IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

<Name>
<Date>

Outline

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

Executive Summary

- Summary of methodologies

  Data Collection through API

  Data Collection with Web Scarping

  Data Wrangling

  Exploratory Data Analysis with Data Visualisation

  Interactive Visual Analytics with Folium

  Machine Learning Prediction

- Summary of all results

  Exploratory Data Analysis result

  Interactive analytics screenshots

  Predictive Analytics result with Machine Learning lab

Introduction

SpaceX is a revolutionary company who has disrupt the space industry by offering a rocket launches specifically Falcon 9 as low as 62 million dollars; while other providers cost upward of 165 million dollar each. Most of this saving thanks to SpaceX astounding idea to reuse the first stage of the launch by re-land the rocket to be used on the next mission. Repeating this process will make the price down even further. As a data scientist of a startup rivaling SpaceX, the goal of this project is to create the machine learning pipeline to predict the landing outcome of the first stage in the future. This project is crucial in identifying the right price to bid against SpaceX for a rocket launch.

The problems included:

- Identifying all factors that influence the landing outcome.

- The relationship between each variables and how it is affecting the outcome.

- The best condition needed to increase the probability of successful landing

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - Data was collected using SpaceX REST API and web scarping from Wikipedia
- Perform data wrangling
  - Data was processed using one-hot encoding for 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

Data collection is the process of gathering and measuring information on targeted variables in an established system, which then enables one to answer relevant questions and evaluate outcomes. As mentioned, the dataset was collected by REST API and Web Scrapping from Wikipedia

For REST API, its started by using the get request. Then, we decoded the response content as Json and turn it into a pandas dataframe using json_normalize(). We then cleaned the data, checked for missing values and fill with whatever needed.

For web scrapping, we will use the BeautifulSoup to extract the launch records as HTML table, parse the table and convert it to a pandas dataframe for further analysis.

Data Collection – SpaceX API

From:
[Applied-Data-Science-Coursera/jupyter-labs-spacex-data-collection-api.ipynb at main · Momitha/Applied-Data-Science-Coursera (github.com)]

```
spacex_url="https://api.spacexdata.com/v4/launches/past"
```

```
response = requests.get(spacex_url)
```

```
static_json_path='ex.json'
```

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

```
response.status_code==200
```

```
True
```

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

```
# Use json_normalize meethod to convert the json result into a dataframe
USE_STATIC_RESULT = True

if USE_STATIC_RESULT:
    import json
    data = pd.json_normalize(json.load(open(static_json_path, 'r')))
else:
    data = pd.json_normalize(response.json())
```

Get request for rocket launch data using API

Use json_normalize method to convert json result to dataframe

Performed data cleaning and filling the missing value

Data Collection - Scraping

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

```
200
```

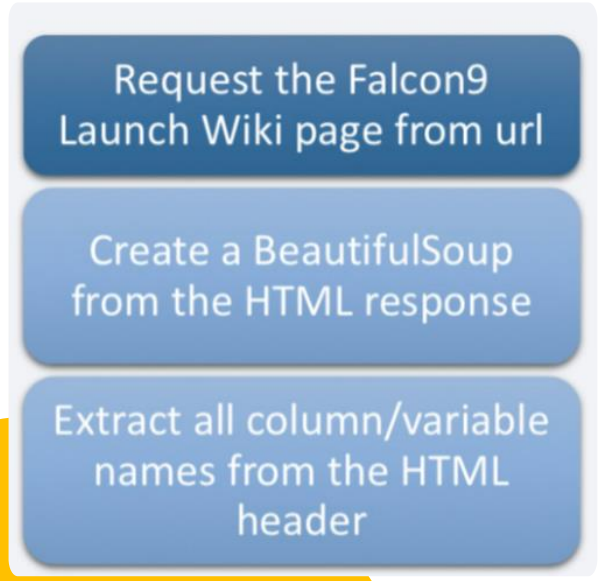
Create a `BeautifulSoup` object from the HTML `response`

```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(response.text)
```

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

```
# Use soup.title attribute
print(soup.title)
```

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



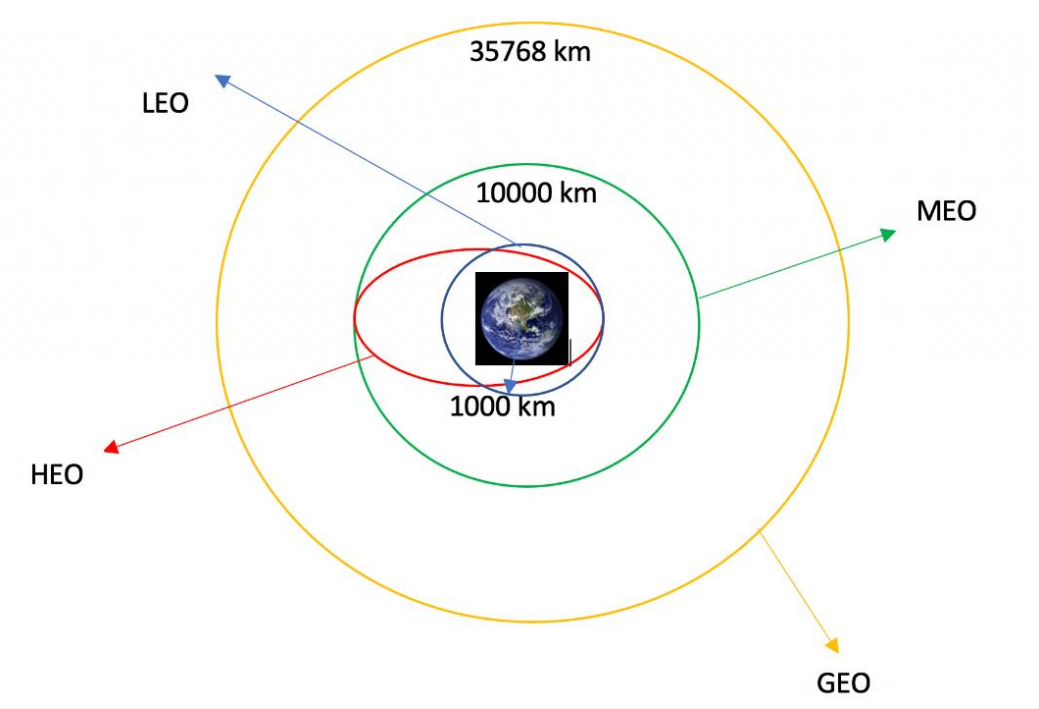
- From:[Applied-Data-Science-Coursera/jupyter-labs-webscraping.ipynb at main · Momitha/Applied-Data-Science-Coursera (github.com)]

```
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)
            launch_dict['Flight No.'].append(flight_number)
            datatimelist=date_time(row[0])

            # Date value
            # TODO: Append the date into launch_dict with key `Date`
```

Data Wrangling

From:[Applied-Data-Science-Coursera/IBM-DS0321EN-SkillsNetwork_labs_module_1_L3_labs-jupyter-spacex-data_wrangling_jupyterlite.jupyterlite.ipynb at main · Momitha/Applied-Data-Science-Coursera (github.com)]

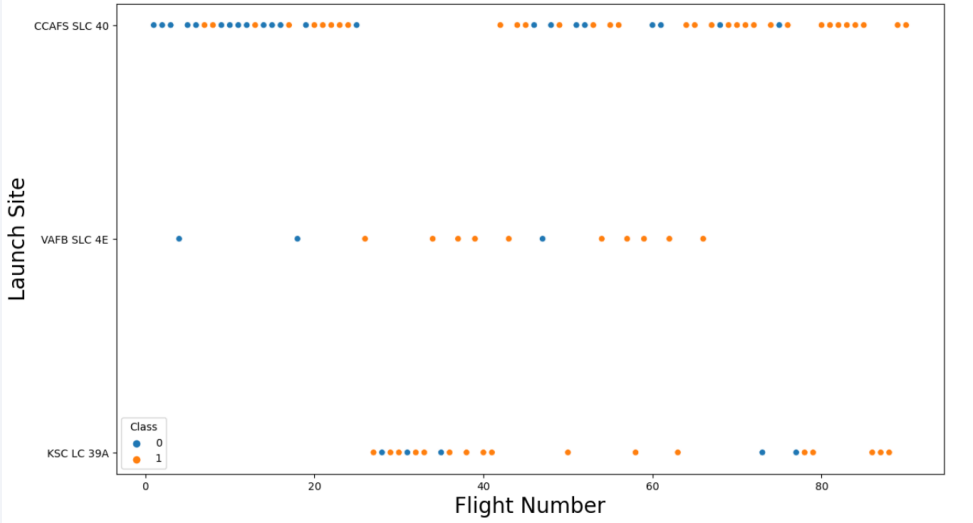


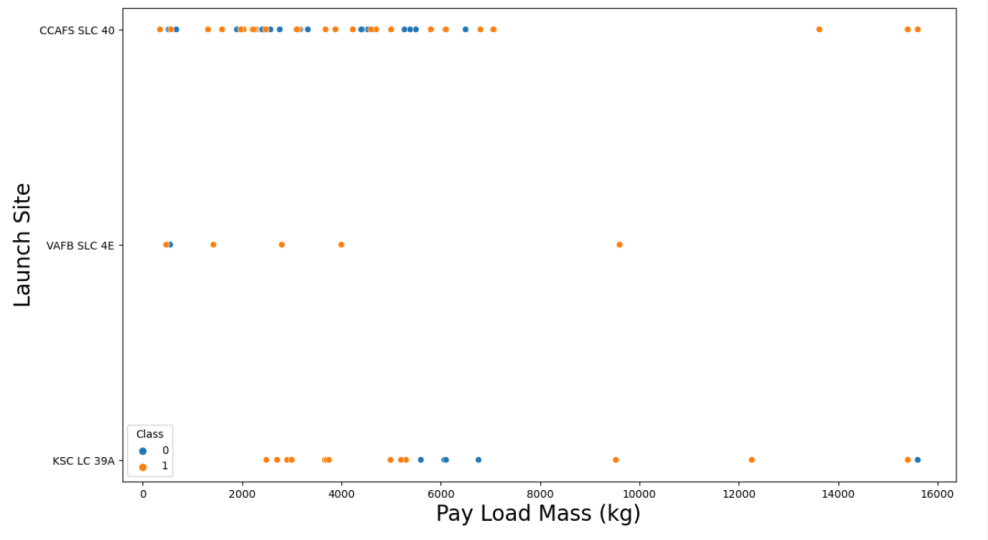
Data Wrangling is the process of cleaning and unifying messy and complex data sets for easy access and Exploratory Data Analysis (EDA).

We will first calculate the number of launches on each site, then calculate the number and occurrence of mission outcome per orbit type.

We then create a landing outcome label from the outcome column. This will make it easier for further analysis, visualization, and ML. Lastly, we will export the result to a CSV.

EDA with Data Visualization





[Applied-Data-Science-Coursera/IBM-DS0321EN-SkillsNetwork_labs_module_2_jupyter-labs-eda-dataviz.ipynb.jupyterlite.ipynb at main · Momitha/Applied-Data-Science-Coursera (github.com)]

We first started by using scatter graph to find the relationship between the attributes such as between:

- Payload and Flight Number.

- Flight Number and Launch Site.

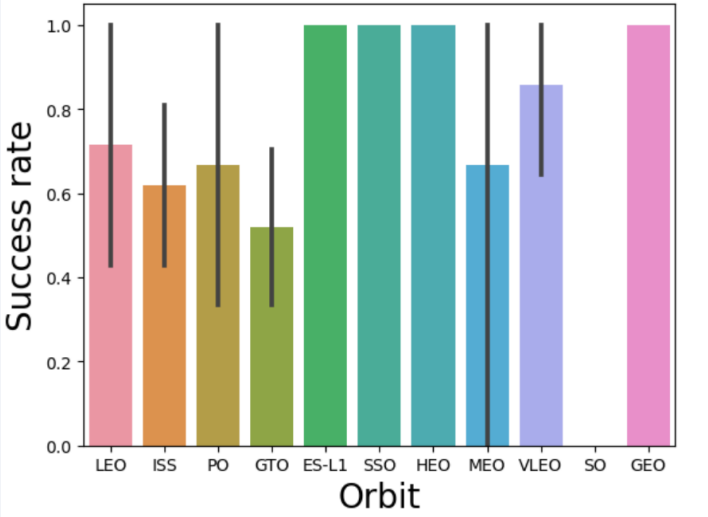
- Payload and Launch Site.

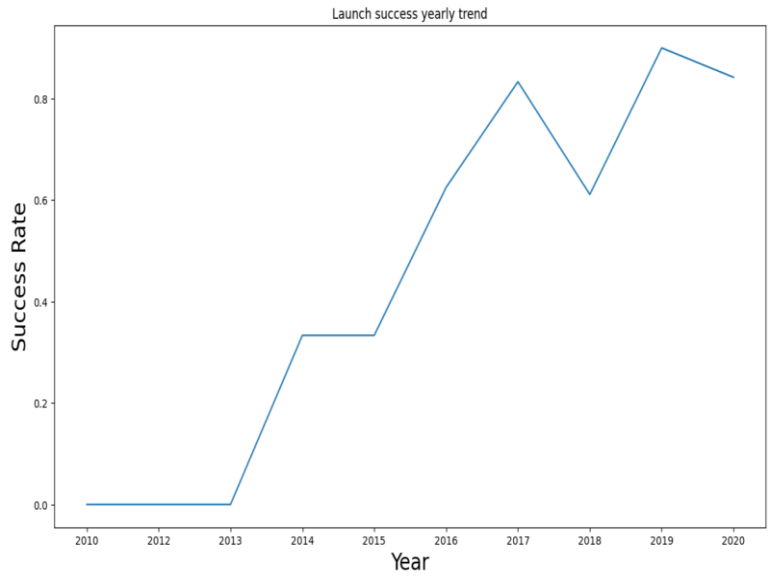
- Flight Number and Orbit Type.

- Payload and Orbit Type.

Scatter plots show dependency of attributes on each other. Once a pattern is determined from the graphs. It's very easy to see which factors affecting the most to the success of the landing outcomes.

EDA with Data Visualisation





[Applied-Data-Science-Coursera/IBM-DS0321EN-SkillsNetwork_labs_module_2_jupyter-labs-eda-dataviz.ipynb.jupyterlite.ipynb at main · Momitha/Applied-Data-Science-Coursera (github.com)]

Once we get a hint of the relationships using scatter plot. We will then use further visualization tools such as bar graph and line plots graph for further analysis.

Bar graphs is one of the easiest way to interpret the relationship between the attributes. In this case, we will use the bar graph to determine which orbits have the highest probability of success.

We then use the line graph to show a trends or pattern of the

attribute over time which in this case, is used for see the

launch success yearly trend.

We then use Feature Engineering to be used in success prediction in the future module by created the dummy. variables to categorical columns.

EDA with SQL

Applied-Data-Science-Coursera/jupyter-labs-eda-sql-coursera_sqllite.ipynb at main · Momitha/Applied-Data-Science-Coursera (github.com)

Using SQL, we had performed many queries to get better understanding of the dataset, Ex:

- Displaying the names of the launch sites.
- Displaying 5 records where launch sites begin with the string 'CCA'.
- Displaying the total payload mass carried by booster launched by NASA (CRS).
- Displaying the average payload mass carried by booster version F9 v1.1. - Listing the date when the first successful landing outcome in ground pad was achieved.
- Listing the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000.
- Listing the total number of successful and failure mission outcomes.
- Listing the names of the booster_versions which have carried the maximum payload mass.
- Listing the failed landing_outcomes in drone ship, their booster versions, and launch sites names for in year 2015.
- Rank the count of landing outcomes or success between the date 2010-06-04 and 2017-03-20, in descending order.

Build an Interactive Map with Folium

Applied-Data-Science-Coursera/IBM-DS0321EN-SkillsNetwork_labs_module_3_lab_jupyter_launch_site_location.jupyterlite.ipynb at main · Momitha/Applied-Data-Science-Coursera (github.com)

To visualize the launch data into an interactive map. We took the latitude and longitude coordinates at each launch site and added a circle marker around each launch site with a label of the name of the launch site.

We then assigned the dataframe launch_outcomes (failure, success) to classes 0 and 1 with Red and Green markers on the map in MarkerCluster().

We then used the Haversine's formula to calculated the distance of the launch sites to various landmark to find answer to the questions of:

- How close the launch sites with railways, highways and coastlines?

- How close the launch sites with nearby cities?

Build a Dashboard with Plotly Dash

Build a Dashboard with Plotly Dash

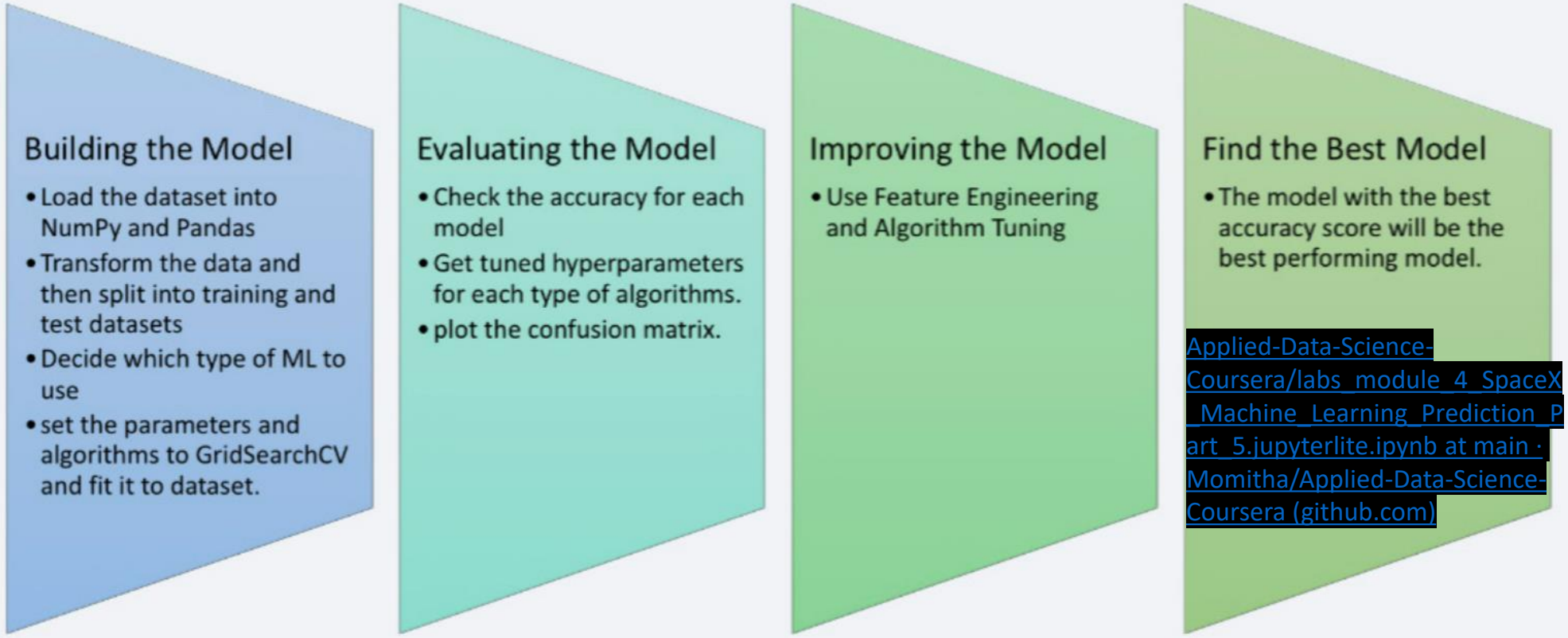
• We built an interactive dashboard with Plotly dash which allowing the user to play around with the data as they need.

• We plotted pie charts showing the total launches by a certain sites.

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

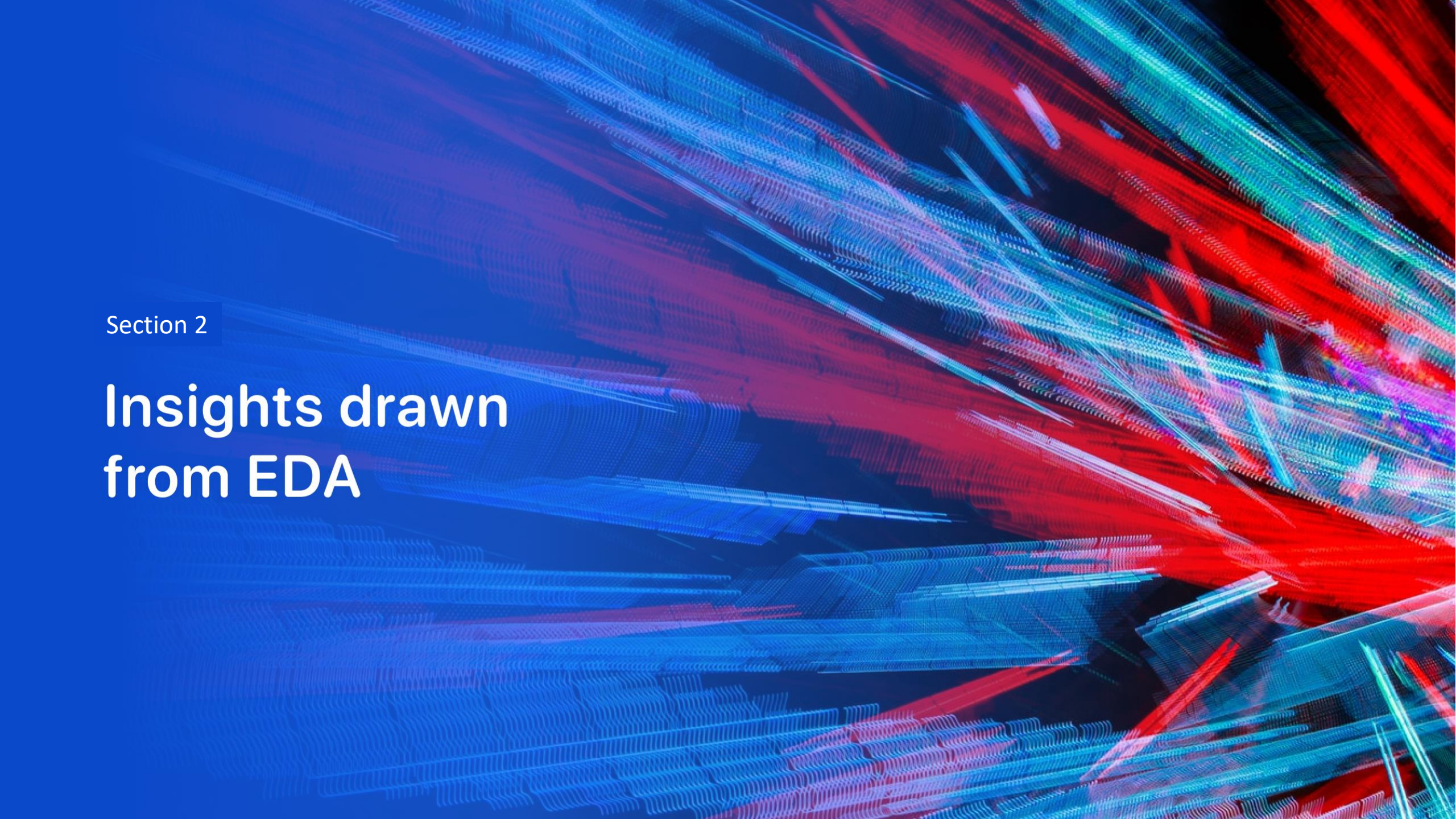
Applied-Data-Science-Coursera/labs_module_4_SpaceX_Machine_Learning_Prediction_Part_5.jupyterlite.ipynb at main · Momitha/Applied-Data-Science-Coursera (github.com)

Predictive Analysis (Classification)



Results

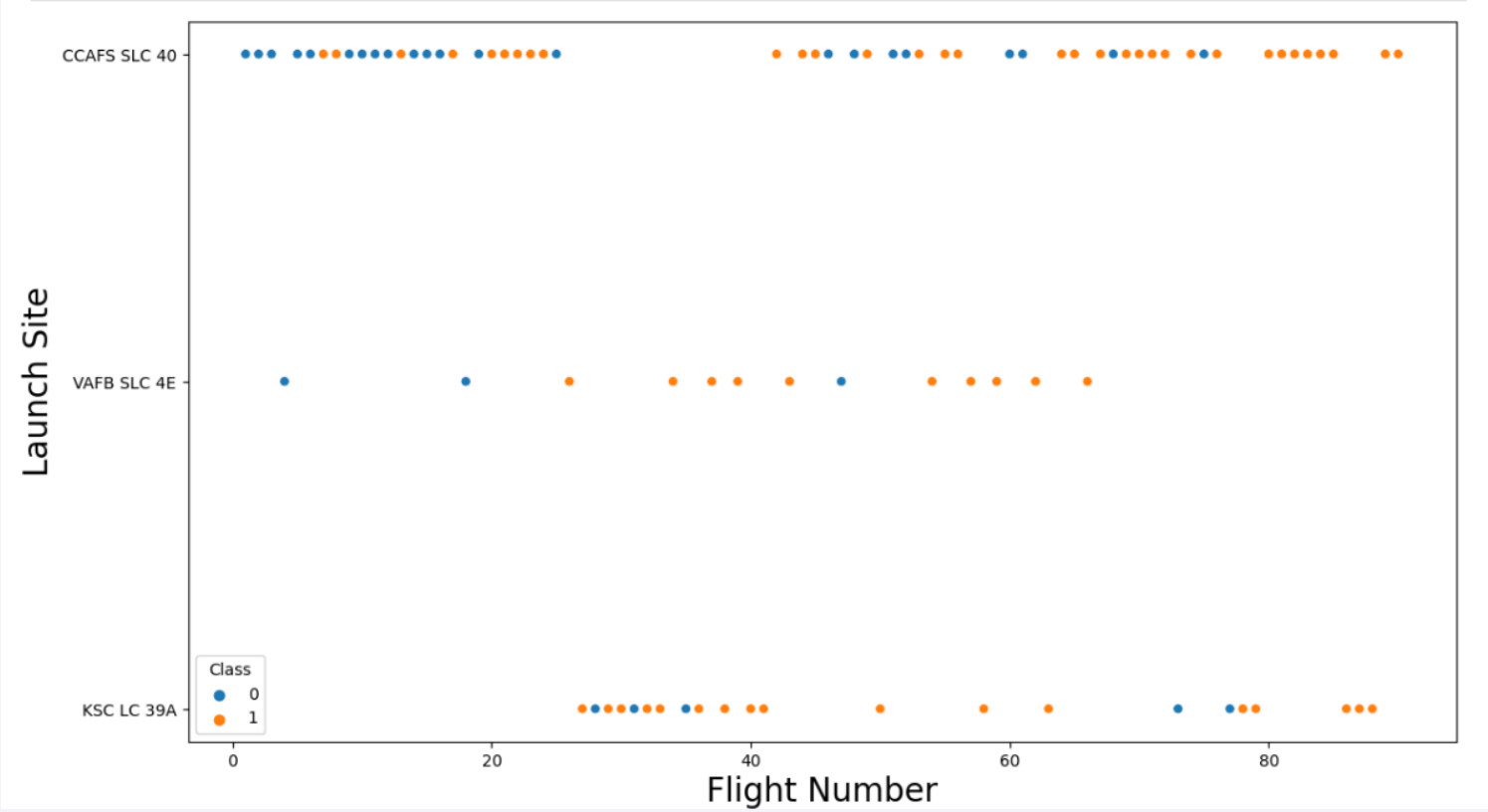
- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results


Section 2
Insights drawn from EDA

Flight Number vs. Launch Site

This scatter plot shows that the larger the flights amount of the launch site, the greater will be the success rate.

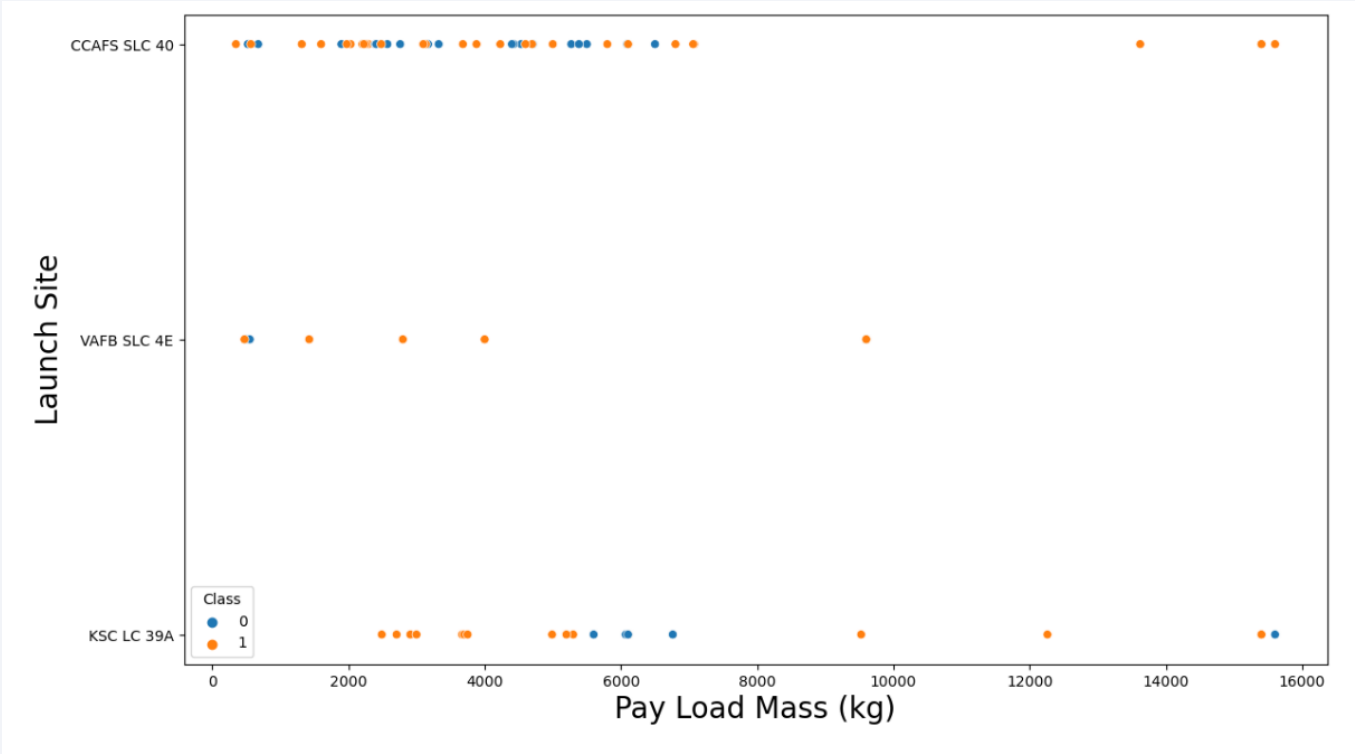
However, site CCAFS SLC40 shows the least pattern of this.



Payload vs. Launch Site

This scatter plot shows once the pay load mass is greater than 7000kg, the probability of the success rate will be highly increased.

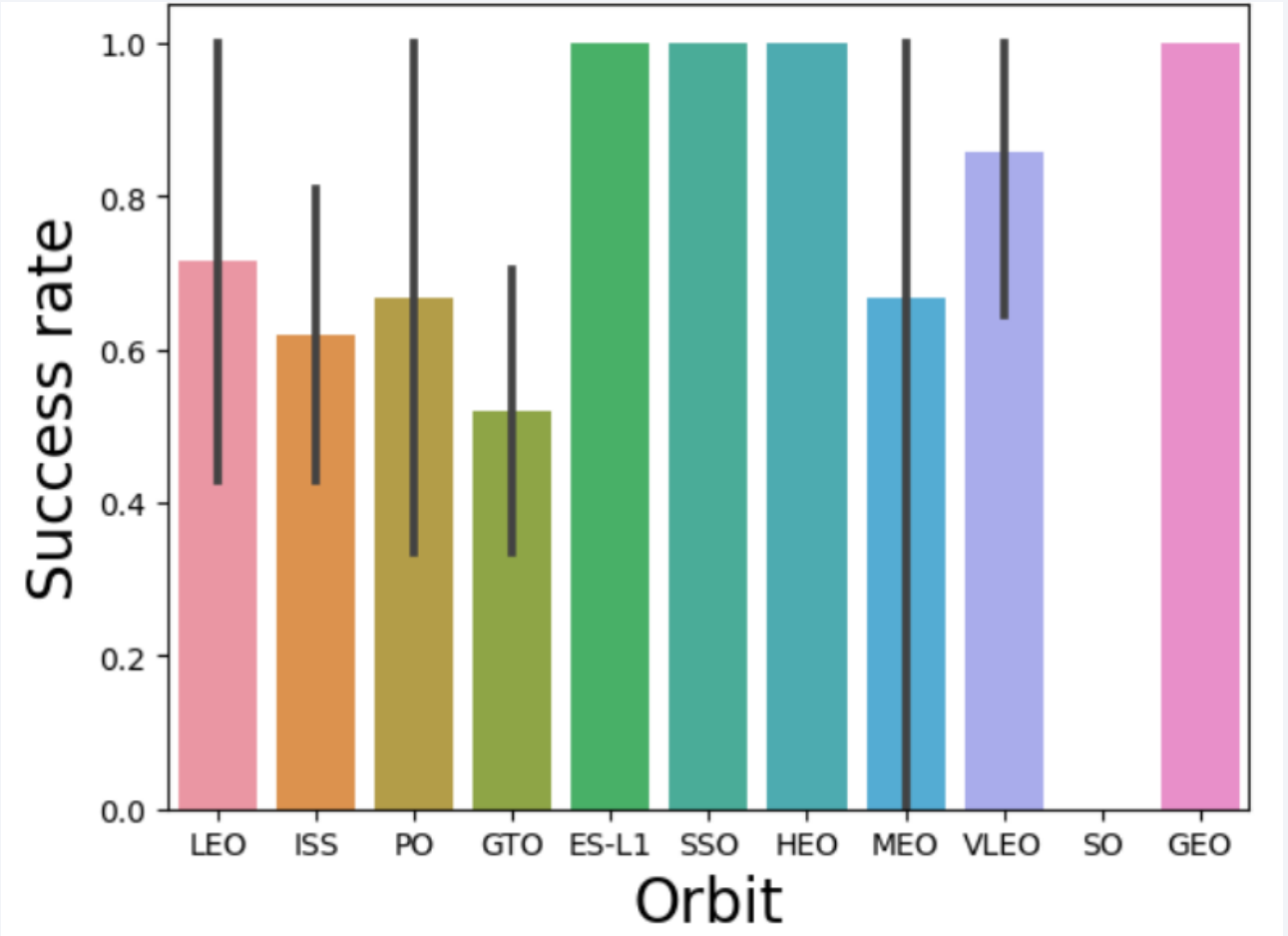
However, there is no clear pattern to say the launch site is dependent to the pay load mass for the success rate.



Success Rate vs. Orbit Type

This figure depicted the possibility of the orbits to influences the landing outcomes as some orbits has 100% success rate such as SSO, HEO, GEO AND ES-L1 while SO orbit produced 0% rate of success.

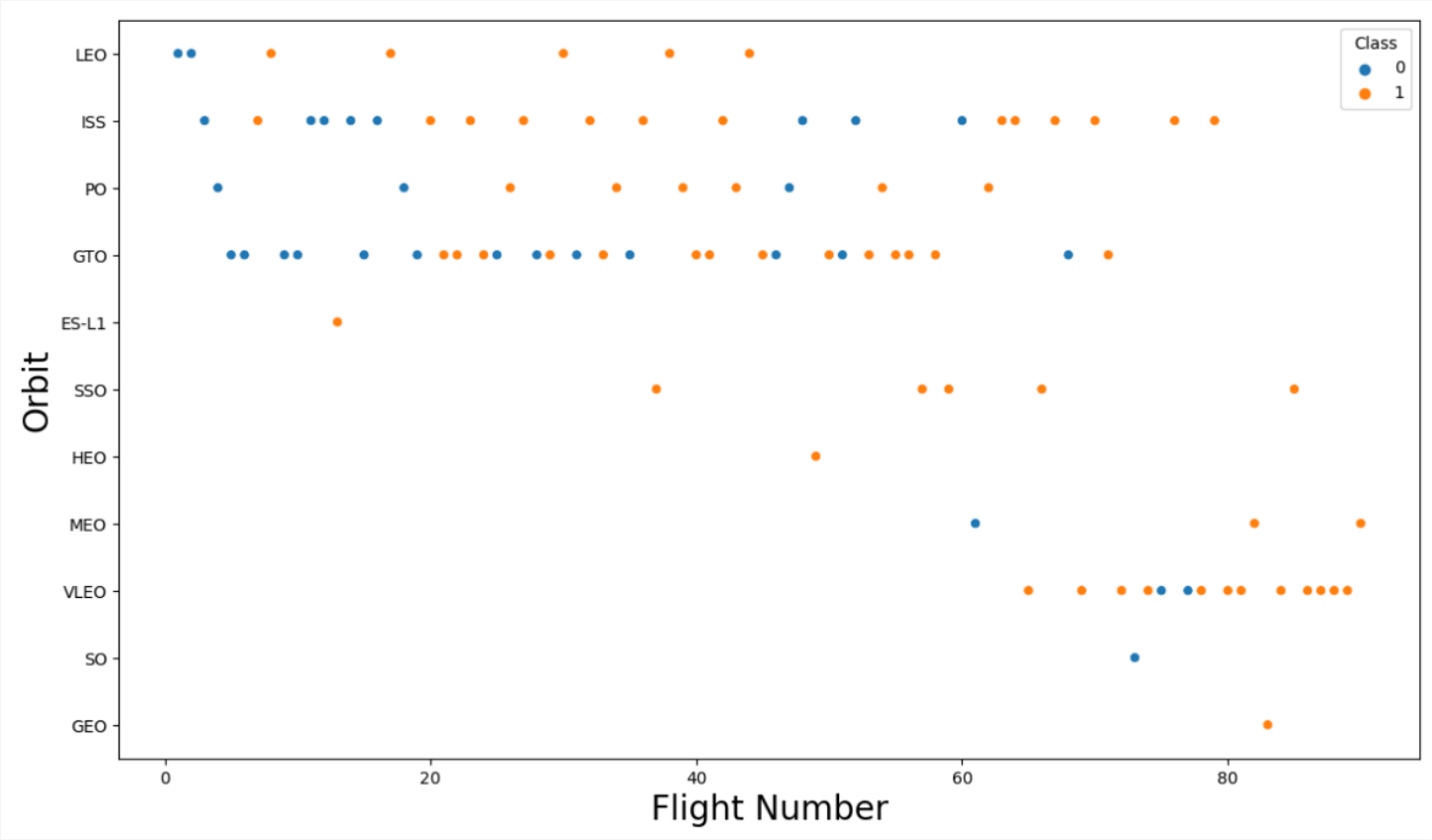
However, deeper analysis show that some of this orbits has only 1 occurrence such as GEO, SO, HEO and ES-L1 which mean this data need more dataset to see pattern or trend before we draw any conclusion.



Flight Number vs. Orbit Type

- This scatter plot shows that generally, the larger the flight number on each orbits, the greater the success rate (especially LEO orbit) except for GTO orbit which depicts no relationship between both attributes.

  Orbit that only has 1 occurrence should also be excluded from above statement as it's needed more dataset.

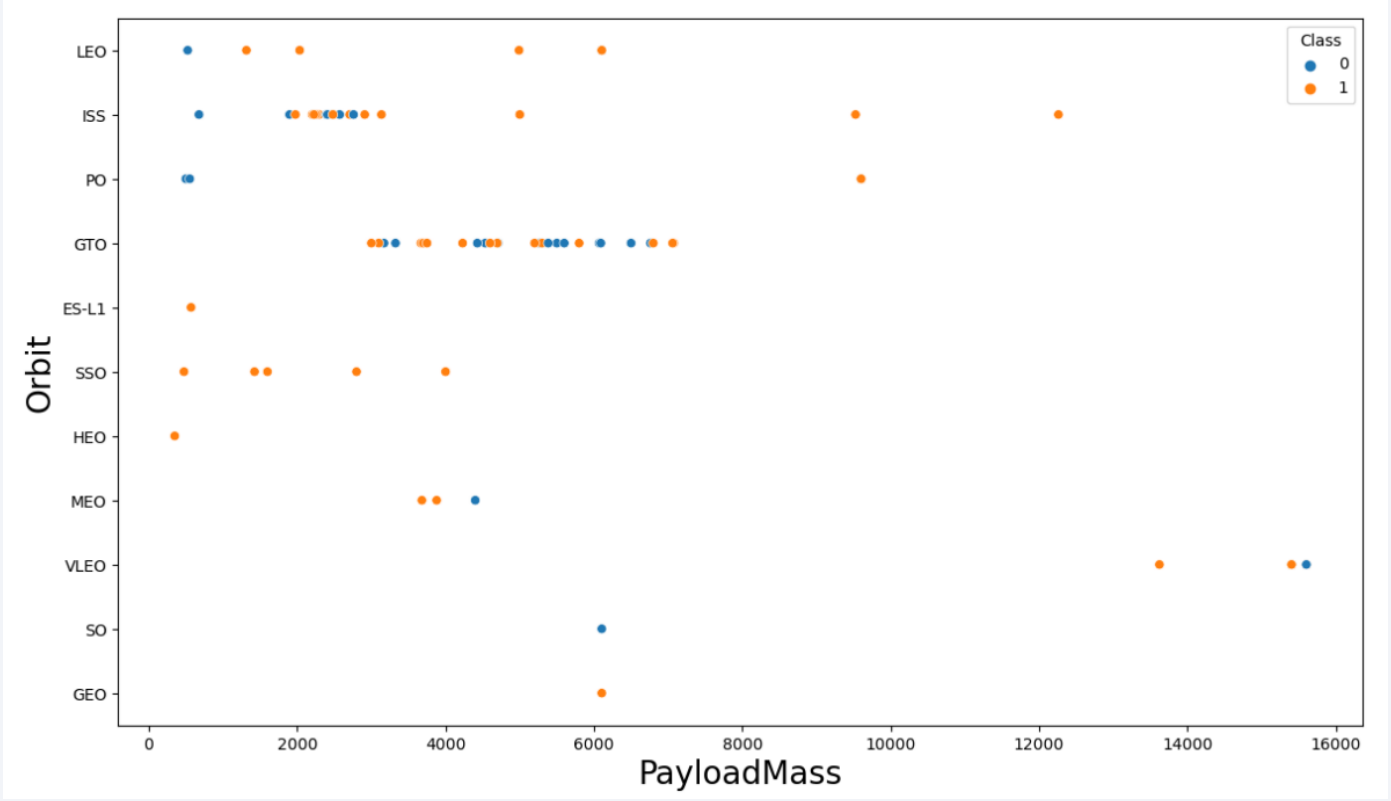


Payload vs. Orbit Type

- Heavier payload has positive impact on LEO, ISS and PO orbit. However, it has negative impact on MEO and VLEO orbit.

  GTO orbit seem to depict no relation between the attributes.

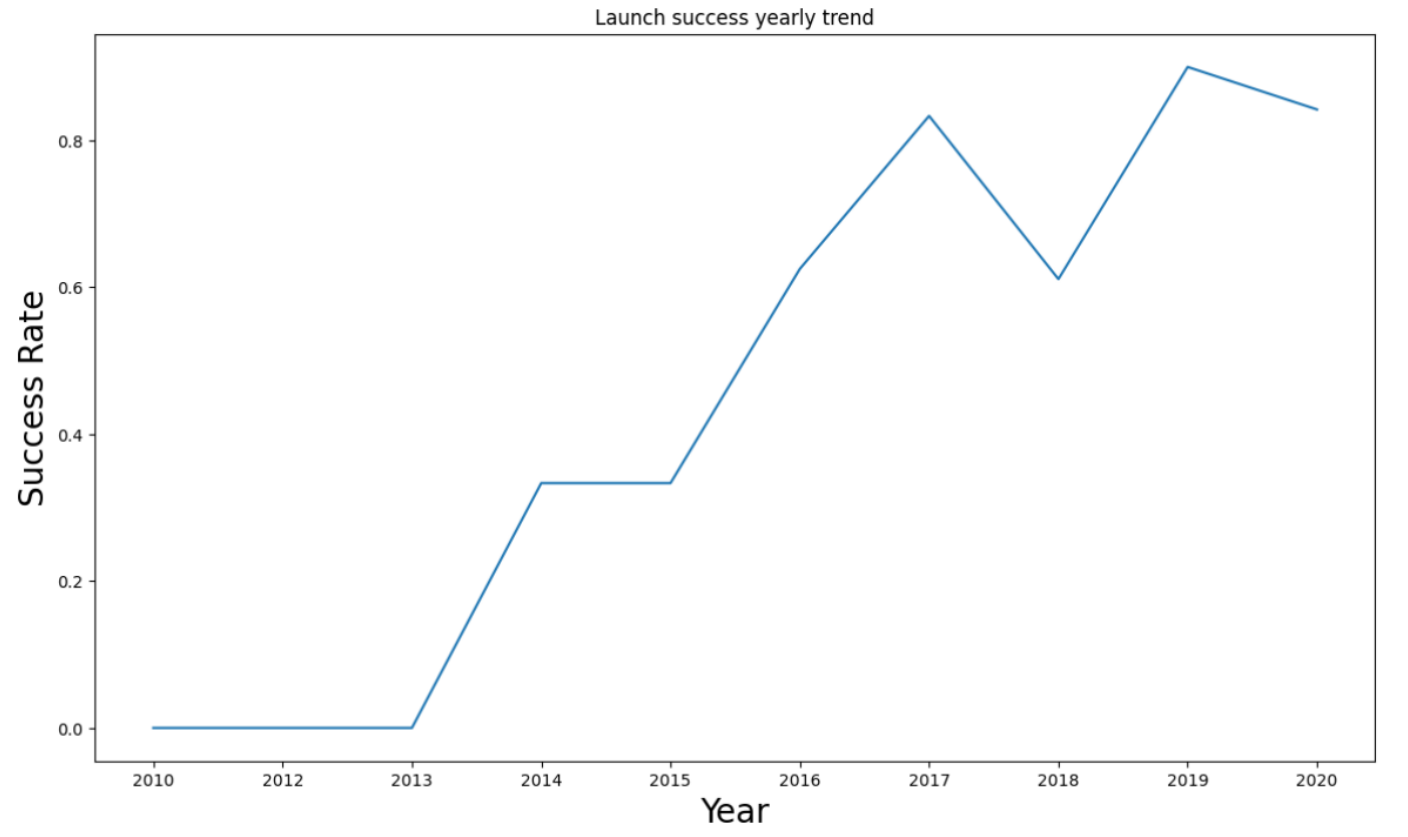
  Meanwhile, again, SO, GEO and HEO orbit need more dataset to see any pattern or trend.



Launch Success Yearly Trend

- This figures clearly depicted and increasing trend from the year 2013 until 2020.

  If this trend continue for the next year onward. The success rate will steadily increase until reaching 1/100% success rate.



All Launch Site Names

We 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

```
%sql  SELECT DISTINCT LAUNCH_SITE  FROM SPACEXTBL;
```

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

Launch_Site
CCAFS LC-40
VAFB SLC-4E
KSC LC-39A
CCAFS SLC-40
None

Launch Site Names Begin with 'CCA'

We used the query below to display 5 records where launch sites begin with ‘CCA’

```
%sql SELECT * FROM SPACEXTBL WHERE LAUNCH_SITE LIKE 'CCA%' LIMIT 5;
```

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

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing
06/04/2010	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0.0	LEO	SpaceX	Success	Failure (
12/08/2010	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0.0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (
22/05/2012	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525.0	LEO (ISS)	NASA (COTS)	Success	N
10/08/2012	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500.0	LEO (ISS)	NASA (CRS)	Success	N
03/01/2013	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677.0	LEO (ISS)	NASA (CRS)	Success	N

Total Payload Mass

We calculated the total payload carried by boosters from NASA as below,

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

```
%sql SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE CUSTOMER = 'NASA(CRS)';
```

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

SUM(PAYLOAD_MASS__KG_)
None

Average Payload Mass by F9 v1.1

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

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

```
%sql SELECT AVG(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE BOOSTER_VERSION = 'F9 v1.1'
```

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

AVG(PAYLOAD_MASS__KG_)
2928.4

First Successful Ground Landing Date

- We use the min() function to find the result.

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

Hint:Use min function

```
%sql select min(DATE) from SPACEXTBL where Landing_Outcome = 'Success (ground pad)';
```

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

min(DATE)
01/08/2018

Successful Drone Ship Landing with Payload between 4000 and 6000

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

%sql SELECT BOOSTER_VERSION from SPACEXTBL WHERE LANDING_OUTCOME = 'Success (drone ship)' and PAYLOAD_MASS__KG_ >4000 and PAYLOAD_MASS__KG_ <6000;

```
%sql SELECT BOOSTER_VERSION from SPACEXTBL WHERE LANDING_OUTCOME = 'Success (drone ship)' and  PAYLOAD_MASS__KG_  >4000
```

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

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

Total Number of Successful and Failure Mission Outcomes

We used COUNT function to find the total number of successful and failure mission outcomes.

%sql select count(MISSION_OUTCOME) from SPACEXTBL where MISSION_OUTCOME = 'Success' or MISSION_OUTCOME = 'Failure (in flight)'

```
%sql select count(MISSION_OUTCOME) from SPACEXTBL where MISSION_OUTCOME = 'Success' or MISSION_OUTCOME = 'Failure (in f
```

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

count(MISSION_OUTCOME)
99

Boosters Carried Maximum Payload

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

```
%sql SELECT BOOSTER_VERSION FROM SPACEXTBL WHERE PAYLOAD_MASS__KG_ = (SELECT max(PAYLOAD_MASS__KG_) FROM SPACEXTBL);
```

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

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 WHERE, LIKE, AND and BETWEEN conditions to filter for failed landing outcomes in drone ship, their booster versions and their launch site names for the year 2015.

%sql SELECT BOOSTER_VERSION,LAUNCH_SITE,LANDING_OUTCOME FROM SPACEXTBL WHERE LANDING_OUTCOME = 'Failure (drone ship)' and DATE('2015')

```
%sql SELECT BOOSTER_VERSION,LAUNCH_SITE,LANDING_OUTCOME FROM SPACEXTBL WHERE LANDING_OUTCOME = 'Failure (drone ship)' a
```

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

Booster_Version	Launch_Site	Landing_Outcome
F9 v1.1 B1012	CCAFS LC-40	Failure (drone ship)
F9 v1.1 B1015	CCAFS LC-40	Failure (drone ship)
F9 v1.1 B1017	VAFB SLC-4E	Failure (drone ship)
F9 FT B1020	CCAFS LC-40	Failure (drone ship)
F9 FT B1024	CCAFS LC-40	Failure (drone ship)

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.

```
%sql select * from SPACEXTBL where Landing_Outcome = 'Success (ground pad)' or (DATE between '2010-06-04' and '2017-03-
```

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

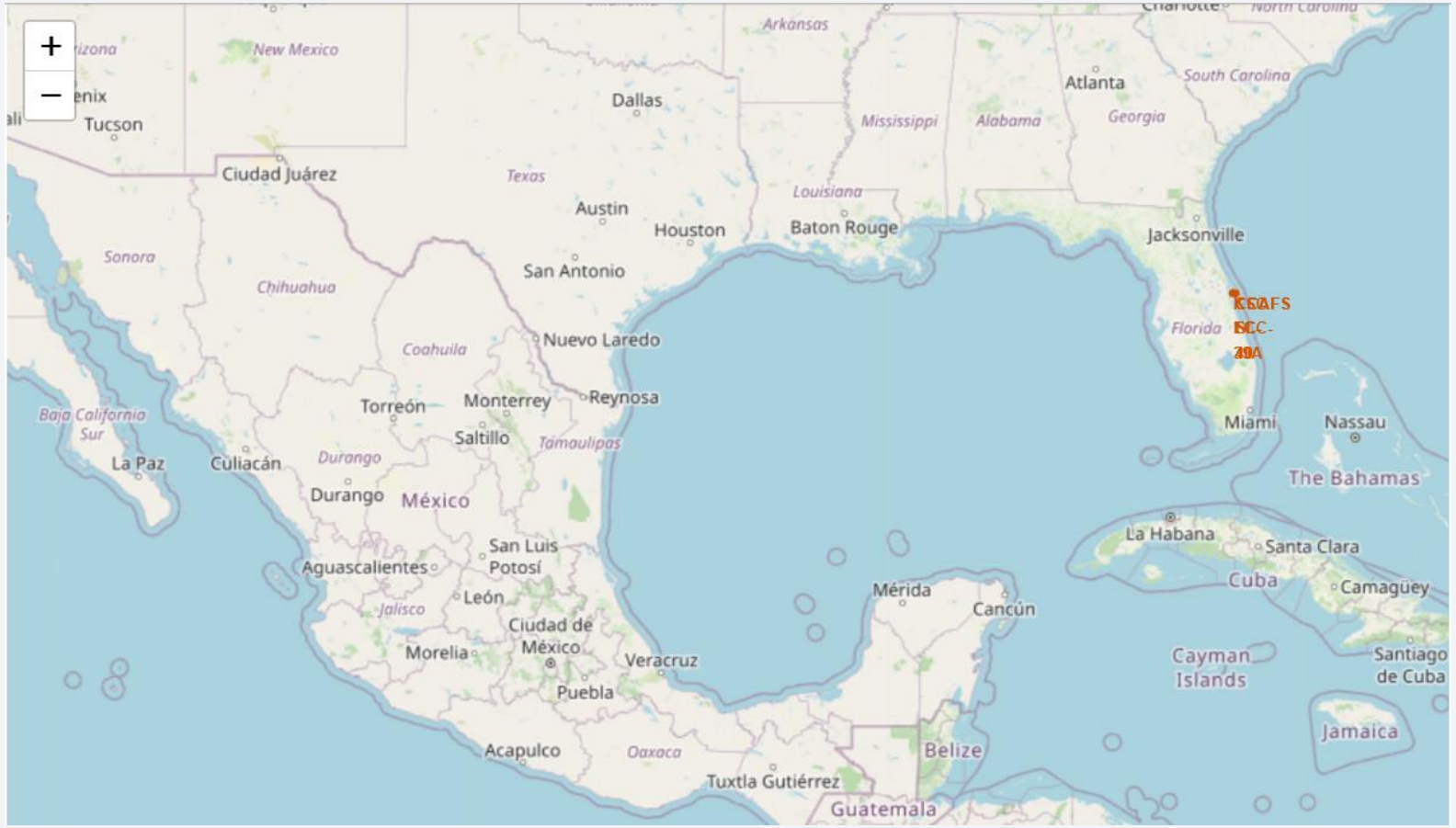
Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_
22/12/2015	1:29:00	F9 FT B1019	CCAFS LC-40	OG2 Mission 2 11 Orbcomm-OG2 satellites	2034.0	LEO	Orbcomm	Success	Succes
19/02/2017	14:39:00	F9 FT B1031.1	KSC LC-39A	SpaceX CRS-10	2490.0	LEO (ISS)	NASA (CRS)	Success	Succes
18/07/2016	4:45:00	F9 FT B1025.1	CCAFS LC-40	SpaceX CRS-9	2257.0	LEO (ISS)	NASA (CRS)	Success	Succes
15/12/2017	15:36:00	F9 FT B1035.2	CCAFS SLC-40	SpaceX CRS-13	2205.0	LEO (ISS)	NASA (CRS)	Success	Succes
14/08/2017	16:31:00	F9 B4 B1039.1	KSC LC-39A	SpaceX CRS-12	3310.0	LEO (ISS)	NASA (CRS)	Success	Succes
09/07/2017	14:00:00	F9 B4 B1040.1	KSC LC-39A	Boeing X-37B OTV-5	4990.0	LEO	U.S. Air Force	Success	Succes
06/03/2017	21:07:00	F9 FT B1035.1	KSC LC-39A	SpaceX CRS-11	2708.0	LEO (ISS)	NASA (CRS)	Success	Succes
05/01/2017	11:15:00	F9 FT B1032.1	KSC LC-39A	NROL-76	5300.0	LEO	NRO	Success	Succes

%sql select * from SPACEXTBL where Landing_Outcome = 'Success (ground pad)' or (DATE between '2010-06-04' and '2017-03-20') order by date desc;

Section 3

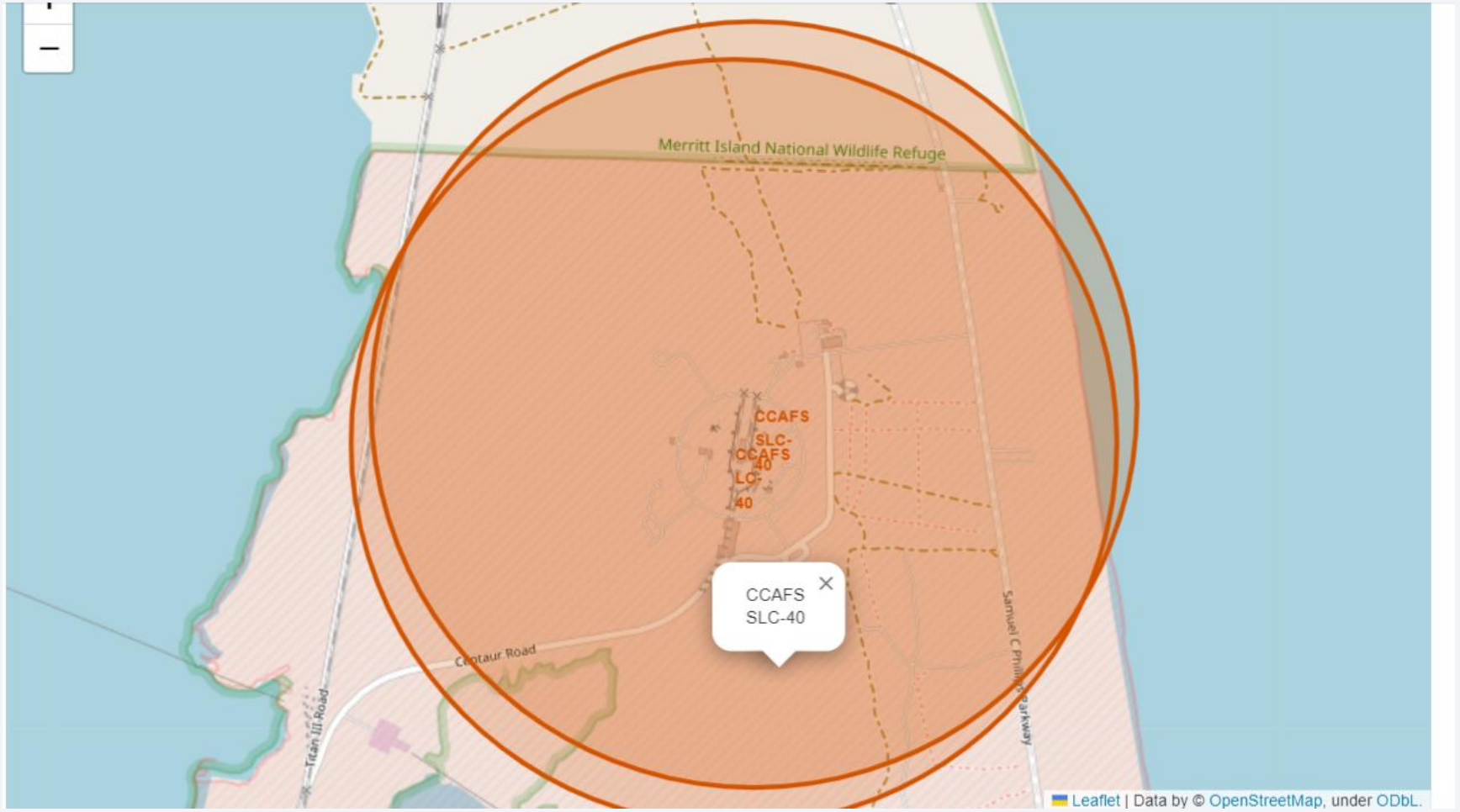
Launch Sites Proximities Analysis

Location of all the Launch Sites

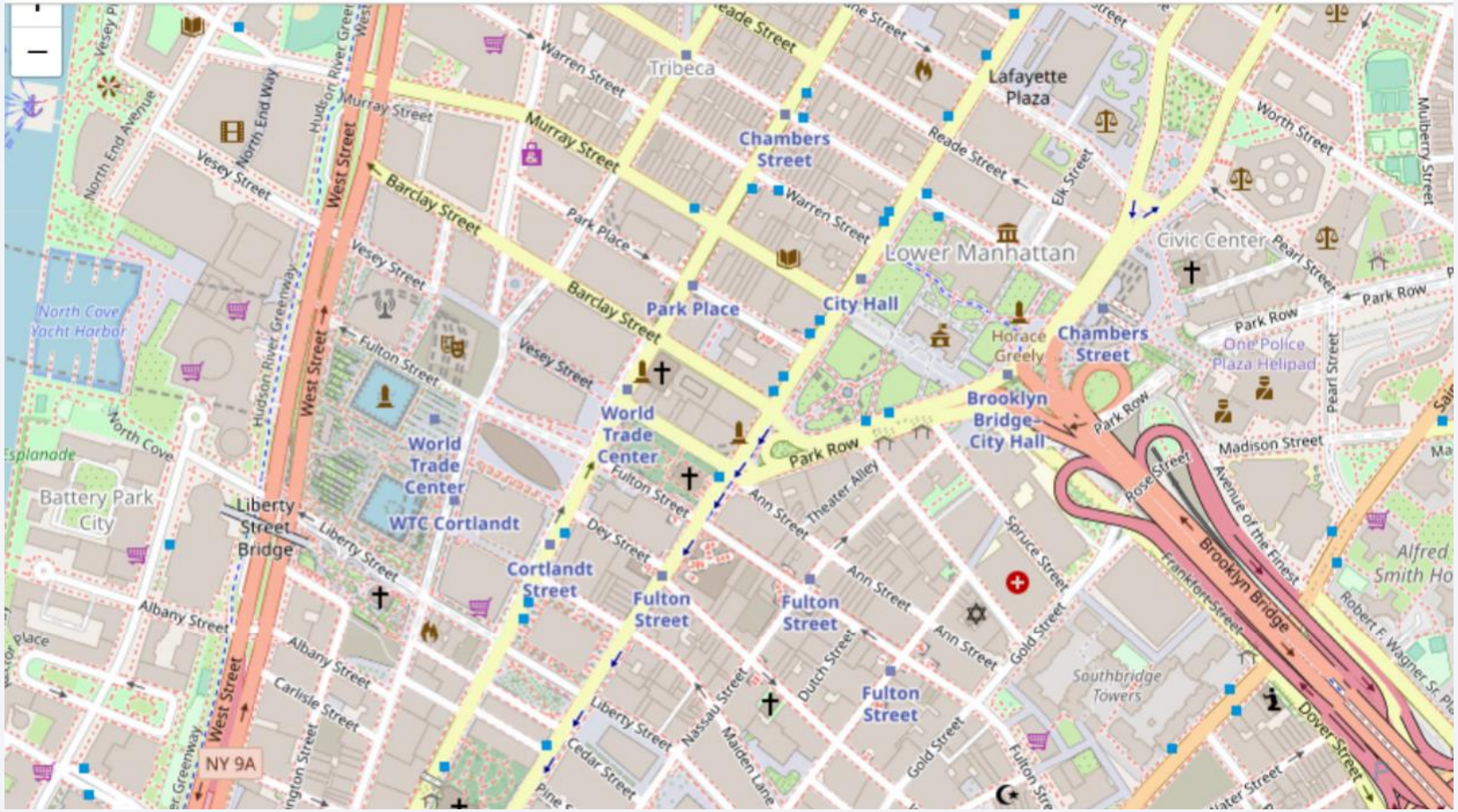


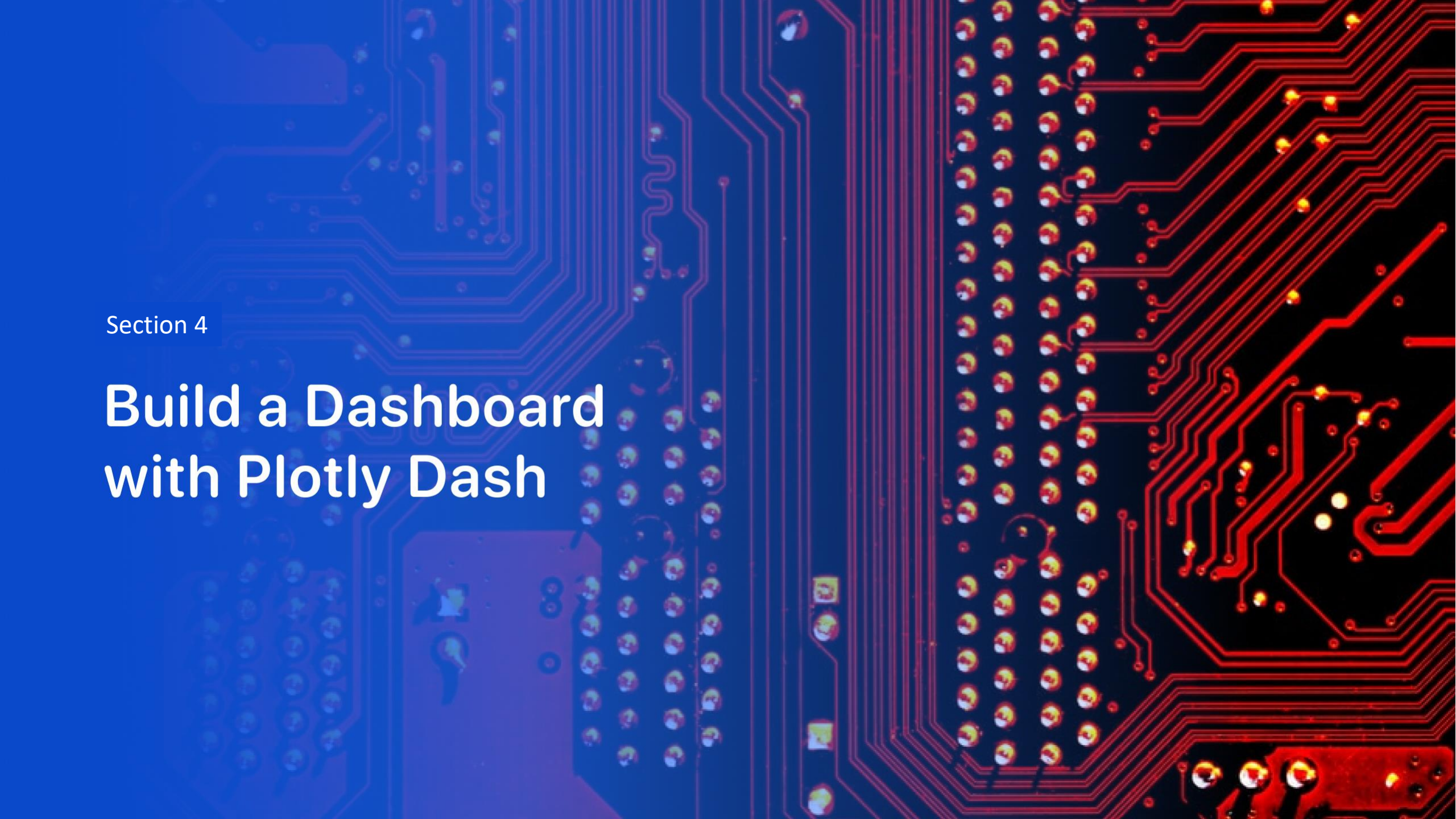
We can see that all the SpaceX launch sites are located inside the United States

Markers showing launch sites with color labels

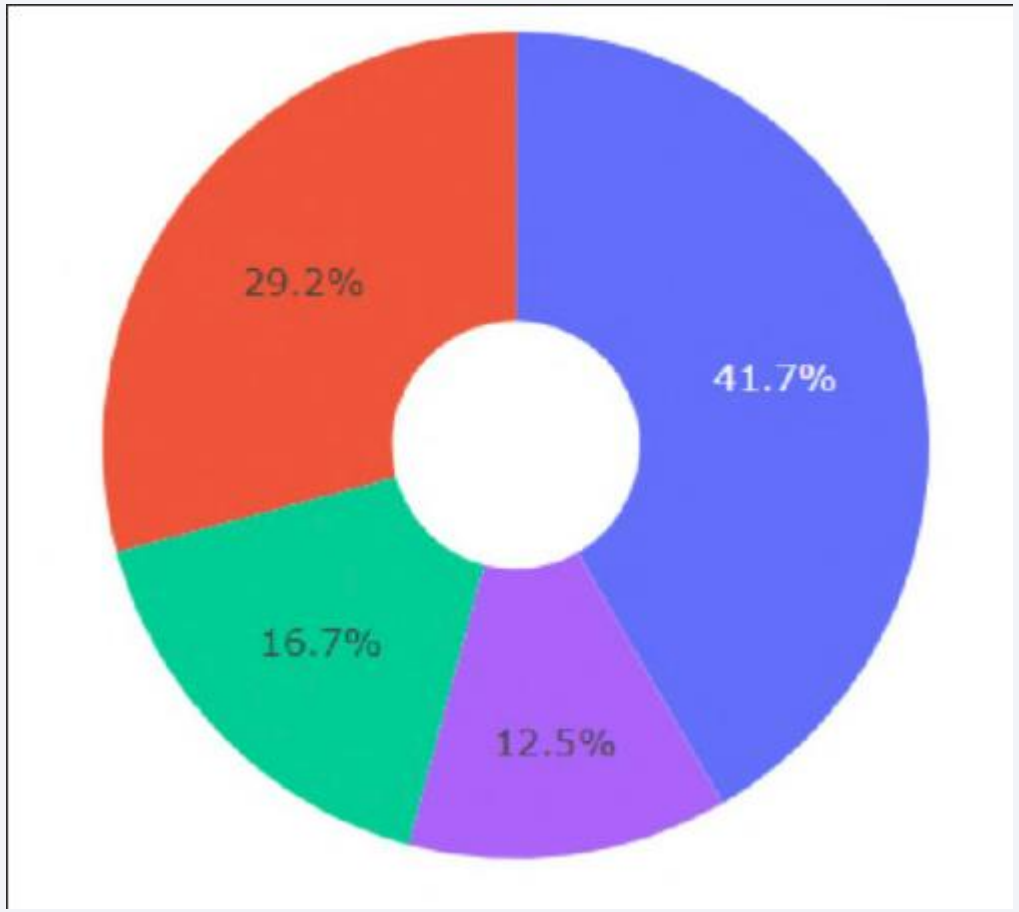


Launch Sites Distance to Landmarks



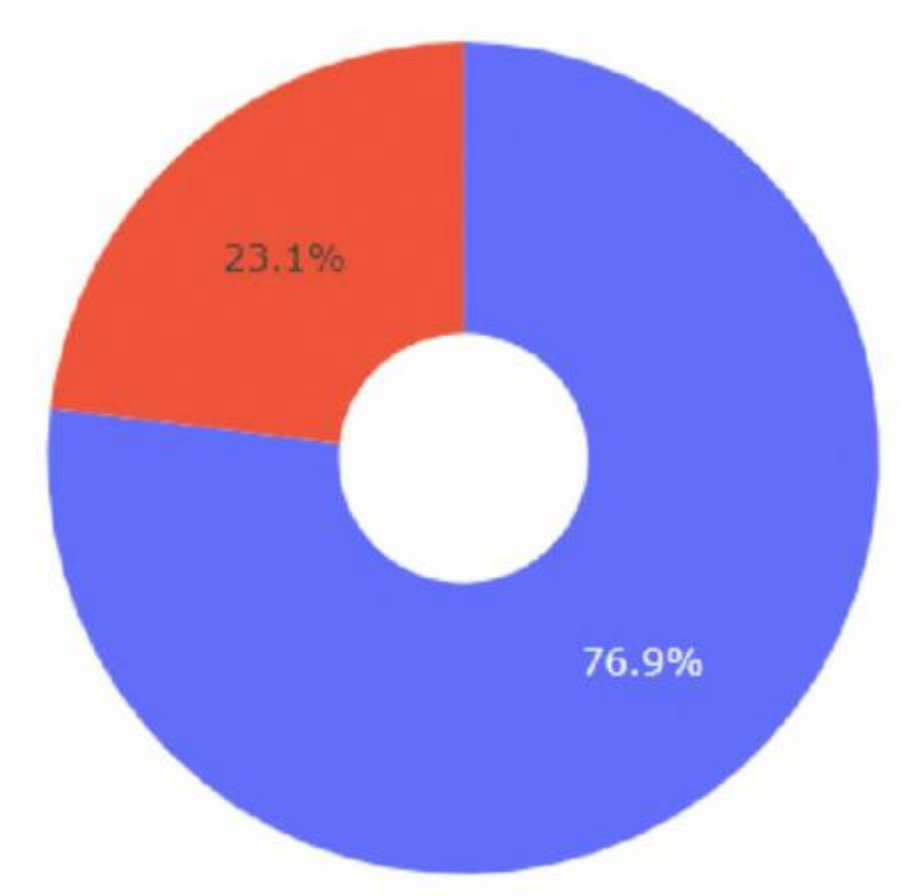

Section 4
Build a Dashboard with Plotly Dash

The success percentage by each site



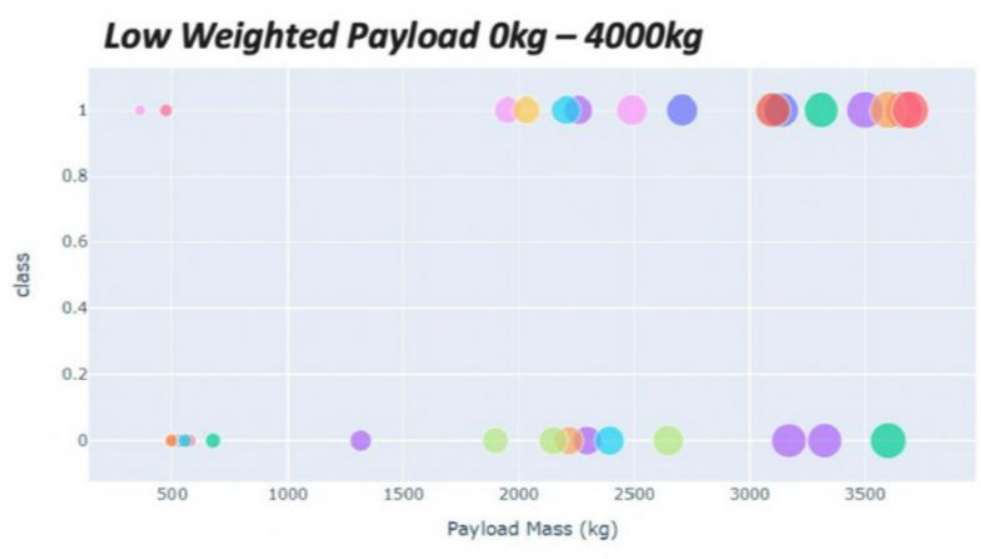
We can see that KSC LC-39A had the most successful launches from all the sites.

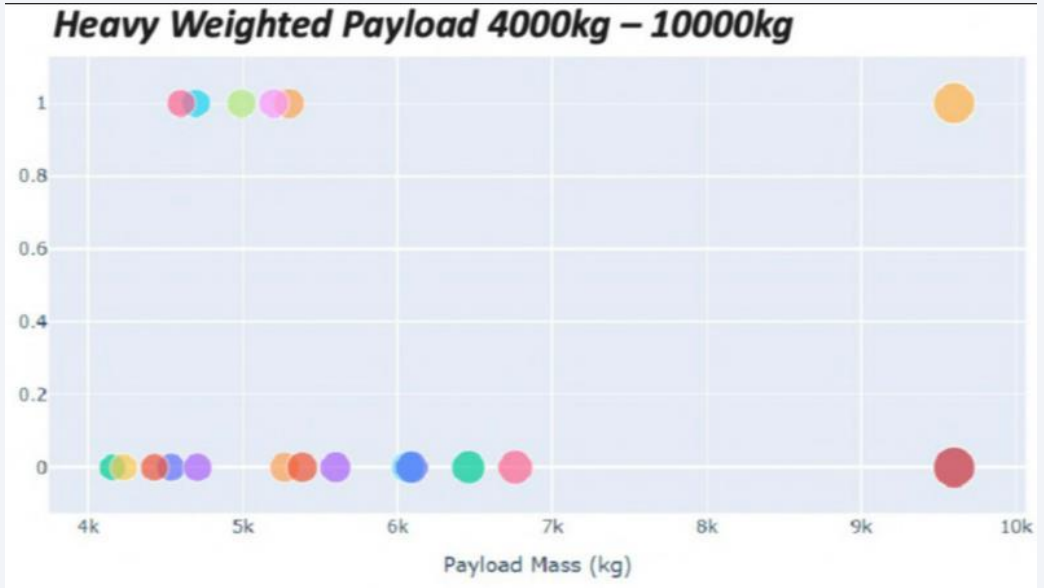
The highest launch-success ratio: KSC LC-39A



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

Payload vs Launch Outcome Scatter Plot





We can see that all the success rate for low weighted payload is higher than heavy weighted payload

Section 5

Predictive Analysis (Classification)

Classification Accuracy

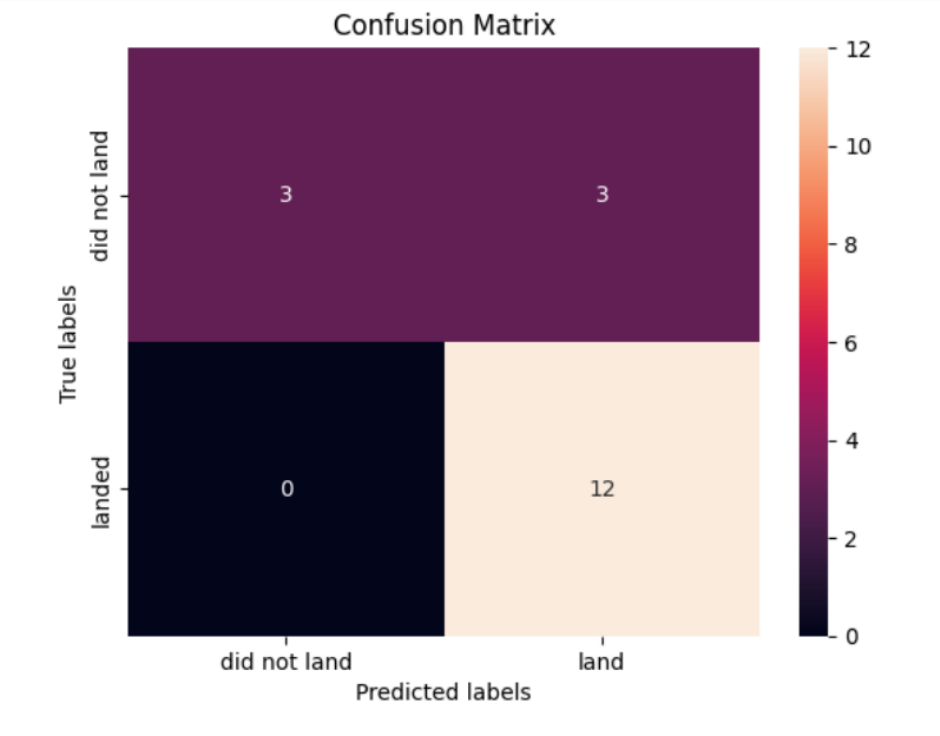
As we can see, by using the code as below: we could identify that the best algorithm to be the Tree Algorithm which have the highest classification accuracy.

```
algorithms={'KNN':knn_cv.best_score_,'Tree':tree_cv.best_score_,'LogisticRegression':logreg_cv.best_score_}
bestalgo=max(algorithms,key=algorithms.get)
print('Best Algorithm is',bestalgo,'with a score of',algorithms[bestalgo])
if bestalgo=='Tree':
    print('Best Params is : ',tree_cv.best_params_)
if bestalgo=='KNN':
    print('Best Params is : ',knn_cv.best_params_)
if bestalgo=='LogisticRegression':
    print('Best Params is : ',logreg_cv.best_params_)
```

```
Best Algorithm is Tree with a score of 0.9
Best Params is :  {'criterion': 'gini', 'max_depth': 2, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 1
0, 'splitter': 'best'}
```

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.



Conclusions

- The Tree Classifier Algorithm is the best Machine Learning approach for this dataset.
- The low weighted payloads (which define as 4000kg and below) performed better than the heavy weighted payloads.
- Starting from the year 2013, the success rate for SpaceX launches is increased, directly proportional time in years to 2020, which it will eventually perfect the launches in the future.
- KSC LC-39A have the most successful launches of any sites; 76.9%
- SSO orbit have the most success rate; 100% and more than 1 occurrence.

Appendix

TASK 1

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

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

TASK 2

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

```
0]: # students get this
transform = preprocessing.StandardScaler()
x = transform.fit(X).transform(X)
```

We split the data into training and testing data using the function `train_test_split`. The training data is divided into validation data, a second set used for training data; then the models are trained and hyperparameters are selected using the function `GridSearchCV`.

TASK 3

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

`X_train, X_test, Y_train, Y_test`

```
1]: X_train, X_test, Y_train, Y_test = train_test_split( x, y, test_size=0.2, random_state=2)
print ('Train set:', X_train.shape,  Y_train.shape)
print ('Test set:', X_test.shape,  Y_test.shape)
```

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

we can see we only have 18 test samples.

```
2]: Y_test.shape
```

```
2]: (18,)
```

TASK 4

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

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

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

```
5]: 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

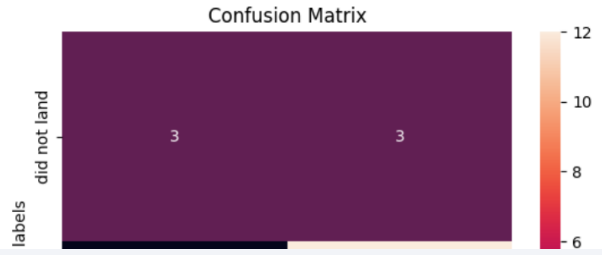
Calculate the accuracy on the test data using the method `score`:

```
6]: logreg_score = logreg_cv.score(X_test, Y_test)
print("score :",logreg_score)
```

```
score : 0.8333333333333334
```

Lets look at the confusion matrix:

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



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`.

```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
svm_cv = GridSearchCV(estimator=svm, cv=10, param_grid=parameters).fit(X_train, Y_train)
```

```
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`:

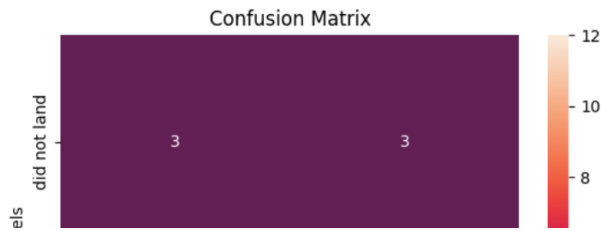
```
svm_cv_score = svm_cv.score(X_test, Y_test)

print("score :",svm_cv_score)
```

```
score : 0.8333333333333334
```

We can plot the confusion matrix

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



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`.

```
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()
```

```
#tree_cv = GridSearchCV(estimator=tree, cv=10, param_grid=parameters).fit(X_train, Y_train)
```

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

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 2, 'max_features': 'sqrt', 'min_samples_leaf': 2,
'min_samples_split': 10, 'splitter': 'best'}
accuracy : 0.9
```

TASK 9

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

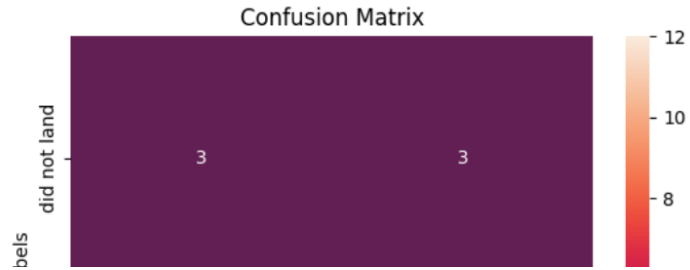
```
tree_cv_score = tree_cv.score(X_test, Y_test)

print("score :",tree_cv_score)
```

```
score : 0.8333333333333334
```

We can plot the confusion matrix

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



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`.

[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()
knn_cv = GridSearchCV(estimator=KNN, cv=10, param_grid=parameters).fit(X_train, Y_train)
```

[31]:
```
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
```

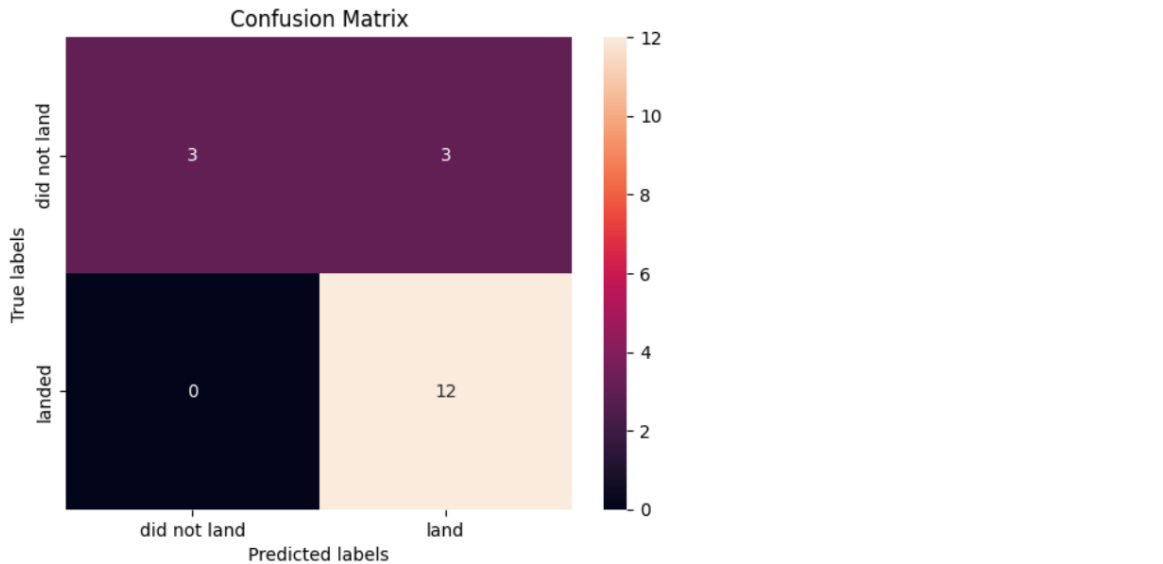
TASK 11

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

[32]:
```
knn_cv_score = knn_cv.score(X_test, Y_test)
print("score :",knn_cv_score)
```

```
score : 0.8333333333333334
```

33]:
```
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



TASK 12

Find the method performs best:

[37]:
```
accuracy = [svm_cv_score, logreg_score, knn_cv_score, tree_cv_score]
accuracy = [i * 100 for i in accuracy]

method = ['Support Vector Machine', 'Logistic Regression', 'K Nearest Neighbour', 'Decision Tree']
models = {'ML Method':method, 'Accuracy Score (%)':accuracy}

ML_df = pd.DataFrame(models)
ML_df
```

[37]:

	ML Method	Accuracy Score (%)
0	Support Vector Machine	83.333333
1	Logistic Regression	83.333333
2	K Nearest Neighbour	83.333333
3	Decision Tree	83.333333

Thank you!