IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Alejandro Cardona Murillo
February 18, 2022

Outline

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

Executive Summary

The main objective of this project is to analyze the SpaceX Falcon 9 data and develop and implement ML models to predict the success rate of the first stage landing.

- Summary of methodologies
  - Data Collection.
  - Data Wrangling.
  - EDA with data visualization.
  - EDA with SQL.
  - Building an interactive map with Folium.
  - Building a Dashboard with Plotly Dash.
  - Predictive Analysis (Clasification).
- Summary of all results
  - Exploratory Data Analysis (EDA) results.
  - Interactive Analytics.
  - Predictive Analytics results.

Introduction

- Project background and context
  - In recent times private space travel companies are taking giant steps into making space travel accessible to the general public, these companies are trying to reduce the cost of each launch by different means. One of the main companies in this area is SpaceX. SpaceX advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is because SpaceX can reuse the first stage, this gives SpaceX a huge advantage versus their competition.
- Problems you want to find answers
  - In this project, we will predict if the Falcon 9 first stage will land successfully.
  - Impact that different parameters or variables have in the landing outcome.
  - Correlation between launch site and success rate.

Section 1

Methodology

Methodology

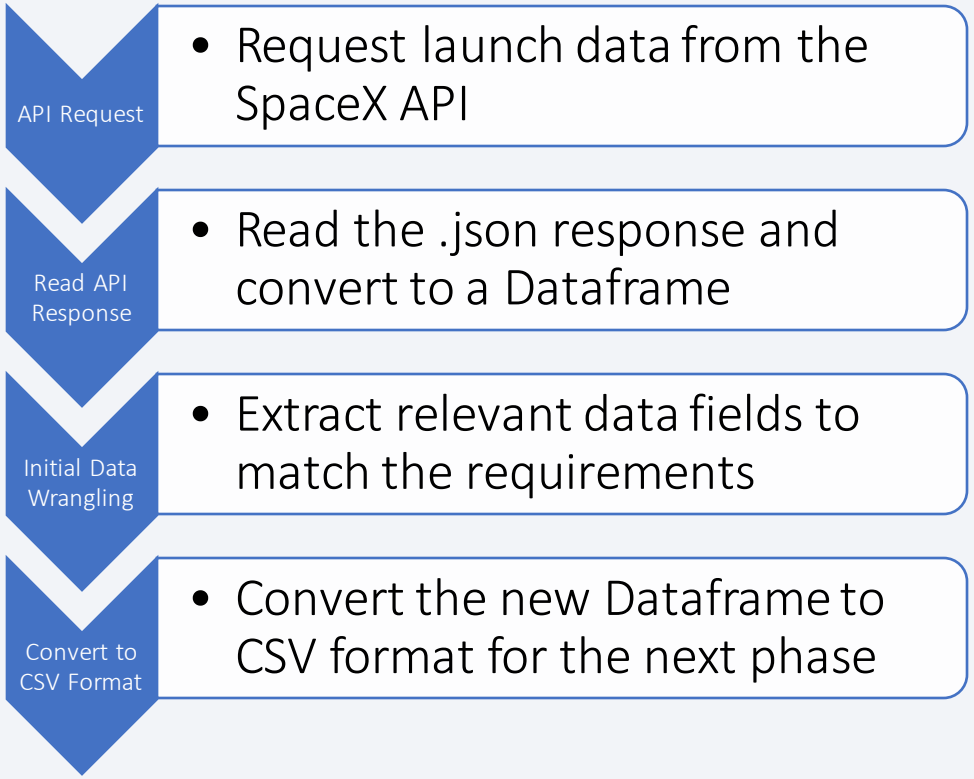
Executive Summary

- Data collection methodology:
  - SpaceX API.
  - Web Scrapping of Falcon 9 and Falcon Heavy launch records from Wikipedia.
- Perform data wrangling
  - Created a landing outcome label from the Outcome column to train the supervised models (Failure – 0, Success - 1)
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - Standardize and transform the data, split the data into train and test sets, build the models with different algorithms (Logistic regression, SVM, decision tree, & KNN), use Grid Search to find the best parameters for each algorithm, evaluate each model with test data and use a confusion matrix to check each model.

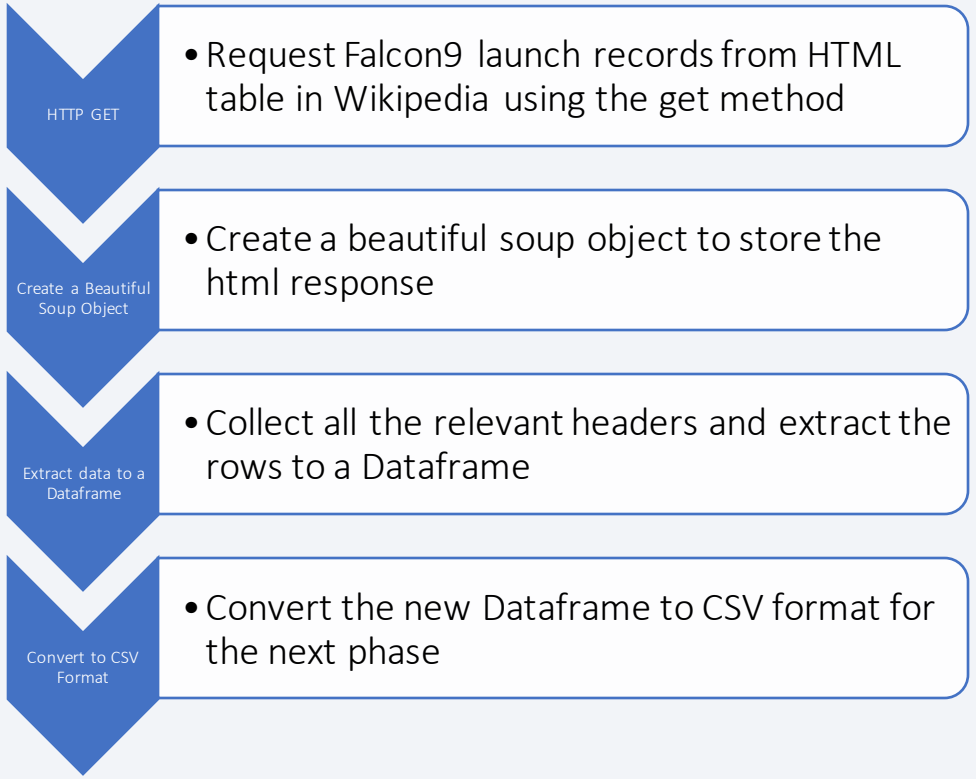
Data Collection

For the project, the data was collected via SpaceX API and Web scrapping Wiki pages for relevant launch data.

SpaceX API



Web Scrapping from Wikipedia



Data Collection – SpaceX API

1. API request, read response and turn into a Dataframe
2. Declare Global Variables
3. Call helper functions with API calls to populate the variables
4. Construct the dataset with the obtained data
5. Convert dict to Dataframe, filter for Falcon 9 launches and convert to CSV

1. API GET request, normalize data and save into a Dataframe.

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

response = requests.get(spacex_url)

# Use json_normalize meethod to convert the json result into a dataframe

data = pd.json_normalize(response.json())
```

2. Declare Global variables to store the data returned by the helper functions using the API calls.

```
#Global variables
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
ReusedCount = []
Serial = []
Longitude = []
Latitude = []
```

3. Call the helper functions to get the relevant data.

```
# Call getBoosterVersion
getBoosterVersion(data)
```

```
# Call getLaunchSite
getLaunchSite(data)
```

```
# Call getPayloadData
getPayloadData(data)
```

```
# Call getCoreData
getCoreData(data)
```

4. Construct the dataset from the obtained data and combine the columns into a dictionary.

```
launch_dict = {'FlightNumber': list(data['flight_number']),
'Date': list(data['date']),
'BoosterVersion':BoosterVersion,
'PayloadMass':PayloadMass,
'Orbit':Orbit,
'LaunchSite':LaunchSite,
'Outcome':Outcome,
'Flights':Flights,
'GridFins':GridFins,
'Reused':Reused,
'Legs':Legs,
'LandingPad':LandingPad,
'Block':Block,
'ReusedCount':ReusedCount,
'Serial':Serial,
'Longitude': Longitude,
'Latitude': Latitude}
```

5. Convert the dictionary to a Dataframe, filter the Dataframe to only include Falcon9 launches and then export it to a csv.

```
# Create a data from launch_dict
data = pd.DataFrame.from_dict(launch_dict)
```

```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = data[data['BoosterVersion']!='Falcon 1']
```

```
data_falcon9.to_csv('dataset_part\_1.csv', index=False)
```

GitHub

Data Collection – Scraping

1. Perform an HTTP GET request
2. Create a Beautiful soup object
3. Extract column names from HTML table headers
4. Create a dictionary with the keys extracted from the column names
5. Call the helper functions to fill the dictionary with launch records
6. Convert the dictionary to a Dataframe and then to CSV

1. Perform an HTTP GET method to request the Falcon9 Launch HTML page, as an HTTP response.

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

```
response = requests.get(static_url)
```

2. Create a Beautiful Soup object.

```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(response.text, "html.parser")
```

3. Extract all column/variable names from the HTML table header.

```
# 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')
```

```
column_names = []

# Apply find_all() function with `th` element on j
# Iterate each th element and apply the provided e
# Append the Non-empty column name (`if name is no
for header in first_launch_table.find_all('th'):
    name = extract_column_from_header(header)
    if name is not None and len(name) > 0:
        column_names.append(name)
```

4. Create an empty dictionary with keys from the extracted column names, to then convert it to a pandas Dataframe.

```
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 to be an empty list
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']=[]
```

6. Convert the dictionary to a Dataframe and the that Dataframe to CSV.

```
df=pd.DataFrame(launch_dict)
```

```
df.to_csv('spacex_web_scraped.csv', index=False)
```

5. Use the helper functions to fill up the launch dictionary with the launch records (examples of the functions and the calls).

```
def booster_version(table_cells):
    """
    This function returns the booster version from the HTML  table cell
    Input: the  element of a table data cell extracts extra row
    """
    out=''.join([booster_version for i,booster_version in enumerate( table_cells.strings) if i%2==0][0:-1])
    return out
```

```
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
```

```
# 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)
```

GitHub

Data Wrangling

For this process, after reading the data, there were made some verifications like the percentage of missing values and the type of each column, then we calculated the number of launches on each site, the number and occurrence of each orbit and mission outcome per orbit type, and finally created a landing outcome label.

1. Load the data into a Dataframe
2. Calculate the number of launches on each site
3. Calculate the number and occurrence of each orbit
4. Calculate the number and occurrence of mission outcome per orbit type
5. Create a landing outcome label from the Outcome column
6. Export the Dataframe to CSV

1. Load the CSV into a Dataframe.

```
df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_1.csv")
```

2. Calculate the number of launches on each site.

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

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

3. Calculate the number and occurrence of each orbit.

```
# Apply value_counts on Orbit column
df['Orbit'].value_counts()
```

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

4. Calculate the number and occurrence of mission outcome per orbit type.

```
# landing_outcomes = values on Outcome column
landing_outcomes = df['Outcome'].value_counts()
landing_outcomes
```

```
True ASDS       41
None None       19
True RTLS       14
False ASDS       6
True Ocean       5
None ASDS        2
False Ocean      2
False RTLS       1
Name: Outcome, dtype: int64
```

6. Export the Dataframe to CSV.

```
df.to_csv("dataset_part\_2.csv", index=False)
```

5. Create a landing outcome label from Outcome column.

```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise

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

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

	Class
0	0
1	0
2	0
3	0
4	0
5	0
6	1
7	1

GitHub

EDA with Data Visualization

For the exploratory data analysis with data visualization, the following charts were plotted to gain further insights:

1. Scatter plot:
   - This type of plot shows the correlation between two variables making the possible patterns easy to detect.
   - Plotted charts:
     - Relationship between Flight Number and Launch Site.
     - Relationship between Payload and Launch Site.
     - Relationship between Flight Number and Orbit type.
     - Relationship between Payload and Orbit type.

2. Bar chart:
   - This type of chart is used to compare the values of a variable at a given time. This charts make it easy to detect which groups are more common and how they compare to each other.
   - Plotted chart:
     - Success rate of each Orbit type.

3. Line chart:
   - This type of chart is commonly used to check the changes of a variable over a period of time and helps to identify trends.
   - Plotted chart:
     - Average Launch Success yearly trend.

GitHub

EDA with SQL

For further understanding the SpaceX dataset, the following SQL queries were performed on an IBM Db2 instance:

- Display the names of the unique launch sites in the space mission.
- Display 5 records where launch sites begin with the string 'CCA'.
- Display the total payload mass carried by boosters launched by NASA (CRS).
- Display average payload mass carried by booster version F9 v1.1.
- List the date when the first successful landing outcome in ground pad was achieved.
- List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000.
- List the total number of successful and failure mission outcomes.
- List the names of the booster_versions which have carried the maximum payload mass. Use a subquery.
- List the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015.
- 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.

GitHub

Build an Interactive Map with Folium

- Developing an interactive map with Folium helps to analyze geospatial data implementing visual analytics to improve the understanding of different factors like location and proximity of launch sites, to finally check the impact of those factors to the success rate of the launches.
- Map objects created and added to the map:
  - Mark all launch sites on the map. This markers help to identify the launch sites on the map.
    - Added a 'folium.circle' object and a 'folium.marker' object to highlight a circle area with a text label over each launch site.
  - Mark the success/failed launches for each site on the map.
    - Added a 'MarkerCluster()' object to show launch successes (green) and failures (red) markers for each launch site.
  - Calculate the distances between a launch site to its proximities.
    - Added a 'MousePosition()' object to get coordinate for the mouse over a point on the map.
    - Added a 'folium.Marker()' object to display the distance in km from the point on the map to the nearest launch site.
    - Added a 'folium.Polyline()' object to draw a line between the point on the map and the launch site.
    - Repeated the process above to add the markers and draw the lines between the launch site and its proximities (coastline, railroad, highway, city)

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. Plots, graphs and interactions added:
  - Launch Site Drop-down Input component to provide filtering functions to the dashboard visualizations by all launch sites or a particular launch site.
  - Pie chart to show the total success launches when 'All Sites' is selected and to show the success and failure count when a particular launch site is selected.
  - Payload range slider to facilitate the selection of different payload ranges to identify visual patterns with the dashboard.
  - Scatter chart to observe the possible correlation between Payload and mission outcomes for the selected sites.
- The dashboard helped to answer the following questions:
  - Which site has the largest successful launches? **KSC LC-39A with 10**
  - Which site has the highest launch success rate? **KSC LC-39A with a 76.9% success rate**
  - Which payload range(s) has the highest launch success rate? **2000 – 5000 kg**
  - Which payload range(s) has the lowest launch success rate? **0 – 2000 kg and 5500 – 7000 kg**
  - Which F9 Booster version (v1.0, v1.1, FT, B4, B5, etc.) has the highest launch success rate? **FT**

GitHub

Predictive Analysis (Classification)

This was the process to find the best performing model: standardize and transform the data, split the data into train and test sets, build the models with different algorithms (Logistic regression, SVM, decision tree, & KNN), use Grid Search to find the best parameters for each algorithm, evaluate each model with test data and use a confusion matrix to check each model.

1. Load the data into a Dataframe and create a 'Class' array
2. Standardize the data
3. Split the data into train and test data sets
4. Create the models and find the best parameters
5. Evaluate each model with test data
6. Find the best performing model

1. Load the SpaceX CSV into a Dataframe and create a Numpy array from the class column.

```
data = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_2.csv")
```

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

2. Standardize the data in X then reassign it to the variable X.

```
# students get this
scaler = preprocessing.StandardScaler()
```

```
X = scaler.fit_transform(X)
```

3. Split the data into train and test sets.

```
# Split data for training and testing data sets
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
print ('Train set:', X_train.shape,  Y_train.shape)
print ('Test set:', X_test.shape,  Y_test.shape)
```

```
Train set: (72, 83) (72,)
Test set: (18, 83) (18,)
```

4. Create the models and find the best parameters with Grid Search (logistic regression example).

```
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, parameters, cv=10)
logreg_cv.fit(X_train, Y_train)
```

```
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
```

5. Evaluate each model with test data (logistic regression example).

```
print("Test data accuracy score - Logistic Regression: ", logreg_cv.score(X_test, Y_test))
```

```
Test data accuracy score - Logistic Regression:  0.8333333333333334
```

6. Find the best performing model.

```
# Create a DF for algorithm type and respective best scores

Model_Performance_df = pd.DataFrame({'Algo Type': ['Logistic Regression', 'SVM','Decision Tree','KNN'],
'Accuracy Score': [logreg_cv.best_score_, svm_cv.best_score_, tree_cv.best_score_, knn_cv.best_score_],
'Test Data Accuracy Score': [logreg_cv.score(X_test, Y_test), svm_cv.score(X_test, Y_test),
tree_cv.score(X_test, Y_test), knn_cv.score(X_test, Y_test)]})
```

```
Model_Performance_df.sort_values(['Accuracy Score'], ascending = False, inplace=True)
Model_Performance_df
```

	Algo Type	Accuracy Score	Test Data Accuracy Score
2	Decision Tree	0.887500	0.833333
3	KNN	0.848214	0.833333
1	SVM	0.848214	0.833333
0	Logistic Regression	0.846429	0.833333

```
i = Model_Performance_df['Accuracy Score'].idxmax()
print('The best performing alogrithm is '+ Model_Performance_df['Algo Type'][i]
      + ' with score ' + str(Model_Performance_df['Accuracy Score'][i]))
```

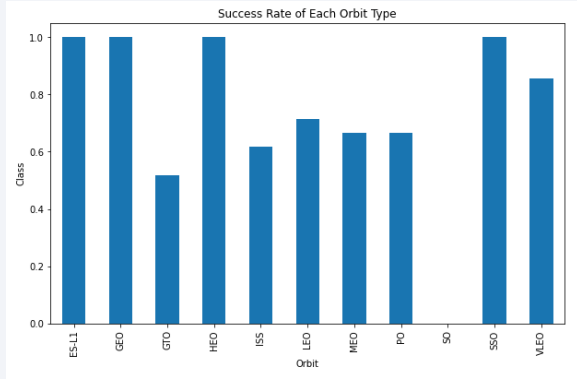
```
The best performing alogrithm is Decision Tree with score 0.8875
```

GitHub

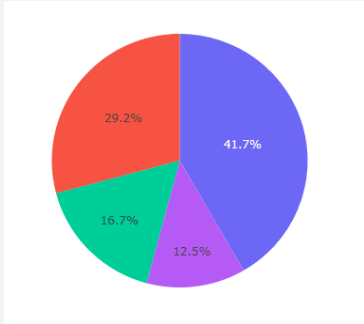
Results

The following sections explain the obtained results.

- Exploratory data analysis results



- Interactive analytics demo in screenshots

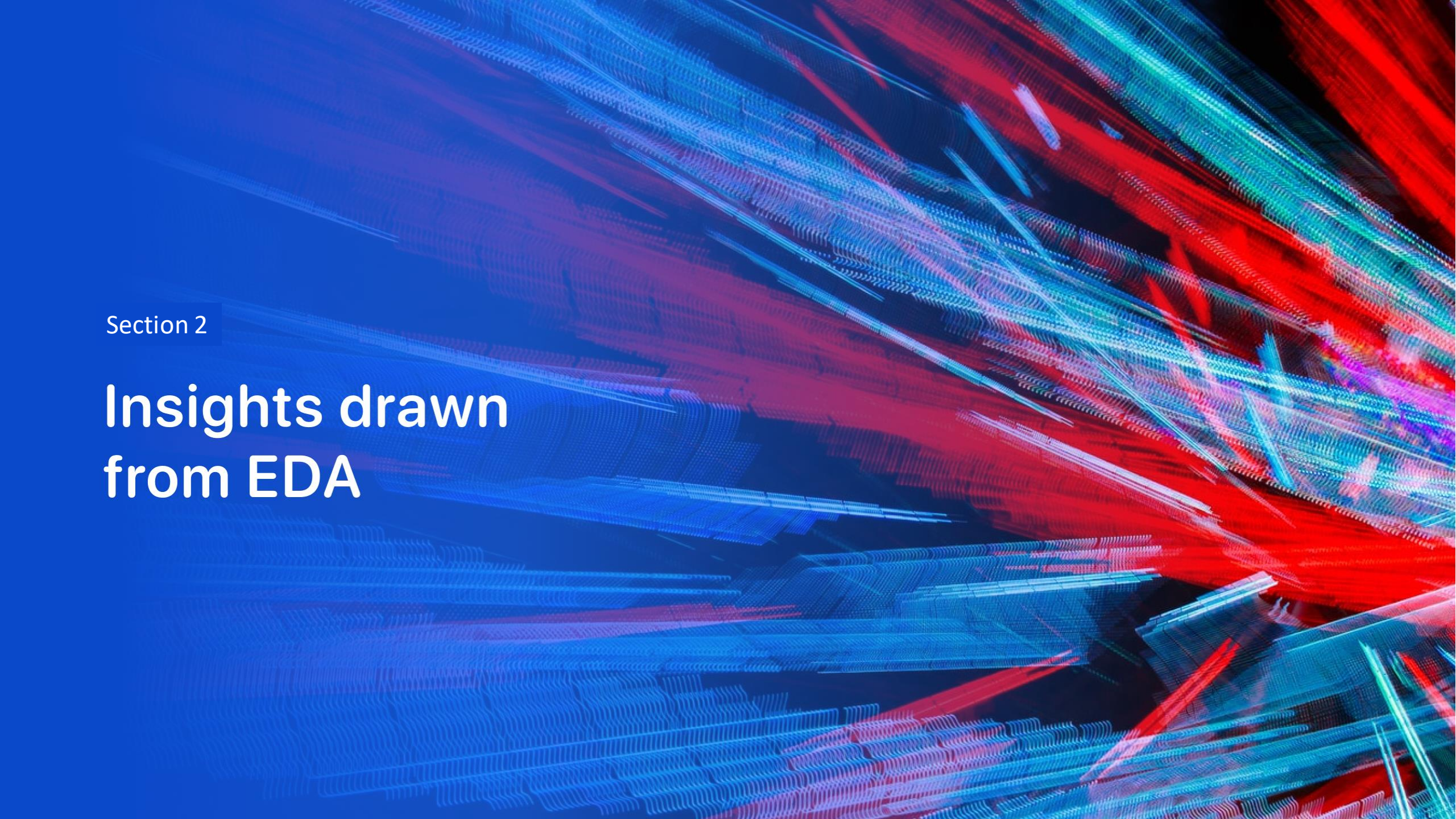


- Predictive analysis results

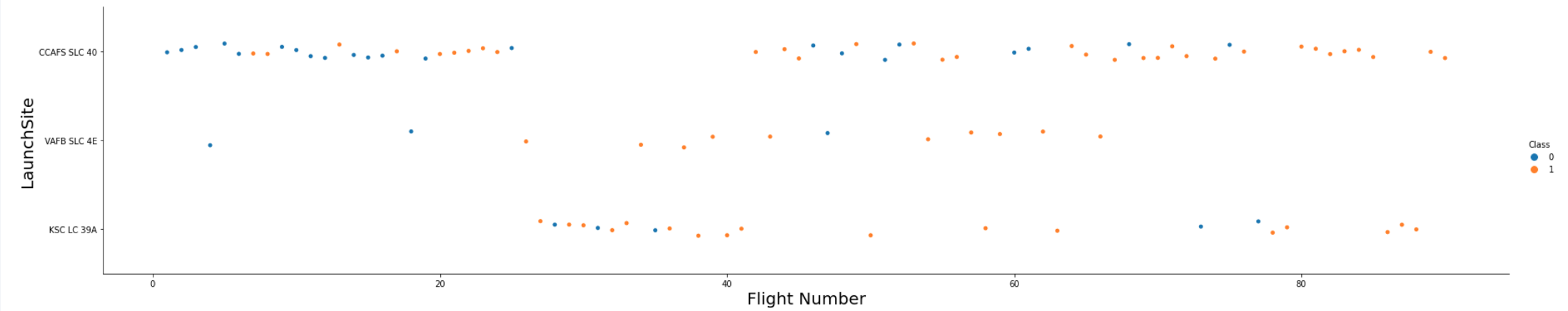
	Algo Type	Accuracy Score	Test Data Accuracy Score
2	Decision Tree	0.887500	0.833333
3	KNN	0.848214	0.833333
1	SVM	0.848214	0.833333
0	Logistic Regression	0.846429	0.833333

Section 2

Insights drawn from EDA

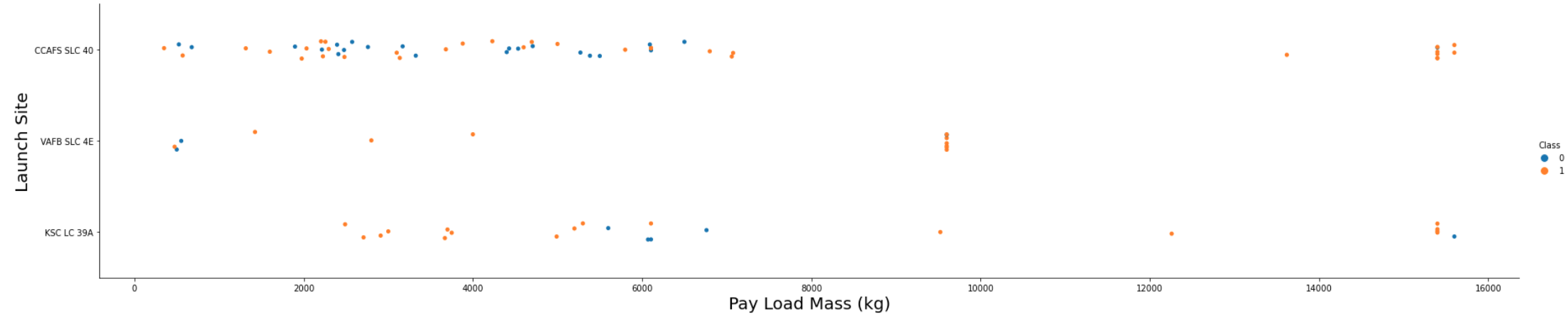


Flight Number vs. Launch Site



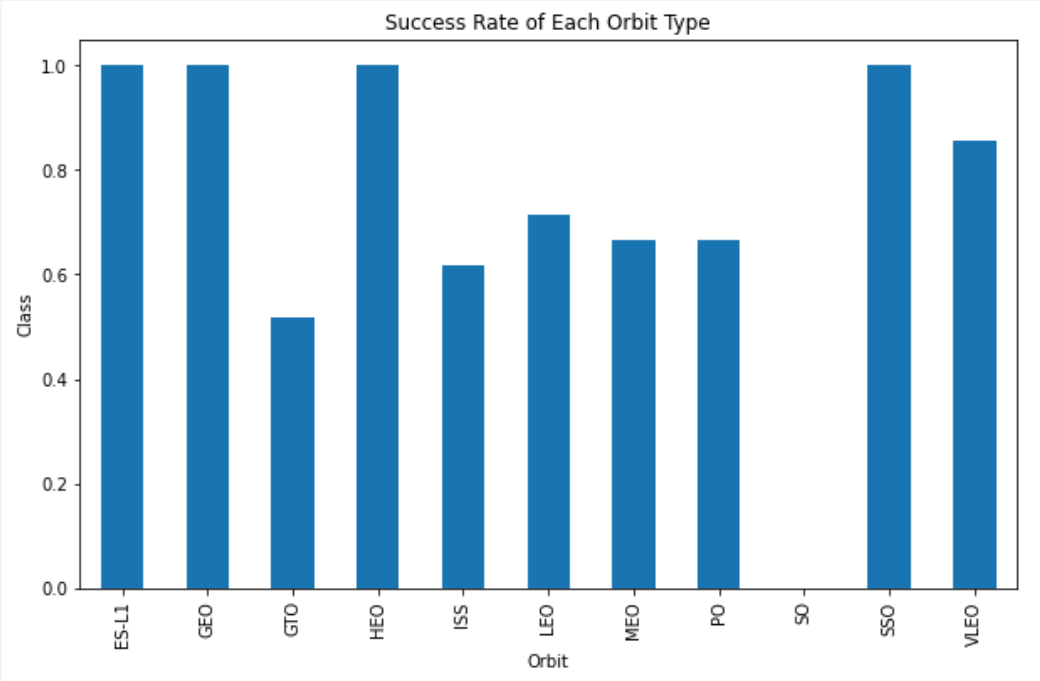
- Success rates (Class=1) increases as the number of flights increase.
- For the launch site KSC LC-39A, it takes at least approximately 25 launches before the first successful launch.

Payload vs. Launch Site



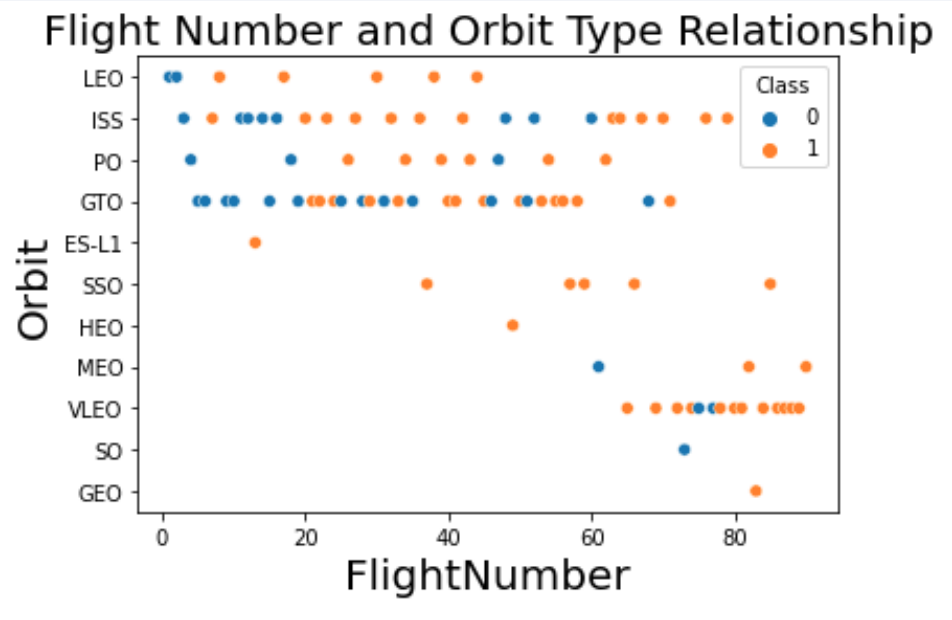
- For the VAFB SLC 4E launch site there are no launches for a payload greater than 10.000 kg.
- The CCAFS SLC 40 launch site has its best performance rate with a high payload mass, around 16.000 kg.

Success Rate vs. Orbit Type



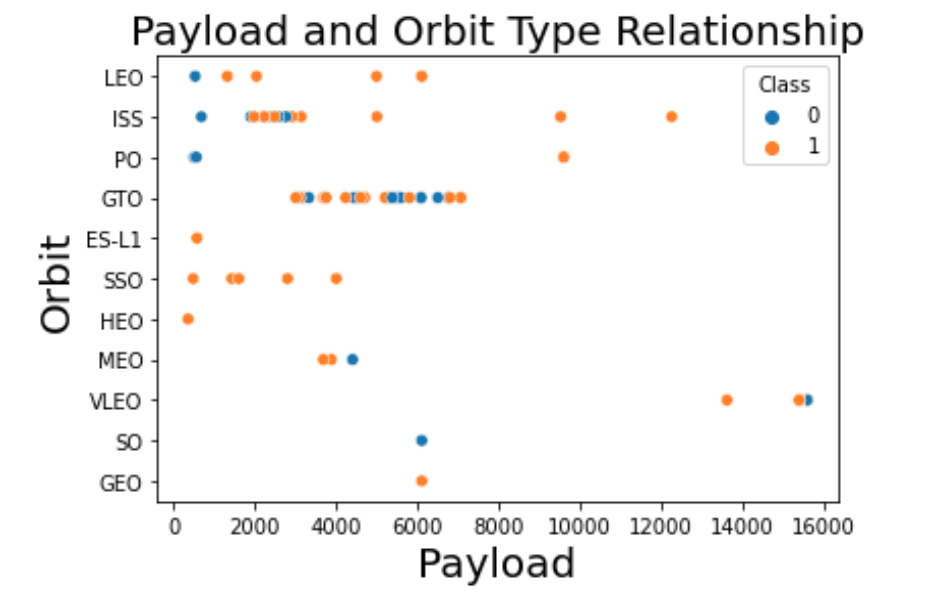
- The orbit types with the best success rates are ES-LI, GEO, HEO, SSO.
- The GTO orbit type has the lowest success rate, around 0.5.

Flight Number vs. Orbit Type



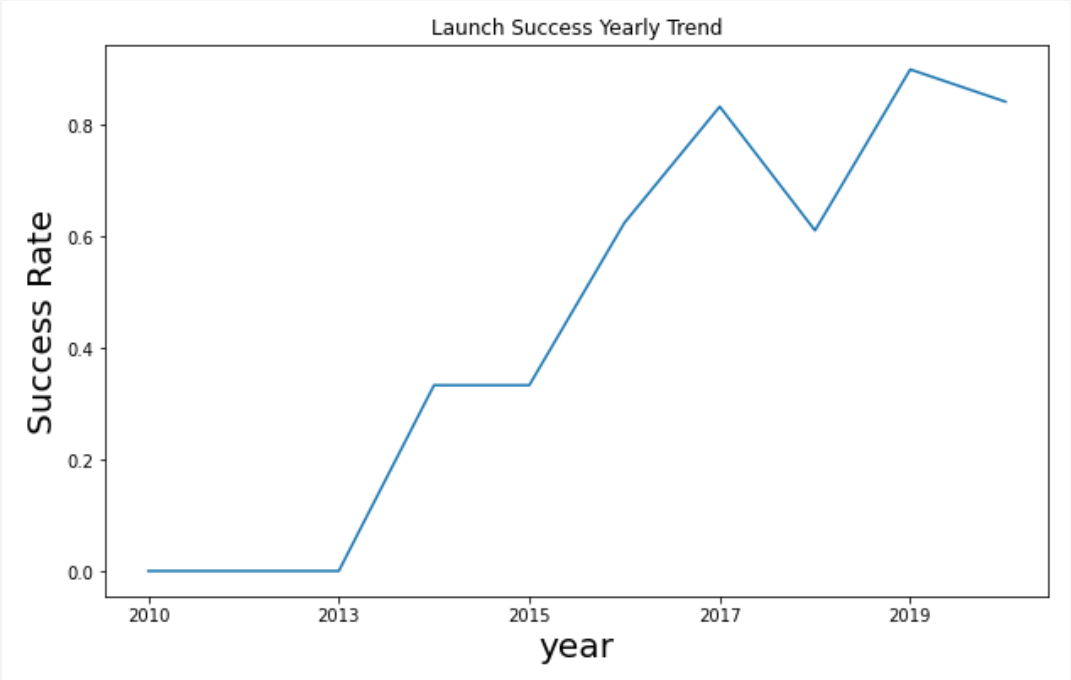
- It appears to be a relationship between the success reate of the LEO orbit and the number of flights
- Because of GTO success rate it appears that there is not a correlation with the number of flights.

Payload vs. Orbit Type



- With higher payload it seems that the orbits LEO, ISS and PO have a better success rate.
- It is evident that there is not a correlation in payload and succes rate for the GTO orbit.

Launch Success Yearly Trend



- It is very clear that the success rate kept increasing since 2013, until 2020.
- The success rate increased to 80% from 2013 to 2017, approximately.
- From 2017 to 2018 and 2019 to 2020 the success rate decreased.

All Launch Site Names

- Find the names of the unique launch sites
- Query:

```
%sql SELECT DISTINCT LAUNCH_SITE FROM SPACEX
```

- Explanation:

  The 'Select distinct' helps to identify the unique values from the launch_site column.

- Result:

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

Launch Site Names Begin with 'CCA'

- Find 5 records where launch sites begin with `CCA`
- Query:

```
%%sql
SELECT *
FROM SPACEX
WHERE LAUNCH_SITE LIKE 'CCA%'
LIMIT 5
```

- Explanation:

  In the where condition the like 'CCA%' helps to get those launch sites and limit allow us to get only 5 records.

- Result:

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	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	00:35:00	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

Total Payload Mass

- Calculate the total payload carried by boosters from NASA
- Query:

```
%%sql
SELECT SUM(PAYLOAD_MASS__KG_) AS TOTAL_PAYLOAD_MASS
FROM SPACEX
WHERE CUSTOMER = 'NASA (CRS)'
```

- Explanation:

  The SUM function helps to sum all the values in the payload_mass_kg_ column, and the where clause guarantees that the customer is NASA (CRS).

- Result:

total_payload_mass
45596

Average Payload Mass by F9 v1.1

- Calculate the average payload mass carried by booster version F9 v1.1
- Query:

```
%%sql
SELECT AVG(PAYLOAD_MASS__KG_) AS AVERAGE_PAYLOAD_MASS
FROM SPACEX
WHERE BOOSTER_VERSION = 'F9 v1.1'
```

- Explanation:

  The AVG function helps to get the average in the payload_mass_kg_ column, and the where clause guarantees that the booster version is F9 v1.1.

- Result:

average_payload_mass
2928

First Successful Ground Landing Date

- Find the dates of the first successful landing outcome on ground pad
- Query:

```
%%sql
SELECT MIN(DATE) AS DATE
FROM SPACEX
WHERE LANDING_OUTCOME = 'Success (ground pad)'
```

- Explanation:

  The MIN function helps to get the first date in date column, and the where clause guarantees that the landing outcome is Success (ground pad).

- Result:

DATE
2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

- List the names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000
- Query:

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

- Explanation:

  The where clause guarantees that the landing outcome is Success (drone ship), and that the payload is in the indicated range, selecting the data from the booster version column.

- Result:

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

Total Number of Successful and Failure Mission Outcomes

- Calculate the total number of successful and failure mission outcomes
- Query:

```
%%sql
SELECT COUNT(MISSION_OUTCOME) AS NUMBER_OF_MISSIONS
FROM SPACEX
WHERE MISSION_OUTCOME = 'Success' OR MISSION_OUTCOME LIKE 'Failure%'
```

- Explanation:

  The Select count function return the number of missions conditioned by the where clause, in this case it guarantees that the mission outcome is Successful and Failure, to sum up both there was necessary to use the OR and the LIKE 'Failure%', because there are two types or Success and Failure is (Failure (in flight)).

- Result:

number_of_missions
100

Boosters Carried Maximum Payload

- List the names of the booster which have carried the maximum payload mass
- Query:

```
%%sql
SELECT BOOSTER_VERSION
FROM SPACEX
WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEX)
```

- Explanation:

  Using a subquery we only select the booster version that fulfil the where condition for the payload, the subquery selects the max payload and then the where clause filters the data with that value.

- Result:

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

2015 Launch Records

- List the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015
- Query:

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

- Explanation:

  The query select the landing outcome, booster version and launch site, with the conditions of the where clause, in this case, the landing outcome is 'Failure (drone ship)' and the year 2015, that's the use of LIKE '2015%'.

- Result:

landing_outcome	booster_version	launch_site
Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

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

- 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

- Query:

```
%%sql
SELECT LANDING_OUTCOME, COUNT(*) AS COUNT
FROM SPACEX
WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY LANDING_OUTCOME
ORDER BY COUNT DESC
```

- Explanation:

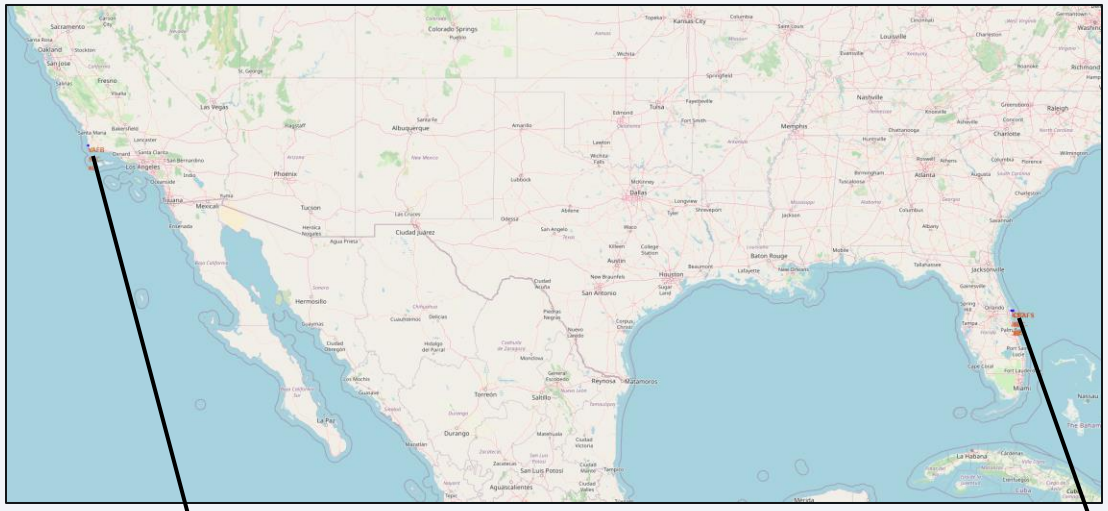
  The query select the landing outcome and use the count function to get the number for each landing outcome, using the where clause the date condition is implemented (with between both dates), then with the group by function the data is arranged in groups and then with the order by keyword the list is ordered or ranked in descending order with the count column.

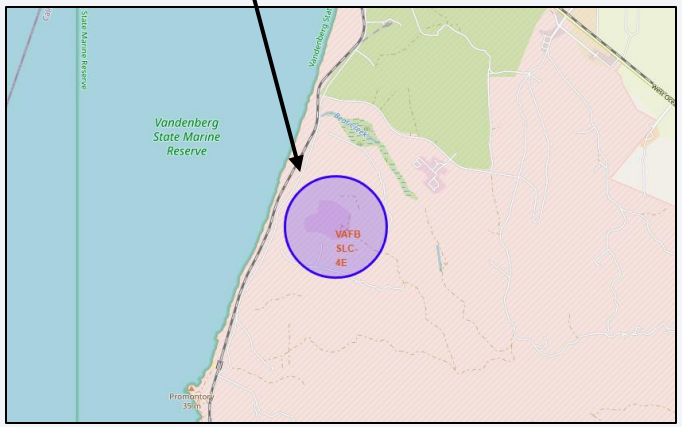
- Result:

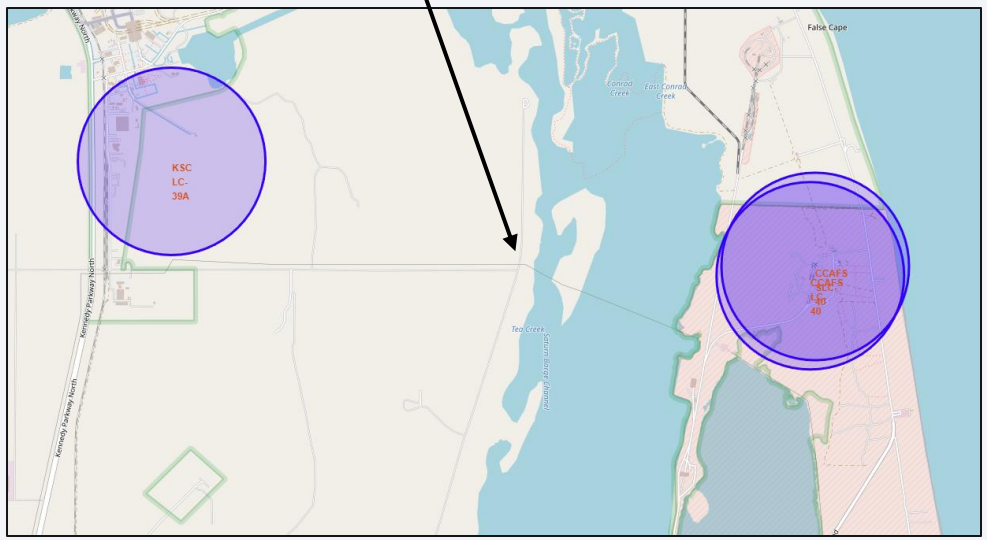
landing_outcome	COUNT
No attempt	10
Failure (drone ship)	5
Success (drone ship)	5
Controlled (ocean)	3
Success (ground pad)	3
Failure (parachute)	2
Uncontrolled (ocean)	2
Precluded (drone ship)	1


Section 3
Launch Sites
Proximities Analysis

SpaceX Falcon9 – Launch Sites Map

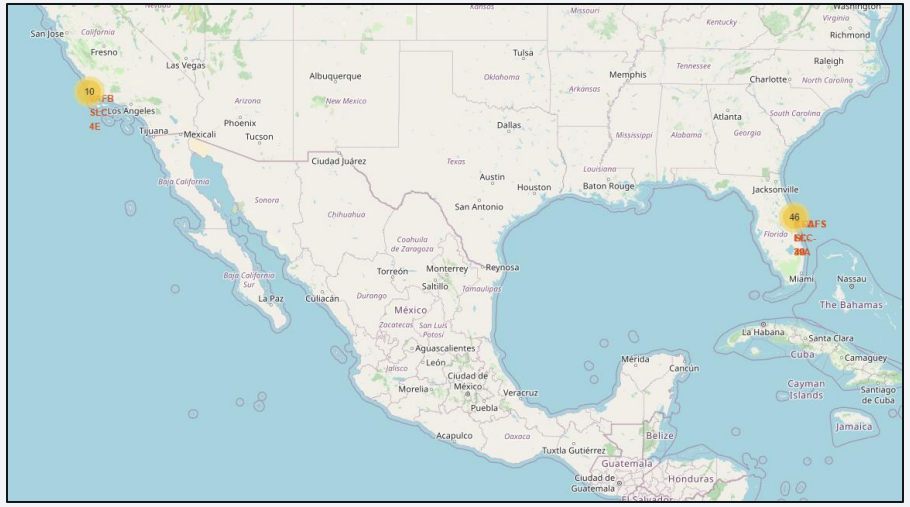




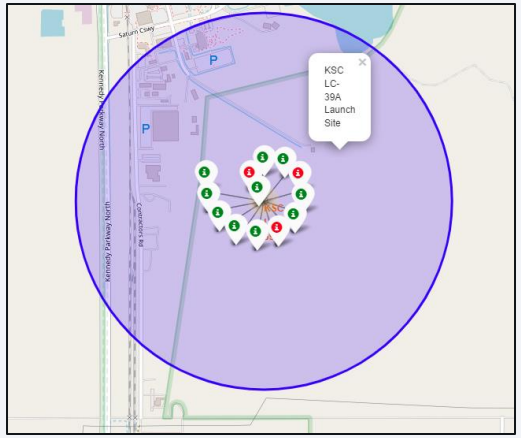


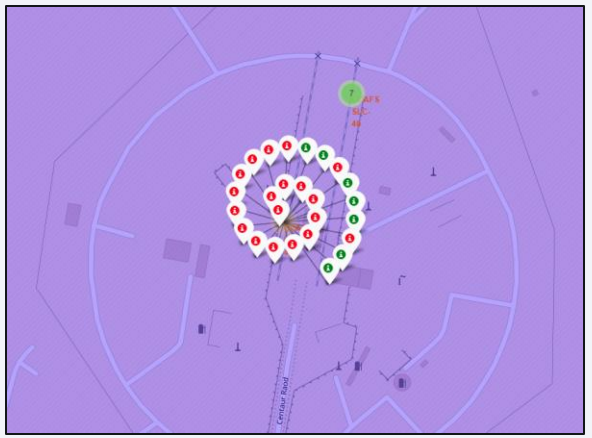
- In the top figure is the map with the Falcon9 launch sites located in the United States (California and Florida). Each launch site contains a circle, a label and a pop up with the location and the name of the launch site
- The other two figures are zooms of those launch sites to be able to display appropriately each launch site.

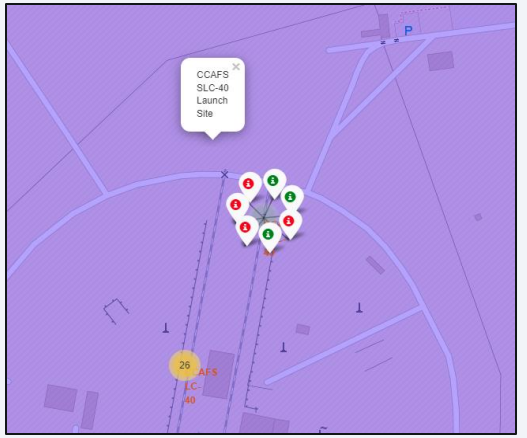
SpaceX Falcon9 – Successful/Failed Launches Map



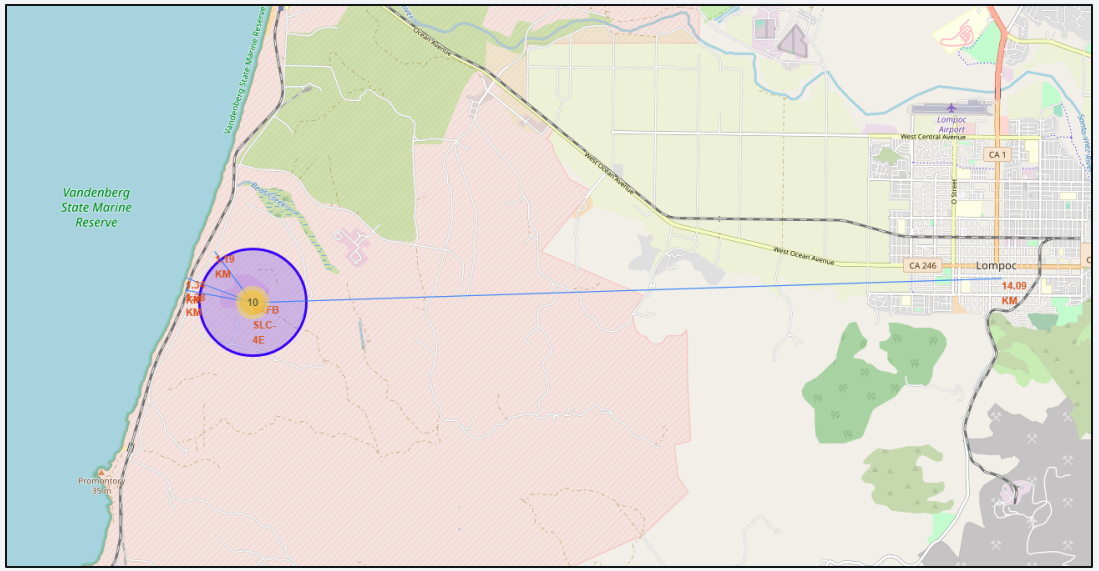
- In the top figure is the US map, the numbers on each side present the total number of successful and failed launches.
- The other four figures shows the marker clusters for each launch in each one of the launch sites.
- Looking the map and each figure, the KSC LC-39A launch site is the one with the highest success rate.

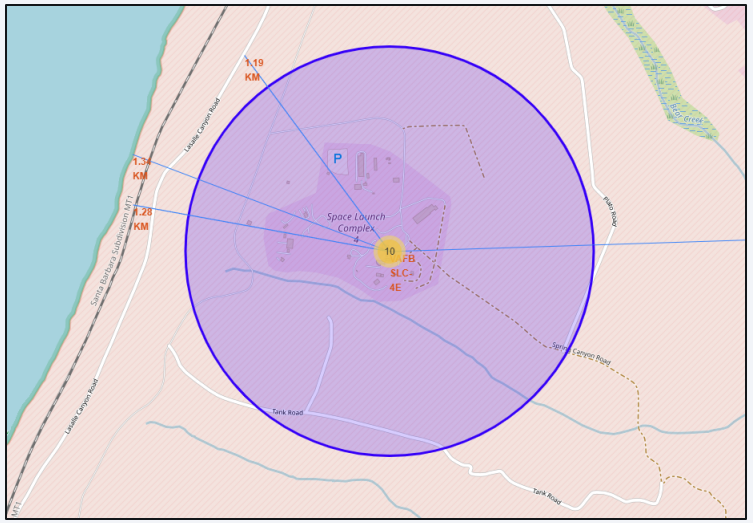




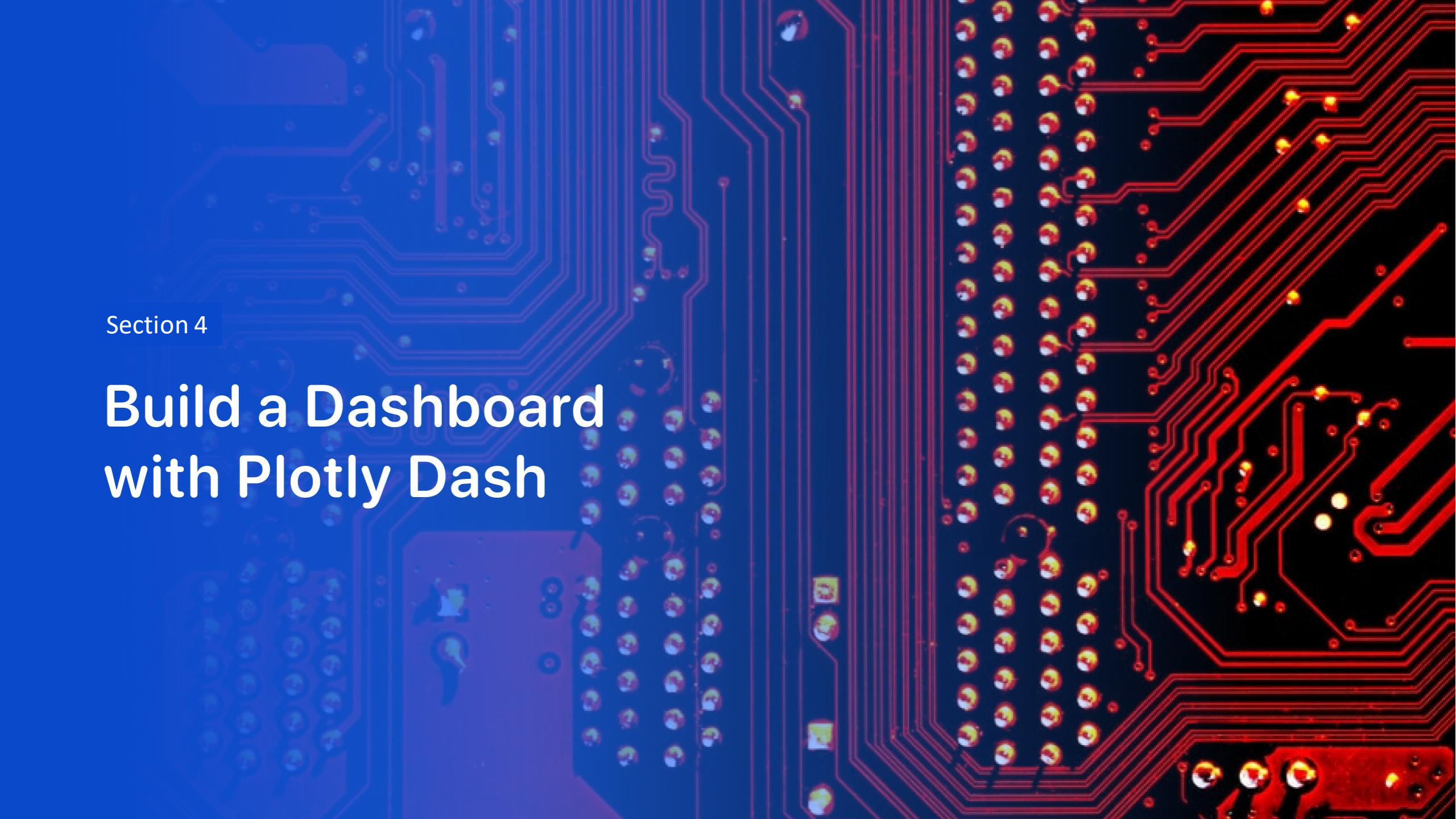


SpaceX Falcon9 – Launch Site to Proximity Distance Map

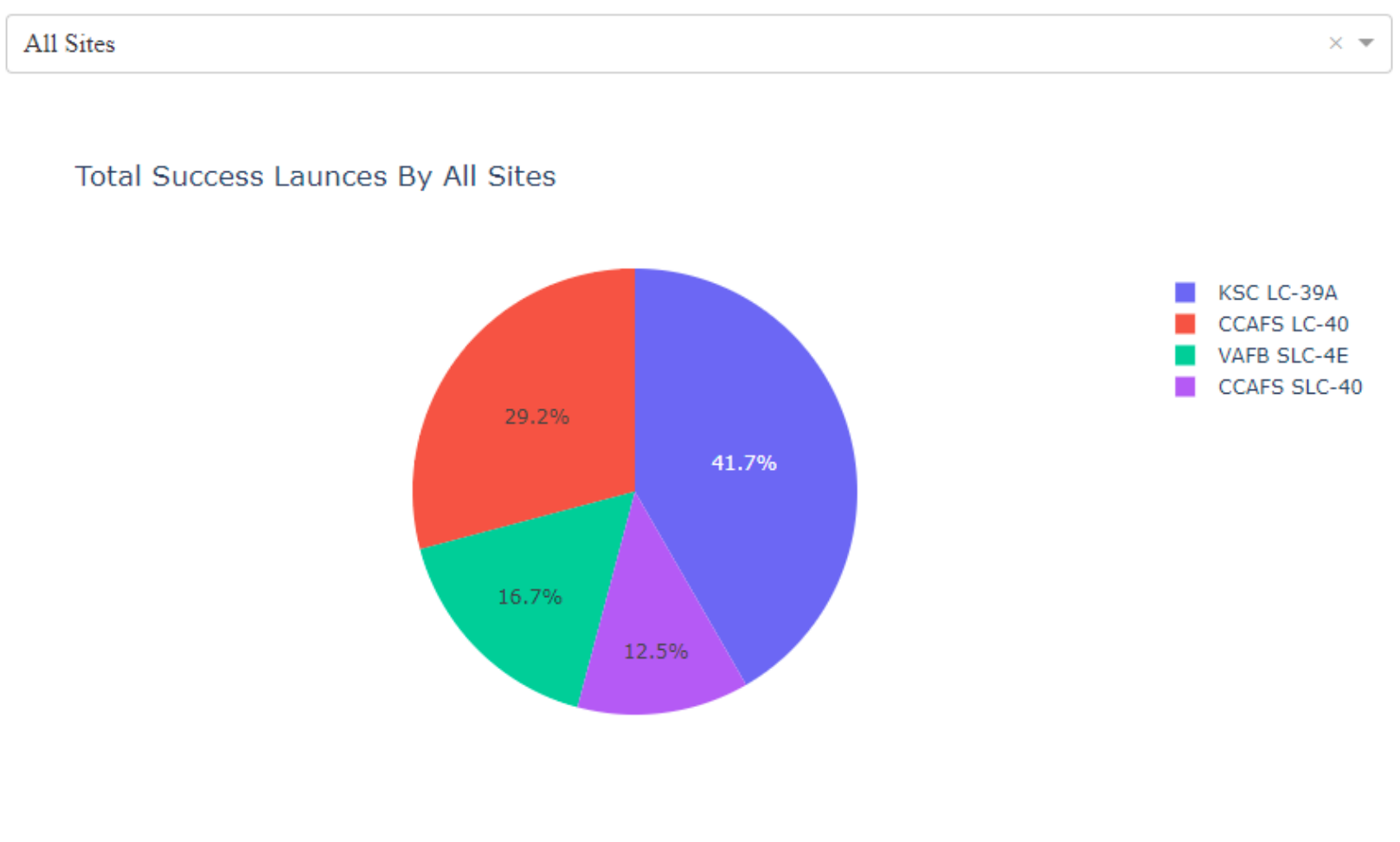




- Both figures displays the proximity sites marked on the map from the VAFB SLC-4E launch site. The city of Lompoc is relatively located further away in comparison to other locations, like the coastline, the highway, the railroad, etc. The marker shows that the city is at 14.09 km from the launch site.
- The second figure displays a zoom of the launch site to identify the polylines and the markers indicating the different distances from each location to the launch site.
- It is reasonable to think that the launch sites should be located away from cities, but relative close to the coastline, railroads and highways to provide easy access to resources.

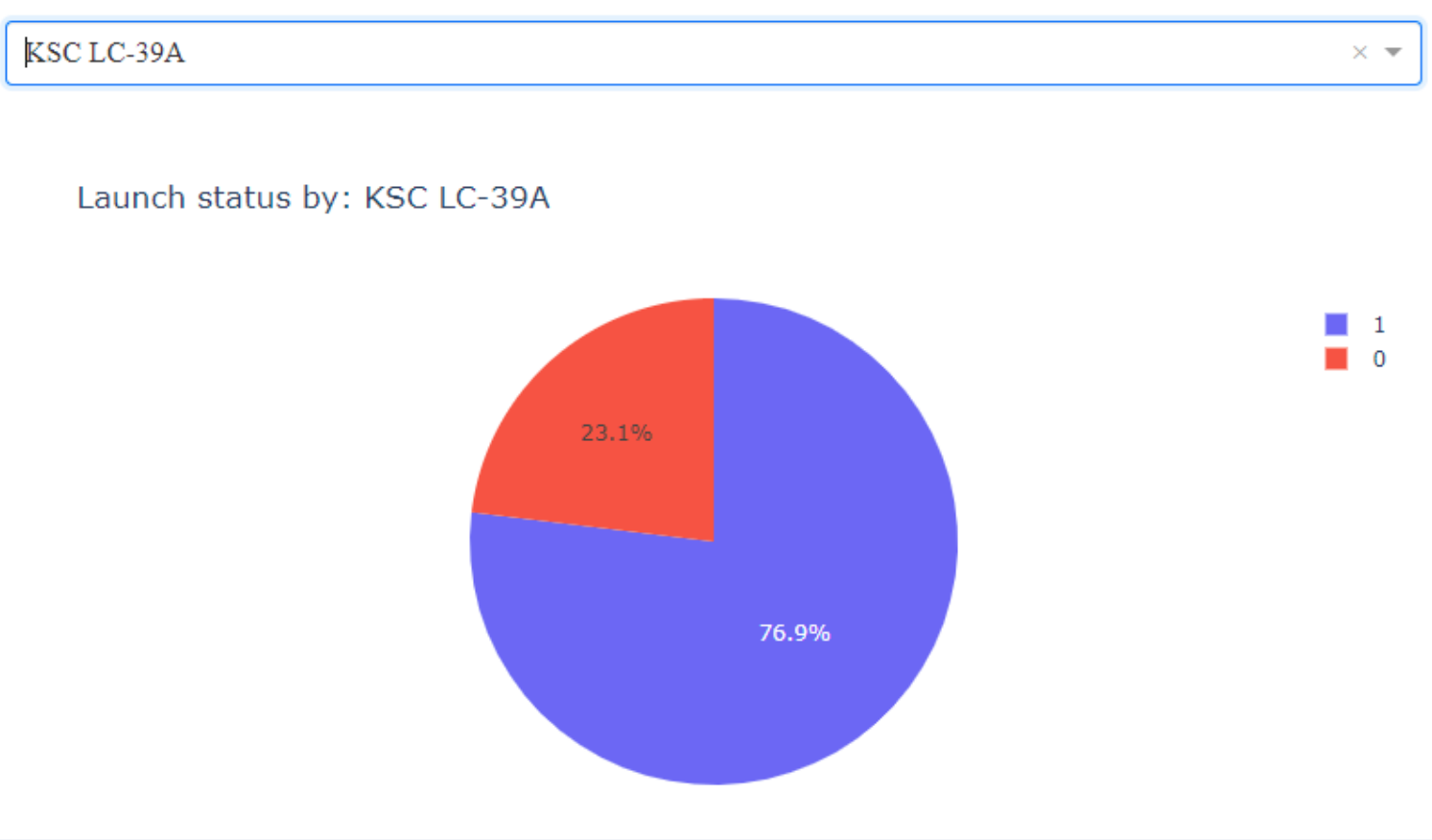

Section 4
Build a Dashboard with Plotly Dash

Launch Success Counts For All Sites



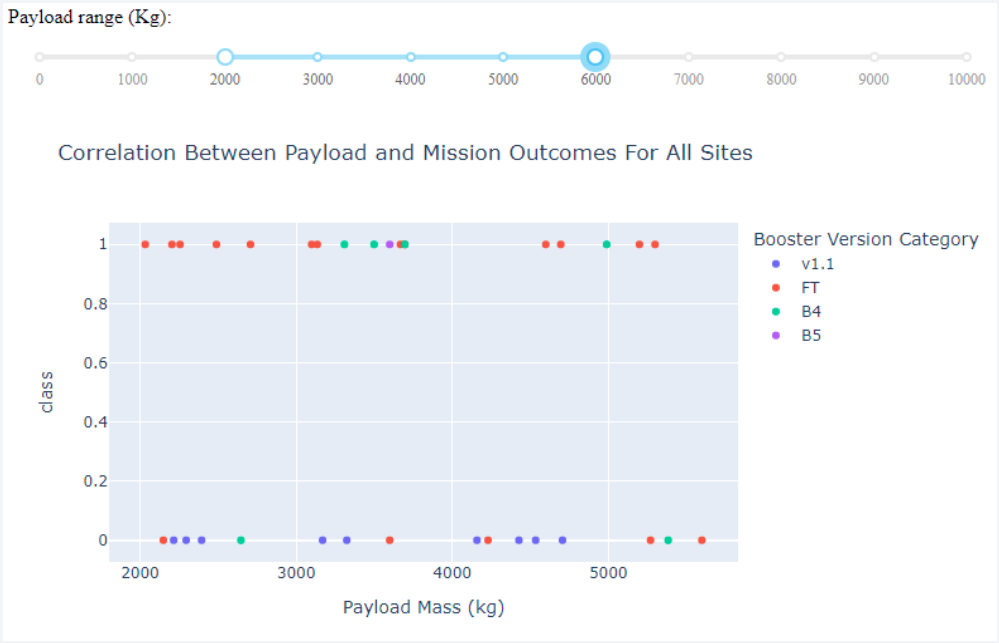
- The KSC LC-39A launch site has the highest launch success rate.
- The CCAFS SLC-4E launch site has the lowest launch success rate.

Launch Site with the Highest Launch Success Ratio



- The KSC LC-39A launch site has the highest launch success rate.
- The launch success rate is 76.9%.
- The launch failure rate is 23.1%.

Payload vs Launch Outcome Scatter Plots for All Sites



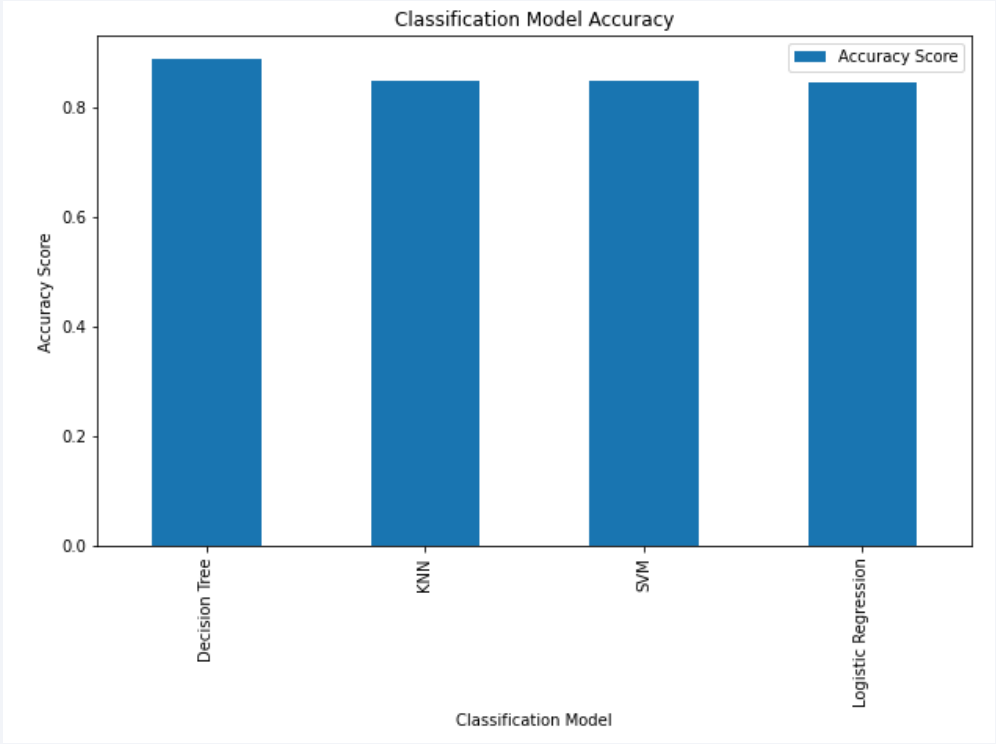
- The booster version with the most successful launches is FT.
- The payload range with a payload between 2000 and 6000 kg appears to have the most successful launches.
- The only booster version that has successful launches beyond the 6000 kg payload is B4.

Section 5

Predictive Analysis (Classification)

Classification Accuracy

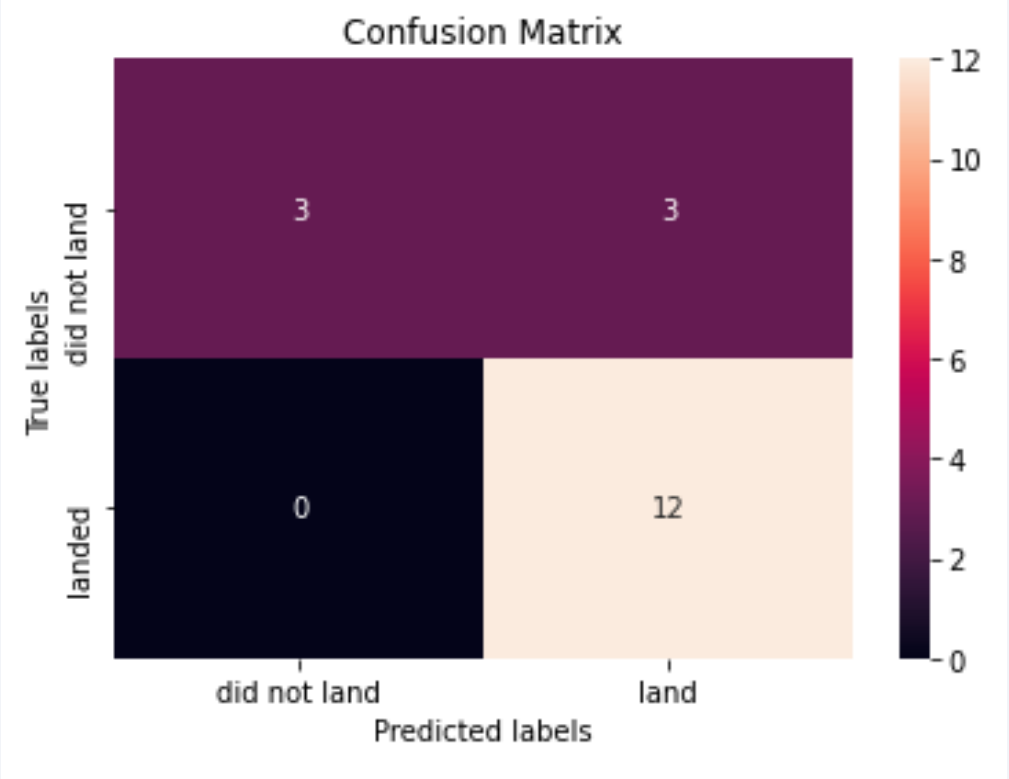
- According to both the bar chart and the Dataframe the model with the highest accuracy score is the Decision Tree algorithm.
- Both the Accuracy Score and the Test Data Accuracy Score are the highest for the Decision Tree, with 0.87 and 0.89 respectively.



	Algo Type	Accuracy Score	Test Data Accuracy Score
2	Decision Tree	0.876786	0.888889
3	KNN	0.848214	0.833333
1	SVM	0.848214	0.833333
0	Logistic Regression	0.846429	0.833333

Confusion Matrix

- The confusion matrix is the same for all the models.
- In the confusion matrix the classifier made 18 predictions.
- 12 True positives, scenarios that were predicted successful and landed successfully.
- 3 True Negatives in the top left, scenarios that were predicted failures and were failures.
- 3 False positives in the top right, scenarios that were predicted successful but did not land successfully.



Conclusions

- It appears that as the number of flights increases, the first stage is more likely to land successfully.
- Success rates appears to improve as the payload increases, but there is no clear relationship with both variables.
- The orbit types with the highest success rates are ES-LI, GEO, HEO, SSO, meanwhile GTO is the one with the lowest success rate.
- The success rate increased to 80% from 2013 to 2017, approximately.
- The KSC LC-39A launch site has the highest launch success rate, meanwhile the CCAFS SLC-4E launch site has the lowest launch success rate.
- Launch sites are located strategically away from the cities and closer to locations that could provide resources as coastlines, railroads and highways.
- The best performing Machine Learning Classification Model was the Decision Tree with and accuracy of 87.67%. More data may be needed to further tune and improve the models to get a better result.

Appendix

All the information related to the labs developed in the Jupyter notebooks is in the following link:

alejandrocarmu/IBM-Applied-Data-Science-Capstone-Final (github.com)

Thank you!