Coursera
IBM Applied Data Science Capstone
IBM Developer Skills Network

SpaceX Falcon9Landing Report

Submitted in the fulfillment of obtaining IBM Data Science Professional Certificate
2021

Submitted by:
Amna Bouhoreira

Saturday, October 09th, 2021.

Content:

1. Excusive Summay:

This project aims to predict the Falcon9 first stage succesfull landing, the project adopted some extraction methods to collect data(request, get, beautifulsoup). Initially the data has been collected and processed. The project relied on SQL and data visualisation to dive in data and have a clearer vision on it and to understand it properly. After the data has been presented in interactive map and dashboard which helped us to determine the best launch site for Falcon9 successful landing(KSC LC-39A). The Logistic Regression, SVM, C Tree and KNN prove that the launch site that we determined as a successful landing launch sites are really the best launch site(score= 0.8333…4).

Key words: SpaceX, Falcon9, Falcon9 successfull landing, SpaceX Falcon9.

2.Introduction:

Background :

SpaceX provides different types of orbits, one of those orbits are Falcon9. the success of the orbit landing depends on different factors, and the main problem is how can Falcon9 land succesfully, and how to determine the successful landing mission. This project predict the Falcon9 succesfull landing and determine the launch location that realise this successfull landing.

The Problem:

So the main question is:

- What is the successfull landing location for Falcon9?

Semi-Questions:

- Has the launch sites have a successfull rate?
- What is the best launch site for Falcon9 succesfull landing?

Hypothesis:

- Launch sites have a success rate of :
  - H0: Less than 30%,
  - H1: 30% to 60%,
  - H2: More than 60%
- LR, SVM, C Tree, KNN score:
  - H0: Less than 0.5
  - H1: 0.5 to 0.75
  - H2: More than 0.75

3.Research Method:

In this section we display the data collection and data processing methods.

3.1. Data Collection:

The data has been collected by prsing the SpaceX Launch from URL by using requests method, convert the json results to data frame using json_normalize method.

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

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 [11]: # Use json_normalize meethod to convert the json result into a dataframe
r = requests.get(static_json_url)
d= r.json()
data = pd.json_normalize(d)
data
```

The result(Output):

Out[11]:

	static_fire_date_utc	static_fire_date_unix	tbd	net	window	rocket	success	details	cr
0	2006-03-17T00:00:00.000Z	1.142554e+09	False	False	0.0	5e9d0d95eda69955f709d1eb	False	Engine failure at 33 seconds and loss of vehicle	
1	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	

Display 5 raws of the DataFrame:

```
In [12]: # Get the head of the dataframe
data.head(5)
```

Out[12]:

	static_fire_date_utc	static_fire_date_unix	tbd	net	window	rocket	success	details	crew	ships	capsules	
0	2006-03-17T00:00:00.000Z	1.142554e+09	False	False	0.0	5e9d0d95eda69955f709d1eb	False	Engine failure at 33 seconds and loss of vehicle	[]	[]	[]	[5eb0e4
								Successful first stage burn and transition to				

3.1. Data Collection:

We've dropped the Falcon1 from Booster Version column and reset the flight number column.

In [24]:
```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = launch.loc[launch['BoosterVersion']!="Falcon 1"]
```

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

In [25]:
```
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9
```

The result(Output):

Out[25]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad
4	1	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None
5	2	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False	False	False	None
6	3	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False	False	False	None
7	4	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None
8	5	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	False	False	False	None
...	...	...	...	...	...	...	...	...	...	...	...	...
89	86	2020-09-03	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	2	True	True	True	5e9e3032383ecb6bb234e7ca
90	87	2020-10-06	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	3	True	True	True	5e9e3032383ecb6bb234e7ca
91	88	2020-10-18	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	6	True	True	True	5e9e3032383ecb6bb234e7ca
92	89	2020-10-24	Falcon 9	15600.0	VLEO	CCSFS SLC 40	True ASDS	3	True	True	True	5e9e3033383ecbb9e534e7cc
93	90	2020-11-05	Falcon 9	3681.0	MEO	CCSFS SLC 40	True ASDS	1	True	False	True	5e9e3032383ecb6bb234e7ca

90 rows × 17 columns

3.1. Data Collection:

We have take a look on the data, there was some missing values.

```
In [26]: data_falcon9.isnull().sum()

Out[26]: FlightNumber      0
         Date              0
         BoosterVersion    0
         PayloadMass       5
         Orbit             0
         LaunchSite        0
         Outcome           0
         Flights           0
         GridFins          0
```

Dealing with missing values by replacing it with the mean of the values.

```
In [27]: # Calculate the mean value of PayloadMass column
         mean = data_falcon9['PayloadMass'].mean()
         # Replace the np.nan values with its mean value
         data_falcon9['PayloadMass'] = data_falcon9['PayloadMass'].fillna(mean)

<ipython-input-27-5ea96cd453d5>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning
-a-view-versus-a-copy
  data_falcon9['PayloadMass'] = data_falcon9['PayloadMass'].fillna(mean)

In [28]: data_falcon9.isnull().sum()

Out[28]: 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
```

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

3.1. Data Collection:

we have used HTML Get method, requests and BeuatifulSoup to get Falcon9 Launch data from Flcon9 html page.

```
In [5]: # use requests.get() method with the provided static_url
        # assign the response to a object
        falcon9 = requests.get(static_url)
        falcon9
```

```
Out[5]: <Response [200]>
```

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

```
In [6]: # Use BeautifulSoup() to create a BeautifulSoup object from a response text content
        html = falcon9.text
        soup = BeautifulSoup(html, '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>
```

We have **e**xtracted all column/variable names from the HTML table header.

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

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

```
In [9]: # Let's print the third table and check its content
        first_launch_table = html_tables[2]
        print(first_launch_table)
```

```
<table class="wikitable plainrowheaders collapsible" style="width: 100%;">
<tbody><tr>
<th scope="col">Flight No.
</th>
<th scope="col">Date and<br/>time (<a href="/wiki/Coordinated_Universal_Time" title="Coordinated Universal Time">UT
C</a>)
</th>
```

```
In [10]: # 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
         column_names = []
         temp = soup.find_all('th')
         for x in range(len(temp)):
             try:
              name = extract_column_from_header(temp[x])
              if (name is not None and len(name) > 0):
                 column_names.append(name)
             except:
              pass
```

Check the extracted column names

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

['Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcom
'Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome
'Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome
'Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome
'N/A', 'Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch
ome', 'Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch o
me', 'Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch ou
e', 'FH 2', 'FH 3', 'Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Custom
'Launch outcome', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch out
e', 'Date and time ( )', 'Launch site', 'Payload', 'Orbit', 'Customer', 'Date and time ( )', 'Launch site', 'Payl
d', 'Orbit', 'Customer', 'Date and time ( )', 'Launch site', 'Payload', 'Orbit', 'Customer', 'Date and time ( )',
unch site', 'Payload', 'Orbit', 'Customer', 'Demo flights', 'logistics', 'Crewed missions', 'Commercial satellite
'Scientific satellites', 'Military satellites', 'Rideshares', 'Current', 'In development', 'Retired', 'Cancelled'
```

```
In [12]: launch_dict= dict.fromkeys(column_names)

         # Remove an irrelvant column
         del launch_dict['Date and time ( )']

         # Let's initial the launch_dict with each value to be an empty list
         launch_dict['Flight No.'] = []
         launch_dict['Launch site'] = []
         launch_dict['Payload'] = []
         launch_dict['Payload mass'] = []
         launch_dict['Orbit'] = []
         launch_dict['Customer'] = []
         launch_dict['Launch outcome'] = []
         # Added some new columns
         launch_dict['Version Booster']=[]
         launch_dict['Booster landing']=[]
         launch_dict['Date']=[]
         launch_dict['Time']=[]
```

We have extracted the booster landing by:

```
In [13]: extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders collapsible")):
   # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number corresponding to launch a number
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
        else:
            flag=False
        #get table element
        row=rows.find_all('td')
        #if it is number save cells in a dictonary
        if flag:
            extracted_row += 1
            # Flight Number value
            # TODO: Append the flight_number into launch_dict with key `Flight No.`
            #print(flight_number)
            datatimelist=date_time(row[0])

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

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

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

            # Launch Site
            # TODO: Append the bv into launch_dict with key `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`
            payload = row[3].a.string
            #print(payload)
            print(payload)
            # Payload Mass
            # TODO: Append the payload_mass into launch_dict with key `Payload mass`
            payload_mass = get_mass(row[4])
            #print(payload_mass)
            print(payload_mass)
            # Orbit
            # TODO: Append the orbit into launch_dict with key `Orbit`
            orbit = row[5].a.string
            #print(orbit)
            print(orbit)
            # Customer
            # TODO: Append the customer into launch_dict with key `Customer`
            customer = row[6].a.string
            #print(customer)
            print(customer)
            # Launch outcome
            # TODO: Append the launch_outcome into launch_dict with key `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`
            booster_landing = landing_status(row[8])
            #print(booster_landing)
            print(booster_landing)
```

```
F9 v1.0B0003.1
CCAFS
Dragon Spacecraft Qualification Unit
0
LEO
SpaceX
Success

Failure
F9 v1.0B0004.1
CCAFS
Dragon
0
LEO
NASA
Success
Failure
F9 v1.0B0005.1
CCAFS
```

3.1. Data Collection:

We have calculated the number of launches on each site, the number of occurrence of each orbit and the number of accurence of mission outcome by orbit type :

```
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
```

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

Out[6]: GTO      0.300000
        ISS      0.233333
        VLEO     0.155556
        PO       0.100000
        LEO      0.077778
        SSO      0.055556
        MEO      0.033333
        SO       0.011111
        HEO      0.011111
        ES-L1    0.011111
        GEO      0.011111
        Name: Orbit, dtype: float64
```


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

Out[6]: GTO      0.300000
        ISS      0.233333
        VLEO     0.155556
        PO       0.100000
        LEO      0.077778
        SSO      0.055556
        MEO      0.033333
        SO       0.011111
        HEO      0.011111
        ES-L1    0.011111
        GEO      0.011111
        Name: Orbit, dtype: float64
```


```
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
```

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

Out[9]: {'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

3.2. Data Processing:

Using the SQL Alchemy and ibm_db method we have connected to IBM db2 database, we have take a deeper look on the data set by:

a. Displayed the names of the unique launch sites in the space mission

```
In [4]: %sql SELECT DISTINCT launch_site FROM SPACEX
 * ibm_db_sa://gyh83869:***@21fecfd8-47b7-4937-840d-d791d0218660.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31864/bludb
Done.
```

Out[4]:

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

b. Displayed 5 records where launch sites begin with the string 'CCA'

```
In [5]: %sql select * from SPACEX where launch_site like 'CCA%' limit 5
 * ibm_db_sa://gyh83869:***@21fecfd8-47b7-4937-840d-d791d0218660.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31864/bludb
Done.
```

Out[5]:

DATE	time__utc_	booster_version	launch_site	payload	payload_mass__kg_	orbit	customer	mission_outcome	landing__outcome
2010-04-06	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-08-12	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2012-08-10	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-01-03	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-12	22:41:00	F9 v1.1	CCAFS LC-40	SES-8	3170	GTO	SES	Success	No atte

c. Displayed the total payload mass carried by boosters launched by NASA (CRS)

```
In [6]: %sql select SUM(payload_mass__kg_) totalpayloadmass from SPACEX Where customer = 'NASA(CRS)';
 * ibm_db_sa://gyh83869:***@21fecfd8-47b7-4937-840d-d791d0218660.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31864/bludb
Done.
```

Out[6]:

totalpayloadmass
None

d. Displayed average payload mass carried by booster version F9 v1.1

```
In [7]: %sql select AVG(PAYLOAD_MASS__KG_) AveragePayloadMass from SPACEX where BOOSTER_VERSION = 'F9 v1.1'
 * ibm_db_sa://gyh83869:***@21fecfd8-47b7-4937-840d-d791d0218660.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31864/bludb
Done.
```

Out[7]:

averagepayloadmass
3676

3.2. Data Processing :

e. Listed the date when the first successful landing outcome in ground pad was acheived.

```
In [8]: %sql select MIN(DATE) from SPACEX Where mission_outcome = 'Success'
         * ibm_db_sa://gyh83869:***@21fecfd8-47b7-4937-840d-d791d0218660.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31864/bludb
        Done.
Out[8]:
```

1
2010-04-06

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

```
In [9]: %sql select booster_version from SPACEX where mission_outcome= 'Success' and payload_mass__kg_>4000 AND payload_mass__kg_< 6000
         * ibm_db_sa://gyh83869:***@21fecfd8-47b7-4937-840d-d791d0218660.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31864/bludb
        Done.
Out[9]:
```

booster_version
F9 v1.1
F9 v1.1 B1011
F9 v1.1 B1014
F9 FT B1020
F9 FT B1022
F9 FT B1032.1
F9 B4 B1040.1
F9 FT B1031.2
F9 B4 B1040.2
F9 B5 B1046.2
F9 B5 B1051.2
F9 B5B1062.1

g. Listed the total number of successful and failure mission outcomes

```
In [10]: %sql select count(MISSION_OUTCOME) as mission_outcomes from SPACEX GROUP BY MISSION_OUTCOME
          * ibm_db_sa://gyh83869:***@21fecfd8-47b7-4937-840d-d791d0218660.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31864/bludb
         Done.
Out[10]:
```

mission_outcomes
44
1

3.2. Data Processing :

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

```
In [11]: %sql SELECT MAX(payload_mass__kg_) booster_version from SPACEX DECS
```

```
 * ibm_db_sa://gyh83869:***@21fecfd8-47b7-4937-840d-d791d0218660.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31864/bludb
Done.
```

Out[11]:

booster_version
15600

i. Listed the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015

```
In [12]: %sql SELECT MONTH(DATE),MISSION_OUTCOME,BOOSTER_VERSION,LAUNCH_SITE FROM SPACEX where EXTRACT(YEAR FROM DATE)='2015'
```

```
 * ibm_db_sa://gyh83869:***@21fecfd8-47b7-4937-840d-d791d0218660.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31864/bludb
Done.
```

Out[12]:

1	mission_outcome	booster_version	launch_site
10	Success	F9 v1.1 B1012	CCAFS LC-40
11	Success	F9 v1.1 B1013	CCAFS LC-40
2	Success	F9 v1.1 B1014	CCAFS LC-40

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

```
In [13]: %sql SELECT LANDING__OUTCOME FROM SPACEX WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20' ORDER BY DATE DESC;
```

```
 * ibm_db_sa://gyh83869:***@21fecfd8-47b7-4937-840d-d791d0218660.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31864/bludb
Done.
```

Out[13]:

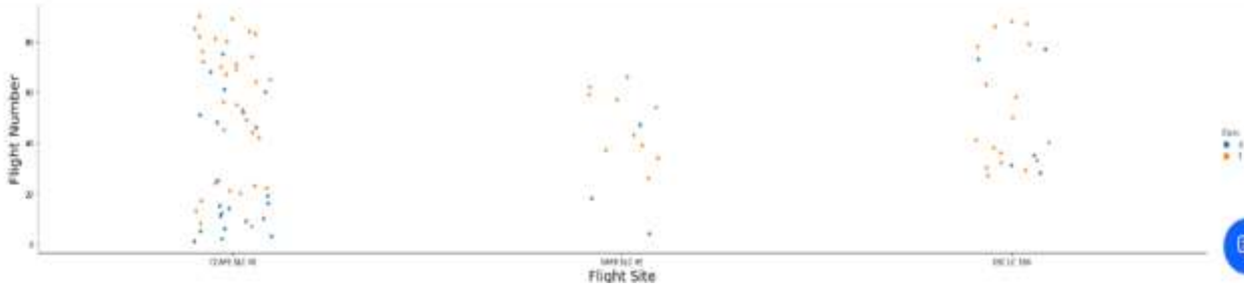
landing__outcome
Success (ground pad)
Success (ground pad)
Success (drone ship)
Success (drone ship)
Failure (drone ship)
Controlled (ocean)
Failure (drone ship)
No attempt
No attempt
No attempt
No attempt
No attempt
No attempt
No attempt
Failure (parachute)

3.2. Data Processing :

Dive in data and get better vision of the data:

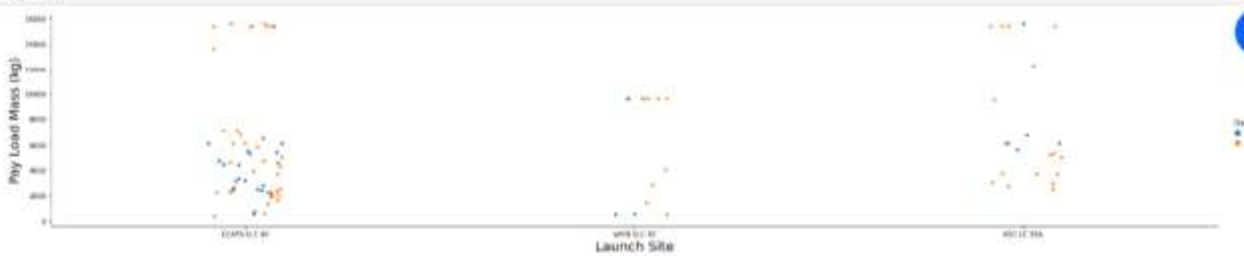
a. Visualize the relationship between Flight Number and Launch Site:

In [4]:
```
# 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="FlightNumber", x="LaunchSite", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Site",fontsize=20)
plt.ylabel("Flight Number",fontsize=20)
plt.show()
```



b.Visualize the relationship between Payload and Launch Site

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



c.Visualize the relationship between success rate of each orbit type

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

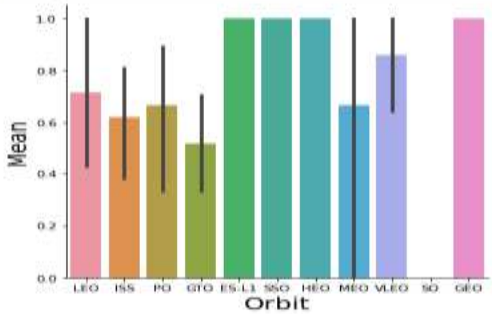
Out[6]:

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

3.2. Data Processing :

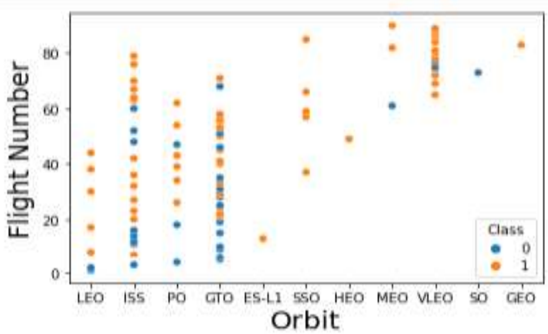
d.Visualize the relationship between FlightNumber and Orbit type

In [7]:
```
# 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="Orbit",y="Class", kind="bar",data=df)
plt.xlabel("Orbit",fontsize=20)
plt.ylabel("Mean",fontsize=20)
plt.show()
```



e. Visualize the relationship between Payload and Orbit type

In [8]:
```
# 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.scatterplot(x="Orbit",y="FlightNumber",hue="Class",data = df)
plt.xlabel("Orbit",fontsize=20)
plt.ylabel("Flight Number",fontsize=20)
plt.show()
```

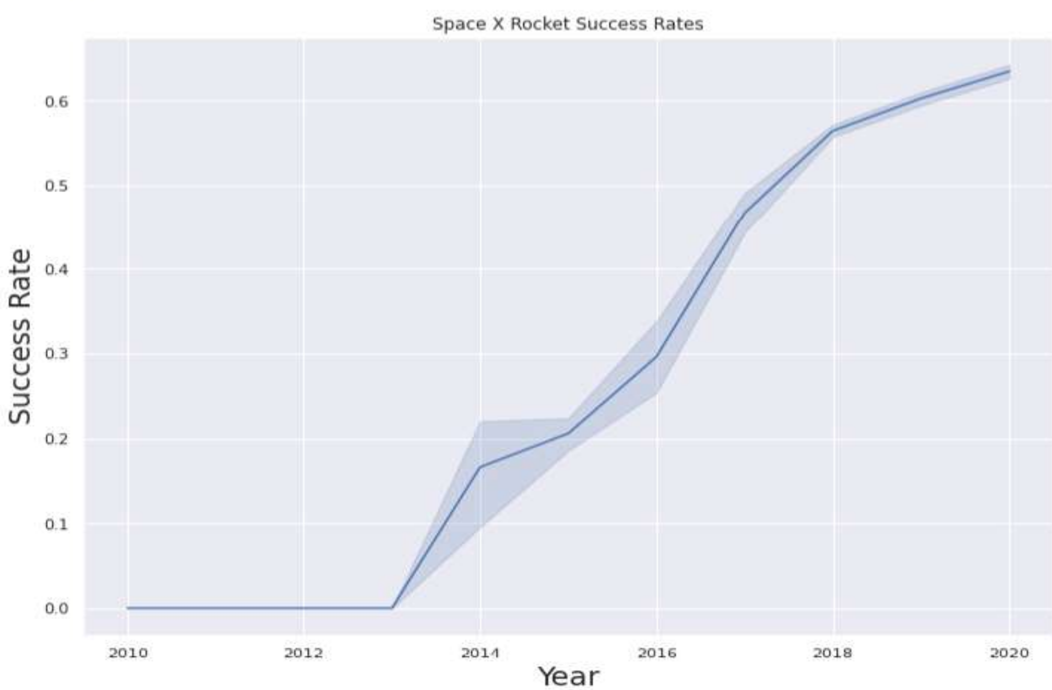


3.2. Data Processing :

f. Visualize the launch success yearly trend

In [9]:
```
# 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
```

In [10]:
```
# Plot a line chart with x axis to be the extracted year and y axis to be the success rate
year = pd.DatetimeIndex(df['Date']).year
year = np.array(list(year))
successratelist = []
successrate = 0.00
records = 1
data = 0
for x in df['Class']:
    data = x + data
    successrate = data/records
    successratelist.append(successrate)
    records= records +1

successratelist = np.array(successratelist)
d = {'successrate':successratelist,'year':year}
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.lineplot(data=d, x="year", y="successrate" )

plt.xlabel("Year",fontsize=20)
plt.title('Space X Rocket Success Rates')
plt.ylabel("Success Rate",fontsize=20)
plt.show()
```



3.3. Analysis Method:

The analysis method used are interactive foluim map, and dash board. The Prediction method is Machine Learning Algorithms: Support Vector Machines, Logistic Regression, Classification Tree and K Nearest Nieghbors. We shall illustrate that in the next section.

4.Empirical Analysis and Prediction:

In this section we have used foluim map to determine the successful and the failed landing locations, displayed the successful landing by launch site in an interactive dashboard, predicted the Falcon9 first stage landing using Machine Learning Algorithms(Logistic Regression, SVM, Classification Tree, K Nearest Nieghbors).

4.1. Interactive Foluim Map:

a. The launches site location analysis using foluim:

Create dummy variables to categorical columns

```
In [12]: # HINT: Use get_dummies() function on the categorical columns
features_hot = df[['Orbit','LaunchSite','LandingPad','Serial']]
features_hot['Orbit'] = pd.get_dummies(df['Orbit'])
features_hot['LaunchSite'] = pd.get_dummies(df['LaunchSite'])
features_hot['LandingPad'] = pd.get_dummies(df['LandingPad'])
features_hot['Serial'] = pd.get_dummies(df['Serial'])
features_hot.head()
```

Out[12]:

	Orbit	LaunchSite	LandingPad	Serial
0	0	1	0	1
1	0	1	0	0
2	0	1	0	0
3	0	0	0	0
4	0	1	0	0

Cast all numeric columns to float64

```
In [13]: # HINT: use astype function
features_hot.astype('float64')
features_hot

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

4.1. Interactive Foluim Map

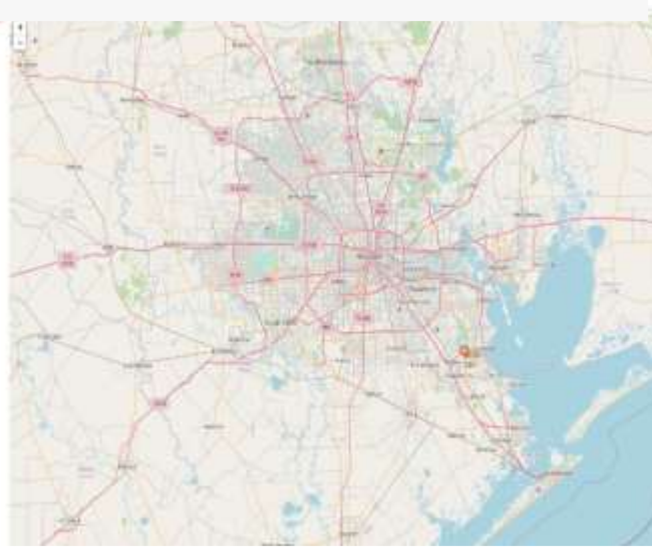
Mark all launch sites on a map

In [22]:
```
# Start location is NASA Johnson Space Center
nasa_coordinate = [29.559684888503615, -95.0830971930759]
site_map = folium.Map(location=nasa_coordinate, zoom_start=10)
```

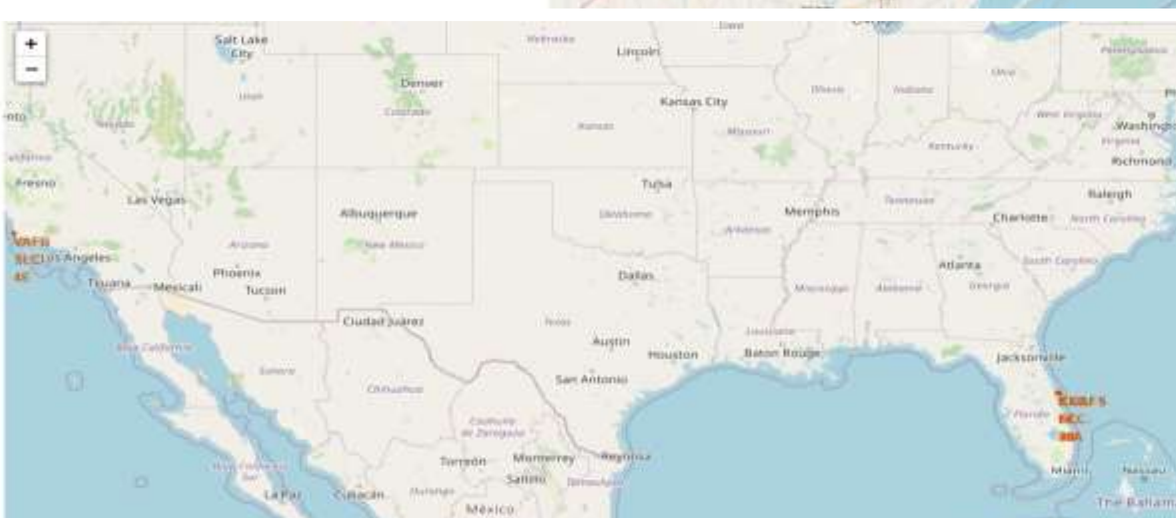
We could use `folium.Circle` to add a highlighted circle area with a text label on a specific coordinate. For example,

In [23]:
```
# Create a blue circle at NASA Johnson Space Center's coordinate with a popup label showing its name
circle = folium.Circle(nasa_coordinate, radius=1000, color='#d35400', fill=True).add_child(folium.Popup('NASA Johnson Space Center'))
# Create a blue circle at NASA Johnson Space Center's coordinate with a icon showing its name
marker = folium.map.Marker(
    nasa_coordinate,
    # Create an icon as a text label
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % 'NASA JSC',
        )
    )
site_map.add_child(circle)
site_map.add_child(marker)
```

You may see that the spacex launches sites are USA Florida and California.



Out[7]:



4.1. Interactive Foluim Map:

b. Mark the success/failed launches for each site on the map:

Let's first create a `MarkerCluster` object

```
In [26]: marker_cluster = MarkerCluster()
```

TODO: Create a new column in `launch_sites` dataframe called `marker_color` to store the marker colors based on the `class` value

```
In [27]:
# Apply a function to check the value of `class` column
# If class=1, marker_color value will be green
# If class=0, marker_color value will be red
launch_sites=[]
for i in enumerate(spacex_df['class']):
    if i==1:
        launch_sites.append('Green')
    else:
        launch_sites.append('Red')
```

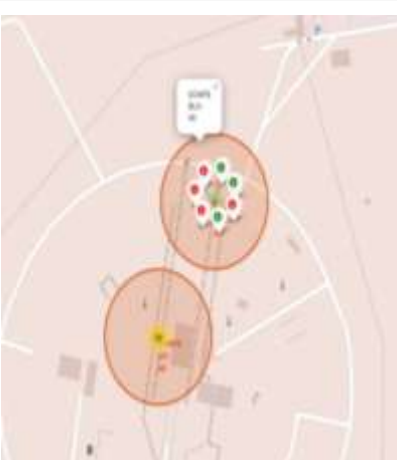
```
In [28]: # Function to assign color to launch outcome
def assign_marker_color(launch_outcome):
    if launch_outcome == 1:
        return 'green'
    else:
        return 'red'

spacex_df['marker_color'] = spacex_df['class'].apply(assign_marker_color)
spacex_df.tail(10)
```

Out[28]:

	Launch Site	Lat	Long	class	marker_color
46	KSC LC-39A	28.573255	-80.646895	1	green
47	KSC LC-39A	28.573255	-80.646895	1	green
48	KSC LC-39A	28.573255	-80.646895	1	green
49	CCAFS SLC-40	28.563197	-80.576820	1	green
50	CCAFS SLC-40	28.563197	-80.576820	1	green

Green Marks are Successful Launches site, Red Marks are Failed Launches sites

```
# Add the Marker cluster to the site map
site_map.add_child(marker_cluster)
```



```
In [30]: # Add Mouse Position to get the coordinate (Lat, Long) for a mouse over on the map
formatter = "function(num) {return L.Util.formatNum(num, 5);};"
mouse_position = MousePosition(
    position='topright',
    separator=' Long: ',
    empty_string='NaN',
    lng_first=False,
    num_digits=20,
    prefix='Lat:',
    lat_formatter=formatter,
    lng_formatter=formatter,
)

site_map.add_child(mouse_position)
site_map
```

4.1. Interactive Foluim Map:

c. Calculate the distances between a launch site to its proximities:

```
In [31]: from math import sin, cos, sqrt, atan2, radians

         def calculate_distance(lat1, lon1, lat2, lon2):
             # approximate radius of earth in km
             R = 6373.0

             lat1 = radians(lat1)
             lon1 = radians(lon1)
             lat2 = radians(lat2)
             lon2 = radians(lon2)

             dlon = lon2 - lon1
             dlat = lat2 - lat1

             a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
             c = 2 * atan2(sqrt(a), sqrt(1 - a))

             distance = R * c
             return distance
```

```
In [33]: # create and add a folium.Marker on your selected closest raiwaly point on the map
         # show the distance to the launch site using the icon property
         #Work out distance to coastline
         coordinates = [
             [28.56342, -80.57674],
             [28.56342, -80.56756]]

         lines=folium.PolyLine(locations=coordinates, weight=1)
         site_map.add_child(lines)
         distance = calculate_distance(coordinates[0][0], coordinates[0][1], coordinates[1][0], coordinates[1][1])
         distance_circle = folium.Marker(
             [28.56342, -80.56794],
             icon=DivIcon(
                 icon_size=(20,20),
                 icon_anchor=(0,0),
                 html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:10.2f} KM".format(distance),
                 )
             )
         site_map.add_child(distance_circle)
         site_map
```

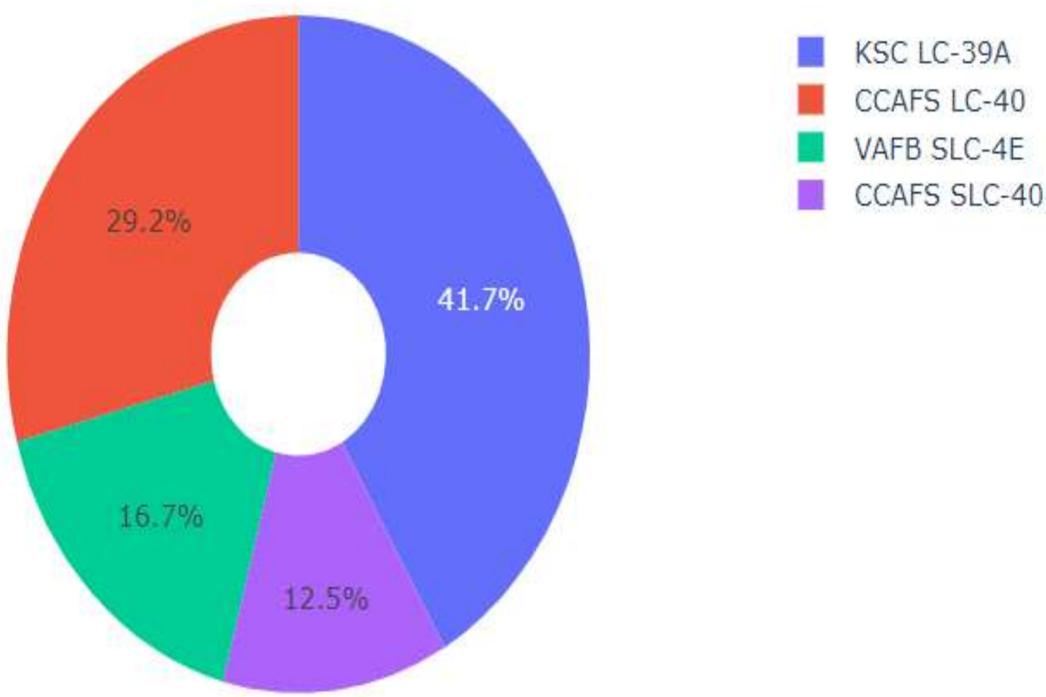
```
In [34]: # Create a `folium.PolyLine` object using the railway point coordinate and launch site coordinate
         #Distance to Highway
         coordinates = [
             [28.56342, -80.57674],
             [28.411780, -80.820630]]

         lines=folium.PolyLine(locations=coordinates, weight=1)
         site_map.add_child(lines)
         distance = calculate_distance(coordinates[0][0], coordinates[0][1], coordinates[1][0], coordinates[1][1])
         distance_circle = folium.Marker(
             [28.411780, -80.820630],
             icon=DivIcon(
                 icon_size=(20,20),
                 icon_anchor=(0,0),
                 html='<div style="font-size: 12; color:#252526;"><b>%s</b></div>' % "{:10.2f} KM".format(distance),
                 )
             )
         site_map.add_child(distance_circle)
         site_map
```

Green Marks are Successful Launches site, Red Marks are Failed Launches sites

4.2.Interactive DashBoard :

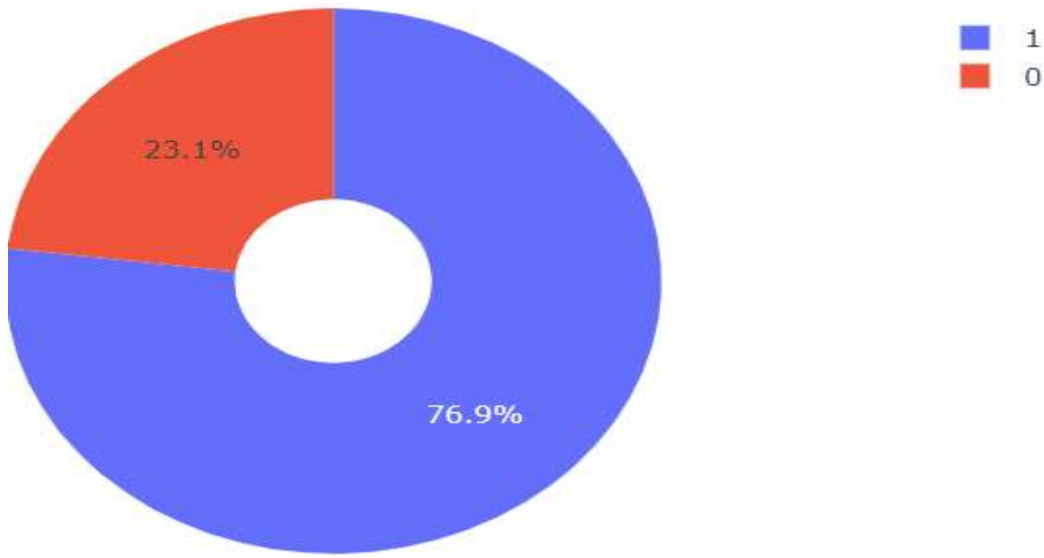
The dropdown of all Launches site:

The pie plot represent the launches by site, you may see that KSC LC-39A have the most success rate.

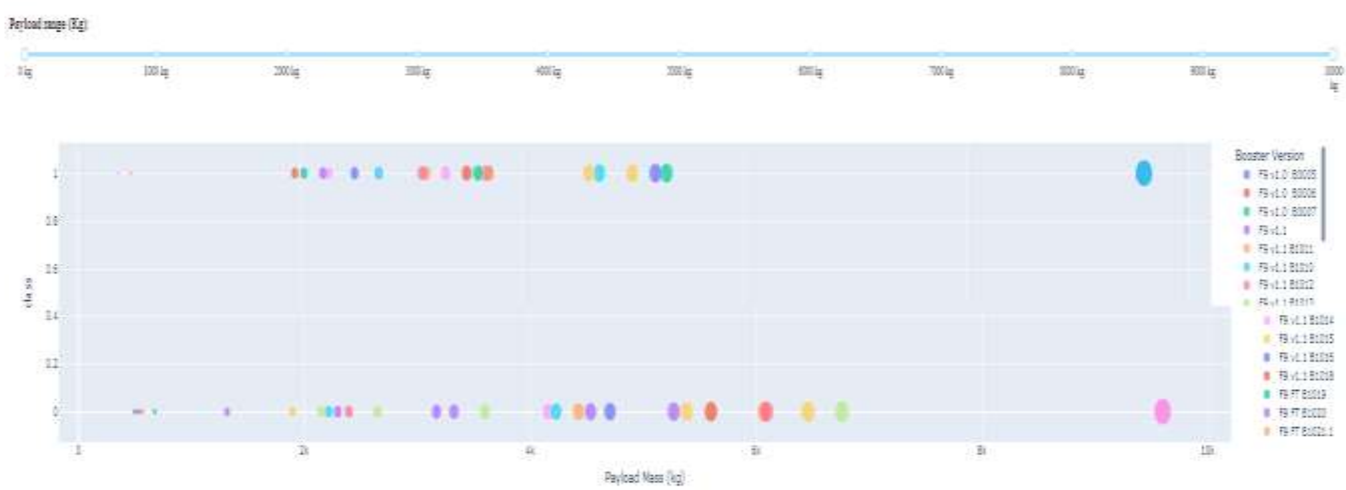


4.2.Interactive DashBoard :

The KSC LC-39A have 76.9% rate of success:



The scatter dislpay the booster versions by payload mass kg:



4.3. Spacex Falcon9 First Stage Prediction:

Create numpy array, standerlize the data and split data into train and test data:

In [5]:
```
Y=data['Class'].to_numpy()
Y
```

Out[5]:
```
array([0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1,
       1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1])
```

In [8]:
```
# students get this
X = preprocessing.StandardScaler().fit(X).transform(X)
```

In [9]:
```
X
```

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

In [10]:
```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=2)
```

we can see we only have 18 test samples.

In [11]:
```
Y_test.shape
```

Out[11]:
```
(18,)
```

4.3. Spacex Falcon9 First Stage Prediction:

Use Logistic Rgression to predict Falcon9 first stage landing:

```
In [12]: parameters ={'C':[0.01,0.1,1],
                     'penalty':['l2'],
                     'solver':['lbfgs']}
```

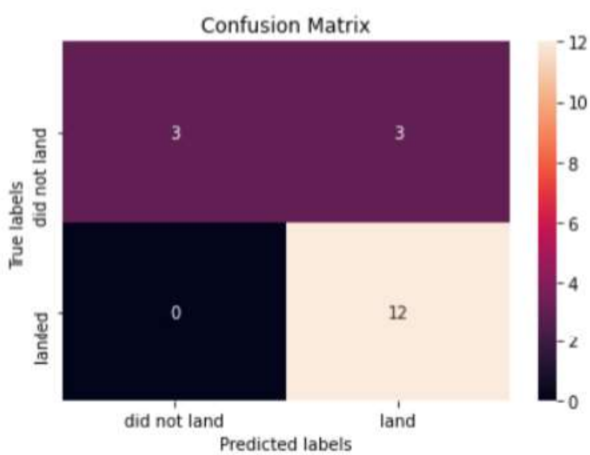
```
In [15]: parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
         lr=LogisticRegression()
         logreg_cv=GridSearchCV(lr,parameters,cv=10)
         logreg_cv.fit(X_train,Y_train)

Out[15]: GridSearchCV(cv=10, estimator=LogisticRegression(),
                      param_grid={'C': [0.01, 0.1, 1], 'penalty': ['l2'],
                                  'solver': ['lbfgs']})
```

```
In [16]: 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
```

```
In [17]: lr_score=logreg_cv.score(X_test,Y_test)
         lr_score

Out[17]: 0.8333333333333334
```

Lets look at the confusion matrix:

```
In [18]: yhat=logreg_cv.predict(X_test)
         plot_confusion_matrix(Y_test,yhat)
```



4.3. Spacex Falcon9 First Stage Prediction:

Use Support Vector Machines to predict Falcon9 first stage landing:

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

In [21]:
```
svm_cv = GridSearchCV(svm,parameters,cv=10)
svm_cv.fit(X_train,Y_train)
```

Out[21]:
```
GridSearchCV(cv=10, estimator=SVC(),
             param_grid={'C': array([1.00000000e-03, 3.16227766e-02, 1.00000000e+00, 3.16227766e+01,
       1.00000000e+03]),
                         'gamma': array([1.00000000e-03, 3.16227766e-02, 1.00000000e+00, 3.16227766e+01,
       1.00000000e+03]),
                         'kernel': ('linear', 'rbf', 'poly', 'rbf', 'sigmoid')})
```

In [22]:
```
print("tuned hpyerparameters :(best parameters) ",svm_cv.best_params_)
print("accuracy :",svm_cv.best_score_)
```

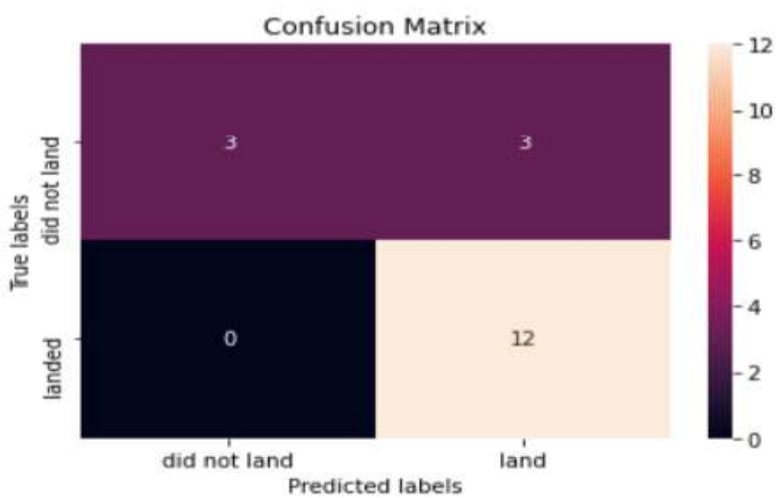
```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
accuracy : 0.8482142857142856
```

In [23]:
```
svm_score = svm_cv.score(X_test,Y_test)
svm_score
```

Out[23]: 0.8333333333333334

We can plot the confusion matrix

In [24]:
```
yhat=svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



4.3. Spacex Falcon9 First Stage Prediction:

Using Classification Tree to predict Falcon9 first stage landing:

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

         tree = DecisionTreeClassifier()
```

```
In [26]: tree_cv = GridSearchCV(tree,parameters,cv=10)
         tree_cv.fit(X_train,Y_train)
```

```
Out[26]: GridSearchCV(cv=10, estimator=DecisionTreeClassifier(),
                      param_grid={'criterion': ['gini', 'entropy'],
                                  'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18],
                                  'max_features': ['auto', 'sqrt'],
                                  'min_samples_leaf': [1, 2, 4],
                                  'min_samples_split': [2, 5, 10],
                                  'splitter': ['best', 'random']})
```

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

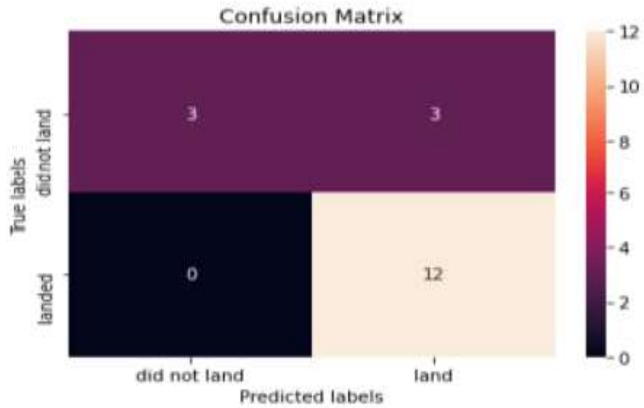
```
tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', 'max_depth': 4, 'max_features': 'sqrt', 'min_sampl
es_leaf': 2, 'min_samples_split': 2, 'splitter': 'random'}
accuracy : 0.875
```

```
In [28]: tree_score = tree_cv.score(X_test,Y_test)
         tree_score
```

```
Out[28]: 0.8333333333333334
```

We can plot the confusion matrix

```
In [29]: yhat = svm_cv.predict(X_test)
         plot_confusion_matrix(Y_test,yhat)
```



```
In [30]: 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()
```

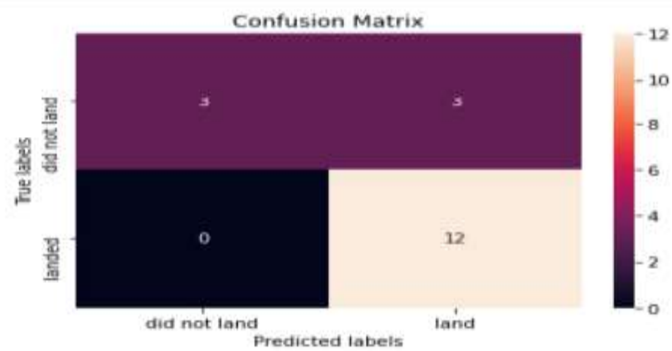
```
In [31]: knn_cv = GridSearchCV(KNN,parameters,cv=10)
         knn_cv.fit(X_train,Y_train)

Out[31]: GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
                      param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                                  'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                                  'p': [1, 2]})
```

```
In [32]: 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
```

```
In [33]: knn_score = knn_cv.score(X_test,Y_test)
         knn_score

Out[33]: 0.8333333333333334
```

We can plot the confusion matrix

```
In [34]: yhat = knn_cv.predict(X_test)
         plot_confusion_matrix(Y_test,yhat)
```




```
In [35]: scores = [lr_score,svm_score,tree_score,knn_score]
         print(scores)
         print(scores.index(max(scores)))

[0.8333333333333334, 0.8333333333333334, 0.8333333333333334, 0.8333333333333334]
0
```

5. Discussion and Interpretations:

5. Discussion and Interpretation:

The interactive dashboard shows that: CCAFS LC-40 launch site have the success rate 29.1% among all the other launches, VAFB SLC-4 E have the rate 16.7%, CCAFS SLC-4O have 12.5% rate, whereas KSC LC-39A has the most success rate by 41.6%, and has the sucess rate of 76.9%. In the 4.3. Falcon9 first stage prediction; Logistic Regression, SVM, Classification Tree, KNN has the same score 0.833..34, which means that it is the best score for the prediction. in other word the Falcon9 have a acceptable success rate in the launch site KSC LC-39A.

6. Conclusion:

As the results shows the Falcon9 will land successfully in the rate of 71.9%, in the KSC LC-39A launch site, whereas the other launches have a lower rate of successful landing which means that the hypothese H1 acceptable rate(success rate:30% to 60%). The Prediction shows the score 0.833…4 in the Logistic Regression, SVM, Classification Tree and KNN which means that the positive hypothese H2 best score(Score =>0.75) is proved. Based on the maps the launch site locations are the factor that influence the success of landing mission which mean that the orbits should land in the KSC LC-39A location in California and Florida, which where marked with green marks.