IBM Developer
SKILLS NETWORK

# Winning Space Race with Data Science

Rohan Ballapragada
December 8th 2023

# Outline

# Executive Summary

- Data Collection via API
- Data Collection via Wikipedia
- Data Wrangling filling null values
- EDA with Data Visualization
- EDA with SQL
- Building interactive map with Folium
- Building a dashboard with Plotly
- Predictive Analytics

# Introduction

- Project background and context

SpaceX has demonstrated to be the largest customer for both public and private enterprises in space travel. They have shown their successes in ventures like sending spacecraft to the space station, their starlink internet constellation, and sending manned missions to the International Space Station. The reason as to why most enterprises seek them out is their ability to launch their rockets cost effectively, a SpaceX falcon 9 rocket costs $62 million dollars whereas rockets from other providers cost $165 million dollars, that is due to their ability to recover and reuse the first stage. If we can determine if the first stage can land, we can determine the cost of the launch. This information can be used if another company wants to compete against SpaceX.

- Problems you want to find answers
    - What are the defining factors in a successful Stage 1 landing?
    - What sort of conditions do we need to place the Falcon 9 so we get a successful Stage 1 landing?

Section 1

# Methodology

# Methodology

## Executive Summary

- Data collection methodology:
    - How data was collected
- Perform data wrangling
    - How data was processed
- 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

- Describe how data sets were collected.

  Data was collected through an API call from the spacexdata.com website

  We next decode the response to a JSON using.json() and turn it into a Pandas dataframe using .json_normalize

  Get information for the dataframe that we would need like BoosterVersion, PayloadMass, and Outcome

  Filter the dataframe to only include Falcon 9 launches and filled in missing values

  Continued to scrape data from Wikipedia for Falcon 9 launchs with BeatifulSoup.

# Data Collection – SpaceX API

- Utilized the get request to the SpaceX API for data collection, cleansing, and formatting.
- https://github.com/ballaprr/IBM_FinalProject_DSAppliedCapstone/blob/main/Week_1/.ipynb_checkpoints/jupyter-labs-spacex-data-collection-api-checkpoint.ipynb

# Data Collection - Scraping

- Used web scraping to Falcon 9 launch data with BeatifulSoup
- https://github.com/ballaprr/IBM_FinalProject_DSAppliedCapstone/blob/main/Week_1/.ipynb_checkpoints/jupyter-labs-webscraping-checkpoint.ipynb

To keep the lab tasks consistent, you will be asked to scrape the data from a snapshot of the `List of Falcon 9 and Falcon Heavy launches` Wikipage updated on `9th June 2021`

In [4]:
```
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 [26]:
```
# 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 [27]:
```
# 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 [7]:
```
# Use soup.title attribute
soup.title
```

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

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

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

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

In [28]:
```
# 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 [30]:
```
# Let's print the third table and check its content
first_launch_table = html_tables[2]
```

You should able to see the columns names embedded in the table header elements `<th>` as follows:

```
<tr>
<th scope="col">Flight No.
</th>
<th scope="col">Date and<br/>time (<a href="/wiki/Coordinated_Universal_Time" title="Coordinated Universal Time">UTC</a
>)
</th>
<th scope="col"><a href="/wiki/List_of_Falcon_9_first-stage_boosters" title="List of Falcon 9 first-stage boosters">Vers
```

# Data Wrangling

- Checked the null values from the SpaceX dataset
- Looked into the amount of missing values for each feature and the types
- Looked at the different and amount of mission outcomes
- Created a binary based landing outcome where all bad outcomes are set to zero and good to one
- https://github.com/ballaprr/IBM_FinalProject_DSAppliedCapstone/blob/main/Week_1/.ipynb_checkpoints/labs-jupyter-spacex-Data%20wrangling-checkpoint.ipynb

In [12]:
```
for i,outcome in enumerate(landing_outcomes.keys()):
    print(i,outcome)
```

```
0 True ASDS
1 None None
2 True RTLS
3 False ASDS
4 True Ocean
5 False Ocean
6 None ASDS
7 False RTLS
```

We create a set of outcomes where the second stage did not land successfully:

In [13]:
```
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
bad_outcomes
```

Out[13]:
```
{'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

**TASK 4: Create a landing outcome label from Outcome column**

Using the `Outcome`, create a list where the element is zero if the corresponding row in `Outcome` is in the set `bad_outcome`; otherwise, it's one. Then assign it to the variable `landing_class`:

In [16]:
```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
landing_class = []
for outcome in df['Outcome']:
    if outcome in bad_outcomes:
        landing_class.append(0)
    else:
        landing_class.append(1)
```

This variable will represent the classification variable that represents the outcome of each launch. If the value is zero, the first stage did not land successfully; one means the first stage landed Successfully

In [17]:
```
df['Class']=landing_class
df[['Class']].head(8)
```

Out[17]:

| | Class |
|---|---|
| 0 | 0 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 1 |
| 7 | 1 |

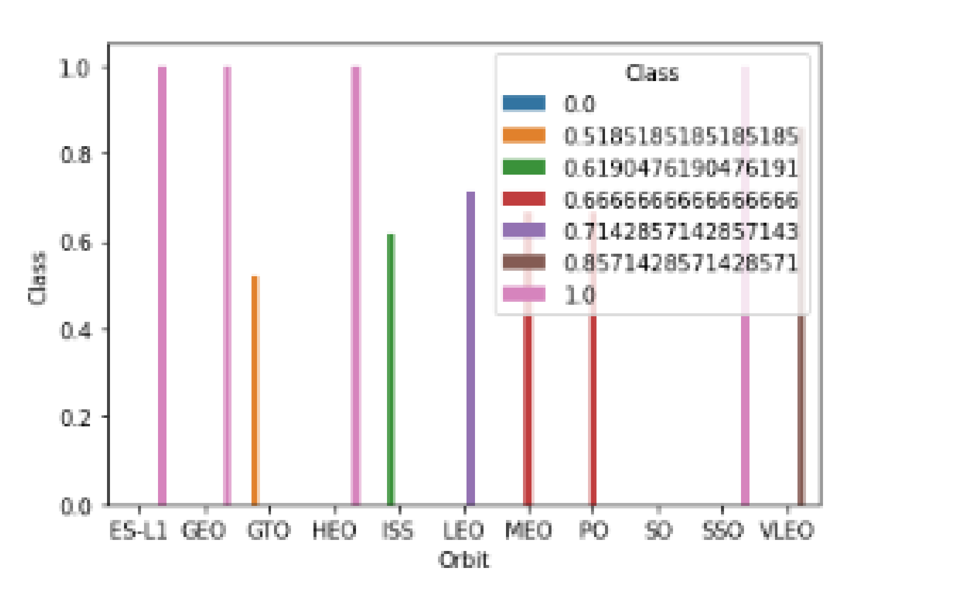
In [18]:
```
df.head(5)
```

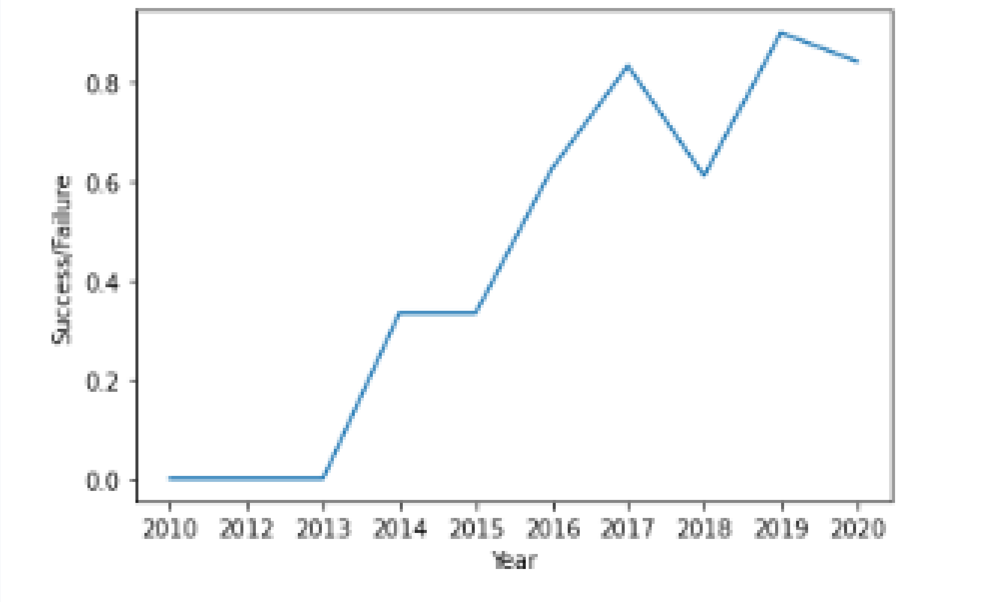
Out[18]:

| | FlightNumber | Date | BoosterVersion | PayloadMass | Orbit | LaunchSite | Outcome | Flights | GridFins | Reused | Legs | LandingPad | Block | ReusedCount | [illegible] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2010-06-04 | Falcon 9 | 6104.959412 | LEO | CCAFS SLC 40 | None None | 1 | False | False | False | NaN | 1.0 | 0 | B |
| 1 | 2 | 2012- | Falcon 9 | 525.000000 | LEO | CCAFS [illegible] | None [illegible] | 1 | False | False | False | NaN | 1.0 | 0 | B |

# EDA with Data Visualization

- The graph on the bottom right shows the success/failure rate of each Falcon 9 launch from 2010 onwards, the success rate continually trending upwards. The graph on the left shows the success rate for each orbit type.
- https://github.com/ballaprr/IBM_FinalProject_DSAppliedCapstone/blob/main/Week_2/jupyter-labs-eda-dataviz.ipynb



# EDA with SQL

- Some of the following SQL queries I have written:
  - Names of the unique launch sites in the space mission
  - 5 records where launch sites begin starting with the string ‘CCA’
  - Total payload launch by NASA (CRS) and average payload by booster version F9 v1.1
  - Date of the first successful landing outcome in ground pad was achieves
  - Names of boosters which have success in drone ship and payload between 4000 and 6000
  - Rank the count of landing outcomes (Failure (drone ship) or Success (ground pad)) between 2010-06-04 and 2017-03-20 in descending order
- https://github.com/ballaprr/IBM_FinalProject_DSAppliedCapstone/blob/main/Week_2/jupyter-labs-eda-sql-coursera_sqllite.ipynb

# Build an Interactive Map with Folium

- On the folium map are marked coordinates for the Launch Sites. The success/failed launch attempts, and the distances between each launch site proximity.
- The markers are colored so we know which launch sites had the most or least successful launches.
- https://github.com/ballaprr/IBM_FinalProject_DSAppliedCapstone/blob/main/Week_3/lab_jupyter_launch_site_location.ipynb

# Build a Dashboard with Plotly Dash

- On the plotly interactive dashboard there is a pie chart on each of the SpaceX sites and there is a scatter plot showing the relationship between the payload mass and the success.
- These plots were added to determine how successful each of the orbital launch sites are how much the payload had to do with it.
- https://github.com/ballaprr/IBM_FinalProject_DSAppliedCapstone/blob/main/Week_3/spacex_dash_app.py

# Predictive Analysis (Classification)

- I imported the desired libraries, pandas, numpy, and sickit-learn, loaded the data, standardized the data, split into train/test splits.
- Built different machine learning models and tuned to different hyperparameters using GridSearchCV
- Used accuracy as our metric to test the models
- Add the GitHub URL of your completed predictive analysis lab, as an external reference and peer-review purpose

### TASK 2

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

```
In [8]:
# students get this
from sklearn.preprocessing import StandardScaler
sc_1 = StandardScaler()
sc_1.fit(X).transform(X)
```

```
Out[8]: 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]])
```

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`

```
In [9]:
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 [10]:
Y_test.shape
Out[10]: (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`.

```
In [11]:
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

```
In [12]:
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()
logreg_cv = GridSearchCV(lr, parameters, cv=10)
```

# Results

- When looking at the different models:
  - Logistics Regression
  - Support Vector Machines
  - Decision Tree
  - K nearest neighbor
- The Decision tree is giving the best fit model

In [31]:
```
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 [33]:
```
knn = GridSearchCV(KNN,parameters,scoring='accuracy',cv=10)
knn_cv = knn.fit(X_train, Y_train)
```

In [34]:
```
print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
print("accuracy :",knn_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 3, 'p': 1}
accuracy : 0.6642857142857143
```

### TASK 11

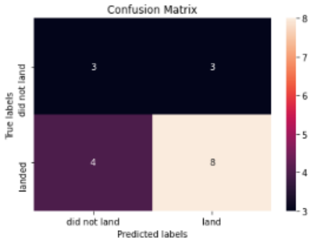
Calculate the accuracy of knn_cv on the test data using the method `score`:

In [35]:
```
knn_cv.score(X_test, Y_test)
```

Out[35]: 0.6111111111111112

We can plot the confusion matrix

In [36]:
```
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```




### TASK 12

Find the method performs best:

In [37]:
```
print('Accuracy for Logistics Regression method:', logreg_cv.score(X_test, Y_test))
print( 'Accuracy for Support Vector Machine method:', svm_cv.score(X_test, Y_test))
print('Accuracy for Decision tree method:', tree_cv.score(X_test, Y_test))
print('Accuracy for K nearsdt neighbors method:', knn_cv.score(X_test, Y_test))
```

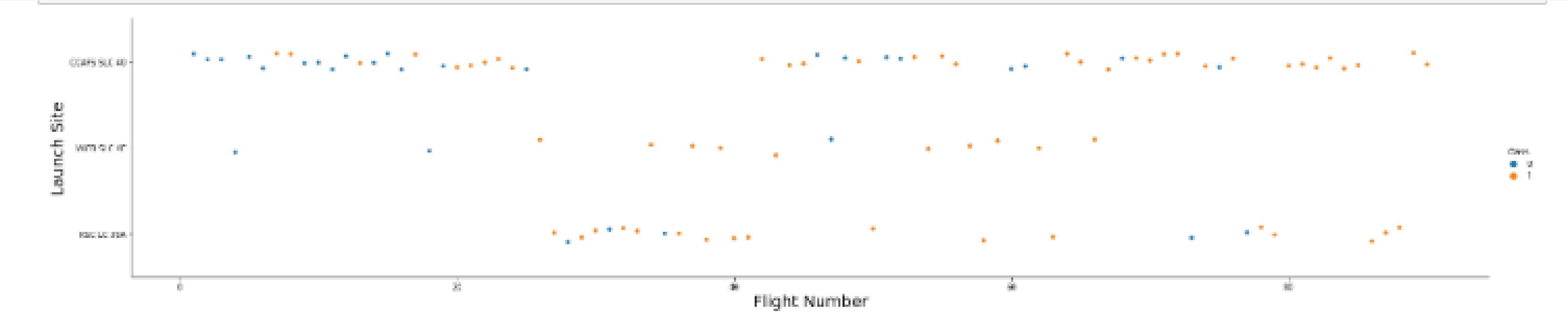
```
Accuracy for Logistics Regression method: 0.8333333333333334
Accuracy for Support Vector Machine method: 0.6666666666666666
Accuracy for Decision tree method: 0.8333333333333334
Accuracy for K nearsdt neighbors method: 0.6111111111111112
```

Section 2

# Insights drawn from EDA

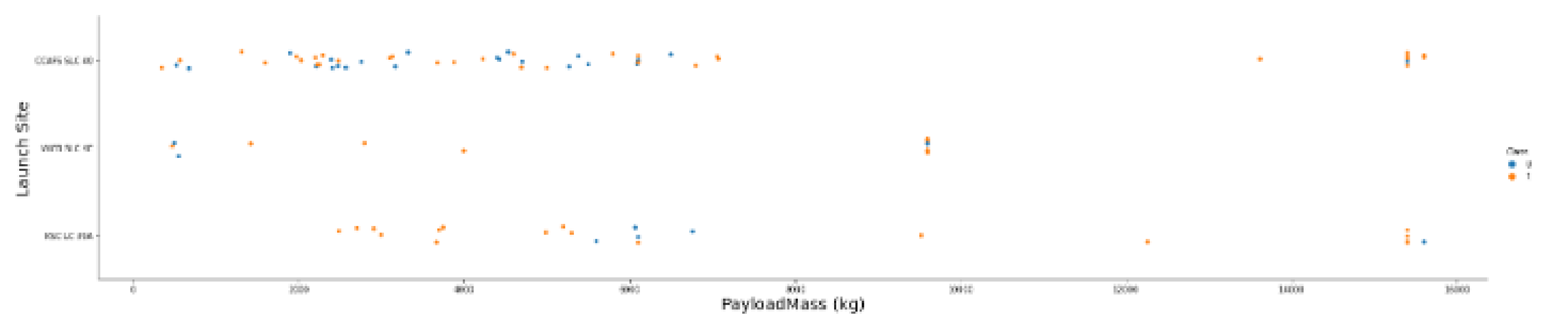
# Flight Number vs. Launch Site

- The more flights the higher the success rate



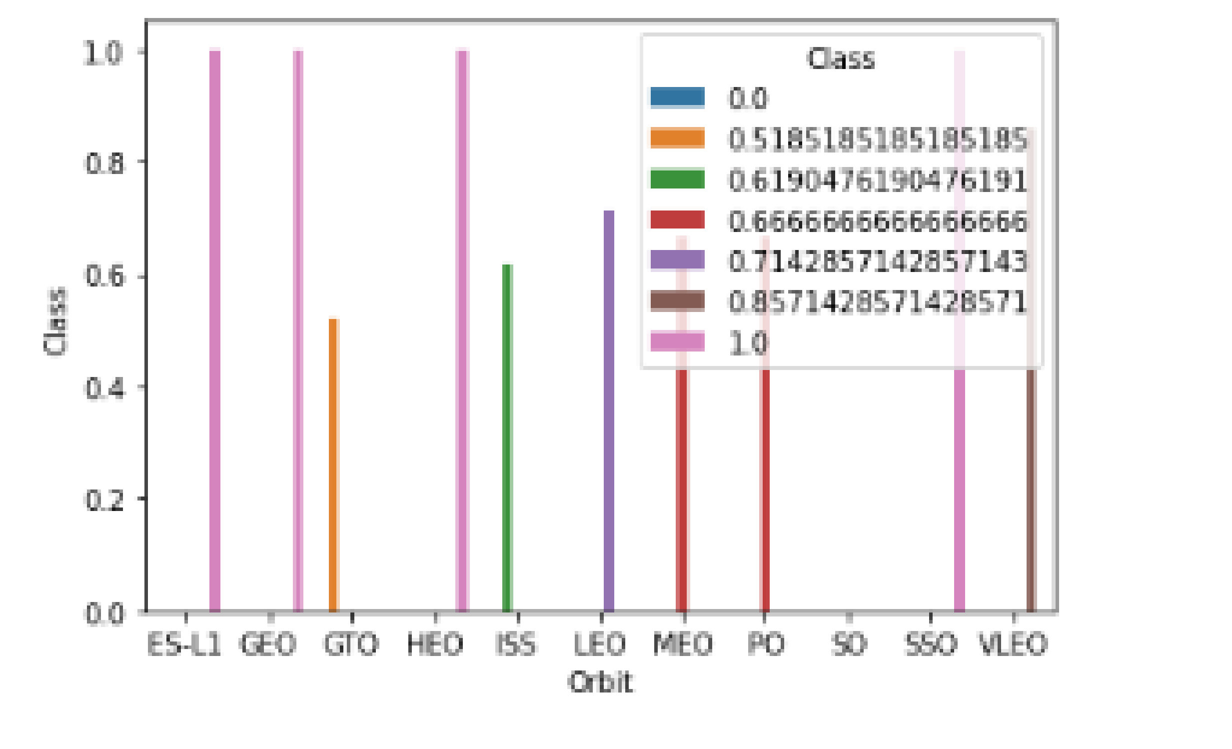
# Payload vs. Launch Site

- The success between the launch and its payload are not correlated



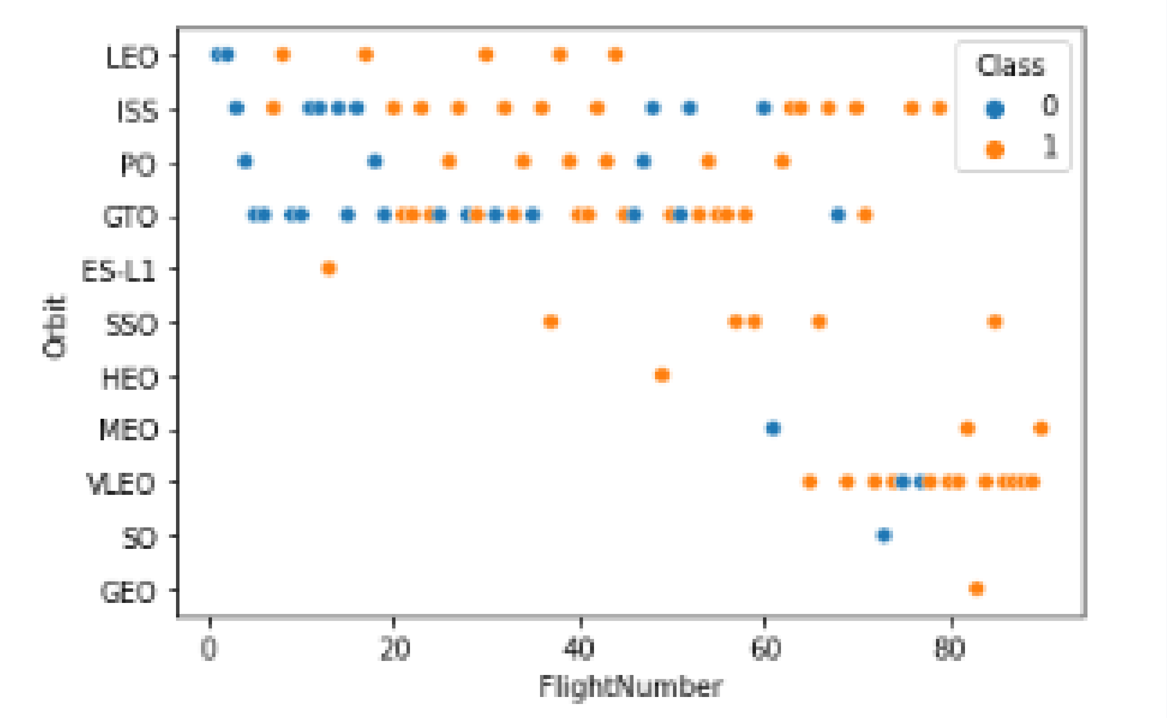
# Success Rate vs. Orbit Type

- The bar chart demonstrates the success rate for each orbit. The following are ES-L1, GEO, HEO, and SSO.



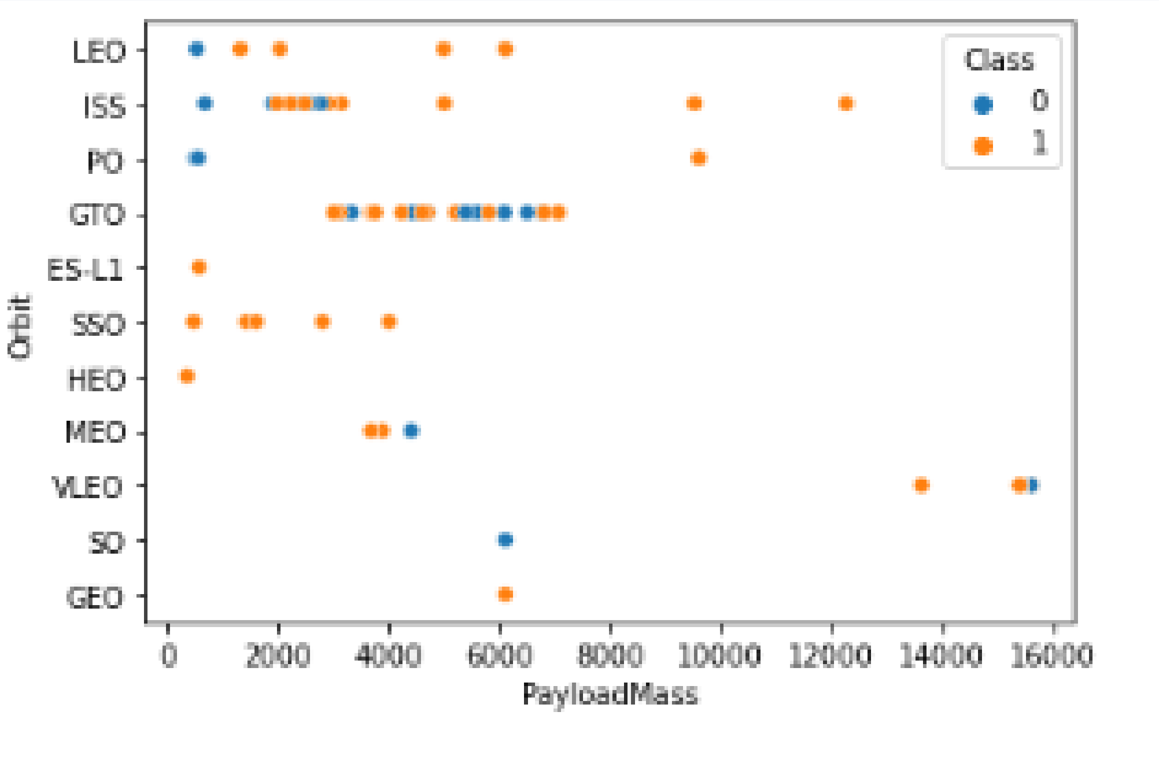
# Flight Number vs. Orbit Type

- Based off our findings from the plot there is higher success for more flights in each orbit with the exception of GTO.



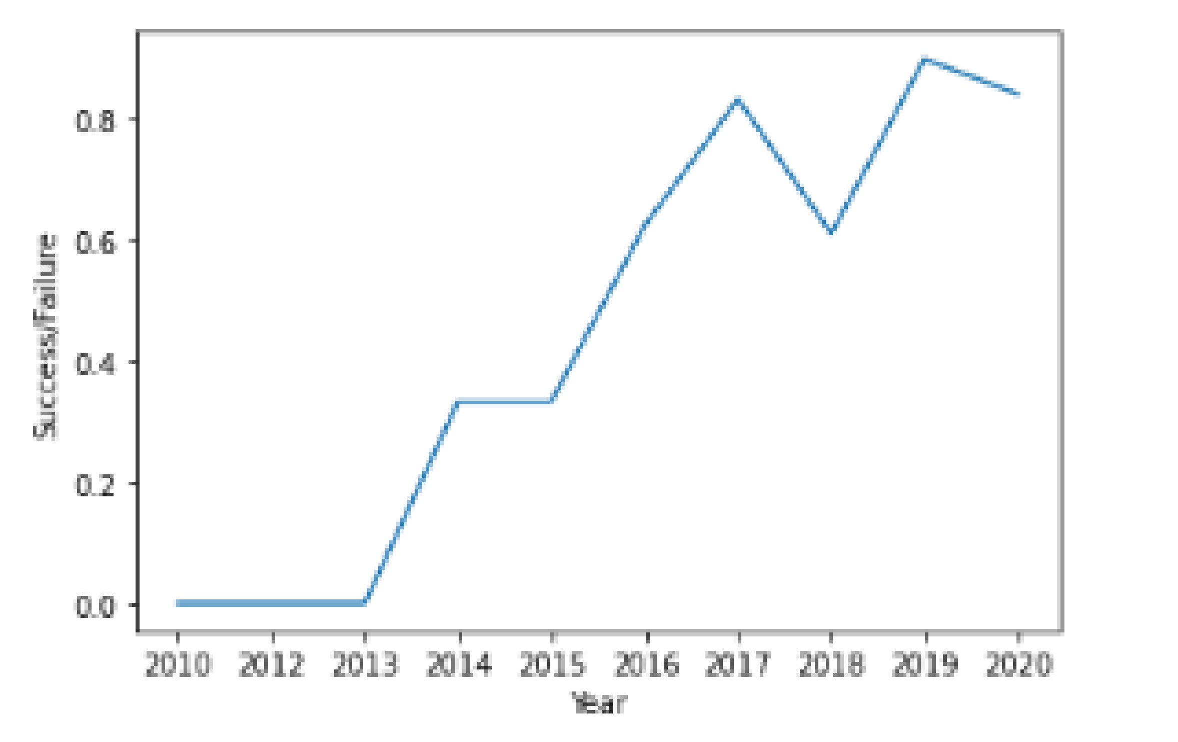
# Payload vs. Orbit Type

- Looking at the graph, there isn't a correlation between the orbit and the payload mass no matter the orbit.
- There will need to be more launches with a high payload



# Launch Success Yearly Trend

- As the years trend higher, the success rate grows.



# All Launch Site Names

- I used the distinct keyword to obtain the unique launch sites from the table.
- Displayed are the following launch sites: CCAFS LC-40, VAFB SLC-4E, KSC LC-39A, CCAFS SLC-40.

**Note: If the column names are in mixed case enclose it in double quotes For Example "Landing_Outcome"**

### Task 1

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

```
[9]: %sql select distinct Launch_Site from SPACEXTABLE;
```

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

[9]:

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

# Launch Site Names Begin with 'CCA'

- Used the like operator to search for a specified pattern, the Launch Site starts with "CCA" and ends with "%" to denote zero or more characters. And limit 5 to limit our query to 5 records.

## Task 2

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

[10]:
```
%sql select * from SPACEXTABLE where Launch_Site like 'CCA%' limit 5;
```

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

[10]:

| 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 | 7:44:00 | F9 v1.0 B0005 | CCAFS LC-40 | Dragon demo flight C2 | 525 | LEO (ISS) | NASA (COTS) | Success | No attempt |
| 2012-10-08 | 0:35:00 | 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

- Uses the sum query to calculate the total of the payload and the where clause to specify the launch customer.

## Task 3

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

```
[11]: %sql select sum(PAYLOAD_MASS__KG_) from SPACEXTABLE where Customer = 'NASA (CRS)';
```

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

[11]:

| sum(PAYLOAD_MASS__KG_) |
| --- |
| 45596 |

# Average Payload Mass by F9 v1.1

- Calculate the average payload mass carried by booster version F9 v1.1

## Task 4

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

```
[12]: %sql SELECT AVG(PAYLOAD_MASS__KG_) FROM SPACEXTABLE where Booster_Version like 'F9 v1.1%'
```

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

[12]:

| AVG(PAYLOAD_MASS__KG_) |
| --- |
| 2534.6666666666665 |

# First Successful Ground Landing Date

- Calculate the minimum date and use the where clause to only include the successful landings on the ground pad.

## Task 5

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

*Hint:Use min function*

```
[13]: %sql SELECT min(Date) from SPACEXTABLE where Landing_Outcome = 'Success (ground pad)';
```

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

[13]:

| min(Date) |
|---|
| 2015-12-22 |

# Successful Drone Ship Landing with Payload between 4000 and 6000

- Uses the where clause to set the Landing Outcome to the successful first stage landings on the drone ship and the between clause to specify the boosters between 4000 and 6000.

### Task 6

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

[14]:
```
%sql select Booster_Version from SPACEXTABLE where Landing_Outcome = 'Success (drone ship)' and PAYLOAD_MASS__KG_ between 4000 and 6000;
```

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

[14]:

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

# Total Number of Successful and Failure Mission Outcomes

- Use the group by operator to group all the mission outcomes and the count operator to count see the number for each outcome.

## Task 7

List the total number of successful and failure mission outcomes

```
[15]: %sql select Mission_Outcome, count(*) from SPACEXTABLE group by Mission_Outcome;
```

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

[15]:

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

# Boosters Carried Maximum Payload

- Use the subquery to find the maximum payload from the table to find the corresponding Booster Version.

### Task 8

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

[16]:
```
%sql select Booster_Version from SPACEXTABLE where PAYLOAD_MASS__KG_ = (select max(PAYLOAD_MASS__KG_) from SPACEXTABLE);
```

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

[16]:

| 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

- List the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015

## Task 9

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

**Note: SQLLite does not support monthnames. So you need to use substr(Date, 6,2) as month to get the months and substr(Date,0,5)='2015' for year.**

[25]:
```
%sql select substr(Date, 6,2) as month, Booster_Version, Launch_Site, Landing_Outcome from SPACEXTABLE \
where Landing_Outcome = 'Failure (drone ship)' and substr(Date,0,5)='2015';
```

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

[25]:

| month | Booster_Version | Launch_Site | Landing_Outcome |
|---|---|---|---|
| 01 | F9 v1.1 B1012 | CCAFS LC-40 | Failure (drone ship) |
| 04 | F9 v1.1 B1015 | CCAFS LC-40 | Failure (drone ship) |

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

- Selected the Landing outcomes from the data and specified the dates we want using the where clause.
- Used the Group by operator to specify the landing outcome and order by to order by landing outcome in descending order

### Task 10

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.

```
29]: %sql select Landing_Outcome, count(*) as count_landing_outcome from SPACEXTABLE where DATE between '2010-06-04' and '2017-03-20' group by Landing_Outcome order by count_landing_outcome desc;
```

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

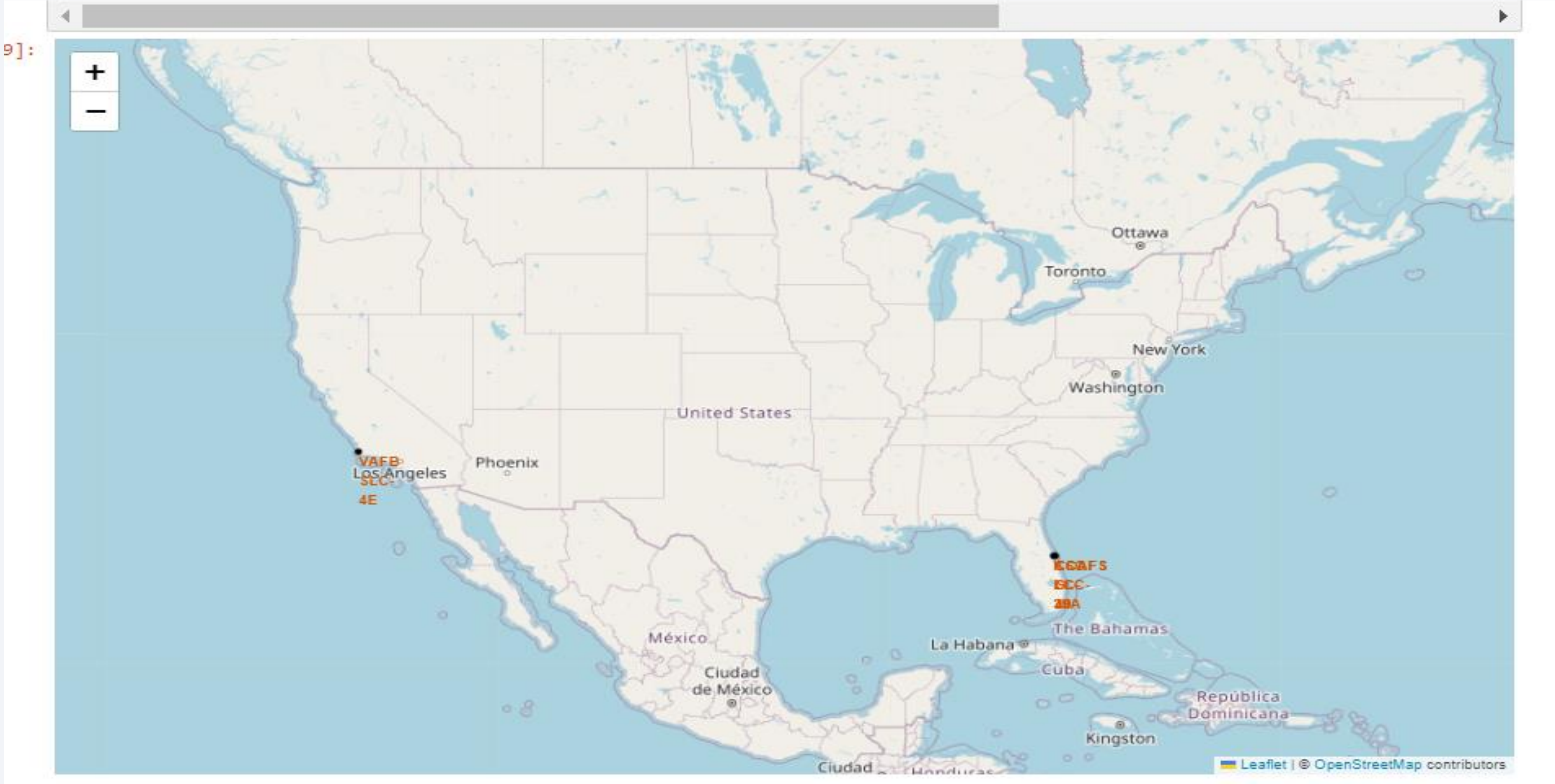
29]:

| Landing_Outcome | count_landing_outcome |
|---|---|
| No attempt | 10 |
| Success (drone ship) | 5 |
| Failure (drone ship) | 5 |
| Success (ground pad) | 3 |
| Controlled (ocean) | 3 |
| Uncontrolled (ocean) | 2 |
| Failure (parachute) | 2 |
| Precluded (drone ship) | 1 |

Section 3

# Launch Sites Proximities Analysis

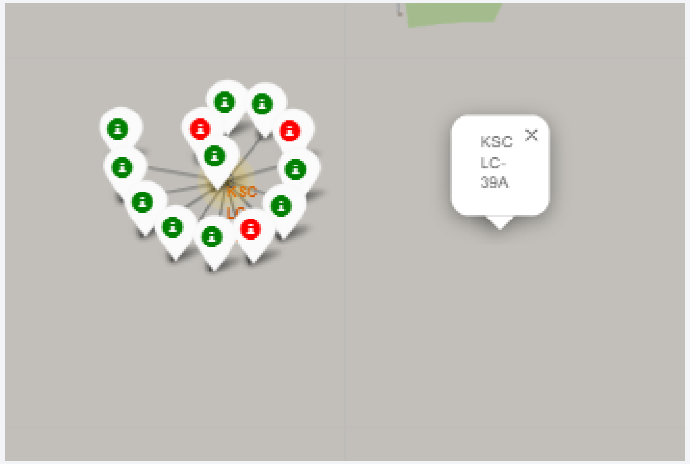
# Global Launch map

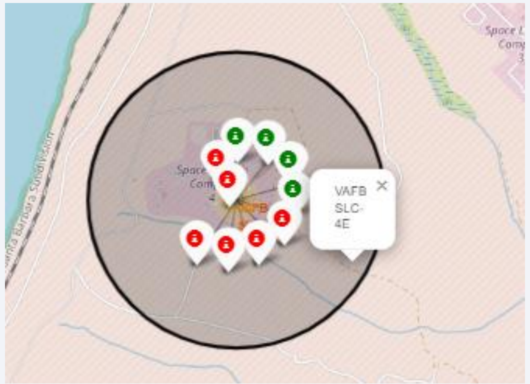


# California and Florida Launches

**Florida Launches:**

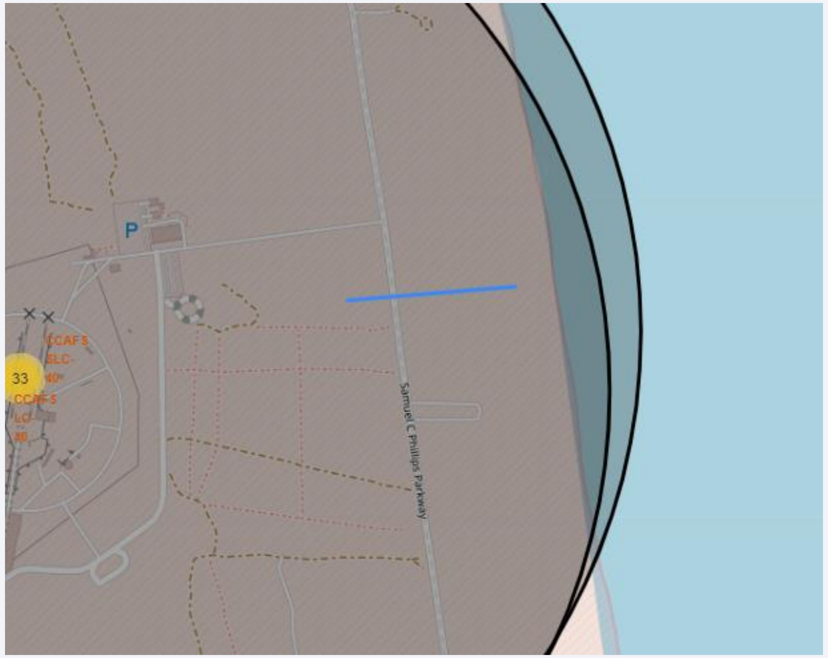

**California launches:**

# Close proximity to launch site

Close to the coastline but not near the railway, highway, nor city.

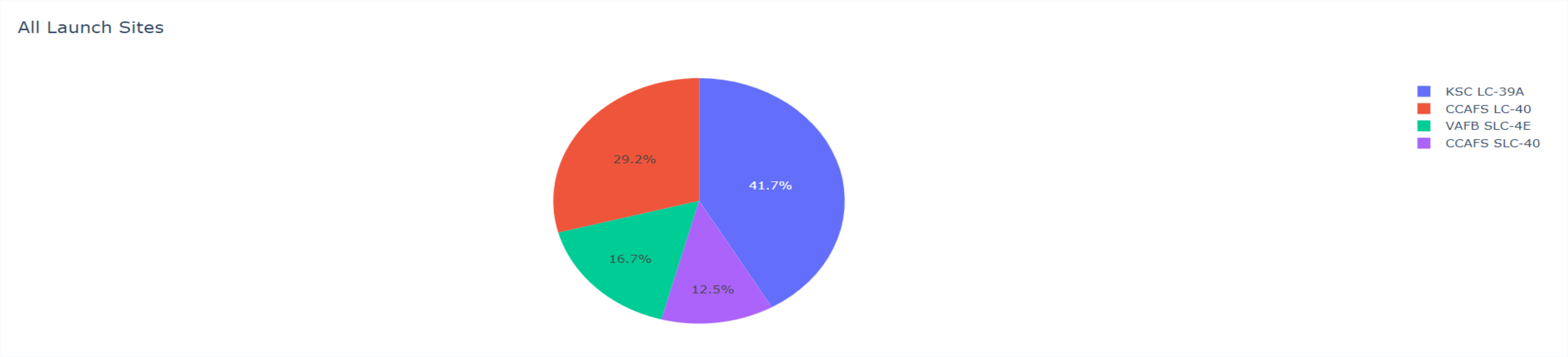



Section 4

# Build a Dashboard with Plotly Dash

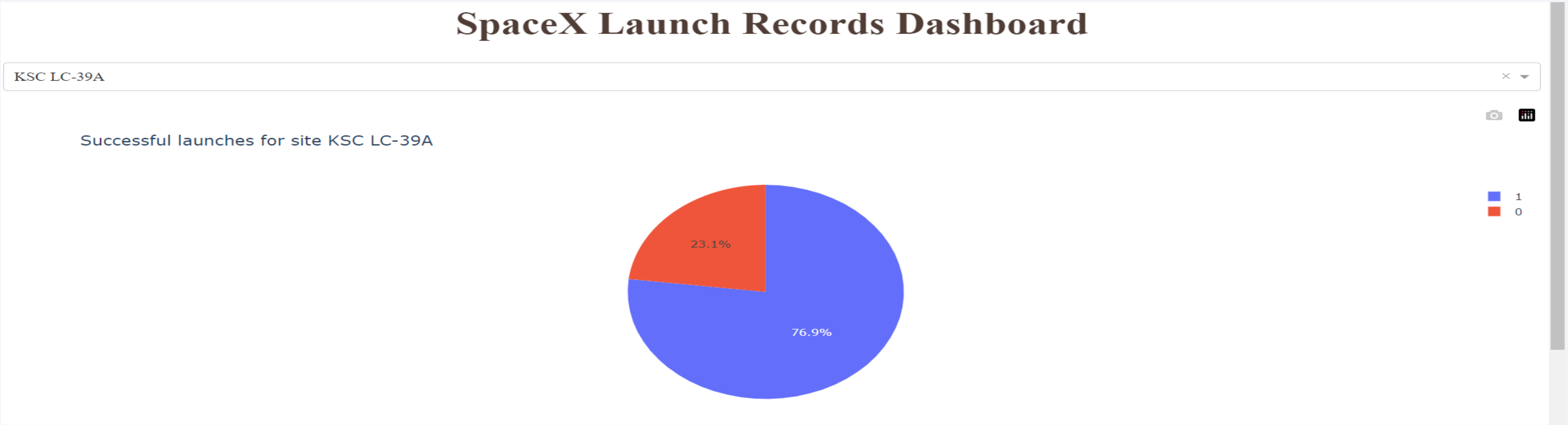
# Total Successful launch sites

All successful launch sites, the KSCLA-39A had the most successful launch site



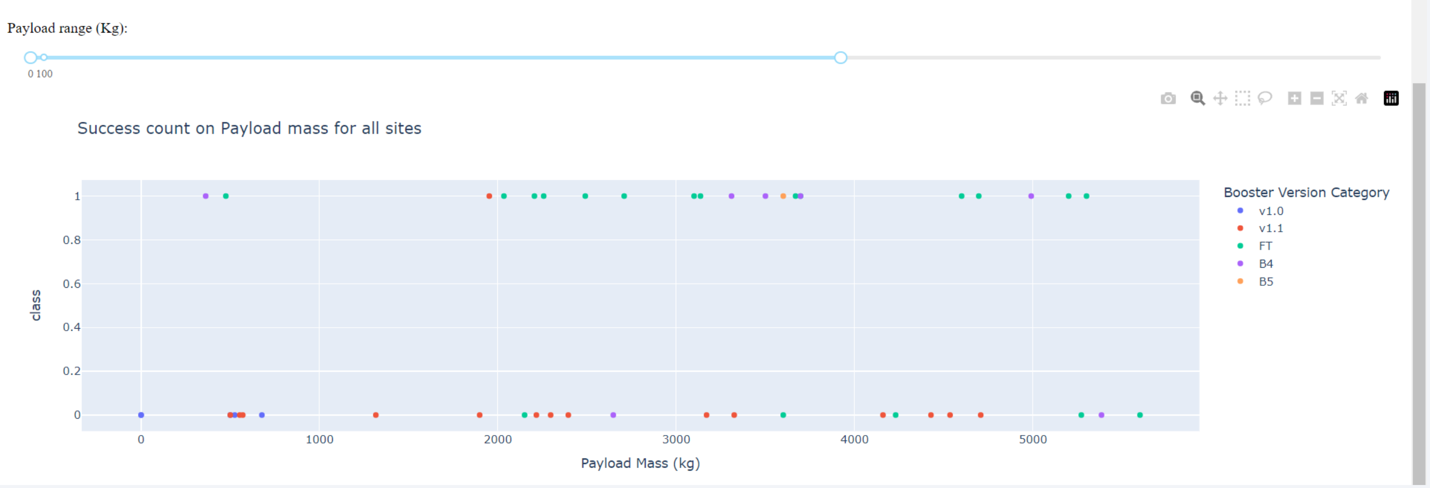
# Most successful launch site

KSC LC-39A is the most successful launch site with a 76.9% rating



# <Dashboard Screenshot 3>

- The success rates between high and low payloads are nonvariable.

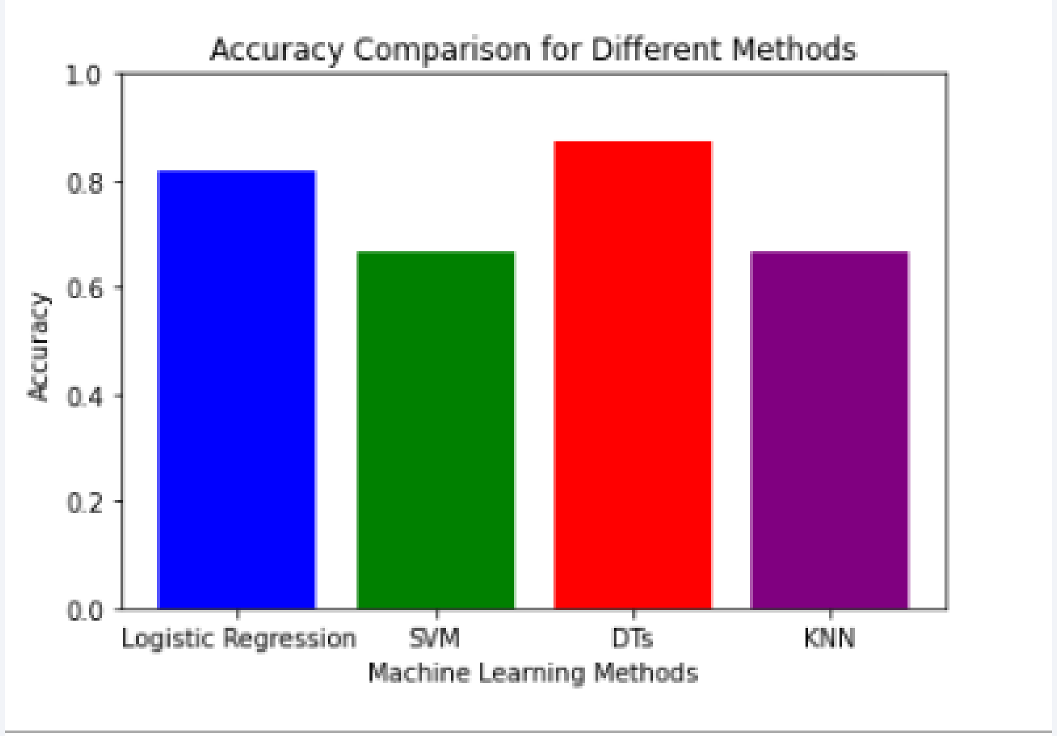



Section 5

# Predictive Analysis (Classification)

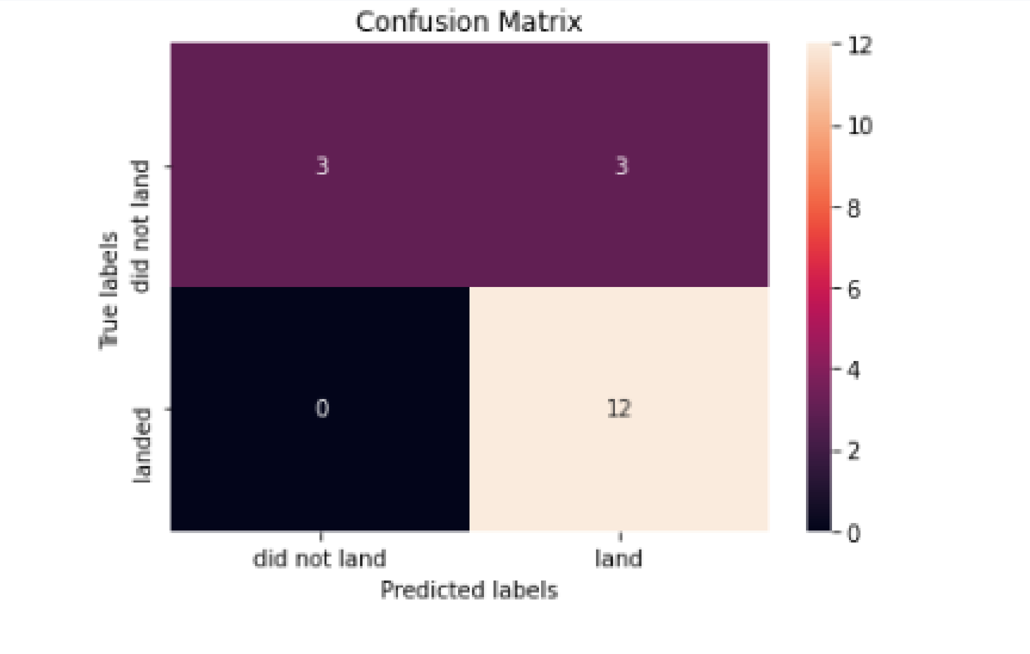
# Classification Accuracy

- When looking at the accuracy comparisons between the four models, the Decision Tree comes out as the best model.



# Confusion Matrix

- Looking at the Confusion Matrix there are 12 True positives and 3 false negatives which is good.
- There is the issue of true negatives where rockets that did not land were identified as successful.



# Conclusions

- Over time from 2013 onwards SpaceX flights get more successful over time, that includes every orbit except for GTO.
- No correlation between a successful SpaceX flight and payload
- The most successful landing site is KSC LC-39A
- Decision Tree classifier is the best machine learning algorithm for this task

# Appendix

- https://github.com/ballaprr/IBM_FinalProject_DSAppliedCapstone/tree/main/CSV_files
- Rest of data already included in slides

Thank you!