IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Nilay Shah
25/10/2021

Outline

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

Executive Summary

Summary of methodologies:

In this project, Spacex Flacon 9 data was collected using the SpaceX Rest API and wrangled to calculate the number and occurrence of mission outcome per orbit type. From there, exploratory analysis using SQL, Pandas and MatplotLib was carried out to identify which attributes determine if the first stage can be reused. Finally, a machine learning model was built using these features to automatically predict if the first stage can land successfully.

Summary of all results:

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

Introduction

The commercial space age is here, companies are making space travel affordable for everyone. Perhaps the most successful is SpaceX. One reason SpaceX can do this is the rocket launches are relatively inexpensive. SpaceX advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upwards of 165 million dollars each, much of the savings is because SpaceX can reuse the first stage.

Therefore, if we can determine if the first stage will land successfully, we can determine the cost of a launch.

<u>Common problems that needed solving.</u>

- What influences if the rocket will land successfully?

- The effect each relationship with certain rocket variables will impact in determining the success rate of a successful landing.

- What conditions does SpaceX have to achieve to get the bestresults and ensure the best rocket success landing rate

First Stage vs Second Stage

The payload is enclosed in the fairings.

Stage two bring the payload to orbit, but most of the work is done by the first stage.

This first stage is much larger than the second stage and provides the initial thrust to send the rocket skywards.

The first stage is shown here.

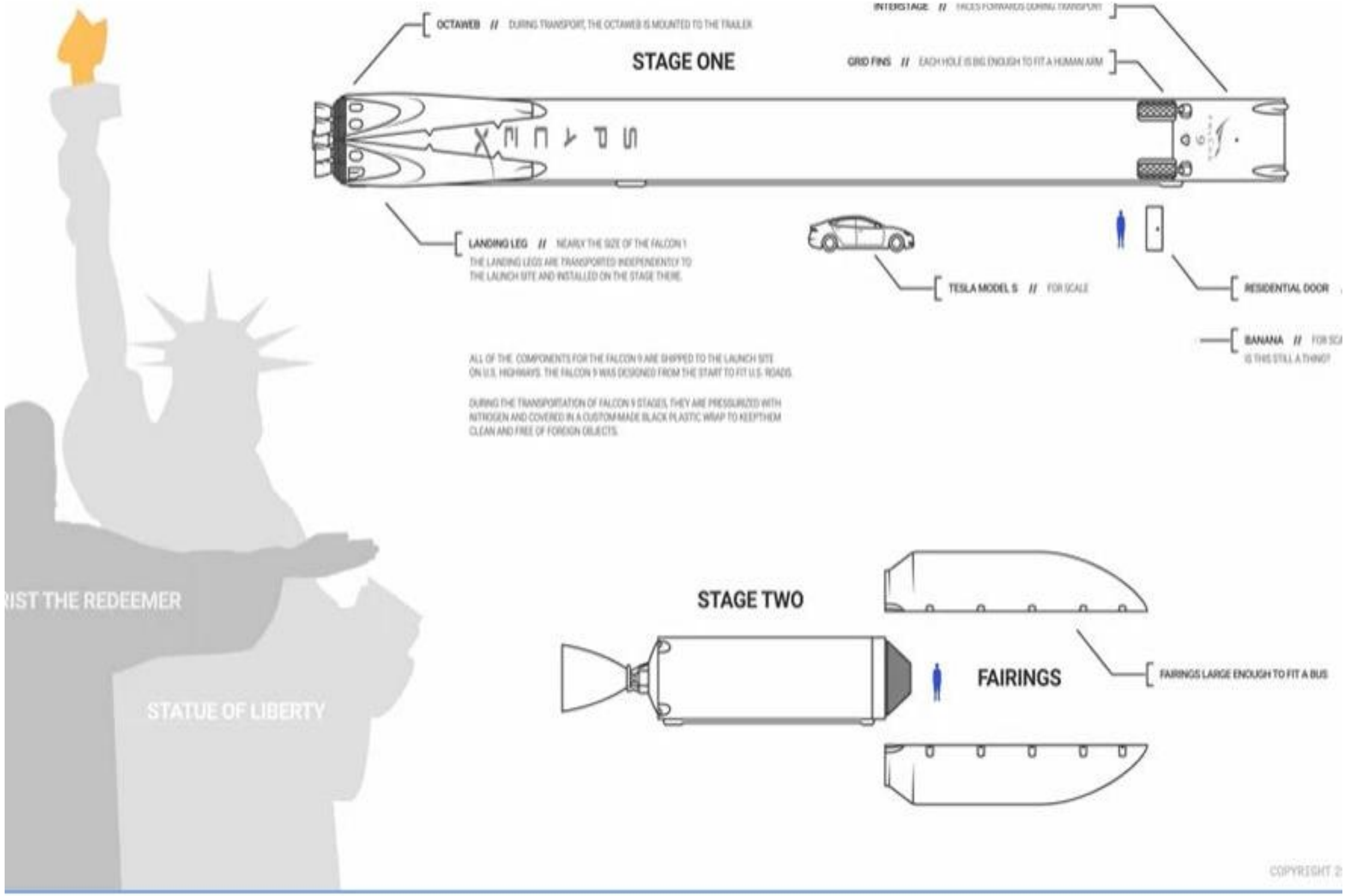
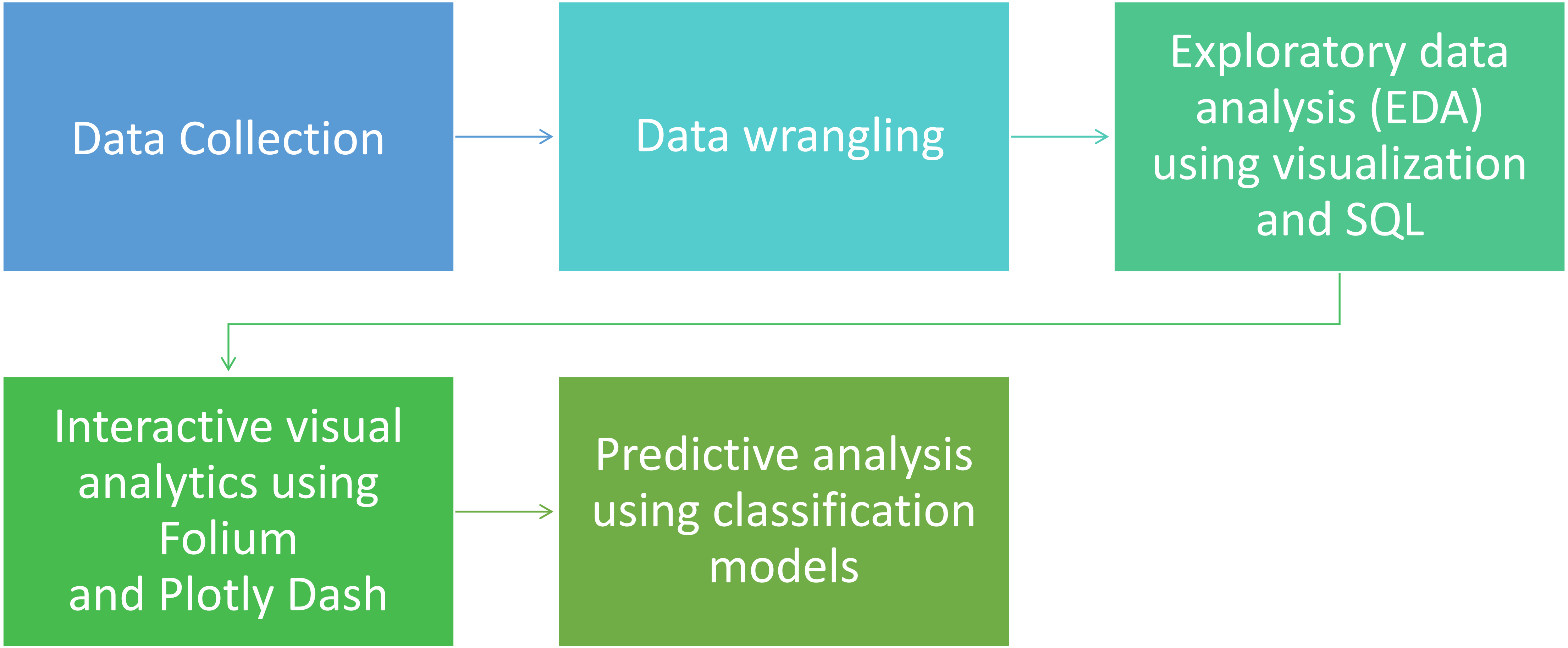


Figure 1: SpaceX First and Second Stage

Section 1

Methodology

Method


Data Collection
Data wrangling
Exploratory data analysis (EDA) using visualization and SQL
Interactive visual analytics using Folium and Plotly Dash
Predictive analysis using classification models

Data Collection

The SpaceX data was collected in two different ways, which are both outlined below.

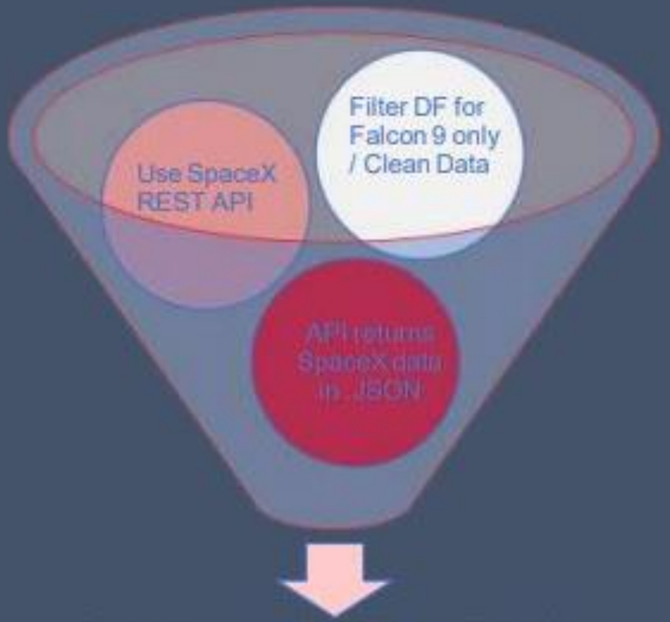
Method 1: API

1. Gather SpaceX launch data from the SpaceX REST API using the endpoint api.spacexdata.com/v4/launches/past.
2. Perform a get request using the requests library to obtain the launch data, which we will use to get the data from the API.
3. Our response will be in the form of a JSON, specifically a list of JSON objects. This can be viewed by calling the .json() method.
4. Finally, convert this JSON to a dataframe using the json_normalize function.

Method 2: Web Scraping

1. Request the HTML page from the URL using urllib.request.
2. Create a BeautifulSoup object from the HTML.
3. Extract all column/variable names from the HTML table header by using the .find_all() method in BeautifulSoup.
4. Create a data frame by parsing the launch HTML tables and using the .DataFrame() method.

Data collection – SpaceX API



Normalize data into flat data file such as .csv

https://github.com/Nilay1997/Project/blob/master/API.ipynb

simplified flow chart

1 .Getting Response from API

```
spacex_url="https://api.spacexdata.com/v4/launches/past"
response = requests.get(spacex_url).json()
```

2. Converting Response to a .json file

```
response = requests.get(static_json_url).json()
data = pd.json_normalize(response)
```

3. Apply custom functions to clean data

```
getLaunchSite(data)
getPayloadData(data)
getCoreData(data)
```

```
getBoosterVersion(data)
```

4. Assign list to dictionary then dataframe

```
launch_dict = {'FlightNumber': list(data['flight_number']),
'Date': list(data['date']),
'BoosterVersion':BoosterVersion,
'PayloadMass':PayloadMass,
'Orbit':Orbit,
'LaunchSite':LaunchSite,
'Outcome':Outcome,
'Flights':Flights,
'GridFins':GridFins,
'Reused':Reused,
'Legs':Legs,
'LandingPad':LandingPad,
'Block':Block,
'ReusedCount':ReusedCount,
'Serial':Serial,
'Longitude': Longitude,
'Latitude': Latitude}
```

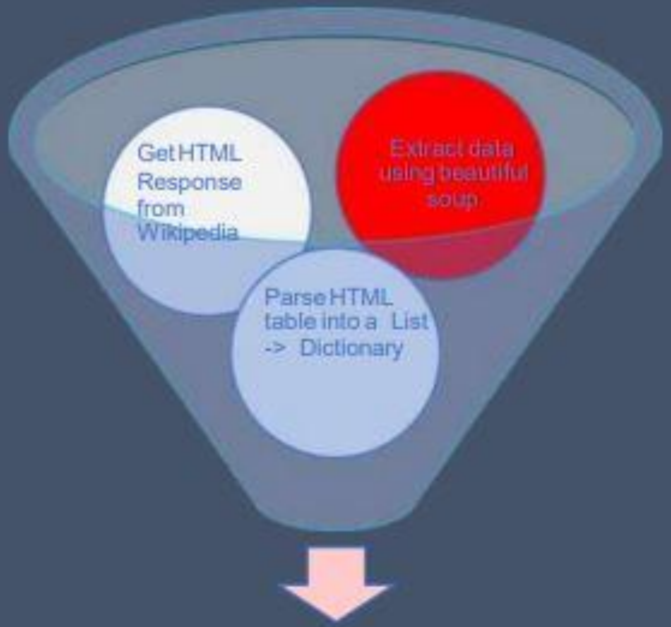
```
df = pd.DataFrame.from_dict(launch_dict)
```

5. Filter dataframe and export to flat file (.csv)

```
data_falcon9 = df.loc[df['BoosterVersion']!="Falcon 1"]
```

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

Data collection – Web Scrapping



Normalize data into flat data file such as .csv

https://github.com/Nilay1997/Project/blob/master/Web%20Scraping.ipynb

simplified flow chart

1 .Getting Response from HTML

```
page = requests.get(static_url)
```

2. Creating BeautifulSoup Object

```
soup = BeautifulSoup(page.text, 'html.parser')
```

3. Finding tables

```
html_tables = soup.find_all('table')
```

4. Getting 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
```

5. Creation of dictionary

```
launch_dict= dict.fromkeys(column_names)

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


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'] = []
launch_dict['Version Booster']=[]
launch_dict['Booster landing']=[]
launch_dict['Date']=[]
launch_dict['Time']=[]
```

6. Appending data to keys (refer) to notebook block 12

```
In [12]: extracted_row = 0
         #Extract each table
         for table_number,table in enumerate(
           # get table row
            for rows in table.find_all("tr")
              #check to see if first table
```

7. Converting dictionary to dataframe

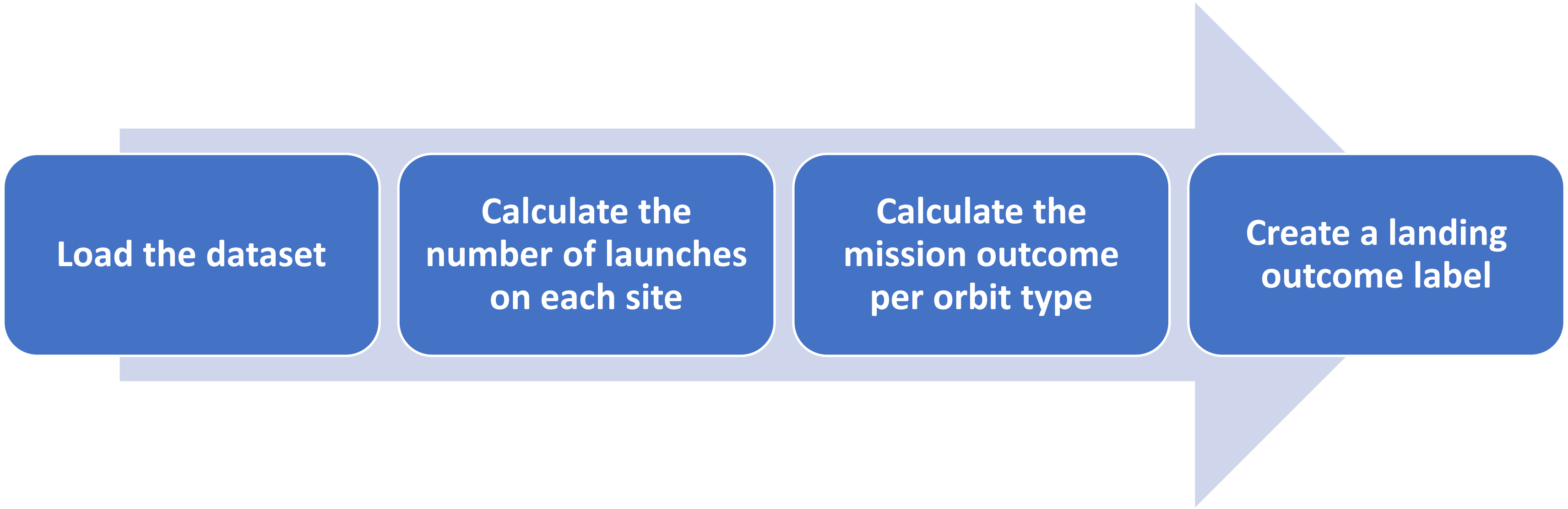
```
df = pd.DataFrame.from_dict(launch_dict)
```

8. Dataframe to .CSV

```
df.to_csv('spacex_web_scraped.csv', index=False)
```

Data Wrangling

In the data set, there are several different cases where the booster did not land successfully. The number and quantity of the mission outcome per orbit type was calculated using the function .value_counts(). Finally, these outcomes were converted into Training Labels by creating a list where the element was zero if the landing was unsuccessful and 1 if the landing was a success and assigning it a variable ‘Landing Class’ using .append().



GitHub link: https://github.com/Nilay1997/Project/blob/master/EDA.ipynb

EDA with Data Visualization

- **Flight Number vs. Payload Mass** using a catplot - Firstly, we wanted see how the Flight Number (indicating the continuous launch attempts.) and Payload variables would affect the launch outcome and using a catplot would allow us to overlay the outcome of the launch.
- **Payload Vs Launch Site** using a scatter chart – We wanted to observe if there were any relationship between launch sites and their payload mass
- **Success rate of each Orbit Type** using a bar chat **-** we wanted to visually check if there were any differences between the success rate for each orbit type
- **Flight Number Vs Orbit Type** using a scatter point chart to reveal the relationship between the flight and orbit type.
- **Payload vs. Orbit Type** using a scatter point chart to reveal the relationship between the payload and orbit type.
- **Year vs Average Success Rate** using a line chart to to get the average launch success trend.

GitHub link: https://github.com/Nilay1997/Project/blob/master/EDA%20with%20Visualisation.ipynb

EDA with SQL

To answer questions about our data, SQL queries were run on the dataset.

Firstly, we loaded the SQL extension and established a connection with the database using %load_ext sql. Several queries were then run to explore the data.

Some of the questions answered can be seen below:

- To display the names of the unique launch sites in the space mission, the query '%sql SELECT UNIQUE(LAUNCH_SITE) FROM SPACEXTBL' was used.
- To select the total and average Payload carried by boosters, the queries 'SELECT SUM(PAYLOAD_MASS_KG)' and 'SELECT AVG(PAYLOAD_MASS_KG)' were used.
- To list the date when the first successful landing outcome in ground pad was achieved, the '%sql SELECT MIN(DATE) FROM SPACEXTBL' was used.
- To list the total number of successful and failure mission outcomes the query '%sql SELECT COUNT(MISSION_OUTCOME) FROM SPACEXTBL GROUP BY MISSION_OUTCOME' was used.

GitHub link: https://github.com/Nilay1997/Project/blob/master/EDA%20with%20SQL.ipynb

Build an Interactive Map with Folium

The dataset contains coordinates These are just plain numbers that can not give any intuitive insights about where the location of the launch sites. By pinning these on a map we can easily visualize their locations. A Folium Map was thus created.

- folium.Circle was added to create a highlighted circle area and text label for each launch coordinate so that they can be identified and each one clearly distinguished.
- folium.Marker was used to distinguish between successful and failed launches. If a launch was successful (class=1), then we use a green marker and if a launch was failed, we use a red marker (class=0).
- A MarkerCluster() object was created to simplify the map since it contained many markers having the same coordinate.
- A MousePosition was added on the map to get coordinates for a mouse over a point on the map. As such, while you are exploring the map, you can easily find the coordinates of any points of interests,
- folium.PolyLine was used to easily visualize the distance between two points on the map based on their Lat and Long values.

https://github.com/Nilay1997/Project/blob/master/Interactive%20Visual%20Analytics%20with%20Folium%20lab.ipynb

Build a Dashboard with Plotly Dash

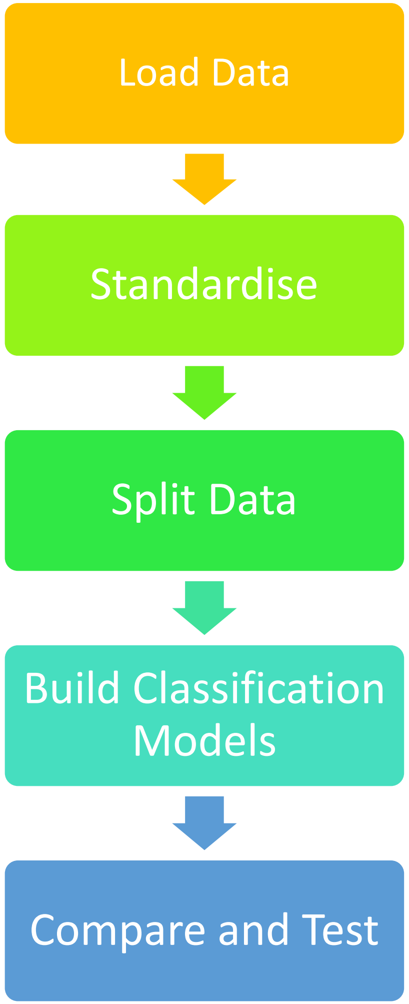
A dashboard was built with Flask and Dash web framework.

- A drop-down menu was added as there are four different launch sites and this will allow the user to select one specific site and check its detailed success rate ((class=0 vs. class=1).
- A Range Slider was added to select Payload. This will allow the user to easily select different payload ranges and see if we can identify some visual patterns, and thus see if variable payload is correlated to mission outcome.
- A Pie chart was added to easily visualize launch success counts and compare the four different launch sites.
- A Scatter plot showing the launch outcome for each payload mass, was also added to visually observe how payload may be correlated with mission outcomes for selected site.

GitHub link: https://nilay0097-8050.theiadocker-5-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/

Predictive Analysis (Classification)

GitHub link:
https://github.com/Nilay1997/Project/blob/master/Machine%20Learning.ipynb



<u>1. Load the data and create a NumPy Array</u>

The data is loaded using the .read_csv command, the 'Class' column is converted to a NumPy array by applying the method to_numpy() and assigned to the variable Y

<u>2. Standardize the data in X</u>

Since the data will have varying scales, it needs to be standardised so that the resultant distribution has a unit standard deviation. This will be done using the technique StandardScalar

<u>3. Split the data into training and test data</u>

Using the function train_test_split the data X and Y can be split into training and test data. The parameter test_size will be set to 0.2 and random_state to 2

<u>4. Build Classification models</u>

Different models should be implemented, including Logistic Regression, Support Vector Machines, Decision Trees and K Nearest Neighbour and the data fit to the models using .fit()

<u>5. Find the best model</u>

The accuracy of the test data can be found using the method score, and a confusion matrix can be plotted in each case to see if the model can distinguish between the different classes.

Results

EXPLORATORY DATA ANALYSIS RESULTS

INTERACTIVE ANALYTICS DEMO IN SCREENSHOTS

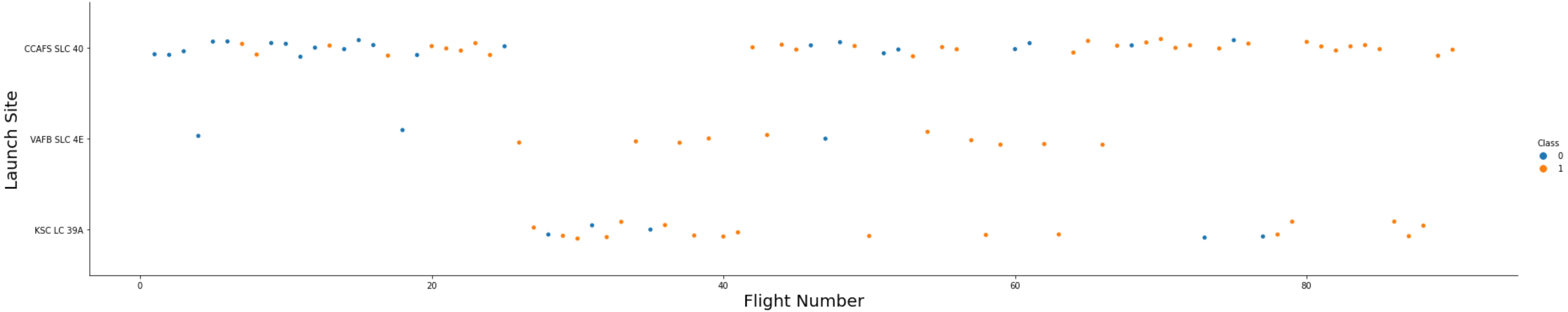
PREDICTIVE ANALYSIS RESULTS

Section 2

Insights drawn from EDA

Flight Number vs. Launch Site

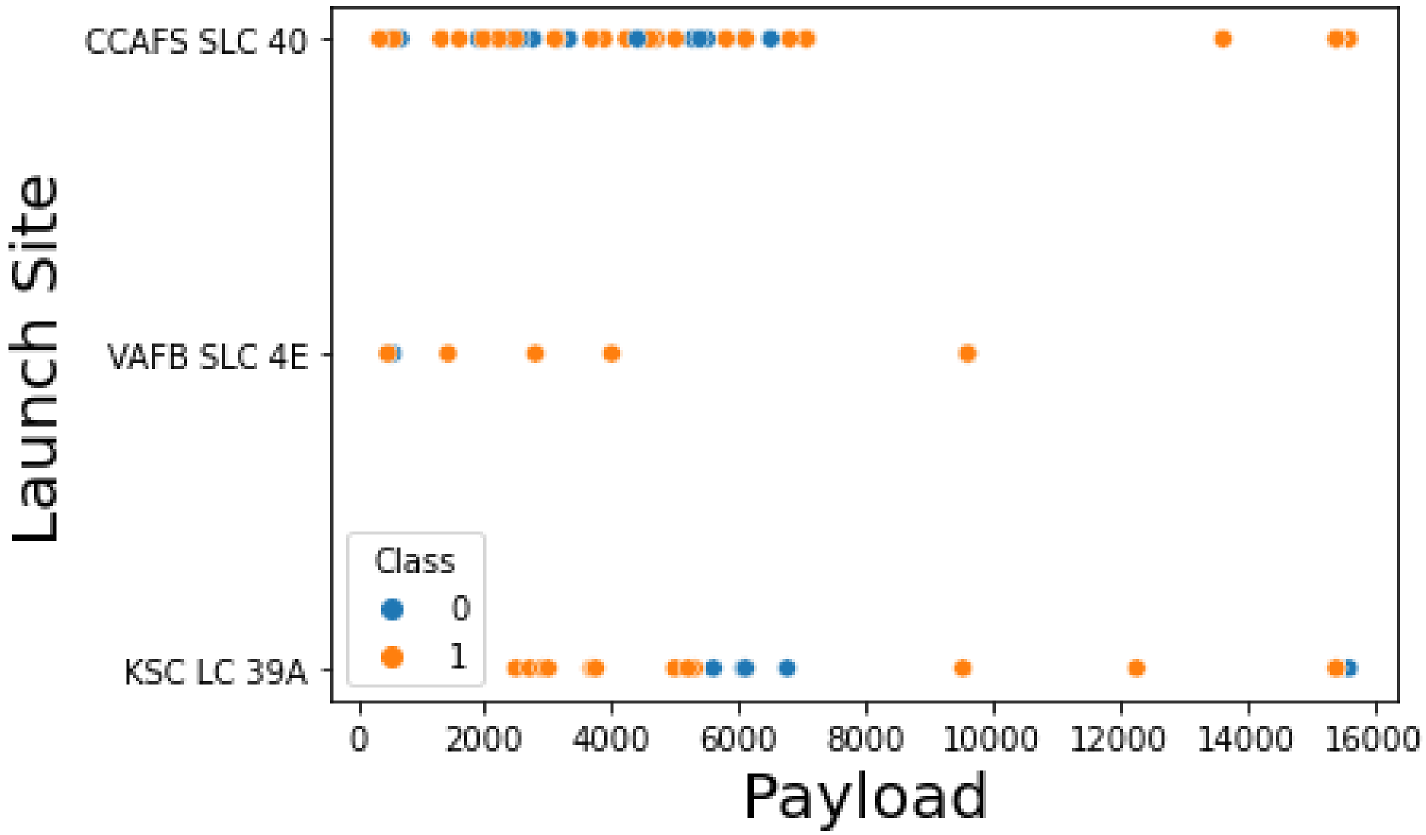
From the graph we can see that as the flight number increases, the greater the number of successful flights (orange marker) at that launch site.



Payload vs. Launch Site

The greater the payload mass for launchsite CCAFS SLC 40 the higher the success rate for the Rocket.

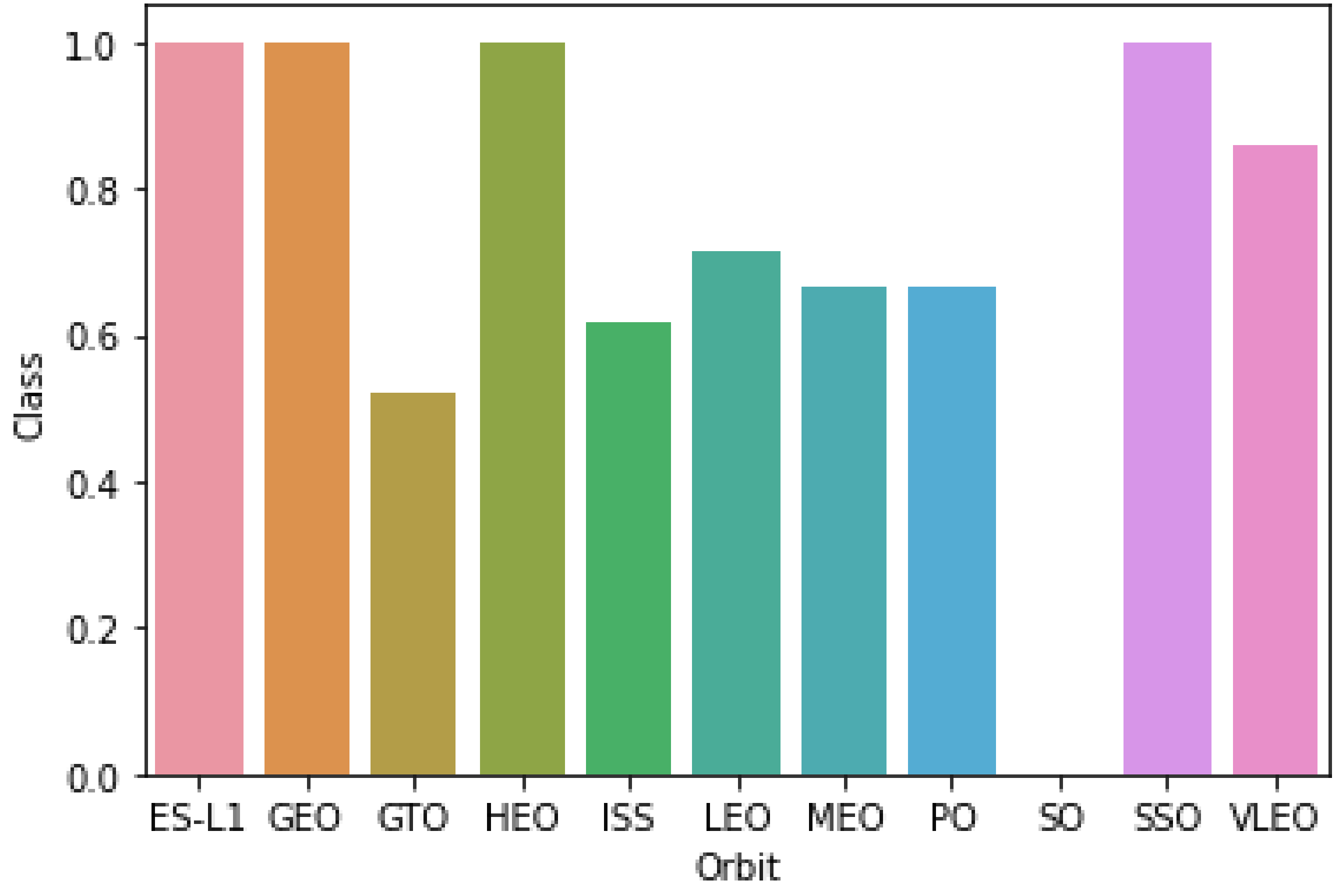
For the VAFB-SLC launchsite there are no rockets launched for heavy payload mass (greater than 10000).



Success Rate vs. Orbit Type

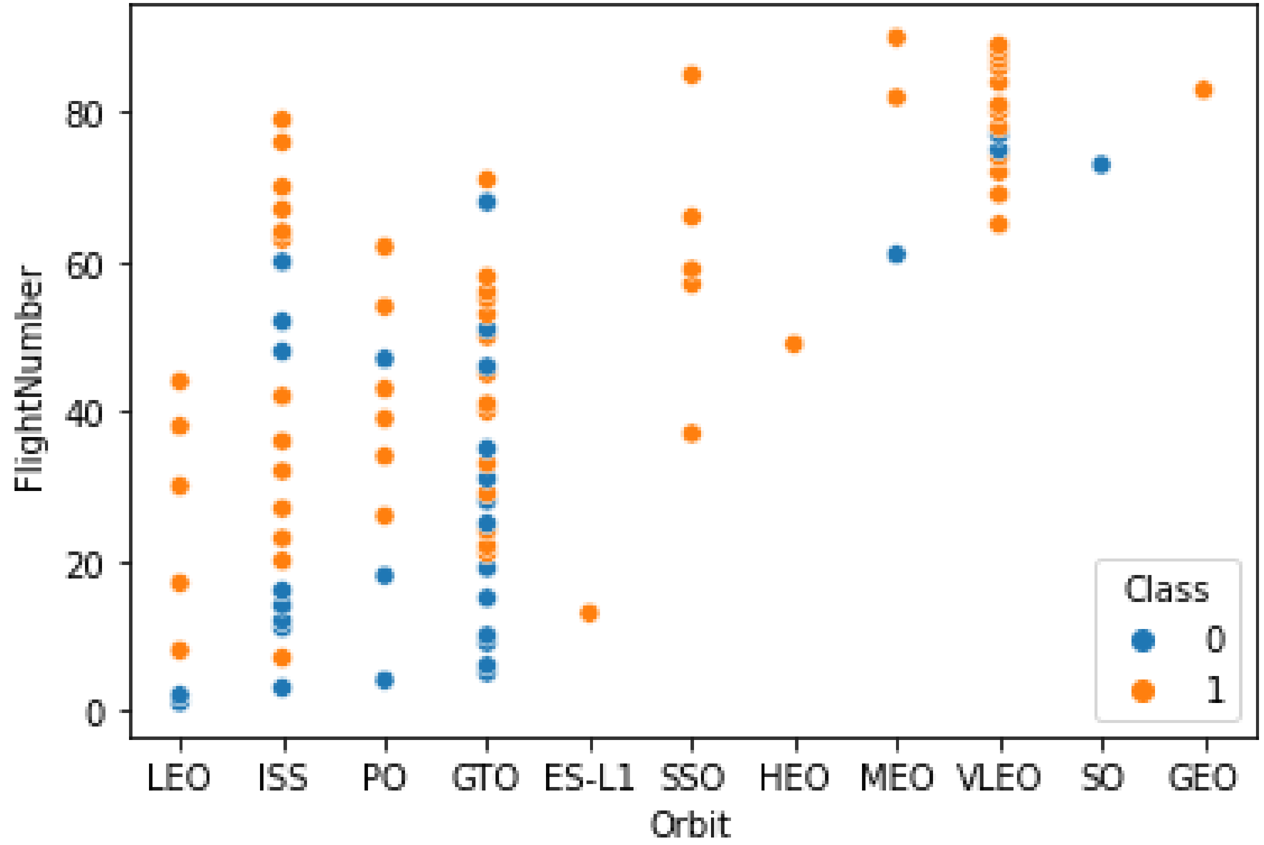
From the graph we can see that:

- ➢Orbit GEO,HEO,SSO,ES-L1 have the highest Success Rate.
- ➢Orbit GTO has the lowest success rate.
- ➢The success rate for the Orbit SO is null.



Flight Number vs. Orbit Type

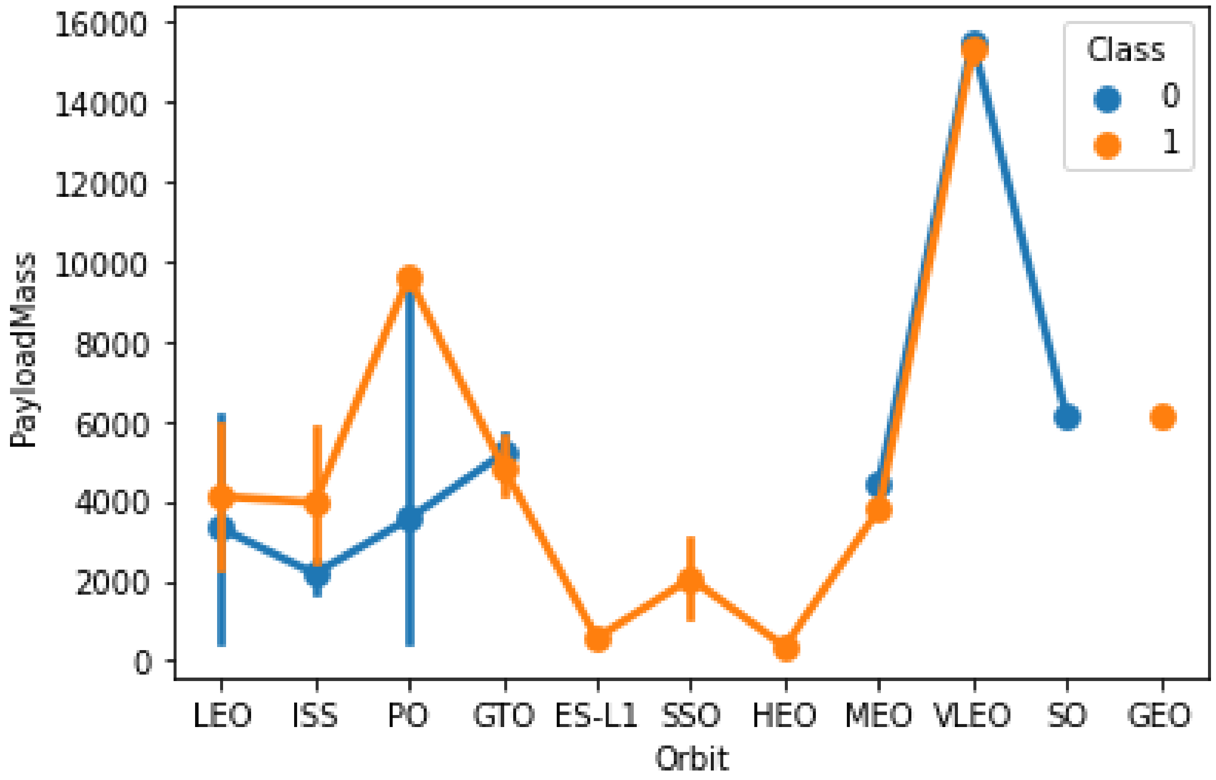
From the graph we can see that in the LEO orbit the Success appears related to the number of flights; on the other hand, there seems to be no relationship between flight number when in GTO orbit. The SO orbit also has no recorded successful flights.



Payload vs. Orbit Type

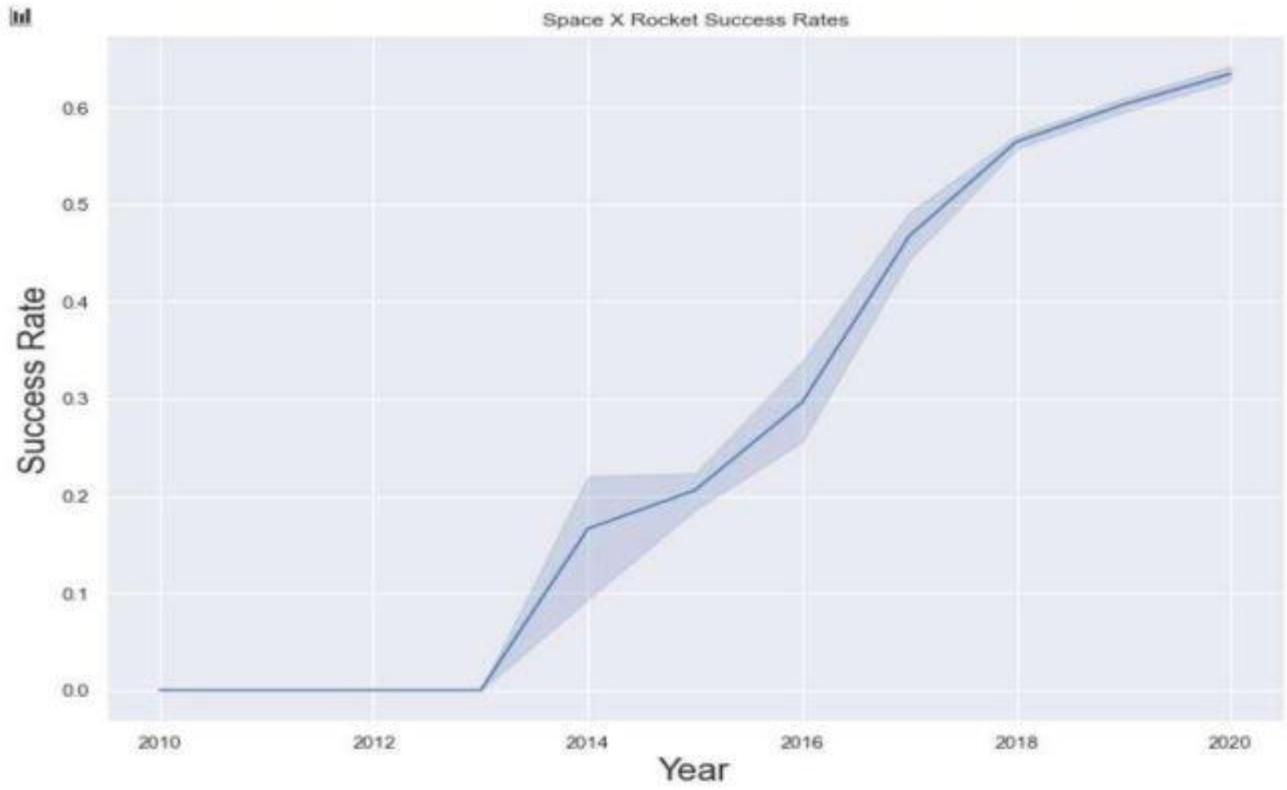
From the graph we can see that:

- With heavy payloads the successful landing or positive landing rate are more for Polar,LEO and ISS.
- For GTO we cannot distinguish this well as both positive landing rate and negative landing(unsuccessful mission) are both there here



Launch Success Yearly Trend

From the graph we can observe that between 2010 and 2013 there were no successful launches. However, from 2013, the success rate kept increasing till 2020.



Section 3

EDA with SQL

All Launch Site Names

SQL QUERY

```
select DISTINCT Launch_Site
from tblSpaceX
```

Unique Launch Sites
CCAFS LC - 40
CCAFS SLC - 40
CCAFS SLC - 40
KSC LC – 39A
VAFB SLC -4E

QUERY EXPLAINATION

Using the word ***DISTINCT*** in the query means that it will only show Unique values in the ***Launch_Site*** column from ***tblSpaceX***

Launch Site Names Begin with 'CCA'

SQL QUERY

```
select TOP 5 * from tblSpaceX
WHERE Launch_Site LIKE 'KSC%'
```

QUERY EXPLAINATION

Using the word ***TOP 5*** in the query means that it will only show 5 records from ***tblSpaceX*** and ***LIKE*** keyword has a wild card with the words ***'KSC%'*** the percentage in the end suggests that the Launch_Site name must start with KSC.

	Date	Time_UTC	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS_KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
0	19-02-2017	2021-07-02 14:39:00.0000000	F9 FT B1031.1	KSC LC-39A	SpaceX CRS-10	2490	LEO (ISS)	NASA (CRS)	Success	Success (ground pad)
1	16-03-2017	2021-07-02 06:00:00.0000000	F9 FT B1030	KSC LC-39A	EchoStar 23	5600	GTO	EchoStar	Success	No attempt
2	30-03-2017	2021-07-02 22:27:00.0000000	F9 FT B1021.2	KSC LC-39A	SES-10	5300	GTO	SES	Success	Success (drone ship)
3	01-05-2017	2021-07-02 11:15:00.0000000	F9 FT B1032.1	KSC LC-39A	NROL-76	5300	LEO	NRO	Success	Success (ground pad)
4	15-05-2017	2021-07-02 23:21:00.0000000	F9 FT B1034	KSC LC-39A	Inmarsat-5 F4	6070	GTO	Inmarsat	Success	No attempt

Total Payload Mass

SQL QUERY

```
select SUM(PAYLOAD_MASS_KG_) TotalPayloadMass
from tblSpaceX
where Customer = 'NASA (CRS)'",'TotalPayloadMass
```

	Total Payload Mass
0	45596

QUERY EXPLAINATION

Using the function ***SUM*** summates the total in the column ***PAYLOAD_MASS_KG_***

The ***WHERE*** clause filters the dataset to only perform calculations on ***Customer NASA (CRS)***

Average Payload Mass by F9 v1.1

SQL QUERY

```
select AVG(PAYLOAD_MASS_KG_) AveragePayloadMass

from tblSpaceX

where Booster_Version = 'F9 v1.1'
```

	Average Payload Mass
0	2928

QUERY EXPLAINATION

Using the function ***AVG*** works out the average in the column ***PAYLOAD_MASS_KG_***

The ***WHERE*** clause filters the dataset to only perform calculations on ***Booster_version F9 v1.1***

First Successful Ground Landing Date

SQL QUERY

select MIN(Date) SLO from tblSpaceX where Landing_Outcome = "Success (drone ship)"

	Date which first Successful landing outcome in drone ship was acheived.
0	06-05-2016

QUERY EXPLAINATION

Using the function ***MIN*** works out the minimum date in the column ***Date***

The ***WHERE*** clause filters the dataset to only perform calculations on ***Landing_Outcome Success (drone ship)***

Successful Drone Ship Landing with Payload between 4000 and 6000

SQL QUERY

select Booster_Version

from tblSpaceX

where Landing_Outcome = 'Success (ground pad)'

AND Payload_MASS_KG_ > 4000 AND Payload_MASS_KG_ < 6000

	Date which first Successful landing outcome in drone ship was acheived.
0	F9 FT B1032.1
1	F9 B4 B1040.1
2	F9 B4 B1043.1

QUERY EXPLAINATION

Selecting only ***Booster_Version***

The ***WHERE*** clause filters the dataset to ***Landing_Outcome = Success (drone ship)***

The ***AND*** clause specifies additional filter conditions ***Payload_MASS_KG_*** > 4000 AND ***Payload_MASS_KG_ < 6000***

Total Number of Successful and Failure Mission Outcomes

SQL QUERY

SELECT(SELECT Count(Mission_Outcome) from tblSpaceX where Mission_Outcome

LIKE '%Success%') as Successful_Mission_Outcomes,

(SELECT Count(Mission_Outcome) from tblSpaceX where Mission_Outcome

LIKE '%Failure%') as Failure_Mission_Coutcomes

	Successful_Mission_Outcomes	Failure_Mission_Outcomes
0	100	1

QUERY EXPLAINATION

a much harder query I must say, we used subqueries here to produce the results. The ***LIKE '%foo%'*** wildcard shows that in the record the ***foo*** phrase is in any part of the string in the records for example.

PHRASE "(Drone Ship was a Success)"
LIKE '%Success%'
Word 'Success' is in the phrase the filter will include it in the dataset

Boosters Carried Maximum Payload

SQL QUERY

SELECT DISTINCT Booster_Version, MAX(PAYLOAD_MASS _KG_) AS [Maximum Payload Mass]

FROM tblSpaceX

GROUP BY Booster_Version

ORDER BY [Maximum Payload Mass] DESC

QUERY EXPLAINATION

Using the word ***DISTINCT*** in the query means that it will only show Unique values in the ***Booster_Version*** column from ***tblSpaceX***
GROUP BY puts the list in order set to a certain condition.
DESC means its arranging the dataset into descending order

	Booster_Version	Maximum Payload Mass
0	F9 B5 B1048.4	15600
1	F9 B5 B1048.5	15600
2	F9 B5 B1049.4	15600
3	F9 B5 B1049.5	15600
4	F9 B5 B1049.7	15600
...	...	...
92	F9 v1.1 B1003	500
93	F9 FT B1038.1	475
94	F9 B4 B1045.1	362
95	F9 v1.0 B0003	0
96	F9 v1.0 B0004	0

97 rows × 2 columns

2015 Launch Records

SQL QUERY

```
SELECT DATENAME(month, DATEADD(month,
MONTH(CONVERT(date, Date, 105)), 0) - 1) AS Month,
Booster_Version, Launch_Site, Landing_Outcome
FROM   tblSpaceX
WHERE (Landing_Outcome LIKE N'%Success%') AND
(YEAR(CONVERT(date, Date, 105)) = '2017')
```

QUERY EXPLAINATION

a much more complex query as I had my ***Date*** fields in SQL Server stored as ***NVARCHAR*** the ***MONTH*** function returns name month. The function ***CONVERT*** converts ***NVARCHAR*** to ***Date***.

WHERE clause filters ***Year*** to be 2017

Month	Booster_Version	Launch_Site	Landing_Outcome
January	F9 FT B1029.1	VAFB SLC-4E	Success (drone ship)
February	F9 FT B1031.1	KSC LC-39A	Success (ground pad)
March	F9 FT B1021.2	KSC LC-39A	Success (drone ship)
May	F9 FT B1032.1	KSC LC-39A	Success (ground pad)
June	F9 FT B1035.1	KSC LC-39A	Success (ground pad)
June	F9 FT B1029.2	KSC LC-39A	Success (drone ship)
June	F9 FT B1036.1	VAFB SLC-4E	Success (drone ship)
August	F9 B4 B1039.1	KSC LC-39A	Success (ground pad)
August	F9 FT B1038.1	VAFB SLC-4E	Success (drone ship)
September	F9 B4 B1040.1	KSC LC-39A	Success (ground pad)
October	F9 B4 B1041.1	VAFB SLC-4E	Success (drone ship)
October	F9 FT B1031.2	KSC LC-39A	Success (drone ship)
October	F9 B4 B1042.1	KSC LC-39A	Success (drone ship)
December	F9 FT B1035.2	CCAFS SLC-40	Success (ground pad)

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

SQL QUERY

```
SELECT COUNT(Landing_Outcome)
FROM  tblSpaceX
WHERE (Landing_Outcome LIKE '%Success%’)
AND (Date > '04-06-2010’) AND
(Date < '20-03-2017')
```

QUERY EXPLAINATION

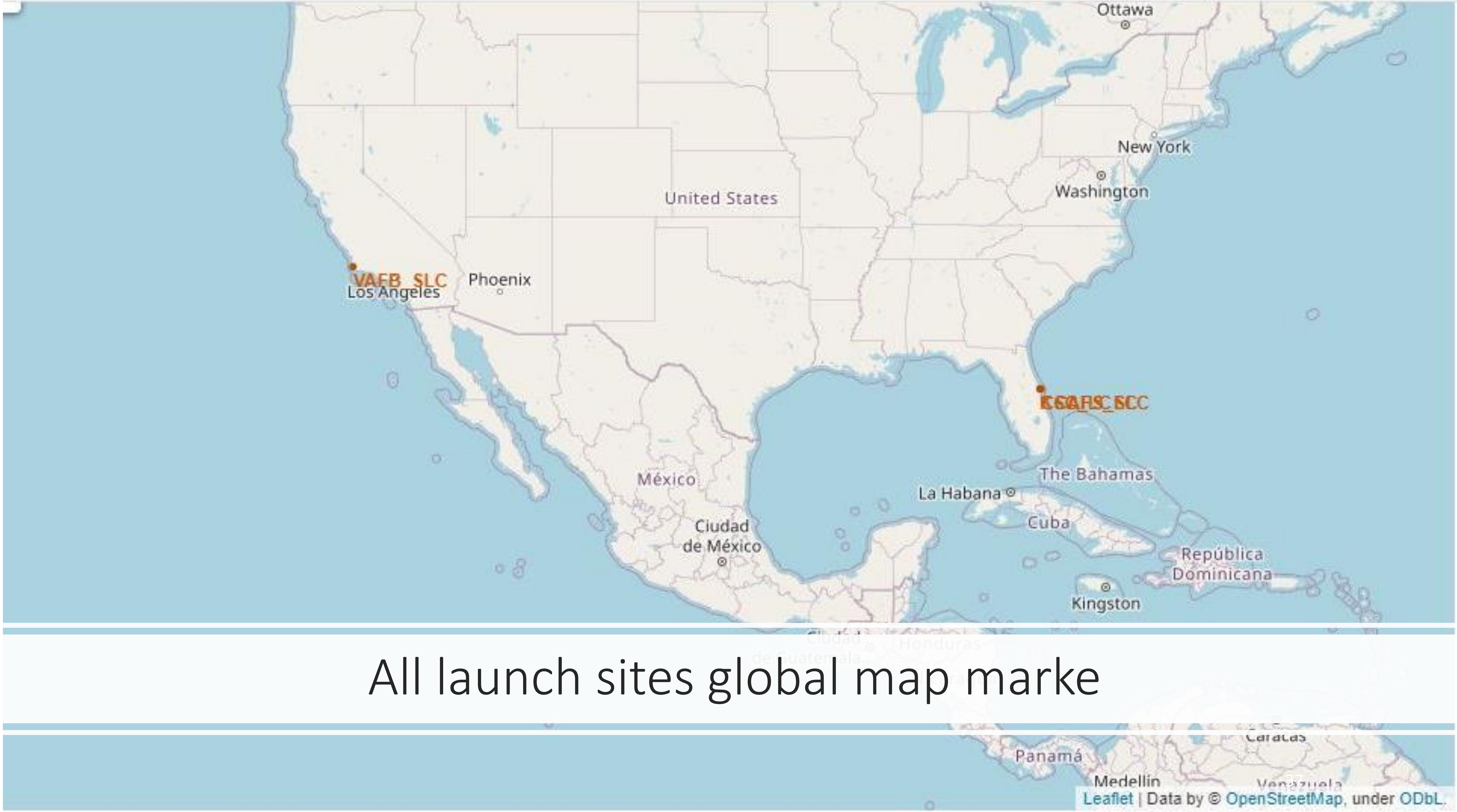
Function ***COUNT*** counts records in column
WHERE filters data

LIKE (wildcard)
AND (conditions)
AND (conditions)

	Successful Landing Outcomes Between 2010-06-04 and 2017-03-20
0	34

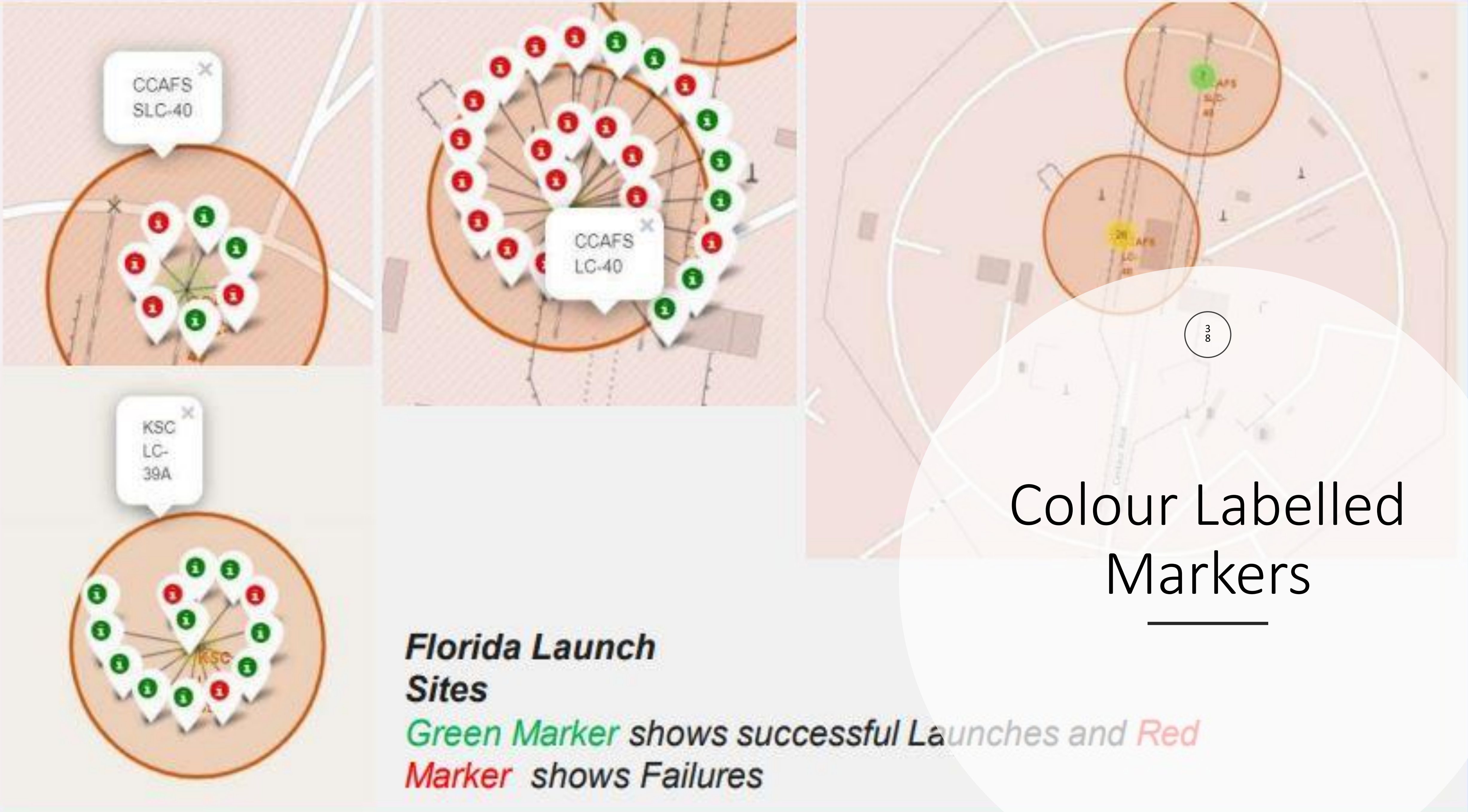
Section 4

Launch Sites Proximities Analysis



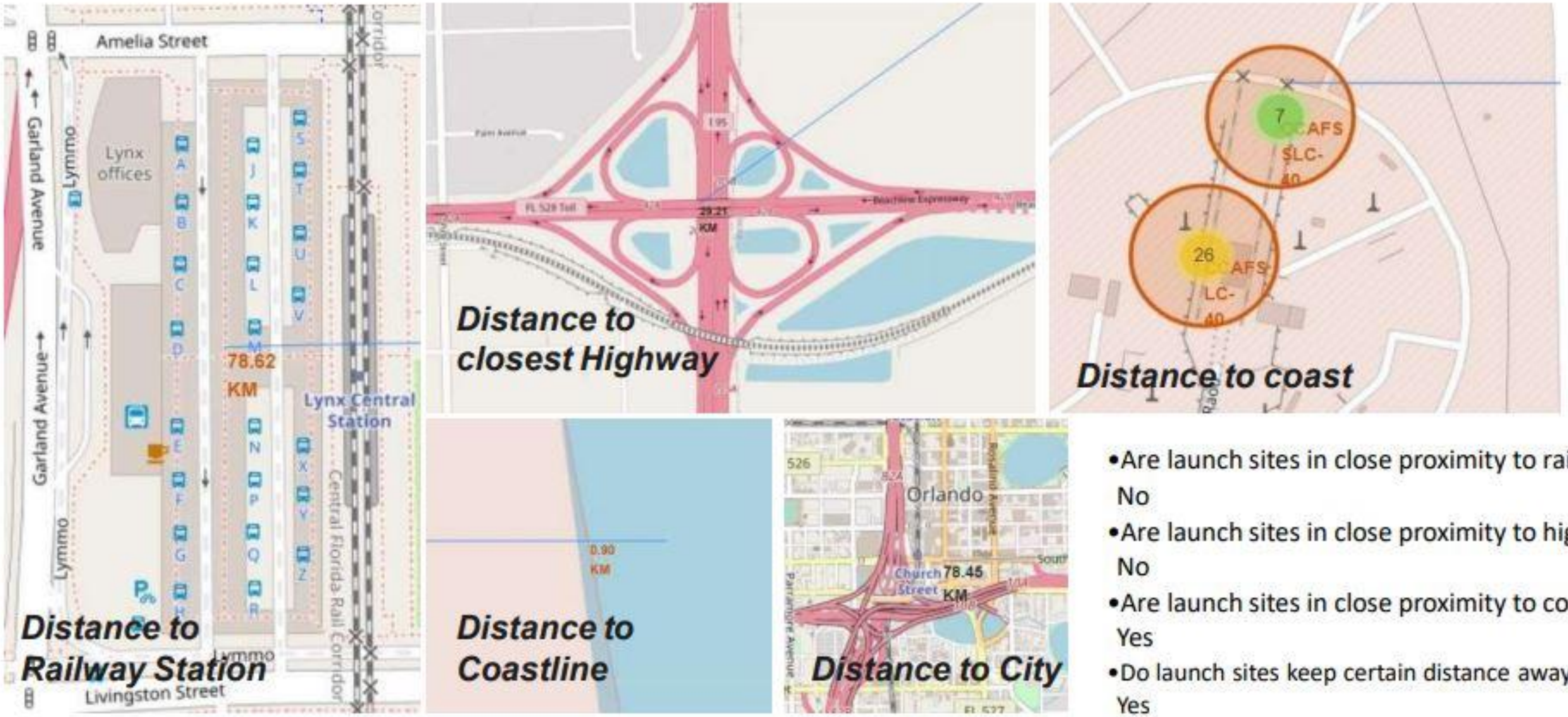
All launch sites global map marke

Colour Labelled Markers



Florida Launch Sites

Green Marker shows successful Launches and Red Marker shows Failures



- 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

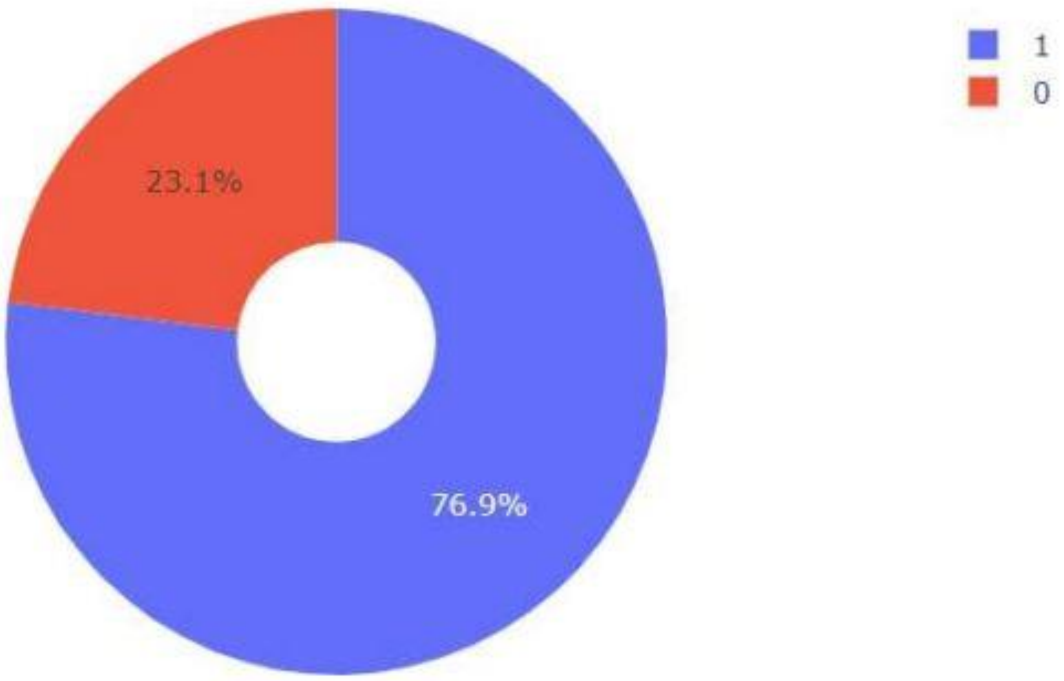
Working out Launch Site CCAFS-SLC-40 proximity

Section 5

Build a Dashboard with Plotly Dash

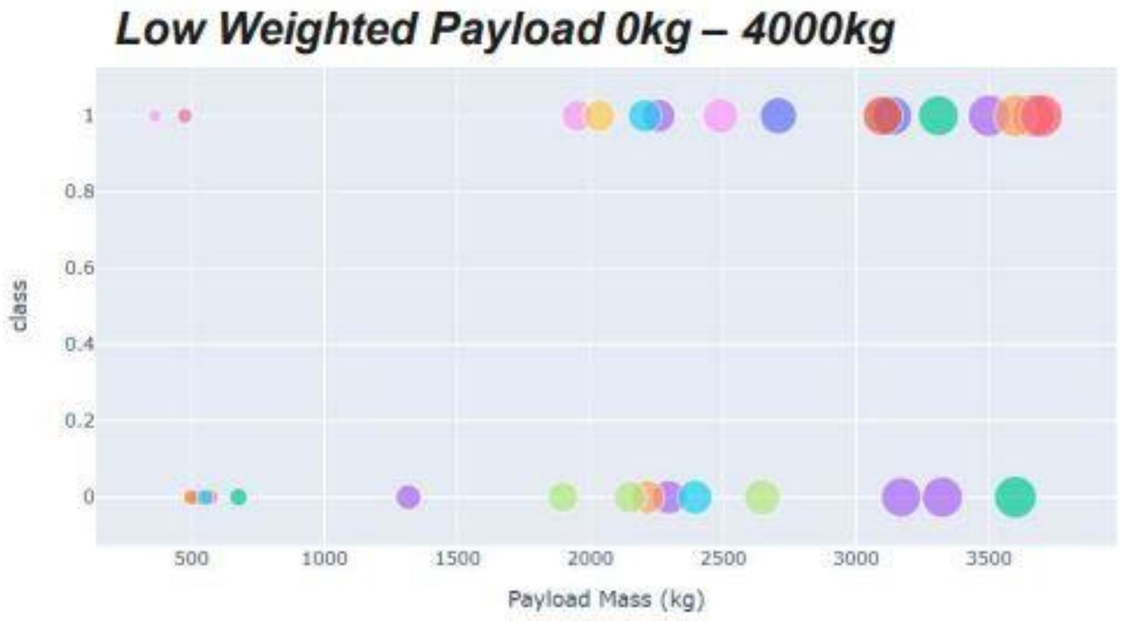
We can see that KSC LC-39A had the most successful launches from all the sites

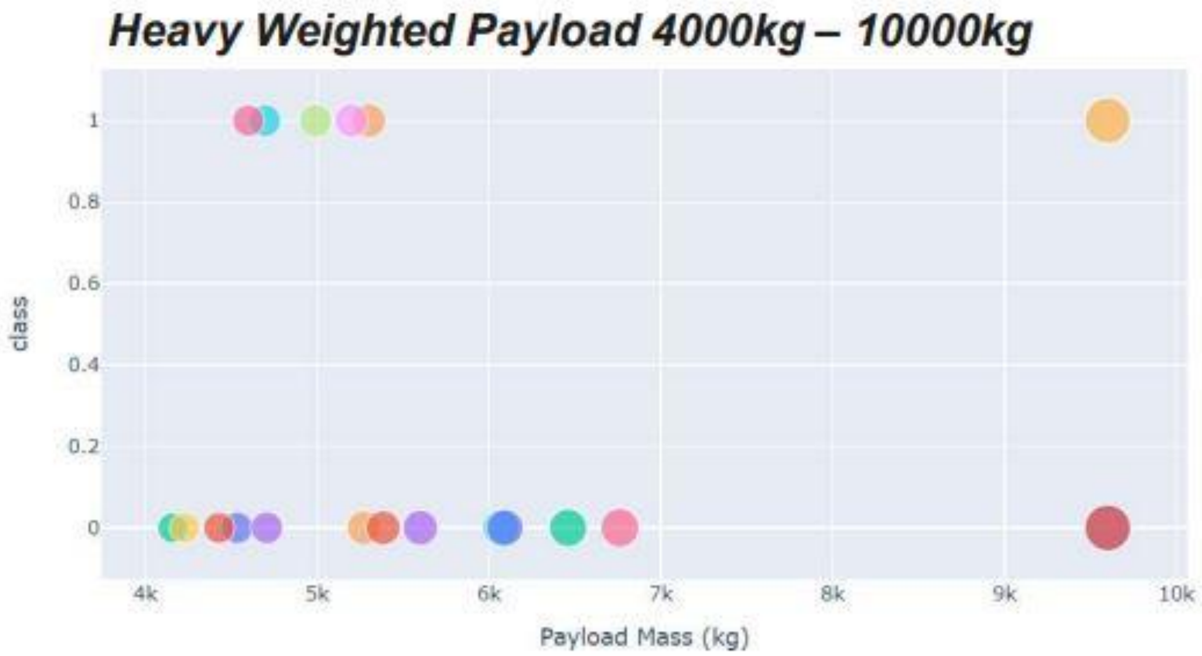
DASHBOARD Pie chart for successful launches



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

DASHBOARD – Pie chart for the launch site with highest success





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

DASHBOARD Payload vs. Launch Outcome scatter plot for all sites

Section 6

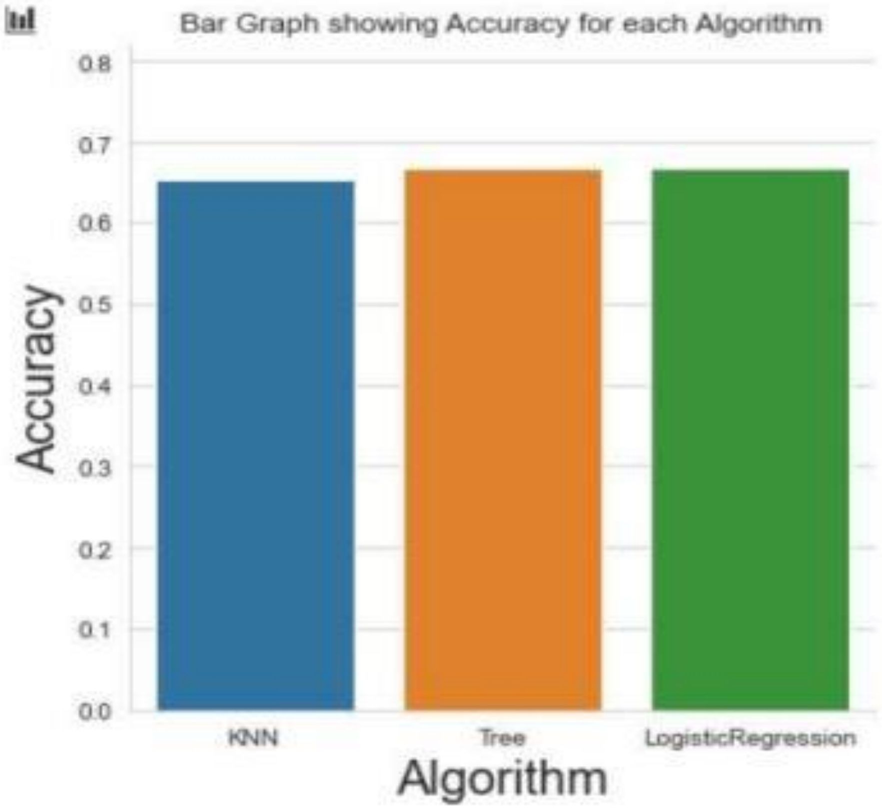
Predictive Analysis (Classification)

Classification Accuracy

As you can see our accuracy is extremely close but we do have a winner its down to decimal places! using this function

```
bestalgorithm = max(algorithms, key=algorithms.get)
```

	Accuracy	Algorithm
0	0.653571	KNN
1	0.667857	Tree
2	0.667857	LogisticRegression



The tree algorithm wins!!

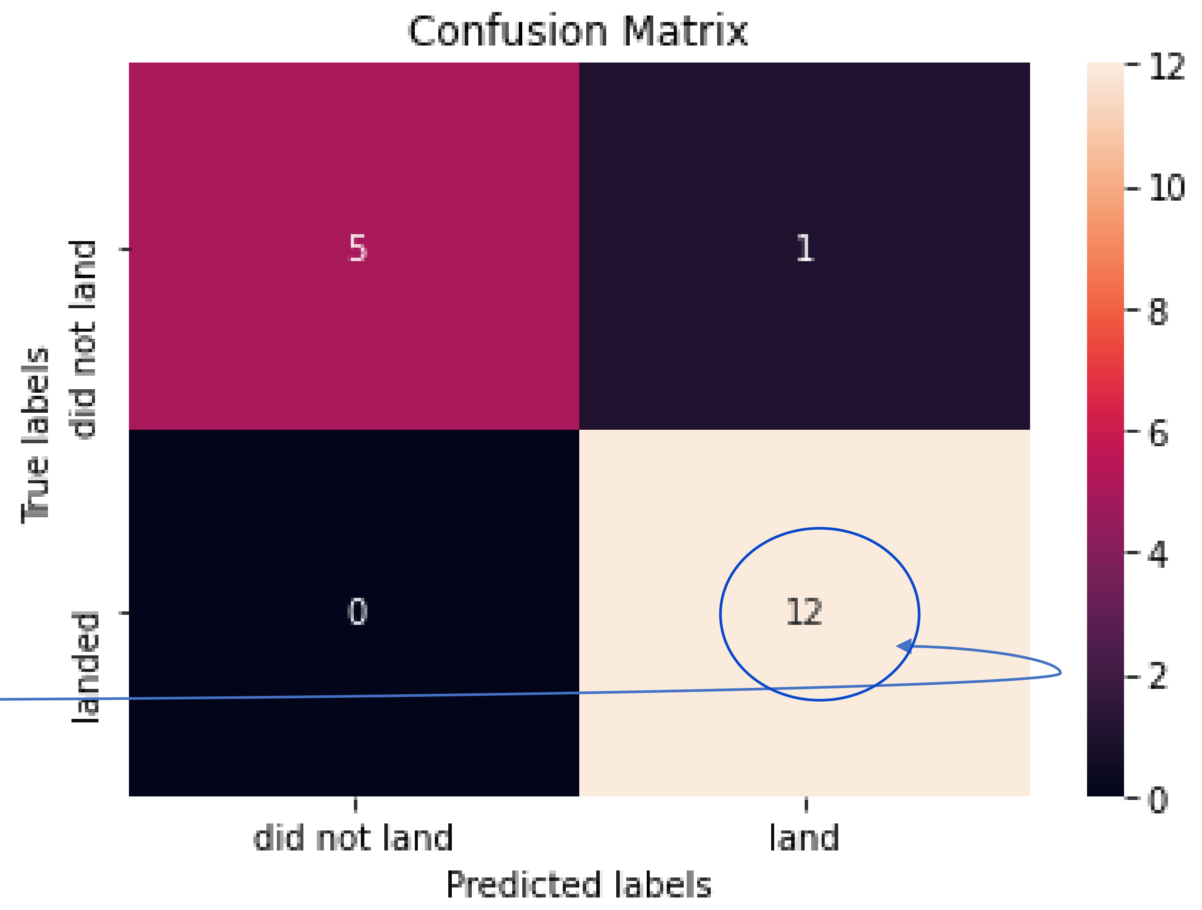
```
Best Algorithm is Tree with a score of 0.6678571428571429
Best Params is : {'criterion': 'gini', 'max_depth': 2, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 2, 'splitter': 'best'}
```

After selecting the best hyperparameters for the decision tree classifier using the validation data, we achieved 83.33% accuracy on the test data.

Confusion Matrix for Tree

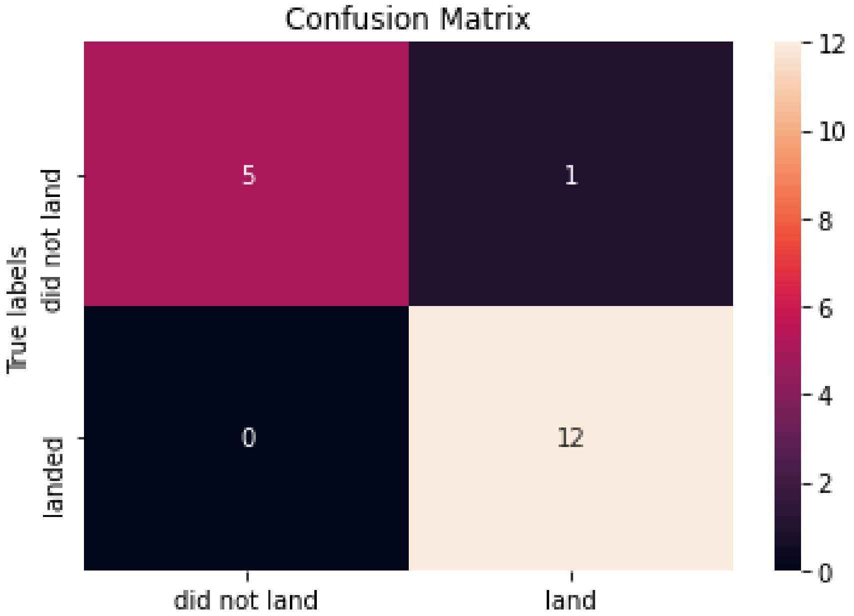
From the model testing, the Decision Tree gave the greatest accuracy. The confusion matrix shows that the model can distinguish between the different classes.

We can also see that the major problem is false positives.



Conclusions

- The Tree Classifier Algorithm is the best for Machine Learning for this dataset.
- Low weighted payloads perform better than the heavier payloads.
- The success rates for SpaceX launches is directly proportional time in years they will eventually perfect the launches.
- We can see that KSC LC-39A had the most successful launches from all the sites.
- Orbit GEO,HEO,SSO,ES-L1 has the best Success Rate.




```
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(tree, parameters, cv=10)
tree_cv.fit(X,Y)
```

```
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']})
```

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

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 14, 'max_
'min_samples_split': 5, 'splitter': 'best'}
accuracy : 0.8888888888888889
```

Appendix A

Appendix B

Haversine Formula

Introduction

The haversine formula determines the great-circle distance between two points on a sphere given their longitudes and latitudes. Important in navigation, it is a special case of a more general formula in spherical trigonometry, the law of haversines, that relates the sides and angles of spherical triangles.

Usage

Why did I use this formula? First of all, I believe the Earth is round/elliptical. I am not a Flat Earth Believer! Jokes aside when doing Google research for integrating my ADGGoogleMaps API with a Python function to calculate the distance using two distinct sets of {longitudinal, latitudinal} list sets. Haversine was the trigonometric solution to solve my requirements above.

Formula

$$a = \sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos\varphi 1 \cdot \cos\varphi 2 \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right)$$

$$c = 2 \cdot \operatorname{atan2}\left(\sqrt{a}, \sqrt{(1-a)}\right)$$

$$d = R \cdot c$$

Thank you!