IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Hoang Phong
22/12/2024

Outline

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

Executive Summary

- Summary of methodologies: Data was collected from the public SpaceX API and the SpaceX Wikipedia page, with a new column, ‘class,’ created to classify successful landings. The data was explored using SQL, visualizations, Folium maps, and dashboards. Relevant columns were selected as features, and categorical variables were converted to binary using one-hot encoding. The data was standardized, and GridSearchCV was used to identify the optimal parameters for machine learning models. The accuracy scores of all models were visualized.
- Summary of all results: Four machine learning models were developed: Logistic Regression, Support Vector Machine, Decision Tree Classifier, and K-Nearest Neighbors. Each model achieved a similar accuracy of approximately 83.33%. However, all models tended to overpredict successful landings, indicating the need for more data to improve model selection and accuracy.

Introduction

- Project background and context: The commercial space age is here, companies are making space travel affordable for everyone. Perhaps the most successful is SpaceX. SpaceX advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upwards of 165 million dollars each, much of the savings is because SpaceX can reuse the first stage. Therefore, if we can determine if the first stage will land, we can determine the cost of a launch.
- Problems you want to find answers: Instead of using rocket science to determine if the first stage will land successfully, we will train a machine learning model and use public information to predict if SpaceX will reuse the first stage.

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - Combined data from SpaceX public API and SpaceX Wikipedia page
- Perform data wrangling
  - Classifying true landings as successful and unsuccessful otherwise
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - Tuned models using GridSearchCV

Data Collection

- Data collection process involved a combination of API requests from Space X public API and web scraping data from a table in Space X's Wikipedia entry.

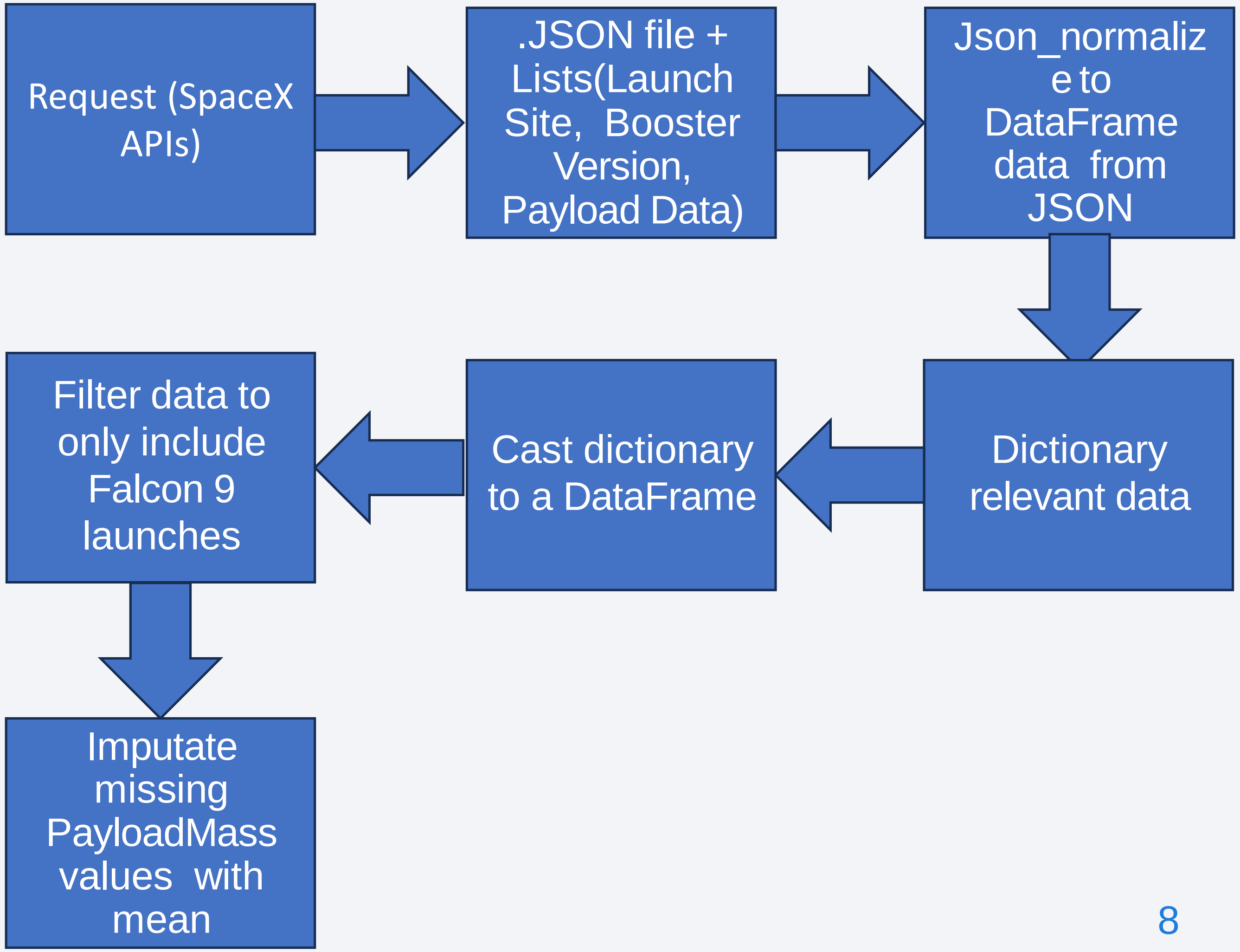
- Data columns from SpaceX API: FlightNumber, Date, BoosterVersion, PayloadMass, Orbit, LaunchSite, Outcome, Flights, GridFins, Reused, Legs, LandingPad, Block, ReusedCount, Serial, Longitude, Latitude.

- Data columns from Wikipedia: Flight No., Launch site, Payload, PayloadMass, Orbit, Customer, Launch outcome, Version Booster, Booster landing, Date, Time

Data Collection – SpaceX API

- Github URL:

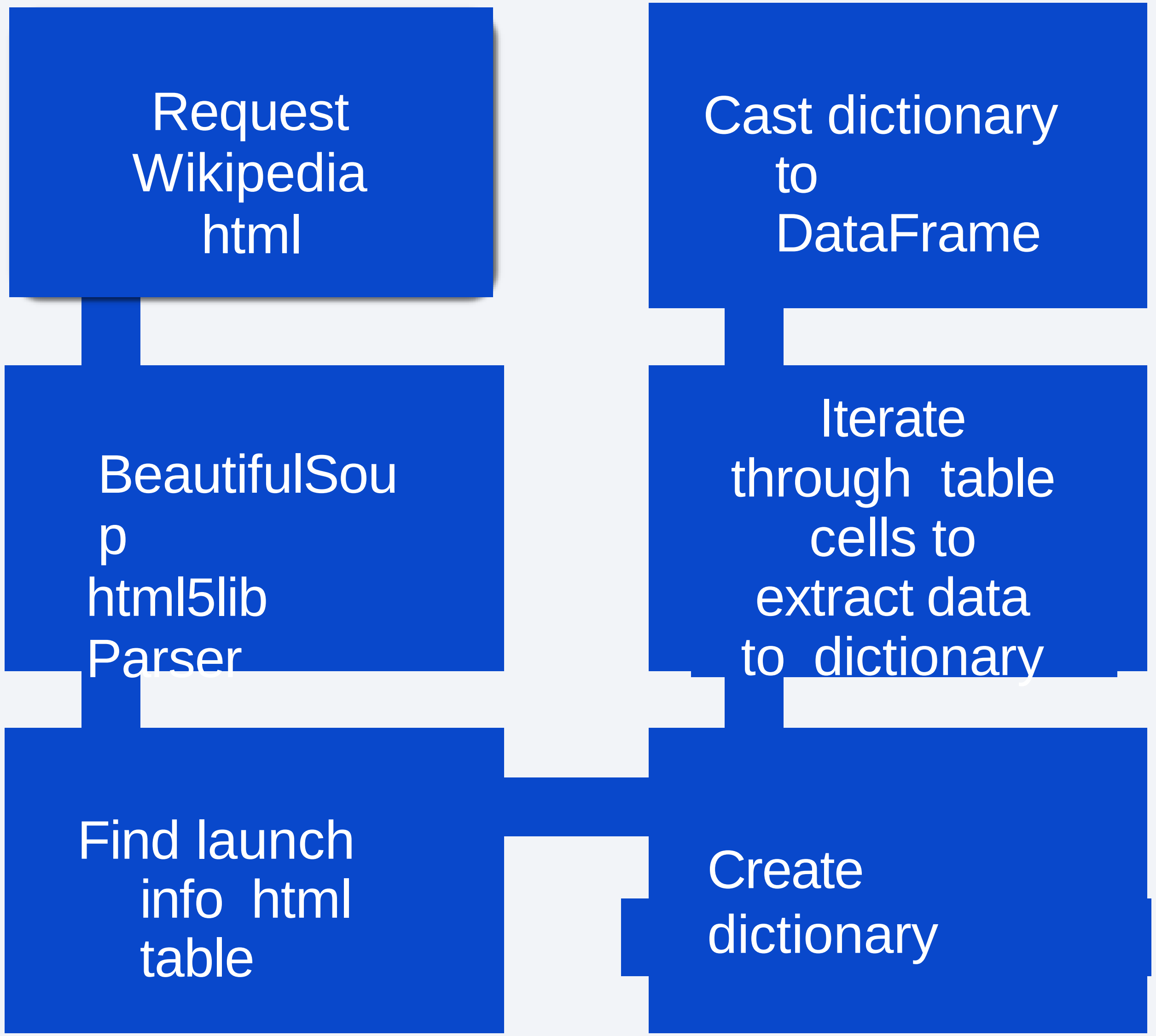
https://github.com/hoangphong111213/ibm-applied-ds-capstone/blob/main/jupyter-labs-spacex-data-collection-api.ipynb



Data Collection - Scraping

- Github URL:

https://github.com/hoangphong111213/ibm-applied-ds-capstone/blob/main/jupyter-labs-webscraping.ipynb



Data Wrangling

- Create a new column called class to indicate landing outcomes, where successful landings are labeled as 1 and failures as 0. The Outcome column consists of two parts: Mission Outcome and Landing Location. For the new column class:
- Assign a value of 1 if Mission Outcome is True (e.g., True ASDS, True RTLS, or True Ocean).
- Assign a value of 0 for all other cases (e.g., None None, False ASDS, None ASDS, False Ocean, or False RTLS).
- Github URL: https://github.com/hoangphong111213/ibm-applied-ds-capstone/blob/main/labs-jupyter-spacex-Data%20wrangling.ipynb

EDA with Data Visualization

- Exploratory Data Analysis (EDA) was carried out on the variables: Flight Number, Payload Mass, Launch Site, Orbit, Class, and Year.
- Plots Generated: Flight Number vs. Payload Mass, Flight Number vs. Launch Site, Payload Mass vs. Launch Site, Orbit vs. Success Rate, Flight Number vs. Orbit, Payload vs Orbit, and Success Yearly Trend
- Scatter plots, line charts, and bar graphs were utilized to examine relationships among these variables. The goal was to determine whether meaningful patterns or associations exist that could contribute to the effectiveness of the machine learning model.
- Github URL: https://github.com/hoangphong111213/ibm-applied-ds-capstone/blob/main/edadataviz.ipynb

EDA with SQL

- The dataset was loaded into the IBM DB2 Database and analyzed using SQL integrated with Python.
- **Queries Performed:** Extracted **launch site names,** Retrieved data on **mission outcomes,** Analyzed **payload sizes** associated with different customers, Investigated **booster versions** and their respective **landing outcomes**.
- These queries were aimed at gaining deeper insights into the dataset to better understand the key attributes and relationships for further analysis.
- Github URL: https://github.com/hoangphong111213/ibm-applied-ds-capstone/blob/main/jupyter-labs-eda-sql-coursera_sqllite.ipynb

Build an Interactive Map with Folium

- Folium maps were used to plot the **Launch Sites**, **successful and unsuccessful landings**, and proximity to key landmarks, including **Railways**, **Highways**, **Coasts**, and **Cities**.
- **Purpose:**
  1. To analyze the strategic placement of launch sites based on their proximity to essential infrastructure and geographical features.
  2. To visualize the distribution and outcomes of landings relative to their locations.
- This mapping helps understand the rationale behind launch site locations and provides insights into landing success rates in relation to these key factors.
- GitHub URL: https://github.com/hoangphong111213/ibm-applied-ds-capstone/blob/main/lab_jupyter_launch_site_location.ipynb

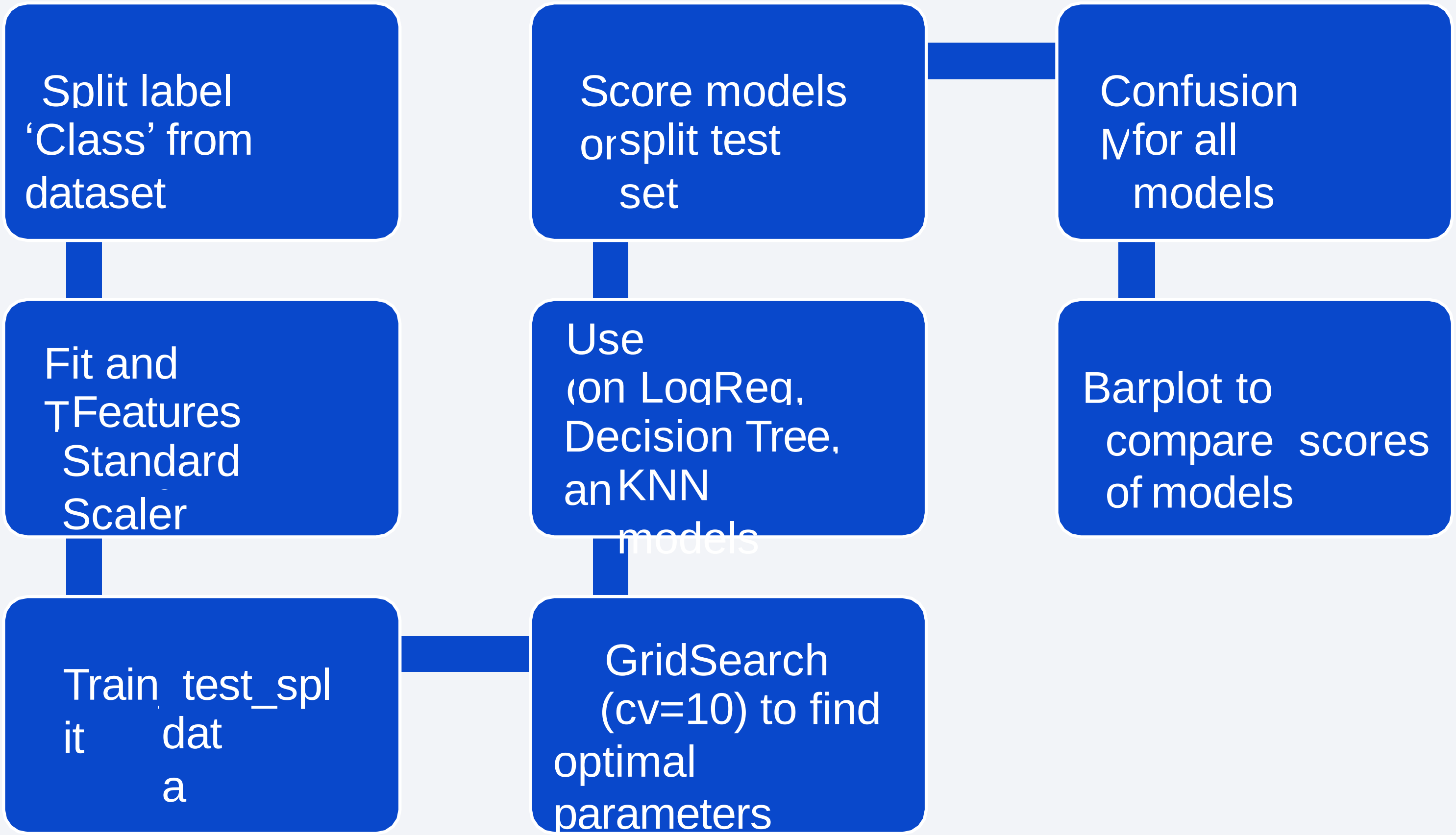
Build a Dashboard with Plotly Dash

- The dashboard features a **pie chart** and a **scatter plot** for interactive data visualization:
  - **Pie Chart:**
    - Displays the **distribution of successful landings** across all launch sites.
    - Can be toggled to show **success rates for individual launch sites**.
    - Helps visualize the performance of launch sites in terms of landing success.
  - **Scatter Plot:**
    - Accepts two inputs:
      - Selection of **all sites** or a specific **individual site**.
      - **Payload mass** adjustable via a slider (range: 0–10,000 kg).
- Highlights variations in landing success based on **launch site**, **payload mass**, and **booster version category**.
- GitHub URL: https://github.com/hoangphong111213/ibm-applied-ds-capstone/blob/main/spacex_dash_app.py

Predictive Analysis (Classification)

- Github URL:

https://github.com/hoangphong111213/ibm-applied-ds-capstone/blob/main/SpaceX_Machine%20Learning%20Prediction_Part_5.ipynb



Results

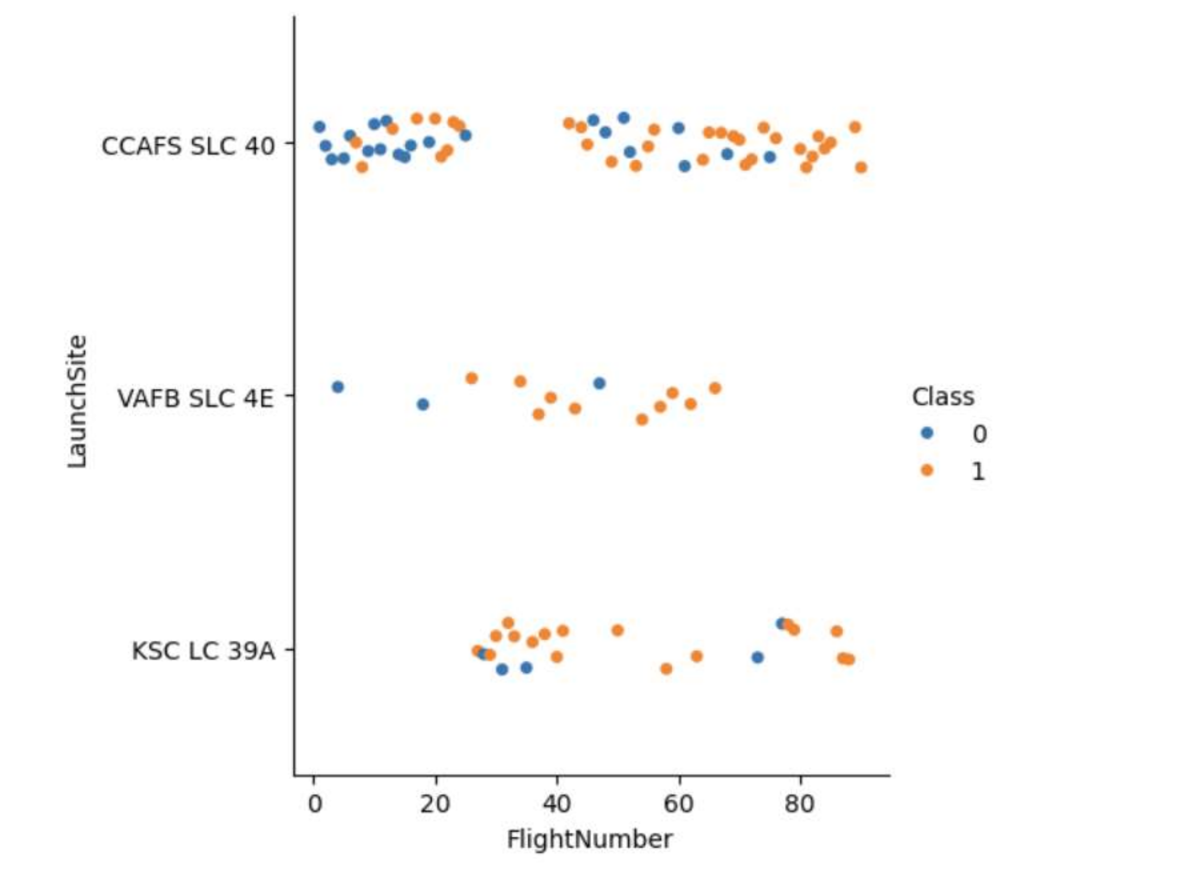
- Key findings from visual and SQL-based data exploration, including insights into launch site performance, payload trends, and landing outcomes.
- Screenshots of interactive visualizations from the dashboard, such as Folium maps, pie charts, and scatter plots, demonstrating key insights.
- Model performance summary, achieving approximately **83% accuracy**, indicating the effectiveness of predictions based on the analyzed variables.

Section 2

Insights drawn from EDA

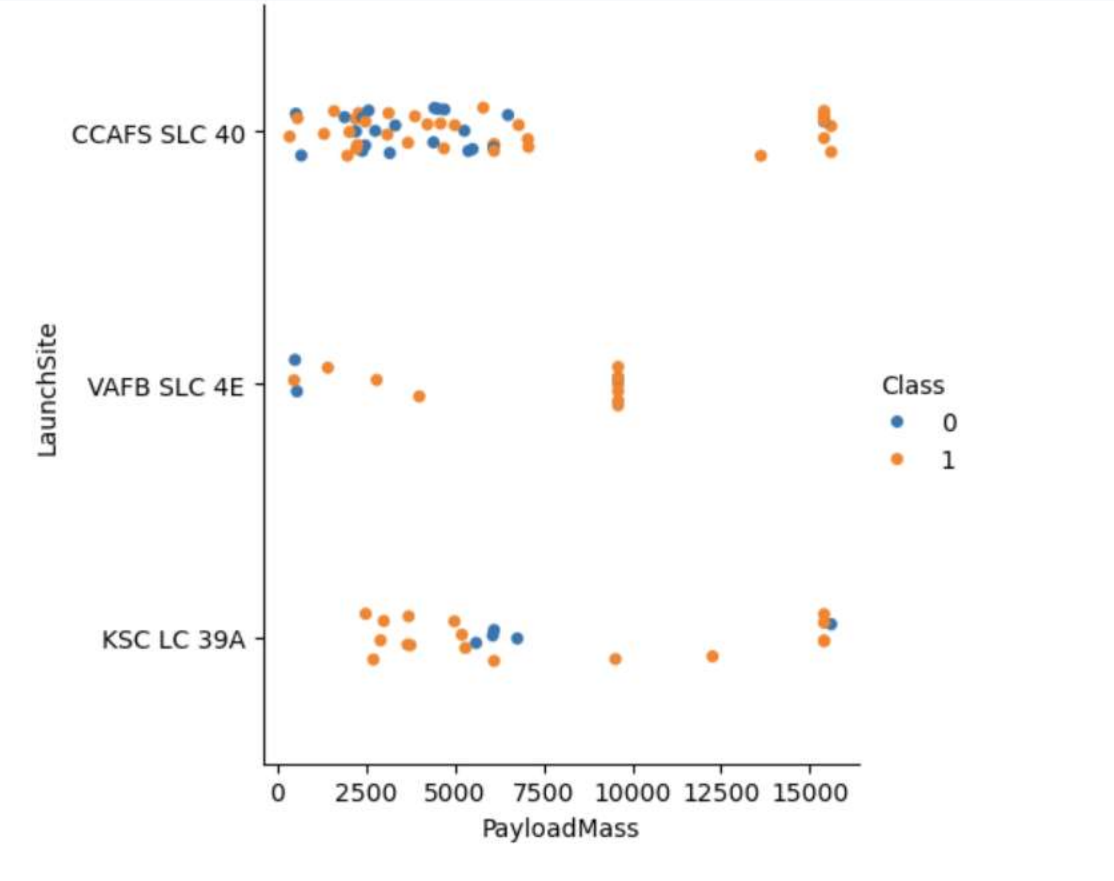
Flight Number vs. Launch Site

- The larger the flight numbers, the greater the success.
- CCAFS SLC 40 shows the least pattern.



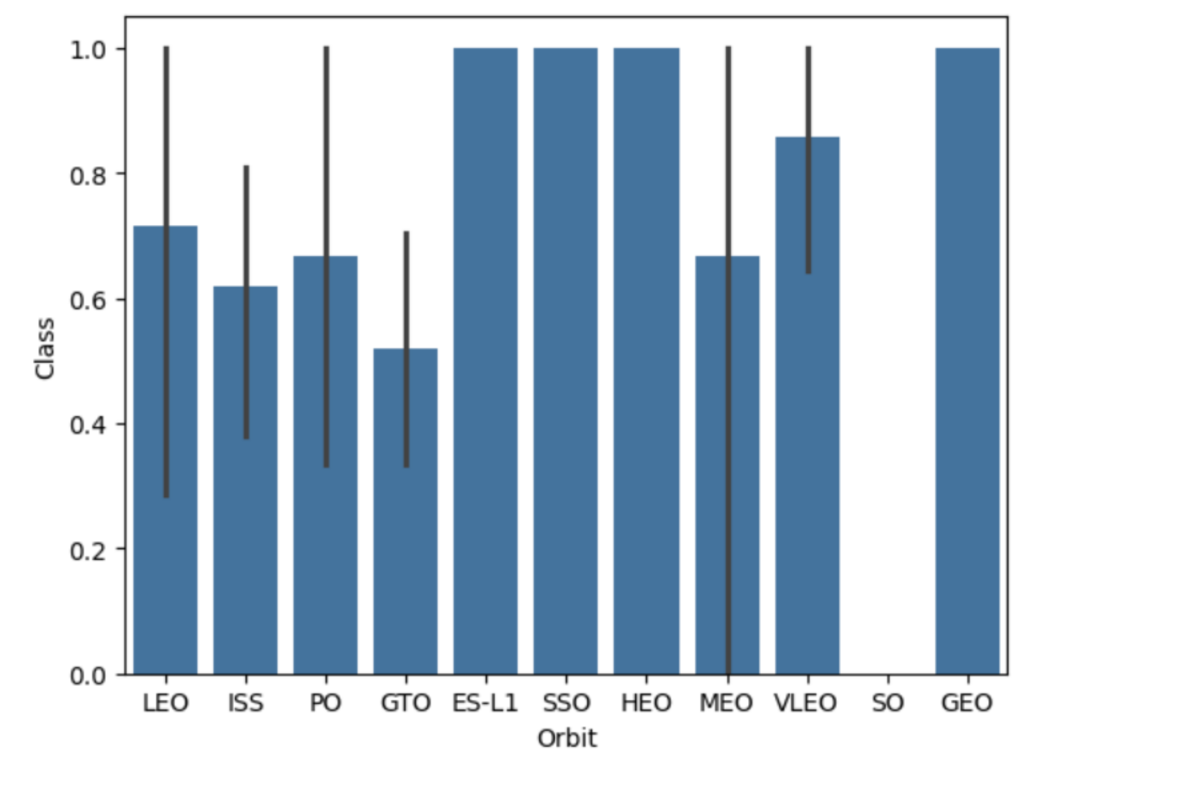
Payload vs. Launch Site

- For payload mass greater than 7500kg, the success rate increases significantly, but no obvious pattern



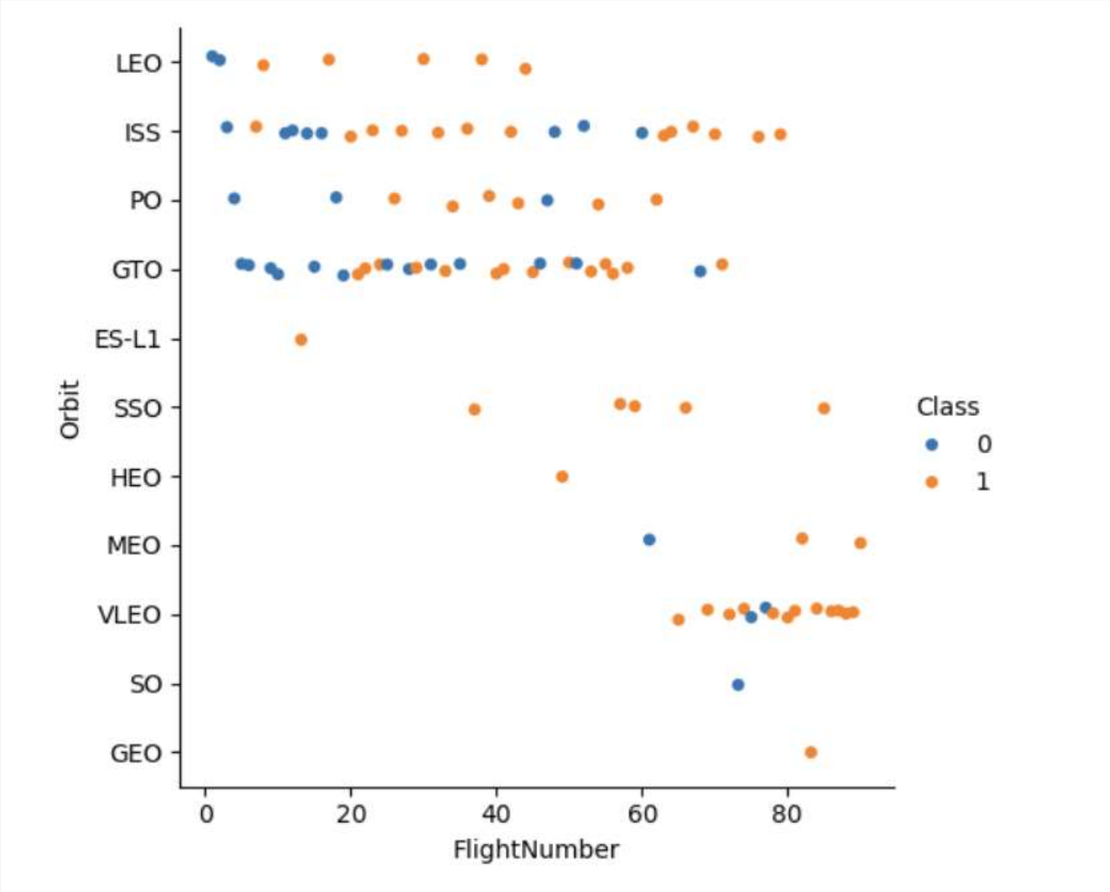
Success Rate vs. Orbit Type

- SO orbit shows 0% rate of success
- ES-L1, SSO, HEO, GEO orbits show only 1 occurrence for each, and they all have 100% success rate



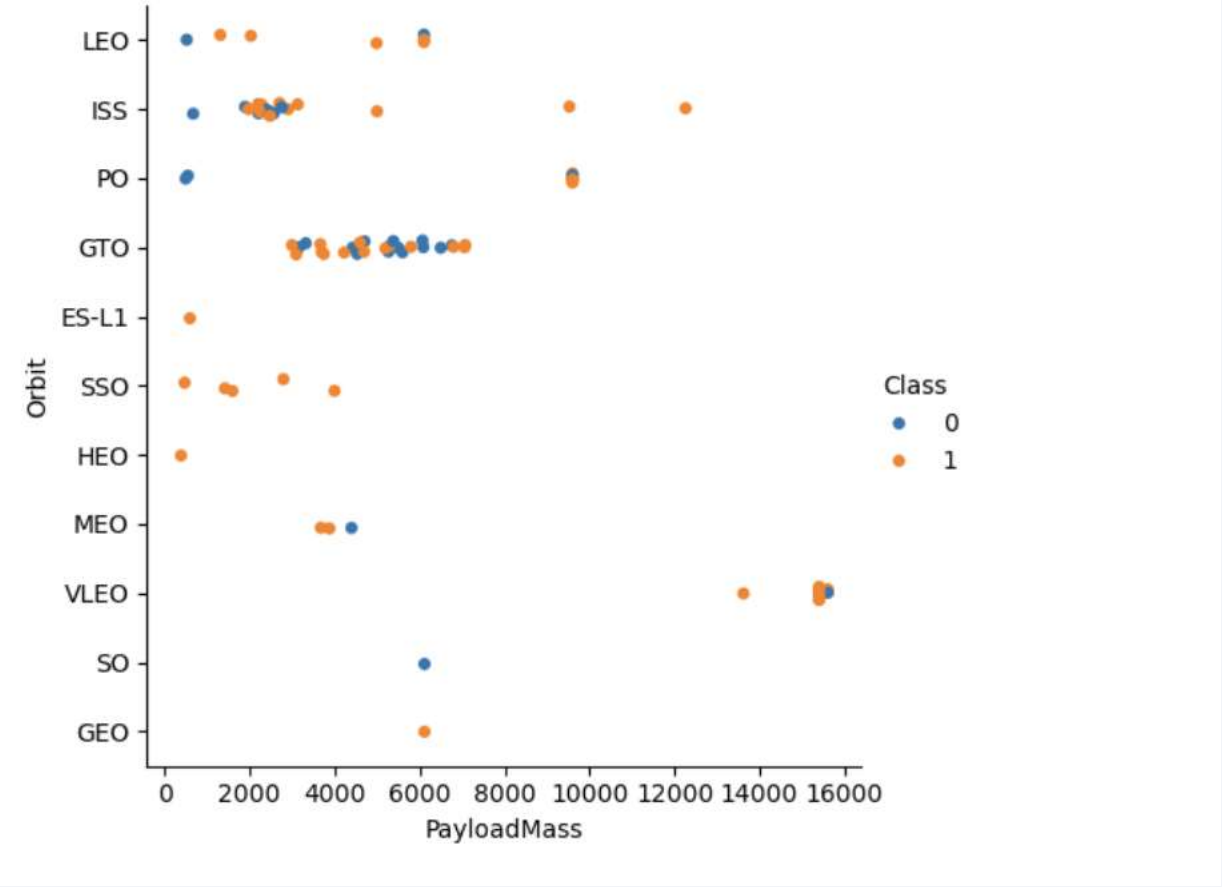
Flight Number vs. Orbit Type

- The larger the flight number, the greater the success rate, except for GTO orbit, which shows less pattern.



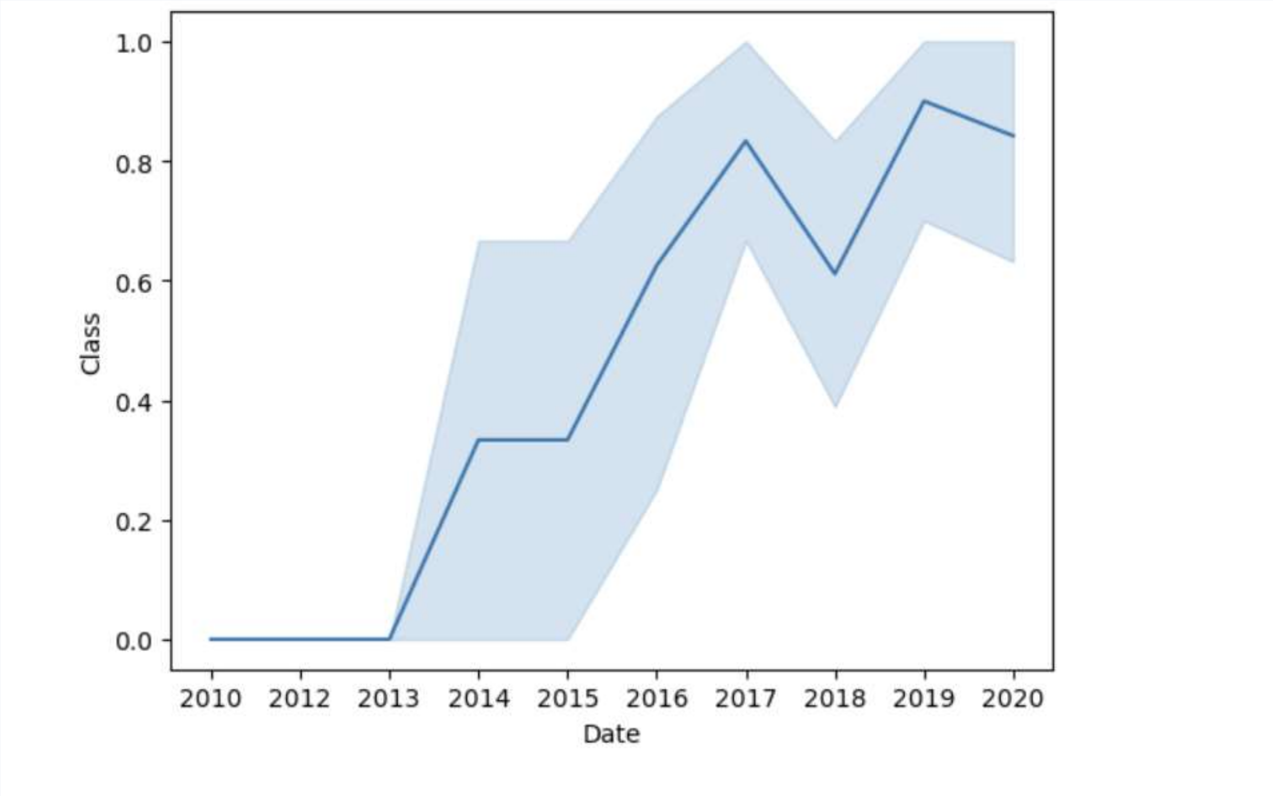
Payload vs. Orbit Type

- Heavier payload has more success rate
- GTO orbit shows the least pattern



Launch Success Yearly Trend

- Success rate increases throughout the years
- Max success rate = 90%



All Launch Site Names

- DISTINCT: select unique names only, avoid duplicates

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

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

[10]:

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

Launch Site Names Begin with 'CCA'

- Display 5 records where launch site names begin with CCA

Task 2

Display 5 records where launch sites begin with the string '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
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Total Payload Mass

- Total payload carried by boosters from NASA

```
[13]: %sql select sum(PAYLOAD_MASS__KG_) from SPACEXTABLE where customer = 'NASA (CRS)'

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

[13]:

sum(PAYLOAD_MASS__KG_)
45596

Average Payload Mass by F9 v1.1

- Average payload mass carried by booster version F9 v1.1

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

```
[14]: %sql select avg(PAYLOAD_MASS__KG_) from SPACEXTABLE where Booster_Version = 'F9 v1.1'

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

[14]:

avg(PAYLOAD_MASS__KG_)
2928.4

First Successful Ground Landing Date

- Use min() function to find the first successful ground landing date

```
[15]: %sql select min(Date) from SPACEXTABLE where Landing_Outcome = 'Success (ground pad)'

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

[15]:

min(Date)
2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

- Use Where ... And … clause to filter boosters that meet the requirement

```
[18]: %sql select Booster_Version from SPACEXTABLE where Landing_Outcome = 'Success (drone ship)' and PAYLOAD_MASS__KG_ > 4000 and PAYLOAD_MASS__KG_ < 6000
 * sqlite:///my_data1.db
Done.
```

[18]:

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

Total Number of Successful and Failure Mission Outcomes

- Use WHERE … LIKE with wildcard to filter Mission Outcomes that are successful or failure

List the total number of successful and failure mission outcomes

[19]:
```
%sql SELECT s.success, f.failure FROM (SELECT COUNT(*) AS success FROM SPACEXTABLE WHERE Mission_Outcome LIKE 'Success%') AS s, (SELECT COUNT(*) AS failure FROM SPACEXTABLE WHERE Mission_Outcome LIKE 'Failure%') AS f;
```

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

[19]:

success	failure
100	1

Boosters Carried Maximum Payload

- Use WHERE clause with max() function to find

```
[21]: %sql select Booster_Version from SPACEXTABLE where PAYLOAD_MASS__KG_ = (select max(PAYLOAD_MASS__KG_) from SPACEXTABLE)

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

[21]:

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

- Use CASE clause to retrieve month names
- Use WHERE … AND to filter all required records

[23]:

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

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

[23]:

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

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

- Use WHERE … BETWEEN ... AND to filter all required records
- Use GROUP BY to group the landing outcomes
- Use ORDER BY … DESC to sort the result by Outcome_Count in descending order

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.

[24]:
```
%sql \
SELECT \
    Landing_Outcome, \
    COUNT(*) AS Outcome_Count \
FROM SPACEXTABLE \
WHERE Date BETWEEN '2010-06-04' AND '2017-03-20' \
GROUP BY Landing_Outcome \
ORDER BY Outcome_Count DESC;
```

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

[24]:

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

Section 3

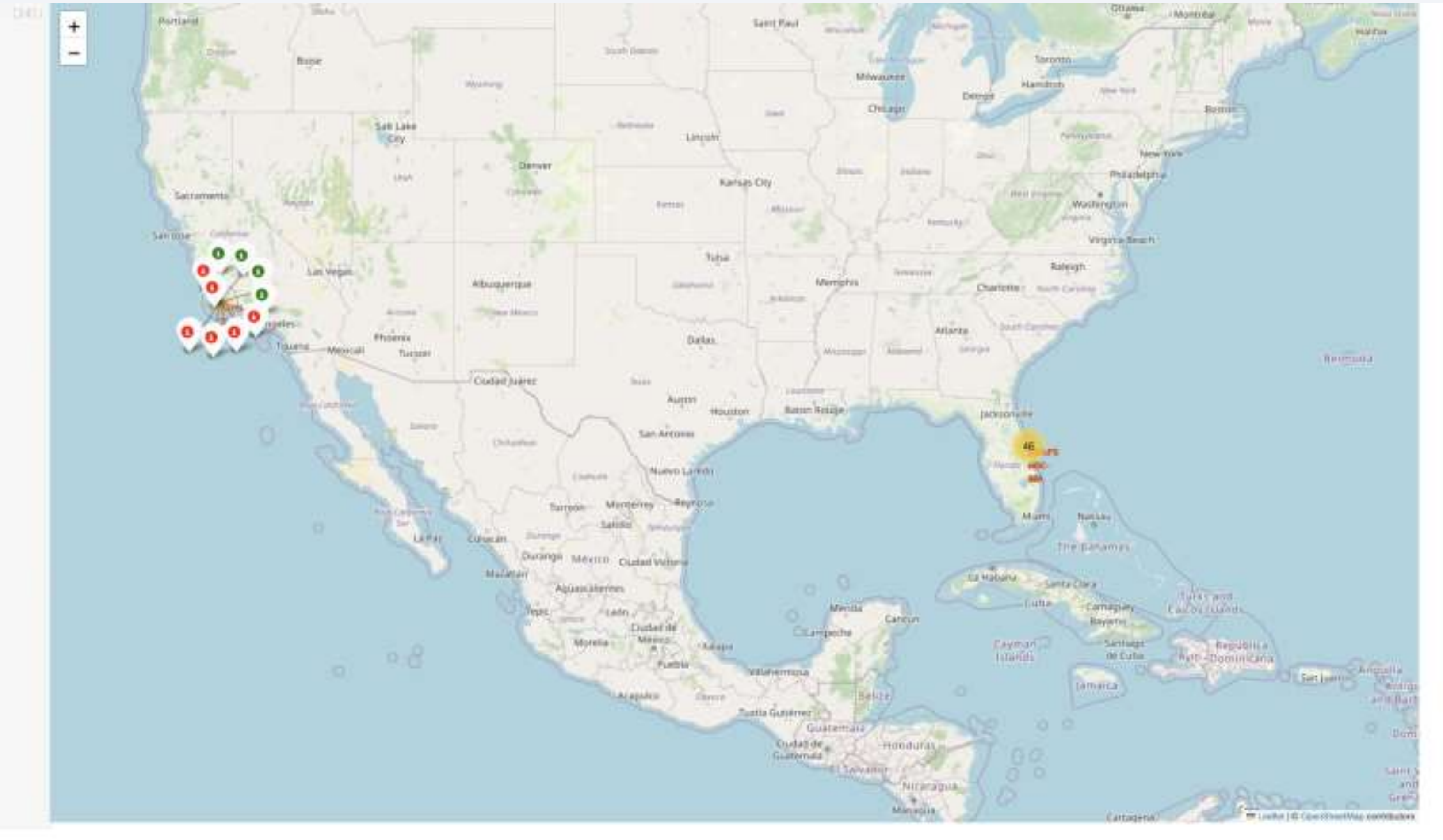
Launch Sites Proximities Analysis

Location of All Launch Sites

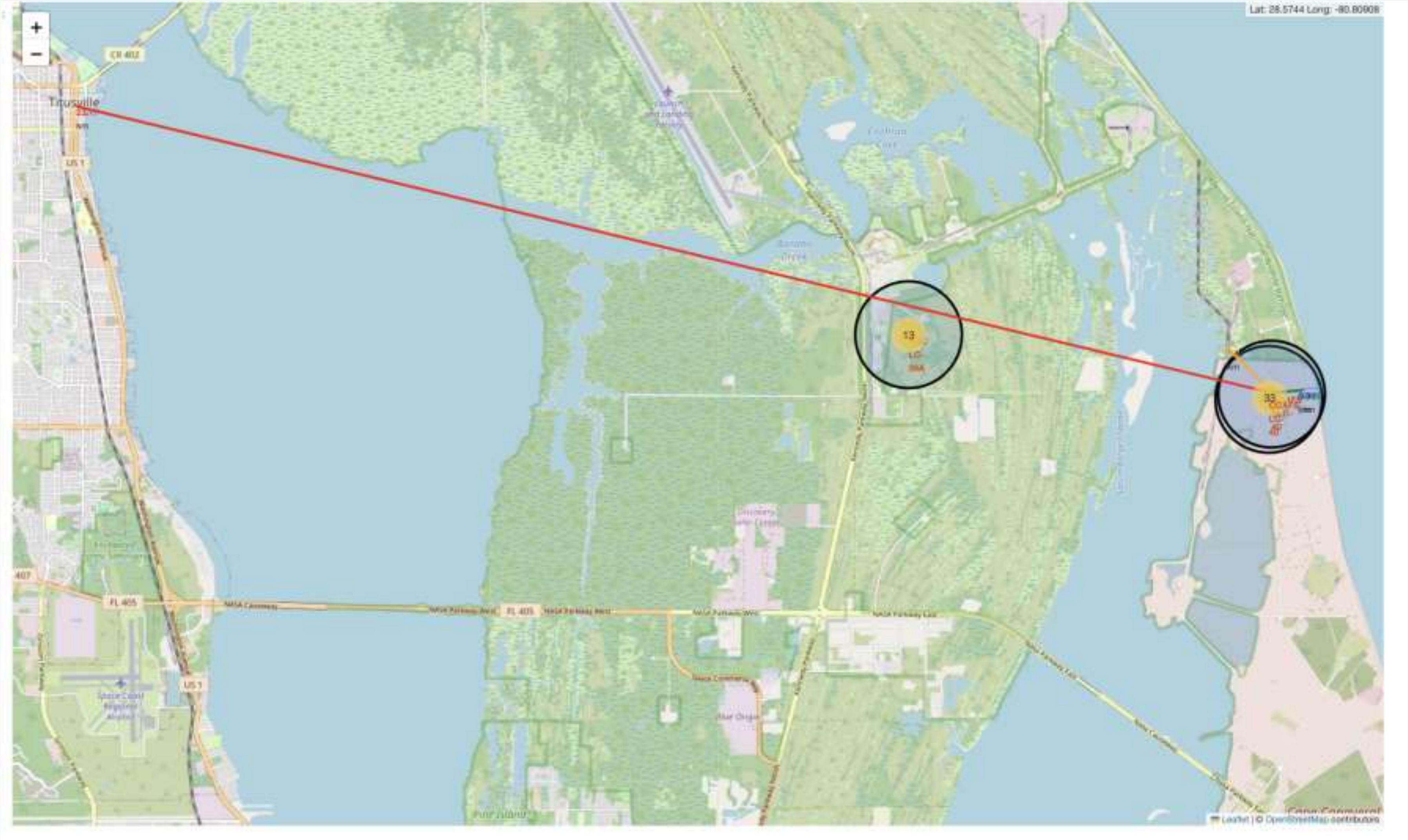
- Mark all launch sites on the map

Marker showing launch sites with color labels

- Mark the success/failed launch for each site



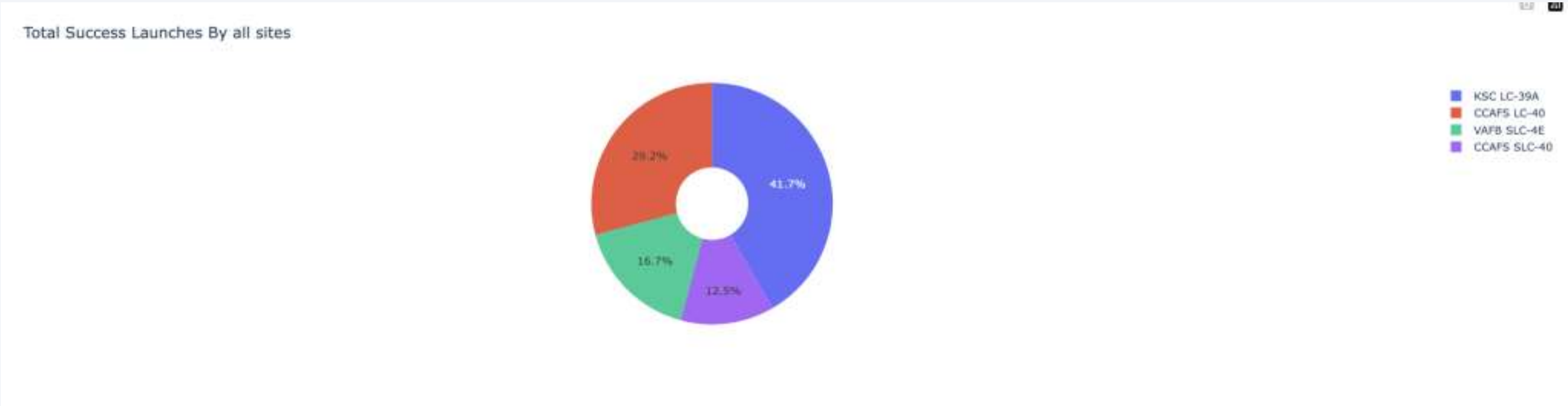
Launch Site distance to some locations



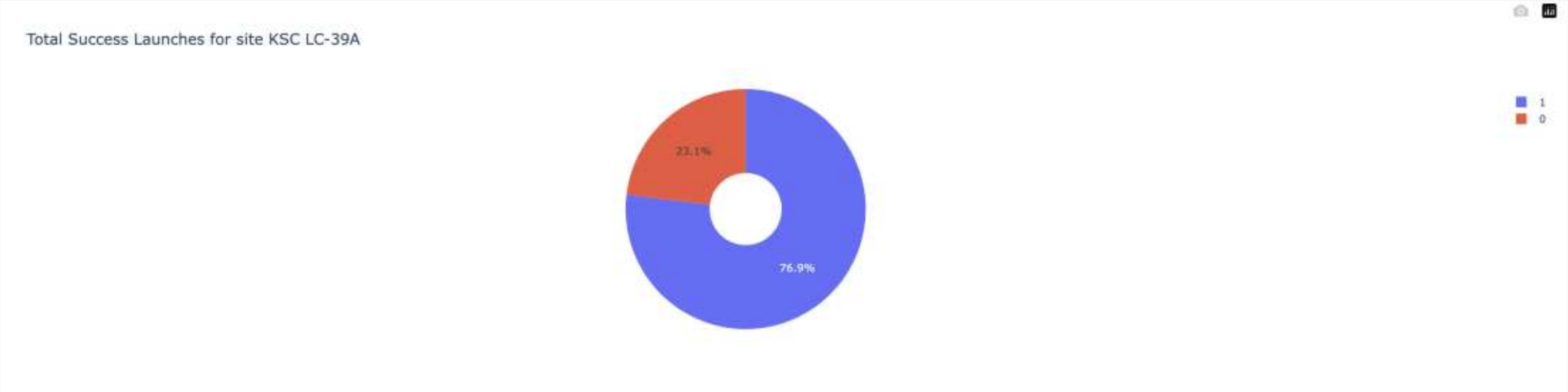
Section 4

Build a Dashboard with Plotly Dash

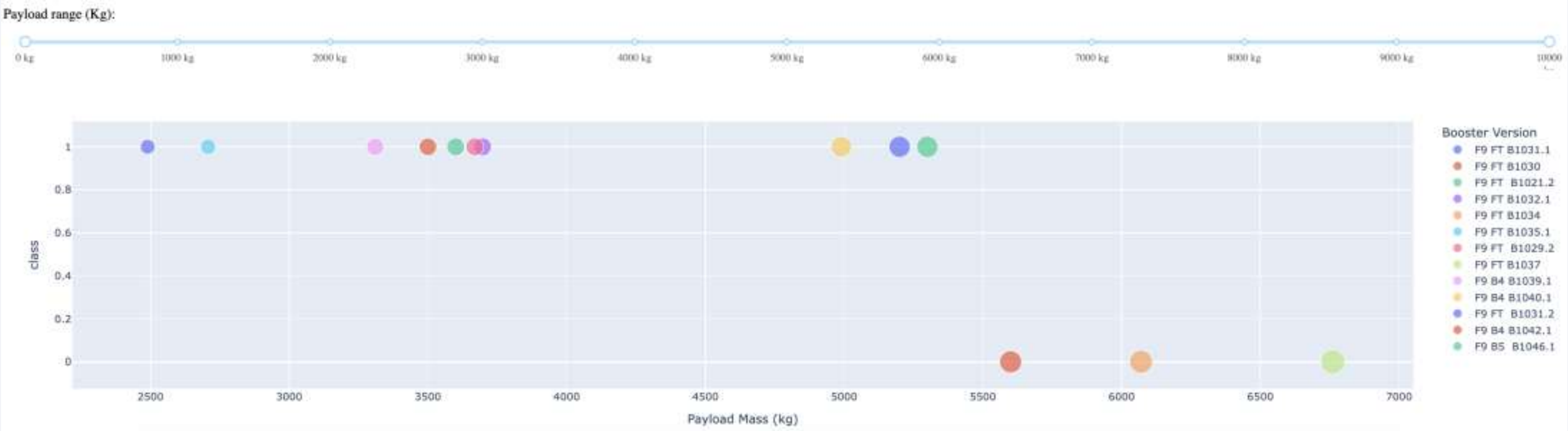
Total success launches by all sites



Highest launch success rate



Payload vs Launch Outcome



Section 5

Predictive Analysis (Classification)

Classification Accuracy

We output the `GridSearchCV` object for logistic regression. We display the best parameters using the data attribute `best_params_` and the accuracy on the validation data using the data attribute `best_score_`.

```
[17]: print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
      print("accuracy :",logreg_cv.best_score_)

tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8464285714285713
```

TASK 5

Calculate the accuracy on the test data using the method `score` :

```
[20]: accuracy = logreg_cv.score(X_test, Y_test)
      accuracy

[20]: 0.8333333333333334
```

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

tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', 'max_depth': 4, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 5, 'splitter': 'random'}
accuracy : 0.875
```

TASK 9

Calculate the accuracy of tree_cv on the test data using the method `score` :

```
[29]: accuracy = tree_cv.score(X_test, Y_test)
      accuracy

[29]: 0.7777777777777778
```

```
[32]: print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
      print("accuracy :",knn_cv.best_score_)

tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}
accuracy : 0.8482142857142858
```

TASK 11

Calculate the accuracy of knn_cv on the test data using the method `score` :

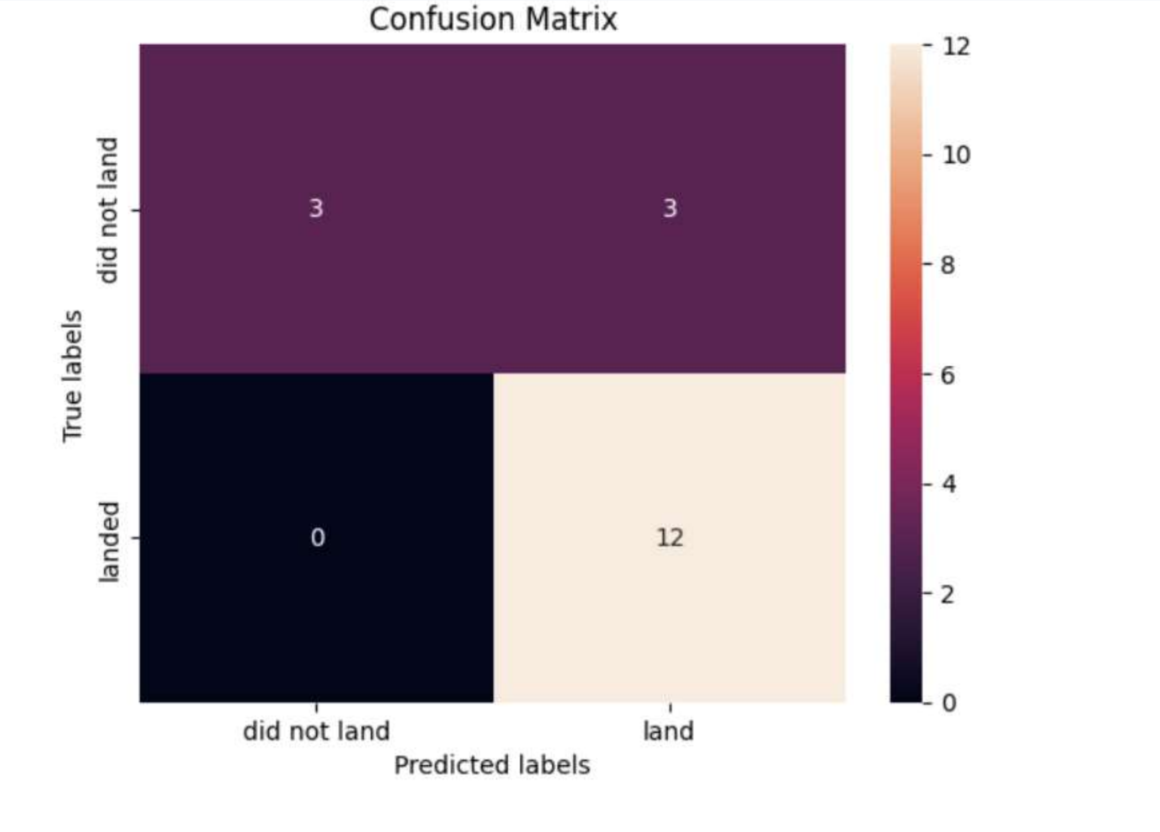
```
[33]: accuracy = knn_cv.score(X_test, Y_test)
      accuracy

[33]: 0.8333333333333334
```

- Best train accuracy: Decision Tree
- Best test accuracy: Support Vector Machine and K nearest neighbor

=> Best model: K nearest neighbor

Confusion Matrix



- Model predicts 3/6 correct for “did not land” true labels and 12/12 correct for “land” true labels

Conclusions

- Our objective was to build a machine learning model for Space Y, aiming to compete with SpaceX. The model's goal is to predict when Stage 1 will land successfully, potentially saving around $100 million USD.
- We utilized data from a public SpaceX API and scraped information from the SpaceX Wikipedia page. The data was labeled and stored in a DB2 SQL database. A dashboard was created for visualization purposes.
- The resulting machine learning model achieved **83% accuracy**. This model can help Allon Mask of SpaceY predict with high accuracy whether a Stage 1 landing will succeed before launch, aiding decisions on whether to proceed with the launch.
- To further enhance model performance, additional data collection could help identify the best machine learning model and improve its accuracy.

Appendix

- Github URL: https://github.com/hoangphong111213/ibm-applied-ds-capstone

Thank you!