SpaceX: IBM Applied Data Science Capstone

Shabish Chithradewa

18-07-2024

https://github.com/shabishc/IBM-SpaceX-Applied-DataScience-Capstone.git

IBM Developer
SKILLS NETWORK

Outline

Executive Summary

Summary of Methodologies

- Data Collection through API and Web Scraping
- Data Wrangling
- Exploratory Data Analysis with SQL and Data Visualization
- Interactive Visual Analytics with Folium and Plotly Dash
- Machine Learning Predictive Analysis (Classification - LR, SVM, DT, KNN)

Results

- Exploratory Data Analysis results
- Interactive analytics / Visualization
- Predictive Analysis results

Introduction

Background

SpaceX, a leader in the space industry, is dedicated to making space travel affordable for everyone. A significant factor in SpaceX's ability to achieve these feats cost-effectively is their innovative reuse of the first stage of the Falcon 9 rocket. This reuse allows SpaceX to offer rocket launches at a relatively low cost of $62 million per launch, unlike other providers whose non-reusable rockets lead to costs exceeding $165 million per launch. The primary goal of this project is to develop a machine learning pipeline capable of predicting the successful landing of the Falcon 9 first stage. Accurate predictions will enable stakeholders to estimate launch costs effectively, facilitating competitive bids against SpaceX. Therefore, accurately predicting the success of the first landing stage can directly impact the cost estimation of a launch. This prediction is particularly valuable for alternative companies competing with SpaceX for rocket launch contracts.

Methodology

1

Data Collection

Data was collected from SpaceX's official API, providing detailed information about past launches, and web scraping from Wikipedia, supplementing data on specific missions.

2

Data Wrangling

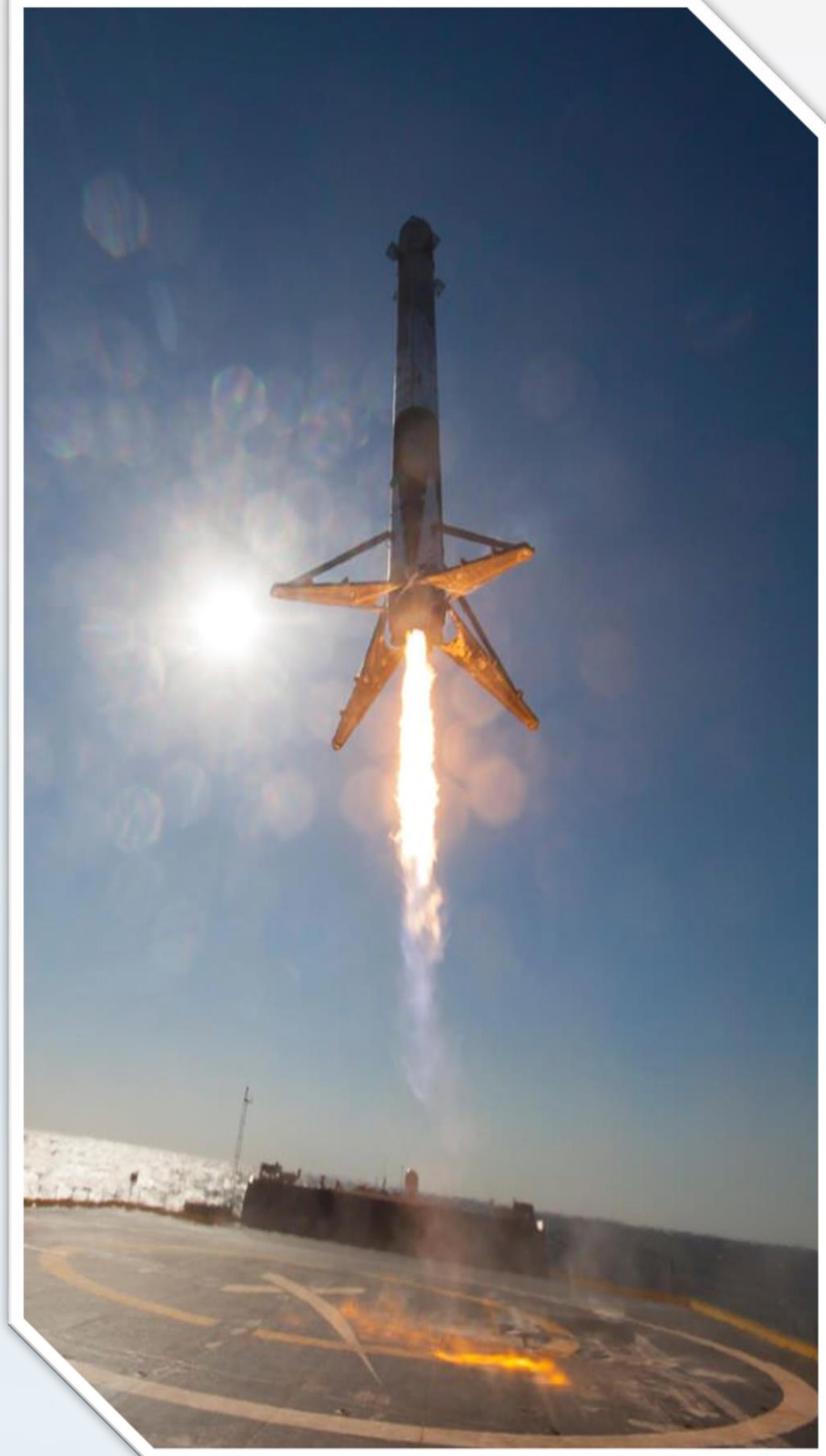
Data was cleaned, missing values were addressed, and categorical features were encoded (one-hot encoding) for use in machine learning models.

3

EDA with SQL & Data Visualization

Data visualized by plotting scatter and bar graphs to understand the relationships between flight number, launch site, payload, and success rate, analyzing yearly trends in launch success rates, and utilizing SQL queries to extract data on unique launch sites, payload mass statistics, and mission outcomes.

IBM Developer SKILLS NETWORK



Methodology Cont.

4

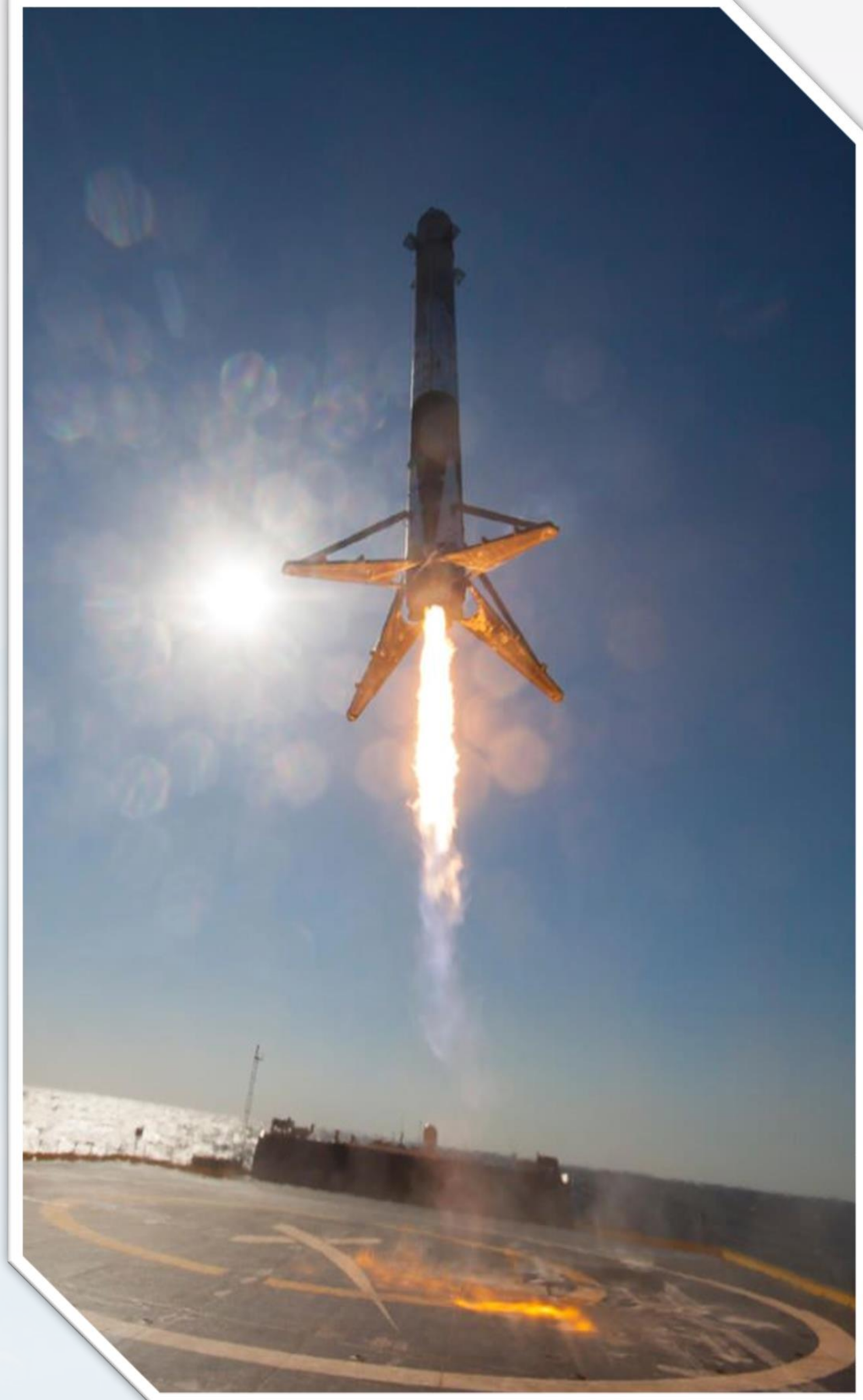
Interactive Visual Analytics & Dashboards

Interactive visual analytics performed using Folium for map-based analysis and Plotly Dash for interactive dashboards, providing insights into success rates by launch sites and the relationship between payload mass and launch outcomes.

5

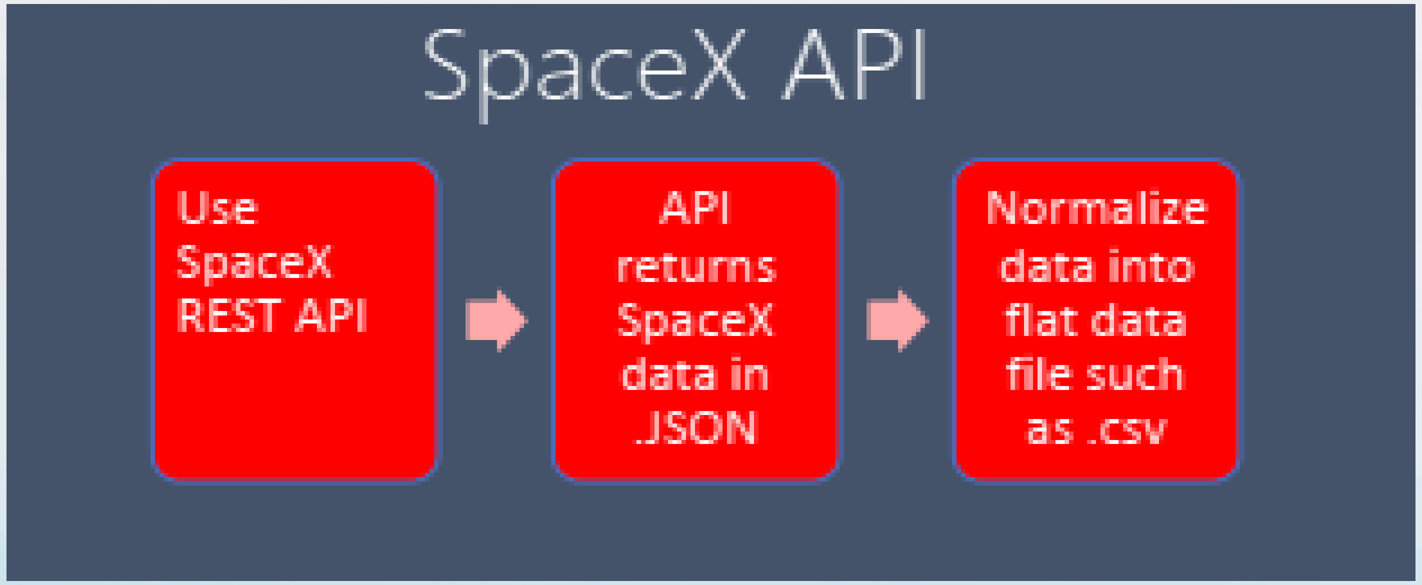
Predictive Analysis (Classification)

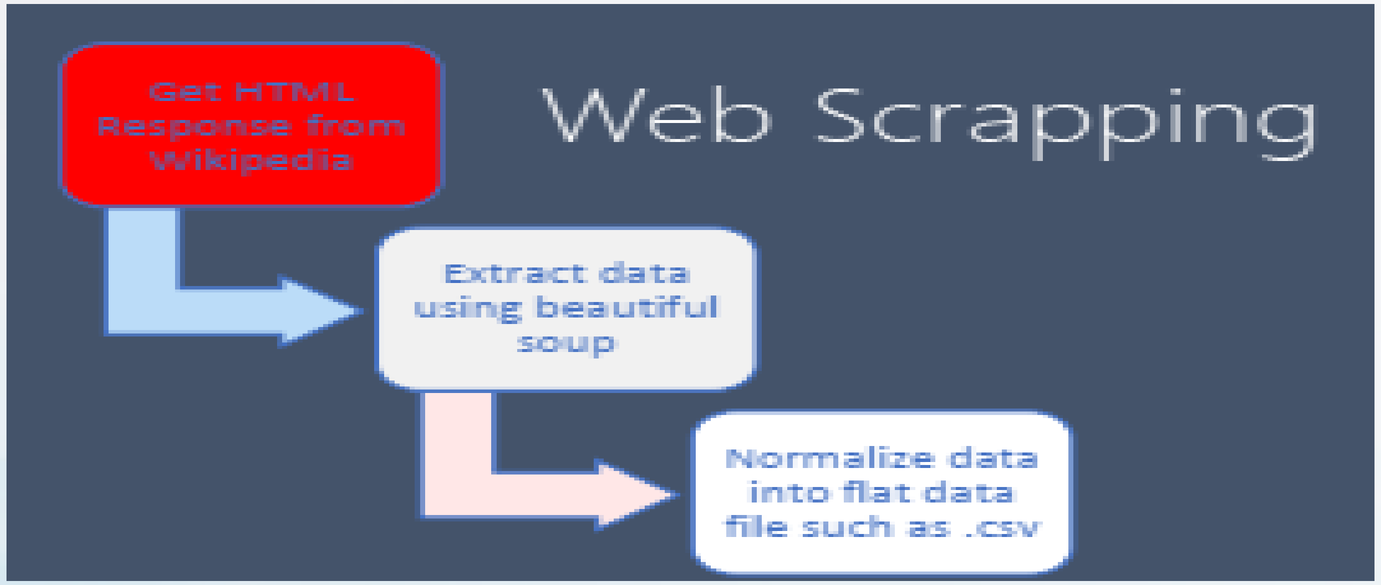
Predictive analysis machine learning prediction process involved data splitting, model training, and hyper-parameter tuning, followed by accuracy and performance evaluation, to determine the best model for predicting landing outcomes.



Data Collection

Data was collected using GET requests to the SpaceX API and web scraping from Wikipedia, then processed into pandas Data Frames for analysis.


SpaceX API
Use SpaceX REST API
API returns SpaceX data in .JSON
Normalize data into flat data file such as .csv


Get HTML Response from Wikipedia
Web Scrapping
Extract data using beautiful soup
Normalize data into flat data file such as .csv

SpaceX API

- Request rocket launch data from the **SpaceX API**, decode the response using .json(), and convert it to a DataFrame using json_normalize().
- Use custom functions to request additional launch information from the SpaceX API, create a dictionary from the data, and convert it into a DataFrame.
- Filter the DataFrame to include only Falcon 9 launches and replace missing values in the Payload Mass column with the calculated mean.
- Export the cleaned and filtered DataFrame to a CSV file.

Web Scraping

- Request Falcon 9 launch data from Wikipedia and create a **BeautifulSoup** object from the HTML response.
- Extract column names from the HTML table header and collect data by parsing the HTML tables.
- Create a dictionary from the parsed data and convert it into a DataFrame.
- Export the DataFrame to a CSV file.

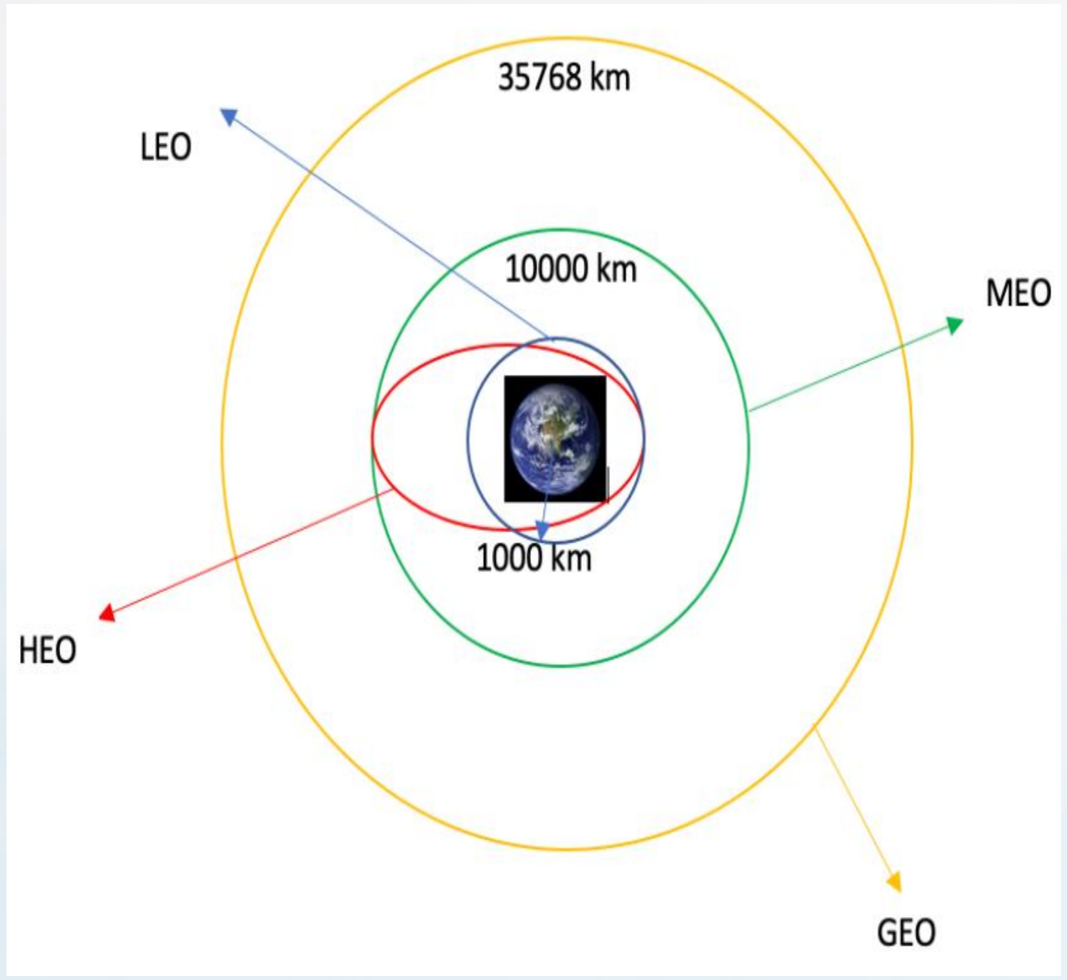
IBM Developer SKILLS NETWORK

Data Wrangling

- Calculate the number of launches at each site
- Calculate the number and occurrence of each orbit
- Calculate the number and occurrence of mission outcomes per orbit type
- Create a landing outcome label from the Outcome column
- Work out the success rate for every landing in the dataset
- Export the dataset as a .CSV file

Landing Outcomes

- **True Ocean**: The mission outcome had a successful landing in a specific region of the ocean
- **False Ocean**: Represented an unsuccessful landing in a specific region of the ocean
- **True RTLS**: Mission had a successful landing on a ground pad
- **False RTLS**: Represented an unsuccessful landing on a ground pad
- **True ASDS**: Mission outcome had a successful landing on a drone ship
- **False ASDS**: Represented an unsuccessful landing on a drone ship
- Outcomes were converted into ‘**1**’ for a successful landing and **‘0’** for an unsuccessful landing



Common Orbit types

IBM Developer SKILLS NETWORK

Exploratory Data Analysis (Data Visualization)

Scatter Graphs

Scatter plots illustrate the extent to which one variable influences another. The connection between these two variables is referred to as their correlation. Typically, scatter plots contain substantial data, and the scatter plots below are drawn for analysis.

- Flight Number vs. Payload
- Flight Number vs. Launch Site
- Payload Mass (kg) vs. Launch Site
- Payload Mass (kg) vs. Orbit type
- Launch success yearly trend

Bar Graph

A bar chart simplifies the comparison of data sets across various groups at a glance. It displays categories on one axis and discrete values on the other, aiming to illustrate the relationship between these two axes. Additionally, bar charts can highlight significant changes in data over time.

- Mean VS. Orbit

Line Graph

Line graphs are valuable because they clearly display data variables and trends, aiding in making predictions about unrecorded data results.

- Success Rate VS. Year

Exploratory Data Analysis (SQL)

We executed SQL queries to extract information from the dataset. For example, we utilized SQL queries to obtain the necessary information from the dataset to answer specific questions about the data. Below is information extracted from the dataset using SQL queries.

- Displaying the names of the unique launch sites in the space mission
- Displaying 5 records where launch sites begin with the string 'CCA’
- Displaying the total payload mass carried by boosters launched by NASA (CRS)
- Displaying average payload mass carried by booster version F9 v1.1
- Listing the date where the successful landing outcome in drone ship was achieved.
- Listing the names of the boosters which have success in ground pad and have payload mass greater than 4000 but less than 6000
- Listing the total number of successful and failure mission outcomes
- Listing the names of the booster_versions which have carried the maximum payload mass.
- Listing the records which will display the month names, successful landing_outcomes in ground pad, booster versions, launch_site for the months in year 2017
- Ranking the count of successful landing_outcomes between the date 2010-06-04 and 2017-03-20 in descending order

IBM Developer
SKILLS NETWORK

Interactive Visual Analytics

Folium Mapping

- We identified all launch sites and added visual elements like markers, circles, and lines on the folium map to indicate the success or failure of launches at each site.
- Launch outcomes were categorized into classes 0 for failure and 1 for success, and using colour-coded marker clusters successful (green) and failure (red) launches at each launch site; we pinpointed launch sites with comparatively high success rates.
- Distances between each launch site and its nearby features were calculated and added colored lines to show distance between launch site CCAFS SLC-40 and its proximity to the nearest coastline, railway, highway, and city

Plotly Dashboards

- We developed an interactive dashboard using Plotly Dash.
- Pie charts were created to display the total number of launches by specific sites.
- A scatter plot was generated to illustrate the relationship between launch outcomes and payload mass (in kilograms) for various booster versions.

Machine Learning Model Development - Predictive Analysis (Classification)

1

Model Implementation

First, we load our dataset into NumPy and Pandas for efficient data manipulation and analysis. Next, we transform the data as needed, such as normalizing features or encoding categorical variables. After transformation, we split our data into training and test datasets to evaluate model performance. We then check the number of test samples to ensure an adequate split. With the dataset prepared, we decide on suitable machine learning algorithms, such as Logistic Regression, SVM, Decision Tree, and KNN. We set our parameters and algorithms for GridSearchCV to find the optimal hyper-parameters. Finally, we fit our datasets into the GridSearchCV objects and train our model to achieve the best performance.

2

Model Evaluations

After training our models, we check the accuracy of each one to evaluate their performance. We then retrieve the tuned hyper-parameters for each type of algorithm to understand the optimal settings found during the GridSearchCV process. Finally, we plot the confusion matrix to visualize and analyze the performance of our models, providing insights into the number of true positives, true negatives, false positives, and false negatives.

3

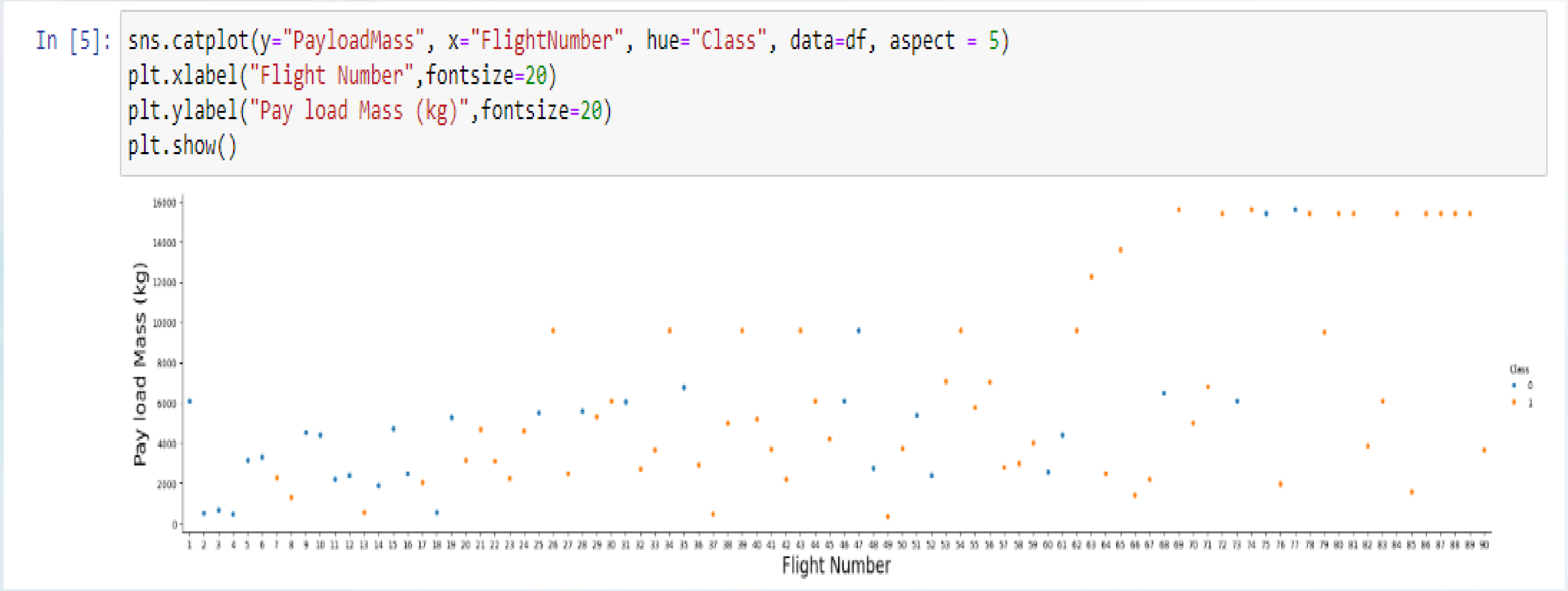
Model Selection

We start with feature engineering, creating and selecting the most relevant features to improve model performance. Next, we focus on algorithm tuning and adjusting hyperparameters to optimize each model's accuracy. Finally, we evaluate all the models, and the best accuracy score is declared the best-performing model, ensuring it provides the most accurate predictions based on our engineered features and tuned algorithms.

IBM Developer
SKILLS NETWORK

Results

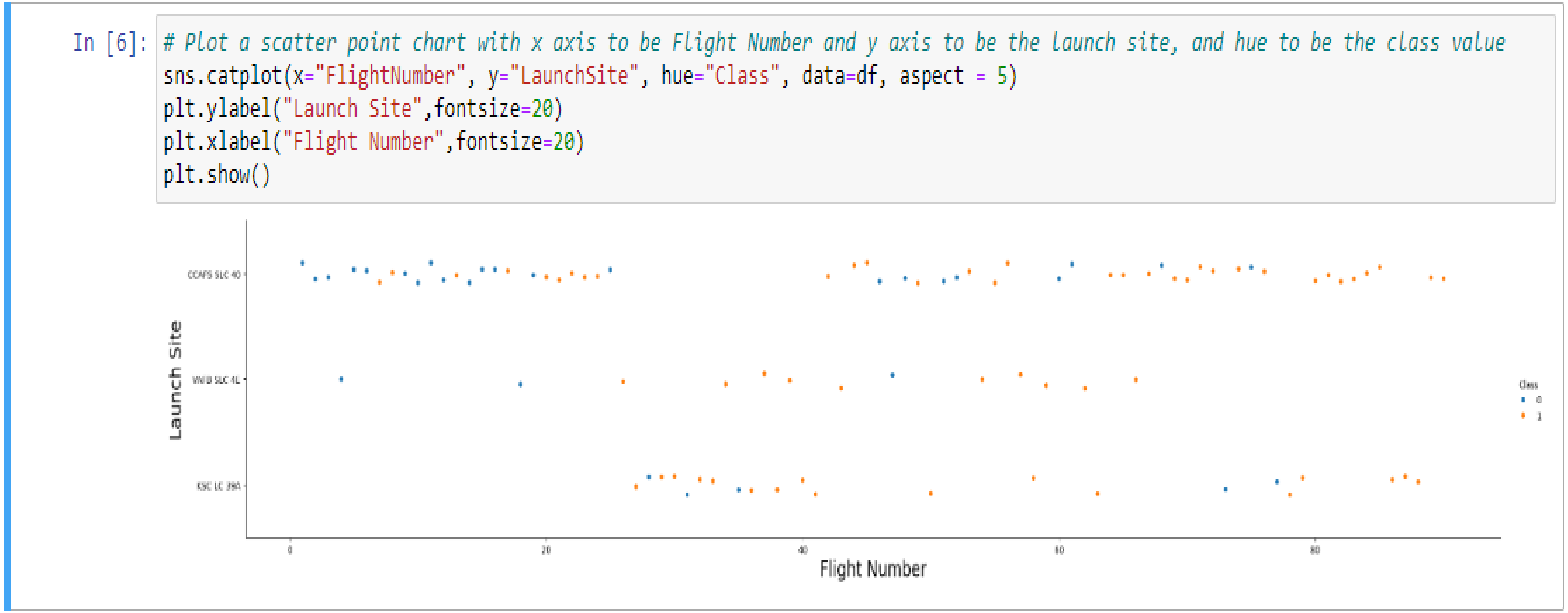
EDA with visualization results – 1. FlightNumber vs. PayloadMass



```
In [5]: sns.catplot(y="PayloadMass", x="FlightNumber", hue="Class", data=df, aspect = 5)
        plt.xlabel("Flight Number",fontsize=20)
        plt.ylabel("Pay load Mass (kg)",fontsize=20)
        plt.show()
```

EDA with visualization results – 2. Flight Number vs. Launch Site

The plot indicates that a higher number of flights at a launch site correlates with a greater success rate at that site.

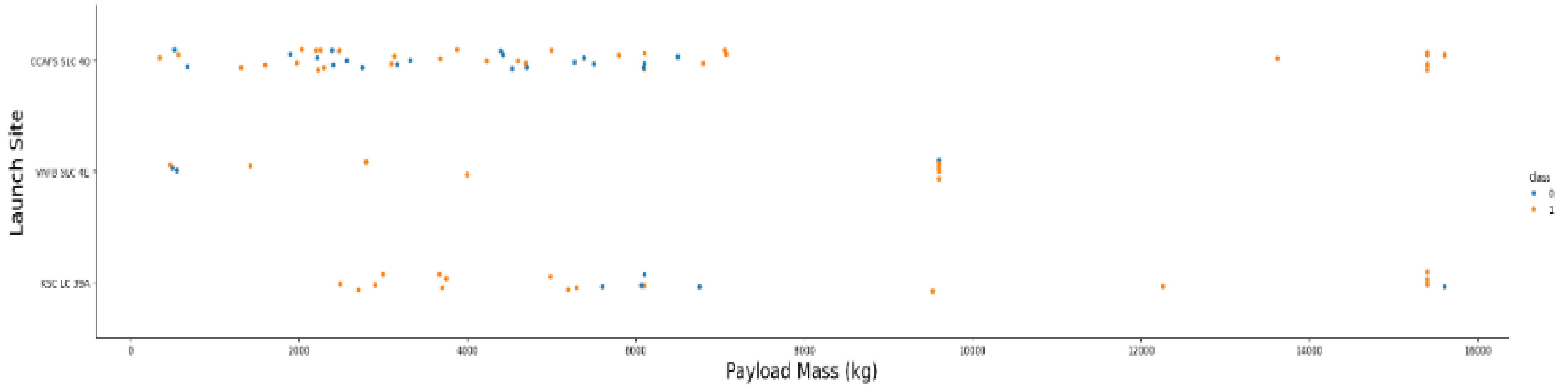


```
In [6]: # Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to be the class value
        sns.catplot(x="FlightNumber", y="LaunchSite", hue="Class", data=df, aspect = 5)
        plt.ylabel("Launch Site",fontsize=20)
        plt.xlabel("Flight Number",fontsize=20)
        plt.show()
```

EDA with visualization results – 3. Payload vs. Launch Site

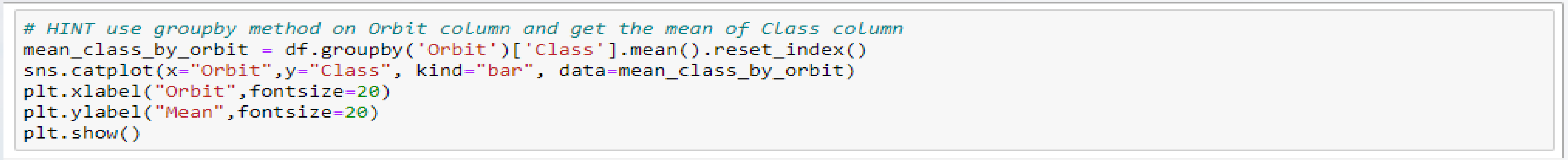
A higher payload mass at Launch Site CCAFS SLC 40 corresponds to a higher rocket success rate.

```
In [7]: # Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch site, and hue to be the class value
        sns.catplot(x="PayloadMass", y="LaunchSite", hue="Class", data=df, aspect = 5)
        plt.ylabel("Launch Site",fontsize=20)
        plt.xlabel("Payload Mass (kg)",fontsize=20)
        plt.show()
```

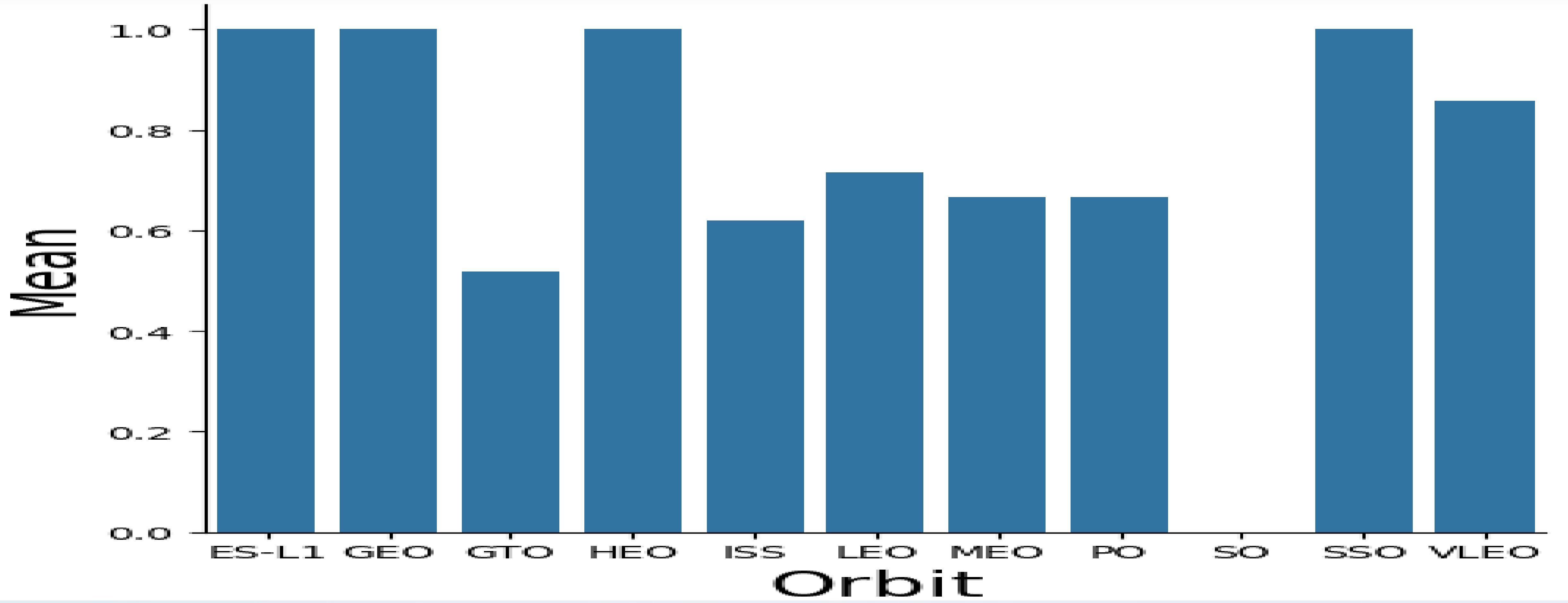


EDA with visualization results – 4. Success rate of each Orbit type

The success rates are as follows: 100% for ES-L1, GEO, HEO, and SSO; 50%-80% for GTO, ISS, LEO, MEO, and PO; and 0% for SO.

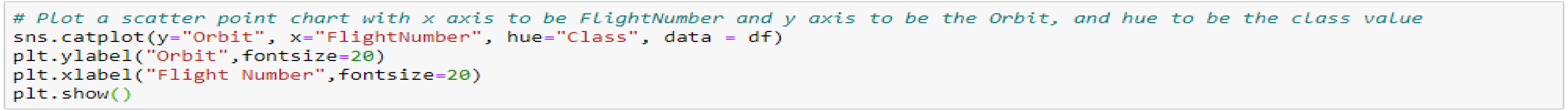


```
# HINT use groupby method on Orbit column and get the mean of Class column
mean_class_by_orbit = df.groupby('Orbit')['Class'].mean().reset_index()
sns.catplot(x="Orbit",y="Class", kind="bar", data=mean_class_by_orbit)
plt.xlabel("Orbit",fontsize=20)
plt.ylabel("Mean",fontsize=20)
plt.show()
```

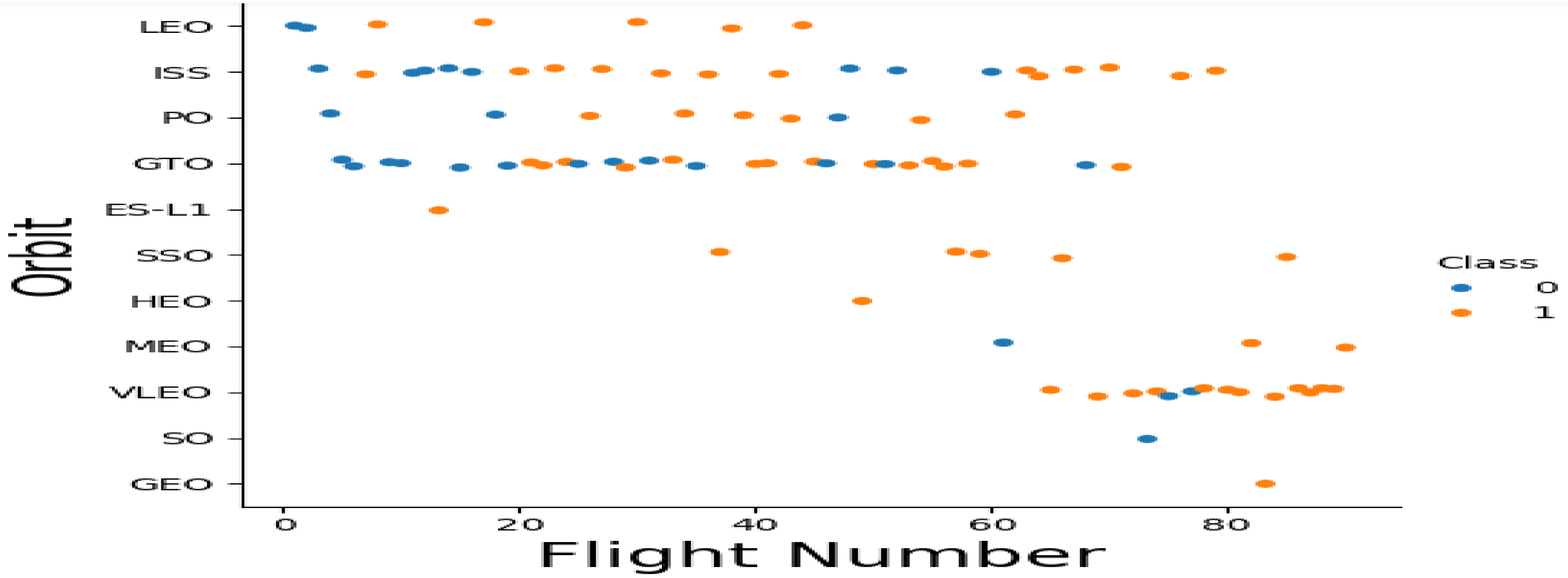


EDA with visualization results – 5. FlightNumber vs. Orbit type

The success rate generally increases with the number of flights for each orbit, a trend that is particularly evident for LEO orbit, whereas GTO orbit does not follow this pattern.



```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class value
sns.catplot(y="Orbit", x="FlightNumber", hue="Class", data = df)
plt.ylabel("Orbit",fontsize=20)
plt.xlabel("Flight Number",fontsize=20)
plt.show()
```

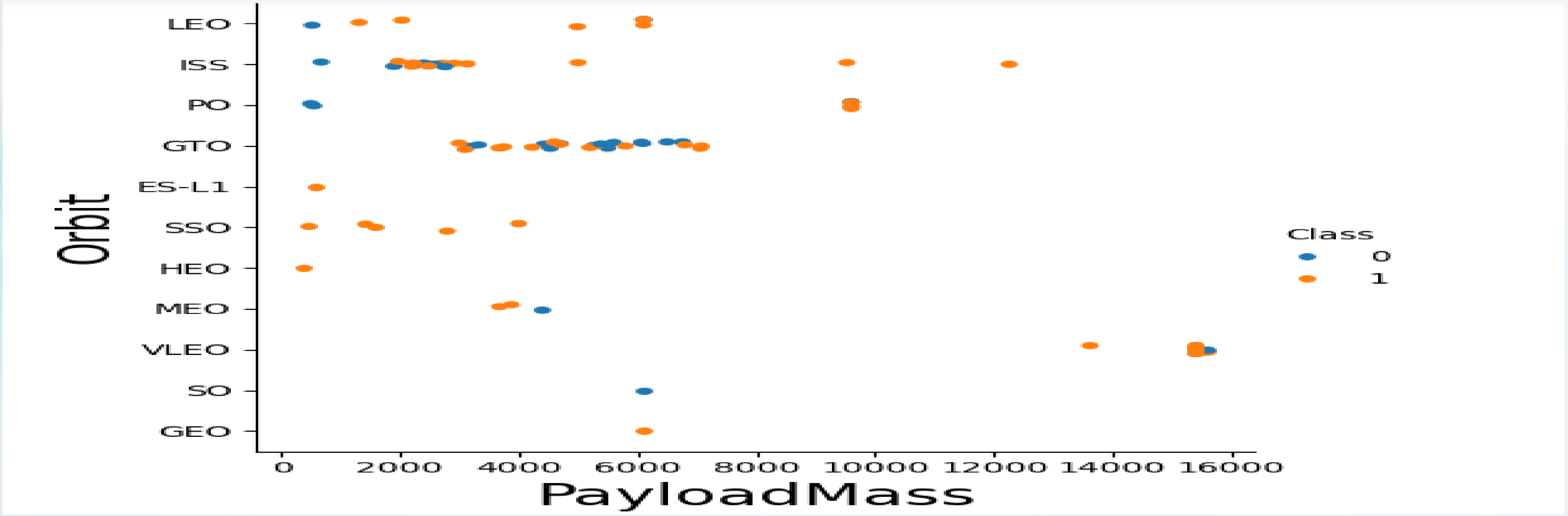


IBM Developer SKILLS NETWORK

EDA with visualization results – 6. Payload vs. Orbit type

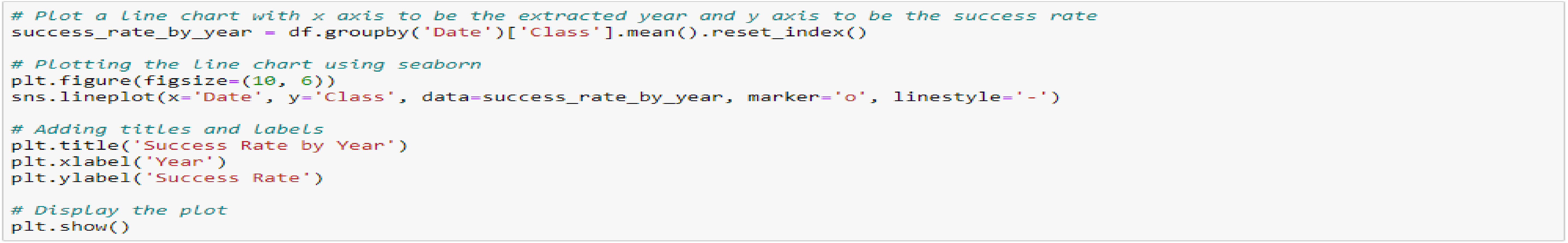
Heavy payloads perform better in LEO, ISS, and PO orbits, while the GTO orbit has mixed success with heavier payloads, demonstrating a negative influence on GTO and a positive influence on Polar LEO (ISS) orbits.

```
# Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, and hue to be the class value
sns.catplot(y="Orbit", x="PayloadMass", hue="Class", data = df)
plt.ylabel("Orbit",fontsize=20)
plt.xlabel("PayloadMass",fontsize=20)
plt.show()
```

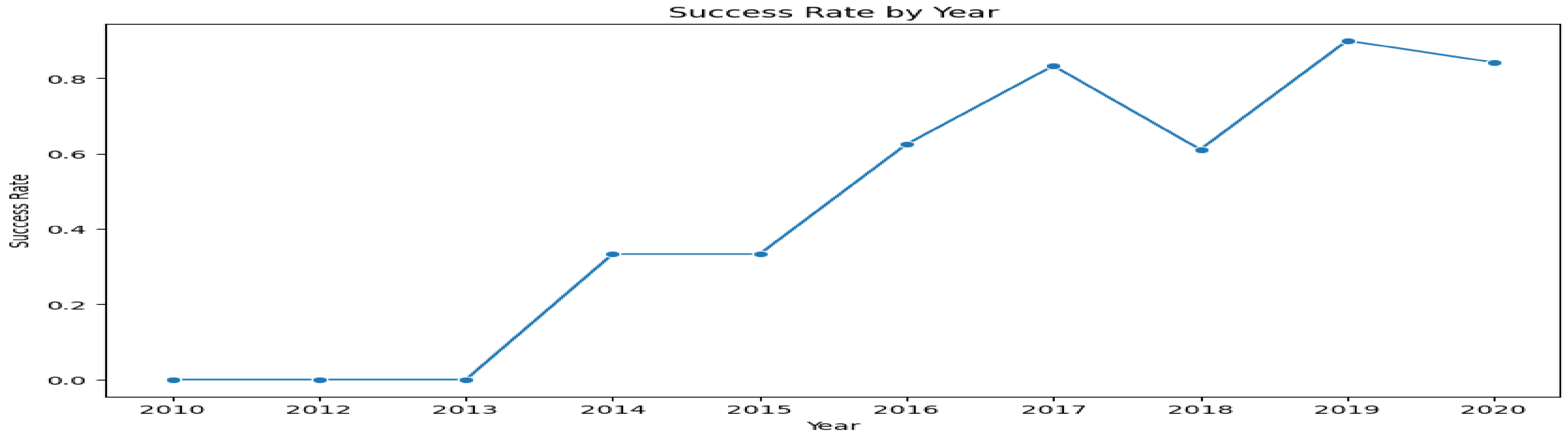


EDA with visualization results – 7. Launch success yearly trend

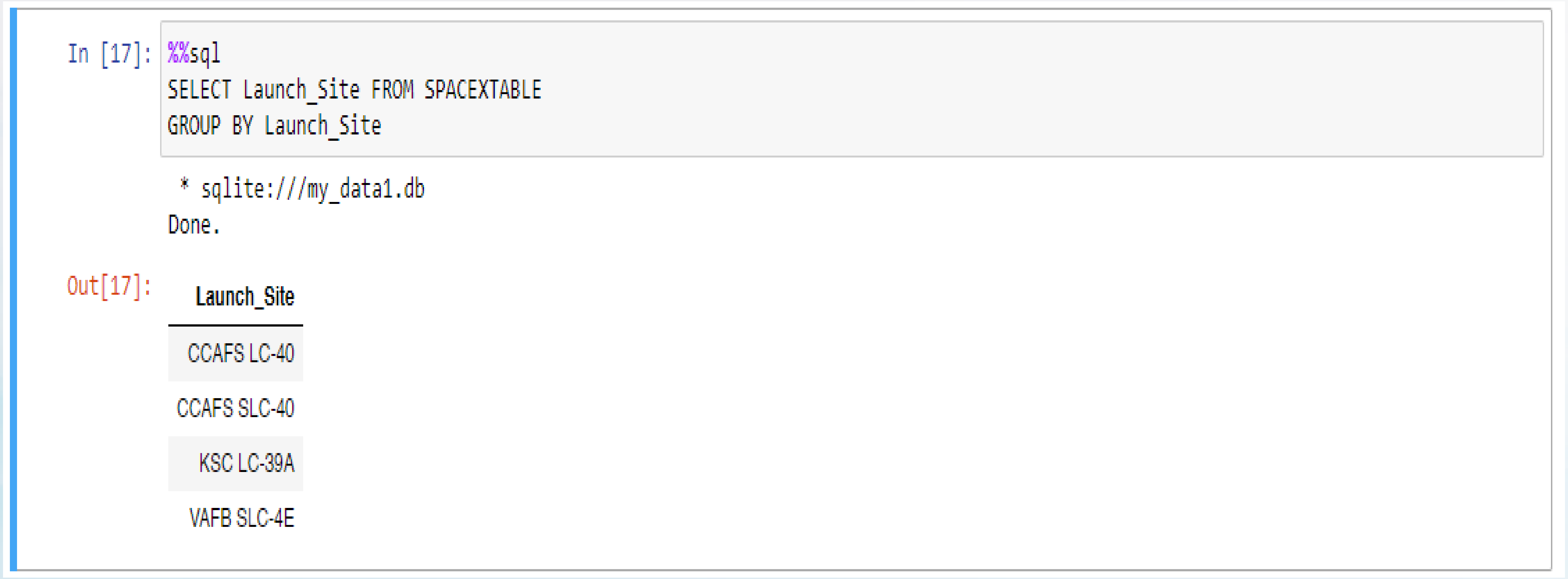
The success rate improved from 2013-2017 and 2018-2019, decreased from 2017-2018 and 2019-2020. Overall, success rate since 2013 kept on increasing till 2020.



```
# Plot a line chart with x axis to be the extracted year and y axis to be the success rate
success_rate_by_year = df.groupby('Date')['Class'].mean().reset_index()

# Plotting the line chart using seaborn
plt.figure(figsize=(10, 6))
sns.lineplot(x='Date', y='Class', data=success_rate_by_year, marker='o', linestyle='-')

# Adding titles and labels
plt.title('Success Rate by Year')
plt.xlabel('Year')
plt.ylabel('Success Rate')

# Display the plot
plt.show()
```



EDA with SQL results – 1. Display the names of the unique launch sites in the space mission

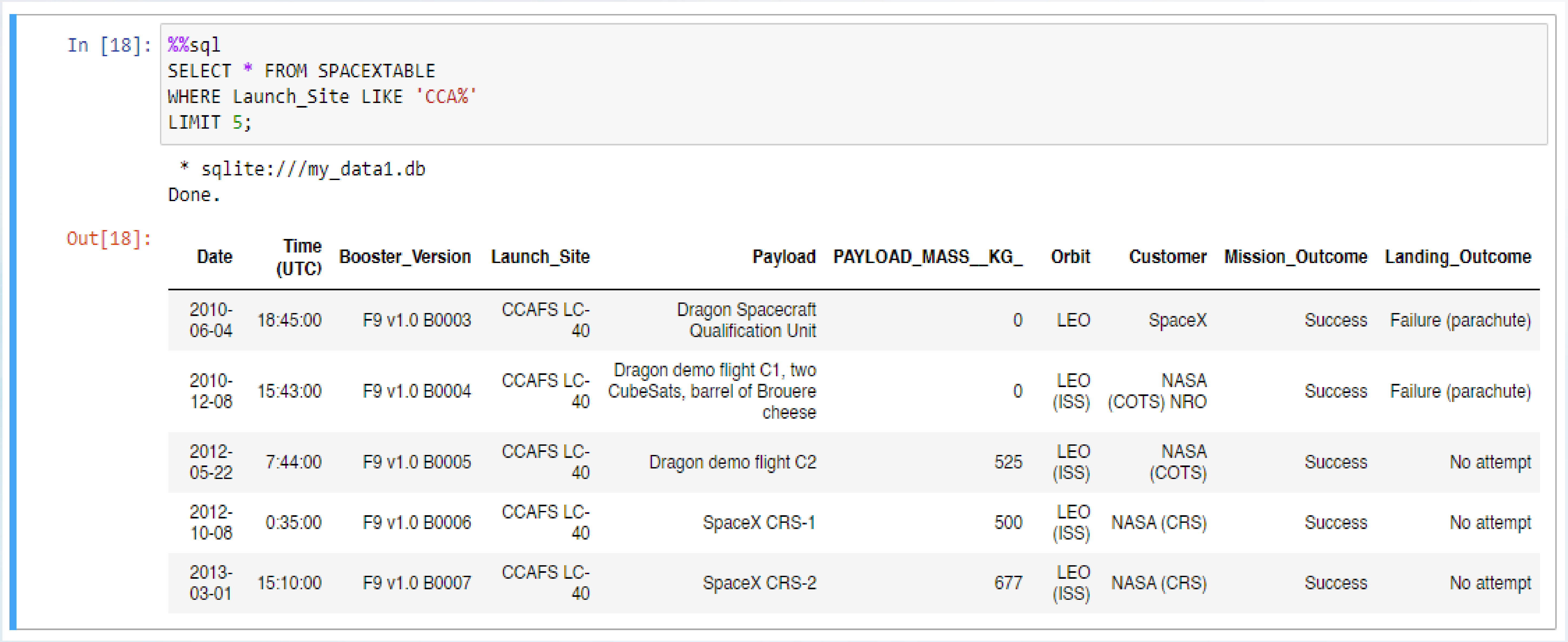


```
In [17]: %%sql
         SELECT Launch_Site FROM SPACEXTABLE
         GROUP BY Launch_Site

          * sqlite:///my_data1.db
         Done.

Out[17]:   Launch_Site
           CCAFS LC-40
          CCAFS SLC-40
            KSC LC-39A
          VAFB SLC-4E
```

The launch sites names are CCAFS LC-40, CCAFS SLC-40, KSC LC-39A, and VAFB SLC-4E.

EDA with SQL results – 2. Display 5 records where launch sites begin with the string 'CCA'

Using the term LIMIT(5) in the query will limit the results to 5 records from SPACEXTABLE, and the LIKE keyword with the wildcard ‘CCA%' specifies that the Launch_Site name must begin with ‘CCA'.



In [18]:

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

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

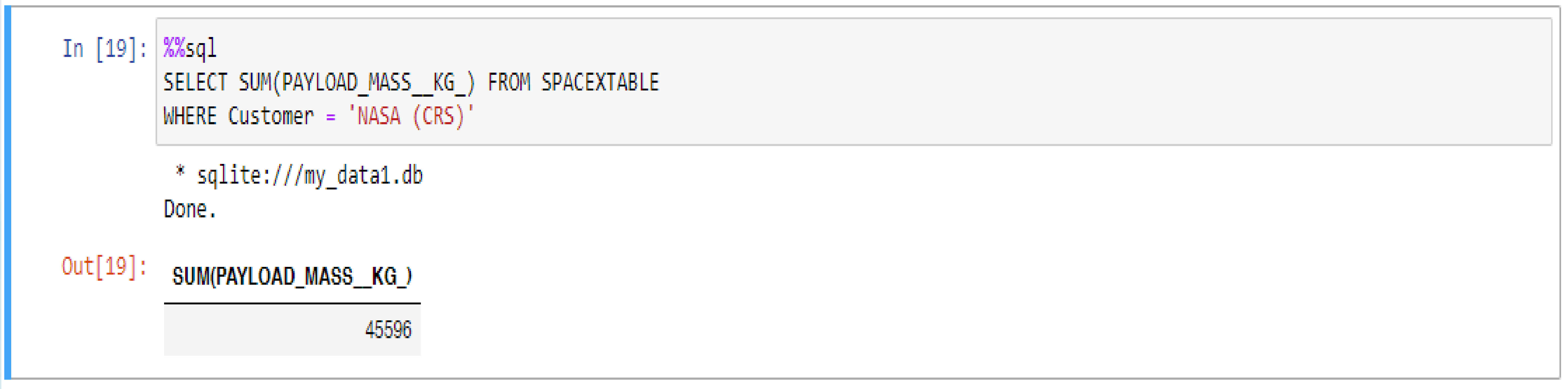
Out[18]:

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

IBM Developer SKILLS NETWORK

EDA with SQL results – 3. Display the total payload mass carried by boosters launched by NASA (CRS)

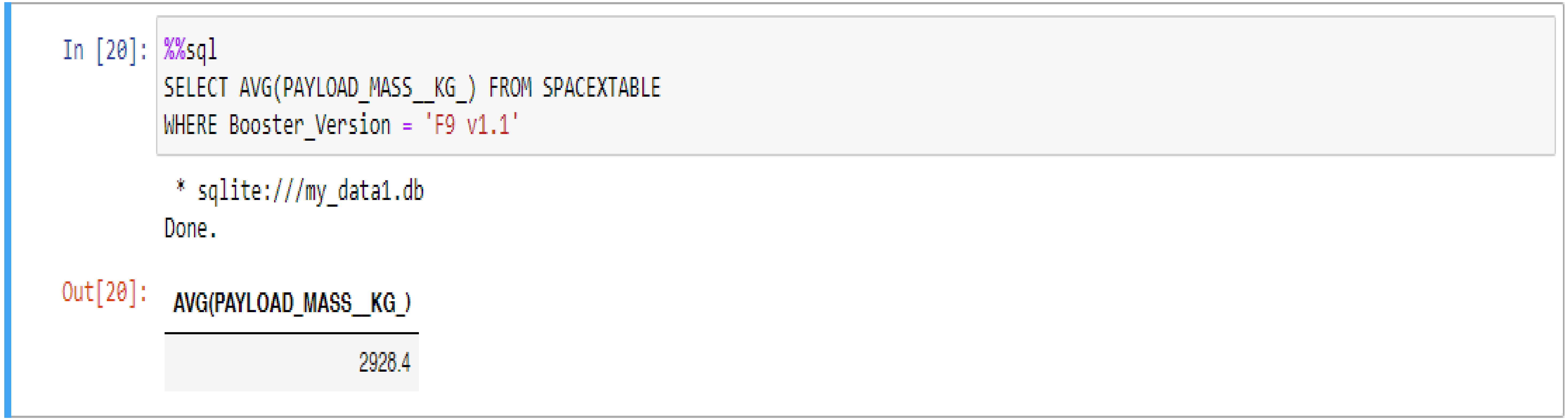
The SUM function calculates the total value in the PAYLOAD_MASS_KG_ column, while the WHERE clause filters the dataset to include only entries for Customer NASA (CRS)



```
In [19]: %%sql
         SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEXTABLE
         WHERE Customer = 'NASA (CRS)'

          * sqlite:///my_data1.db
         Done.

Out[19]:  SUM(PAYLOAD_MASS__KG_)
                           45596
```

EDA with SQL results – 4. Display average payload mass carried by booster version F9 v1.1

The AVG function calculates the average value in the PAYLOAD_MASS_KG_ column, and the WHERE clause restricts the dataset to entries with the Booster_version F9 v1.1.



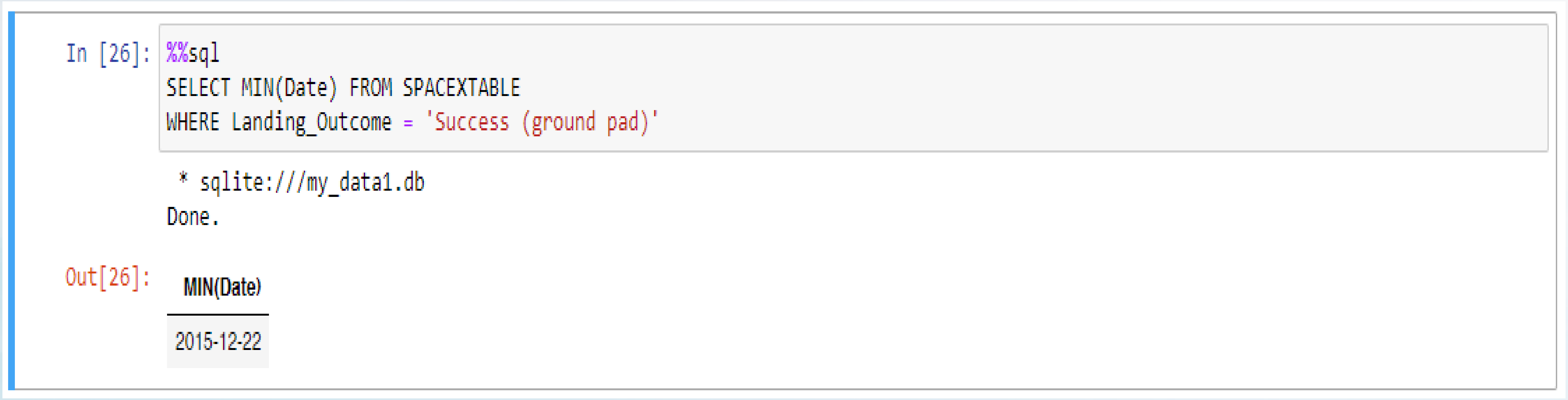
```
In [20]: %%sql
         SELECT AVG(PAYLOAD_MASS__KG_) FROM SPACEXTABLE
         WHERE Booster_Version = 'F9 v1.1'

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

Out[20]:

AVG(PAYLOAD_MASS__KG_)
2928.4

EDA with SQL – 5. List the date when the first successful landing outcome in ground pad was achieve

The MIN function determines the earliest date in the Date column, while the WHERE clause filters the dataset to include only entries with a Landing_Outcome of Success (drone ship).

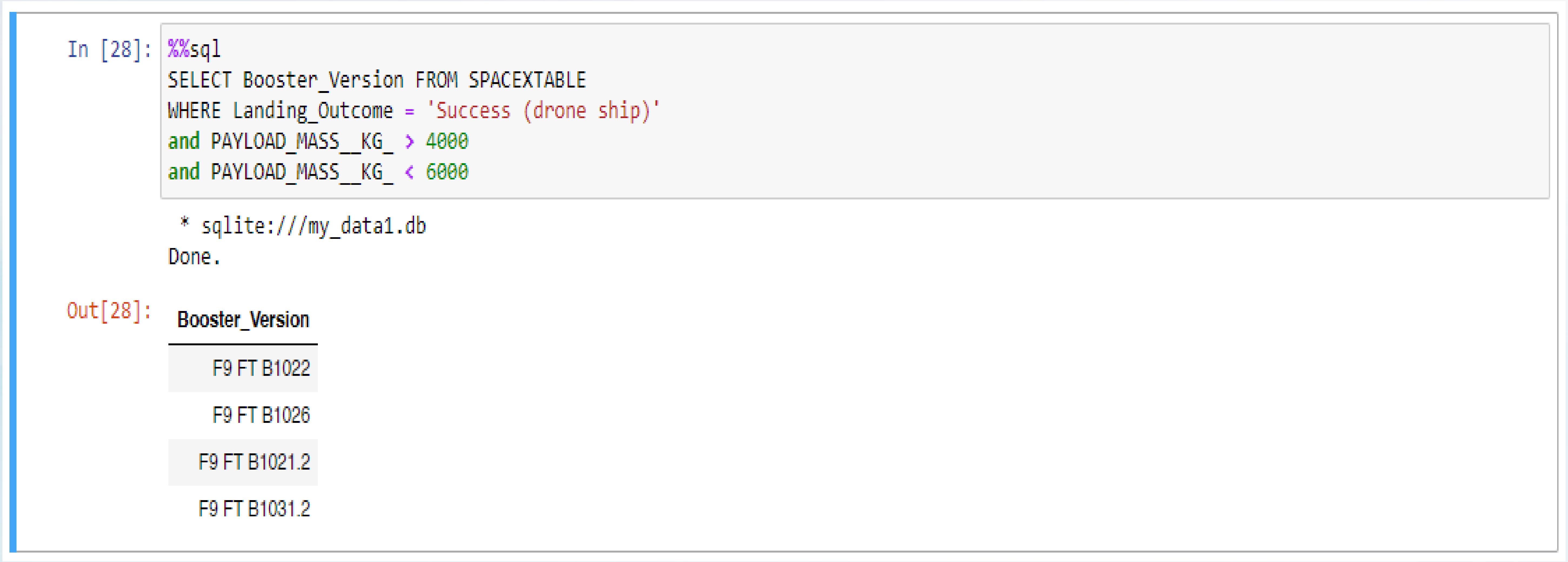


```
In [26]: %%sql
         SELECT MIN(Date) FROM SPACEXTABLE
         WHERE Landing_Outcome = 'Success (ground pad)'

          * sqlite:///my_data1.db
         Done.

Out[26]:  MIN(Date)
          2015-12-22
```

IBM Developer SKILLS NETWORK

EDA with SQL results – 6. Successful Drone Ship Landing with Payload between 4000 and 6000

Selecting only the Booster_Version column, the WHERE clause filters the dataset to include entries with a Landing_Outcome of Success (drone ship), while the AND clause adds additional conditions for Payload_MASS_KG_ to be greater than 4000 and less than 6000.



In [28]:
```
%%sql
SELECT Booster_Version FROM SPACEXTABLE
WHERE Landing_Outcome = 'Success (drone ship)'
and PAYLOAD_MASS__KG_ > 4000
and PAYLOAD_MASS__KG_ < 6000
```

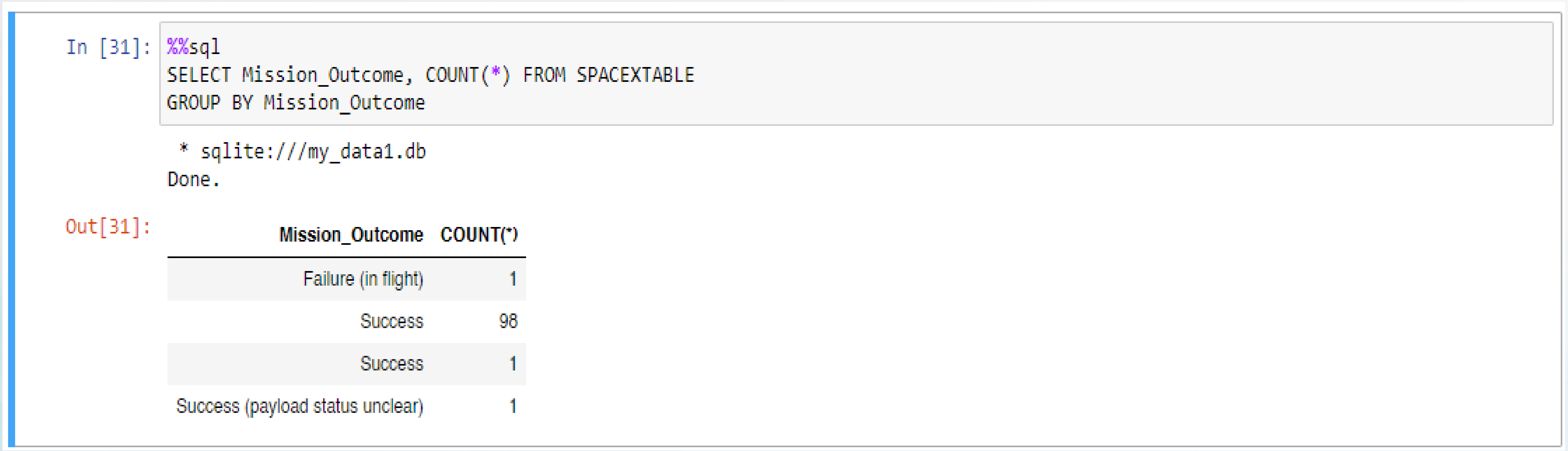
```
 * sqlite:///my_data1.db
Done.
```

Out[28]:

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

EDA with SQL results – 7. List the total number of successful and failure mission outcomes

The query counts the number of missions for each distinct mission outcome in the SPACEXTABLE and displays these counts.



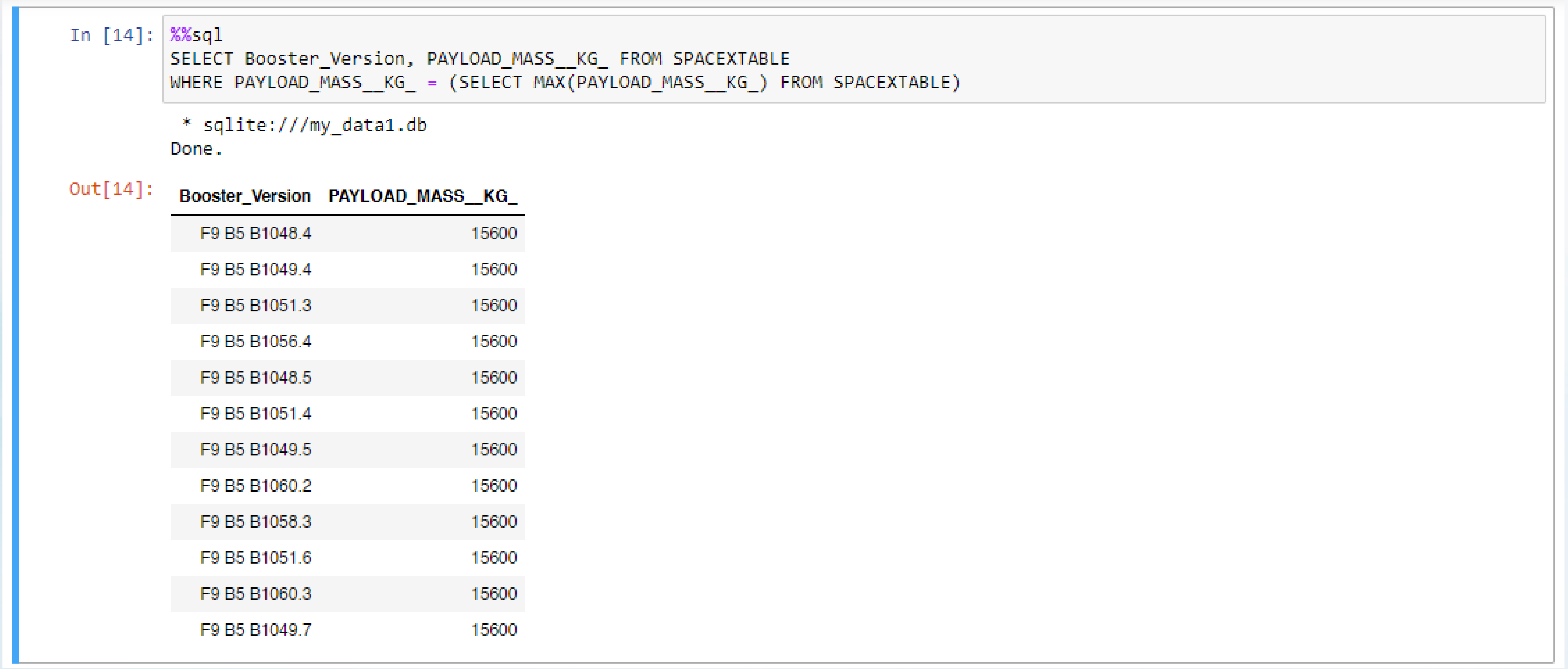
```
In [31]: %%sql
         SELECT Mission_Outcome, COUNT(*) FROM SPACEXTABLE
         GROUP BY Mission_Outcome

          * sqlite:///my_data1.db
         Done.

Out[31]:
```

Mission_Outcome	COUNT(*)
Failure (in flight)	1
Success	98
Success	1
Success (payload status unclear)	1

IBM Developer SKILLS NETWORK

EDA with SQL results – 8. List the names of the booster_versions which have carried the maximum payload mass

The query retrieves the Booster_Version and PAYLOAD_MASS__KG_ for the record with the maximum payload mass from the SPACEXTABLE.

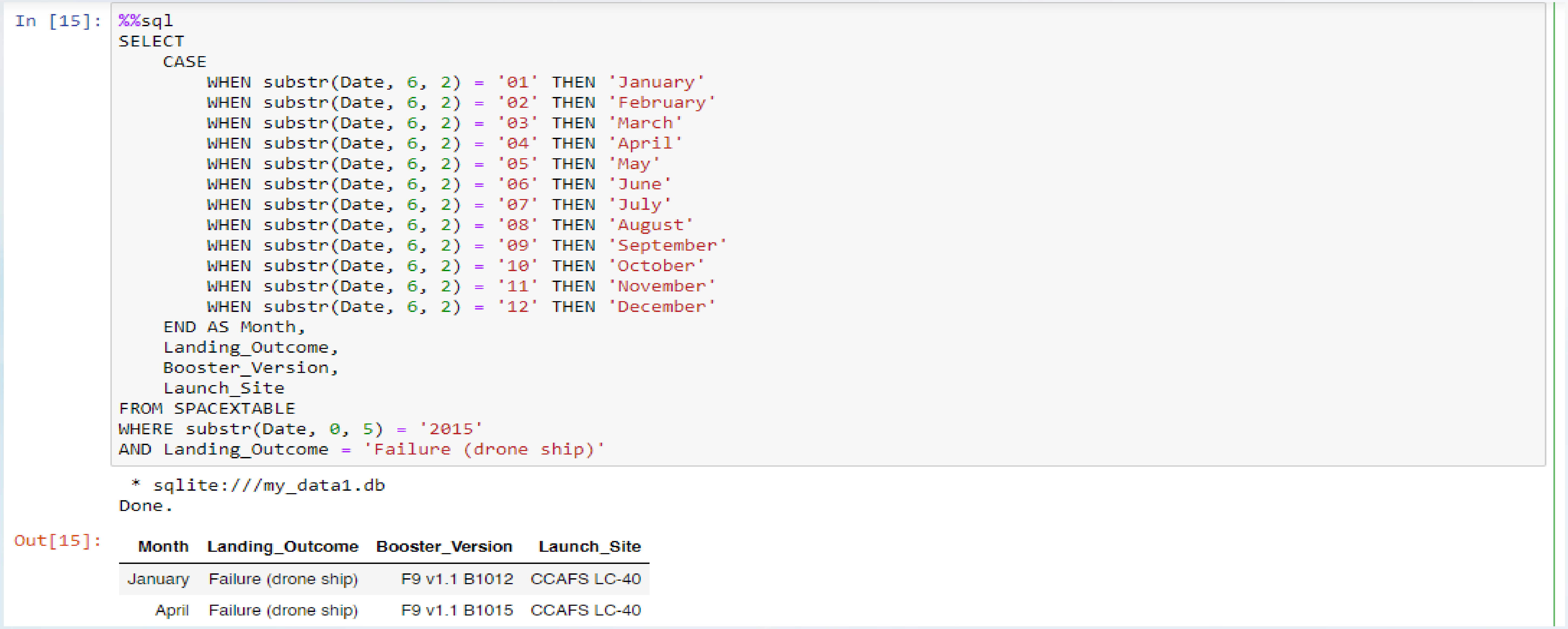


```
In [14]: %%sql
         SELECT Booster_Version, PAYLOAD_MASS__KG_ FROM SPACEXTABLE
         WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTABLE)

          * sqlite:///my_data1.db
         Done.

Out[14]:
```

Booster_Version	PAYLOAD_MASS__KG_
F9 B5 B1048.4	15600
F9 B5 B1049.4	15600
F9 B5 B1051.3	15600
F9 B5 B1056.4	15600
F9 B5 B1048.5	15600
F9 B5 B1051.4	15600
F9 B5 B1049.5	15600
F9 B5 B1060.2	15600
F9 B5 B1058.3	15600
F9 B5 B1051.6	15600
F9 B5 B1060.3	15600
F9 B5 B1049.7	15600

EDA with SQL results – 9. List the records which will display the month names, failure landing_outcomes in drone ship, booster versions, launch_site for the months in year 2015.

The query is intended to retrieve information about SpaceX launches that occurred in the year 2015 and where the landing outcome was specifically noted as 'Failure (drone ship)'. It enhances the readability of the output by converting the numeric month representation into its corresponding month name using the CASE statement in the SELECT clause.



In [15]:

```
%%sql
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 SPACEXTABLE
WHERE substr(Date, 0, 5) = '2015'
AND Landing_Outcome = 'Failure (drone ship)'
```

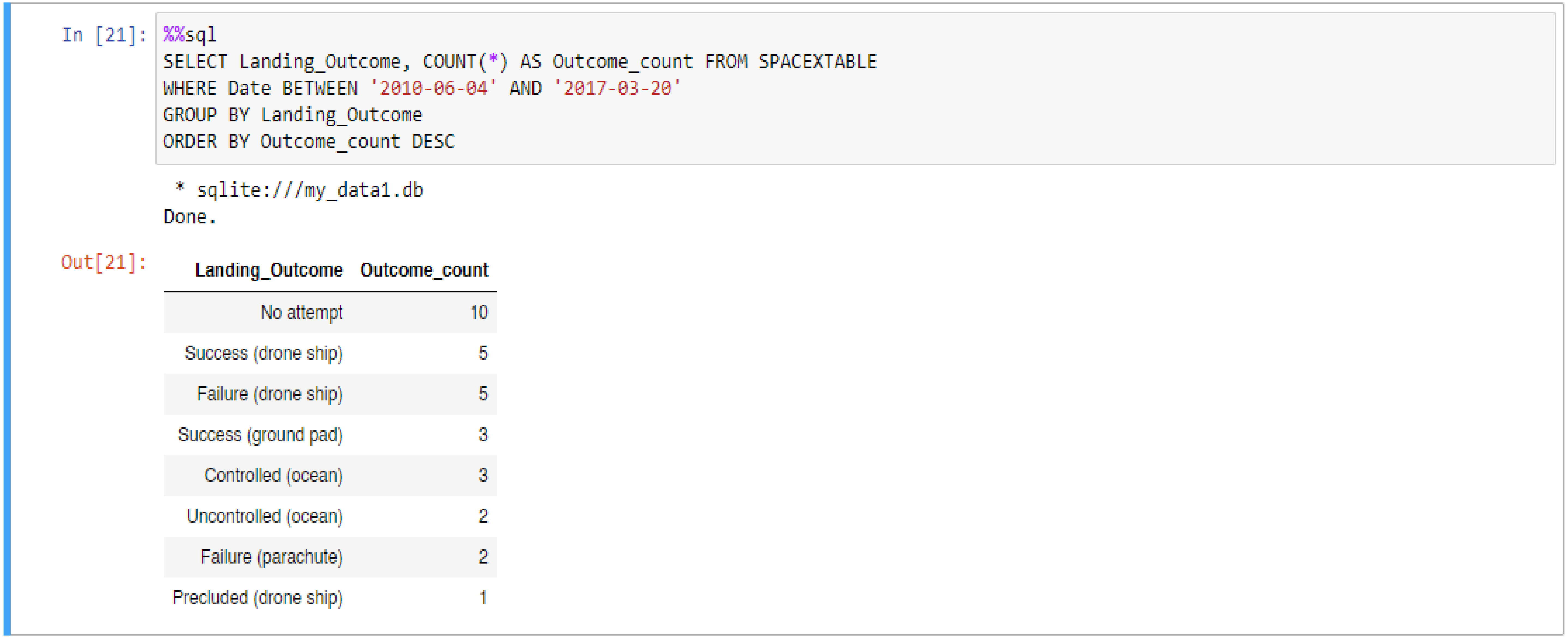
```
 * sqlite:///my_data1.db
Done.
```

Out[15]:

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

EDA with SQL results – 10. 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.

This SQL query aggregates and counts occurrences of different landing outcomes from the SPACEXTABLE within the date range from June 4, 2010, to March 20, 2017, providing a summary of each landing outcome ordered by frequency

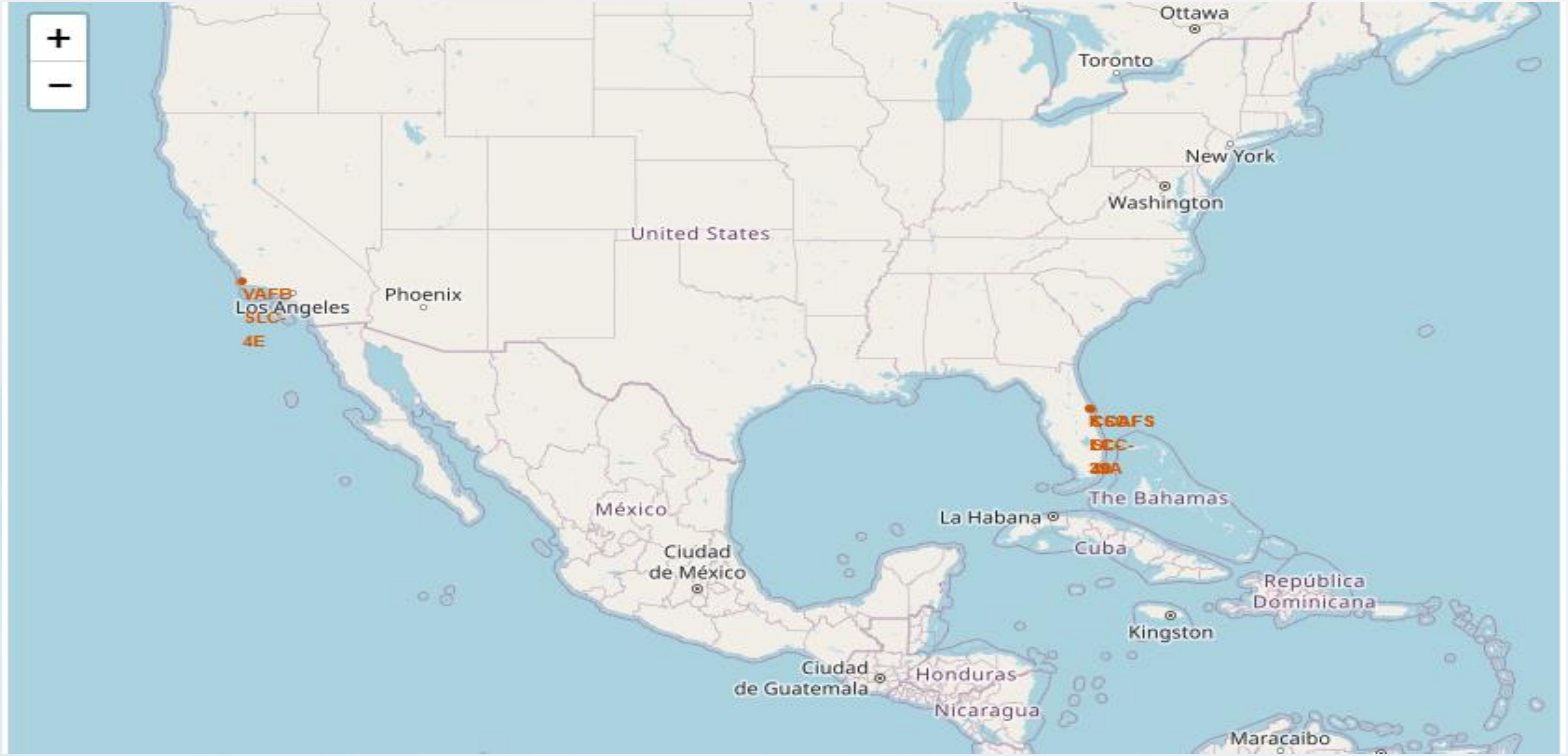


```
In [21]: %%sql
         SELECT Landing_Outcome, COUNT(*) AS Outcome_count FROM SPACEXTABLE
         WHERE Date BETWEEN '2010-06-04' AND '2017-03-20'
         GROUP BY Landing_Outcome
         ORDER BY Outcome_count DESC

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

Out[21]:

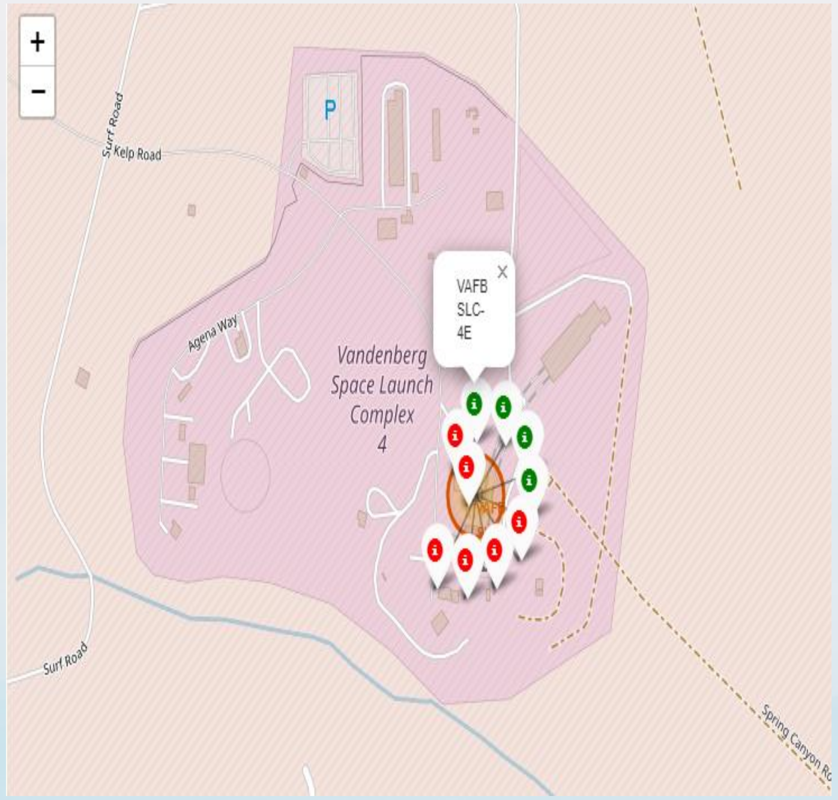
Landing_Outcome	Outcome_count
No attempt	10
Success (drone ship)	5
Failure (drone ship)	5
Success (ground pad)	3
Controlled (ocean)	3
Uncontrolled (ocean)	2
Failure (parachute)	2
Precluded (drone ship)	1

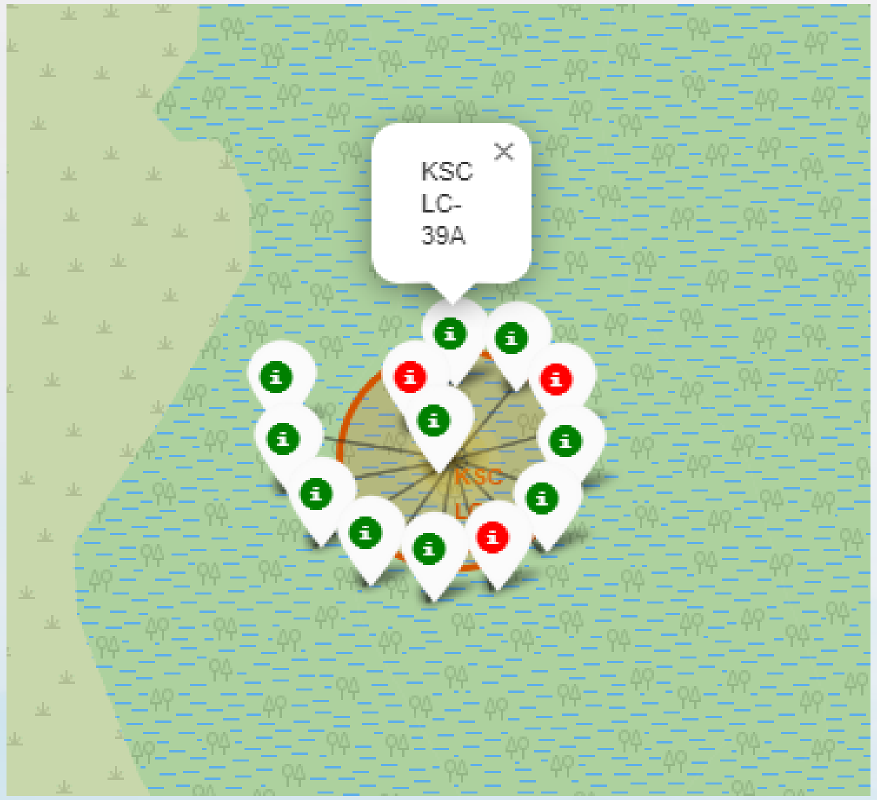
Interactive map with Folium results – 1. Mark all launch sites on a map

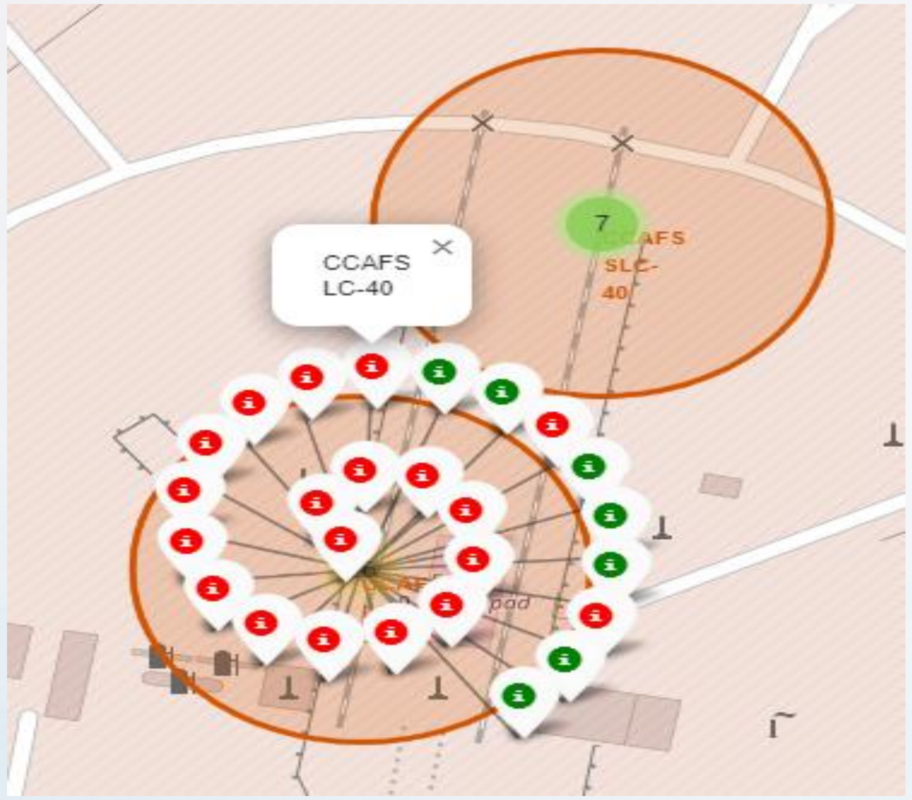


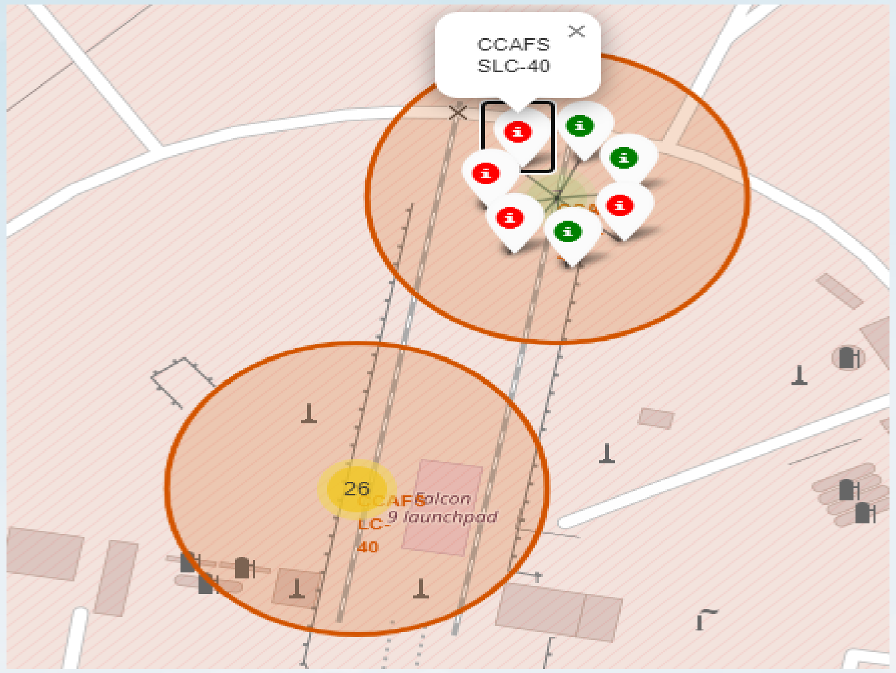
IBM Developer SKILLS NETWORK

Interactive map with Folium results – 2. Mark the success/failed launches for each site on the map





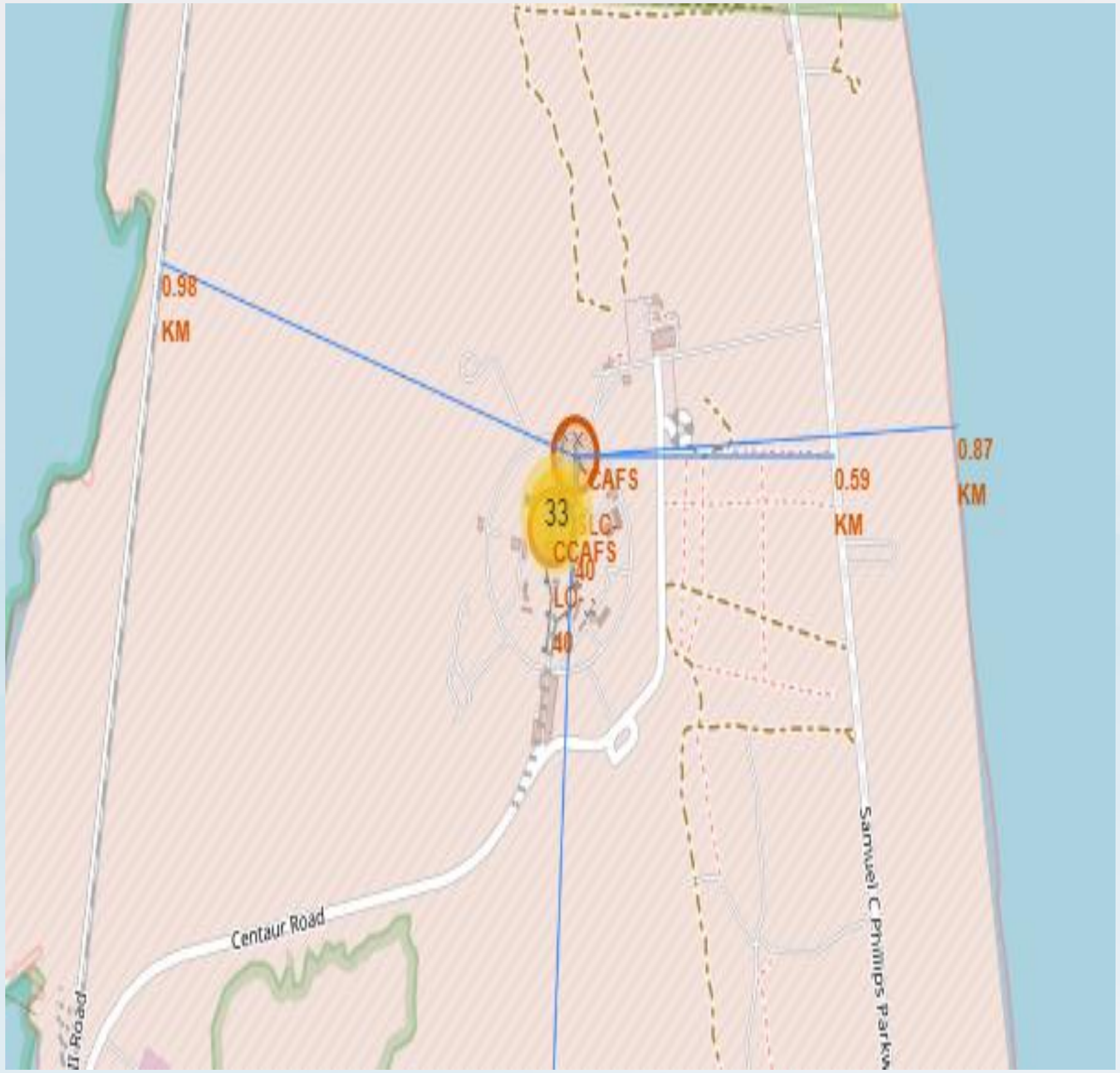


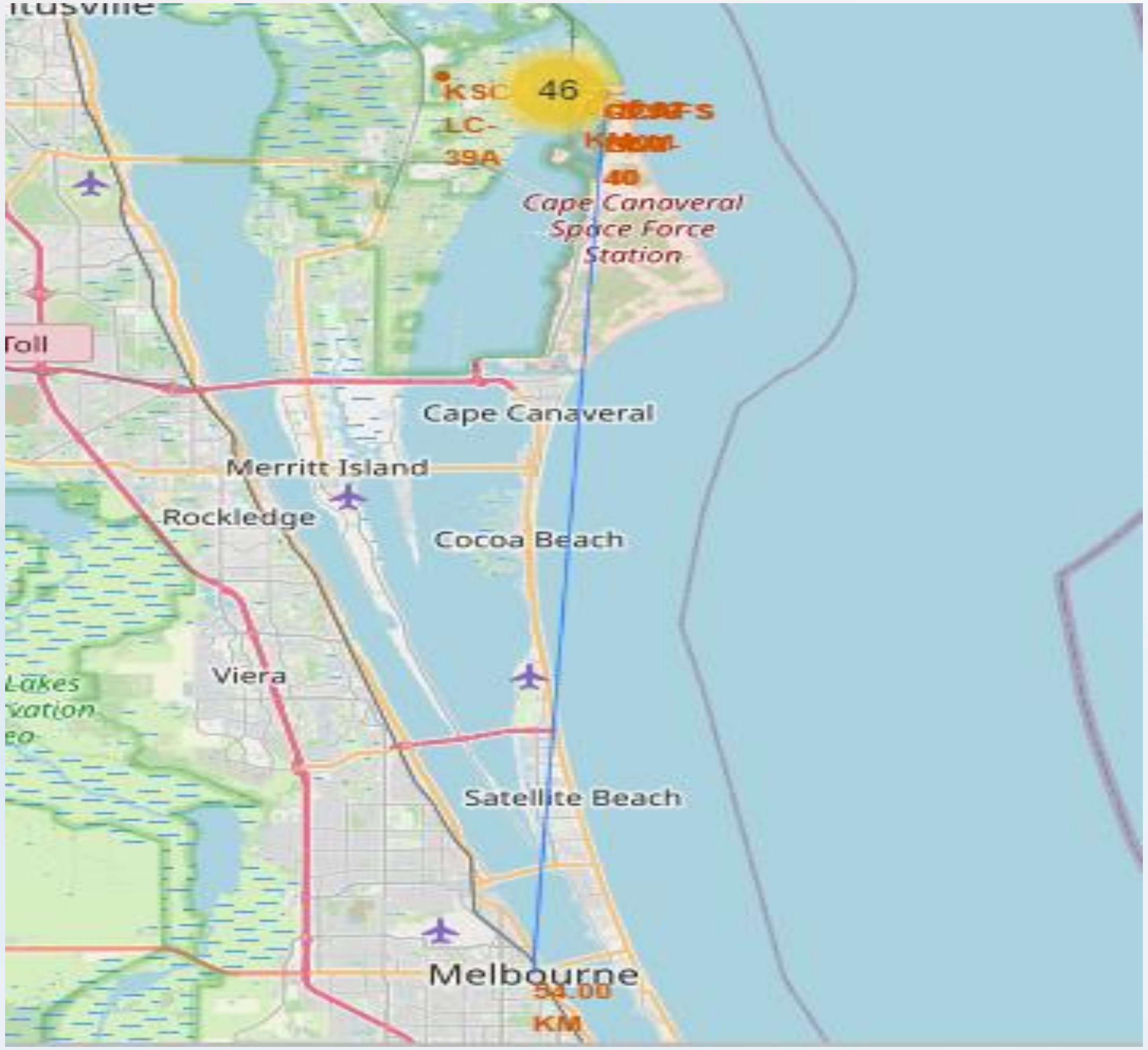


Green Marker shows successful Launches and Red Marker shows Failures

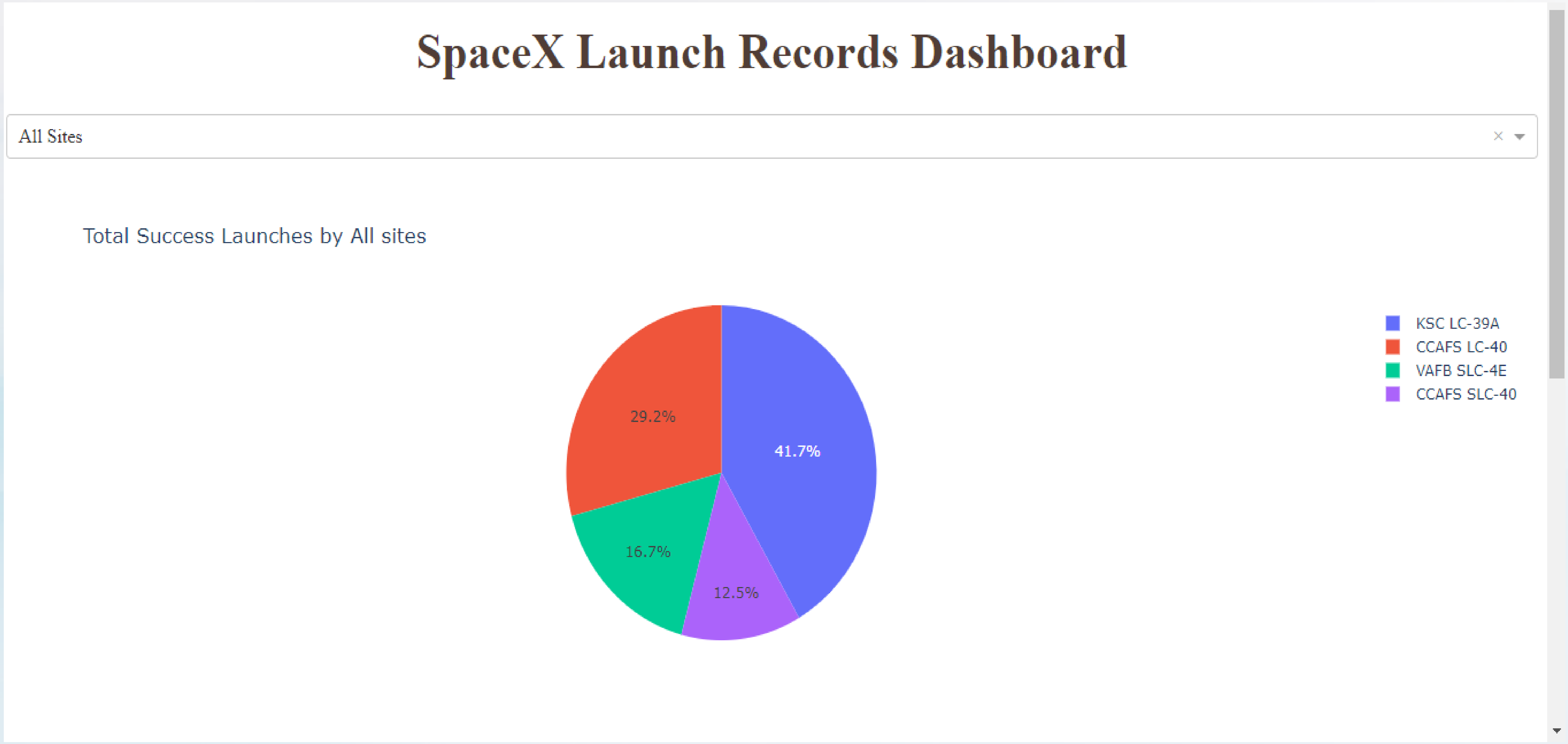
IBM Developer SKILLS NETWORK

Interactive map with Folium results – 3. Calculate the distances between a launch site to its proximities





Plotly Dash dashboard results – 1. Pie chart showing the success percentage achieved by each launch site

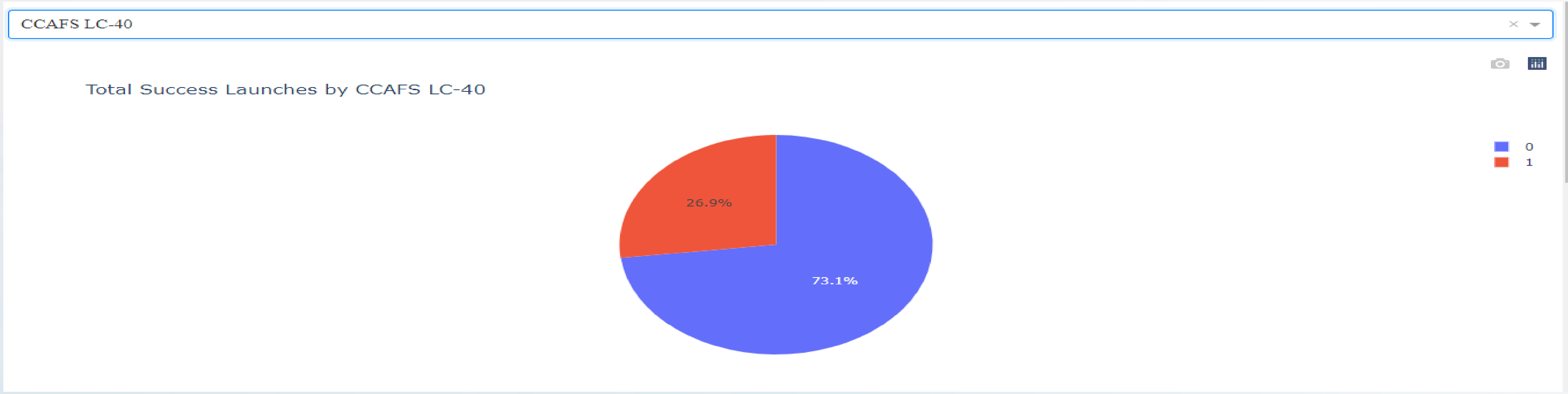

SpaceX Launch Records Dashboard

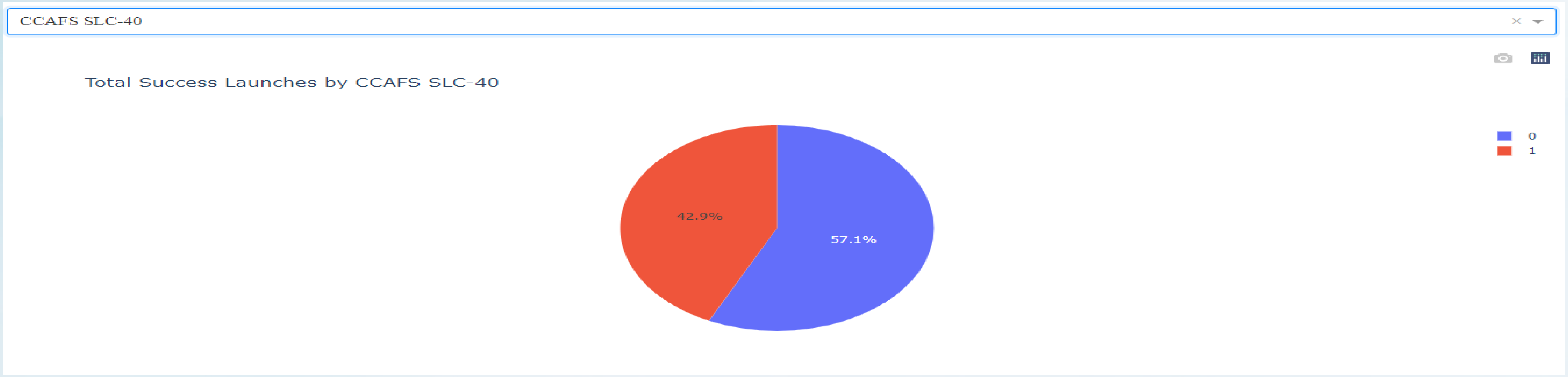
All Sites

Total Success Launches by All sites

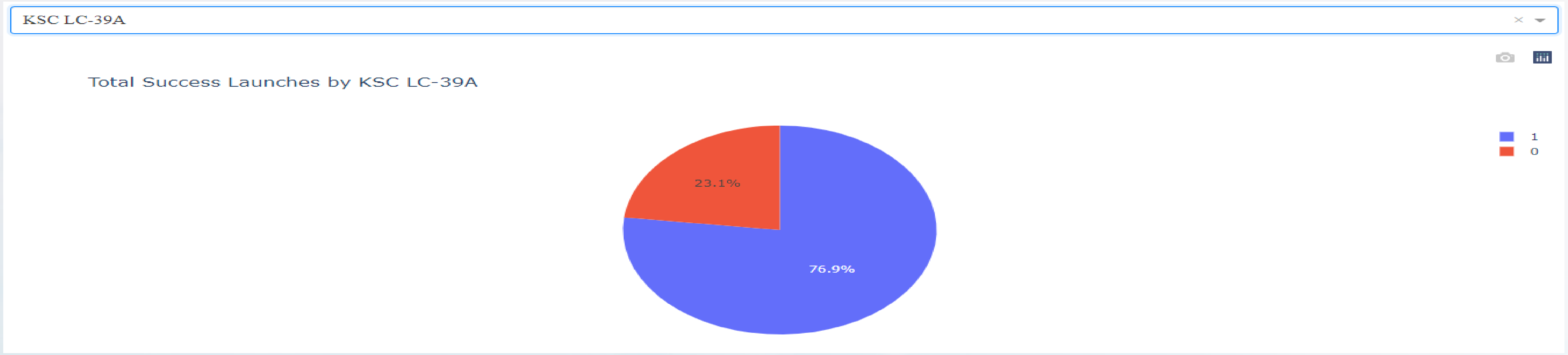
IBM Developer SKILLS NETWORK

Plotly Dash dashboard results– 2. Pie chart showing the Launch sites with the highest launch success ratio

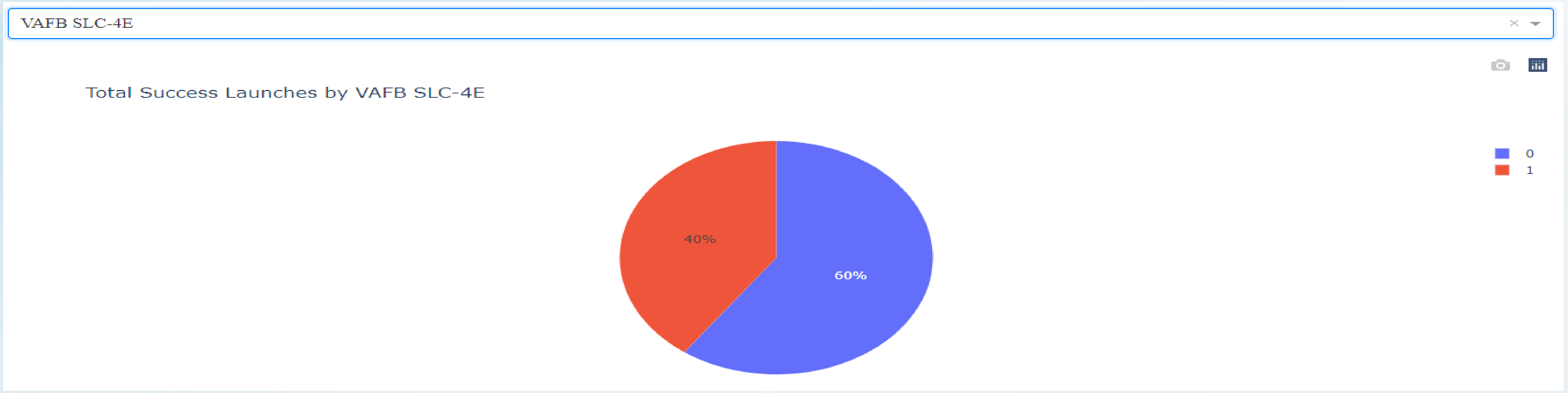

CCAFS LC-40


CCAFS SLC-40

Plotly Dash dashboard results – 2. Pie chart showing the Launch sites with the highest launch success ratio Cont.



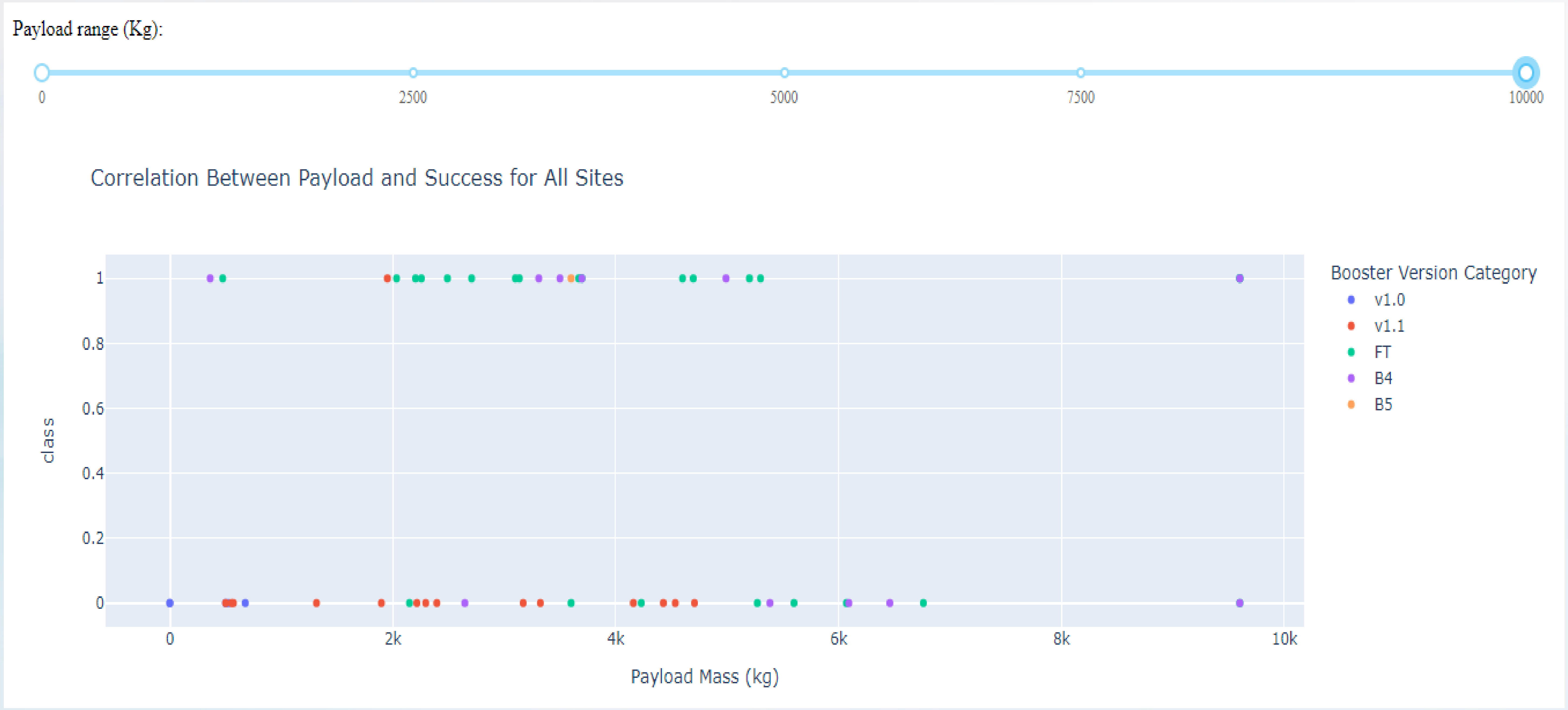
KSC LC-39A



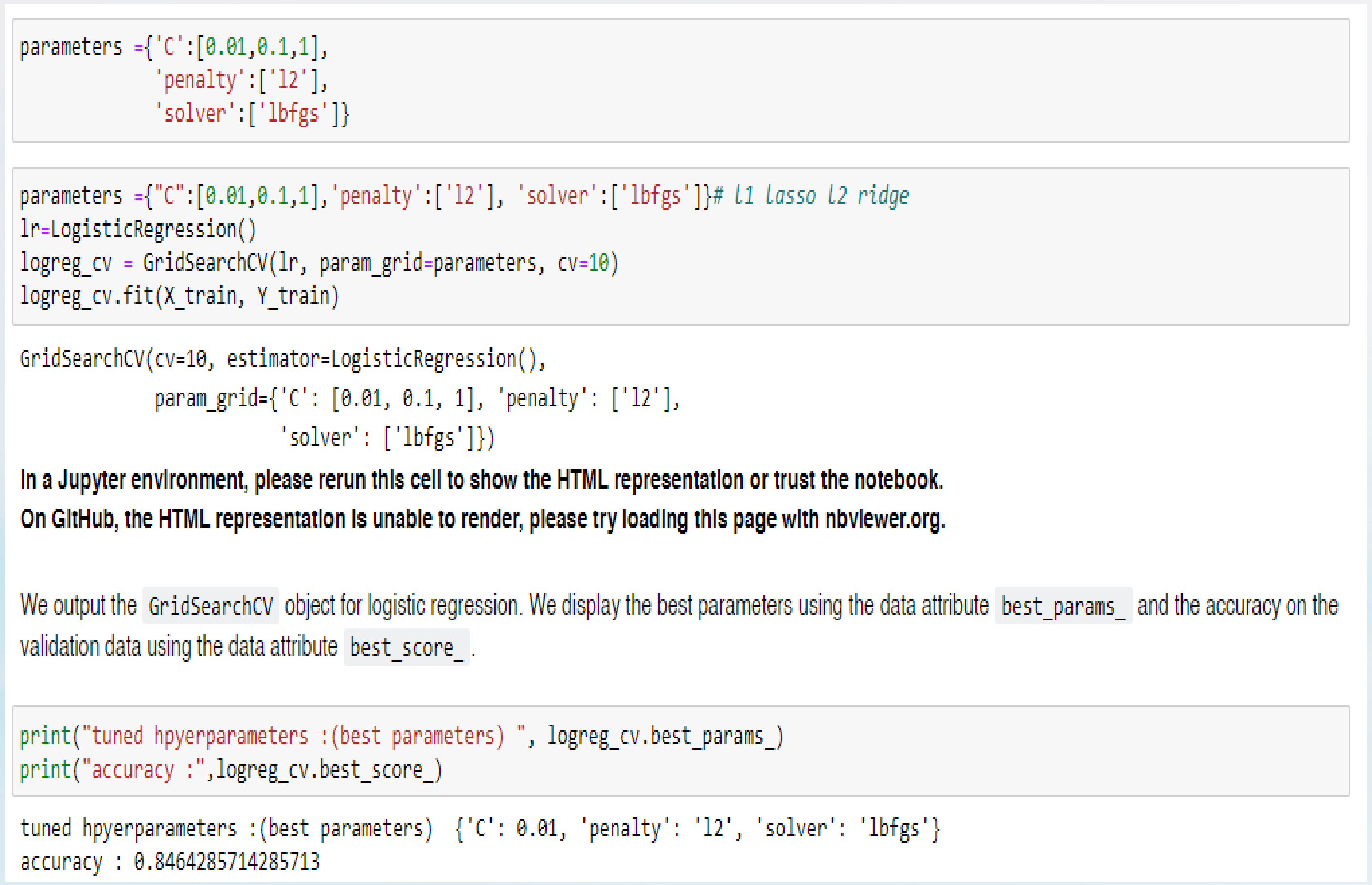
VAFB SLC-4E

IBM Developer SKILLS NETWORK

Plotly Dash dashboard results– 3. Scatter plot of Payload vs Launch Outcome for all sites, with different payload selected in the range slider



Predictive Analysis (Classification) results – 1. Accuracy of the models – Logistic Regression



```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()
logreg_cv = GridSearchCV(lr, param_grid=parameters, cv=10)
logreg_cv.fit(X_train, Y_train)
```

```
GridSearchCV(cv=10, estimator=LogisticRegression(),
             param_grid={'C': [0.01, 0.1, 1], 'penalty': ['l2'],
                         'solver': ['lbfgs']})
```

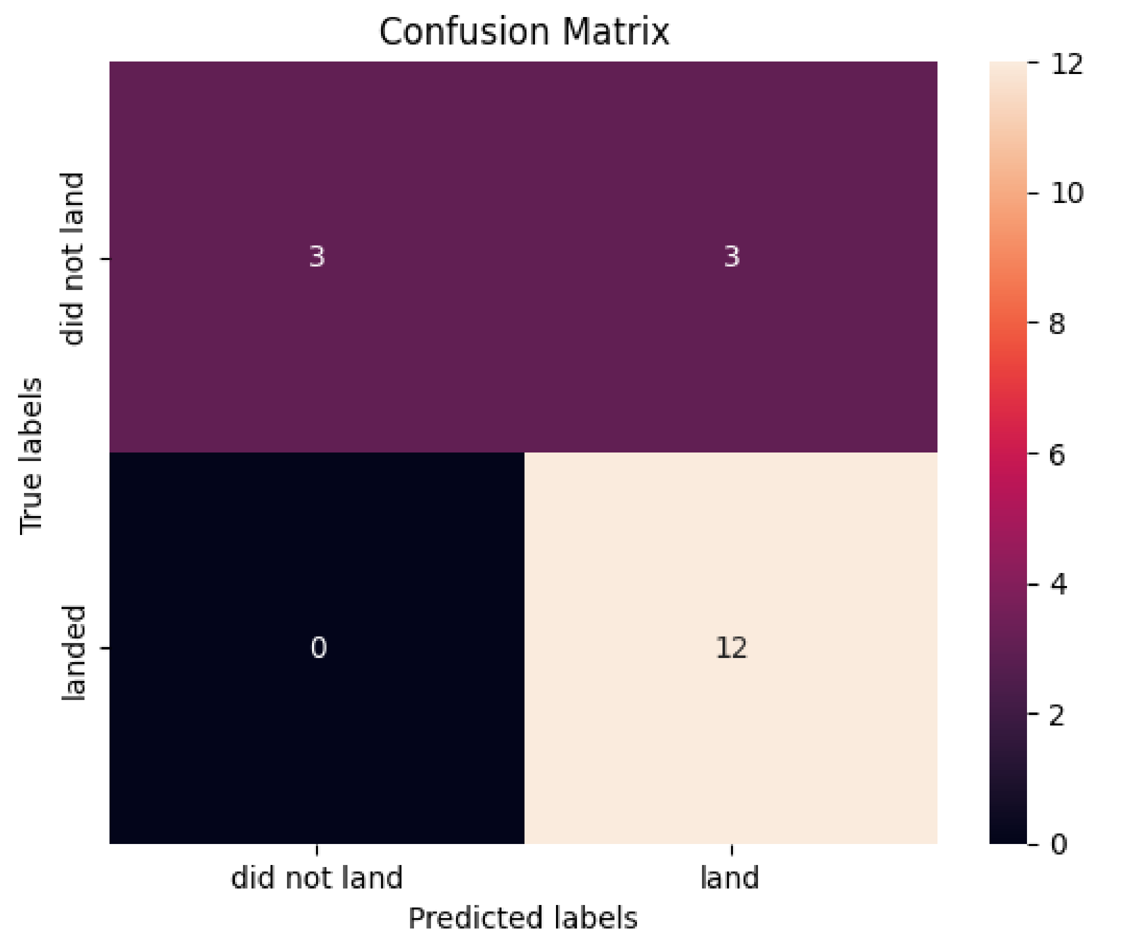
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

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_`.

```
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
```

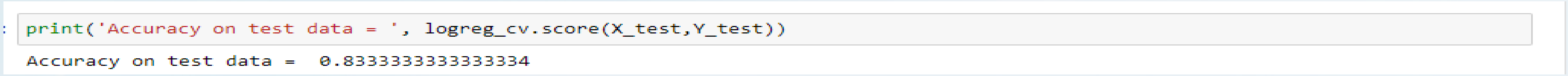
```
yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test, yhat)
```


Confusion Matrix

True labels \ Predicted labels	did not land	land
did not land	3	3
landed	0	12

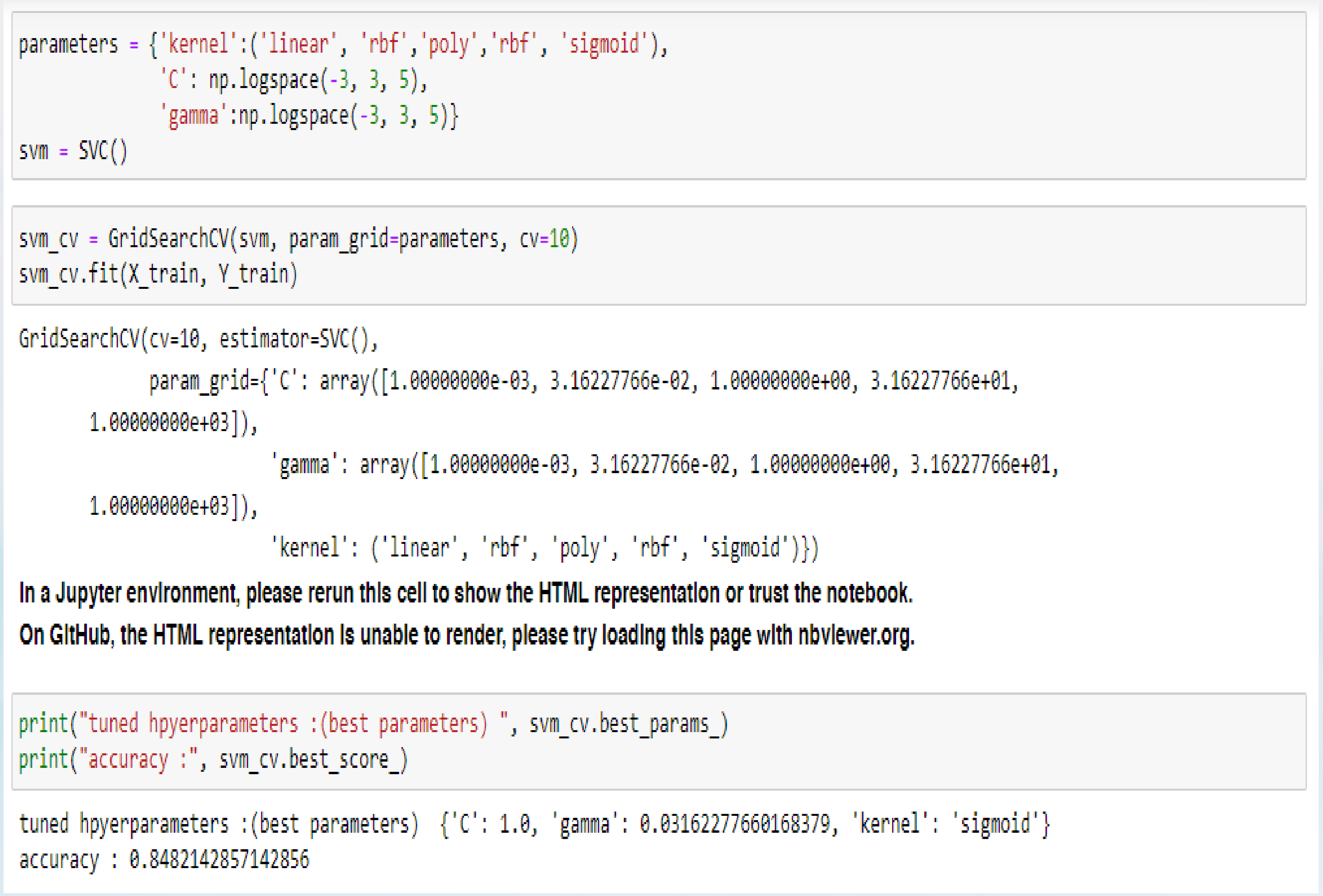
```
print('Accuracy on test data = ', logreg_cv.score(X_test,Y_test))
```

```
Accuracy on test data =  0.8333333333333334
```



IBM Developer SKILLS NETWORK

Predictive Analysis (Classification) results – 1. Accuracy of the models – Support Vector Machine



```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

```
svm_cv = GridSearchCV(svm, param_grid=parameters, cv=10)
svm_cv.fit(X_train, Y_train)
```

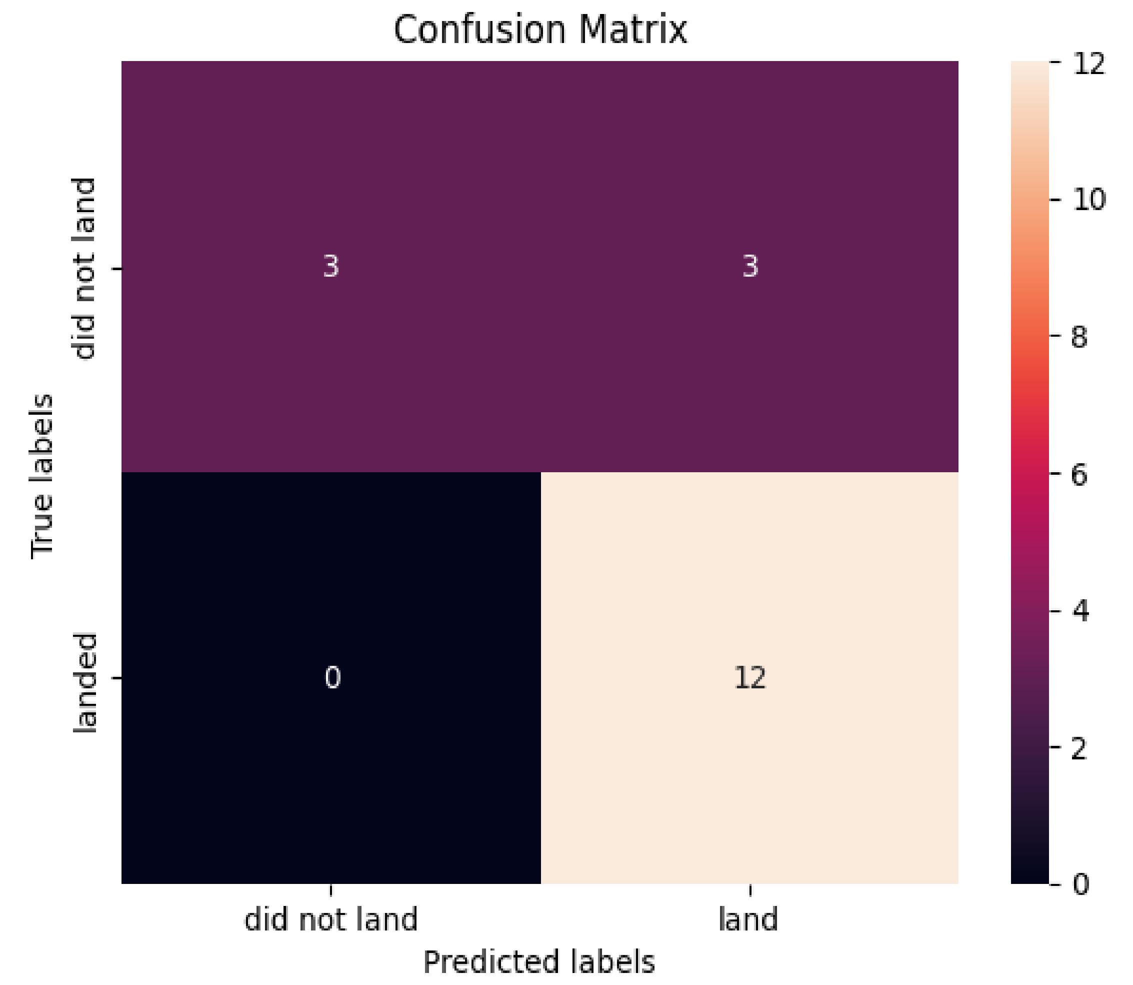
```
GridSearchCV(cv=10, estimator=SVC(),
             param_grid={'C': array([1.00000000e-03, 3.16227766e-02, 1.00000000e+00, 3.16227766e+01,
       1.00000000e+03]),
                         'gamma': array([1.00000000e-03, 3.16227766e-02, 1.00000000e+00, 3.16227766e+01,
       1.00000000e+03]),
                         'kernel': ('linear', 'rbf', 'poly', 'rbf', 'sigmoid')})
```

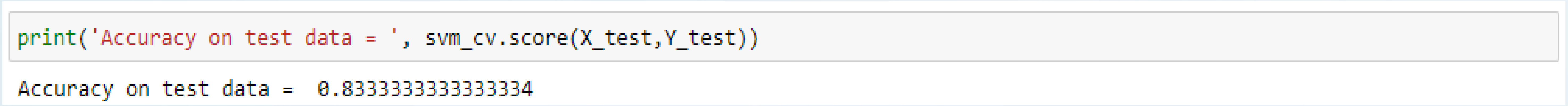
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
print("tuned hpyerparameters :(best parameters) ", svm_cv.best_params_)
print("accuracy :", svm_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
accuracy : 0.8482142857142856
```

```
yhat=svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```

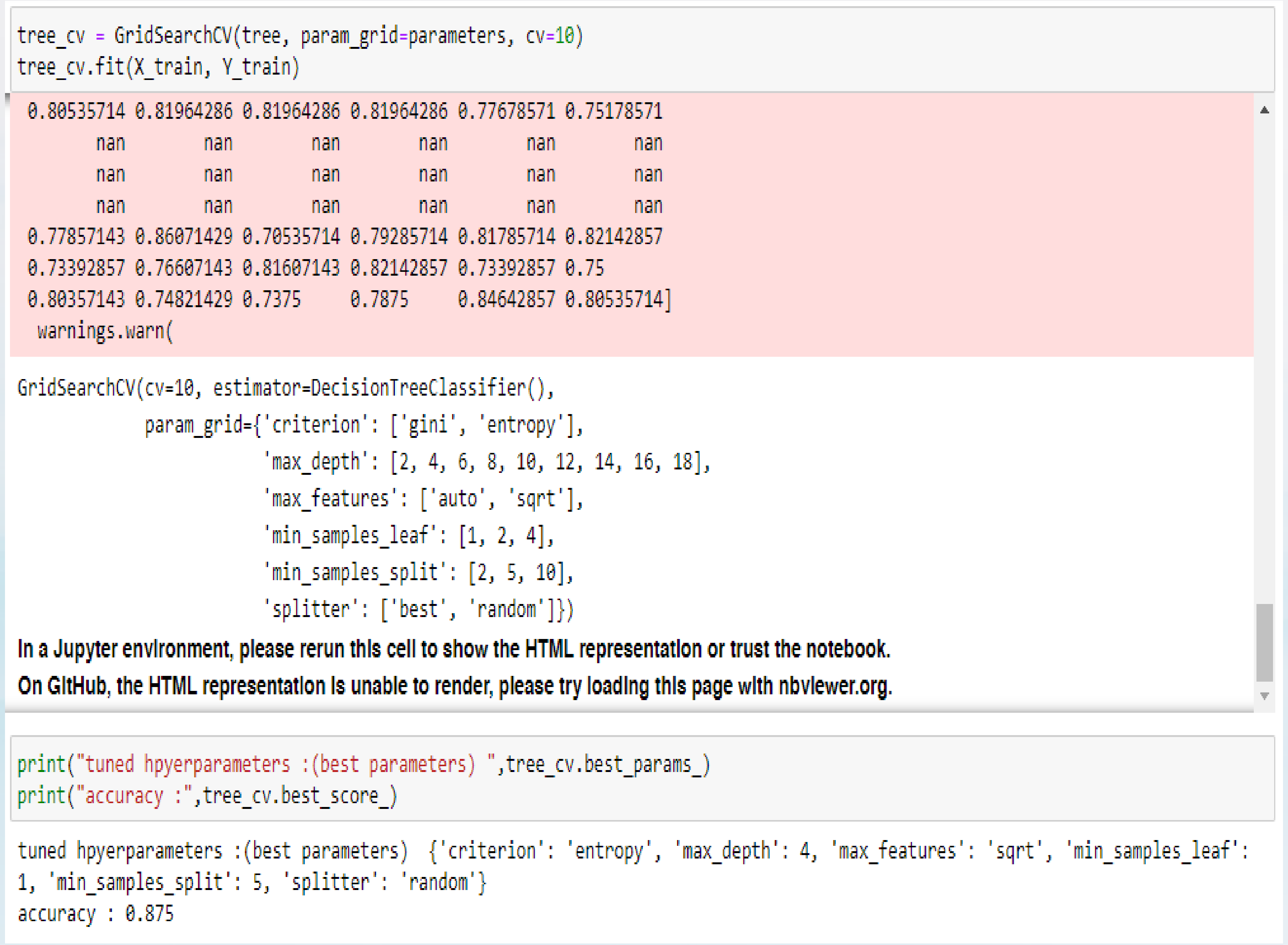




```
print('Accuracy on test data = ', svm_cv.score(X_test,Y_test))
```

```
Accuracy on test data =  0.8333333333333334
```

Predictive Analysis (Classification) results – 1. Accuracy of the models – Decision Tree Classifier



```
tree_cv = GridSearchCV(tree, param_grid=parameters, cv=10)
tree_cv.fit(X_train, Y_train)
```

```
 0.80535714 0.81964286 0.81964286 0.81964286 0.77678571 0.75178571
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.77857143 0.86071429 0.70535714 0.79285714 0.81785714 0.82142857
 0.73392857 0.76607143 0.81607143 0.82142857 0.73392857 0.75
 0.80357143 0.74821429 0.7375     0.7875     0.84642857 0.80535714]
  warnings.warn(
```

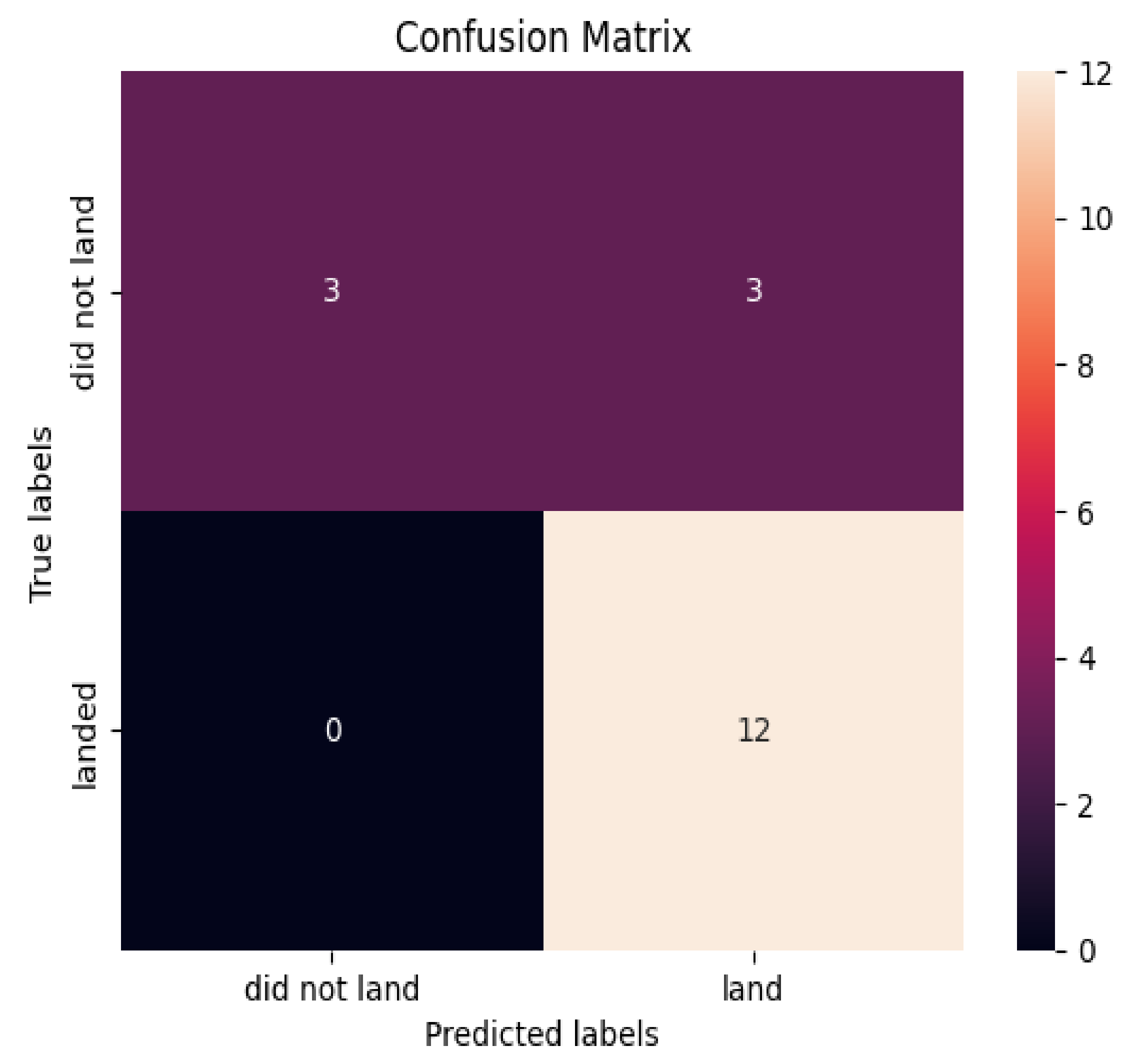
```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18],
                         'max_features': ['auto', 'sqrt'],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'splitter': ['best', 'random']})
```

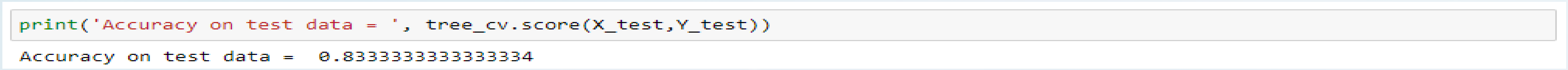
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
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
```

```
yhat = tree_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```

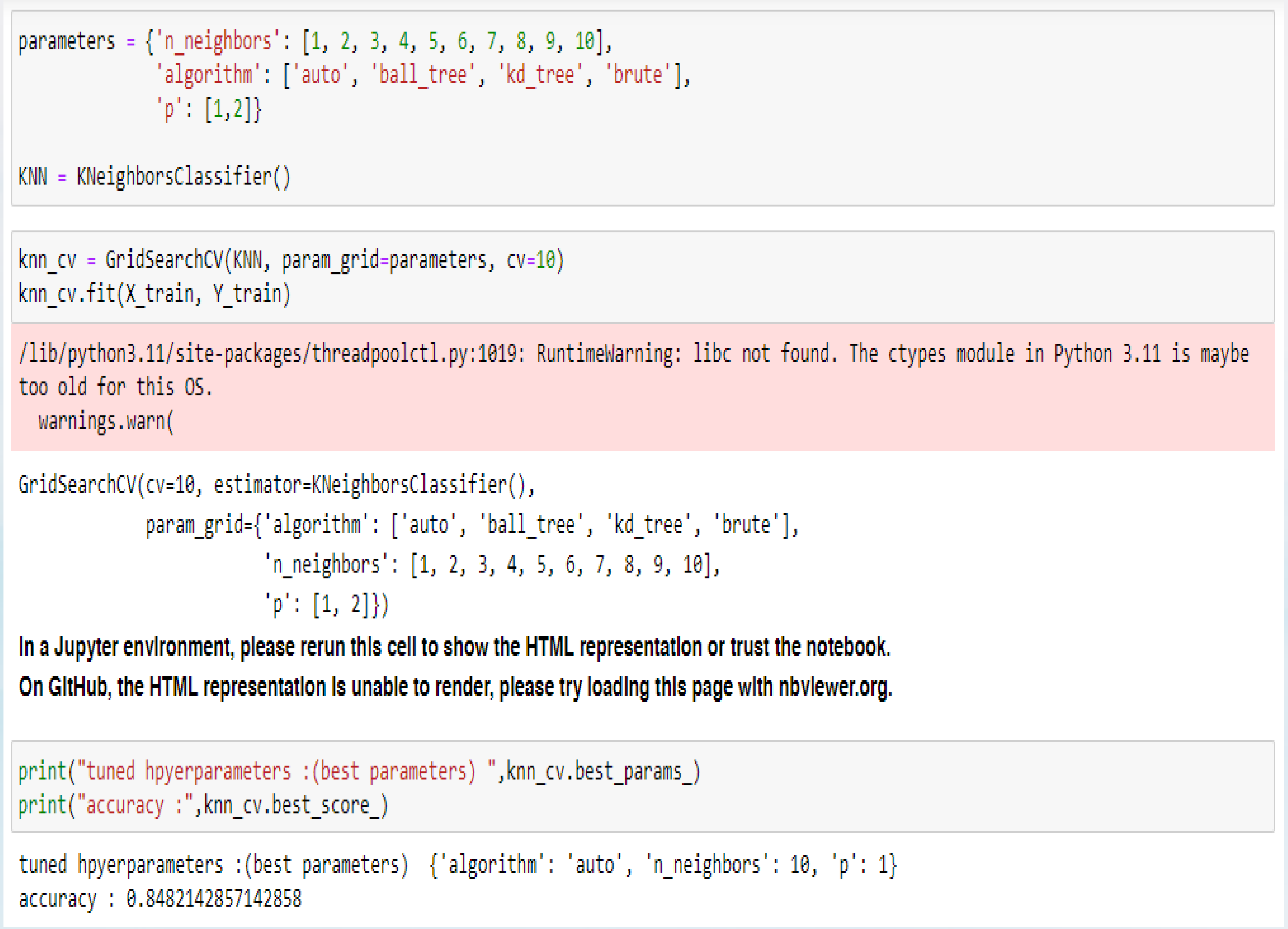




```
print('Accuracy on test data = ', tree_cv.score(X_test,Y_test))
```

```
Accuracy on test data =  0.8333333333333334
```

Predictive Analysis (Classification) results – 1. Accuracy of the models – KNN Classifier



```
parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'p': [1,2]}

KNN = KNeighborsClassifier()
```

```
knn_cv = GridSearchCV(KNN, param_grid=parameters, cv=10)
knn_cv.fit(X_train, Y_train)
```

```
/lib/python3.11/site-packages/threadpoolctl.py:1019: RuntimeWarning: libc not found. The ctypes module in Python 3.11 is maybe
too old for this OS.
  warnings.warn(
```

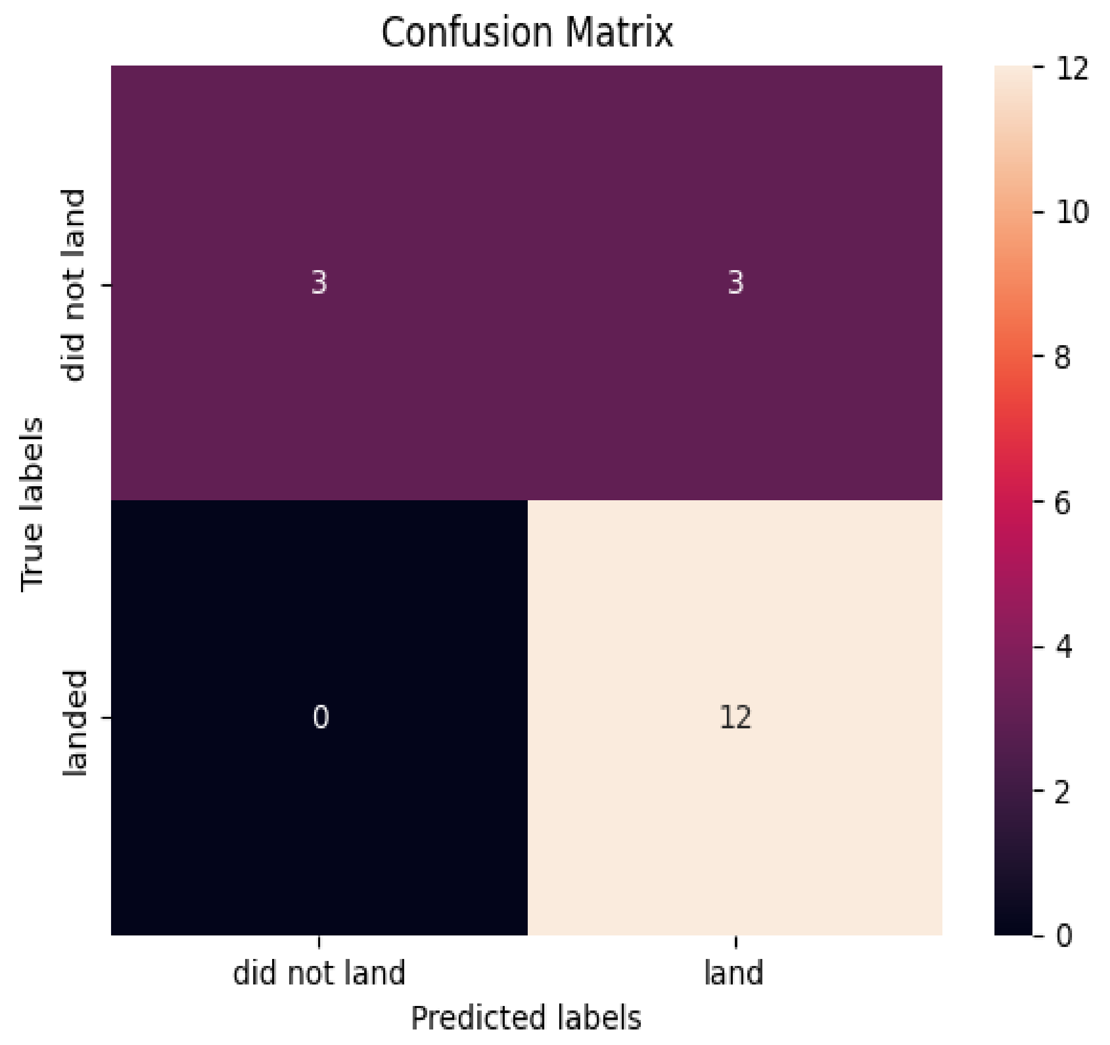
```
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                         'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'p': [1, 2]})
```

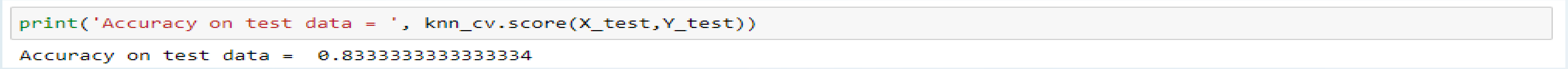
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
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
```

```
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```

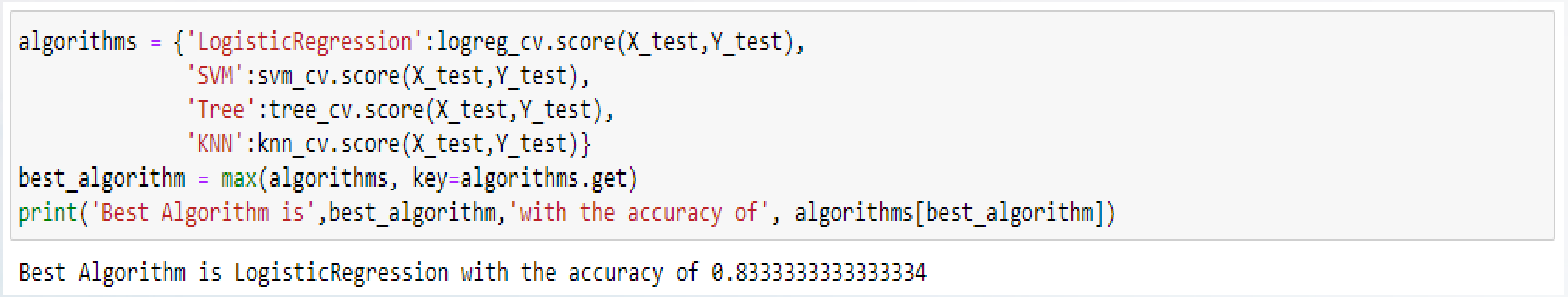




```
print('Accuracy on test data = ', knn_cv.score(X_test,Y_test))
```

```
Accuracy on test data =  0.8333333333333334
```

Predictive Analysis (Classification) results – 2. Best Model Selection



```
algorithms = {'LogisticRegression':logreg_cv.score(X_test,Y_test),
              'SVM':svm_cv.score(X_test,Y_test),
              'Tree':tree_cv.score(X_test,Y_test),
              'KNN':knn_cv.score(X_test,Y_test)}
best_algorithm = max(algorithms, key=algorithms.get)
print('Best Algorithm is',best_algorithm,'with the accuracy of', algorithms[best_algorithm])
```

```
Best Algorithm is LogisticRegression with the accuracy of 0.8333333333333334
```

As you can see, our accuracy is extremely close, but we do have a winner, even if it's down to decimal places! After selecting the best hyper-parameters for the logistic regression classifier using the validation data, we achieved 83.33% accuracy on the test data. The logistic regression classifier is the model with the highest classification accuracy. All the models performed at about the same level and had similar scores and accuracy, likely due to the small dataset. The logistic regression model slightly outperformed the rest when looking at the best score. A confusion matrix summarizes the performance of a classification algorithm, and in this case, all the confusion matrices were identical. However, the presence of false positives (Type 1 error) is not ideal.

Conclusion

Based on our research and analysis, we can conclude the following key points:

- The logistic regression classifier demonstrated the highest classification accuracy, outperforming other models slightly.
- Launch sites near the equator benefit from the Earth's rotational speed, providing a natural boost that reduces the need for additional fuel and boosters.
- All launch sites are situated close to the coast, which is a common feature.
- The overall launch success rate has shown an increasing trend over time.
- KSC LC-39A stands out with the highest success rate among launch sites, achieving a 100% success rate for launches with payloads under 5,500 kg.
- Orbits such as ES-L1, GEO, HEO, and SSO have maintained a 100% success rate.
- There is a positive correlation between higher payload mass and success rates across all launch sites.
- To improve predictive analytics, a larger dataset is essential to ensure the findings are generalizable to a broader context.
- Additional feature analysis or principal component analysis (PCA) should be considered to further enhance accuracy.
- Exploring the XGBoost model, which was not included in this study, could provide insights into its performance relative to other classification models.
- Lower-weighted payloads tend to perform better than heavier payloads in terms of success rates.
- SpaceX launch success rates have improved over time, indicating potential for near-perfect launches in the future.
- KSC LC-39A consistently had the most successful launches among all sites.
- Orbits GEO, HEO, SSO, and ES-L1 achieved the highest success rates, making them reliable choices for missions.

Thank you

IBM Developer
SKILLS NETWORK