IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

J Aquino – November 2023

Table of contents

1

Executive Summary

Outline
Background and
Problem Statement

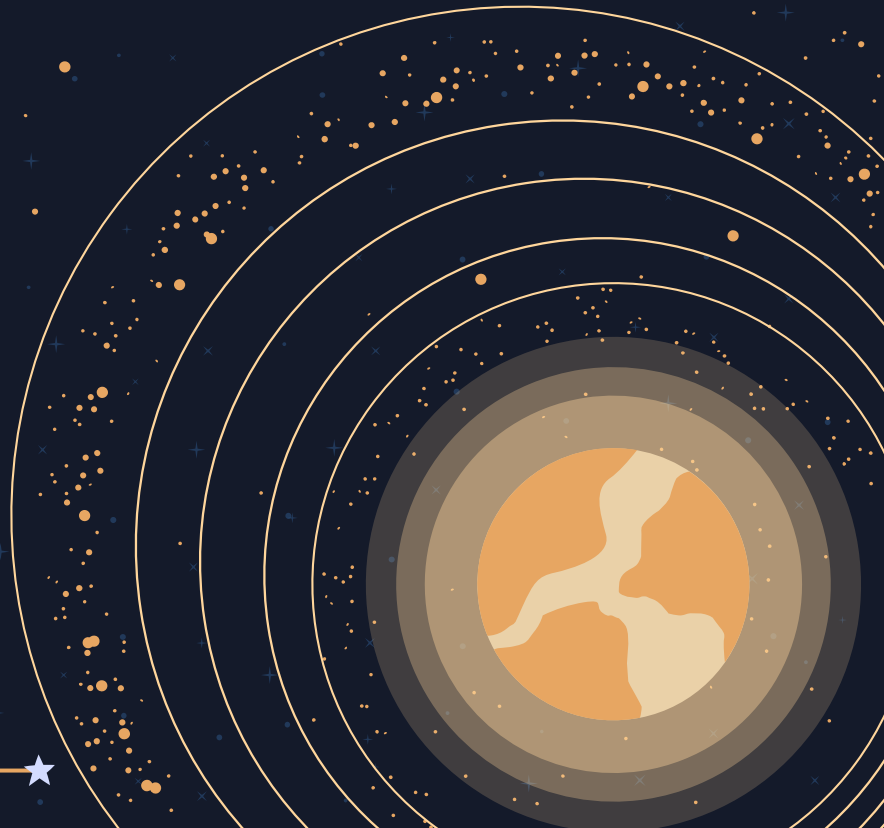
Executive Summary

The following methodologies were used…

- Data collection using API call or webscraping
- Data wrangling
- EDA with SQL and visualisation
- Building an interactive map with Folium
- Building a dashboard with Plotly Dash
- Predictive analytics (classification)

To obtain the results below…

- Exploratory data analysis
- Visual analytics
- Predictive analytics
- Conclusions



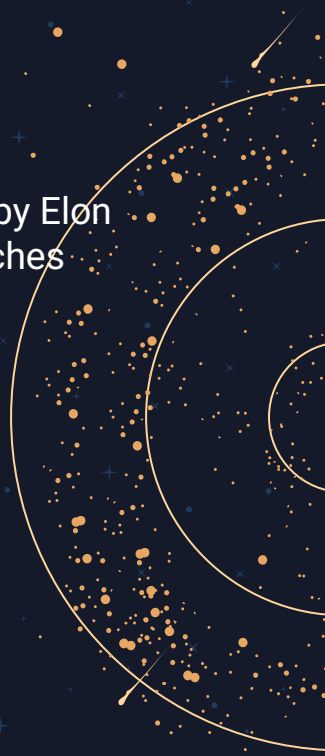
Introduction

Project Background and Context

Our client SpaceY, would like to bid against SpaceX for a rocket launch. SpaceX, founded by Elon Musk, is currently the leading company in this space. They advertise Falcon 9 rocket launches on their website at a cost of $62million. Other companies provide their cost upwards of $165million each. Much of the savings is because SpaceX can reuse the first stage.

To help Space Y, we have been tasked to answer the following…

1. Can we predict if the first landing will be successful?
2. What factors influence a successful landing?
3. Is there a machine learning model we can apply to predict?



II

Data Methodology

Data Collection via API
Webscraping
Wrangling

Data Collection via API

Github notebook link: https://github.com/danyzephyr/IBM-Final-Capstone-Project/blob/main/Week%201.1%20-%20SpaceX%20Data%20Collection%20-%20API%20request_JA.ipynb

1. Call the API

Request and parse the SpaceX launch data using GET request. The file downloaded from the API is a JSON file.

```
In [6]: spacex_url="https://api.spacexdata.com/v4/launches/past"

In [7]: response = requests.get(spacex_url)
```

```
In [28]: # Use json_normalize meethod to convert the json result into a dataframe
         data = pd.read_json(static_json_url)
         pd.json_normalize(data)
```

2. Convert to DF

Use .json_normalize() to decode the JSON file and turn it to a Pandas dataframe.

3. Normalise

Understand the downloaded dataset. Construct the dataset by pulling your required fields. Normalise where required, e.g. date formats, restrict date range or remove any redundant data.

```
In [30]: # 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
         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)]
```

Data Collection via Webscraping

Github notebook link: https://github.com/danyzephyr/IBM-Final-Capstone-Project/blob/main/Week%201.2%20-%20SpaceX%20Webscraping_JA.ipynb

1. Extract data from the website

Falcon 9 launch records are available from an HTML table in Wikipedia. BeautifulSoup, a Python feature can be used to parse the data from the website.

2. Construct dataset

Find the 'th' (table headers) and tables from the website and assign these to a variable to start constructing the dataset. The parsed HTML tables are used to create an empty dictionary with keys.

3. Convert to DF

Convert the empty dictionary created in step # 2 to a Pandas dataframe.

NB. Screenshots for this are provided on the next slide

Data Collection via Webscraping

Github notebook link: https://github.com/danyzephyr/IBM-Final-Capstone-Project/blob/main/Week%201.2%20-%20SpaceX%20Webscraping_JA.ipynb

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.

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

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

```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
falconobj = BeautifulSoup(falcondata,"html5lib")
```

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

```
print(falconobj.prettify()) # extra code ran to display the HTML in a nested structure

# Use soup.title attribute
tag_object=falconobj.title
print(tag_object)
```

```
<title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

```
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'] = []
```

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

Data Wrangling

Github notebook link: https://github.com/danyzephyr/IBM-Final-Capstone-Project/blob/main/Week%201.3%20-%20SpaceX%20Data%20Wrangling_JA.ipynb

1. Identify missing or NaN values

In order to properly use the dataset, the data types must be understood, along with any missing or NaN values.

2. Perform required calculations

Some calculations were performed on the dataset:

- value_counts for # of launches per site, # of orbit occurrences, # of different landing outcomes
- Successful vs. bad landings

3. Add calculated variables to dataframe

Any calculated variables from step # 2 are added to the created dataframe for future querying.

NB. Screenshots for this are provided on the next slide

Data Wrangling

Github notebook link: https://github.com/danyzephyr/IBM-Final-Capstone-Project/blob/main/Week%201.3%20-%20SpaceX%20Data%20Wrangling_JA.ipynb

In [3]:
```
df.isnull().sum()/len(df)*100
```

In [4]:
```
df.dtypes
```

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

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

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

```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise

landing_class = [0 if outcome in bad_outcomes else 1 for outcome in df['Outcome']]
landing_class
```

In [15]:
```
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
bad_outcomes
```

Out[15]: {'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}

EDA

Data Visualisation
SQL
Folium
Plotly Dash

EDA with Data Visualisation

To help predict the parameters for a successful landing, the following charts were created to compare the variables against each other. The outputs are on **Section V – Results**. The overview of charts that were plotted are below:

1. Scatter chart on Flight Number vs. Payload
2. Scatter chart on Flight Number vs. Launch Site
3. Scatter chart on Flight Number vs. Orbit Type
4. Scatter chart on Payload vs. Orbit Type
5. Scatter chart on Payload vs. Launch Site
6. Bar chart to represent Success Rate per Orbit Type
7. Line chart to represent Yearly Success trend

Github notebook link: https://github.com/danyzephyr/IBM-Final-Capstone-Project/blob/main/Week%202.2%20-%20SpaceX%20EDA%20with%20Visualization_JA.ipynb

EDA with SQL

To help predict the parameters for a successful landing, the following queries were ran against the database table. The outputs are on **Section V – Results**. The overview of the SQL queries are below:

1. Identify all unique launch sites
2. Identify all launch sites beginning with 'CCA'
3. Calculate the total payload mass carried by boosters launched by NASA (CRS)
4. Calculate average payload mass carried by booster version F9 v1.1
5. Identify earliest date when first successful landing in ground pad was achieved
6. List all boosters which have success in drone ship with payload mass between 4000 to 6000
7. Calculate total number of successful vs. non-successful outcomes
8. List all boosters which have the max payload mass
9. Display all months where landing failed for drone ships in 2015
10. Rank the count of landing outcomes between 2010-06-04 and 2017-03-20

Github notebook link: https://github.com/danyzephyr/IBM-Final-Capstone-Project/blob/main/Week%202.1%20-%20SpaceX%20EDA%20with%20SQL_JA.pdf (a PDF file has been uploaded to GitHub as the Python file kept encountering error as Invalid Notebook. The screenshots prove that the SQL queries were running correctly in the kernel.)

Interactive Maps with Folium

A Folium site map has been created in order to geographically explore the physical location of the launch sites. This will help to determine **whether the geographic location contributes to a successful landing**. The following objects have been added to the site map.

Markers were placed on the map using the mentioned sites' latitudes and longitudes

1. A **blue** circle to highlight the NASA Johnson Space centre
2. **Orange** circles for all launch sites with name labels
3. **Green** and **red** pop-up markers to differentiate between successful (green) and failed (red) landings at each launch site

Plot lines to highlight the proximity of the launch sites to some landmarks

1. Plot lines with distance displayed to the nearest coast or highway
2. Plot lines with the distance displayed to the nearest city, airport or railway station

Github notebook link: https://github.com/danyzephyr/IBM-Final-Capstone-Project/blob/main/Week%203.1%20-%20SpaceX%20Visual%20Analytics%20with%20Folium_JA.ipynb (NB. Github can only show static data so the Folium maps are not shown properly. In order to view the Folium maps and interact with them, you have to type the notebook link and view using nbviewer.org.)

Interactive Dashboard with Plotly Dash

A dashboard has been created to **allow the client to view dynamic data on successful vs. failed landings per launch site**. It also provides a **scatter chart on payload mass which can be analyzed to check whether it contributes to a successful landing**. The following features / charts are in the dashboard. The dashboard can be further enhanced should the client require it.

1. Dropdown list showing all unique launch sites as well as ‘All sites’
2. Pie chart showing the % rate of successful vs. failed launches of the selected dropdown option
3. Range slider for payload mass range to allow users to select the payload range
4. Scatter chart to show the correlation between payload and success rate by booster version

Github notebook link: https://github.com/danyzephyr/IBM-Final-Capstone-Project/blob/main/Week%203.2%20-%20SpaceX_Dash_App_JA.py

IV

Predictive Analytics

Logistic Regression
SVM
Decision Tree
KNN

Predictive Analysis (Classification)

Github notebook link: https://github.com/danyzephyr/IBM-Final-Capstone-Project/blob/main/Week%204%20-%20SpaceX%20Machine%20Learning_JA.ipynb

1. Preprocess the data

A NumPy array is created from the class column. The data is then standardised by using StandardScaler to fit and transform it.

```
Y = data['Class'].to_numpy()
Y
```

```
array([0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1,
       1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1], dtype=int64)
```

2. Split test and train data

The dataset was split into test and train data via train_test_split, with the overall parameter test size set to 20%.

```
# students get this
transform = preprocessing.StandardScaler()
X = transform.fit_transform(X)
X
```

```
array([[-1.71291154e+00, -1.94814463e-16, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       [-1.67441914e+00, -1.19523159e+00, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       [-1.63592675e+00, -1.16267307e+00, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       ...,
       [ 1.63592675e+00,  1.99100483e+00,  3.49060516e+00, ...,
         1.19684269e+00, -5.17306132e-01,  5.17306132e-01],
       [ 1.67441914e+00,  1.99100483e+00,  1.00389436e+00, ...,
         1.19684269e+00, -5.17306132e-01,  5.17306132e-01],
       [ 1.71291154e+00, -5.19213966e-01, -6.53912840e-01, ...,
        -8.35531692e-01, -5.17306132e-01,  5.17306132e-01]])
```

3. Create, train and determine the best model

4 different models were created: logistic regression, SVM, decision tree and KNN. All 4 models' accuracy scores were calculated using .score on the test data. A confusion matrix was also used to assess each model.

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
```

Creating, Evaluating and Picking the Best Model

1. GridSearchCV

After the array, test and train data have been created, a model object can be created by using GridSearchCV. This is a cross-validation technique used to search and find the optimal combination of hyperparameters of a given model. All these models have been assigned a value of 10.

```
parameters = {'criterion': ['gini', 'entropy'],
     'splitter': ['best', 'random'],
     'max_depth': [4, 4, 4, 4],
     'max_features': ['sqrt'],
     'min_samples_leaf': [1, 2, 4],
     'min_samples_split': [2, 5, 10]}

tree = DecisionTreeClassifier()
```

```
tree_cv = GridSearchCV(tree, parameters, cv=10)
tree_cv.fit(X_train, Y_train)
```

```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [4, 4, 4, 4], 'max_features': ['sqrt'],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'splitter': ['best', 'random']})
```

2. .Score

Calculate the accuracy on the test data using the method .score.

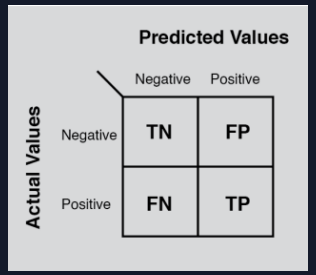
```
svm_score = svm_cv.score(X_test, Y_test)
svm_score
```

```
0.8333333333333334
```

Creating, Evaluating and Picking the Best Model

3. Confusion Matrix

A confusion matrix is a performance evaluation tool. It is another way to test the model's accuracy by presenting different outcomes of the prediction.




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

4. Find the best model

Once steps 1 to 3 have been applied for each model, we can compare each accuracy against each other. Here, we are using a bar chart to visualise.

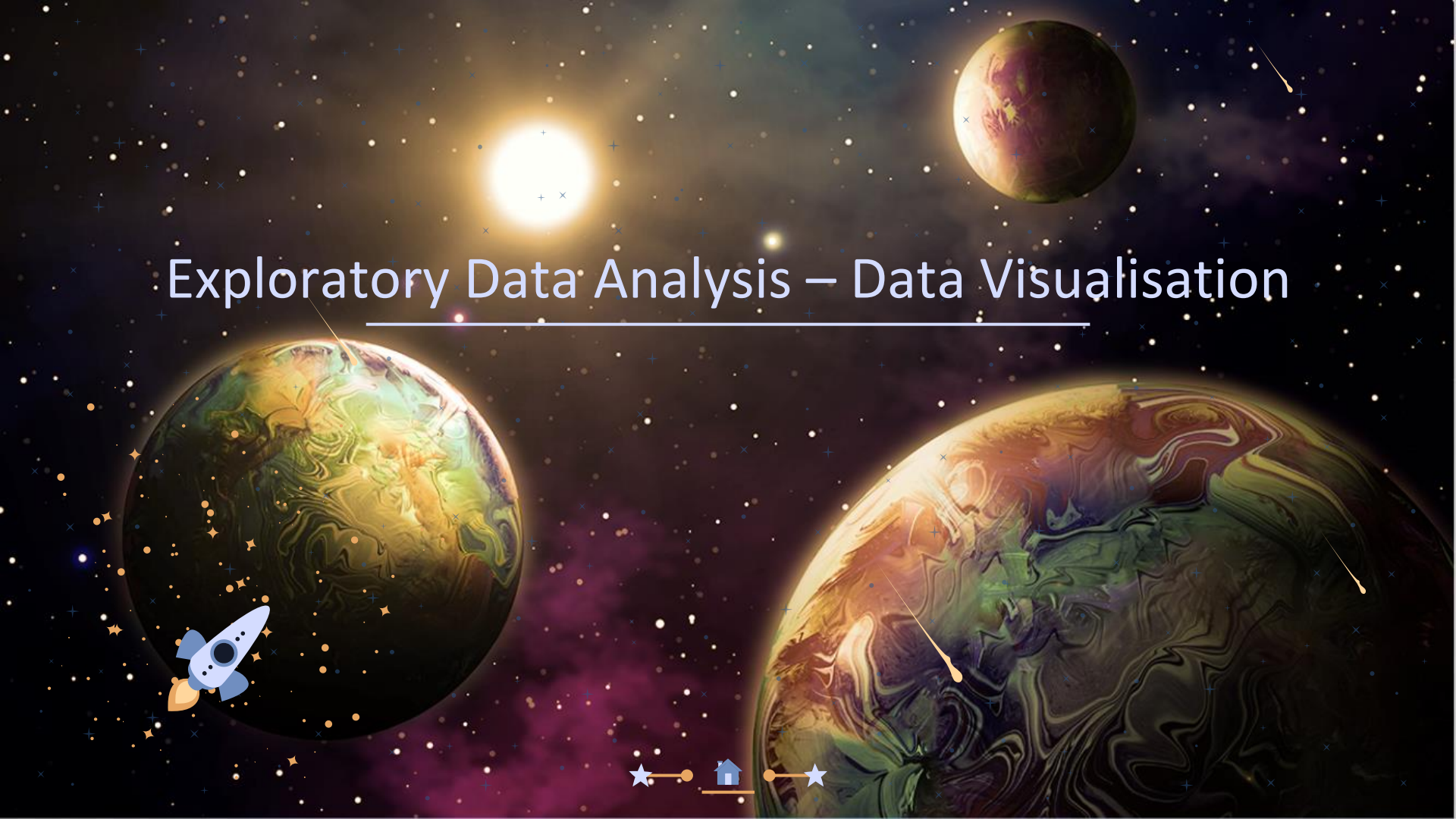
```
X = ['Logistic Regression', 'SVM', 'Decision Tree', 'KNN']
Y = [log_score, svm_score, tree_score, knn_score]
plt.bar(X,Y)
plt.xlabel('ML Model')
plt.ylabel('Accuracy')
plt.title('Accuracy Rating of the ML Models')
plt.show()
```

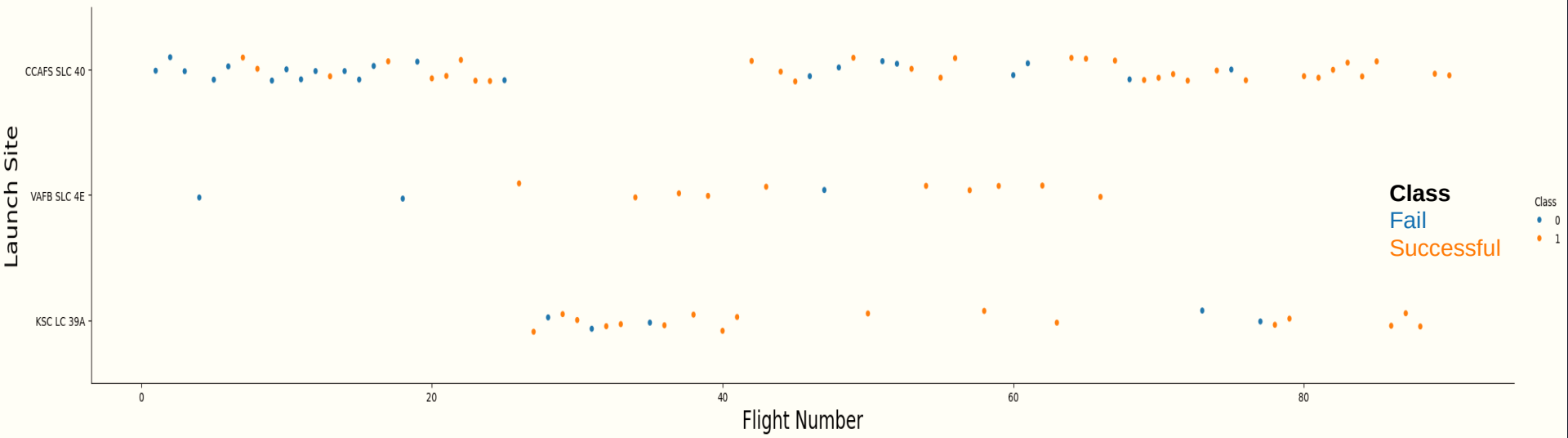
V

Results

Visualisation Charts
SQL Outputs
Folium Maps
Plotly Dashboard
Predictive Analysis

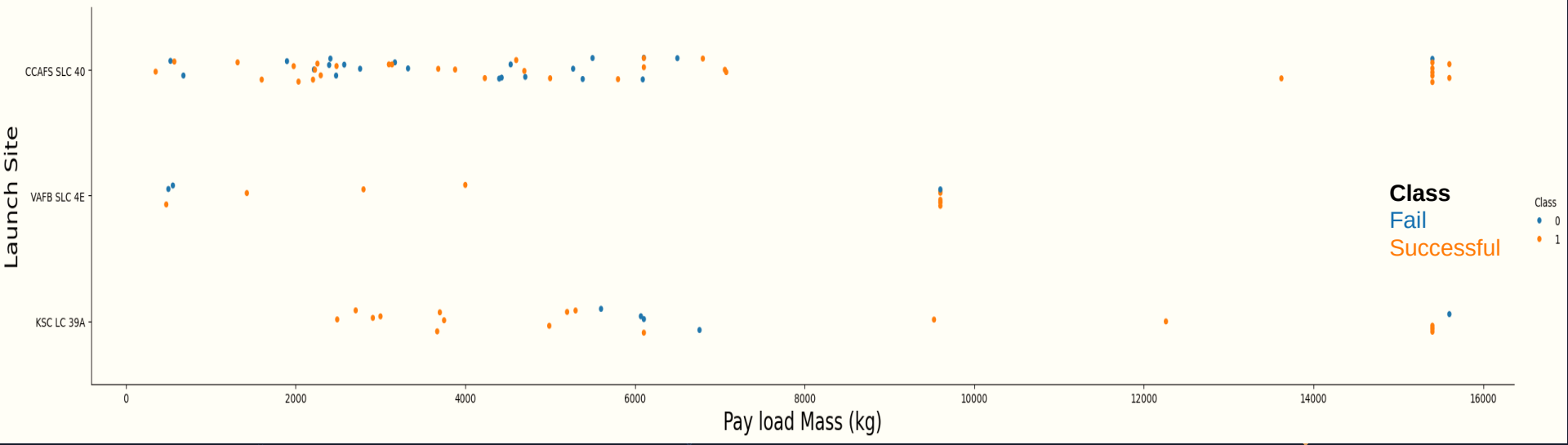
Exploratory Data Analysis – Data Visualisation





1. Flight Number vs. Launch Site

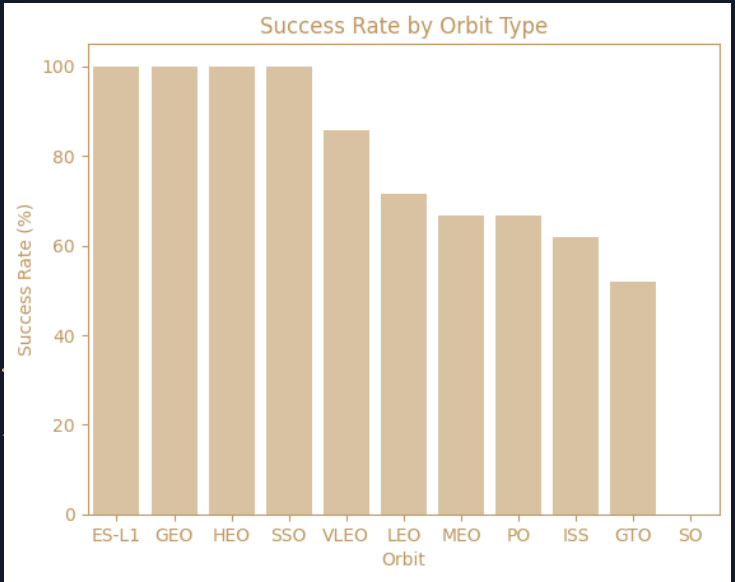
- There was a higher % of failure with earlier flights. Later flights were more successful.
- Out of the 3 launch sites, the least successful was CCAFS SLC-40.
- We can infer that new launches have a higher success rate, probably due to the lessons learned from earlier flights or improvement of technology.



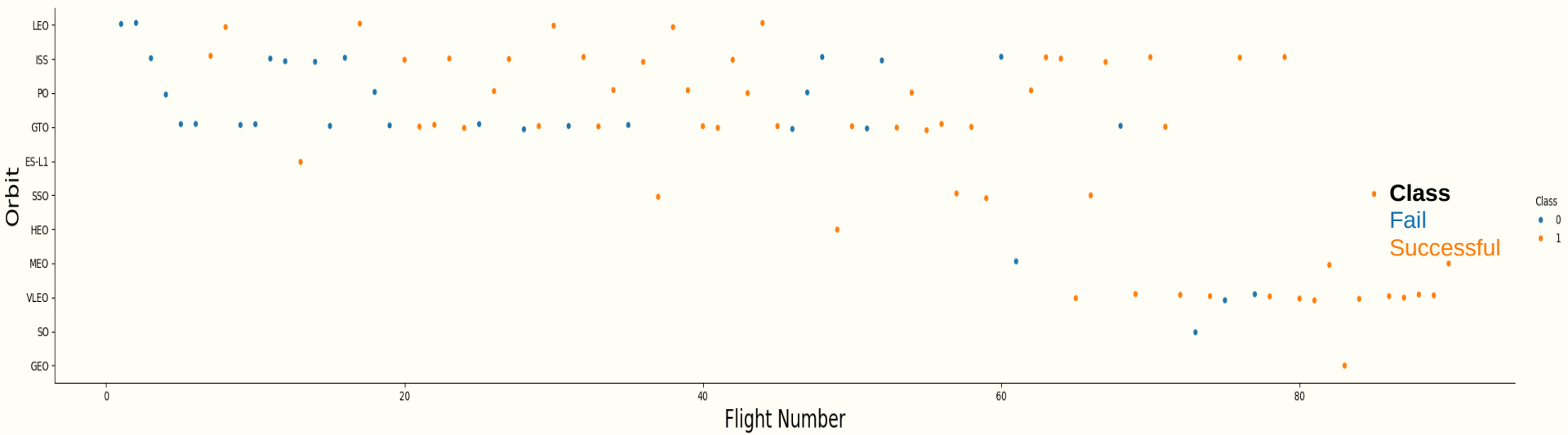
2. Payload vs. Launch Site

- There seems to be a higher success rate with payload masses greater than ~ 9,000 kg.
- VAFB SLC-4E has not launched anything greater than 10,000 kg.
- CCAFS SLC-40 has not launched anything with payload mass between 7,000 kg to 13,500 kg.but seems to be quite successful in launching above 13,500 kg.
- KSC LC-39A has a 100% success rate with launching payload masses below ~ 5,500 kg.

3. Success Rate vs. Orbit Type

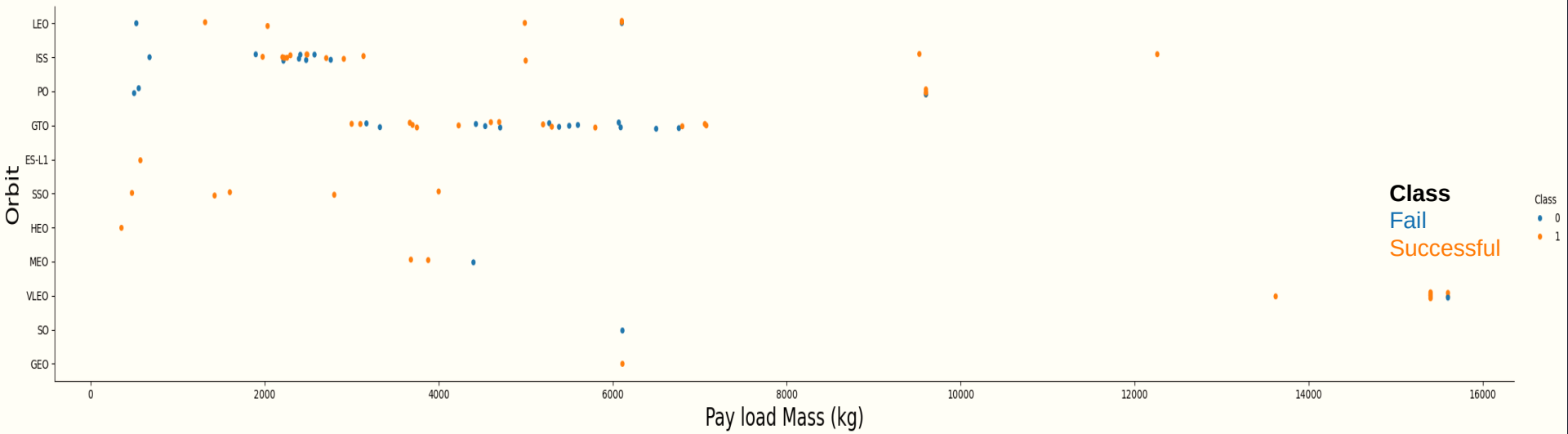


- ES-L1, GEO, HEO and SSO have 100% success rates.
- This is followed by the variable success rates of VLEO, LEO, MEO, ISS and GTO orbits – ranging from 50% to ~ 85%.
- SO orbit has 0% success rate.



4. Flight Number vs. Orbit Type

- As the flight numbers increase, the success rate also seems to increase. The exception is the GTO orbit which still presents mixed results.
- The first observation is the most noticeable in the LEO orbit.



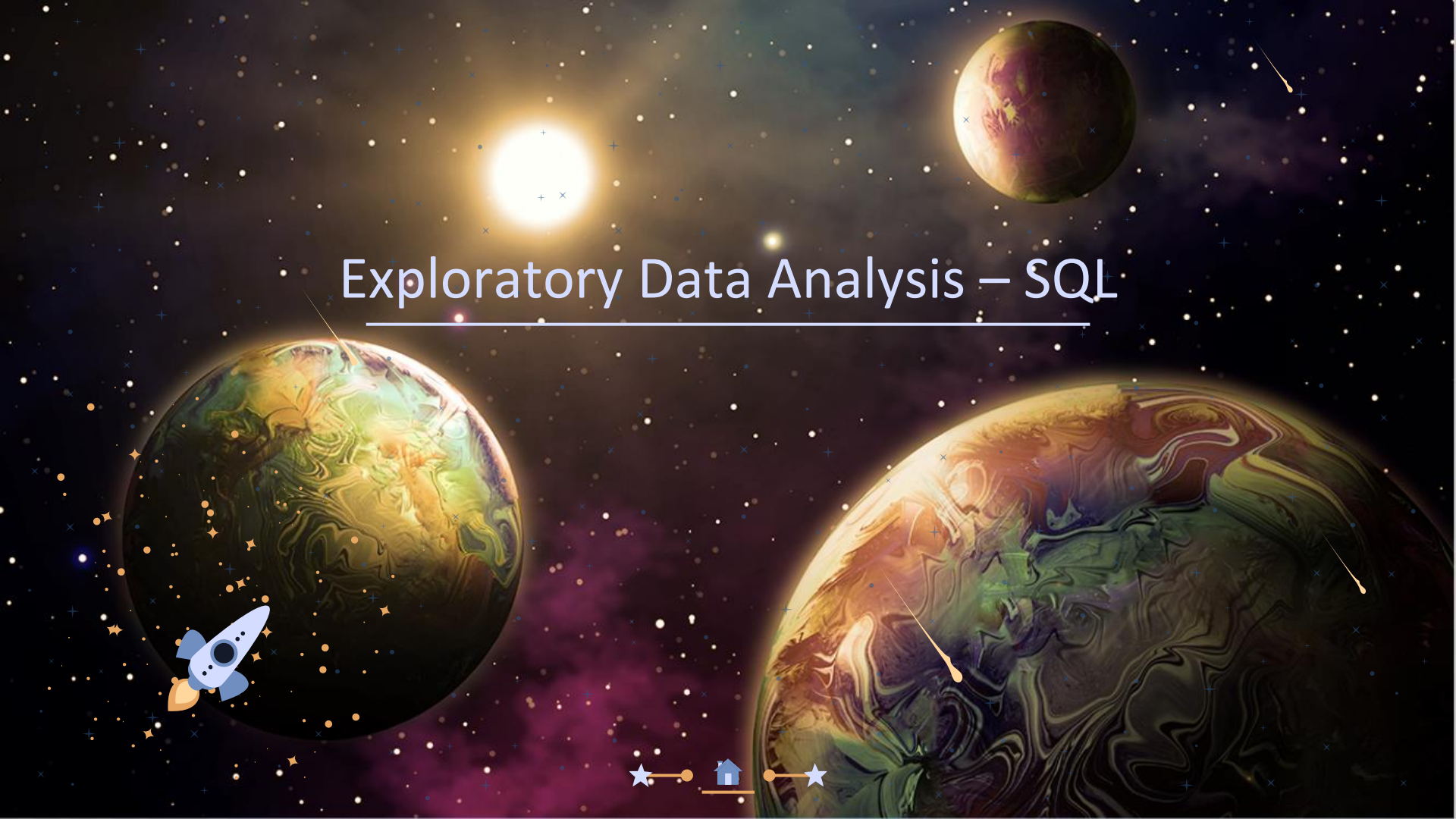
5. Payload vs. Orbit Type

- GTO has a mixed success rate.
- SSO has 100% success rate with launching payload masses at 4,000 kg and below.
- ISS has the most success with launching payload masses at 3,000 kg and above.

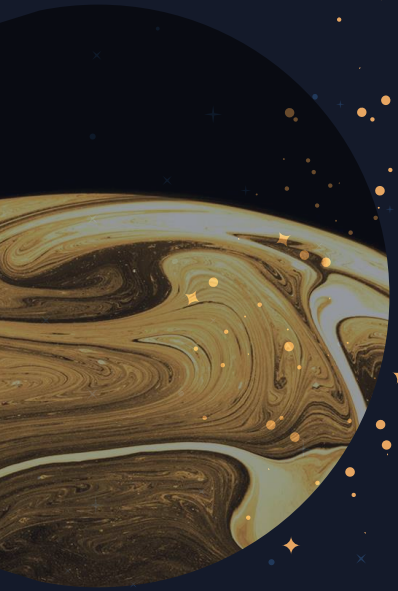
6. Yearly Trend of Launch Outcome

- All launches have failed prior to 2013.
- The success rate started to increase sharply from 2013 onwards.
- There was a steep downturn in 2017 to 2018.
- There is 50% success rate increase from 2014 to 2020.

Exploratory Data Analysis – SQL



1. SQL – all launch sites' names



```
[9]: %%sql

     select distinct(Launch_Site) from SPACEXTABLE order by Launch_Site

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

[9]:

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

DISTINCT is used to display only unique launch site names.

2. SQL – 5 records with names starting with CCA

```
[11]: %%sql

      select * from SPACEXTABLE where Launch_Site like 'CCA%' limit 5
```

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

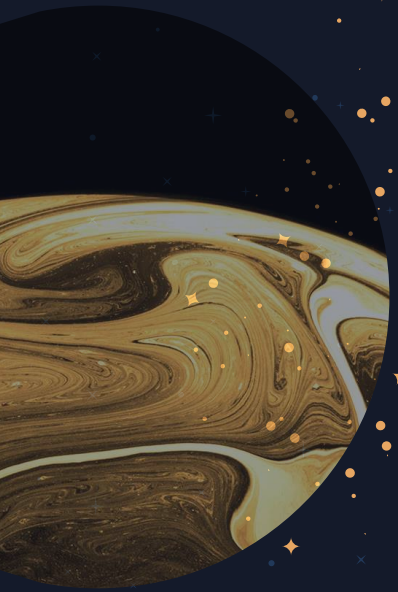
[11]:

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

LIKE is used to query wildcard values. ‘CCA%’ indicates the query should only produce launch site names beginning with CCA.

LIMIT 5 is used to limit the results to 5 records only.

3. SQL – total payload mass by NASA (CRS)



```
%%sql

select sum(PAYLOAD_MASS__KG_) as 'Total Payload Mass', Customer
from SPACEXTABLE
where Customer = 'NASA (CRS)'
```

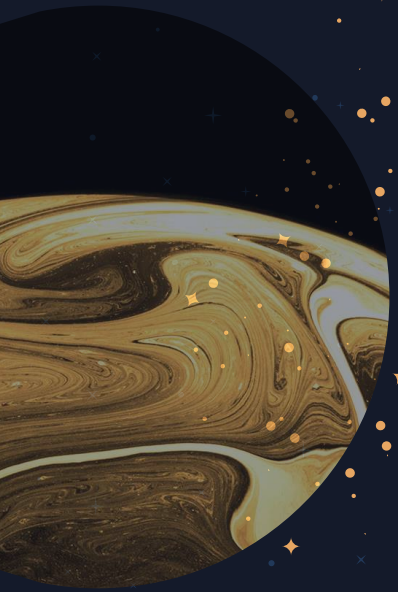
```
 * sqlite:///my_data1.db
Done.
```

Total Payload Mass	Customer
45596	NASA (CRS)

SUM is used to calculate the total of a column.

AS is used to provide a new name for the calculated column.

4. SQL – average payload mass by F9 v1.1



```
%%sql

select avg(PAYLOAD_MASS__KG_) as 'Average Payload Mass', Booster_Version
from SPACEXTABLE
where Booster_Version like 'F9 v1.1%'
```

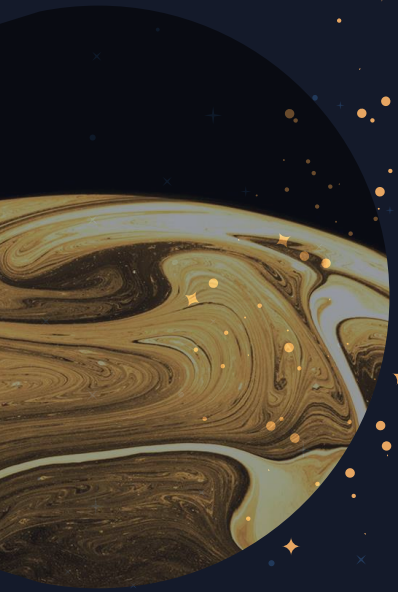
```
 * sqlite:///my_data1.db
Done.
```

Average Payload Mass	Booster_Version
2534.6666666666665	F9 v1.1 B1003

AVG is used to calculate the average of a column. This query uses the AS column again. Its purpose was explained on the previous slide.

LIKE and % limits the output to values where Booster Version starts with F9 v1.1

5. SQL – 1st successful ground landing date



```
%%sql

select min(Date), Landing_Outcome from SPACEXTABLE
where Landing_Outcome = 'Success (ground pad)'
```

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

min(Date)	Landing_Outcome
2015-12-22	Success (ground pad)

MIN is used to provide the lowest value of a column. Here, MIN was used on the DATE column to get the first date that meets the WHERE criteria.

6. SQL – successful drone ship landings with payloads between 4,000 kg and 6,000 kg

```
%%sql

select Booster_Version, Landing_Outcome, PAYLOAD_MASS__KG_
from SPACEXTABLE
where Landing_Outcome = 'Success (drone ship)'
and PAYLOAD_MASS__KG_ between 4000 and 6000
order by PAYLOAD_MASS__KG_
```

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

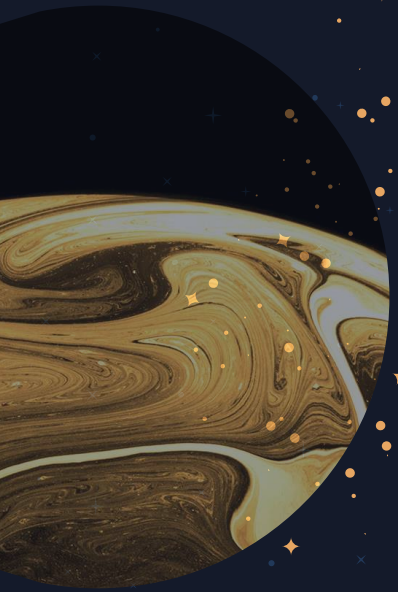
Booster_Version	Landing_Outcome	PAYLOAD_MASS__KG_
F9 FT B1026	Success (drone ship)	4600
F9 FT B1022	Success (drone ship)	4696
F9 FT B1031.2	Success (drone ship)	5200
F9 FT B1021.2	Success (drone ship)	5300

Here, the WHERE clause has multiple conditions:

- Landing outcome must be successful drone ships only
- Payload mass must be between 4,000 kg to 6,000 kg

I used an additional ORDER_BY clause to sort out Payload values in ascending order in order to present the findings better.

7. SQL – total number of successful vs. failed mission outcomes



```
[49]: %%sql

select count(Mission_Outcome) as 'Count of Mission Outcomes', Mission_Outcome
from SPACEXTABLE
group by Mission_Outcome
```

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

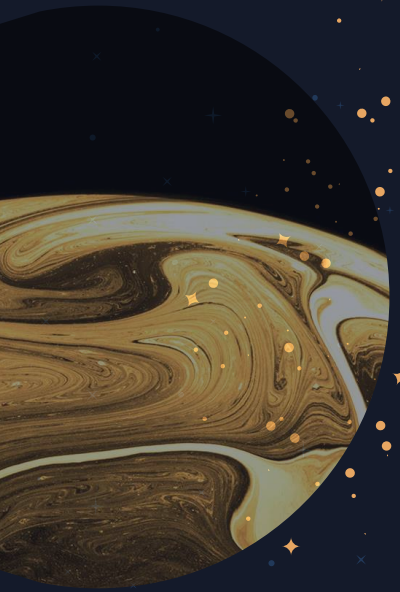
[49]:

Count of Mission Outcomes	Mission_Outcome
1	Failure (in flight)
99	Success
1	Success (payload status unclear)

COUNT is used to count the number of records that meet a specified criteria.

Here, it counts the grouped records specified in the GROUP BY clause.

8. SQL – booster names which carried the max payload mass using a subquery



```
%%sql

select Booster_Version, PAYLOAD_MASS__KG_
from SPACEXTABLE
where PAYLOAD_MASS__KG_ in (select max(PAYLOAD_MASS__KG_)
                            from SPACEXTABLE)
```

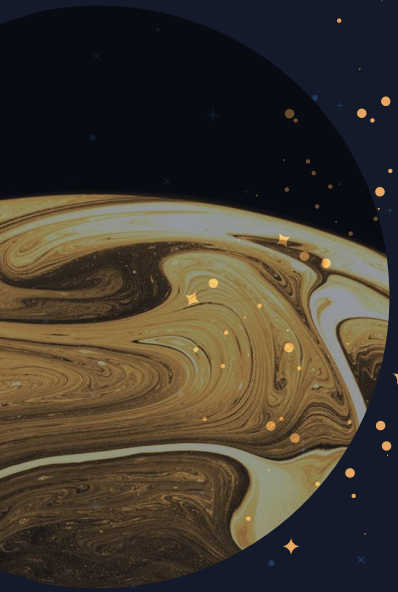
```
 * sqlite:///my_data1.db
Done.
```

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

The SUBQUERY contains the query to search for the MAX Payload Mass from the same database table. The subquery limits the results from the outside query.

Here, you can see that the MAX Payload Mass is 15,600 kg. Only Boosters which have carried this weight are outputted by the query.

9. SQL – 2015 launch records for failed drone ship landings



```
[24]: %%sql

select substr(Date,6,2) as 'Month', substr(Date,0,5) as 'Year', Landing_Outcome, Booster_Version, Launch_Site
from SPACEXTABLE
where Landing_Outcome = 'Failure (drone ship)'
and substr(Date,0,5) = '2015'
```

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

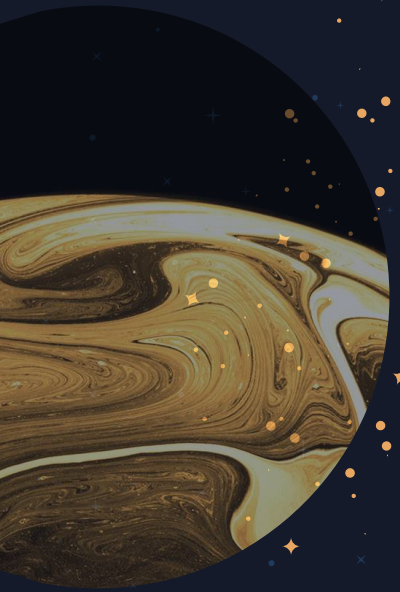
[24]:

Month	Year	Landing_Outcome	Booster_Version	Launch_Site
10	2015	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
04	2015	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

In this query, SUBSTR is used to extract only the months and the years from the Date field.

AS is again used to provide more meaningful column names.

10. SQL – ranked count of successful landings between 2010-06-04 and 2017-03-20



```
%%sql

select count(*) as 'Count', Landing_Outcome
from SPACEXTABLE
where Date between '2010-06-04' and '2017-03-20'
group by Landing_Outcome
order by count(*) desc
```

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

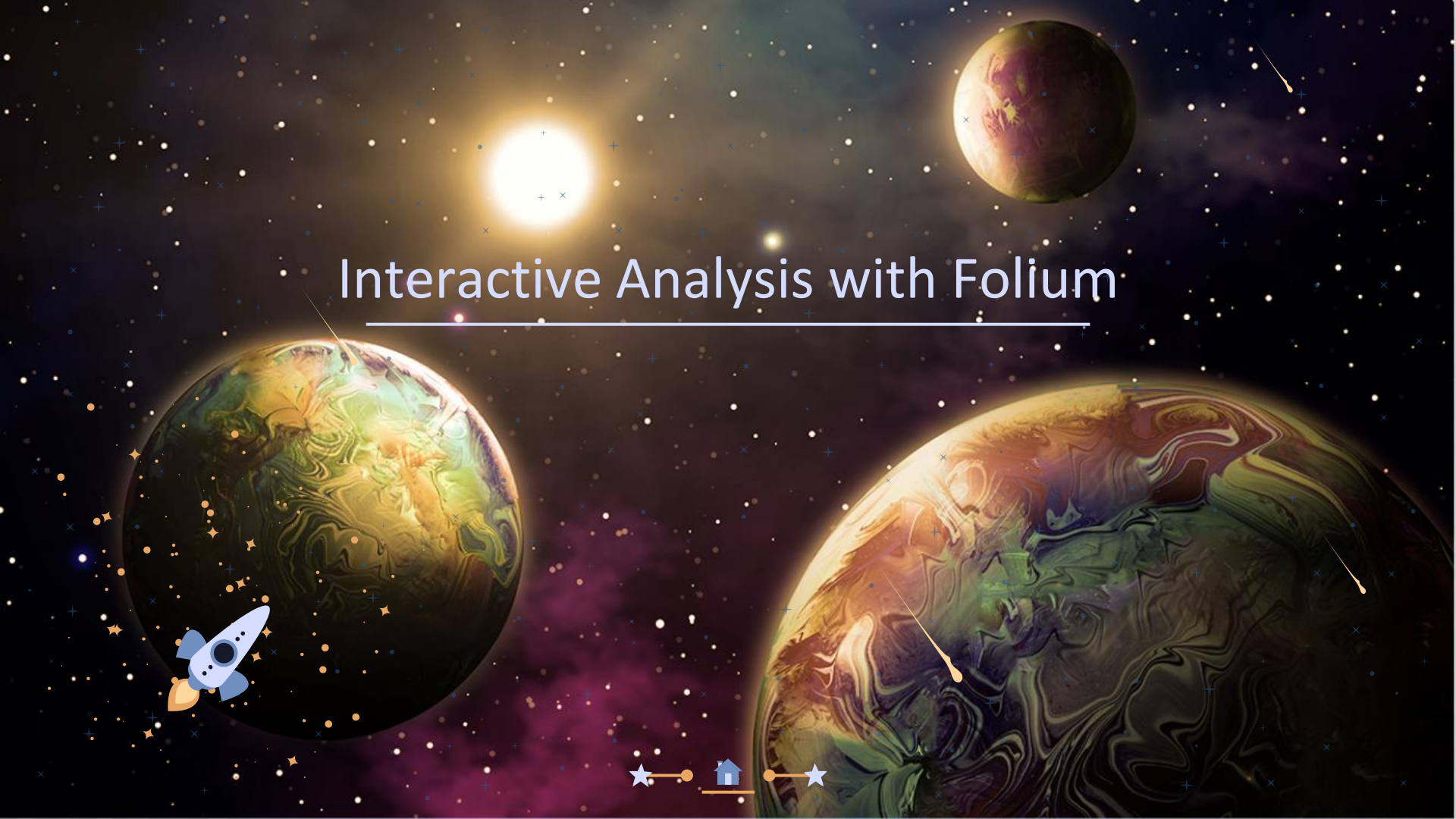
Count	Landing_Outcome
10	No attempt
5	Success (ground pad)
5	Success (drone ship)
5	Failure (drone ship)
3	Controlled (ocean)
2	Uncontrolled (ocean)
1	Precluded (drone ship)
1	Failure (parachute)

COUNT counts the records per group / landing outcome.

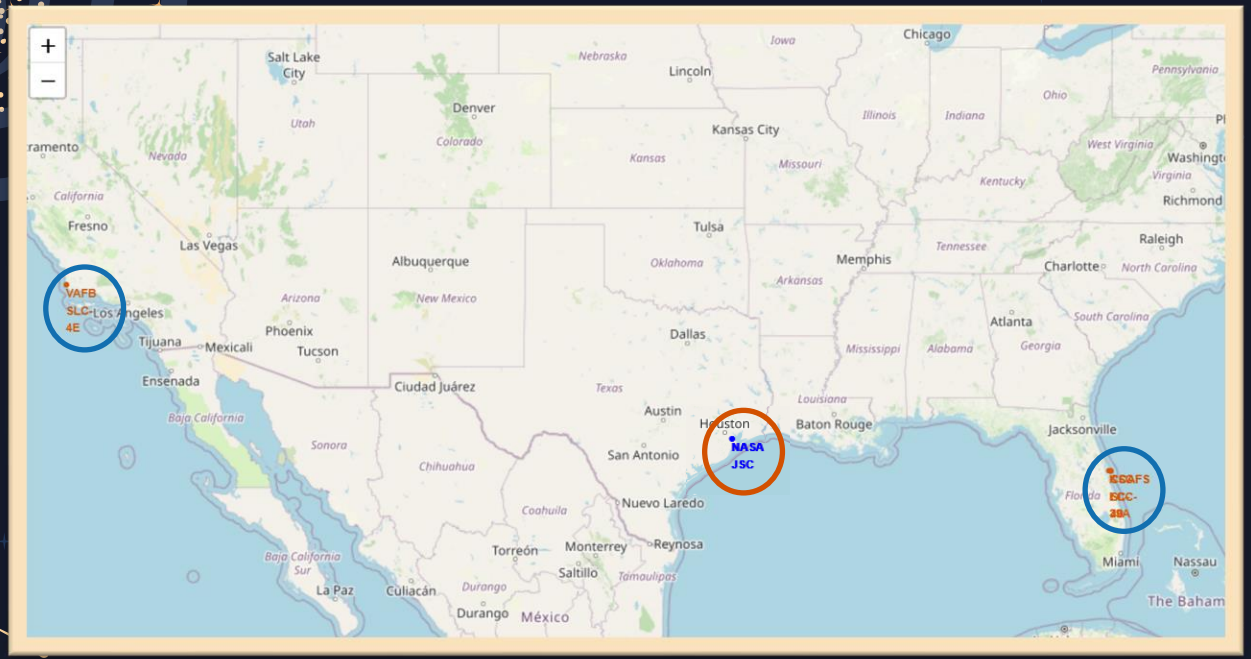
WHERE restricts the records to the date range specified.

ORDER BY sorts out the count of landing outcomes in descending order.

Interactive Analysis with Folium

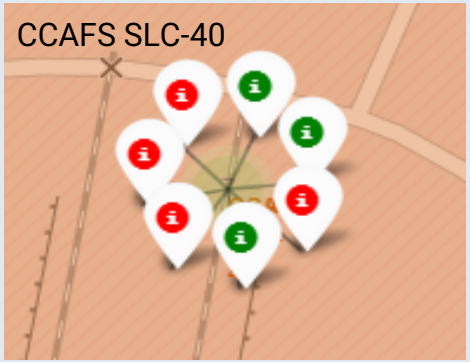


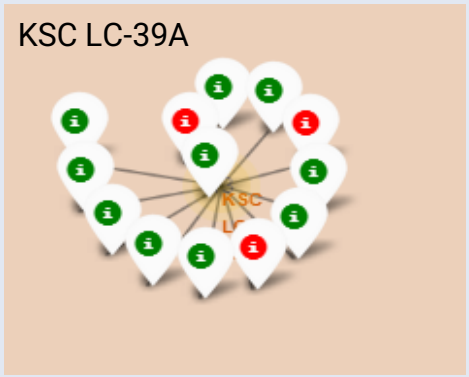
1. SpaceX's Launch Sites

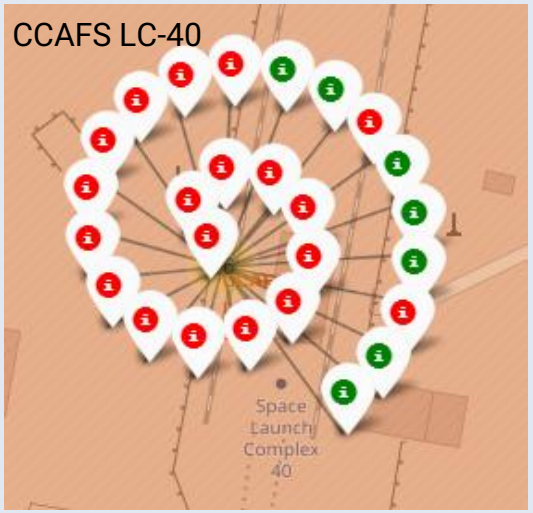


SpaceX launch sites' (circled in blue) are located in 2 main areas: California and Florida. Both sites are relatively close to the NASA Johnson Space centre (circled in orange), although Florida is closer.

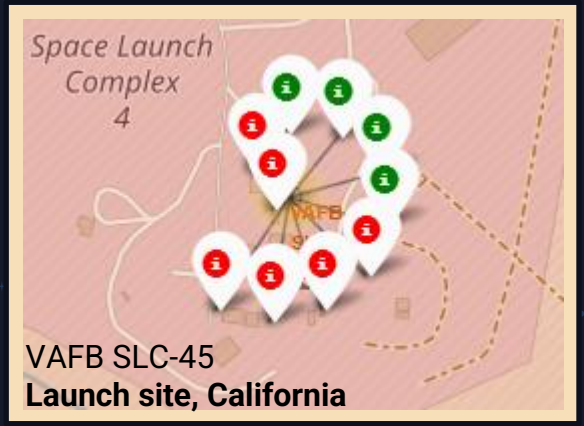
2. Successful vs. Failed Landing Outcomes







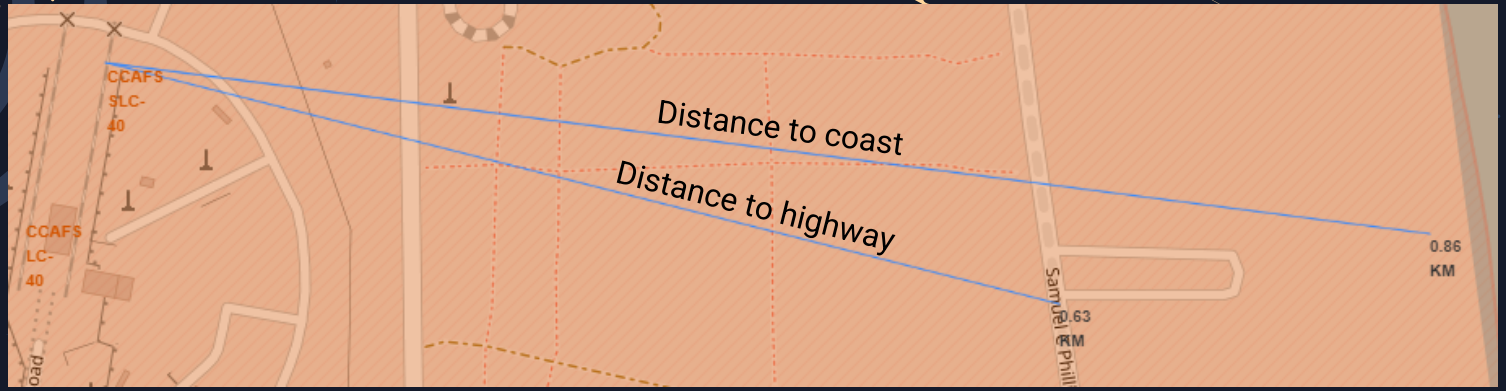
Launch sites, Florida

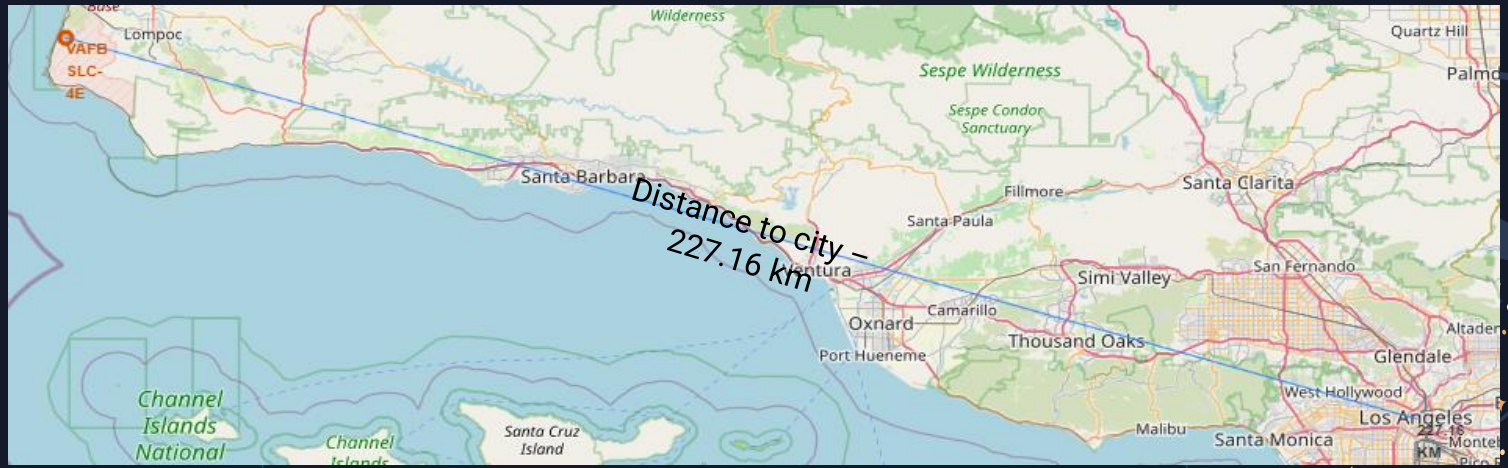


Launch site, California

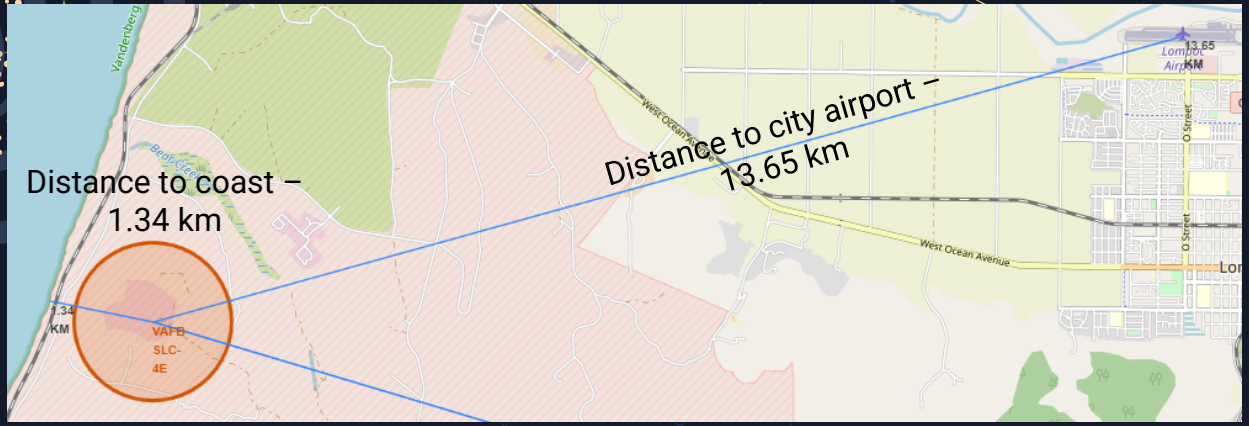
On average, there is a 43.5% success rate across all 3 launch sites in Florida. VAFB, the site in Cali, has a 40% success rate. There's also more launch attempts from Florida (ratio 23:5). **This is because rockets launched closer to the equator can take optimum advantage of the earth's substantial rotational speed. We can infer that equatorial launches help contribute to a successful landing.**

3. Launch Sites' Proximity to Landmarks





3. Launch Sites' Proximity to Landmarks



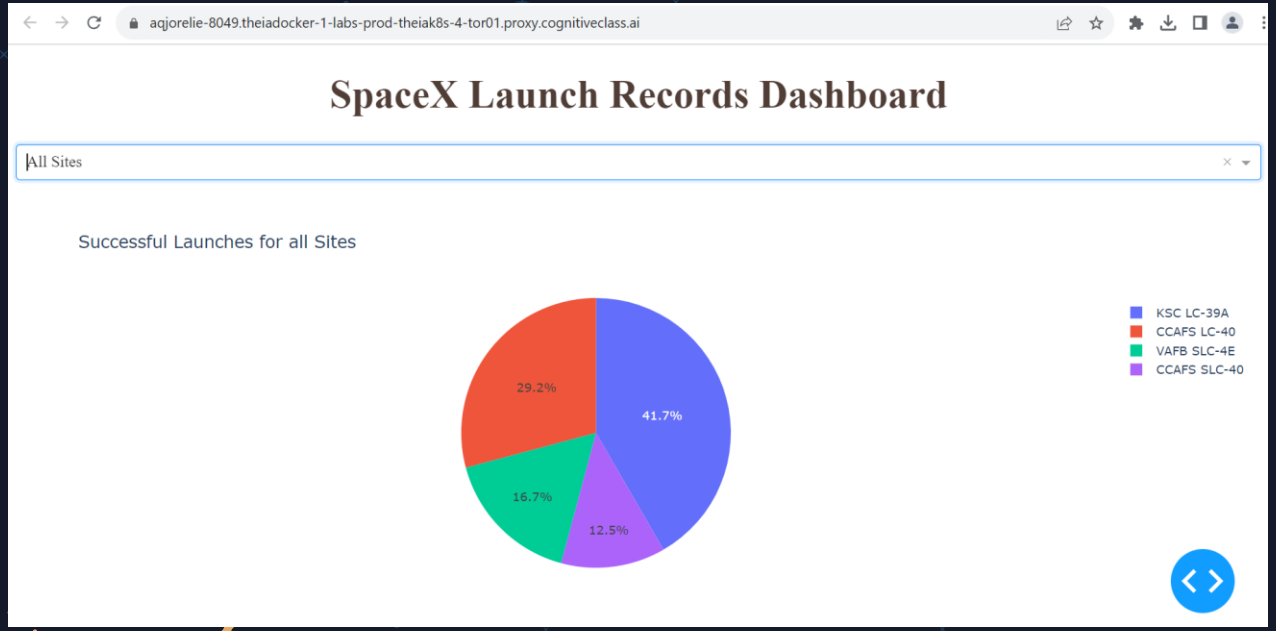
Multiple plot lines have been drafted to answer the following questions:

1. Are launch sites in close proximity to railways? No
2. Are launch sites in close proximity to highways? No
3. Are launch sites in close proximity to coastline? Yes
4. Do launch sites keep certain distance away from cities? Yes

The geographic locations of these launch sites prove that they are built as far away as possible from places where people usually gather, e.g. airports, railway stations, cities, for public safety. These hubs are still close enough to provide support when required. From our analyses, the launch sites are close to the coasts. There are advantages to this, i.e. in the event of a catastrophic fire, the water is a possible source of extinguisher.

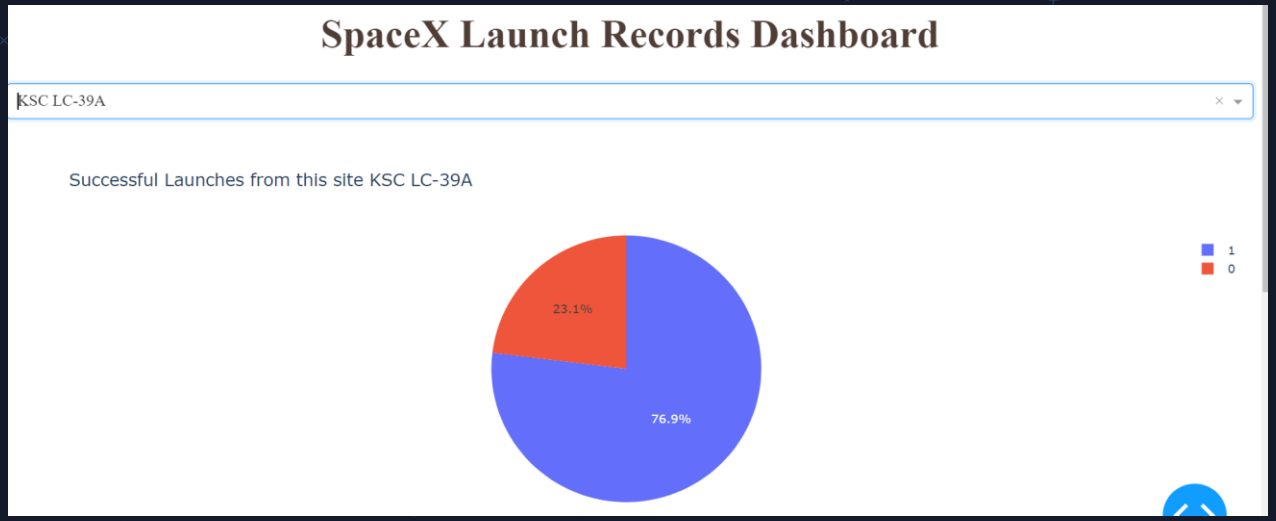
Interactive Dashboard with Plotly Dash

1. Launch success count for all sites



- KSC LC-39A has the highest success rate out of all the sites at 41.7%.
- CCAFS SLC-40 has the lowest success rate at 12.5%. Interestingly, this was one of the launch sites in Florida. VAFB, based in California, is the 2nd lowest.

2. Launch site with the highest launch rate

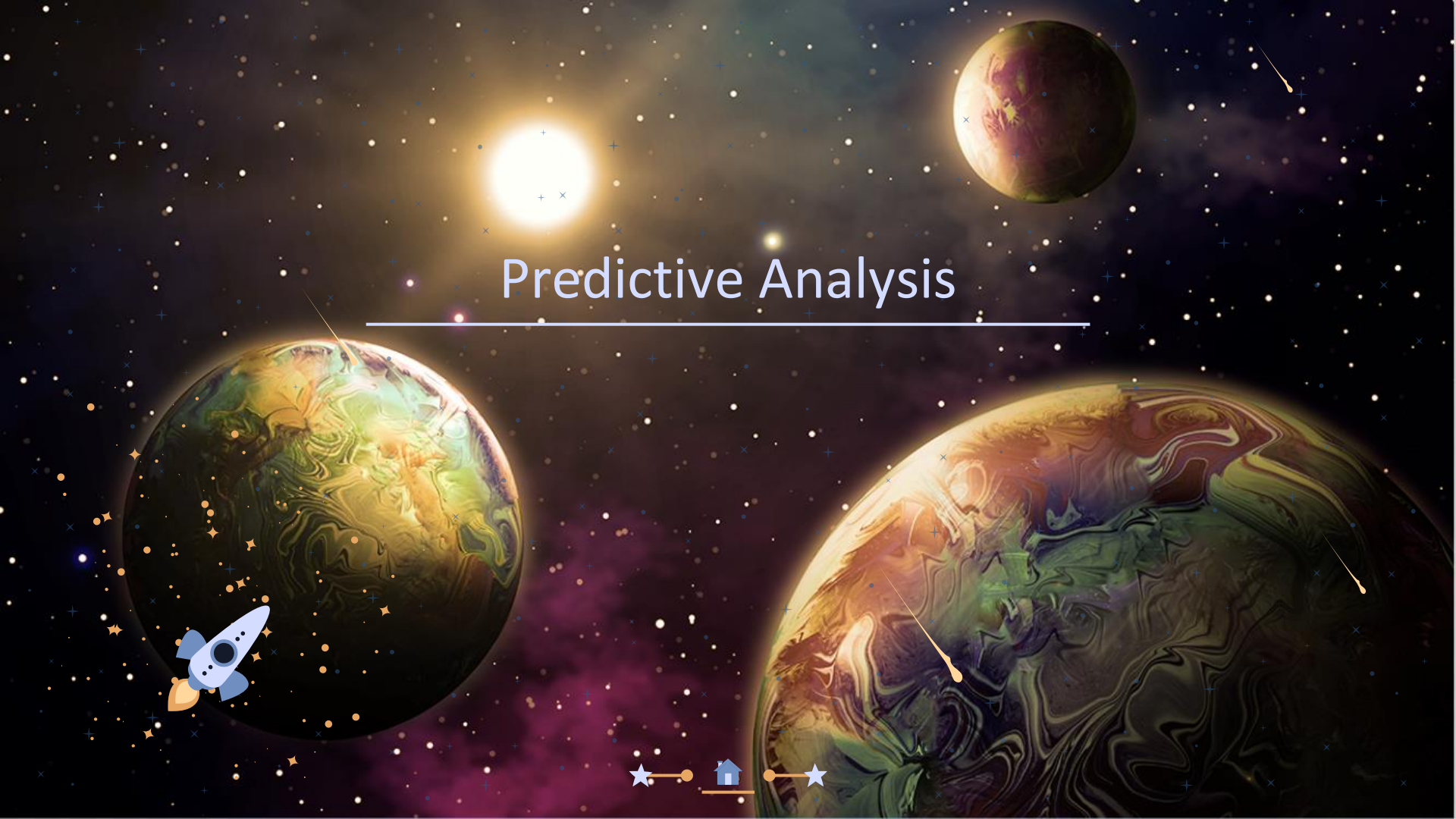


- The dropdown option now shows the selected site = KSC LC-39A. As mentioned on the previous slide, this site has the highest success rate. Its failure rate is considerably lower than the other sites.

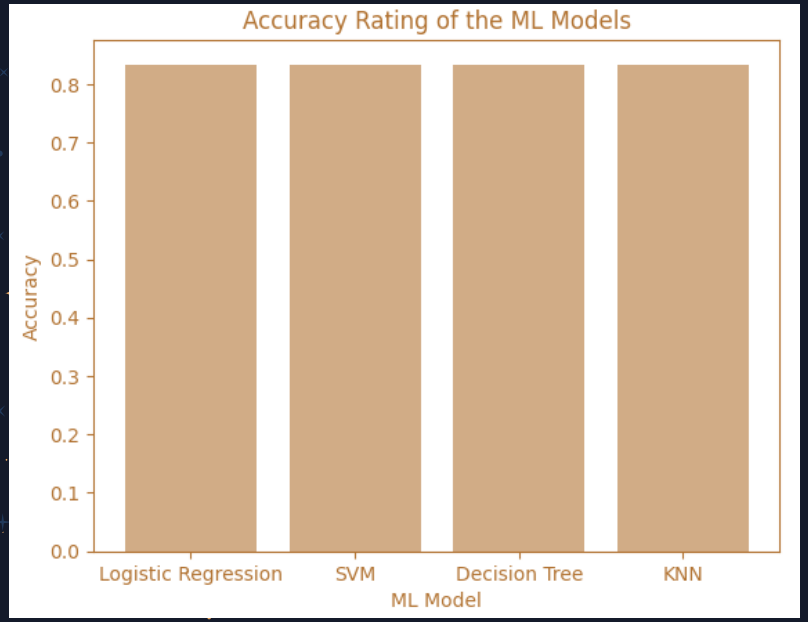
3. Payload vs. Launch scatter plot

- Payloads with mass between 2,000 kg to 5,000 kg have a higher success rate.
- V1.1 booster is the least successful booster version.

Predictive Analysis



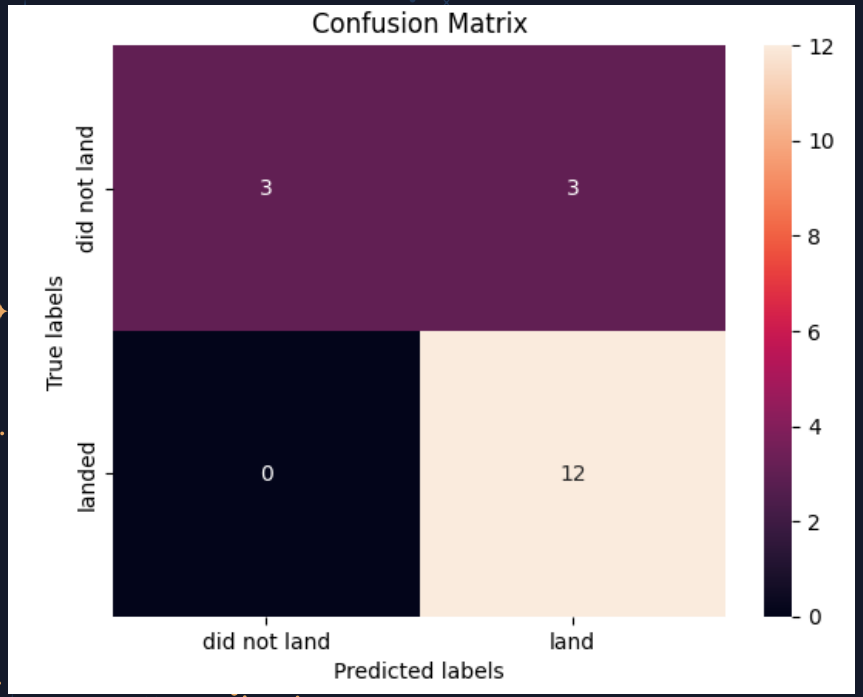
Accuracy Performance of the 4 Models



- As mentioned on **slide 19**, the accuracy of the 4 models were tested using .score.
- This bar chart shows that **all 4 models have the exact accuracy rating as the other, meaning that no 1 model outperforms the other**.

CAVEAT: We need a bigger sample data to do further test on these models.

Confusion Matrix of the 4 Models



- The purpose of the confusion matrix has been mentioned on **slide 20.**

- Again, **<u>all 4 models have the same confusion matrix</u>**. This matrix means:

1. 12 True Positives
2. 3 True Negatives
3. 3 False Positives – the best FP rate is 0.0. This is an issue indicator. However as all models have the exact same number of FPs, no 1 model outperforms the other.
4. 0 False Negatives

CAVEAT: We need a bigger sample data to do further test on these models.

VI

Conclusion

EDA findings
Interactive analytic findings
Predictive analytics findings

Key Findings

Exploratory Data Analysis

- As the flight number increases, the first stage landing is more likely to be successful.
- Orbits ES-L1, GEO, HEO and SSO have a 100% success rate.

Visualisation Analysis

- KSC LC-39A has the highest success rate for launches at 41.7%.
- The proximity plot lines in the Folium site map shows that the launch sites are of sufficient distance from busy landmarks, like the coastline or the highway.

Predictive Analysis

- All 4 models have an 83% accuracy so currently, there is not one model that outperforms the other.

Conclusion

These were the questions we intended to answer to give SpaceY a competitive advantage over SpaceX.

Can we predict if the first landing will be successful?

Based on our analyses, we are able to make predictions on whether the first stage can be reused. If the first stage can be reused, this will cost the company less.

What factors influence a successful landing?

We know that launches are more successful when:

- Launches happen from KSC LC-39A, with payload masses above 5,500 kg.
- We apply the lessons learned from the previous years as exhibited by the sharp increase yearly.
- Launches in orbits ES-L1, GEO, HEO and SSO.
- Launched in a site closer to the equator.

Is there a machine learning model we can use to predict?

We created 4 models. Based on our initial evaluation, all 4 models: logistic regression, KNN, decision tree and SVM, can all be used to predict, with the caveat that we can train and test better with a bigger sample data.

THANK YOU