IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Vicente Soler
14/11/2024

Outline

Executive Summary

Methodology & Results

- Methodology
  - Data Collection: Gather data using APIs or web scraping techniques.
  - Data Wrangling: Clean and structure the data, ensuring it is in the proper format.
  - Exploratory Data Analysis (EDA): Analyze data to ensure correctness and identify patterns.
  - Data Analysis: Conduct in-depth analysis using appropriate statistical or machine learning techniques.
  - Result Extraction and Presentation: Present key findings clearly and compellingly.

- Results
  - Rocket first-stage reuse significantly reduces costs.
  - SpaceX is succeeding with first-stage rocket reuses.
  - FT Booster has the best performance.
  - Kennedy Space Center (KSC LC-39A) has the highest launch success rate.

Background & Problems

Introduction

- Project background and context
  - First-stage rocket reuse is crucial for reducing costs.
  - Investigating the relationship between boosters, launch sites, and success rates.
- Problems you want to find answers
  - How does first-stage reuse impact launch costs?
  - Which booster has the highest success rate?
  - Which launch site has the best success rate?

Section 1

Methodology

Methodology

Data collection methodology:

Data collection is gathered from two primary sources:

- Via API, specifically the Space X REST API.
- Via Web Scrapping, Falcon 9 launch records HTML table from Wikipedia

Perform data wrangling

Clean and wrangle data, correcting errors and categorizing attributes.

Perform exploratory data analysis (EDA) using visualization and SQL

Perform interactive visual analytics using Folium and Plotly Dash

Perform predictive analysis using classification models

Build and evaluate classification models in a machine learning pipeline.

Identify the best-performing model through iterative analysis.

Data Collection

- **API Access**:
  - The notebook utilizes the requests library to make HTTP GET requests to the SpaceX API (e.g., https://api.spacexdata.com/v4/rockets/{id}).
  - The API calls are made iteratively for each rocket ID found in the dataset. This allows for the extraction of detailed information such as rocket names, configurations, and technical specifications.
  - Data received from the API is parsed in JSON format and processed to extract relevant fields, which are then appended to a list or DataFrame for further analysis.
- **Web Scraping**:
  - Although not explicitly confirmed from the code cells reviewed, web scraping in Python typically involves using libraries like BeautifulSoup or selenium to parse HTML content and extract specific information.
  - This method would be used if certain data were unavailable through the API or to supplement API data with additional context (e.g., web pages with specific launch details or historical data).

Data Collection – SpaceX API

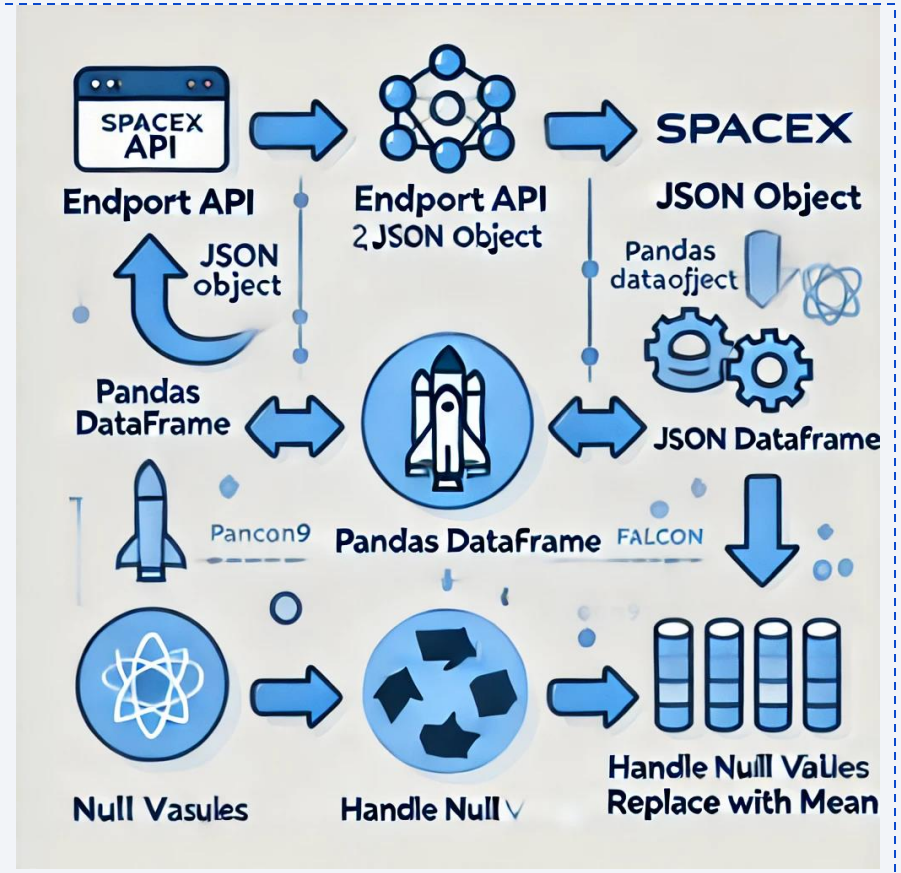
Endpoints:

- https://api.spacexdata.com/v4/rockets/
- https://api.spacexdata.com/v4/launchpads/
- https://api.spacexdata.com/v4/launches/past
- https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_call_spacex_api.json

Get sample:

```
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'])
```

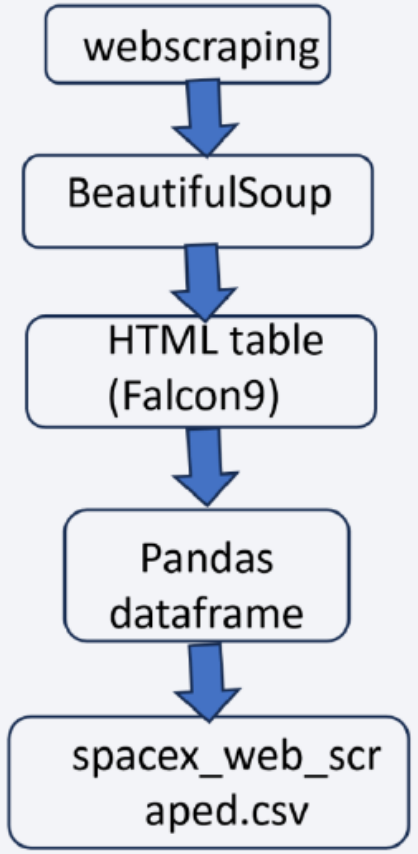
- https://github.com/visolerga/IBMCapstone/blob/bf146689678e474e3c74fd72b1db580ca2e34b8c/jupyter-labs-spacex-data-collection-api-v2.ipynb



Data Collection - Scraping

Web Scraping Process Overview:

- **Web Scraping**:
  - The process begins with sending a request to the target web page using a library like `requests` or automating a browser session with `selenium`.
  - The web page's raw HTML content is retrieved and prepared for parsing.
- **BeautifulSoup**:
  - The HTML content is parsed using `BeautifulSoup`, a Python library that makes it easy to navigate and search the document structure.
  - This step allows for locating specific data elements within the page, such as tables, by targeting tags, classes, or IDs.
- **HTML Table Extraction**:
  - Specific tables, such as those with data about Falcon 9 launches, are extracted by identifying their HTML structure.
  - The data is cleaned and prepared to ensure relevant rows and columns are selected.
- **Pandas DataFrame**:
  - The extracted data is then converted into a `pandas` DataFrame for better manipulation and analysis.
  - This allows for further data cleaning and transformations as needed.
- **Export to CSV**:
  - Finally, the structured DataFrame is exported to a CSV file, making it easy to store and share the collected data.
  - The file is saved with a relevant name.



- https://github.com/visolerga/IBMCapstone/blob/bf146689678e474e3c74fd72b1db580ca2e34b8c/jupyter-labs-webscraping.ipynb

Data Wrangling

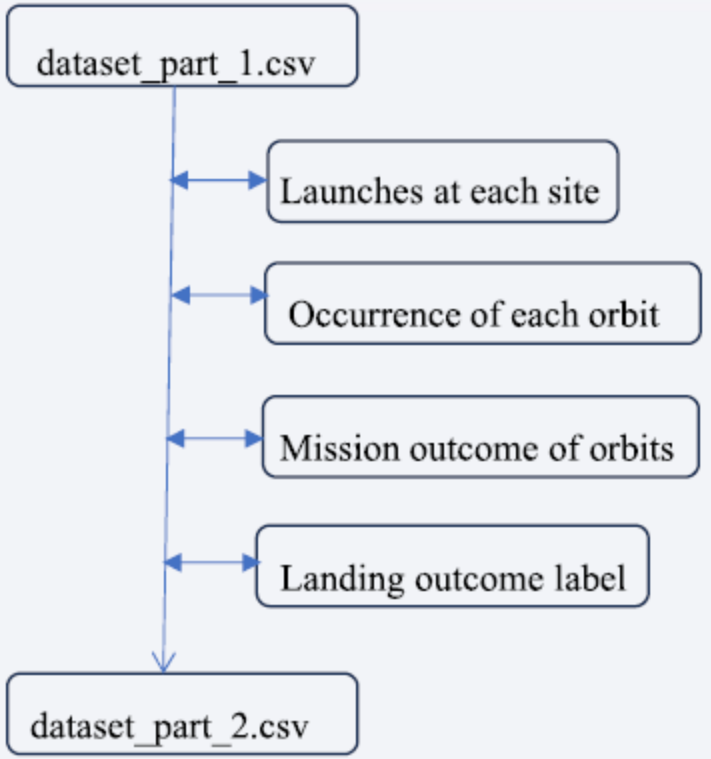
We could consider the main key phrases like this:

```
landing_outcomes = df['Outcome'].value_counts()
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
landing_class = [0 if outcome in bad_outcomes else 1 for outcome in df['Outcome']]
df['Class']=landing_class
```

```
for i,outcome in enumerate(landing_outcomes.keys()):
    print(i,outcome)
```

```
0 True ASDS
1 None None
2 True RTLS
3 False ASDS
4 True Ocean
5 False Ocean
6 None ASDS
7 False RTLS
```

In bad_outcomes, we have identified which results are failures, and by following the process, we can obtain the successful outcomes for the orbits and launch sites



https://github.com/visolerga/IBMCapstone/blob/bf146689678e474e3c74fd72b1db580ca2e34b8c/labs-jupyter-spacex-Data%20wrangling-v2.ipynb

EDA with Data Visualization

Summary of charts.

- Visualize the relationship between different parameters
- Payload and flight number for how the Flight
- Number Payload variables would affect the launch outcome
- Flight number and launch site to find the patterns in the scatter point plots
- Payload mass and launch site to find relationship launching rockets with different masses
- Orbits and class to find success rate
- Flight number and orbit type to know outcomes at these orbits
- Launch success rate and year to know success rate trend

We need to analyze the data visually to identify anomalies or patterns, observe how different parameters influence the outcomes, and track their evolution over time.

https://github.com/visolerga/IBMCapstone/blob/bf146689678e474e3c74fd72b1db580ca2e34b8c/jupyter-labs-eda-dataviz-v2.ipynb

EDA with SQL

Summary.

- Some %sql magic for preparation.
- "SELECT DISTINCT Launch_Site FROM {table_name}"
- "SELECT * FROM {table_name} WHERE Launch_Site LIKE 'CCA%' LIMIT 5"
- "SELECT * FROM {table_name} WHERE Launch_Site LIKE 'CCA%' LIMIT 5"
- "SELECT SUM(PAYLOAD_MASS__KG_) AS TOTAL_PAYLOAD FROM {table_name} WHERE Customer LIKE '%NASA (CRS)%'"
- "SELECT AVG(PAYLOAD_MASS__KG_) AS AVG_PAYLOAD FROM {table_name} WHERE Booster_Version LIKE 'F9 v1.1%'"
- "SELECT MIN(Date) as minDate FROM {table_name} WHERE Landing_Outcome = 'Success (ground pad)'"
- "SELECT Date, "Time (UTC)" FROM {table_name} WHERE Date = ? and Landing_Outcome = 'Success (ground pad)' ORDER BY "Time (UTC)" ASC LIMIT 1"
- "SELECT DISTINCT Booster_Version FROM {table_name} WHERE Landing_Outcome = 'Success (drone ship)' AND PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000"
- "SELECT Mission_Outcome, COUNT(*) as outcome_count FROM {table_name} GROUP BY Mission_Outcome"
- "SELECT CASE WHEN Mission_Outcome like '%Success%' THEN 'Success' ELSE 'Failure' END as Outcome_Type, COUNT(*) as outcome_count FROM {table_name} GROUP BY Outcome_Type"
- "SELECT distinct Booster_Version FROM {table_name} WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_)FROM {table_name})"
- "SELECT CASE WHEN substr(Date, 6, 2) = '01' THEN 'January' ---- WHEN substr(Date, 6, 2) = '12' THEN 'December' END as Month_Name, Booster_Version, Launch_Site FROM {table_name} WHERE Landing_Outcome = 'Failure (drone ship)' AND substr(Date, 0, 5) = '2015'"
- "SELECT Landing_Outcome, COUNT(*) as outcome_count FROM {table_name} WHERE Date BETWEEN '2010-06-04' AND '2017-03-20' GROUP BY Landing_Outcome ORDER BY outcome_count DESC "

https://github.com/visolerga/IBMCapstone/blob/bf146689678e474e3c74fd72b1db580ca2e34b8c/jupyter-labs-eda-sql-coursera_sqllite.ipynb

Build an Interactive Map with Folium

Markers.

- Launch locations as circles
- Launch sites Markers on a map representing the success/failed launches for each site on the map
- Line to measure distance between nearest coast to location, location to nearest railway station and location to nearest town.

Reasons:

- Added to get insights about the requirements and the possible locations to avoid for setting launch sites.
- Also asked about distance to equatorian line.

https://github.com/visolerga/IBMCapstone/blob/02c56e9f3b8bcc12de348f79adb820e65c7cb01f/my_map.html

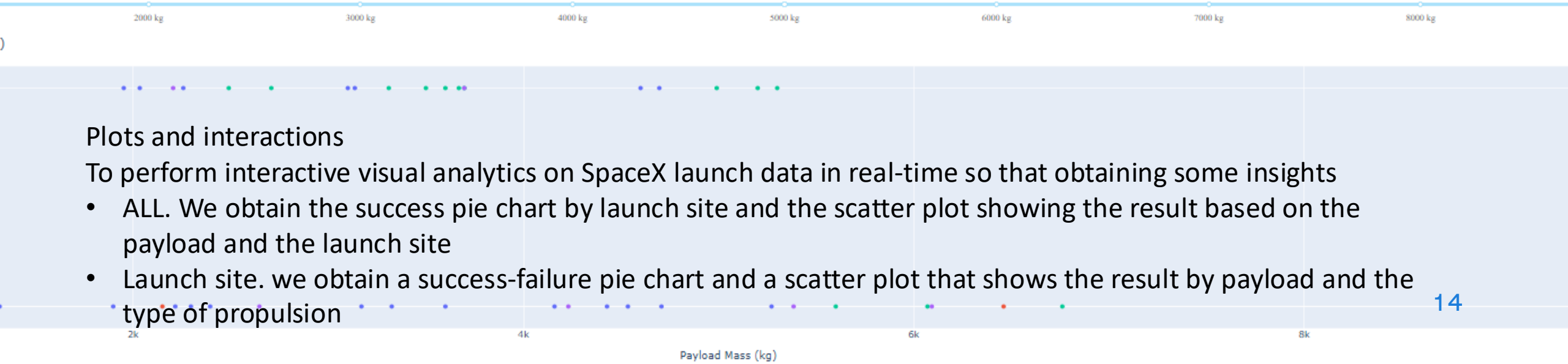
https://github.com/visolerga/IBMCapstone/blob/02c56e9f3b8bcc12de348f79adb820e65c7cb01f/lab-jupyter-launch-site-location-v2.ipynb

Build a Dashboard with Plotly Dash

Summary:

- SpaceX launch records dashboard
  - A pie chart visualizing
    - Launch success counts per launch site if ALL is selected
    - Launch success-fail per lauch site if a lauch site is selected
- A range slider to choose payload.
- A payload-outcome scatter plot to observe how payload may be correlated with mission outcomes for selected site/booster version.

https://github.com/visolerga/IBMCapstone/blob/02c56e9f3b8bcc12de348f79adb820e65c7cb01f/InteractiveDashboard/spacex_dash_app.py



Plots and interactions

To perform interactive visual analytics on SpaceX launch data in real-time so that obtaining some insights

- ALL. We obtain the success pie chart by launch site and the scatter plot showing the result based on the payload and the launch site
- Launch site. we obtain a success-failure pie chart and a scatter plot that shows the result by payload and the type of propulsion

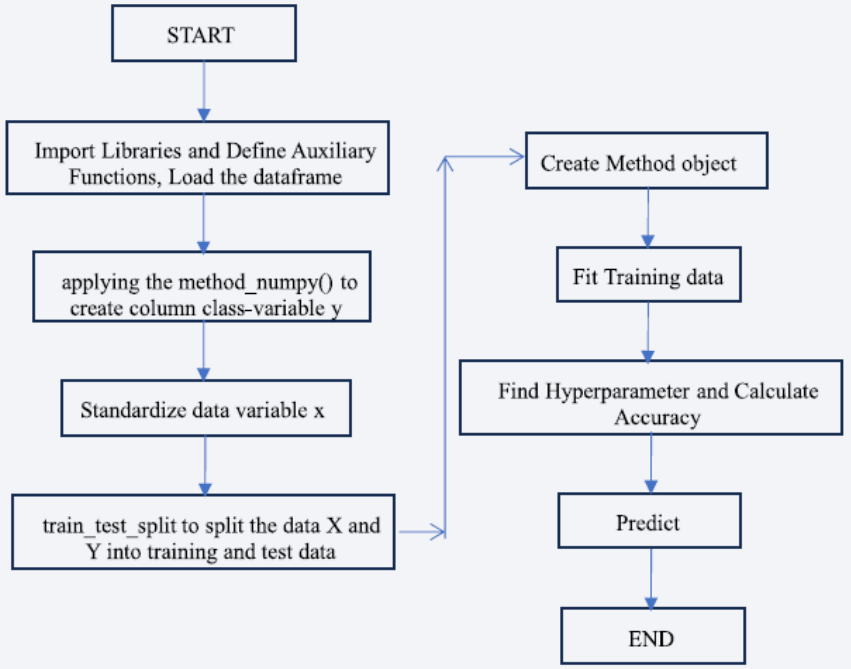
Predictive Analysis (Classification)

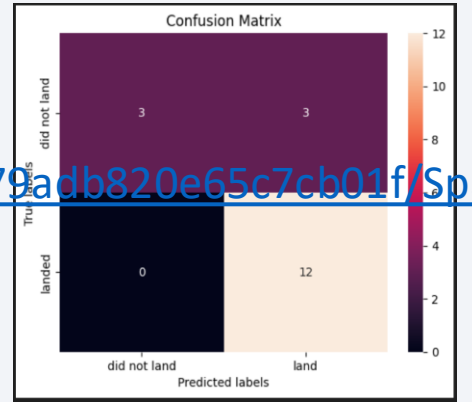
- Standardize data for proper evaluation.
- Split data for training and testing, train for each model.
- Calculate accuracy with test data after finding best hyperparameters.
- Find out which methods performs best.
- Create machine learning pipeline with that method for predicting new outcomes.

Some Phrases.

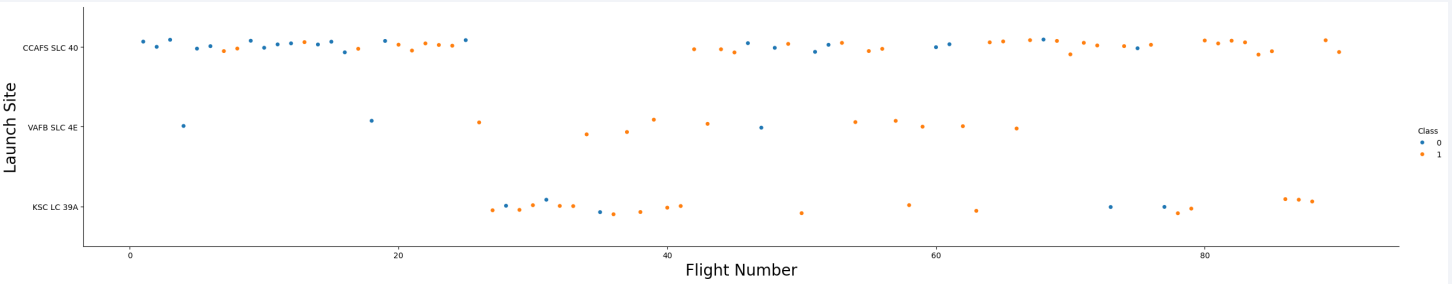
```
transform = preprocessing.StandardScaler()
X = transform.fit_transform(X)
logreg_test_accuracy = logreg_cv.best_estimator_.score(X_test, Y_test)
svm_test_accuracy = svm_cv.best_estimator_.score(X_test, Y_test)
tree_test_accuracy = tree_cv.best_estimator_.score(X_test, Y_test)
knn_test_accuracy = knn_cv.best_estimator_.score(X_test, Y_test)
```

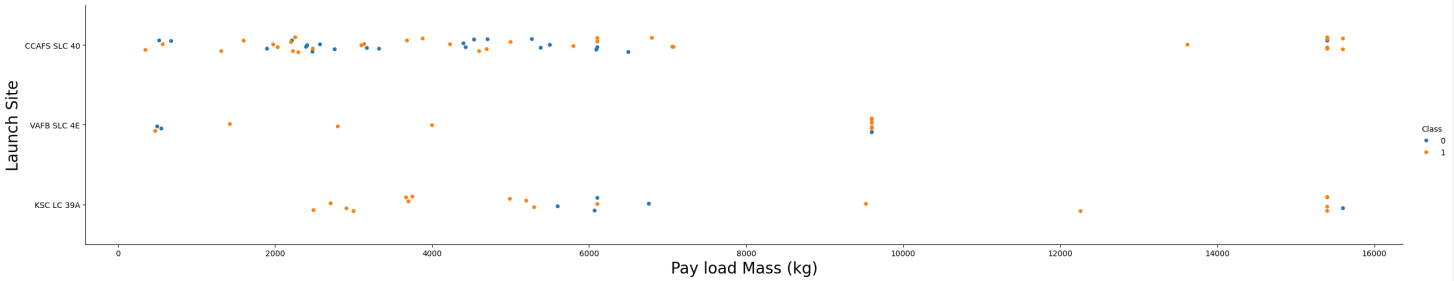
https://github.com/vishnu/IBMCapstone/blob/02566f3b8bcc12de348f79adb820e65c7cb01f/SpaceX-Machine-Learning-Prediction-Part-5-v1.ipynb

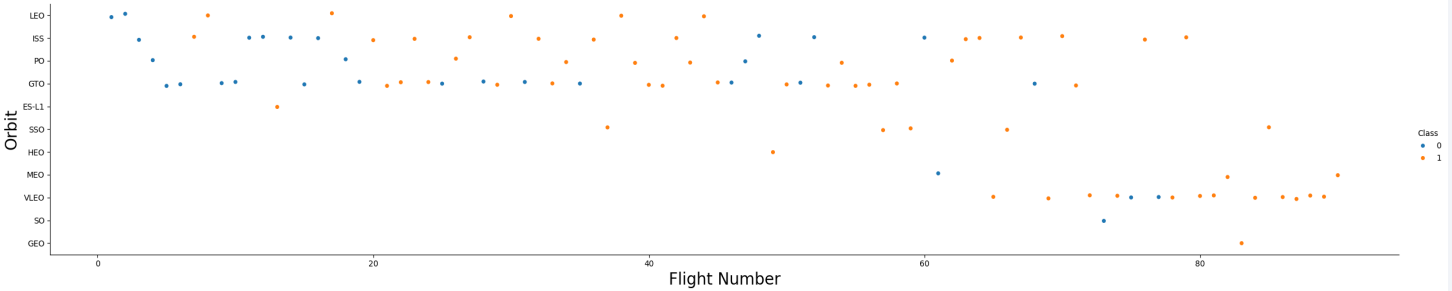


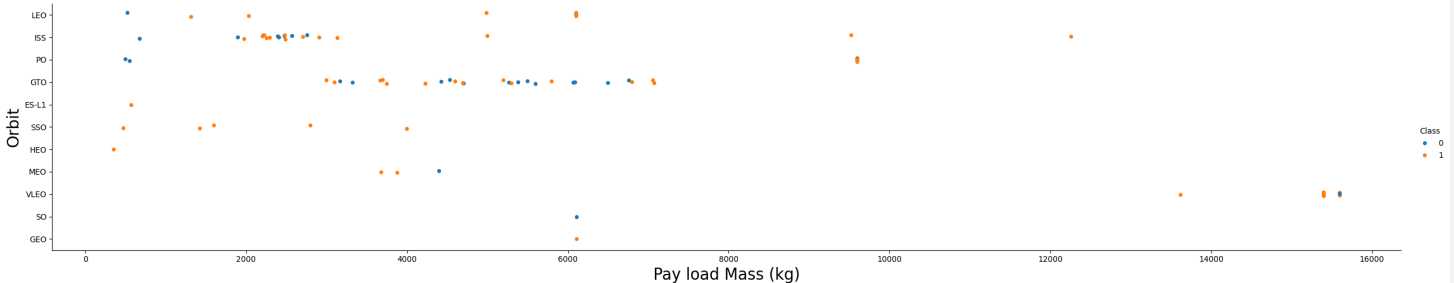


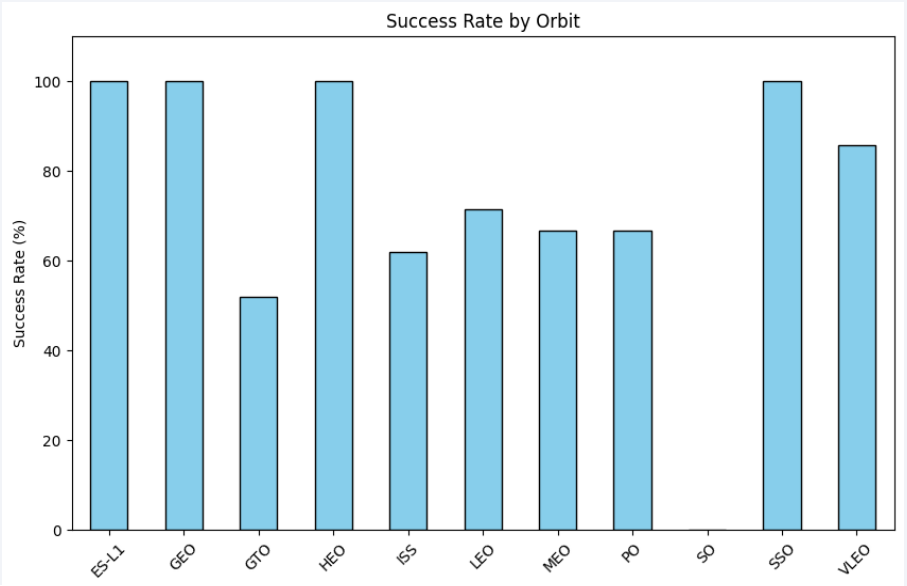
Results: Exploratory data analysis results

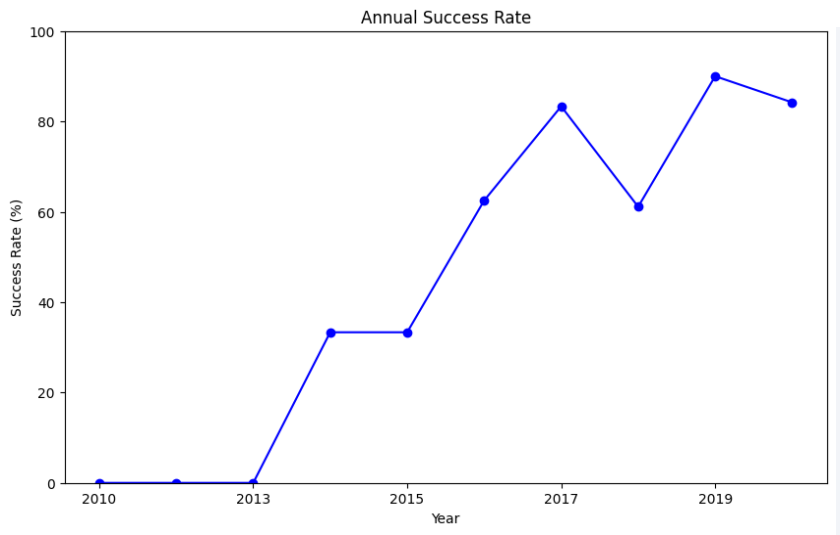




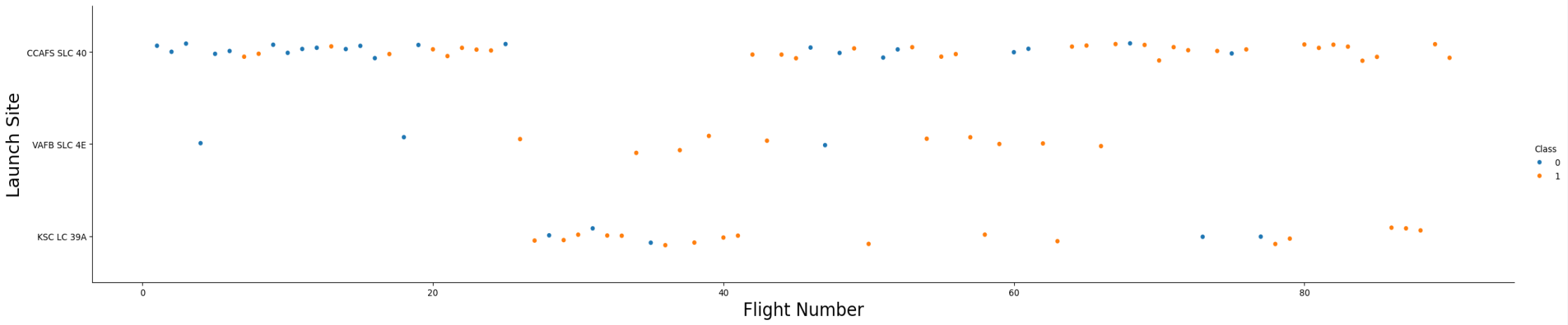


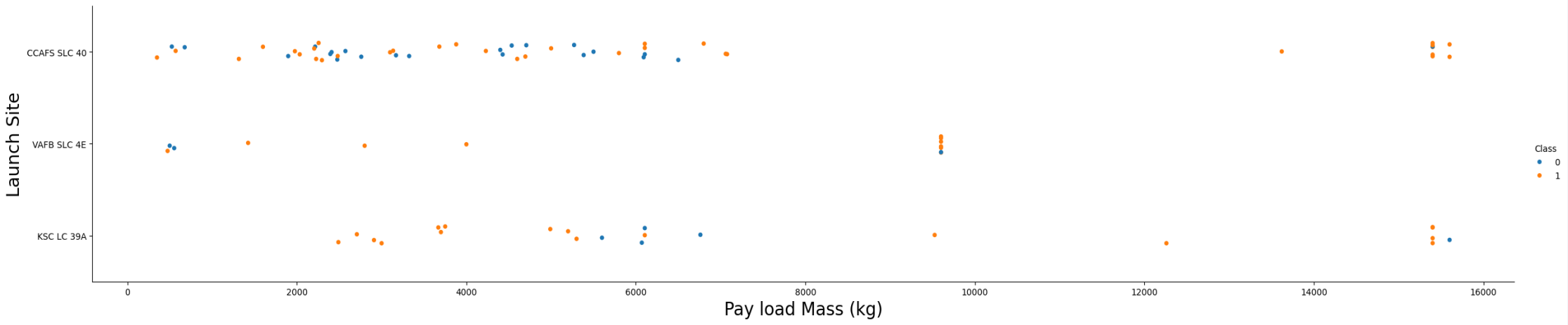




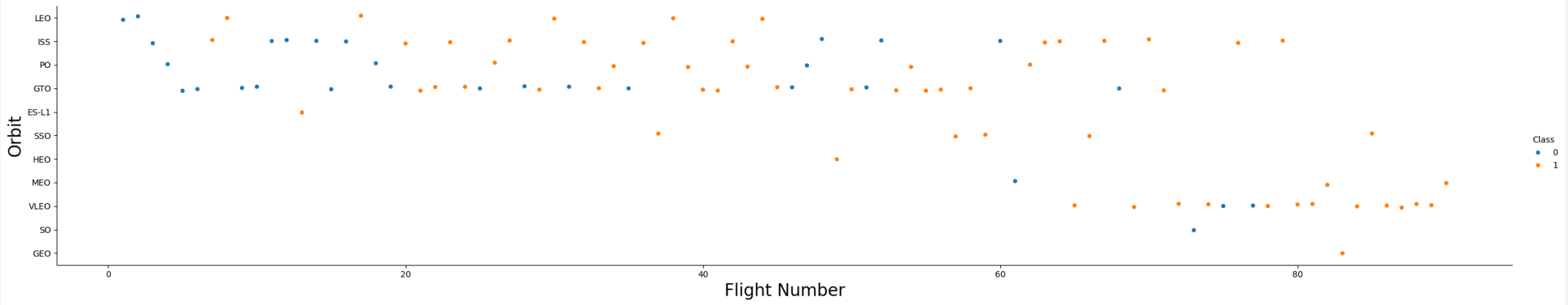


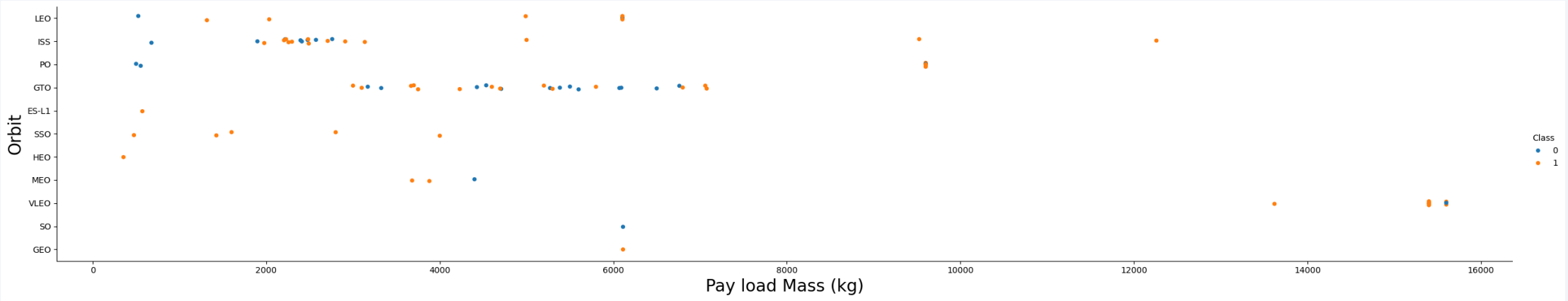
Results: Exploratory data analysis results



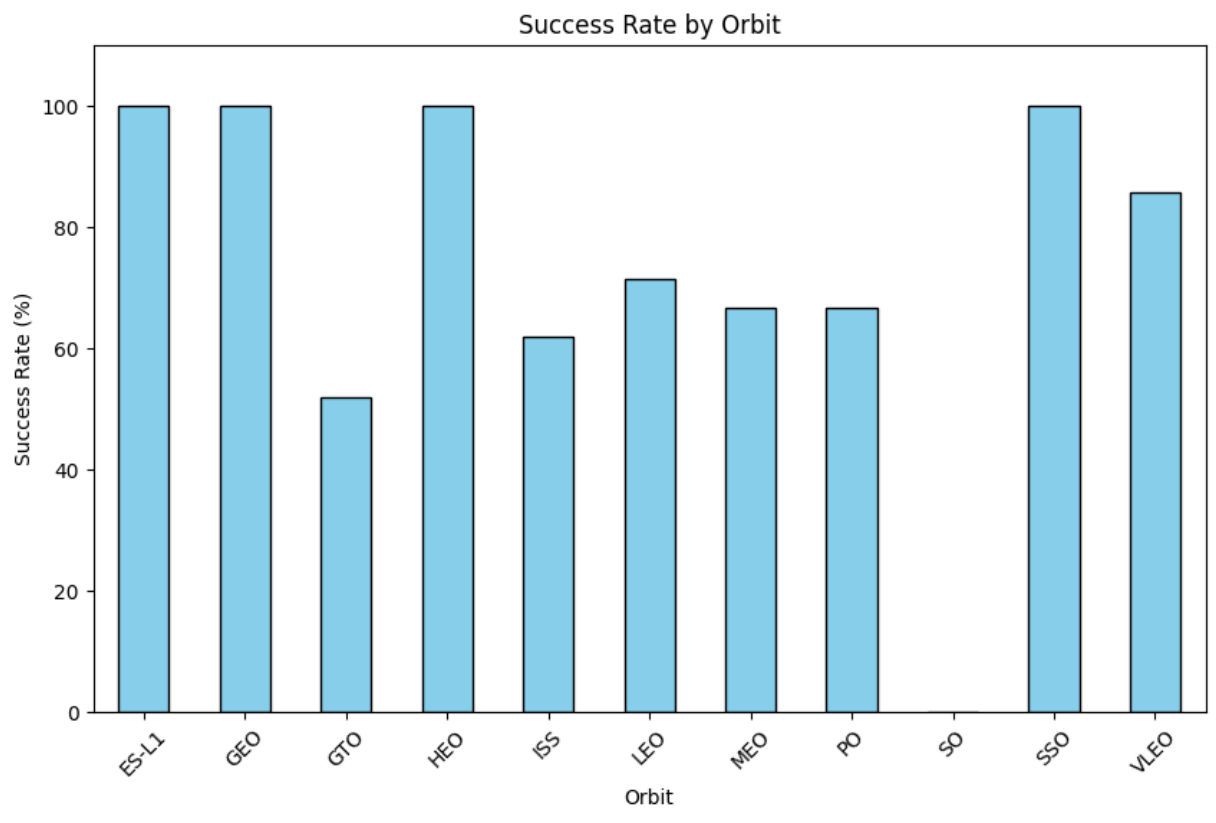


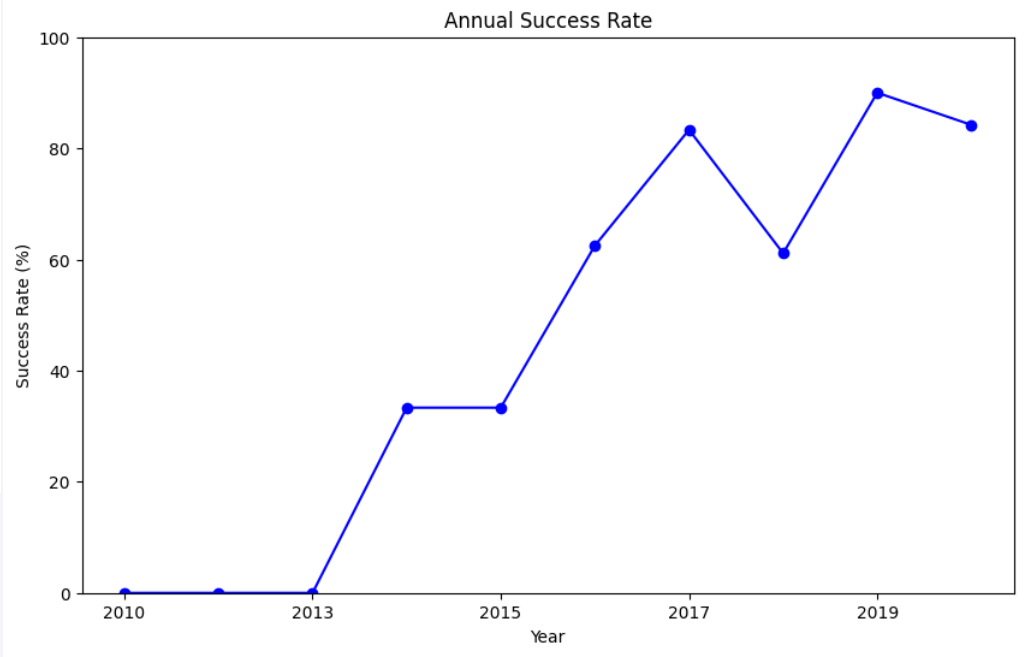
Results: Exploratory data analysis results



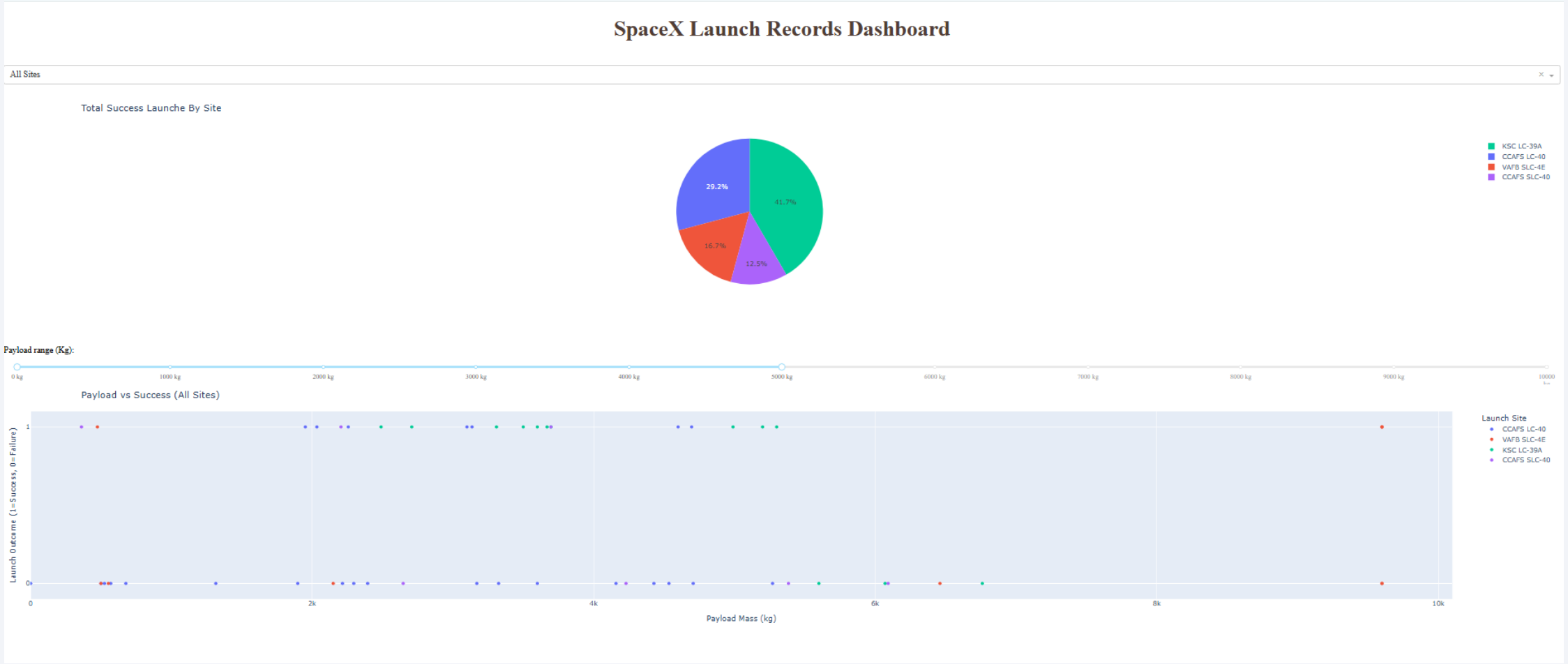


Results: Exploratory data analysis results

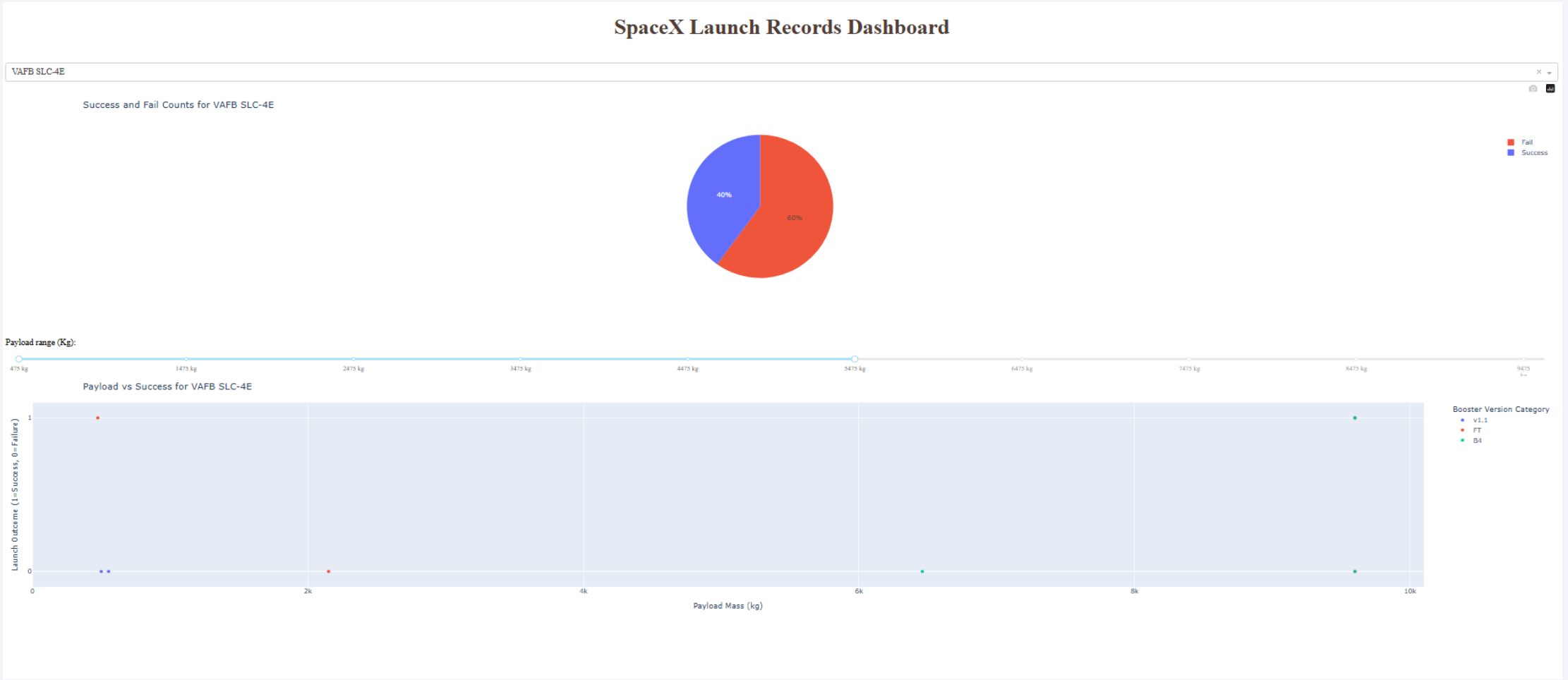




Results: Interactive analytics demo in screenshots



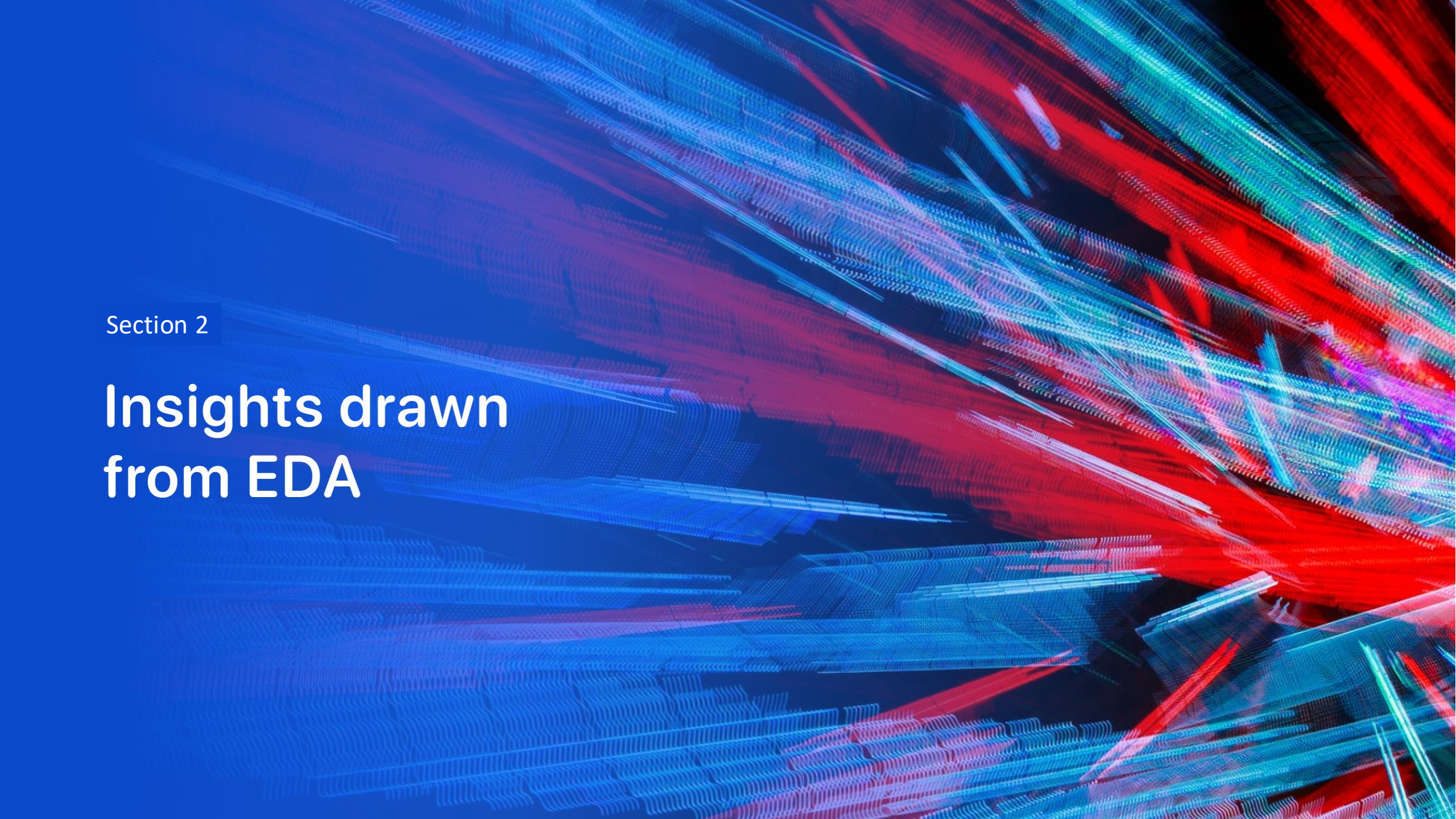
Results: Interactive analytics demo in screenshots



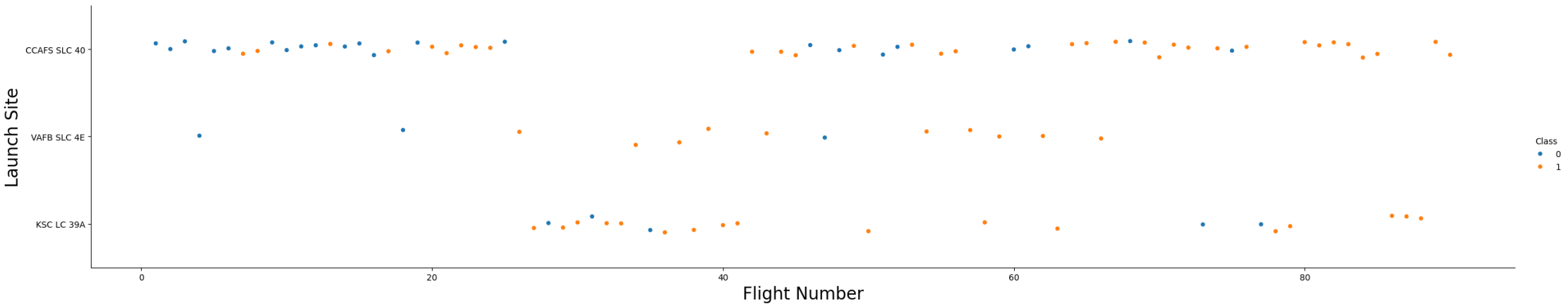
Results: Predictive analysis results

```python
logreg_test_accuracy = logreg_cv.best_estimator_.score(X_test, Y_test)
svm_test_accuracy = svm_cv.best_estimator_.score(X_test, Y_test)
tree_test_accuracy = tree_cv.best_estimator_.score(X_test, Y_test)
knn_test_accuracy = knn_cv.best_estimator_.score(X_test, Y_test)

# Mostramos
print("Logistic Regression Test Accuracy:", logreg_test_accuracy)
print("SVM Test Accuracy:", svm_test_accuracy)
print("Decision Tree Test Accuracy:", tree_test_accuracy)
print("KNN Test Accuracy:", knn_test_accuracy)

# Pero esto nos muestra el mejor (que tb vale a ojo)
best_model = max(
    [("Logistic Regression", logreg_test_accuracy),
     ("SVM", svm_test_accuracy),
     ("Decision Tree", tree_test_accuracy),
     ("KNN", knn_test_accuracy)],
    key=lambda x: x[1]
)

print(f"\nThe best performing model is {best_model[0]} with a test accuracy of {best_model[1]}")
```

```
Logistic Regression Test Accuracy: 0.8333333333333334
SVM Test Accuracy: 0.8333333333333334
Decision Tree Test Accuracy: 0.8333333333333334
KNN Test Accuracy: 0.8333333333333334

The best performing model is Logistic Regression with a test accuracy of 0.8333333333333334
```


Section 2
Insights drawn from EDA

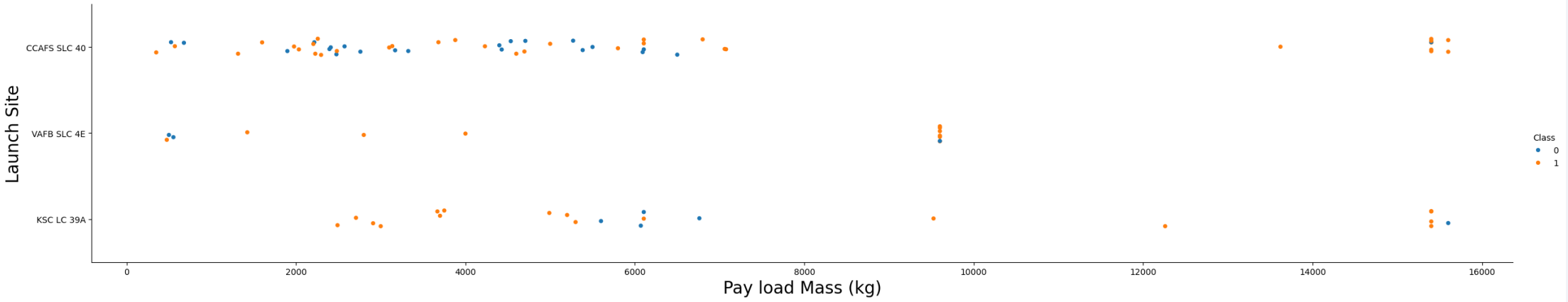
Flight Number vs. Launch Site



We can observe that KSC LC 39A has a higher success rate of 0.7727, compared to the next closest CCAFS SLC 40 with 0.7692.

However, it should be noted that CCAFS SLC 40 has a total of 33 successful launches, while X has only KSC LC 39A.

Payload vs. Launch Site



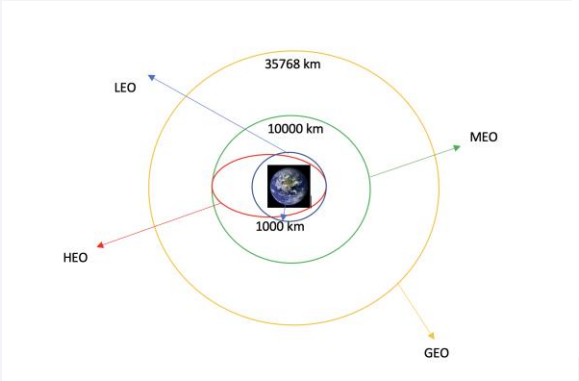
In this case seems simple, the higher the payload the better success rate.

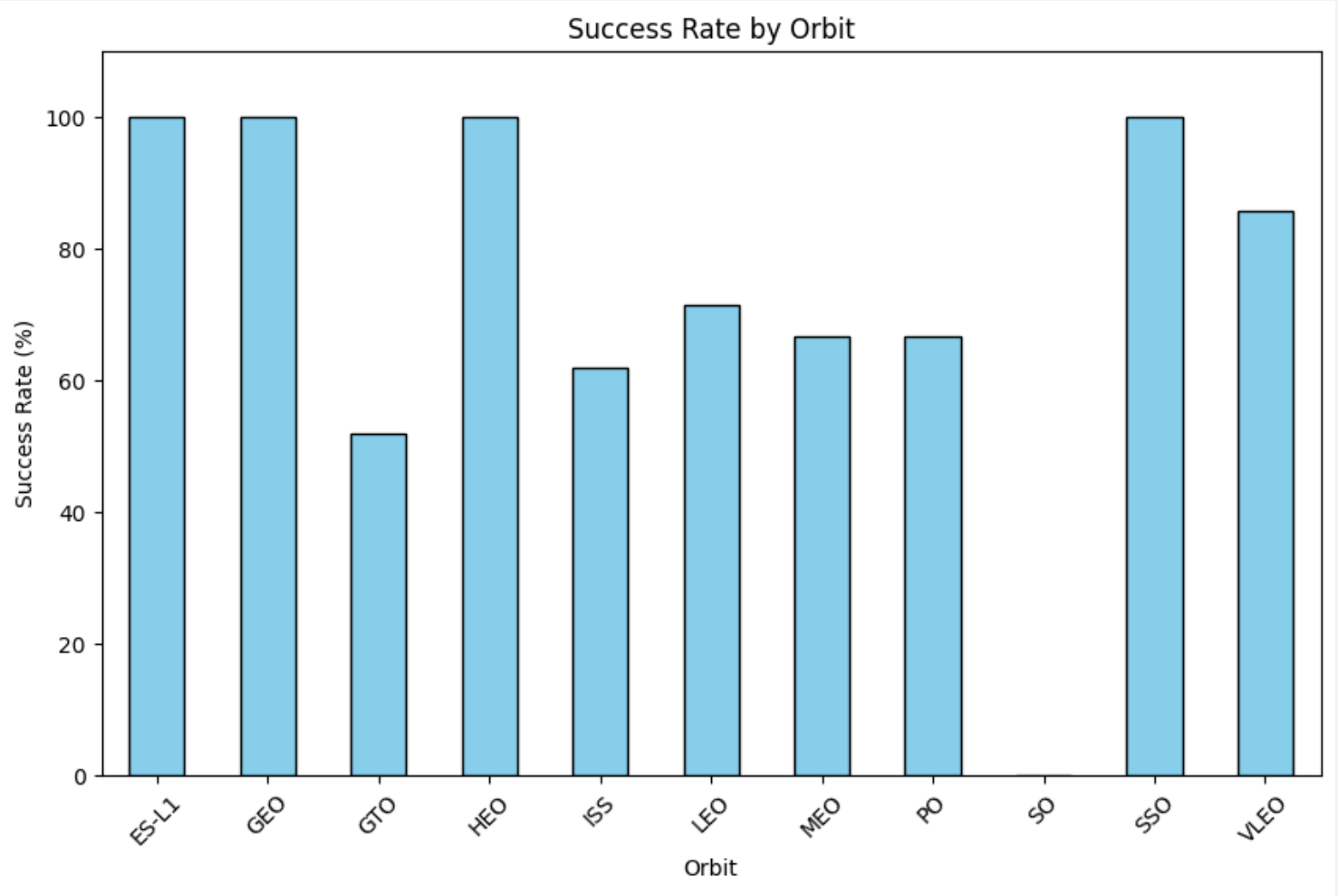
Some launch sites has payload limits, but within their capabilities works for all.

Success Rate vs. Orbit Type

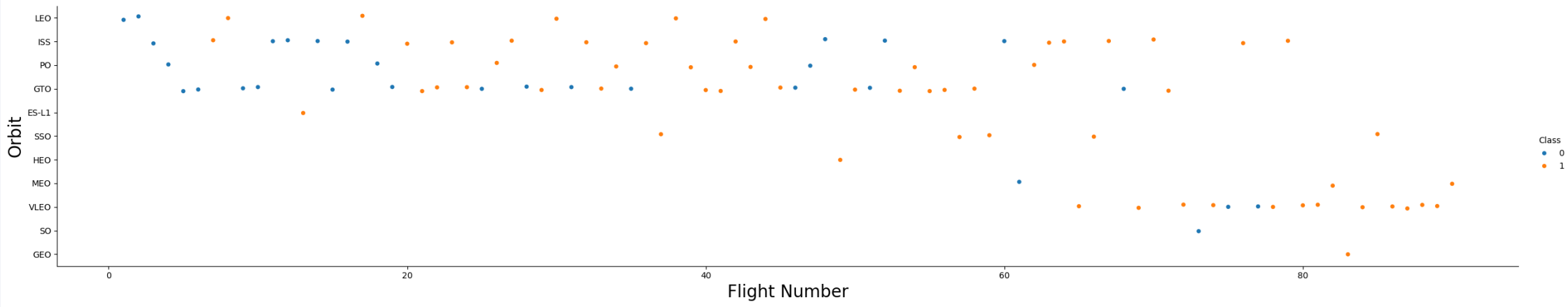
We have a perfect success rate of 100% for a group of 4 orbits.

- ES-L1
- GEO
- HEO
- SSO



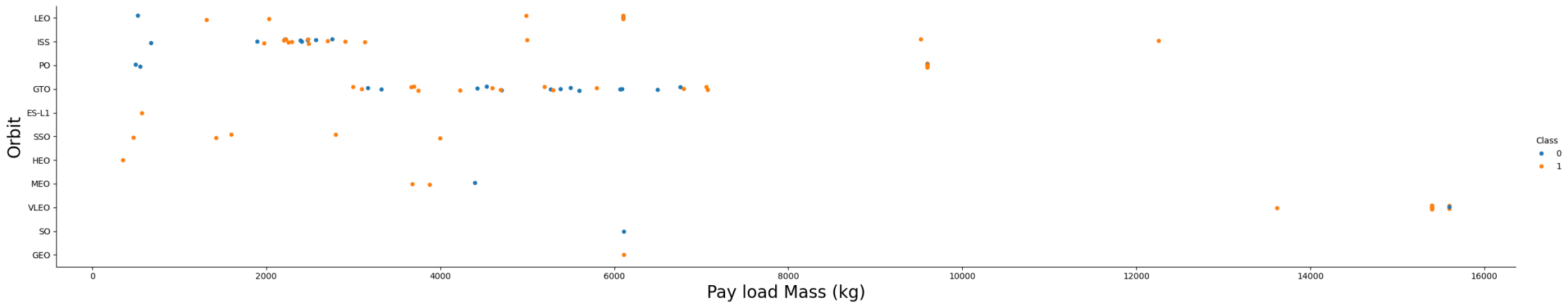


Flight Number vs. Orbit Type



As we could appreciate at lab you should see that in the LEO orbit the Success appears related to the number of flights; on the other hand, there seems to be no relationship between flight number when in GTO orbit.

Payload vs. Orbit Type

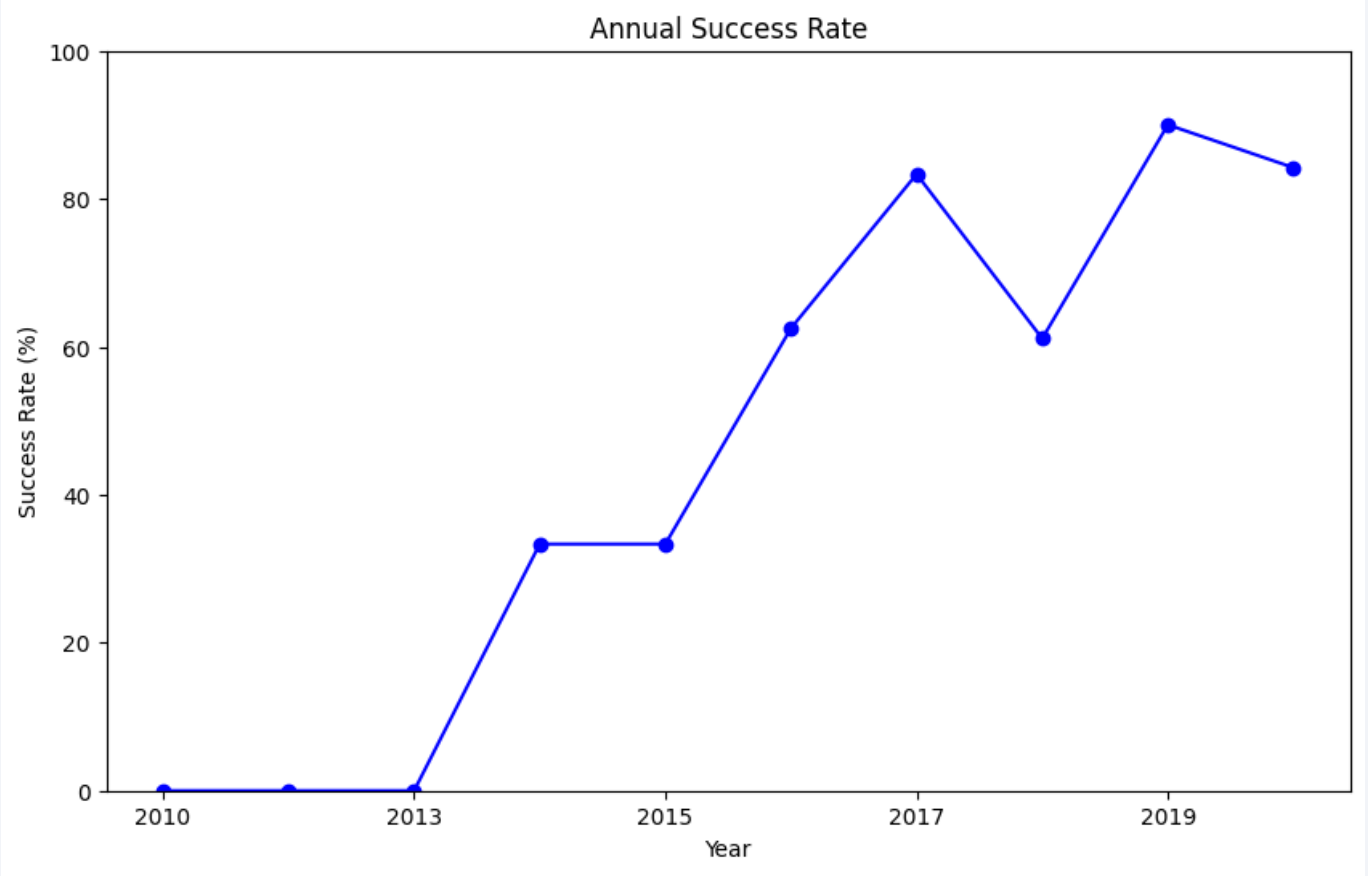


As we could observe at lab the observations we achieve are that with heavy payloads the successful landing or positive landing rate are more for Polar,LEO and ISS.

However for GTO we cannot distinguish this well as both positive landing rate and negative landing(unsuccessful mission) are both there here.

Launch Success Yearly Trend

We can appreciate that took from 2010 to 2013 was null, but after success stalling on 2014 it kept increasing steadilly with a minor problem at 2018.



All Launch Site Names

```
# Unique launch sites
cur.execute(f"SELECT DISTINCT Launch_Site FROM {table_name}")
unique_launch_sites = cur.fetchall()

# Display the records
for site in unique_launch_sites:
    print(site[0])
```

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

- We have all data into a table called SPACEXTABLE, keeping its reference into a variable called table_name, for this and all incoming queries.
- In this case we want the Launch_Site field, that contains the names, but we use distinct for avoiding repetitions, that is better than using a python loop.

```python
cur.execute(f"SELECT * FROM {table_name} WHERE Launch_Site LIKE 'CCA%' LIMIT 5")
records = cur.fetchall()
# tambien se pueden seleccionar todos y limitarlo en el bucle, pero es mas costoso y menos preciso

# Display the records
for record in records:
    print(record)
```

Python

```
('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')
```

Launch Site Names Begin with 'CCA'

- This time we narrow our search using LIKE and LIMIT, first one will filter within the string search inside the name, and LIMIT will help us avoiding use of resources because we will get only the 5 required records.

```
cur.execute(f"SELECT SUM(PAYLOAD_MASS__KG_) AS TOTAL_PAYLOAD FROM {table_name} WHERE Customer LIKE '%NASA (CRS)'")
record = cur.fetchone()
print(f'total payload mass carried by boosters launched by NASA (CRS): {record}')
```

```
total payload mass carried by boosters launched by NASA (CRS): (48213,)
```

Total Payload Mass

Just an aggregate function with the sum of the payload from the filtered customer NASA.

```
    cur.execute(f"SELECT AVG(PAYLOAD_MASS__KG_) AS AVG_PAYLOAD FROM {table_name} WHERE Booster_Version LIKE 'F9 v1.1%'")
    record = cur.fetchone()
    print(f'average payload mass carried by booster version F9 v1.1: {record}')

]
```

```
average payload mass carried by booster version F9 v1.1: (2534.6666666666665,)
```

Average Payload Mass by F9 v1.1

Another aggregate function, in this case the average mass of the filtered booster version.

```
cur.execute(f"""
        SELECT min(Date), "Time (UTC)"
        FROM {table_name}
        WHERE Landing_Outcome = 'Success (ground pad)'
        ORDER BY "Time (UTC)" ASC
        LIMIT 1
    """)
min_date_record = cur.fetchone()  # Usar fetchone() ya que solo esperamos un resultado

#if min_date_record and min_date_record[0] is not None and min_date_record[1] is not None:
if not (min_date_record[0] is None or min_date_record[1] is None):
    print(f'date when the first successful landing outcome in ground pad was achieved: {min_date_record[0]} at {min_date_record[1]}')
else:
    print("No records found or incorrect for Date or Time")
```

✓ 0.0s

```
date when the first successful landing outcome in ground pad was achieved: 2015-12-22 at 1:29:00
```

First Successful Ground Landing Date

- We use a compound sql now, filtering the minimum date from a previous result of selecting the successful groud landing, and ordering it, so this way we are sure we get the first one.

Successful Drone Ship Landing with Payload between 4000 and 6000

```
    cur.execute(f"""
    ····SELECT DISTINCT Booster_Version
    ····FROM {table_name}
    ····WHERE Landing_Outcome = 'Success (drone ship)'
    ····AND PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000
    """)
    boosters = cur.fetchall()

    # usamos un fetchall, por lo que es una lista
    if boosters:
        for booster in boosters:
            print(booster[0])
    else:
        print("No boosters found with the specified conditions.")
]
```

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

At this occasion we get the booster version from a previous query where we find out successful drone ship landings, but filtering with the BETWEEN the payload withing the stablished limits.

Total Number of Successful and Failure Mission Outcomes

```
cur.execute(f"""
    SELECT Mission_Outcome, COUNT(*) as outcome_count
    FROM {table_name}
    GROUP BY Mission_Outcome
""")

# Fetch all records
outcome_counts = cur.fetchall()

# Display the counts
if outcome_counts:
    for outcome in outcome_counts:
        print(f"Mission Outcome: {outcome[0]}, Count: {outcome[1]}")
else:
    print("No mission outcomes found.")
```

```
Mission Outcome: Failure (in flight), Count: 1
Mission Outcome: Success, Count: 98
Mission Outcome: Success , Count: 1
Mission Outcome: Success (payload status unclear), Count: 1
```

- With this code we are counting the total number of records that we can group depending on the mission outcome.
- First we group results depending on the outcome and then we sum them for getting a total for each group.

Boosters Carried Maximum Payload

Here we are filtering to get a unique value (using distinct) from the records that fit the search conditions, that we get from a nested query, queriying If the payload they had it's the same as the maximum available within our records.

```
cur.execute(f"""
    SELECT distinct Booster_Version
    FROM {table_name}
    WHERE PAYLOAD_MASS__KG_ = (
        SELECT MAX(PAYLOAD_MASS__KG_)
        FROM {table_name}
    )
""")
boosters_with_max_payload = cur.fetchall()
# Display
if boosters_with_max_payload:
    for booster in boosters_with_max_payload:
        print(booster[0])
else:
    print("No boosters found with the maximum payload mass.")
```

✓ 0.0s

```
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
```

```
cur.execute(f"""
    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_Name, Booster_Version, Launch_Site
    FROM {table_name}
    WHERE Landing_Outcome = 'Failure (drone ship)' AND substr(Date, 0, 5) = '2015' """)
# Fetch all records
failure_records_2015 = cur.fetchall()
# Display the records
if failure_records_2015:
    for record in failure_records_2015:
        print(f"Month: {record[0]}, Booster Version: {record[1]}, Launch Site: {record[2]}")
else:
    print("No records found for failure landing outcomes in 2015.")
```

```
Month: January, Booster Version: F9 v1.1 B1012, Launch Site: CCAFS LC-40
Month: April, Booster Version: F9 v1.1 B1015, Launch Site: CCAFS LC-40
```

2015 Launch Records

- Problem here is that we must filter data that is numerical and then write all the possible outcomes we can find, it' s our task not leaving out any possibility.

```
    cur.execute(f"""
        SELECT Landing_Outcome, COUNT(*) as outcome_count
        FROM {table_name}
        WHERE Date BETWEEN '2010-06-04' AND '2017-03-20'
        GROUP BY Landing_Outcome
        ORDER BY outcome_count DESC
    """)
    landing_outcome_counts = cur.fetchall()

    if landing_outcome_counts:
        for outcome in landing_outcome_counts:
            print(f"Landing Outcome: {outcome[0]}, Count: {outcome[1]}")
    else:
        print("No landing outcomes found in the specified date range.")
```

```
Landing Outcome: No attempt, Count: 10
Landing Outcome: Success (drone ship), Count: 5
Landing Outcome: Failure (drone ship), Count: 5
Landing Outcome: Success (ground pad), Count: 3
Landing Outcome: Controlled (ocean), Count: 3
Landing Outcome: Uncontrolled (ocean), Count: 2
Landing Outcome: Failure (parachute), Count: 2
Landing Outcome: Precluded (drone ship), Count: 1
```

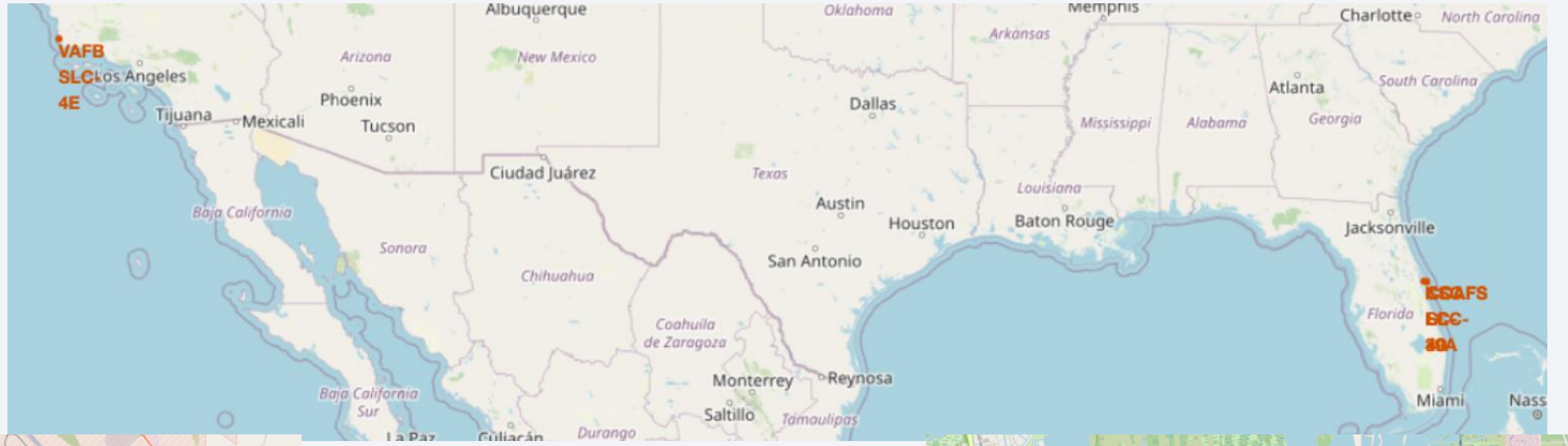
Rank Landing Outcomes Between 2010-06-04 and 2017-03-20

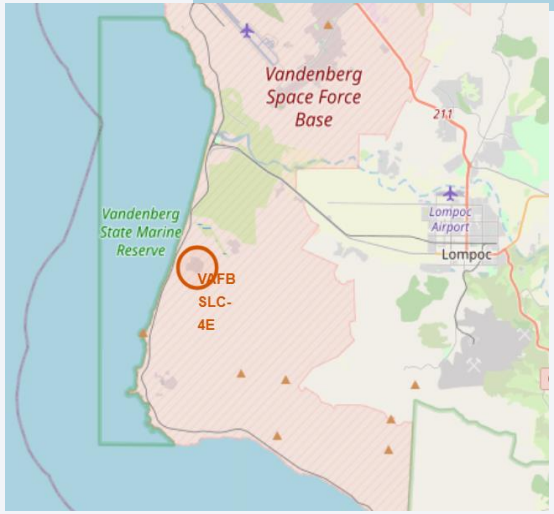
- Just remember the right order when you set the options for the sql, must group by before trying to order anything.

Section 3

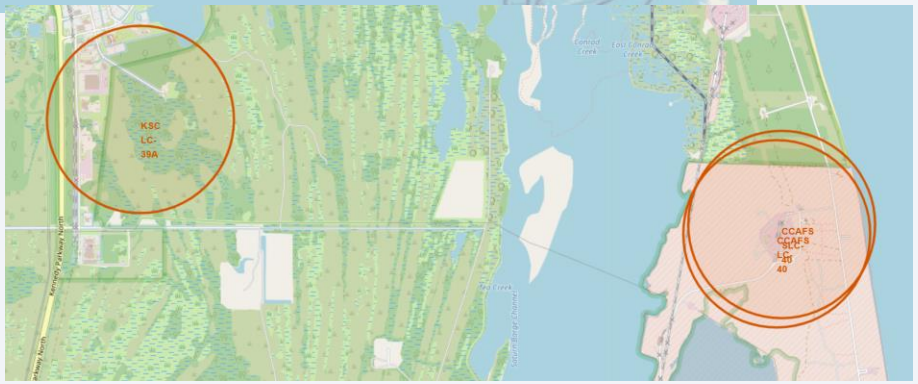
Launch Sites Proximities Analysis

Global Launch Sites Folium Map peek view





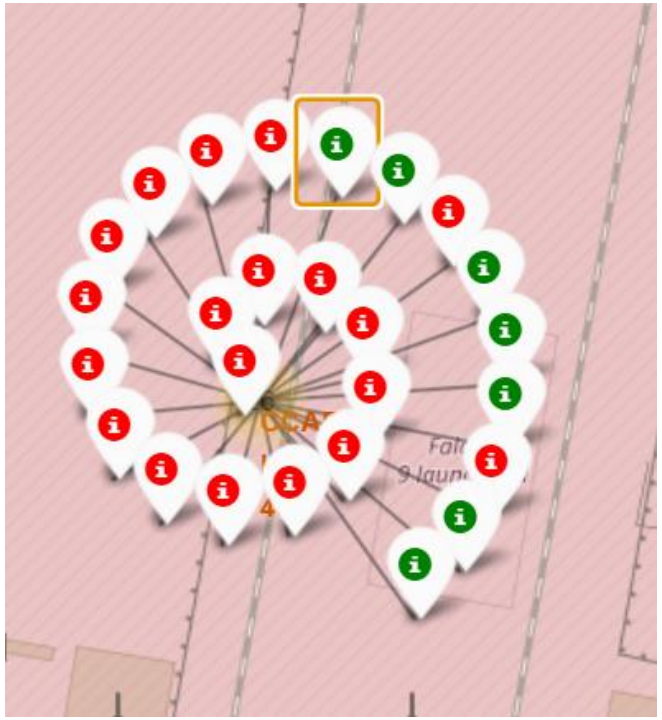
Global view allows us to locate the aprox. Situation of launch sites, but too far for proper inspection.

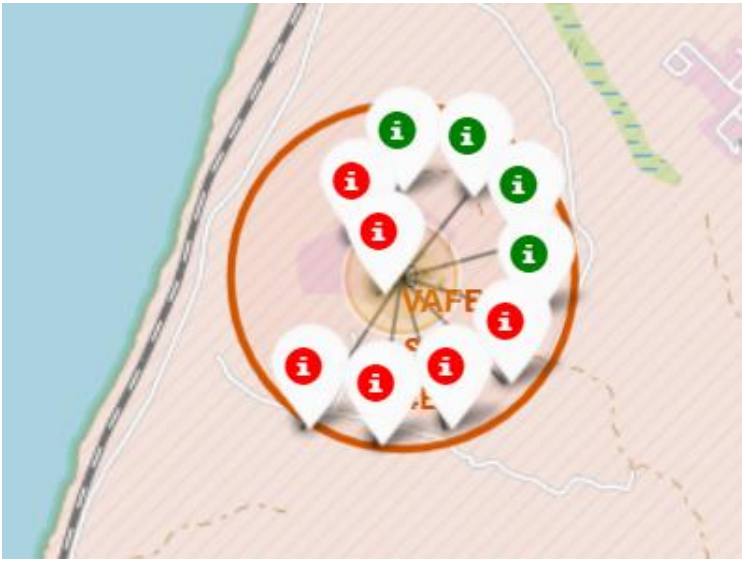


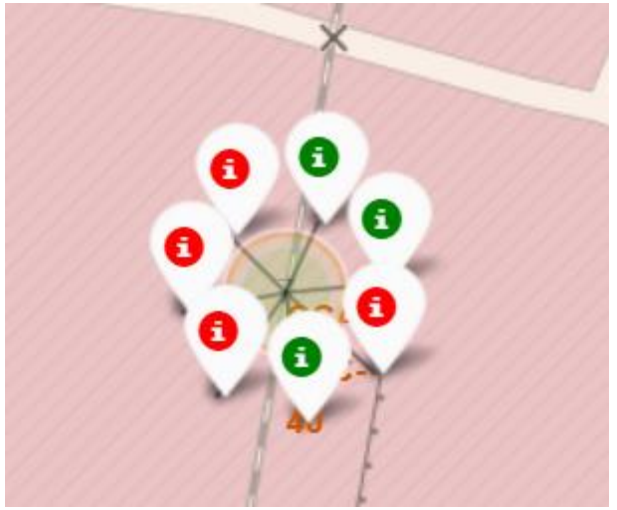
We can observe how a folium circe circles the situation and a marker lets us identify each one of the launch sites.

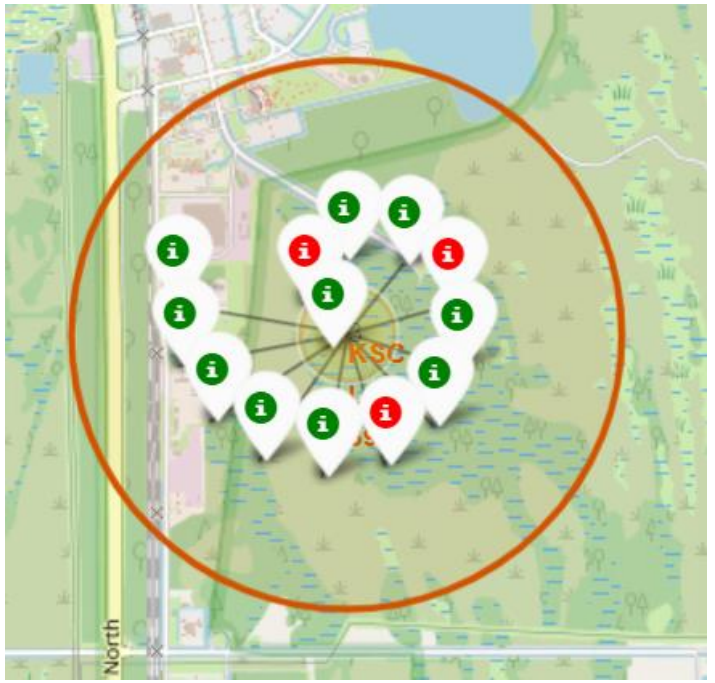
Spiraling markers with launch results

- Adding a marker for each launch, we set red for fail and green for successful launches.
- This markers are set around the previous one setting name for the site, once the number is to high for keeping a proper circle it spirals around the site.





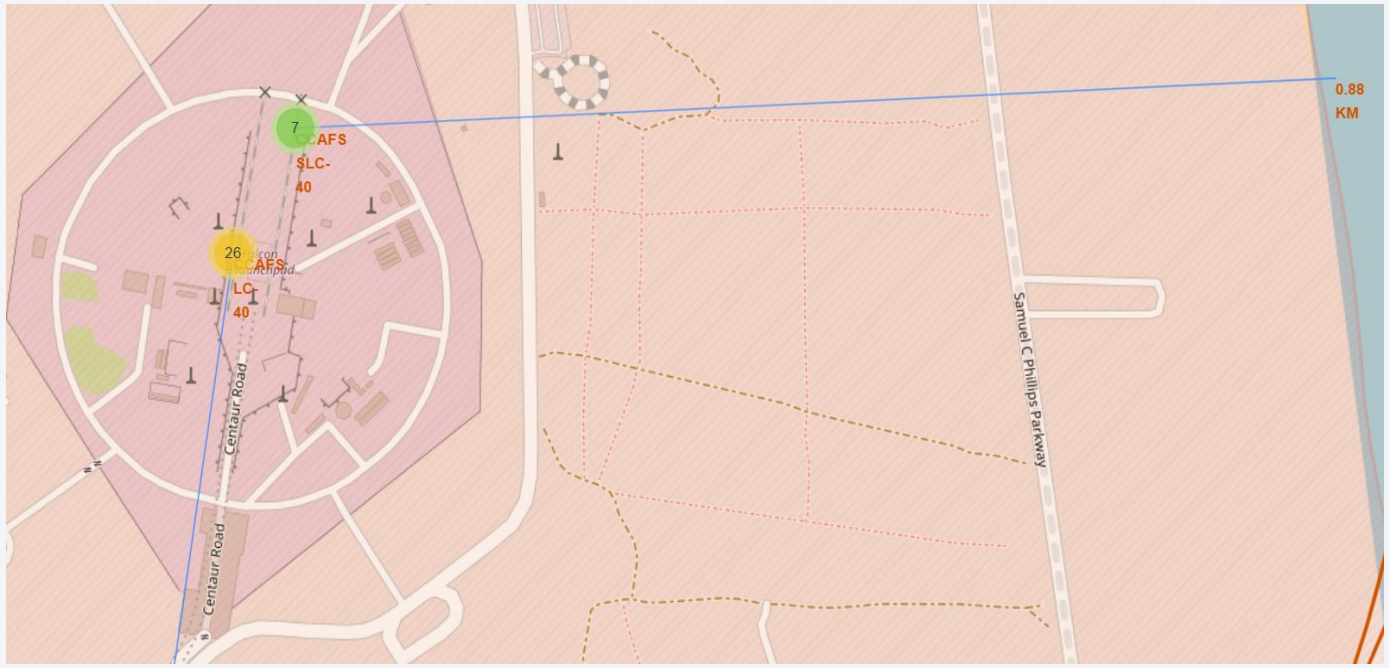




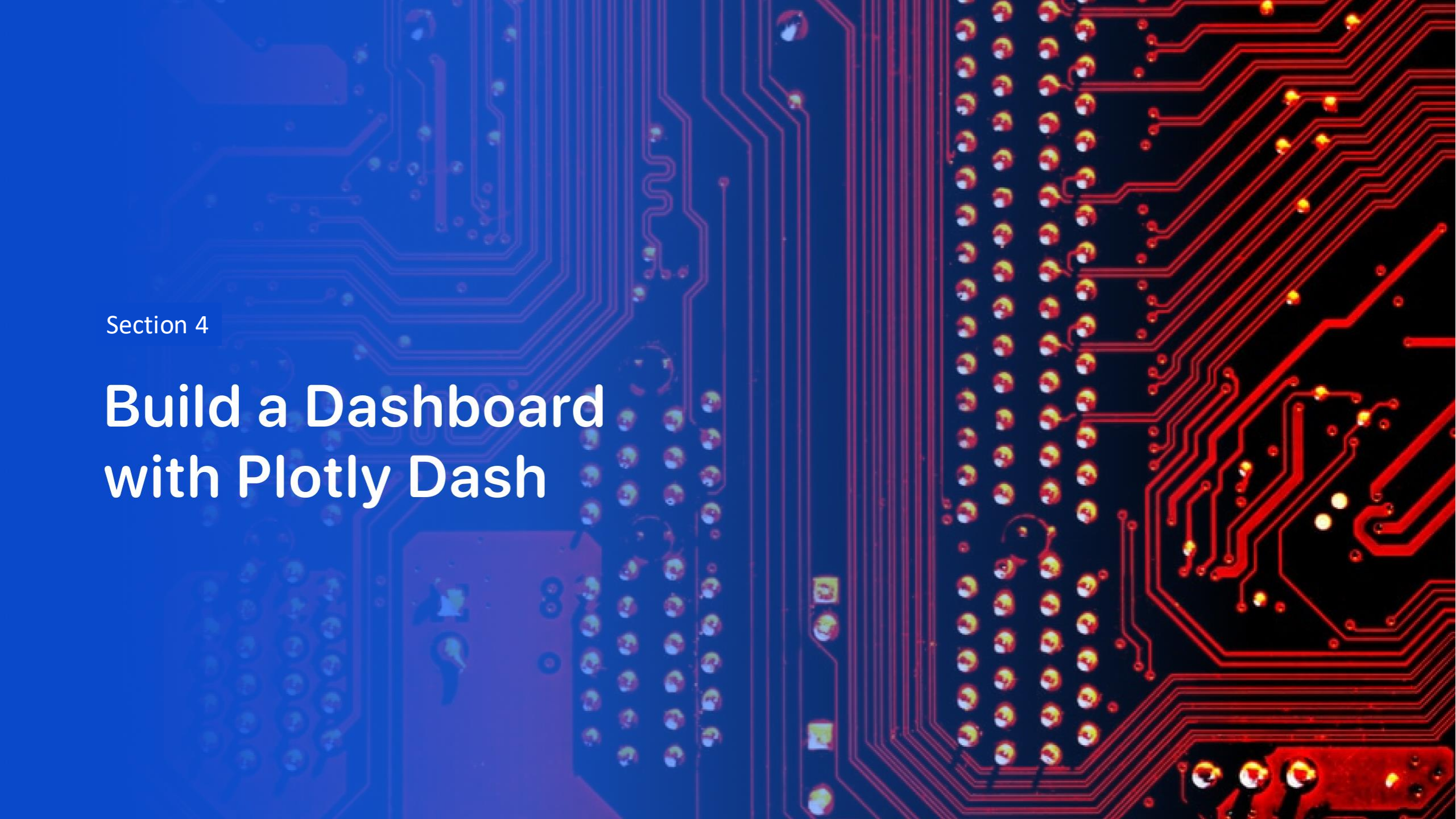
Boundaries inspection with Folium

With distance calculation and marking relevant spots we can set insights about the needs or the avoids from launch sites

This way we can learn how far do we have available resources or calculate the possible dangers to a town or any suitable place.



An easy find is that launch sites prefer to be near the sea (just for security?) and as far as they can from towns.
The most reusable they become the less need they have to be near railways for heavy pieces.

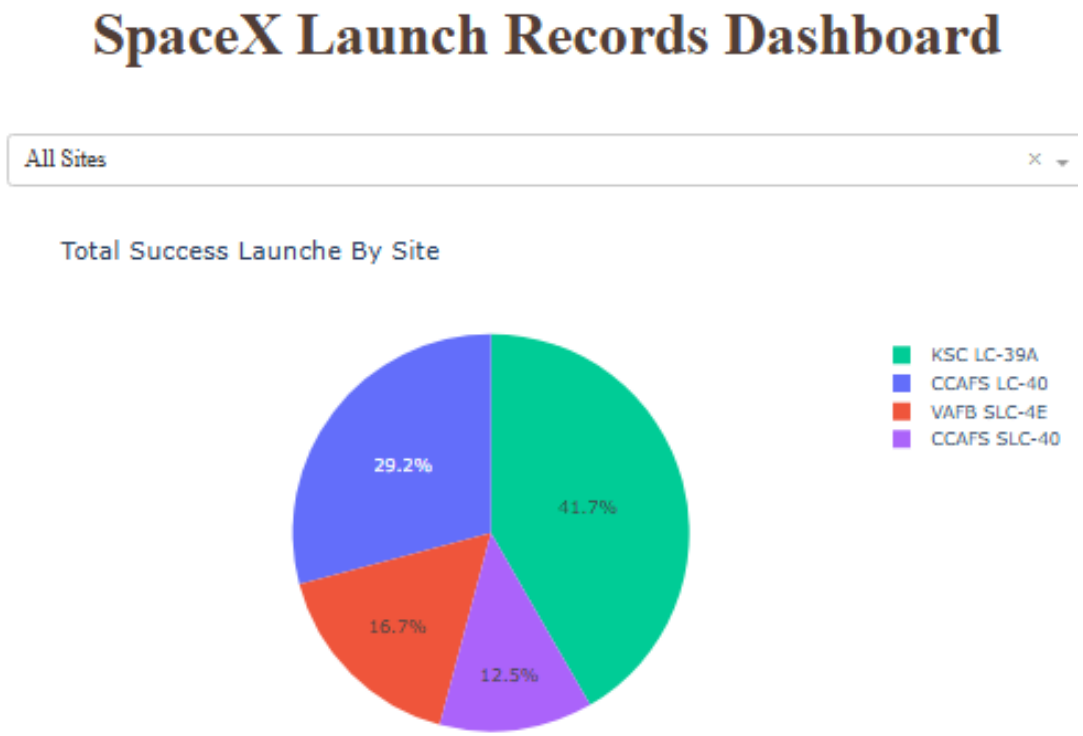

Section 4
Build a Dashboard with Plotly Dash

Dashboard. Pie chart.

Default option, with all sites.

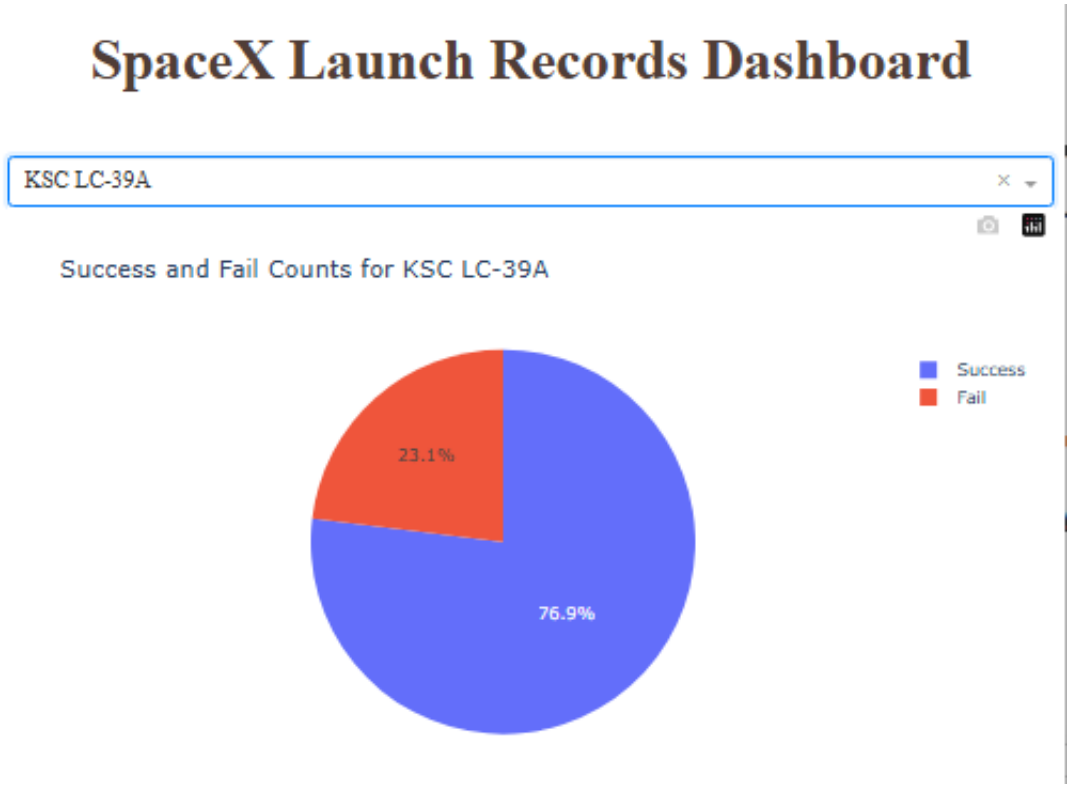
- All elements are important, the pie chart and the selector.
    - Pie chart show the results.
    - Selector allows us to pick the info we want to see.

Easy finding is that KSC LC-39A is the launch site with more success while CCAFS SLC-40 has the worst results.



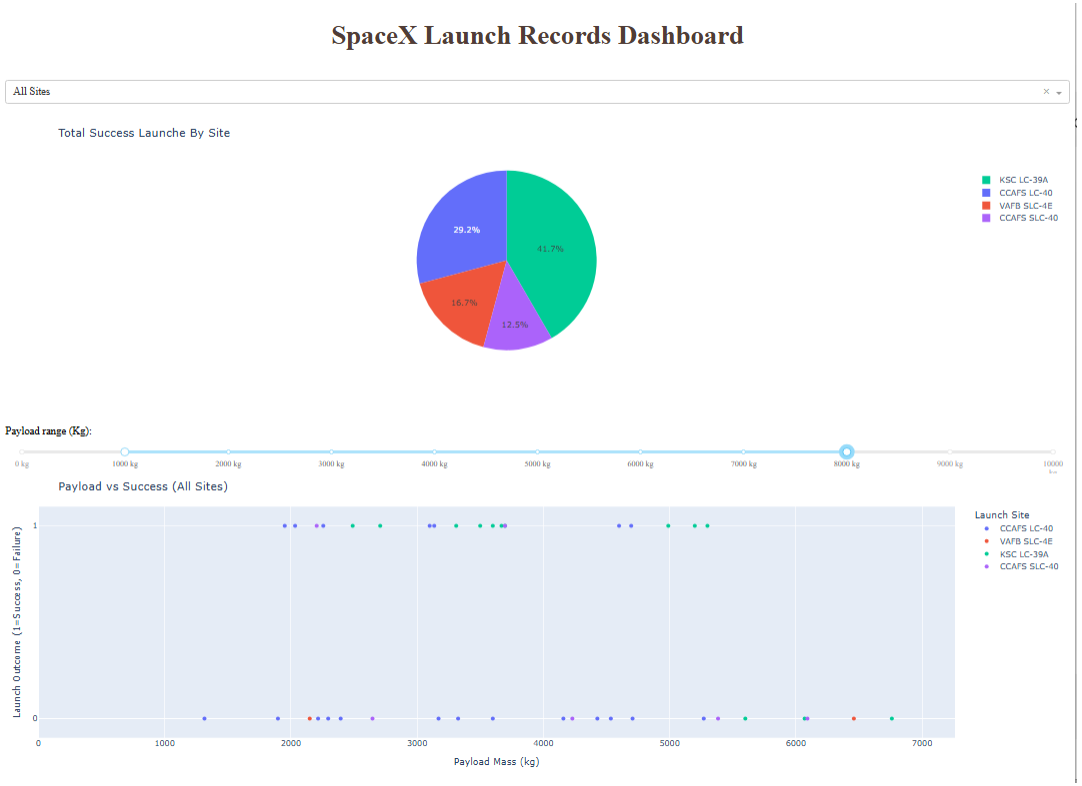
Dashboard. Best pie.

- We keep with the same elements, just changed the pie chart, when we select ALL sites we can observe the full-sites results.
- When we choose a single launch site we can see its real performance, showing us the % of failed and successful launches.
- With the best results launch site we can observe that it has a 76.9% success ratio.



Dashboard. Payload Selector.

- Now the new payload range selector gets some more importance, because it allow us to filter a bit more.
- We can observe that the higher payload the best results we get.
- Some insights come that the B5 booster has no failures and best success with a 58.8% comes into 2000-4000 Kgs payload range.

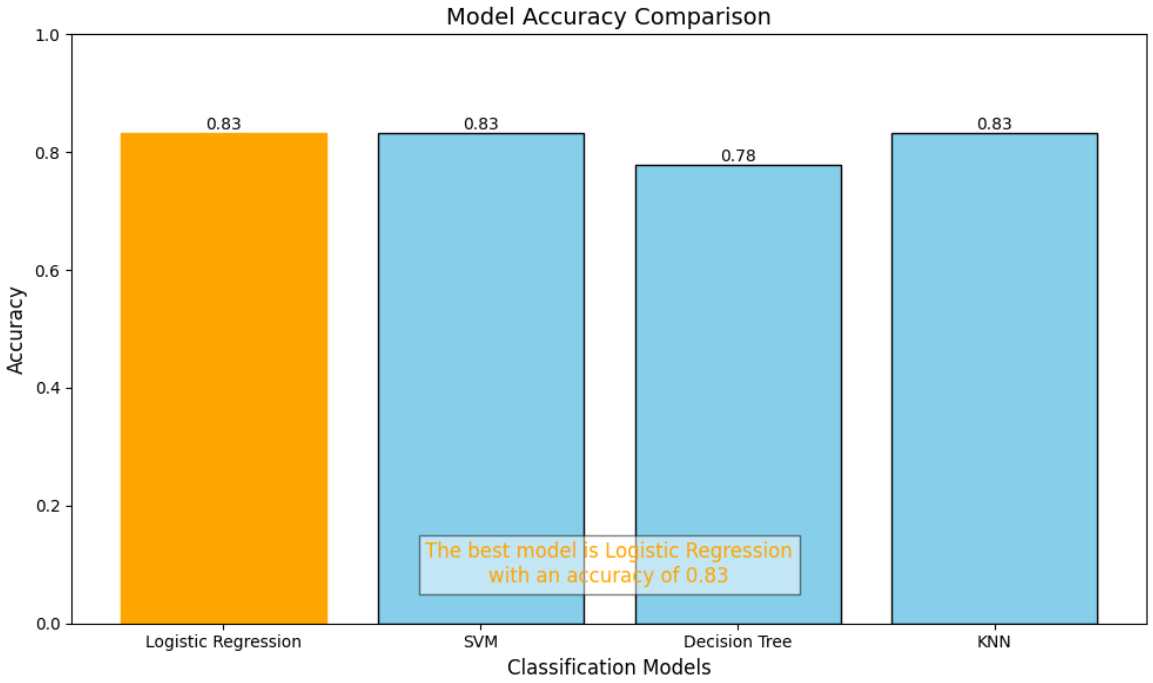


Section 5

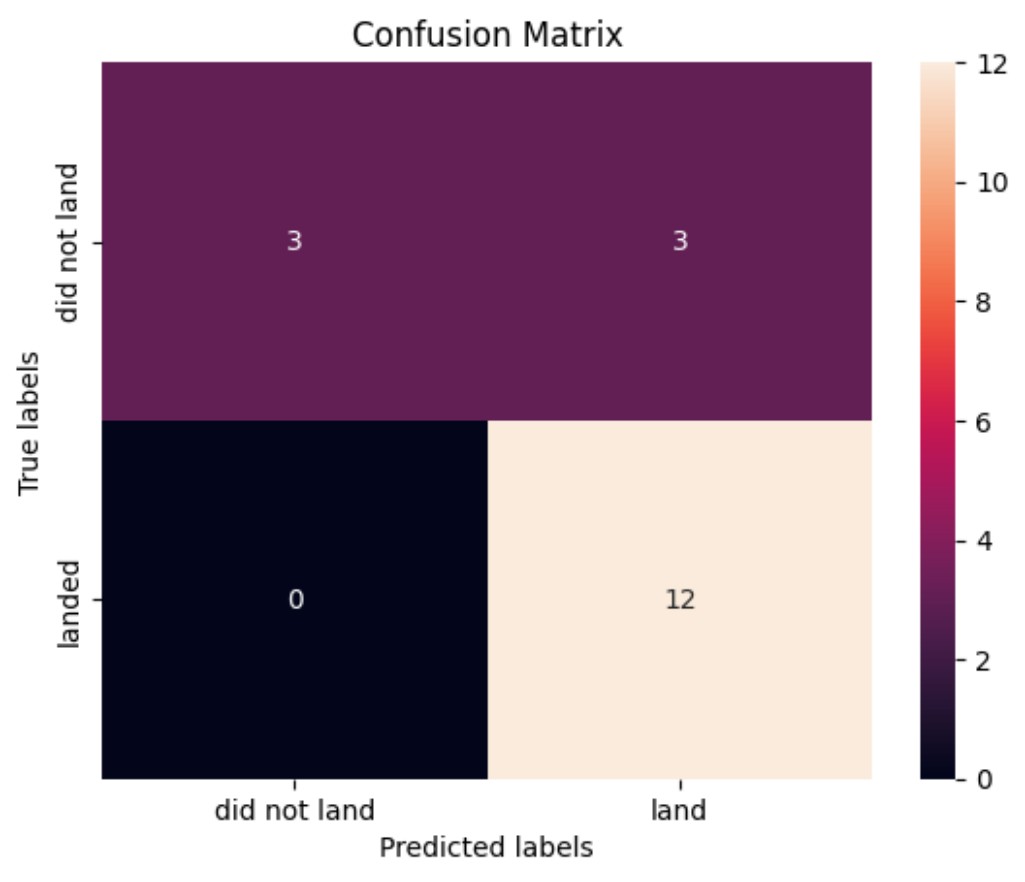
Predictive Analysis (Classification)

Algorithm presents logistic regression as the best one, but its results are same as SVM and KNN.

Classification Accuracy



Confusion Matrix

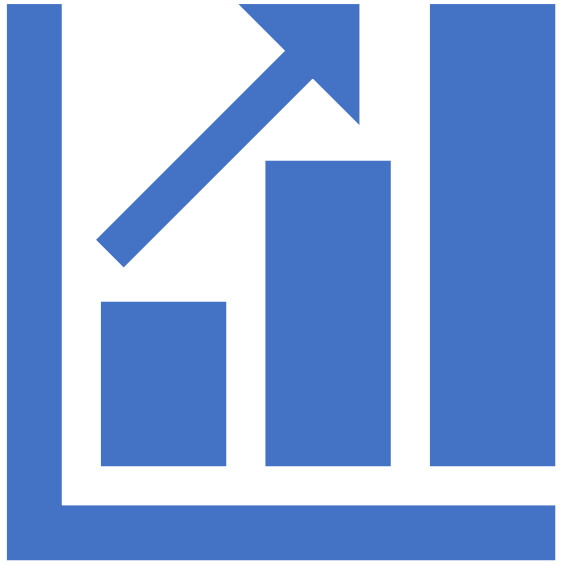


We can find out the the performance getting True positives is perfect, but we can see that NN and NT are same same.

So we can relay in finding real success, but will fail when trying to find out possible fails.

Conclusions

- Yearly trend of increase in success rate from 2013 onwards
- Max. success counts from CCAFS LC-40
- Two records in 2015, both mission outcomes successful whereas landing outcomes failure
- Date of the first successful landing outcome on ground pad 22.12.2015
- Four numbers of F9 FT BXXXX series of boosters have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000
- No failure of B5
- B4 lifted 9600kg successfully



Appendix

Just a python snipet for the range selector of payload, instead of allowing full range it sets max and min depending on the values that launch site has managed before.

Of course must add some variations to the corresponding code at the dashboard.

```python
# TASK 3.5
# Update the slider range based on the selected site
@app.callback(
    Output(component_id='payload-slider', component_property='marks'),
    Output(component_id='payload-slider', component_property='min'),
    Output(component_id='payload-slider', component_property='max'),
    Input(component_id='site-dropdown', component_property='value')
)
def update_slider_range(selected_site):
    if selected_site == 'ALL':
        # Si seleccionamos "ALL", usamos los valores min y max globales
        min_payload = int(spacex_df['Payload Mass (kg)'].min())  # Convertir a int
        # max_payload = int(spacex_df['Payload Mass (kg)'].max())  # Convertir a int
        max_payload = 10000
        # Definimos marcas cada 1000 kg
        marks = {i: f'{i} kg' for i in range(min_payload, max_payload + 1, 1000)}
    else:
        # Si se selecciona un sitio específico, filtramos por ese sitio
        site_df = spacex_df[spacex_df['Launch Site'] == selected_site]
        min_payload = int(site_df['Payload Mass (kg)'].min())  # Convertir a int
        max_payload = int(site_df['Payload Mass (kg)'].max())  # Convertir a int
        # Definimos marcas cada 1000 kg
        marks = {i: f'{i} kg' for i in range(min_payload, max_payload + 1, 1000)}

    return marks, min_payload, max_payload
```

Thank you!