IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Francisco Rodrigo Parente da Ponte
August 08th, 2024

Outline

- Executive Summary
- Introduction
- Methodology
- Results
- Conclusion

Executive Summary

The main goal of this study is to analyze SpaceX Falcon 9 launch data and use Machine Learning models to identify the underlying reasons for the first-stage landing success. This information is crucial for SpaceY to compete with SpaceX.

Methodologies

- **Data Collection:** utilizing APIs and web scraping methods to gather relevant data;
- **Data Transformation:** applying data wrangling techniques to clean and organize the collected data;
- **Exploratory Data Analysis:** using SQL, Pandas, and other data visualization tools to understand the data better;
- **Interactive Dashboard:** creating an interactive dashboard using Plotly Dash and Folium;
- **Predictive Modeling:** building machine learning models to predict the success of the first stage landing of the Falcon 9.

Executive Summary

Results

- **Data Analysis:** detailed insights derived from the exploratory data analysis;
- **Data Visuals and Interactive Dashboards:** visual representations and interactive tools to explore the data;
- **Predictive Model Analysis:** outcomes of the machine learning model predictions regarding the success landing.

Introduction

Project Background and Context

With recent advancements in private space travel, the space industry is becoming increasingly mainstream and accessible to the general population. Despite these advancements, the cost of launches remains a significant barrier for new competitors entering the space race. SpaceX has an advantage over its competitors because it is a pioneer in the technology of landing and reusing the first stage of rockets. Each SpaceX launch costs approximately $62 million, and the company can reuse the first stage for future missions. In contrast, other competitors spend upwards of $165 million per launch. This cost efficiency gives SpaceX a substantial edge in the competitive space industry.

Introduction

Problems to Address

- **Predicting Landing Success:** can we determine if the first stage of the SpaceX Falcon 9 will land successfully?
- **Impact Analysis:** what impact do different parameters and variables, such as launch site, payload mass, and booster version, have on the landing outcomes?
- **Correlation Studies:** what are the correlations between different parameters and variables and their respective success rates?

By addressing these questions, the research aims to provide valuable insights that could help other space agencies make informed decisions about competing with SpaceX.

Section 1

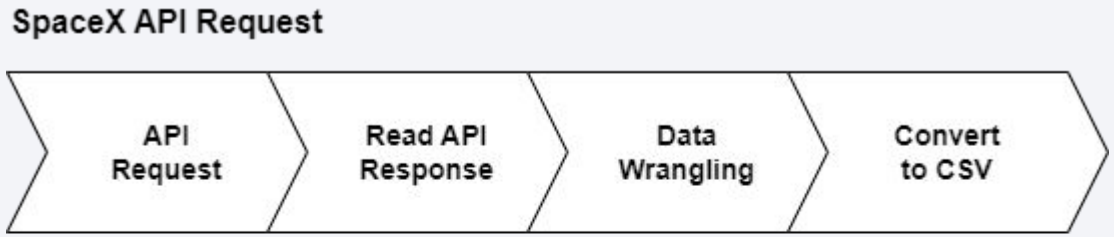
Methodology

Methodology

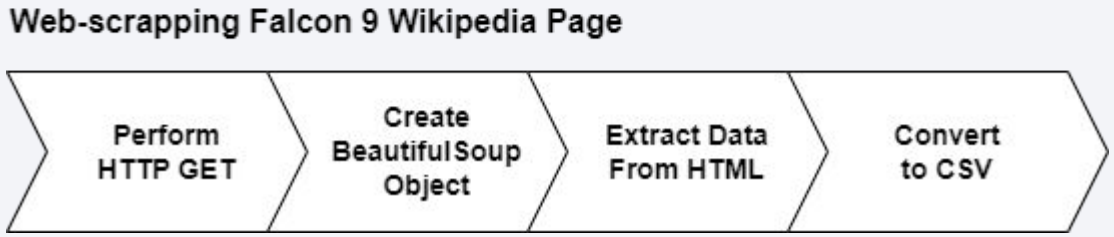
- **Data collection methodology**
  - Utilized the SpaceX API and web-scraped Falcon 9 Wikipedia page to gather data.
- **Perform data wrangling**
  - Performed data wrangling to clean the data and converted mission outcomes (0 for failure, 1 for successful).
- **Perform exploratory data analysis (EDA) using visualization and SQL**
- **Perform interactive visual analytics using Folium and Plotly Dash**
- **Perform predictive analysis using classification models**
  - Standardized, transformed, and split the data into training and test sets;
  - Evaluated different classification algorithms (Logistic Regression, SVM, Decision Tree, and KNN) to determine the best model using test data.

Data Collection

- **SpaceX REST API:** gathered comprehensive data on rocket launches from SpaceX REST API, including information on rockets used, launch dates, payload masses, launch success or failure, launch site name and location (latitude and longitude), booster version (focused on Falcon 9 boosters), landing outcome, and etc.



- **Web Scraping:** used BeautifulSoup to collect Falcon 9 launch data from Wikipedia, including data and time, booster version, payload, orbit, and etc.



Data Collection – SpaceX API

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

[7]:
```
spacex_url = "https://api.spacexdata.com/v4/launches/past"
```

[8]:
```
response = requests.get(spacex_url)
```

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

[9]:
```
response.status_code
```

[9]:
```
200
```

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

[10]:
```
data = pd.json_normalize(response.json())
```

Data Collection – SpaceX API

Task 2: Filter the dataframe to only include `Falcon 9` launches

Finally we will remove the Falcon 1 launches keeping only the Falcon 9 launches. Filter the data dataframe using the `BoosterVersion` column to only keep the Falcon 9 launches. Save the filtered data to a new dataframe called `data_falcon9`.

[23]:
```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = data.loc[data['BoosterVersion'] != 'Falcon 1']
```

Now that we have removed some values we should reset the FlgihtNumber column

[24]:
```
data_falcon9.loc[:, 'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9
```

[24]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	Reused
4	1	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	
5	2	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	

Data Collection – SpaceX API

Task 3: Dealing with Missing Values

Calculate below the mean for the `PayloadMass` using the `.mean()`. Then use the mean and the `.replace()` function to replace `np.nan` values in the data with the mean you calculated.

[26]:
```
# Calculate the mean value of PayloadMass column
mean_payload_mass = data_falcon9['PayloadMass'].mean()

# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'] = data_falcon9['PayloadMass'].replace(np.nan, mean_payload_mass)
```

You should see the number of missing values of the `PayLoadMass` change to zero.

[27]:
```
data_falcon9.isnull().sum()
```

[27]:
```
FlightNumber       0
Date               0
BoosterVersion     0
PayloadMass        0
Orbit              0
LaunchSite         0
Outcome            0
Flights            0
GridFins           0
Reused             0
Legs               0
LandingPad        26
```

Data Collection – SpaceX API

Github

Data Collection - Scraping

TASK 1: Request the Falcon9 Launch Wiki page from its URL

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

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

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

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

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

[6]:
```
soup.title
```

[6]:
```
<title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

Data Collection - Scraping

TASK 2: Extract all column/variable names from the HTML table header

Next, we want to collect all relevant column names from the HTML table header

Let's try to find all tables on the wiki page first. If you need to refresh your memory about `BeautifulSoup`, please check the external reference link towards the end of this lab

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

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

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

Next, we just need to iterate through the `<th>` elements and apply the provided `extract_column_from_header()` to extract colum

[9]:
```
column_names = []

# Apply find_all() function with `th` element on first_launch_table
# Iterate each th element and apply the provided extract_column_from_header() to get a column name
# Append the Non-empty column name (`if name is not None and len(name) > 0`) into a list called column_names
for th in first_launch_table.find_all('th'):
    name = extract_column_from_header(th)
    if name:
        column_names.append(name)
```

Check the extracted column names

[10]:
```
print(column_names)
```

```
['Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome']
```

Data Collection - Scraping

TASK 3: Create a data frame by parsing the launch HTML tables

We will create an empty dictionary with keys from the extracted column names in the previous task. Later, this dictionary will be converted into a Pandas dataframe

[11]:
```
launch_dict= dict.fromkeys(column_names)

# Remove an irrelvant column
del launch_dict['Date and time ( )']

# Let's initial the launch_dict with each value
launch_dict['Flight No.'] = []
launch_dict['Launch site'] = []
launch_dict['Payload'] = []
launch_dict['Payload mass'] = []
launch_dict['Orbit'] = []
launch_dict['Customer'] = []
launch_dict['Launch outcome'] = []
# Added some new columns
launch_dict['Version Booster'] = []
launch_dict['Booster landing'] = []
launch_dict['Date'] = []
launch_dict['Time'] = []
```

[12]:
```
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)
            datatimelist=date_time(row[0])
```

Data Collection - Scraping

Github

Data Wrangling

- Exploratory Data Analysis (EDA) was performed to identify patterns and create training labels for supervised models;
- The dataset included various mission outcomes which were converted into binary training labels:
    - 0 for an unsuccessful landing;
    - 1 for a successful landing.
- The labels were defined based on different landing scenarios:
    - **True Ocean:** successfully landed in the ocean;
    - **False Ocean:** unsuccessfully landed in the ocean;
    - **RTLS (Return to Launch Site):** successfully landed on a ground pad;
    - **False RTLS:** unsuccessfully landed on a ground pad;
    - **True ASDS (Autonomous Spaceport Drone Ship):** successfully landed on a drone ship;
    - **False ASDS:** unsuccessfully landed on a drone ship.

Data Wrangling

TASK 1: Calculate the number of launches on each site

The data contains several Space X launch facilities: Cape Canaveral Space Launch Complex 40 **VAFB SLC 4E** , Vandenberg Air Force Base Space Launch Complex 4E **(SLC-4E)**, Kennedy Space Center Launch Complex 39A **KSC LC 39A** .The location of each Launch Is placed in the column `LaunchSite`

Next, let's see the number of launches for each site.

Use the method `value_counts()` on the column `LaunchSite` to determine the number of launches on each site:

[5]:
```
# Apply value_counts() on column LaunchSite
df['LaunchSite'].value_counts()
```

[5]:
```
LaunchSite
CCAFS SLC 40    55
KSC LC 39A      22
VAFB SLC 4E     13
Name: count, dtype: int64
```

Data Wrangling

TASK 2: Calculate the number and occurrence of each orbit

Use the method `.value_counts()` to determine the number and occurrence of each orbit in the column `Orbit`

[6]:
```
df['Orbit'].value_counts()
```

[6]:
```
Orbit
GTO      27
ISS      21
VLEO     14
PO        9
LEO       7
SSO       5
MEO       3
HEO       1
ES-L1     1
SO        1
GEO       1
Name: count, dtype: int64
```

Data Wrangling

TASK 3: Calculate the number and occurence of mission outcome of the orbits

Use the method `.value_counts()` on the column `Outcome` to determine the number of `landing_outcomes` .Then assign it to a variable landing_outcomes.

[7]:
```
landing_outcomes = df['Outcome'].value_counts()
```

`True Ocean` means the mission outcome was successfully landed to a specific region of the ocean while `False Ocean` means the mission outcome was unsuccessfully landed to a specific region of the ocean. `True RTLS` means the mission outcome was successfully landed to a ground pad `False RTLS` means the mission outcome was unsuccessfully landed to a ground pad. `True ASDS` means the mission outcome was successfully landed to a drone ship `False ASDS` means the mission outcome was unsuccessfully landed to a drone ship. `None ASDS` and `None None` these represent a failure to land.

[8]:
```
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
```

We create a set of outcomes where the second stage did not land successfully:

[9]:
```
bad_outcomes = set(landing_outcomes.keys()[[1, 3, 5, 6, 7]])
bad_outcomes
```

[9]:
```
{'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

Data Wrangling

TASK 4: Create a landing outcome label from Outcome column

Using the `Outcome`, create a list where the element is zero if the corresponding row in `Outcome` is in the set `bad_outcome`; otherwise, it's one. Then assign it to the variable `landing_class`:

[10]:
```
landing_class = df['Outcome'].apply(lambda x: 0 if x in bad_outcomes else 1)
```

This variable will represent the classification variable that represents the outcome of each launch. If the value is zero, the first stage did not land successfully; one means the first stage landed Successfully

[11]:
```
df['Class'] = landing_class
df[['Class']].head(8)
```

[11]:

	Class
0	0
1	0
2	0
3	0

Data Wrangling

Github

EDA with Data Visualization

- As part of the Exploratory Data Analysis (EDA), several charts were plotted to gain insights;
- **Scatter Plots** were used to visualize the relationships and correlations between variables:
  - Flight Number and Launch Site;
  - Payload and Launch Site;
  - Flight Number and Orbit Type;
  - Payload and Orbit Type.
- **Bar Charts** were used to compared values of a variable at a given point in time, making it easy to see which groups are highest or most common:
  - Success rate of each orbit type.
- **Line Charts** were used to track changes over time to depict trends:
  - Average launch success yearly trend.

EDA with Data Visualization

Github

EDA with SQL

To better understand the SpaceX dataset, several SQL queries were executed:

1. **Unique Launch Sites:** displayed the names of unique launch sites in space missions;
2. **Launch Sites Starting:** retrieved 5 records where launch sites begin with 'CCA';
3. **Total Payload Mass:** displayed the total payload mass carried by boosters launched by NASA (CRS);
4. **Average Payload Mass by Booster Version:** calculated the average payload mass carried by the booster F9 v1.1;
5. **First Successful Ground Pad Landing:** listed the date of the first successful landing on a ground pad;

EDA with SQL

6. **Boosters with Success on Drone Ship and Specific Payload Mass:** listed boosters that successfully landed on a drone ship and carried a payload mass between 4000 and 6000;
7. **Total Successful and Failed Missions:** displayed the total number of successful and failed mission outcomes;
8. **Booster Versions with Maximum Payload Mass:** listed the booster versions that carried the maximum payload mass using a subquery;
9. **Failed Drone Ship Landings in 2015:** listed the failed landing outcomes on drone ships in 2015, along with their booster versions and launch site names;
10. **Ranking Landing Outcomes:** ranked the count of different landing outcomes (e.g., Failure on drone ship, Success on ground pad) between the dates 2010-06-04 and 2017-03-20 in descending order.

EDA with SQL

Github

Build an Interactive Map with Folium

The following features were implemented using Folium:

- **Launch Site Markers:** marked all launch sites to visually identify their locations;
- **Highlighted Areas:** used `folium.circle` and `folium.marker` to emphasize areas with text labels over each launch site;
- **Success and Failure Markers:** added `MarkerCluster()` to display launch success (green) and failure (red) markers for each site;
- **Distance Calculations:** calculated distances from launch sites to nearby features (coastline, railroad, highway, city);
- **Mouse Position Tracking:** Included `MousePosition()` to get coordinates of the mouse position on the map;
- **Distance Display:** Used `folium.Marker()` to show distances (in KM) from launch sites to nearby features;
- **Polyline Connections:** Added `folium.Polyline()` to draw lines between launch sites and proximate features.

Build an Interactive Map with Folium

Github

Build a Dashboard with Plotly Dash

A Plotly Dash web application was developed to perform interactive visual analytics on SpaceX launch data in real-time.

The dashboard includes components:

- **Launch Site Drop-down:** allows users to filter visualizations by all launch sites or a specific site;
- **Payload Range Slider:** facilitates the selection of different payload ranges to identify visual patterns.

And charts:

- **Pie Chart:** displays total successful launches when "All Sites" is selected, and shows success and failure counts for a selected site;
- **Scatter Chart:** visualizes the correlation between payload and mission outcomes for selected site(s), with color labels indicating different booster versions for each scatter point.

These features enhance user interaction and enable detailed analysis of the launch data.

Build a Dashboard with Plotly Dash

Github

Predictive Analysis (Classification)

Building the Model

- Scaled the columns to ensure uniformity;
- Divided the dataset into testing and training sets;
- Chose machine learning algorithms for classification, including:
  - KNN;
  - Decision Tree;
  - SVM;
  - Logistic Regression.
- Employed Grid Search and Cross Validation to identify the best tuning parameters for each model based on the training sets.

Predictive Analysis (Classification)

Evaluating the Model

- Assessed the accuracy of each model on both training and testing sets;
- Plotted a confusion matrix to visualize model performance.

Improving the Model

- Implemented feature engineering and algorithm tuning to enhance model performance.

Selecting the Best Classifier

- Identified the model with the highest accuracy score on the test set as the best model;
- In case of a tie, the accuracy on the training set was also considered.

Predictive Analysis (Classification)

TASK 1

Create a NumPy array from the column `Class` in `data` , by applying the method `to_numpy()` then assign it to the variable `Y` ,make sure the output is a Pandas series (only one bracket df['name of column']).

[8]:
```
Y = data['Class'].to_numpy()
```

[9]:
```
type(Y)
```

[9]:
```
numpy.ndarray
```

Predictive Analysis (Classification)

TASK 2

Standardize the data in `X` then reassign it to the variable `X` using the transform provided below.

[10]:
```
# students get this
scaler = preprocessing.StandardScaler()
```

[11]:
```
X = scaler.fit_transform(X)
```

We split the data into training and testing data using the function `train_test_split`. The training data is divided into validation data, a second set used for training data; then the models are trained and hyperparameters are selected using the function `GridSearchCV`.

Predictive Analysis (Classification)

TASK 3

Use the function train_test_split to split the data X and Y into training and test data. Set the parameter test_size to 0.2 and random_state to 2. The training data and test data should be assigned to the following labels.

`X_train, X_test, Y_train, Y_test`

[12]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
```

we can see we only have 18 test samples.

[13]:
```
y_test.shape
```

[13]:
```
(18,)
```

Predictive Analysis (Classification)

TASK 4

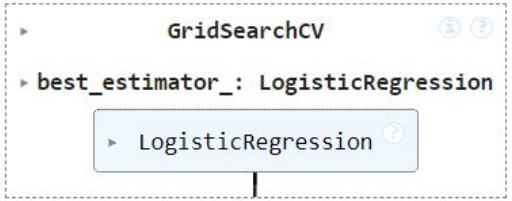
Create a logistic regression object then create a GridSearchCV object `logreg_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

[14]:
```
parameters = {
    'C':[0.01,0.1,1],
    'penalty':['l2'],
    'solver':['lbfgs']
}
```

[15]:
```
lr = LogisticRegression()
logreg_cv = GridSearchCV(lr, param_grid=parameters, cv=10)
```

[16]:
```
logreg_cv.fit(X_train, y_train)
```

[16]:



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

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

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

Predictive Analysis (Classification)

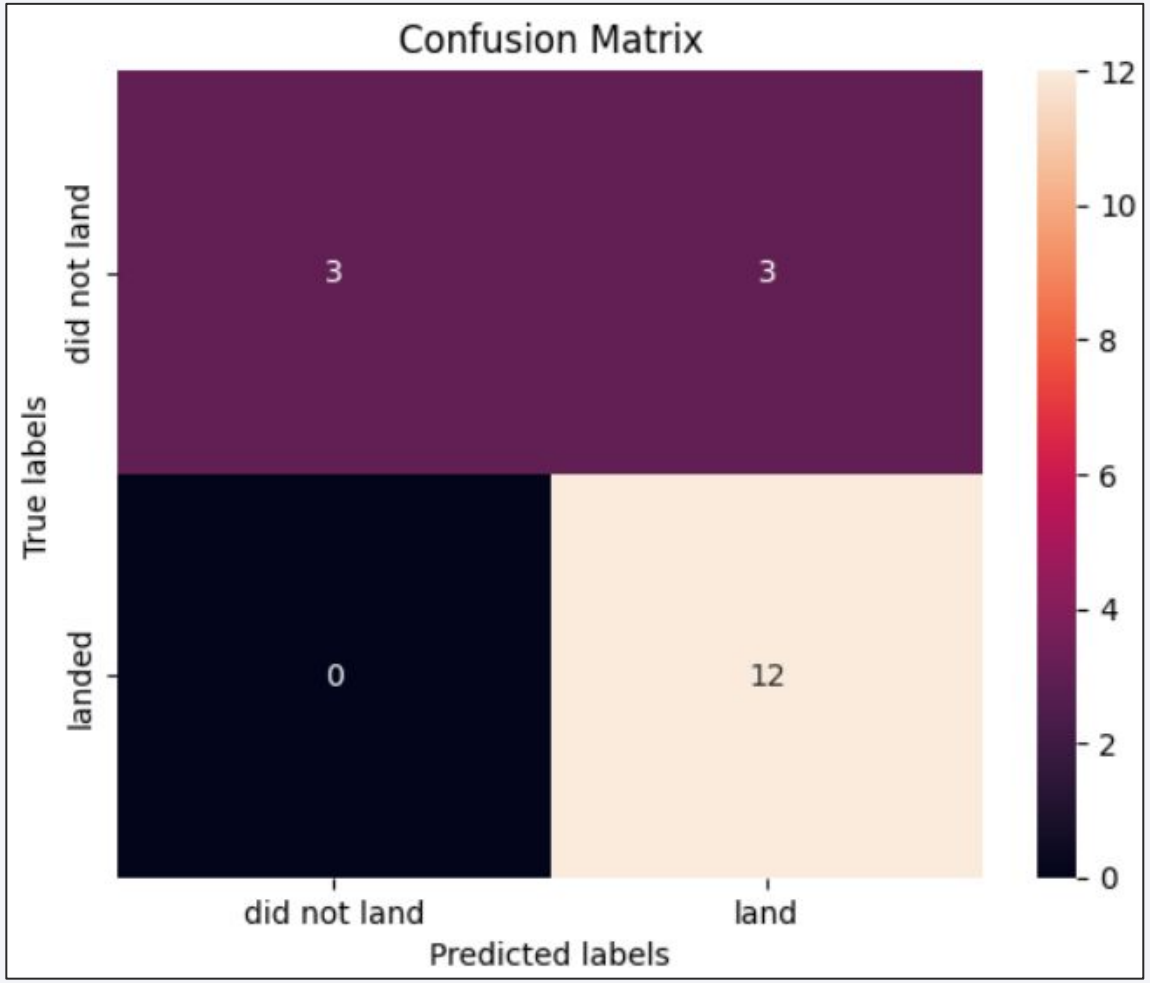
TASK 5

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

```
[18]: lr_acc = logreg_cv.score(X_test, y_test)

[19]: print("Test data accuracy for Logistic Regression: ", lr_acc)

Test data accuracy for Logistic Regression:  0.8333333333333334
```



Predictive Analysis (Classification)

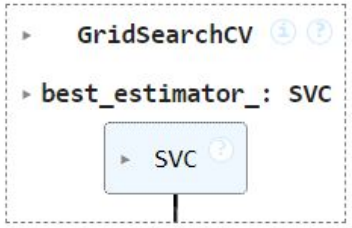
TASK 6

Create a support vector machine object then create a `GridSearchCV` object `svm_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

[21]:
```
parameters = {
    'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
    'C': np.logspace(-3, 3, 5),
    'gamma':np.logspace(-3, 3, 5)
}
```

[22]:
```
svm = SVC()
svm_cv = GridSearchCV(svm, param_grid=parameters, cv=10)
svm_cv.fit(X_train, y_train)
```

[22]:



[23]:
```
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
```

Predictive Analysis (Classification)

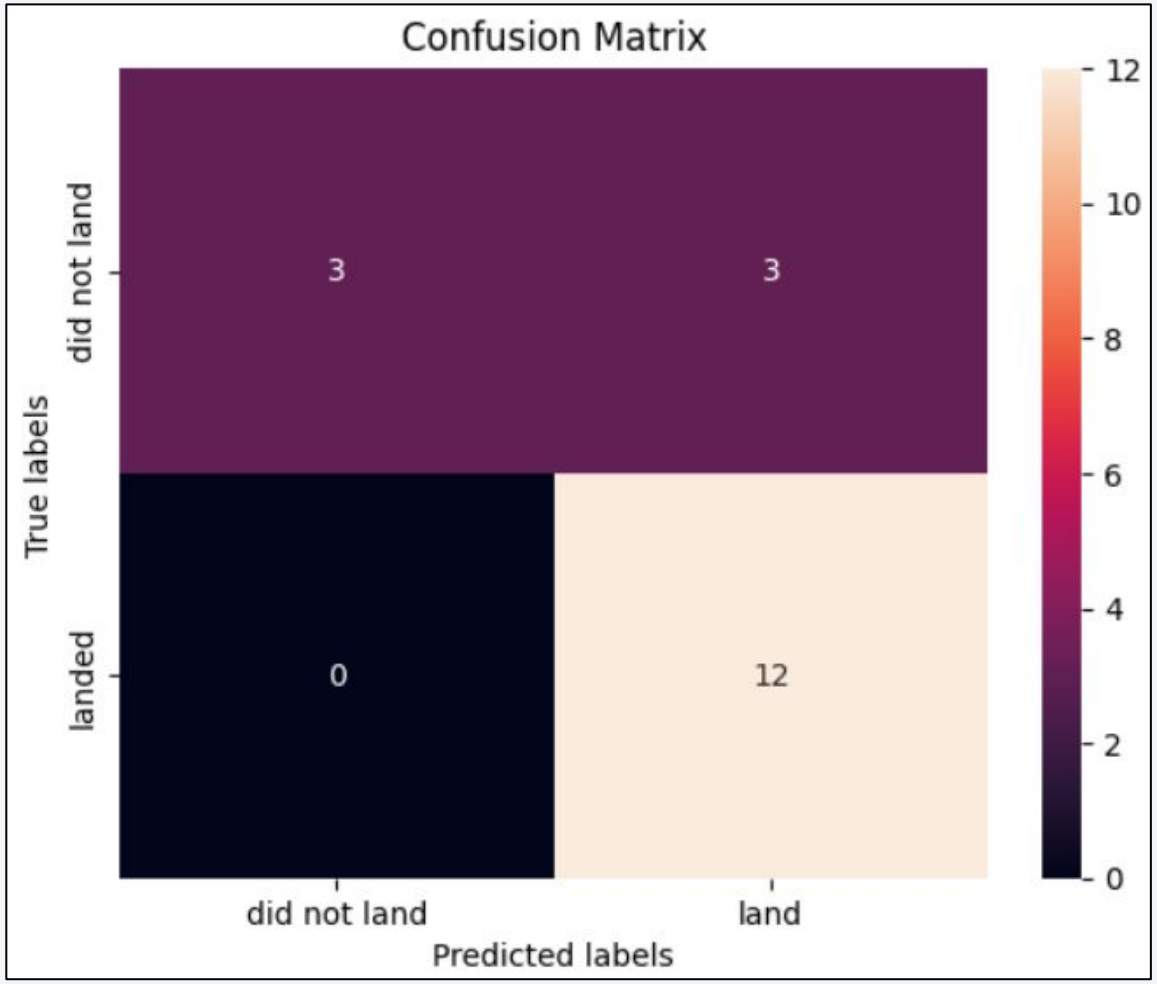
TASK 7

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

```
[24]: svm_acc = svm_cv.score(X_test, y_test)

[25]: print("Test data accuracy for SVM: ", svm_acc)
```

```
Test data accuracy for SVM:  0.8333333333333334
```



Predictive Analysis (Classification)

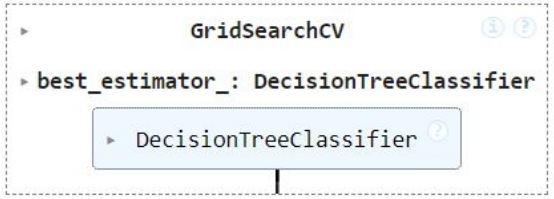
TASK 8

Create a decision tree classifier object then create a `GridSearchCV` object `tree_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

[27]:
```
parameters = {
    '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]
}
```

[28]:
```
tree = DecisionTreeClassifier()
tree_cv = GridSearchCV(tree, param_grid=parameters, cv=10)
tree_cv.fit(X_train, y_train)
```

[28]:



[29]:
```
print("tuned hpyerparameters :(best parameters) ", tree_cv.best_params_)
print("accuracy :", tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 14, 'max_features': 'sqrt', 'min_samples_leaf': 4, 'min_samples_split': 10,
'splitter': 'random'}
accuracy : 0.8732142857142857
```

Predictive Analysis (Classification)

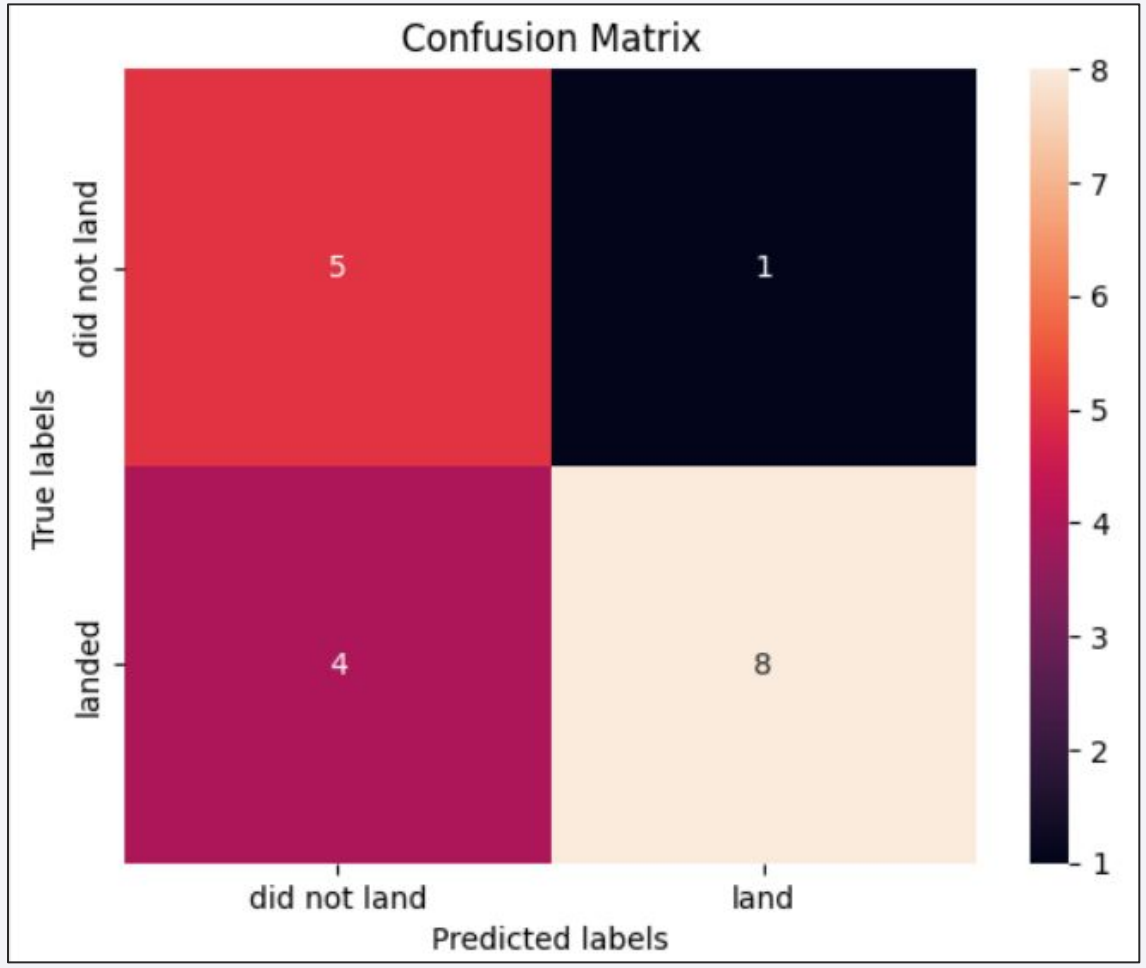
TASK 9

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

[30]:
```
tree_acc = tree_cv.score(X_test, y_test)
```

[31]:
```
print("Test data accuracy for Decision Tree: ", tree_acc)
```

```
Test data accuracy for Decision Tree:  0.7222222222222222
```



Predictive Analysis (Classification)

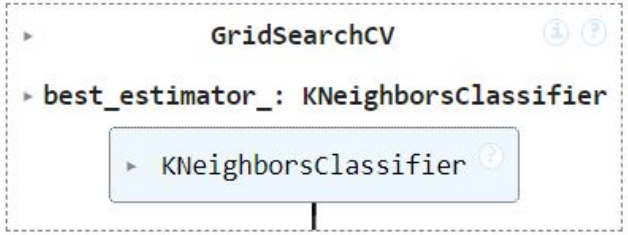
TASK 10

Create a k nearest neighbors object then create a `GridSearchCV` object `knn_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

[33]:
```
parameters = {
    'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'p': [1,2]
}
```

[34]:
```
KNN = KNeighborsClassifier()
knn_cv = GridSearchCV(KNN, param_grid=parameters, cv=10)
knn_cv.fit(X_train, y_train)
```

[34]:



[35]:
```
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
```

Predictive Analysis (Classification)

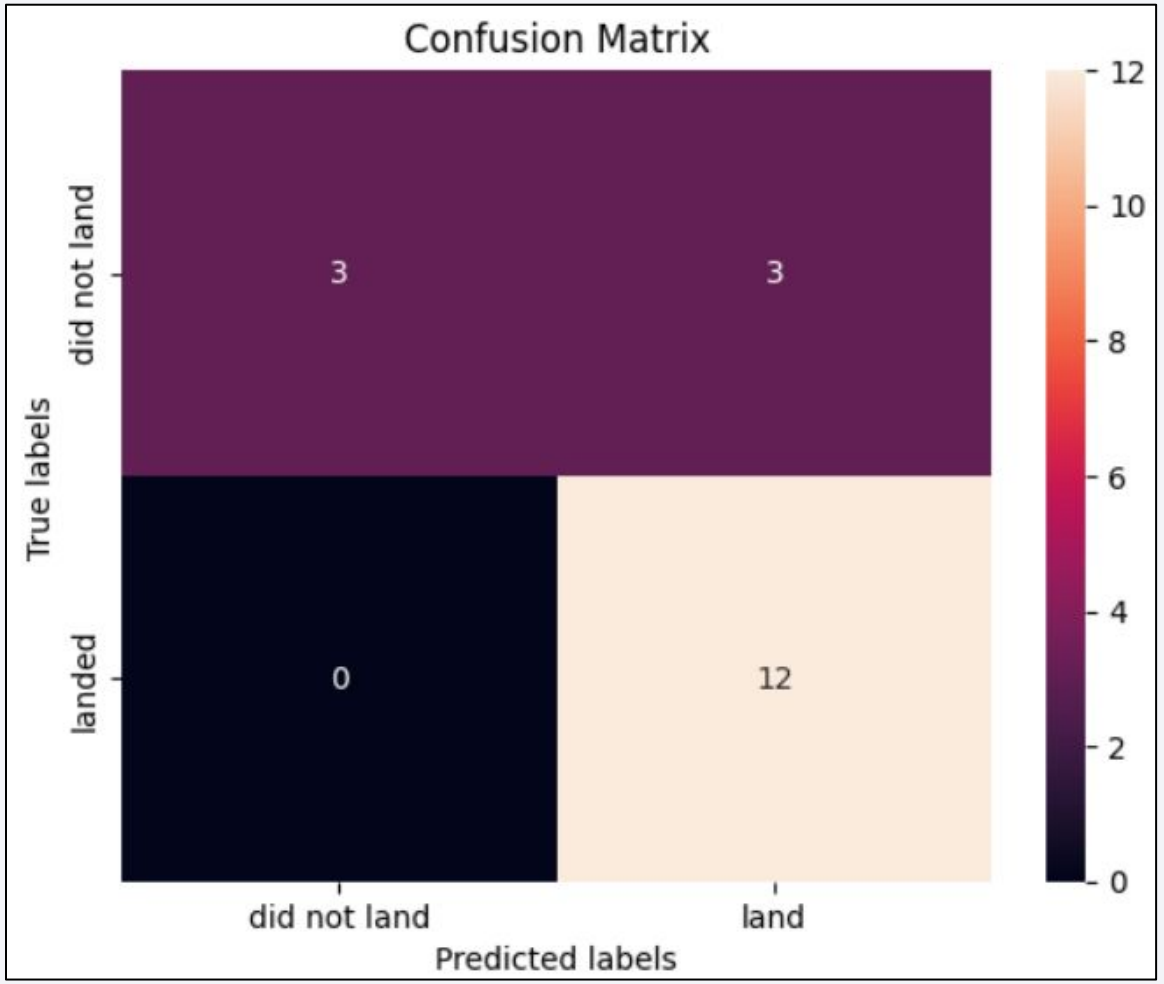
TASK 11

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

[36]:
```
knn_acc = knn_cv.score(X_test, y_test)
```

[37]:
```
print("Test data accuracy for KNN: ", knn_acc)
```

```
Test data accuracy for KNN:  0.8333333333333334
```



Predictive Analysis (Classification)

TASK 12

Find the method performs best:

[39]:
```
Report = pd.DataFrame({
    'algorithm': ['Logistic Regression', 'SVM','Decision Tree','KNN'],
    'train accuracy': [logreg_cv.best_score_, svm_cv.best_score_, tree_cv.best_score_, knn_cv.best_score_],
    'test accuracy': [lr_acc, svm_acc, tree_acc, knn_acc]
})
```

[40]:
```
Report
```

[40]:

	algorithm	train accuracy	test accuracy
0	Logistic Regression	0.846429	0.833333
1	SVM	0.848214	0.833333
2	Decision Tree	0.873214	0.722222
3	KNN	0.848214	0.833333

[41]:
```
index = Report['train accuracy'].idxmax()
algorithm = Report['algorithm'][index]
accuracy = Report['train accuracy'][index]
```

[42]:
```
print(f'The best performing alogrithm is {algorithm} with score {accuracy}')
```

```
The best performing alogrithm is Decision Tree with score 0.8732142857142857
```

Predictive Analysis (Classification)

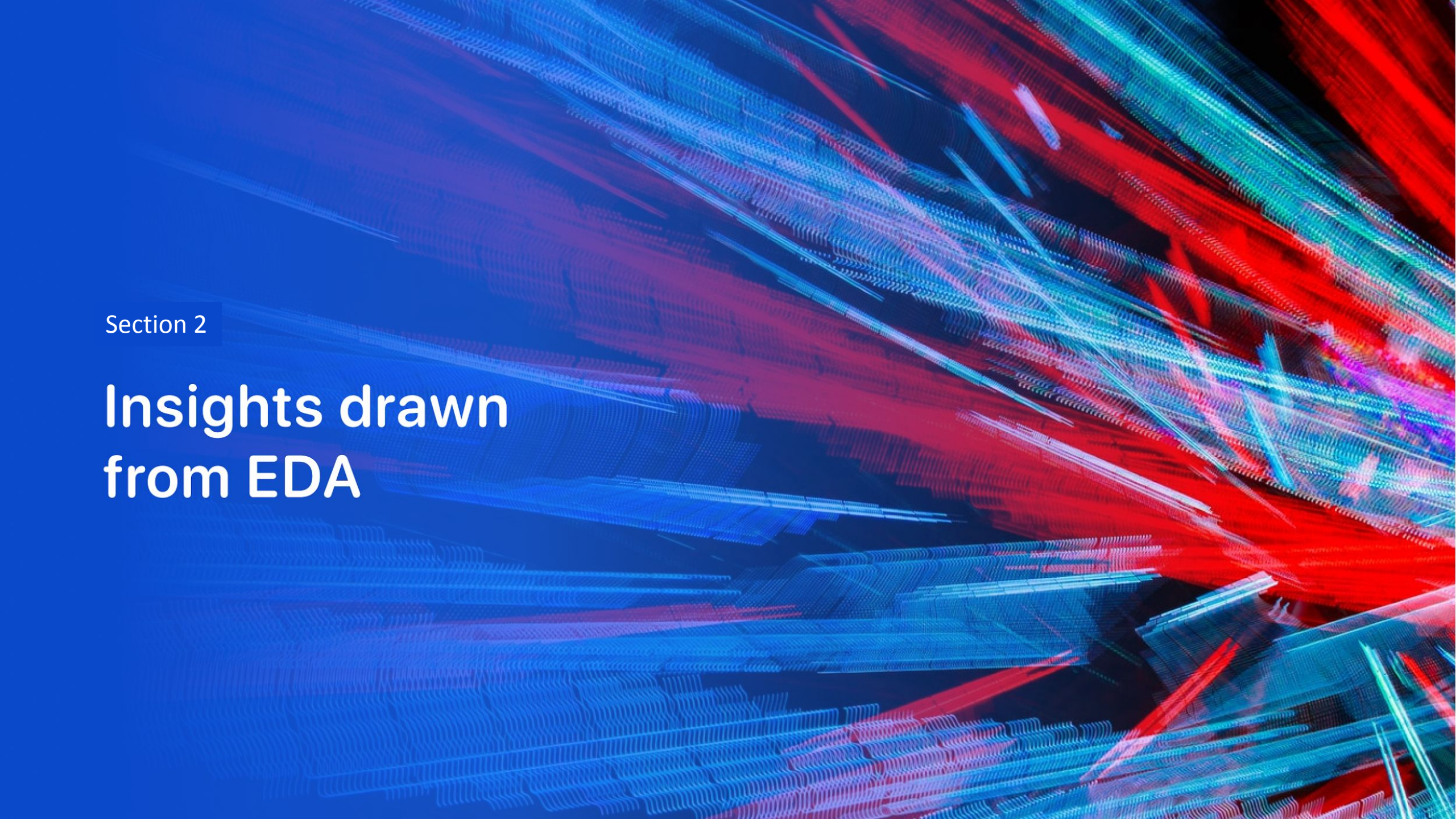
Github

Results

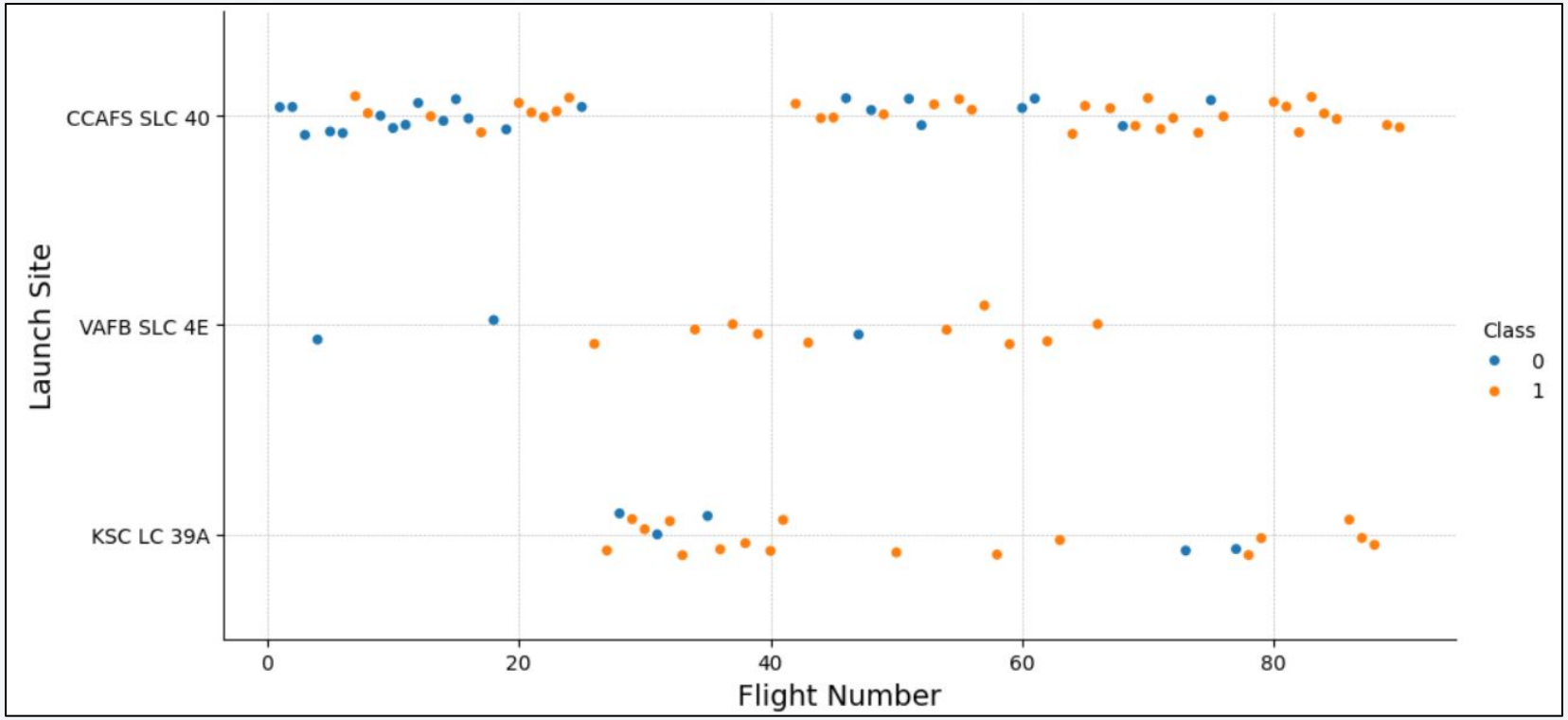
- Exploratory data analysis results;
- Interactive analytics demo in screenshots;
- Predictive analysis results.

Section 2

Insights drawn from EDA

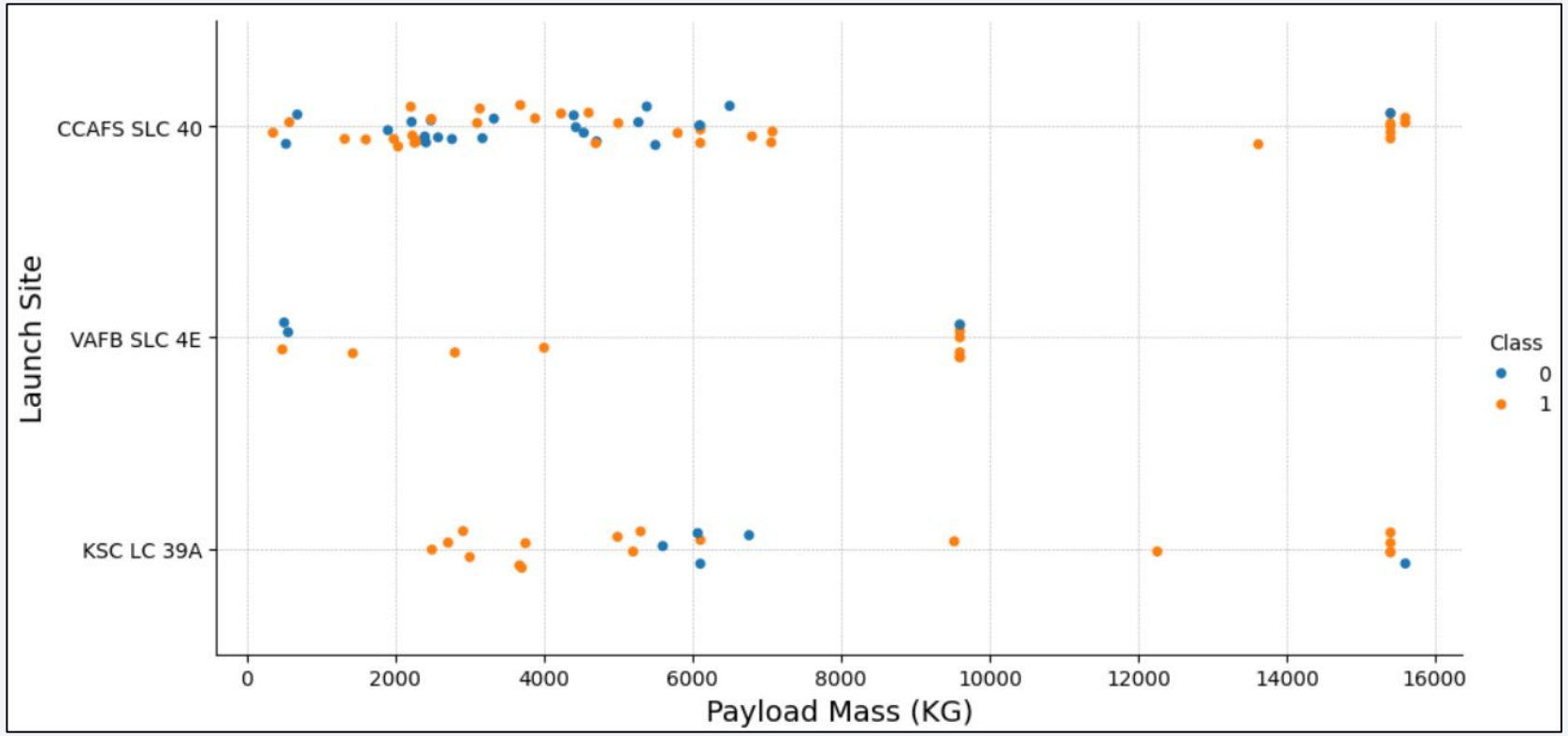


Flight Number vs. Launch Site



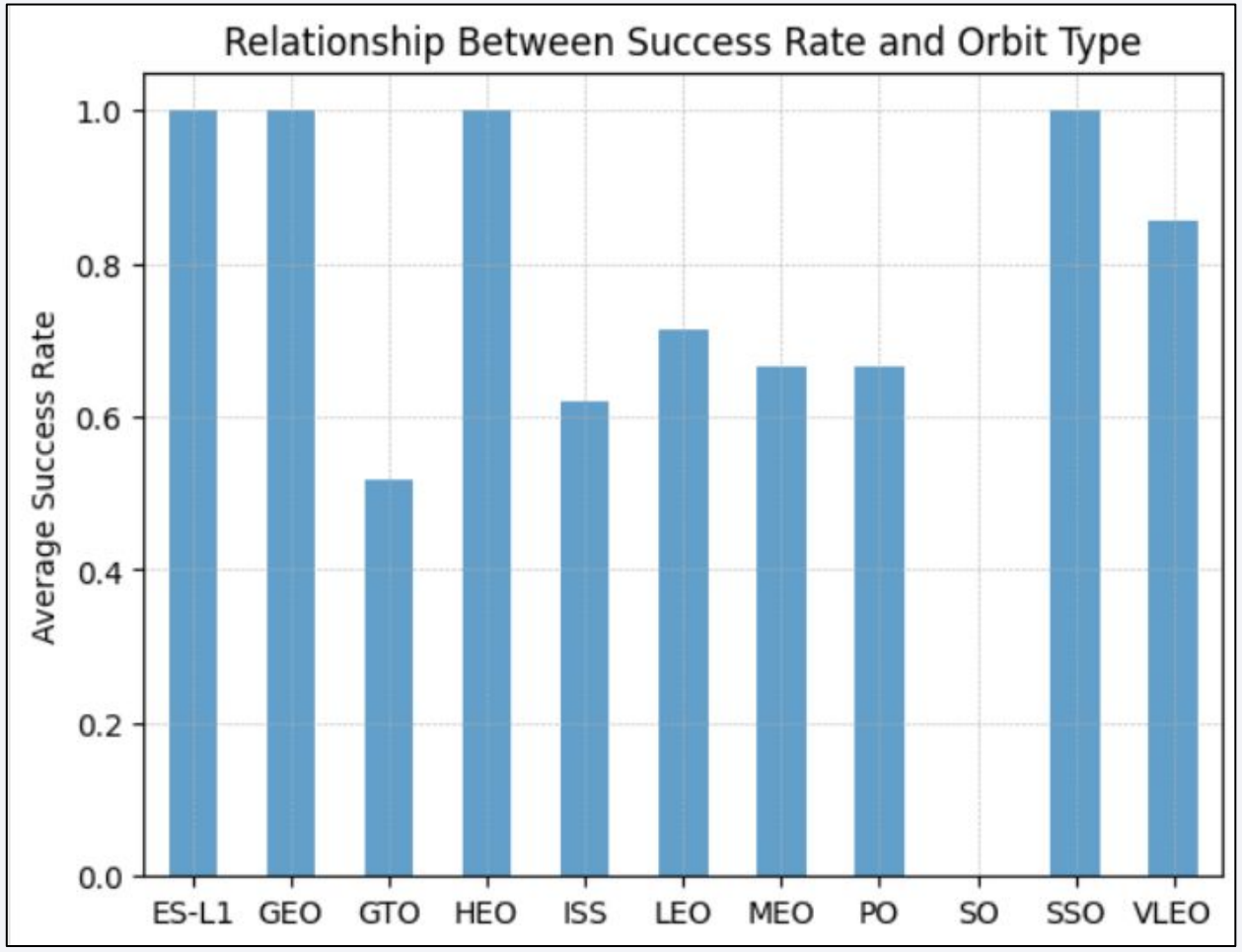
- Across all launch sites, there seems to be an improvement in the success rate over time;
- In the earlier flights, there are more failures (Class 0), but as flight numbers increase, the proportion of successful landings (Class 1) also increases;
- This indicates a trend of increasing reliability and success in rocket landings.

Payload vs. Launch Site



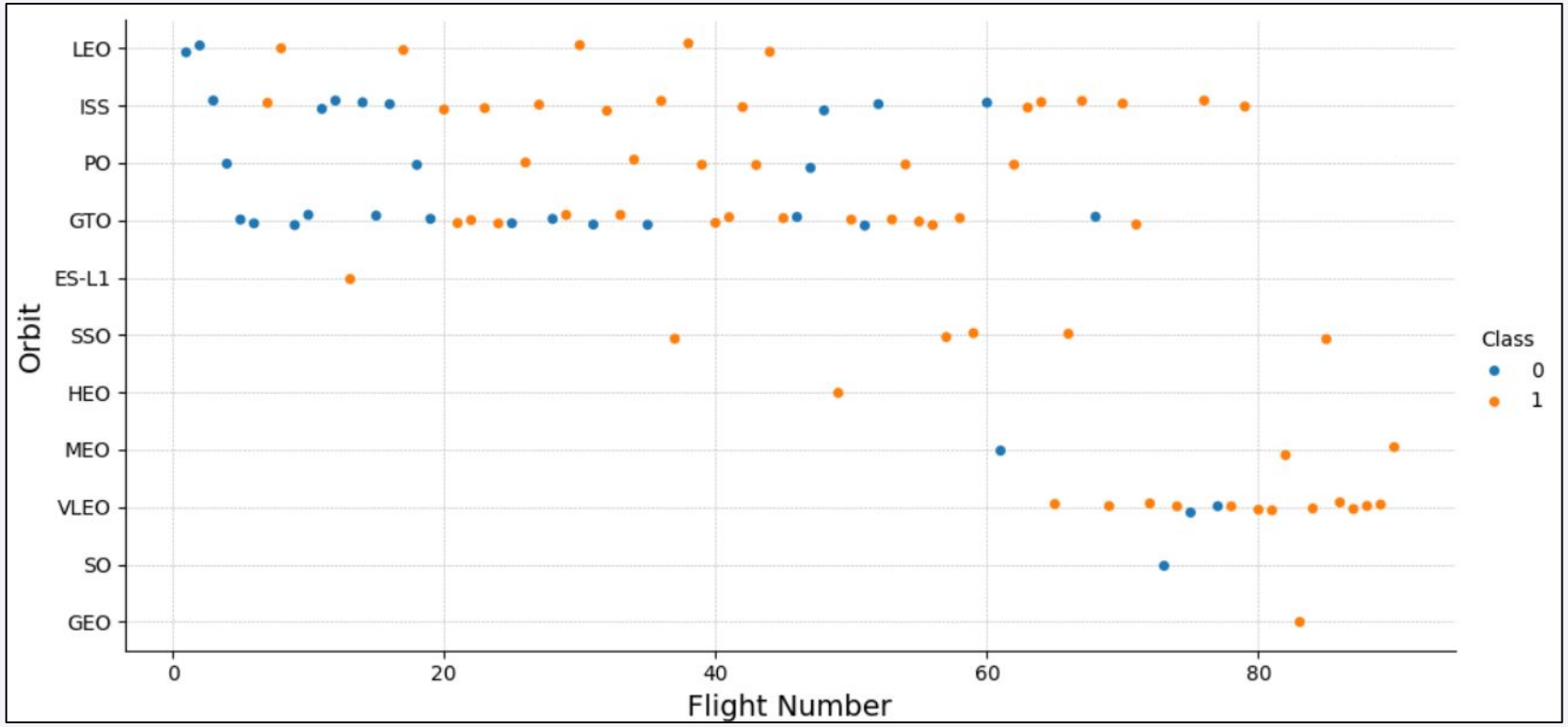
- For the VAFB-SLC launch site there are no rockets launched for heavy payload mass greater than 10000 kg.

Success Rate vs. Orbit Type



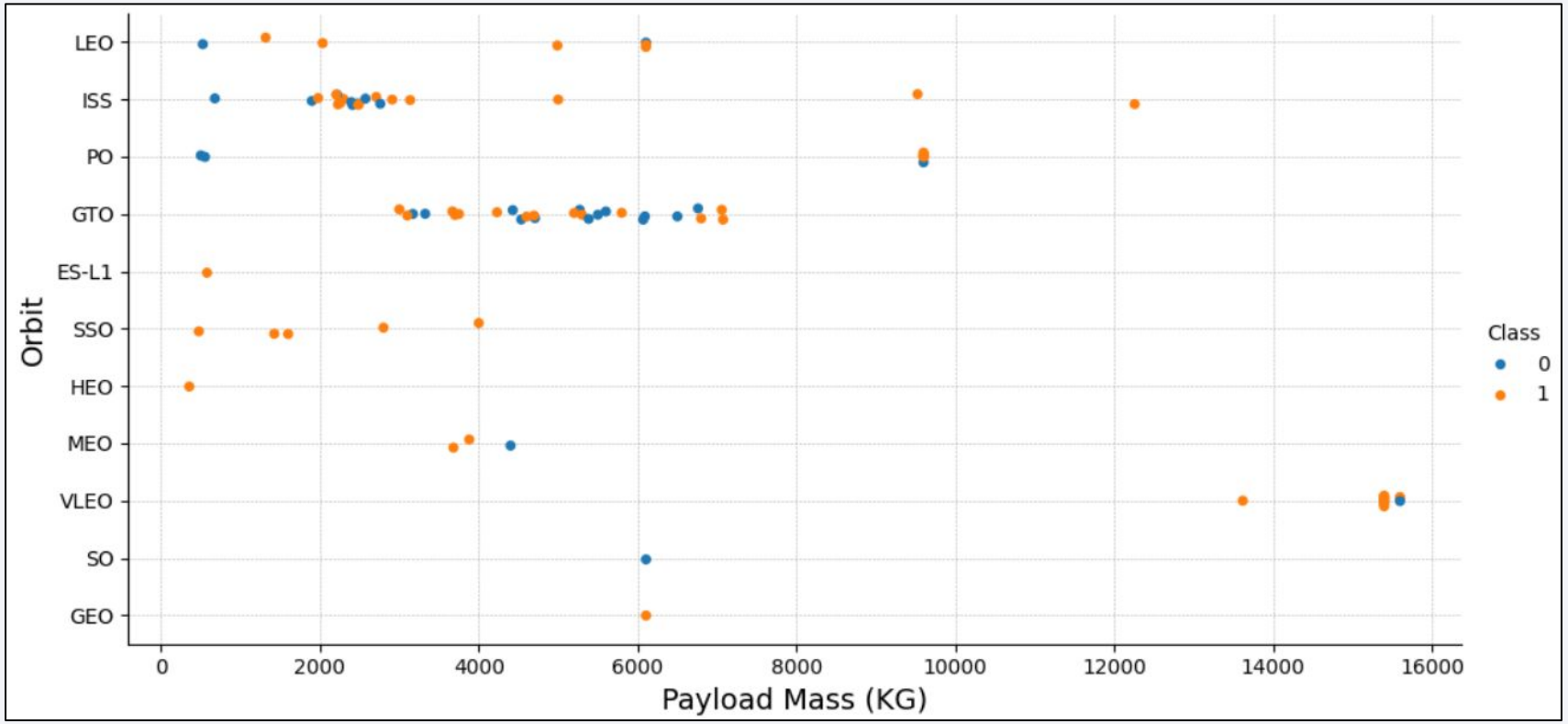
- We can see from the graph that orbits ES-L1, GEO, HEO, and SSO have the highest success rate;
- The SO orbit has the lowest success rate.

Flight Number vs. Orbit Type



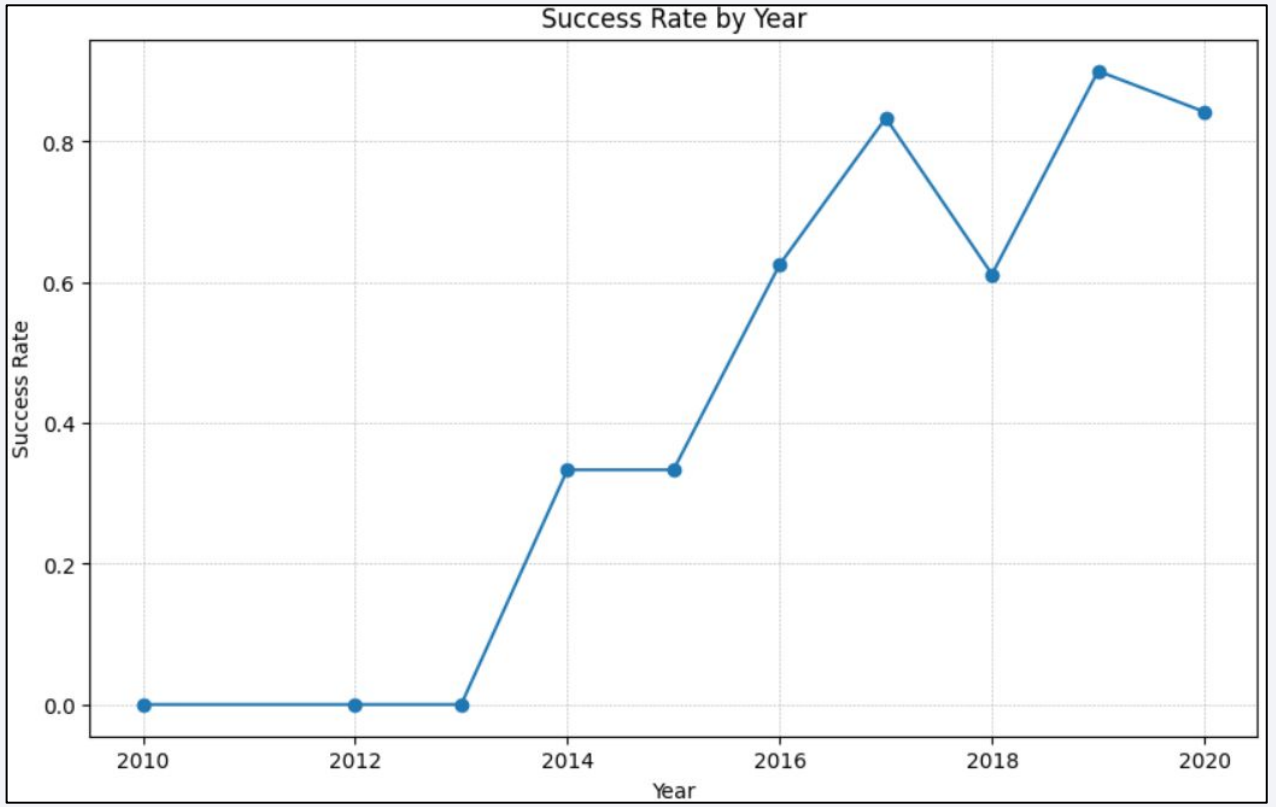
- We can see that in the LEO orbit the Success appears related to the number of flights;
- There seems to be no relationship between flight number when in GTO orbit.

Payload vs. Orbit Type



- With heavy payloads the successful landing rate is higher for PO, LEO and ISS;
- For GTO we cannot distinguish this well as both classes appear together.

Launch Success Yearly Trend



- We can observe that the success rate increased from 2013 to 2017;
- Between 2017 and 2018 and between 2019 and 2020 the success rate decreased.

All Launch Site Names

Task 1

Display the names of the unique launch sites in the space mission

```
[8]: %sql SELECT DISTINCT "Launch_Site" FROM SPACEXTABLE;
```

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

[8]:

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

- DISTINCT returns only unique values from the queries column "*Launch_Site*";
- There are 4 unique launch sites.

Launch Site Names Begin with 'CCA'

Task 2

Display 5 records where launch sites begin with the string 'CCA'

[9]:
```
%sql SELECT * FROM SPACEXTABLE WHERE "Launch_Site" LIKE "%CCA%" LIMIT 5;
```

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

[9]:

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

- Using keyword LIKE and format "*%CCA%*", returns records where *"Launch_Site"* column starts with "*CCA*";
- LIMIT 5, limits the number of returned records to 5.

Total Payload Mass

Task 3

Display the total payload mass carried by boosters launched by NASA (CRS)

```
[10]: %sql SELECT SUM("PAYLOAD_MASS__KG_") AS total_payload_mass FROM SPACEXTABLE WHERE "Customer" = "NASA (CRS)";
```

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

[10]:

total_payload_mass
45596

- SUM adds column "*PAYLOAD_MASS__KG_*" and returns total payload mass;
- WHERE customers named is "*NASA (CRS)*".

Average Payload Mass by F9 v1.1

Task 4

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

```
[11]: %sql SELECT AVG("PAYLOAD_MASS__KG_") AS average_payload_mass FROM SPACEXTABLE WHERE "Booster_version" LIKE "%F9 V1.1%";
```

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

[11]:

average_payload_mass
2534.6666666666665

- AVG keyword returns the average of payload mass in "*PAYLOAD_MASS_KG*" column;
- WHERE booster version is LIKE *"%F9 v1.1%*".

First Successful Ground Landing Date

Task 5

List the date when the first succesful landing outcome in ground pad was acheived.

Hint:Use min function

```
[12]: %sql SELECT MIN("Date") FROM SPACEXTABLE WHERE "Landing_Outcome" LIKE "%ground pad%" AND "Mission_Outcome" = "Success";
 * sqlite:///my_data1.db
Done.
[12]: MIN("Date")
      2015-12-22
```

- MIN selects the first or the oldest date from the "*Date*" column where first successful landing on group pad was achieved;
- WHERE clause defines the criteria to return date for scenarios where "*Landing_Outcome*" value is equal to "*Success*".

Successful Drone Ship Landing with Payload between 4000 and 6000

Task 6

List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000

[13]:
```
%%sql
SELECT DISTINCT "Booster_Version"
FROM SPACEXTABLE WHERE
    "Landing_Outcome" LIKE "%drone ship%" AND
    "Mission_Outcome" LIKE "%Success%" AND
    "PAYLOAD_MASS__KG_" > 4000 AND
    "PAYLOAD_MASS__KG_" < 6000;
```

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

[13]:

Booster_Version
F9 FT B1020
F9 FT B1022
F9 FT B1026
F9 FT B1021.2
F9 FT B1031.2

- The query returns the "*Booster_Version*" WHERE "*PAYLOAD_MASS__KG_*" is greater than 4000 but less than 6000, the "*Landing_Outcome*" is LIKE *"%drone ship%",* and "*Mission_Outcome*" is LIKE *"%Success%";*
- The AND operator returns the "*Booster_Version*" where all conditions are true.

Total Number of Successful and Failure Mission Outcomes

Task 7

List the total number of successful and failure mission outcomes

[14]:
```
%%sql
SELECT
    COUNT(CASE WHEN "Mission_Outcome" LIKE "%Success%" THEN 1 END) AS total_number_success,
    COUNT(CASE WHEN "Mission_Outcome" LIKE "%Failure%" THEN 1 END) AS total_number_failure
FROM SPACEXTABLE;
```

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

[14]:

total_number_success	total_number_failure
100	1

- We used the COUNT and the CASE WHEN clause to count each occurrence

Boosters Carried Maximum Payload

Task 8

List the names of the booster_versions which have carried the maximum payload mass. Use a subquery

[15]:
```
%%sql
SELECT DISTINCT "Booster_Version" FROM SPACEXTABLE WHERE "PAYLOAD_MASS__KG_" = (SELECT MAX("PAYLOAD_MASS__KG_") FROM SPACEXTABLE);
```

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

[15]:

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

- The subquery returns the maximum payload mass by using function MAX on the "*PAYLOAD_MASS__KG_*" column;
- The main query returns "*Booster_Version*" WHERE the "*PAYLOAD_MASS__KG_*" is maximum (i.e., with value of 15600).

2015 Launch Records

Task 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.

Note: SQLLite does not support monthnames. So you need to use substr(Date, 6,2) as month to get the months and substr(Date,0,5)='2015' for year.

[16]:
```
%%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_Name",
    "Booster_Version",
    "Launch_Site",
    "Mission_Outcome",
    "Landing_Outcome"
FROM SPACEXTABLE
WHERE
    "Landing_Outcome" LIKE "%drone ship%" AND
    "Mission_Outcome" LIKE "%Failure%" AND
    substr("Date", 0, 5) = '2015';
```

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

[16]:

Month_Name	Booster_Version	Launch_Site	Mission_Outcome	Landing_Outcome
June	F9 v1.1 B1018	CCAFS LC-40	Failure (in flight)	Precluded (drone ship)

- SELECT "*Landing_Outcome*", "*Booster_Versions*"," *Launch_Site*", and "*Month_name*";
- Where "*Landing_Outcome*" is LIKE "*drone ship*", "*Mission_Outcome*" is LIKE "*Failure*", and "*Date*" in the year "*2015*".

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

Task 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.

[17]:
```
%%sql
SELECT
    "Landing_Outcome",
    COUNT(*) AS "Count"
FROM SPACEXTABLE
WHERE "Date" BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY "Landing_Outcome"
ORDER BY "Count" DESC;
```

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

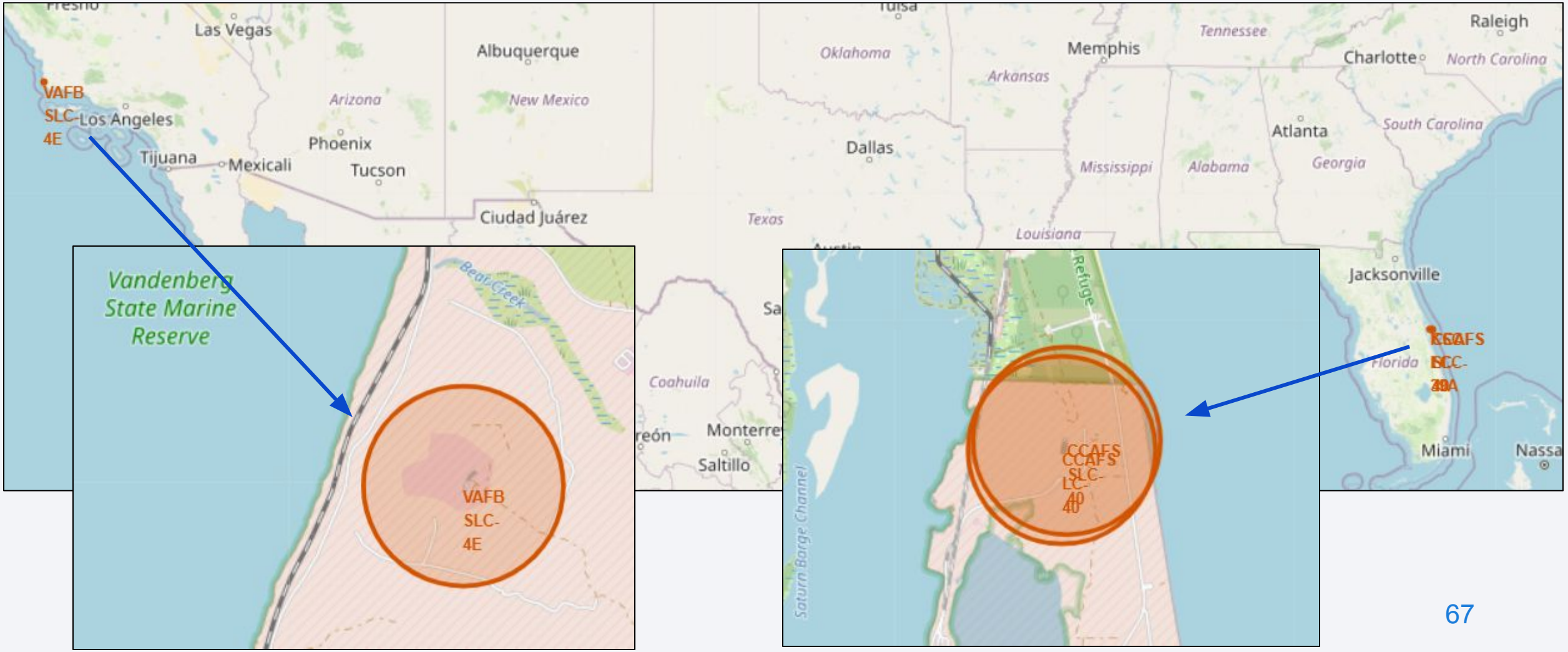
[17]:

Landing_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

- The GROUP BY clause arranges data in column *"Landing_Outcome"* into groups;
- The BETWEEN return data between "*2010-06-04*" and "*2017-03-20*";
- The ORDER BY arranges the counts column in descending order;
- The result is a ranked list of "*Landing_Outcome*" counts per the specified date.


Section 3
Launch Sites
Proximities Analysis

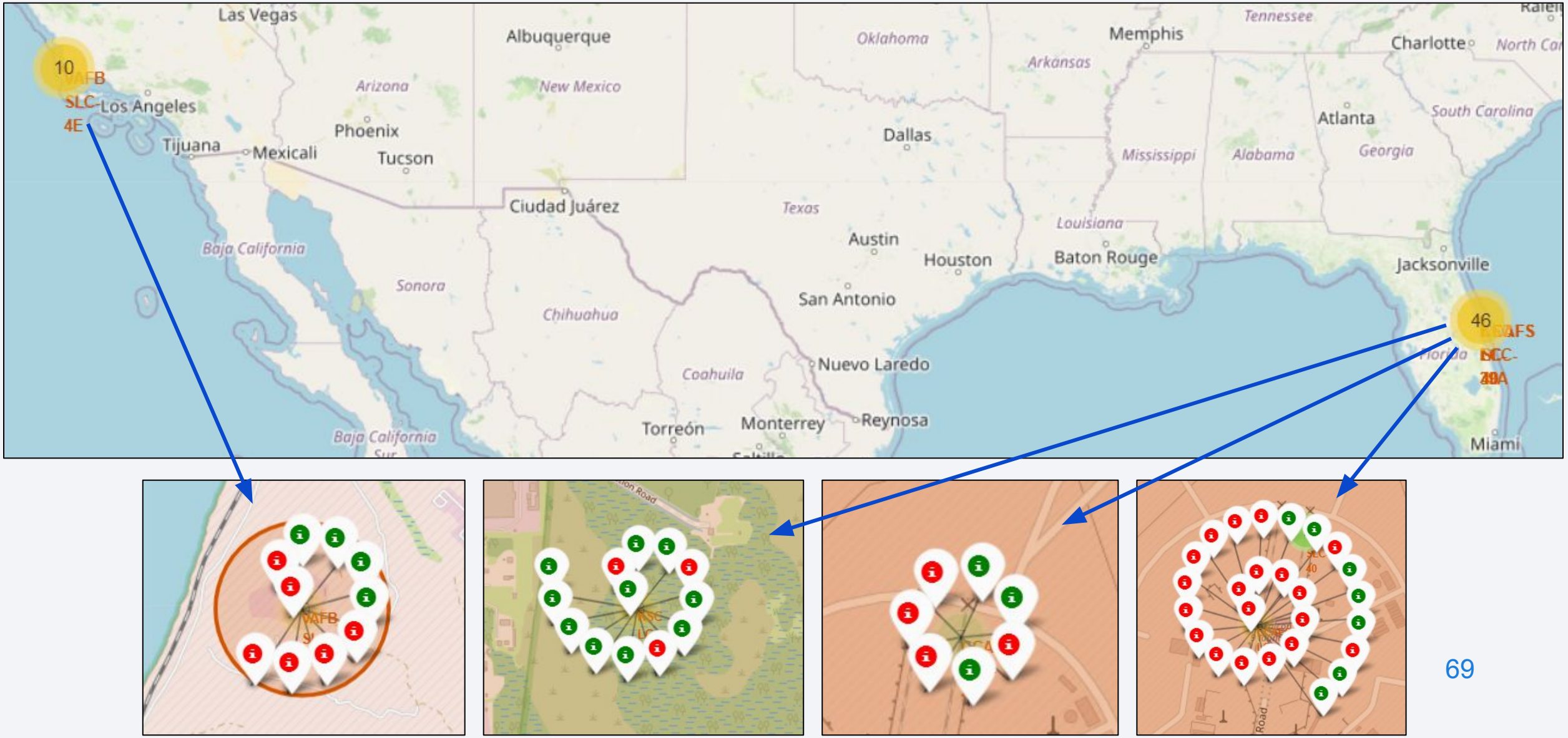
SpaceX Falcon9 - Launch Sites Map



SpaceX Falcon9 - Launch Sites Map

- The figures displays a global map highlighting Falcon 9 launch sites located in the United States, specifically in California and Florida;
- Each launch site is marked with a circle, label, and popup to indicate its location and name, demonstrating that all launch sites are situated near the coast;
- The four launch sites:
  - VAFB SLC-4E (CA);
  - CCAFS LC-40 (FL);
  - KSC LC-39A (FL);
  - CCAFS SLC-40 (FL).

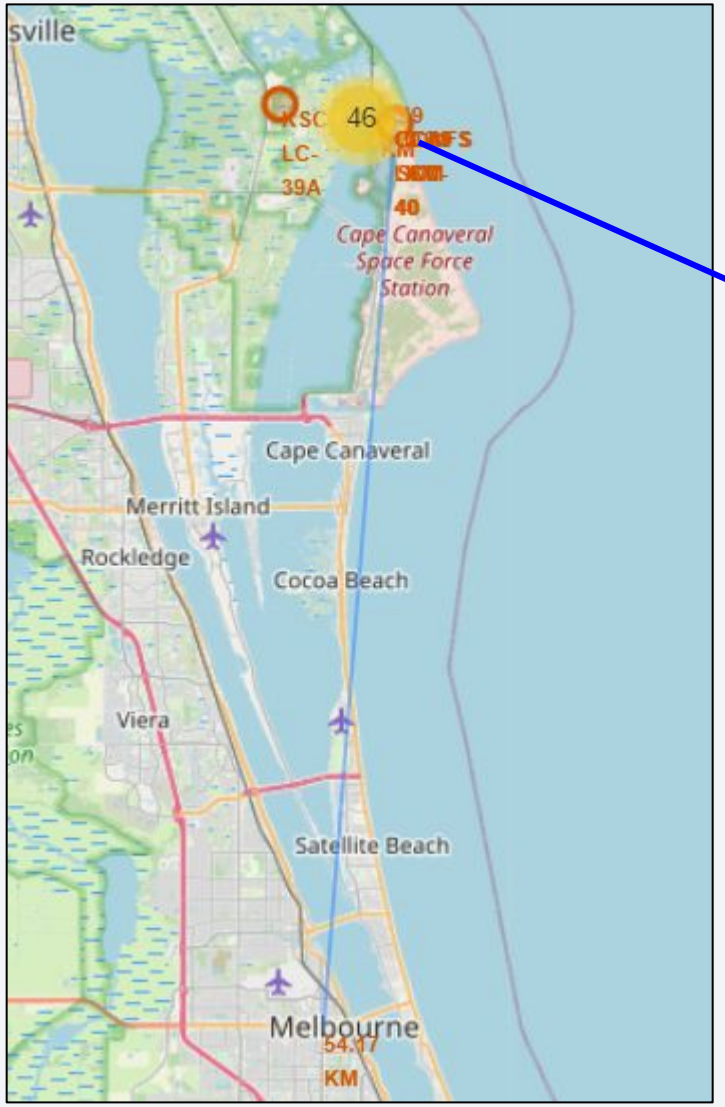
SpaceX Falcon9 – Success/Failed Launch Map for all Launch Sites

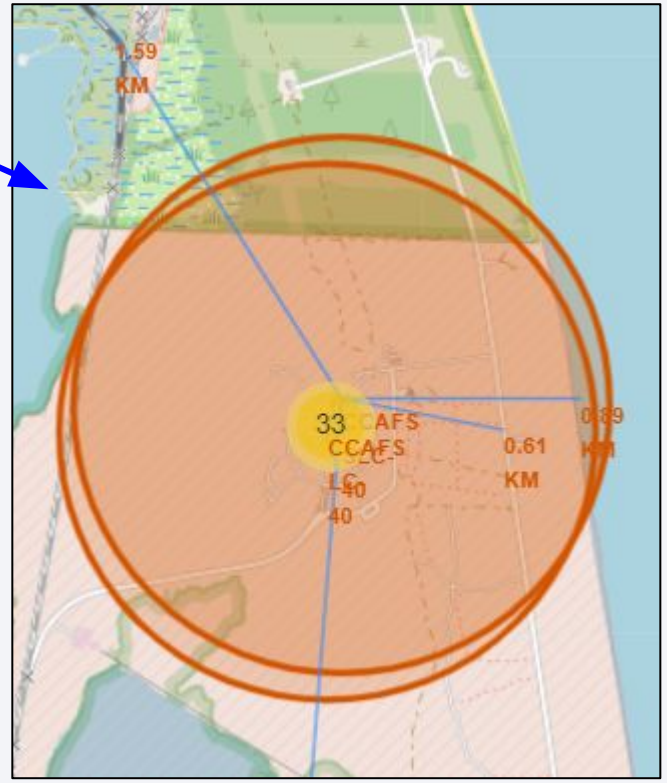


SpaceX Falcon9 – Success/Failed Launch Map for all Launch Sites

- The figure shows a map of the United States with all launch sites, where the numbers indicate the total count of successful and failed launches;
- For each site, we are displaying success (green) and failure (red) markers for launches;
- The KSC LC-39A Launch Site has the highest number of successful launches among all sites.

SpaceX Falcon9 – Launch Site to proximity Distance Map





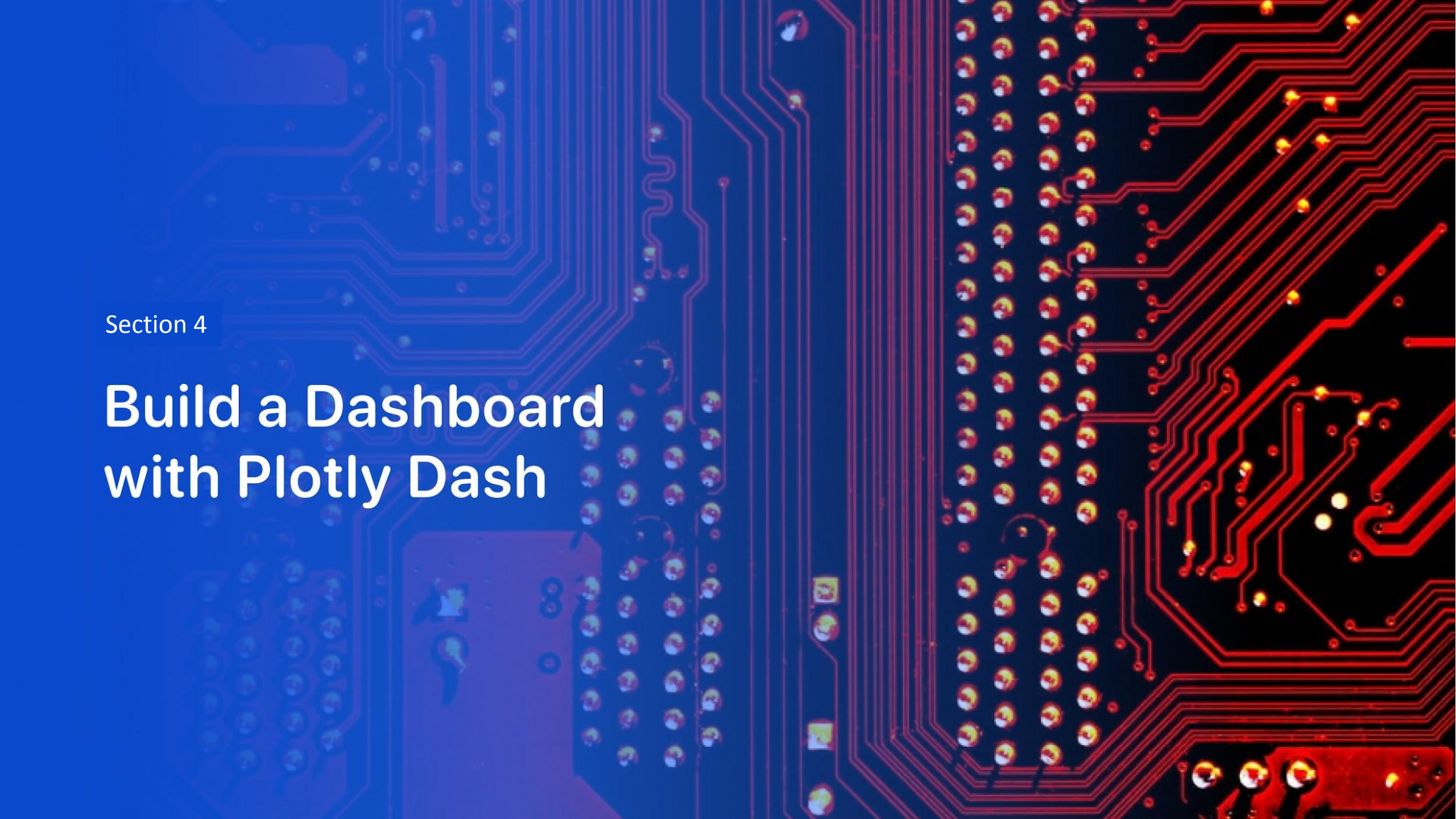
SpaceX Falcon9 – Launch Site to proximity Distance Map

- The figure shows all proximity sites for the CCAFS SLC-40 launch site, highlighting that the city of Melbourne is further away (54.09 km) compared to other proximities such as the coastline, railroad, and highway;

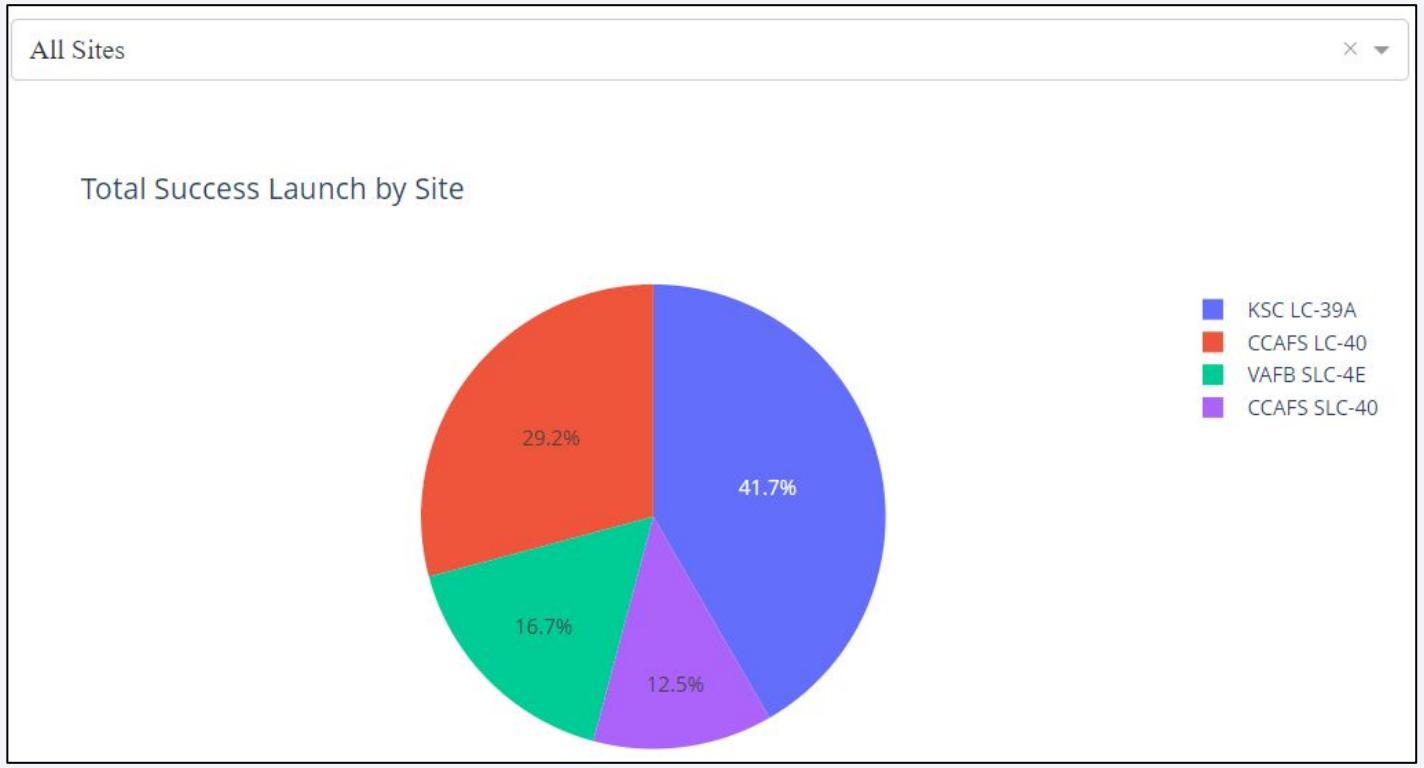
- The launch site is strategically positioned near resources to easy access:

  - Coastlines (0.89 km);
  - Railroads (1.59 km);
  - Highways (0.61 km).

- Cities are typically located away from launch sites to minimize risks to the public and infrastructure.

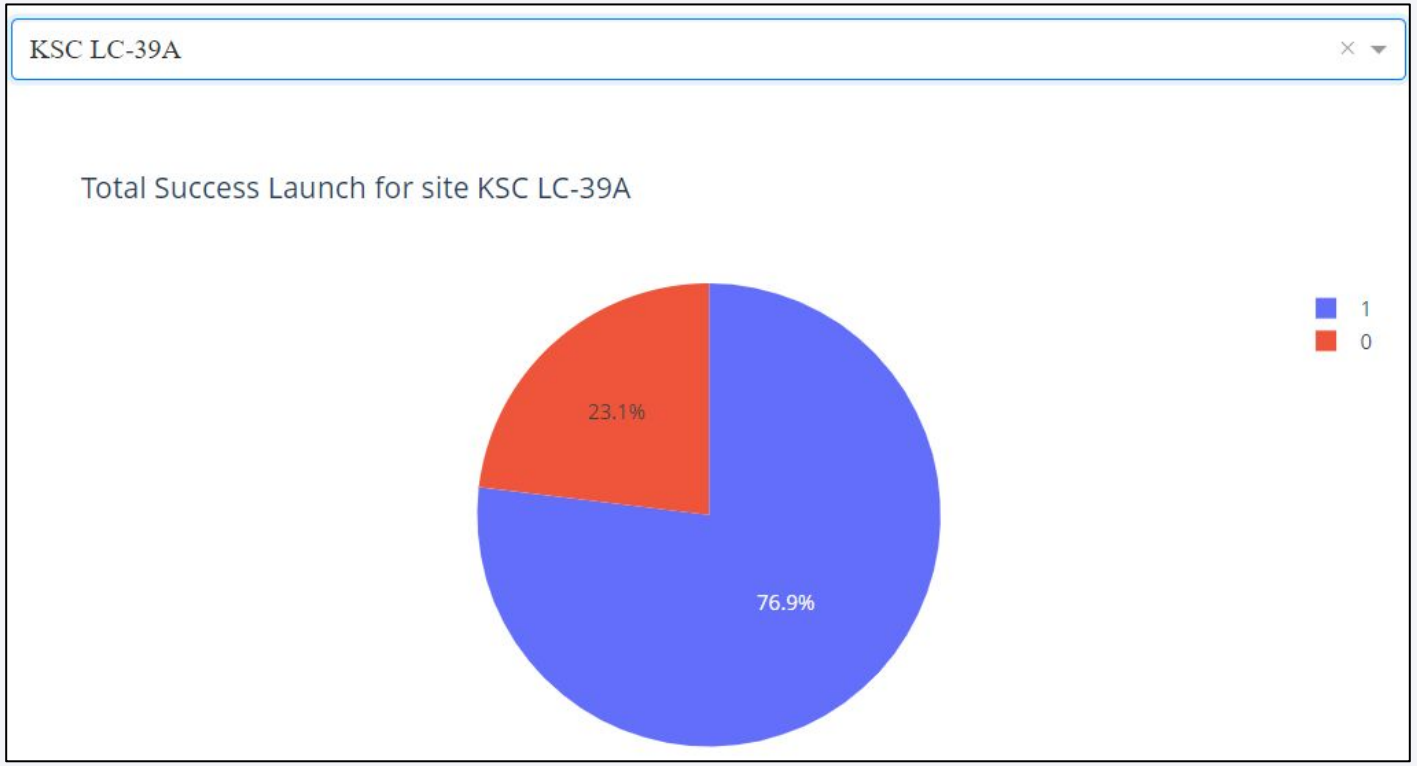

Section 4
Build a Dashboard with Plotly Dash

Launch Success Counts For All Sites



- The launch site "KSC LC-39A" has the highest launch success rate;
- The launch site "CCAFS SLC-40" has the lowest launch success rate.

Launch Site with Highest Launch Success Ratio



- The KSC LC-39A launch site has the highest launch success rate of 76.9%

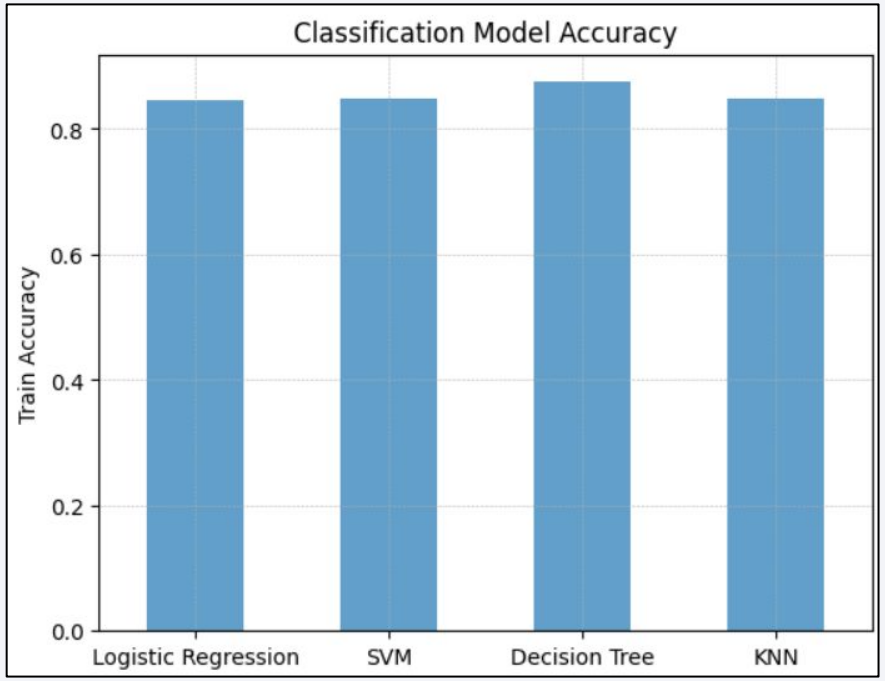
Payload vs. Launch Outcome Scatter Plot for All Sites

- Most successful launches have payloads ranging from 2000 to 5500 kg;
- The only booster with a successful launch for payloads greater than 8000 kg is 'B4'.

Section 5

Predictive Analysis (Classification)

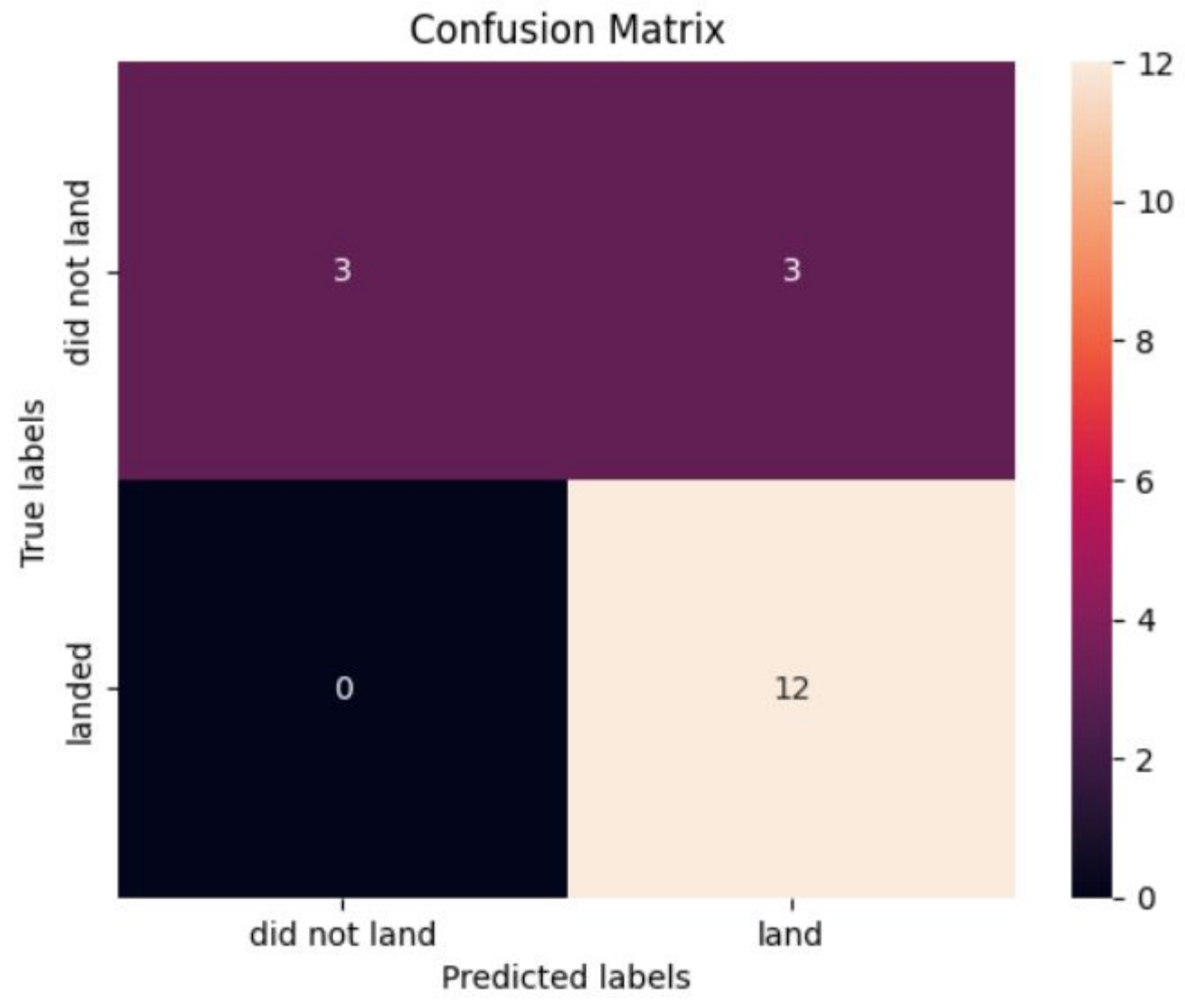
Classification Accuracy



	algorithm	train accuracy	test accuracy
0	Logistic Regression	0.846429	0.833333
1	SVM	0.848214	0.833333
2	Decision Tree	0.875000	0.944444
3	KNN	0.848214	0.833333

- The Decision Tree algorithm has the highest classification score with an accuracy of 0.8750, as shown in the bar chart;
- The accuracy score on the test data is 0.8333 for all classification algorithms;
- Due to the close accuracy scores and identical test scores, a broader dataset is needed for further model tuning.

Confusion Matrix



- The classifier made 18 predictions in total:
  - 12 true positives;
  - 3 true negatives;
  - 3 false positives.

- Overall, the classifier is correct about 83% of the time with a misclassification or error rate of about 16.5%.

Conclusions

- The likelihood of the first stage landing successfully increases with the number of flights;
- While success rates tend to rise with increasing payload mass, there is no clear correlation between payload mass and success rates;
- The launch success rate increased by approximately 80% from 2013 to 2020;
- KSC LC-39A has the highest success rate, while CCAFS SLC-40 has the lowest.

Conclusions

- Orbits ES-L1, GEO, HEO, and SSO have the highest success rates, whereas orbit GTO has the lowest;
- Launch sites are strategically placed away from cities and closer to coastlines, railroads, and highways;
- The Decision Tree is the best performing model with an accuracy of about 87.5%;
- All models scored an accuracy of about 83% on test data, indicating the need for more data to further tune the models.

Thank you!