IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Alfonso Mondragón Montero
13 June 2022

Outline

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

Executive Summary

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

Introduction

- Project background and context
    - 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
    - What factors determine if the rocket will land successfully?
    - What features determine the success rate of a successful landing?

Section 1

Methodology

Methodology

Executive Summary

- 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
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
    - How to build, tune, evaluate classification models

Data Collection

- The data was collected:
  - ➢ Using get request to the Space API
  - ➢ Then, I decoded the response content as a Json using .json() function call and turn it into a pandas dataframe using .json_normalize().
  - ➢ I filter the dataframe to only include Falcon 9 launches.
  - ➢ I dealt with missing values
  - ➢ The goal 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

- I used the get request to the SpaceX API to collect data, clean the requested data and did some basic data wrangling and formatting.

- The link to the notebook is https://github.com/amm2307/Capstone_Project/blob/master/Capstone%20Final%20Project%20IBM%20Machine%20Learning.ipynb

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

In [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

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

Out[11]:
```
200
```

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

In [12]:
```
# Use json_normalize meethod to convert the json result into a dataframe
data = pd.json_normalize(response.json())
```

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

In [32]:
```
# Calculate the mean value of PayloadMass column
payloadmassavg = data_falcon9['PayloadMass'].mean()
# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'].replace(np.nan, payloadmassavg, inplace=True)
data_falcon9.isnull().sum()
```

Out[32]:
```
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

- I applied web scrapping to webscrap Falcon 9 launch records with BeautifulSoup, and I parsed the table and converted it into a pandas dataframe.
- The link to the notebook is https://github.com/amm2307/Capstone_Project/blob/master/Hands-on%20Lab%20Web%20Scraping.ipynb

In [8]:
```
static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"
```

Next, request the HTML page from the above URL and get a `response` object

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 [9]:
```
# use requests.get() method with the provided static_url
# assign the response to a object
data  = requests.get(static_url).text
```

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

In [10]:
```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(data, 'html5lib')
```

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

In [11]:
```
# Use soup.title attribute
print(soup.title)
```

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

In [14]:
```
column_names = []

# Apply find_all() function with `th` element on first_launch_table
# Iterate each th element and apply the provided extract_column_from_header() to get a column name
# Append the Non-empty column name (`if name is not None and len(name) > 0`) into a list called column_names
for row in first_launch_table.find_all('th'):
    name = extract_column_from_header(row)
    if (name != None and len(name) > 0):
        column_names.append(name)
```

Check the extracted column names

In [15]:
```
print(column_names)
```

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

Data Wrangling

- I performed exploratory data analysis and determined the training labels.
- I calculated the number of launches at each site, and the number and occurrence of each orbits
- The link to the notebook is https://github.com/amm2307/Capstone_Project/blob/master/EDA%20Lab.ipynb

In [3]:
```
# check for null values
df.isnull().sum()/df.count()*100
```

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

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
```

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()
```

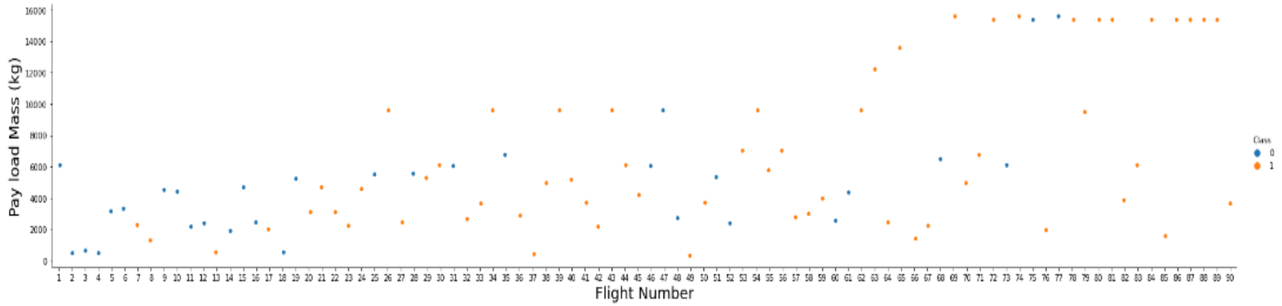
Out[6]:
```
GTO      27
ISS      21
VLEO     14
PO        9
LEO       7
SSO       5
MEO       3
ES-L1     1
HEO       1
SO        1
GEO       1
Name: Orbit, dtype: int64
```

EDA with Data Visualization

- I plot the FlightNumber vs PayloadMass and overlay the outcome of the launch and I plot the relationship between success rate of each orbit type.
- The link to the notebook is https://github.com/amm2307/Capstone_Project/blob/master/EDA%20with%20Visualization%20Lab.ipynb

In [3]:

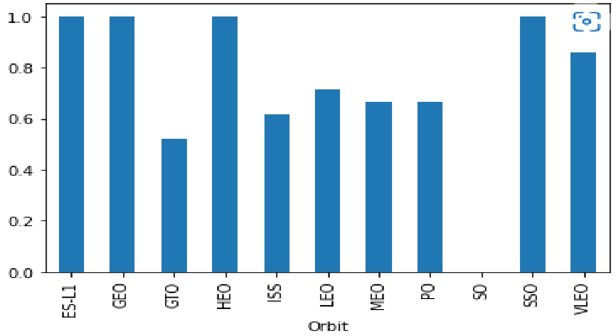
```
sns.catplot(y="PayloadMass", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Pay load Mass (kg)",fontsize=20)
plt.show()
```



In [6]:

```
# HINT use groupby method on Orbit column and get the mean of Class column
df.groupby("Orbit").mean()['Class'].plot(kind='bar')
```

Out[6]: <AxesSubplot:xlabel='Orbit'>



EDA with SQL

- I display the names of the unique launch sites in the space mission.
- I display 5 records where launch sites begin with the string 'KSC'.
- I List the date where the first succesful landing outcome in drone ship was acheived.
- The link to the notebook is https://github.com/amm2307/Capstone_Project/blob/master/EDA%20with%20SQL%20Lab.ipynb

In [7]:
```
%sql SELECT Distinct LAUNCH_SITE FROM SPACEXTBL
```
```
 * ibm_db_sa://bnz29907:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:31321/bludb
Done.
```
Out[7]:

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

In [8]:
```
%sql SELECT * FROM SPACEXTBL WHERE LAUNCH_SITE LIKE 'CCA%' LIMIT 5
```
```
 * ibm_db_sa://bnz29907:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:31321/bludb
Done.
```
Out[8]:

DATE	time__utc_	booster_version	launch_site	payload	payload_mass__kg_	orbit	customer	mission_outcome	landing__outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2012-05-22	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

In [11]:
```
%sql SELECT min(DATE) FROM SPACEXTBL WHERE LANDING__OUTCOME='Success (ground pad)'
```
```
 * ibm_db_sa://bnz29907:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:31321/bludb
Done.
```
Out[11]:

1
2015-12-22

Build an Interactive Map with Folium

- I marked all launch sites, and I added map objects such as markers, circles, lines to mark the success or failure of launches for each site on the folium map.
- I assigned the feature launch outcomes (failure or success) to class 0 and 1.i.e., 0 for failure, and 1 for success
- Using the color-labeled marker clusters, I identified which launch sites have relatively high success rate.
- I calculated the distances between a launch site to its proximities.
- The link to the notebook is https://github.com/amm2307/Capstone_Project/blob/master/Interactive%20Visual%20Analytics%20with%20Folium%20lab.ipynb

Build a Dashboard with Plotly Dash

Predictive Analysis (Classification)

- I loaded the data using numpy and pandas, transformed the data, split our data into training and testing.
- I built different machine learning models and tune different hyperparameters using GridSearchCV.
- I used accuracy as the metric for our model, improved the model using feature engineering and algorithm tuning.
- The link to the notebook is https://github.com/amm2307/Capstone_Project/blob/master/Machine%20Learning%20Prediction%20lab.ipynb

```
In [4]:  # students get this
         transform = preprocessing.StandardScaler()

In [10]: X
```

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

we can see we only have 18 test samples.

```
In [9]:  Y_test.shape

Out[9]:  (18,)
```

```
In [13]:  logreg_cv.score(X_test, Y_test)

Out[13]:  0.8333333333333334
```

Results

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

Section 2

Insights drawn from EDA

Flight Number vs. Launch Site

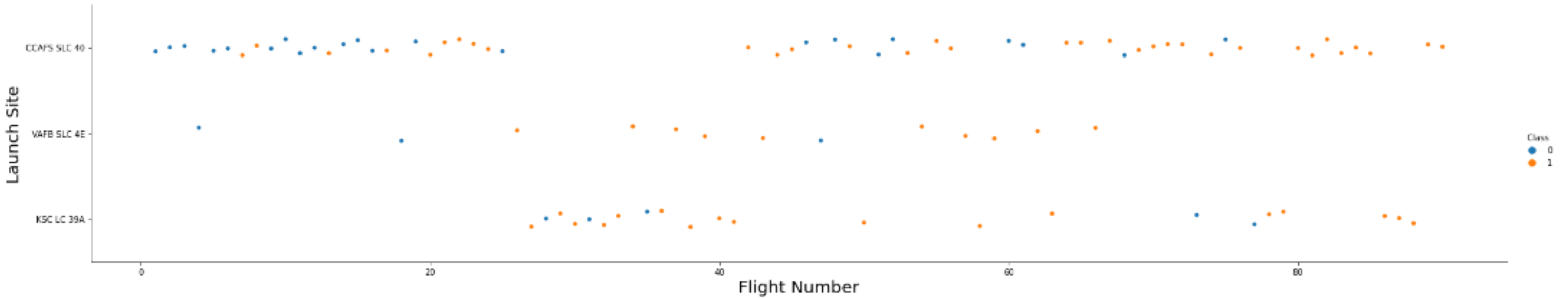
- From the plot, we found that the larger the flight amount at a launch site, the greater the success rate at a launch site.

In [4]:

```python
# 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(x="FlightNumber", y="LaunchSite", data=df, hue="Class", aspect=5)
plt.xlabel("Flight Number", fontsize=20)
plt.ylabel("Launch Site", fontsize=20)
```

Out[4]:

```
Text(23.199484374999997, 0.5, 'Launch Site')
```



Payload vs. Launch Site

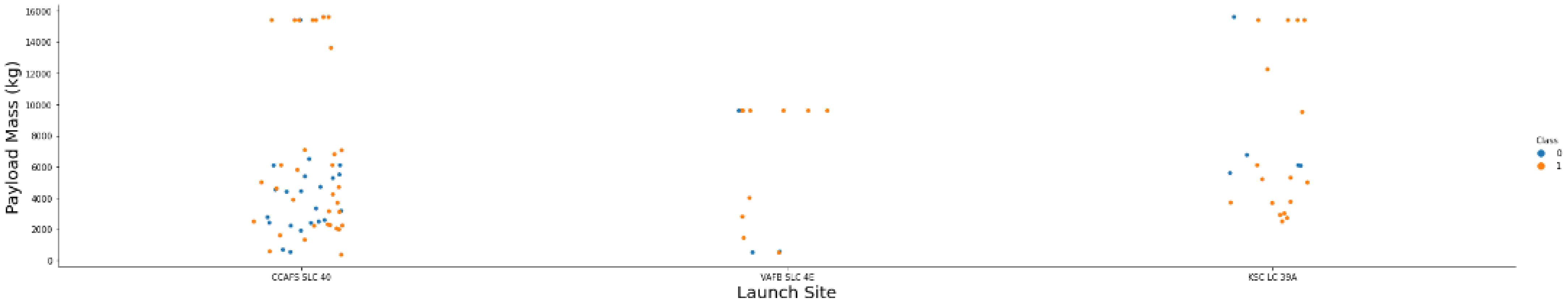
- I find for the VAFB-SLC launchsite there are no rockets launched for heavypayload mass.

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(x="LaunchSite", y="PayloadMass", hue="Class", data=df, aspect = 5)
plt.xlabel("Launch Site", size=20)
plt.ylabel("Payload Mass (kg)", size=20)
```

Out[5]:

```
Text(22.299015624999996, 0.5, 'Payload Mass (kg)')
```

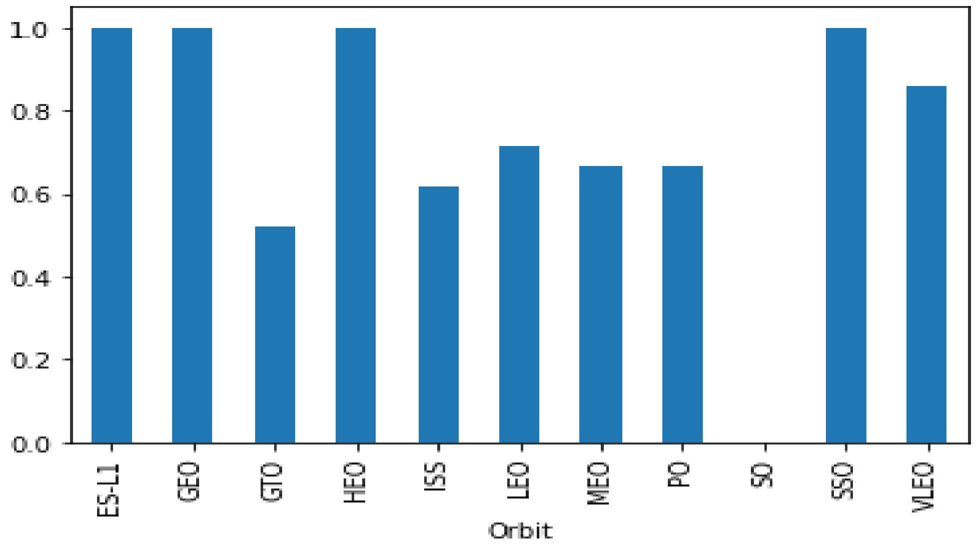


Success Rate vs. Orbit Type

- From the plot, we can see that ES-L1, GEO, HEO, SSO, VLEO had the most success rate.

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

Out[6]:
```
<AxesSubplot:xlabel='Orbit'>
```



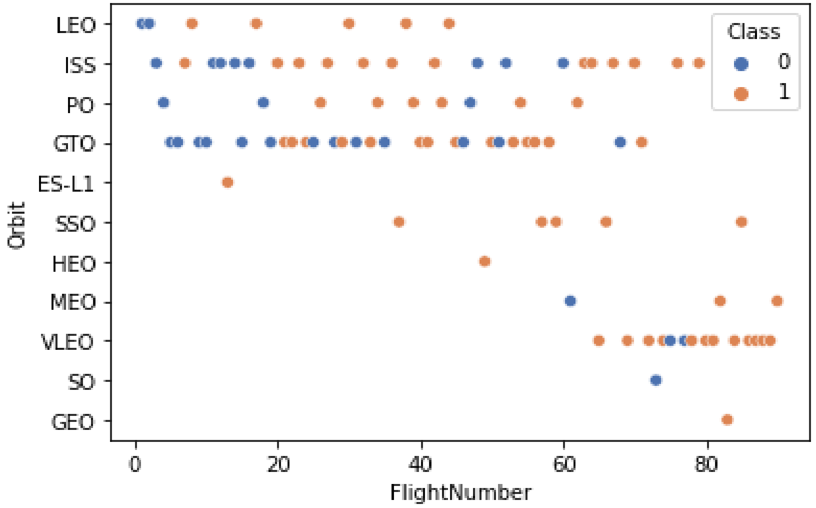
Flight Number vs. Orbit Type

- The plot below shows the Flight Number vs. Orbit type. We observe that in the LEO orbit, success is related to the number of flights whereas in the GTO orbit, there is no relationship between flight number and the orbit.

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.scatterplot(data=df, x="FlightNumber", y="Orbit", hue="Class", palette="deep")
```

Out[7]: <AxesSubplot:xlabel='FlightNumber', ylabel='Orbit'>

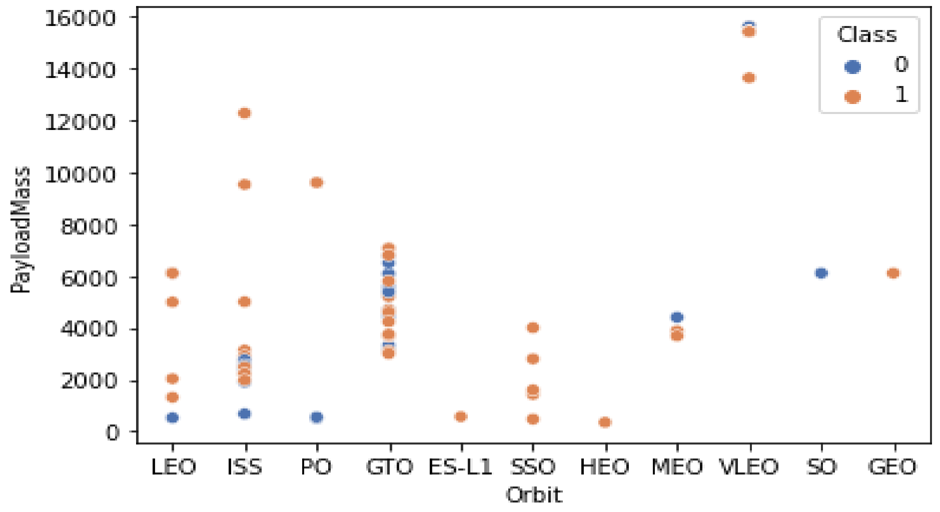


Payload vs. Orbit Type

- I can observe that with heavy payloads, the successful landing are more for PO, LEO and ISS orbits.

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(data=df, x="Orbit", y="PayloadMass", hue="Class", palette="deep")
```

Out[8]:
```
<AxesSubplot:xlabel='Orbit', ylabel='PayloadMass'>
```



Launch Success Yearly Trend

- I can observe that the sucess rate since 2013 kept increasing till 2020.

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
Extract_year(df["Date"])
zipped = zip(df['Date'], df['Orbit'], df['Outcome'],df['Class'], year)
df1=pd.DataFrame(zipped, columns=['Date', 'Orbit', 'Outcome', 'Class', 'Year'])
df1
```

Out[10]:

	Date	Orbit	Outcome	Class	Year
0	2010-06-04	LEO	None None	0	2010
1	2012-05-22	LEO	None None	0	2012
2	2013-03-01	ISS	None None	0	2013
3	2013-09-29	PO	False Ocean	0	2013
4	2013-12-03	GTO	None None	0	2013
...	...	...	...	...	...
85	2020-09-03	VLEO	True ASDS	1	2020
86	2020-10-06	VLEO	True ASDS	1	2020
87	2020-10-18	VLEO	True ASDS	1	2020
88	2020-10-24	VLEO	True ASDS	1	2020
89	2020-11-05	MEO	True ASDS	1	2020

90 rows × 5 columns

All Launch Site Names

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

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

In [7]:
```
%sql SELECT Distinct LAUNCH_SITE FROM SPACEXTBL
```

```
 * ibm_db_sa://bnz29907:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:31321/bludb
Done.
```

Out[7]:

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

Launch Site Names Begin with 'KSC'

- I Display 5 records where launch sites begin with the string 'KSC'

In [8]:
```
%sql SELECT * FROM SPACEXTBL WHERE LAUNCH_SITE LIKE 'CCA%' LIMIT 5
```

```
 * ibm_db_sa://bnz29907:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:31321/bludb
Done.
```

Out[8]:

DATE	time_utc_	booster_version	launch_site	payload	payload_mass_kg_	orbit	customer	mission_outcome	landing_outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2012-05-22	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Total Payload Mass

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

```
In [9]: %sql SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE CUSTOMER='NASA (CRS)'

 * ibm_db_sa://bnz29907:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:31321/bludb
Done.

Out[9]:   1
      45596
```

Average Payload Mass by F9 v1.1

- I calculated the average payload mass carried by booster version F9 v1.1 as 2928

```
In [10]: %sql SELECT AVG(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE BOOSTER_VERSION='F9 v1.1'

 * ibm_db_sa://bnz29907:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:31321/bludb
Done.

Out[10]:    1
         2928
```

First Successful Ground Landing Date

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

```
In [11]: %sql SELECT min(DATE) FROM SPACEXTBL WHERE LANDING__OUTCOME='Success (ground pad)'

 * ibm_db_sa://bnz29907:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:31321/bludb
Done.

Out[11]:
         1
2015-12-22
```

Successful Drone Ship Landing with Payload between 4000 and 6000

- I 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

In [12]:
```
%sql SELECT BOOSTER_VERSION FROM SPACEXTBL WHERE PAYLOAD_MASS__KG_ between 4000 and 6000 AND LANDING__OUTCOME='Success (drone ship)'
```

```
 * ibm_db_sa://bnz29907:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:31321/bludb
Done.
```

Out[12]:

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

Total Number of Successful and Failure Mission Outcomes

- I used wildcard like '%' to filter for WHERE MissionOutcome was a success or a failure.

In [13]:

```
%sql SELECT COUNT(*) FROM SPACEXTBL WHERE MISSION_OUTCOME LIKE '%Success%' OR MISSION_OUTCOME LIKE '%Failure%'
```

```
 * ibm_db_sa://bnz29907:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:31321/bludb
Done.
```

Out[13]:

1
101

Boosters Carried Maximum Payload

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

In [14]:
```
%sql SELECT BOOSTER_VERSION FROM SPACEXTBL WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTBL)
```

```
 * ibm_db_sa://bnz29907:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:31321/bludb
Done.
```

Out[14]:

booster_version
F9 B5 B1048.4
F9 B5 B1049.4
F9 B5 B1051.3
F9 B5 B1056.4
F9 B5 B1048.5
F9 B5 B1051.4
F9 B5 B1049.5
F9 B5 B1060.2
F9 B5 B1058.3
F9 B5 B1051.6
F9 B5 B1060.3
F9 B5 B1049.7

2015 Launch Records

- I 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

In [15]:
```
%sql SELECT TO_CHAR(TO_DATE(MONTH("DATE"), 'MM'), 'MONTH') AS MONTH_NAME, \
    LANDING__OUTCOME AS LANDING__OUTCOME, \
    BOOSTER_VERSION AS BOOSTER_VERSION, \
    LAUNCH_SITE AS LAUNCH_SITE \
    FROM SPACEXTBL WHERE LANDING__OUTCOME = 'Failure (drone ship)' AND "DATE" LIKE '%2015%'
```

```
 * ibm_db_sa://bnz29907:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:31321/bludb
Done.
```

Out[15]:

month_name	landing__outcome	booster_version	launch_site
JANUARY	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
APRIL	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

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

- I 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.
- I applied the GROUP BY clause to group the landing outcomes and the ORDER BY clause to order the grouped landing outcome in descending order.

In [16]:

```
%sql SELECT "DATE", COUNT(LANDING__OUTCOME) as COUNT FROM SPACEXTBL \
    WHERE "DATE" BETWEEN '2010-06-04' and '2017-03-20' AND LANDING__OUTCOME LIKE '%Success%' \
    GROUP BY "DATE" \
    ORDER BY COUNT(LANDING__OUTCOME) DESC
```

```
 * ibm_db_sa://bnz29907:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:31321/bludb
Done.
```

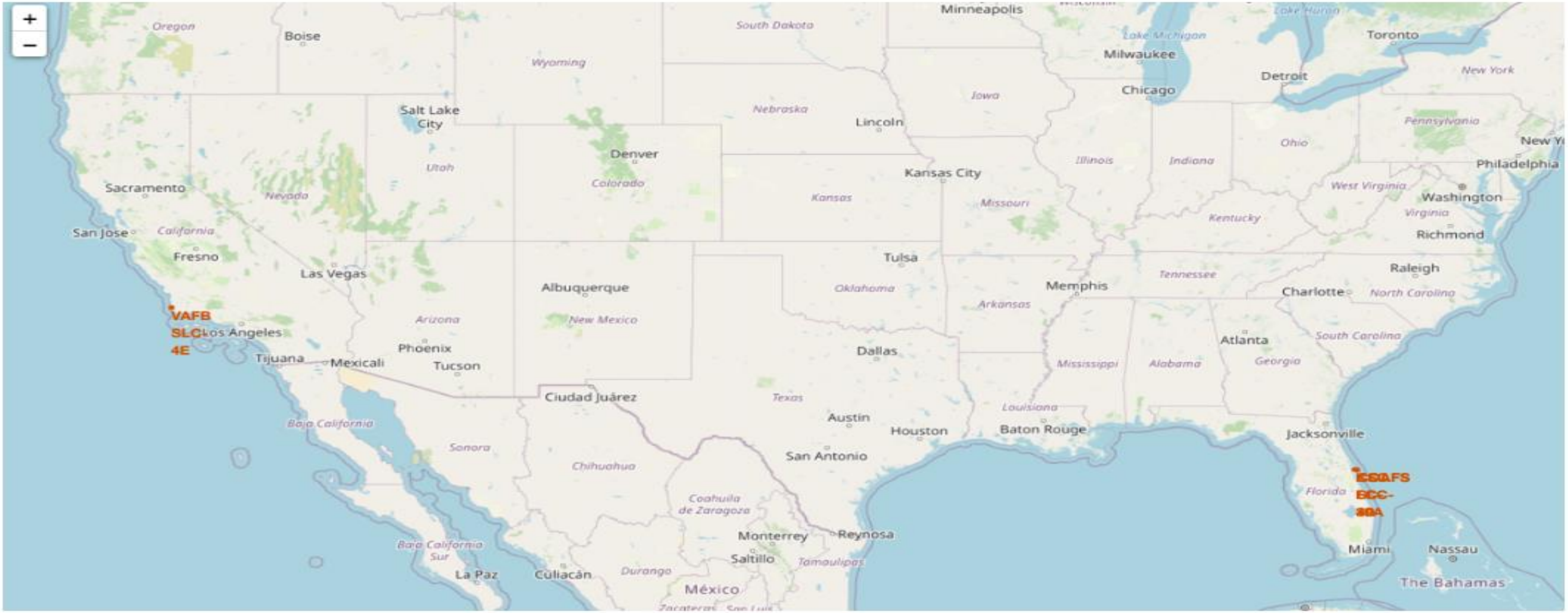
Out[16]:

DATE	COUNT
2015-12-22	1
2016-04-08	1
2016-05-06	1
2016-05-27	1
2016-07-18	1
2016-08-14	1
2017-01-14	1
2017-02-19	1

Section 3

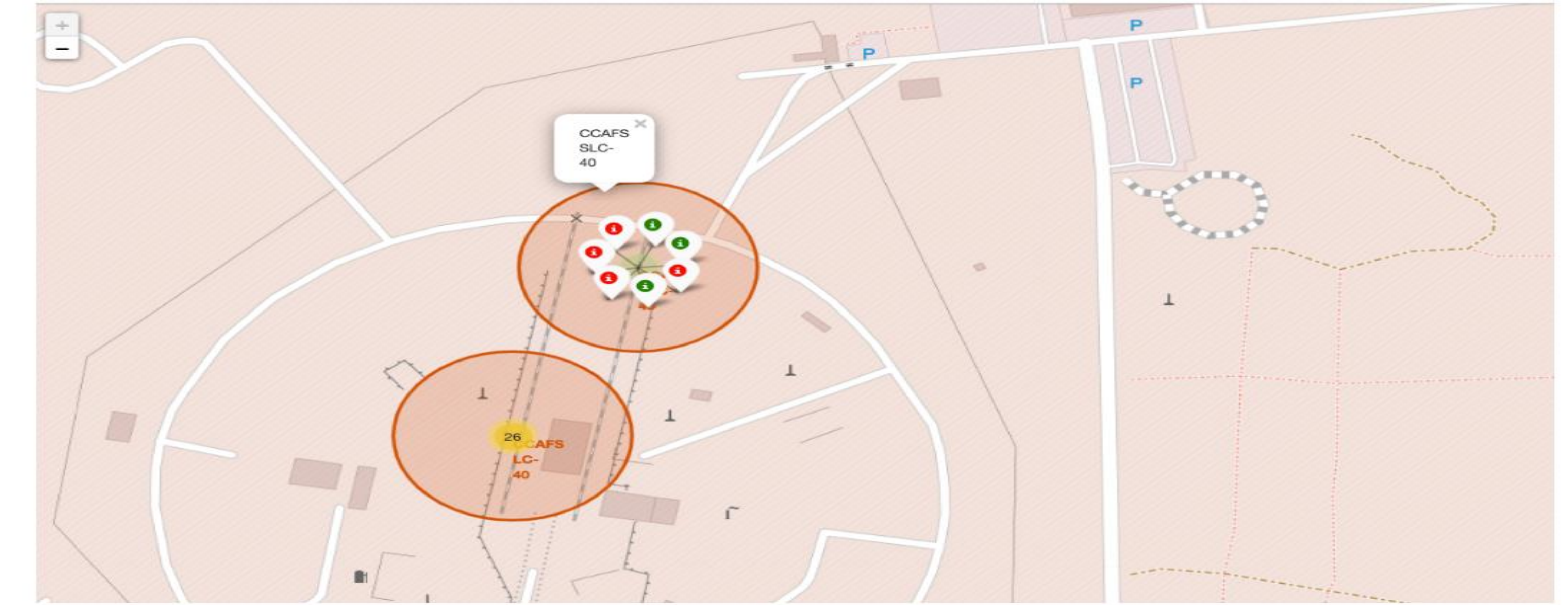
Launch Sites Proximities Analysis

All launch sites global map markers



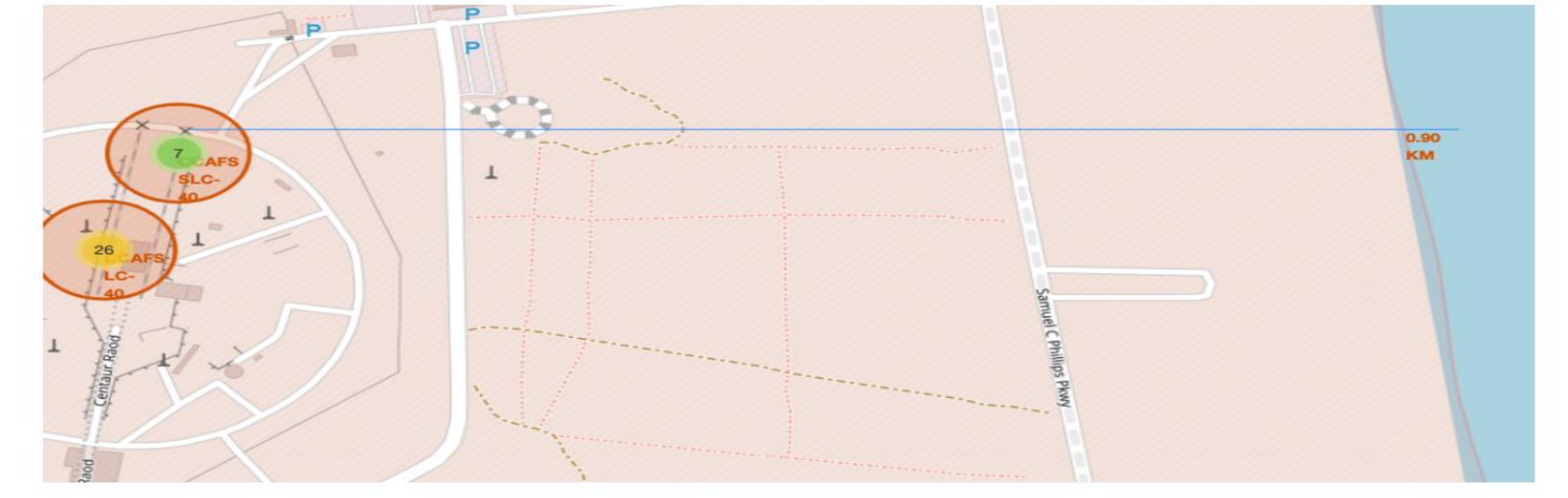
- I can see that the SpaceX launch sites are in the United States of America coasts, Florida and California.

Markers showing launch sites with color labels



- Green Marker shows successful Launches and Red Marker shows Failures.

Launch Site distance to landmarks



- Are launch sites in close proximity to railways? No.
- 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.

Section 4

Build a Dashboard with Plotly Dash

Pie chart showing the success percentage achieved by each launch site

Pie chart showing the Launch site with the highest launch success ratio

Scatter plot of Payload vs Launch Outcome for all sites, with different payload selected in the range slider

Section 5

Predictive Analysis (Classification)

Classification Accuracy

- The decision tree classifier is the model with the highest classification accuracy

Confusion Matrix

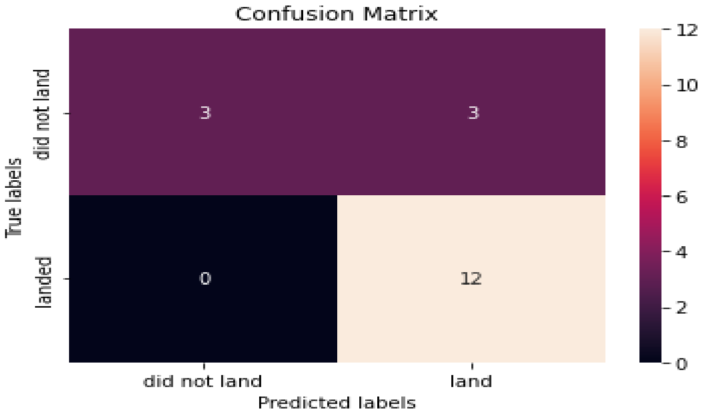
- The confusion matrix for the decision tree classifier shows that the classifier can distinguish between the different classes. The major problem is the false positives .i.e., unsuccessful landing marked as successful landing by the classifier.

In [13]:
```
logreg_cv.score(X_test, Y_test)
```

Out[13]:
```
0.8333333333333334
```

Lets look at the confusion matrix:

In [14]:
```
yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



Conclusions

I can conclude that:

- Launch success rate started to increase in 2013 till 2020.
- Orbits ES-L1, GEO, HEO, SSO, VLEO had the most success rate.
- KSC LC-39A had the most successful launches of any sites.
- The Decision tree classifier is the best machine learning algorithm.

Appendix

```
In [9]:
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_par

X.head(100)
```

Out[9]:

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	...	Serial_B1058	Serial_B1059	Serial_B1060	Serial_B1062
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	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
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
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
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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
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
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
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
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
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

90 rows × 83 columns

Thank you!