# IBM Data Science Capstone Project

Space X Falcon 9 Landing Analysis

Kartik Ningaraj Yanigar

01-01-2023

# Outline

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

# Executive Summary

- **Summary of Methodologies:**
  - Data Collection
  - Data Wrangling
  - Exploratory Data Analysis
  - Interactive Visual Analytics
  - Predictive Analysis (Classification)

- **Summary of Results:**
  1. Exploratory Data Analysis (EDA) results
  2. Geospatial analytics
  3. Interactive dashboard
  4. Predictive analysis of classification models

# Introduction

- Project background and context
  - Falcon 9 rocket launches by SpaceX cost about $62 million. This is significantly less expensive than other providers (which typically cost more than $165 million), and a large portion of the savings is due to SpaceX's ability to land and reuse the rocket's first stage.
- Problems you want to find answers
  - We can estimate the cost of a launch and use this data to decide whether or not a different company should compete against SpaceX for a rocket launch if we can anticipate whether the first stage will land.

In the end, this project will be able to forecast if the Space X Falcon 9 first stage will land safely.

# Methodology

## Executive Summary

- **Data collection methodology:**
  - Making GET requests to the SpaceX REST API
  - Web Scraping

- **Perform data wrangling**
  - Using the .fillna() method to remove NaN values
  - Using the .value_counts() method to determine the following:
    - Number of launches on each site
    - Number and occurrence of each orbit
    - Number and occurrence of mission outcome per orbit type
  - Creating a landing outcome label that shows the following:
    - 0 when the booster did not land successfully
    - 1 when the booster did land successfully

- **Perform exploratory data analysis (EDA) using visualization and SQL**
  - Using Pandas and Matplotlib to visualize relationships between variables, and determine patterns

- **Perform interactive visual analytics using Folium and Plotly Dash**
  - Geospatial analytics using Folium
  - Creating an interactive dashboard using Plotly Dash

- **Perform predictive analysis using classification models**
  - Using Scikit-Learn to:
    - Pre-process (standardize) the data
    - Split the data into training and testing data using `train_test_split`
    - Train different classification models
    - Find hyperparameters using `GridSearchCV`
  - Plotting confusion matrices for each classification model
  - Assessing the accuracy of each classification model

Using the SpaceX API to retrieve data about launches, including information about the rocket used, payload delivered, launch specifications, landing specifications, and landing outcome.

- Make a GET response to the SpaceX REST API
- Convert the response to a .json file then to a Pandas DataFrame

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

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

```
# Use json_normalize method to convert the json result into a dataframe
data = pd.json_normalize(response.json())
```

- Use custom logic to clean the data (see Appendix)
- Define lists for data to be stored in
- Call custom functions (see Appendix) to retrieve data and fill the lists
- Use these lists as values in a dictionary and construct the dataset

```
#Global variables
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
ReusedCount = []
Serial = []
Longitude = []
Latitude = []
```

```
# Call getBoosterVersion
getBoosterVersion(data)
```

```
# Call getLaunchSite
getLaunchSite(data)
```

```
# Call getPayloadData
getPayloadData(data)
```

```
# Call getCoreData
getCoreData(data)
```

```
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}
```

- Create a Pandas DataFrame from the constructed dictionary dataset

```
# Create a data from launch_dict
df = pd.DataFrame.from_dict(launch_dict)
```

- Filter the DataFrame to only include Falcon 9 launches
- Reset the FlightNumber column
- Replace missing values of PayloadMass with the mean PayloadMass value

```
data_falcon9 = df[df['BoosterVersion']!='Falcon 1']
```

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

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

GitHub Link

Web scraping to collect Falcon 9 historical launch records from a Wikipedia page titled List of Falcon 9 and Falcon Heavy launches.

- Request the HTML page from the static URL
- Assign the response to an object

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

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

- Create a BeautifulSoup object from the HTML response object
- Find all tables within the HTML page

```
soup = BeautifulSoup(data, 'html5lib')
html_tables = soup.find_all('table')
```

- Collect all column header names from the tables found within the HTML page

```
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)
```

- Use the column names as keys in a dictionary
- Use custom functions and logic to parse all launch tables (see Appendix) to fill the dictionary values

```
launch_dict= dict.fromkeys(column_names)

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

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

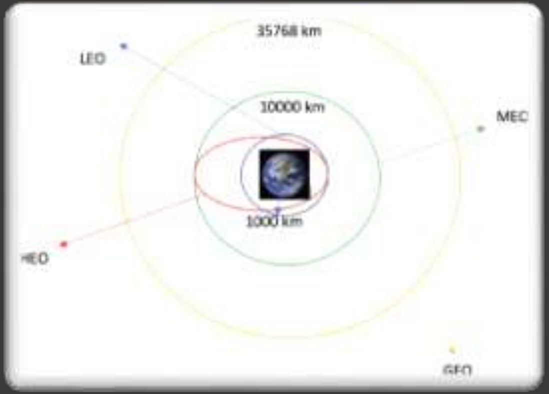
- Convert the dictionary to a Pandas DataFrame ready for export

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

GitHub Link

### Context:

- The SpaceX dataset contains several Space X launch facilities, and each location is in the `LaunchSite` column.
- Each launch aims to a dedicated orbit, and some of the common orbit types are shown in the figure below. The orbit type is in the `Orbit` column.




### Initial Data Exploration:

- Using the `.value_counts()` method to determine the following:

1. Number of launches on each site
2. Number and occurrence of each orbit
3. Number and occurrence of landing outcome per orbit type

```
# Apply value_counts() on column LaunchSite
df['LaunchSite'].value_counts()
```

```
CCAFS SLC 40    55
KSC LC 39A      22
VAFB SLC 4E     13
Name: LaunchSite, dtype: int64
```

```
# Apply value_counts on Orbit column
df['Orbit'].value_counts()
```

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

```
# landing_outcomes = values on Outcome column
landing_outcomes = df['Outcome'].value_counts()
landing_outcomes
```

```
True ASDS      41
None None      19
True RTLS      14
False ASDS      6
True Ocean      5
None ASDS       2
False Ocean     2
False RTLS      1
Name: Outcome, dtype: int64
```

GitHub Link

## Context:

- The landing outcome is shown in the Outcome column:
  - True Ocean – the mission outcome was successfully landed to a specific region of the ocean
  - False Ocean – the mission outcome was unsuccessfully landed to a specific region of the ocean.
  - True RTLS – the mission outcome was successfully landed to a ground pad
  - False RTLS – the mission outcome was unsuccessfully landed to a ground pad.
  - True ASDS – the mission outcome was successfully landed to a drone ship
  - False ASDS – the mission outcome was unsuccessfully landed to a drone ship.
  - None ASDS and None None – these represent a failure to land.

## Data Wrangling:

- To determine whether a booster will successfully land, it is best to have a binary column, i.e., where the value is 1 or 0, representing the success of the landing.
- This is done by:

1. Defining a set of unsuccessful (bad) outcomes, bad_outcome
2. Creating a list, landing_class, where the element is 0 if the corresponding row in Outcome is in the set bad_outcome, otherwise, it's 1.
3. Create a Class column that contains the values from the list landing_class
4. Export the DataFrame as a .csv file.

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

```
{'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

```
# 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)
```

```
df['Class']=landing_class
```

```
df.to_csv("dataset_part\_2.csv", index=False)
```

GitHub Link

# Exploratory data analysis (eda) – visualization

- SCATTER CHARTS

- Scatter charts were produced to visualize the relationships between:
- Flight Number and Launch Site
- Payload and Launch Site
- Orbit Type and Flight Number
- Payload and Orbit Type

Scatter charts are useful to observe relationships, or correlations, between two numeric variables.

- BAR CHART

- A bar chart was produced to visualize the relationship between:
- Success Rate and Orbit Type

Bar charts are used to compare a numerical value to a categorical variable. Horizontal or vertical bar charts can be used, depending on the size of the data.

- LINE CHARTS

- Line charts were produced to visualize the relationships between:
- Success Rate and Year (i.e. the launch success yearly trend)

Line charts contain numerical values on both axes, and are generally used to show the change of a variable over time.

GitHub Link

- To gather some information about the dataset, some SQL queries were performed.

- The SQL queries performed on the data set were used to:

1. Display the names of the unique launch sites in the space mission
2. Display 5 records where launch sites begin with the string ‘CCA’
3. Display the total payload mass carried by boosters launched by NASA (CRS)
4. Display the average payload mass carried by booster version F9 v1.1
5. List the date when the first successful landing outcome on a ground pad was achieved
6. List the names of the boosters which had success on a drone ship and a payload mass between 4000 and 6000 kg
7. List the total number of successful and failed mission outcomes
8. List the names of the booster versions which have carried the maximum payload mass
9. List the failed landing outcomes on drone ships, their booster versions, and launch site names for 2015
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

GitHub Link

- The following steps were taken to visualize the launch data on an interactive map:

1. Mark all launch sites on a map
    - Initialise the map using a Folium `Map` object
    - Add a `folium.Circle` and `folium.Marker` for each launch site on the launch map

2. Mark the success/failed launches for each site on a map
    - As many launches have the same coordinates, it makes sense to cluster them together.
    - Before clustering them, assign a marker colour of successful (class = 1) as green, and failed (class = 0) as red.
    - To put the launches into clusters, for each launch, add a `folium.Marker` to the `MarkerCluster()` object.
    - Create an icon as a text label, assigning the `icon_color` as the `marker_colour` determined previously.

3. Calculate the distances between a launch site to its proximities
    - To explore the proximities of launch sites, calculations of distances between points can be made using the `Lat` and `Long` values.
    - After marking a point using the `Lat` and `Long` values, create a `folium.Marker` object to show the distance.
    - To display the distance line between two points, draw a `folium.PolyLine` and add this to the map.

GitHub Link

- The following plots were added to a Plotly Dash dashboard to have an interactive visualisation of the data:

1. Pie chart (`px.pie()`) showing the total successful launches per site
   - This makes it clear to see which sites are most successful
   - The chart could also be filtered (using a `dcc.Dropdown()` object) to see the success/failure ratio for an individual site

2. Scatter graph (`px.scatter()`) to show the correlation between outcome (success or not) and payload mass (kg)
   - This could be filtered (using a `RangeSlider()` object) by ranges of payload masses
   - It could also be filtered by booster version

GitHub Link

The following steps were taking to develop, evaluate, and find the best performing classification model:

## Model Development

- To prepare the dataset for model development:
  - Load dataset
  - Perform necessary data transformations (standardise and pre-process)
  - Split data into training and test data sets, using `train_test_split()`
  - Decide which type of machine learning algorithms are most appropriate
- For each chosen algorithm:
  - Create a `GridSearchCV` object and a dictionary of parameters
  - Fit the object to the parameters
  - Use the training data set to train the model

## Model Evaluation

- For each chosen algorithm:
  - Using the output GridSearchCV object:
    - Check the tuned hyperparameters (`best_params_`)
    - Check the accuracy (`score` and `best_score_`)
  - Plot and examine the Confusion Matrix

## Finding the Best Classification Model

- Review the accuracy scores for all chosen algorithms
- The model with the highest accuracy score is determined as the best performing model

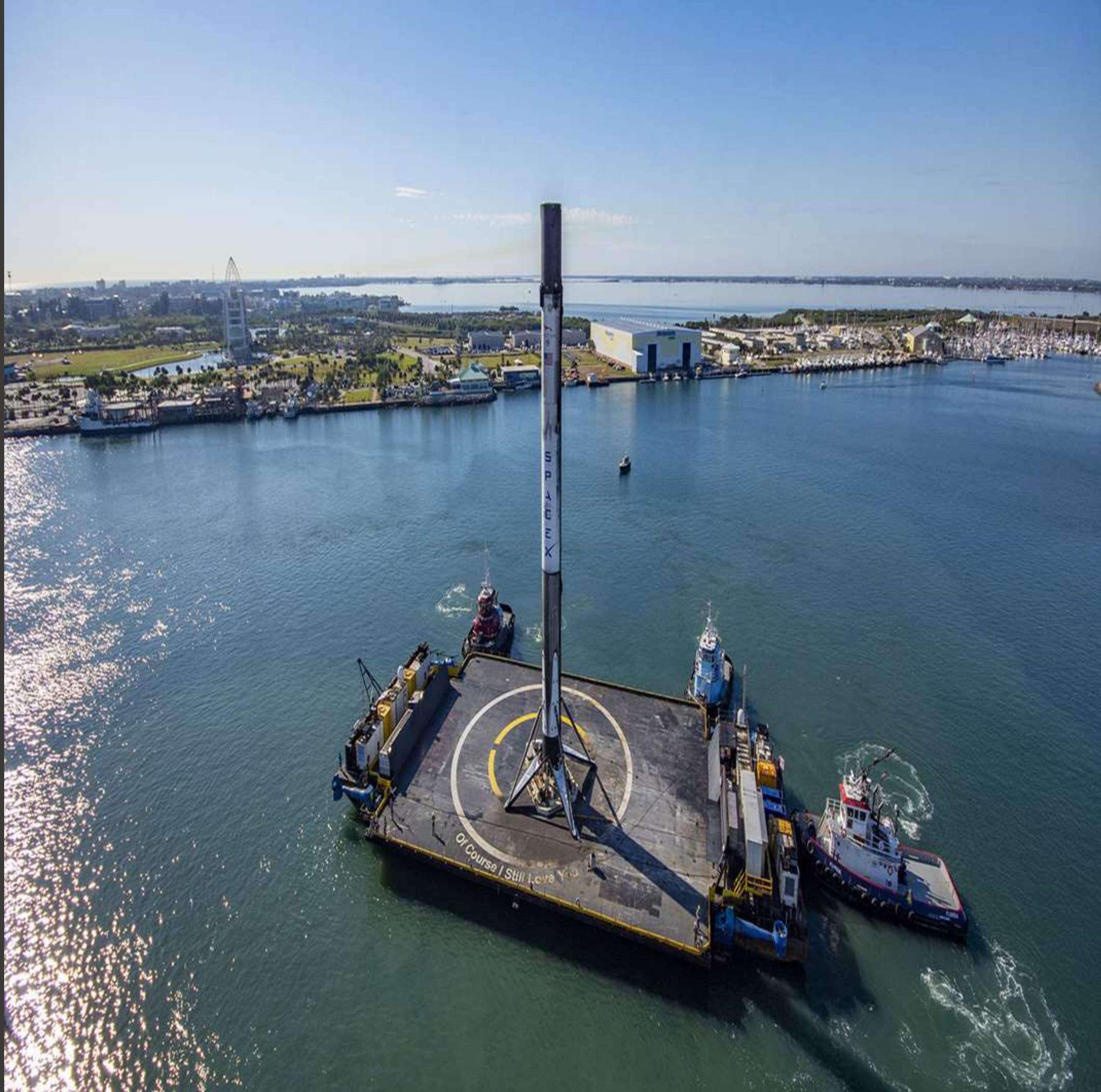
GitHub Link

# results

Exploratory Data Analysis

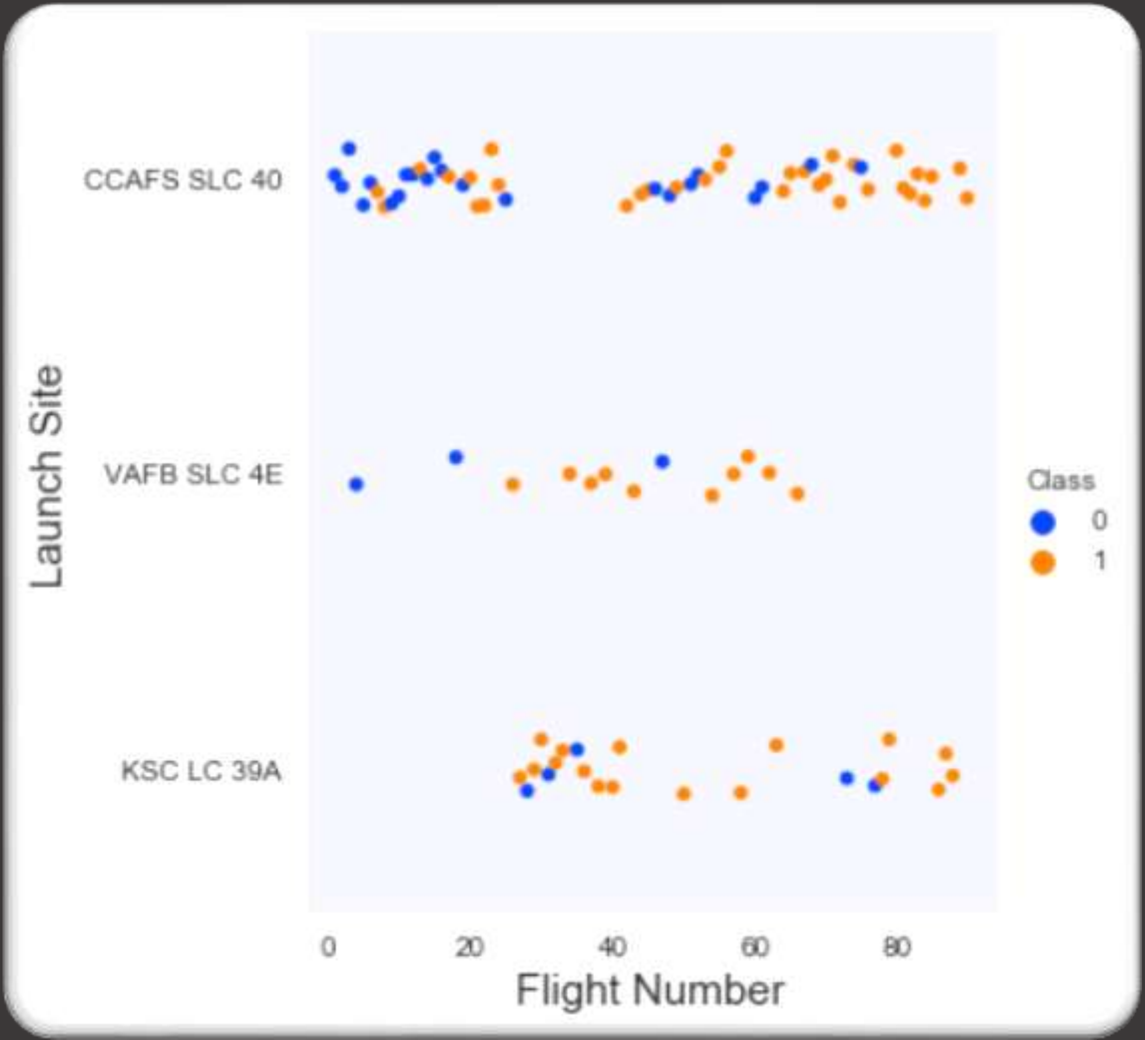
Interactive Analytics

Predictive Analysis

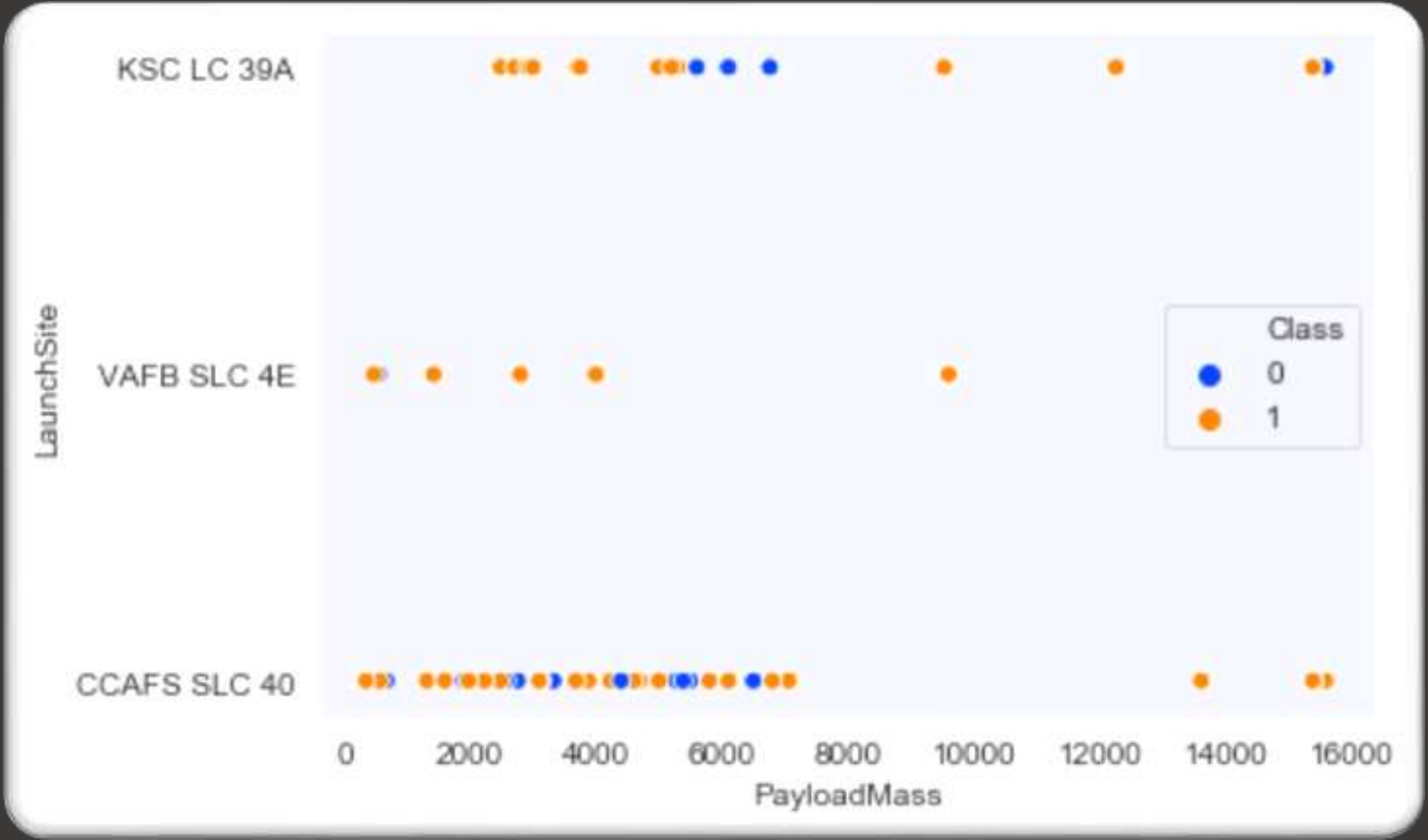


# EDA - WITH VISUALIZATION

- The scatter plot of Launch Site vs. Flight Number shows that:
- As the number of flights increases, the rate of success at a launch site increases.
- Most of the early flights (flight numbers < 30) were launched from CCAFS SLC 40, and were generally unsuccessful.
- The flights from VAFB SLC 4E also show this trend, that earlier flights were less successful.
- No early flights were launched from KSC LC 39A, so the launches from this site are more successful.
- Above a flight number of around 30, there are significantly more successful landings (Class = 1).

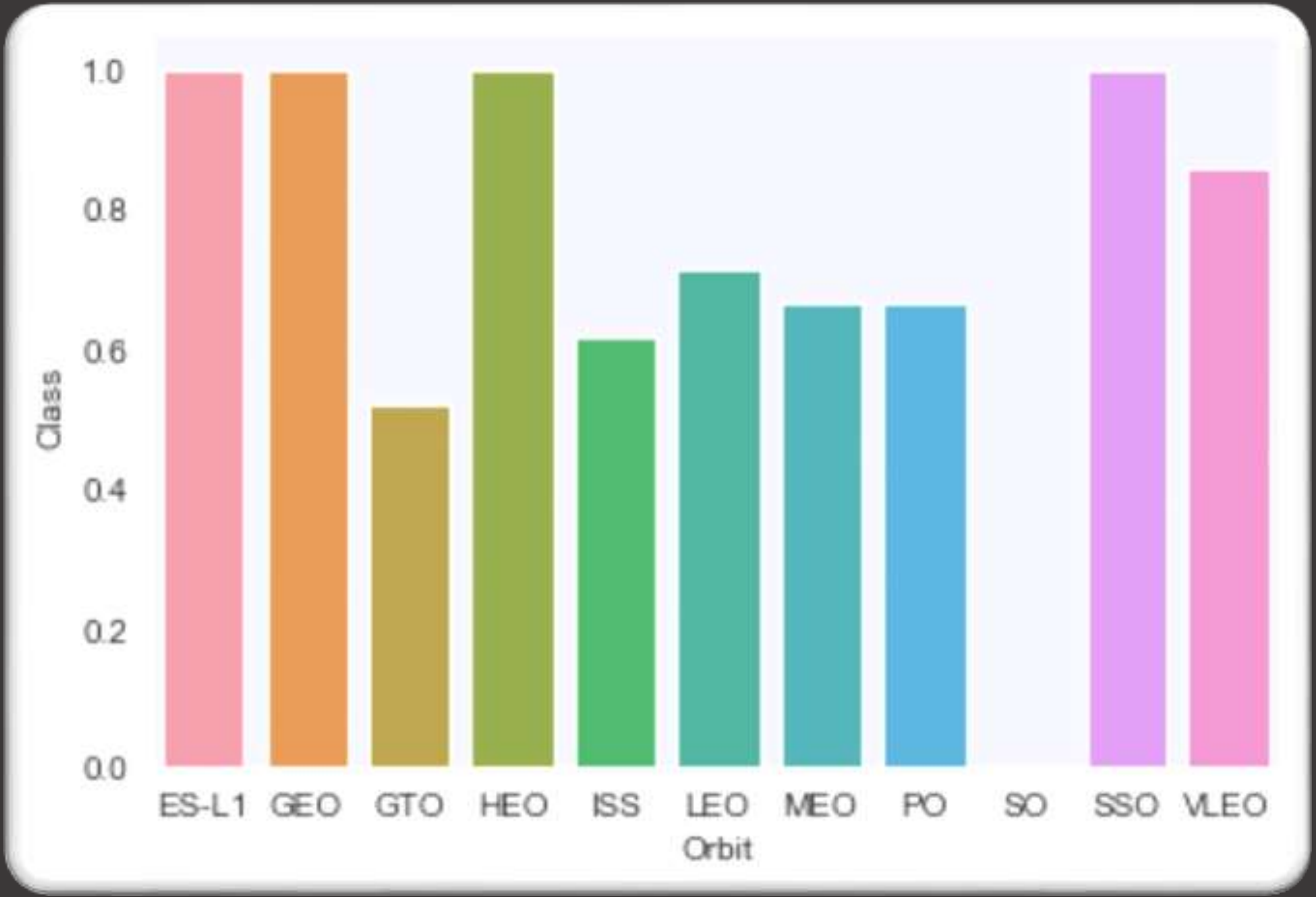


- The scatter plot of Launch Site vs. Payload Mass shows that:
- Above a payload mass of around 7000 kg, there are very few unsuccessful landings, but there is also far less data for these heavier launches.
- There is no clear correlation between payload mass and success rate for a given launch site.
- All sites launched a variety of payload masses, with most of the launches from CCAFS SLC 40 being comparatively lighter payloads (with some outliers).

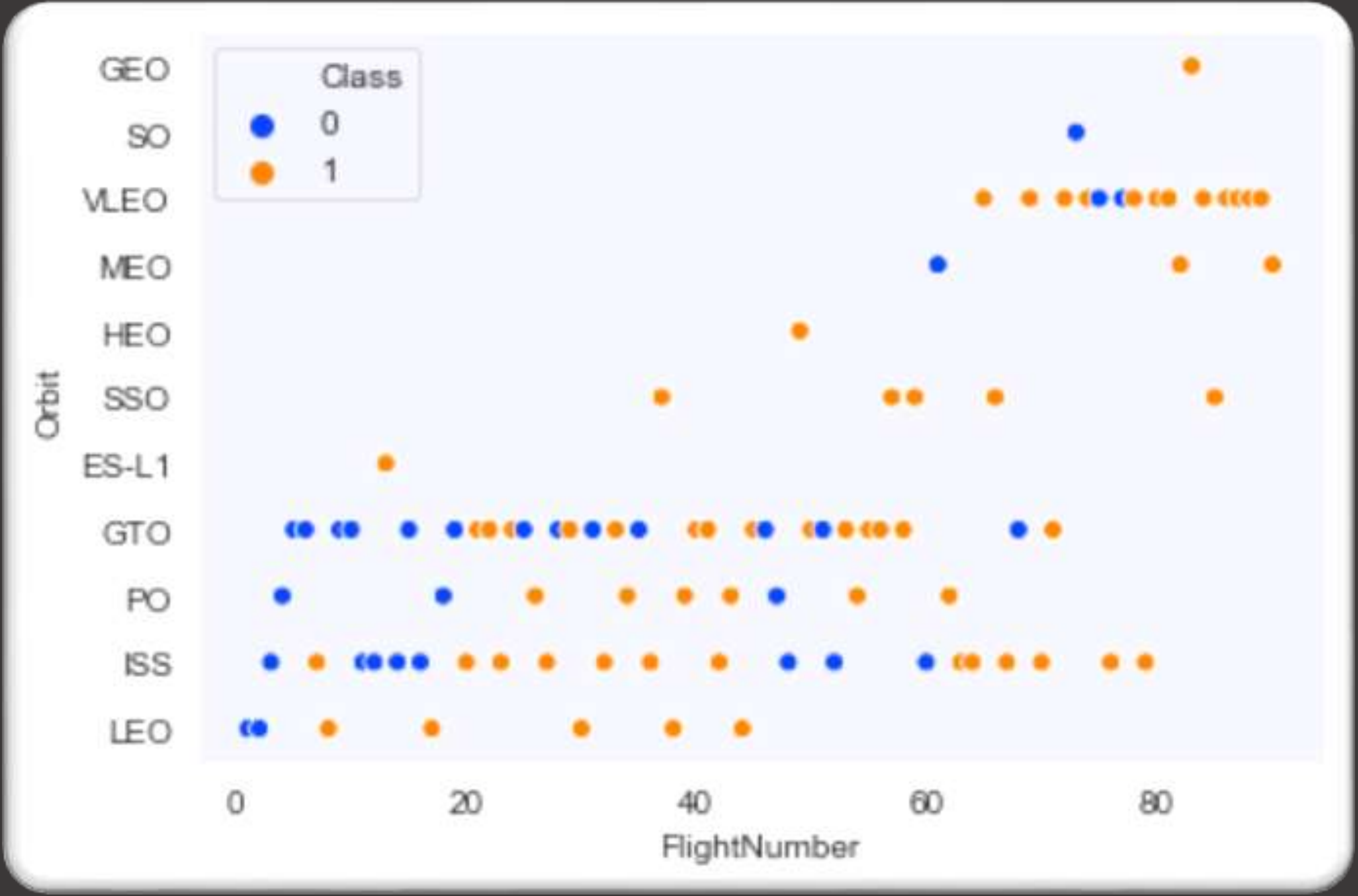


- The bar chart of Success Rate vs. Orbit Type shows that the following orbits have the highest (100%) success rate:
- ES-L1 (Earth-Sun First Lagrangian Point)
- GEO (Geostationary Orbit)
- HEO (High Earth Orbit)
- SSO (Sun-synchronous Orbit)

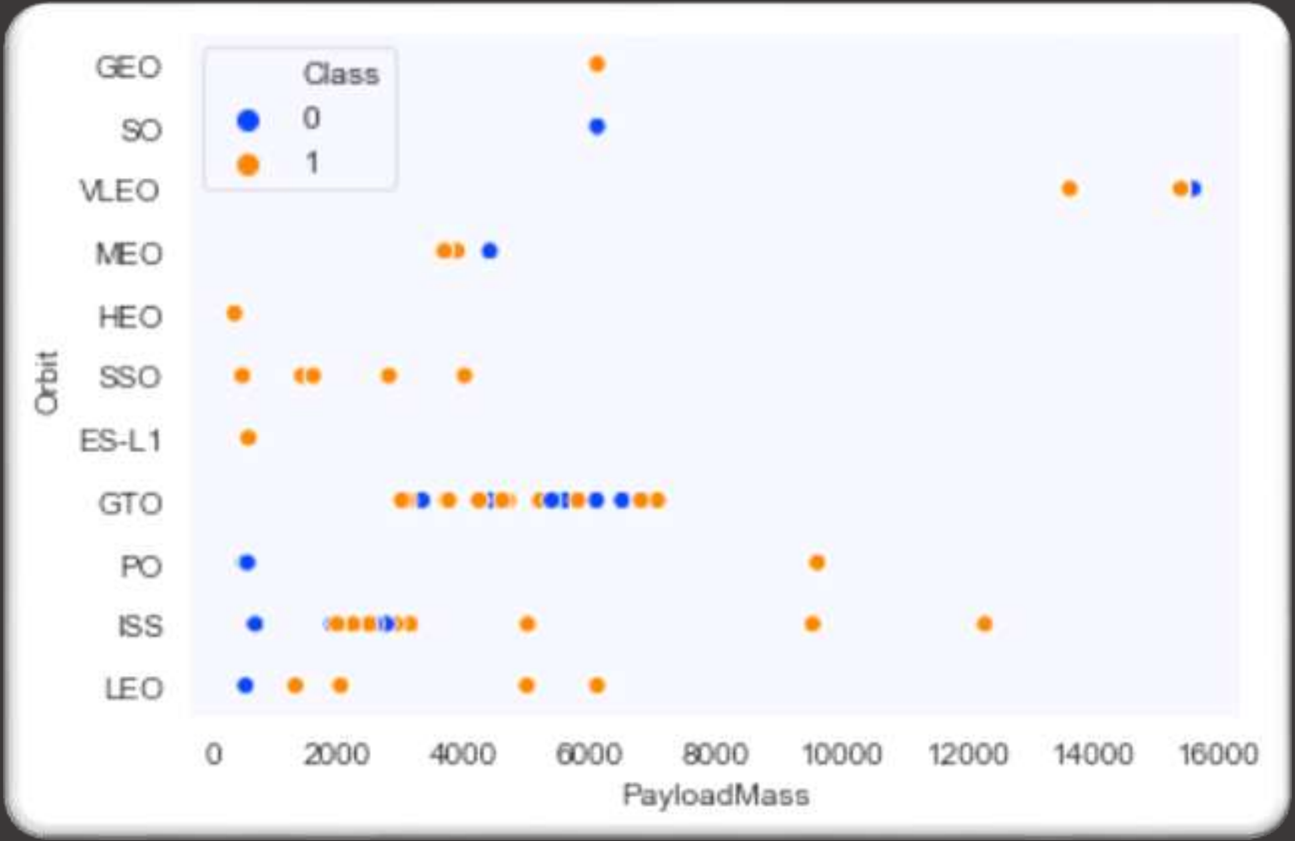
- The orbit with the lowest (0%) success rate is:
- SO (Heliocentric Orbit)



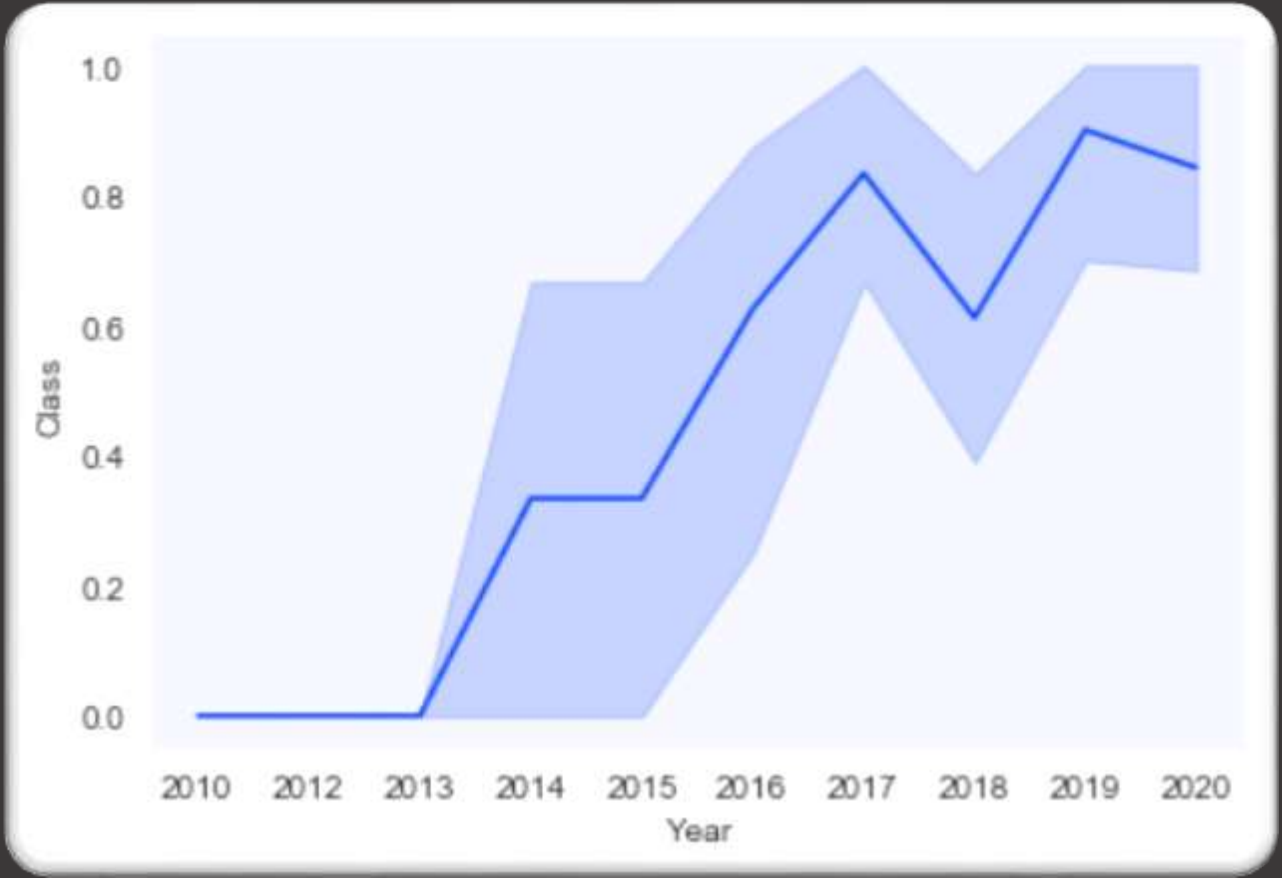
- This scatter plot of Orbit Type vs. Flight number shows a few useful things that the previous plots did not, such as:
- The 100% success rate of GEO, HEO, and ES-L1 orbits can be explained by only having 1 flight into the respective orbits.
- The 100% success rate in SSO is more impressive, with 5 successful flights.
- There is little relationship between Flight Number and Success Rate for GTO.
- Generally, as Flight Number increases, the success rate increases. This is most extreme for LEO, where unsuccessful landings only occurred for the low flight numbers (early launches).



- This scatter plot of Orbit Type vs. Payload Mass shows that:
- The following orbit types have more success with heavy payloads:
  - PO (although the number of data points is small)
  - ISS
  - LEO
- For GTO, the relationship between payload mass and success rate is unclear.
- VLEO (Very Low Earth Orbit) launches are associated with heavier payloads, which makes intuitive sense.



- The line chart of yearly average success rate shows that:
- Between 2010 and 2013, all landings were unsuccessful (as the success rate is 0).
- After 2013, the success rate generally increased, despite small dips in 2018 and 2020.
- After 2016, there was always a greater than 50% chance of success.



# EDA - WITH SQL

- Find the names of the unique launch sites.

```sql
%sql SELECT UNIQUE(LAUNCH_SITE) FROM SPACEXTBL;
```

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

- The word UNIQUE returns only unique values from the LAUNCH_SITE column of the SPACEXTBL table.

- Find 5 records where launch sites begin with ‘CCA’.

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

| launch_site |
|---|
| CCAFS LC-40 |
| CCAFS LC-40 |
| CCAFS LC-40 |
| CCAFS LC-40 |
| CCAFS LC-40 |

- `LIMIT 5` fetches only 5 records, and the `LIKE` keyword is used with the wild card `‘CCA%’` to retrieve string values beginning with ‘CCA’.

- Calculate the total payload carried by boosters from NASA.

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

| total_payload_mass |
| --- |
| 45596 |

- The SUM keyword is used to calculate the total of the LAUNCH column, and the SUM keyword (and the associated condition) filters the results to only boosters from NASA (CRS).

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

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

| average_payload_mass |
|---|
| 2928 |

- The AVG keyword is used to calculate the average of the PAYLOAD_MASS__KG_ column, and the WHERE keyword (and the associated condition) filters the results to only the F9 v1.1 booster version.

- Find the dates of the first successful landing outcome on ground pad.

```
%sql SELECT MIN(DATE) AS FIRST_SUCCESSFUL_GROUND_LANDING FROM SPACEXTBL \
    WHERE LANDING__OUTCOME = 'Success (ground pad)';
```

| first_successful_ground_landing |
|---|
| 2015-12-22 |

- The MIN keyword is used to calculate the minimum of the DATE column, i.e. the first date, and the WHERE keyword (and the associated condition) filters the results to only the successful ground pad landings.

- List the names of boosters which have successfully landed on drone ship and had 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_ BETWEEN 4000 AND 6000);
```

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

- The WHERE keyword is used to filter the results to include only those that satisfy both conditions in the brackets (as the AND keyword is also used). The BETWEEN keyword allows for 4000 < x < 6000 values to be selected.

- Calculate the total number of successful and failure mission outcome.

```
%sql SELECT MISSION_OUTCOME, COUNT(MISSION_OUTCOME) AS TOTAL_NUMBER FROM SPACEXTBL GROUP BY MISSION_OUTCOME;
```

| mission_outcome | total_number |
|---|---|
| Failure (in flight) | 1 |
| Success | 99 |
| Success (payload status unclear) | 1 |

- The COUNT keyword is used to calculate the total number of mission outcomes, and the GROUPBY keyword is also used to group these results by the type of mission outcome.

- List the names of the booster which have carried the maximum payload mass.

```
%sql SELECT DISTINCT(BOOSTER_VERSION) FROM SPACEXTBL \
    WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTBL);
```

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

- A subquery is used here. The SELECT statement within the brackets finds the maximum payload, and this value is used in the WHERE condition. The DISTINCT keyword is then used to retrieve only distinct /unique booster versions.

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

```sql
%sql SELECT BOOSTER_VERSION, LAUNCH_SITE FROM SPACEXTBL \
    WHERE (LANDING__OUTCOME = 'Failure (drone ship)') AND (EXTRACT(YEAR FROM DATE) = '2015');
```

| booster_version | launch_site |
|---|---|
| F9 v1.1 B1012 | CCAFS LC-40 |
| F9 v1.1 B1015 | CCAFS LC-40 |

- The WHERE keyword is used to filter the results for only failed landing outcomes, AND only for the year of 2015.

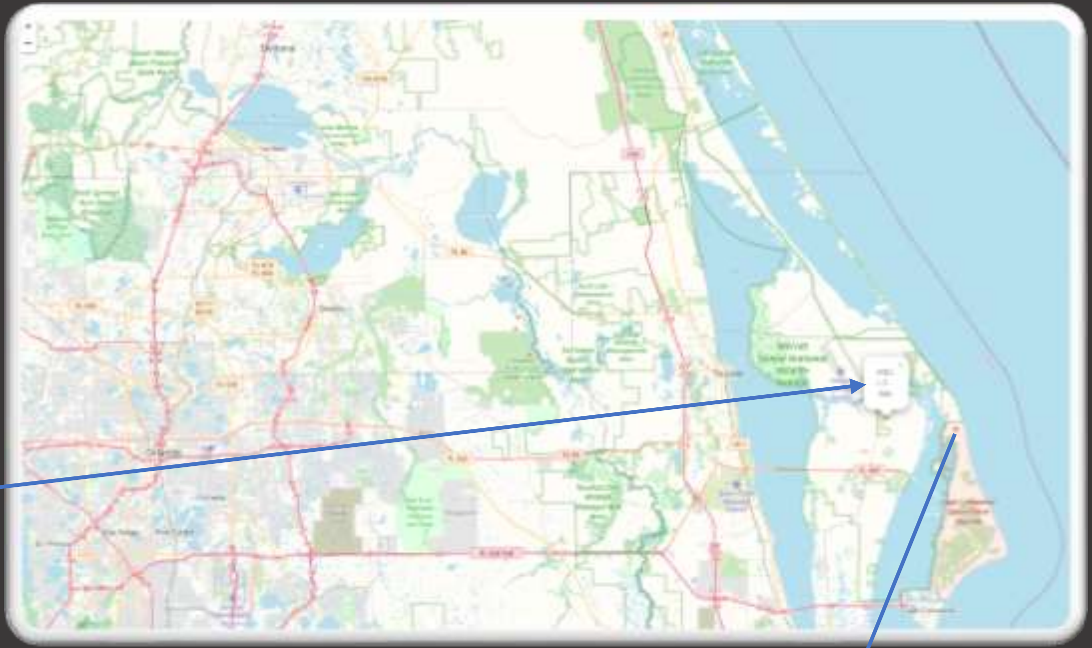
- Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order.

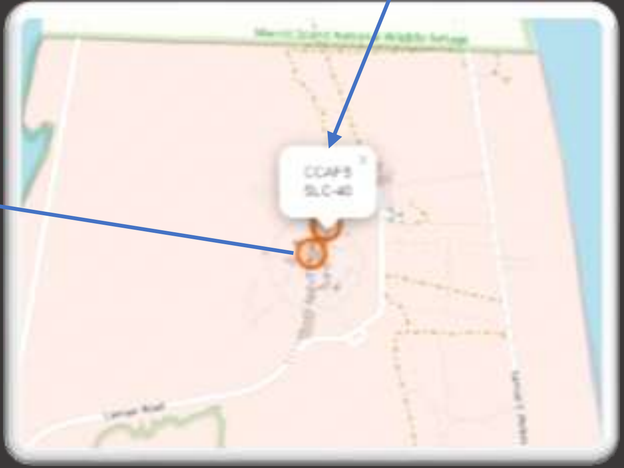
```
%sql SELECT LANDING__OUTCOME, COUNT(LANDING__OUTCOME) AS TOTAL_NUMBER FROM SPACEXTBL \
    WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20' \
    GROUP BY LANDING__OUTCOME \
    ORDER BY TOTAL_NUMBER DESC;
```

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

- The WHERE keyword is used with the BETWEEN keyword to filter the results to dates only within those specified. The results are then grouped and ordered, using the keywords GROUP BY and ORDER BY, respectively, where DESC is used to specify the descending order.

# LAUNCH SITES PROXIMITY ANALYSIS – FOLIUM INTERACTIVE MAP



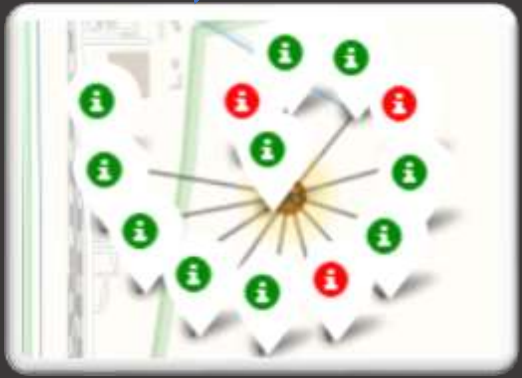



- All SpaceX launch sites are on coasts of the United States of America, specifically Florida and California.

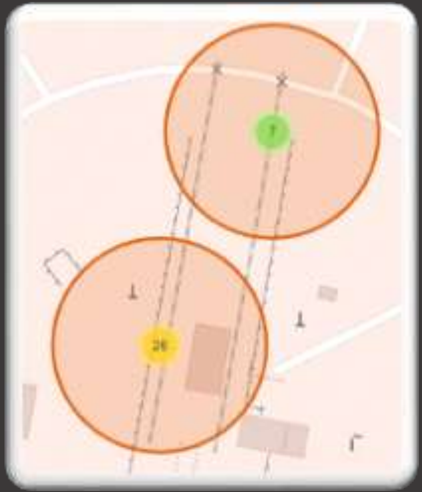
- Launches have been grouped into clusters, and annotated with green icons for successful launches, and red icons for failed launches.

VAFB SLC-4E

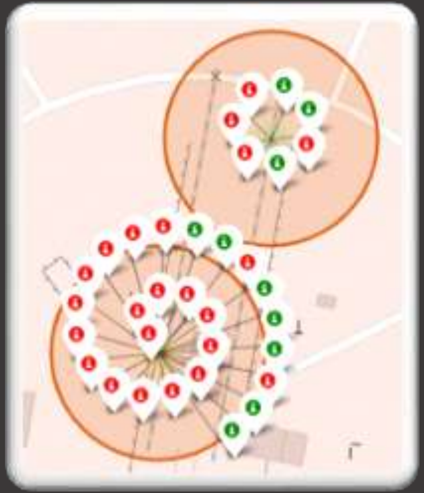
KSC LC-39A



CCAFS SLC-40 and CCAFS LC-40

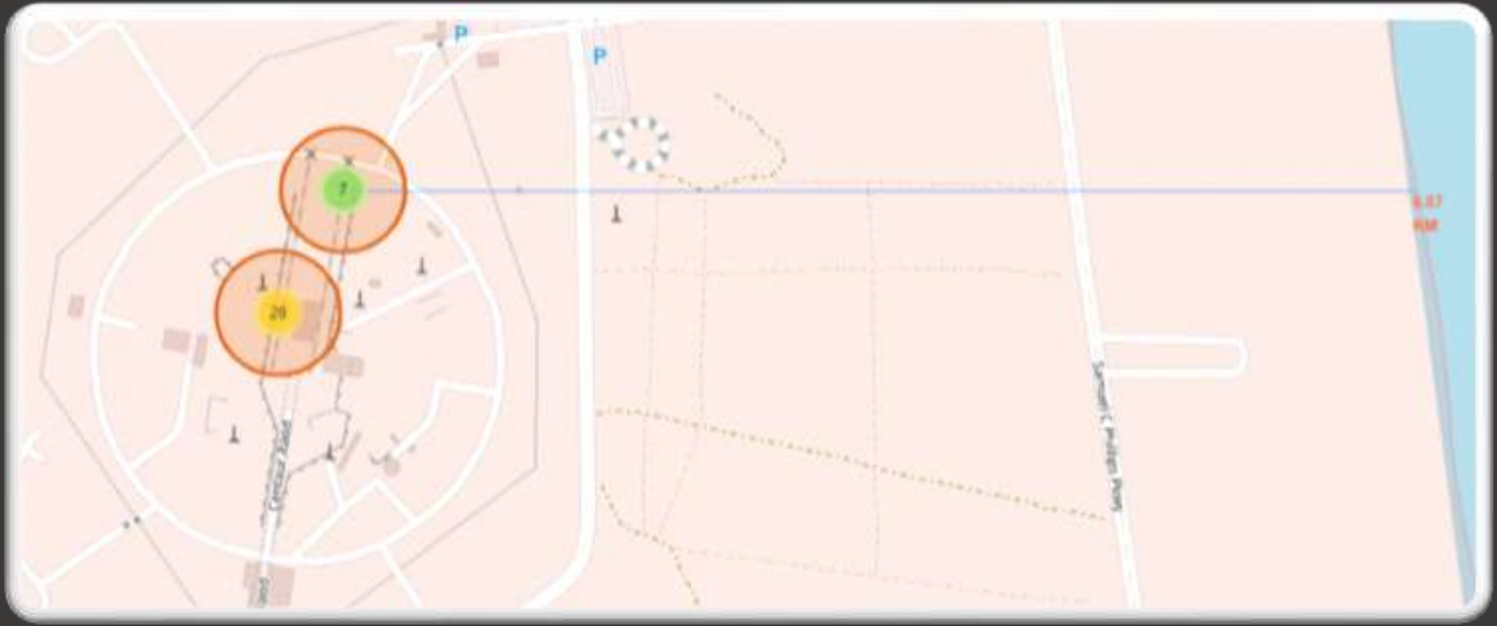


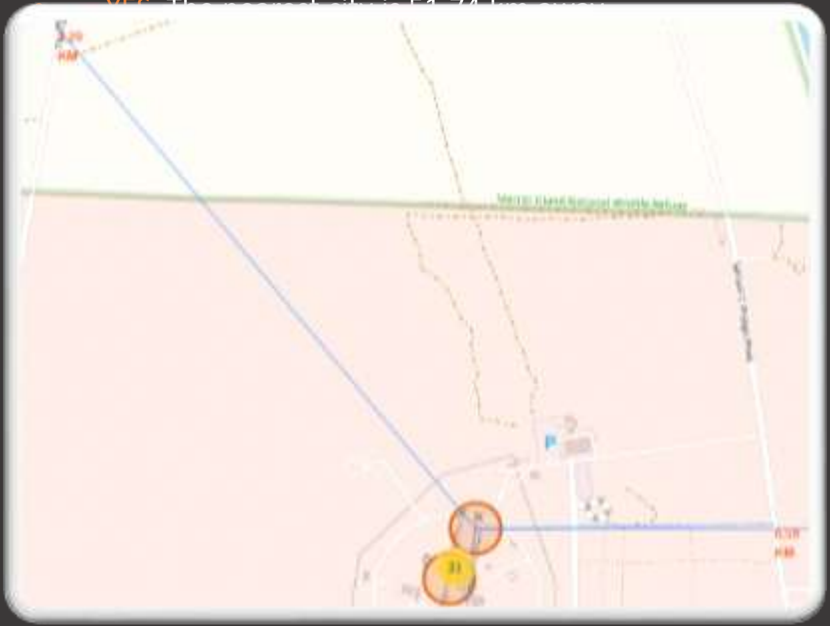
=



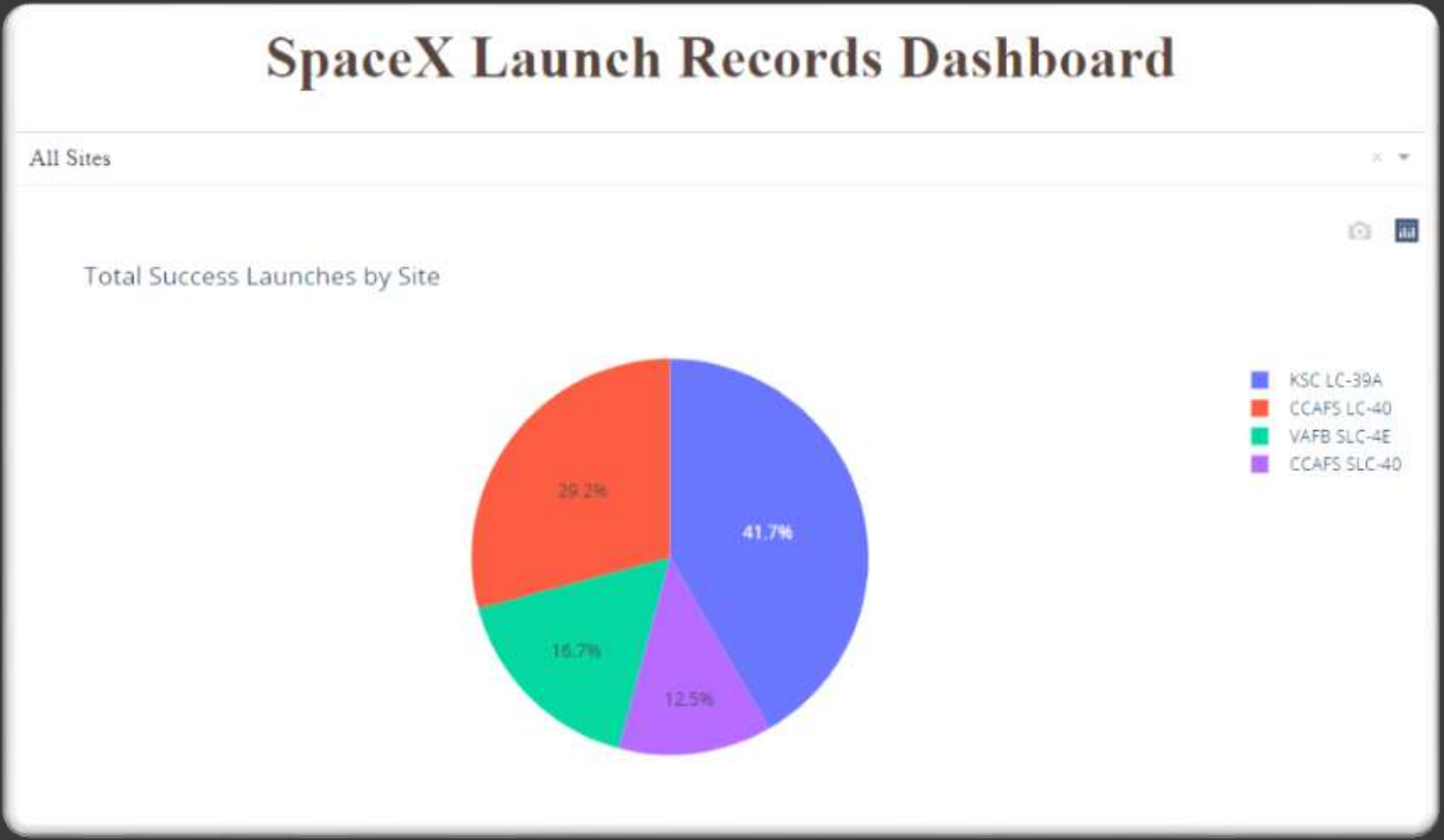
Using the CCAFS SLC-40 launch site as an example site, we can understand more about the placement of launch sites.


- Are launch sites in close proximity to railways?
- YES. The coastline is only 0.87 km due East.
- Are launch sites in close proximity to highways?
- YES. The nearest highway is only 0.59km away.
- Are launch sites in close proximity to railways?
- YES. The nearest railway is only 1.29 km away.
- Do launch sites keep certain distance away from cities?
- YES. The nearest city is 51.74 km away.

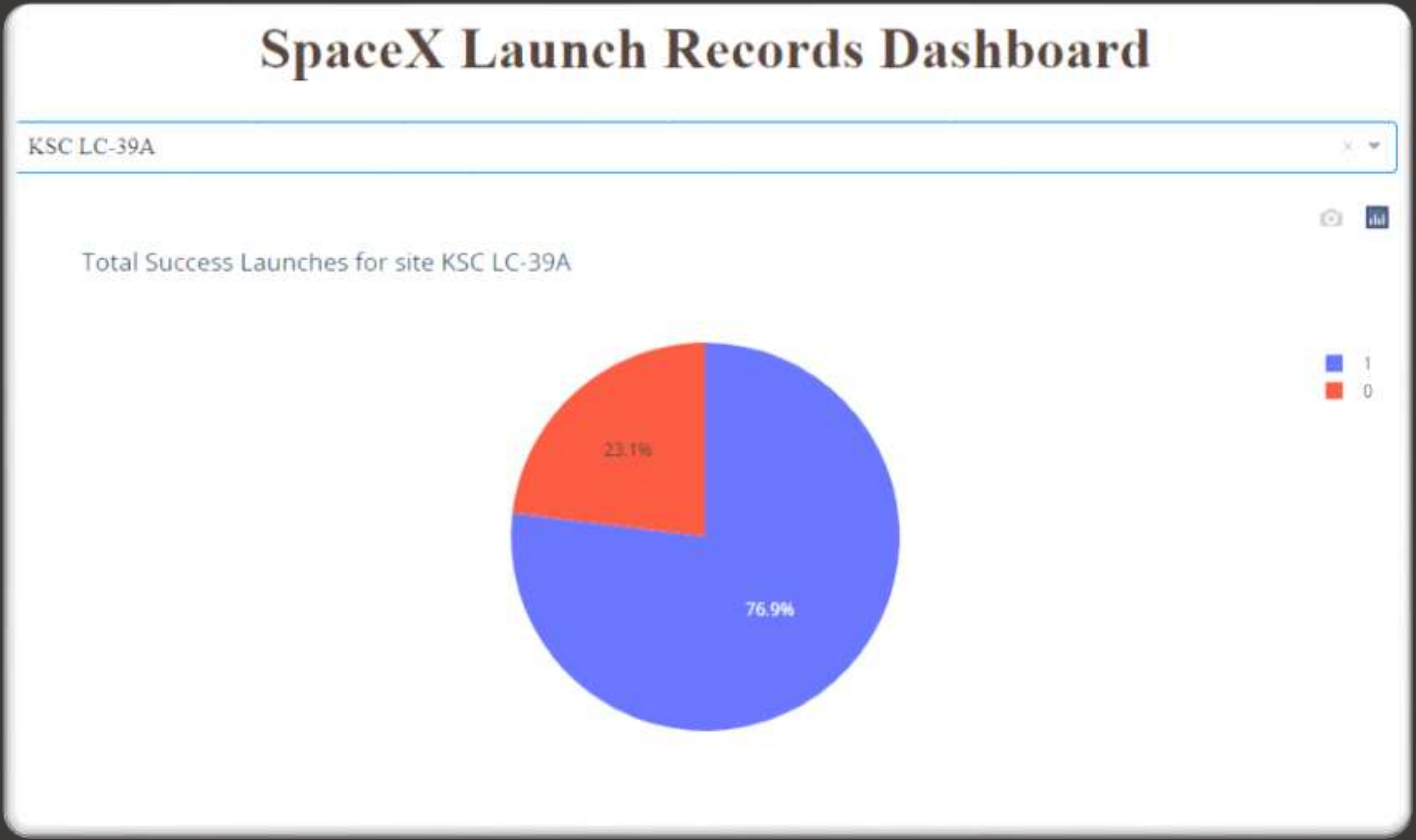




# interactive dashboard  - **Plotly Dash**




- The launch site KSC LC-39 A had the most successful launches, with 41.7% of the total successful launches.




The launch site KSC LC-39 A also had the highest rate of successful launches, with a 76.9% success rate.

Note: $class \begin{cases} 0, Failure \\ 1, Success \end{cases}$


1


2

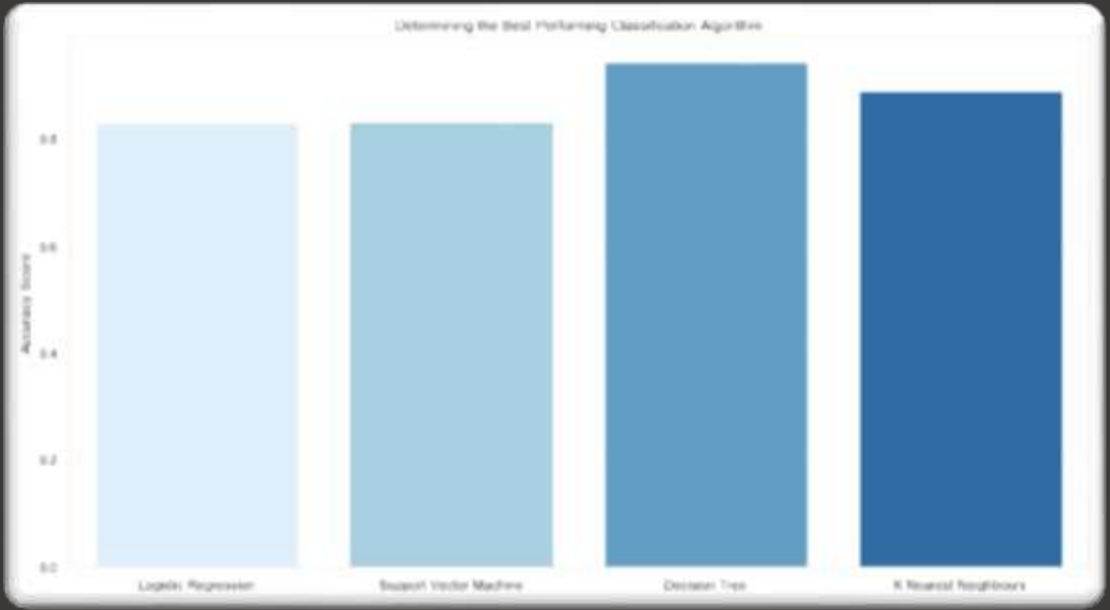

- Plotting the launch outcome vs. payload for all sites shows a gap around 4000 kg, so it makes sense to split the data into 2 ranges:
  - 0 – 4000 kg (low payloads)
  - 4000 – 10000 kg (massive payloads)
- From these 2 plots, it can be shown that the success for massive payloads is lower than that for low payloads.
- It is also worth noting that some booster types (v1.0 and B5) have not been launched with massive payloads.

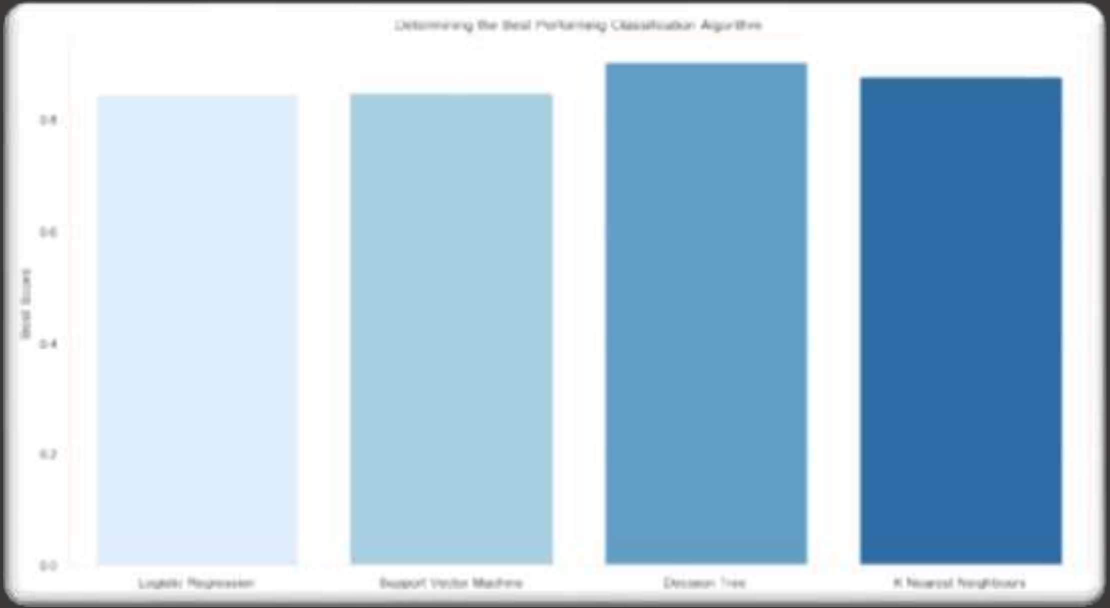
Note: $class \begin{cases} 0, Failure \\ 1, Success \end{cases}$

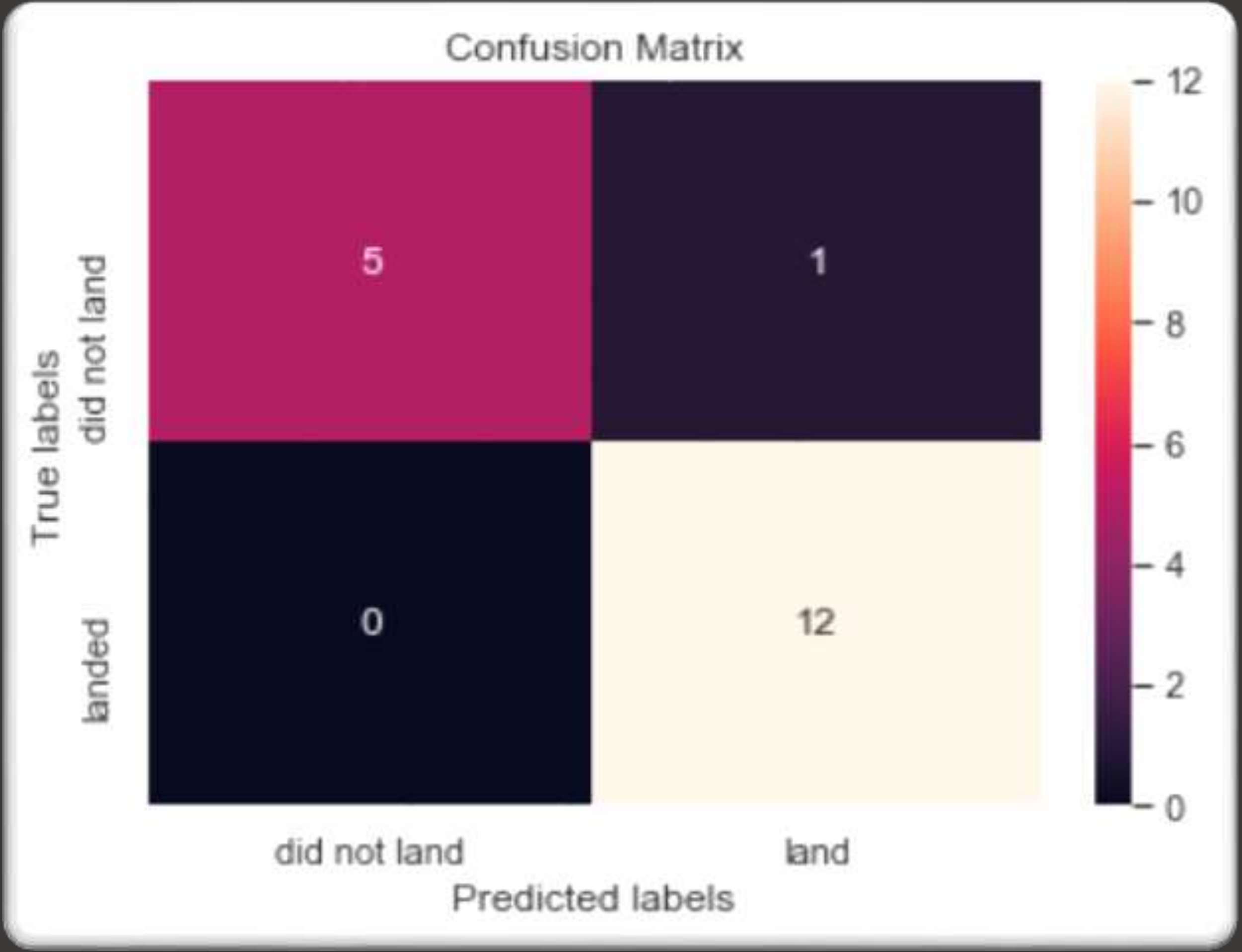
# PREDICTIVE ANALYSIS - CLASSIFICATION

- Plotting the Accuracy Score and Best Score for each classification algorithm produces the following result:
- The Decision Tree model has the highest classification accuracy

| Algorithm | Accuracy Score | Best Score |
|---|---|---|
| Logistic Regression | 0.833333 | 0.846429 |
| Support Vector Machine | 0.833333 | 0.848214 |
| Decision Tree | 0.944444 | 0.903571 |
| K Nearest Neighbours | 0.888889 | 0.876786 |





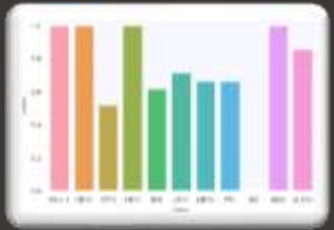



- As shown previously, best performing classification model is the Decision Tree model, with an accuracy of 94.44%.
- This is explained by the confusion matrix, which shows only 1 out of 18 total results classified incorrectly (a false positive, shown in the top-right corner).
- The other 17 results are correctly classified (5 did not land, 12 did land).

# CONCLUSIONS

- As the number of flights increases, the rate of success at a launch site increases, with most early flights being unsuccessful. I.e. with more experience, the success rate increases.
  - Between 2010 and 2013, all landings were unsuccessful (as the success rate is 0).
  - After 2013, the success rate generally increased, despite small dips in 2018 and 2020.
  - After 2016, there was always a greater than 50% chance of success.

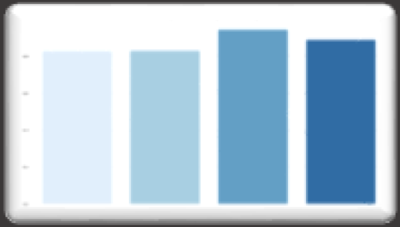
- Orbit types ES-L1, GEO, HEO, and SSO, have the highest (100%) success rate.
  - The 100% success rate of GEO, HEO, and ES-L1 orbits can be explained by only having 1 flight into the respective orbits.
  - The 100% success rate in SSO is more impressive, with 5 successful flights.
  - The orbit types PO, ISS, and LEO, have more success with heavy payloads:
  - VLEO (Very Low Earth Orbit) launches are associated with heavier payloads, which makes intuitive sense.



- The launch site KSC LC-39 A had the most successful launches, with 41.7% of the total successful launches, and also the highest rate of successful launches, with a 76.9% success rate.

- The success for massive payloads (over 4000kg) is lower than that for low payloads.

- The best performing classification model is the Decision Tree model, with an accuracy of 94.44%.



# APPENDIX

- Custom functions to retrieve the required information
- Custom logic to clean the data

```python
# Lets take a subset of our dataframe keeping only the features we want and the flight number, and date_utc.
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]

# We will remove rows with multiple cores because those are falcon rockets with 2 extra rocket boosters
# and rows that have multiple payloads in a single rocket.
data = data[data['cores'].map(len)==1]
data = data[data['payloads'].map(len)==1]

# Since payloads and cores are lists of size 1 we will also extract the single value in the list and replace the feature.
data['cores'] = data['cores'].map(lambda x : x[0])
data['payloads'] = data['payloads'].map(lambda x : x[0])

# We also want to convert the date_utc to a datetime datatype and then extracting the date leaving the time
data['date'] = pd.to_datetime(data['date_utc']).dt.date

# Using the date we will restrict the dates of the launches
data = data[data['date'] <= datetime.date(2020, 11, 13)]
```

Python

From the `rocket` column we would like to learn the booster name.

```python
# Takes the dataset and uses the rocket column to call the API and append the data to the list
def getBoosterVersion(data):
    for x in data['rocket']:
        response = requests.get("https://api.spacexdata.com/v4/rockets/"+str(x)).json()
        BoosterVersion.append(response['name'])
```

Python

From the `launchpad` we would like to know the name of the launch site being used, the longitude, and the latitude.

```python
# Takes the dataset and uses the launchpad column to call the API and append the data to the list
def getLaunchSite(data):
    for x in data['launchpad']:
        response = requests.get("https://api.spacexdata.com/v4/launchpads/"+str(x)).json()
        Longitude.append(response['longitude'])
        Latitude.append(response['latitude'])
        LaunchSite.append(response['name'])
```

Python

From the `payload` we would like to learn the mass of the payload and the orbit that it is going to.

```python
# Takes the dataset and uses the payloads column to call the API and append the data to the lists
def getPayloadData(data):
    for load in data['payloads']:
        response = requests.get("https://api.spacexdata.com/v4/payloads/"+load).json()
        PayloadMass.append(response['mass_kg'])
        Orbit.append(response['orbit'])
```

Python

From `cores` we would like to learn the outcome of the landing, the type of the landing, number of flights with that core, whether gridfins were used, whether the core is reused, whether legs were used, the landing pad used, the block of the core (which is a number used to separate versions of cores), the number of times this specific core has been reused, and the serial of the core.

```python
# Takes the dataset and uses the cores column to call the API and append the data to the lists
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cores/"+core['core']).json()
                Block.append(response['block'])
                ReusedCount.append(response['reuse_count'])
                Serial.append(response['serial'])
            else:
                Block.append(None)
                ReusedCount.append(None)
                Serial.append(None)
            Outcome.append(str(core['landing_success'])+' '+str(core['landing_type']))
            Flights.append(core['flight'])
            GridFins.append(core['gridfins'])
            Reused.append(core['reused'])
            Legs.append(core['legs'])
            LandingPad.append(core['landpad'])
```

Python

- Custom functions for web scraping
- Custom logic to fill up the launch_dict values with values from the launch tables

```python
def date_time(table_cells):
    """
    This function returns the data and time from the HTML  table cell
    Input: the  element of a table data cell extracts extra row
    """
    return [data_time.strip() for data_time in list(table_cells.strings)][0:2]

def booster_version(table_cells):
    """
    This function returns the booster version from the HTML  table cell
    Input: the  element of a table data cell extracts extra row
    """
    out=''.join([booster_version for i,booster_version in enumerate( table_cells.strings) if i%2==0][0:-1])
    return out

def landing_status(table_cells):
    """
    This function returns the landing status from the HTML table cell
    Input: the  element of a table data cell extracts extra row
    """
    out=[i for i in table_cells.strings][0]
    return out


def get_mass(table_cells):
    mass=unicodedata.normalize("NFKD", table_cells.text).strip()
    if mass:
        mass.find("kg")
        new_mass=mass[0:mass.find("kg")+2]
    else:
        new_mass=0
    return new_mass


def extract_column_from_header(row):
    """
    This function returns the landing status from the HTML table cell
    Input: the  element of a table data cell extracts extra row
    """
    if (row.br):
        row.br.extract()
    if row.a:
        row.a.extract()
    if row.sup:
        row.sup.extract()

    column_name = ' '.join(row.contents)

    # Filter the digit and empty names
    if not(column_name.strip().isdigit()):
        column_name = column_name.strip()
        return column_name
```

```python
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
            # Append the flight_number into launch_dict with key `Flight No.`
            launch_dict["Flight No."].append(flight_number)

            # Date value
            #Append the date into launch_dict with key `Date`
            datatimelist=date_time(row[0])
            date = datatimelist[0].strip(',')
            launch_dict["Date"].append(date)

            # Time value
            #Append the time into launch_dict with key `Time`
            time = datatimelist[1]
            launch_dict["Time"].append(time)

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

            # Launch Site
            #Append the bv into launch_dict with key `Launch Site`
            launch_site = row[2].a.string
            launch_dict['Launch site'].append(launch_site)

            # Payload
            #Append the payload into launch_dict with key `Payload`
            payload = row[3].a.string
            launch_dict['Payload'].append(payload)

            # Payload Mass
            #Append the payload_mass into launch_dict with key `Payload mass`
            payload_mass = get_mass(row[4])
            launch_dict['Payload mass'].append(payload_mass)

            # Orbit
            #Append the orbit into launch_dict with key `Orbit`
            orbit = row[5].a.string
            launch_dict['Orbit'].append(orbit)

            # Customer
            #Append the customer into launch_dict with key `Customer`
            if row[6].a != None:
                customer = row[6].a.string
            else:
                customer = 'None'

            launch_dict['Customer'].append(customer)

            # Launch outcome
            #Append the launch_outcome into launch_dict with key `Launch outcome`
            launch_outcome = list(row[7].strings)[0]
            launch_dict['Launch outcome'].append(launch_outcome)

            # Booster landing
            #Append the booster landing into launch_dict with key `Booster landing`
            booster_landing = landing_status(row[8])
            launch_dict['Booster landing'].append(booster_landing)


            print(f"Flight Number: {flight_number}, Date: {date}, Time: {time} \n \
            Booster Version {bv}, Launch Site: {launch_site} \n \
            Payload: {payload}, Orbit: {orbit} \n \
            Customer: {customer}, Launch Outcome: {launch_outcome}\n \
            Booster Landing: {booster_landing} \n \
            *** ")
```