IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Eszter Tóth
2. July, 2023

Outline

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

Executive Summary

Methodologies

The methodologies used in the projects includes **data collection, data wrangling, data processing, exploratory data analysis, visual analytics, and predictive analysis:**

- Data was collected from various online resources such as **APIs, public databases, and web scraping**. The collected data was then wrangled, a process that involved cleaning, structuring, and enriching the raw data to make it suitable for analysis.

- The data was processed using numerous tools and techniques. This included data **encoding for categorical features, normalization for numerical features, and handling of missing values**. Furthermore, data was partitioned into training and testing sets to enable effective model evaluation later in the process.

- **Exploratory Data Analysis (EDA)** was conducted using SQL and visualization tools to **understand the underlying structure of the data, identify outliers, and detect patterns or trends**. This was critical for gaining insights and guiding the subsequent predictive analysis stage.

- In terms of interactive visual analytics, tools like Folium and Plotly Dash were used. Folium was employed to create interactive maps for geographical data visualization, while Plotly Dash was used to build interactive dashboards that aid in understanding complex data narratives.

- For predictive analysis, a range of classification models, including but not limited to **Decision Trees, Random Forests, and Support Vector Machines** were built. Each of these models was tuned using methods like GridSearchCV to optimize hyperparameters and maximize model performance.

Results

These models were then evaluated based on metrics such as **accuracy, precision, recall, and F1-score**. The process of building, tuning, and evaluating these classification models gave valuable insights into the behavior of these algorithms and their suitability for different types of classification problems.

Introduction

- Project background and context
- Problems you want to find answers

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - Data is collected from the SpaceX API by making an HTTP GET request to the API endpoint, receiving the JSON response, extracting the relevant information, and storing it for further processing or analysis.
- Perform data wrangling
  - SpaceX API was processed by extracting relevant information such as flight number, mission name, and launch success from the API response, categorizing the launches based on specific criteria or key phrases, and storing the processed data in appropriate data structures for further analysis or presentation.
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - Prepare the data, select an appropriate algorithm, train the model using the training dataset, tune the hyperparameters for optimal performance, evaluate the model using appropriate metrics on the testing dataset, and compare different models before selecting the best-performing one for deployment.

Data Collection

When collecting SpaceX data, there are two primary approaches: utilizing the SpaceX API or scraping data from Wikipedia.

SpaceX API:

Collecting data from the SpaceX API involves making HTTP requests to specific endpoints provided by SpaceX. You can retrieve structured and up-to-date information on launches, rockets, capsules, and more. The API provides a direct and reliable source of data, ensuring accuracy and consistency. However, the API may have limitations, such as rate limits and restricted access to certain data fields.

Web scraping from Wikipedia:

On the other hand, scraping data from Wikipedia involves extracting information from HTML web pages. With Wikipedia being a collaborative platform, the data may be community-driven and subject to potential inconsistencies or inaccuracies. Scraping requires parsing and extracting relevant data from HTML, which can be more complex and time-consuming compared to accessing structured data through an API.

While the SpaceX API provides authoritative and structured data, it may have limitations on available fields or historical data. Scraping Wikipedia allows access to a broader range of information, including historical context and details not available through the API. However, scraping requires careful handling of HTML structure changes, potential legal considerations, and the need for additional data cleaning and preprocessing steps.

The choice between API usage and scraping depends on the specific data needs, time constraints, available resources, and the level of control and accuracy required for the data collection process.

Data Collection – SpaceX API

1. Send a GET request to the SpaceX API endpoint (`https://api.spacexdata.com/v4/launches`).

2. Receive the JSON response containing the launch data.

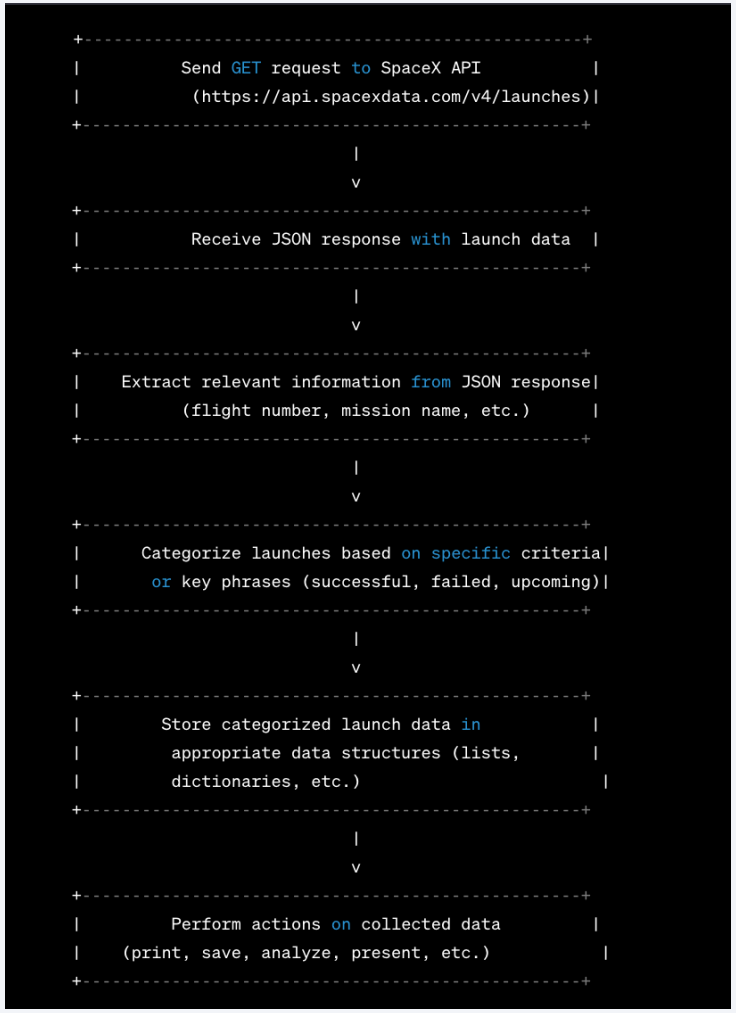
3. Extract the relevant information from the JSON response, such as flight number, mission name, and launch success.

4. Categorize the launches based on specific criteria or key phrases, such as successful launches, failed launches, or upcoming launches.

5. Store the categorized launch data in appropriate data structures, such as lists or dictionaries.

6. Perform actions on the collected data, such as printing, saving, analyzing, or presenting it as needed.

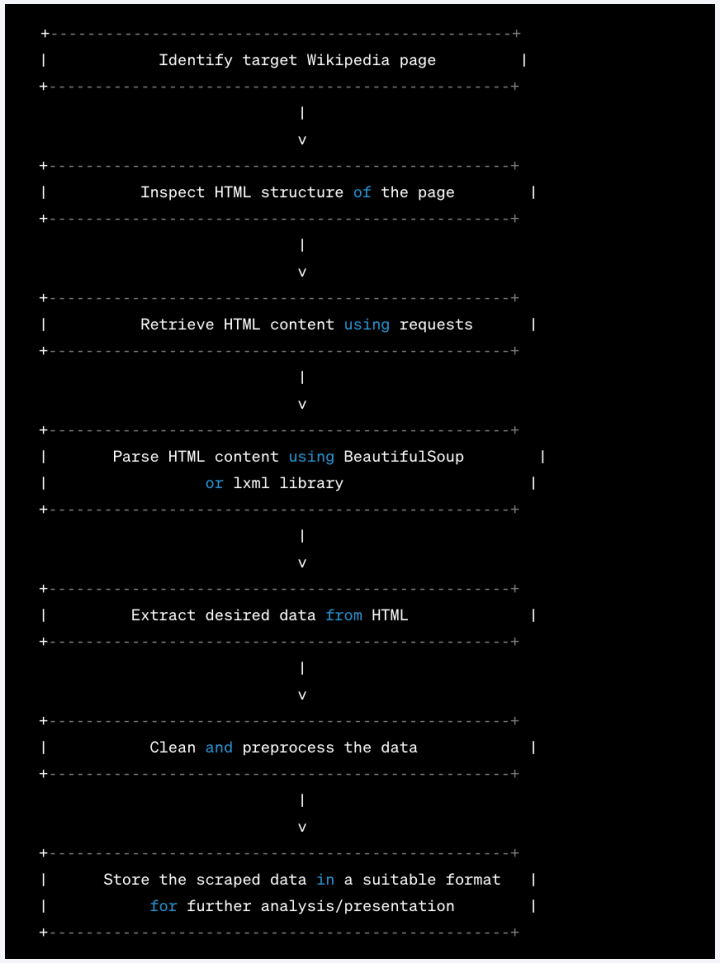
- https://github.com/t-eszter/IBMAppliedDataScienceCapstone/blob/main/01-jupyter-labs-webscraping.ipynb



Data Collection - Scraping

- Identify the target Wikipedia page containing the desired data
- Inspect the HTML structure of the page to locate the relevant elements and tags
- Retrieve the HTML content of the page using a library like requests.
- Parse the HTML content using an HTML parsing library such as BeautifulSoup or lxml.
- Extract the desired data by targeting specific elements or tags within the parsed HTML.
- Clean and preprocess the extracted data as needed.
- Store the scraped data in a suitable format for further analysis or presentation.

- https://github.com/t-eszter/IBMAppliedDataScienceCapstone/blob/main/01-jupyter-labs-webscraping.ipynb



Data Wrangling

- 1. We loaded the obtained dataset
- 2. We used method `value_counts()` on the column `LaunchSite` to determine the number of launches on each site.
- 3. With the `.value_counts()` method we determined the number and occurrence of each orbit in the column `Orbit`
- `4. With the .value_counts()` on the column `Outcome` to we determined the number of `landing_outcomes`
- `5.` We classified the bad and good outcomes from the Outcome column
- 6. We calculated the success rate with the .mean() of the outcomes
- https://github.com/t-eszter/IBMAppliedDataScienceCapstone/blob/main/labs-jupyter-spacex-Data%20wrangling.ipynb

EDA with Data Visualization

- We used 'catplot' to visualize the relationship between Flight Number and Launch Site, because it handles categorical data well.
- We used scatter point chart to visualize the relationship between Payload and Launch Site, the relationship between FlightNumber and Orbit type, and the relationship between Payload and Orbit type, because it's effective for visualizing relationships between two continuous variables
- We used bar chart to visualize the relationship between success rate of each orbit type, because it's useful for comparing different categories
- We used line chart to visualize the launch success yearly trend, because it's ideal for showing trends over time

- https://github.com/t-eszter/IBMAppliedDataScienceCapstone/blob/main/jupyter-labs-eda-dataviz.ipynb

EDA with SQL

- *Display the names of the unique launch sites in the space mission: %sql SELECT DISTINCT Launch_Site FROM SPACEXTBL;*
- *Display 5 records where launch sites begin with the string 'CCA': %sql SELECT * FROM SPACEXTBL WHERE Launch_Site LIKE 'CCA%' LIMIT 5;*
- *Display the total payload mass carried by boosters launched by NASA (CRS): %sql SELECT SUM(Payload_Mass__kg_) FROM SPACEXTBL WHERE Customer = 'NASA (CRS)';*
- *Display average payload mass carried by booster version F9 v1.1: %sql SELECT AVG(Payload_Mass__kg_) FROM SPACEXTBL WHERE Booster_Version = 'F9 v1.1';*
- *List the date when the first succesful landing outcome in ground pad was acheived: %sql SELECT MIN(Date) FROM SPACEXTBL WHERE Landing_Outcome = 'Success (ground pad)';*
- *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 Booster_Version FROM SPACEXTBL WHERE Landing_Outcome = 'Success (drone ship)' AND Payload_Mass__kg_ > 4000 AND Payload_Mass__kg_ < 6000;*
- *List the total number of successful and failure mission outcomes: %sql SELECT Mission_Outcome, COUNT(*) FROM SPACEXTBL GROUP BY Mission_Outcome;*
- *List the names of the booster_versions which have carried the maximum payload mass. Use a subquery: %sql SELECT Booster_Version FROM SPACEXTBL WHERE Payload_Mass__kg_ = (SELECT MAX(Payload_Mass__kg_) FROM SPACEXTBL);*

- https://github.com/t-eszter/IBMAppliedDataScienceCapstone/blob/main/jupyter-labs-eda-sql-coursera_sqllite.ipynb

Build an Interactive Map with Folium

- https://github.com/t-eszter/IBMAppliedDataScienceCapstone/blob/main/lab_jupyter_launch_site_location.ipynb

Predictive Analysis (Classification)

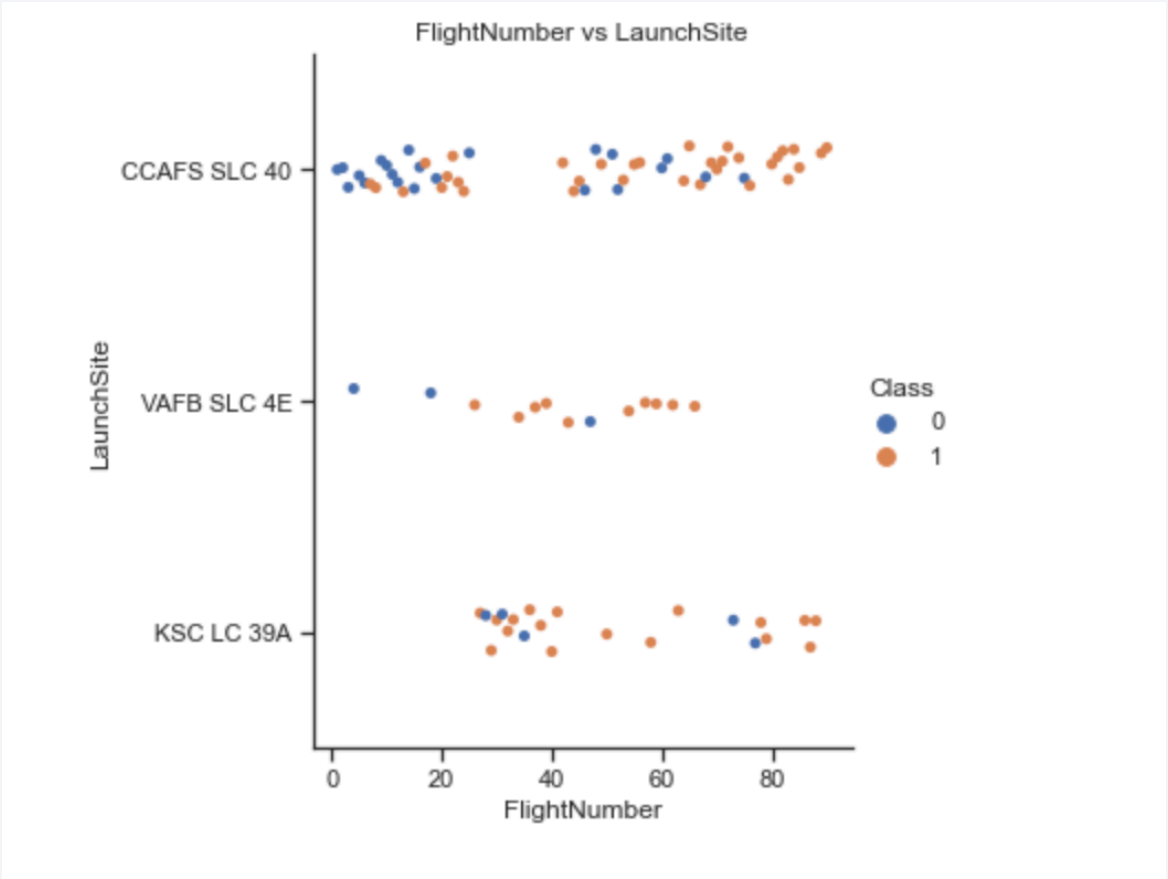
- https://github.com/t-eszter/IBMAppliedDataScienceCapstone/blob/main/module_4_SpaceX_Machine_Learning_Prediction_Part_5.jupyterlite.ipynb

Section 2

Insights drawn from EDA

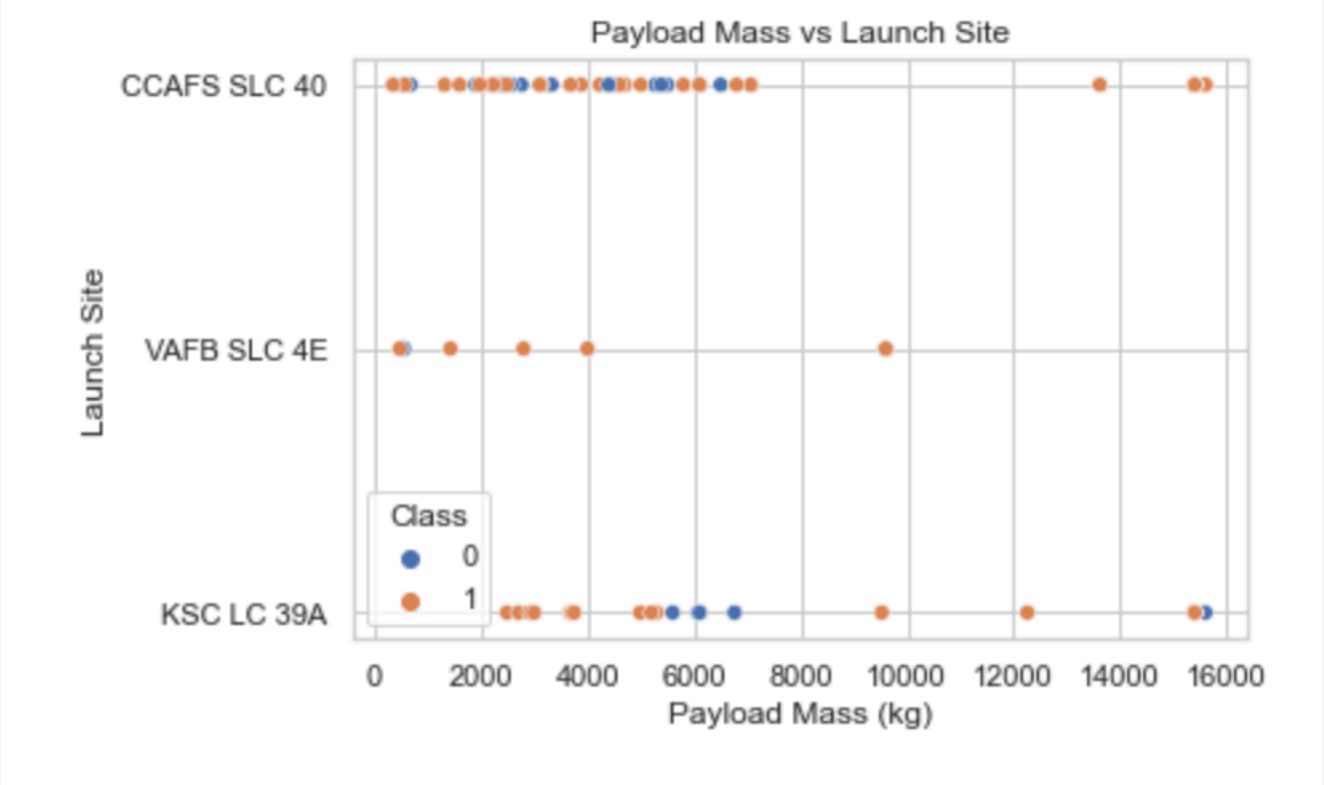
Flight Number vs. Launch Site

- This scatter plot represents the relationship between flight numbers (x-axis) and launch sites (y-axis), with the color of the points indicating the class value.
- This chart allows the visualization of patterns or relationships between flight numbers, launch sites, and class values. It can help identify any trends or groupings in the data, such as flights with similar characteristics being clustered together.
- It uses the seaborn library's `catplot()` function to create the plot. The title and axis labels are set, and the plot is displayed using `plt.show()`.



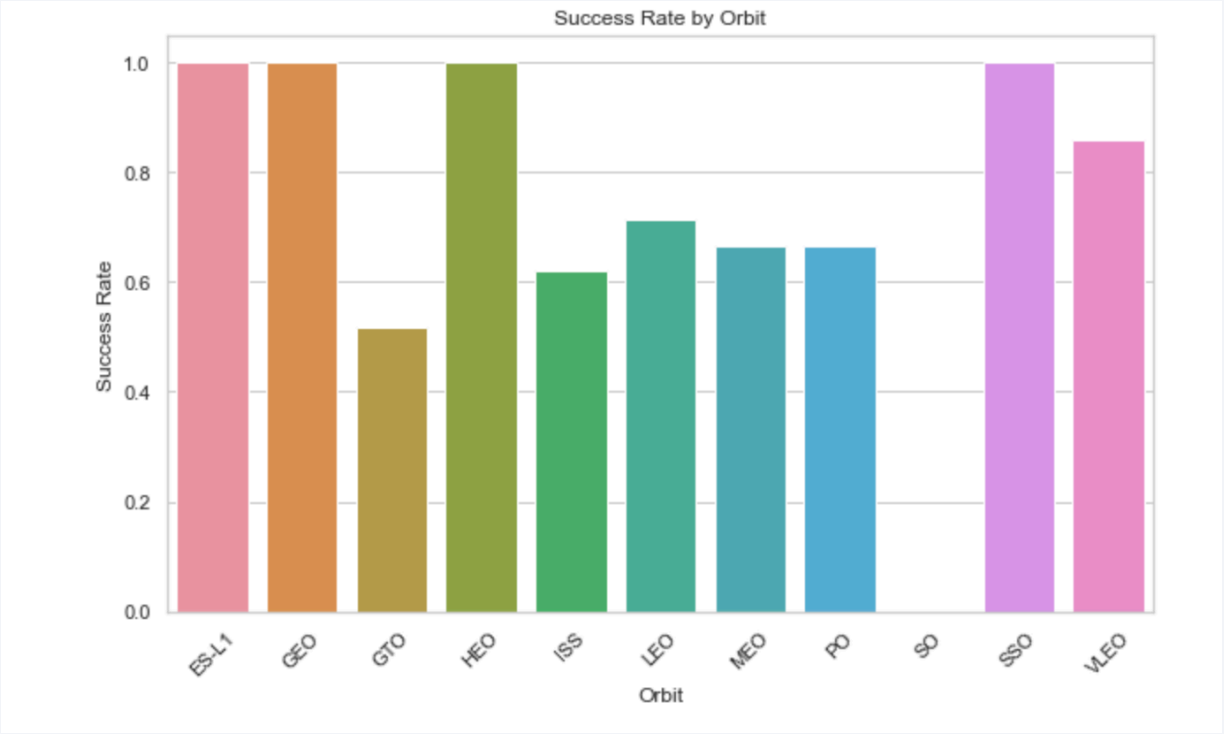
Payload vs. Launch Site

- This scatter plot represents the relationship between payload mass (x-axis) and launch site (y-axis), with the color of the points indicating the class value.
- Each point on the plot corresponds to a specific launch, with its position on the x-axis representing the payload mass and its position on the y-axis representing the launch site. The color of each point indicates the class value, which is another categorical variable.
- It uses the seaborn library's `scatterplot()` function to create the plot. The title and axis labels are set, and the plot is displayed using `plt.show()`.



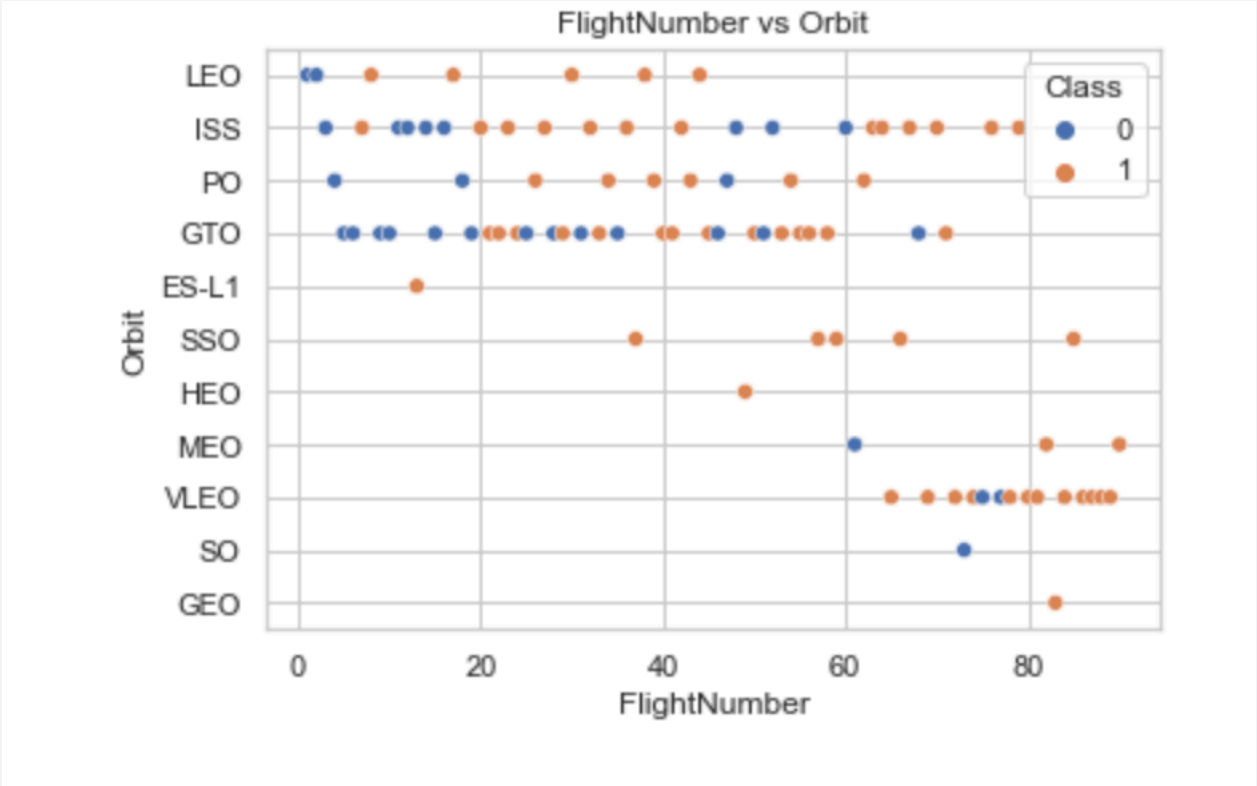
Success Rate vs. Orbit Type

- This bar chart represents the success rate of different orbits. It calculates the mean of the 'Class' column (which likely represents the success rate) grouped by the 'Orbit' column.
- The groupby() method is used on the DataFrame df to group the data by the 'Orbit' column. Then, the mean() function is applied to the 'Class' column to calculate the average success rate for each orbit. The resulting data is stored in the variable orbit_success_rate.
- It uses the seaborn library's barplot() function to create the chart.



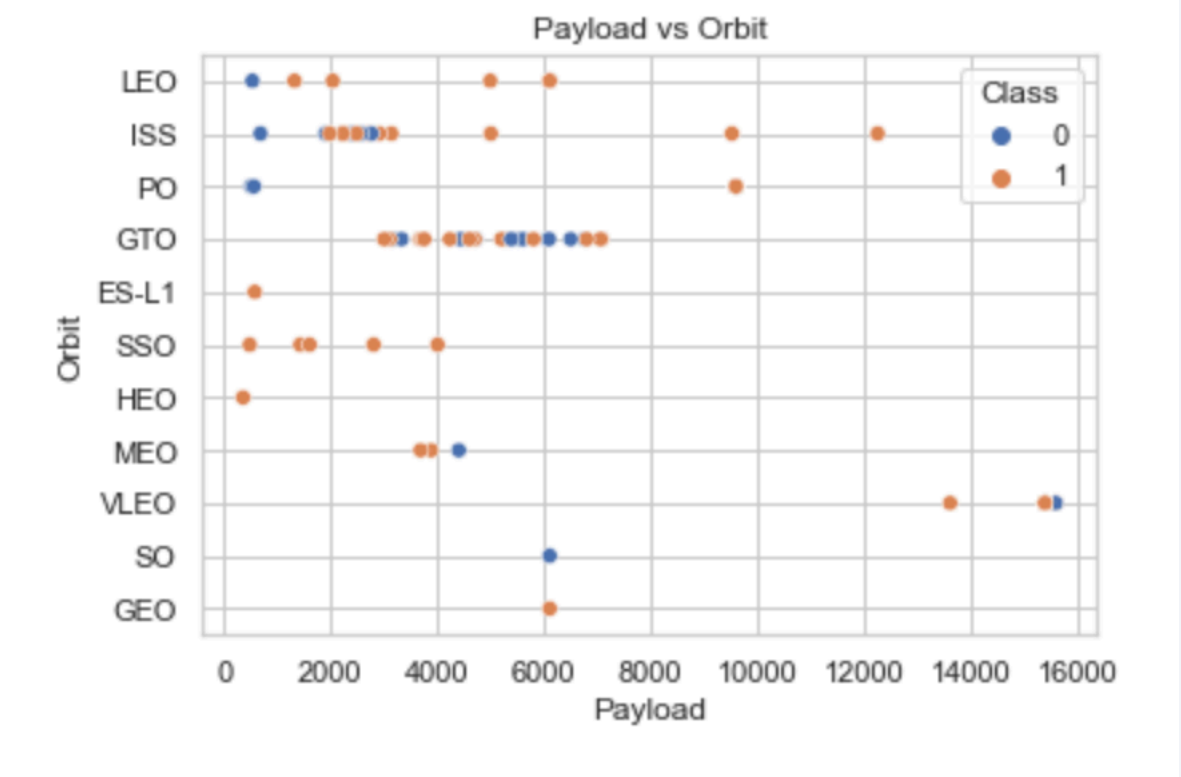
Flight Number vs. Orbit Type

- This scatter plot represents the relationship between flight numbers (x-axis) and orbits (y-axis), with the color of the points indicating the class value.
- Each point on the plot represents a specific flight, with its position on the x-axis representing the flight number and its position on the y-axis representing the orbit. The color of each point indicates the class value, which is another categorical variable.
- It uses the seaborn library's scatterplot() function to create the plot.



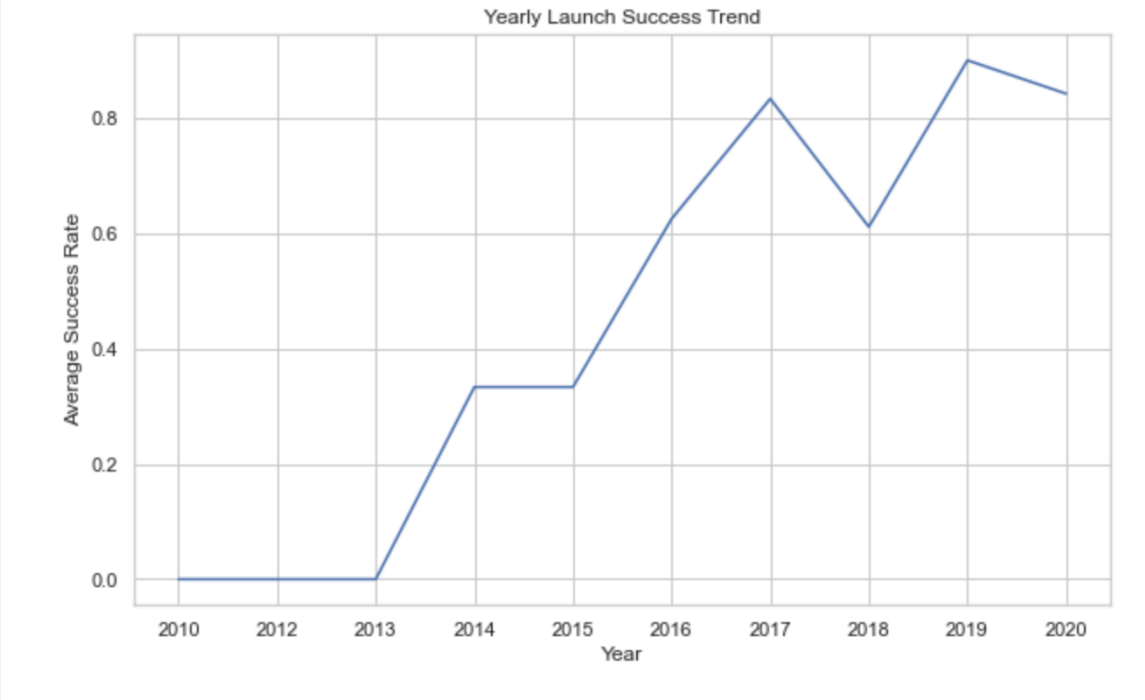
Payload vs. Orbit Type

- This scatter plot shows the relationship between payload mass (x-axis) and orbits (y-axis), with the class value represented by the color of the points.
- Each point on the plot represents a specific launch, with its position on the x-axis representing the payload mass and its position on the y-axis representing the orbit. The color of each point indicates the class value, which is another categorical variable.
- It uses seaborn's scatterplot() function, with the style set to "whitegrid", title set to "Payload vs Orbit", and axis labels specified as "Payload" and "Orbit".



Launch Success Yearly Trend

- This line chart displays the yearly trend of the average launch success rate.
- The years are extracted from the 'Date' column using the function `Extract_year(date)`. The data is then grouped by 'Year', and the mean of the 'Class' column is calculated to get the average success rate for each year.
- The line chart is created using seaborn's `lineplot()` function, with the style set to "whitegrid", title set to "Yearly Launch Success Trend", and axis labels specified as "Year" and "Average Success Rate".



All Launch Site Names

- %sql SELECT DISTINCT Launch_Site FROM SPACEXTBL;
- SELECT: This keyword is used to retrieve data from the database.
- DISTINCT: This keyword ensures that only unique values are returned, eliminating any duplicates from the result set.
- Launch_Site: It is the column name from which we want to retrieve distinct values.
- FROM SPACEXTBL: It specifies the table name "SPACEXTBL" from which the data is being retrieved.

Launch_Site
CCAFS LC-40
VAFB SLC-4E
KSC LC-39A
CCAFS SLC-40
None

Launch Site Names Begin with 'CCA'

- %sql SELECT * FROM SPACEXTBL WHERE Launch_Site LIKE 'CCA%' LIMIT 5;
- SELECT *: This keyword is used to retrieve all columns from the table.
- FROM SPACEXTBL: It specifies the table name "SPACEXTBL" from which the data is being retrieved.
- WHERE Launch_Site LIKE 'CCA%': This is the condition to filter the rows. It selects rows where the "Launch_Site" column starts with the letters 'CCA'.
- LIMIT 5: This limits the result to only the first 5 rows that meet the specified condition.

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
06/04/2010	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0.0	LEO	SpaceX	Success	Failure (parachute)
12/08/2010	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0.0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
22/05/2012	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525.0	LEO (ISS)	NASA (COTS)	Success	No attempt
10/08/2012	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500.0	LEO (ISS)	NASA (CRS)	Success	No attempt
03/01/2013	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677.0	LEO (ISS)	NASA (CRS)	Success	No attempt

Total Payload Mass

- %sql SELECT SUM(Payload_Mass__kg_) FROM SPACEXTBL WHERE Customer = 'NASA (CRS)';
- SELECT SUM(Payload_Mass__kg_): This part of the query selects the sum of the "Payload_Mass__kg_" column from the table.
- FROM SPACEXTBL: It specifies the table name "SPACEXTBL" from which the data is being retrieved.
- WHERE Customer = 'NASA (CRS)': This is the condition to filter the rows. It selects rows where the "Customer" column is equal to 'NASA (CRS)'.

SUM(Payload_Mass__kg_)
45596.0

Average Payload Mass by F9 v1.1

- %sql SELECT AVG(Payload_Mass__kg_) FROM SPACEXTBL WHERE Booster_Version = 'F9 v1.1';
- SELECT AVG(Payload_Mass__kg_): This part of the query selects the average of the "Payload_Mass__kg_" column from the table.
- FROM SPACEXTBL: It specifies the table name "SPACEXTBL" from which the data is being retrieved.
- WHERE Booster_Version = 'F9 v1.1': This is the condition to filter the rows. It selects rows where the "Booster_Version" column is equal to 'F9 v1.1'.

AVG(Payload_Mass__kg_)
2928.4

First Successful Ground Landing Date

- %sql SELECT MIN(Date) FROM SPACEXTBL WHERE Landing_Outcome = 'Success (ground pad)';
- SELECT MIN(Date): This part of the query selects the minimum value from the "Date" column.
- FROM SPACEXTBL: It specifies the table name "SPACEXTBL" from which the data is being retrieved.
- WHERE Landing_Outcome = 'Success (ground pad)': This is the condition to filter the rows. It selects rows where the "Landing_Outcome" column is equal to 'Success (ground pad)';

MIN(Date)
01/08/2018

Successful Drone Ship Landing with Payload between 4000 and 6000

- %sql SELECT Booster_Version FROM SPACEXTBL WHERE Landing_Outcome = 'Success (drone ship)' AND Payload_Mass__kg_ > 4000 AND Payload_Mass__kg_ < 6000;
- SELECT Booster_Version: This part of the query specifies that we want to retrieve values from the "Booster_Version" column.
- FROM SPACEXTBL: It specifies the table name "SPACEXTBL" from which the data is being retrieved.
- WHERE Landing_Outcome = 'Success (drone ship)': This is the first condition to filter the rows. It selects rows where the "Landing_Outcome" column is equal to 'Success (drone ship)'.
- AND Payload_Mass__kg_ > 4000 AND Payload_Mass__kg_ < 6000: These are additional conditions using the logical AND operator. They further filter the rows to only include rows where the "Payload_Mass__kg_" is greater than 4000 and less than 6000.

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

Total Number of Successful and Failure Mission Outcomes

- %sql SELECT Mission_Outcome, COUNT(*) FROM SPACEXTBL GROUP BY Mission_Outcome;
- SELECT Mission_Outcome, COUNT(*): This part of the query selects the "Mission_Outcome" column and counts the occurrences of each unique value using COUNT(*).
- FROM SPACEXTBL: It specifies the table name "SPACEXTBL" from which the data is being retrieved.
- GROUP BY Mission_Outcome: This clause groups the result set based on the unique values in the "Mission_Outcome" column.

Mission_Outcome	COUNT(*)
None	898
Failure (in flight)	1
Success	98
Success	1
Success (payload status unclear)	1

Boosters Carried Maximum Payload

- %sql SELECT Booster_Version FROM SPACEXTBL WHERE Payload_Mass__kg_ = (SELECT MAX(Payload_Mass__kg_) FROM SPACEXTBL);
- SELECT Booster_Version: This part of the query specifies that we want to retrieve values from the "Booster_Version" column.
- FROM SPACEXTBL: It specifies the table name "SPACEXTBL" from which the data is being retrieved.
- WHERE Payload_Mass__kg_ = (SELECT MAX(Payload_Mass__kg_) FROM SPACEXTBL): This is the condition to filter the rows. It selects rows where the "Payload_Mass__kg_" is equal to the maximum payload mass value obtained from the subquery (SELECT MAX(Payload_Mass__kg_) FROM SPACEXTBL).

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

```
df['Date_Time_UTC'] = df['Date'] + ' ' + df['Time (UTC)']

df['Date_Time_UTC'] = pd.to_datetime(df['Date_Time_UTC'], format='%d/%m/%Y %H:%M:%S')

df['Month'] = df['Date_Time_UTC'].dt.month

df['Year'] = df['Date_Time_UTC'].dt.year

filtered_df = df[

    (df['Year'] == 2015) &

    (df['Landing_Outcome'] == 'Failure (drone ship)')

]

result_df = filtered_df[['Month', 'Booster_Version', 'Launch_Site', 'Payload', 'Landing_Outcome']]

print(result_df)
```

- **Combining 'Date' and 'Time (UTC)' columns:** the code combines the 'Date' and 'Time (UTC)' columns to create a new column that represents both the date and time in a single string format.
- **Converting 'Date_Time_UTC' to datetime format:** After combining the 'Date' and 'Time (UTC)' columns, the code converts the new combined column into the pandas datetime format. The datetime format allows for easier manipulation and extraction of date and time components.
- **Extracting month and year:** Once the 'Date_Time_UTC' column is converted to the datetime format, the code extracts the month and year from it and creates two new columns to store this information.
- **Filtering the DataFrame:** The code filters the DataFrame to include only those records where the year is 2015 and the landing outcome is specified as "Failure (drone ship)". This filtering operation allows us to focus only on the relevant data.
- **Selecting desired columns for the result:** After filtering the DataFrame, the code selects only specific columns (such as 'Month', 'Booster_Version', 'Launch_Site', 'Payload', and 'Landing_Outcome') from the filtered DataFrame. This step ensures that the final result contains only the relevant information.

	Month	Booster_Version	Launch_Site	Payload	Landing_Outcome
13	10.0	F9 v1.1 B1012	CCAFS LC-40	SpaceX CRS-5	Failure (drone ship)
16	4.0	F9 v1.1 B1015	CCAFS LC-40	SpaceX CRS-6	Failure (drone ship)

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

```
df['Date'] = pd.to_datetime(df['Date'])

filtered_df = df[(df['Date'] >= '2010-06-04') & (df['Date'] <= '2017-03-20')]

landing_outcomes_count = filtered_df['Landing_Outcome'].value_counts().reset_index()

landing_outcomes_count.columns = ['Landing_Outcome', 'Count']

landing_outcomes_count = landing_outcomes_count.sort_values(by='Count',
ascending=False)

print(landing_outcomes_count)
```

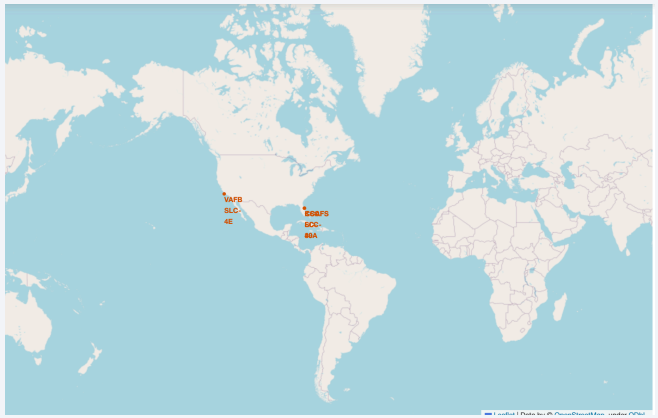
	Landing_Outcome	Count
0	No attempt	10
1	Failure (drone ship)	5
2	Success (drone ship)	5
3	Controlled (ocean)	3
4	Success (ground pad)	3
5	Failure (parachute)	2
6	Uncontrolled (ocean)	2
7	Precluded (drone ship)	1

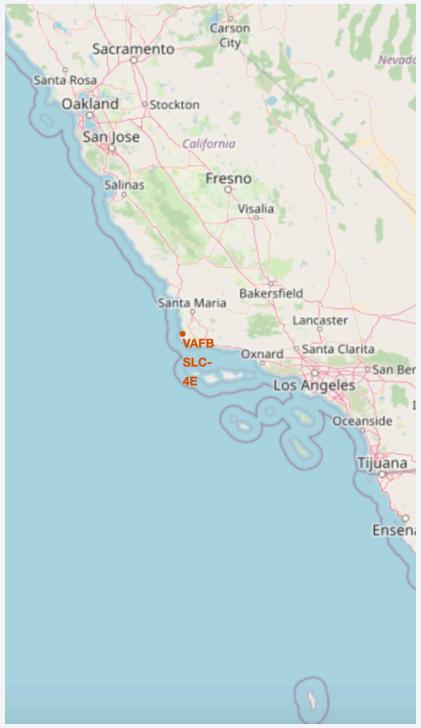
- Convert the 'Date' column of the DataFrame df to datetime format using pd.to_datetime.
- Filter the DataFrame df to keep only the rows where the date falls between '2010-06-04' and '2017-03-20', inclusive.
- Use value_counts to count the occurrences of each unique value in the 'Landing_Outcome' column of the filtered DataFrame. The reset_index function is used to convert the resulting Series into a DataFrame and reset the index, and then column names are assigned for clarity.
- Sort the landing_outcomes_count DataFrame in descending order based on the 'Count' column, so that the most frequent landing outcomes appear first.

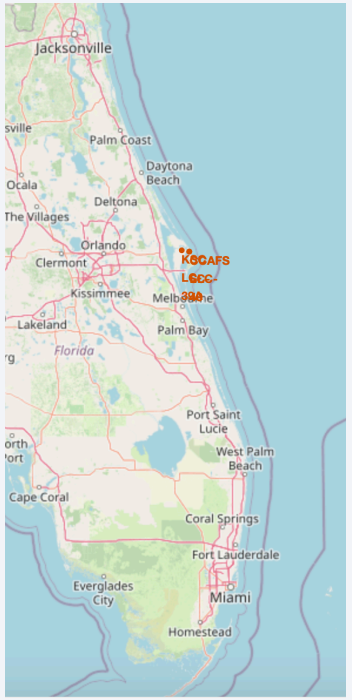
Section 3

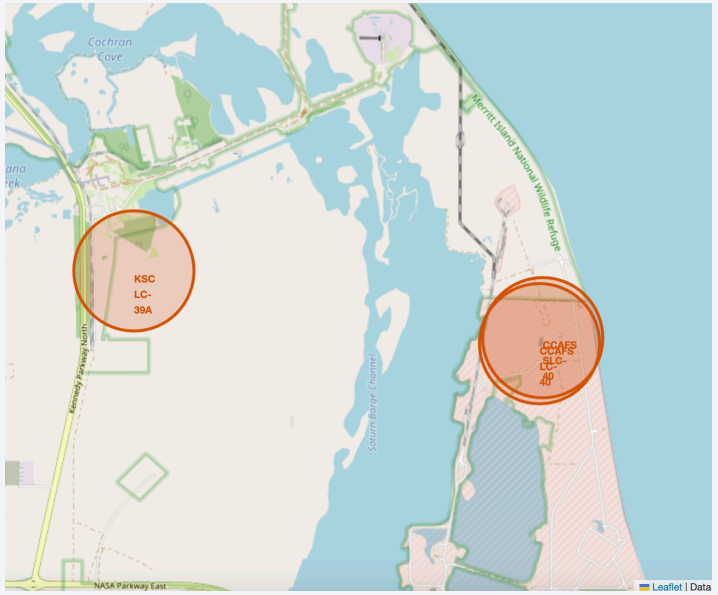
Launch Sites Proximities Analysis

SpaceX Launch sites

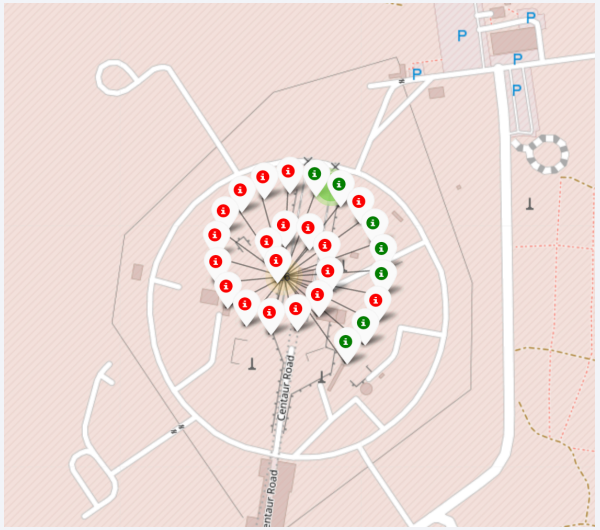


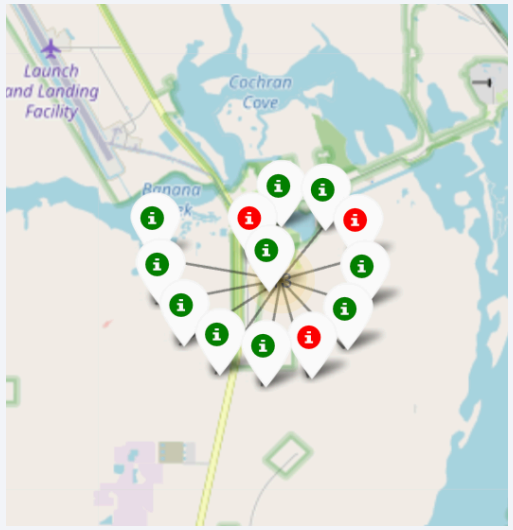


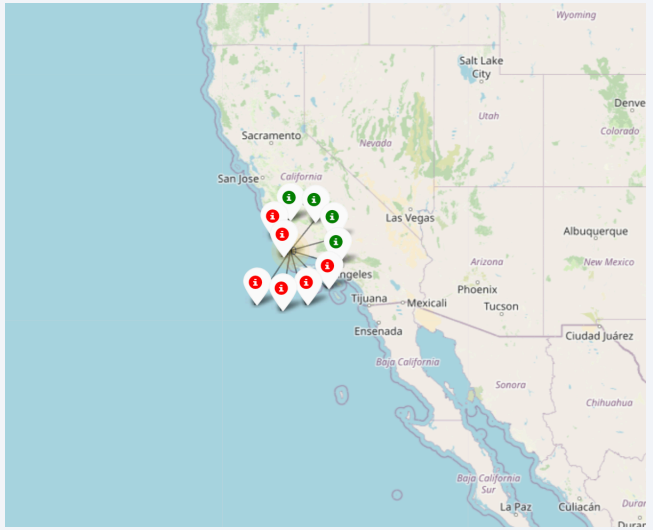


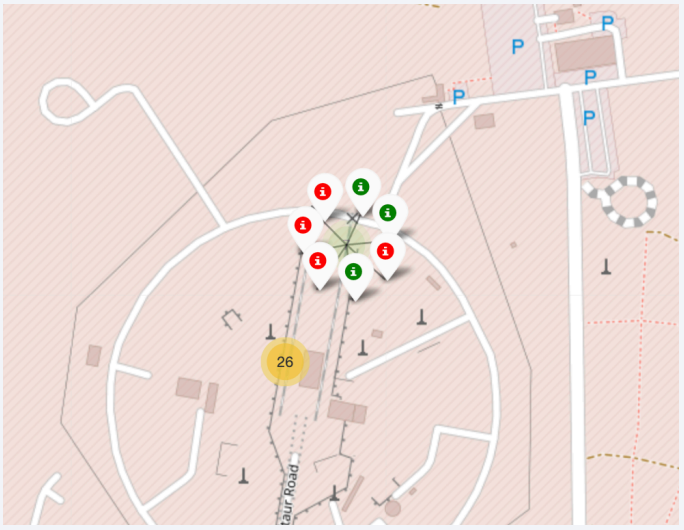


Success/failed launches for each site

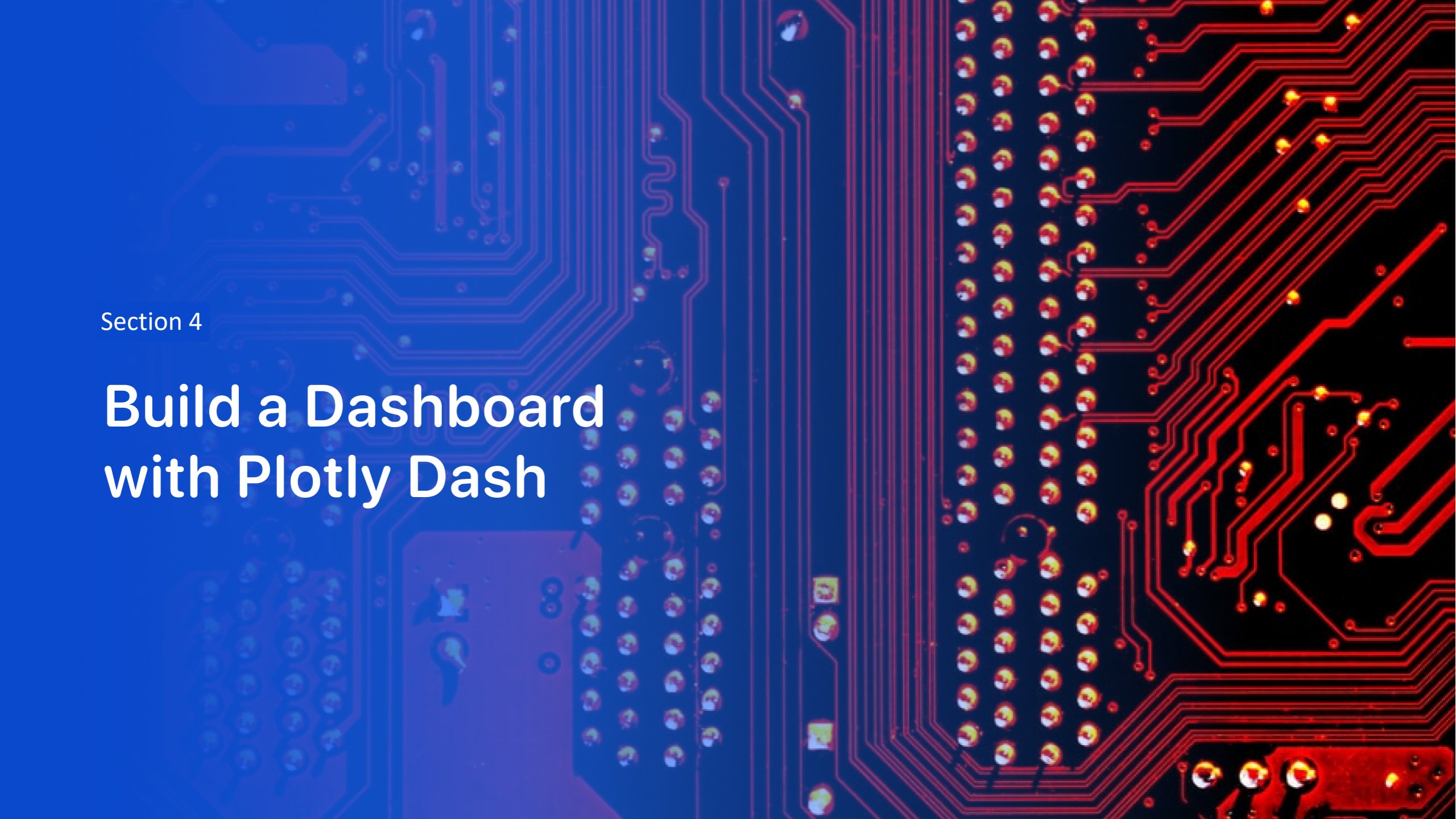






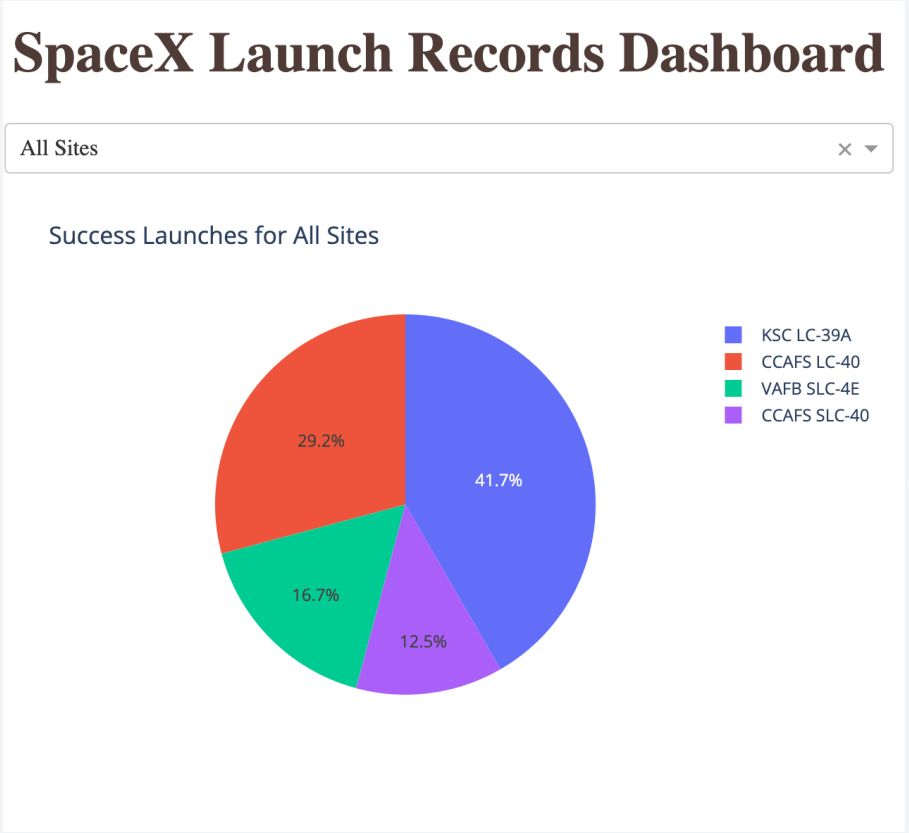


Launch Site	Failed	Successful
CCAFS LC-40	19	7
CCAFS SLC-40	4	3
KSC LC-39A	3	10
VAFB SLC-4E	6	4


Section 4
Build a Dashboard with Plotly Dash

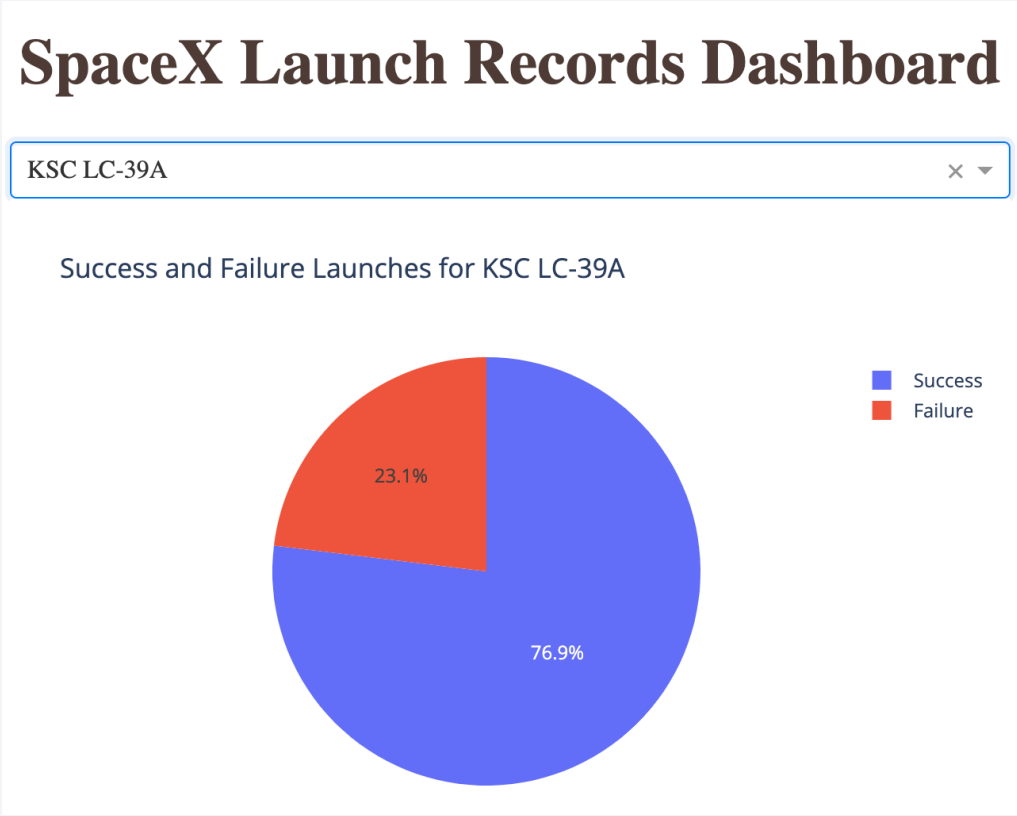
Success Launches for all sites

- The most successful launch site is KSC LC-39A, with 41.7% success rate
- The least successfull is CCAFS SLC-40 with 12.5% sccess rate



KSC LC-39A

- The launches at KSC LC-39A were 76.9% successful and 23.1% unsuccesful.



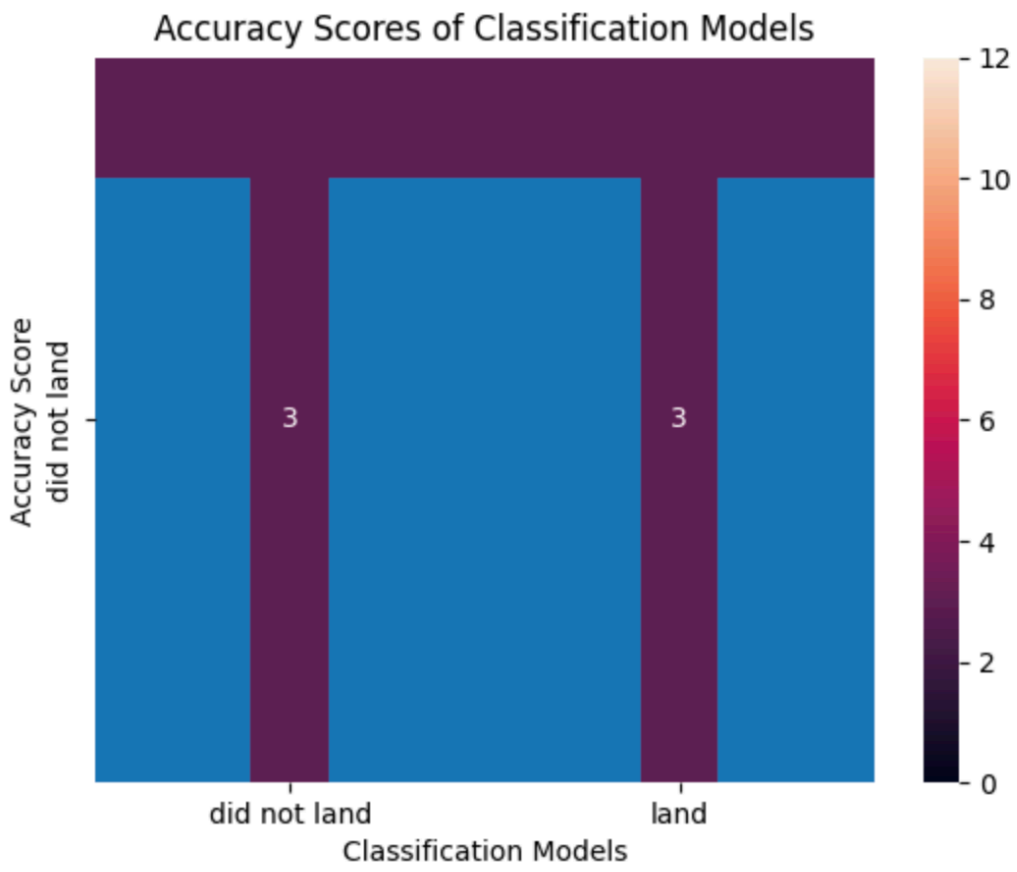
Payload and Launch Success

- The launches at KSC LC-39A were 76.9% successful and 23.1% unsuccesful.

Section 5

Predictive Analysis (Classification)

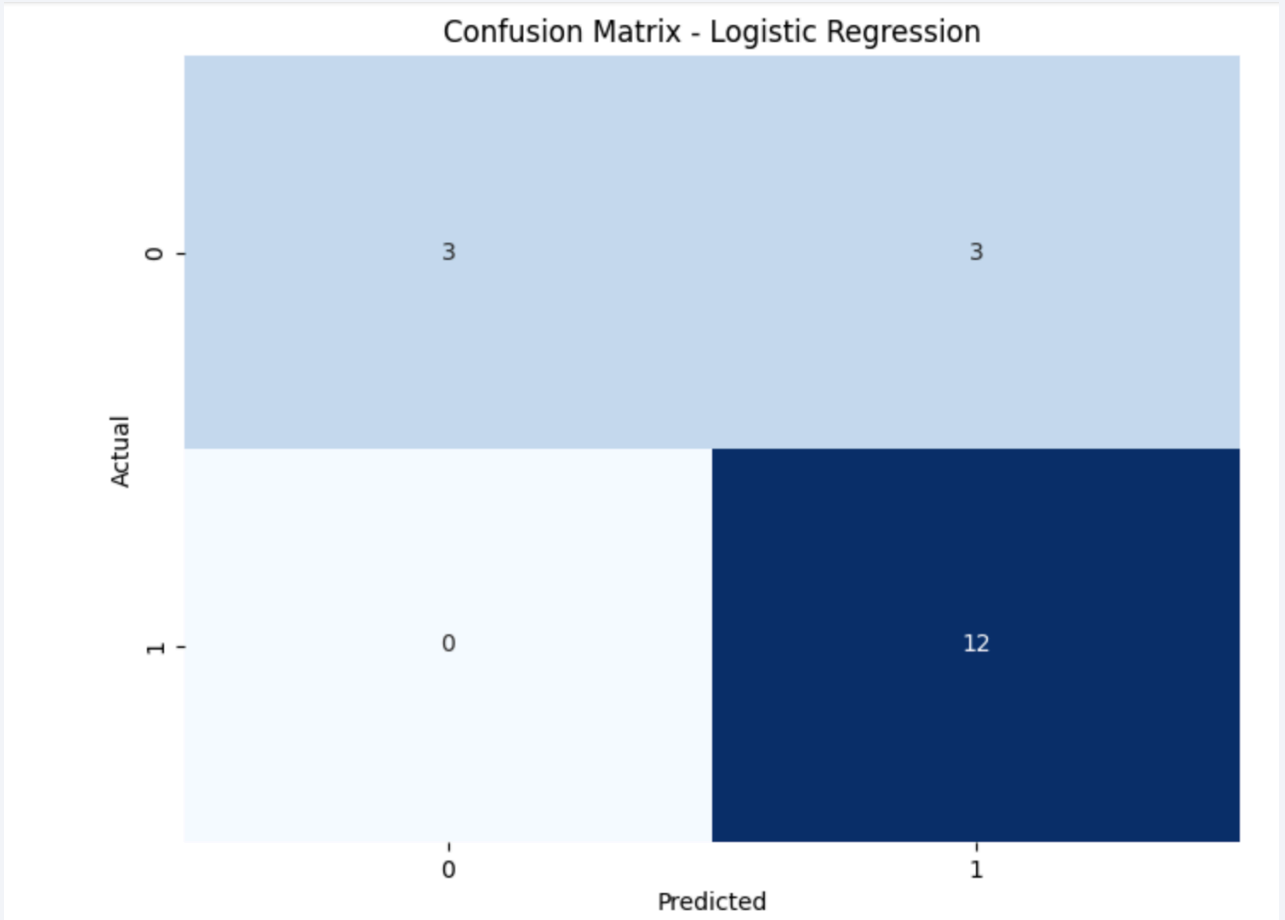
Classification Accuracy




```
[43]: # Calculate accuracy scores
logreg_score = logreg_cv.score(X_test, Y_test)
svm_score = svm_cv.score(X_test, Y_test)
tree_score = tree_cv.score(X_test, Y_test)
knn_score = knn_cv.score(X_test, Y_test)

# Create a dictionary to store the accuracy scores
accuracy_scores = {'Logistic Regression': logreg_score,
                   'Support Vector Machine': svm_score,
                   'Decision Tree': tree_score,
                   'K-Nearest Neighbors': knn_score}

# Find the model with the highest accuracy
best_model = max(accuracy_scores, key=accuracy_scores.get)
best_accuracy = accuracy_scores[best_model]

# Print the best model and its accuracy
print("Best Model:", best_model)
print("Accuracy:", best_accuracy)
```

```
Best Model: Logistic Regression
Accuracy: 0.8333333333333334
```

Confusion Matrix




```
Best Performing Model: Logistic Regression
Confusion Matrix:
[[ 3  3]
 [ 0 12]]
```

Thank you!