WINNING SPACE RACE WITH DATA SCIENCE

Paweł Baczyński

2/22/2023

Outline

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

Executive Summary

- SpaceX advertises Falcon 9 rocket launches on its website with a cost of 67 million dollars (as of February 2023); other providers cost upward of 165 million dollars each, much of the savings is since SpaceX can reuse the first stage.
- This study examinate the underlying conditions that lead to the successful rocket launch and then subsequent landing of the first stage of the Falcon 9 rocket. If we can determine if the first stage will land, we can estimate the cost of a launch. That information can be used if an alternate company wants to bid against SpaceX for a rocket launch.
- In order to be able to address the above-mentioned problem, first we collected data from SpaceX REST API at api.spacexdata.com/v4/ and wiki pages at: https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922
- Next, we performed the data wrangling process to transform and map the data from "raw" format into another, more appropriate for analytics. The main goal of the data wrangling at this stage was to map the booster landing status for each row of the data (where: 1 row = 1 mission) into one of two statuses: “1” - if the booster successfully landed and “0” - if it was unsuccessful, as the raw data set distinguish several different cases where the booster landed or did not land successfully. The results are then stored in the new column called: *Class.*
- We used exploratory data analysis (EDA) to investigate and analyze the data sets with help of the statistical graphs and graphical visualization methods to better understand the data variables and the relationship among them.
- We concluded that the success of the missions depends on multiple factors, the most important are Launch Site, Flight Number, Payload, and destination Orbit.
- We can observe that the mission success trend grows over the years, as SpaceX company accumulate the experience: as the flight number increases, the first stage is more likely to land successfully
- At that point we were finally ready to built machine learning models to predict if the Falcon 9 first stage will land successfully in the future. We created 4 prediction models: tree classifier, logistic regression, support vector machine, k nearest neighbors.
- All of them produced the same results and have the same accuracy score, therefore the model we ultimately choose depends on our preferences.

Introduction

In recent years we are witnessing the rapidly decreasing cost of launching objects into orbit.

NASA's space shuttles, which were retired in 2011, costed an average of $1.6 billion per flight, or nearly $30,000 per pound of payload (in 2021 dollars) to reach low-Earth orbit, according to an analysis by the Center for Strategic and International Studies.

Russia's Soyuz rockets can cost anywhere from $53 million to $225 million per launch, working out to more than $8,000 per pound of payload to reach Earth orbit, while Chinese Long March 3B cost around US$50-70 million per launch.

Today however, commercial companies like SpaceX and Rocket Lab created a competitive market that drove down the price of rides to space.

Rocket Lab, an aerospace company from California, debuted its Electron rocket in 2018. The company provides launches to Earth orbit for small satellites, which range from CubeSats roughly the size of a loaf of bread to minifridge-sized spacecraft that weigh less than 1,100 pounds. Rocket Lab charges around $5 million per flight, a cost that works out to roughly $10,000 per pound of payload.

SpaceX offers even more competitive pricing for rides aboard its medium-lift Falcon 9 rocket. The company typically charges around $67 million per launch, or around $1,200 per pound of payload to reach low-Earth orbit, much of the savings is because SpaceX can reuse the first stage

In this analysis, we will predict if the Falcon 9 first stage will land successfully based on some underlaying conditions, such as payload mass, launch site, destination orbit, booster version, etc. this way we can determine the cost of a launch. That information can be later used if an alternate company wants to bid against SpaceX for a rocket launch.

Section 1

Methodology

Data Collection

Data used in the project were collected from two primary sources:

- SpaceX REST API, which provides data about launches, information about the rocket used, payload delivered, launch specifications, landing specifications, and landing outcome. The SpaceX REST API endpoints starts with api.spacexdata.com/v4/.

- Second source for obtaining Falcon 9 Launch data comes from web scraping the Wiki pages at: https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922 using the BeautifulSoup library.

SpaceX REST API

*Please refer to the below GitHub URL for the completed SpaceX API notebook:
https://github.com/Rektorian/stairstostarts/blob/main/1_jupyter_labs_spacex_data_collection_api.ipynb

Data Collection – SpaceX API

Request SpaceX launch data from SpaceX API with the URL

```
spacex_url="https://api.spacexdata.com/v4/launches/past"
response = requests.get(spacex_url)
```

We decode the response content as a Json using .json() and turn it into a Pandas dataframe

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

From the rocket column we would like to learn the booster's name.

```
# Takes the dataset and uses the rocket column to call the API and append the data to the list
def getBoosterVersion(data):
    for x in data['rocket']:
       if x:
        response = requests.get("https://api.spacexdata.com/v4/rockets/"+str(x)).json()
        BoosterVersion.append(response['name'])
```

Data Collection – SpaceX API

From the launchpad we would like to know the name of the launch site being used, the longitude, and the latitude.

```
# Takes the dataset and uses the launchpad column to call the API and append the data to the list
def getLaunchSite(data):
    for x in data['launchpad']:
       if x:
         response = requests.get("https://api.spacexdata.com/v4/launchpads/"+str(x)).json()
         Longitude.append(response['longitude'])
         Latitude.append(response['latitude'])
         LaunchSite.append(response['name'])
```

From the payload we would like to learn the mass of the payload and the orbit that it is going to.

```
# Takes the dataset and uses the payloads column to call the API and append the data to the lists
def getPayloadData(data):
    for load in data['payloads']:
       if load:
        response = requests.get("https://api.spacexdata.com/v4/payloads/"+load).json()
        PayloadMass.append(response['mass_kg'])
        Orbit.append(response['orbit'])
```

Data Collection – SpaceX API

From cores we would like to learn the outcome of the landing, the type of the landing, number of flights with that core, whether gridfins were used, wheter the core is reused, wheter legs were used, the landing pad used, the block of the core which is a number used to separate version of cores, the number of times this specific core has been reused, and the serial of the core.

```
# Takes the dataset and uses the cores column to call the API and append the data to the lists
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cores/"+core['core']).json()
                Block.append(response['block'])
                ReusedCount.append(response['reuse_count'])
                Serial.append(response['serial'])
            else:
                Block.append(None)
                ReusedCount.append(None)
                Serial.append(None)
            Outcome.append(str(core['landing_success'])+' '+str(core['landing_type']))
            Flights.append(core['flight'])
            GridFins.append(core['gridfins'])
            Reused.append(core['reused'])
            Legs.append(core['legs'])
            LandingPad.append(core['landpad'])
```

Data Collection – SpaceX API

Finally let's construct our dataset using the data we have obtained. We combine the columns into a dictionary. Then, we need to create a Pandas data frame from the dictionary launch_dict.

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

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

Data Collection – SpaceX API

Now, we want to keep only Falcon 9 launches. In order to achieve this, we filter the data dataframe using the BoosterVersion column to only keep the Falcon 9 launches. Next, we save the filtered data to a new dataframe called data_falcon9.

```
data_falcon9 = data[data['BoosterVersion'].map(lambda x : x != 'Falcon 1')]
print('Row count after removing Falcon 1 launches:', len(data_falcon9))

# reset the FlgihtNumber column
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9
```

Before we can continue we must deal with these missing values. The LandingPad column will retain None values to represent when landing pads were not used.

Data Collection – SpaceX API

```
data_falcon9.isnull().sum()
```

```
FlightNumber      0
Date              0
BoosterVersion    0
PayloadMass       5
Orbit             0
LaunchSite        0
Outcome           0
Flights           0
GridFins          0
Reused            0
Legs              0
LandingPad       26
Block             0
ReusedCount       0
Serial            0
Longitude         0
Latitude          0
dtype: int64
```

We will deal with the missing values for PayloadMass by replacing np.nan values with the mean value for the column

Data Collection – SpaceX API

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

# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'].fillna(value=PayloadMassMean, inplace=True)
```

```
data_falcon9.isnull().sum()
```

```
FlightNumber       0
Date               0
BoosterVersion     0
PayloadMass        0
Orbit              0
LaunchSite         0
Outcome            0
Flights            0
GridFins           0
Reused             0
Legs               0
LandingPad        26
Block              0
ReusedCount        0
Serial             0
Longitude          0
Latitude           0
dtype: int64
```

Data Collection – SpaceX API

Now we are ready for further data wrangling.

FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial
1	2010-06-04	Falcon 9	6123.547647	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0003
2	2012-05-22	Falcon 9	525.000000	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0005
3	2013-03-01	Falcon 9	677.000000	ISS	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0007
4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None	1.0	0	B1003
5	2013-12-03	Falcon 9	3170.000000	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B1004

Wikipedia

*Please refer to the below GitHub URL for the completed Web Scrapping notebook:
https://github.com/Rektorian/stairstostarts/blob/main/2_jupyter-labs-webscraping.ipynb

Data Collection - Scraping

First, we define the URL of the page from which we want to scrap the data, then we use 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"
# use requests.get() method with the provided static_url
# assign the response to a object
falcon9RawLaunchData = requests.get(static_url)
falcon9RawLaunchData
```

The we create a BeautifulSoup object from the HTML response

```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
falcon9RawLaunchData_html_file = BeautifulSoup(falcon9RawLaunchData.text, "html.parser")
```

Data Collection - Scraping

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 = falcon9RawLaunchData_html_file.find_all('table')
# we take data from the 3rd table
first_launch_table = html_tables[2]
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'):
    #print(th)
    colName = extract_column_from_header(th)
    if colName is not None and len(colName) > 0:
        column_names.append(colName)
```

Data Collection - Scraping

Now we can create a data frame by parsing the launch HTML tables and we are ready for data wrangling

	Flight No.	Launch site	Payload	Payload mass	Orbit	Customer	Launch outcome	Version Booster	Booster landing	Date	Time
0	1	CCAFS	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success\n	F9 v1.0B0003.1	Failure	4 June 2010	18:45
1	2	CCAFS	Dragon	0	LEO	NASA	Success	F9 v1.0B0004.1	Failure	8 December 2010	15:43
2	3	CCAFS	Dragon	525 kg	LEO	NASA	Success	F9 v1.0B0005.1	No attempt\n	22 May 2012	07:44
3	4	CCAFS	SpaceX CRS-1	4,700 kg	LEO	NASA	Success\n	F9 v1.0B0006.1	No attempt	8 October 2012	00:35
4	5	CCAFS	SpaceX CRS-2	4,877 kg	LEO	NASA	Success\n	F9 v1.0B0007.1	No attempt\n	1 March 2013	15:10

Data Wrangling

In this part, we will look at some of the attributes of the collected data.

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.

In this part we will convert those outcomes into Labels with 1 means the booster successfully landed 0 means it was unsuccessful for each row of the data.

*Please refer to the below GitHub URL for the completed Data Wrangling notebook:
https://github.com/Rektorian/stairstostarts/blob/main/3_labs-jupyter-spacex-Data%20wrangling.ipynb

Data Wrangling

First, we use the method .value_counts() on the column Outcome to determine the number of landing_outcomes. Then assign it to a variable landing_outcomes.

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

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

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

Data Wrangling

```
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
```

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

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

Data Wrangling

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

for outcome in df['Outcome']:
    if outcome in bad_outcomes:
        landing_class.append(0)
    else:
        landing_class.append(1)
```

Finally, we add our list into our main dataframe

```
df['Class']=landing_class
```

Data Wrangling

BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577366	28.561857	0
Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366	28.561857	0
Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.577366	28.561857	0
Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	0	B1003	-120.610829	34.632093	0
Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561857	0
Falcon 9	3325.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1005	-80.577366	28.561857	0
Falcon 9	2296.000000	ISS	CCAFS SLC 40	True Ocean	1	False	False	True	NaN	1.0	0	B1006	-80.577366	28.561857	1
Falcon 9	1316.000000	LEO	CCAFS SLC 40	True Ocean	1	False	False	True	NaN	1.0	0	B1007	-80.577366	28.561857	1

EDA with Data Visualization

In this part we used the statistical graphs and graphical visualization methods to analyze and investigate the data sets. Exploratory data analysis allows us to manipulate the data in order to discover patterns, spot anomalies, test a hypothesis, check assumptions and understand the data variables and the relationship among them.

The following charts were created:

- FlightNumber vs. PayloadMass – to get the outcome of the launch based on the FlightNumber and PayloadMass
- FlightNumber vs LaunchSite - to get the outcome of the launch based on the FlightNumber and LaunchSite
- Relationship between Payload and Launch Site Plot Plot
- Relationship between success rate of each orbit type Plot
- Relationship between FlightNumber and Orbit Plot
- Relationship between Payload and Orbit Plot
- Launch success yearly trend - to get the average launch success trend over the year

*Please refer to the below GitHub URL for the completed notebook:

https://github.com/Rektorian/stairstostarts/blob/main/5_jupyter-labs-eda-dataviz.ipynb

EDA with SQL

Next, we perform EDA using data gathered in the database to understand the data even better.

The following SQL Queries were invoked:

- **Display the names of the unique launch sites in the space mission:**
  %sql SELECT DISTINCT Launch_Site AS Unique_Launch_Sites FROM SPACEXTBL;
- **Display 5 records where launch sites begin with the string 'CCA':**
  %sql SELECT * FROM SPACEXTBL WHERE Launch_Site LIKE '%CCA%' LIMIT 5;
- **Display the total payload mass carried by boosters launched by NASA (CRS):**
  %sql SELECT SUM(PAYLOAD_MASS__KG_) AS total_payload_mass FROM SPACEXTBL WHERE Customer='NASA (CRS)';
- **Display average payload mass carried by booster version F9 v1.1:**
  %sql SELECT AVG(PAYLOAD_MASS__KG_) AS average_payload_mass FROM SPACEXTBL WHERE Booster_Version='F9 v1.1';
- **List the date when the first succesful landing outcome in ground pad was achieved:**
  %sql SELECT MIN(Date) AS First_Succesful_Landing FROM SPACEXTBL WHERE LANDING__OUTCOME = 'Success (ground pad)';
- **List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000:**
  %sql SELECT BOOSTER_VERSION, PAYLOAD_MASS__KG_, LANDING__OUTCOME FROM SPACEXTBL WHERE \
  landing__outcome = 'Success (drone ship)' AND (PAYLOAD_MASS__KG_ > 4000 AND PAYLOAD_MASS__KG_ < 6000);
- **List the total number of successful and failure mission outcomes:**
  %sql SELECT mission_outcome, COUNT(*) as number FROM Spacextbl GROUP BY mission_outcome;
- **List the names of the booster_versions which have carried the maximum payload mass:**
  %sql SELECT booster_version, payload, payload_mass__kg_ FROM Spacextbl WHERE payload_mass__kg_ = (SELECT MAX(payload_mass__kg_) FROM Spacextbl);

EDA with SQL

- **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:**
  %sql SELECT MONTH(DATE) AS Month, booster_version, launch_site, landing__outcome \
  FROM Spacextbl \
  WHERE landing__outcome LIKE '%Failure (drone ship)%' AND YEAR(DATE) = '2015';
- **Rank the count of successful landing_outcomes between the date 04-06-2010 and 20-03-2017 in descending order:**
  %sql SELECT YEAR(Date) AS year, Count(*) as successful_landings FROM Spacextbl \
  WHERE Date BETWEEN '2010-06-04' AND '2017-03-20' AND (landing__outcome LIKE '%Success%') GROUP BY YEAR(Date) ORDER BY successful_landings DESC;

*Please refer to the below GitHub URL for the completed notebook:

https://github.com/Rektorian/stairstostarts/blob/main/4_jupyter-labs-eda-sql-coursera_sqllite.ipynb

Interactive Map with Folium

The launch success rate may depend on many factors such as payload mass, orbit type, and so on. It may also depend on the location and proximities of a launch site, i.e., the initial position of rocket trajectories. Finding an optimal location for building a launch site certainly involves many factors and hopefully we could discover some of the factors by analyzing the existing launch site locations.

EDA provided us some preliminary correlations between the launch site and success rates. In this section, we performed interactive visual analytics using Folium. The created map has the following attributes:

- It shows all launch sites based on their coordinates (Longitude & Latitude). In order to achieve this, we used *folium.Circle* and *folium.Marker* to add a highlighted circle area with a text label on specific coordinates.
- It marks the success/failed launches for each site on the map using the *MarkerCluster* object.
- Finally, it shows the distances between the launch site CCAFS LC-40 and its important proximities (city, railroad, highway and coastline) that may impact the overall success of the missions using the *folium.PolyLine* object.

*Please refer to the below GitHub URL for the completed notebook:

https://github.com/Rektorian/stairstostarts/blob/main/6_lab_jupyter_launch_site_location.ipynb

Dashboard with Plotly Dash

In this section we build a dashboard application with the Python Plotly Dash package. The dashboard application contains interactive components that allow us to find more insights from the SpaceX dataset more easily than with static graphs. We can do this via input components:

- A dropdown list from which user can select the launch site
- a slider to select payload range

By specifying those we can interact with Pie Chart that shows the success rate per launch site and a Scatter Chart that show the correlation between payload and launch success

*Please refer to the below GitHub URL for the completed notebook:

https://github.com/Rektorian/stairstostarts/blob/main/7_spacex_dash_app.py

Predictive Analysis (Classification)

In this part, we created machine learning models to predict if the first stage will land based on the previously collected and wrangled data.

In order to achieve this, the data was standardized and then split into training and test sets. Then we built logistic regression, support vector machine, k nearest neighbors and decision tree classifier models, and evaluated them based on their accuracy.

*Please refer to the below GitHub URL for the completed notebook:

https://github.com/Rektorian/stairstostarts/blob/main/8_SpaceX_Machine%20Learning%20Prediction_Part_5.ipynb

Predictive Analysis (Classification)

Predictive Analysis (Classification)

We create four different machine learning models

1.
We create a **logistic regression** *lr* object and a GridSearchCV object *logreg_cv* to find the best parameters for our model.

```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression() # create logistic regression object
logreg_cv = GridSearchCV(lr, param_grid = parameters, scoring='accuracy', 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
```

We calculate the accuracy of our model on the test data using the method ***score***

```
logreg_cv.score(X_test, Y_test)
```

```
0.8333333333333334
```

We plot the confusion matrix to visualize our results:
yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)

2.
We create a **support vector machine** *svm* object and a GridSearchCV object *svm_cv* to find the best parameters for our model.

```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
svm_cv = GridSearchCV(svm, param_grid = parameters, scoring='accuracy', cv = 10)
svm_cv.fit(X_train, Y_train)
```

```
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
```

We calculate the accuracy of our model on the test data using the method ***score***

```
svm_cv.score(X_test, Y_test)
```

```
0.8333333333333334
```

We plot the confusion matrix to visualize our results:
yhat = svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)

3.
We create a **a k nearest neighbors** *KNN* object and a GridSearchCV object *knn_cv* to find the best parameters for our model.

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

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

```
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
```

We calculate the accuracy of our model on the test data using the method ***score***

```
knn_cv.score(X_test, Y_test)
```

```
0.8333333333333334
```

We plot the confusion matrix to visualize our results:
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)

Predictive Analysis (Classification)

4.
We create four different machine learning models

We create a **decision tree classifier** *tree* object and a GridSearchCV object *tree_cv* to find the best parameters for our model.

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

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

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 8, 'max_features': 'auto', 'min_samples_leaf': 4, 'min_samples_split': 10, 'splitter': 'random'}
accuracy : 0.876785714285714
```

We calculate the accuracy of our model on the test data using the method ***score***

```
tree_cv.score(X_test, Y_test)
```

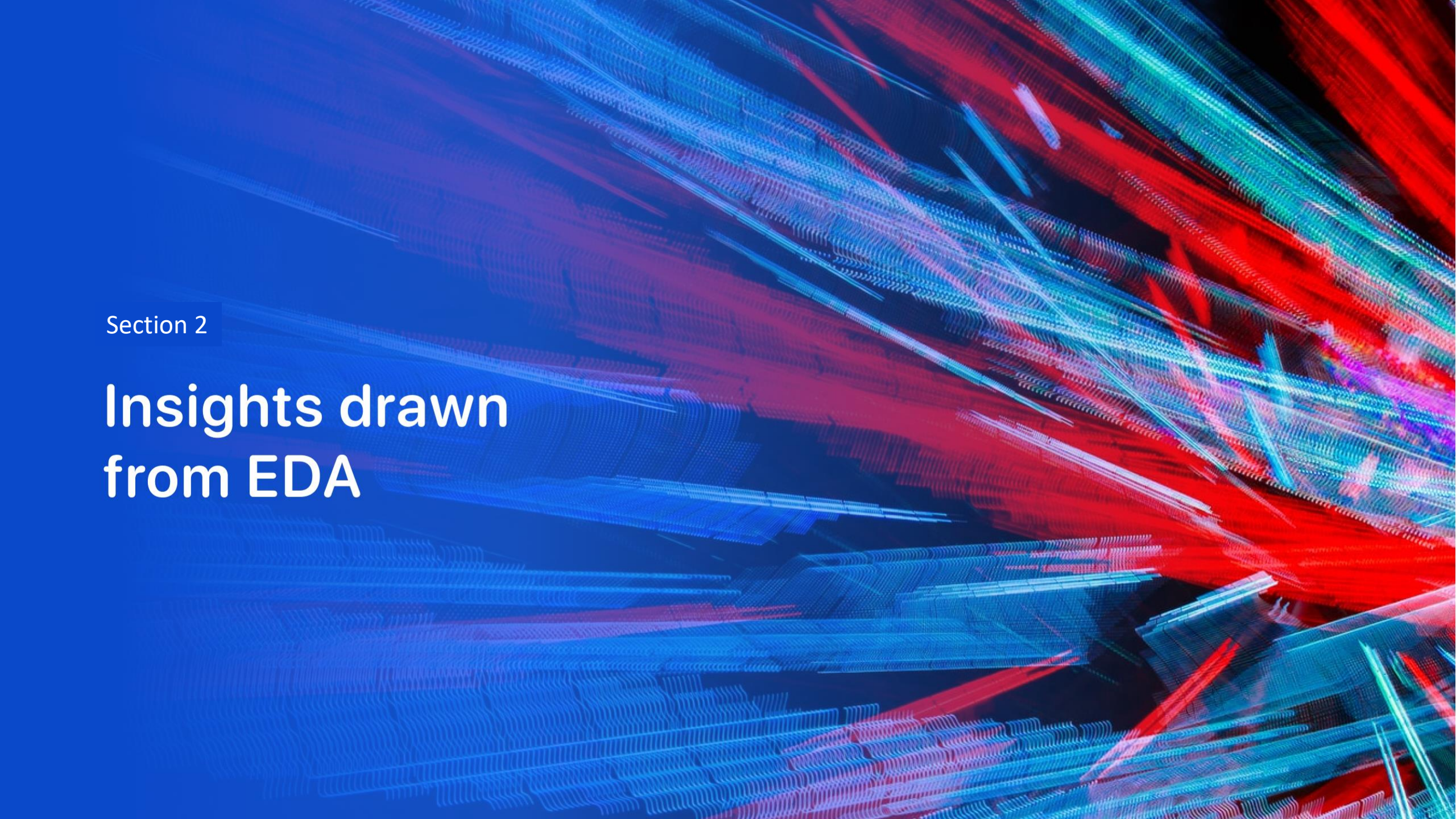
```
0.8333333333333334
```

We plot the confusion matrix to visualize our results:
yhat = tree_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)

Results

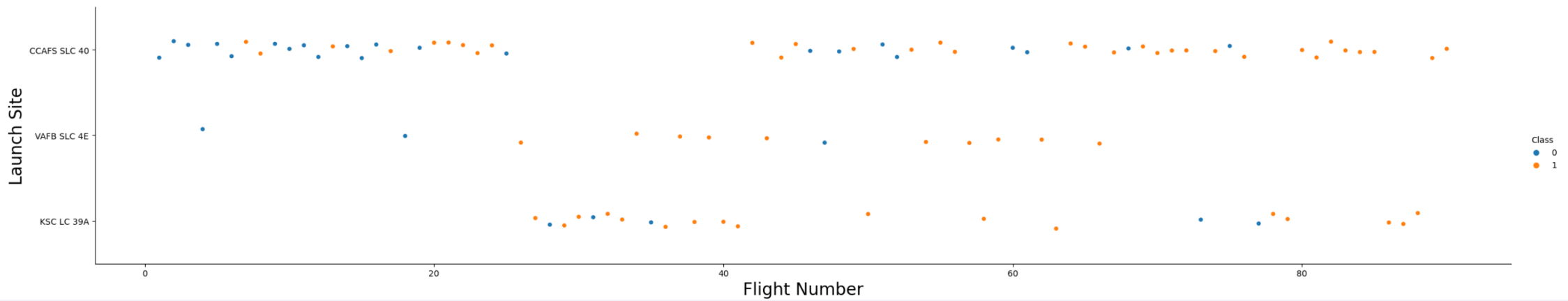
Exploratory data analysis results

As visible on the above screenshots and in the referenced notebook, all tested algorithms basically gave the same results with accuracy around **8.3**, which is a pretty decent score.


Section 2
Insights drawn from EDA

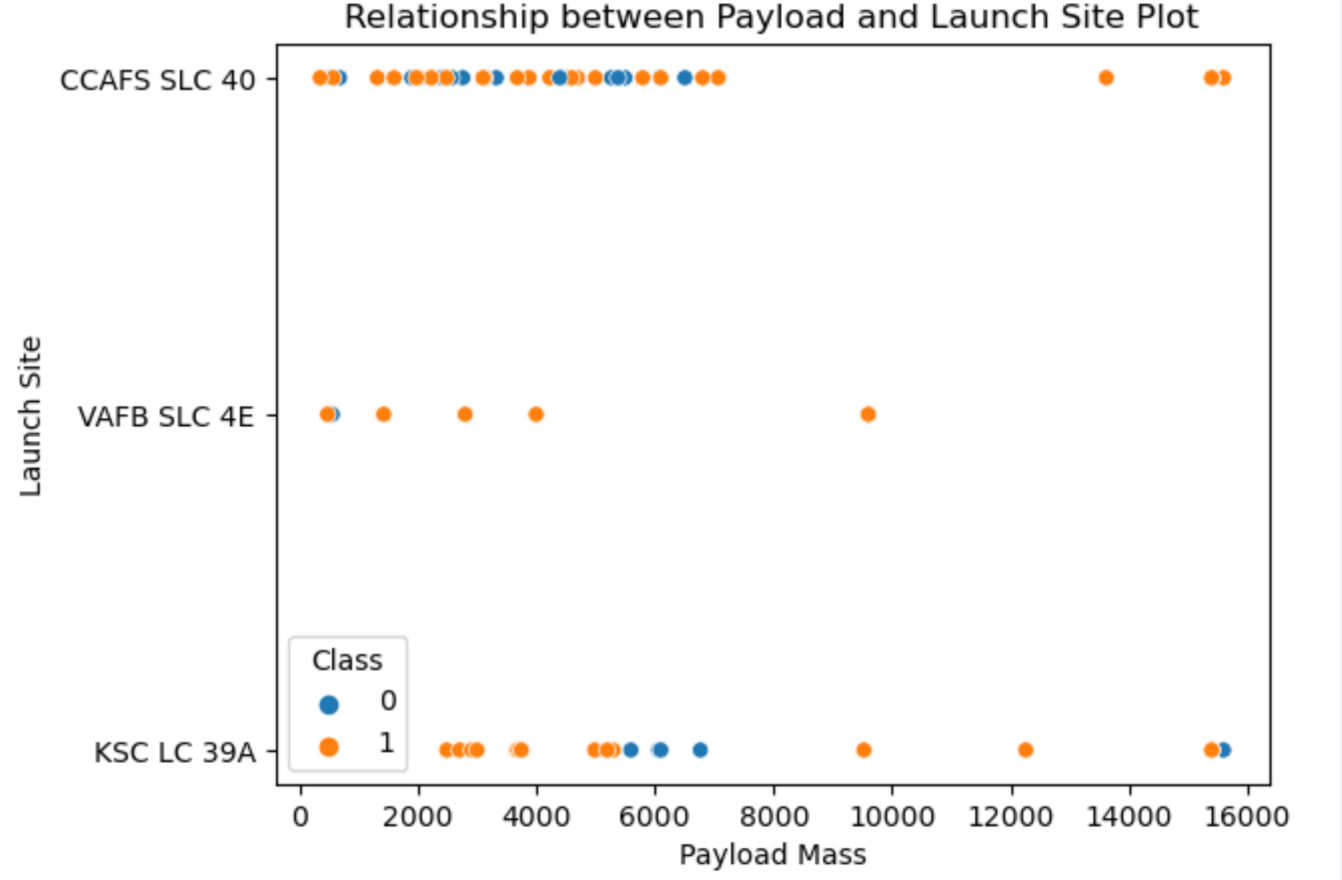
Flight Number vs. Launch Site

Below plot shows the FlightNumber vs LaunchSite and overlay the outcome of the launch.
We see that as the flight number increases, the first stage is more likely to land successfully.
We can also notice that the LaunchSite CCAFS SLC 40 has the lowest success ratio compared to the others.



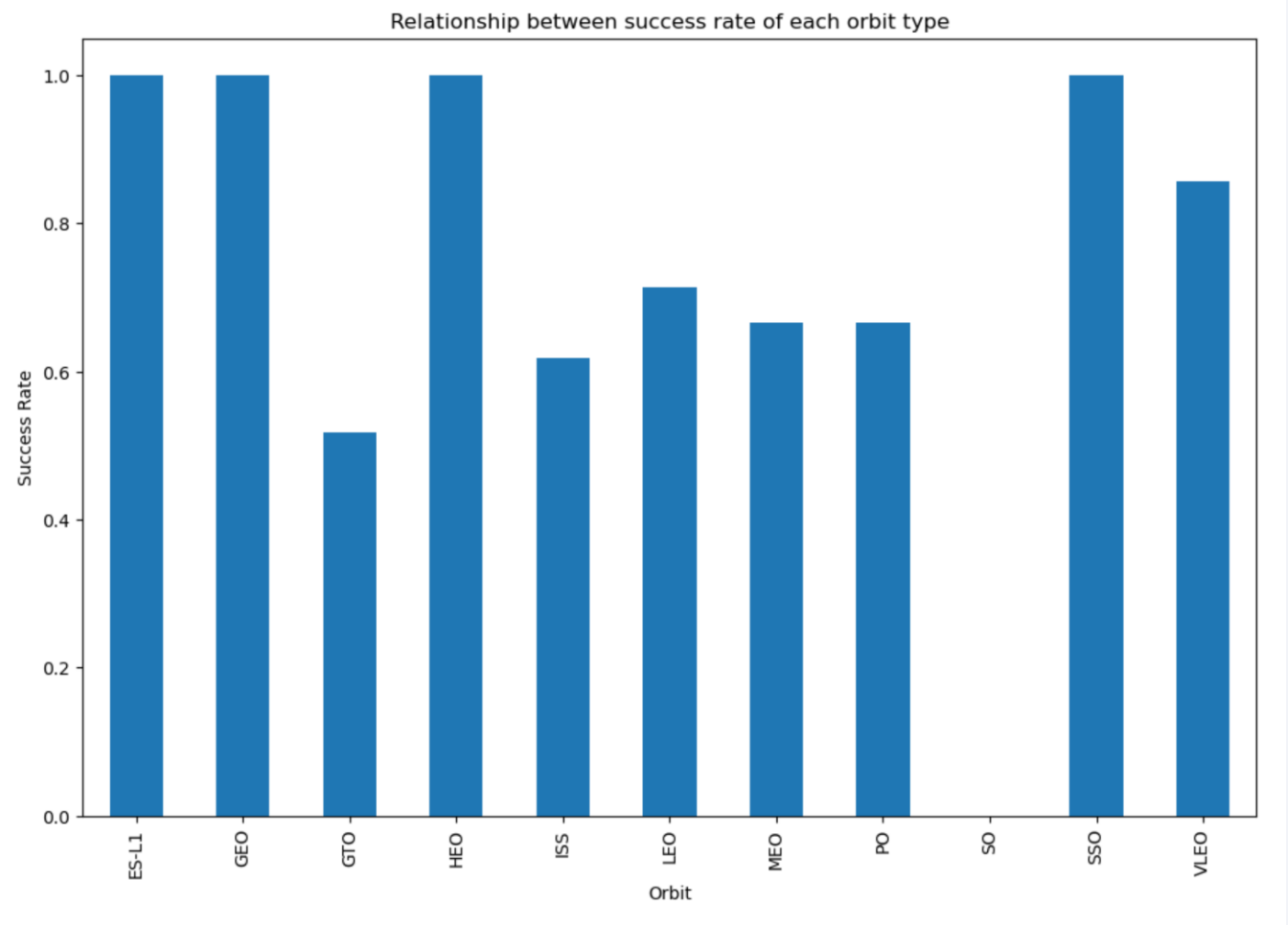
Payload vs. Launch Site

In order to observe the relationship between launch sites and their payload mass, we create the “Relationship between Payload and Launch Site Plot”. We see that for the VAFB-SLC launch site there are no rockets launched for heavypayload mass (greater than 10000).



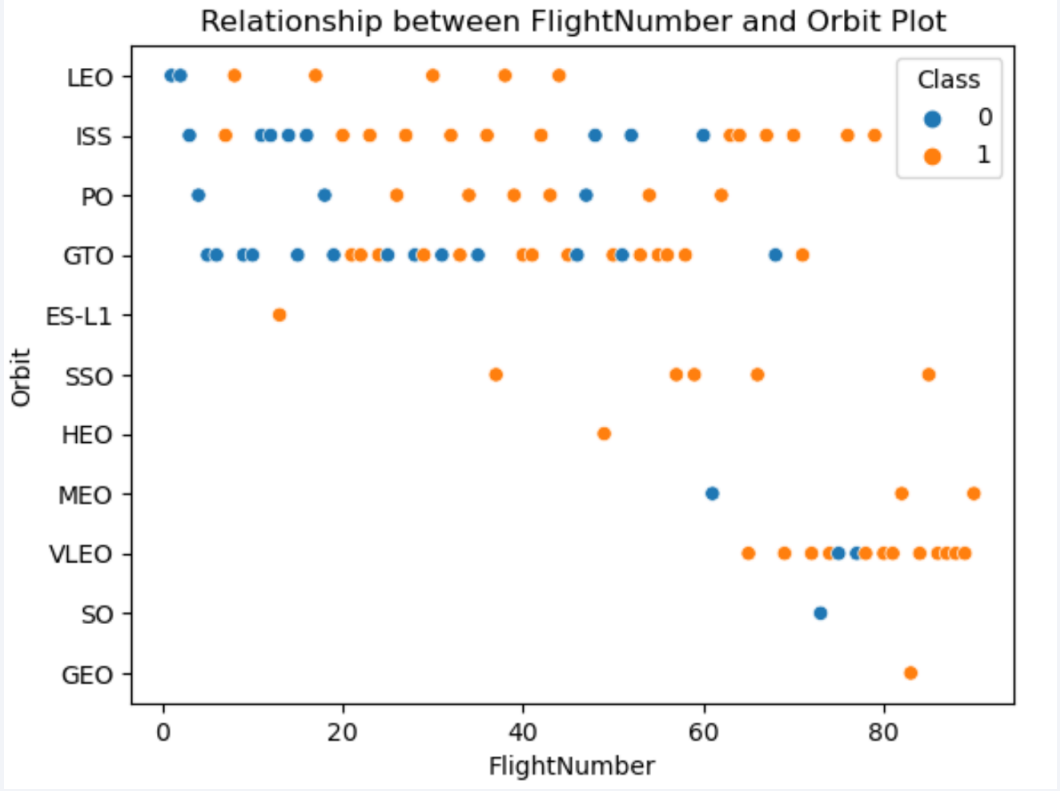
Success Rate vs. Orbit Type

On the “Relationship between success rate of each orbit type” plot, we can observe that launches for orbits ES-L1, GEO, HEO and SSO have 100% success rate, while launch for orbit SO has success rate of 0%. Other missions are between those two extrema's.



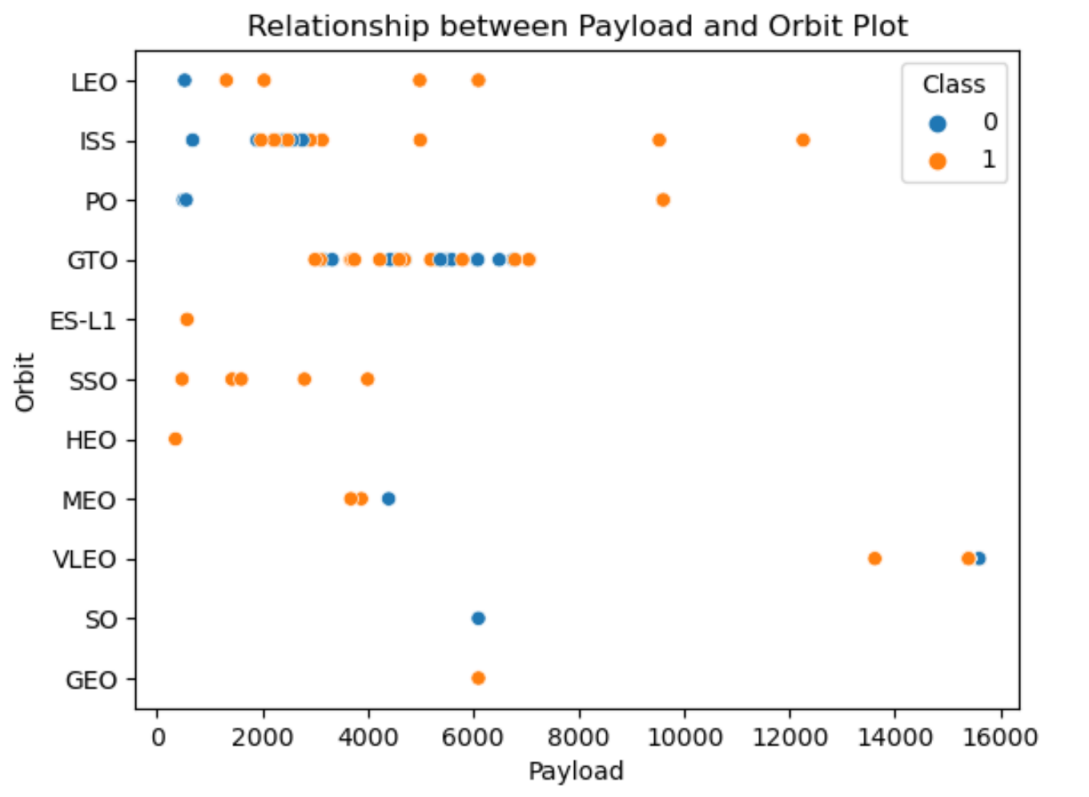
Flight Number vs. Orbit Type

On the below plot, for each orbit, we want to see if there is any relationship between FlightNumber and Orbit type. We can see that in the LEO orbit the Success appears related to the number of flights; on the other hand, there seems to be no relationship between flight number when in GTO orbit.



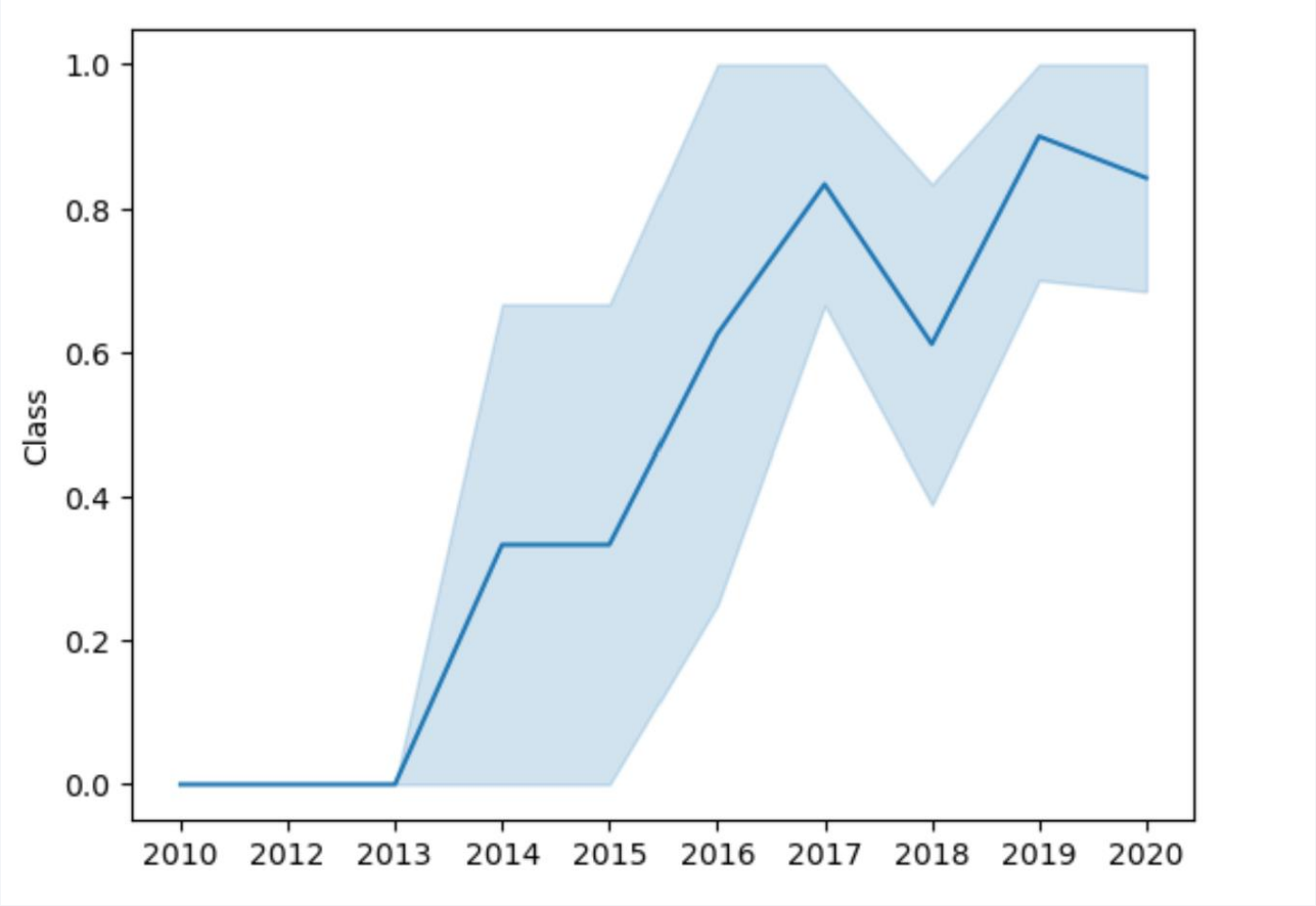
Payload vs. Orbit Type

Similarly, we can plot the Payload vs. Orbit scatter point charts to reveal the relationship between Payload and Orbit type. With heavy payloads the successful landing or positive landing rate are more for Polar, LEO and ISS. However, for GTO we cannot distinguish this well as both positive landing rate and negative landing (unsuccessful mission) are both there here.



Launch Success Yearly Trend

Finally, we create a plot of the average missions success rate, starting from 2013. We can see that the success rate since 2013 kept increasing overlay until 2020 where our dataset ends.



All Launch Site Names

The following query displays the names of the unique launch sites in the space mission:

```
%sql SELECT DISTINCT Launch_Site AS Unique_Launch_Sites FROM SPACEXTBL;
```

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

Launch Site Names Begin with 'CCA'

Display 5 records where launch sites begin with the string ‘CCA:

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

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

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

```
%sql SELECT SUM(PAYLOAD_MASS__KG_) AS total_payload_mass FROM SPACEXTBL WHERE Customer='NASA (CRS)';
```

total_payload_mass
45596

Average Payload Mass by F9 v1.1

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

```
%sql SELECT AVG(PAYLOAD_MASS__KG_) AS average_payload_mass FROM SPACEXTBL WHERE Booster_Version='F9 v1.1';
```

average_payload_mass
2928

First Successful Ground Landing Date

List the date when the first succesful landing outcome in ground pad was achieved:

```
%sql SELECT MIN(Date) AS First_Succesful_Landing FROM SPACEXTBL WHERE LANDING__OUTCOME = 'Success (ground pad)';
```

first_succesful_landing
2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

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

```
%sql SELECT BOOSTER_VERSION, PAYLOAD_MASS__KG_, LANDING__OUTCOME FROM SPACEXTBL WHERE \

landing__outcome = 'Success (drone ship)' AND (PAYLOAD_MASS__KG_ > 4000 AND PAYLOAD_MASS__KG_ < 6000);
```

booster_version	payload_mass__kg_	landing__outcome
F9 FT B1022	4696	Success (drone ship)
F9 FT B1026	4600	Success (drone ship)
F9 FT B1021.2	5300	Success (drone ship)
F9 FT B1031.2	5200	Success (drone ship)

Total Number of Successful and Failure Mission Outcomes

List of the total number of successful and failure mission outcomes:

%sql SELECT mission_outcome, COUNT(*) as number FROM Spacextbl GROUP BY mission_outcome;

mission_outcome	number
Failure (in flight)	1
Success	99
Success (payload status unclear)	1

Boosters Carried Maximum Payload

List of the List the names of the booster_versions which have carried the maximum payload mass:

%sql SELECT booster_version, payload, payload_mass__kg_ FROM Spacextbl WHERE payload_mass__kg_ = (SELECT MAX(payload_mass__kg_) FROM Spacextbl);

booster_version	payload	payload_mass__kg_
F9 B5 B1048.4	Starlink 1 v1.0, SpaceX CRS-19	15600
F9 B5 B1049.4	Starlink 2 v1.0, Crew Dragon in-flight abort test	15600
F9 B5 B1051.3	Starlink 3 v1.0, Starlink 4 v1.0	15600
F9 B5 B1056.4	Starlink 4 v1.0, SpaceX CRS-20	15600
F9 B5 B1048.5	Starlink 5 v1.0, Starlink 6 v1.0	15600
F9 B5 B1051.4	Starlink 6 v1.0, Crew Dragon Demo-2	15600
F9 B5 B1049.5	Starlink 7 v1.0, Starlink 8 v1.0	15600
F9 B5 B1060.2	Starlink 11 v1.0, Starlink 12 v1.0	15600
F9 B5 B1058.3	Starlink 12 v1.0, Starlink 13 v1.0	15600
F9 B5 B1051.6	Starlink 13 v1.0, Starlink 14 v1.0	15600
F9 B5 B1060.3	Starlink 14 v1.0, GPS III-04	15600
F9 B5 B1049.7	Starlink 15 v1.0, SpaceX CRS-21	15600

2015 Launch Records

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

```
%sql SELECT MONTH(DATE) AS Month, booster_version, launch_site, landing__outcome \

FROM Spacextbl \

WHERE landing__outcome LIKE '%Failure (drone ship)%' AND YEAR(DATE) = '2015';
```

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

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

Rank the count of successful landing_outcomes between the date 04-06-2010 and 20-03-2017 in descending order:

```
%sql SELECT YEAR(Date) AS year, Count(*) as successful_landings FROM Spacextbl \

WHERE Date BETWEEN '2010-06-04' AND '2017-03-20' AND (landing__outcome LIKE '%Success%') GROUP BY YEAR(Date)
ORDER BY successful_landings DESC;
```

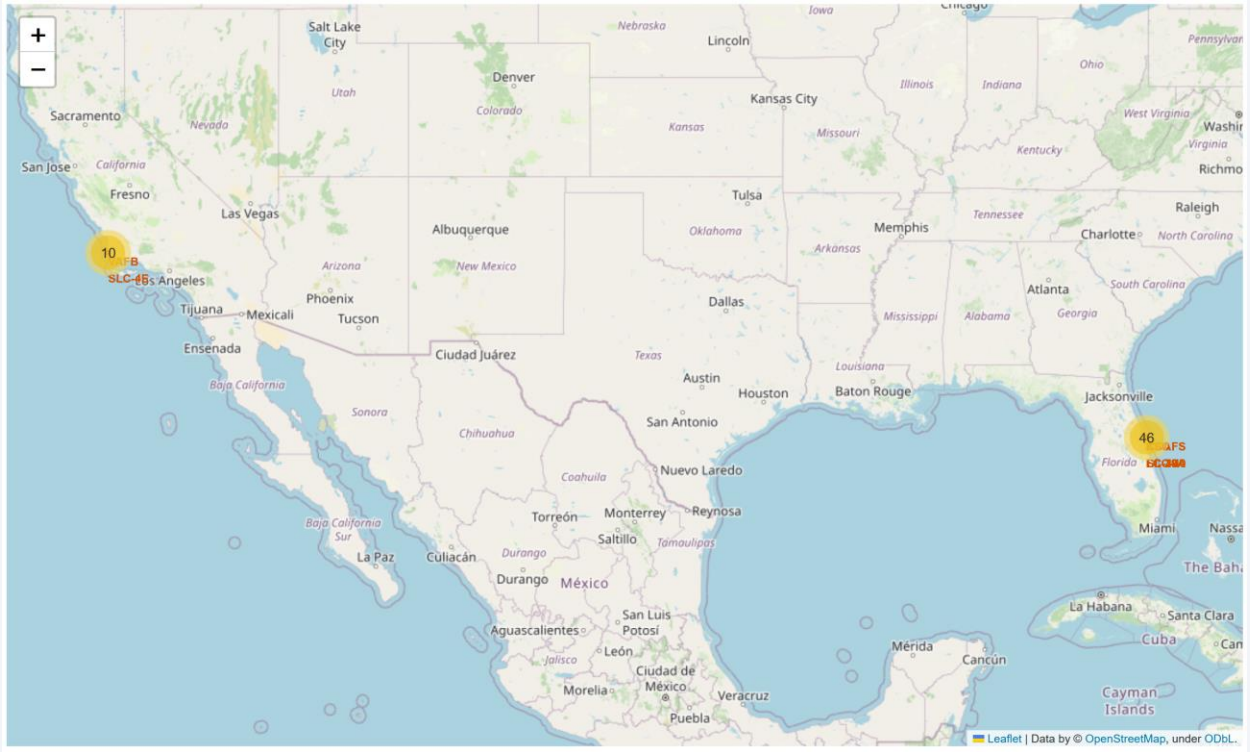
YEAR	successful_landings
2016	5
2017	2
2015	1

Section 3

Launch Sites Proximities Analysis

SPACEX Launch Sites Map

- Map on the right shows all launch sites based on their coordinates (Longitude & Latitude). In order to achieve this, we used folium.Circle and folium.Marker to add a highlighted circle area with a text label on specific coordinates.
- We can see that the launch sites are localized in close proximity to ocean in the south part of the USA. This is because rockets can most easily reach satellite orbits if launched near the equator in an easterly direction, as this maximizes use of the Earth's rotational speed (465 m/s at the equator).*



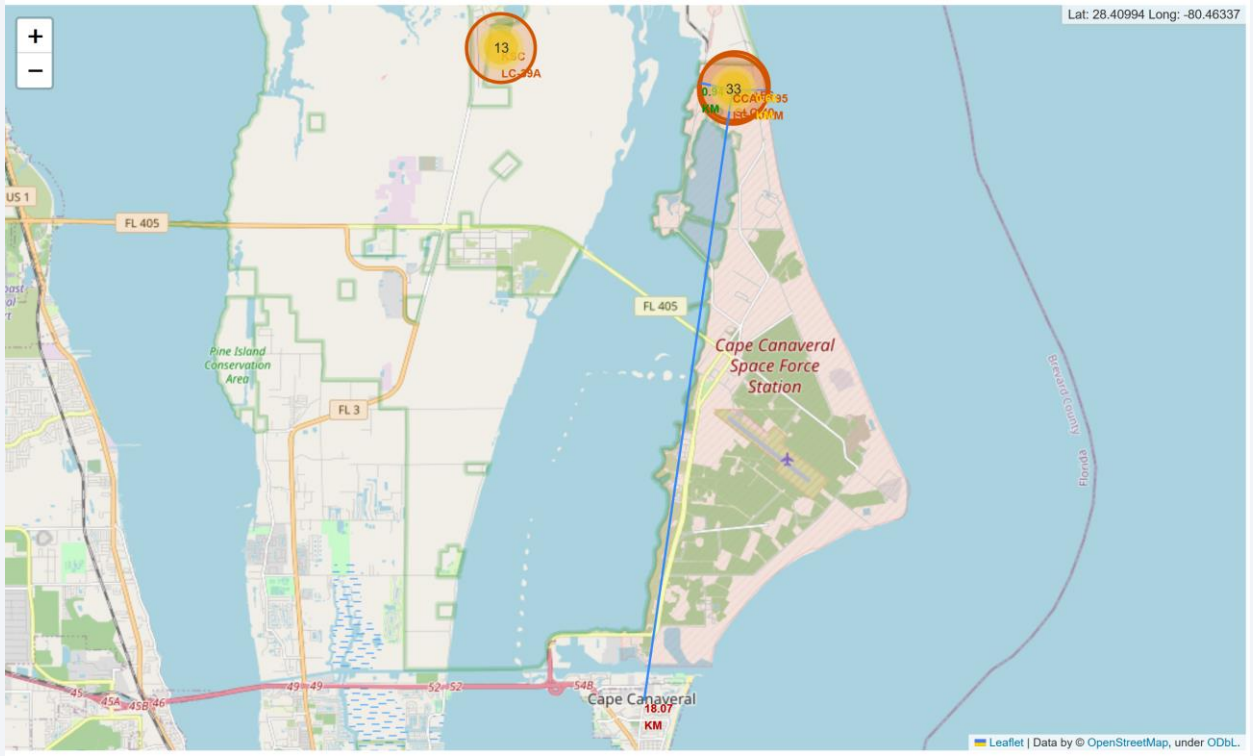
*https://en.wikipedia.org/wiki/Spaceport

Success/failed launches per each site

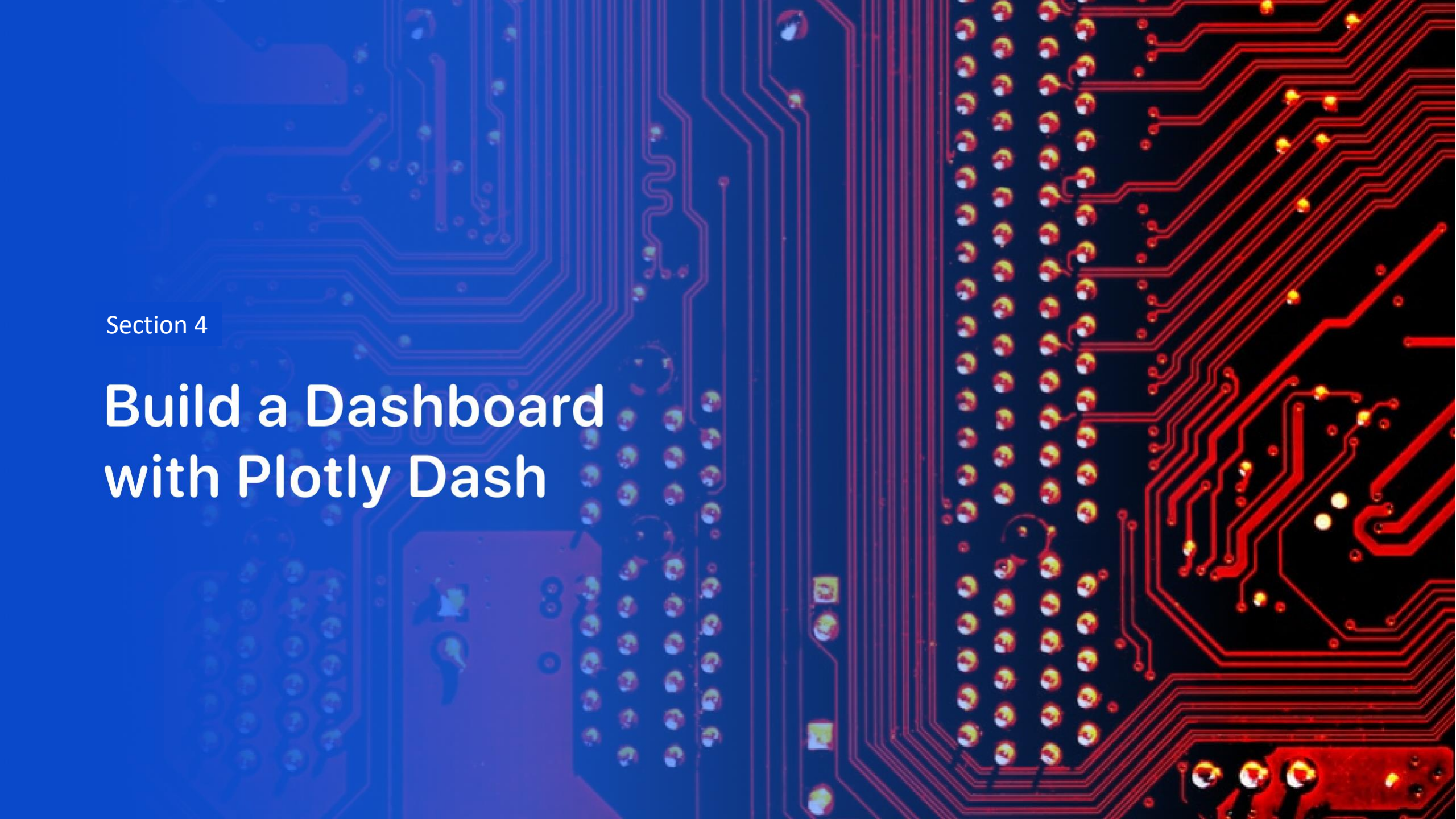
- Screen on the right zooms on the **KSC LC-39A** launch site, displaying the color-labeled markers in marker clusters that show success to fail ratio of all launches on this specific site
- With our map, we can zoom on the other launch sites to display their own statistics

Distances between a launch site to its proximities

- Finally, our map shows the distances between the launch site CCAFS LC-40 and its important proximities: city, railroad, highway and coastline using the folium.PolyLine object.
- Rocket sites are built as far as possible away from major population centers in order to mitigate risk to bystanders should a rocket experience a catastrophic failure. In many cases a launch site is built close to major bodies of water to ensure that no components are shed over populated areas. Typically a spaceport site is large enough that, should a vehicle explode, it will not endanger human lives or adjacent launch pads.
- Planned sites of spaceports for sub-orbital spaceflight often make use of existing ground infrastructure, including runways.*

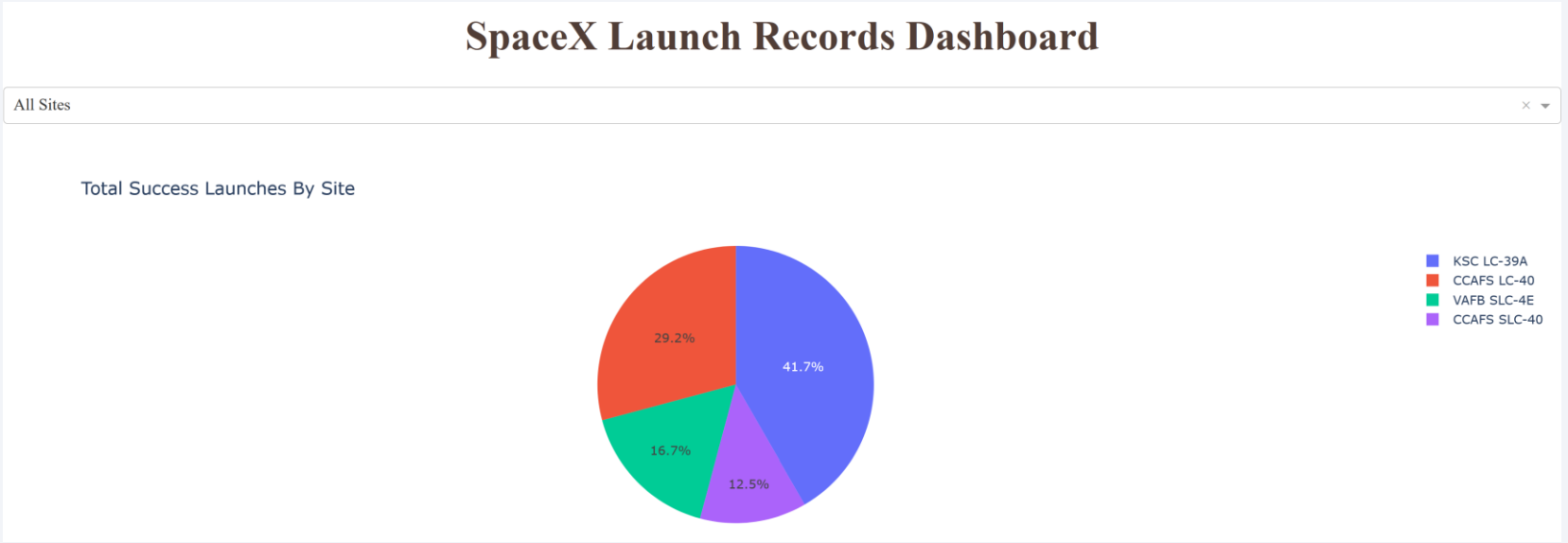


*https://en.wikipedia.org/wiki/Spaceport


Section 4
Build a Dashboard with Plotly Dash

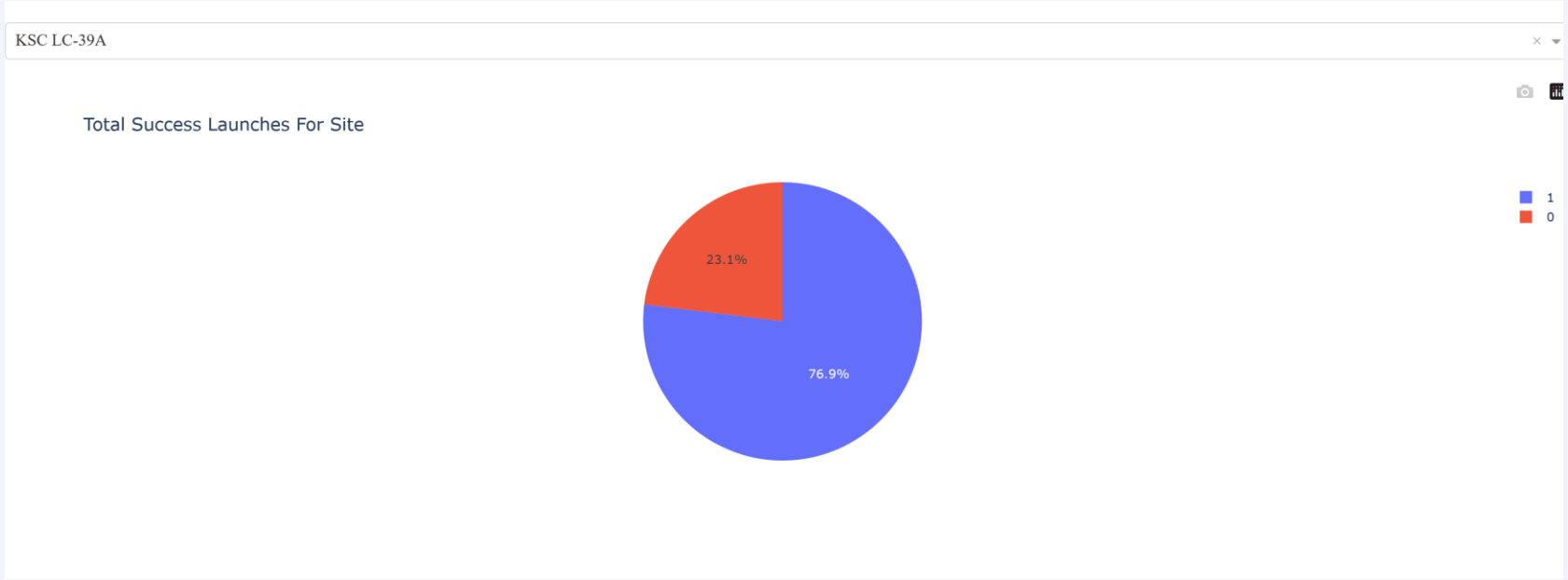
Dashboard: Total Success Launches For Site Chart

- On the piechart we can see Total Success Launches for all sites
- The launch site which contributes the most to the mission's success is KSC LC-39A: 41.7%
- The lowest one is on CCAFS SLC-40: 12.5%



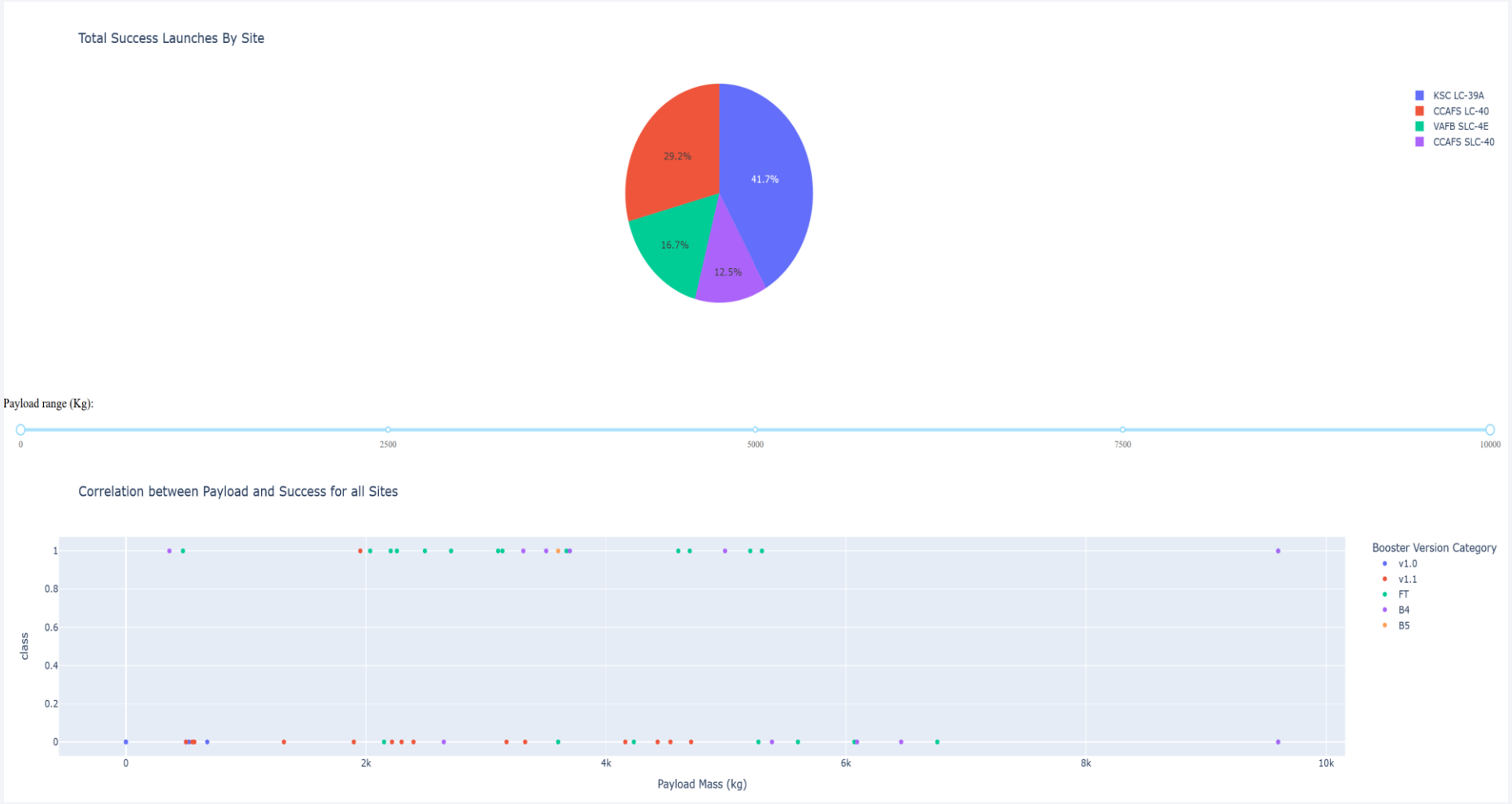
Dashboard: Total Success Launches For Site KSC LC-39A Chart

- On this chart we can see details regarding the success launches for Site KSC LC-39A
- 76.9% of the missions ended in success, while 23.1% failed.



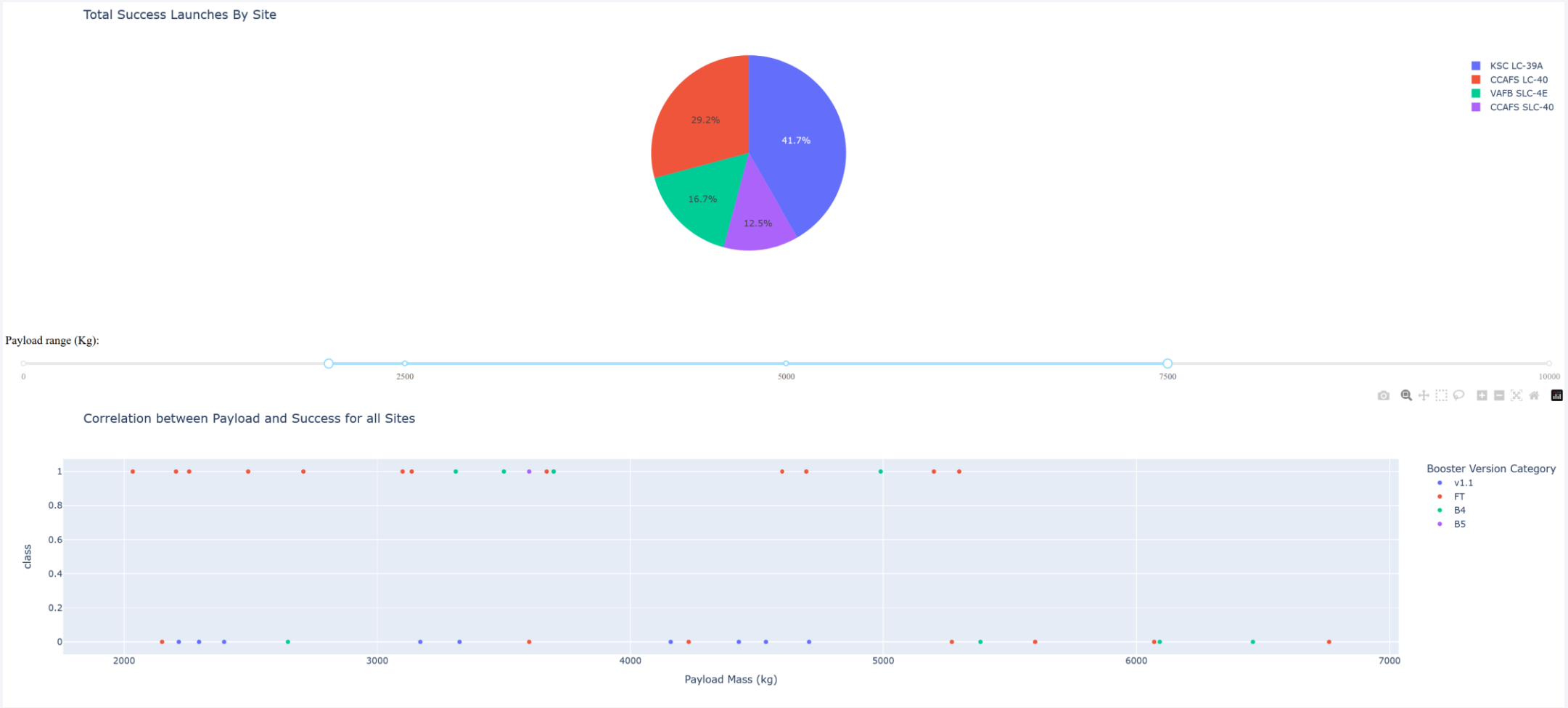
Dashboard: Payload vs. Launch Outcome scatter plot for all sites

- The scatter plot below the Pie Chart shows correlation between Payload (which range can be selected using slider) and Success for all Sites taking Booster Version Category into account.
- In the selected range we can see that booster B4 is used for the heaviest payload
- FT booster seems to be the most effective
- The highest success rate is for payload between 2k – 6k



Dashboard: Payload vs. Launch Outcome scatter plot for all sites

- Below we can see the scatter plot with the different Payload range selected

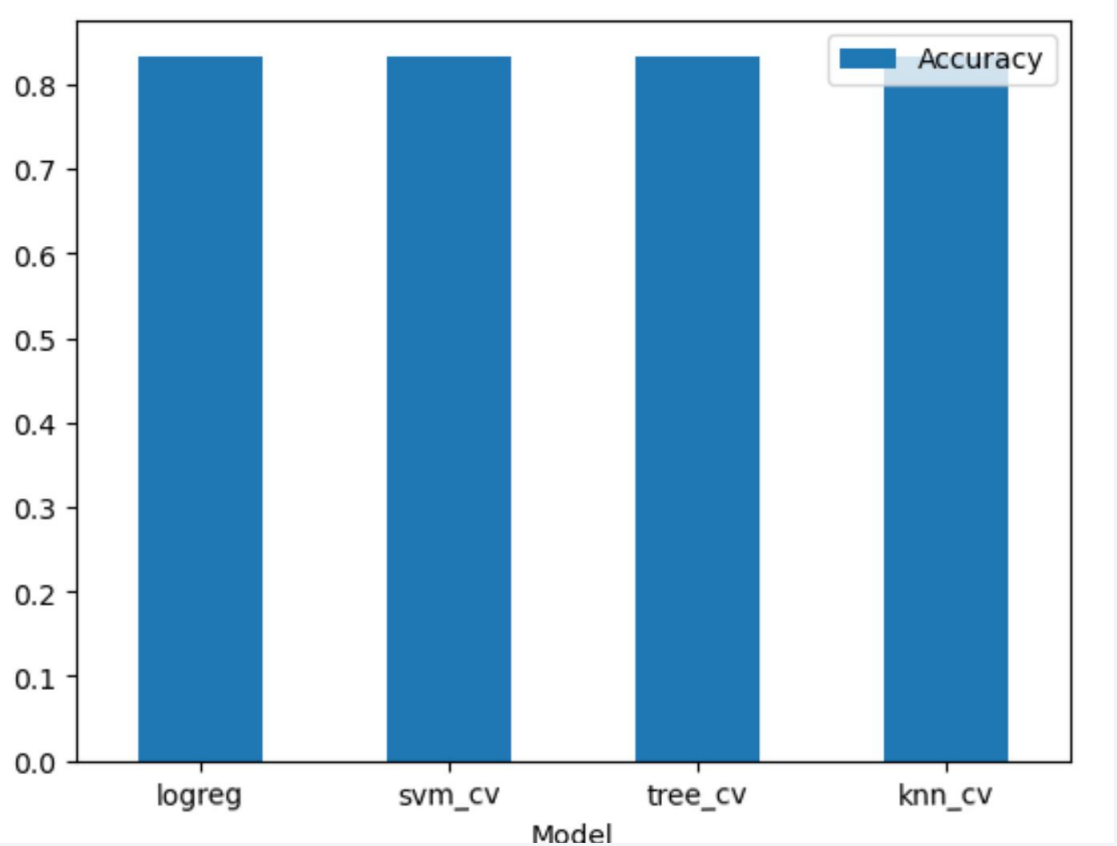


Section 5

Predictive Analysis (Classification)

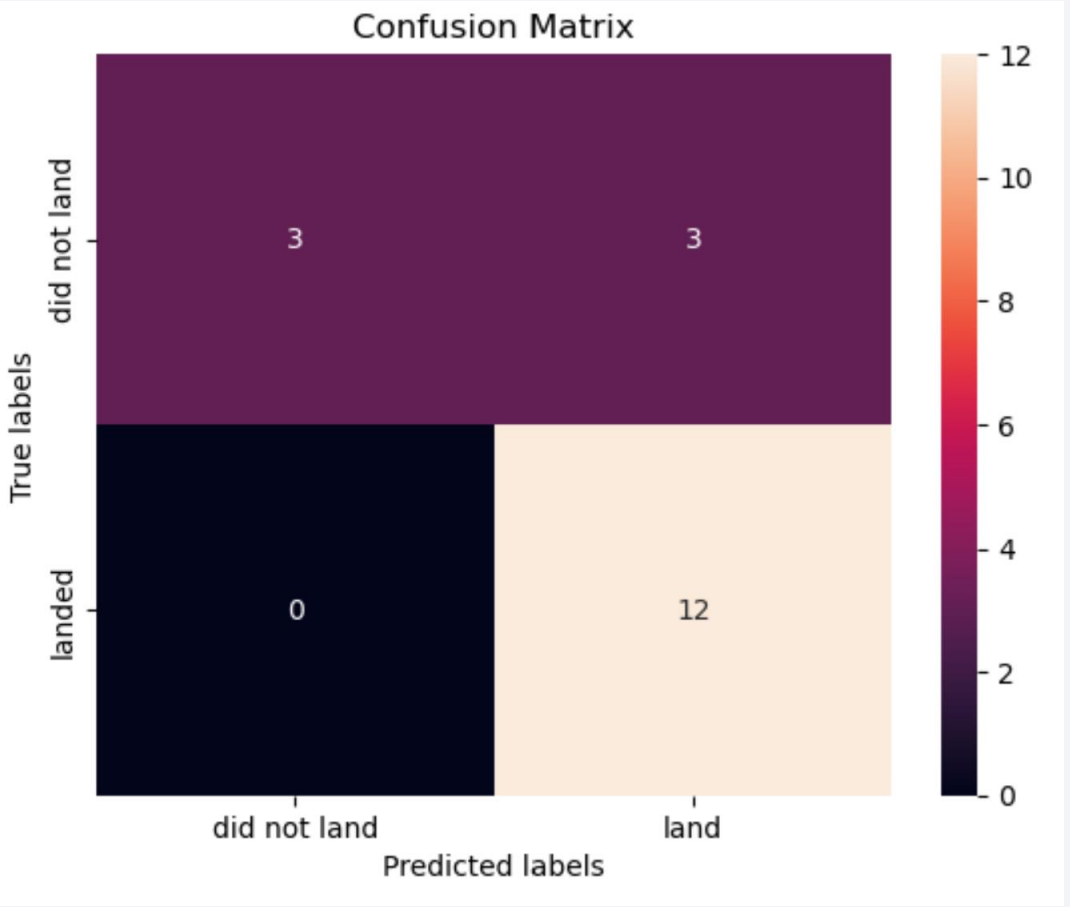
Classification Accuracy

- On the right side we see a bar chart with the built model accuracy for all built classification models
- As visible, all models have basically the same accuracy score, which is approximately: 8.33



Confusion Matrix

- Confusion Matrix for all 4 models are exactly the same, on the right we display the matrix for **k nearest neighbors** as an illustration
- We can see that the model correctly predicted15 successful landings, from which 12 are correct and 3 are false positive.
- We can also notice that according to the model only 3 missions failed, while in reality: 6 landings failed.



Conclusions

- Based on our research we can conclude that the success of the missions depends on many factors, the most important are Launch Site, Flight Number, Payload, and destination Orbit.
- We found out that that the Launch Site with the highest success rate is KSC LC-39A on Florida (76.9% of the missions ended in success, while 23.1% failed)
- We can observe that the mission success trend grows with years and experience accumulated by SPACEX company: as the flight number increases, the first stage is more likely to land successfully
- With heavy payloads the successful landing or positive landing rate are more for Polar, LEO and ISS
- On the “Relationship between success rate of each orbit type” plot, we can observe that launches for orbits ES-L1, GEO, HEO and SSO have 100% success rate
- We built 4 prediction models: tree classifier, logistic regression, support vector machine, k nearest neighbors
- All of them produced the same results and have the same accuracy score, therefore the model we ultimately choose depends on our preferences
- If we would like to bid against SpaceX for a rocket launch, we should take all of the above findings into account, have big budget to cover all the infrastructural, RD, materials, work costs and posses the necessary persistence to withstand the long process that leads to the successful spaceflight.

Appendix

- All relevant assets like Python code snippets, SQL queries, charts, Notebook outputs can be found in GitHub repository at:

- https://github.com/Rektorian/stairstostarts

Thank you!