Winning Space Race with Data Science

Kristine Halago
December , 2021

Outline

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

Executive Summary

- 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
- Summary of all results
  - Exploratory data analysis results
  - Interactive analytics demo in screenshot
  - Predictive analysis results

Introduction

- Project background and context

  The commercial space age is here and SPACE Y wants to compete in the market by knowing how to SPACE X FALCON 9 enables to launch and REUSE THE FIRST STAGE of racket making it an inexpensive commercial firm for rocket launching.

- Problems you want to find answers

  - Determine if SpaceX will reuse the first stage.
  - The effect each relationship with certain rocket variables will impact in determining the success rate of a successful landing.
  - Investigate the each variables for each rocket that provide impact in determining the success rate of successful landing.
  - Find a pattern that can determine successful landing and failure landing, such as payload, orbit and customer.

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
- Perform data wrangling
  - Describe how data was processed
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - How to build, tune, evaluate classification models

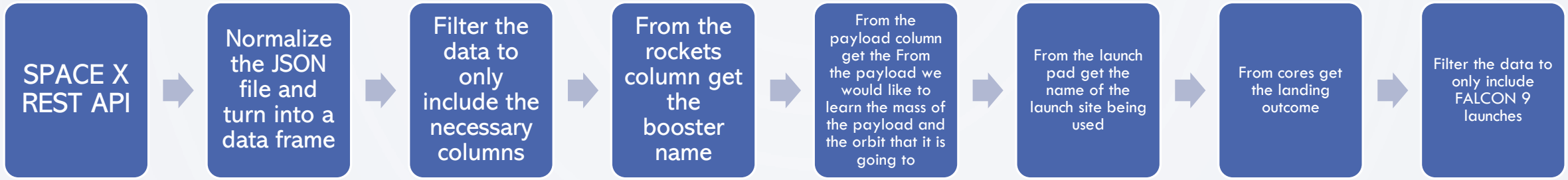
Data Collection Methodology

1. REQUEST API
   a. Using requests library to obtain the launch data
   b. This API will give us data about launches, including information about the rocket used,
   - payload delivered, launch specification, landing specifications, and landing outcome.

2. WEB SCRAPPING – WIKI PAGES
   a. Using beautiful soup package to web scape some html tables that contain valuable Falcon 9 launch records.

Data Collection Methodology : FLOWCHART

SpaceX API



SpaceX Web scrapping

Data Collection – SpaceX API

1. Request rocket launch data from Space X API

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

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

In [8]: response.status_code
Out[8]: 200
```

2. Normalize the response content and turn into a DataFrame

```
In [12]: # Use json_normalize meethod to convert the json result into a dataframe
         data = pd.json_normalize(response.json())
```

3. Filter the data to only include the necessary columns

```
In [14]: # 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']]
```

4. From the rockets column get the booster name

```
In [18]: # Call getBoosterVersion
         getBoosterVersion(data)
```

5. From the payload column get the mass of the payload and the orbit that it is going to

```
In [20]: # Call getLaunchSite
         getLaunchSite(data)
```

6. From the launch pad get the name of the launch site being used

```
In [21]: # Call getPayloadData
         getPayloadData(data)
```

7. From cores get the landing outcome

```
In [22]: # Call getCoreData
         getCoreData(data)
```

8. Filter the data to only include FALCON 9 launches

```
In [26]: # Hint data['BoosterVersion']!='Falcon 1'
         data_falcon9 = df[(df['BoosterVersion'] == 'Falcon 9')]
         data_falcon9
```

GitHub link:
https://github.com/TineAG/Data-Science/blob/main/Capstone/W1_jupyter-labs-spacex-data-collection-api.ipynb

Data Collection – Web Scrapping

1. Request rocket launch data from Space X API

```
In [19]: # use requests.get() method with the provided static_url
         response = requests.get(static_url)
         # assign the response to a object
         response.status_code
Out[19]: 200
```

```
In [24]: # Use BeautifulSoup() to create a BeautifulSoup object from a response text content
         data  = requests.get(static_url).text
         soup = BeautifulSoup(data,"html5lib")
```

2. Extract all column/Variables name from HTML table header

```
In [ ]: # Use the find_all function in the Bea
        # Assign the result to a list called `
        htmt_tables = soup.find_all('table')
```

3. Iterate through the table to extract column name

```
In [34]: 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
         temp = soup.find_all('th')
         for x in range(len(temp)):
             try:
              name = extract_column_from_header(temp[x])
              if (name is not None and len(name) > 0):
                 column_names.append(name)
             except:
              pass
```

4. Create DataFrame by parsing the launch HTML tables

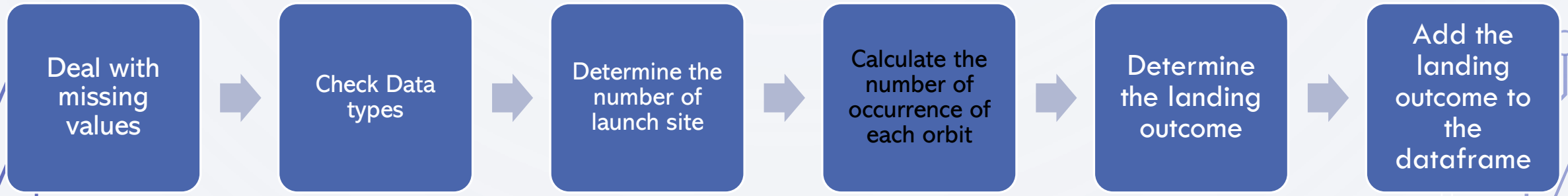
```
In [36]: launch_dict= dict.fromkeys(column_names)

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

         # Let's initial the launch_dict with each
         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']=[]
```

```
In [37]: extracted_row = 0
         #Extract each table
         for table_number,table in enumerate(soup.find_all('
            # get table row
             for rows in table.find_all("tr"):
                 #check to see if first table heading is as
                 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 l
                     #print(flight_number)
```

GitHub link:
https://github.com/TineAG/Data-Science/blob/main/Capstone/W2_jupyter-labs-webscraping%20(1).ipynb

Data Wrangling Methodology

In the data set, there are several different cases where the booster did not land successfully. Sometimes a landing was attempted but failed due to an accident; for example, 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 on a drone ship False ASDS means the mission outcome was unsuccessfully landed on a drone ship.



Data Wrangling Methodology

1. Check if there are missing value

```
In [3]: df.isnull().sum()/df.count()*100
```

2. Check columns data type

```
In [4]: df.dtypes
```

3. Determine the number of launch site

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

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

4. Calculate the number of occurrence of each orbit

```
In [7]: # landing_outcomes = values on Outcome column
        #landing_outcomes = df.groupby('Outcome')['Orbit'
        landing_outcomes = df['Outcome'].value_counts()
        landing_outcomes
```

5. Create a landing outcome label

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

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

        for i,outcome in enumerate(landing_outcomes.keys()):
            if outcome in bad_outcomes:
                landing_class.append(0)
            else:
                landing_class.append(1)


        df_landing_class = pd.DataFrame(landing_class)
```

6. Add the landing outcome to the dataframe

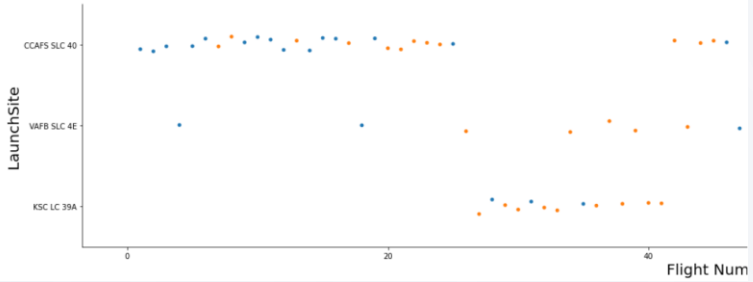
```
In [11]: df['Class']=df_landing_class
         df[['Class']].head(8)
```

GitHub link:
https://github.com/TineAG/Data-Science/blob/main/Capstone/W1_labs-jupyter-spacex-Data%20wrangling.ipynb

Exploratory Data Analysis using Visual

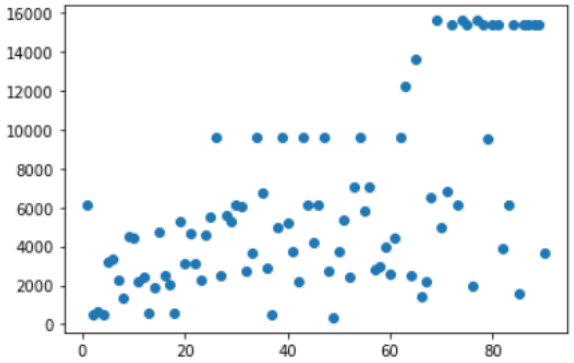
1. Visualize the relation between Flight Number and Launch Site

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



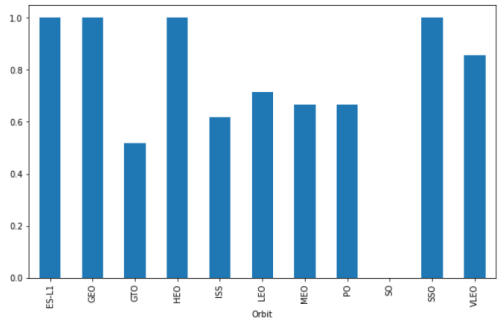
2. Visualize the relationship between Payload and Launch Site

```
In [20]: # Plot a scatter point chart with x axis to be Pay Load M
fig = plt.Figure(figsize = (10,20))
plt.scatter(x = df["FlightNumber"], y=df["PayloadMass"])
plt.show()
```



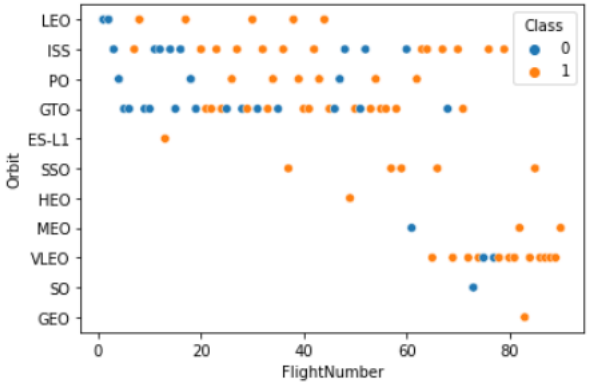
3. Visualize the relationship between success rate of each orbit type

```
In [28]: # HINT use groupby method on Orbit column and get the mean of Class column
bar_df = df.groupby('Orbit')['Class'].mean()
bar_df.plot(kind='bar', figsize=(10, 6))
plt.show()
```



4. Visualize the relationship between Flight Number & Orbit Type

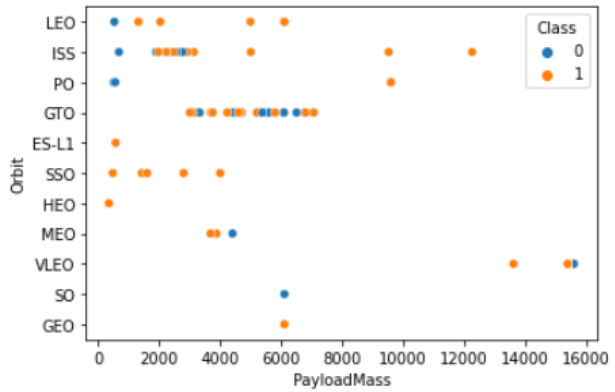
```
In [31]: # Plot a scatter point chart with x axis to be FlightNumber and y axis to
sns.scatterplot(x = "FlightNumber", y="Orbit", hue = 'Class', data = df)
plt.show()
```



Exploratory Data Analysis using Visual

5. Visualize the relation between Payload & Orbit Type

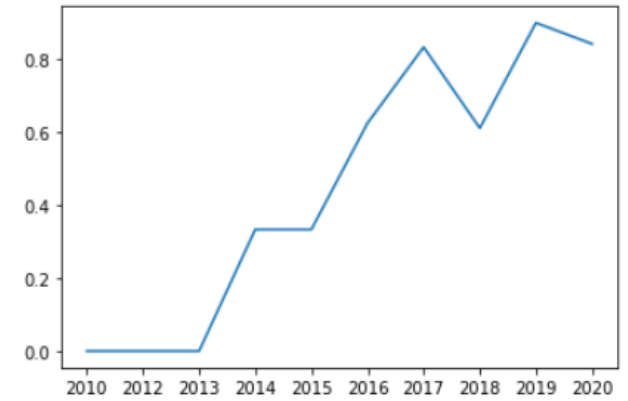
```
In [33]: # Plot a scatter point chart with x axis to be Payload and y axis to be
sns.scatterplot(x = "PayloadMass", y="Orbit", hue = 'Class', data = df)
plt.show()
```



6. Visualize the launch success yearly trend

```
In [45]: # A function to Extract years from the date
year=[]
def Extract_year(date):
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year

#year =pd.unique(Extract_year(df))
df['years'] = Extract_year(df)
```

```
In [50]: new_df = df.groupby('years')['Class'].mean().to_frame(
new_df.head()
plt.plot(new_df['years'],new_df['mean'])  # Plot the
plt.show()
```



GitHub link:
https://github.com/TineAG/Data-Science/blob/main/Capstone/W2_jupyter-labs-eda-dataviz.ipynb

Exploratory Data Analysis using SQL

1. Name of the unique launch site in the space mission

```
In [3]: %sql select distinct(LAUNCH_SITE) from SPACEXTBL
 * ibm_db_sa://mdb13977:***@2f3279a5-73d1-4859-88
Done.
```

Out[3]:

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

2. Records with launch site begin with string CCA

```
In [4]: %sql select * from SPACEXTBL where LAUNCH_SITE like 'CCA%' limit 5
```

Out[4]:

DATE	time__utc_	booster_version	launch_site	payload	payload_mass__kg_	orbit	customer	mission_outcome	landing__outcome
2010-06-04	None	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	None	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	None	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	None	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	None	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

3. Total payload mass carried by booster version F9 v1.1

```
In [5]: %sql select sum(PAYLOAD_MASS__KG_) from SPACEXTBL where CUSTOMER = 'NASA (CRS)'
```

Out[5]:

1
45596

4. Average payload mass carried by booster version F9 v1.1

```
In [6]: %sql select avg(PAYLOAD_MASS__KG_) from SPACEXTBL where BOOSTER_VERSION = 'F9 v1.1'
```

Out[6]:

1
2928

5. List of dates the first successful landing outcome in ground

```
In [7]: %sql select min(DATE) from SPACEXTBL where Landing__Outcome = 'Success (ground pad)'
```

Out[7]:

1
2015-12-22

Exploratory Data Analysis using SQL

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

```
In [8]: %sql select BOOSTER_VERSION from SPACEXTBL where Landing__Outcome = 'Success (drone ship)' and PAYLOAD_MASS__KG_ > 4000 and PAYLO
```

Out[8]:

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

7. List of total number of successful and failure mission outcomes

```
In [9]: %sql select count(MISSION_OUTCOME) from SPACEXTBL where MISSION_OUTCOME = 'Success' or MISSION_OUTCOME = 'Failure (in flight)'
```

8. List of the booster version which have carried the maximum payload mass

```
In [10]: %sql select BOOSTER_VERSION from SPACEXTBL where PAYLOAD_MASS__KG_ = (select max(PAYLOAD_MASS__KG_) from SPACEXTBL)
```

Out[10]:

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

9. List of failed landing outcomes in drop ship

```
In [11]: %sql SELECT EXTRACT(MONTH, select min(DATE) from SPACEXTBL where Landing__Outcome = 'Success (ground pad)')
```

10. Rank the count of landing outcome between 2010-06-04 and 2017-03-20

```
In [12]: %sql select * from SPACEXTBL where Landing__Outcome like 'Success%' and (DATE between '2010-06-04' and '2017-03-20') order by dat
```

Out[12]:

DATE	time__utc_	booster_version	launch_site	payload	payload_mass__kg_	orbit	customer	mission_outcome	landing__outcome
2017-02-19	None	F9 FT B1031.1	KSC LC-39A	SpaceX CRS-10	2490	LEO (ISS)	NASA (CRS)	Success	Success (ground pad)
2017-01-14	None	F9 FT B1029.1	VAFB SLC-4E	Iridium NEXT 1	9600	Polar LEO	Iridium Communications	Success	Success (drone ship)
2016-08-14	None	F9 FT B1026	CCAFS LC-40	JCSAT-16	4600	GTO	SKY Perfect JSAT Group	Success	Success (drone ship)
2016-07-18	None	F9 FT B1025.1	CCAFS LC-40	SpaceX CRS-9	2257	LEO (ISS)	NASA (CRS)	Success	Success (ground pad)
2016-05-27	None	F9 FT B1023.1	CCAFS LC-40	Thaicom 8	3100	GTO	Thaicom	Success	Success (drone ship)
2016-05-06	None	F9 FT B1022	CCAFS LC-40	JCSAT-14	4696	GTO	SKY Perfect JSAT Group	Success	Success (drone ship)
2016-04-08	None	F9 FT B1021.1	CCAFS LC-40	SpaceX CRS-8	3136	LEO (ISS)	NASA (CRS)	Success	Success (drone ship)
2015-12-22	None	F9 FT B1019	CCAFS LC-40	OG2 Mission 2 11 Orbcomm-OG2 satellites	2034	LEO	Orbcomm	Success	Success (ground pad)

GitHub link:
https://github.com/TineAG/Data-Science/blob/main/Capstone/W2_jupyter-labs-eda-sql-coursera.ipynb

Build an Interactive Map using Folium

1. Mark all launch site in Map

```
In [8]: # Initial the map
site_map = folium.Map(location=nasa_coordinate, zoom_start=5)
# For each launch site, add a Circle object based on its coordin
for row in zip(launch_sites_df['Launch Site'],launch_sites_df['L
    launch_site = row[0]
    lat = row[1]
    long = row[2]
    circle = folium.Circle([lat, long], radius=1000, color='#d35
    marker = folium.map.Marker(
        [lat, long],
        # Create an icon as a text label
        icon=DivIcon(
            icon_size=(20,20),
            icon_anchor=(0,0),
            html='<div style="font-size: 12; color:#d35400;"><b>
            )
        )
    site_map.add_child(circle)
    site_map.add_child(marker)

site_map
```

2. Mark success/failed launches for each site on the map

```
launch_sites=[]
for i in enumerate(spacex_df['class']):
    if i==1:
        launch_sites.append('Green')
    else:
        launch_sites.append('Red')
launch_sites
```

```
In [13]: # Function to assign color to launch outcome
def assign_marker_color(launch_outcome):
    if launch_outcome == 1:
        return 'green'
    else:
        return 'red'

spacex_df['marker_color'] = spacex_df['class'].apply(assign_marker_color)
spacex_df.tail(10)
```


GitHub link:
https://github.com/TineAG/Data-Science/blob/main/Capstone/W3_jupyter_lab_folium_launch_site_location.ipynb

Build an Interactive Map using Folium

3. Calculate the distance between a launch site and to its proximity

```
coordinates = [
    [28.56342, -80.57674],
    [28.56342, -80.56756]]

lines=folium.PolyLine(locations=coordinates, weight=1)
site_map.add_child(lines)
distance = calculate_distance(coordinates[0][0], coordinates[0][1
distance_circle = folium.Marker(
    [28.56342, -80.56794],
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b
        )
    )
site_map.add_child(distance_circle)
site_map
```

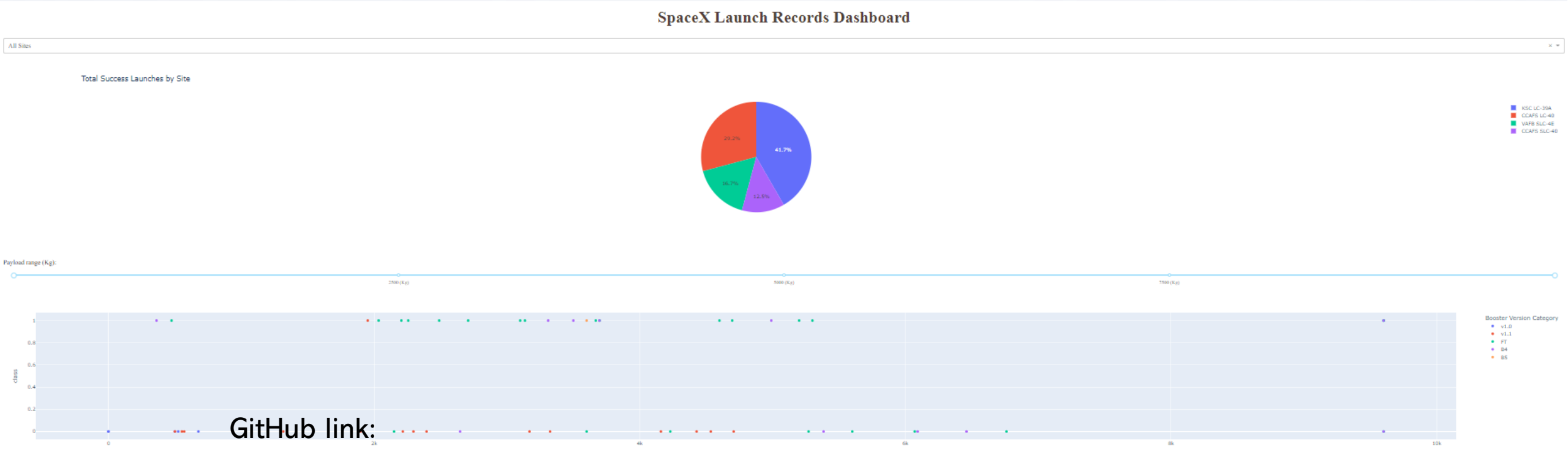
```
#Distance to Highway
coordinates = [
    [28.56342, -80.57674],
    [28.411780, -80.820630]]

lines=folium.PolyLine(locations=coordinates, weight=1)
site_map.add_child(lines)
distance = calculate_distance(coordinates[0][0], coordinates[0][1
distance_circle = folium.Marker(
    [28.411780, -80.820630],
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#252526;"><b>%s</b
        )
    )
site_map.add_child(distance_circle)
site_map
```

GitHub link:
https://github.com/TineAG/Data-Science/blob/main/Capstone/W3_jupyter_lab_folium_launch_site_location.ipynb

Build Dashboard with Plotly Dash

Dashboard includes a pie chart and a scatter plot.

- Pie chart can be selected to show distribution of successful landings across all launch sites and can be selected to show individual launch site success rates.
- The scatter plot can help us see how success varies across launch sites, payload mass, and booster version category.



GitHub link:
https://github.com/TineAG/Data-Science/blob/main/Capstone/W3_jupyter-lab-dash-app.ipynb

Predictive Analysis :

1. Create a NumPy array from the column Class

```
In [5]: Y = data['Class'].to_numpy()
        Y[0:10], type(Y)
Out[5]: (array([0, 0, 0, 0, 0, 0, 1, 1, 0, 0], dtype=int64), numpy.ndarray)
```

2. Standardize the data

```
In [8]: # students get this
        transform = preprocessing.StandardScaler()
In [15]: x = transform.fit_transform(X)
```

3. Split the data

```
In [16]: 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.

```
In [17]: Y_test.shape, X_test.shape
Out[17]: ((18,), (18, 83))
```

4. Modelling : LOGISTIC REGRESSION

```
In [18]: parameters ={'C':[0.01,0.1,1],
                      'penalty':['l2'],
                      'solver':['lbfgs']}
```

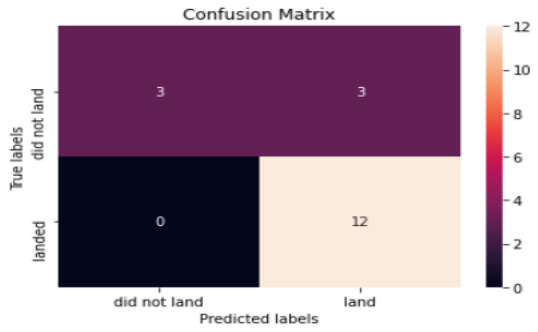
```
In [24]: parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['l
         lr=LogisticRegression(random_state=1)
         logreg_cv = GridSearchCV(lr, parameters, cv=10, refit=True)
         logreg_cv.fit(X_train, Y_train)
```

5. Calculate the accuracy of Logistic Regression Model

```
In [13]: log_acc=logreg_cv.score(X_test, Y_test)
         print("Test set accuracy: {:.1%}".format(log_acc))
Test set accuracy: 83.3%
```

Lets look at the confusion matrix:

```
In [32]: log_yhat=logreg_cv.predict(X_test)
         plot_confusion_matrix(Y_test,log_yhat)
```



GitHub link:
https://github.com/TineAG/Data-Science/blob/main/Capstone/W3_jupyter-lab-dash-app.ipynb

Predictive Analysis :

6. Modelling : SUPPORT VECTOR MACHINE

```
In [44]: parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
                       'C': np.logspace(-3, 3, 5),
                       'gamma':np.logspace(-3, 3, 5)}
         svm = SVC(probability=True, random_state=1)

In [45]: svm_cv = GridSearchCV(svm, parameters, cv=10)
         svm_cv.fit(X_train, Y_train)
```

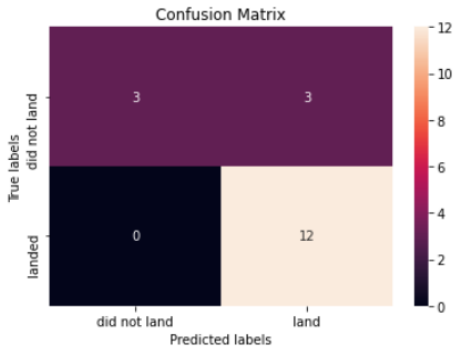
7. Calculate the accuracy of Support Vector Machine Model

```
In [18]: svm_acc=svm_cv.score(X_test, Y_test)
         print("Test set accuracy: {:.1%}".format(svm_acc))
```

Test set accuracy: 83.3%

We can plot the confusion matrix

```
In [19]: svm_yhat=svm_cv.predict(X_test)
         plot_confusion_matrix(Y_test,svm_yhat)
```



8. Modelling : DECISION TREE

```
In [20]: 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]}

         tree = DecisionTreeClassifier()

In [21]: tree_cv = GridSearchCV(tree, parameters, cv=10)
         tree_cv.fit(X_train, Y_train)

Out[21]: GridSearchCV(cv=10, estimator=DecisionTreeClassifier(),
                      param_grid={'criterion': ['gini', 'entropy'],
                                  'max_depth': [2, 4, 6, 8, 10, 12, 14
                                  'max_features': ['auto', 'sqrt'],
                                  'min_samples_leaf': [1, 2, 4],
                                  'min_samples_split': [2, 5, 10],
                                  'splitter': ['best', 'random']})
```

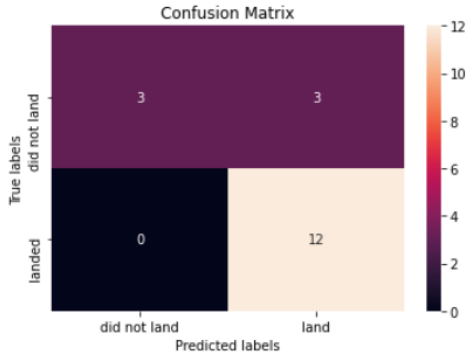
9. Calculate the accuracy of Decision Tree Model

```
In [23]: tree_acc=tree_cv.score(X_test, Y_test)
         print("Test set accuracy: {:.1%}".format(tree_acc))
```

Test set accuracy: 72.2%

We can plot the confusion matrix

```
In [25]: yhat = svm_cv.predict(X_test)
         plot_confusion_matrix(Y_test,yhat)
```



Predictive Analysis :

10. Modelling : K NEAREST NEIGHBOR

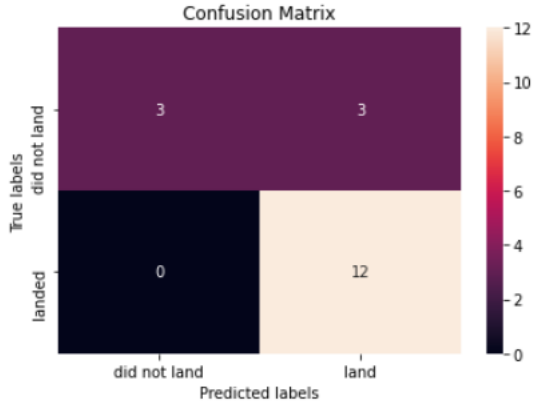
```
In [26]: parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                       'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                       'p': [1,2]}

         KNN = KNeighborsClassifier()

In [27]: knn_cv = GridSearchCV(KNN, parameters, cv=10)
         knn_cv.fit(X_train, Y_train)

Out[27]: GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
                      param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                                  'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                                  'p': [1, 2]})

In [28]: print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
         print("accuracy :{:.1%}".format(knn_cv.best_score_))

         tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}
         accuracy :84.8%
```

11. Calculate the accuracy KNN Model

```
In [29]: knn_acc=knn_cv.score(X_test, Y_test)
         print("Test set accuracy: {:.1%}".format(knn_acc))

         Test set accuracy: 83.3%
```

We can plot the confusion matrix

```
In [33]: knn_yhat = knn_cv.predict(X_test)
         plot_confusion_matrix(Y_test,knn_yhat)
```



GitHub link:
https://github.com/TineAG/Data-Science/blob/main/Capstone/W3_jupyter-lab-dash-app.ipynb

Predictive Analysis (Classification)

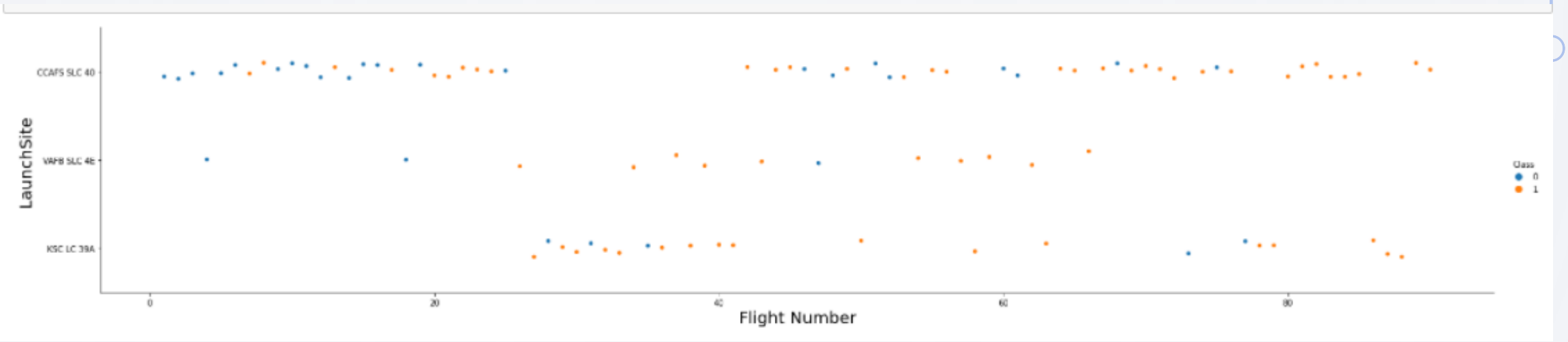
- Summarize how you built, evaluated, improved, and found the best performing classification model
- You need present your model development process using key phrases and flowchart
- Add the GitHub URL of your completed predictive analysis lab, as an external reference and peer-review purpose

GitHub link:
https://github.com/TineAG/Data-Science/blob/main/Capstone/W3_jupyter-lab-dash-app.ipynb

Section 2

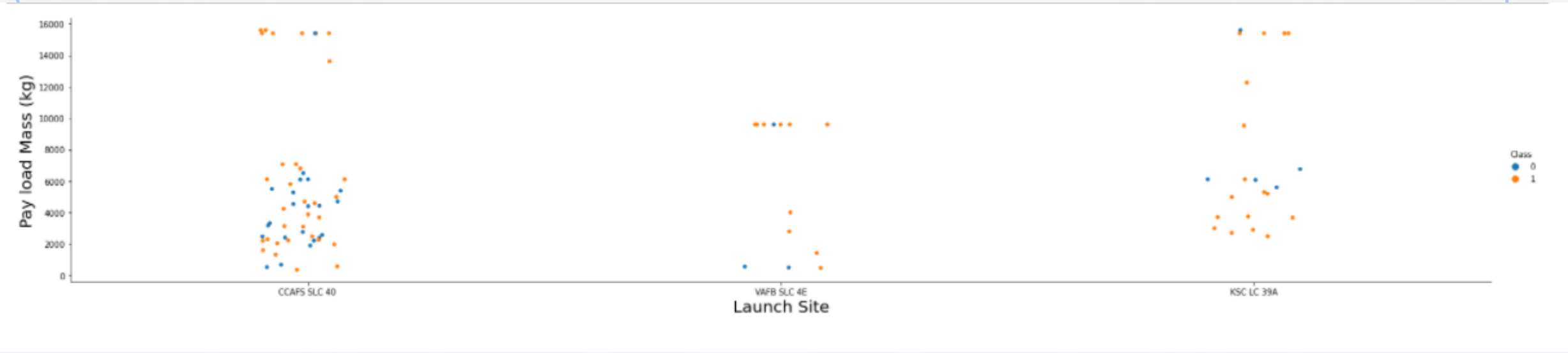
Insights drawn from EDA

Flight Number vs. Launch Site



As shown in the graph, launch success occurs more in launch site CCAFS SLC 40

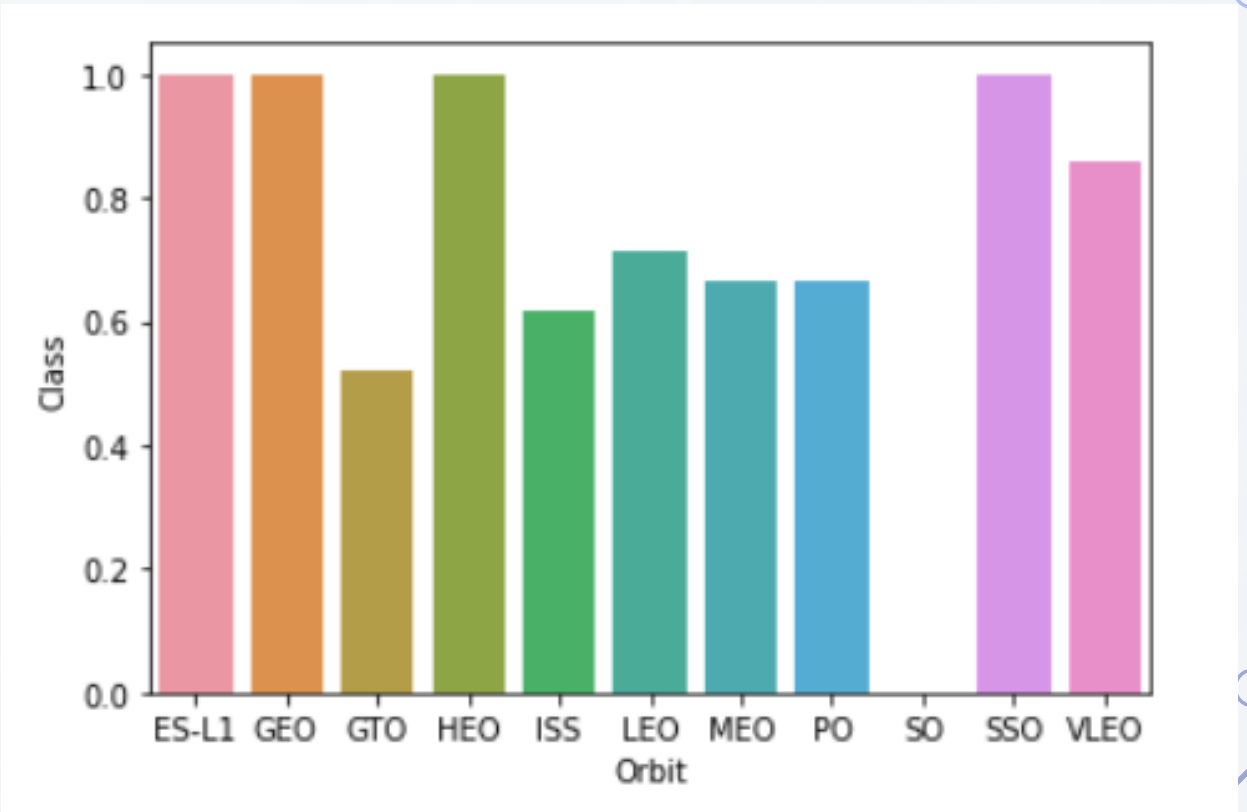
Payload vs. Launch Site



Successful launch is more evident with payload mass of below 6000

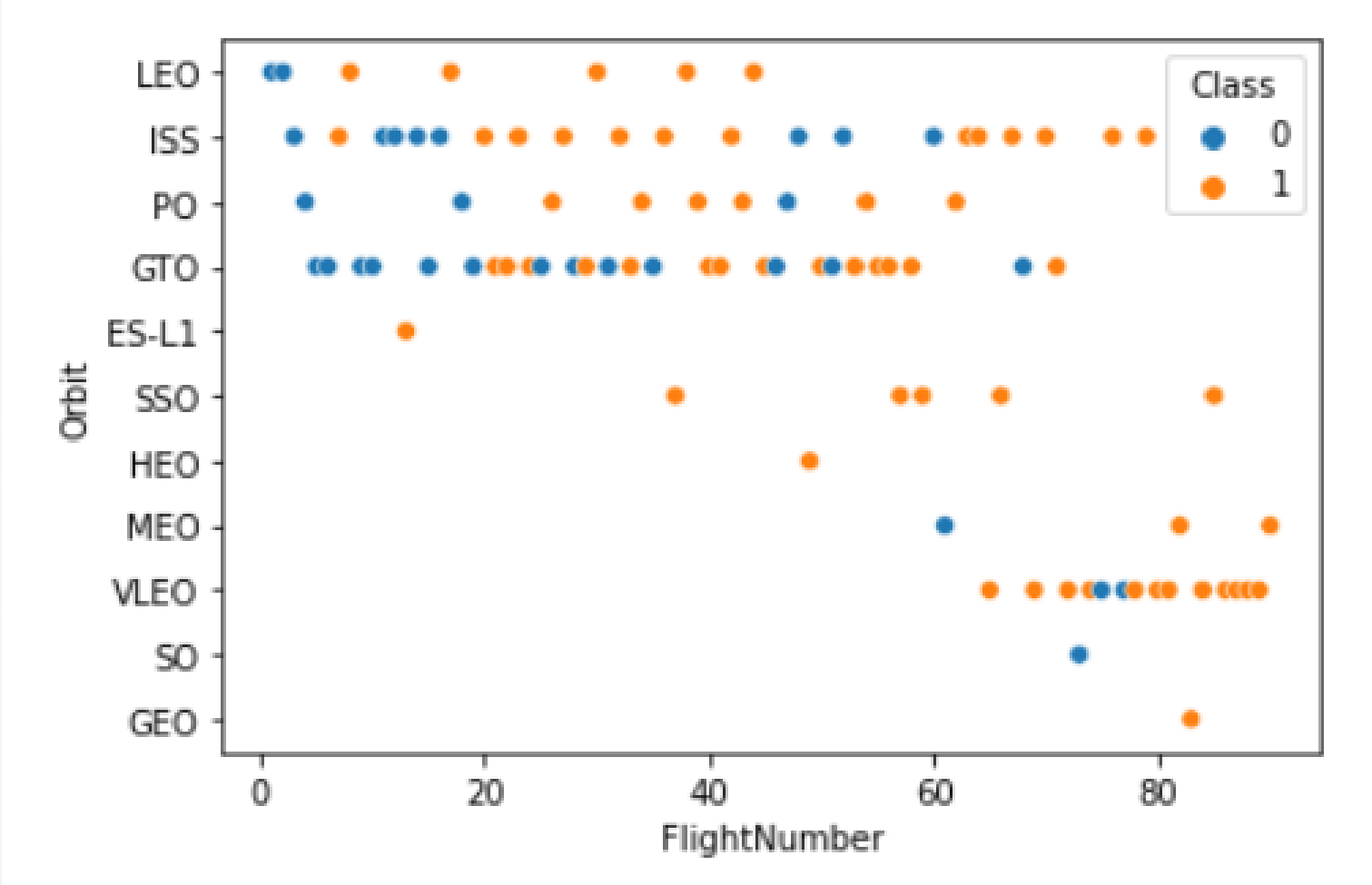
Success Rate vs. Orbit Type

Orbit with best success rate are in ES-L1, GEO, HEO, and SSO



Flight Number vs. Orbit Type

Most of the fights where in GTO orbit and success rate and failure are evenly occur



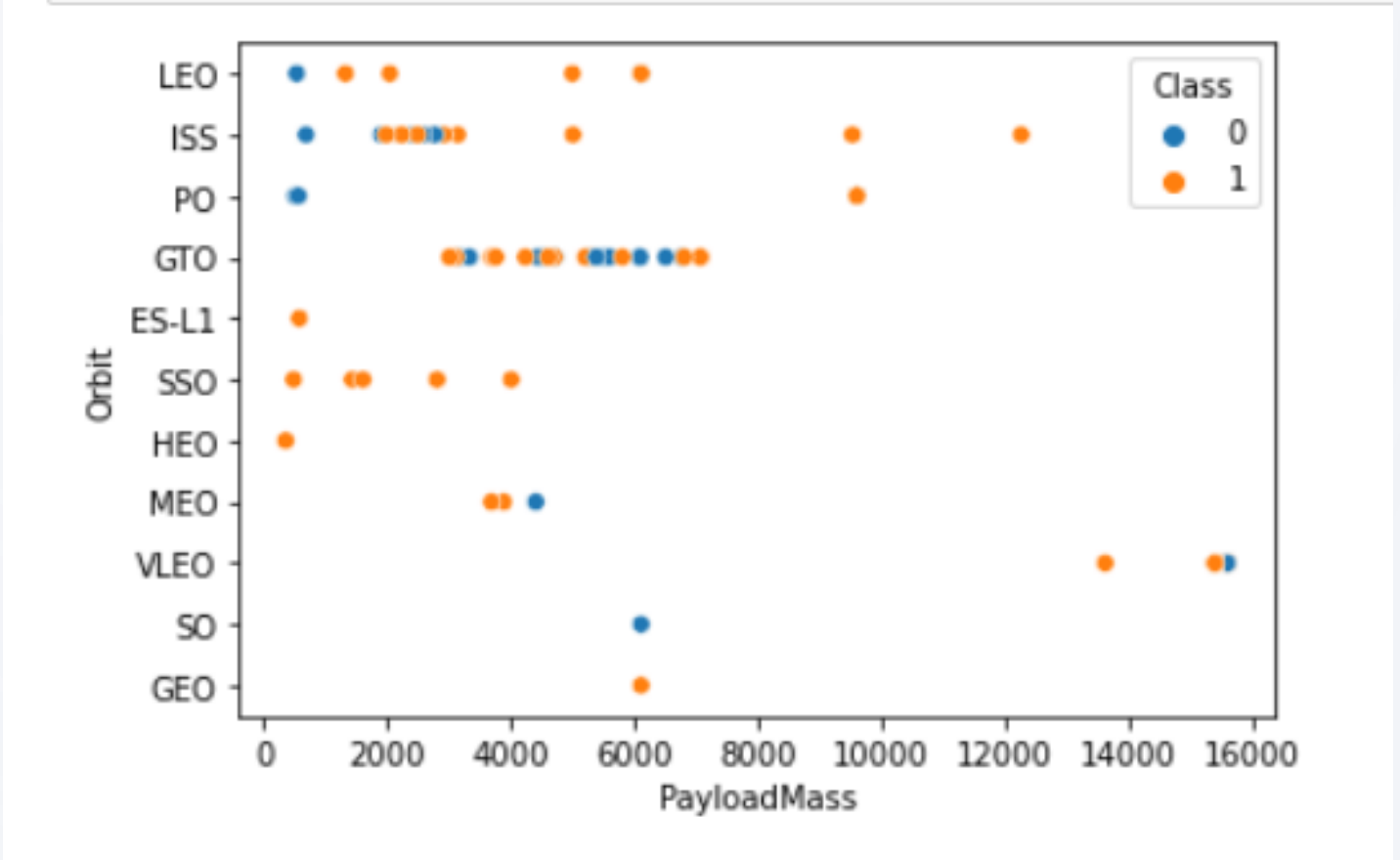
Payload vs. Orbit Type

GTO – payload mass are concentrated between Payload Mass of 2500 & 8000

VLEO – orbit have the highest payload mass among all the orbits

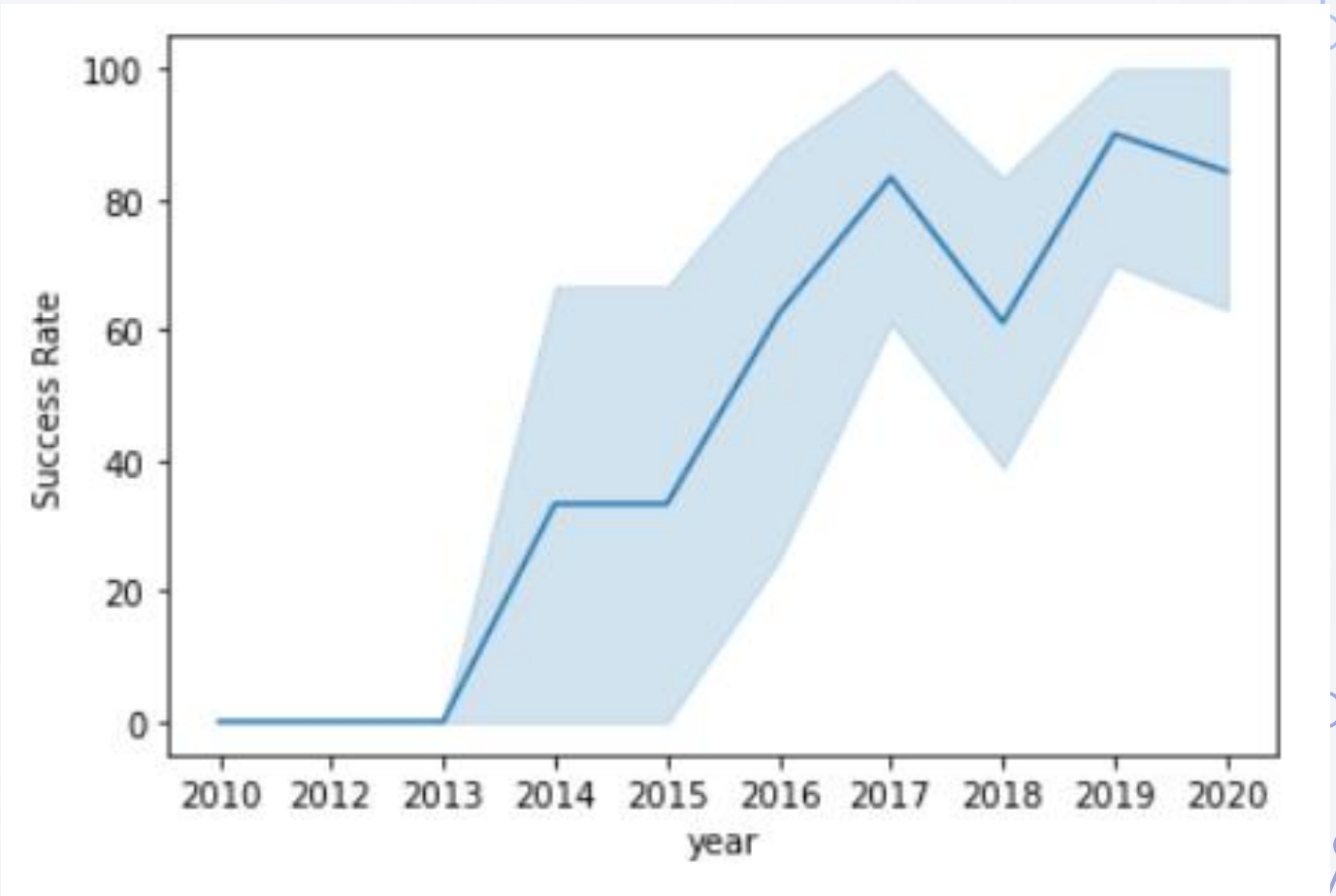
ISS – payload mass are concentrated between 2000 & 4000

Small payload mass only went to LEO, ISS, PO, ES-L1, HEO



Launch Success Yearly Trend

There were a drastic increase of success launches in 2015 till 2017.



All Launch Site Names

```
In [3]: %sql select distinct(LAUNCH_SITE) from SPACEXTBL
```

Using the distinct function in queries will provide all the unique values in the dataset

We have 4 launch site in our data set

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

Launch Site Names Begin with 'CCA'

Using a limit argument will send a command to our dataset to only display the first 5 result that satisfy the query

```
In [4]: %sql select * from SPACEXTBL where LAUNCH_SITE like 'CCA%' limit 5
```

Out[4]:

DATE	time__utc_	booster_version	launch_site	payload	payload_mass__kg_	orbit	customer	mission_outcome	landing__outcome
2010-06-04	None	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	None	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	None	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	None	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	None	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Total Payload Mass

```
In [5]: %sql select sum(PAYLOAD_MASS__KG_) from SPACEXTBL where CUSTOMER = 'NASA (CRS)'
```

Out[5]:

1
45596

To calculate the total payload mass we need to use the sum() function and ensure that it only calculate the data in NASA(CRS) using the WHERE clause

Average Payload Mass by F9 v1.1

```
In [6]: %sql select avg(PAYLOAD_MASS__KG_) from SPACEXTBL where BOOSTER_VERSION = 'F9 v1.1'
```

```
Out[6]:
   1
2928
```

To calculate the average payload mass we need to use the avg() function and ensure that it only calculate the data BOOSTER VERSION = F9 v1.1 using the WHERE clause

First Successful Ground Landing Date

```
In [7]: %sql select min(DATE) from SPACEXTBL where Landing__Outcome = 'Success (ground pad)'
```

Out[7]:

1
2015-12-22

To get the first successful date we can use the min() function on the date column and using the where argument for landing_outcome

Successful Drone Ship Landing with Payload between 4000 and 6000

```
In [8]: %sql select BOOSTER_VERSION from SPACEXTBL where Landing__Outcome = 'Success (drone ship)' and PAYLOAD_MASS__KG_ > 4000 and PAYL
```

Apply a argument to the where clause to only include booster version with payload mass of 4000 to 6000 and ensure only successful launched were included

Out[8]:

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

Here is the list the names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000

Boosters Carried Maximum Payload

```
In [10]: %sql select BOOSTER_VERSION from SPACEXTBL where PAYLOAD_MASS__KG_ = (select max(PAYLOAD_MASS__KG_) from SPACEXTBL)
```

Here is the list the names of the booster which have carried the maximum payload mass

Apply a subquery in the where clause to get the max payload mass

Out[10]:

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

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

```
In [12]: %sql select * from SPACEXTBL where Landing__Outcome like 'Success%' and (DATE between '2010-06-04' and '2017-03-20') order by dat
```

Here is the rank of 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

Out[12]:

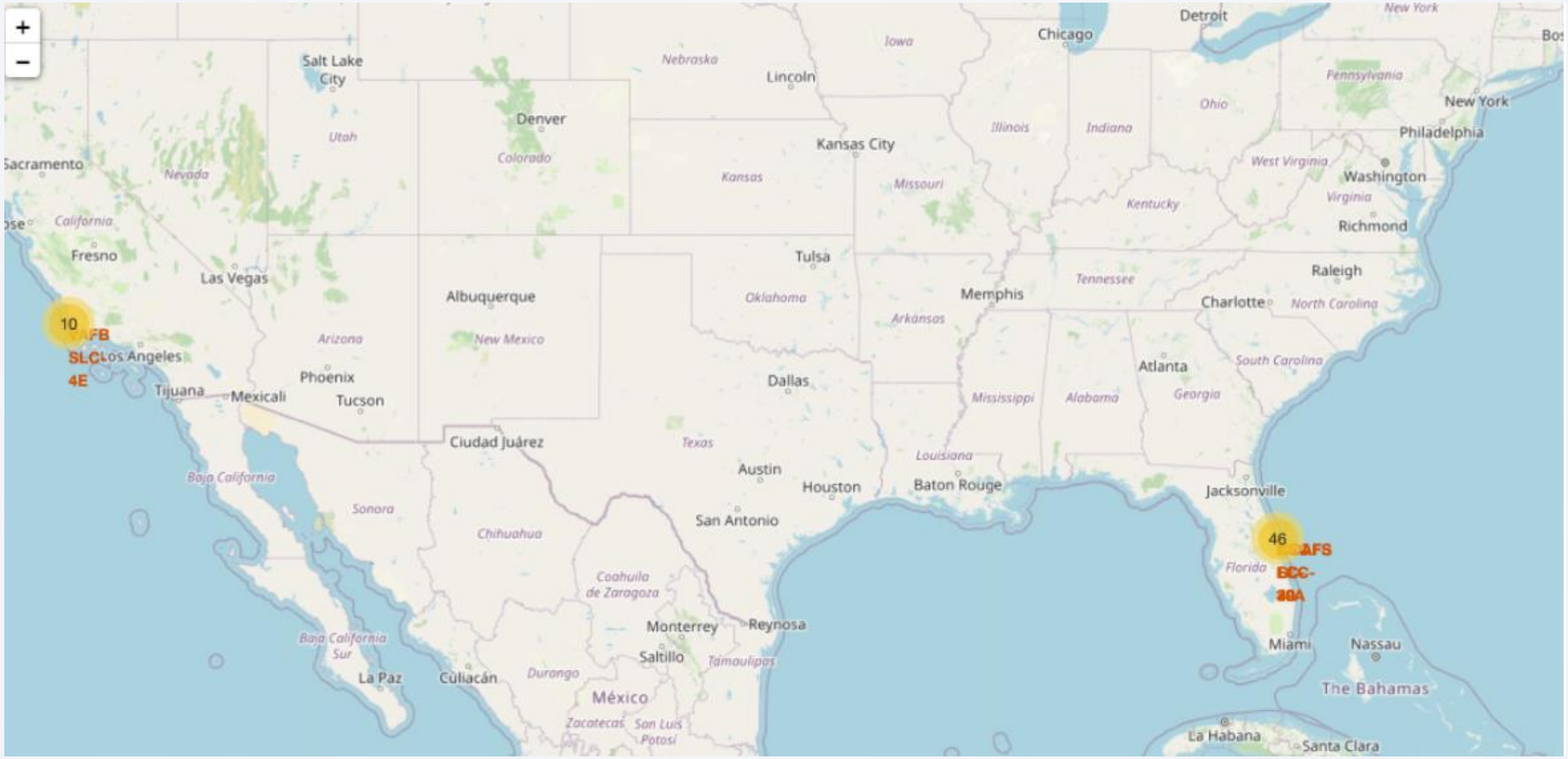
DATE	time__utc_	booster_version	launch_site	payload	payload_mass__kg_	orbit	customer	mission_outcome	landing__outcome
2017-02-19	None	F9 FT B1031.1	KSC LC-39A	SpaceX CRS-10	2490	LEO (ISS)	NASA (CRS)	Success	Success (ground pad)
2017-01-14	None	F9 FT B1029.1	VAFB SLC-4E	Iridium NEXT 1	9600	Polar LEO	Iridium Communications	Success	Success (drone ship)
2016-08-14	None	F9 FT B1026	CCAFS LC-40	JCSAT-16	4600	GTO	SKY Perfect JSAT Group	Success	Success (drone ship)
2016-07-18	None	F9 FT B1025.1	CCAFS LC-40	SpaceX CRS-9	2257	LEO (ISS)	NASA (CRS)	Success	Success (ground pad)
2016-05-27	None	F9 FT B1023.1	CCAFS LC-40	Thaicom 8	3100	GTO	Thaicom	Success	Success (drone ship)
2016-05-06	None	F9 FT B1022	CCAFS LC-40	JCSAT-14	4696	GTO	SKY Perfect JSAT Group	Success	Success (drone ship)
2016-04-08	None	F9 FT B1021.1	CCAFS LC-40	SpaceX CRS-8	3136	LEO (ISS)	NASA (CRS)	Success	Success (drone ship)
2015-12-22	None	F9 FT B1019	CCAFS LC-40	OG2 Mission 2 11 Orbcomm-OG2 satellites	2034	LEO	Orbcomm	Success	Success (ground pad)

Multiple filter must be done. First is the date, then success launch then order it by date.

Section 4

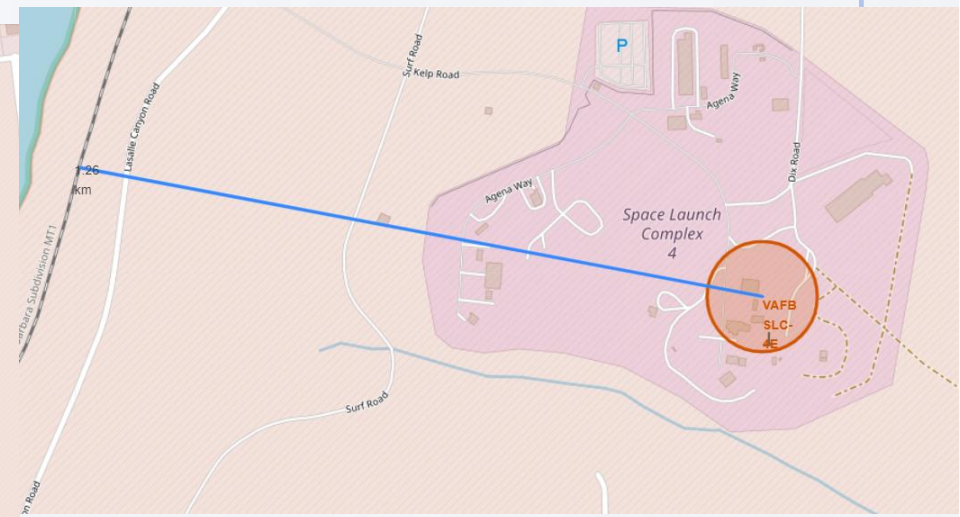
Launch Sites Proximities Analysis

Launch Site Location

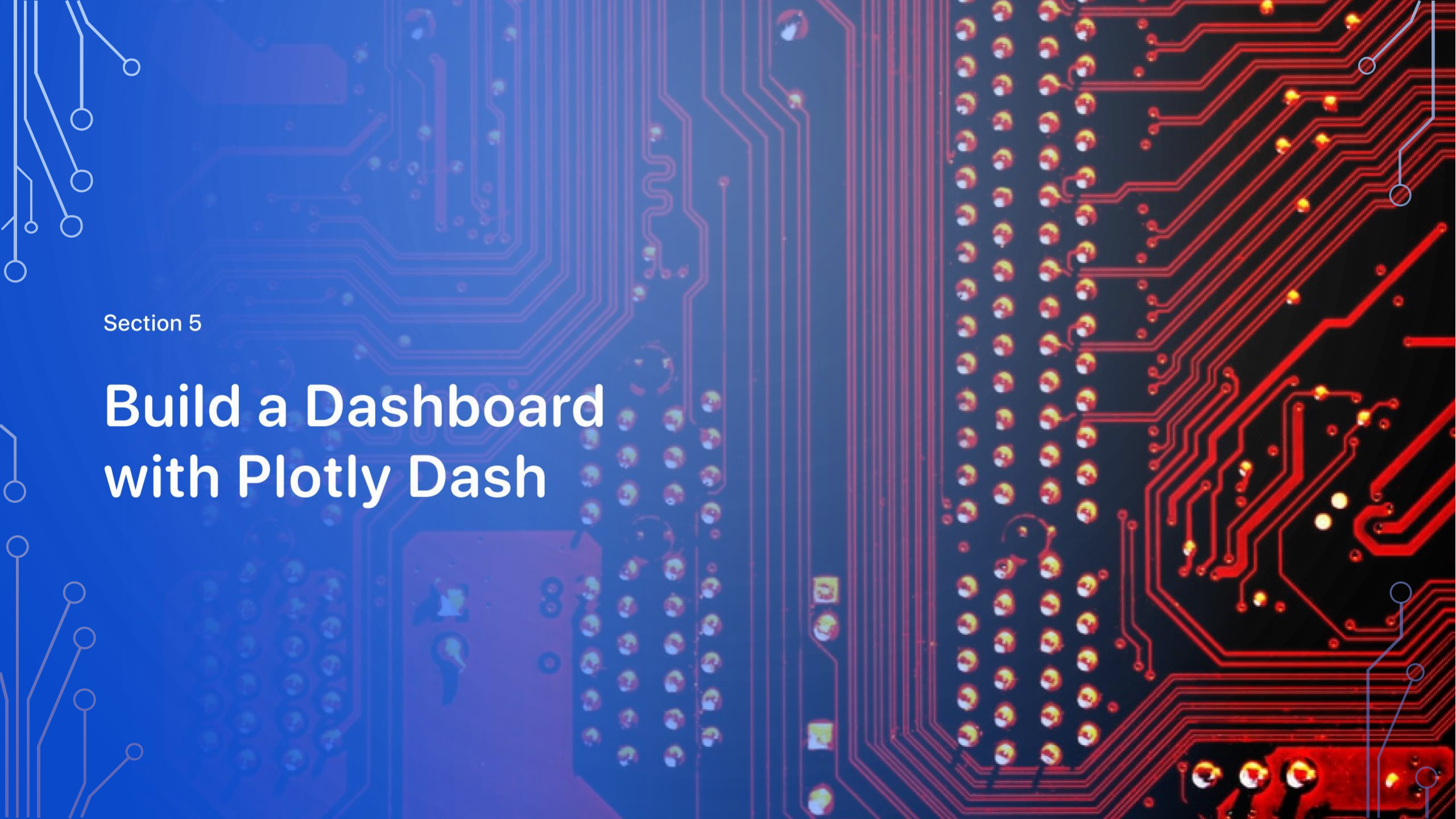


Space X launch site in Florida USA are all near the ocean

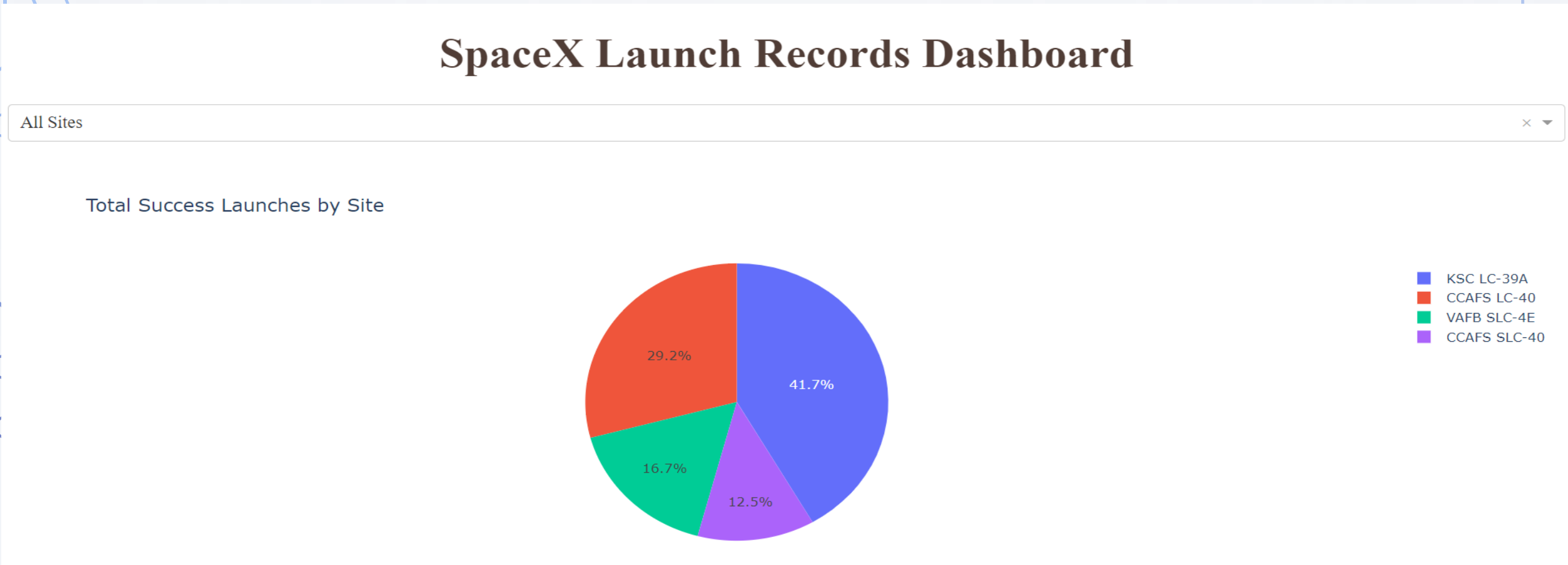
Markers and proximity



Launch site all near railways and sea as shown in the map


Section 5
Build a Dashboard with Plotly Dash

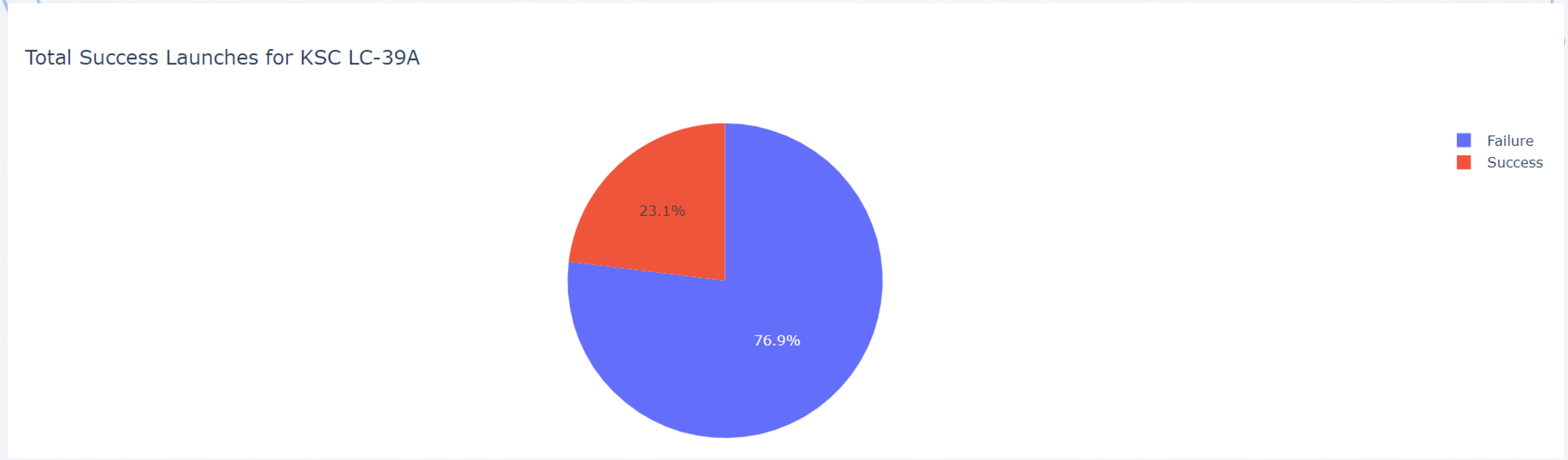
Space X Dashboard



Majority of successful launch site are in KSL LC39A with 41.7%

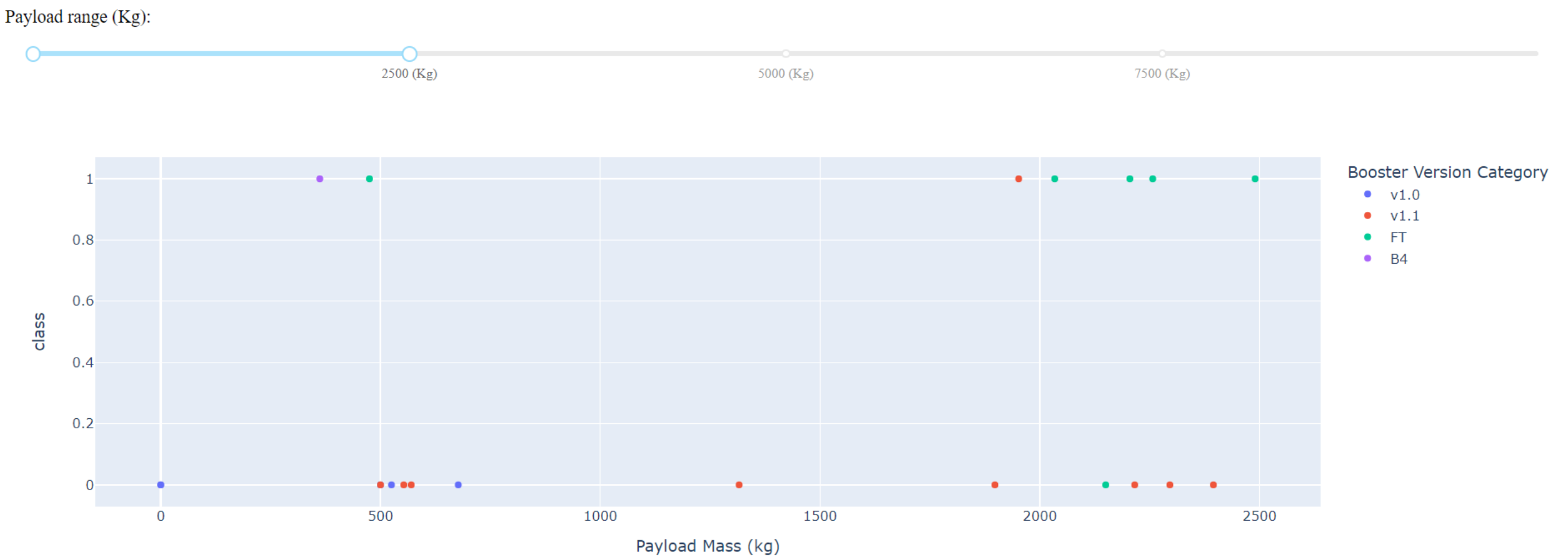
and low successful rate are in CCAFS SLC- 40

Highest Launch Success Ratio : KSC LC 39A



Success rate for KSC LC 39 A is 76.9%

Payload vs Launch Outcome



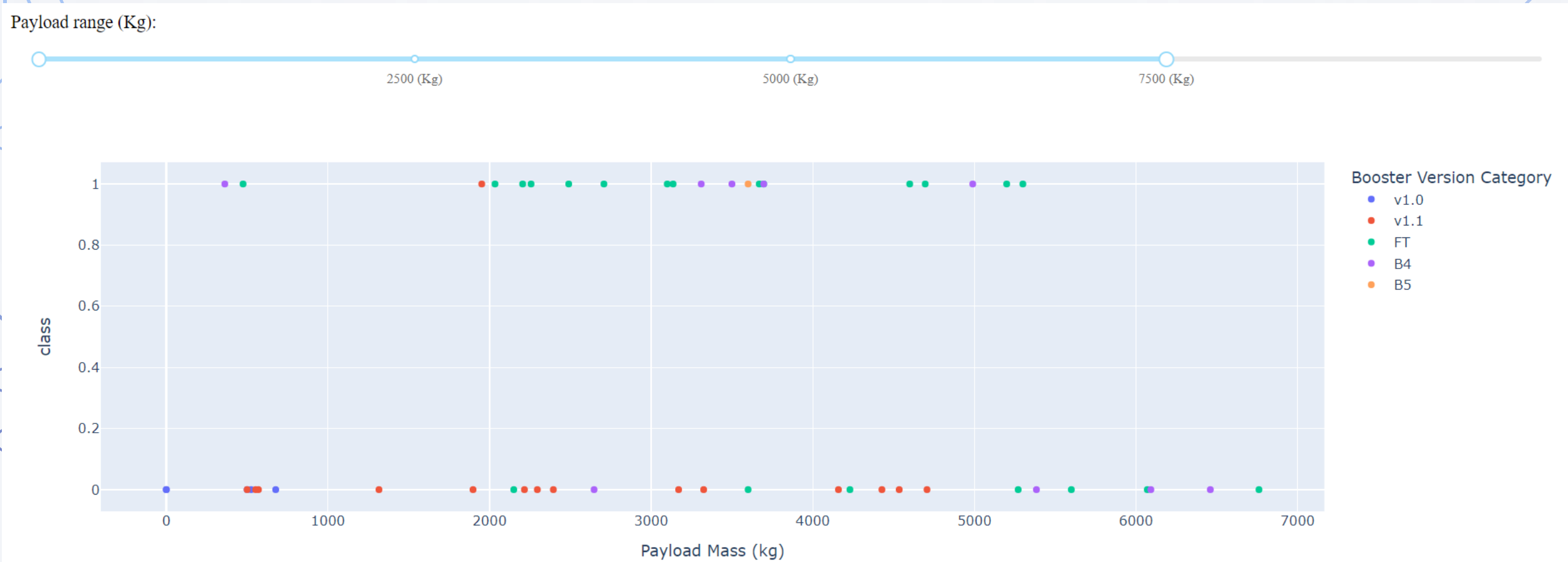
Payload range below 2500kg

There were more failed launches that successful

Payload vs Launch Outcome

Payload range below 5000kg

Payload vs Launch Outcome



Payload range below 7500kg

Payload from 5000kg to 7500kg were all unsuccessful launches

Payload vs Launch Outcome

Payload range below 1000kg

Payload from beyond 8000kg are evenly likeable to failed or succeed.

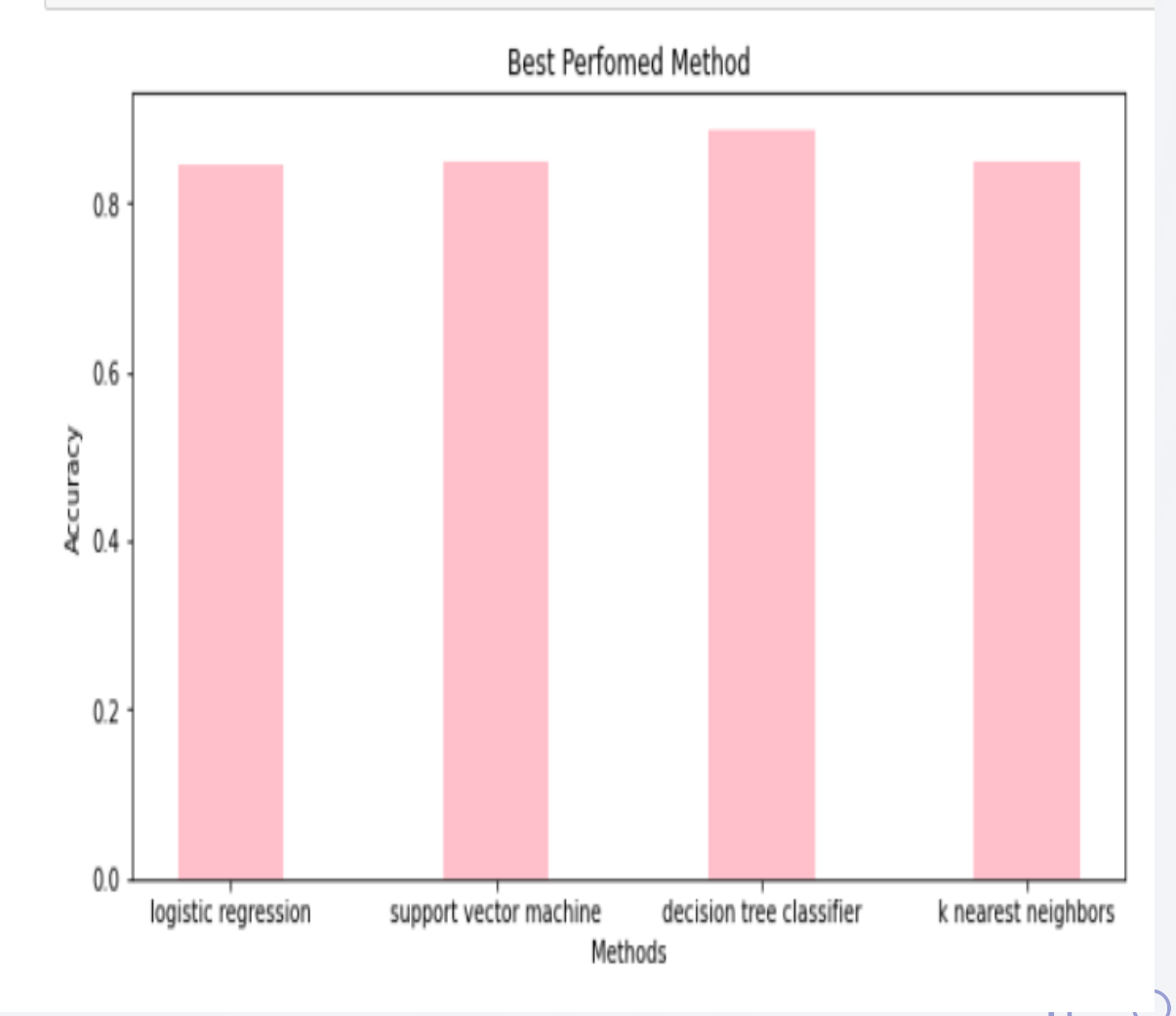
Section 6

Predictive Analysis (Classification)

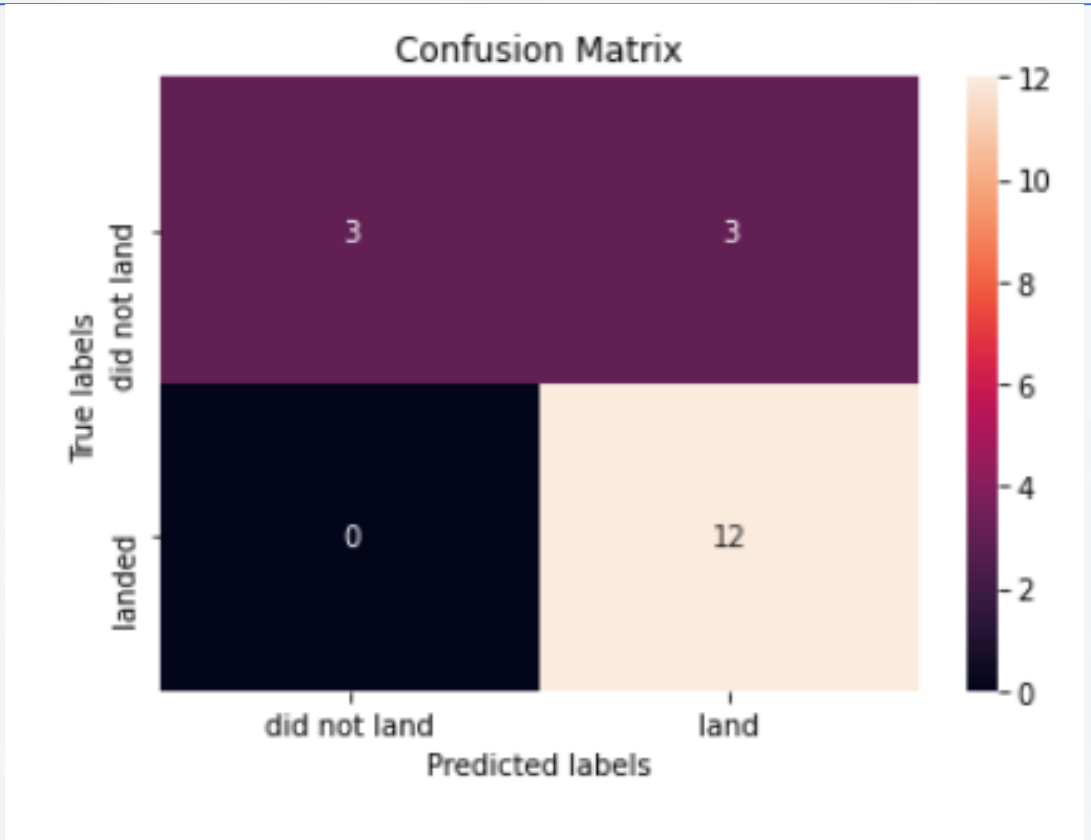
Classification Accuracy

The highest accuracy model is the DECISION TREE with 88% accuracy score

The remaining three have 84% accuracy score



Confusion Matrix



The decision tree correctly predicted 12 launch that landed successfuly

Conclusions

- For higher success rate per launch it is observed that its better to have a payload of 5000kg to 7000kg
- Orbit GTO also have a high success rate
- CCAFS – LS launch site have the top stop of success rate in orbit LEO, CRS, etc
- Decision Tree modeling is the best classification algorithm for our dataset

Thank you!