IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

AGUSTIN VICENTE GIMENEZ
22/02/2025

https://github.com/agusvicgim/Final-IBM-data-course/tree/main

Outline

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

Executive Summary

- **Summary of methodologies**
- Data collection and data wrangling methodology
- EDA and interactive visual analytics
- Predictive analysis methodology
- EDA with visualization results
- EDA with SQL results
- Interactive map with Folium
- Plotly Dash dashboard
- Predictive analysis

- **Summary of all results**
- EDA
- Interactive
- Predictive

Introduction

- **Project background and context**
- We, Space Y, a new rocket company, aims to compete by predicting whether SpaceX will reuse the first stage, impacting launch costs.
- The commercial space industry is growing, with companies like SpaceX leading the way in reducing launch costs through reusable rockets. SpaceX's Falcon 9 reuses its first stage, lowering costs to $62 million per launch compared to competitors at $165 million.

- **Problems you want to find answers**
- What factors affect first-stage recovery?
- How can we predict the launch would be successful?
- How can we do to make sure the launch would be succesful?

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - Space X open source API
  - Web scrapping from Wikipedia "List of falcon 9 and Falcon heavy launches"
- Perform data wrangling
  - Transforming categorical data using OneHotEncoding for machine learning algorithym and removing any empty or unnecessary data.
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - Logistic regression, KNN, SVM, Decision Tree used to find the best classification method.

Data Collection

- Data collection was achieve using SpaceX API, decoded with json and normalized in a Panda's data frame
- Data was cleaned and Created a new data fame with only Falcon 9 Launches
- Checked for missing values.

Data Collection – SpaceX API

Task 1: Request and parse the SpaceX launch data using the GET request

To make the requested JSON results more consistent, we will use the following static response object for this project:

In [9]:
```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_
```

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

In [10]:
```
response=requests.get(static_json_url)
```

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

Out[11]: 200

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

In [24]:
```
# Decode the JSON content from the response
data = pd.json_normalize(response.json())
```

Using the dataframe `data` print the first 5 rows

In [25]:
```
# Display the first 5 rows of the dataframe
print(df.head())
```

```
      static_fire_date_utc  static_fire_date_unix    tbd    net  window  \
0  2006-03-17T00:00:00.000Z           1.142554e+09  False  False     0.0
```

Use Get request to get info form spaceX API and create new dataset

Pl
he

```
getLaunchSite(data)
```

In [33]:
```
# Call getPayloadData
getPayloadData(data)
```

In [34]:
```
# Call getCoreData
getCoreData(data)
```

Finally lets construct our dataset using the data we have obtained. We we combine the columns into a dictionary.

In [35]:
```
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}
```

Then, we need to create a Pandas data frame from the dictionary launch_dict.

In [36]:
```
# Create a DataFrame from the launch_dict
launch_df = pd.DataFrame(launch_dict)
```

Show the summary of the dataframe

In [38]:
```
# Show the head of the dataframe

print(launch_df.head())
```

```
Empty DataFrame
```

https://github.com/agusvicgim/Final-IBM-data-course/blob/main/jupyter-labs-spacex-data-collection-api%20(1).ipynb

Data Collection – SpaceX API

Task 2: Filter the dataframe to only include `Falcon 9` launches

Finally we will remove the Falcon 1 launches keeping only the Falcon 9 launches. Filter the data dataframe using the `BoosterVersion` column to only keep the Falcon 9 launches. Save the filtered data to a new dataframe called `data_falcon9`.

In [39]:

```
# Hint data['BoosterVersion']!='Falcon 1'
# Filter the data to only keep Falcon 9 launches
data_falcon9 = launch_df[launch_df['BoosterVersion'] != 'Falcon 1']

# Display the first 5 rows of the filtered dataframe
print(data_falcon9.head())
```

```
Empty DataFrame
Columns: [FlightNumber, Date, BoosterVersion, PayloadMass, Orbit, LaunchSite, Outcome, Flights, GridFins, Reused, Legs, Landi
ngPad, Block, ReusedCount, Serial, Longitude, Latitude]
Index: []
```

Now that we have removed some values we should reset the FlgihtNumber column

In [40]:

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

Out[40]:

FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block

New df with only Falcon 9 launches

Data Wrangling

We can see below that some of the rows are missing values in our dataset.

In [41]:

```
data_falcon9.isnull().sum()
```

Task 3: Dealing with Missing Values

Calculate below the mean for the `PayloadMass` using the `.mean()`. Then use the mea `np.nan` values in the data with the mean you calculated.

```
# Calculate the mean value of PayloadMass column

# Calculate the mean value of PayloadMass column
mean_payload_mass = data_falcon9['PayloadMass'].mean()

# Replace the np.nan values in PayloadMass with the mean value
data_falcon9['PayloadMass'] = data_falcon9['PayloadMass'].replace(np.nan, m

# Verify if there are any remaining missing values
print(data_falcon9.isnull().sum())
```

```
FlightNumber      0.0
Date              0.0
BoosterVersion    0.0
PayloadMass       0.0
Orbit             0.0
LaunchSite        0.0
Outcome           0.0
Flights           0.0
GridFins          0.0
Reused            0.0
Legs              0.0
LandingPad        0.0
Block             0.0
ReusedCount       0.0
Serial            0.0
Longitude         0.0
```

Removing missing values

https://github.com/agusvicgim/Final-IBM-data-course/blob/main/jupyter-labs-spacex-data-collection-api%20(1).ipynb

Data Collection - Scraping

- WebScrapping form Wikipedia using Beautiful Soup
- Extract colums and Create a new data frame

https://github.com/agusvicgim/Final-IBM-data-course/blob/main/jupyter-labs-webscraping%20(1).ipynb

Data Collection - Scraping

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

```
4]:
static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launch
```

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.

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

```
200
```

Request data form Wikipedia using beautiful soup

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

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

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

```
7]:
# Use soup.title attribute
print(soup.title)
```

```
<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 reference link towards the end of this lab

```
# Use the find_all function in the BeautifulSoup object, with element type `table`
# Assign the result to a list called `html_tables`
# Find all tables on the Wikipedia page
html_tables = soup.find_all('table')

# Print the number of tables found to verify the search
print(f"Number of tables found: {len(html_tables)}")

# Extract the column names from the first table (assuming the first table is the one with the launch data)
header_row = html_tables[0].find_all('th')

# Extract and clean the column names
column_names = [extract_column_from_header(row) for row in header_row]

# Display the extracted column names
print("Column Names:", column_names)
```

Extract colums form HTLM

```
lumber of tables found: 26
olumn Names: []
```

Starting from the third table is our target table contains the actual launch records.

```
# Let's print the third table and check its content
first_launch_table = html_tables[2]
print(first_launch_table)
```

```
table class="wikitable plainrowheaders collapsible" style="width: 100%;">
```


https://github.com/agusvicgim/Final-IBM-data-course/blob/main/jupyter-labs-webscraping%20(1).ipynb

Data Collection - Scraping

TASK 3: Create a data frame by parsing the launch HTML tables

We will create an empty dictionary with keys from the extracted column names in the previous task. Later, this dictionary w
into a Pandas dataframe

```
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']=[]
```

Create New dataframe

Next, we just need to fill up the `launch_dict` with launch records extracted from table rows.

Usually, HTML tables in Wiki pages are likely to contain unexpected annotations and other types of noises, such as referer
`B0004.1[8]`, missing values `N/A [e]`, inconsistent formatting, etc.

To simplify the parsing process, we have provided an incomplete code snippet below to help you to fill up the `launch_d`
complete the following code snippet with TODOs or you can choose to write your own logic to parse all launch tables:

```
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:
```

```
bv=booster_version(row[1])
if not(bv):
    bv=row[1].a.string
print(bv)

# Launch Site
# TODO: Append the bv into launch_dict with key `Launch Site`
launch_site = row[2].a.string
#print(launch_site)

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

# Payload Mass
# TODO: Append the payload_mass into launch_dict with key `Payload mass`
payload_mass = get_mass(row[4])
#print(payload)

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

# Customer
# TODO: Append the customer into launch_dict with key `Customer`
customer = row[6].a.string
#print(customer)

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

# Booster landing
# TODO: Append the launch_outcome into launch_dict with key `Booster landing`
booster_landing = landing_status(row[8])
#print(booster_landing)
```

After you have fill in the parsed launch record values into `launch_dict`, you can create a dataframe from it.

```
df= pd.DataFrame({ key:pd.Series(value) for key, value in launch_dict.items() })
```


https://github.com/agusvicgim/Final-IBM-data-course/blob/main/jupyter-labs-webscraping%20(1).ipynb

Data Wrangling

- We calculate the number of launches
- Calculate the launches and ocurrence for each orbit
- Calculate the outcome for the orbits
- Create landing outcome label

- https://github.com/agusvicgim/Final-IBM-data-course/blob/main/labs-jupyter-spacex-Data%20wrangling%20(2).ipynb

Data Wrangling

TASK 1: Calculate the number of launches on each site

The data contains several Space X launch facilities: Cape Canaveral Space Launch Complex 40 V/
Space Launch Complex 4E **(SLC-4E)**, Kennedy Space Center Launch Complex 39A **KSC LC 39A** .T
column `LaunchSite`

Next, let's see the number of launches for each site.

Use the method `value_counts()` on the column `LaunchSite` to determine the number of l

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

# Display the result
print(launch_counts)
```

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

calculate number launches

Each launch aims to an dedicated orbit, and here are some common orbit types:

TASK 2: Calculate the number and occurrence of each orbit

Use the method `.value_counts()` to determine the number and occurrence of each orbit

```
# Apply value_counts on Orbit column

# Calculate the number and occurrence of each orbit
orbit_counts = df['Orbit'].value_counts()

# Display the result
print(orbit_counts)
```

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

Calculate for each orbit and outcome

- https://github.com/agusvicgim/Final-IBM-data-course/blob/main/labs-jupyter-spacex-Data%20wrangling%20(2).ipynb

TASK 3: Calculate the number and occurence of mission outcome of the orbits

Use the method `.value_counts()` on the column `Outcome` to determine the number of `landing_outcomes` .The variable landing_outcomes.

```
# landing_outcomes = values on Outcome column

# Calculate the number and occurrence of mission outcomes
landing_outcomes = df['Outcome'].value_counts()

# Display the result
print(landing_outcomes)
```

```
Outcome
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: count, dtype: int64
```

`True Ocean` means the mission outcome was successfully landed to a specific region of the ocean while `False Oce` outcome was unsuccessfully landed to a specific region of the ocean. `True RTLS` means the mission outcome was su ground pad `False RTLS` means the mission outcome was unsuccessfully landed to a ground pad. `True ASDS` mear outcome was successfully landed to a drone ship `False ASDS` means the mission outcome was unsuccessfully lande `None ASDS` and `None None` these represent a failure to land.

```
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:

Create lancind outcome label for colum

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 one. Then assign it to the variable `landing_class`:

```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
# Define the bad_outcomes set
bad_outcomes = {'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}

# Create the landing_class list
landing_class = [0 if outcome in bad_outcomes else 1 for outcome in df['Outcome']]

# Assign the landing_class list to the dataframe as a new column
df['landing_class'] = landing_class

# Display the updated dataframe
df.head()

# Count how many mission outcomes were successfully landed to a drone ship (True ASDS)
successful_drone_ship_landings = df[df['Outcome'] == 'True ASDS'].shape[0]

# Print the result
print(successful_drone_ship_landings)
```

41

This variable will represent the classification variable that represents the outcome of each launch. If the valu land successfully; one means the first stage landed Successfully

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

	Class
0	0
1	0
2	0

Calculate success rate

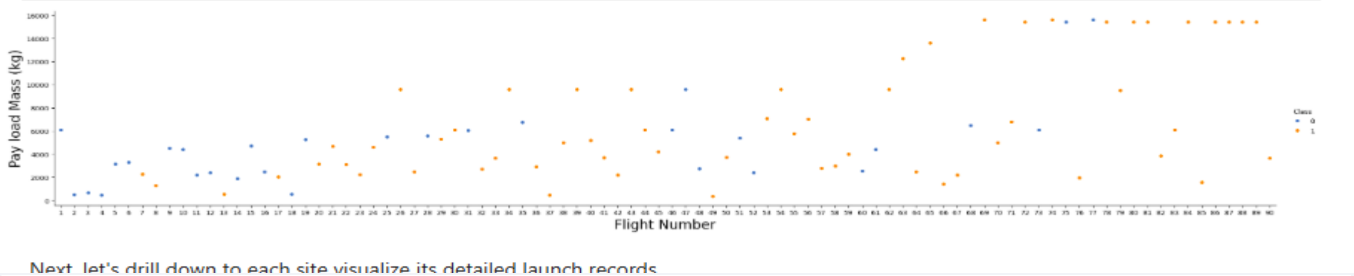
- https://github.com/agusvicgim/Final-IBM-data-course/blob/main/labs-jupyter-spacex-Data%20wrangling%20(2).ipynb

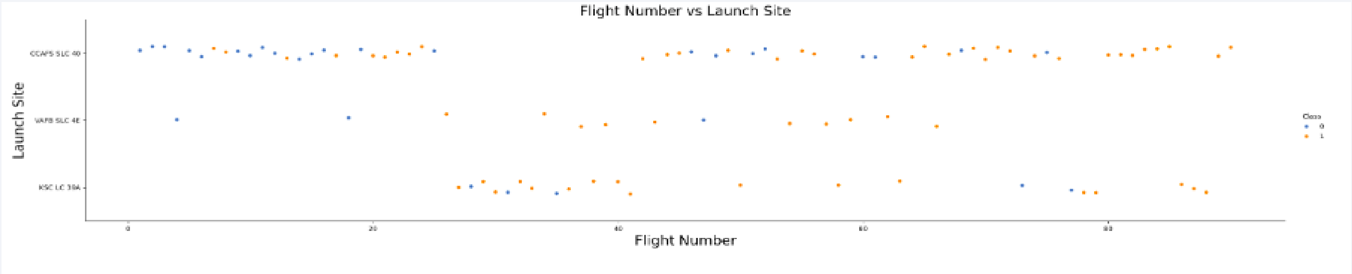
EDA with Data Visualization

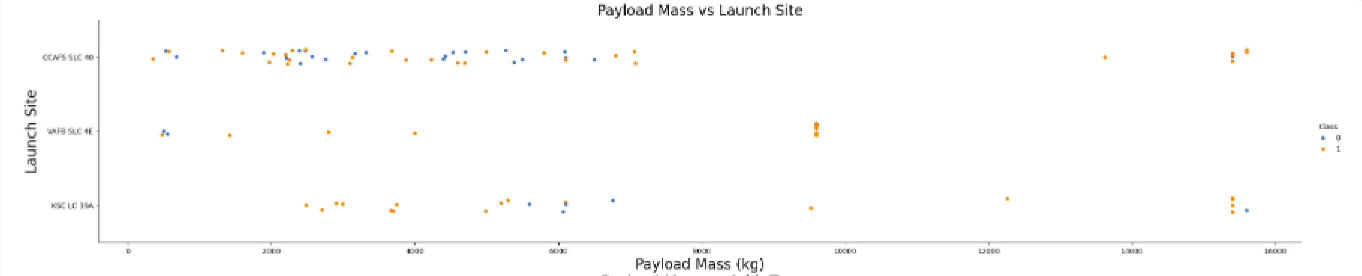
- Visualize the relationship between Flight Number and Launch Site
- Visualize the relationship between Payload Mass and Launch Site
- Visualize the relationship between FlightNumber and Orbit type
- Visualize the relationship between Payload Mass and Orbit type
- Visualize the relationship between success rate of each orbit type
- Visualize the launch success yearly trend

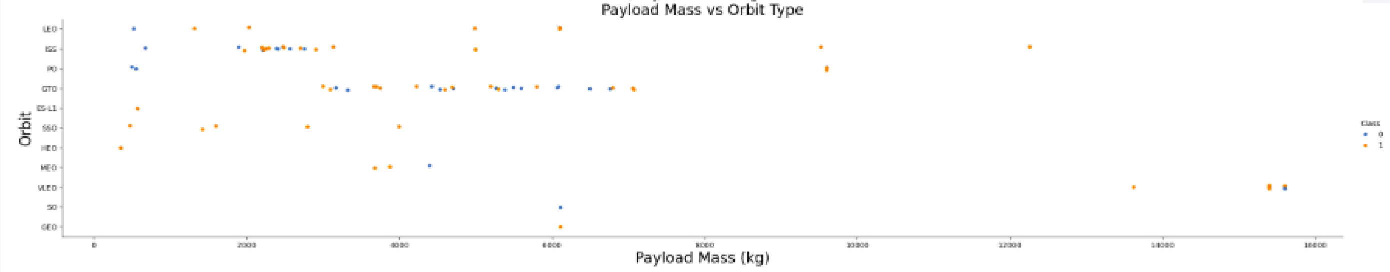
https://github.com/agusvicgim/Final-IBM-data-course/blob/main/edadataviz%20(1).ipynb

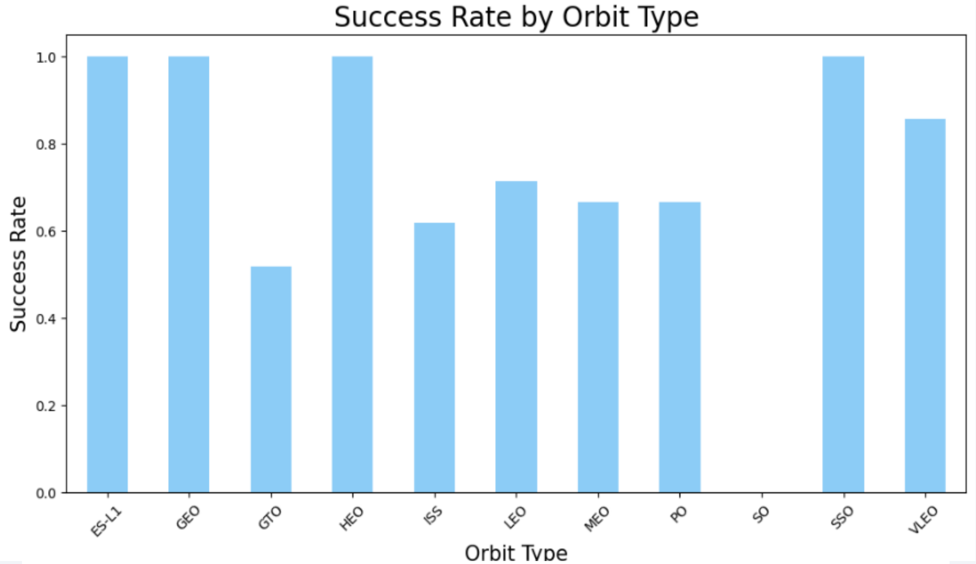
EDA with Data Visualization

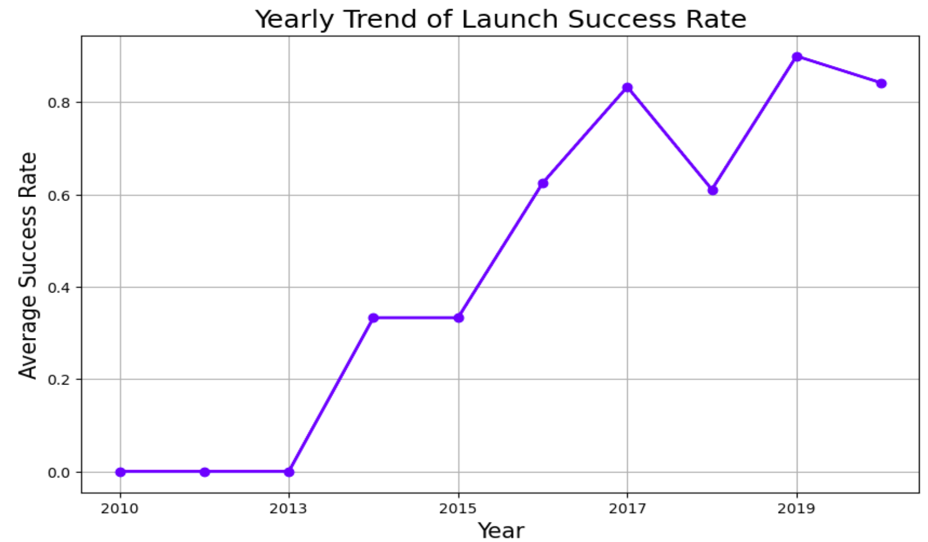












- https://github.com/agusvicgim/Final-IBM-data-course/blob/main/edadataviz%20(1).ipynb

EDA with SQL

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 average payload mass carried by booster version F9 v1.1
5. List the date when the first successful landing outcome in ground pad was achieved.
6. List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000
7. List the total number of successful and failure mission outcomes
8. List the names of the booster versions which have carried the maximum payload mass using a subquery
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.
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

https://github.com/agusvicgim/Final-IBM-data-course/blob/main/jupyter-labs-eda-sql-coursera_sqllite%20(1).ipynb

Build an Interactive Map with Folium

- We created a map using folium with the different launches areas.
- We add markers to know the name and how many launches were done in each area.
- We added a marker with color green or red to visualize succesful or not launches.
- We calculated the distance to proximities as the coastlines.

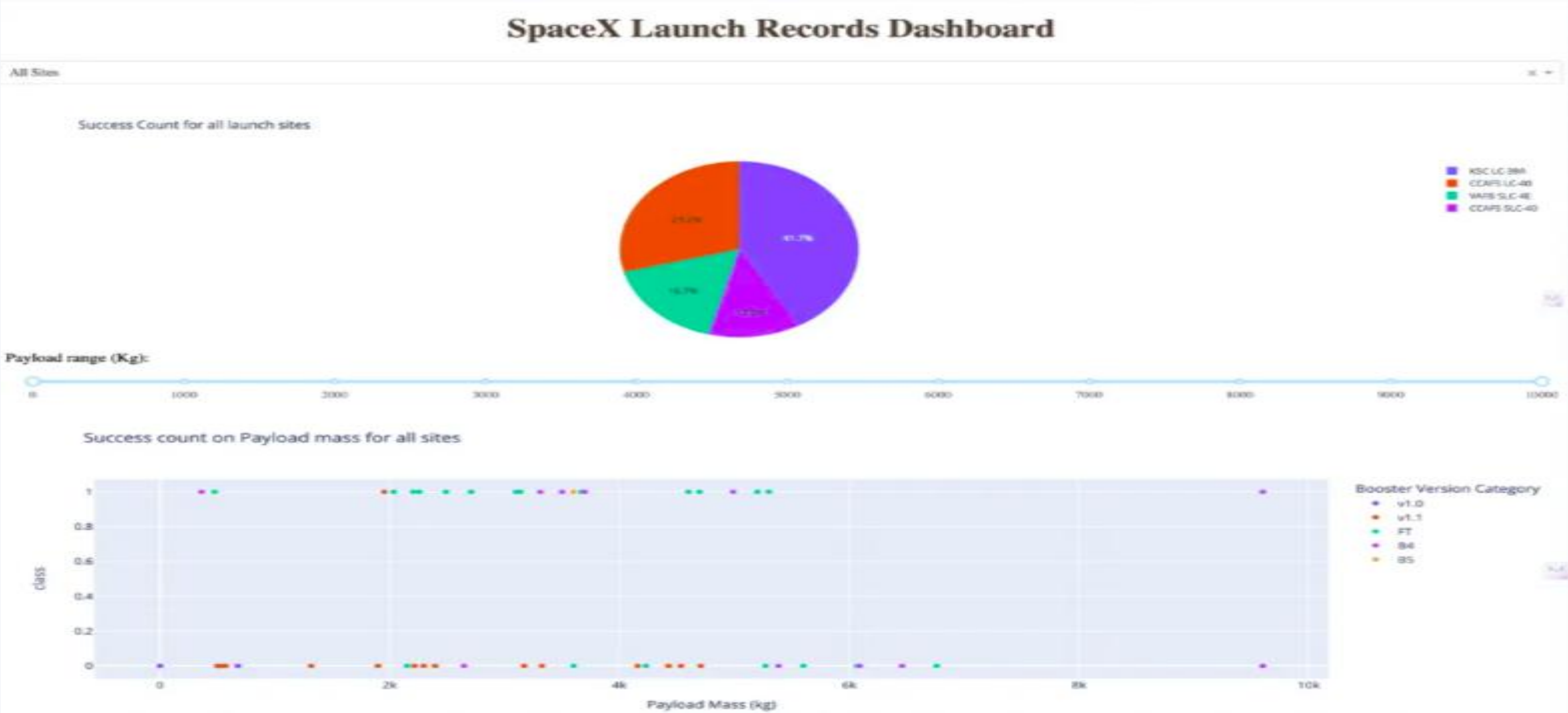
- https://github.com/agusvicgim/Final-IBM-data-course/blob/main/lab_jupyter_launch_site_location%20(1).ipynb
- Use this link to visualize the maps, as at gethub will now appear, you need to use the lab: https://labs.cognitiveclass.ai/v2/tools/jupyterlite?ulid=ulid-f6ab70283e72c16d2ae5741ccd08bc625cf53bda

Build a Dashboard with Plotly Dash

- We build an interactive Dash with Plotly Dash
- We created a pie chart as it is the best type to visualize the different launches and a Scatter plot for different booster versions with the relation of outcome and mass (kg)

- Please find the code below. The access to the chart does not allow me to add it to github but you can see the correct code, I will add pictures below: **https://github.com/agusvicgim/Final-IBM-data-course/blob/main/Build%20an%20Interactive%20Dashboard%20with%20Ploty%20Dash**

Build a Dashboard with Plotly Dash



Predictive Analysis (Classification)

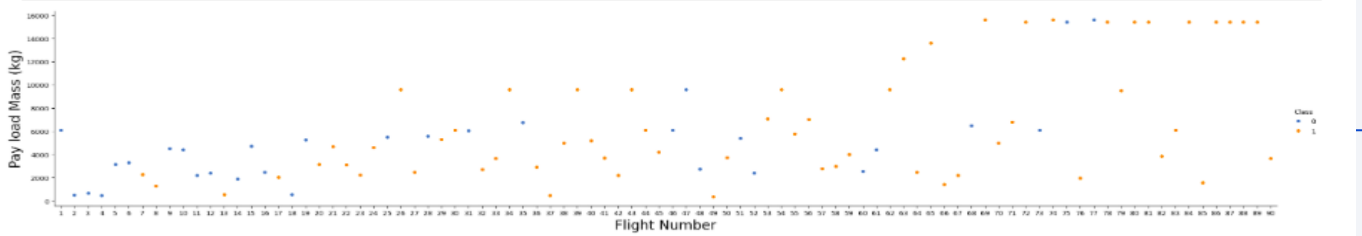
- We created a NumPy array and standarize the data, split the data for test and training and performed: logistic regression, support machine vector object, decision tree classifier, KNN and the accuracy for each one
- We obtain the best accuracy of 0.857 with decision tree
- https://github.com/agusvicgim/Final-IBM-data-course/blob/main/SpaceX_Machine%20Learning%20Prediction_Part_5%20(1).ipynb

Results

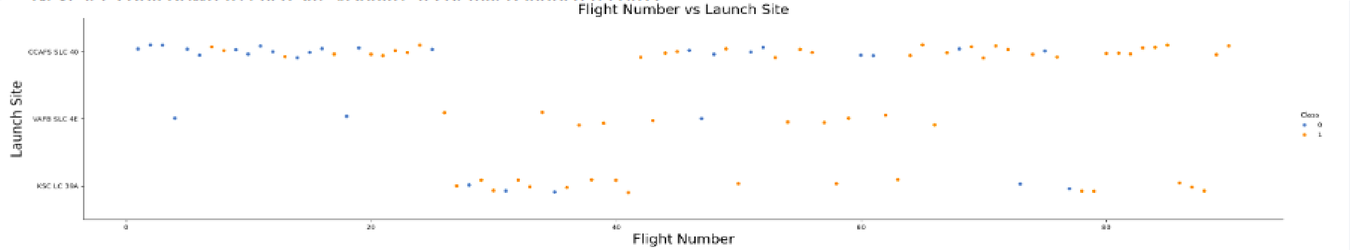
Exploratory data analysis results

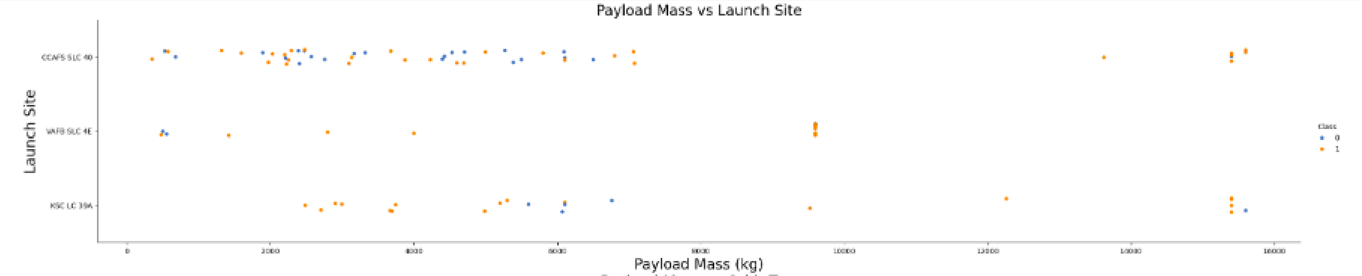
- There is a positive correlation of successful launches and years
- Higher payload mass did not affect to the latest launches
- KSC CL A39 is the location with the best successful launches rate
- Launches are close to the coast for safety reasons

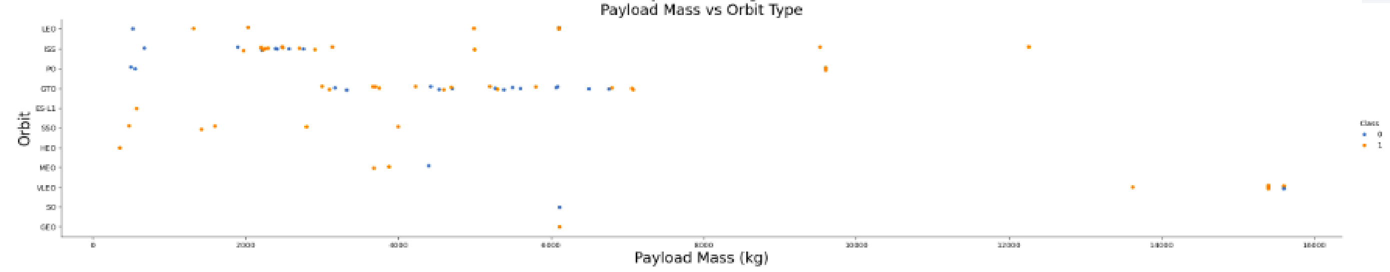
Interactive analytics demo in screenshots

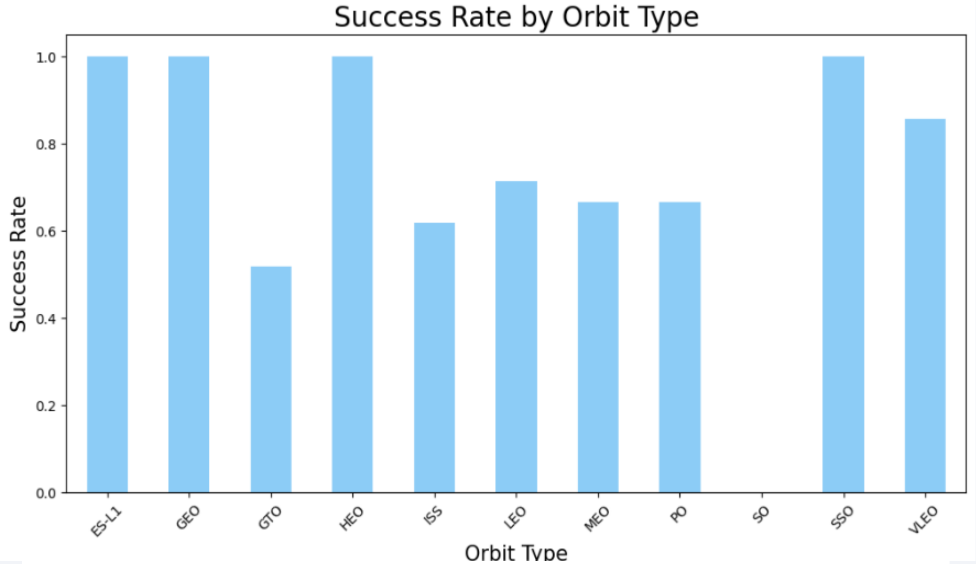


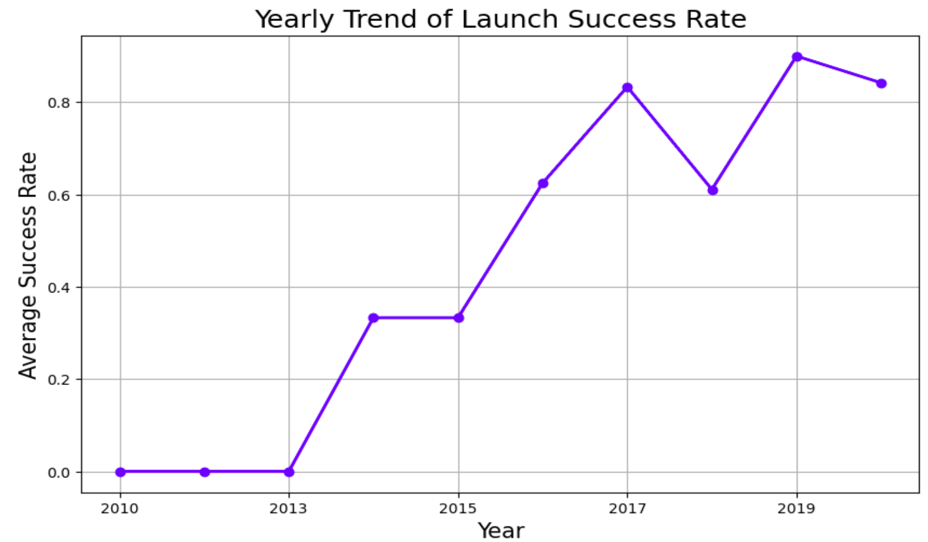
Next, let's drill down to each site visualize its detailed launch records





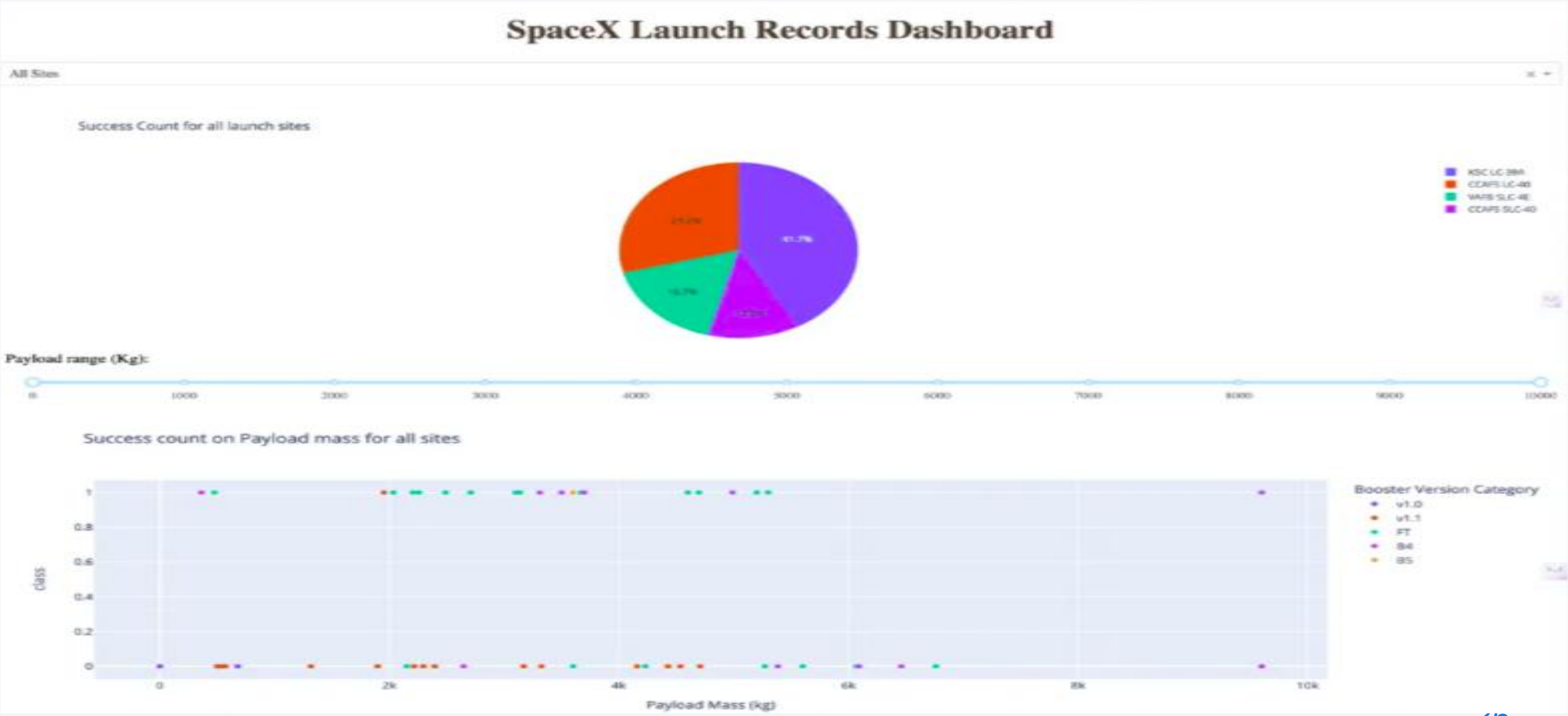






Interactive analytics demo in screenshots

Interactive analytics demo in screenshots



Results

- Best model is decision tree with an accuracy of 0.857.

Find the method performs best:

```
# List the accuracy scores of all models
model_accuracies = {
    'Logistic Regression': 0.8333333333333334,  # Replace with the actual accuracy of logreg_cv
    'SVM': 0.8482142857142856,                  # Replace with the actual accuracy of svm_cv
    'Decision Tree': 0.8571428571428571,        # Replace with the actual accuracy of tree_cv
    'KNN': accuracy_knn_test                    # The accuracy calculated for knn_cv
}

# Find the model with the best performance
best_model = max(model_accuracies, key=model_accuracies.get)
best_accuracy = model_accuracies[best_model]

# Output the best performing model
print(f"The best performing model is {best_model} with an accuracy of {best_accuracy:.4f}")
```

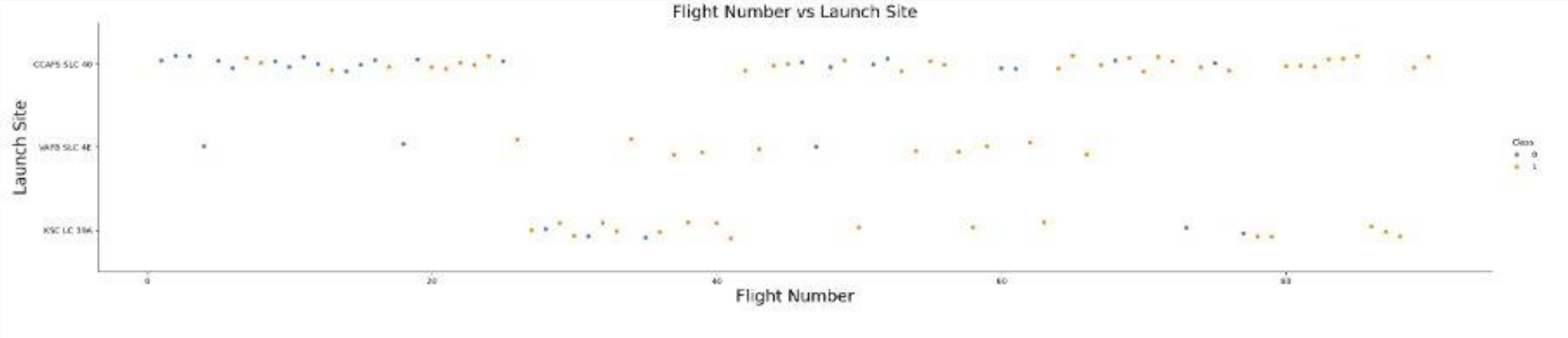
```
ne best performing model is Decision Tree with an accuracy of 0.8571
```

Section 2

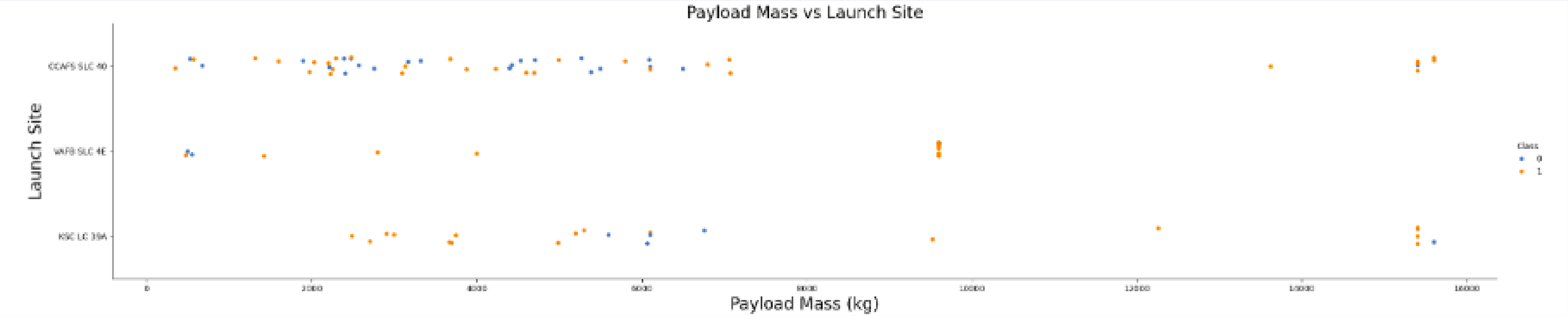
Insights drawn from EDA

Flight Number vs. Launch Site

- Success rate improve over time, we cannot conclude a place is better but the CCAP5 has a better success rate but as well the most number of launches



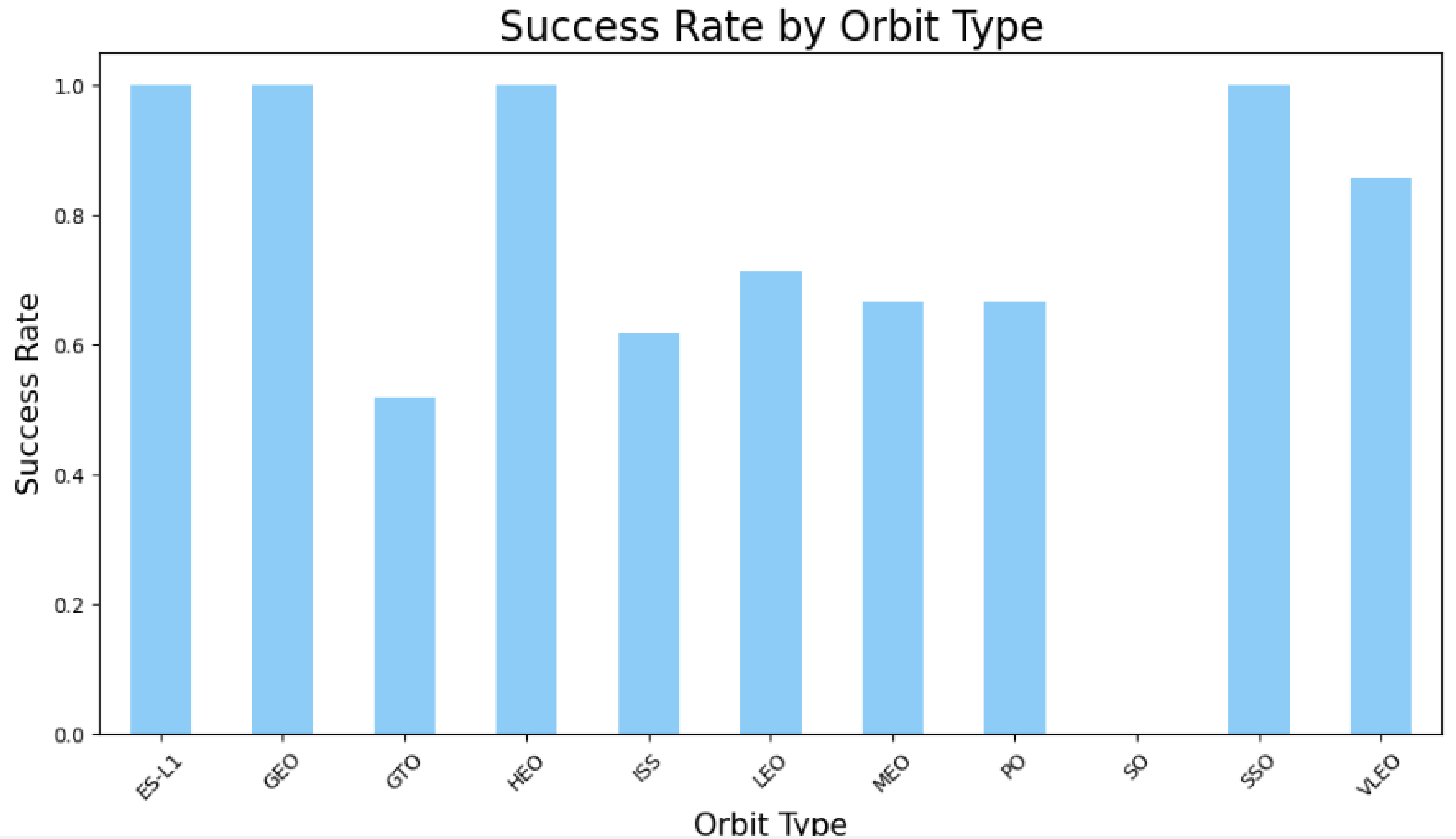
Payload vs. Launch Site



- The higher the mass, the best success rate.

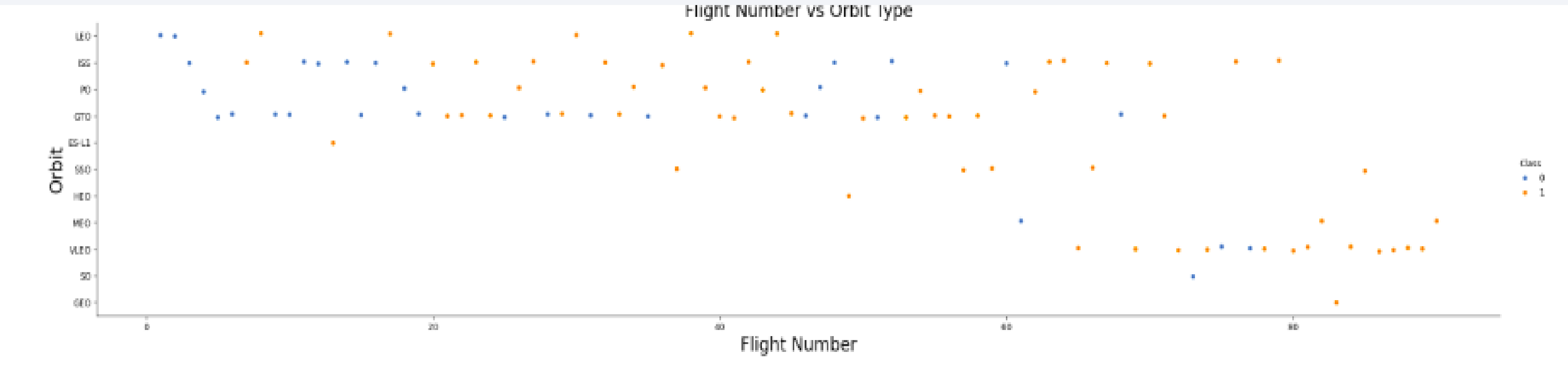
Success Rate vs. Orbit Type

- ESL1, GEO, HEO and SSO are the most successful Orbits



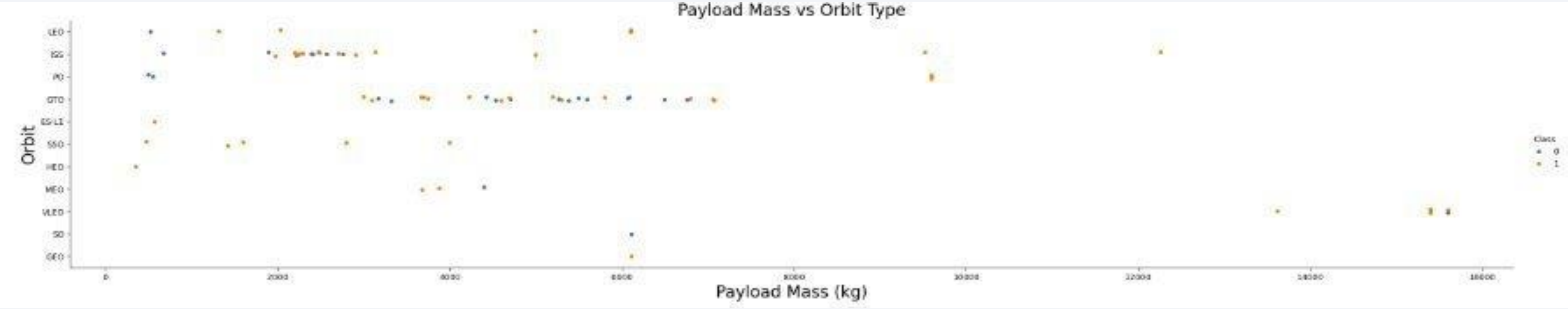
Flight Number vs. Orbit Type

- Leo, ISS, PO, GTO are the most successful but this change to VLEO with number of flights



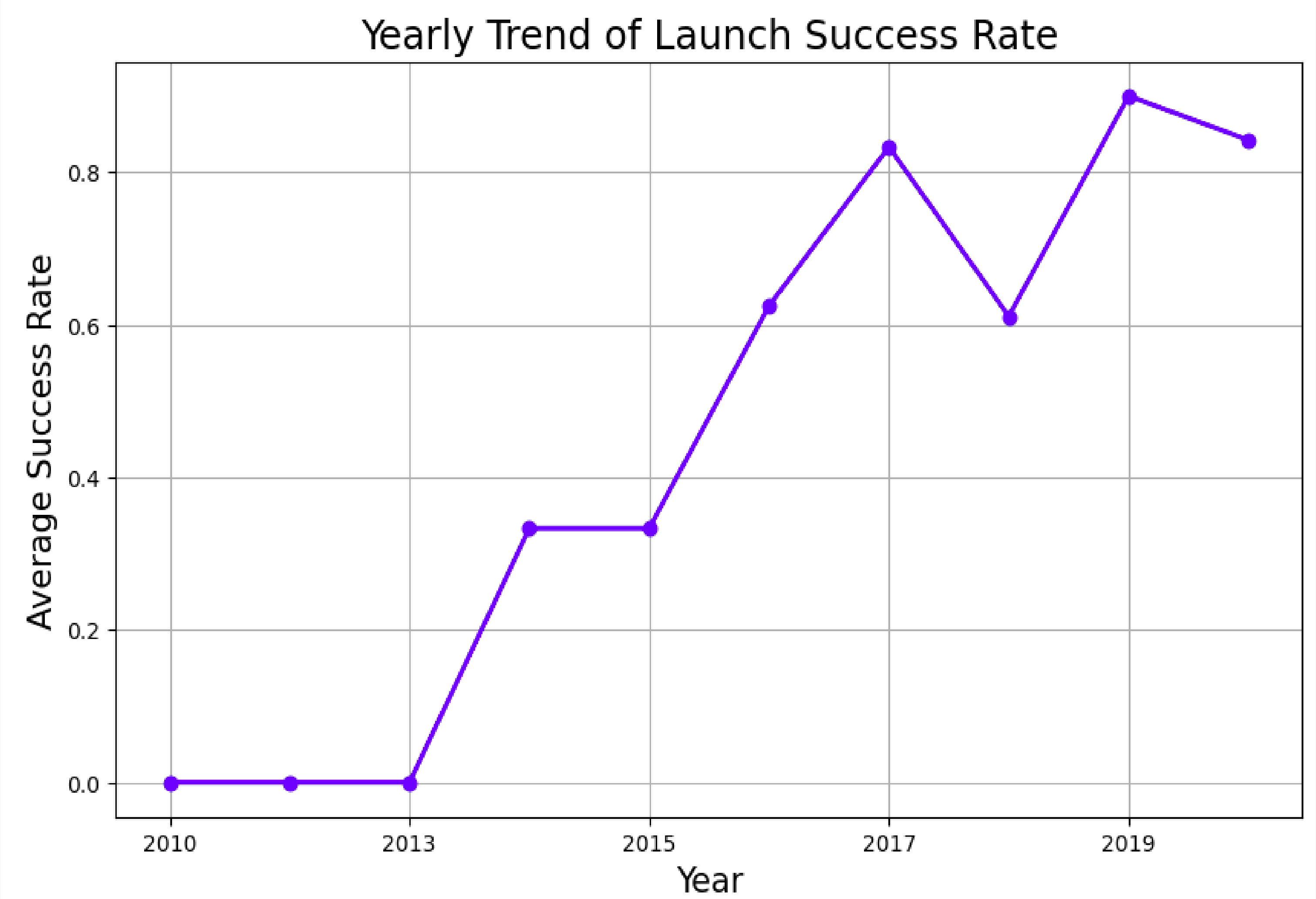
Payload vs. Orbit Type

- LEO, PO, ISS are the most successful with heavy playload



Launch Success Yearly Trend

- The success rate keep increasing since 2013 to 2020



All Launch Site Names

- Using DISTINCT in the query, it will give the different sites of the launches.

Task 1

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

```
# To execute SQL in Jupyter, use the following:
%sql SELECT DISTINCT "Launch_Site" FROM SPACEXTABLE;
```

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

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

Launch Site Names Begin with 'CCA'

- Using LIKE CCA% will tell us the records starting with CCA and LIMIT 5 give us only the first 5 records.

Task 2

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

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

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

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_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

- Using AS give us the result as the name we want and WHERE limits to only NADA (CRS) the results

Task 3

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

```
%sql SELECT SUM("Payload_Mass__kg_") AS Total_Payload_Mass FROM SPACEXTABLE WHERE "Customer" = 'NASA (CRS)';
```

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

Total_Payload_Mass
45596

Average Payload Mass by F9 v1.1

Task 4

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

```
%sql SELECT AVG("Payload_Mass__kg_") AS Avg_Payload_Mass FROM SPACEXTABLE WHERE "Booster_Version" = 'F9 v1.1';
```

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

Avg_Payload_Mass
2928.4

- AVG give us the average result (WHERE =) retrieve the data of F9 v1.1

First Successful Ground Landing Date

Task 5

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

Hint:Use min function

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

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

First_Successful_Landing_Date
2015-12-22

- Using MIN will give us the 1st, WHERE = success (ground pad) will give us the successful landing

Successful Drone Ship Landing with Payload 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

```
%sql SELECT DISTINCT "Booster_Version" FROM SPACEXTABLE WHERE "Landing_Outcome" = 'Success (drone ship)' AND "Payload_Mass_
```

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

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

- Using DISTINCT and AND will sort for the booster versions >4000 and < 6000 will sort for only version with those masses.

Total Number of Successful and Failure Mission Outcomes

Task 7

List the total number of successful and failure mission outcomes

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

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

Mission_Outcome	Total_Count
Failure (in flight)	1
Success	98
Success	1
Success (payload status unclear)	1

- Using COUNT will tell us the amount of mission

Boosters Carried Maximum Payload

- Using a subquery SELECT MAX will retrieve the data with the maximum payload for booster version

Task 8

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

```
%sql SELECT "Booster_Version" FROM SPACEXTABLE WHERE "Payload_Mass__kg_" = (SELECT MAX("Payload_Mass__kg_") FROM SPACEXTABLI
```

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

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

2015 Launch Records

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.

```
%sql SELECT substr(Date, 6, 2) AS Month, "Booster_Version", "Launch_Site" FROM SPACEXTABLE WHERE substr(Date, 0, 5) = '2015
```

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

Month	Booster_Version	Launch_Site
01	F9 v1.1 B1012	CCAFS LC-40
04	F9 v1.1 B1015	CCAFS LC-40

- Using substr(Date, 6,2) as month to get the months and substr(Date,0,5)='2015' for year.

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

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.

```
%sql SELECT "Landing_Outcome", COUNT(*) AS Outcome_Count FROM SPACEXTABLE WHERE "Date" BETWEEN '2010-06-04' AND '2017-03-20
```

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

Landing_Outcome	Outcome_Count
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

Using COUNT GROUP BY and ORDER BY DESC, we can 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

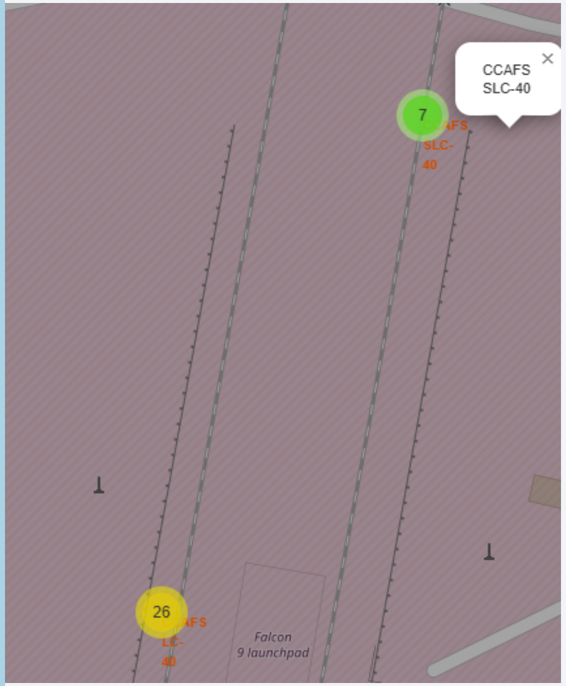
Section 3

Launch Sites Proximities Analysis

<Folium Map Screenshot 1>

All launch sites on a map

<Folium Map Screenshot 2>



Success/failed launches for each site on the map

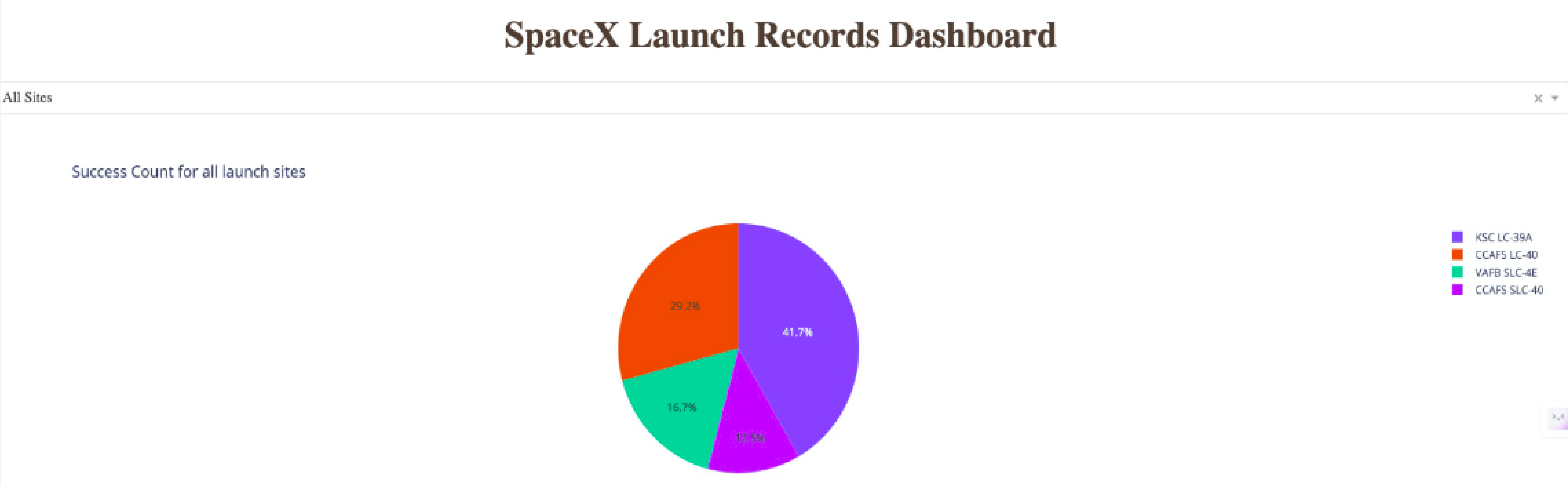
<Folium Map Screenshot 3>

Distances between a launch site to its proximities

Section 4

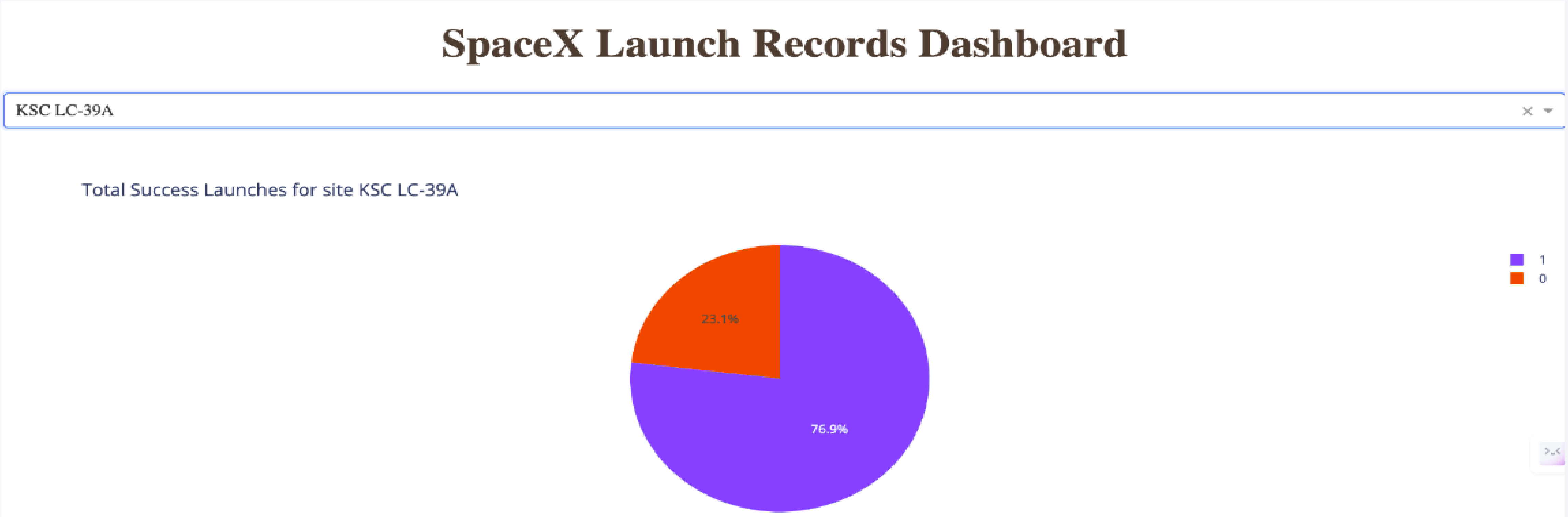
Build a Dashboard with Plotly Dash

SpaceX Launch Records Dashboard



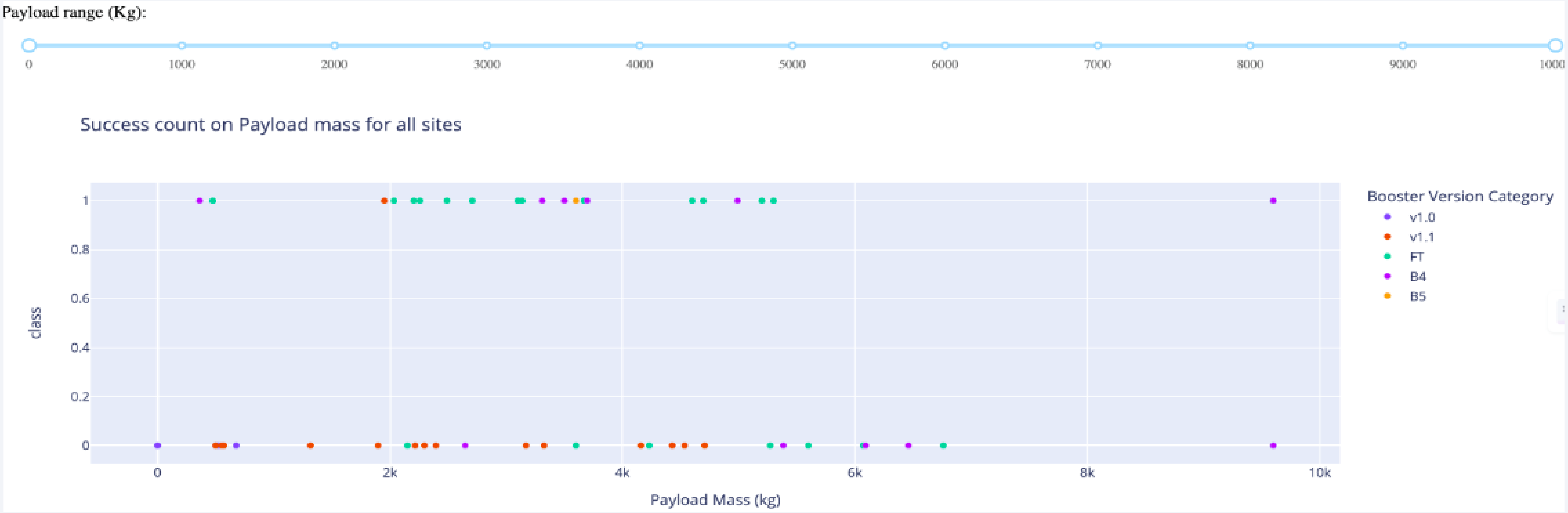
- KSC LC 39A is the most successful launch site with 41.7%

<Dashboard Screenshot 2>



KSC LC 39A has 76.9% success launches and 23.1% failure

<Dashboard Screenshot 3>



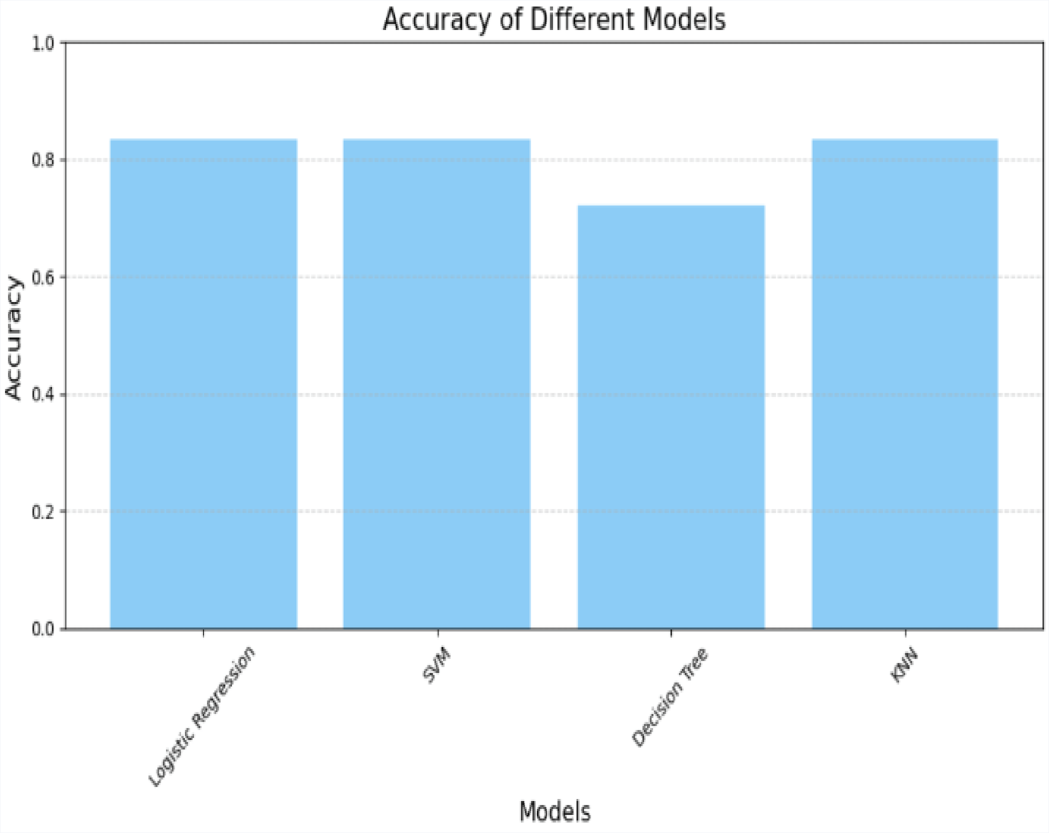
Payload rate most successful is between 2-4k kg followed by 4-6k kg.
Booster version FT has the most successful launches (green)

Section 5

Predictive Analysis (Classification)

Classification Accuracy

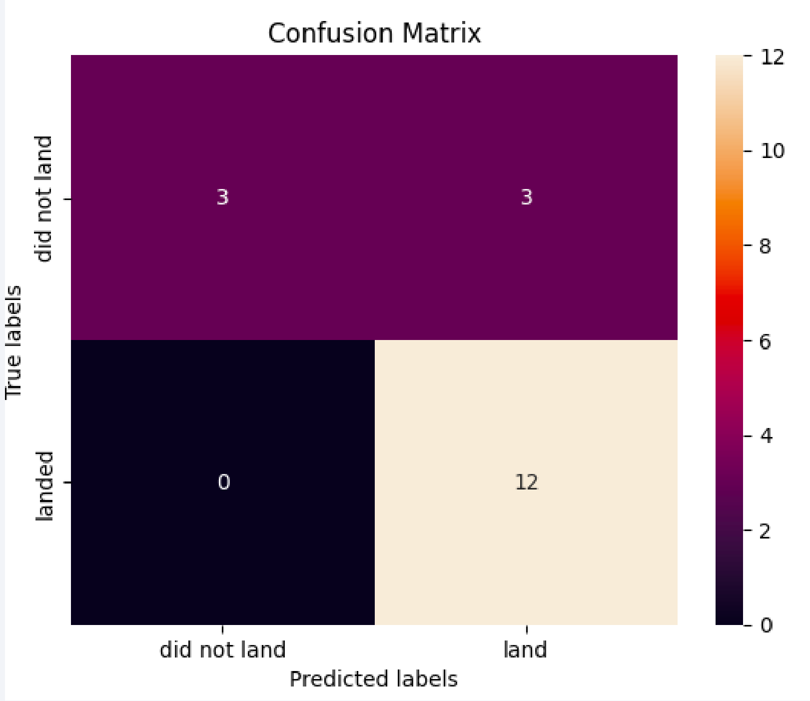
- model_accuracies = {
- 'Logistic Regression': 0.8333333333333334, *logreg_cv*
- 'SVM': 0.8482142857142856, *svm_cv*
- 'Decision Tree': 0.8571428571428571, *tree_cv*
- 'KNN': accuracy_knn_test *# The accuracy calculated for knn_cv*
- }

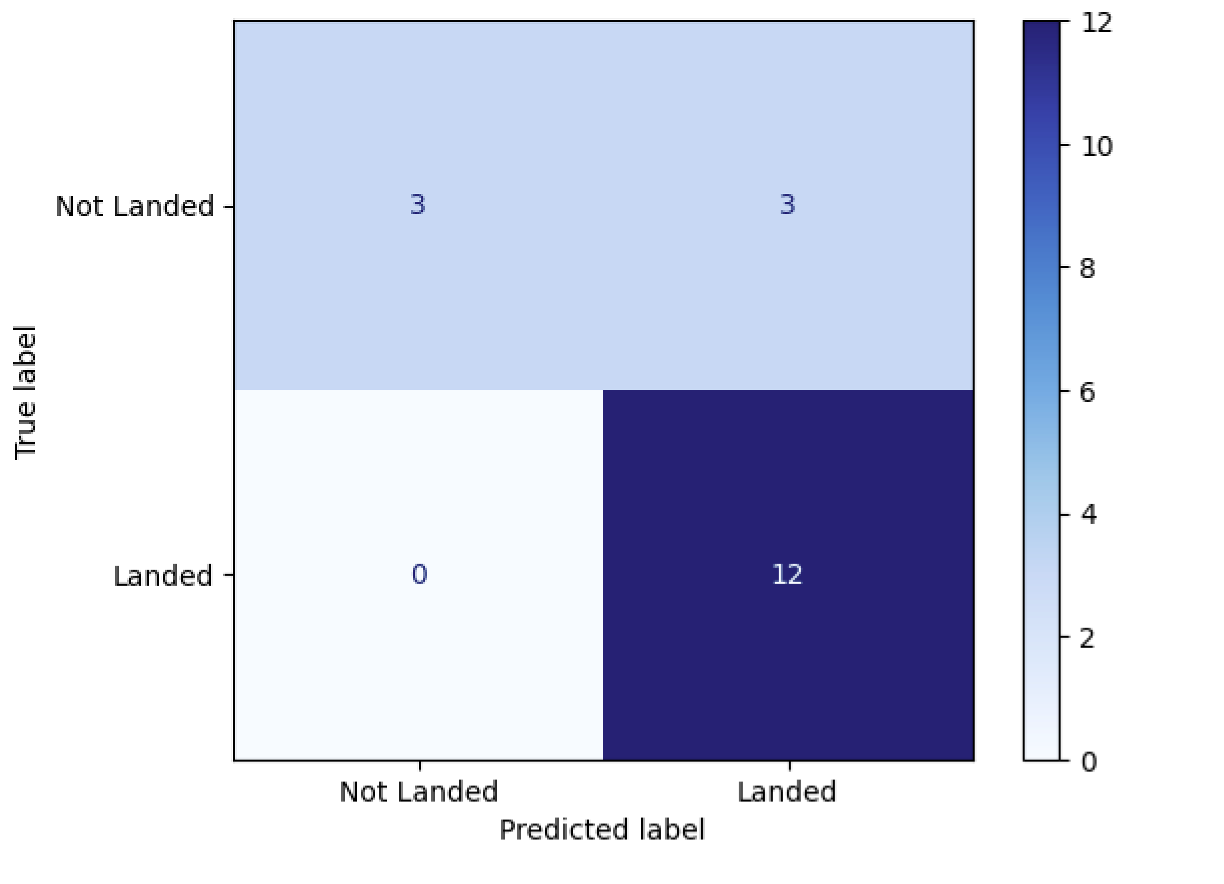


Confusion Matrix

- Decision tree is the best performing model as has the highest accuracy

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






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

```
tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', 'max_depth': 16, '
f': 4, 'min_samples_split': 5, 'splitter': 'random'}
accuracy : 0.8732142857142856
```

Conclusions

- 39A is the locaiton with the best successful rate
- ESL1, GEO, HEO and SSO are the most successful Orbits
- Increses the success with years
- Increase payload increases the success rate
- The best model performance is the decision tree due to the accuracy

Appendix

- Include any relevant assets like Python code snippets, SQL queries, charts, Notebook outputs, or data sets that you may have created during this project

Thank you!