IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Hossam Abdeen
27/01/2025

Outline

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

Executive Summary

- Summary of methodologies
  Data Collection
  Data Preparation
  Predictive Analysis
- Leveraging data science achieved an impressive 83.33% accuracy, simplifying tasks that would traditionally require complex mathematics, advanced dynamics, and literal rocket science.

Introduction

- This project looks to Leveraging data science to Predict The success rate of the first stage of a rocket landing this traditionally require complex mathematics, advanced dynamics, and literal rocket science.

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - Data collected from SpaceX and Coursera websites.
- Perform data wrangling
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium
- Perform predictive analysis using classification models
  - Classification models built and tuned using Jupyter Notebook.
  - Four models created: Logistic Regression, SVM, Decision Tree, KNN.

Data Collection

- Data collected from SpaceX and Coursera websites.
- Using requests, BeautifulSoup and Pandas libraries.

Data Collection – SpaceX API

- Using requests and Pandas libraries.
  spacex_url = https://api.spacexdata.com/v4/launches/past
  response = requests.get(spacex_url)
  data = response.json()
  data = pd.json_normalize(data)

- GitHub URL: https://github.com/HossamAbdeenO/SpaceX
  File : APIs.ipynb

In [8]:
```
# SpaceX API URL
spacex_url = "https://api.spacexdata.com/v4/launches/past"

# Get the response from the API
response = requests.get(spacex_url)

static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/

response.status_code

```

Out[8]: 200

In [11]:
```

# Decode the JSON content
data = response.json()

# Convert the JSON list to a DataFrame
data = pd.json_normalize(data)

# Display the first 5 rows of the DataFrame
data.head()

```

Data Collection - Scraping

- Using requests, BeautifulSoup and Pandas libraries.

- GitHub URL: https://github.com/HossamAbdeen0/SpaceX
  File: webscraping.ipynb

```
In [3]: static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"

In [4]: response = requests.get(static_url)

In [5]: soup = BeautifulSoup(response.text, 'html.parser')

In [6]: print(soup.title)

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

In [7]: # Use the find_all function in the BeautifulSoup object, with element type `table`
        # Assign the result to a list called `html_tables`
        html_tables = soup.find_all('table')

In [8]: # Let's print the third table and check its content
        first_launch_table = html_tables[2]
        print(first_launch_table)

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

Data Wrangling

- **Data Import**
- Loaded dataset from a CSV file using Pandas.
- **Missing Values**
- Checked for missing values
- **Data Types**
- Reviewed and confirmed column data types
- **Feature Exploration**
- Examined and counted unique values in key columns like "LaunchSite", "Orbit", and "Outcome".
- **Handling Categorical Variables**
- Identified bad outcomes in the "Outcome" column
- Created a new binary "Class" column, marking good outcomes as 1 and bad outcomes as 0.
- **Data Export**
- Saved the cleaned and wrangled dataset to a new CSV file for further analysis.

GitHub URL: https://github.com/HossamAbdeen0/SpaceX
File: Data wrangling.ipynb

EDA with Data Visualization

•**Payload Mass vs Flight Number**
•**Chart**: Catplot.
•**Purpose**: To show the relationship between flight number and payload mass, highlighting launch success.
•**Flight Number vs Launch Site**
•**Chart**: Stripplot.
•**Purpose**: To examine how flight numbers vary across launch sites, colored by success.
•**Success Rate by Orbit Type**
•**Chart**: Barplot.
•**Purpose**: To compare the average success rate for different orbit types.
•**Launch Success Rate Trend**
•**Chart**: Lineplot.
•**Purpose**: To observe trends in launch success over time.

GitHub URL: https://github.com/HossamAbdeen0/SpaceX
File: EDA_dataviz.ipynb

EDA with SQL

•Selected distinct launch sites from the database.
•Retrieved records from SPACEXTBL for launch sites starting with 'CCA'.
•Calculated the total payload mass for NASA (CRS) missions.
•Calculated the average payload mass for the 'F9 v1.1' booster version.
•Found the date of the first successful landing on a ground pad.
•Identified booster versions with successful drone ship landings and payload mass between 4000 and 6000 kg.
•Counted the total number of successful and failed mission outcomes.
•Retrieved booster versions with the maximum payload mass.
•Extracted mission details for failed drone ship landings in 2015, grouped by month.
•Counted landing outcomes between specific date ranges and ordered by outcome frequency.

GitHub URL: https://github.com/HossamAbdeen0/SpaceX
File: EDA_SQL.ipynb

Build an Interactive Map with Folium

1. **Circles**: Represented launch sites with a 1000m radius for geographic context.
2. **Markers**: Placed at launch sites with custom icons showing site names.
3. **Marker Clustering**: Grouped nearby markers to reduce map clutter.
4. **Mouse Position**: Displays the real-time latitude and longitude of the cursor.
5. **Distance Marker & PolyLine**: Showed the distance from a launch site to a coastline and drew a line between them.

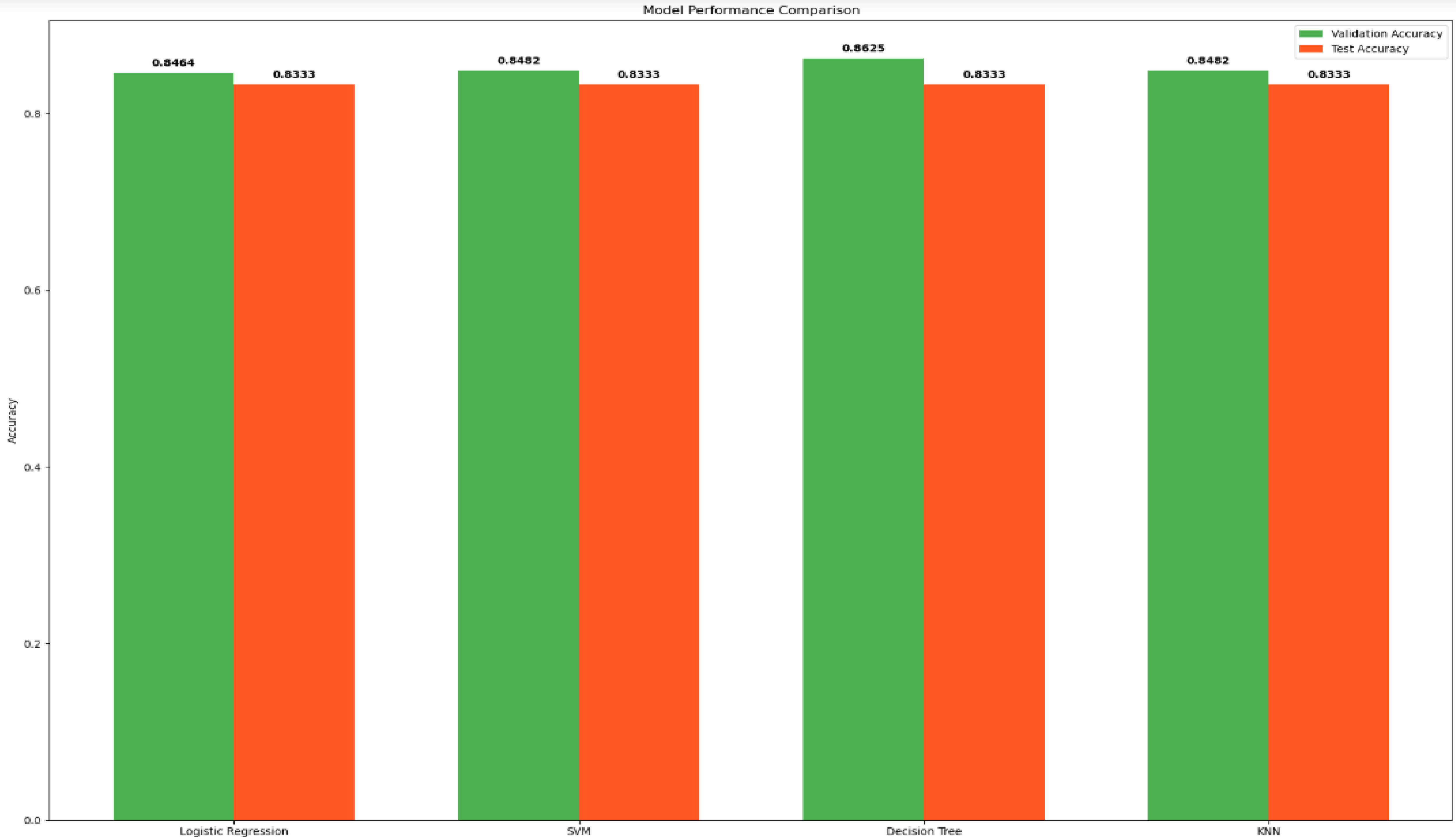
- These objects enhance map interactivity and provide spatial context for the launch sites.

GitHub URL: https://github.com/HossamAbdeen0/SpaceX
File: Location_folium.ipynb

Predictive Analysis (Classification)

- **Data Preparation:**
- Standardized the data using StandardScaler.
- Split the dataset into training (80%) and testing (20%) sets.
- **Model Development: Used** Logistic Regression, SVM, Decision Tree, and K-Nearest Neighbors (KNN) classifiers.
- **Performance Comparison:**
  Although the Decision Tree achieved the highest validation accuracy, all models seem to have similar test performance, suggesting that they generalize similarly well to unseen data.

GitHub URL: https://github.com/HossamAbdeen0/SpaceX
File: Machine_learning_SpaceX.ipynb


Model Performance Comparison
Validation Accuracy
Test Accuracy
0.8464
0.8333
0.8482
0.8333
0.8625
0.8333
0.8482
0.8333
Accuracy
0.0
0.2
0.4
0.6
0.8
Logistic Regression
SVM
Decision Tree
KNN

Results

- Validation Accuracy: The Decision Tree has the highest validation accuracy (86.25%), so it seems to fit the training data best.

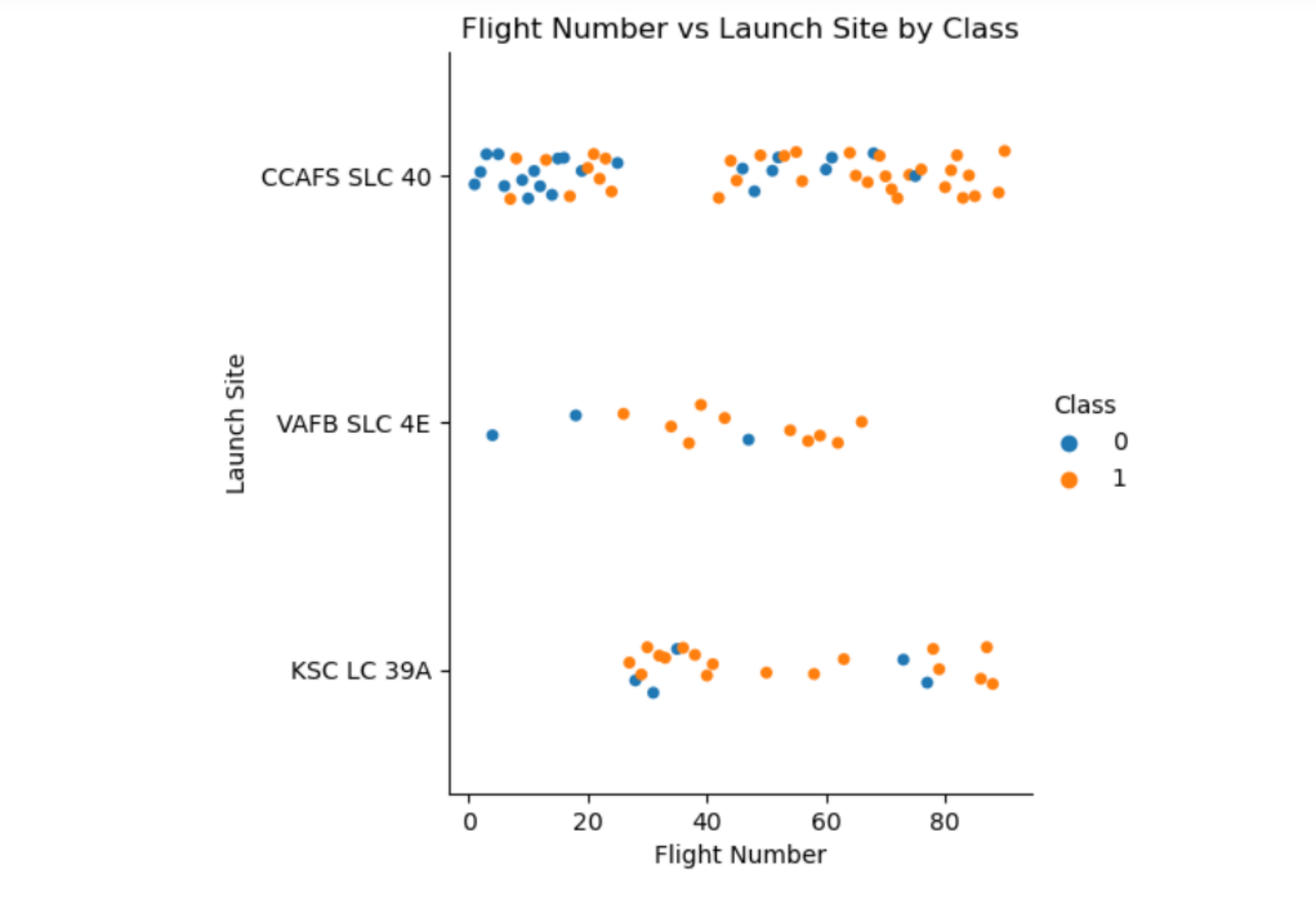
- the accuracy on the test data for all models is the same (83.33%). so, in terms of generalization ability on the test data, they all performed equally well.
- Decision Tree achieved the highest validation accuracy, all models seem to have similar test performance, suggesting that they generalize similarly well to unseen data.

Section 2

Insights drawn from EDA

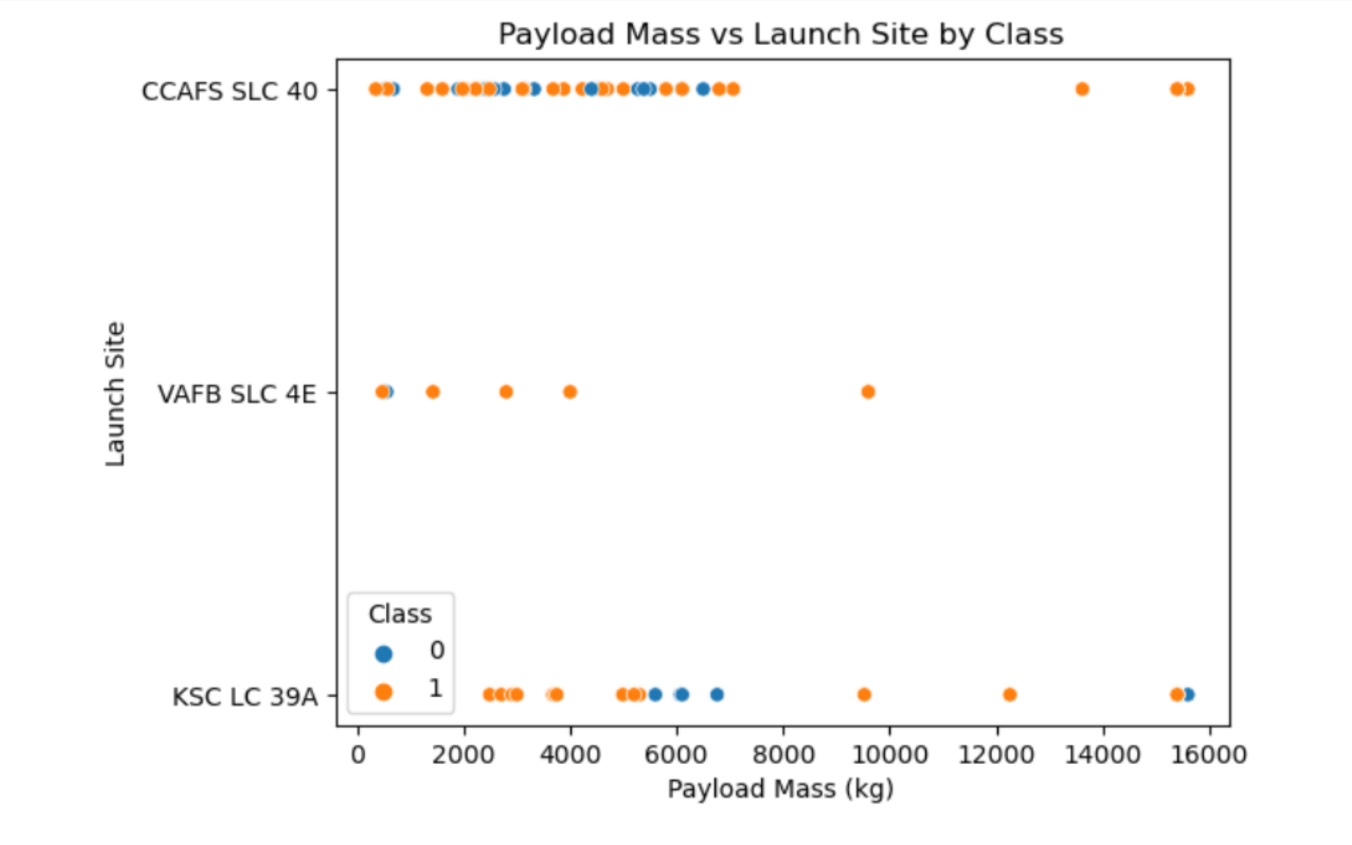
Flight Number vs. Launch Site

- This graph shows the relation between the amount of data for each type and the launch site
- Shows that the site VAFB SLC 4E is the least site that was used
  CCAFS SLC 40 is clearly the most used type



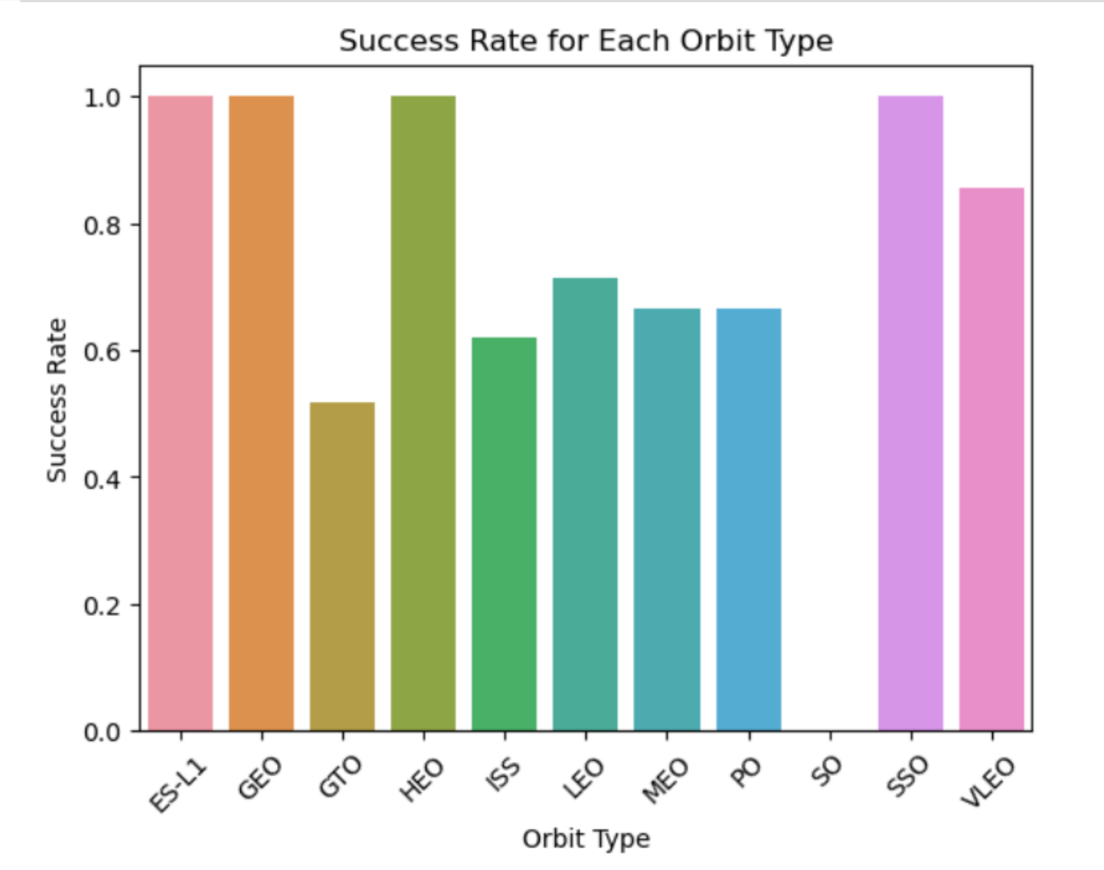
Payload vs. Launch Site

- The graph show that for KSC LC 39A it has a success rate of a 100% from 2000kg to 5500kg
- VAFB SLC 4E works great for both low and intermediate wight with no data for high wight
- CCAFS SLC 40 preforms best with high wights above 13000kg



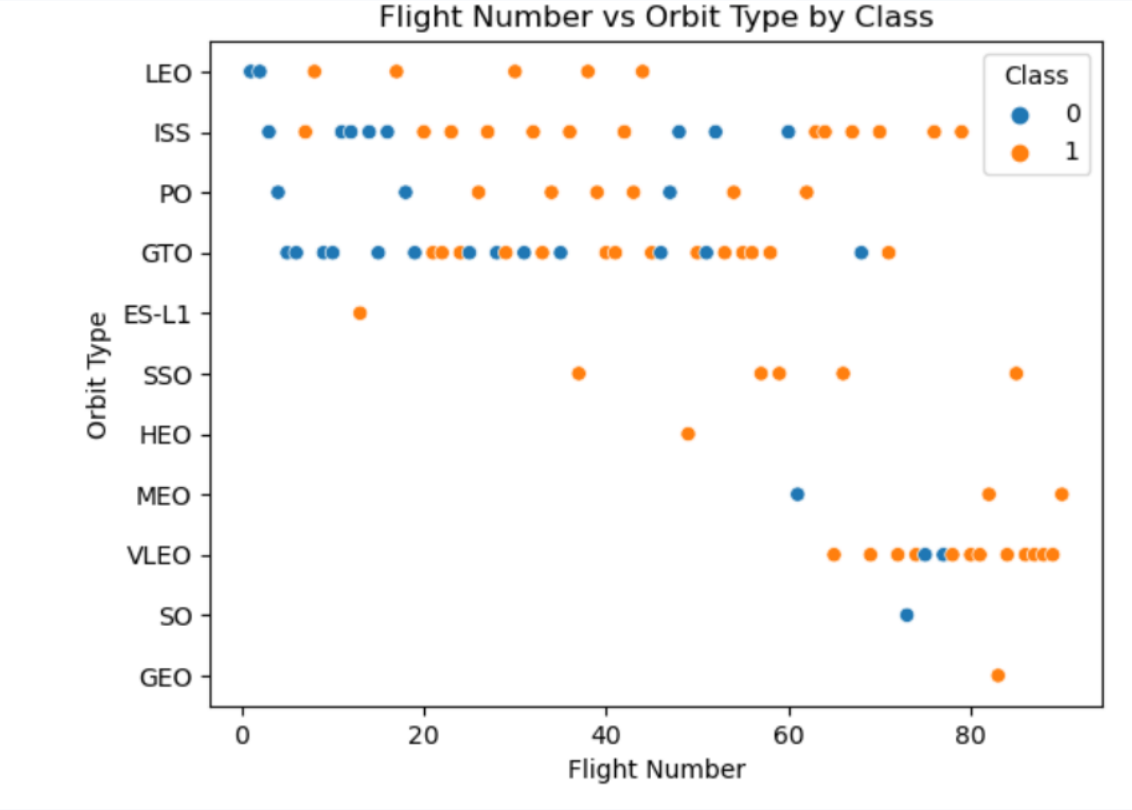
Success Rate vs. Orbit Type

- The graph shows the orbit types that has the best and worst performances
ES-L1 , GEO , HEO and SSO are all tied for 100% success rate
GTO is the worst with around 50% success rate



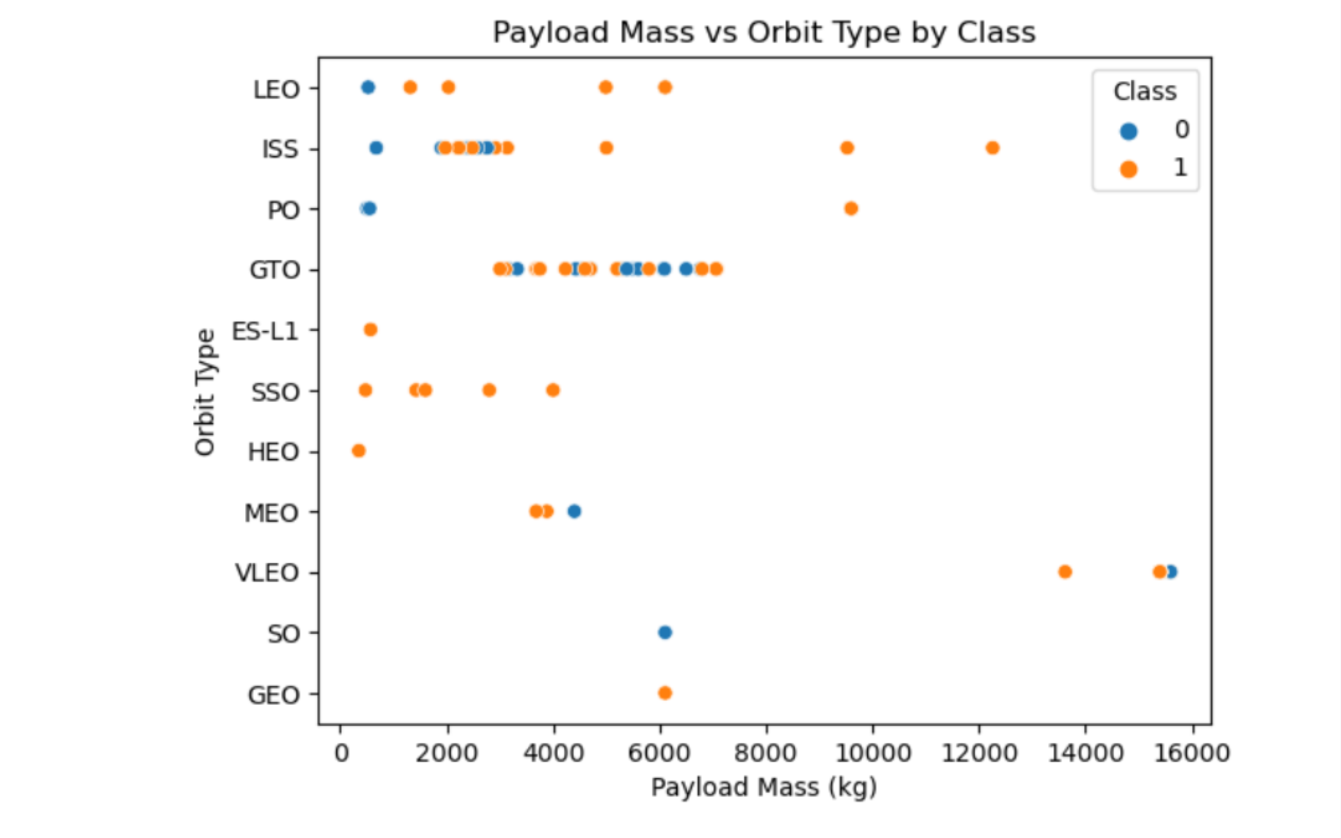
Flight Number vs. Orbit Type

- The graph shows the amount of data we have for each orbit type, so we don't get confused by the percentage point
for example, we have viewed that ES-L1 has a success rate of 100% but now we can see that there was only one trip for that orbit
we would be more impressed with for example VLEO



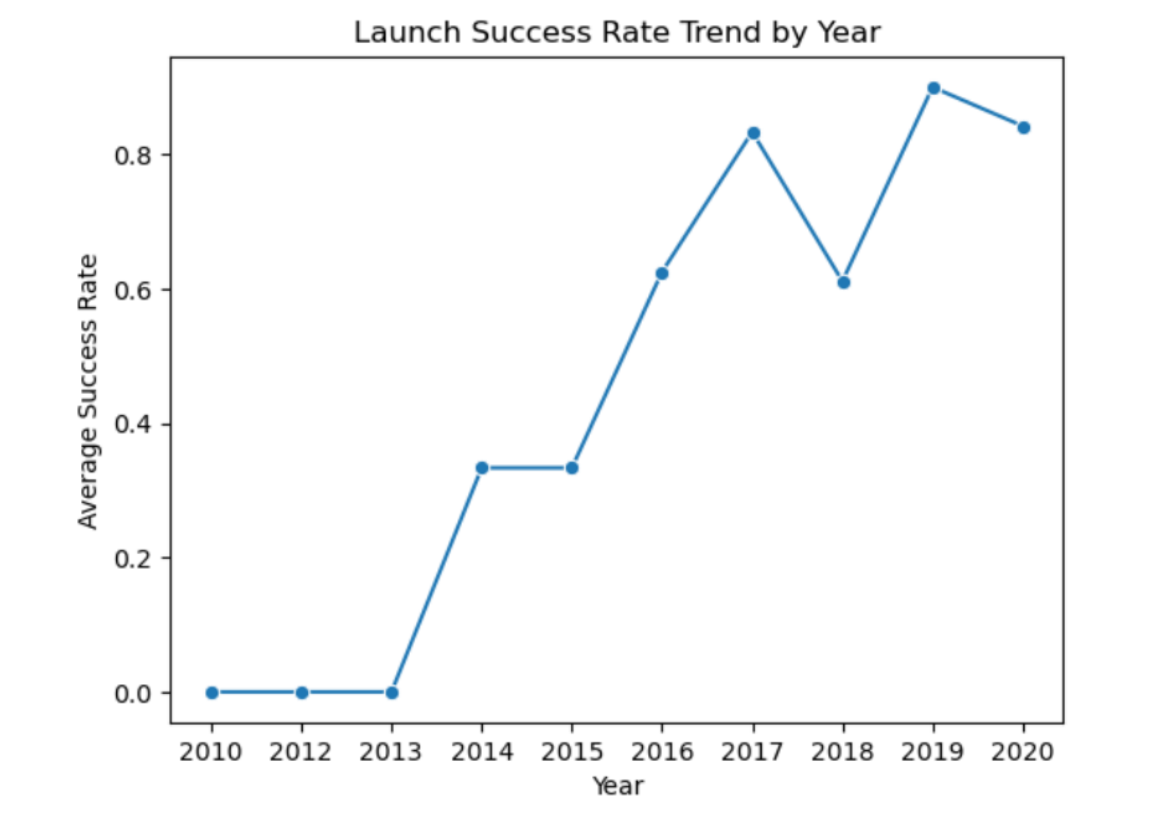
Payload vs. Orbit Type

- The graph shows a lot of orbit types that seems to preform better when the ship's payload is from intermediate to high like LEO, ISS and PO



Launch Success Yearly Trend

- Shows that lunch success rate started improving since 2013 peaked in 2019



All Launch Site Names

- Simple select from query

In [6]:

```python
import sqlite3

# Connect to the database
con = sqlite3.connect("my_data1.db")
cur = con.cursor()

# Execute the SQL query
query = 'SELECT DISTINCT "Launch_Site" FROM SPACEXTBL;'
cur.execute(query)

# Fetch and display the results
launch_sites = cur.fetchall()
for site in launch_sites:
    print(site[0])

# Close the connection
con.close()
```

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

Launch Site Names Begin with 'CCA'

- select query using where and limit
- In [8]:

```
# Connect to the database
con = sqlite3.connect("my_data1.db")
cur = con.cursor()
# Execute the SQL query
query = '''
SELECT *
FROM SPACEXTBL
WHERE "Launch_Site" LIKE 'CCA%'
LIMIT 5;
'''
cur.execute(query)

# Fetch and display the results
records = cur.fetchall()
for record in records:
    print(record)

# Close the connection
con.close()
```

```
('2010-06-04', '18:45:00', 'F9 v1.0  B0003', 'CCAFS LC-40', 'Dragon Spacecraft Qualification Unit', 0, 'LEO', 'SpaceX', 'Suc
cess', '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)', 'Succes
s', '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 at
tempt')
('2013-03-01', '15:10:00', 'F9 v1.0  B0007', 'CCAFS LC-40', 'SpaceX CRS-2', 677, 'LEO (ISS)', 'NASA (CRS)', 'Success', 'No a
ttempt')
```

Total Payload Mass

- Select sum query using where

In [9]:

```
# Connect to the database
con = sqlite3.connect("my_data1.db")
cur = con.cursor()

# Execute the SQL query
query = '''
SELECT SUM("PAYLOAD_MASS__KG_") AS Total_Payload_Mass
FROM SPACEXTBL
WHERE "Customer" = 'NASA (CRS)';
'''
cur.execute(query)

# Fetch and display the result
total_payload_mass = cur.fetchone()
print(f"Total Payload Mass by NASA (CRS): {total_payload_mass[0]} kg")

# Close the connection
con.close()
```

```
Total Payload Mass by NASA (CRS): 45596 kg
```

Average Payload Mass by F9 v1.1

- Select average query using where

In [10]:

```python
# Connect to the database
con = sqlite3.connect("my_data1.db")
cur = con.cursor()

# Execute the SQL query
query = '''
SELECT AVG("PAYLOAD_MASS__KG_") AS Average_Payload_Mass
FROM SPACEXTBL
WHERE "Booster_Version" = 'F9 v1.1';
'''
cur.execute(query)

# Fetch and display the result
average_payload_mass = cur.fetchone()
print(f"Average Payload Mass for Booster Version F9 v1.1: {average_payload_mass[0]} kg")

# Close the connection
con.close()
```

```
Average Payload Mass for Booster Version F9 v1.1: 2928.4 kg
```

First Successful Ground Landing Date

- Select minimum query using where

```
In [11]:
# Connect to the database
con = sqlite3.connect("my_data1.db")
cur = con.cursor()

# Execute the SQL query
query = '''
SELECT MIN("Date") AS First_Successful_Landing
FROM SPACEXTBL
WHERE "Landing_Outcome" = 'Success (ground pad)';
'''
cur.execute(query)

# Fetch and display the result
first_successful_landing = cur.fetchone()
print(f"Date of First Successful Landing on Ground Pad: {first_successful_landing[0]}")

# Close the connection
con.close()
```

```
Date of First Successful Landing on Ground Pad: 2015-12-22
```

Successful Drone Ship Landing with Payload between 4000 and 6000

- Select query using where and “And”

In [12]:

```
# Connect to the database
con = sqlite3.connect("my_data1.db")
cur = con.cursor()

# Execute the SQL query
query = '''
SELECT DISTINCT "Booster_Version"
FROM SPACEXTBL
WHERE "Landing_Outcome" = 'Success (drone ship)'
  AND "PAYLOAD_MASS__KG_" > 4000
  AND "PAYLOAD_MASS__KG_" < 6000;
'''
cur.execute(query)

# Fetch and display the results
boosters = cur.fetchall()
print("Boosters with success in drone ship and payload mass between 4000 and 6000 kg:")
for booster in boosters:
    print(booster[0])

# Close the connection
con.close()
```

```
Boosters with success in drone ship and payload mass between 4000 and 6000 kg:
F9 FT B1022
F9 FT B1026
F9 FT  B1021.2
F9 FT  B1031.2
```

Total Number of Successful and Failure Mission Outcomes

- Select query using group by

In [13]:

```
# Connect to the database
con = sqlite3.connect("my_data1.db")
cur = con.cursor()

# Execute the SQL query
query = '''
SELECT "Mission_Outcome", COUNT(*) AS Total_Count
FROM SPACEXTBL
GROUP BY "Mission_Outcome";
'''
cur.execute(query)

# Fetch and display the results
mission_outcomes = cur.fetchall()
print("Total Number of Successful and Failed Mission Outcomes:")
for outcome in mission_outcomes:
    print(f"{outcome[0]}: {outcome[1]}")

# Close the connection
con.close()
```

```
Total Number of Successful and Failed Mission Outcomes:
Failure (in flight): 1
Success: 98
Success : 1
Success (payload status unclear): 1
```

Boosters Carried Maximum Payload

- Select query using where (with nested select max)

```
# Execute the SQL query
query = '''
SELECT "Booster_Version"
FROM SPACEXTBL
WHERE "PAYLOAD_MASS__KG_" = (
    SELECT MAX("PAYLOAD_MASS__KG_")
    FROM SPACEXTBL
);
'''
cur.execute(query)

# Fetch and display the results
boosters = cur.fetchall()
for booster in boosters:
    print(booster[0])

# Close the connection
con.close()
```

```
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

- Select query using nested cases with when and ending with where

```
query = '''
SELECT
    CASE
        WHEN substr("Date", 6, 2) = '01' THEN 'January'
        WHEN substr("Date", 6, 2) = '02' THEN 'February'
        WHEN substr("Date", 6, 2) = '03' THEN 'March'
        WHEN substr("Date", 6, 2) = '04' THEN 'April'
        WHEN substr("Date", 6, 2) = '05' THEN 'May'
        WHEN substr("Date", 6, 2) = '06' THEN 'June'
        WHEN substr("Date", 6, 2) = '07' THEN 'July'
        WHEN substr("Date", 6, 2) = '08' THEN 'August'
        WHEN substr("Date", 6, 2) = '09' THEN 'September'
        WHEN substr("Date", 6, 2) = '10' THEN 'October'
        WHEN substr("Date", 6, 2) = '11' THEN 'November'
        WHEN substr("Date", 6, 2) = '12' THEN 'December'
    END AS Month,
    "Landing_Outcome",
    "Booster_Version",
    "Launch_Site"
FROM SPACEXTBL
WHERE "Landing_Outcome" = 'Failure (drone ship)'
  AND substr("Date", 0, 5) = '2015';
'''
cur.execute(query)
records = cur.fetchall()
for record in records:
    print(record)
con.close()
```

```
('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

- Select From Where Group by Order by query

In [16]:

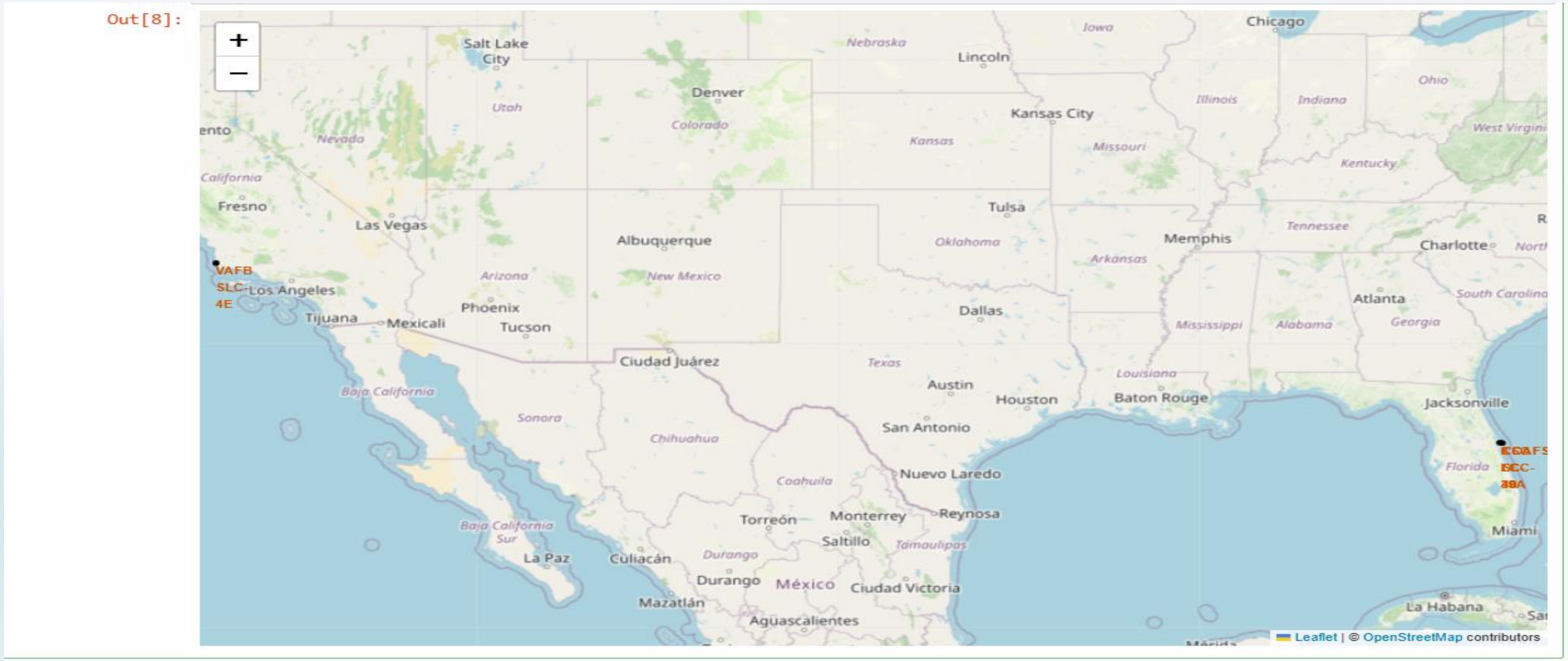
```python
# Connect to the database
con = sqlite3.connect("my_data1.db")
cur = con.cursor()

# Execute the SQL query
query = '''
SELECT "Landing_Outcome", COUNT(*) AS Outcome_Count
FROM SPACEXTBL
WHERE "Date" BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY "Landing_Outcome"
ORDER BY Outcome_Count DESC;
'''
cur.execute(query)

# Fetch and display the results
outcomes = cur.fetchall()
for outcome in outcomes:
    print(f"{outcome[0]}: {outcome[1]}")

# Close the connection
con.close()
```

```
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 map with all launch sites



- Loop through the launch sites _ circle for each launch site_ markers for each launch site

Location map with marker on launch sites

- Iterators _markers _ marker_cluster

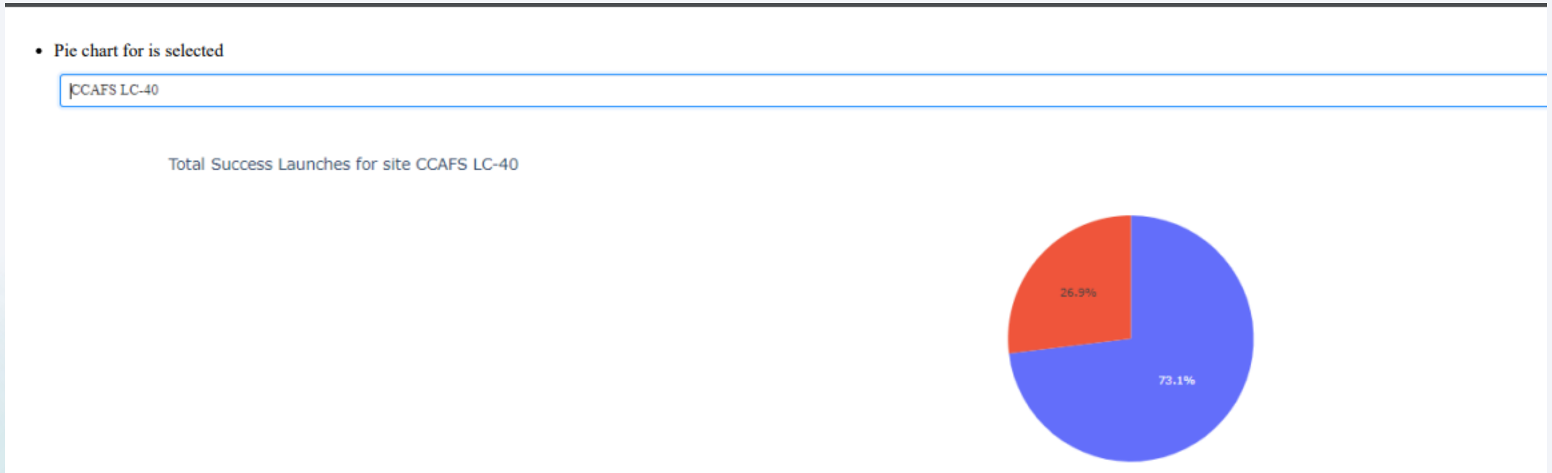
<Folium Map Screenshot 3>

- distance_marker_ markers _ marker_cluster

Section 4

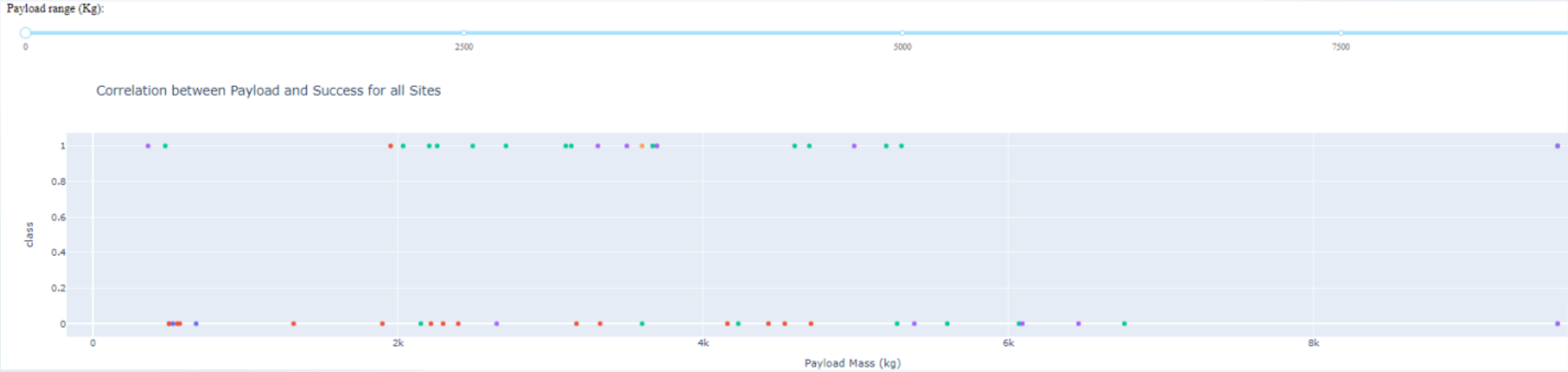
Build a Dashboard with Plotly Dash

SpaceX Launch Dashboard Selected site



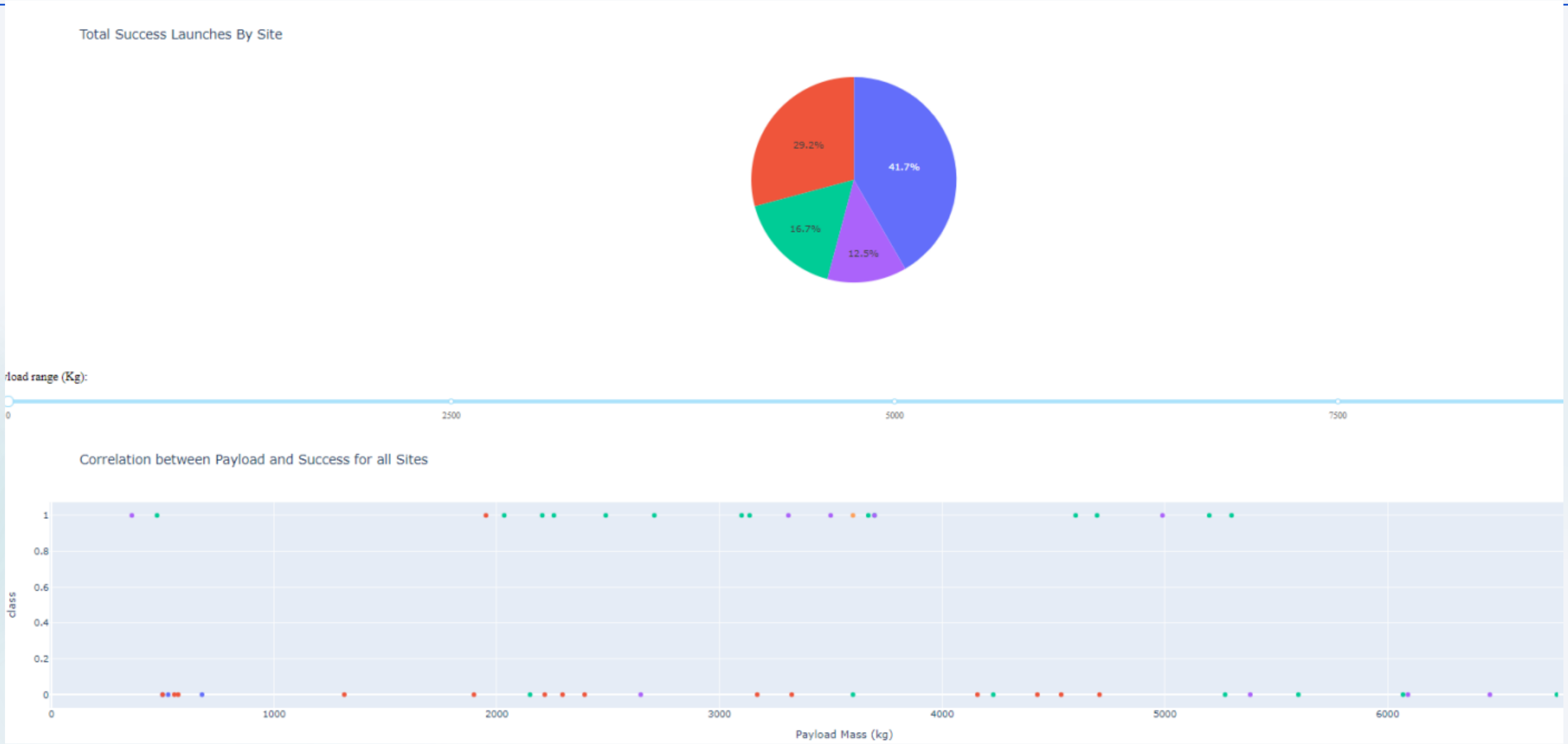
- Total success launches for the selected site

SpaceX Launch Dashboard Selected payload



- The correlation between the payload and the success rate

SpaceX Launch Records Dashboard



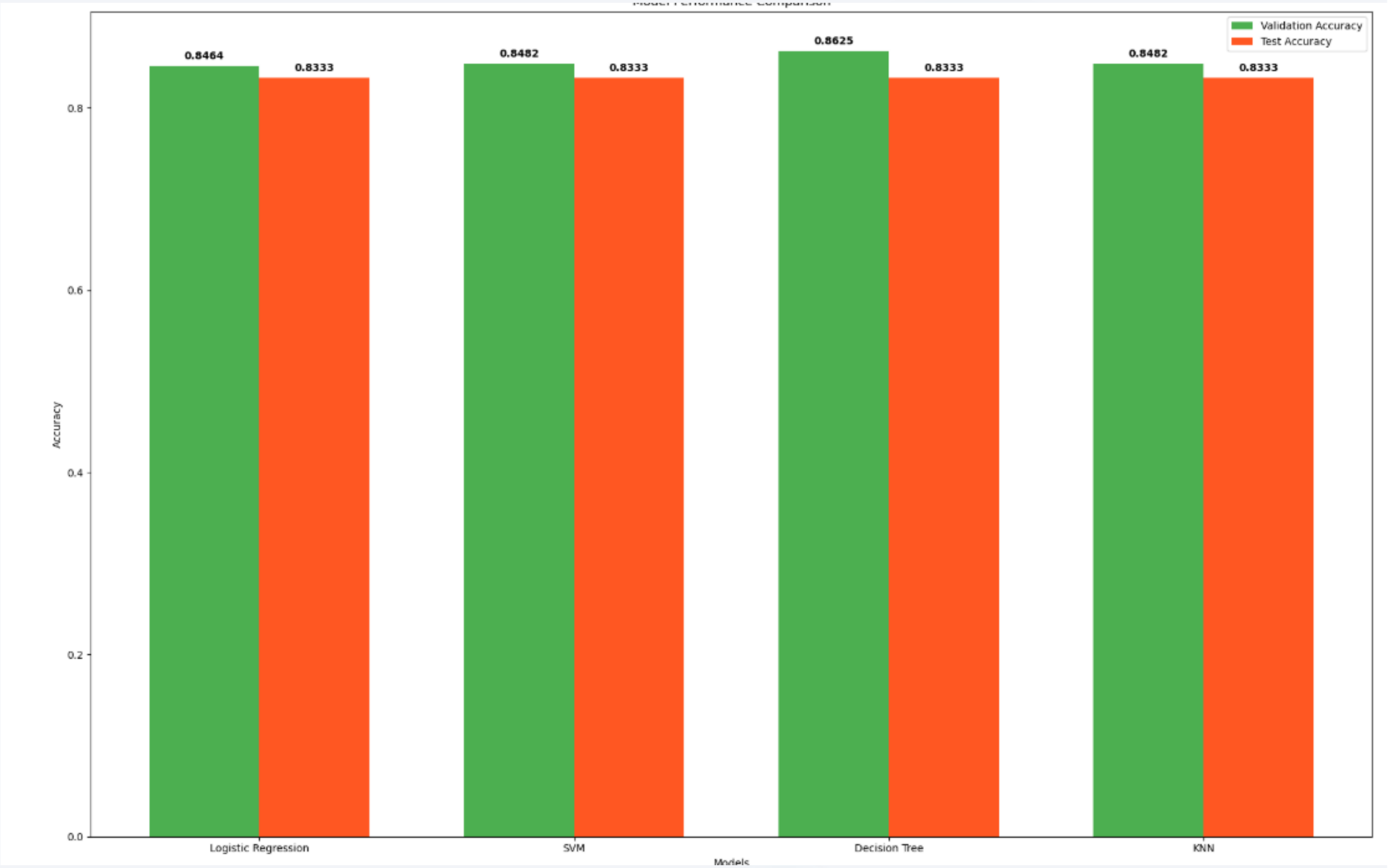
- The total success launch by site and payload mass

Section 5

Predictive Analysis (Classification)

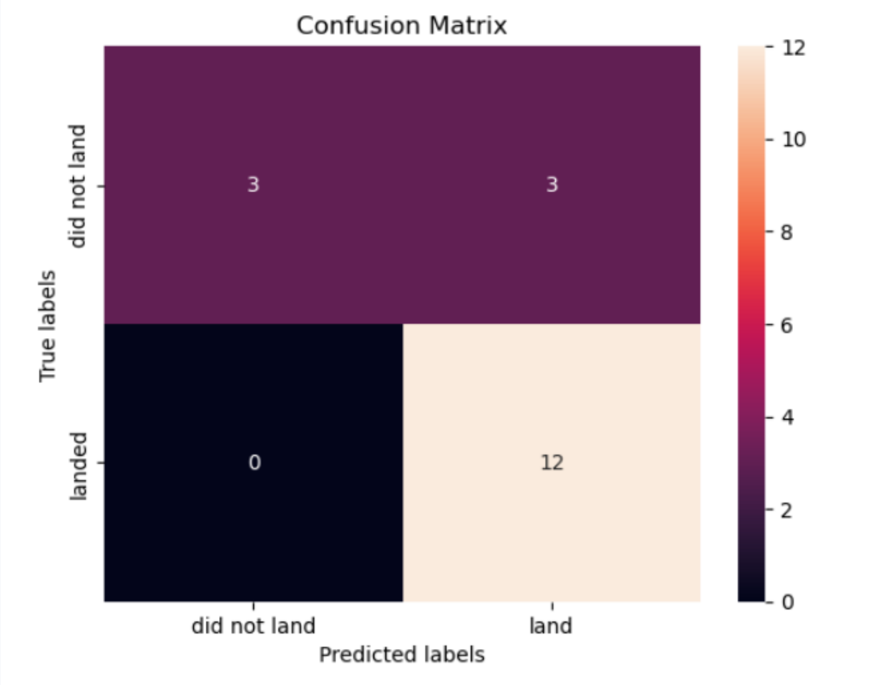
Classification Accuracy

- When it comes to training the Decision Tree model preformed best with 86.25% accuracy and when it comes to Testing all four models preformed with the same accuracy 83.33%
-



Confusion Matrix

- Since All models preformed equally good when it comes to testing all confusion matrices yielded the same result with 12 True lands



Conclusions

- Validation Accuracy: The Decision Tree has the highest validation accuracy (86.25%), so it seems to fit the training data best.

- the accuracy on the test data for all models is the same (83.33%). so in terms of generalization ability on the test data, they all performed equally well.

- Although the Decision Tree achieved the highest validation accuracy, all models seem to have similar test performance, suggesting that they generalize similarly well to unseen data.

Appendix

- https://github.com/HossamAbdeen0/SpaceX

Thank you!