IBM Developer
SKILLS NETWORK

# Winning Space Race with Data Science

Gigih Pambuko
10/08/2023

# Outline

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

# Executive Summary

- Summary of methodologies
  - Data Collection via API, Web Scraping
  - Exploratory Data Analysis (EDA) with Data Visualization
  - EDA with SQL
  - Interactive Map with Folium
  - Dashboards with Plotly Dash
  - Predictive Analysis
- Summary of all results
  - Exploratory Data Analysis results
  - Interactive maps and dashboard
  - Predictive results

# Introduction

- Project background and context

The aim of this project is to predict if the Falcon 9 first stage will successfully land. SpaceX says on its website that the Falcon 9 rocket launch cost 62 million dollars. Other providers cost upward of 165 million dollars each. The price difference is explained by the fact that SpaceX can reuse the first stage. By determining if the stage will land, we can determine the cost of a launch. This information is interesting for another company if it wants to compete with SpaceX for a rocket launch.

- Problems you want to find answers

1. What are the main characteristics of a successful or failed landing ?
2. What are the effects of each relationship of the rocket variables on the success or failure of a landing ?
3. What are the conditions which will allow SpaceX to achieve the best landing success rate ?

Section 1

# Methodology

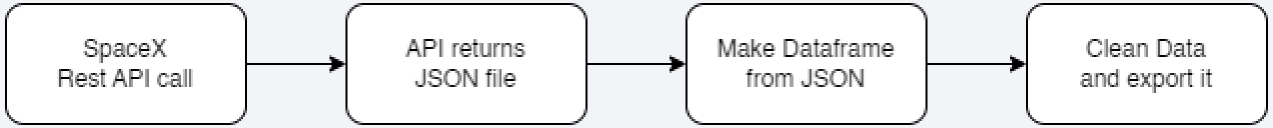
# Methodology

## Executive Summary

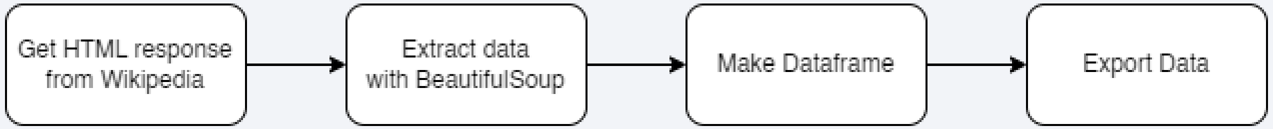
- Data collection methodology:
  - SpaceX REST API
  - Web Scrapping from Wikipedia
- Perform data wrangling
  - Dropping unnecessary columns
  - One Hot Encoding for classification models
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - How to build, tune, evaluate classification models

# Data Collection

- Datasets are collected from Rest SpaceX API and webscrapping Wikipedia
  - The information obtained by the API are rocket, launches, payload information.
    - The space X REST API URL is https://api.spacexdata.com/v4/rockets/




  - The information obtained by the webscrapping of Wikipedia are launces, landing, payload information.
    - URL is https://en.wikipedia.org/wiki/List_of_Falcon_9_and_Falcon_Heavy_launches



# Data Collection – SpaceX API

## 1. Getting Response from API

```
[10]: static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_call_spacex_api.json'
```

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

```
[11]: response.status_code
[11]: 200
```

## 2. Convert Response to JSON File

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

## 3. Transform data

```
[17]: # Call getBoosterVersion
      getBoosterVersion(data)
```

the list has now been update

```
[18]: BoosterVersion[0:5]
[18]: ['Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 9']
```

we can apply the rest of the functions here:

```
[19]: # Call getLaunchSite
      getLaunchSite(data)

[20]: # Call getPayloadData
      getPayloadData(data)

[21]: # Call getCoreData
      getCoreData(data)
```

Link to code

# Data Collection – SpaceX API

## 4. Create dictionary with data

```
[22]: 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}
```

## 5. Create dataframe

```
[23]: # Create a data from launch_dict
      data = pd.DataFrame.from_dict(launch_dict)
```

## 6. Filter dataframe

```
[25]: # Hint data['BoosterVersion']!='Falcon 1'
      data_falcon9 = data[data['BoosterVersion'] != 'Falcon 1']
```

## 7. Export to file

```
[29]: data_falcon9.to_csv('dataset_part_1.csv', index=False)
```

[Link to code]

# Data Collection - Scraping

1. Getting Response from HTML

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

2. Create BeautifulSoup Object

```
[6]: # Use BeautifulSoup() to create a BeautifulSoup object from a response text content
     soup = BeautifulSoup(data, 'html.parser')
```

3. Find all tables

```
# Find all elements of type 'table' in the HTML
html_tables = soup.find_all('table')
```

4. Get column names

```
[10]: column_names = []
      # Apply find_all() function with `th` element on 
      # Iterate each th element and apply the provided
      # Append the Non-empty column name (`if name is n
      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)
```

Link to code

# Data Collection - Scraping

## 5. Create dictionary

```
[12]: launch_dict= dict.fromkeys(column_names)

# Remove an irrelvant 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']=[]
```

## 6. Add data to keys

```
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table
   # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as numbe
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
```

## 7. Create dataframe from dictionary & Export to file

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

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

Link to code

# Data Wrangling

- In the dataset, there are several cases where the booster did not land successully.
  - True Ocean, True RTLS, True ASDS means the mission has been successful.
  - False Ocean, False RTLS, False ASDS means the mission was a failure.
- We need to transform string variables into categorical variables where 1 means the mission has been successful and 0 means the mission was a failure.

Link to code

# Data Wrangling

1. Calculate launches number for each site

```
[8]: # Apply value_counts() on column LaunchSite
     launches = df['LaunchSite'].value_counts()
     launches

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

2. Calculate the number and occurrence of each orbit

```
[9]: # Apply value_counts on Orbit column
     occurence = df['Orbit'].value_counts()
     occurence

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

3. Calculate number and occurrence of mission outcome per orbit type

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

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

4. Create landing outcome label from Outcome column

```
[13]: # landing_class = 0 if bad_outcome
      # landing_class = 1 otherwise
      df['Class'] = df['Outcome'].apply(lambda x: 0 if x in bad_outcomes else 1)
      df['Class'].value_counts()

[13]: 1    60
      0    30
      Name: Class, dtype: int64
```

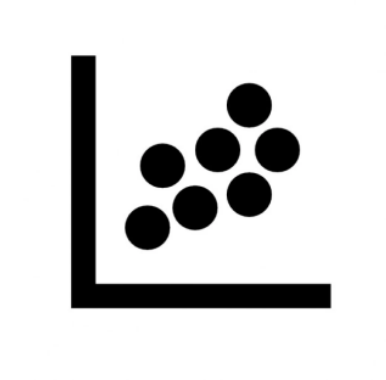
5. Export file

```
[17]: df.to_csv("dataset_part_2.csv", index=False)
```

Link to code

# EDA with Data Visualization

### Scatter Graphs



- Flight Number vs. Payload Mass
- Flight Number vs. Launch Site
- Payload vs. Launch Site
- Orbit vs. Flight Number
- Payload vs. Orbit Type
- Orbit vs. Payload Mass

*Scatter plots show relationship between variables.*

*This relationship is called the correlation.*

### Bar Graph

- Success Rate vs Orbit

*Bar graphs show the relationship between numeric and categorical variables.*

### Line Graph

- Success Rate vs Year

*Line graphs show data variables and their trends. Line graphs can help to show global behavior and make prediction for unseen data.*

Link to code

# EDA with SQL

We performed SQL queries to gather and understand data from dataset:

- Displaying the names of the unique lauunch sites in the space mission.
- Display 5 records where launch sites begin with the string 'CCA'
- Display the total payload mass carried by boosters launched by NASA (CRS).
- Display average payload mass carried by booster version F9 v1.1.
- List the date when the first successful landing outcome in ground pad was achieved.
- List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000.
- List the total number of successful and failure mission outcomes.
- List the names of the booster_versions which have carried the maximum payload mass.
- List the records which will display the month names, faiilure landing_ouutcomes in drone ship, booster versions, launch_site forthe months in year 2015.
- Rank the count of successful landiing_outcomes between the date04-06-2010 and 20-03-2017in descending order.

[Link to code]

# Build an Interactive Map with Folium

Folium map object is a map centered on NASA Johnson Space Center at Houson, Texas

- Red circle at NASA Johnson Space Center's coordinate with label showing its name(*folium.Circle, folium.map.Marker*).
- Red circles at each launch site coordinates with label showing launch site name (*folium.Circle, folium.map.Marker, folium.features.DivIcon*).
- The grouping of points in a cluster to display multiple and different information for the same coordinates (*folium.plugins.MarkerCluster*).
- Markers to show successful and unsuccessful landings. Green for successful landing and Red for unsuccessful landing. (*folium.map.Marker, folium.Icon*).
- Markers to show distance between launch site to key locations (railway, highway, coastway, city) and plot a line between them. (*folium.map.Marker, folium.PolyLine, folium.features.DivIcon)*

These objects are created in order to understand better the problem and the data. We can show easily all launch sites, their surroundings and the number of successful and unsuccessful landings.

[Link to code]

# Build a Dashboard with Plotly Dash

Dashboard has dropdown,pie chart,rangeslider andscatter plot components

- Dropdownallows a user to choose the launch site or all launch sites (*dash_core_components.Dropdown*).
- Pie chart shows the total success and the total failure for the launch site chosen with the dropdown component(*plotly.express.pie*).
- Rangeslider allows a user to select a payload mass in a fixed range (*dash_core_components.RangeSlider*).
- Scatter chart showsthe relationship between two variables, in particular Success vs Payload Mass (*plotly.express.scatter*).

Link to code

# Predictive Analysis (Classification)

Data Preparation

- Load dataset
- Normalize data
- Split data into training and test sets.

Model Preparation

- Selection of machine learning algorithms
- Set parameters for each algorithm to GridSearchCV
- Training GridSearchModel models with training dataset

Model Evaluation

- Get best hyperparameters for each type of model
- Compute accuracy for each model with test dataset
- Plot Confusion Matrix

Model Comparison

- Comparison of models according to their accuracy
- The model with the best accuracy will be chosen (see Notebook for result)

[Link to code]()

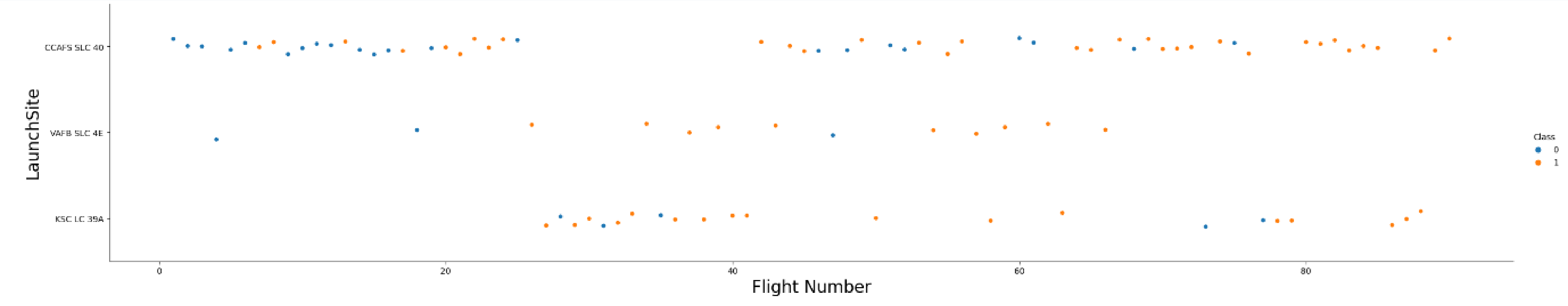
# Results

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

Section 2

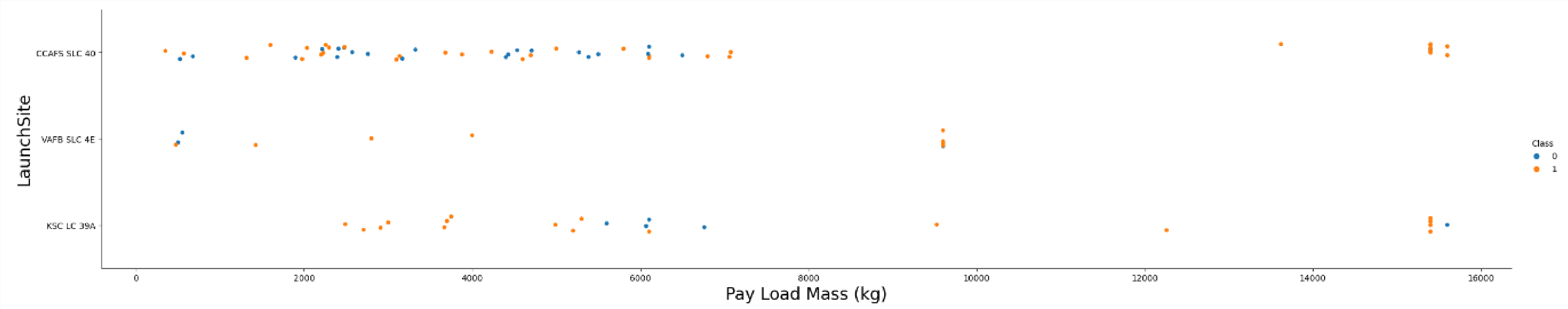
# Insights drawn from EDA

# Flight Number vs. Launch Site




We observe that, for each site, the success rate is increasing.

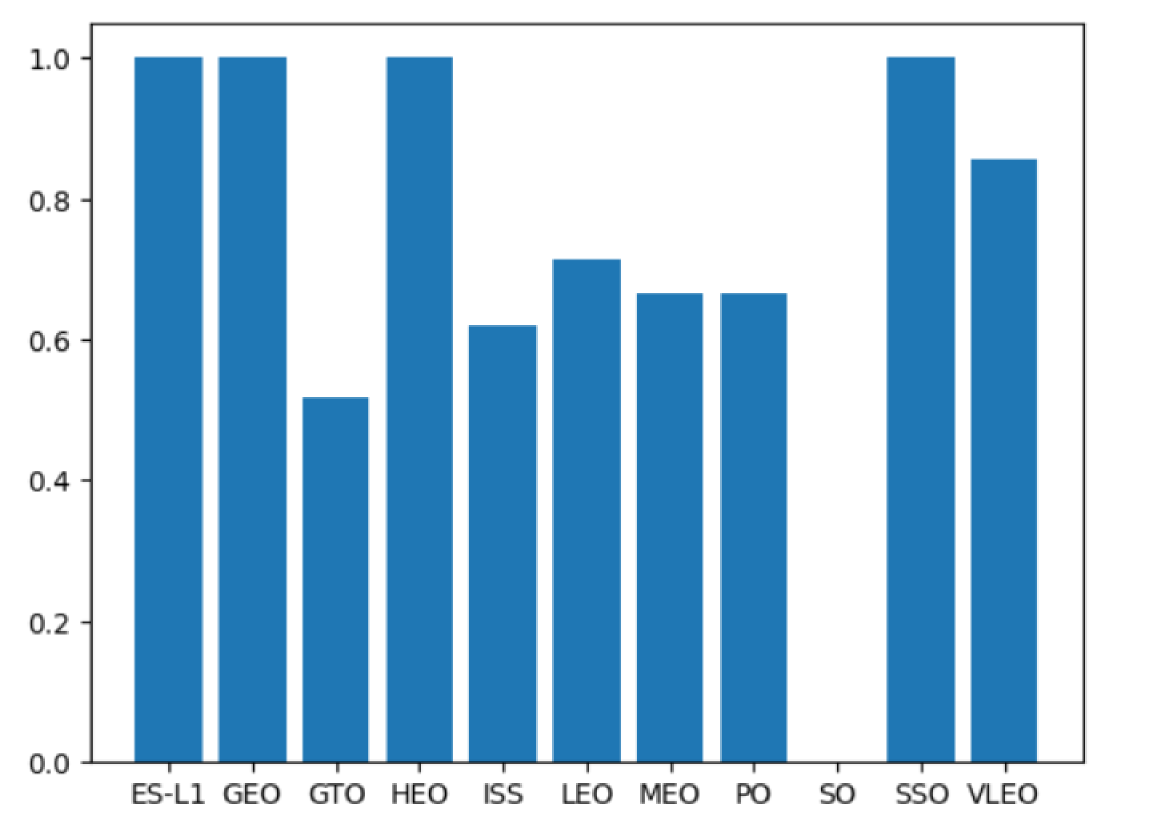
# Payload vs. Launch Site



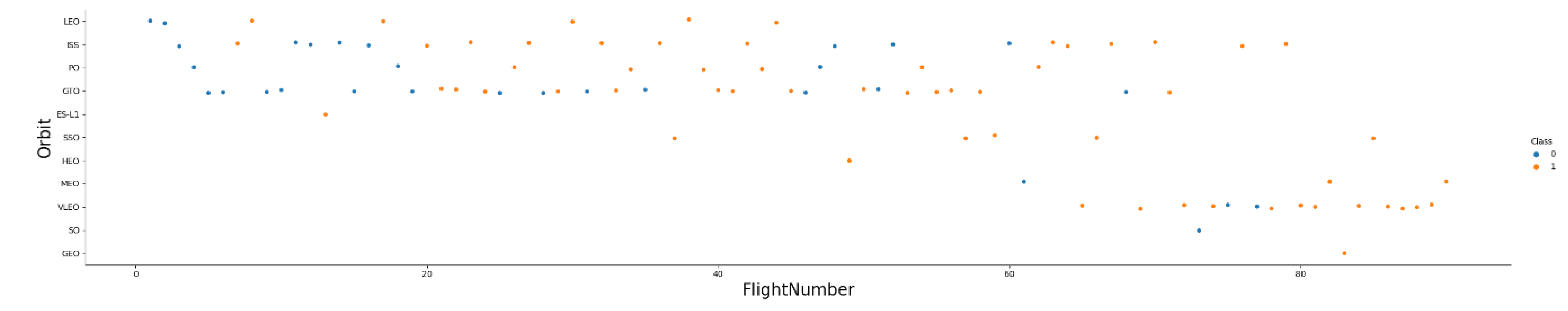

Depending on the launch site, a heavier payload may be a consideration for a successful landing. On the other hand, a too heavy payload can make a landing fail.

# Success Rate vs. Orbit Type

With this plot, we can see success rate for different orbit types. We note that ES-L1, GEO, HEO, SSO have the best success rate.

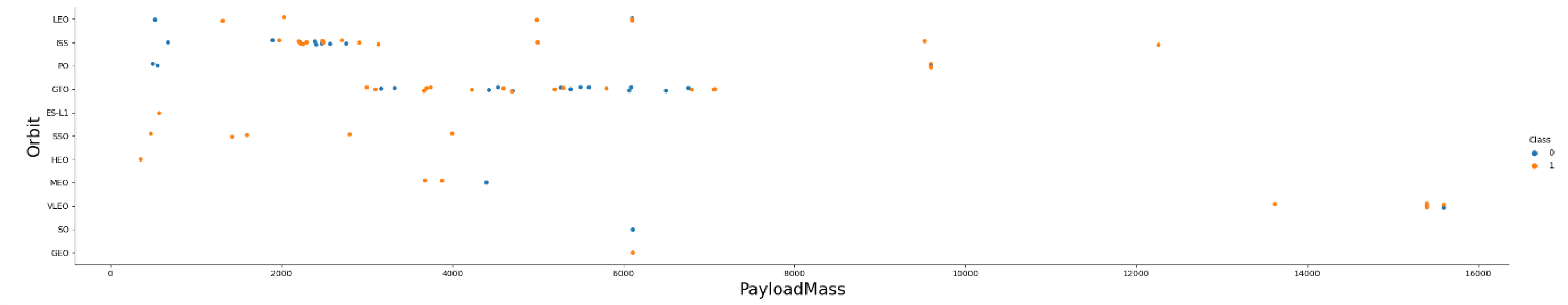


# Flight Number vs. Orbit Type



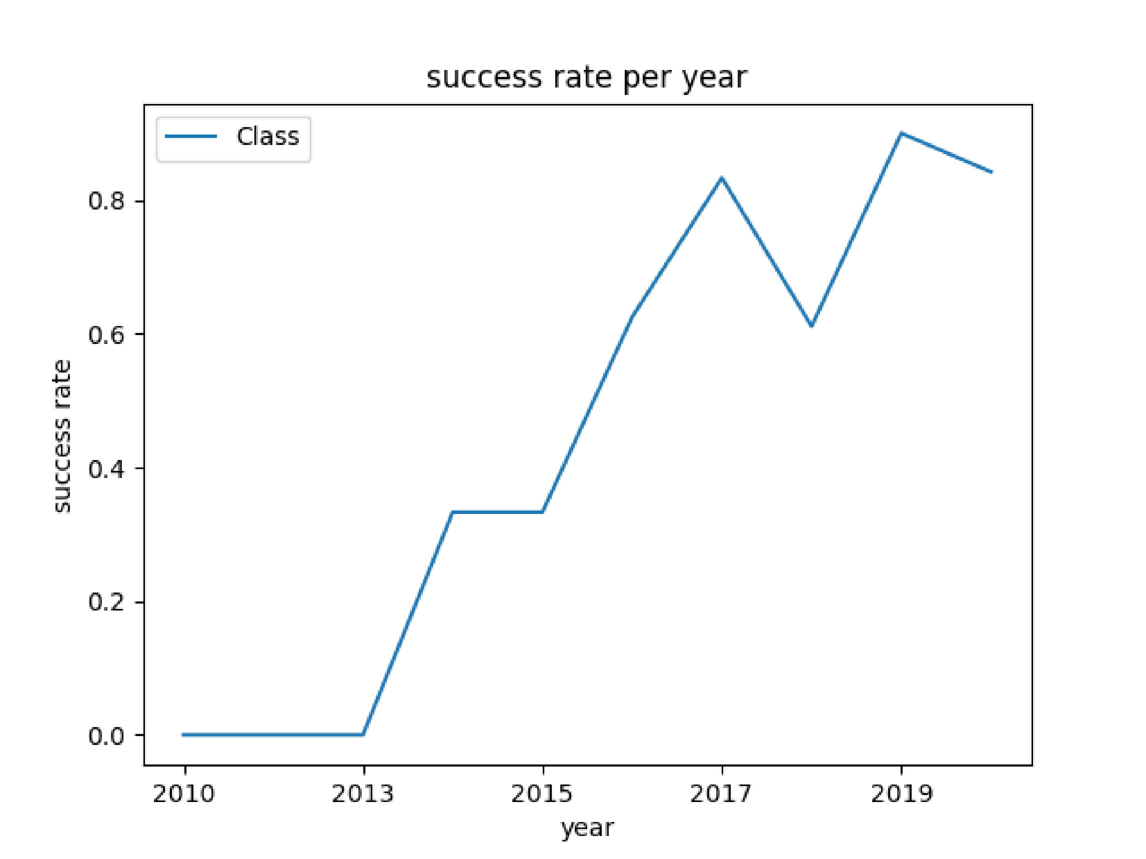

We notice that the success rate increases with the number of flights for the LEO orbit. For some orbits like GTO, there is no relation between the success rate and the number of flights. But we can suppose that the high success rate of some orbits like SSO or HEO is due to the knowledge learned during former launches for other orbits.

# Payload vs. Orbit Type




The weight of the payloads can have a great influence on the success rate of the launches in certain orbits. For example, heavier payloads improve the success rate for the LEO orbit.Another finding is that decreasing the payload weight for a GTO orbit improves the success of a launch.

# Launch Success Yearly Trend




Since 2013, we can see an increase in the Space X Rocket success rate.

# All Launch Site Names

Query

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

Result

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

[9]:

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

Explanation: The use of DISTINCT in the query allows to remove duplicate LAUNCH_SITE.

# Launch Site Names Begin with 'CCA'

Query

```
%%sql
SELECT *
FROM SPACEXTABLE
WHERE Launch_Site LIKE '%CCA%'
LIMIT 5
```

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

Result

| Date | Time (UTC) | Booster_Version | Launch_Site | Payload | PAYLOAD_MASS__KG_ | Orbit | Customer | Mission_Outcome | Landing_Outcome |
|---|---|---|---|---|---|---|---|---|---|
| 2010-04-06 | 18:45:00 | F9 v1.0 B0003 | CCAFS LC-40 | Dragon Spacecraft Qualification Unit | 0 | LEO | SpaceX | Success | Failure (parachute) |
| 2010-08-12 | 15:43:00 | F9 v1.0 B0004 | CCAFS LC-40 | Dragon demo flight C1, two CubeSats, barrel of Brouere cheese | 0 | LEO (ISS) | NASA (COTS) NRO | Success | Failure (parachute) |
| 2012-05-22 | 07:44:00 | F9 v1.0 B0005 | CCAFS LC-40 | Dragon demo flight C2 | 525 | LEO (ISS) | NASA (COTS) | Success | No attempt |
| 2012-08-10 | 00:35:00 | F9 v1.0 B0006 | CCAFS LC-40 | SpaceX CRS-1 | 500 | LEO (ISS) | NASA (CRS) | Success | No attempt |
| 2013-01-03 | 15:10:00 | F9 v1.0 B0007 | CCAFS LC-40 | SpaceX CRS-2 | 677 | LEO (ISS) | NASA (CRS) | Success | No attempt |

Explanation:

The WHERE clause followed by LIKE clause filters launch sitesthat contain the substring CCA.

LIMIT 5 shows 5 records from filtering.

# Total Payload Mass

Query

```
%%sql
SELECT Customer, SUM(PAYLOAD_MASS__KG_) AS total_PAYLOAD_MASS__KG
FROM SPACEXTABLE
WHERE Customer = 'NASA (CRS)'
GROUP BY Customer
```

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

| Customer | total_PAYLOAD_MASS__KG |
| --- | --- |
| NASA (CRS) | 45596 |

Result

Explanation:

This query returns the sum of all payload masses where the customer is NASA (CRS).

# Average Payload Mass by F9 v1.1

Query

```
%%sql
SELECT Booster_Version, AVG(PAYLOAD_MASS__KG_) AS average_PAYLOAD_MASS__KG
FROM SPACEXTABLE
WHERE Booster_Version = 'F9 v1.1'
GROUP BY Booster_Version
```

Result

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

| Booster_Version | average_PAYLOAD_MASS__KG |
|---|---|
| F9 v1.1 | 2928.4 |

Explanation:

This query returns the average of all payload masses where the booster version contains the substring F9 v1.1.

# First Successful Ground Landing Date

Query

```
%%sql
SELECT MIN(Date) AS first_succesfull_landing_date
FROM SPACEXTABLE
WHERE Landing_Outcome = 'Success (ground pad)'
```

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

Result

| first_succesfull_landing_date |
|---|
| 2015-12-22 |

Explanation:

With this query, we select the oldest successful landing.

The WHERE clause filters dataset in order to keep only records where landing was successful.
With the MIN function, we select the record with the oldest date.

# Successful Drone Ship Landing with Payload between 4000 and 6000

Query

```
%%sql
SELECT Booster_Version AS names_of_the_boosters
FROM SPACEXTABLE
WHERE Landing_Outcome = 'Success (drone ship)'
    AND PAYLOAD_MASS__KG_ > 4000
    AND PAYLOAD_MASS__KG_ < 6000
```

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

Result

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

Explanation:

This query returns the booster version where landing was successful and payload mass is between 4000 and 6000 kg. The WHERE and AND clauses filter the dataset.

# Total Number of Successful and Failure Mission Outcomes

Query

```
%%sql
SELECT Mission_Outcome, COUNT(*) AS total_outcomes
FROM SPACEXTABLE
GROUP BY Mission_Outcome;
```

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

Result

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

Explanation:

With the first SELECT, we show the queries counts total mission outcome. The group by classifies data record from mission outcome column.

# Boosters Carried Maximum Payload

Query →

```
%%sql
SELECT Booster_Version, PAYLOAD_MASS__KG_
FROM SPACEXTABLE
WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTABLE)
```

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

Result →

| Booster_Version | PAYLOAD_MASS__KG_ |
|---|---|
| F9 B5 B1048.4 | 15600 |
| F9 B5 B1049.4 | 15600 |
| F9 B5 B1051.3 | 15600 |
| F9 B5 B1056.4 | 15600 |
| F9 B5 B1048.5 | 15600 |
| F9 B5 B1051.4 | 15600 |
| F9 B5 B1049.5 | 15600 |
| F9 B5 B1060.2 | 15600 |
| F9 B5 B1058.3 | 15600 |
| F9 B5 B1051.6 | 15600 |
| F9 B5 B1060.3 | 15600 |
| F9 B5 B1049.7 | 15600 |

Explanation:

We used a subquery to filter data by returning only the heaviest payload mass with MAX function. The main query uses subquery results and returns unique booster version (SELECT DISTINCT) with the heaviest payload mass.

# 2015 Launch Records

Query

```
%%sql
SELECT
    CASE
        WHEN substr(Date, 6, 2) = '01' THEN 'January'
        WHEN substr(Date, 6, 2) = '02' THEN 'February'
        WHEN substr(Date, 6, 2) = '03' THEN 'March'
        WHEN substr(Date, 6, 2) = '04' THEN 'April'
        WHEN substr(Date, 6, 2) = '05' THEN 'May'
        WHEN substr(Date, 6, 2) = '06' THEN 'June'
        WHEN substr(Date, 6, 2) = '07' THEN 'July'
        WHEN substr(Date, 6, 2) = '08' THEN 'August'
        WHEN substr(Date, 6, 2) = '09' THEN 'September'
        WHEN substr(Date, 6, 2) = '10' THEN 'October'
        WHEN substr(Date, 6, 2) = '11' THEN 'November'
        WHEN substr(Date, 6, 2) = '12' THEN 'December'
    END AS Month_Name,
    Landing_Outcome,
    Booster_Version,
    Launch_Site
FROM SPACEXTABLE
WHERE substr(Date, 1, 4) = '2015'
    AND Landing_Outcome = 'Failure (drone ship)'
```

Result

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

| Month_Name | Landing_Outcome | Booster_Version | Launch_Site |
|---|---|---|---|
| October | Failure (drone ship) | F9 v1.1 B1012 | CCAFS LC-40 |
| April | Failure (drone ship) | F9 v1.1 B1015 | CCAFS LC-40 |

Explanation:

This query returns month, booster version, launch site where landing was unsuccessful and landing date took place in 2015. Substr function process date in order to take month or year. Substr(DATE,1, 4) shows year.

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

Query

```
%%sql
SELECT "Landing_Outcome", COUNT(*) as "Outcome_Count"
FROM SPACEXTABLE
WHERE "Date" >= '2010-06-04' AND "Date" <= '2017-03-20'
  AND ("Landing_Outcome" LIKE 'Success (ground pad)' OR "Landing_Outcome" LIKE 'Failure (drone ship)')
GROUP BY "Landing_Outcome"
ORDER BY "Outcome_Count" DESC
```

Result

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

| Landing_Outcome | Outcome_Count |
|---|---|
| Success (ground pad) | 5 |
| Failure (drone ship) | 5 |

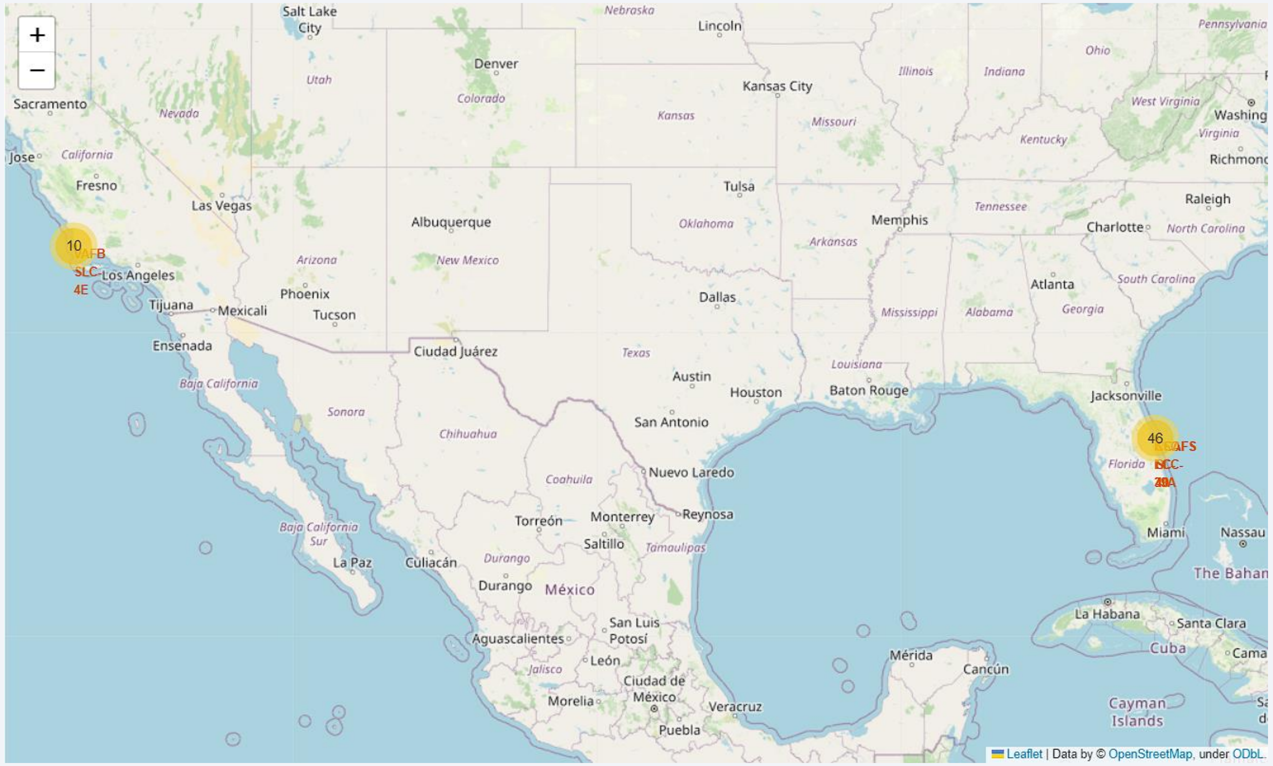
Explanation:

It focuses on missions that happened between June 4, 2010, and March 20, 2017. It counts how many times each outcome is labeled as either "Success (ground pad)" or "Failure (drone ship)". The code then shows these counts, arranging them from the most common outcome to the least common one.

Section 3

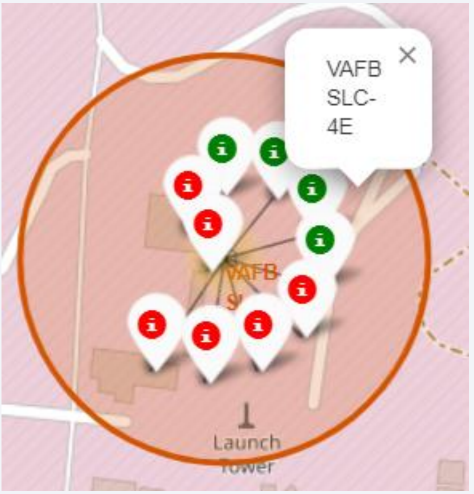
# Launch Sites Proximities Analysis

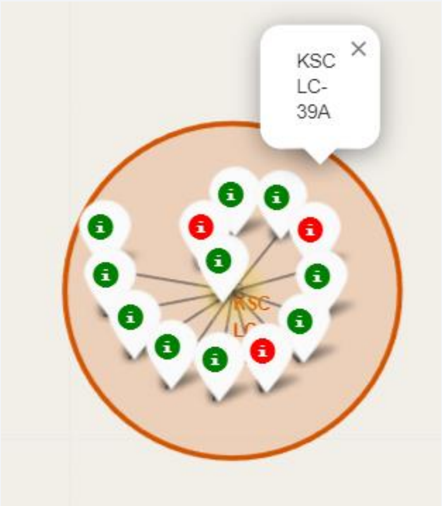
# Folium map – Ground Station

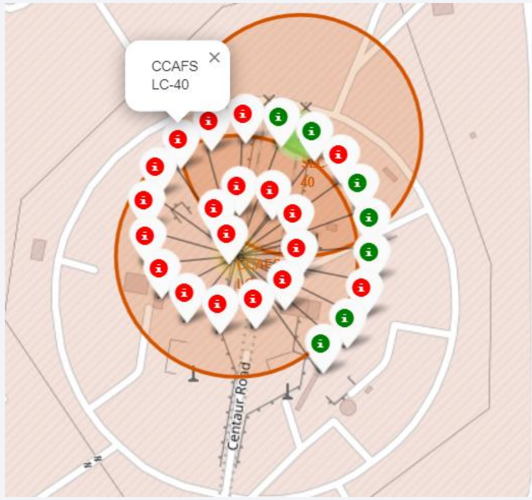


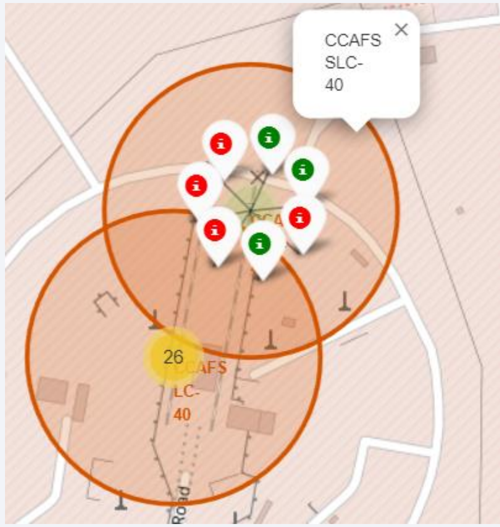

We see that Space X launch sites are located on the coast of the United States

# Folium map – Color Labeled Markers



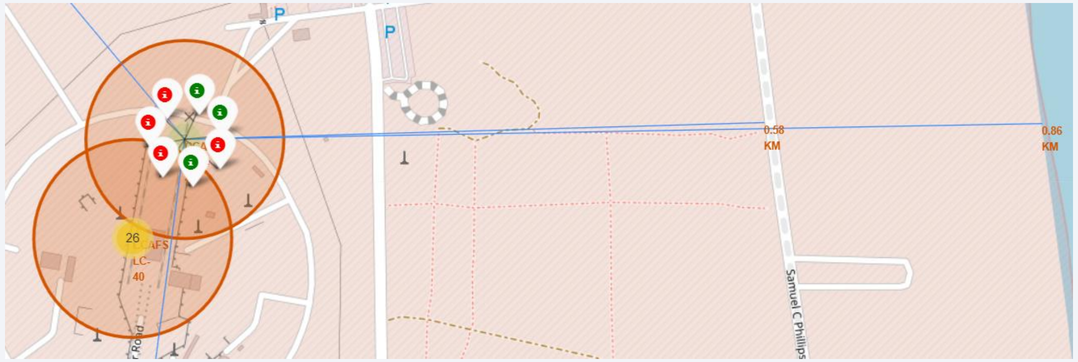



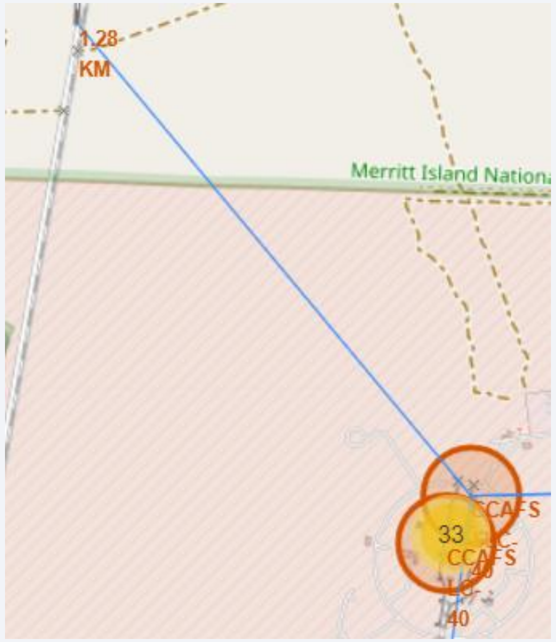



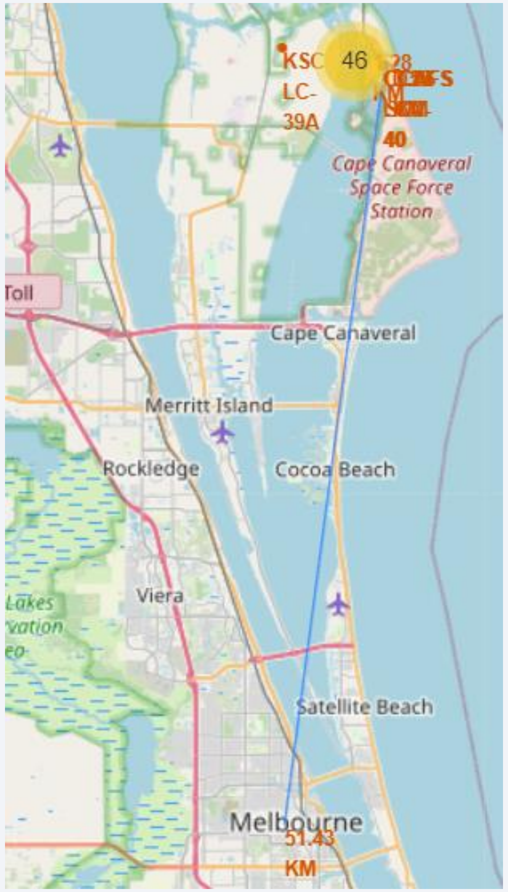




Green marker represents successful launches. Red marker represents unsuccessful launches. We note that KSC LC-39A has a higher launch success rate.

# Folium map – Distance between CCAFS SLC-40 and its proximities



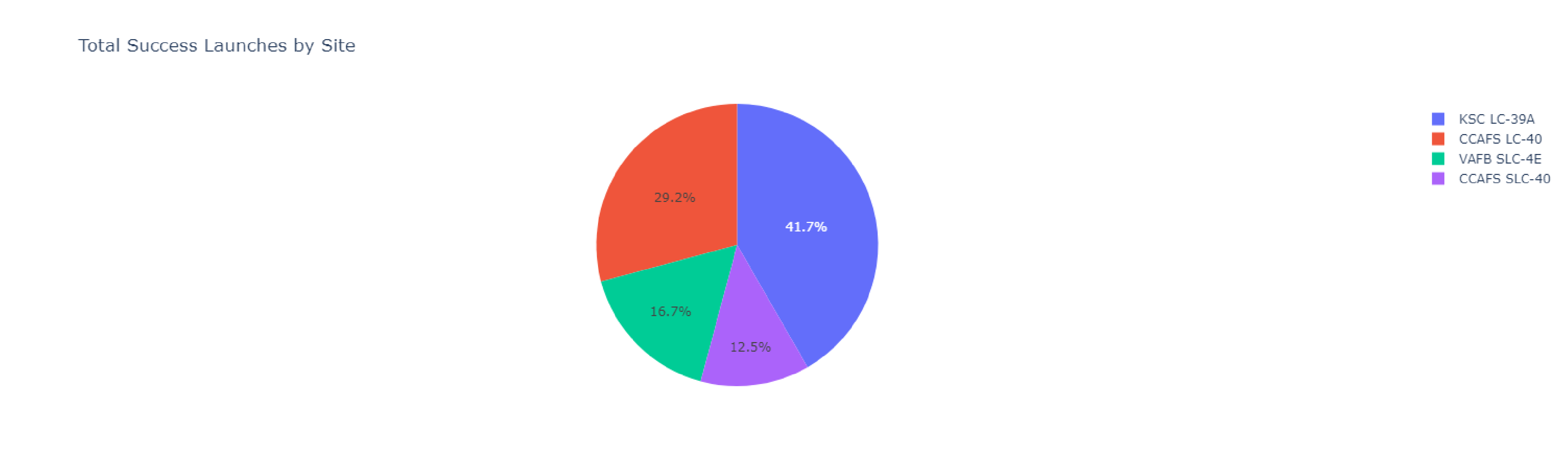




- Is CCAFS SLC-40 in close proximity to railways ? Yes
- Is CCAFS SLC-40 in close proximity to highways ? Yes
- Is CCAFS SLC-40 in close proximity to coastline ? Yes
- DoCCAFS SLC-40 keeps certain distance away from cities ? No

Section 4

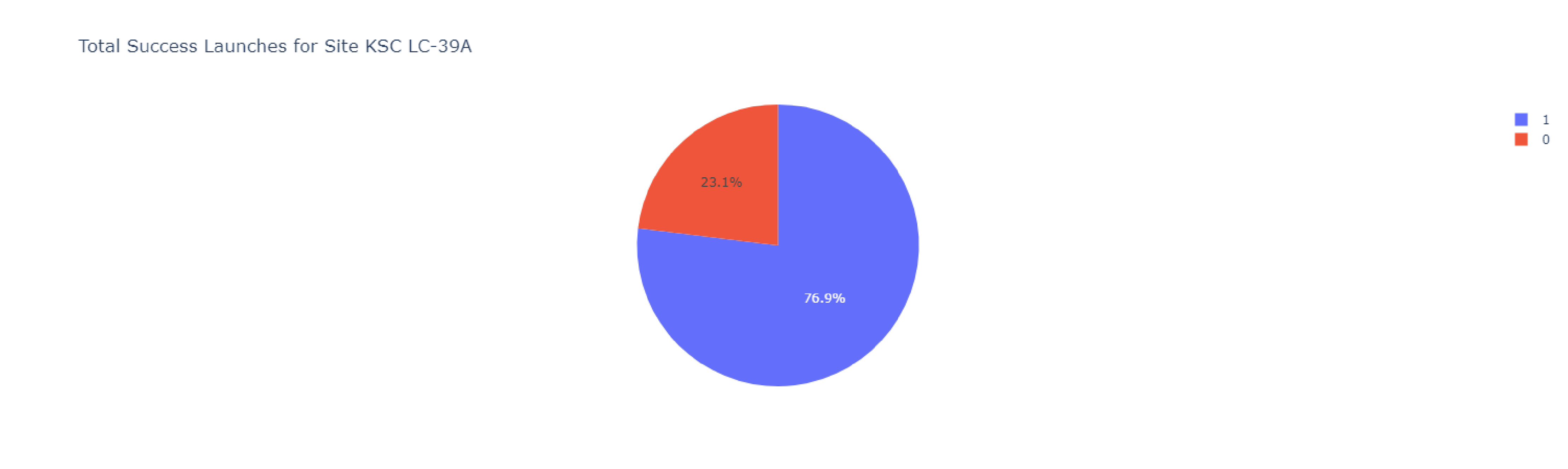
# Build a Dashboard with Plotly Dash

# Dashboard – Total success by Site



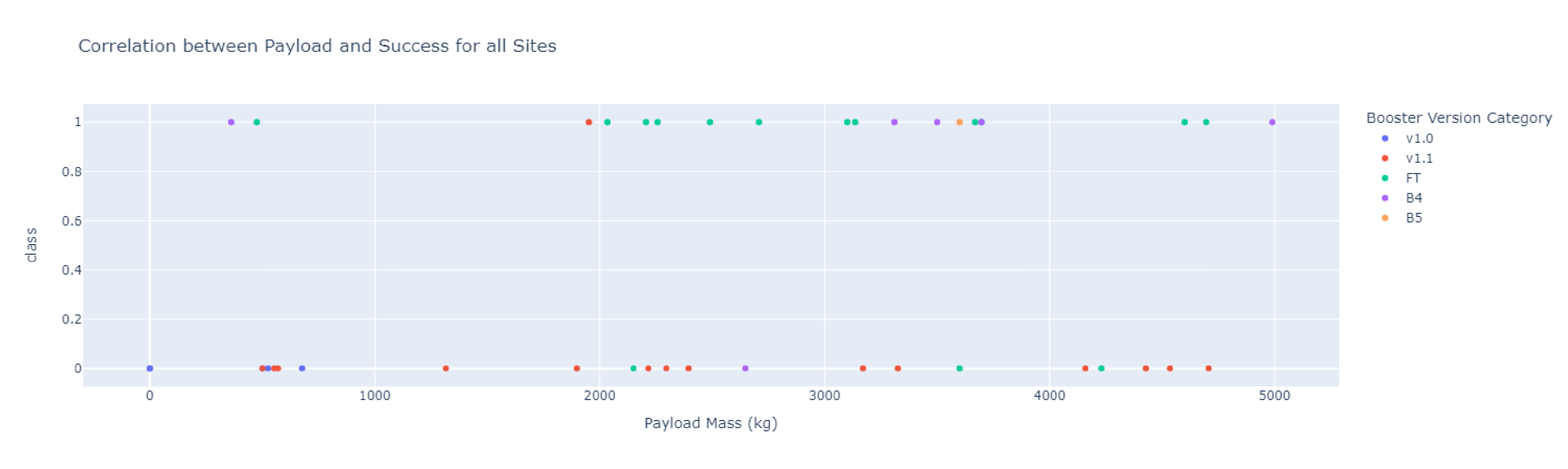

We see that KSC LC-39A has the best success rate of launches.

# Dashboard – Total success launches for Site KSC LC-39 A




We see that KSC LC-39A has achieved a 76.9% success ratewhile getting a 23.1% failure rate.

# Dashboard –Payload mass vs Outcome for all sites with different payload mass selected



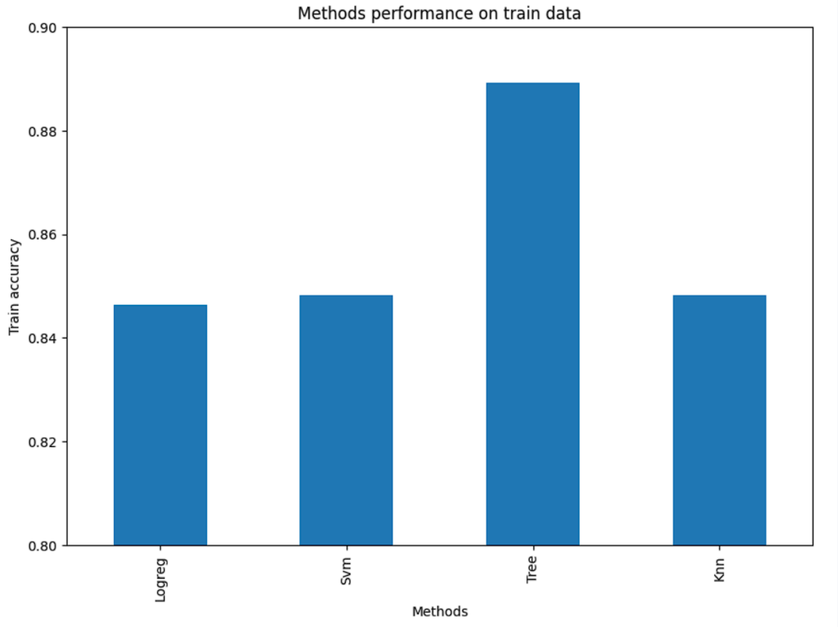

Low weighted payloads have a better success rate than the heavy weighted payloads.

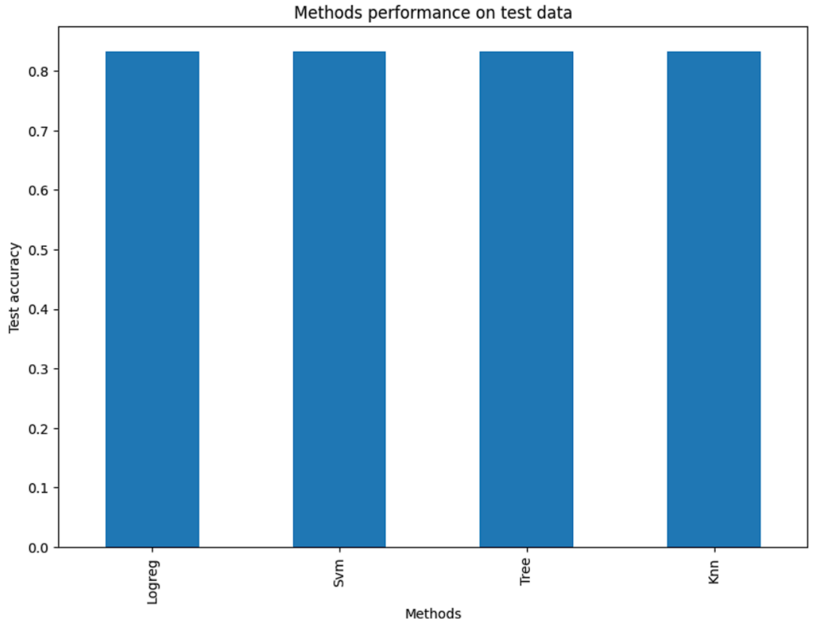
Section 5

# Predictive Analysis (Classification)

# Classification Accuracy

| | Accuracy Train | Accuracy Test |
|---|---|---|
| **Tree** | 0.889286 | 0.833333 |
| **Knn** | 0.848214 | 0.833333 |
| **Svm** | 0.848214 | 0.833333 |
| **Logreg** | 0.846429 | 0.833333 |



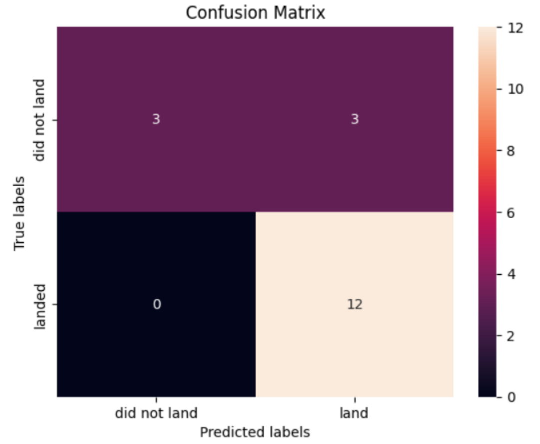

For accuracy test, all methods performed similar. **We could get more test data to decide between them.**But if we really need to choose one right now, we would take the decision tree.

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

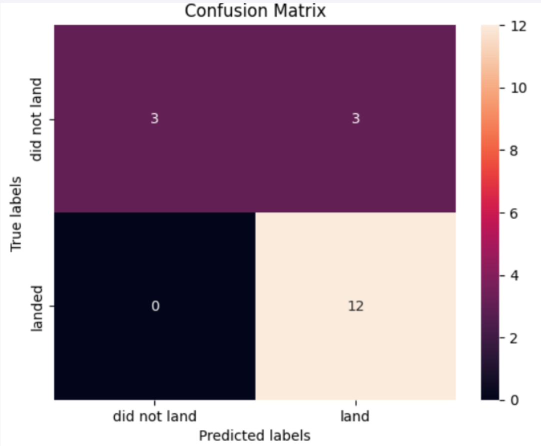
```
tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', 'max_depth': 16, 'max_features': 'auto', 'min_samples_leaf': 4, 'min_samp
les_split': 2, 'splitter': 'random'}
accuracy : 0.8892857142857142
```

# Confusion Matrix

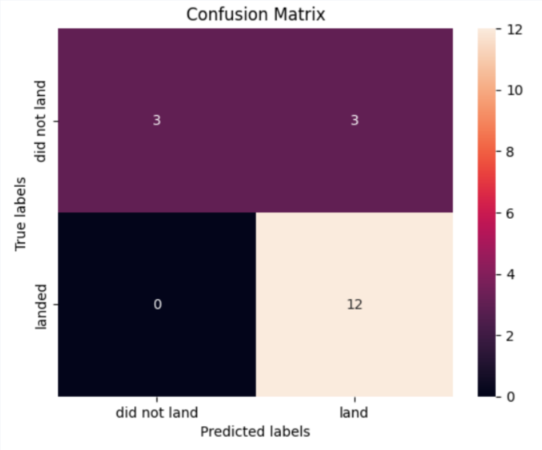
### Logistic regression



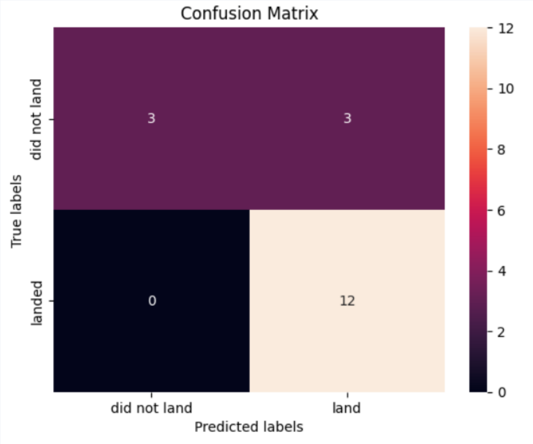

### SVM




### Decision Tree




### KNN




As the test accuracy are all equal, the confusion matrices are also identical. The main problem of these models are false positives.

| | | Actual values | |
|---|---|---|---|
| | | 1 | 0 |
| Predicted values | 1 | TP | FP |
| | 0 | FN | TN |

# Conclusions

- The success of a mission can be explained by several factors such as the launch site, the orbit and especially the number of previous launches. Indeed, we can assume that there has been a gain in knowledge between launches that allowed to go from a launch failure to a success.
- The orbits with the best success rates are GEO, HEO, SSO, ES-L1.
- Depending on the orbits, the payload mass can be a criterion to take into account for the success of a mission. Some orbits require a light or heavy payload mass. But generally low weighted payloads perform better than the heavy weighted payloads.
- With the current data, we cannot explain why some launch sites are better than others (KSC LC-39A is the best launch site). To get an answer to this problem, we could obtain atmospheric or other relevant data.
- For this dataset, we choose the Decision Tree Algorithm as the best model even if the test accuracy between all the models used is identical. We choose Decision Tree Algorithm because it has a better train accuracy.

# Thank you!