Winning Space Race with Data Science

Le Hoang Duy

31 Mar 2024

Outline

1. Executive Summary
2. Introduction
3. Methodology
4. Results
5. Conclusion
6. Appendix

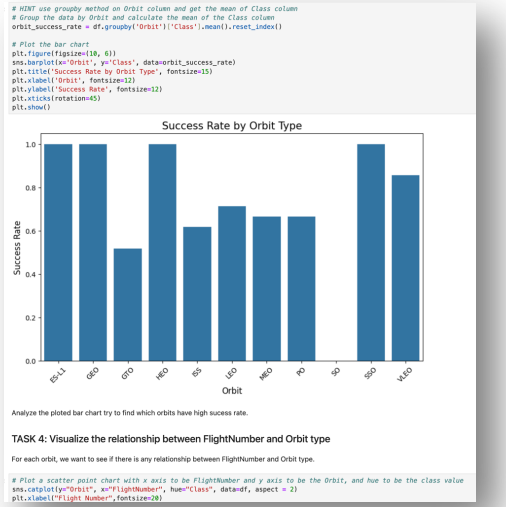
Executive Summary

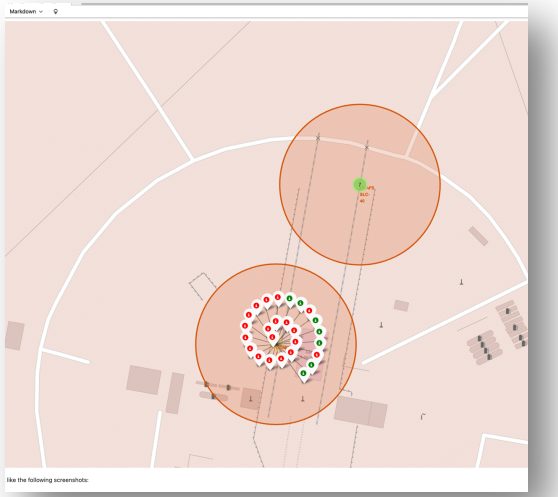
Summary of Methodologies:

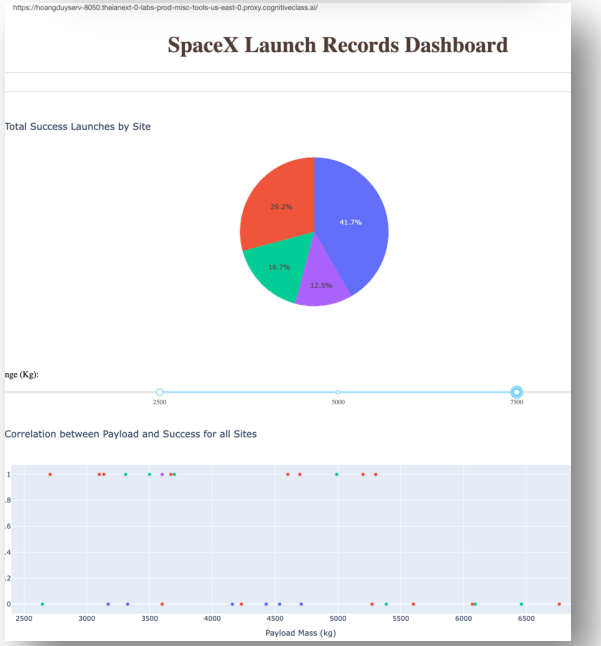
- Data Collection
- Data Wrangling
- Exploratory Data Analysis
- Interactive Visual Analytics
- Predictive Analysis (Classification)

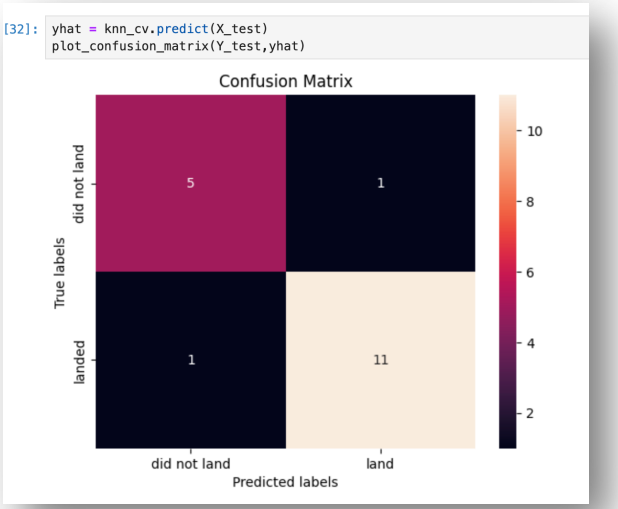
Summary of Results:

- Exploratory Data Analysis (EDA) results
- Geospatial analytics
- Interactive dashboard
- Predictive analysis of classification models









Introduction

- SpaceX launches Falcon 9 rockets at a cost of around $62M, significantly cheaper than competitors' costs of $165M or more, largely due to the ability to land and reuse the first stage.
- Predicting the successful landing of the first stage can determine launch costs, aiding in evaluating competitive bids against SpaceX.
- The project's goal is to predict the successful landing of the SpaceX Falcon 9 first stage.

Section 1

Methodology

Methodology

1. **Data Collection:**
   - Making GET requests to the SpaceX REST API
   - Web Scraping
2. **Data Wrangling:**
   - Using the .fillna() method to remove NaN values
   - Using the .value_counts() method to analyze:
     - Number of launches on each site
     - Number and occurrence of each orbit
     - Number and occurrence of mission outcomes per orbit type
   - Creating a landing outcome label indicating:
     - 0 for unsuccessful booster landings
     - 1 for successful booster landings
3. **Exploratory Data Analysis:**
   - Utilizing SQL queries for data manipulation and evaluation
   - Visualizing relationships between variables and identifying patterns using Pandas and Matplotlib
4. **Interactive Visual Analytics:**
   - Conducting geospatial analytics with Folium
   - Developing an interactive dashboard using Plotly Dash
5. **Data Modeling and Evaluation:**
   - Pre-processing (standardizing) the data using Scikit-Learn
   - Splitting the data into training and testing sets with train_test_split
   - Training various classification models
   - Tuning hyperparameters using GridSearchCV
   - Plotting confusion matrices for each classification model
   - Assessing the accuracy of each classification model

Data Collection – SpaceX API

Utilizing the SpaceX API to fetch data regarding launches, encompassing details about the rocket utilized, payload delivered, launch specifications, landing specifications, and the outcome of the landing.

Step 1:

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

Step 2:

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

Step 3:

- ✓ Create a Pandas DataFrame from the constructed dictionary dataset

Step 4:

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

GitHub Click here

Data Collection - Scraping

Web scraping Wikipedia

Step 1:

•Request the HTML page from the static URL.

•Assign the response to an object.

Step 2:

•Create a BeautifulSoup object from the HTML response object.

•Find all tables within the HTML page.

Step 3:

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

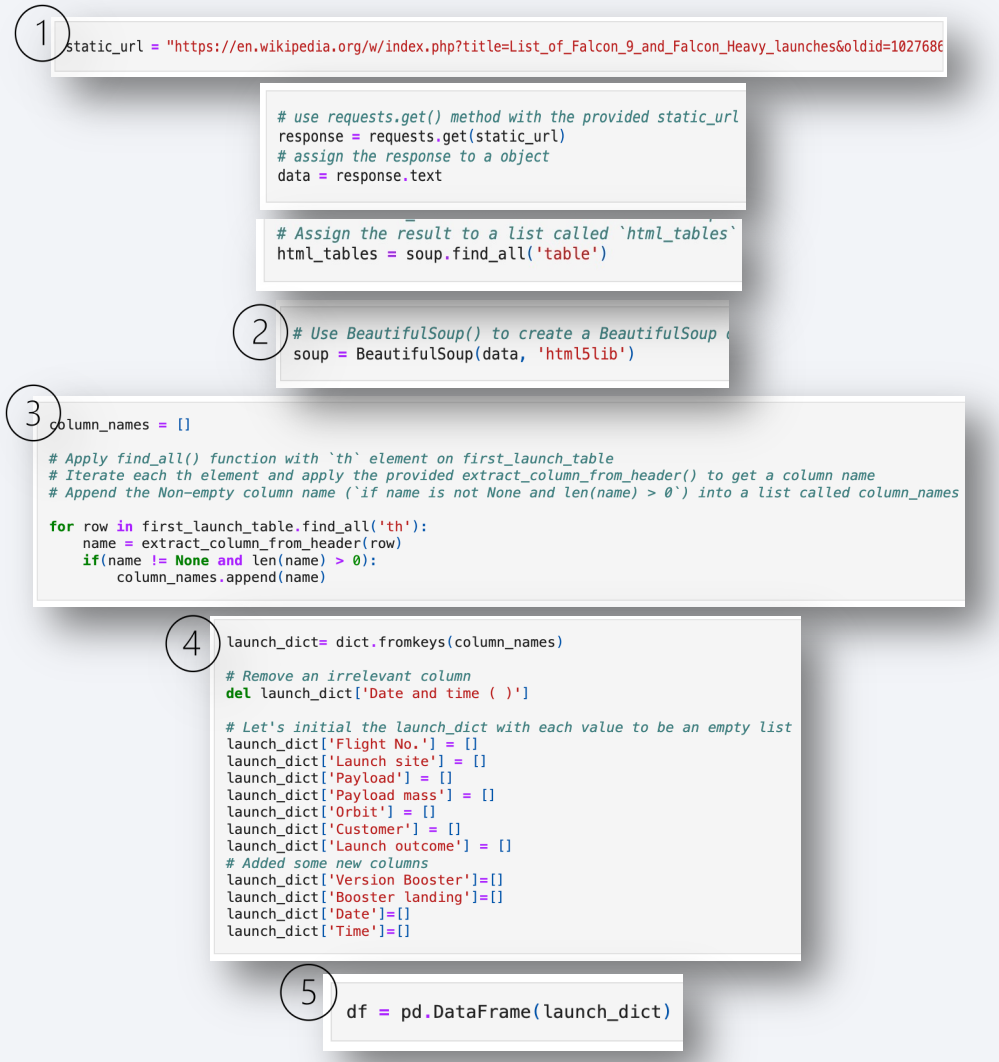
Step 4:

•Use the column names as keys in a dictionary.

•Utilize custom functions and logic to parse all launch tables (refer to Appendix) to fill the dictionary values.

Step 5:

•Convert the dictionary to a Pandas DataFrame ready for export.

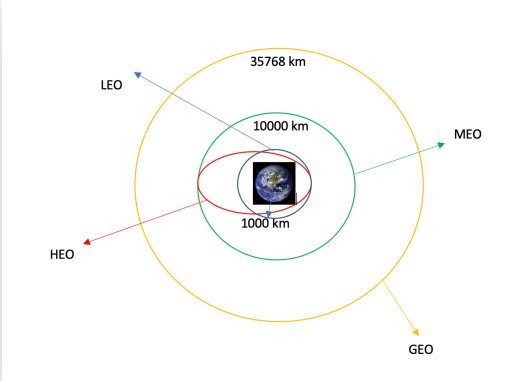


GitHub Click here

Data Wrangling

Context

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



Initial Data Exploration

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

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

1

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

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

2

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

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

3

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

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

GitHub Click here

Data Wrangling

Context

The landing outcome is indicated in the "Outcome" column:

- True Ocean: The mission successfully landed in a specific region of the ocean.
- False Ocean: The mission unsuccessfully landed in a specific region of the ocean.
- True RTLS: The mission successfully landed on a ground pad.
- False RTLS: The mission unsuccessfully landed on a ground pad.
- True ASDS: The mission successfully landed on a drone ship.
- False ASDS: The mission unsuccessfully landed on a drone ship.
- None ASDS and None None: Represent a failure to land.

Data Wrangling

To determine landing success, create a binary column where 1 represents success and 0 represents failure

1. Define a set of unsuccessful outcomes, referred to as "bad_outcome."
2. Create a list named "landing_class," assigning 0 if the corresponding row in "Outcome" is in the set "bad_outcome," otherwise 1.
3. Generate a new column named "Class" containing values from the "landing_class" list.
4. Export the DataFrame as a .csv file for further analysis.

1
```
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
bad_outcomes
```
```
{'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

2
```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise

landing_class = []

for outcome in df['Outcome']:
    if outcome in bad_outcomes:
        landing_class.append(0)
    else:
        landing_class.append(1)
```

3
```
df['Class']=landing_class
df[['Class']].head(8)
```

4
```
df.to_csv("dataset_part\_2.csv", index=False)
```

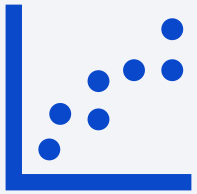
GitHub Click here

EDA with Data Visualization

SCATTER CHARTS

Scatter charts were produced to visualize the relationships between:

- Flight Number and Launch Site
- Payload and Launch Site
- Orbit Type and Flight Number
- Payload and Orbit Type

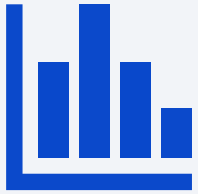


Scatter charts are useful to observe relationships, or correlations, between two numeric variables.

BAR CHART

A bar chart was produced to visualize the relationship between:

- Success Rate and Orbit Type



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

LINE CHARTS

Line charts were produced to visualize the relationships between:

- Success Rate and Year (i.e. the launch success yearly trend)

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

GitHub Click here

EDA with SQL

The SQL queries:

1. Display the names of unique launch sites in the space mission.
2. Display 5 records where launch sites begin with the string 'CCA'.
3. Display the total payload mass carried by boosters launched by NASA (CRS).
4. Display the average payload mass carried by booster version F9 v1.1.
5. List the date when the first successful landing outcome on a ground pad was achieved.
6. List the names of boosters which had success on a drone ship and a payload mass between 4000 and 6000 kg.
7. List the total number of successful and failed mission outcomes.
8. List the names of booster versions which have carried the maximum payload mass.
9. List the failed landing outcomes on drone ships, their booster versions, and launch site names for 2015.
10. Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the dates 2010-06-04 and 2017-03-20, in descending order.

GitHub Click here

Build an Interactive Map with Folium

Visualize the launch data on an interactive map:

1. **Mark All Launch Sites on a Map:**
    1. Initialize the map using a Folium Map object.
    2. Add a folium.Circle and folium.Marker for each launch site on the launch map.
2. **Mark the Success/**Failed **Launches for Each Site on a Map:**
    1. As many launches have the same coordinates, cluster them together.
    2. Before clustering, assign marker colors: green for successful (class = 1) and red for failed (class = 0) launches.
    3. For each launch, add a folium.Marker to the MarkerCluster() object.
    4. Create an icon as a text label, assigning the icon color based on the marker color determined previously.
3. **Calculate Distances Between a Launch Site and Its Proximities:**
    1. To explore launch site proximities, calculate distances between points using their latitude and longitude values.
    2. After marking a point using the latitude and longitude values, create a folium.Marker object to display the distance.
    3. To display the distance line between two points, draw a folium.PolyLine and add it to the map.

GitHub Click here

Build a Dashboard with Plotly Dash

Interactive visualisation Plotly Dash dashboard

1. **Pie Chart (px.pie()):**

   Display the total successful launches per site.

   - Provides a clear visualization of the most successful launch sites.
   - Can be filtered using a dcc.Dropdown() object to view the success/failure ratio for individual sites.

2. **Scatter Graph (px.scatter()):**

   Illustrates the correlation between launch outcome (success or failure) and payload mass (kg).

   - Can be filtered by ranges of payload masses using a RangeSlider() object.
   - Additionally, it can be filtered by booster version to explore correlations specific to different versions.

GitHub Click here

Predictive Analysis (Classification)

Model Development

Prepare the Dataset for Model Development:

- Load the dataset.
- Perform necessary data transformations such as standardization and pre-processing.
- Split the data into training and test datasets using train_test_split().
- Decide which type of machine learning algorithms are most appropriate for the task.

For Each Chosen Algorithm:

- Create a GridSearchCV object and define a dictionary of parameters.
- Fit the GridSearchCV object to the parameters.
- Train the model using the training dataset.

Model Evaluation

- For Each Chosen Algorithm:Utilize the output GridSearchCV object:
- Check the tuned hyperparameters (best_params_).
- Evaluate the accuracy using the score and best_score_ attributes.
- Plot and examine the Confusion Matrix to further assess the model's performance.

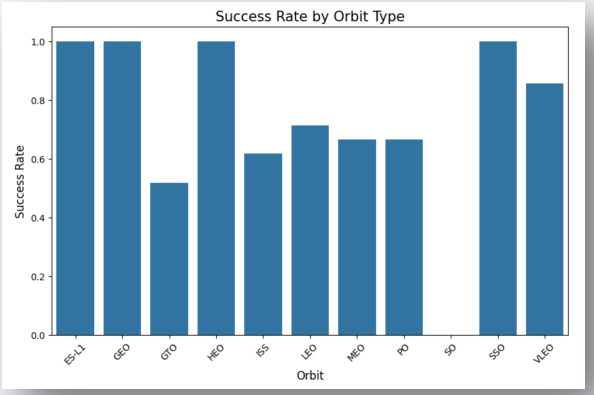
Best Classification Model

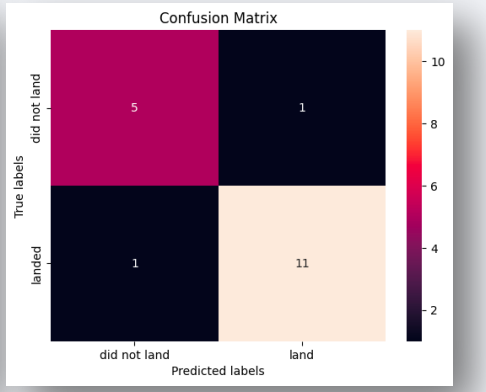
- Review the accuracy scores for all chosen algorithms.
- Determine the model with the highest accuracy score as the best performing model.

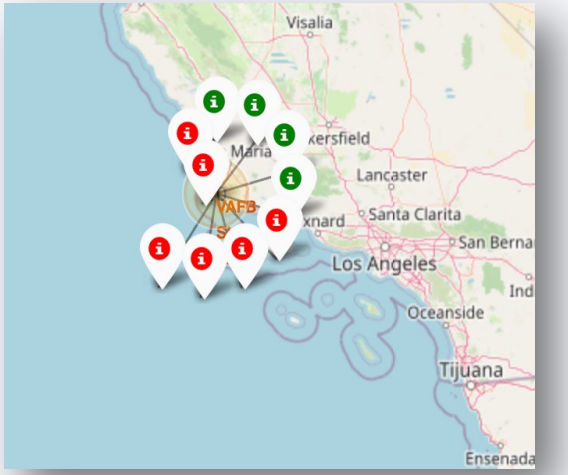
GitHub Click here

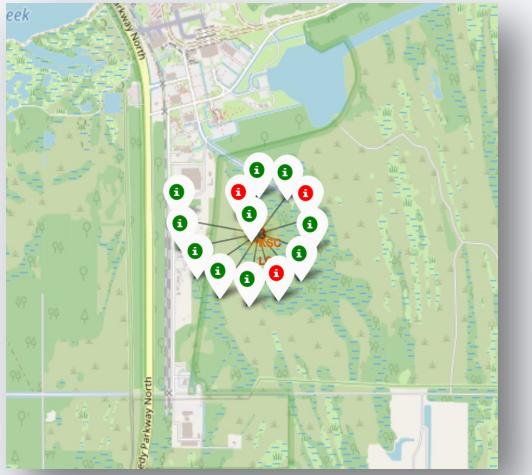
Results

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







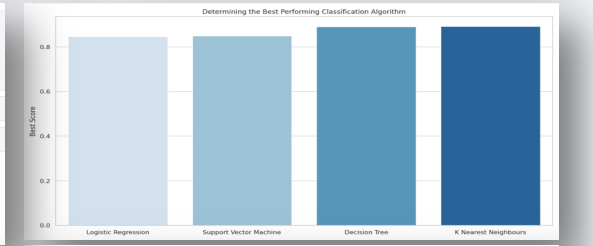


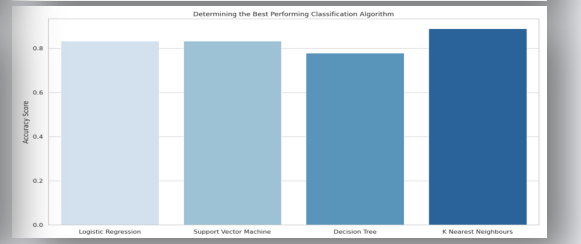
```
algorithms = ['Logistic Regression', 'Support Vector Machine', 'Decision Tree', 'K Nearest Neighbours']
scores = [lr_score, svm_score, tree_score, knn_score]
best_scores = [lr_best_score, svm_best_score, tree_best_score, knn_best_score]
column_names = ['Algorithm', 'Accuracy Score', 'Best Score']

df = pd.DataFrame(list(zip(algorithms, scores, best_scores)),columns = column_names)

df
```

	Algorithm	Accuracy Score	Best Score
0	Logistic Regression	0.833333	0.846429
1	Support Vector Machine	0.833333	0.848214
2	Decision Tree	0.777778	0.889286
3	K Nearest Neighbours	0.888889	0.891071

```
algorithms = ['Logistic Regression', 'Support Vector Machine', 'Decision Tree', 'K Nearest Neighbours']
scores = [lr_score, svm_score, tree_score, knn_score]
best_scores = [lr_best_score, svm_best_score, tree_best_score, knn_best_score]

# Organize the data
df = pd.DataFrame({
    'Algorithm': algorithms,
    'Accuracy Score': scores,
    'Best Score': best_scores
})

# Finding the algorithm with the highest accuracy score
max_accuracy = df['Accuracy Score'].max()
max_index = df['Accuracy Score'].idxmax()
most_appropriate_method = df.loc[max_index, 'Algorithm']

print("Most appropriate method:")
print(most_appropriate_method)
```

```
Most appropriate method:
K Nearest Neighbours
```



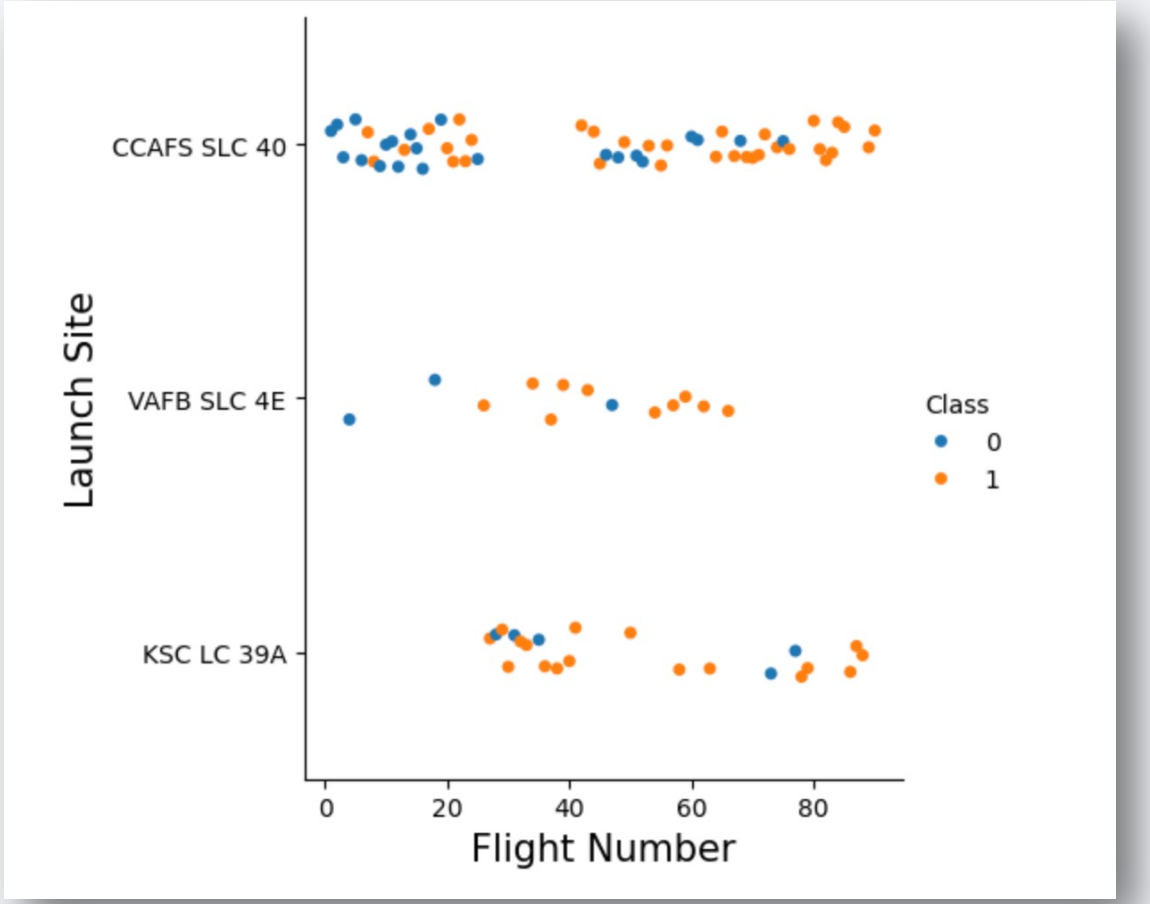


Section 2

Insights drawn from EDA

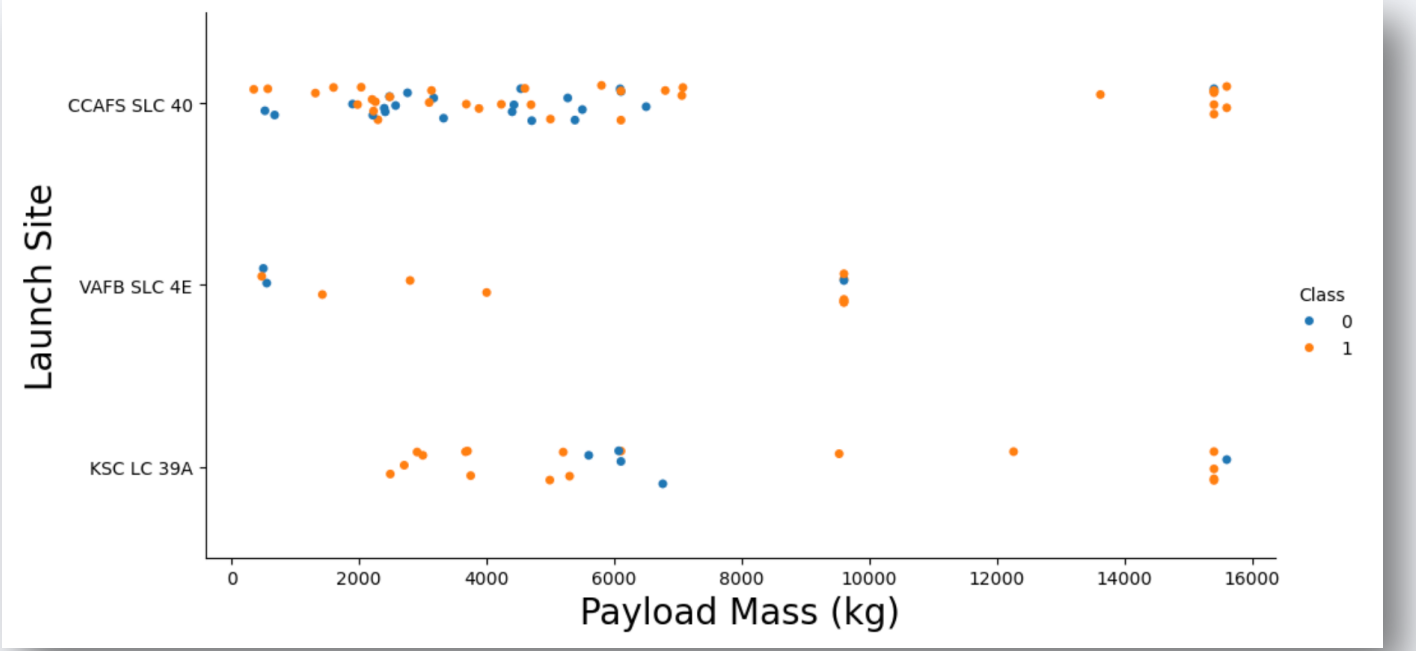
Flight Number vs. Launch Site

- The scatter plot indicates a positive correlation between the number of flights and the success rate at a launch site.
- Early flights (flight numbers < 30) from CCAFS SLC 40 and VAFB SLC 4E were generally unsuccessful, suggesting a learning curve or technical challenges during initial launches.
- KSC LC 39A, having no early flights, shows a higher success rate compared to other sites.
- Beyond a flight number of approximately 30, there is a notable increase in successful landings (Class = 1), indicating improved performance or refinement of processes over time.



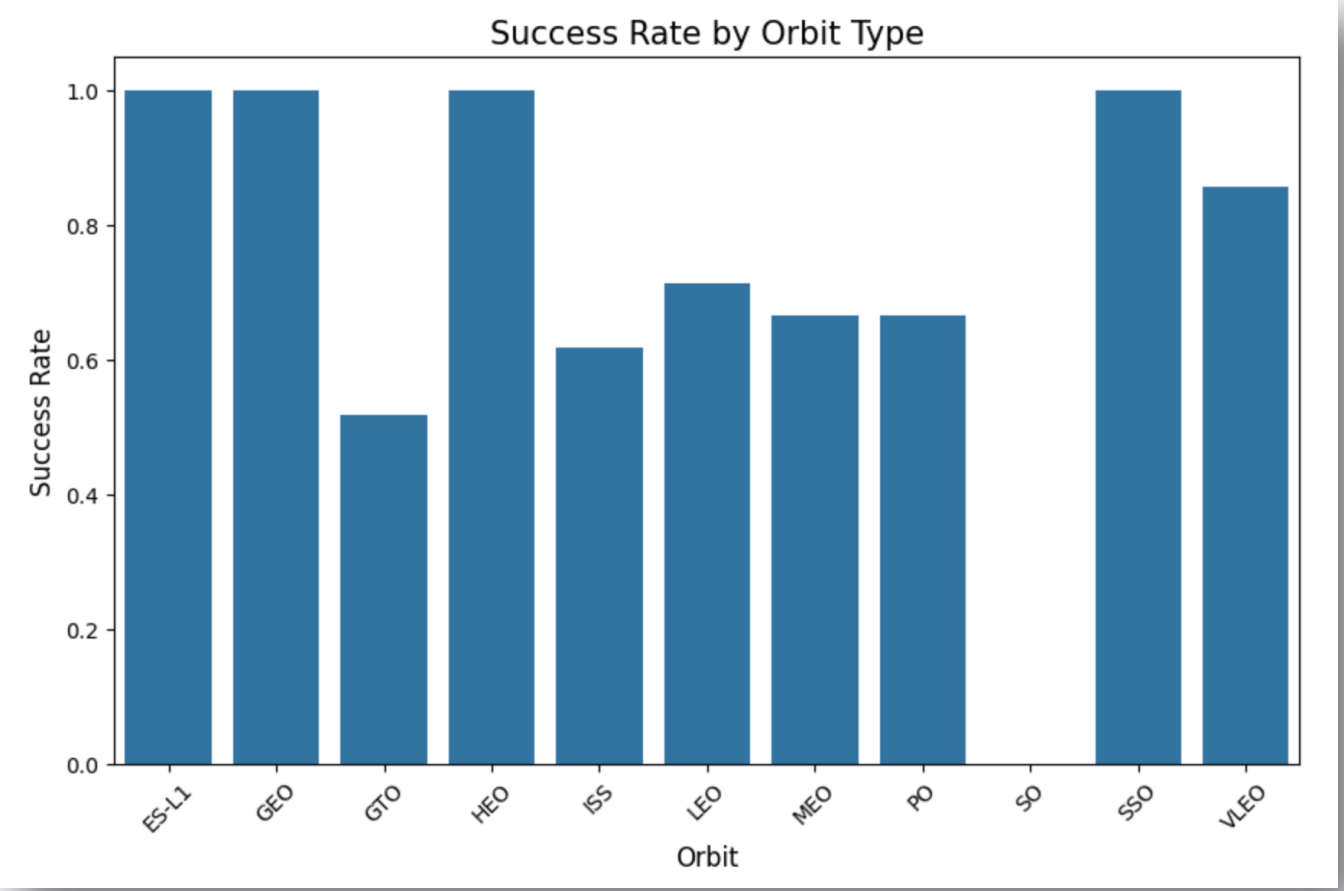
Payload vs. Launch Site

- The scatter plot suggests that above a payload mass of approximately 7000 kg, there are very few unsuccessful landings. However, it's important to note that there is limited data available for these heavier launches.
- There is no discernible correlation between payload mass and success rate for a given launch site.
- All sites launched a variety of payload masses, with the majority of launches from CCAFS SLC 40 involving comparatively lighter payloads, albeit with some outliers.



Success Rate vs. Orbit Type

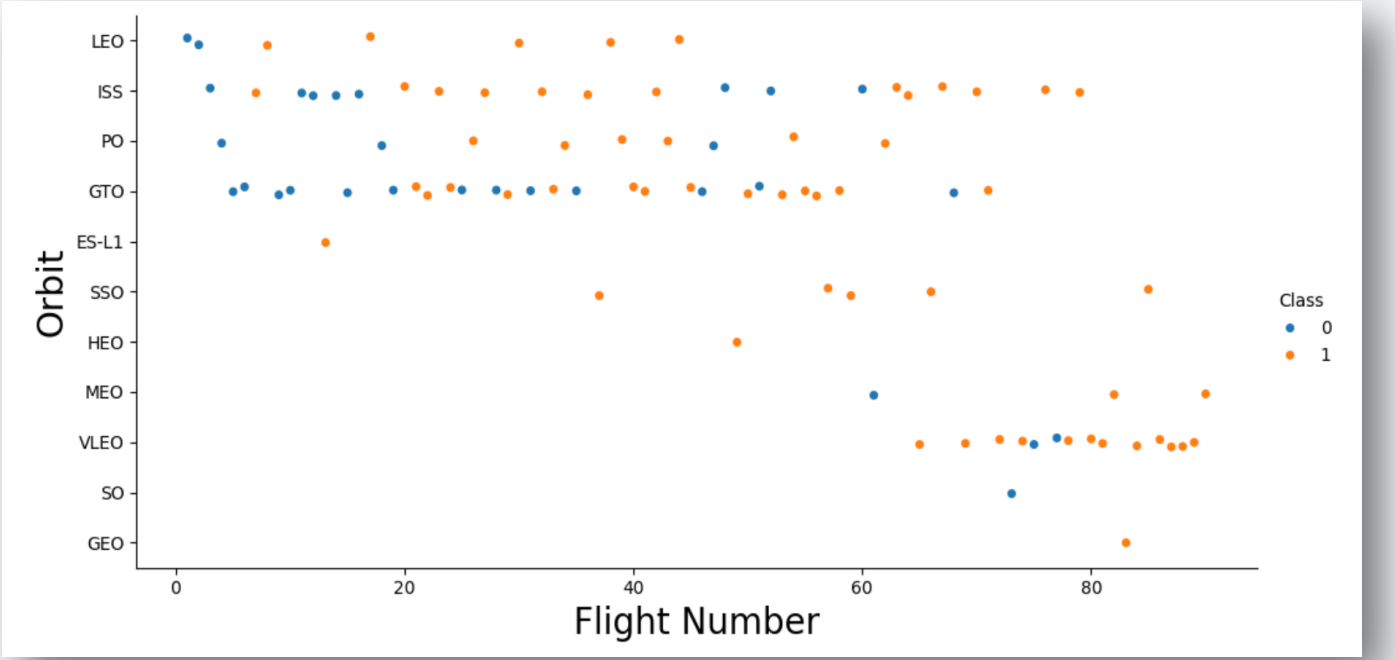
- The bar chart indicates that the following orbits have a 100% success rate:
  - ES-L1 (Earth-Sun First Lagrangian Point)
  - GEO (Geostationary Orbit)
  - HEO (High Earth Orbit)
  - SSO (Sun-synchronous Orbit)
- The orbit with the lowest success rate, at 0%, is:
  - SO (Heliocentric Orbit)



Flight Number vs. Orbit Type

The scatter plot reveals additional insights not evident in previous plots:

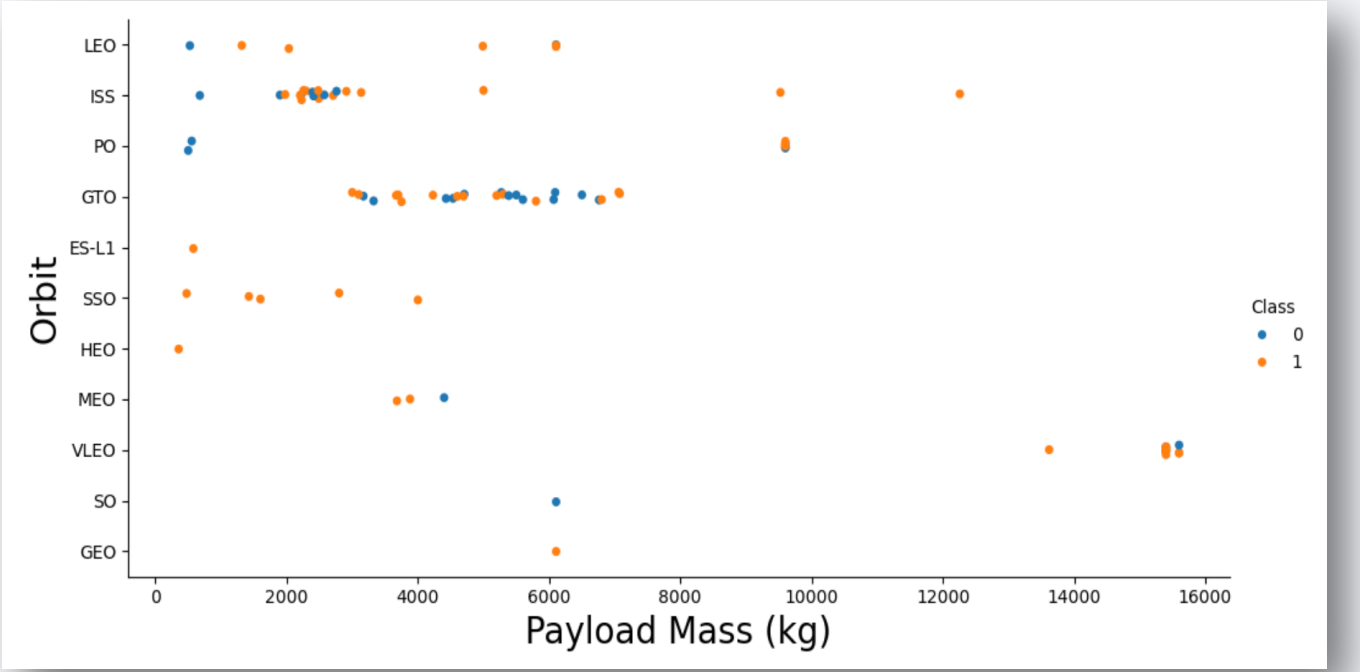
- The 100% success rate of GEO, HEO, and ES-L1 orbits can be attributed to having only 1 flight into each respective orbit.
- The 100% success rate in SSO is notable, with 5 successful flights.
- There is little correlation between Flight Number and Success Rate for GTO.
- Generally, as Flight Number increases, the success rate tends to increase. This trend is particularly pronounced for LEO, where unsuccessful landings occurred predominantly in the early launches.



Payload vs. Orbit Type

The scatter plot highlights the following insights:

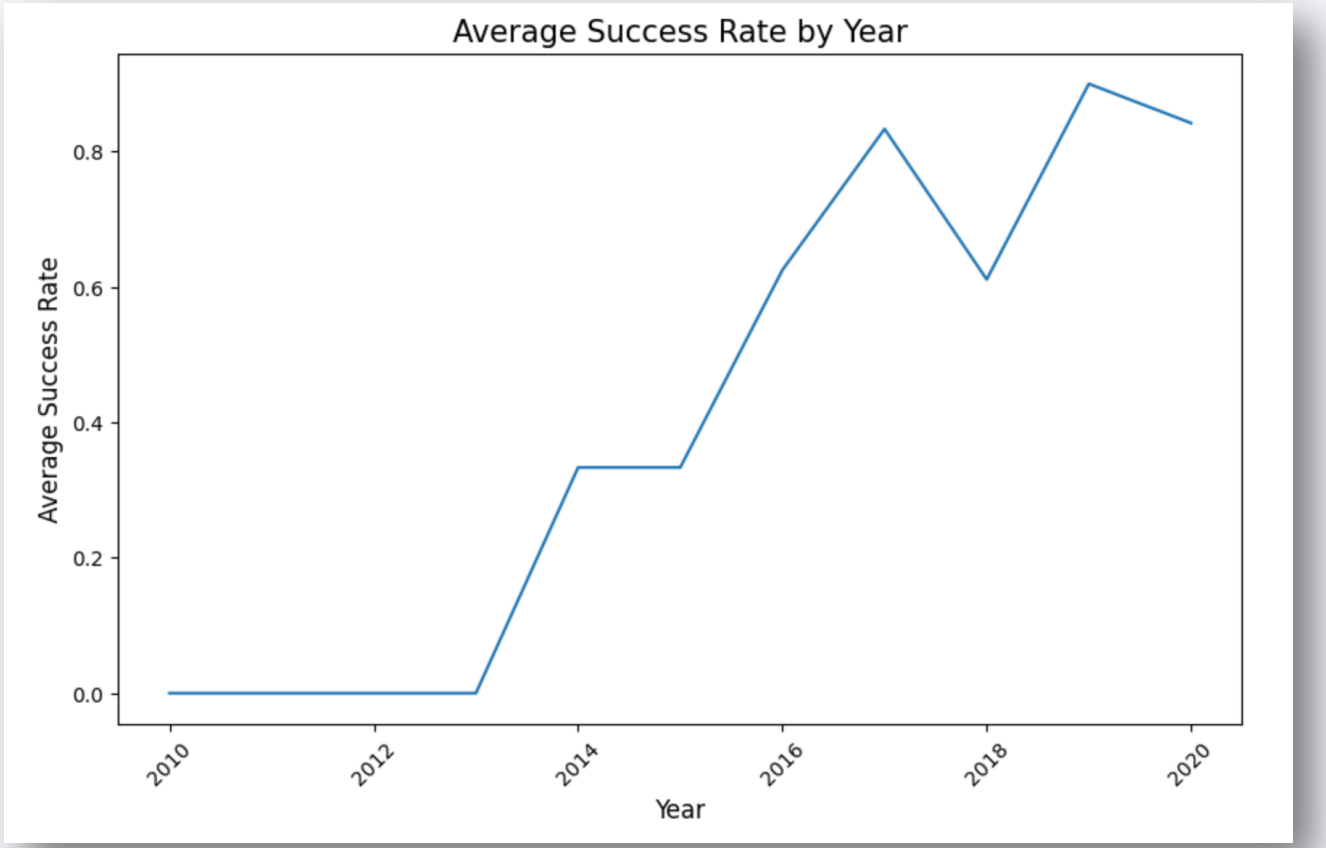
- Orbit types such as PO, ISS, and LEO tend to have higher success rates with heavy payloads.
- For GTO, the relationship between payload mass and success rate is unclear.
- VLEO (Very Low Earth Orbit) launches are associated with heavier payloads, which aligns with intuition.



Launch Success Yearly Trend

The line chart depicts the following trends:

- Between 2010 and 2013, all landings had a 0% success rate.
- From 2013 onwards, the success rate generally increased, with minor dips observed in 2018 and 2020.
- After 2016, the success rate consistently remained above 50%.



All Launch Site Names

- Find the names of the unique launch sites

```
%%sql
SELECT DISTINCT Launch_Site from SPACEXTABLE;
```

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

SELECT DISTINCT specifies that you want to retrieve unique values from the Launch_site column of the table SPACEXTABLE

Launch Site Names Begin with 'CCA'

- Find 5 records where launch sites begin with `CCA`

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

Date	Time (UTC)	Booster_Version	Launch_Site	Payload
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2

- The SQL query "LIMIT 5" fetches only 5 records.
- The LIKE keyword is utilized with the wildcard 'CCA%' to retrieve string values beginning with 'CCA'.

Total Payload Mass

- Calculate the total payload carried by boosters from NASA

```
%%sql
SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEXTABLE WHERE Customer = 'NASA (CRS)';
```

SUM(PAYLOAD_MASS__KG_)
45596

- The SUM keyword is utilized to calculate the total of the LAUNCH column. Additionally, the SUM keyword, along with the associated condition, filters the results to only include boosters from NASA (CRS).

Average Payload Mass by F9 v1.1

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

```
%sql SELECT AVG(PAYLOAD_MASS__KG_) FROM SPACEXTABLE WHERE Booster_Version = 'F9 v1.1';
```

AVG(PAYLOAD_MASS__KG_)
2928.4

- The AVG keyword is employed to calculate the average of the PAYLOAD_MASS__KG_ column. Additionally, the WHERE keyword, along with the associated condition, filters the results to only include data related to the F9 v1.1 booster version.

First Successful Ground Landing Date

- Retrieve the dates of the first successful landing outcome on a ground pad

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

MIN(Date)
2015-12-22

- The MIN keyword is utilized to calculate the minimum of the DATE column, representing the first date. Additionally, the WHERE keyword, along with the associated condition, filters the results to only include successful ground pad landings

Successful Drone Ship Landing with Payload between 4000 and 6000

- List the names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000

```
%sql SELECT BOOSTER_VERSION FROM SPACEXTBL \
    WHERE (LANDING__OUTCOME = 'Success (drone ship)') AND (PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000);
```

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

- Use the WHERE keyword to filter the results to include boosters that have successfully landed on a drone ship and had a payload mass greater than 4000 but less than 6000, utilizing the BETWEEN keyword to select values within the specified range.

Total Number of Successful and Failure Mission Outcomes

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

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

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

- The COUNT keyword is employed to calculate the total number of mission outcomes. Additionally, the GROUP BY keyword is used to group these results by the type of mission outcome.

Boosters Carried Maximum Payload

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

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

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

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

2015 Launch Records

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

```
%%sql
SELECT substr(Date, 6, 2) AS Month, Landing_Outcome, Booster_Version, Launch_Site FROM SPACEXTABLE
WHERE substr(Date, 0, 5) = '2015' AND Landing_Outcome = 'Failure (drone ship)';
```

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

- In this SQL query, the substr(Date, 6, 2) function is used to extract the month from the Date column, renaming it as "Month". Additionally, the Landing_Outcome, Booster_Version, and Launch_Site columns are selected from the SPACEXTABLE.
- The WHERE clause filters the results to include only records where the substring of the Date column from the 0th to the 5th index is equal to '2015', representing the year 2015. Additionally, it filters for records where the Landing_Outcome column is equal to 'Failure (drone ship)'.

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

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

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

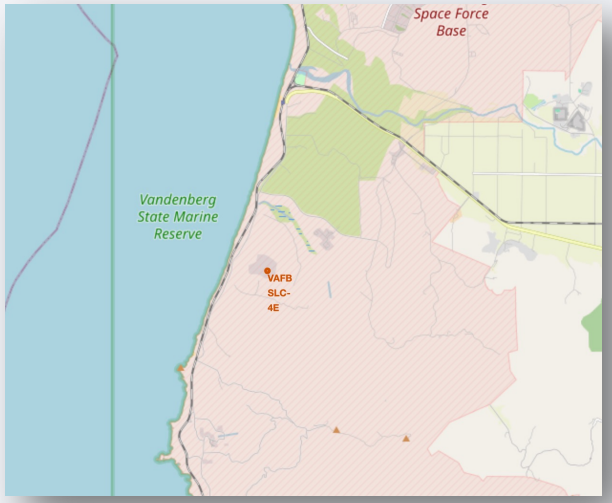
landing__outcome	total_number
No attempt	10
Failure (drone ship)	5
Success (drone ship)	5
Controlled (ocean)	3
Success (ground pad)	3
Failure (parachute)	2
Uncontrolled (ocean)	2
Precluded (drone ship)	1

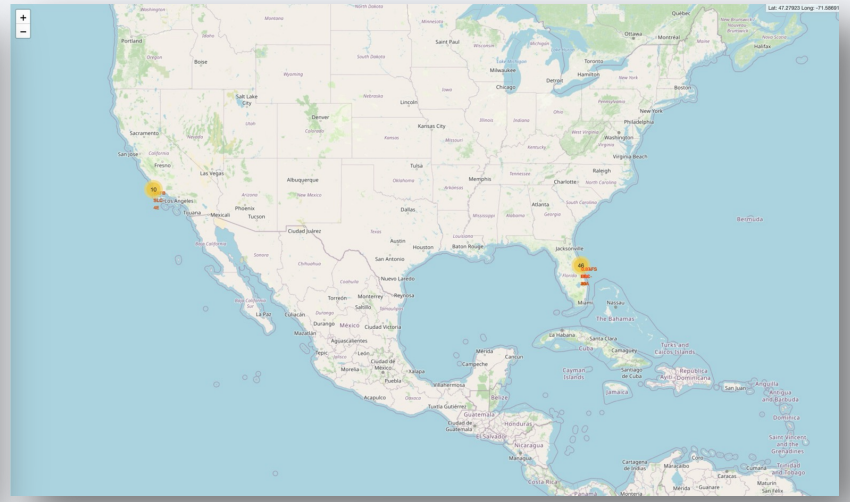
- The WHERE keyword is used with the BETWEEN keyword to filter the results to dates within the specified range.
- The results are then grouped and ordered using the GROUP BY and ORDER BY keywords, respectively. DESC is utilized to specify the descending order.

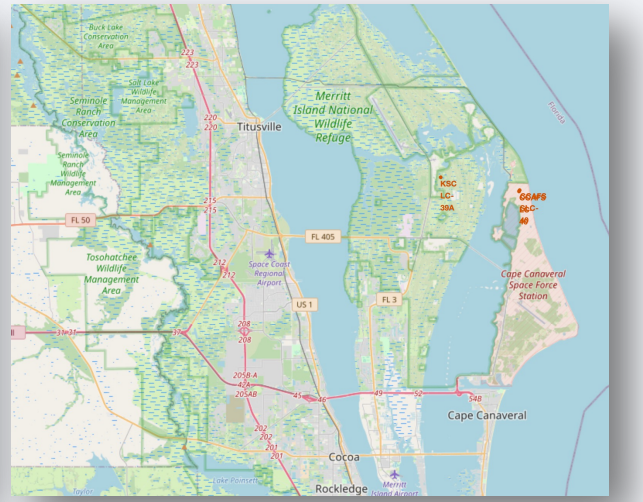
Section 3

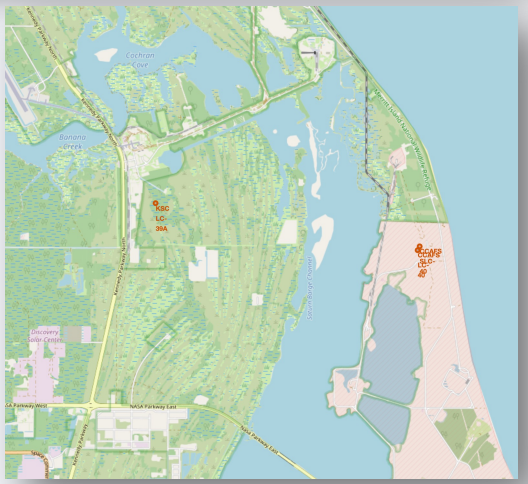
Launch Sites Proximities Analysis

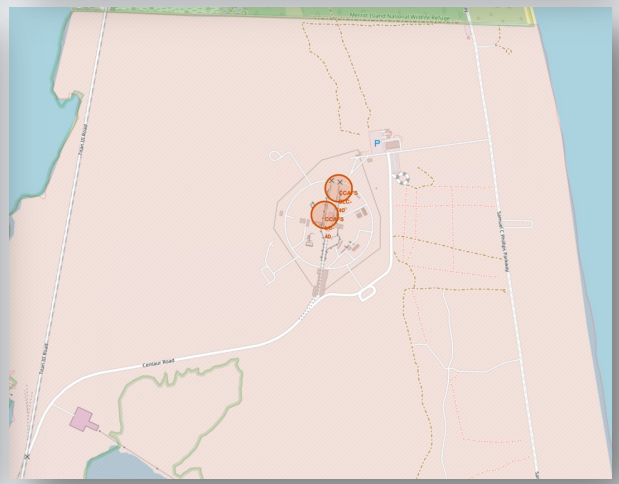
All Launch Sites on a Map





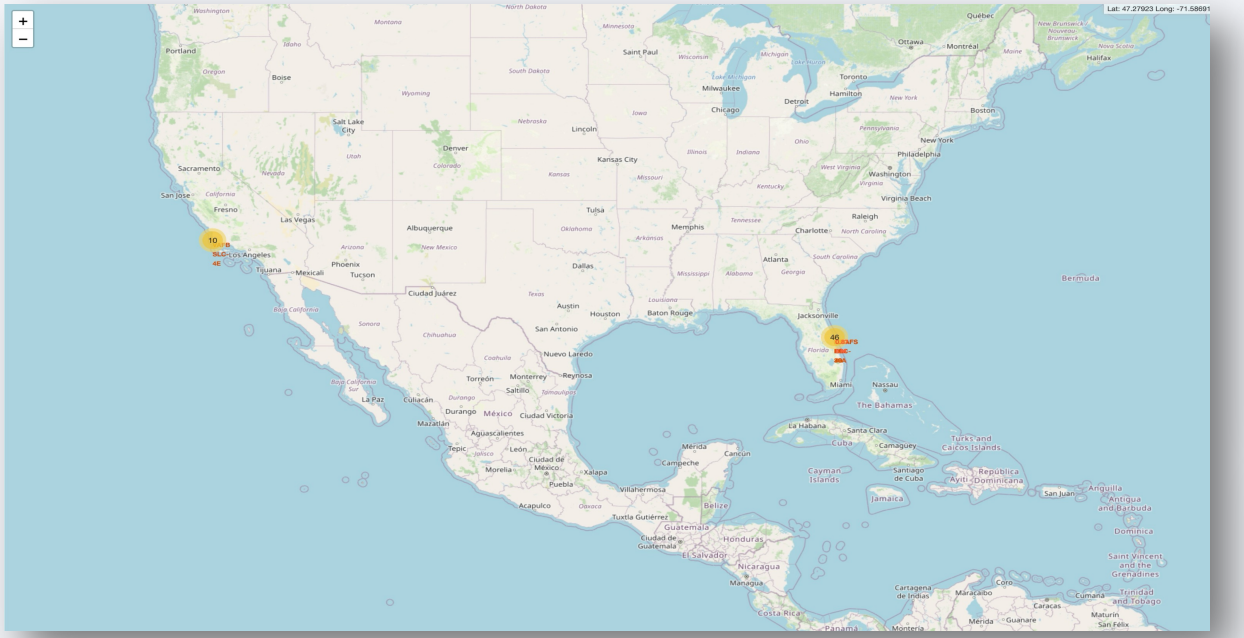


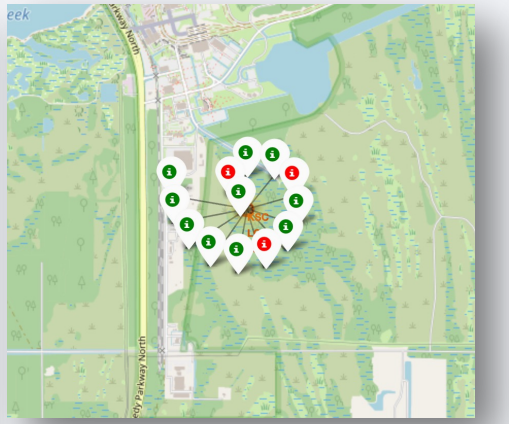


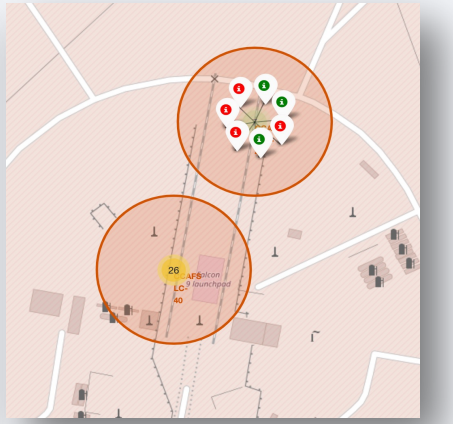


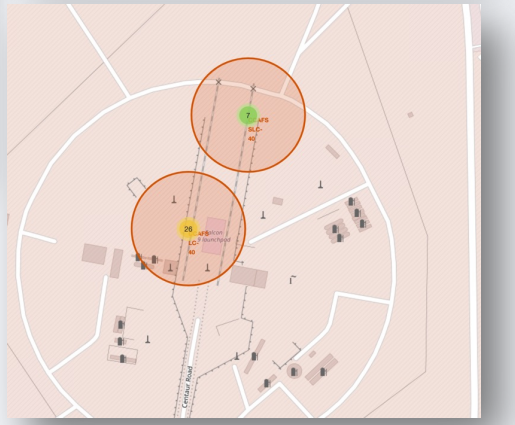
- All SpaceX launch sites are near coast lines of the United States of America, specifically Florida and California

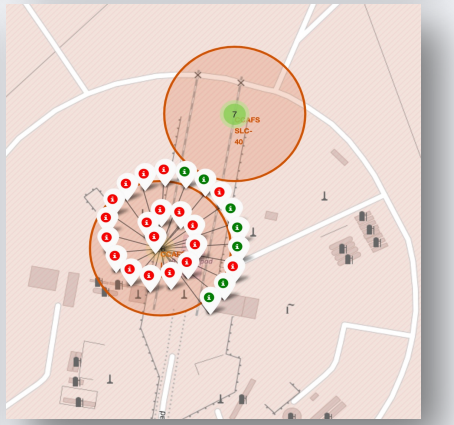
Success/Failed Launches for Each Site

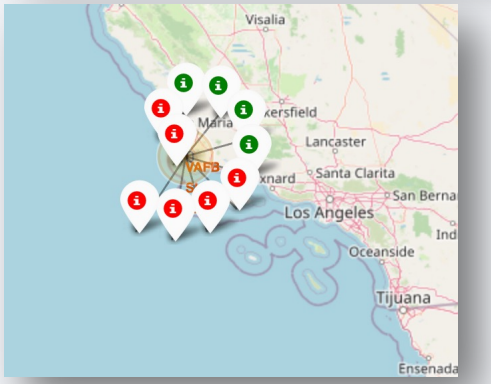












Launches have been grouped into clusters, and annotated with green icons for successful launches, and red icons for failed launches.

Proximity of Launch Sites to Other Points

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

- Are launch sites in close proximity to railways?

  YES. The coastline is only 0.87 km due East.

- Are launch sites in close proximity to highways?

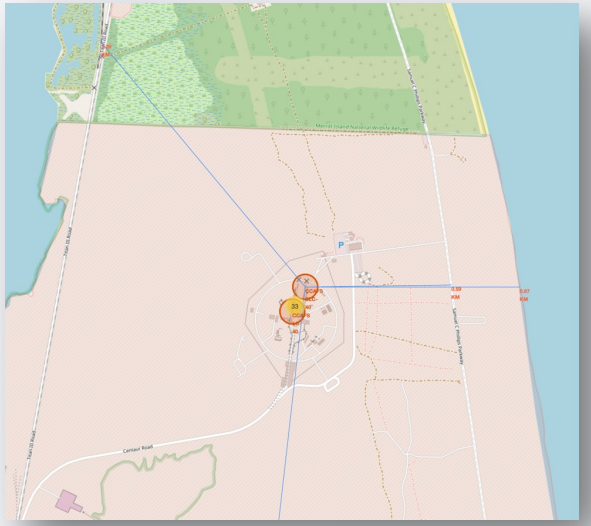
  YES. The nearest highway is only 0.59km away.

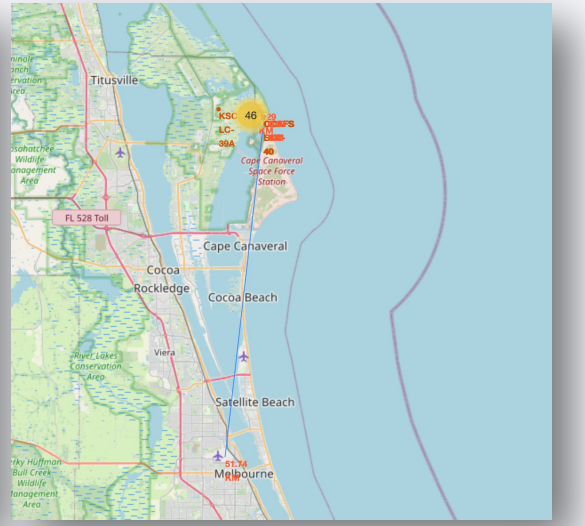
- Are launch sites in close proximity to railways?

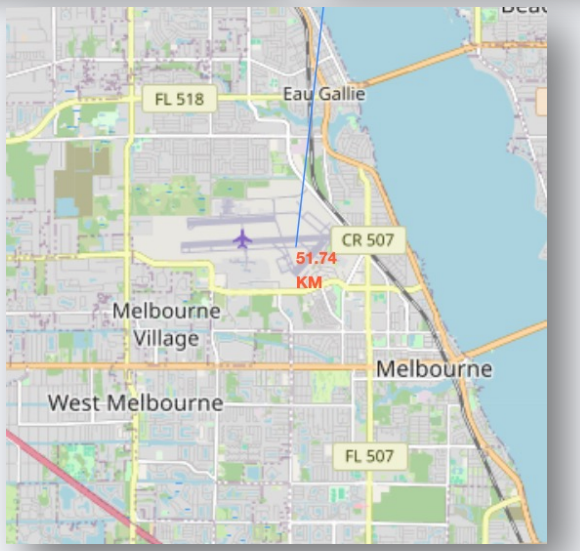
  YES. The nearest railway is only 1.29 km away.

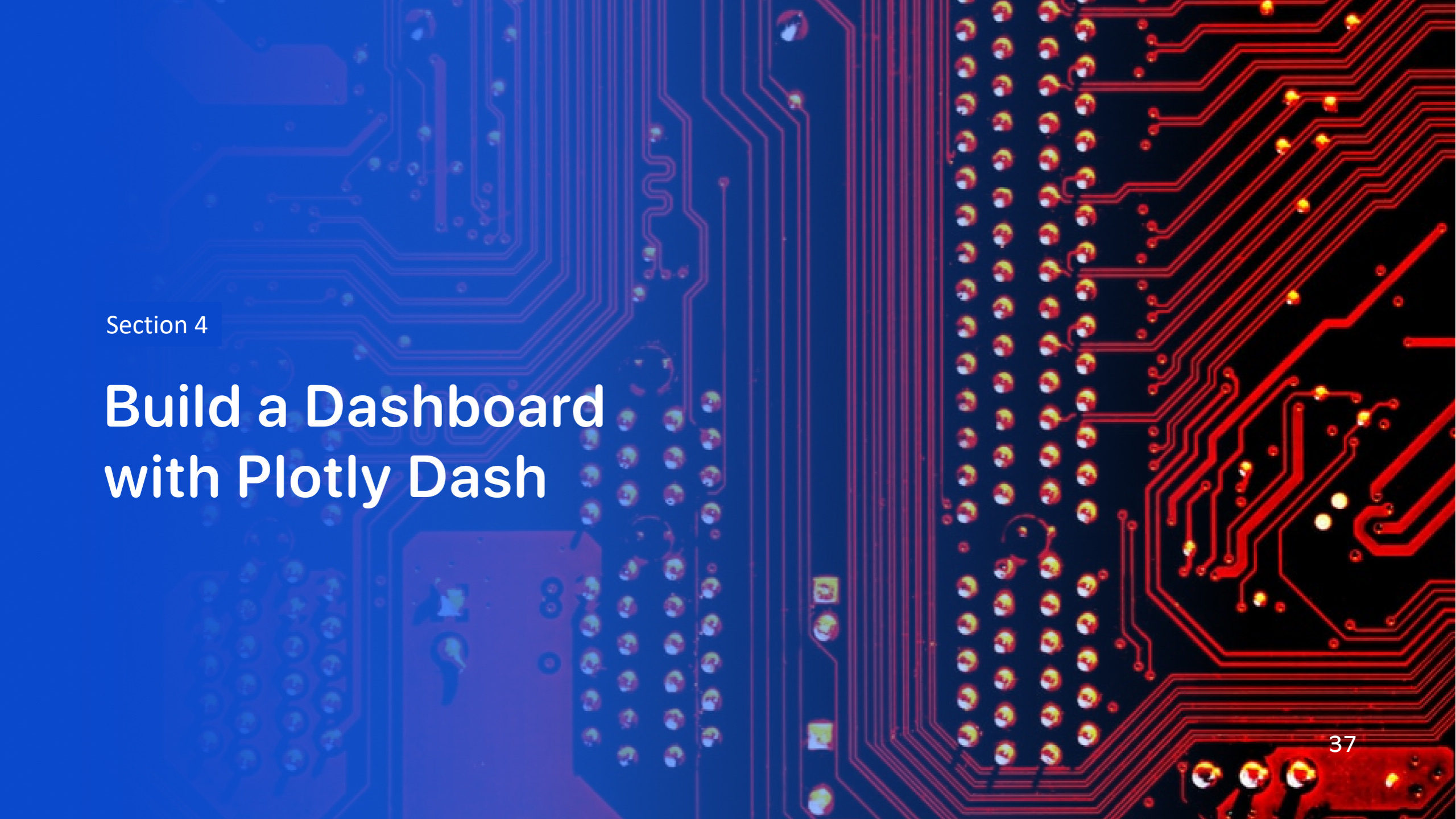
- Do launch sites keep certain distance away from cities?

  YES. The nearest city is 51.74 km away.







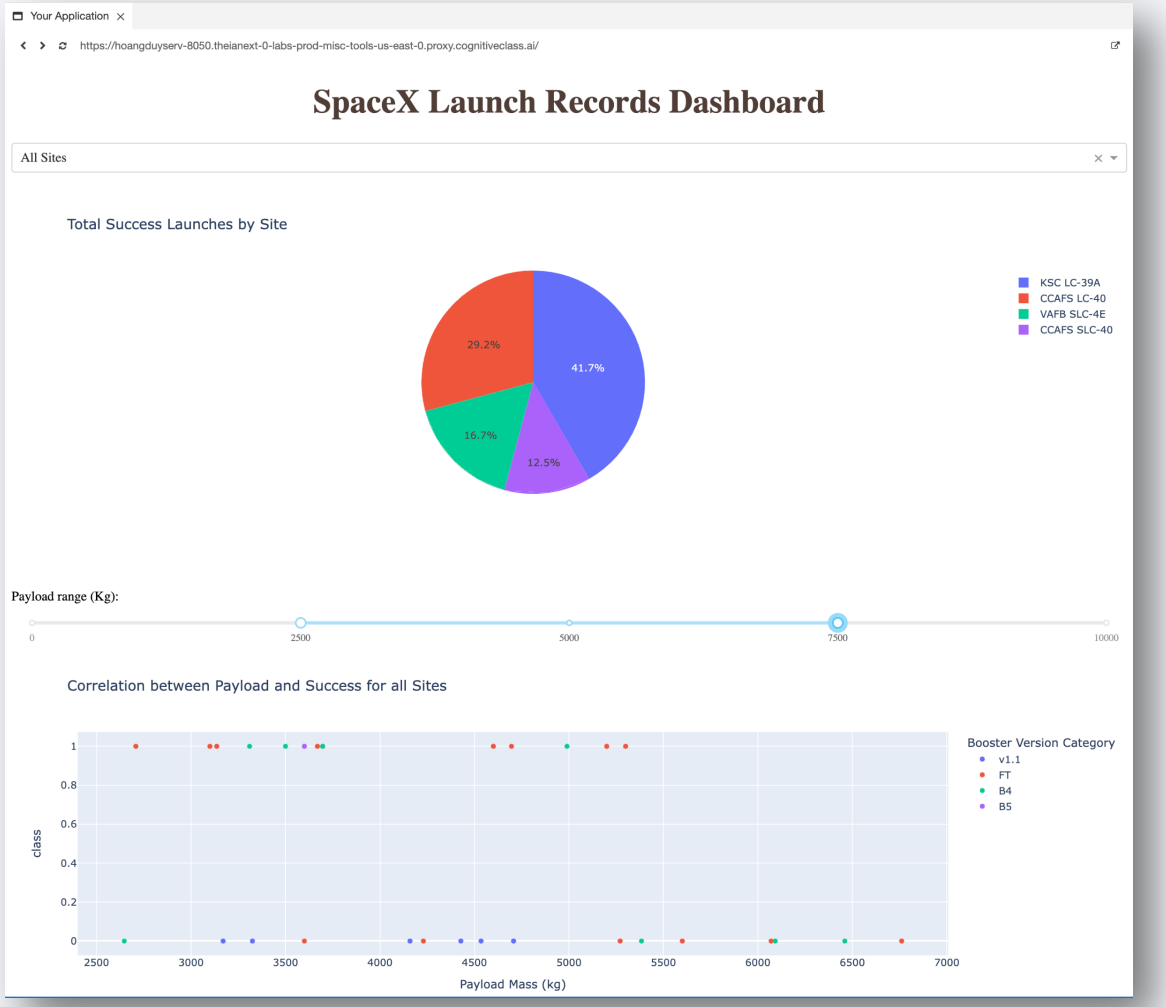


Section 4

Build a Dashboard with Plotly Dash

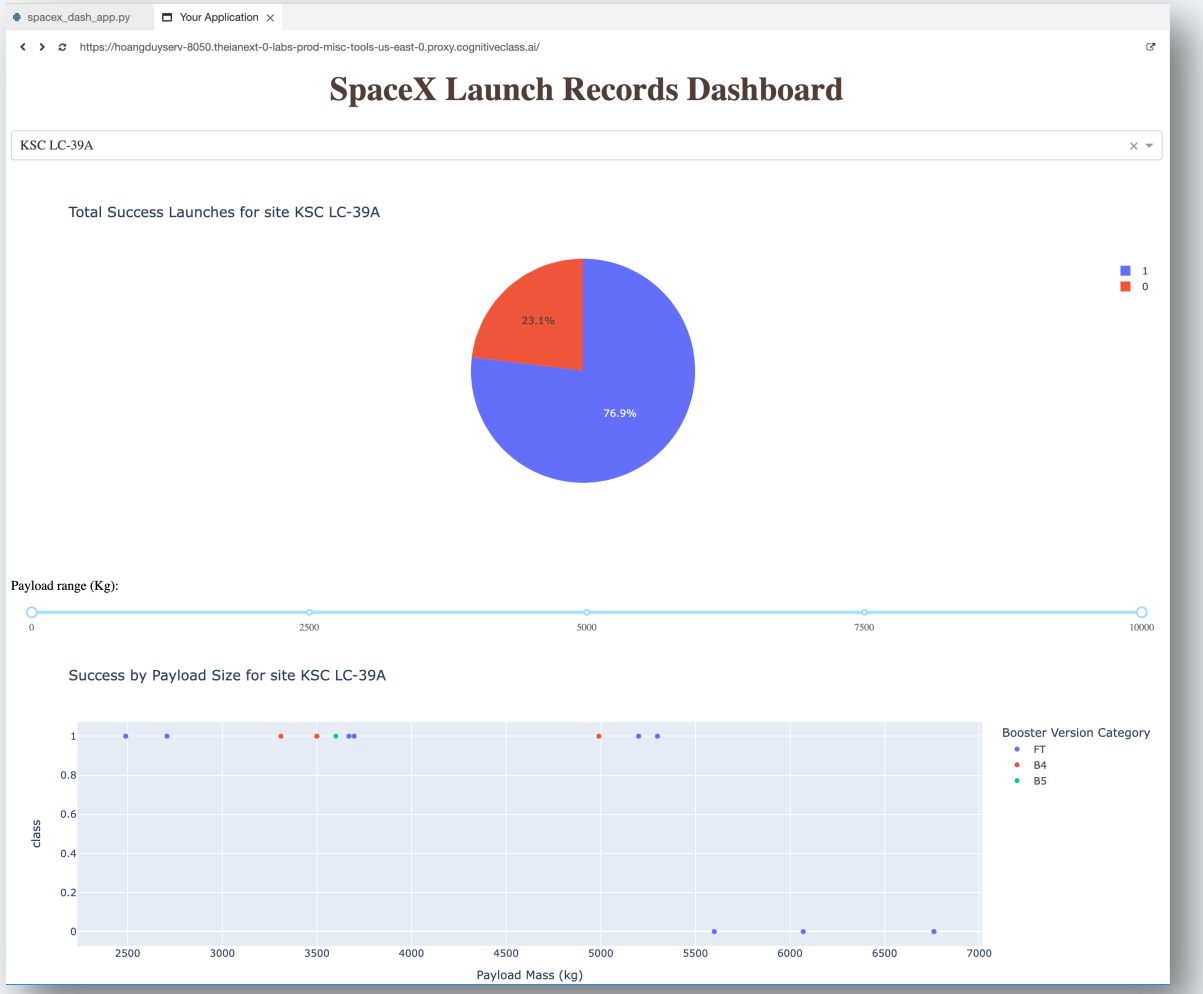
Launch Success Count for All Sites

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

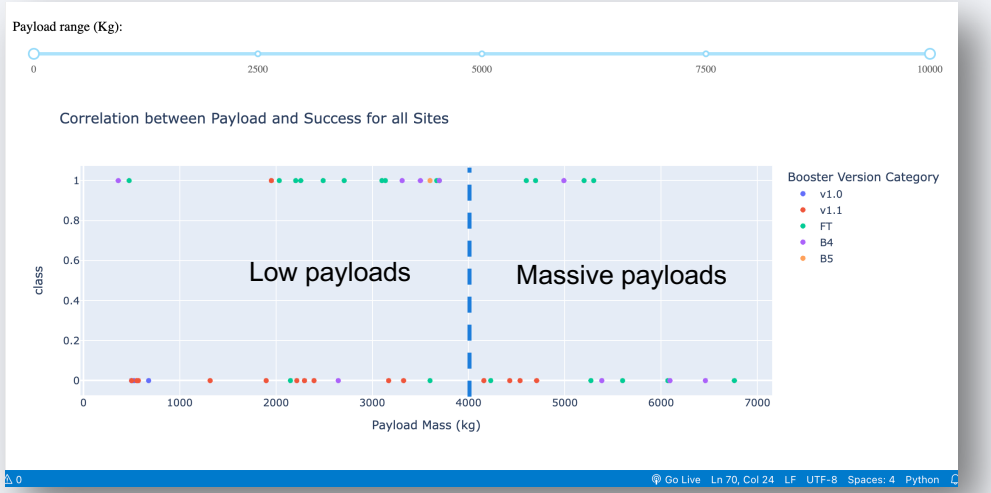


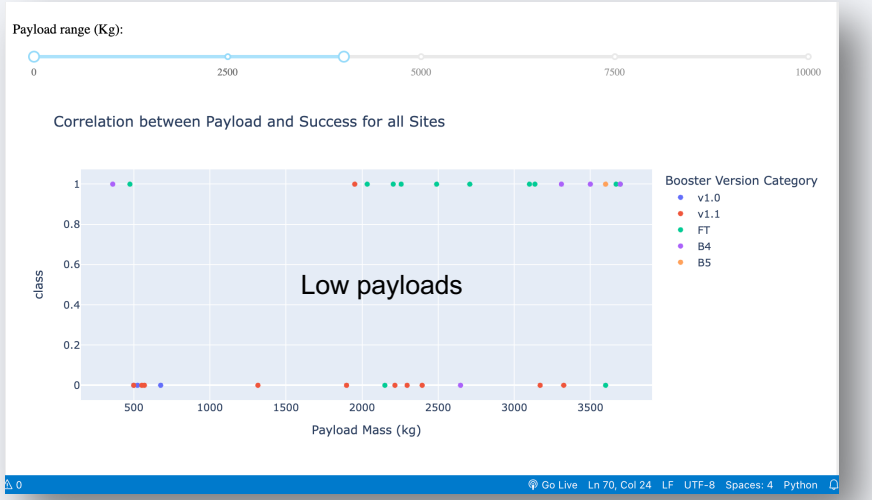
Pie Chart for the Launch Site with Highest Launch Success Ratio

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



Launch Outcome vs. Payload Scatter Plot for All Sites





Plotting the launch outcome vs. payload for all sites reveals a noticeable gap around 4000 kg. Therefore, it is logical to split the data into two ranges:

0 – 4000 kg (low payloads)

4000 – 10000 kg (massive payloads)

From these two plots, it becomes evident that the success rate for massive payloads is lower than that for low payloads. Additionally, it is worth noting that some booster types (v1.0 and B5) have not been launched with massive payloads.

Section 5

Predictive Analysis (Classification)

Classification Accuracy

Plotting the accuracy score and best score for each classification algorithm reveals the following result:

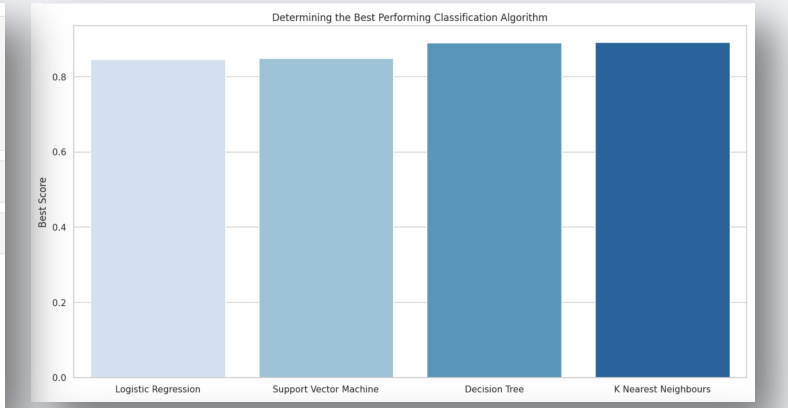
- The K Nearest Neighbours model achieves the highest classification accuracy.
- The accuracy score is 88.89%.
- The best score is 89.11%.

```
algorithms = ['Logistic Regression', 'Support Vector Machine', 'Decision Tree', 'K Nearest Neighbours']

scores = [lr_score, svm_score, tree_score, knn_score]

best_scores = [lr_best_score, svm_best_score, tree_best_score, knn_best_score]

column_names = ['Algorithm', 'Accuracy Score', 'Best Score']
```

```
df = pd.DataFrame(list(zip(algorithms, scores, best_scores)),columns = column_names)
```

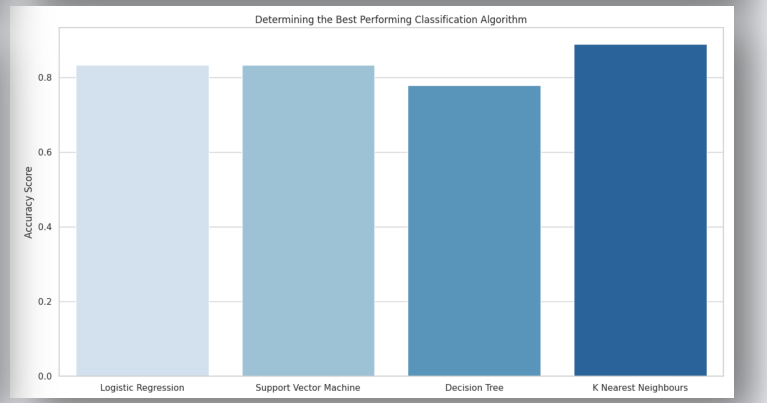
```
df
```

	Algorithm	Accuracy Score	Best Score
0	Logistic Regression	0.833333	0.846429
1	Support Vector Machine	0.833333	0.848214
2	Decision Tree	0.777778	0.889286
3	K Nearest Neighbours	0.888889	0.891071



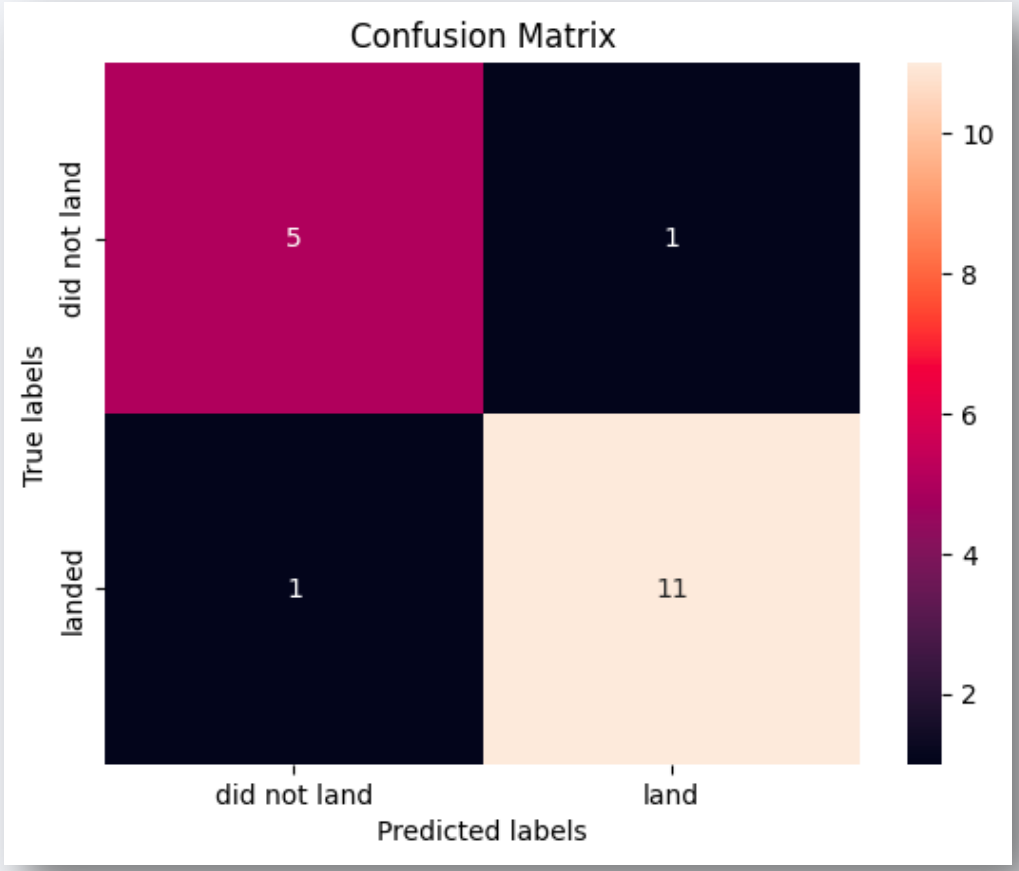

```
algorithms = ['Logistic Regression', 'Support Vector Machine', 'Decision Tree', 'K Nearest Neighbours']
scores = [lr_score, svm_score, tree_score, knn_score]
best_scores = [lr_best_score, svm_best_score, tree_best_score, knn_best_score]

# Organize the data
df = pd.DataFrame({
    'Algorithm': algorithms,
    'Accuracy Score': scores,
    'Best Score': best_scores
})

# Finding the algorithm with the highest accuracy score
max_accuracy = df['Accuracy Score'].max()
max_index = df['Accuracy Score'].idxmax()
most_appropriate_method = df.loc[max_index, 'Algorithm']

print("Most appropriate method:")
print(most_appropriate_method)
```

```
Most appropriate method:
K Nearest Neighbours
```



Confusion Matrix

As demonstrated previously, the best performing classification model is the K Nearest Neighbours model, boasting an accuracy of 88.89%. This exceptional performance is elucidated by the confusion matrix, wherein only 1 out of 18 total results is classified incorrectly. Specifically, this misclassification corresponds to a false positive, as indicated in the top-right corner of the matrix. Notably, the remaining 17 results are correctly classified, with 6 instances of failed landings and 11 instances of successful landings.



Conclusions

- The success rate at a launch site tends to increase with experience, as evidenced by the higher success rates observed as the number of flights increases. Early flights were often unsuccessful, indicating a learning curve.
- Between 2010 and 2013, all landings were unsuccessful, reflecting a period of challenges or technical issues.
- After 2013, the overall success rate improved, although minor dips occurred in 2018 and 2020.
- Since 2016, there has been a consistent success rate of over 50%, indicating improved performance.
- Orbit types ES-L1, GEO, HEO, and SSO have achieved a 100% success rate, although the success rate for SSO is particularly impressive, considering there have been multiple successful flights.
- The success rate in orbit types PO, ISS, and LEO is higher with heavier payloads, while VLEO launches typically involve heavier payloads.
- KSC LC-39 A has the most successful launches, accounting for 41.7% of the total successful launches, with a success rate of 76.9%.
- The success rate for massive payloads (over 4000 kg) is lower than that for low payloads.
- The best performing classification model is the K Nearest Neighbours model, achieving an accuracy of 88.89%.

Appendix

- Custom functions were developed to retrieve the required information efficiently.
- Custom logic was implemented to clean the data effectively, ensuring accuracy and consistency.

We will now use the API again to get information about the launches using the IDs given for each launch. Specifically we will be using columns `rocket`, `payloads`, `launchpad`, and `cores`.

```
# 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
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
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)]
```

GitHub Click here

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

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

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

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

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

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

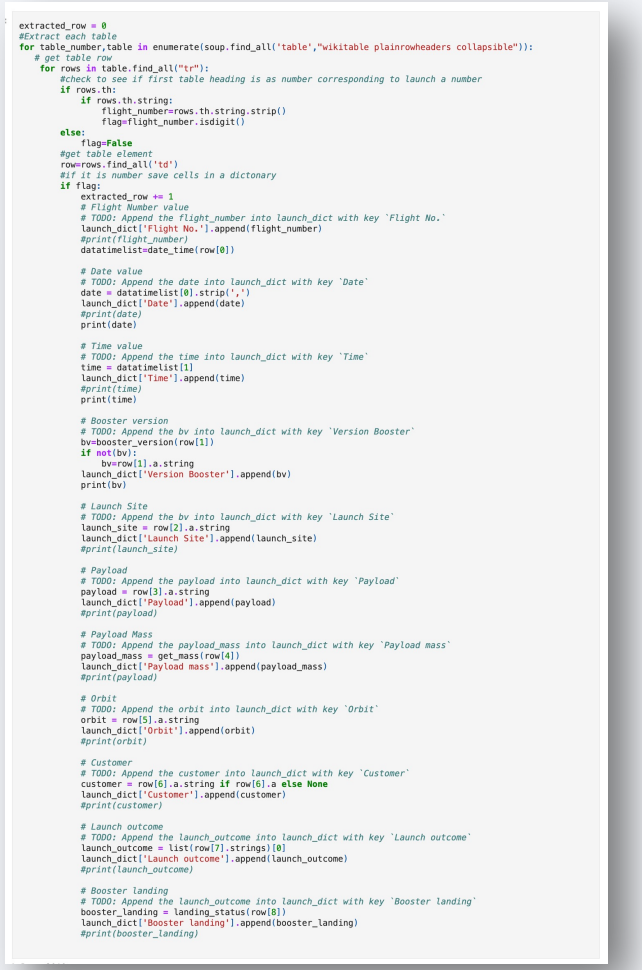
From `cores` we would like to learn the outcome of the landing, the type of the landing, number of flights with that core, whether gridfins were used, wheter the core is reused, wheter legs were used, the landing pad used, the block of the core which is a number used to seperate version of cores, the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

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

Appendix

- Custom functions were created for web scraping, enabling systematic extraction of data from web sources.
- Custom logic was devised to populate the launch_dict with values extracted from the launch tables, ensuring comprehensive data collection and organization.

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

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

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


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


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

    column_name = ' '.join(row.contents)

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



```
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders collapsible")):
   # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number corresponding to launch a number
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
        else:
            flag=False
        #get table element
        row=rows.find_all('td')
        #if it is number save cells in a dictonary
        if flag:
            extracted_row += 1
            # Flight Number value
            # TODO: Append the flight_number into launch_dict with key `Flight No.`
            launch_dict['Flight No.'].append(flight_number)
            #print(flight_number)
            datimelist=date_time(row[0])

            # Date value
            # TODO: Append the date into launch_dict with key `Date`
            date = datimelist[0].strip(',')
            launch_dict['Date'].append(date)
            #print(date)
            print(date)

            # Time value
            # TODO: Append the time into launch_dict with key `Time`
            time = datimelist[1]
            launch_dict['Time'].append(time)
            #print(time)
            print(time)

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

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

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

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

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

            # Customer
            # TODO: Append the customer into launch_dict with key `Customer`
            customer = row[6].a.string if row[6].a else None
            launch_dict['Customer'].append(customer)
            #print(customer)

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

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

Thank you!