IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

shima pilehvari

19/02/2024

Outline

- Executive Summary
- Introduction
- Methodology
- Results (inside of each section)
- Conclusion

Executive Summary

1)Summary of methodologies

- Data was gathered through the SpaceX REST API and by employing web scraping techniques.
- The data was then cleaned to establish a variable indicating the outcome of each launch as a success or failure.
- Various data visualization methods were used to examine factors such as payload, launch site, flight number, and trends over time.
- Using SQL, data analysis was conducted to calculate metrics like total payload, the payload range for successful missions, and the count of successful versus failed launches.
- The research further explored the success rates of different launch sites and their locations relative to key geographical features.
- Visualization techniques highlighted the most successful launch sites and the range of payloads for successful missions.
- Predictive models, including logistic regression, support vector machine (SVM), decision trees, and K-nearest neighbors (KNN), were developed to forecast the outcomes of landing

2) Summary of all results

- Findings from Exploratory Data Analysis:
- The rate of launch success has seen an improvement over time
- Among various launch sites, KSC LC-39A has emerged with the highest rate of successful landings.
- Orbits such as ES-L1, GEO, HEO, and SSO have achieved a 100% success rate.
- Executive Summary of Insights and Predictive Analysis:
- Visualization and analysis reveal that most launch sites are positioned near the equator and are proximate to coastal areas.
- In predictive analysis, all models demonstrated comparable performance on the test data, with the decision(DT) tree model marginally outperforming others

Introduction

Project background and context:

SpaceX, a pioneering force in aerospace, is committed to making space exploration accessible to all. Its notable achievements include dispatching spacecraft to the International Space Station, deploying a satellite network for global internet coverage, and conducting crewed space missions. The affordability of SpaceX launches, priced at approximately $62 million per launch, is attributed to its innovative reuse of Falcon 9's first stage – a feat not matched by competitors, whose launches can exceed $165 million due to the lack of reuse capability. The ability to predict the successful landing of the first stage is crucial for estimating launch costs. Utilizing publicly available data and machine learning techniques, we aim to forecast whether SpaceX or its rivals will be able to recover the first stage for reuse.

Investigation Areas:

- Examining the impact of factors such as payload mass, launch site, flight frequency, and orbit types on the success of first-stage landings.

- Trends in landing success rates over the years.

- Identifying the most effective prediction model for forecasting successful landings, a key aspect of binary classification.

Section 1

Methodology

Methodology

Executive Summary

- To analyze the factors influencing successful rocket landings, the study employed the following condensed approach:
- Data was collected via SpaceX REST API and web scraping to compile detailed datasets.
- Data wrangling prepared the data by filtering, handling missing values, and applying one-hot encoding.
- Exploratory Data Analysis (EDA) was used for data exploration, with SQL for queries and various visualization tools for insights.
- Folium and Plotly Dash visualized data through maps and interactive web apps, respectively, to highlight key patterns.
- Predictive models forecasted landing outcomes using classification algorithms, with fine-tuning and evaluation to identify the most effective model and parameters.

Data Collection

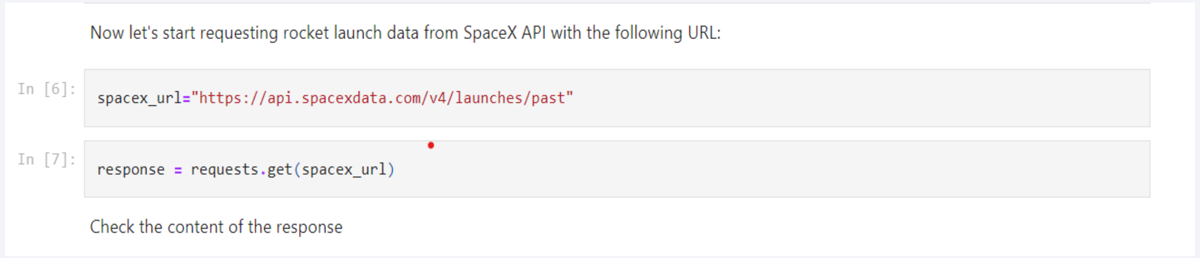
Procedures for data collection :

- Retrieve rocket launch data from the SpaceX API.
- Utilize .json() for decoding and .Json normalize() to transform into a data frame.
- Fetch details on launches via SpaceX API through specialized functions.
- Construct a dictionary based on the gathered data.
- Generate a data frame from this dictionary.
- Narrow down the data frame to include solely Falcon 9 launches.
- Fill in missing Payload Mass values with the average calculated using .mean().
- Save the compiled data into a CSV file

Data Collection – SpaceX API

Process Overview:

- Initiate a GET request to retrieve data on rocket launches via the API.
- Employ the 'json_normalize' function to transform the 'json' output into a DataFrame.
- Undertake data cleaning activities and address any missing values accordingly.

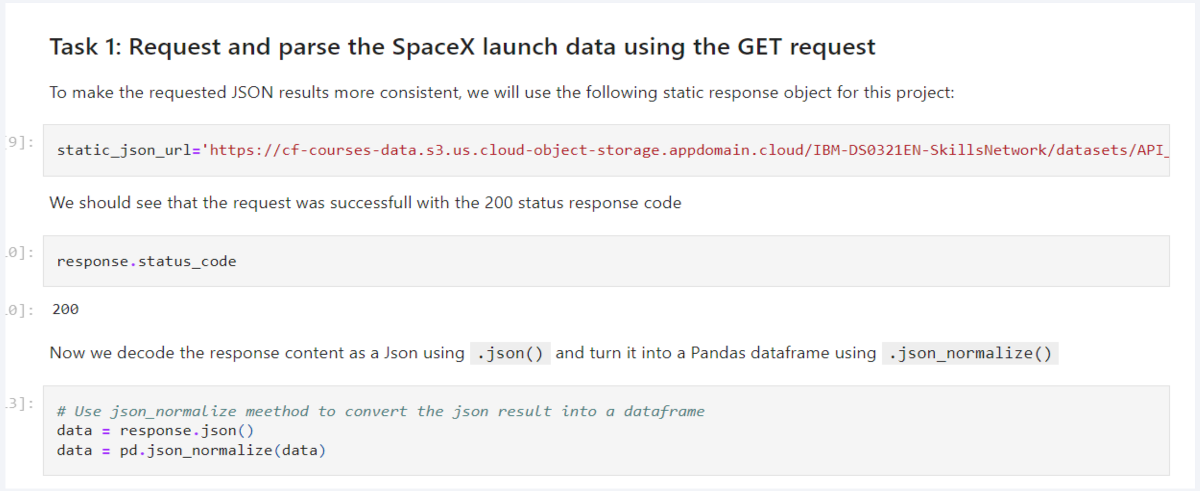


Now let's start requesting rocket launch data from SpaceX API with the following URL:

```
In [6]: spacex_url="https://api.spacexdata.com/v4/launches/past"
```

```
In [7]: response = requests.get(spacex_url)
```

Check the content of the response



Task 1: Request and parse the SpaceX launch data using the GET request

To make the requested JSON results more consistent, we will use the following static response object for this project:

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

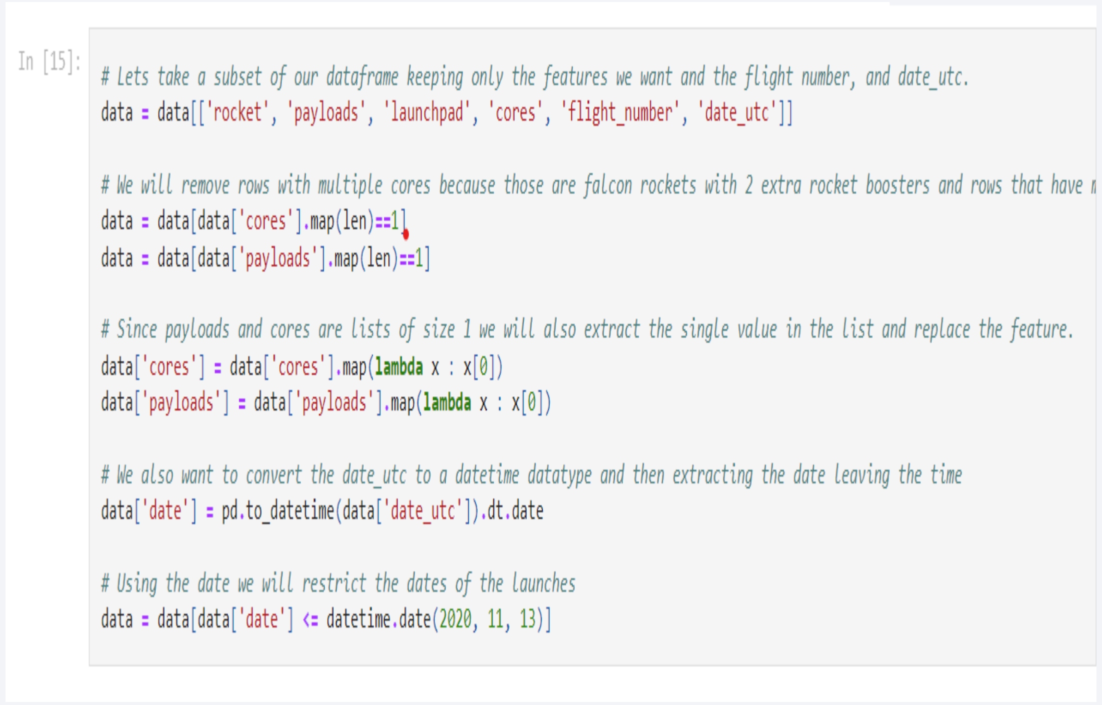
We should see that the request was successfull with the 200 status response code

```
0]: response.status_code
```

```
0]: 200
```

Now we decode the response content as a Json using `.json()` and turn it into a Pandas dataframe using `.json_normalize()`

```
3]: # Use json_normalize meethod to convert the json result into a dataframe
    data = response.json()
    data = pd.json_normalize(data)
```



```
In [15]: # Lets take a subset of our dataframe keeping only the features we want and the flight number, and date_utc.
         data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]

         # We will remove rows with multiple cores because those are falcon rockets with 2 extra rocket boosters and rows that have
         data = data[data['cores'].map(len)==1]
         data = data[data['payloads'].map(len)==1]

         # Since payloads and cores are lists of size 1 we will also extract the single value in the list and replace the feature.
         data['cores'] = data['cores'].map(lambda x : x[0])
         data['payloads'] = data['payloads'].map(lambda x : x[0])

         # We also want to convert the date_utc to a datetime datatype and then extracting the date leaving the time
         data['date'] = pd.to_datetime(data['date_utc']).dt.date

         # Using the date we will restrict the dates of the launches
         data = data[data['date'] <= datetime.date(2020, 11, 13)]
```

Data Collection - Scraping

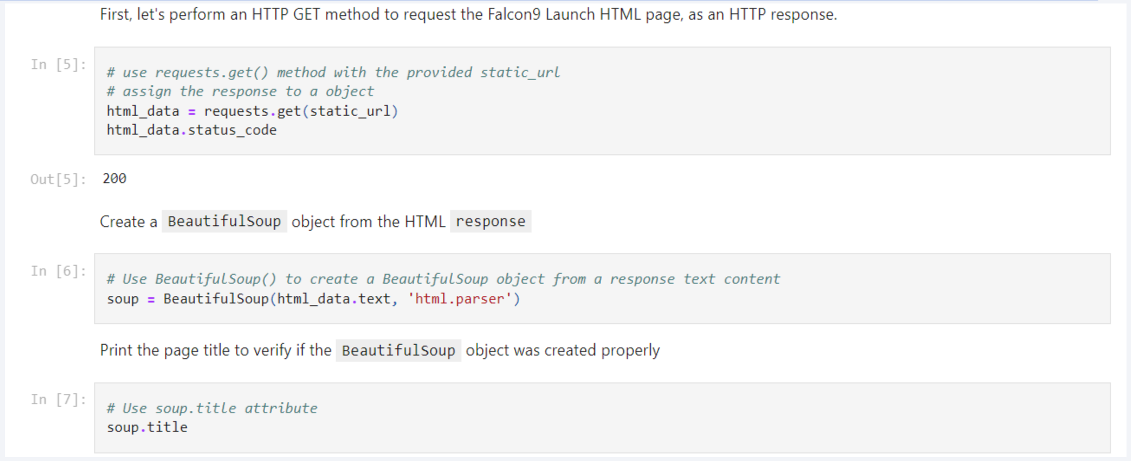
Web Scraping for Data Acquisition:

- A common method to gather data on Falcon 9 launches involves web scraping from associated Wikipedia pages. To accomplish this, we will utilize the Beautiful Soup package in Python, focusing on extracting vital information from HTML tables that hold Falcon 9 launch records.

- Retrieve the Falcon 9 launch information from the specified URL.

- Generate a Beautiful Soup object using the HTML content received.

- Identify and extract the names of all columns/variables from the HTML headers.



First, let's perform an HTTP GET method to request the Falcon9 Launch HTML page, as an HTTP response.

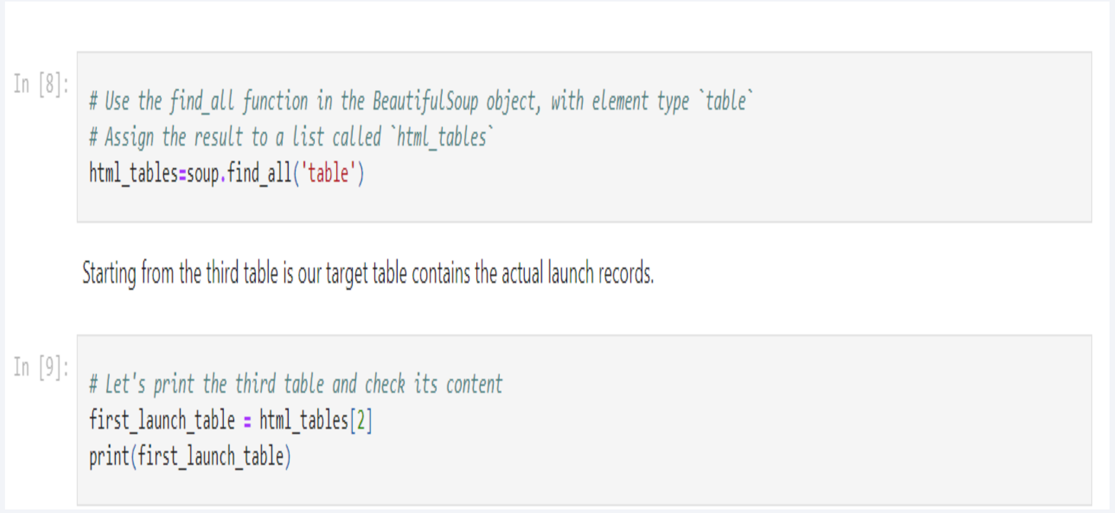
```
In [5]:  # use requests.get() method with the provided static_url
         # assign the response to a object
         html_data = requests.get(static_url)
         html_data.status_code

Out[5]:  200
```

Create a `BeautifulSoup` object from the HTML `response`

```
In [6]:  # Use BeautifulSoup() to create a BeautifulSoup object from a response text content
         soup = BeautifulSoup(html_data.text, 'html.parser')
```

Print the page title to verify if the `BeautifulSoup` object was created properly

```
In [7]:  # Use soup.title attribute
         soup.title
```

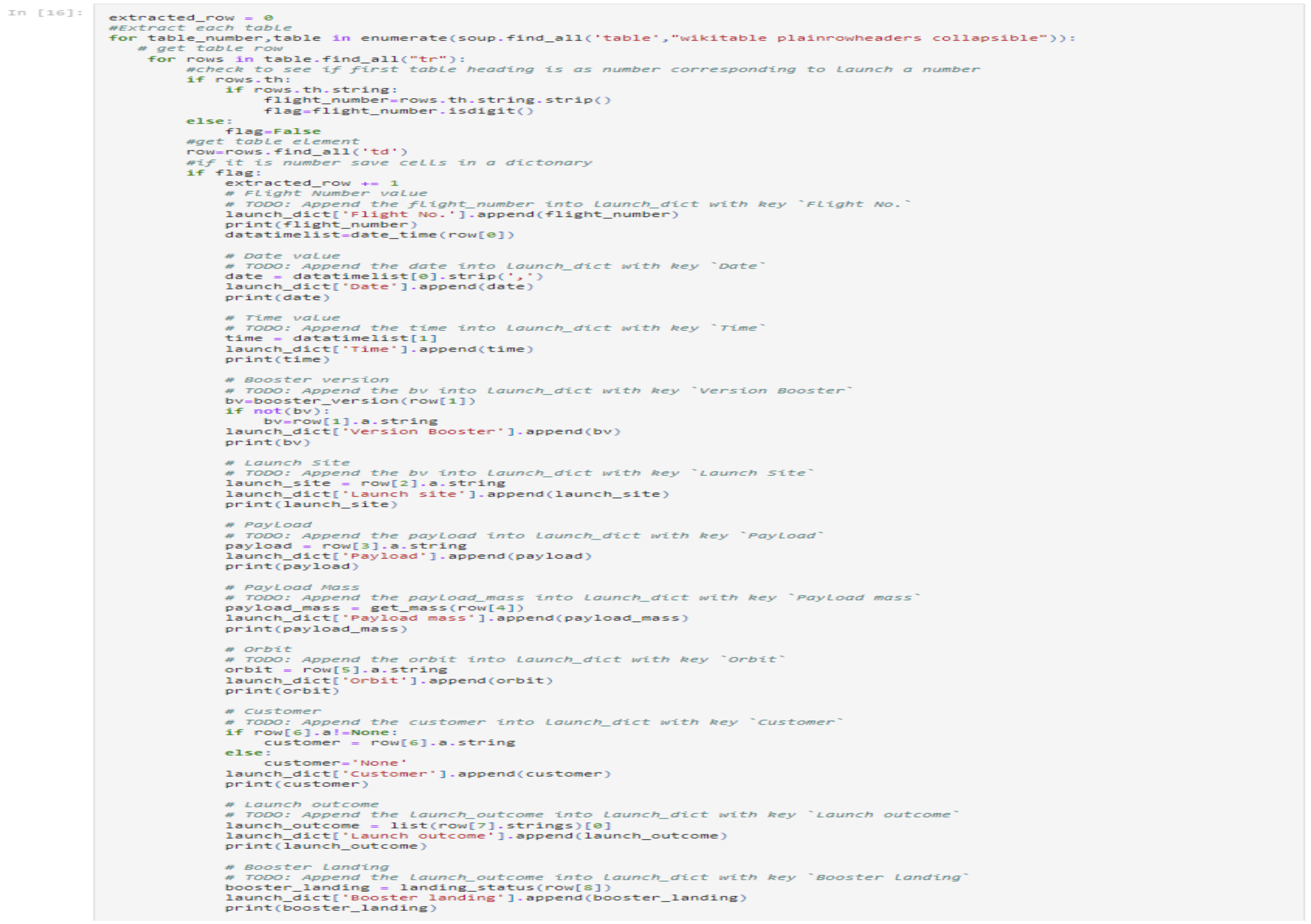


```
In [8]:  # Use the find_all function in the BeautifulSoup object, with element type `table`
         # Assign the result to a list called `html_tables`
         html_tables=soup.find_all('table')
```

Starting from the third table is our target table contains the actual launch records.

```
In [9]:  # Let's print the third table and check its content
         first_launch_table = html_tables[2]
         print(first_launch_table)
```

Data Collection - Scraping



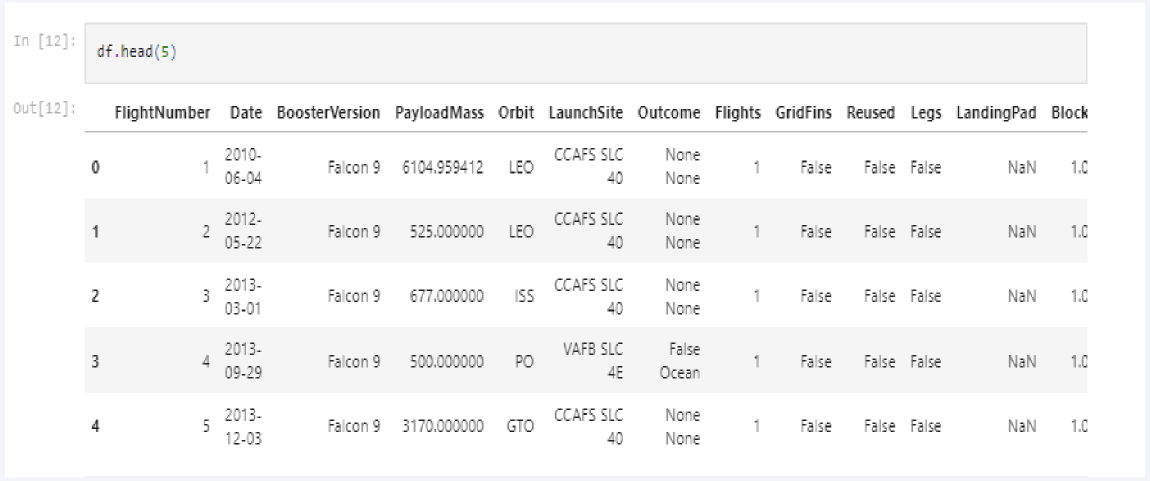
```
In [16]:
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders collapsible")):
   # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number corresponding to launch a number
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
        else:
            flag=False
        #get table element
        row=rows.find_all('td')
        #if it is number save cells in a dictonary
        if flag:
            extracted_row += 1
            # Flight Number value
            # TODO: Append the flight_number into launch_dict with key `Flight No.`
            launch_dict['Flight No.'].append(flight_number)
            print(flight_number)
            datatimelist=date_time(row[0])

            # Date value
            # TODO: Append the date into launch_dict with key `Date`
            date = datatimelist[0].strip(',')
            launch_dict['Date'].append(date)
            print(date)

            # Time value
            # TODO: Append the time into launch_dict with key `Time`
            time = datatimelist[1]
            launch_dict['Time'].append(time)
            print(time)

            # Booster version
            # TODO: Append the bv into launch_dict with key `Version Booster`
            bv=booster_version(row[1])
            if not(bv):
                bv=row[1].a.string
            launch_dict['Version Booster'].append(bv)
            print(bv)

            # Launch Site
            # TODO: Append the bv into launch_dict with key `Launch Site`
            launch_site = row[2].a.string
            launch_dict['Launch site'].append(launch_site)
            print(launch_site)

            # Payload
            # TODO: Append the payload into launch_dict with key `Payload`
            payload = row[3].a.string
            launch_dict['Payload'].append(payload)
            print(payload)

            # Payload Mass
            # TODO: Append the payload_mass into launch_dict with key `Payload mass`
            payload_mass = get_mass(row[4])
            launch_dict['Payload mass'].append(payload_mass)
            print(payload_mass)

            # Orbit
            # TODO: Append the orbit into launch_dict with key `Orbit`
            orbit = row[5].a.string
            launch_dict['Orbit'].append(orbit)
            print(orbit)

            # Customer
            # TODO: Append the customer into launch_dict with key `Customer`
            if row[6].a!=None:
                customer = row[6].a.string
            else:
                customer='None'
            launch_dict['Customer'].append(customer)
            print(customer)

            # Launch outcome
            # TODO: Append the launch_outcome into launch_dict with key `Launch outcome`
            launch_outcome = list(row[7].strings)[0]
            launch_dict['Launch outcome'].append(launch_outcome)
            print(launch_outcome)

            # Booster landing
            # TODO: Append the launch_outcome into launch_dict with key `Booster landing`
            booster_landing = landing_status(row[8])
            launch_dict['Booster landing'].append(booster_landing)
            print(booster_landing)
```

Data Wrangling

Data Preparation and Analysis:

- Data preparation, or "Data Wrangling," involves the process of refining and consolidating complex and disorganized data sets to facilitate accessibility and Exploratory Data Analysis (EDA). Our initial step will involve determining the frequency of launches at each site, followed by assessing the number and frequency of mission outcomes across various orbit types. Subsequently, we will generate a "landing outcome" label derived from the outcome column to simplify subsequent analyses, visualizations, and machine learning processes. The final step will include exporting the organized data into a CSV file.

- The exploratory phase aims to uncover patterns within the data and establish appropriate labels for training supervised learning models. Within our dataset, multiple instances indicate unsuccessful booster landings. In these scenarios, we will categorize the outcomes into Training Labels, where '1' denotes a successful booster landing and '0' signifies an unsuccessful attempt, as indicated by the 'Class' feature.



In [12]:

```
df.head(5)
```

Out[12]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0

Predictive Analysis (Classification)

Model Construction:

- Import the dataset into NumPy and Pandas.
- Process and split the data into training and test sets.
- Select the ML model type.
- Configure GridSearchCV with parameters and algorithms, then apply it to the dataset.

Model Evaluation:

- Assess each model's accuracy.

- Retrieve optimized hyperparameters for all algorithms.

- Generate the confusion matrix.

Model Enhancement:

- Implement Feature Engineering and Algorithm Tuning, utilizing grid search to identify optimal hyperparameters

Section 2

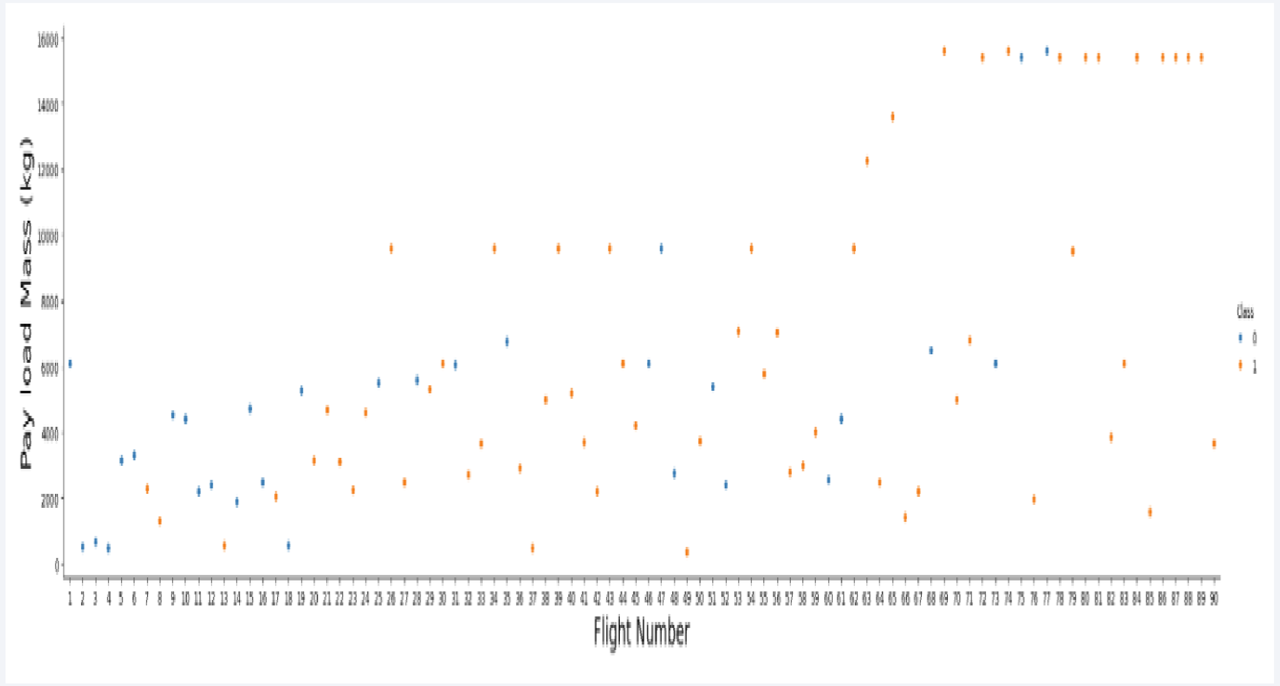
Insights drawn from EDA

EDA with Data Visualization

- We'll dive into Exploratory Data Analysis (EDA) by visualizing data to uncover patterns and relationships. Key focuses include comparisons between Flight Number, Payload Mass, Launch Site, Orbit Type, and annual launch success trends. We aim to highlight geographical details of launch sites, especially their locations relative to coastlines, railways, and highways, using visual maps to represent launch outcomes.

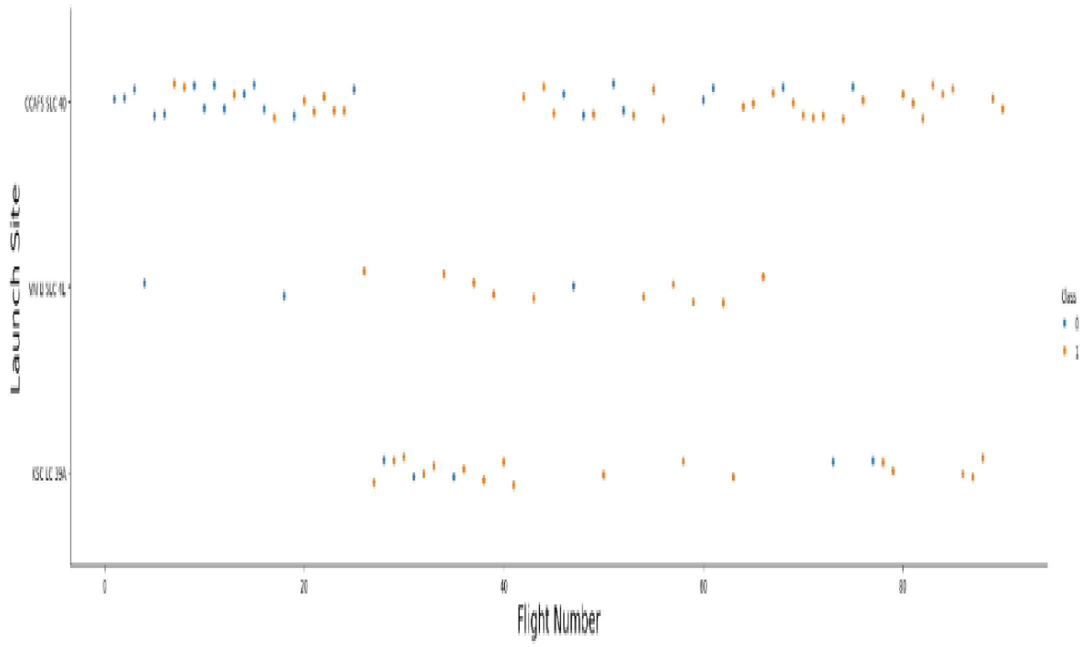
  Our primary tools for initial insights will be scatter plots, examining:

- Payload Mass vs. Flight Number
- Flight Number vs. Launch Site
- Orbit Type correlations with Flight Number and Payload Mass
- This streamlined approach helps us quickly identify significant data trends and relationships.



As observed in the scatter plot, flight numbers increase, successful first-stage landings become more likely. However, heavier payloads reduce this likelihood.

EDA with Data Visualization

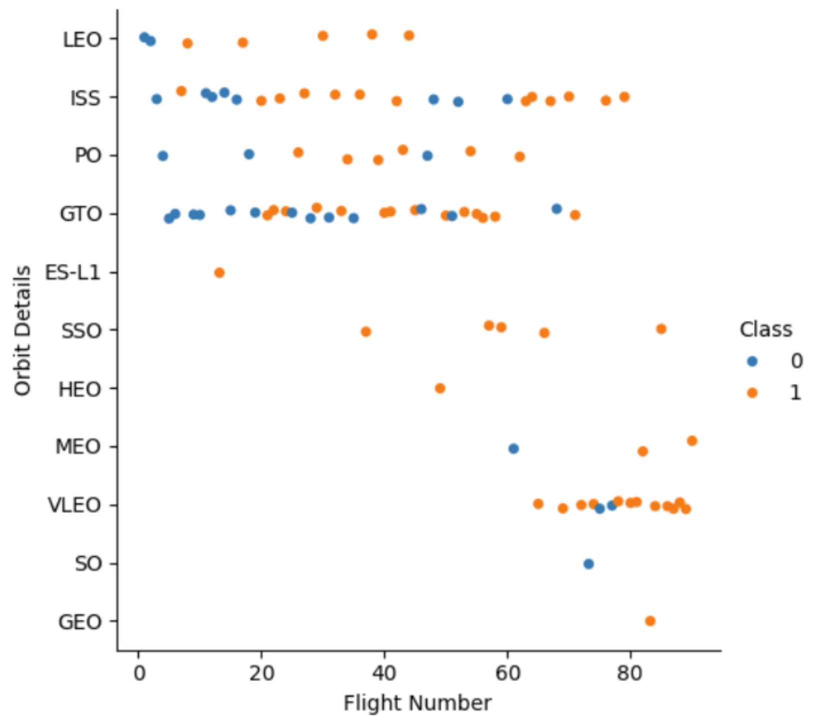


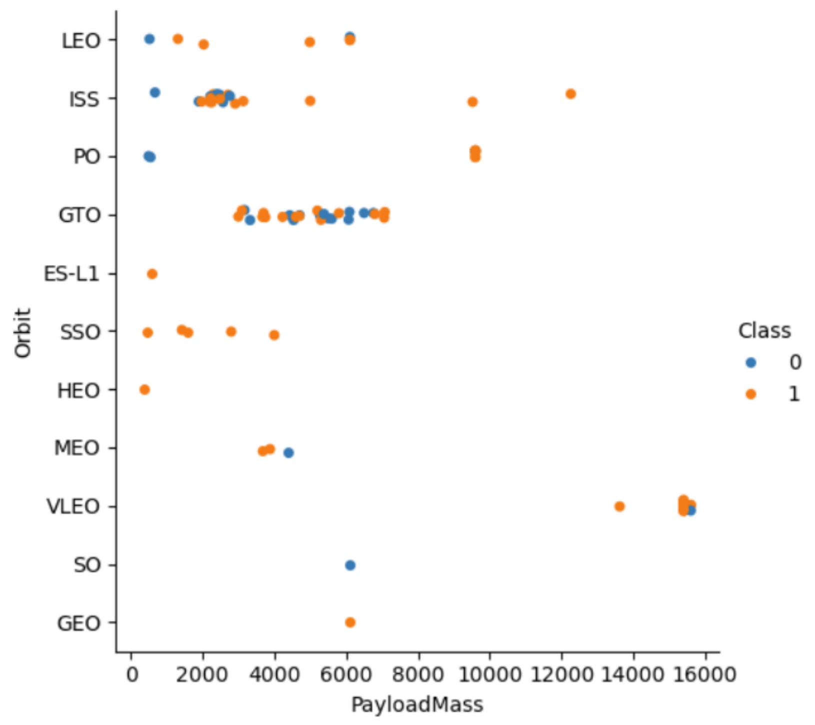
The analysis reveals that the launch site CCAFS LC-40 achieves a 60% success rate, whereas KSC LC-39A and VAFB SLC 4E both exhibit a higher success rate of 77%.

EDA with Data Visualization

we can observe that Success in LEO orbits increases with more flights, while in GTO orbits, flight number doesn't impact success.

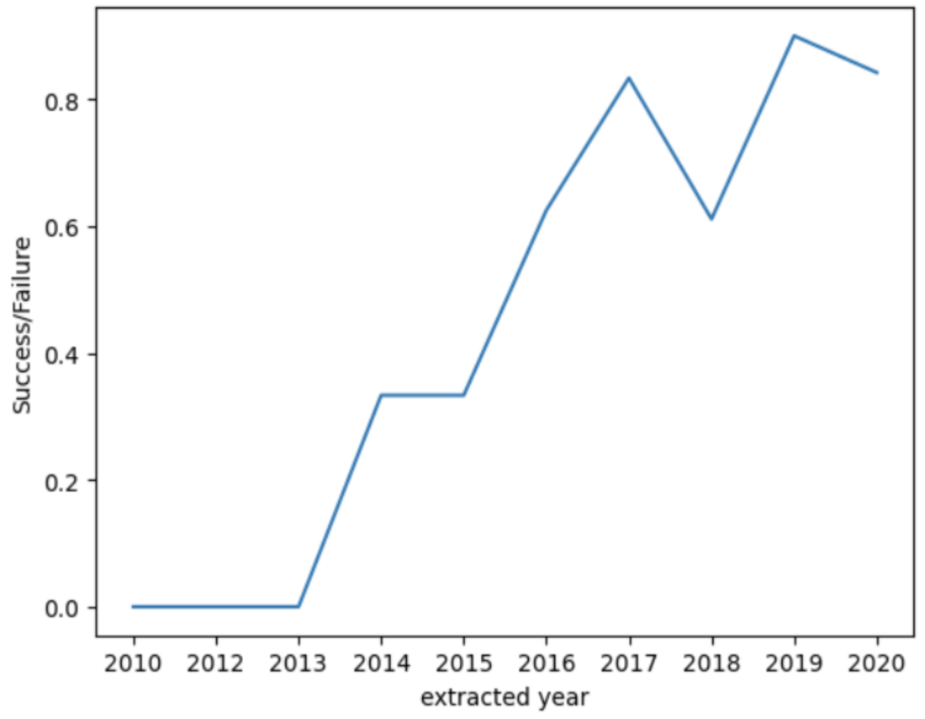
For Polar, LEO, and ISS orbits, heavier payloads correlate with higher successful landing rates. In contrast, for GTO orbits, success rates are mixed, with both successful and unsuccessful missions present, regardless of payload weight.





EDA with Data Visualization

Launch Success over Time



- The success rate see an increase between 2013-2017 and 2018-2019, while it declined from 2017-2018 and from 2019-2020. However, looking at the trend since 2013, there has been an overall improvement in the success rate.

EDA with SQL

Using SQL, we performed several analyses on our dataset, as follows:

- Listed launch site names.
- Showed 5 records for launch sites starting with 'CCA'.
- Calculated the total payload mass for NASA's CRS program launches.
- Determined the average payload mass for the F9 v1.1 booster version.
- Identified the date of the first successful ground pad landing.
- Listed boosters that successfully landed on drone ships with payloads between 4000 and 6000.
- Counted the successful and failed missions.
- Identified booster versions carrying the maximum payload.
- Listed failed drone ship landings in 2015, including booster versions and launch sites.
- Ranked landing outcomes by count, from 2010-06-04 to 2017-03-20, in descending order.

All launch site names

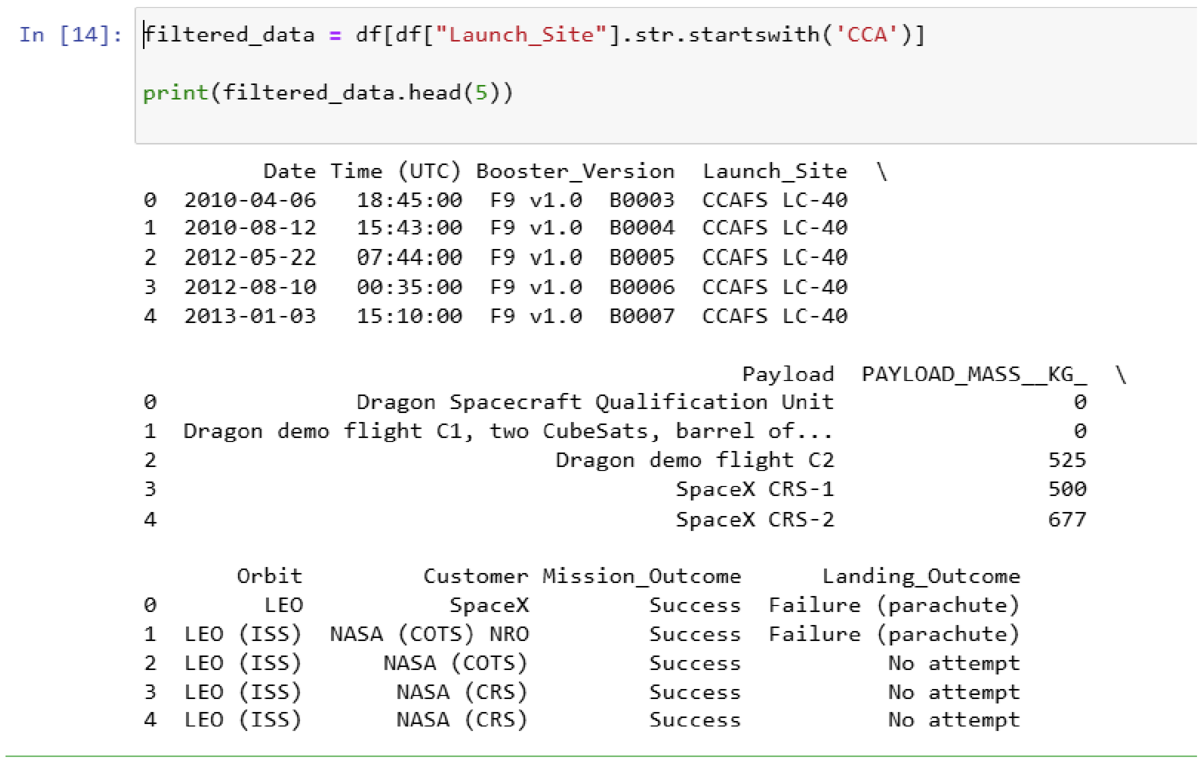
CCAFS LC-40

VAFB SLC-4E

KSC LC-39A

CCAFS SLC-40

Launch site names begin with `CCA



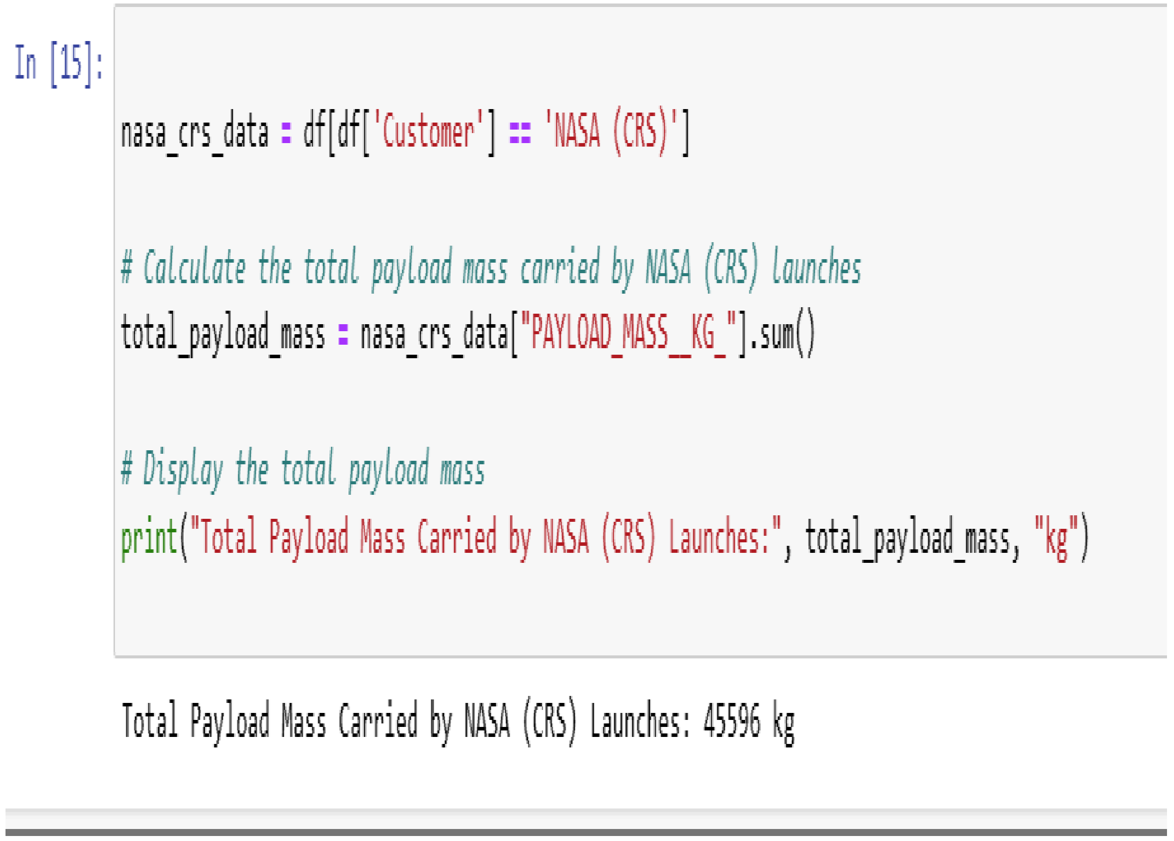
```
In [14]: filtered_data = df[df["Launch_Site"].str.startswith('CCA')]

         print(filtered_data.head(5))
```

```
         Date Time (UTC) Booster_Version  Launch_Site  \
0  2010-04-06   18:45:00  F9 v1.0  B0003  CCAFS LC-40
1  2010-08-12   15:43:00  F9 v1.0  B0004  CCAFS LC-40
2  2012-05-22   07:44:00  F9 v1.0  B0005  CCAFS LC-40
3  2012-08-10   00:35:00  F9 v1.0  B0006  CCAFS LC-40
4  2013-01-03   15:10:00  F9 v1.0  B0007  CCAFS LC-40

                                              Payload  PAYLOAD_MASS__KG_  \
0              Dragon Spacecraft Qualification Unit                  0
1  Dragon demo flight C1, two CubeSats, barrel of...                  0
2                              Dragon demo flight C2                525
3                                      SpaceX CRS-1                500
4                                      SpaceX CRS-2                677

      Orbit         Customer Mission_Outcome      Landing_Outcome
0       LEO           SpaceX         Success  Failure (parachute)
1  LEO (ISS)  NASA (COTS) NRO         Success  Failure (parachute)
2  LEO (ISS)      NASA (COTS)         Success           No attempt
3  LEO (ISS)       NASA (CRS)         Success           No attempt
4  LEO (ISS)       NASA (CRS)         Success           No attempt
```

CCAFS LC-40

CCAFS SLC-40

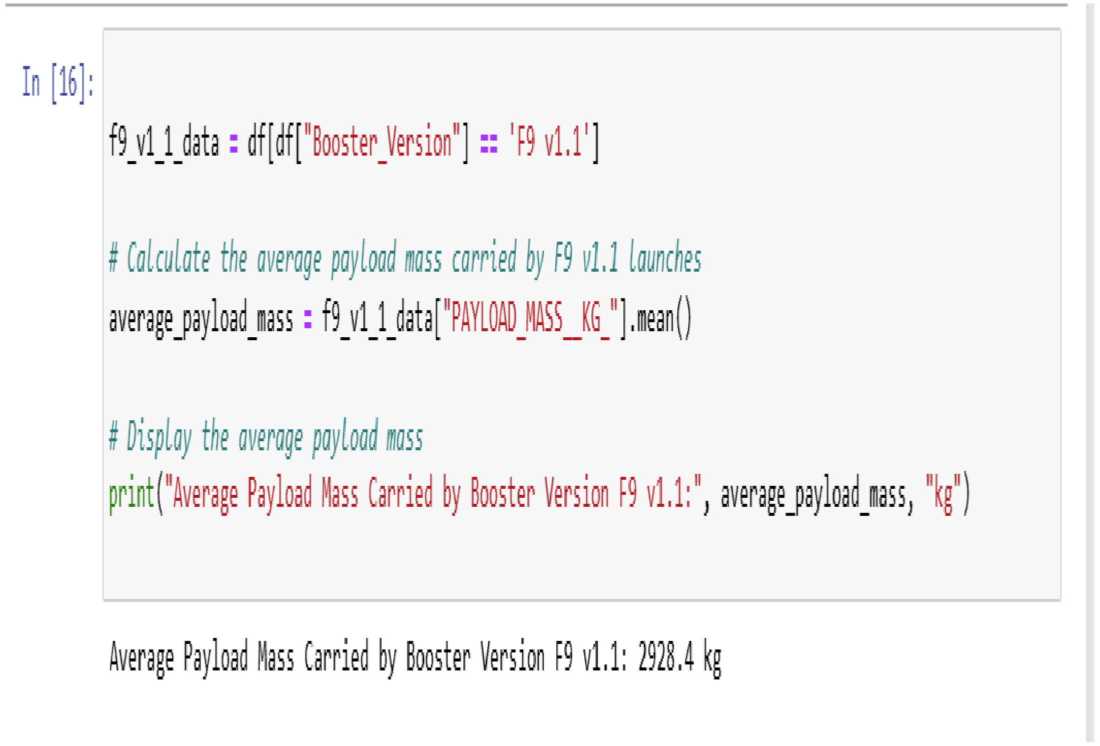
total payload carried by booster from NASA



```
In [15]:
nasa_crs_data = df[df['Customer'] == 'NASA (CRS)']

# Calculate the total payload mass carried by NASA (CRS) launches
total_payload_mass = nasa_crs_data["PAYLOAD_MASS__KG_"].sum()

# Display the total payload mass
print("Total Payload Mass Carried by NASA (CRS) Launches:", total_payload_mass, "kg")
```

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

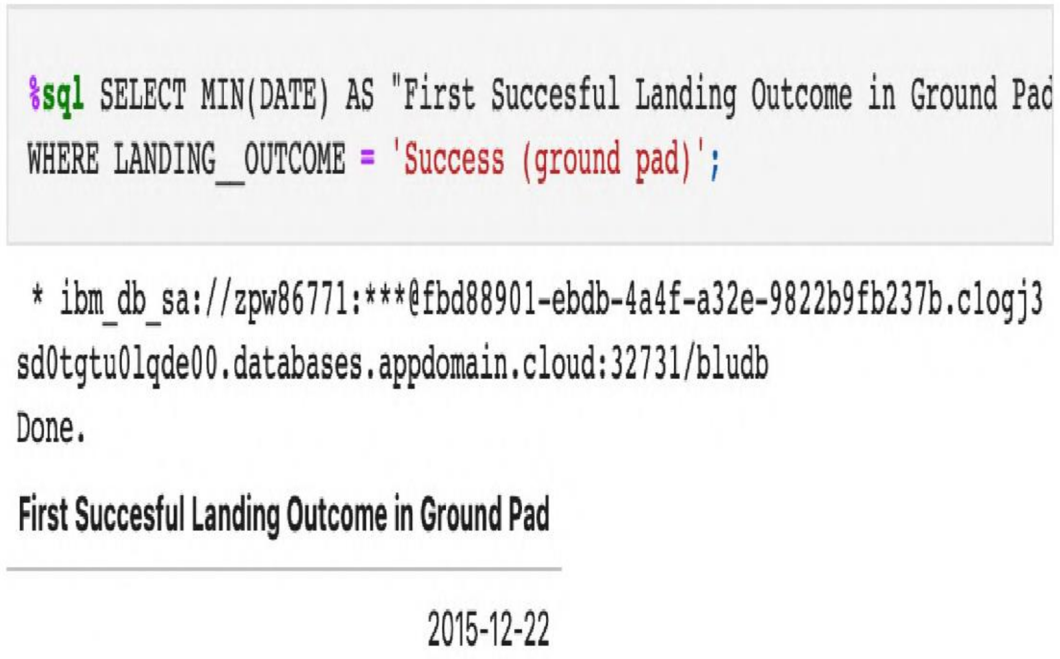
Average payload mass by F9 v1.1



```
In [16]:
f9_v1_1_data = df[df["Booster_Version"] == 'F9 v1.1']

# Calculate the average payload mass carried by F9 v1.1 launches
average_payload_mass = f9_v1_1_data["PAYLOAD_MASS__KG_"].mean()

# Display the average payload mass
print("Average Payload Mass Carried by Booster Version F9 v1.1:", average_payload_mass, "kg")
```

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

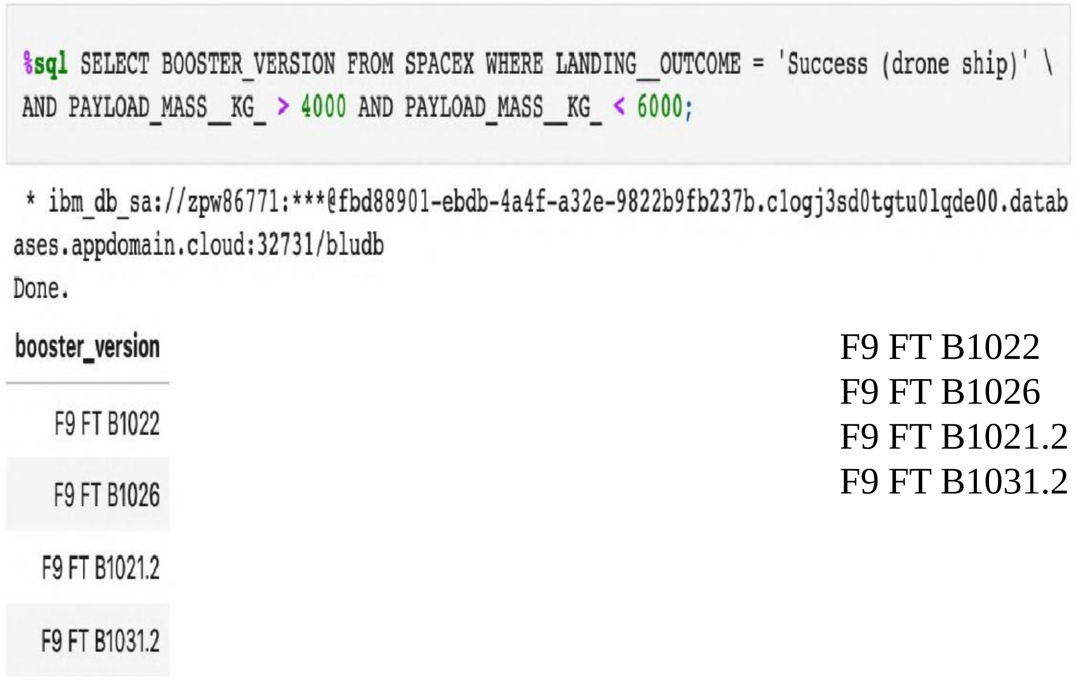
First successful ground landing date:



```
%sql SELECT MIN(DATE) AS "First Succesful Landing Outcome in Ground Pad
WHERE LANDING__OUTCOME = 'Success (ground pad)';
```

```
 * ibm_db_sa://zpw86771:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.clogj3
sd0tgtu0lqde00.databases.appdomain.cloud:32731/bludb
Done.
First Succesful Landing Outcome in Ground Pad
                                   2015-12-22
```

Successful drone ship landing with payload between 4000 and 6000:

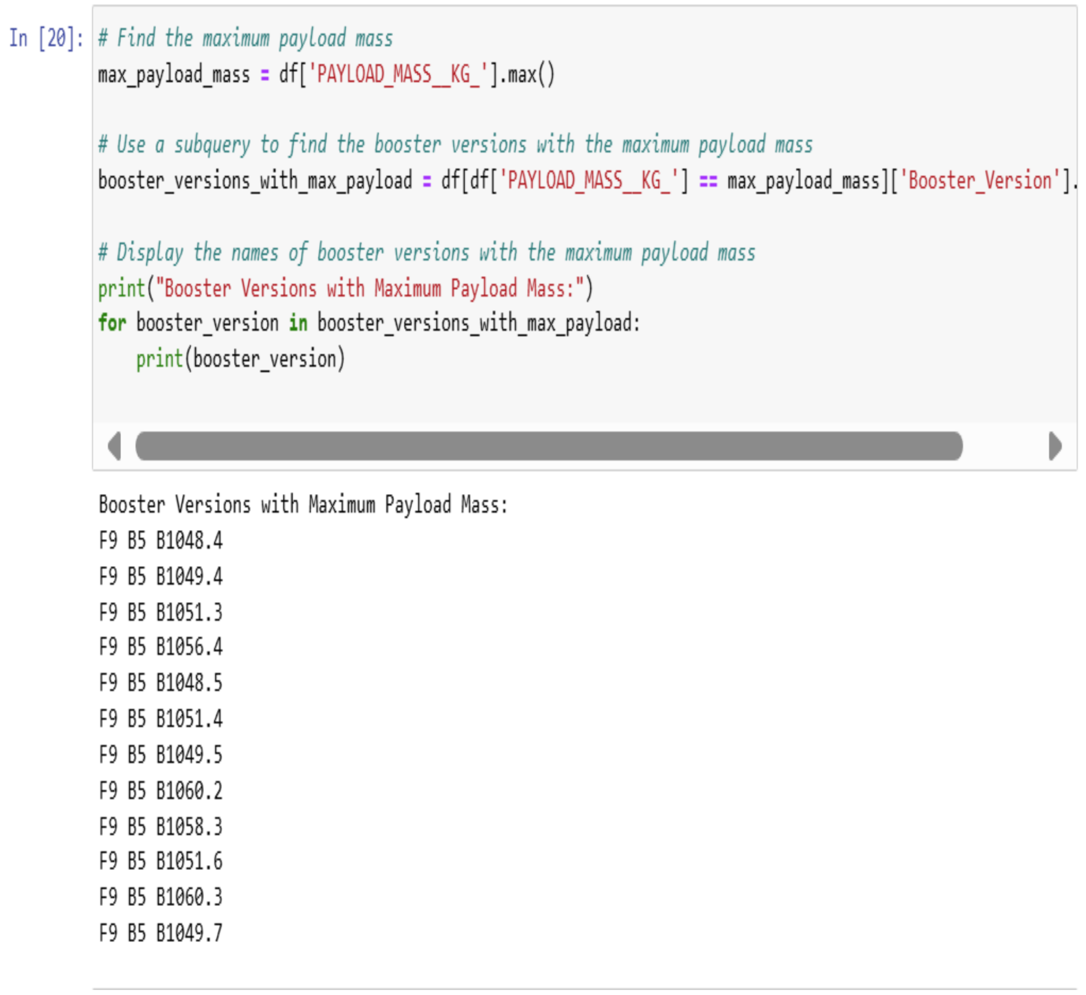


```
%sql SELECT BOOSTER_VERSION FROM SPACEX WHERE LANDING__OUTCOME = 'Success (drone ship)' \
AND PAYLOAD_MASS__KG_ > 4000 AND PAYLOAD_MASS__KG_ < 6000;
```

```
 * ibm_db_sa://zpw86771:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.clogj3sd0tgtu0lqde00.datab
ases.appdomain.cloud:32731/bludb
Done.
booster_version
    F9 FT B1022
    F9 FT B1026
  F9 FT B1021.2
  F9 FT B1031.2
```

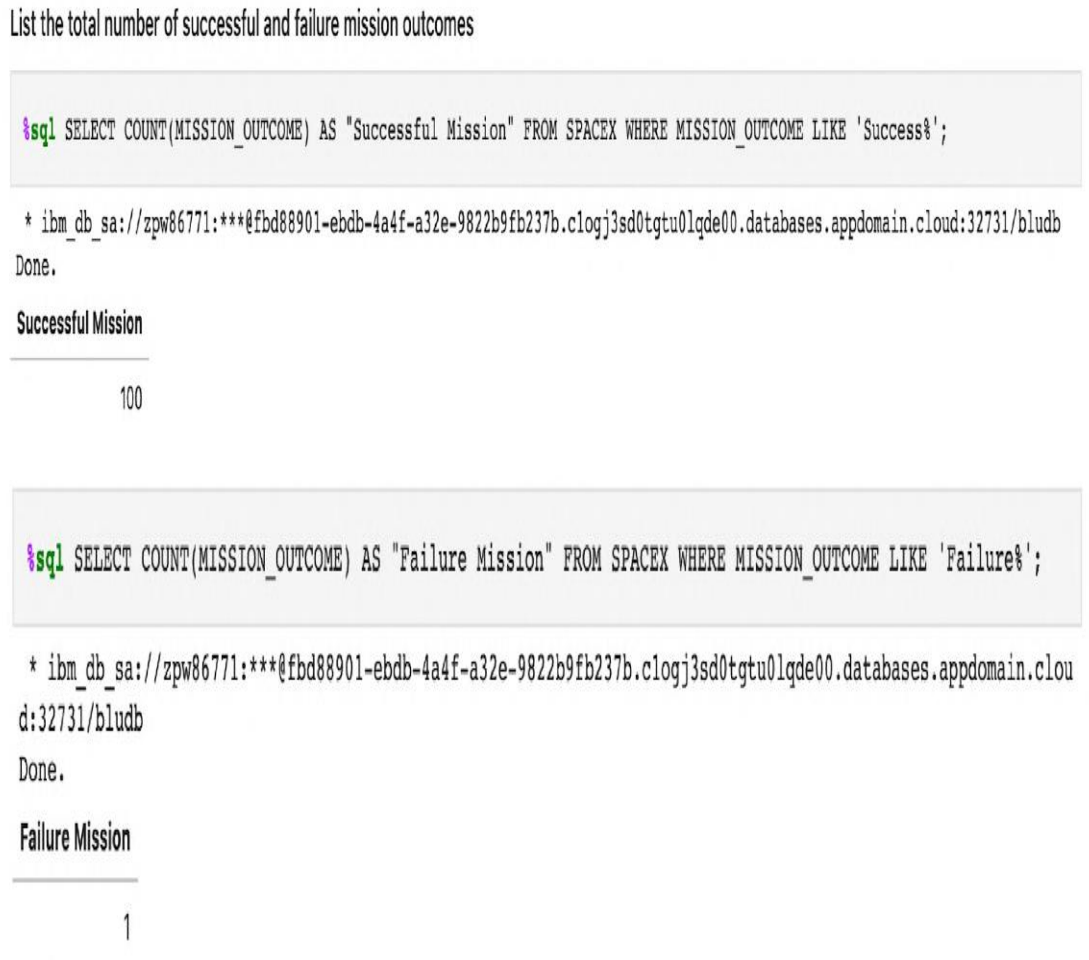
F9 FT B1022
F9 FT B1026
F9 FT B1021.2
F9 FT B1031.2

Total number of successful and failure mission outcomes and Boosters carried maximum payload



```
In [20]: # Find the maximum payload mass
max_payload_mass = df['PAYLOAD_MASS__KG_'].max()

# Use a subquery to find the booster versions with the maximum payload mass
booster_versions_with_max_payload = df[df['PAYLOAD_MASS__KG_'] == max_payload_mass]['Booster_Version'].

# Display the names of booster versions with the maximum payload mass
print("Booster Versions with Maximum Payload Mass:")
for booster_version in booster_versions_with_max_payload:
    print(booster_version)
```

```
Booster Versions with Maximum Payload Mass:
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
```



List the total number of successful and failure mission outcomes

```
%sql SELECT COUNT(MISSION_OUTCOME) AS "Successful Mission" FROM SPACEX WHERE MISSION_OUTCOME LIKE 'Success%';
```

```
 * ibm_db_sa://zpw86771:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:32731/bludb
Done.
```

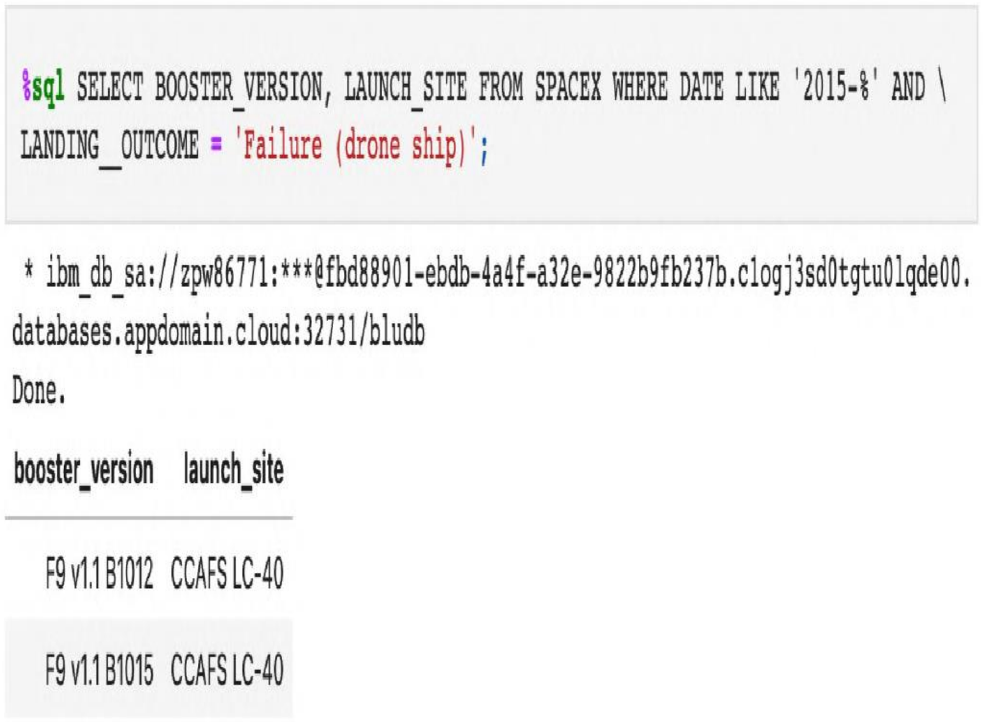
Successful Mission
100

```
%sql SELECT COUNT(MISSION_OUTCOME) AS "Failure Mission" FROM SPACEX WHERE MISSION_OUTCOME LIKE 'Failure%';
```

```
 * ibm_db_sa://zpw86771:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.c1ogj3sd0tgtu0lqde00.databases.appdomain.clou
d:32731/bludb
Done.
```

Failure Mission
1

2015 launch records



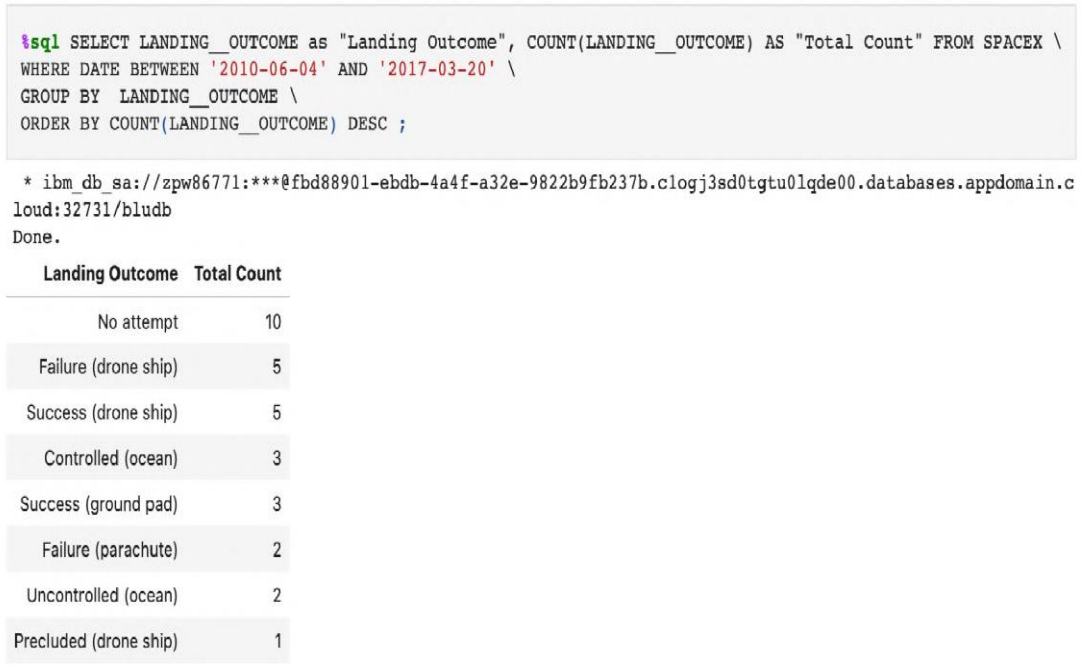
```
%sql SELECT BOOSTER_VERSION, LAUNCH_SITE FROM SPACEX WHERE DATE LIKE '2015-%' AND \
LANDING__OUTCOME = 'Failure (drone ship)';
```

```
 * ibm_db_sa://zpw86771:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.clogj3sd0tgtu0lqde00.
databases.appdomain.cloud:32731/bludb
Done.
```

booster_version	launch_site
F9 v1.1 B1012	CCAFS LC-40
F9 v1.1 B1015	CCAFS LC-40

Rank success count between 2010-06-04 and 2017-03-20

We used the GROUP BY clause to aggregate the landing outcomes and the ORDER BY clause to sort them in descending order, allowing us to identify patterns within the specified time frame.



```
%sql SELECT LANDING__OUTCOME as "Landing Outcome", COUNT(LANDING__OUTCOME) AS "Total Count" FROM SPACEX \
WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20' \
GROUP BY  LANDING__OUTCOME \
ORDER BY COUNT(LANDING__OUTCOME) DESC ;
```

```
 * ibm_db_sa://zpw86771:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.clogj3sd0tgtu0lqde00.databases.appdomain.c
loud:32731/bludb
Done.
```

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

Section 3

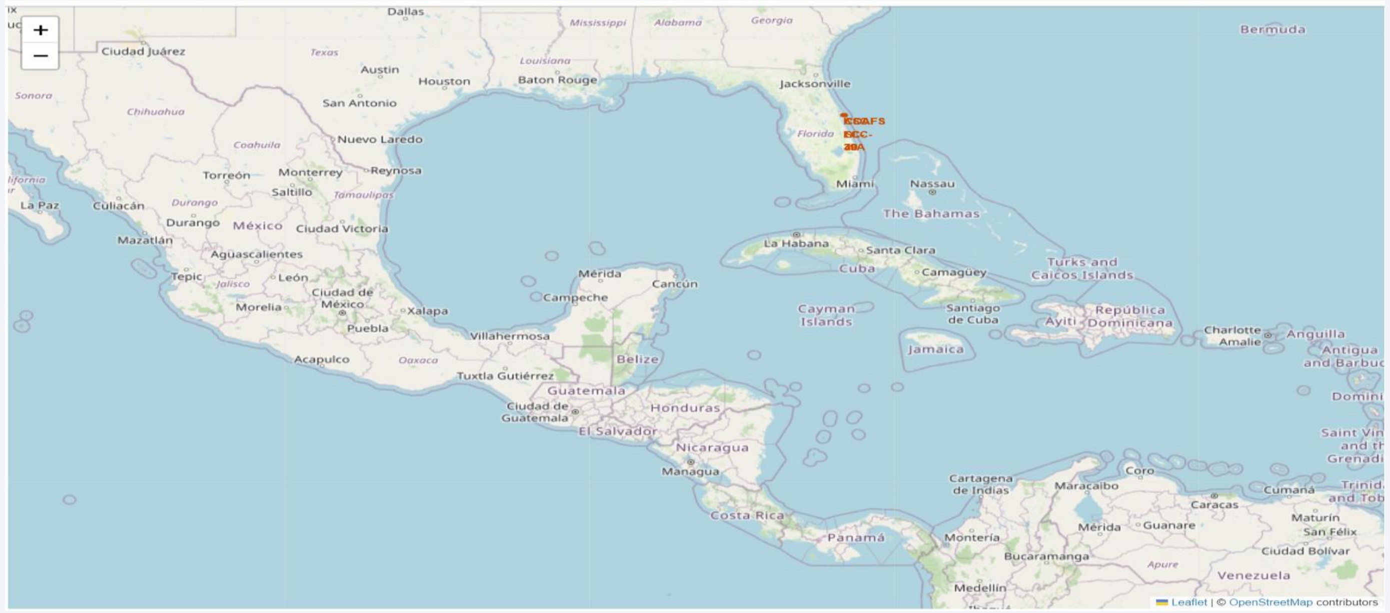
Launch Sites Proximities Analysis

Build an Interactive Map with Folium

- 1. Extracted coordinates for each launch site and marked them with labeled circles.
- 2. Classified launch outcomes as 0 (failure) and 1 (success), displaying them with red and green markers in a clustered layout.

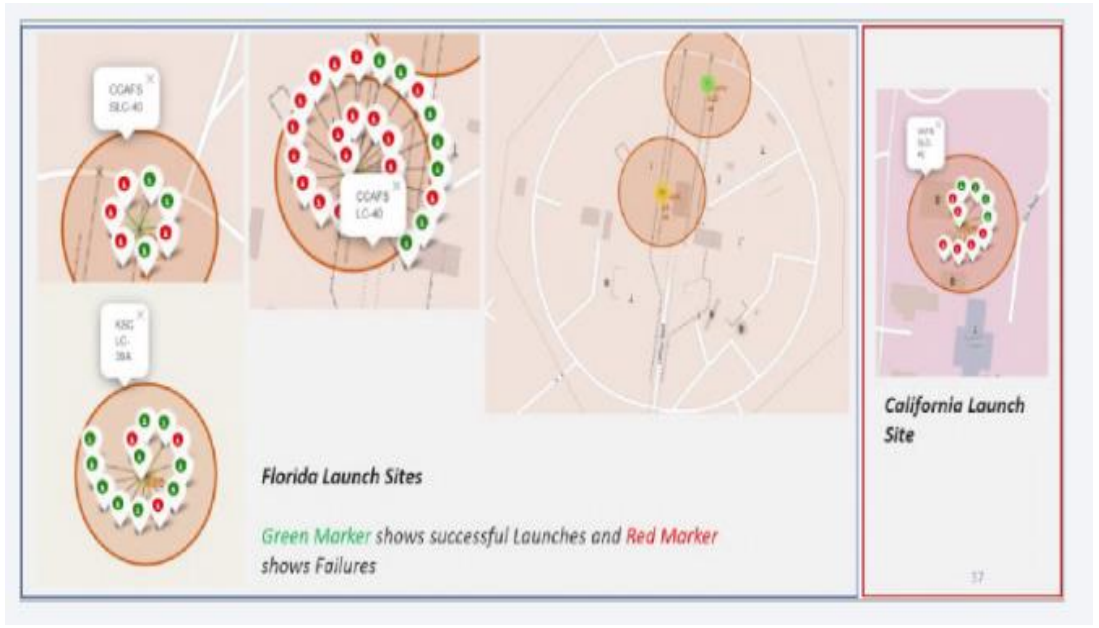
Additionally, distances between launch sites and key landmarks were calculated using Haversine's formula to explore:

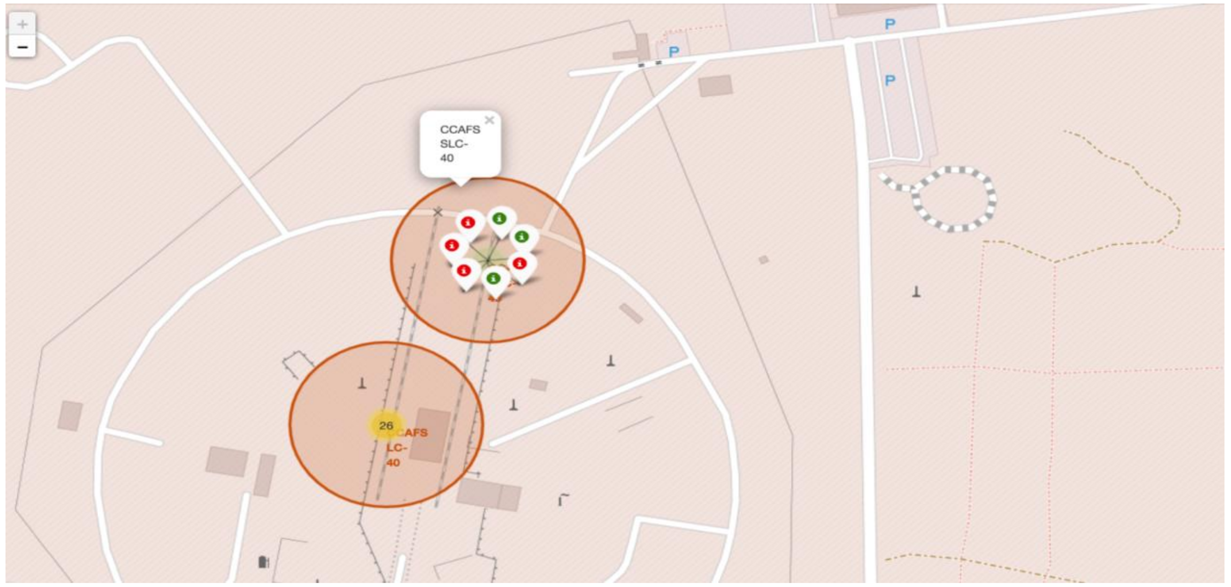
- Distance to railways, highways, and coastlines.
- Proximity to nearby cities.



Launch Outcomes

- At Each Launch Site
- Green markers for successful launches
- Red markers for unsuccessful launches
- Launch site CCAFS SLC-40 has a 3/7 success rate (42.9%)

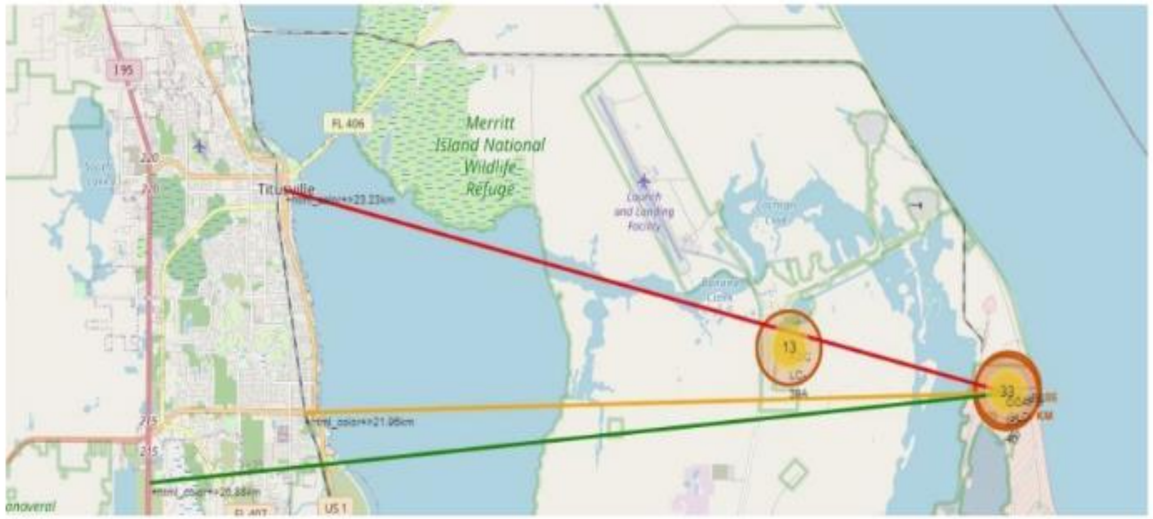


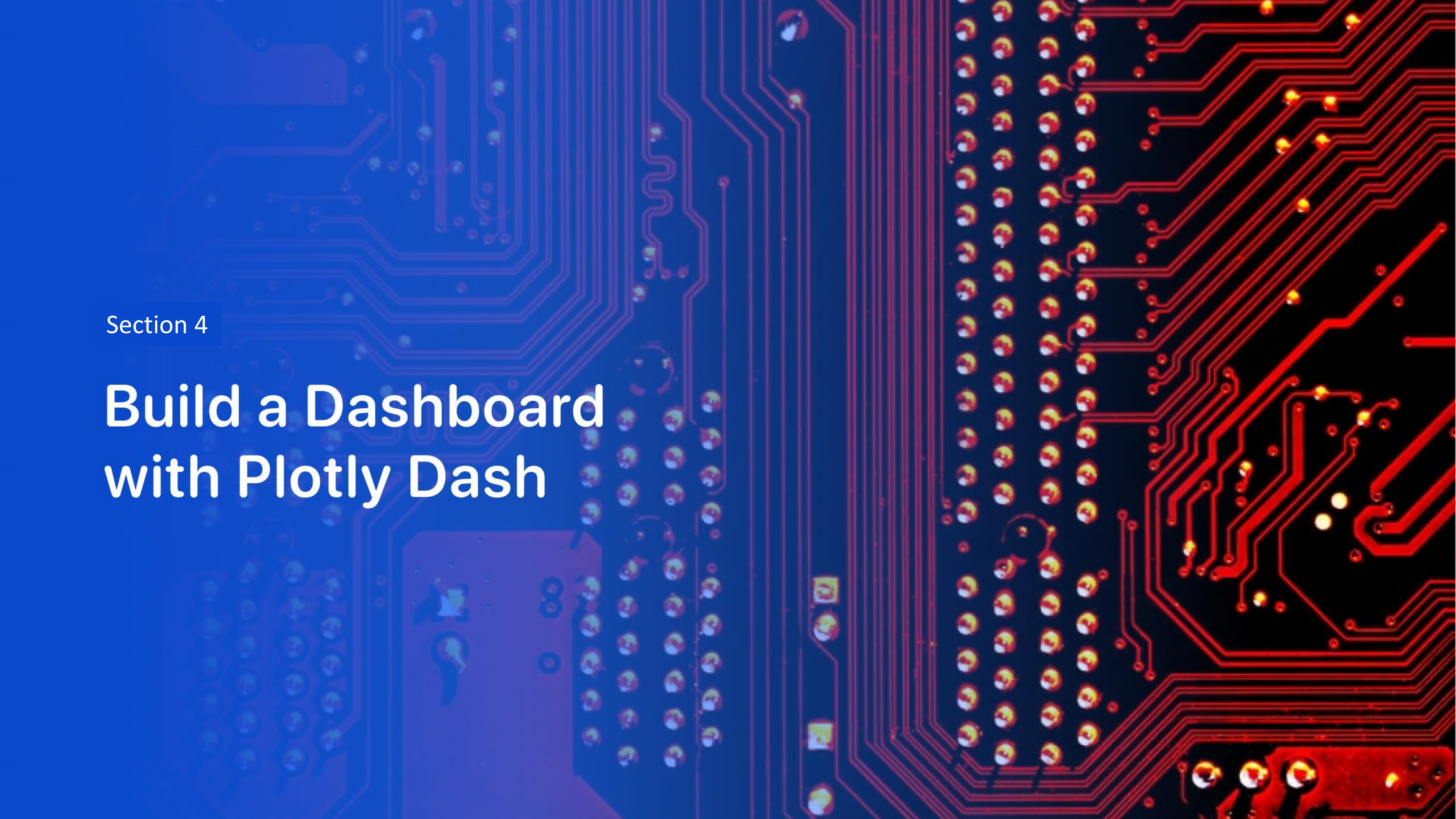


Distance to Proximities

CCAFS SLC-40

- .86 km from nearest coastline
- 21.96 km from nearest railway
- 23.23 km from nearest city
- 26.88 km from nearest highway


Merritt Island National Wildlife Refuge
Titusville
FL 406
Launch and Landing Facility
13
33
US 1
Canaveral



Section 4

Build a Dashboard with Plotly Dash

Build a Dashboard with Plotly Dash

- A dynamic dashboard was created utilizing Plotly Dash, which enables users to adjust the data to meet their needs.
- Within this dashboard:
- The total number of launches from specific sites was displayed using pie charts.
- The relationship between the outcomes of the launches and the mass of the payload (in kilograms) across different booster versions was shown using scatter plots.

Build a Dashboard with Plotly Dash

Indeed, we can cover the following points:

- The launch site with the highest number of successful launches.

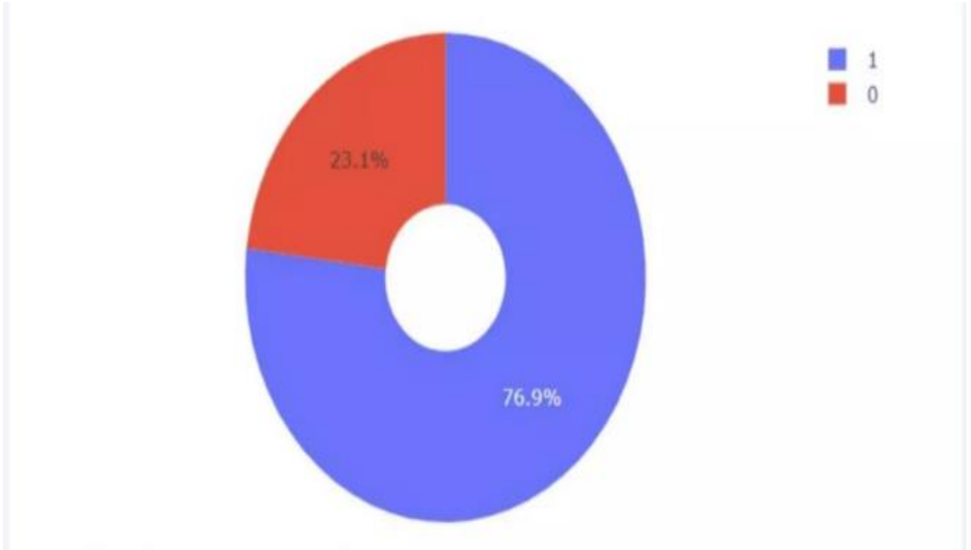
- The launch site with the greatest success rate for launches.

- The payload range(s) with the highest success rate for launches.

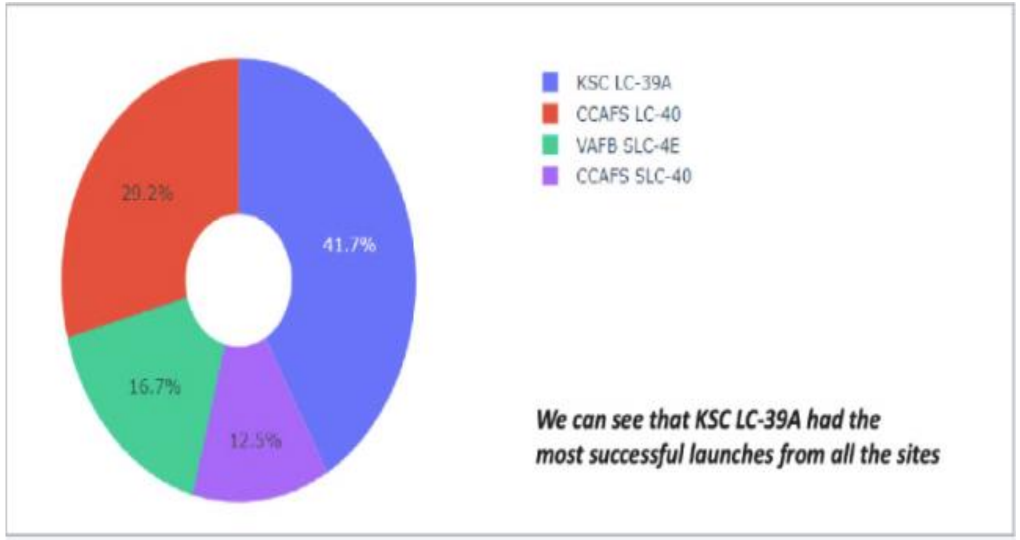
- The payload range(s) with the lowest success rate for launches.

- The Falcon 9 Booster version with the highest launch success rate.of successful launches.

The success percentage by each sites

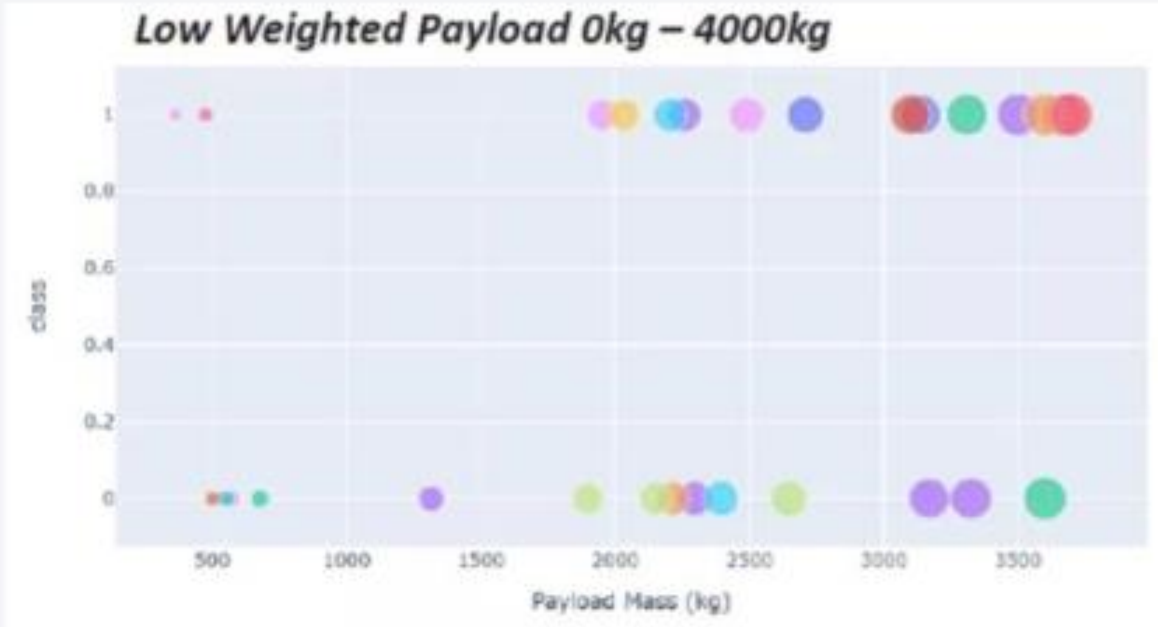


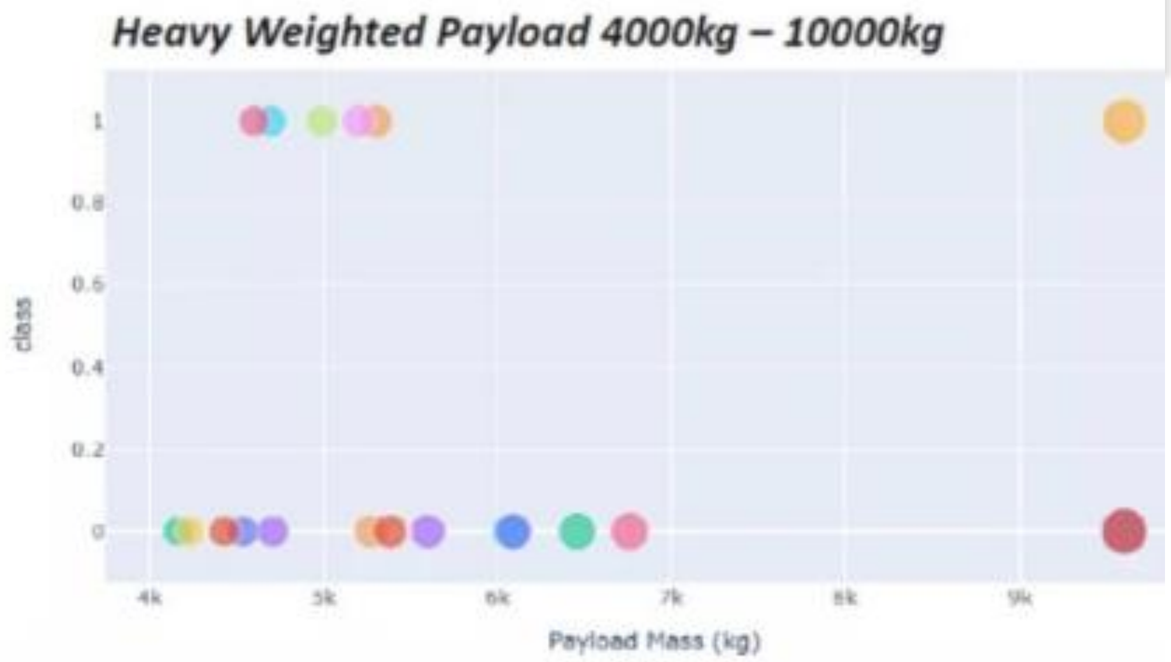
KSC LC-39A achieved a 76.9% success rate while getting a 23.1% failure rate



We can see that KSC LC-39A had the most successful launches from all the sites

Payload vs launch outcome





We observe that success rates are higher for lighter payloads compared to heavier ones.

Results

- The results will be organized into three main segments:

1. Findings from Exploratory Data Analysis (EDA):

    - Observations and insights from the initial data exploration.

    - Identification of trends, patterns, and notable relationships within the data.

2. Interactive Analytics Demonstration (with Screenshots):

    - Visual examples of interactive visualizations and analytics capabilities.

    - Illustration of the analytics dashboard's interactive and dynamic features.

3. Outcomes of Predictive Analysis:

    - Findings from the predictive modeling process.

    - Evaluation of model performance including accuracy, precision, recall, and other relevant metrics.

These sections comprehensively cover the initial analysis, the demonstration of interactive analytics, and the final predictions and evaluations from the predictive analysis phase.

Section 5

Predictive Analysis (Classification)

Prediction process:

Conduct Exploratory Data Analysis (EDA):

Add a class column to the Data Frame

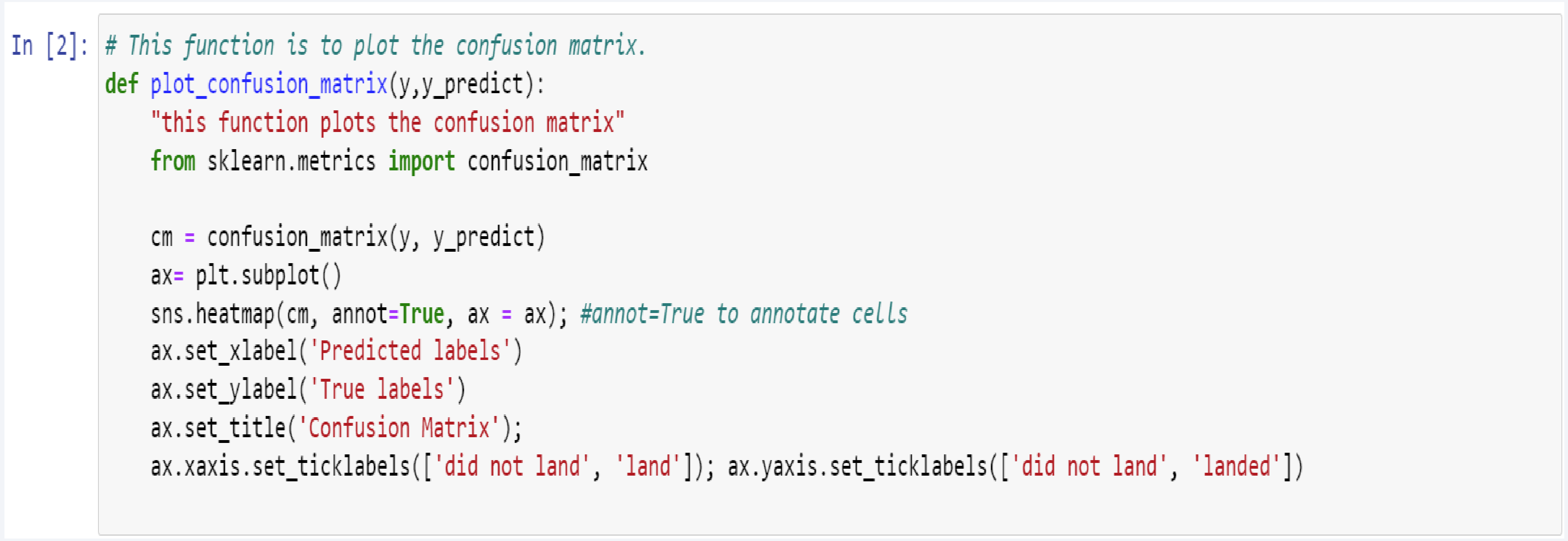
Normalize or standardize the data

Divide the data into training and test sets to prepare it for prediction

Identify optimal hyperparameters for SVM, Classification Trees, and Logistic Regression using the test dataset

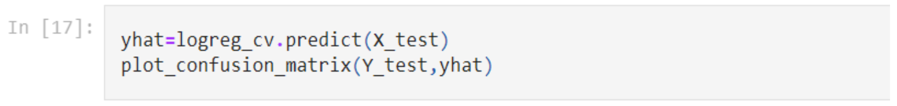
Confusion Matrix

- With this function below, the confusion matrix was defined:

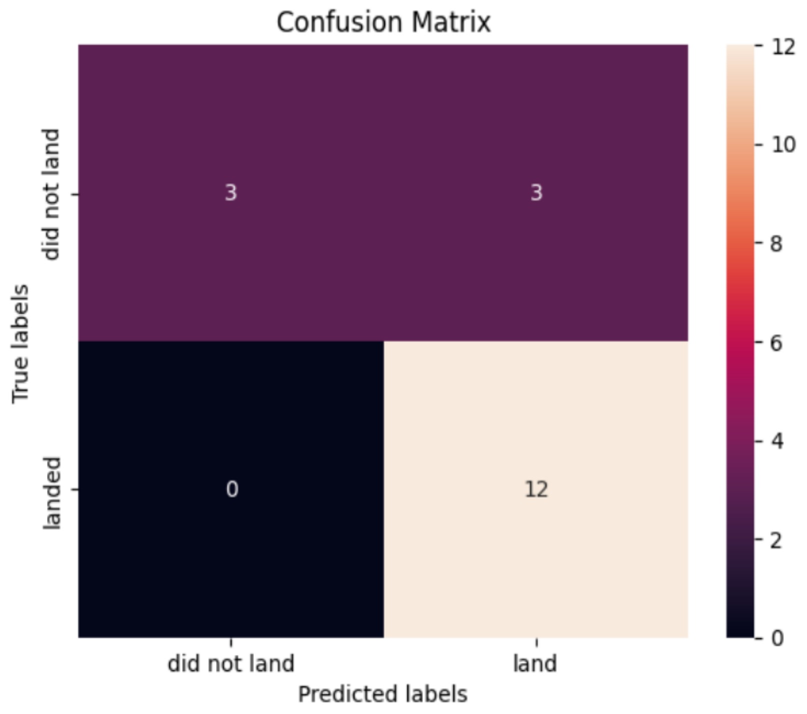


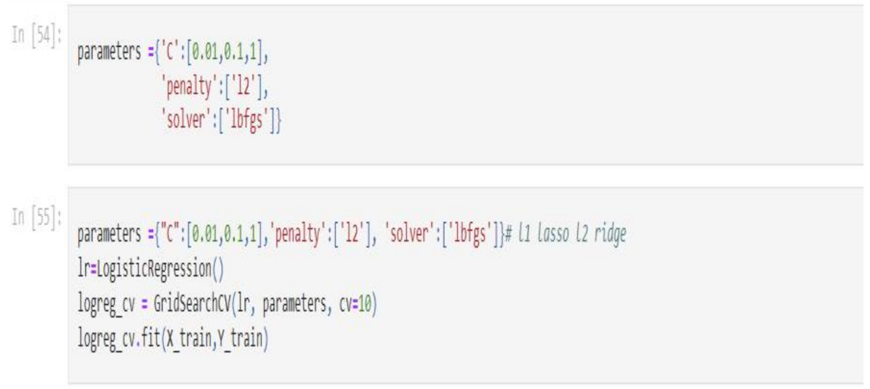
```
In [2]: # This function is to plot the confusion matrix.
        def plot_confusion_matrix(y,y_predict):
            "this function plots the confusion matrix"
            from sklearn.metrics import confusion_matrix

            cm = confusion_matrix(y, y_predict)
            ax= plt.subplot()
            sns.heatmap(cm, annot=True, ax = ax); #annot=True to annotate cells
            ax.set_xlabel('Predicted labels')
            ax.set_ylabel('True labels')
            ax.set_title('Confusion Matrix');
            ax.xaxis.set_ticklabels(['did not land', 'land']); ax.yaxis.set_ticklabels(['did not land', 'landed'])
```

Predicting with Logistic regression:



```
In [17]: yhat=logreg_cv.predict(X_test)
         plot_confusion_matrix(Y_test,yhat)
```

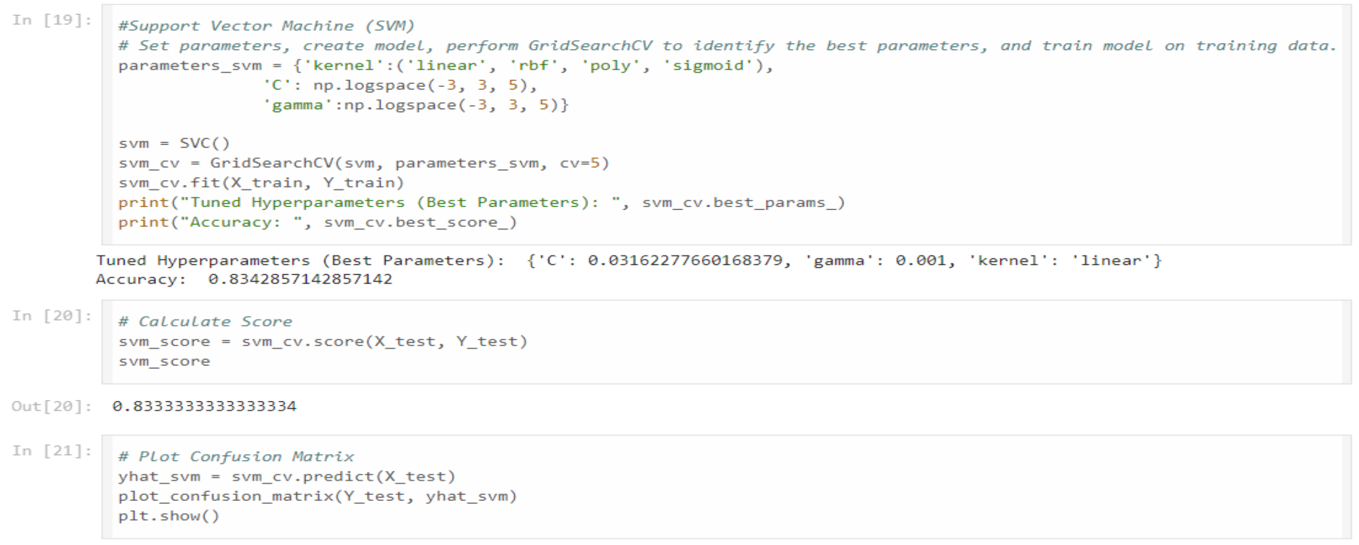




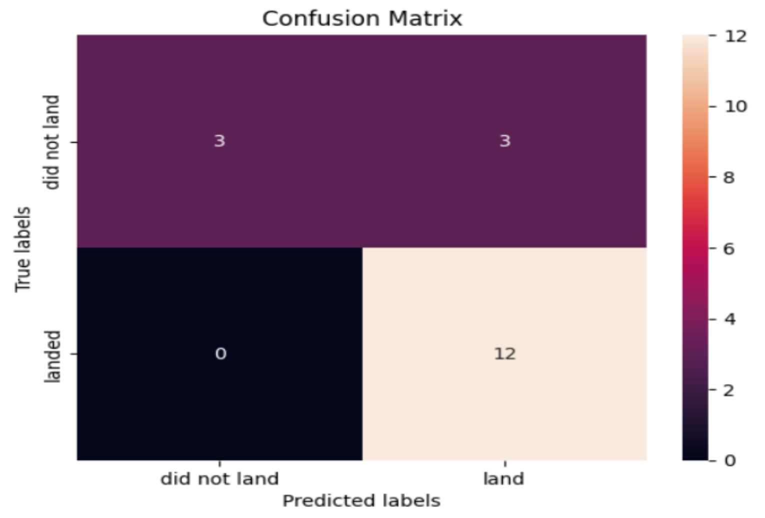
```
In [54]: parameters ={'C':[0.01,0.1,1],
                      'penalty':['l2'],
                      'solver':['lbfgs']}

In [55]: parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
         lr=LogisticRegression()
         logreg_cv = GridSearchCV(lr, parameters, cv=10)
         logreg_cv.fit(X_train,Y_train)
```

True positive in testing set is 12 and 3 is false positive.

Predicting with Support Vector Machin:



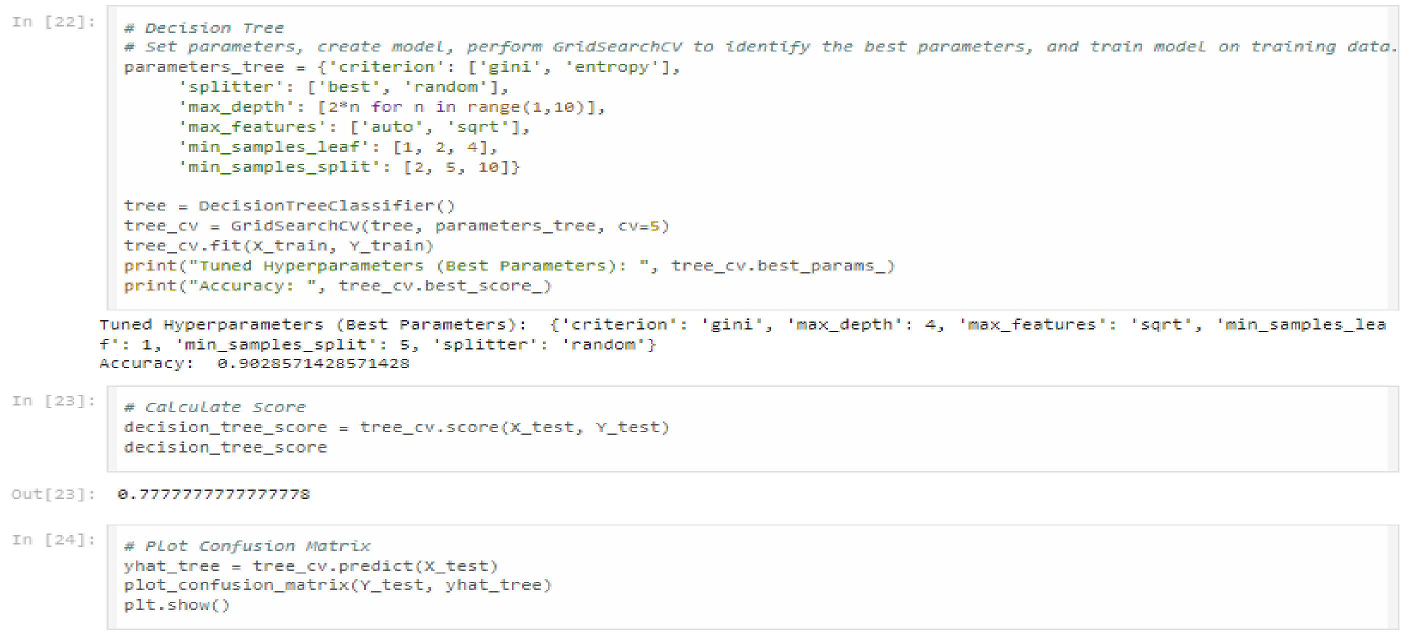
```
In [19]: #Support Vector Machine (SVM)
         # Set parameters, create model, perform GridSearchCV to identify the best parameters, and train model on training data.
         parameters_svm = {'kernel':('linear', 'rbf', 'poly', 'sigmoid'),
                       'C': np.logspace(-3, 3, 5),
                       'gamma':np.logspace(-3, 3, 5)}

         svm = SVC()
         svm_cv = GridSearchCV(svm, parameters_svm, cv=5)
         svm_cv.fit(X_train, Y_train)
         print("Tuned Hyperparameters (Best Parameters): ", svm_cv.best_params_)
         print("Accuracy: ", svm_cv.best_score_)

Tuned Hyperparameters (Best Parameters):  {'C': 0.03162277660168379, 'gamma': 0.001, 'kernel': 'linear'}
Accuracy:  0.8342857142857142

In [20]: # Calculate Score
         svm_score = svm_cv.score(X_test, Y_test)
         svm_score

Out[20]: 0.8333333333333334

In [21]: # Plot Confusion Matrix
         yhat_svm = svm_cv.predict(X_test)
         plot_confusion_matrix(Y_test, yhat_svm)
         plt.show()
```



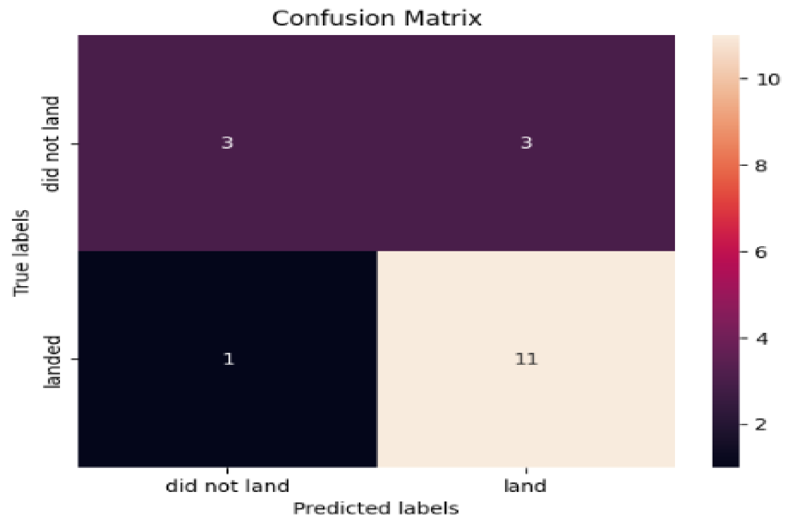
- True positive in testing set is 12, meaning the first stage has the successful landing, and 3 is false positive. Accuracy is %83

Predicting with Decision Tree:

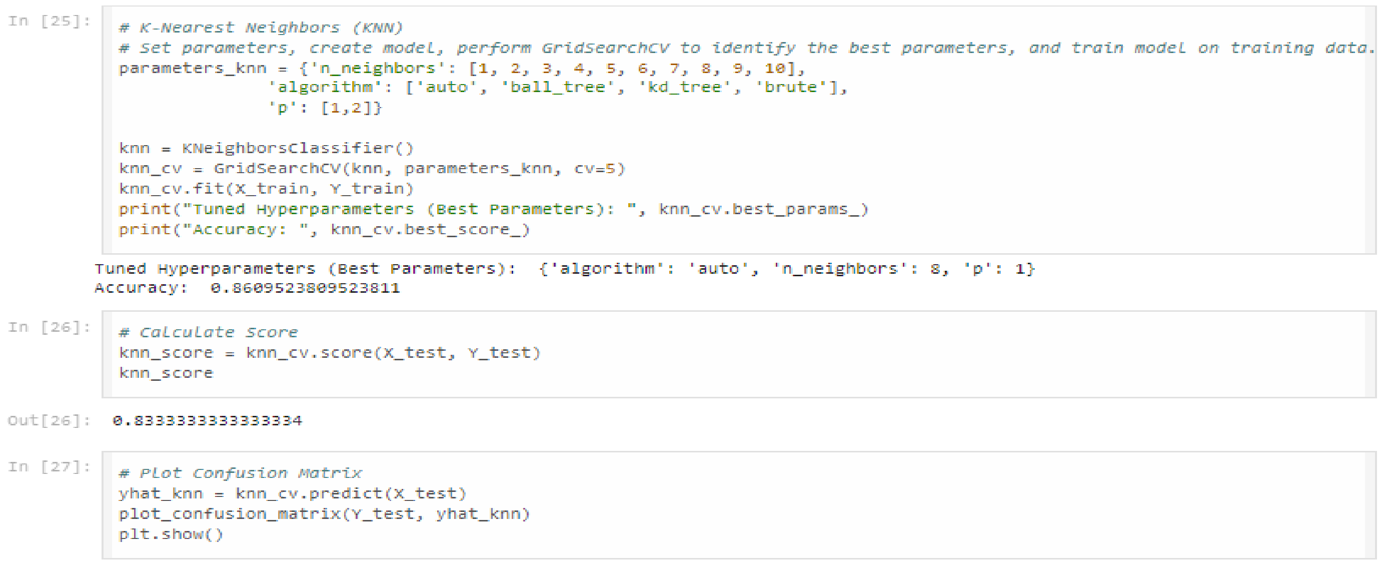
- True positive in testing set in DT model is 11, meaning the first stage has the successful landing, and 3 is false positive.



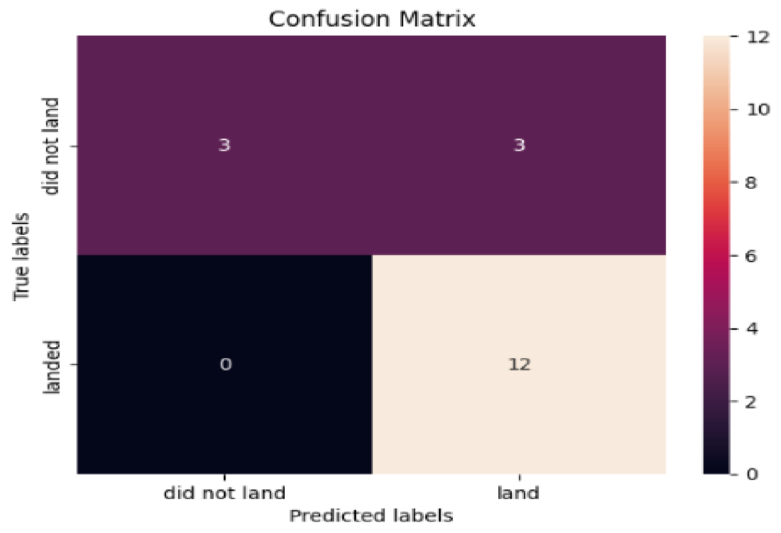
```
In [22]: # Decision Tree
         # Set parameters, create model, perform GridSearchCV to identify the best parameters, and train model on training data.
         parameters_tree = {'criterion': ['gini', 'entropy'],
              'splitter': ['best', 'random'],
              'max_depth': [2*n for n in range(1,10)],
              'max_features': ['auto', 'sqrt'],
              'min_samples_leaf': [1, 2, 4],
              'min_samples_split': [2, 5, 10]}

         tree = DecisionTreeClassifier()
         tree_cv = GridSearchCV(tree, parameters_tree, cv=5)
         tree_cv.fit(X_train, Y_train)
         print("Tuned Hyperparameters (Best Parameters): ", tree_cv.best_params_)
         print("Accuracy: ", tree_cv.best_score_)

        Tuned Hyperparameters (Best Parameters):  {'criterion': 'gini', 'max_depth': 4, 'max_features': 'sqrt', 'min_samples_lea
        f': 1, 'min_samples_split': 5, 'splitter': 'random'}
        Accuracy:  0.9028571428571428

In [23]: # Calculate Score
         decision_tree_score = tree_cv.score(X_test, Y_test)
         decision_tree_score

Out[23]: 0.7777777777777778

In [24]: # Plot Confusion Matrix
         yhat_tree = tree_cv.predict(X_test)
         plot_confusion_matrix(Y_test, yhat_tree)
         plt.show()
```



Predicting with KNN:

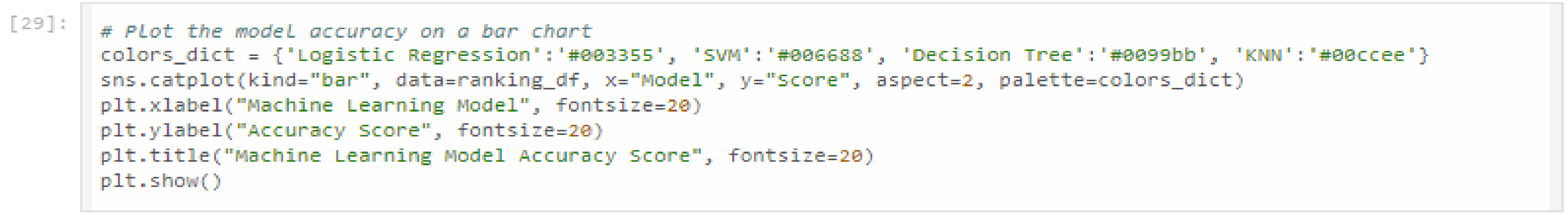


```
In [25]: # K-Nearest Neighbors (KNN)
         # Set parameters, create model, perform GridSearchCV to identify the best parameters, and train model on training data.
         parameters_knn = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                        'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                        'p': [1,2]}

         knn = KNeighborsClassifier()
         knn_cv = GridSearchCV(knn, parameters_knn, cv=5)
         knn_cv.fit(X_train, Y_train)
         print("Tuned Hyperparameters (Best Parameters): ", knn_cv.best_params_)
         print("Accuracy: ", knn_cv.best_score_)

        Tuned Hyperparameters (Best Parameters):  {'algorithm': 'auto', 'n_neighbors': 8, 'p': 1}
        Accuracy:  0.8609523809523811

In [26]: # Calculate Score
         knn_score = knn_cv.score(X_test, Y_test)
         knn_score

Out[26]: 0.8333333333333334

In [27]: # Plot Confusion Matrix
         yhat_knn = knn_cv.predict(X_test)
         plot_confusion_matrix(Y_test, yhat_knn)
         plt.show()
```

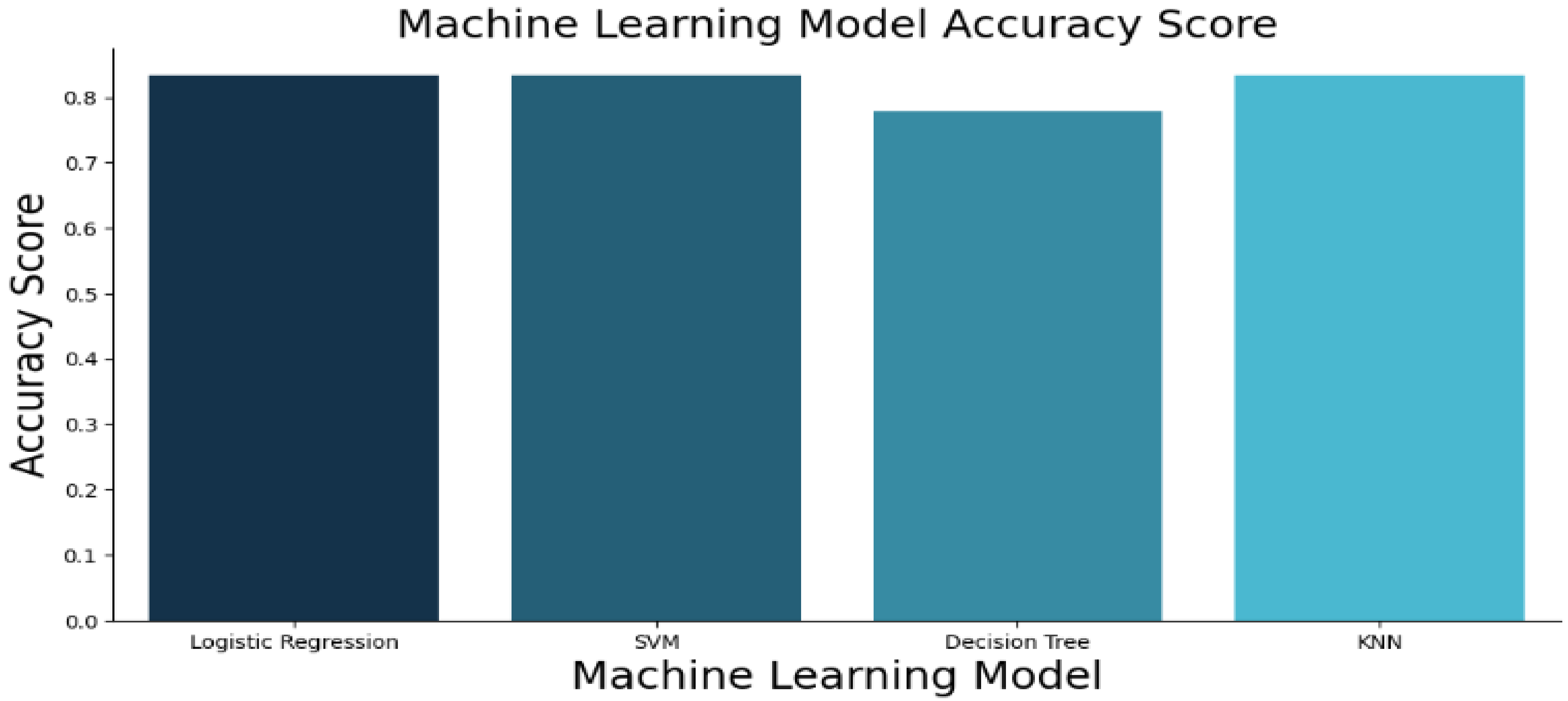


- True positive in testing set in KNN model is 12, meaning the first stage has the successful landing, and 3 is false positive.

Classification Accuracy



```
[29]: # Plot the model accuracy on a bar chart
colors_dict = {'Logistic Regression':'#003355', 'SVM':'#006688', 'Decision Tree':'#0099bb', 'KNN':'#00ccee'}
sns.catplot(kind="bar", data=ranking_df, x="Model", y="Score", aspect=2, palette=colors_dict)
plt.xlabel("Machine Learning Model", fontsize=20)
plt.ylabel("Accuracy Score", fontsize=20)
plt.title("Machine Learning Model Accuracy Score", fontsize=20)
plt.show()
```



Conclusions

- 1. SVM, KNN, Logistic Regression, and the decision tree model demonstrated high predictive accuracy, with the decision tree slightly outperforming the others.
- 2. Launch sites are strategically located near the equator and coastlines to leverage the Earth's rotational velocity, providing a natural boost that minimizes the need for additional fuel and boosters.
- 3. SpaceX's success rate has increased over the years, indicating an improvement in launch operations.
- 4. KSC LC 39A has the highest success rates among all launch sites, achieving a 100% success rate for payloads under 5,500 kg.
- 5. Orbits GEO, HEO, SSO, and ES L1 all recorded a 100% success rate, showcasing the effectiveness of SpaceX's orbital insertion strategies.
- 6. A noticeable correlation exists between payload mass and success rate, with heavier payloads generally seeing higher success rates, reflecting the maturing capabilities of SpaceX's launch technology.

Conclusions

Application of coding to track and enhance classification accuracy effectively,

can be helpful in evaluation of model performance

The importance of a deeper evaluation beyond simple accuracy measures, thereby

improving the presentation's depth and insightfulness.

Innovative Insights

- Dataset Expansion: Expanding the dataset could enhance the predictive analytics outcomes, providing insights into whether the observations can be applied to a broader dataset.

- Feature Examination/PCA: Conducting further analysis of features or applying principal component analysis could be key in boosting the accuracy of the predictions.

- XGBoost Implementation: This study did not incorporate the XGBoost model, known for its effectiveness.

Thank you!