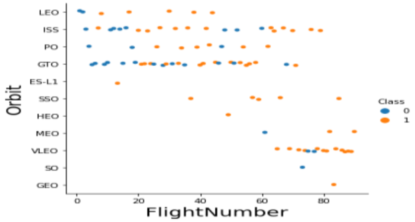
Out[24]: <AxesSubplot:xlabel='Orbit', ylabel='Class'>



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.

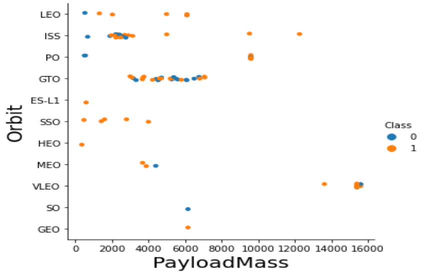
In [25]:
```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class value
sns.catplot(x="FlightNumber",y="Orbit",data=df,hue="Class")
plt.xlabel("FlightNumber",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```



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 [26]:
```
# 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(x="PayloadMass",y="Orbit",data=df,hue="Class")
plt.xlabel("PayloadMass",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```



TASK 6: Visualize the launch success yearly trend

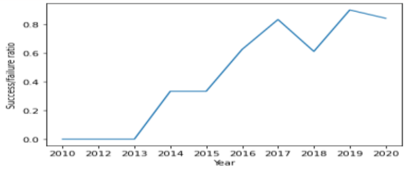
You can plot a line chart with x axis to be `Year` and y axis to be average success rate, to get the average launch success trend.

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

In [30]:
```
# A function to Extract years from the date
year=[]
def Extract_year(date):
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year
Extract_year(1)
df["Year"]=year
average_by_year = df.groupby("Year").mean()
average_by_year.reset_index(inplace=True)
```

In [31]:
```
# Plot a line chart with x axis to be the extracted year and y axis to be the success rate
sns.lineplot(average_by_year["Year"],average_by_year["Class"])
plt.xlabel("Year")
plt.ylabel("Success/failure ratio")
plt.show()
```

```
C:\Users\saket\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



EDA WITH SQL

1) We loaded the SpaceX dataset into a Sqlite3 database with using the assistance of jupyter notebook.

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

- The names of unique launch sites in the space mission.
- The total payload mass carried by boosters launched by NASA (CRS)
- The average payload mass carried by booster version F9 v1.1
- The total number of successful and failure mission outcomes
- The failed landing outcomes in drone ship, their booster version and launch site names.

3) The link to the notebook is: https://github.com/saketh0408/Data-Sciene-Capston-Project/blob/55ba65e97237280cb96a75de3f19e9cfb74a4841/SpaceX_jupyter-labs-eda-sql.ipynb

BUILD AN INTERACTIVE MAP WITH FOLIUM LIBRARY

1) We marked all launch sites, and added map objects such as markers, circles, lines to mark the success or failure of launches for each site on the folium map.

2)We assigned the feature launch outcomes (failure or success) to class 0 and 1.i.e., 0 for failure, and 1 for success.

3)Using the color-labeled marker clusters, we identified which launch sites have relatively high success rate.

4)We calculated the distances between a launch site to its proximities. We answered some question for instance:

- Are launch sites near railways, highways and coastlines.
- Do launch sites keep certain distance away from cities.

5) The link to the notebook is: https://github.com/saketh0408/Data-Sciene-Capston-Project/blob/d7faf4ed754aca346aee28dc1f59e0e360669175/SpaceX_jupyter_Folium_launch_site_location.ipynb

BUILD A DASHBOARD WITH PLOTLY DASH

1) We built an interactive dashboard with Plotly dash.

2) We plotted success pie charts showing the total number of launches from certain sites and its success rate.

3) We plotted scatter graph showing the relationship with Outcome and Payload Mass (Kg) for the different booster version.

4) The link to the notebook is: https://github.com/saketh0408/Data-Sciene-Capston-Project/blob/d7faf4ed754aca346aee28dc1f59e0e360669175/SpaceX_jupyter_dash_board.py

PREDICTIVE ANALYSIS (CLASSIFICATION MODELS)

1) We loaded the data using numpy and pandas, transformed the data, split our data into training and testing.

2)We built different machine learning models like Logistic Regression, KNN, Decision-Tree and SVM , then tuned hyperparameters using GridSearchCV and ploted confusion Matrix plots.

3)We used accuracy as the metric for our model, improved the model using feature engineering and algorithm tuning.

4)We found the best performing classification model.

5) The link to the notebook is: https://github.com/saketh0408/Data-Sciene-Capston-Project/blob/d7faf4ed754aca346aee28dc1f59e0e360669175/SpaceX_Machine%20Learning%20Prediction.ipynb

RESULTS

1) Exploratory data analysis results.

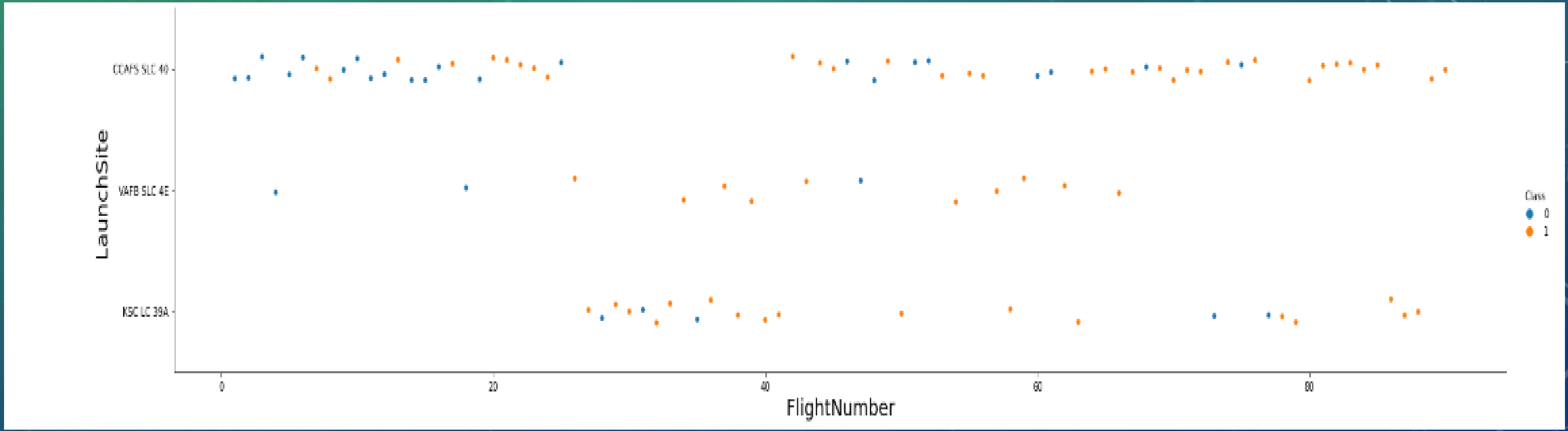
2) Interactive analytics demo in screenshots.

3) Predictive analysis result.s

INSIGHTS DRAWN FROM EXPLORATORY DATA ANALYSIS

FLIGHT NUMBER VS. LAUNCH SITE

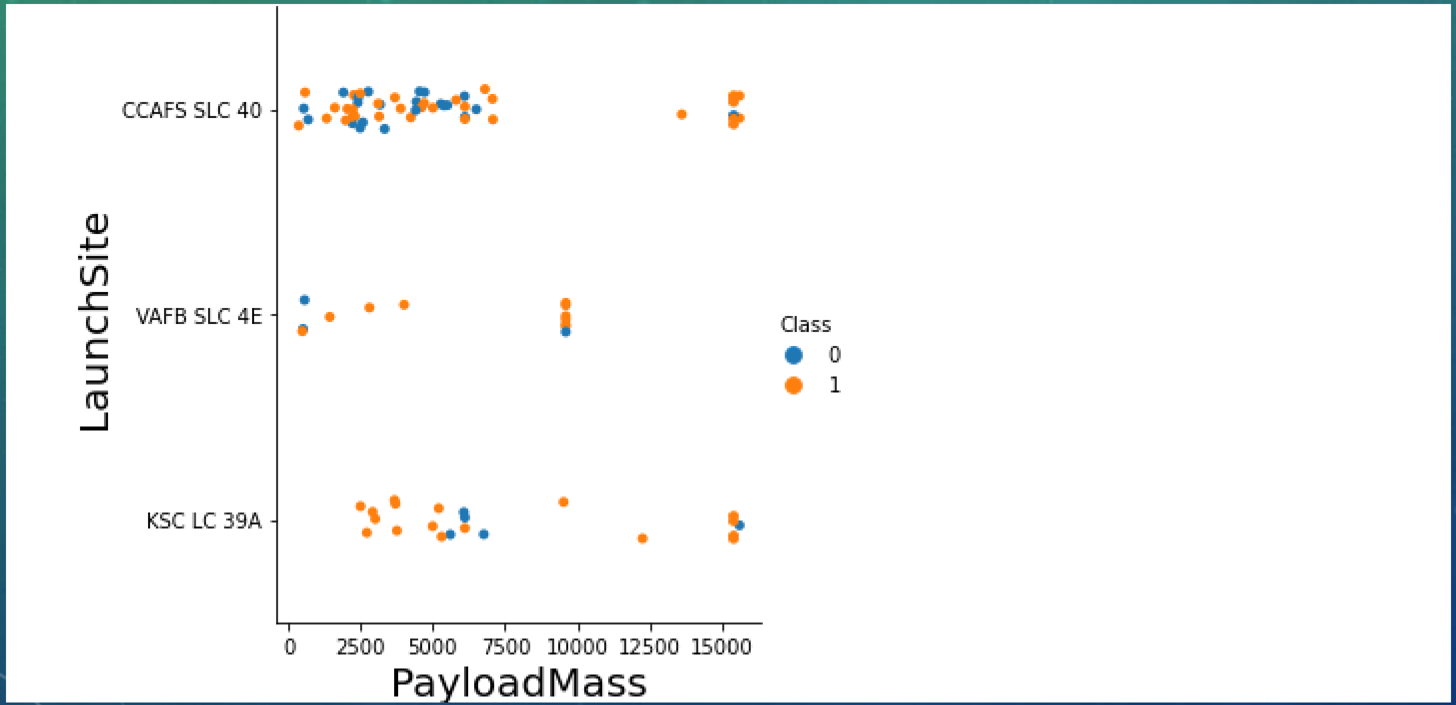
From the below plot we can find out that highest number of flights from earth to space has been launched from launch site that is CCAPS SLC-40.



PAYLOAD VS. LAUNCH SITE

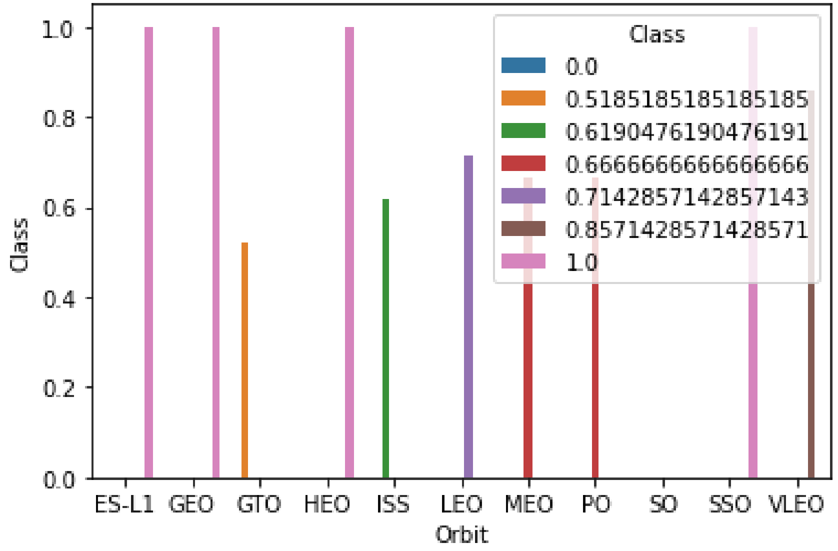
From the below plot we can find out that, large quantity of Payload mass for rockets falls under category 2500 to 8000 and has been launched from launch site CCAFS SLC-40.

And payload mass for rockets grater than 12500 has highest success rate.



SUCCESS RATE VS. ORBIT TYPE

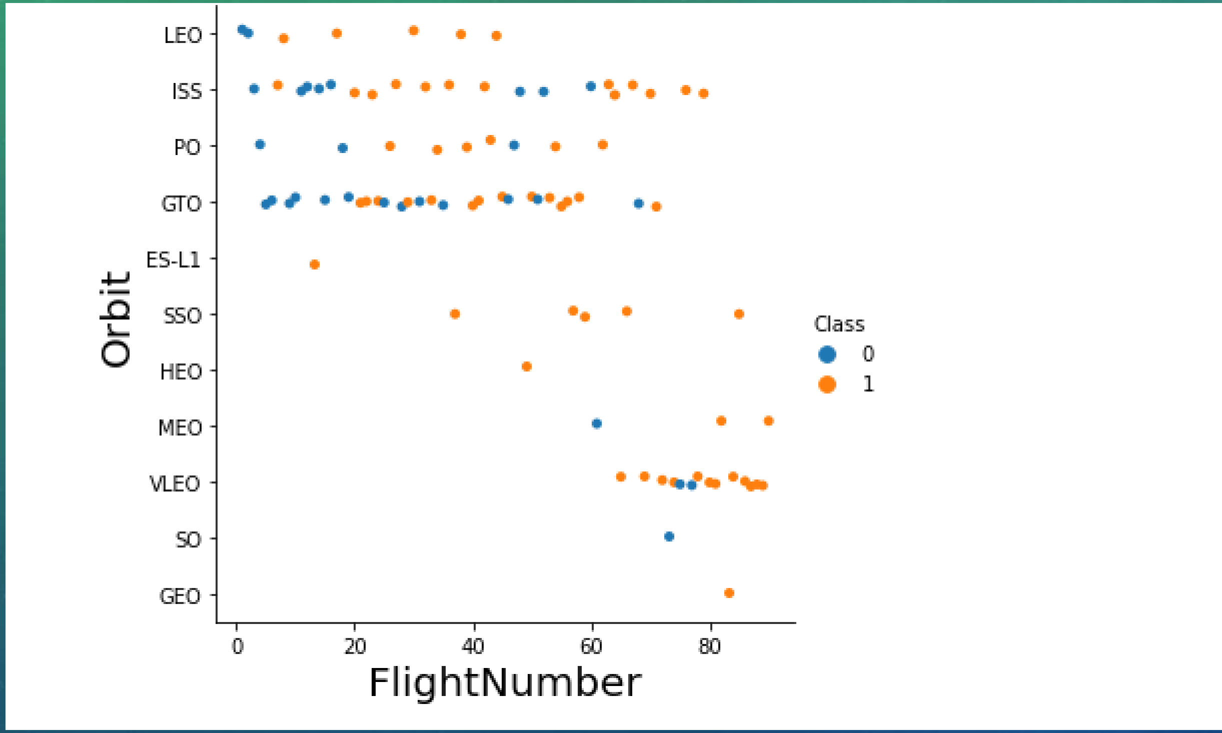
From the below plot we can find out that, flights launched into these orbits ES-L1, GEO, HEO, SSO, VLEO have the most success rate of 1.0. The flights launched into VLEO orbit has a success rate of about 0.8.



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

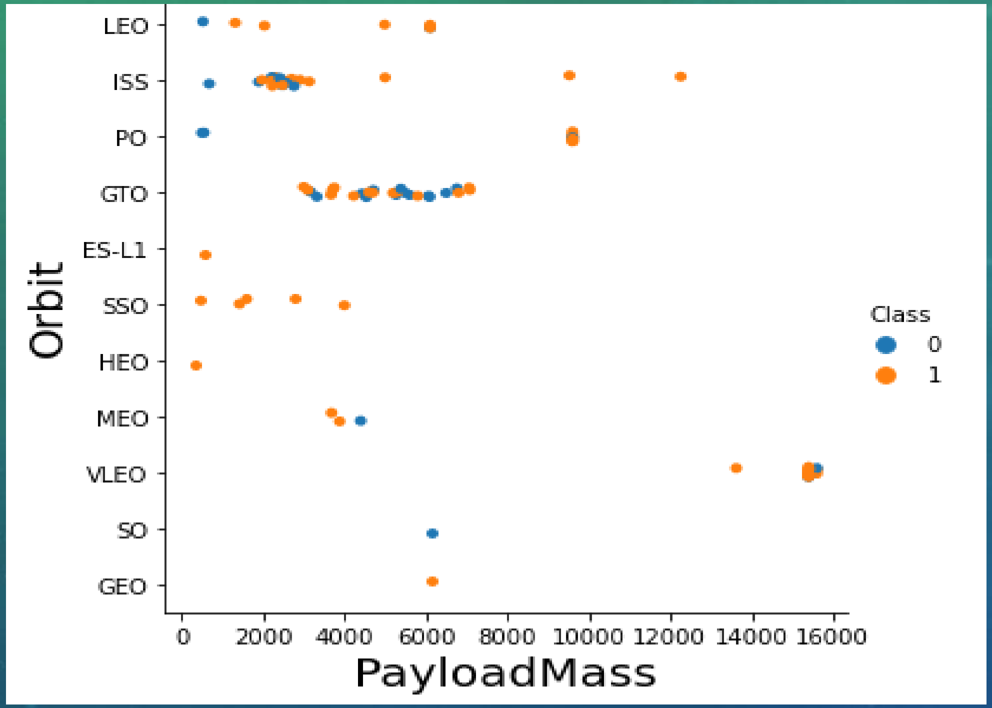
FLIGHT NUMBER VS. ORBIT TYPE

The plot below shows the Flight Number vs. Orbit type. We observe that in the LEO orbit, the number of flights that was launched is successful.



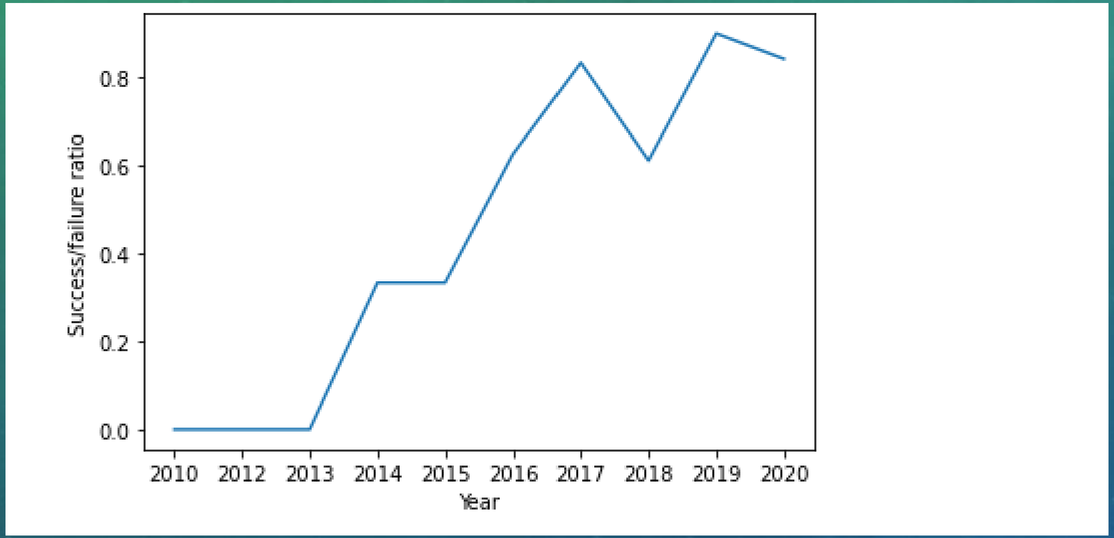
PAYLOAD MASS VS ORBIT TYPE

The plot below shows, We observe that in the LEO , SSO and ISS orbit, has successful landing record of rockets.



SUCCESS YEARLY RECORD

From 2013, the success rate of 1st stage landing of spaceX rocket is increasing drastically and continuous to do also after our analysis.



ALL LAUNCH SITE NAMES

- We used the key word **DISTINCT** to show only unique launch sites from the SpaceX data.

Task 1

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

In [12]:
```
%sql select distinct Launch_Site from SPACEXTBL;
```

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

Out[12]:

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

LAUNCH SITE NAMES BEGINS WITH 'CCA'

- 5 records of launch sites starting with 'CCA'.

Task 2

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

[13]:

```
%sql select * from SPACEXTBL where Launch_Site like 'CCA%'Limit 5;
```

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

[13]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	LandingOutcome
4/6/2010	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
8/12/2010	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)
5/22/2012	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
8/10/2012	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
1/3/2013	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

TOTAL PAYLOAD MASS

- We calculated the total payload carried by boosters from NASA as 45596 using the query below

Task 3

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

In [14]:

```
%sql select sum(PAYLOAD_MASS__KG_) as Total_PayLoad_Mass from SPACEXTBL where Customer=='NASA (CRS)';
```

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

Out[14]:

Total_PayLoad_Mass
45596

AVERAGE PAYLOAD MASS BY F9 V 1.1

- We calculated the average payload mass carried by booster version F9 v1.1 as 2928.4

Task 4

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

In [15]:
```
%sql select Avg(PAYLOAD_MASS__KG_) as Avg_PayLoad_Mass from SPACEXTBL where Booster_Version=='F9 v1.1';
```

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

Out[15]:

Avg_PayLoad_Mass
2928.4

FIRST SUCCESSFUL GROUND LANDING DATE

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

Task 5

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

Hint:Use min function

In [16]:

```
%sql SELECT min(Date) AS FirstSuccessfull_landing_date FROM SPACEXTBL WHERE LandingOutcome LIKE 'Success (ground pad)'
```

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

Out[16]:

FirstSuccessfull_landing_date
12/22/2015

SUCCESSFUL DRONE SHIP LANDING WITH PAYLOAD BETWEEN 4000 AND 6000

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

Task 6

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

In [17]:

```
%sql select Booster_Version from SPACEXTBL where "LandingOutcome"=='Success (drone ship)' and PAYLOAD_MASS__KG_ between 4000 and 6000;
```

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

Out[17]:

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

TOTAL NUMBER OF SUCCESSFUL AND FAILURE MISSION OUTCOMES

- Total Mission_Outcome of both success or a failure mission.

Task 7

List the total number of successful and failure mission outcomes

In [18]:
```
%sql select Mission_Outcome,count(Mission_Outcome) as count from SPACEXTBL group by Mission_Outcome;
```

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

Out[18]:

Mission_Outcome	count
Failure (in flight)	1
Success	98
Success	1
Success (payload status unclear)	1

BOOSTERS CARRIED MAXIMUM PAYLOAD

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

Task 8

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

In [19]:
```
%sql select Booster_Version from SPACEXTBL where PAYLOAD_MASS__KG_ is (select max(PAYLOAD_MASS__KG_) from SPACEXTBL);
```

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

Out[19]:

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

2015 LAUNCH RECORDS

- We used a combinations of the WHERE clause, LIKE, AND, and BETWEEN conditions to filter for failed landing outcomes in drone ship, their booster versions, and launch site names for year 2015

Task 9

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

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

In [24]:

```
%sql select booster_version, launch_site from SPACEXTBL \
where "LandingOutcome" == 'Failure (drone ship)' and substr(Date,7,4) BETWEEN '01-01-2015' AND '31-12-2015' Limit 2 ;
```

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

Out[24]:

Booster_Version	Launch_Site
F9 v1.1 B1012	CCAFS LC-40
F9 v1.1 B1015	CCAFS LC-40

RANK LANDING OUTCOMES BETWEEN 2010-06-04 AND 2017-03-20

- We selected Landing outcomes and the COUNT of landing outcomes from the data and used the WHERE clause to filter for landing outcomes BETWEEN 2010-06-04 to 2010-03-20.
- We applied the GROUP BY clause to group the landing outcomes and the ORDER BY clause to order the grouped landing outcome in descending order.

Task 10

Rank the count of successful landing_outcomes between the date 04-06-2010 and 20-03-2017 in descending order.

In [21]:

```
%sql select count("LandingOutcome"), "LandingOutcome" from SPACEXTBL \
where DATE between '04-06-2010' and '20-03-2017'\
group by "LandingOutcome"\
order by count("LandingOutcome") desc Limit 4;
```

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

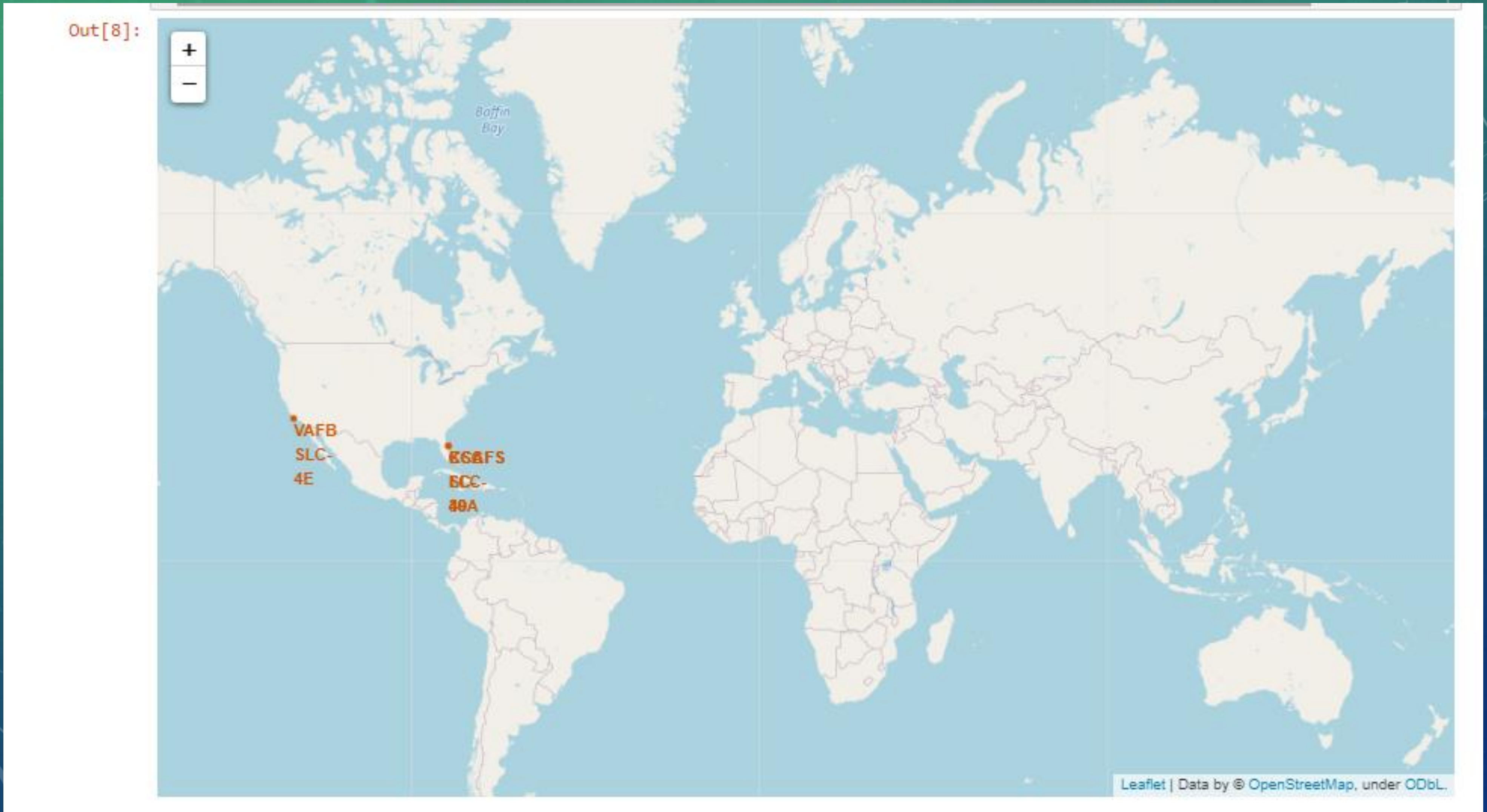
Out[21]:

count("LandingOutcome")	LandingOutcome
14	Success
6	No attempt
4	Success (ground pad)
4	Success (drone ship)

INSIGHTS DRAWN FROM LAUNCH SITES PROXIMITIES ANALYSIS

ALL LAUNCH SITES GLOBAL MAP MARKERS

The location of launch sites in United States of America: 1) Claifornia 2) Florida

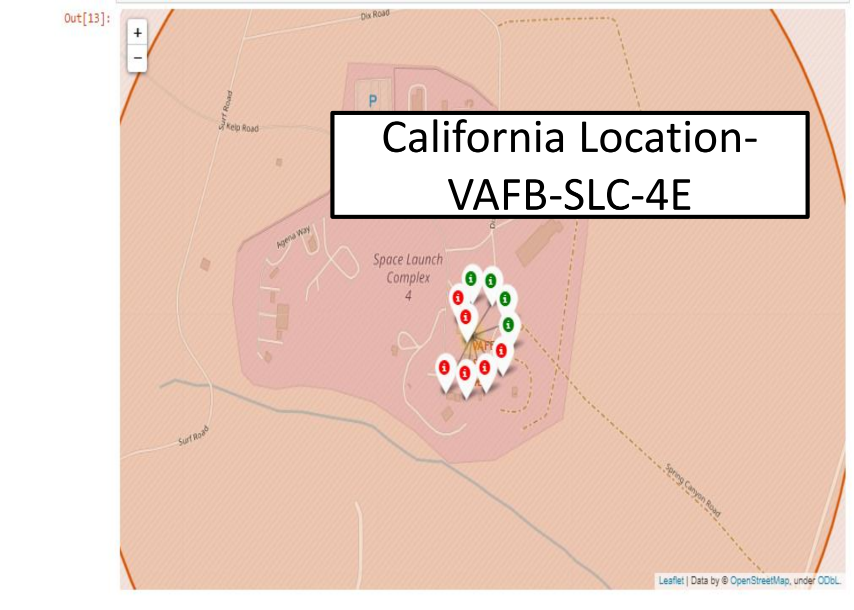


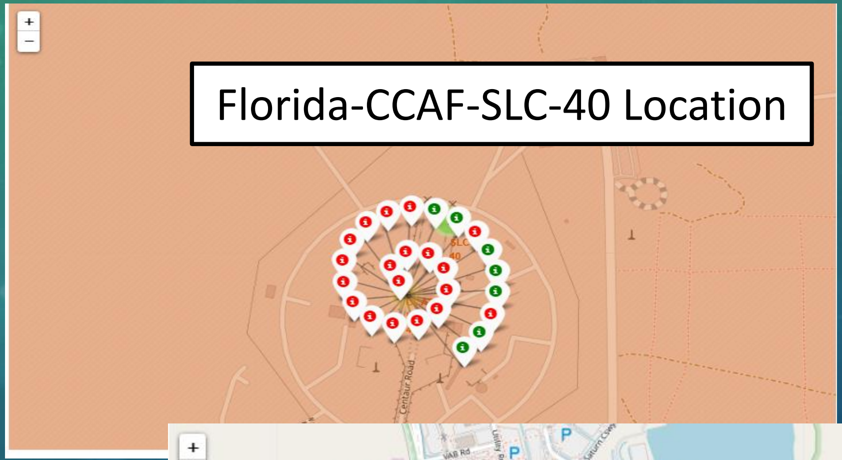
MARKERS SHOWING LAUNCH SITES WITH COLOR LABELS

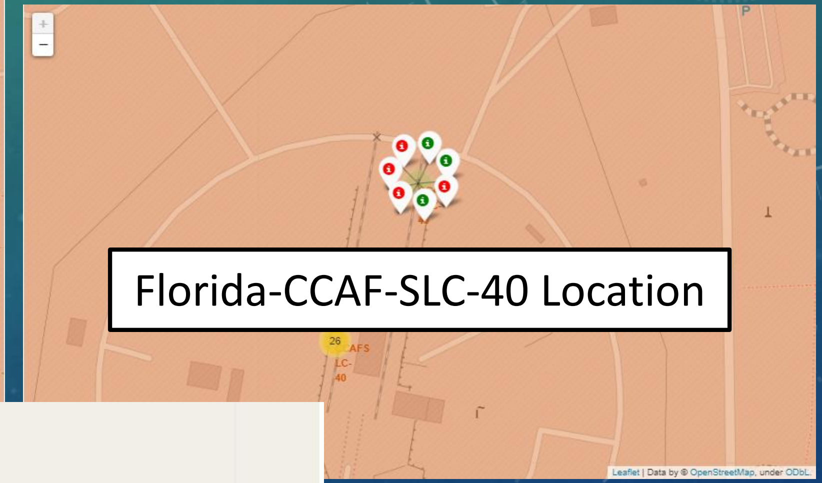
The location of launch sites in United States of America:

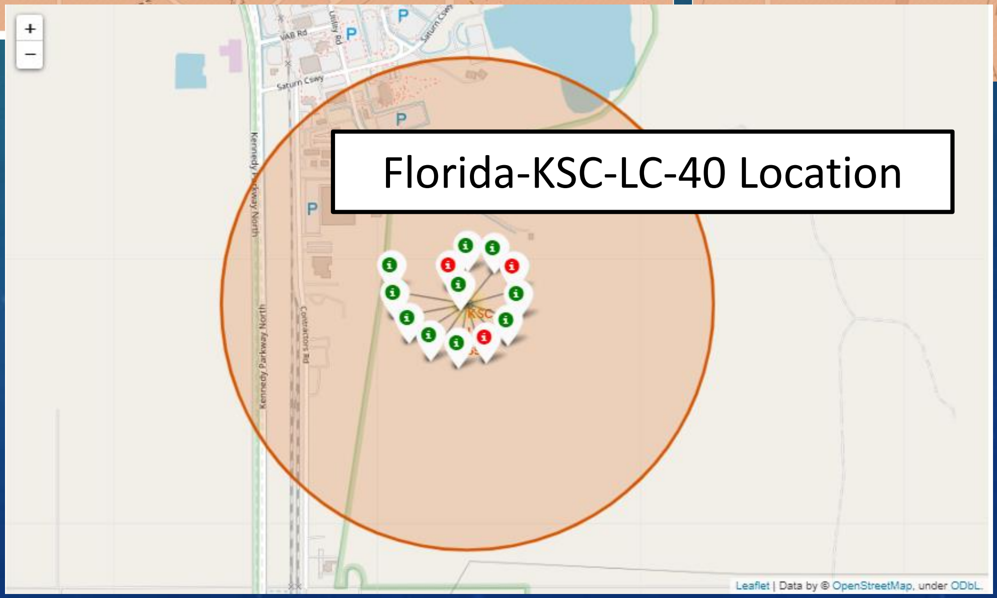
1) Claifornia- (Green markers shows the Successful launches and Red marker shows the Unsuccessful Launches)

2) Florida- (Green markers shows the Successful launches and Red marker shows the Unsuccessful Launches)

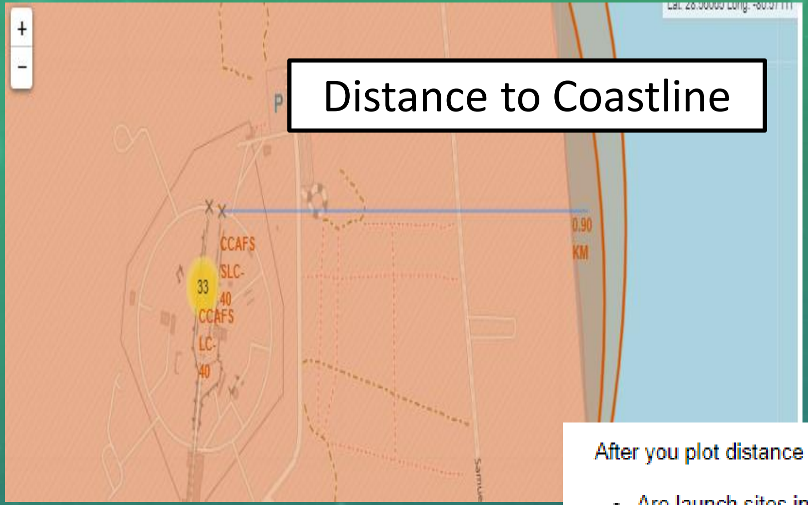


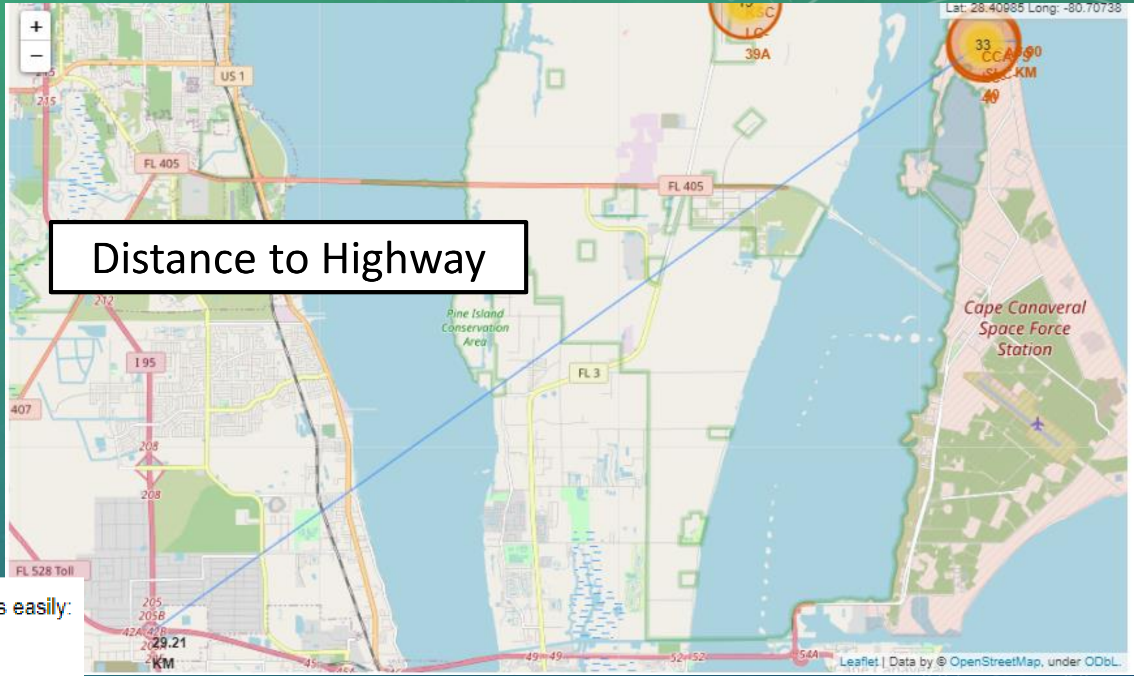






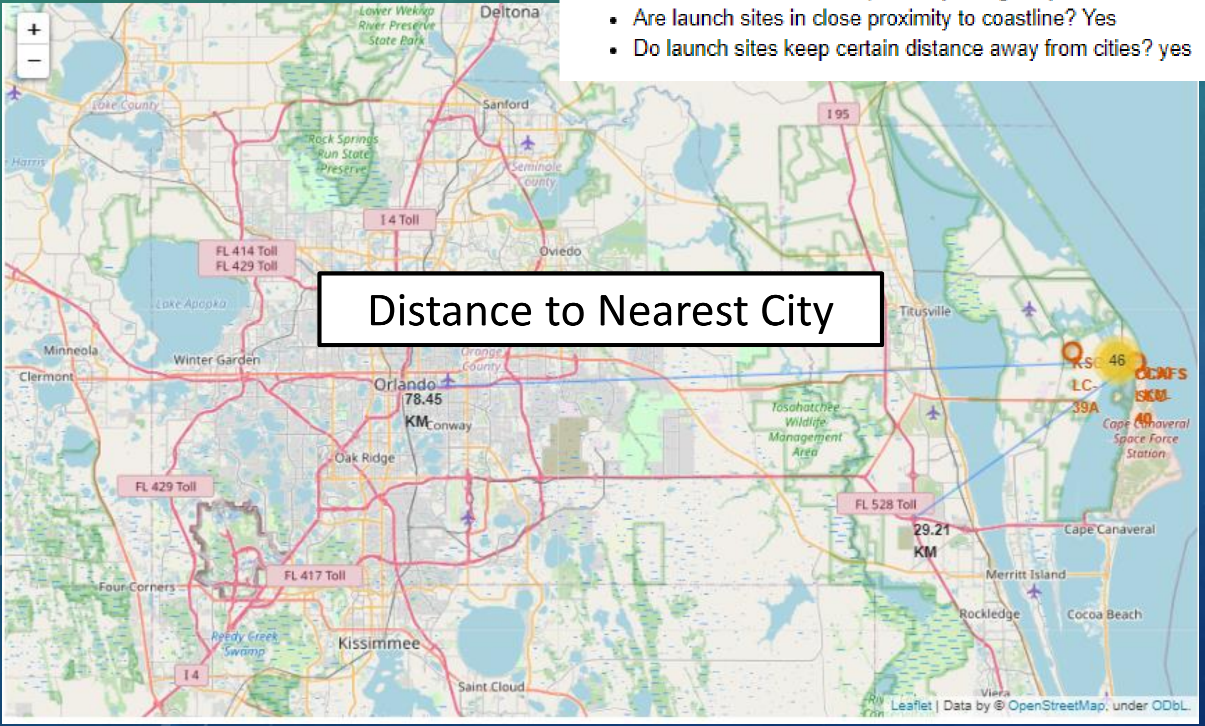
LAUNCH SITES PROXIMITIES DISTANCE TO LANDMARKS

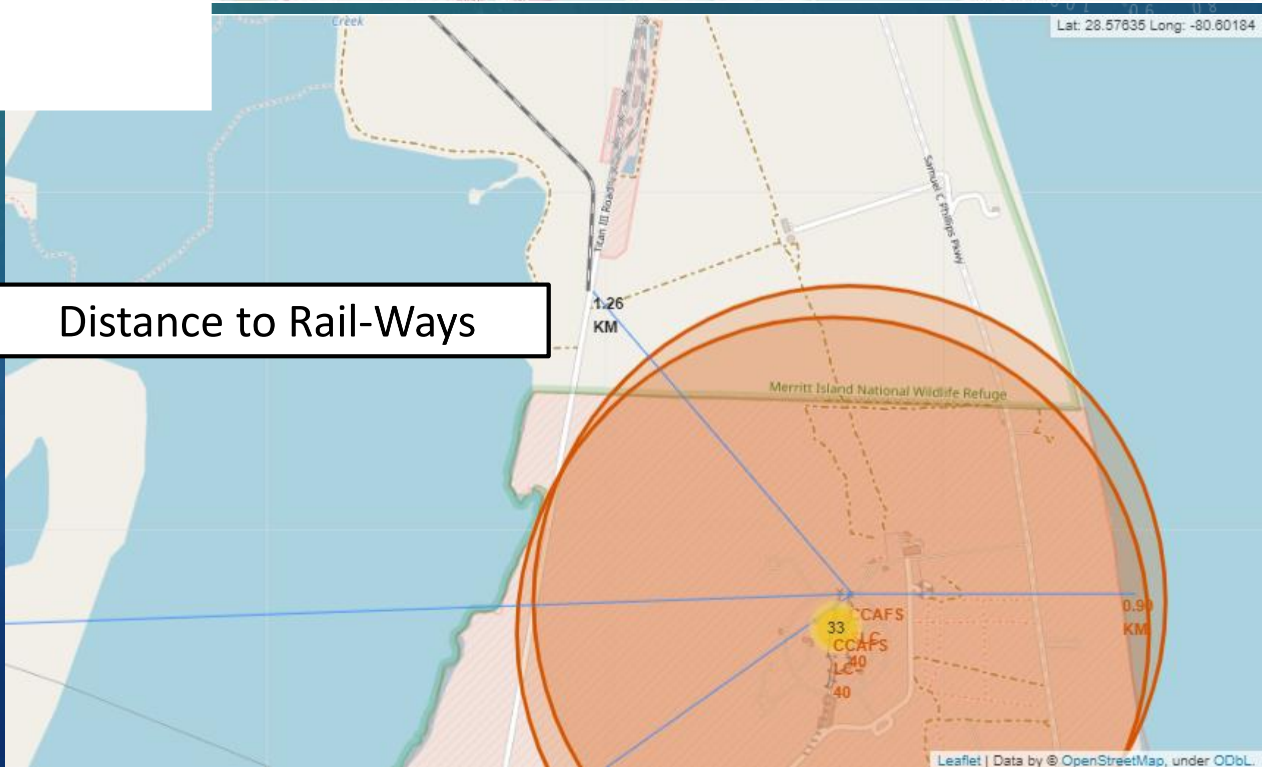




After you plot distance lines to the proximities, you can answer the following questions easily:

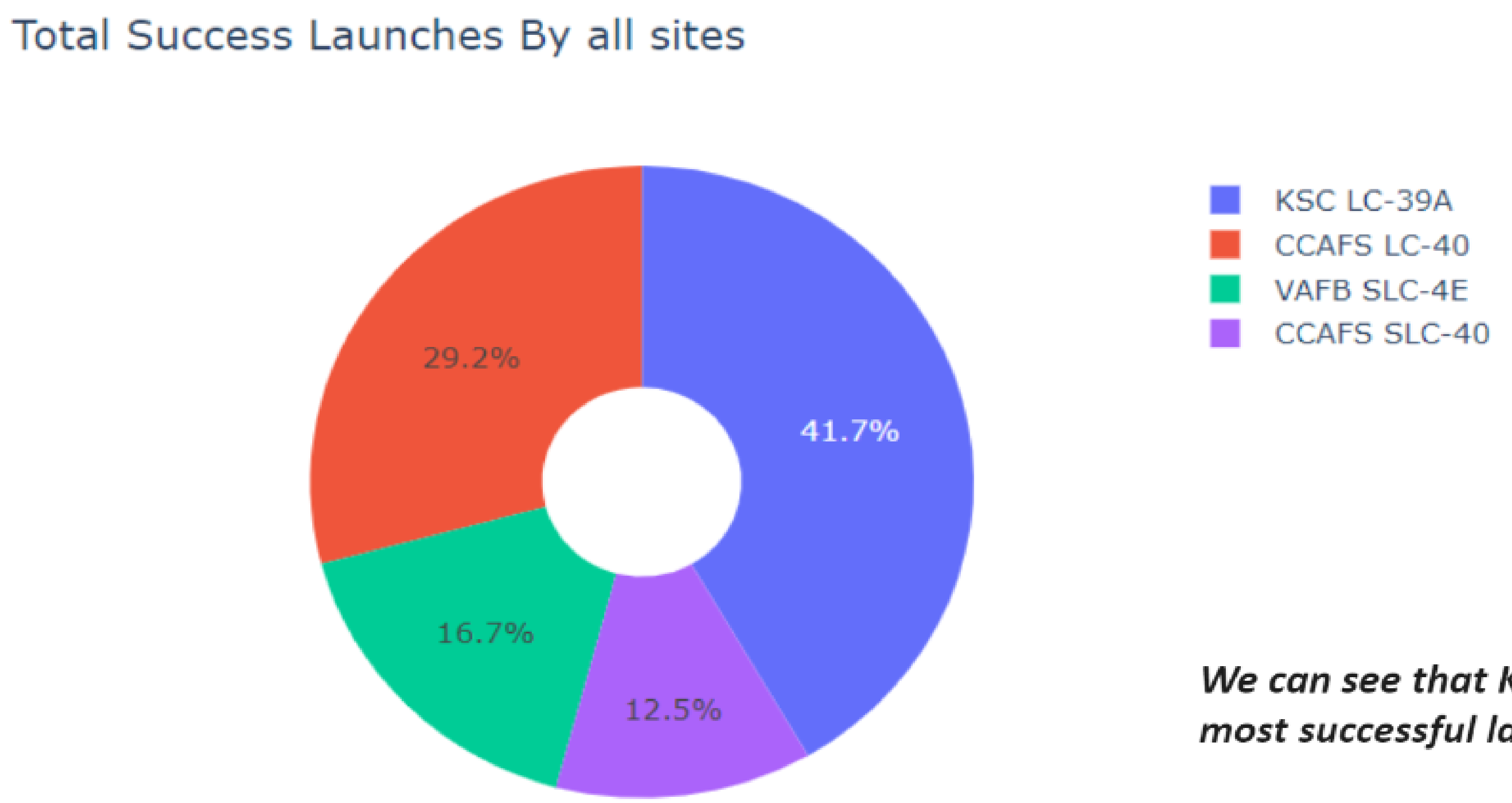
- Are launch sites in close proximity to railways? Yes
- Are launch sites in close proximity to highways? No
- Are launch sites in close proximity to coastline? Yes
- Do launch sites keep certain distance away from cities? yes





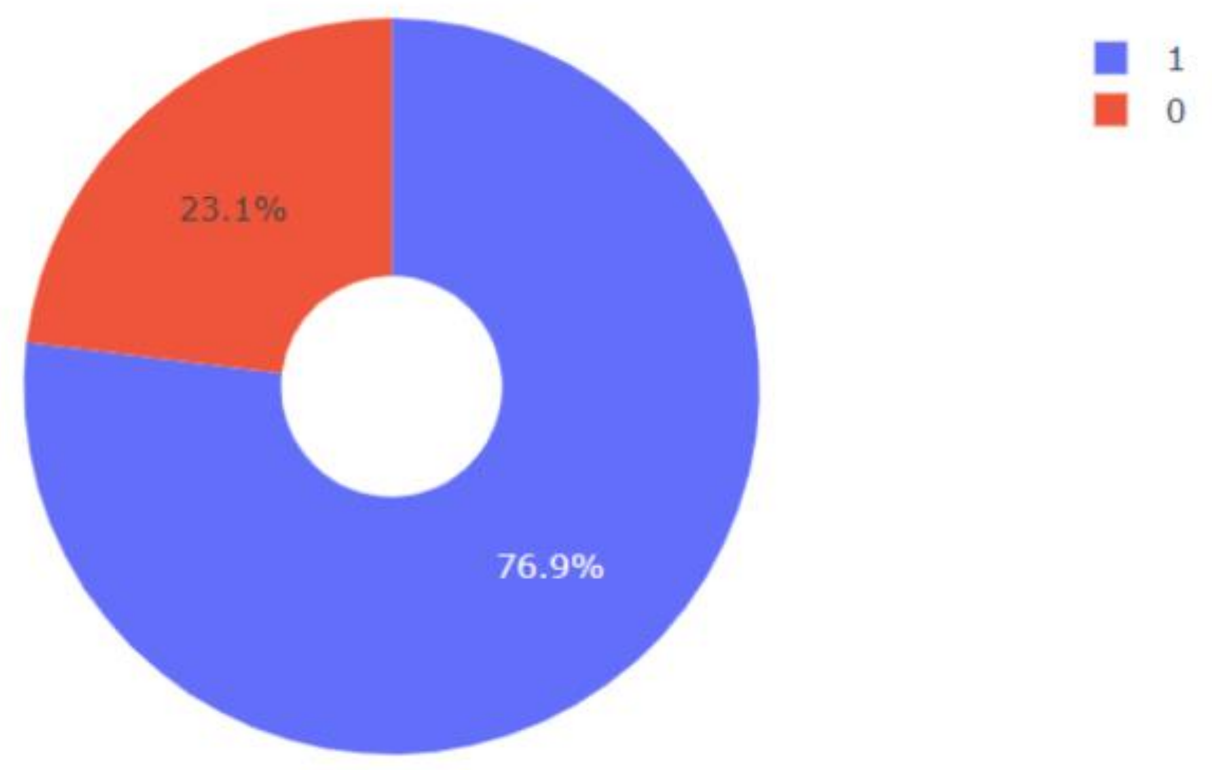
INSIGHTS DRAWN FROM DASHBOARD WITH PLOTLY

PIE-CHART SHOWING PERCENTAGE ACHIEVED BY EACH LAUNCH SITE



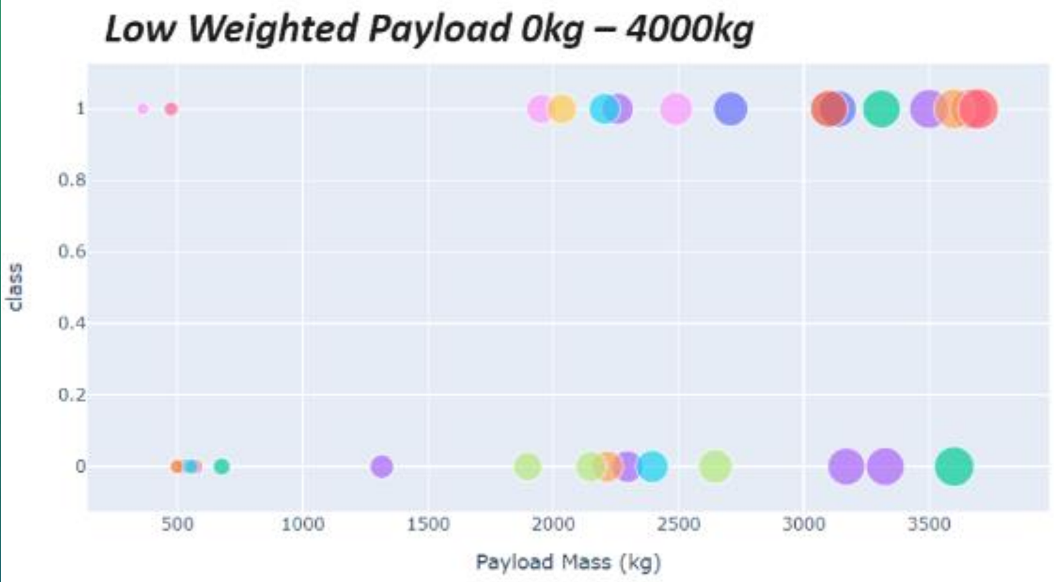
We can see that KSC LC-39A had the most successful launches from all the sites

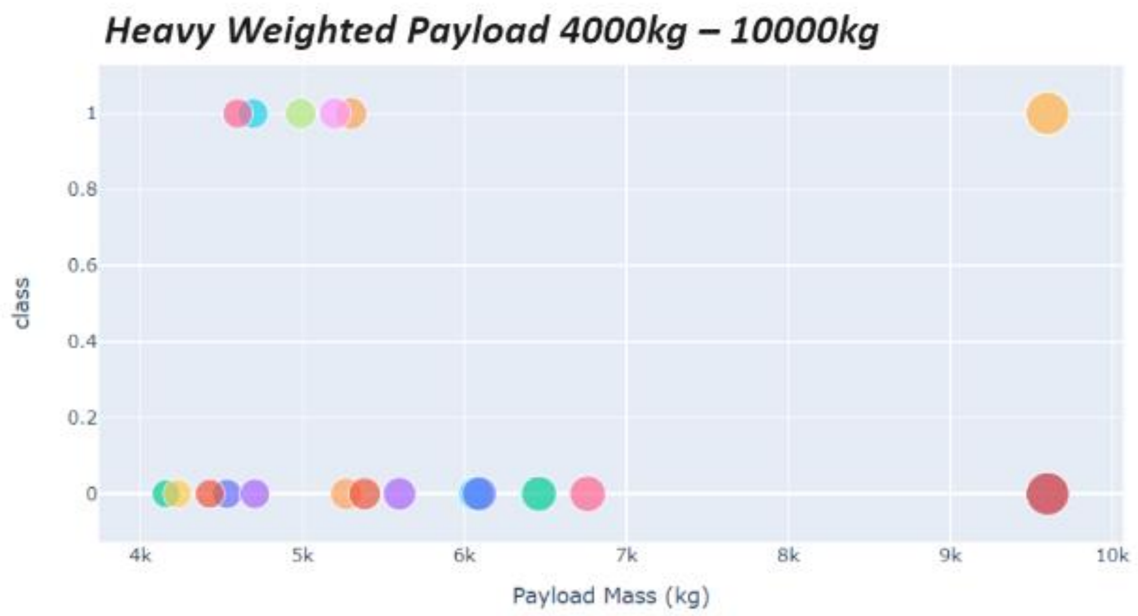
PIE CHART SHOWING THE LAUNCH SITE WITH THE HIGHEST LAUNCH SUCCESS RATIO



KSC LC-39A achieved a 76.9% success rate while getting a 23.1% failure rate

SCATTER PLOT OF PAYLOAD VS LAUNCH OUTCOME FOR ALL SITES, WITH DIFFERENT PAYLOAD SELECTED IN THE RANGE SLIDER





We can see the success rates for low weighted payloads is higher than the heavy weighted payloads

INSIGHTS FROM PREDICTIVE ANALYSIS

DIFFERENT MODELS

Logistic Regression:

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`.

In [15]:
```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

In [55]:
```
parameters_lr ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression(random_state = 12345)
grid_lr=GridSearchCV(
        estimator=lr,
        param_grid=parameters_lr,
        scoring='accuracy',
        cv=10)
logreg_cv=grid_lr.fit(X_train,Y_train)
```

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\_`.

In [56]:
```
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

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

In [57]:
```
print("The Accuracy Score of Test Data is:",logreg_cv.score(X_test,Y_test))
```

```
The Accuracy Score of Test Data is: 0.8333333333333334
```

DIFFERENT MODELS

SVM:

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` .

In [59]:
```
parameters_svm = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC(random_state = 12345)
```

In [60]:
```
grid_svm=GridSearchCV(
        estimator=svm,
        param_grid=parameters_svm,
        scoring='accuracy',
        cv=10)

svm_cv=grid_svm.fit(X_train,Y_train)
```

In [61]:
```
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` :

In [62]:
```
print("The Accuracy Score of Test Data is:",svm_cv.score(X_test,Y_test))
```

```
The Accuracy Score of Test Data is: 0.8333333333333334
```

DIFFERENT MODELS

Decision Tree:

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`.

In [64]:

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

tree = DecisionTreeClassifier(random_state = 12345)
```

In [65]:

```
grid_tree=GridSearchCV(
          estimator=tree,
          param_grid=parameters_tree,
          scoring='accuracy',
          cv=10)
tree_cv=grid_tree.fit(X_train,Y_train)
```

In [66]:

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

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 6, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_samples_split': 5,
'splitter': 'random'}
accuracy : 0.8732142857142856
```

TASK 9

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

In [67]:

```
print("The Accuracy Score of Test Data is:",tree_cv.score(X_test,Y_test))
```

```
The Accuracy Score of Test Data is: 0.8333333333333334
```

DIFFERENT MODELS

KNN:

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`.

In [69]:
```
parameters_knn = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'p': [1,2]}

KNN = KNeighborsClassifier()
```

In [70]:
```
grid_knn=GridSearchCV(
         estimator=KNN,
         param_grid=parameters_knn,
         scoring='accuracy',
         cv=10)
knn_cv=grid_knn.fit(X_train,Y_train)
```

In [71]:
```
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

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

In [72]:
```
print("The Accuracy Score of Test Data is:",knn_cv.score(X_test,Y_test))
```

```
The Accuracy Score of Test Data is: 0.8333333333333334
```

CLASSIFICATION ACCURACY

- Out of 4 classification models we have trained, fitted, found the accuracy score of test data and predicted the best fitting model to analyze the success rate of 1st stage of SpaceX rocket.

- Out of 4 models , the best model was Decision-tress with score of 0.873 with max_depth of 6 and the criterion is gini.

Find the method performs best:

In [75]:

```
models={'LogisticRegression':logreg_cv.best_score_,
        'SVM':svm_cv.best_score_,
        'Decision Tree':tree_cv.best_score_,
        'KNN':knn_cv.best_score_}
BestAlgorithm=max(models.values())
if BestAlgorithm==logreg_cv.best_score_:
    print("The Best Method to Adapt is LogisticRegression and its score is:",logreg_cv.best_score_)
    print("The Best Parameters is:",logreg_cv.best_params_)
if BestAlgorithm==svm_cv.best_score_:
    print("The Best Method to Adapt is SVM and its score is:",svm_cv.best_score_)
    print("The Best Parameters is:",svm_cv.best_params_)
if BestAlgorithm==tree_cv.best_score_:
    print("The Best Method to Adapt is Decision Tree and its score is:",tree_cv.best_score_)
    print("The Best Parameters is:",tree_cv.best_params_)
if BestAlgorithm==knn_cv.best_score_:
    print("The Best Method to Adapt is KNN and its score is:",knn_cv.best_score_)
    print("The Best Parameters is:",knn_cv.best_params_)
```

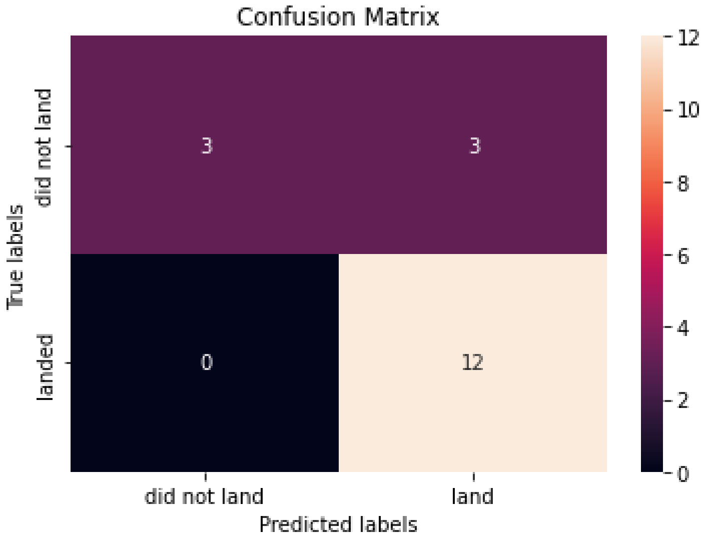
```
The Best Method to Adapt is Decision Tree and its score is: 0.8732142857142856
The Best Parameters is: {'criterion': 'gini', 'max_depth': 6, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_samples_split': 5, 'splitter': 'rand
om'}
```

CONFUSION MATRIX FOR DECISION TREE MODEL

- The confusion matrix for the decision tree classifier shows that the classifier can distinguish between the different classes. The major problem is the false positives .i.e., unsuccessful landing marked as successful landing by the classifier.

In [68]:

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



CONCLUSION

We can conclude that:

- The larger the flight amount at a launch site, the greater the success rate at a launch site.
- Launch success rate started to increase in 2013 till 2020.
- From the below plot we can find out that, flights launched into these orbits ES-L1, GEO, HEO, SSO, VLEO have the most success rate of 1.0. The flights launched into VLEO orbit has a success rate of about 0.8.
- Highest number of flights from earth to space has been launched from launch site that is CCAPS SLC-40.
- The Decision tree classifier is the best machine learning algorithm for this task, with score of 0.873 with max_depth of 6 and the criterion is gini.
- KSC LC-39A had the most successful launches of any sites.

THANK YOU