IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Surya Prakash Murugan

14 JULY 2023

Outline

- Executive Summary
- Introduction
- Methodology
- Results
- Conclusion

Executive Summary

- Summary of methodologies

 - SpaceX Data Collection using SpaceX API

 - SpaceX Data Collection with Web Scraping

 - SpaceX Data Wrangling

 - SpaceX Exploratory Data Analysis using SQL

 - Space-X EDA DataViz Using Python Pandas and Matplotlib

 - Space-X Launch Sites Analysis with Folium-Interactive Visual Analytics and Ploty Dash

 - SpaceX Machine Learning Landing Prediction

- Summary of all results

 - EDA results

 - Interactive Visual Analytics and Dashboards

 - Predictive Analysis(Classification)

Introduction

Context

SpaceX promotes Falcon 9 rocket launches for 62 million dollars; other suppliers charge upwards of 165 million dollars for each launch. A large portion of the savings is due to SpaceX's ability to reuse the first stage. Therefore, if we can figure out if the first stage will land, we can figure out how much a launch will cost. If another business wishes to submit a bid for a rocket launch against SpaceX, it may do so using this information.

Objective:

Using information from Falcon 9 rocket launches that were publicized on the company website, we will forecast whether the Falcon 9 first stage will successfully land in this location.

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - Describe how data was collected
- 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

Data was initially gathered by sending a get request to the SpaceX API, a RESTful API, using the SpaceX API. In order to accomplish this, a series of helper functions that would aid in the use of the API to extract information utilizing identifying numbers in the launch data were first defined. Following that, the SpaceX API url was requested to obtain rocket launch data.

The SpaceX launch data was finally requested and parsed via the GET request, and the return content was then decoded as a JSON result before being turned into a Pandas data frame. This was done to improve the consistency of the requested JSON results.

Additionally, web scraping was done to get past Falcon 9 launch data from a Wikipedia page named "List of Falcon 9 and Falcon Heavy launches," where the data is saved in HTML. I parsed the table and transformed it into a Pandas data frame using BeautifulSoup and request Libraries to extract the Falcon 9 launch HTML table records from the Wikipedia page.

Data Collection – SpaceX API

The SpaceX API (a RESTful API) was used to collect the data. A GET request was made to the SpaceX API, which was used to request and parse the SpaceX launch data. The response content was decoded as a JSON result, and this was then turned into a Pandas data frame.

Github link for the SpaceX API call Notebook

https://github.com/suryaprakash-09/AppliedDatascienceCapstoneproject/blob/a6e6998b36bb0a8a54d4428cee14bd128e2050d5/1.SPACEX-data-collection-api.ipynb

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

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

In [17]:
```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_
```

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

In [18]:
```
response.status_code
```

Out[18]: 200

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

In [19]:
```
# Use json_normalize meethod to convert the json result into a dataframe
response.json()
data=pd.json_normalize(response.json())
```

Using the dataframe `data` print the first 5 rows

In [20]:
```
# Get the head of the dataframe
df=pd.read_json(static_json_url)
df.head(5)
```

Out[20]:

	fairings	links	static_fire_date_utc	static_fire_date_unix	tbd	net	window
0	{'reused': False, 'recovery_attempt': False, 'recovered': False, 'ships': []}	{'patch': {'small': 'https://images2.imgbox.com/3c/0e/T8iJcSN3_o.png', 'large': 'https://images2.imgbox.com/40/e3/GypSkayF_o.png'}, 'reddit': {'campaign': None, 'launch': None, 'media': None, 'recovery': None}, 'flickr': {'small': [], 'original': []}, 'presskit': None, 'webcast': 'https://www.youtube.com/watch?v=0a_00nJ_Y88', 'youtube_id': '0a_00nJ_Y88', 'article': 'https://www.space.com/2196-spacex-inaugural-falcon-1-rocket-lost-launch.html', 'wikipedia': 'https://en.wikipedia.org/wiki/DemoSat'}	2006-03-17T00:00:00.000Z	1.142554e+09	False	False	0.0

Data Collection - Scraping

Web scraping was used to gather historical Falcon 9 launch data from Wikipedia using BeautifulSoup and a request. The historical Falcon 9 launch data were then extracted from the Wikipedia page's HTML table and converted into a data frame utilizing launch HTML parsing.

GitHub URL of the completed web scraping notebook

https://github.com/suryaprakash-09/ApplieddatascienceCapstoneproject/blob/a6e6998b36bb0a8a54d4428cee14bd128e2050d5/2.SPACEX-webscraping.ipynb

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

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

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

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

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

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

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

Out[7]:
```
<title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

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

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

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

In [9]:
```
# Use the find_all function in the BeautifulSoup object, with element type `table`
# Assign the result to a list called `html_tables`
html_tables = soup.find_all('table')
```

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

In [10]:
```
# Let's print the third table and check its content
first_launch_table = html_tables[2]
print(first_launch_table)
```

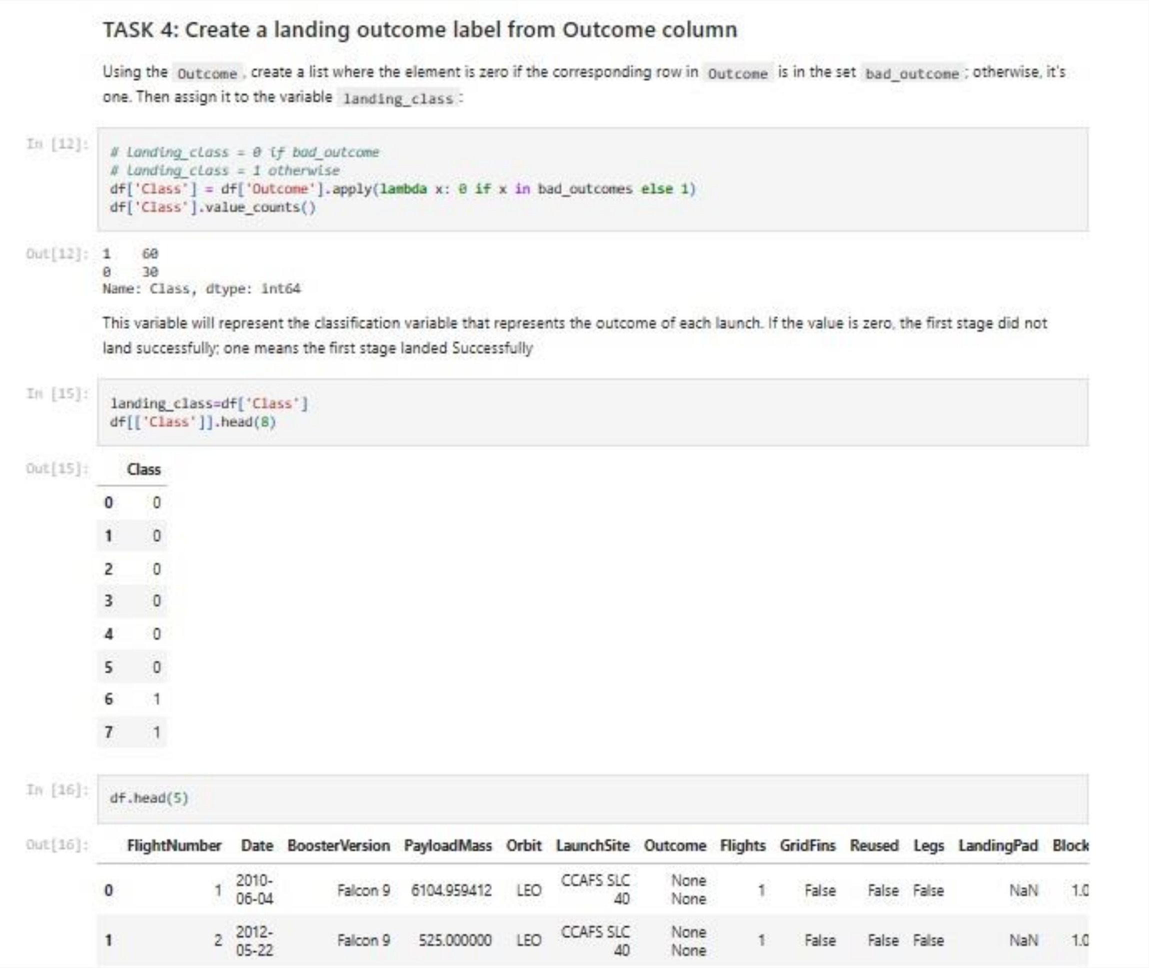
```
<table class="wikitable plainrowheaders collapsible" style="width: 100%;">
```

Data Wrangling

Data was filtered using the BoosterVersion column to only keep the Falcon 9 launches after acquiring and generating a Pandas DF from the gathered information. Next, the missing data values in the LandingPad and PayloadMass columns were handled with. Missing data values for the PayloadMass were replaced with the column's mean value.

GitHub URL of completed data wrangling related notebooks

https://github.com/suryaprakash-09/ApplieddatascienceCapstoneproject/blob/a6e6998b36bb0a8a54d4428cee14bd128e2050d5/3.SPACEX-data_wrangling.ipynb



TASK 4: Create a landing outcome label from Outcome column

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

In [12]:

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

Out[12]:

```
1    60
0    30
Name: Class, dtype: int64
```

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

In [15]:

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

Out[15]:

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

In [16]:

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

Out[16]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0

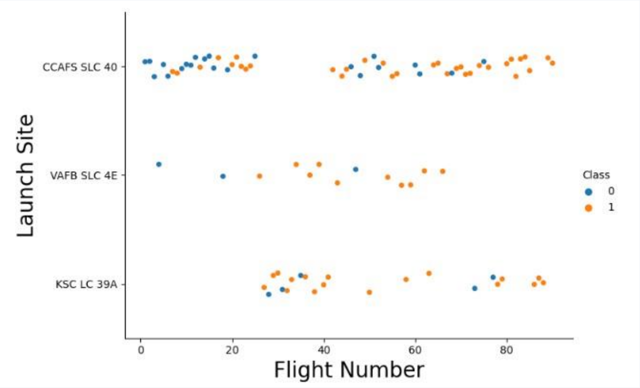
EDA with Data Visualization

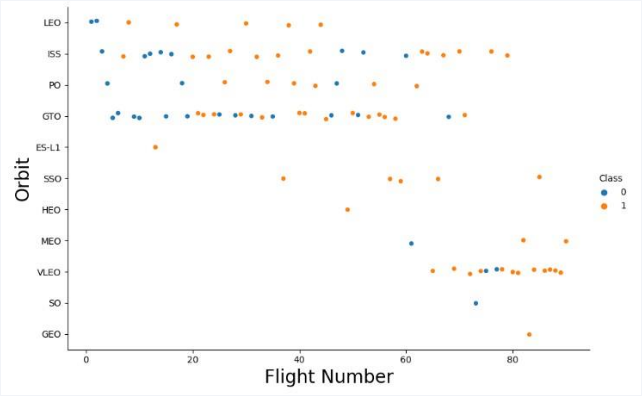
Performed data Analysis and Feature Engineering using Pandas and Matplotlib.

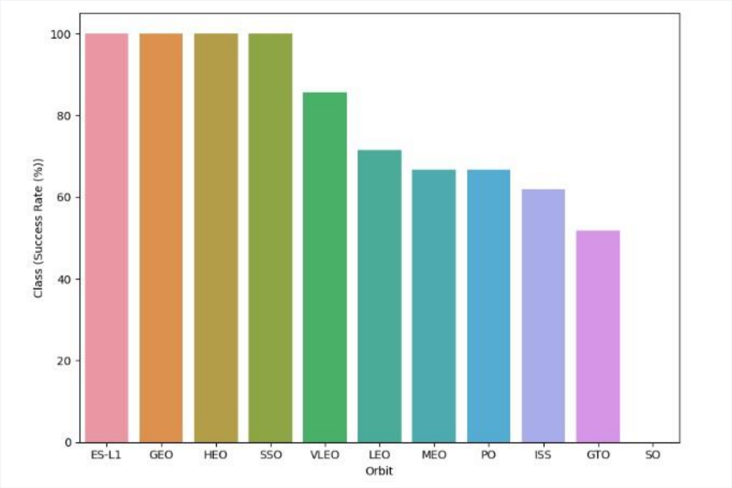
• Exploratory Data Analysis

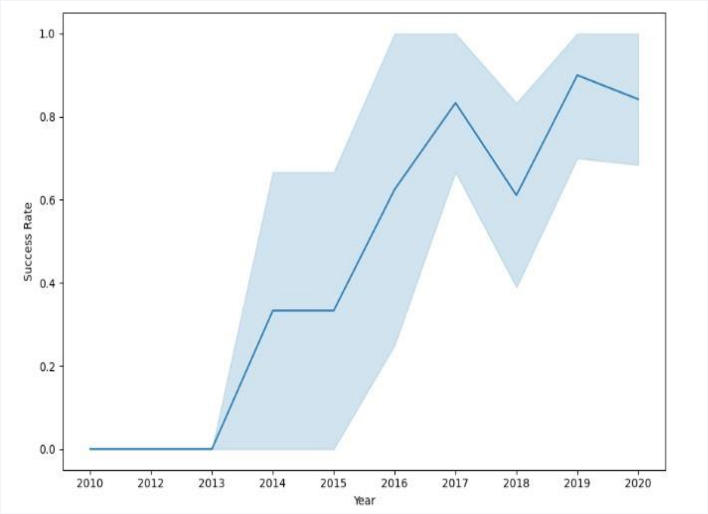
• Preparing Data Feature Engineering

Used scatter plots to Visualize the relationship between Flight Number and Launch Site, Payload and Launch Site, Flight Number and Orbit type, Payload and Orbit type. Used Bar chart to Visualize the relationship between success rate of each orbit type. Line plot to Visualize the launch success yearly trend.









GitHub URL of completed EDA with data visualization notebook

https://github.com/suryaprakash-09/ApplieddatascienceCapstoneproject/blob/a6e6998b36bb0a8a54d4428cee14bd128e2050d5/5.SPACEX-EDA-DataViz%20using%20pandas%20and%20matplotlib.ipynb

EDA with SQL

The following SQL queries were performed for EDA

- 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 EDA with SQL
- 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 (%sql SELECT DISTINCT Booster_Version, Payload 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 outcome

GitHub URL of your completed EDA with SQL notebook

https://github.com/suryaprakash-09/ApplieddatascienceCapstoneproject/blob/a6e6998b36bb0a8a54d4428cee14bd128e2050d5/4.SPACEX-EDA%20using%20SQL.ipynb

Build an Interactive Map with Folium

- Developed a folium map to indicate each launch site, and created map items like markers, circles, and lines to show each launch site's success or failure with launches.
- 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.
- In the previous exploratory data analysis labs, you have visualized the SpaceX launch dataset using matplotlib and seaborn and discovered some preliminary correlations between the launch site and success rates. In this lab, you will be performing more interactive visual analytics using Folium.

GitHub URL of your completed interactive map with Folium map

https://github.com/suryaprakash-09/ApplieddatascienceCapstoneproject/blob/a6e6998b36bb0a8a54d4428cee14bd128e2050d5/6.SPACEX-Launch%20site%20location%20using%20Folium.ipynb

Build a Dashboard with Plotly Dash

Built an interactive dashboard application with Plotly dash by:

- Adding a Launch Site Drop-down Input Component.
- Adding a callback function to render success-pie-chart based on selected site dropdown.
- Adding a Range Slider to Select Payload.
- Adding a callback function to render the success-payload-scatter-chart scatter plot.

GitHub URL of your completed Plotly Dash lab.

https://github.com/suryaprakash-09/ApplieddatascienceCapstoneproject/blob/a6e6998b36bb0a8a54d4428cee14bd128e2050d5/7.SPACEX-DASH%20with%20Plotly%20dash.py

Predictive Analysis (Classification)

In order to undertake exploratory data analysis and identify the training labels, loaded the data as a Pandas Data frame.

• Created a NumPy array from the column Class in the data using the method to numpy(), and I then assigned it to the variable Y as the outcome variable.

• After that, the feature dataset (x) was transformed using preprocessing to achieve standardization Sklearn's StandardScaler() function.

• Then, using the function train_test_split from sklearn.model_selection and setting the test_size and random_state parameters to 0.2 and 2, the data were divided into training and testing sets.

GitHub URL of your completed predictive analysis lab

https://github.com/suryaprakash-09/ApplieddatascienceCapstoneproject/blob/a6e6998b36bb0a8a54d4428cee14bd128e2050d5/8.SPACEX-Machine%20Learning%20Prediction.ipynb

TASK 1

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

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

Out[8]:
```
dtype('int64')
```

TASK 2

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

In [9]:
```
# students get this
transform = preprocessing.StandardScaler()
X = transform.fit_transform(X)
X
```

Out[9]:
```
array([[-1.71291154e+00, -1.94814463e-16, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       [-1.67441914e+00, -1.19523159e+00, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       [-1.63592675e+00, -1.16267307e+00, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       ...,
       [ 1.63592675e+00,  1.99100483e+00,  3.49060516e+00, ...,
         1.19684269e+00, -5.17306132e-01,  5.17306132e-01],
       [ 1.67441914e+00,  1.99100483e+00,  1.00389436e+00, ...,
         1.19684269e+00, -5.17306132e-01,  5.17306132e-01],
       [ 1.71291154e+00, -5.19213966e-01, -6.53912840e-01, ...,
        -8.35531692e-01, -5.17306132e-01,  5.17306132e-01]])
```

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

TASK 3

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

`X_train, X_test, Y_train, Y_test`

In [10]:
```
X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size=0.2, random_state=2)
```

Predictive Analysis (Classification)

SVM, Classification Trees, k Nearest Neighbors, and Logistic Regression were tested against test data in order to determine which machine learning model or method performed the best.

- After making an object for each algorithm, construct a GridSearchCV object and give it a set of model-specific parameters.
- The GridsearchCV object was made for each of the models being evaluated with cv=10, and the training data for each model was then fitted into the GridSearch object to find the optimum hyperparameter.
- After each model was output as a GridSearchCV object after being fitted to the training set, the best parameters and accuracy on the validation data were reported using the data attributes best_params_ and best_score_, respectively.
- The accuracy on the test data for each model was then calculated using the method score, and a confussion matrix was generated for each using the test and projected outcomes.

TASK 3

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

`X_train, X_test, Y_train, Y_test`

In [10]:
```
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 [11]:
```
Y_test.shape
```

Out[11]:
```
(18,)
```

TASK 4

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

In [12]:
```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

In [13]:
```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()
# Create a GridSearchCV object logreg_cv
logreg_cv = GridSearchCV(lr, parameters, cv=10)

#Fit the training data into the GridSearch object
logreg_cv.fit(X_train, Y_train)
```

Out[13]:
```
GridSearchCV(cv=10, estimator=LogisticRegression(),
             param_grid={'C': [0.01, 0.1, 1], 'penalty': ['l2'],
                         'solver': ['lbfgs']})
```

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

In [14]:
```
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
```

Results

The table below compares the test data accuracy scores of SVM, Classification Trees, k nearest neighbors, and Logistic Regression to indicate which approach outperformed the others using the test data.

Out[33]:

	0
Method	Test Data Accuracy
Logistic_Reg	0.833333
SVM	0.833333
Decision Tree	0.833333
KNN	0.833333

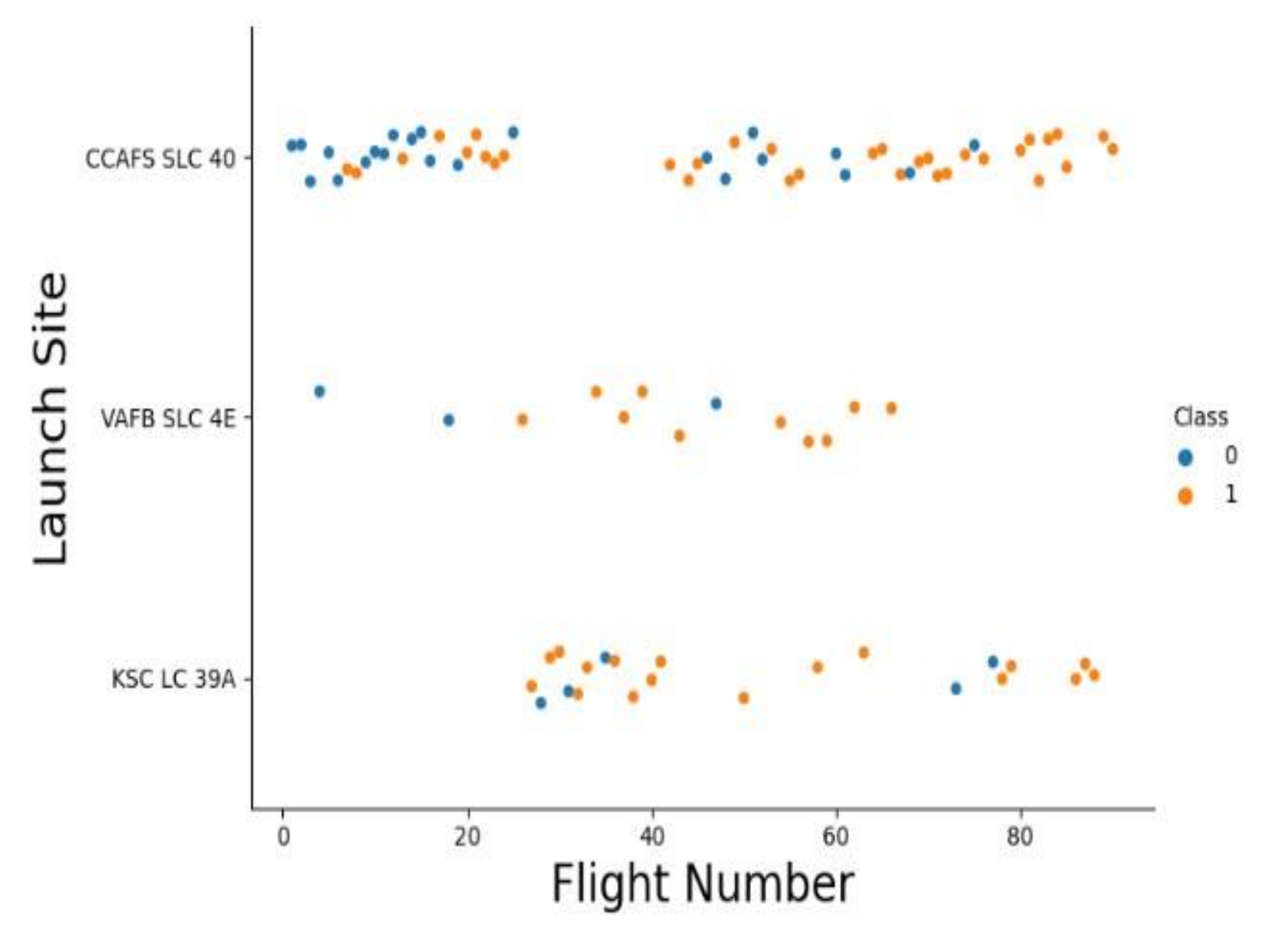
All the method have same accuracy 0.833 on test data. So, all the method perform equally on test data.

Section 2

Insights drawn from EDA

Flight Number vs. Launch Site

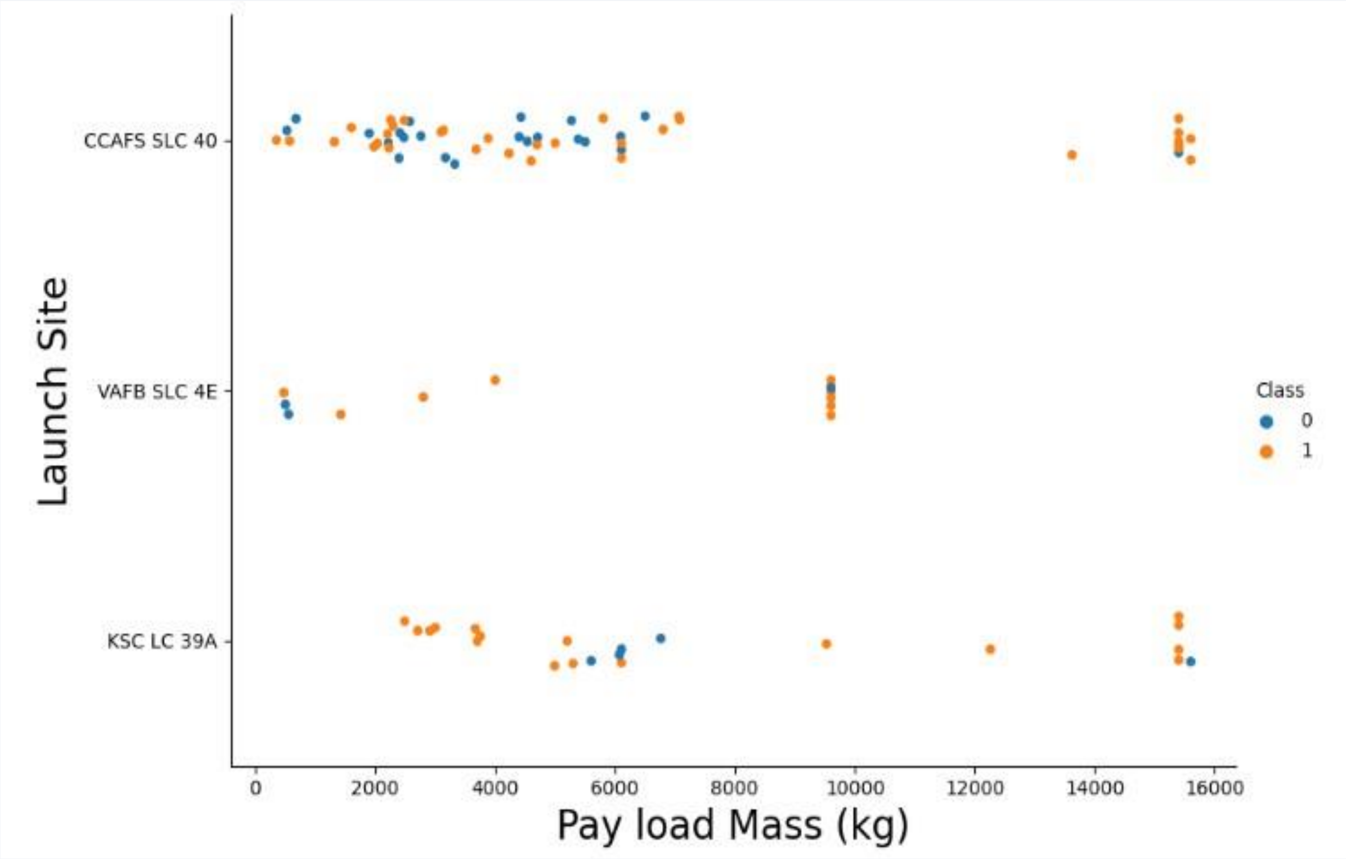
Scatter plot of Flight Number vs. Launch Site



Payload vs. Launch Site

Scatter plot of Payload vs. Launch Site

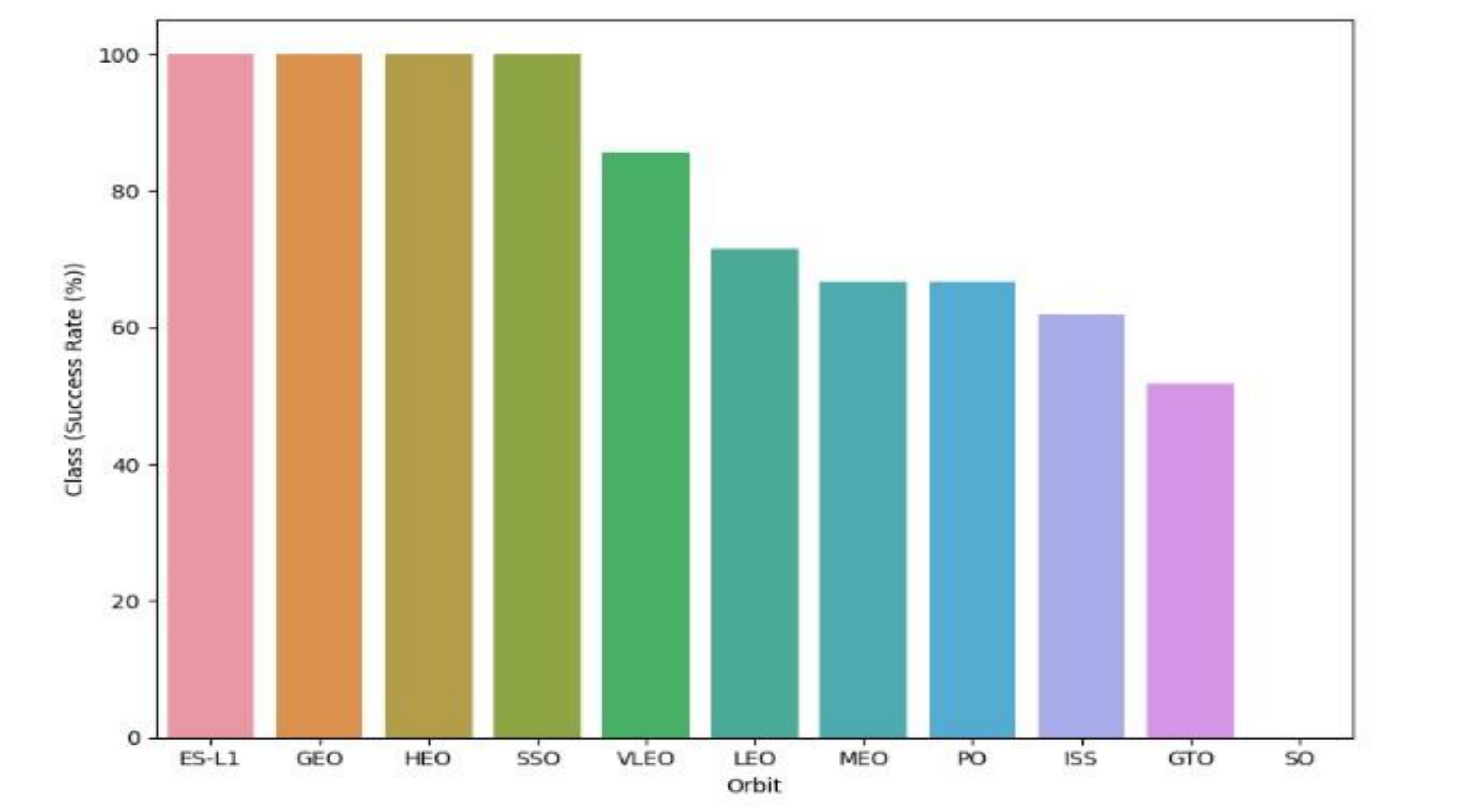
Now if you observe Payload Vs. Launch Site scatter point chart you will find for the VAFB-SLC launchsite there are no rockets launched for heavypayload mass(greater than 10000).



Success Rate vs. Orbit Type

Success rate of each orbit type

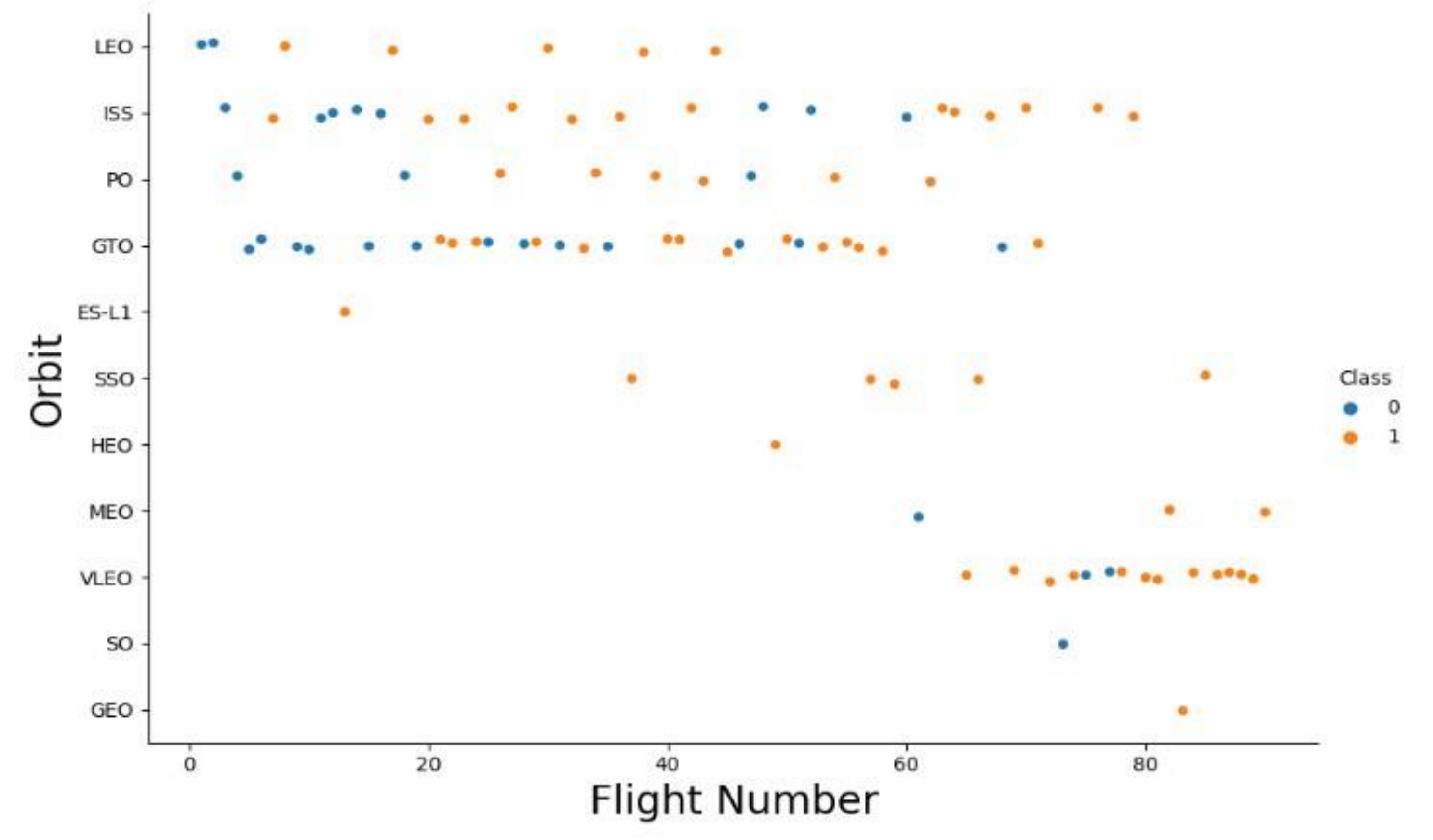
Analyzing the plotted bar chart ES-L1, GEO, HEO, SSO orbits have High Success Rate



Flight Number vs. Orbit Type

Scatter plot of Flight number vs. Orbit type

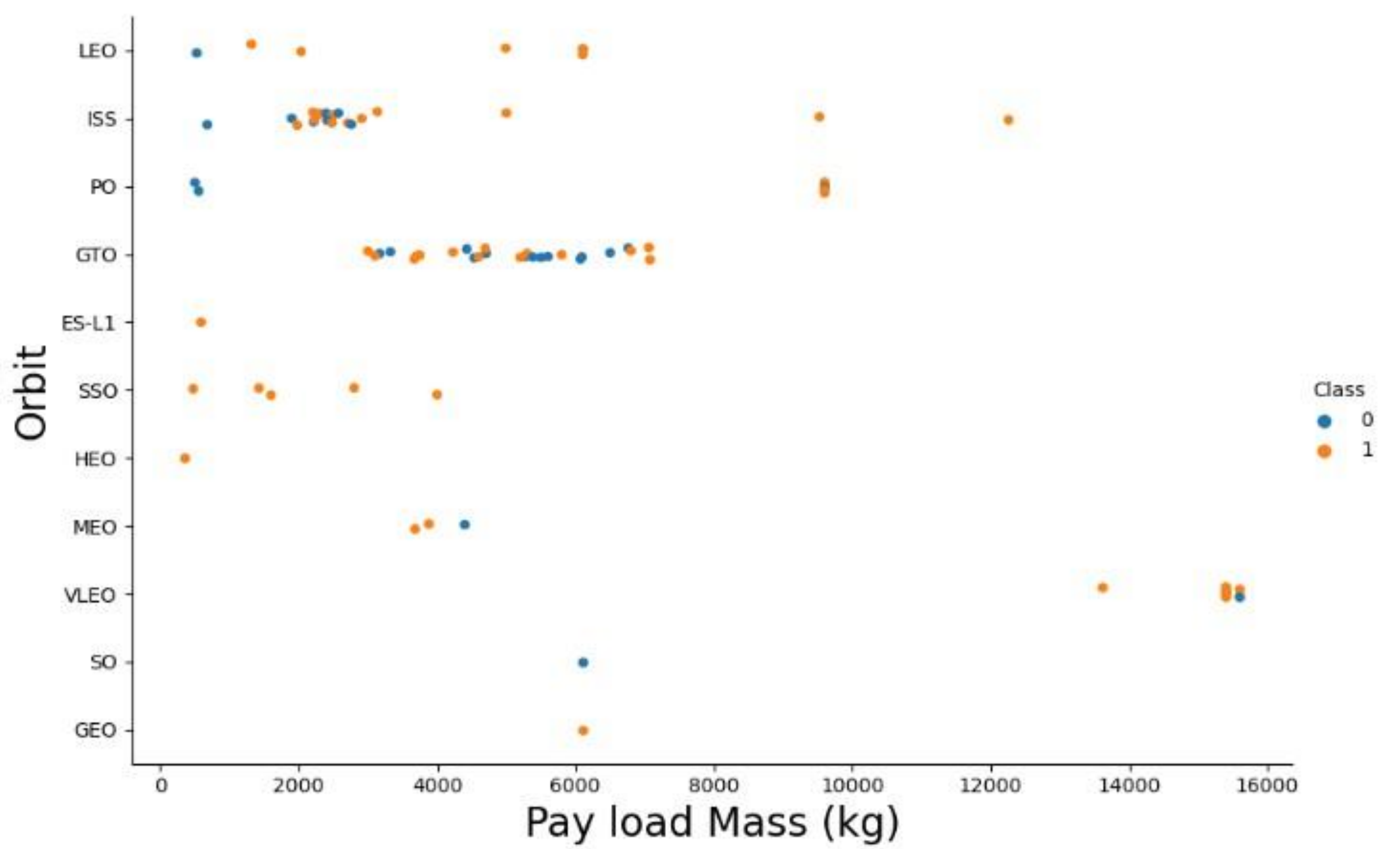
The LEO orbit the Success appears related to the number of flights, on the other hand, there seems to be no relationship between flight number when in GTO orbit.



Payload vs. Orbit Type

Scatter polt of payload vs. orbit type

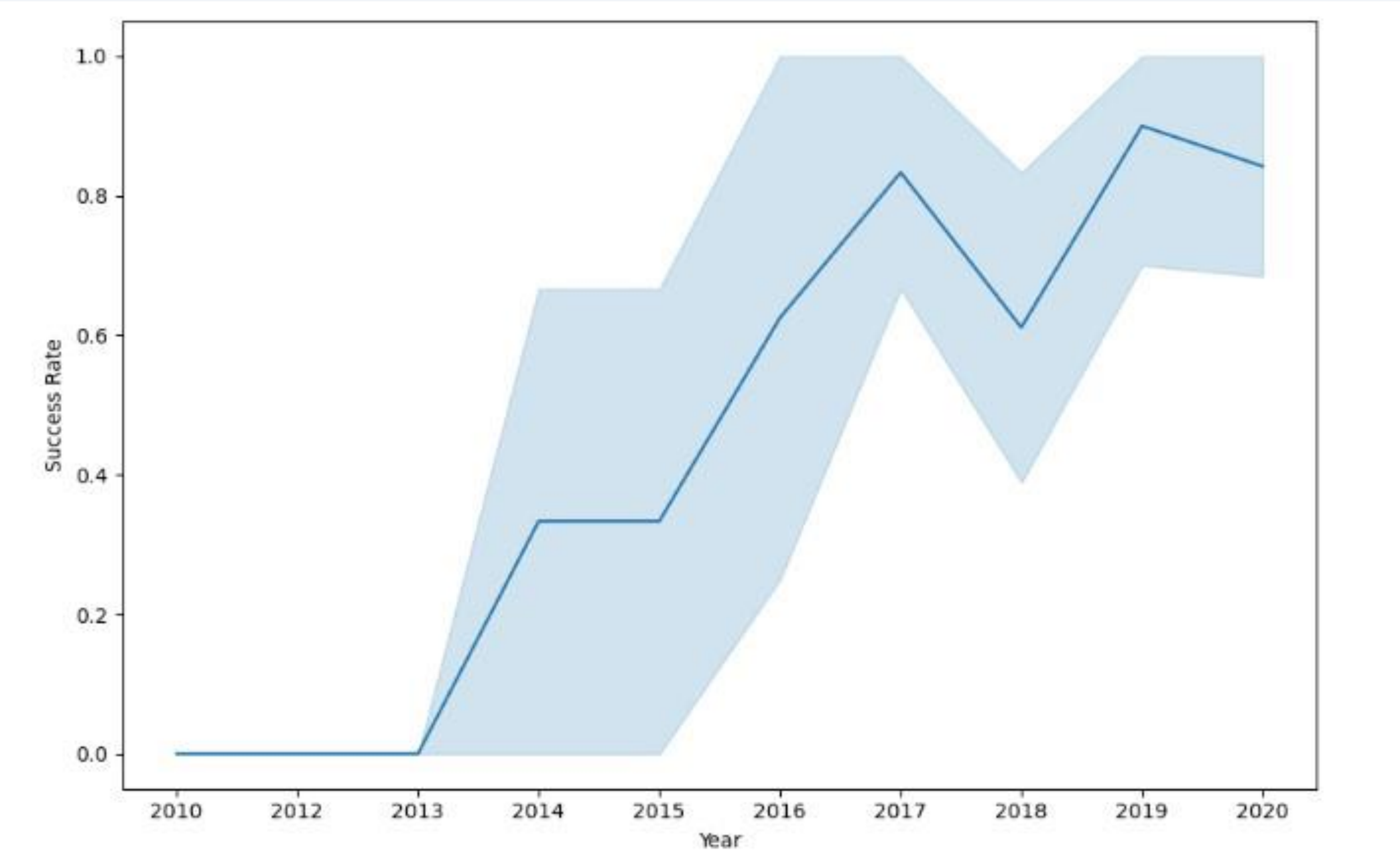
With heavy payloads the successful landing or positive landing rate are more for Polar, LEO and ISS. However for GTO we cannot distinguish this well as both positive landing rate and negative landing(unsuccessful mission) are here.



Launch Success Yearly Trend

Line chart of yearly average success rate

It is observed that the success rate since 2013 kept increasing till 2020



All Launch Site Names

Used ‘SELECT DISTINCT’ statement to return only the unique launch sites from the ‘LAUNCH_SITE’ column of the SPACEXTBL table

Task 1

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

In [7]:
```
%sql SELECT DISTINCT LAUNCH_SITE as "Launch_Sites" FROM SPACEXTBL;
```

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

Out[7]:

Launch_Sites
CCAFS LC-40
VAFB SLC-4E
KSC LC-39A
CCAFS SLC-40
None

Launch Site Names Begin with 'CCA'

Used ‘LIKE’ command with ‘%’ wildcard in ‘WHERE’ clause to select and display a table of 5 records where launch sites begin with the string 'CCA'

Task 2

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

In [8]:
```
%sql SELECT * FROM 'SPACEXTBL' WHERE Launch_Site LIKE 'CCA%' LIMIT 5;
```

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

Out[8]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outc
06/04/2010	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0.0	LEO	SpaceX	Success	Failure (paracl
12/08/2010	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0.0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (paracl
22/05/2012	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525.0	LEO (ISS)	NASA (COTS)	Success	No att
10/08/2012	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500.0	LEO (ISS)	NASA (CRS)	Success	No att
03/01/2013	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677.0	LEO (ISS)	NASA (CRS)	Success	No att

Total Payload Mass

Used the 'SUM()' function to return and display the total sum of 'PAYLOAD_MASS_KG' column for Customer 'NASA(CRS)'

Task 3

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

In [9]:
```
%sql SELECT SUM(PAYLOAD_MASS__KG_) as "Total Payload Mass(Kgs)", Customer FROM 'SPACEXTBL' WHERE Customer = 'NASA (CRS)';
```

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

Out[9]:

Total Payload Mass(Kgs)	Customer
45596.0	NASA (CRS)

Average Payload Mass by F9 v1.1

Used the 'AVG()' function to return and display the average payload mass carried by booster version F9 v1.1

Task 4

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

In [11]:

```
%sql SELECT AVG(PAYLOAD_MASS__KG_) as "Payload Mass Kgs", Customer, Booster_Version FROM 'SPACEXTBL' WHERE Booster_Version
```

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

Out[11]:

Payload Mass Kgs	Customer	Booster_Version
2534.6666666666665	MDA	F9 v1.1 B1003

First Successful Ground Landing Date

Used the ‘MIN()’ function to return and display the first (oldest) date when first successful landing outcome on ground pad ‘Success (ground pad)’happened.

Task 5

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

Hint:Use min function

```
%sql SELECT MIN(DATE) FROM 'SPACEXTBL' WHERE "Landing _Outcome" = "Success (ground pad)";
```

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

MIN(DATE)
01-05-2017

Successful Drone Ship Landing with Payload between 4000 and 6000

Used ‘Select Distinct’ statement to return and list the ‘unique’ names of boosters with operators >4000 and <6000 to only list booster with payloads between 4000-6000 with landing outcome of ‘Success (drone ship)’

Task 6

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

```
# %sql SELECT * FROM 'SPACEXTBL'
```

```
%sql SELECT DISTINCT Booster_Version, Payload FROM SPACEXTBL WHERE "Landing _Outcome" = "Success (drone ship)" AND PAYLOAD_MASS__KG_ > 4000 AND PAYLOA
```

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

Booster_Version	Payload
F9 FT B1022	JCSAT-14
F9 FT B1026	JCSAT-16
F9 FT B1021.2	SES-10
F9 FT B1031.2	SES-11 / EchoStar 105

Total Number of Successful and Failure Mission Outcomes

Used the ‘COUNT()’ together with the ‘GROUP BY’ statement to return total number of missions outcomes

Task 7

List the total number of successful and failure mission outcomes

In [18]:

```
%sql SELECT "Mission_Outcome", COUNT("Mission_Outcome") as Total FROM SPACEXTBL GROUP BY "Mission_Outcome";
```

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

Out[18]:

Mission_Outcome	Total
None	0
Failure (in flight)	1
Success	98
Success	1
Success (payload status unclear)	1

Boosters Carried Maximum Payload

Used a Subquery to return and pass the Max payload and listed all the boosters that have carried the Max payload of 15600kgs

Task 8

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

In [19]:
```
%sql SELECT "Booster_Version",Payload, "PAYLOAD_MASS__KG_" FROM SPACEXTBL WHERE "PAYLOAD_MASS__KG_" = (SELECT MAX("PAYLOAD_
```

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

Out[19]:

Booster_Version	Payload	PAYLOAD_MASS__KG_
F9 B5 B1048.4	Starlink 1 v1.0, SpaceX CRS-19	15600.0
F9 B5 B1049.4	Starlink 2 v1.0, Crew Dragon in-flight abort test	15600.0
F9 B5 B1051.3	Starlink 3 v1.0, Starlink 4 v1.0	15600.0
F9 B5 B1056.4	Starlink 4 v1.0, SpaceX CRS-20	15600.0
F9 B5 B1048.5	Starlink 5 v1.0, Starlink 6 v1.0	15600.0
F9 B5 B1051.4	Starlink 6 v1.0, Crew Dragon Demo-2	15600.0
F9 B5 B1049.5	Starlink 7 v1.0, Starlink 8 v1.0	15600.0
F9 B5 B1060.2	Starlink 11 v1.0, Starlink 12 v1.0	15600.0
F9 B5 B1058.3	Starlink 12 v1.0, Starlink 13 v1.0	15600.0
F9 B5 B1051.6	Starlink 13 v1.0, Starlink 14 v1.0	15600.0
F9 B5 B1060.3	Starlink 14 v1.0, GPS III-04	15600.0
F9 B5 B1049.7	Starlink 15 v1.0, SpaceX CRS-21	15600.0

2015 Launch Records

Used the ‘subsrt()’ in the select statement to get the month and year from the date column where substr(Date,7,4)='2015' for year and Landing_outcome was ‘Failure (drone ship’) and return the records matching the filter

Task 9

List the records which will display the month names, failure landing_outcomes in drone ship ,booster versions, launch_site for the months in year 2015.

```
%sql SELECT substr(Date,7,4), substr(Date, 4, 2),"Booster_Version", "Launch_Site", Payload, "PAYLOAD_MASS__KG_", "Mission_Outcome", "Landing _Outcome"
```

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

substr(Date,7,4)	substr(Date, 4, 2)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Mission_Outcome	Landing _Outcome
2015	01	F9 v1.1 B1012	CCAFS LC-40	SpaceX CRS-5	2395	Success	Failure (drone ship)
2015	04	F9 v1.1 B1015	CCAFS LC-40	SpaceX CRS-6	1898	Success	Failure (drone ship)

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

Task 10

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

```
%sql SELECT * FROM SPACEXTBL WHERE "Landing _Outcome" LIKE 'Success%' AND (Date BETWEEN '04-06-2010' AND '20-03-2017') ORDER BY Date DESC;
```

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

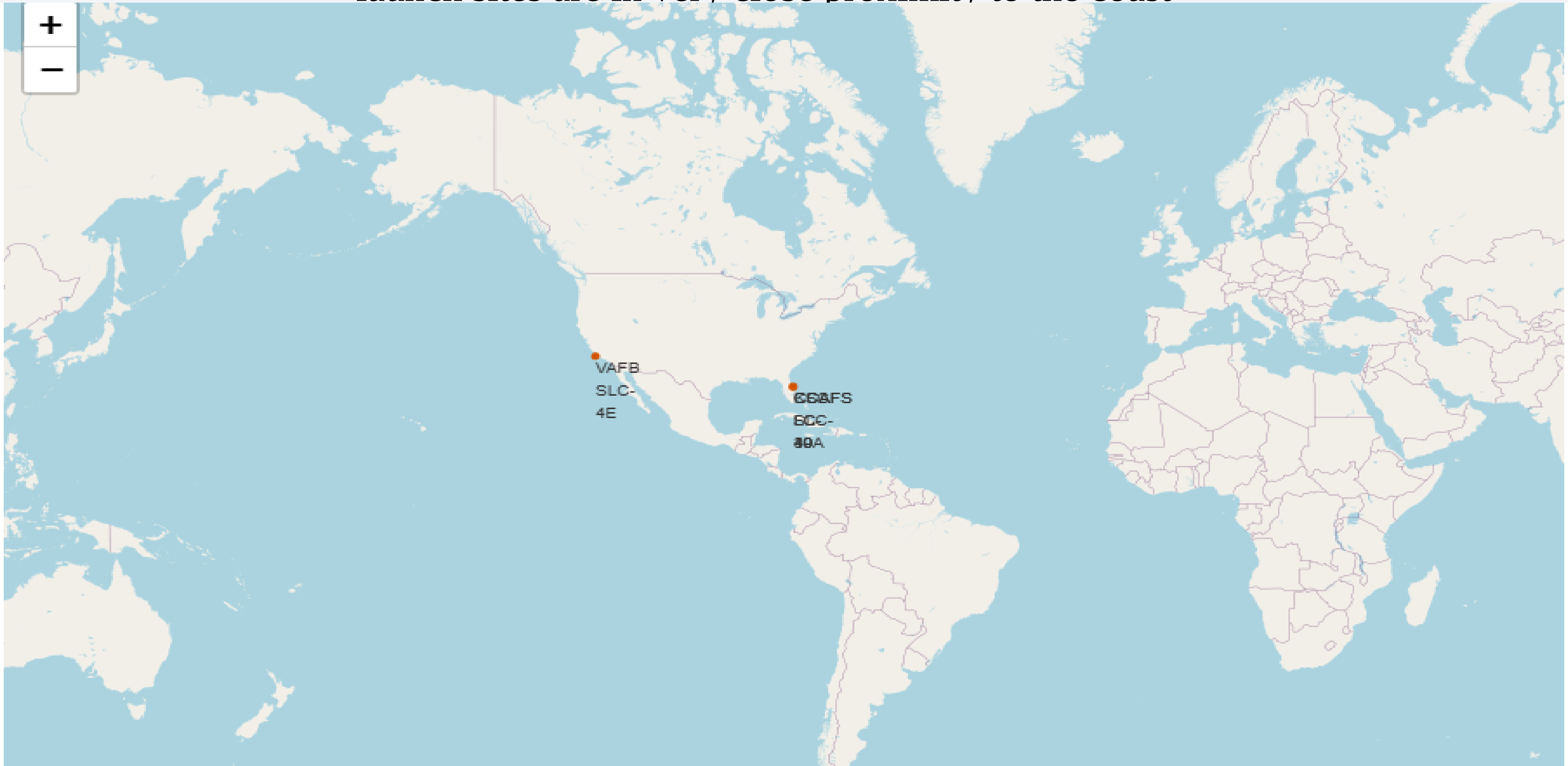
Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing _Outcome
19-02-2017	14:39:00	F9 FT B1031.1	KSC LC-39A	SpaceX CRS-10	2490	LEO (ISS)	NASA (CRS)	Success	Success (ground pad)
18-10-2020	12:25:57	F9 B5 B1051.6	KSC LC-39A	Starlink 13 v1.0, Starlink 14 v1.0	15600	LEO	SpaceX	Success	Success
18-08-2020	14:31:00	F9 B5 B1049.6	CCAFS SLC-40	Starlink 10 v1.0, SkySat-19, -20, -21, SAOCOM 1B	15440	LEO	SpaceX, Planet Labs, PlanetIQ	Success	Success
18-07-2016	04:45:00	F9 FT B1025.1	CCAFS LC-40	SpaceX CRS-9	2257	LEO (ISS)	NASA (CRS)	Success	Success (ground pad)
18-04-2018	22:51:00	F9 B4 B1045.1	CCAFS SLC-40	Transiting Exoplanet Survey Satellite (TESS)	362	HEO	NASA (LSP)	Success	Success (drone ship)

Section 3

Launch Sites Proximities Analysis

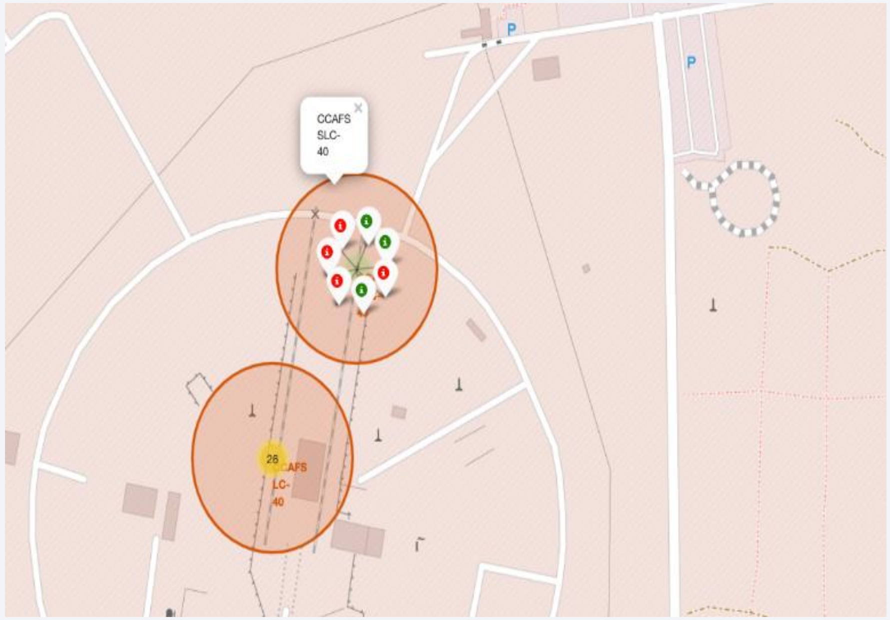
All Launch site Markers on Global map

All launch sites are in proximity to the Equator, (located southwards of the US map). Also all the launch sites are in very close proximity to the coast



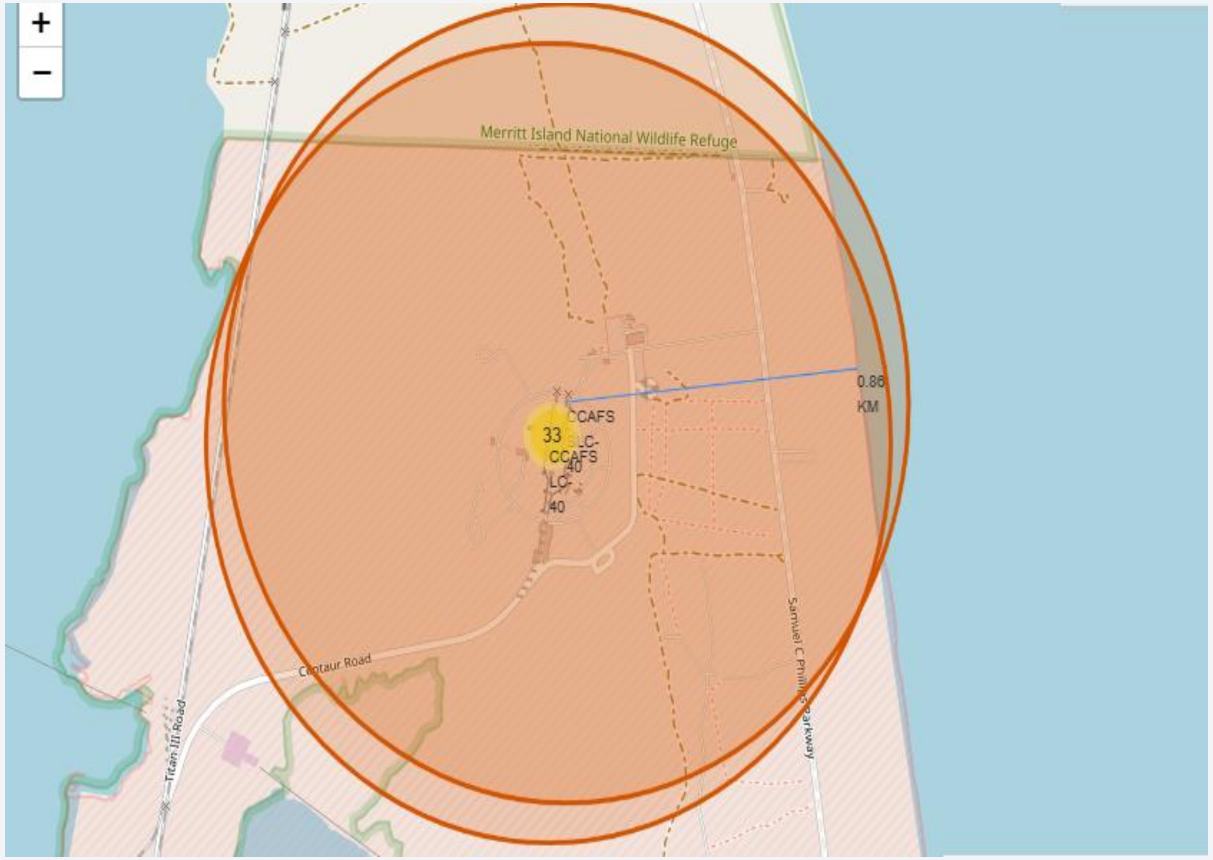
Launch outcomes of each site with color markers on map

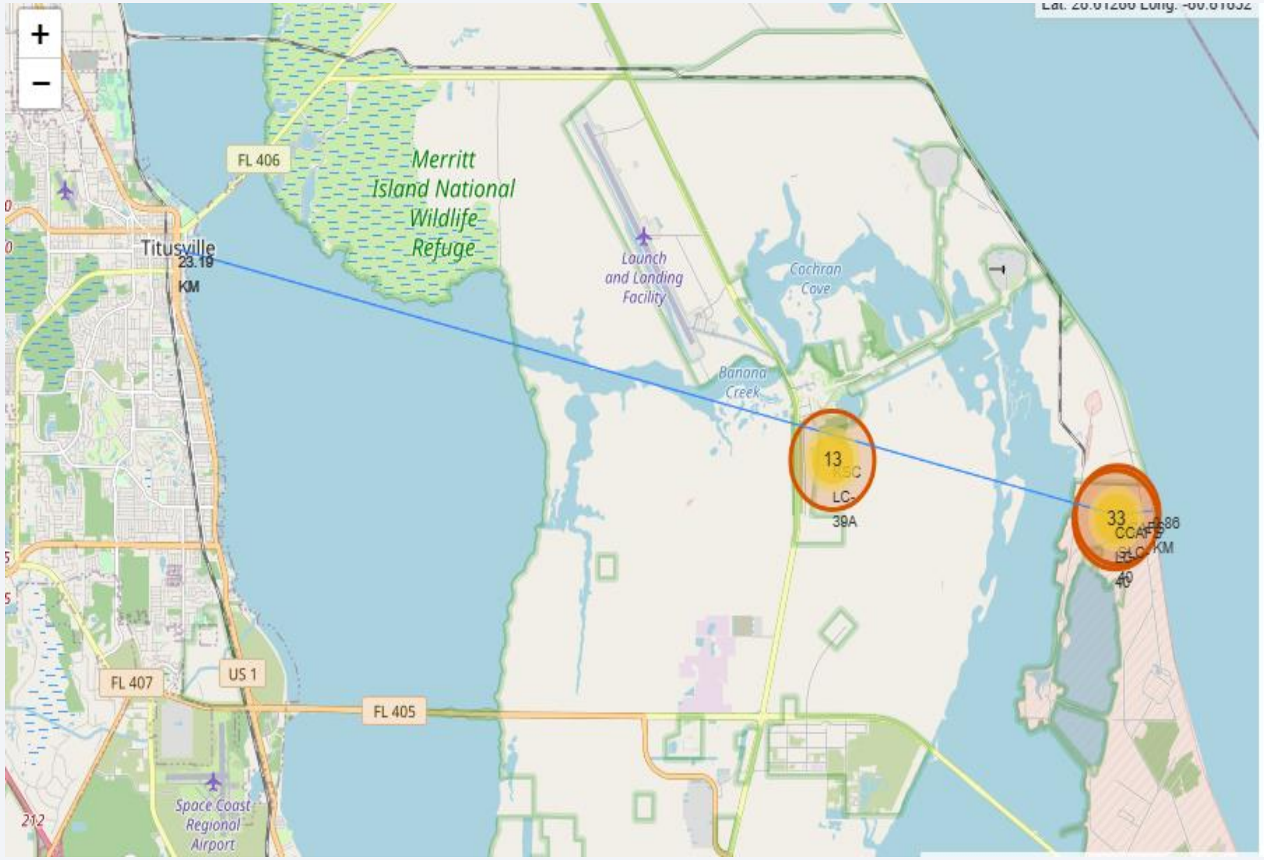
Compared to CCAFS SLC-40 & CCAFS LC-40, the Eastern Coast (Florida) Launch Site KSC LC-39A has relatively high success rates.



Distance between launch sites to its proximities

Launch site CCAFS SLC-40 proximity to coastline is 0.86km and Launch site CCAFS SLC-40 closest to highway (Washington Avenue) is 23.19km



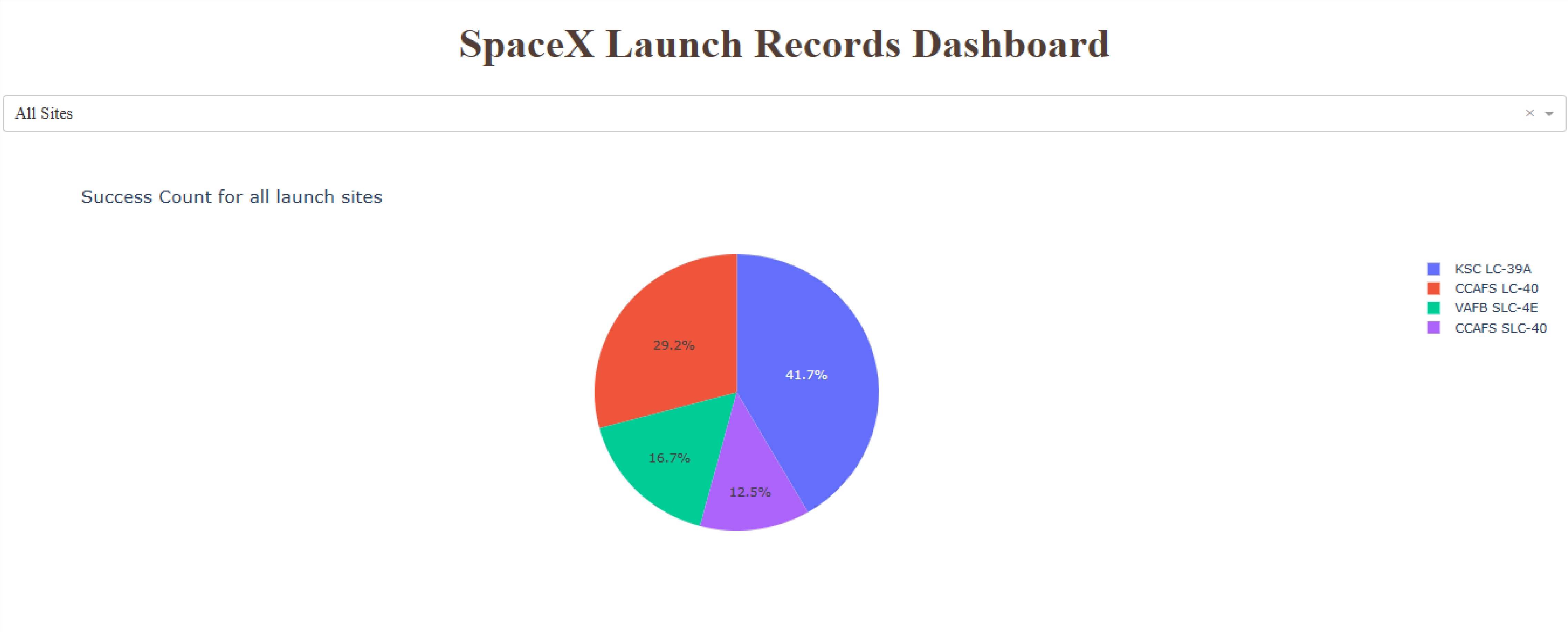


Section 4

Build a Dashboard with Plotly Dash

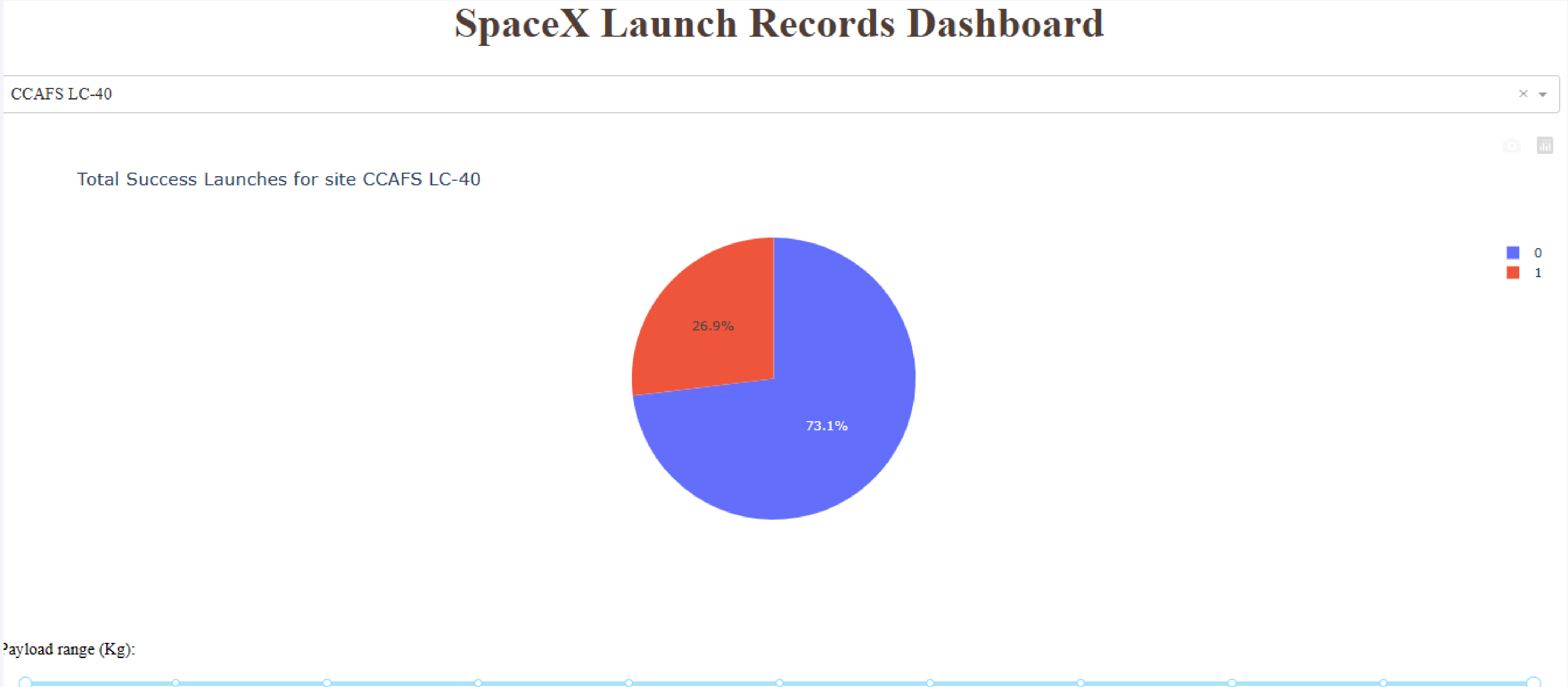
Pie chart for launch success count - all sites

With a success rate of 41.7%, launch site KSC LC-39A leads the field, followed by CCAFS LC-40 with a success rate of 29%, VAFB SLC-4E with a success rate of 17%, and launch site CCAFS SLC-40 with a success rate of 13%.



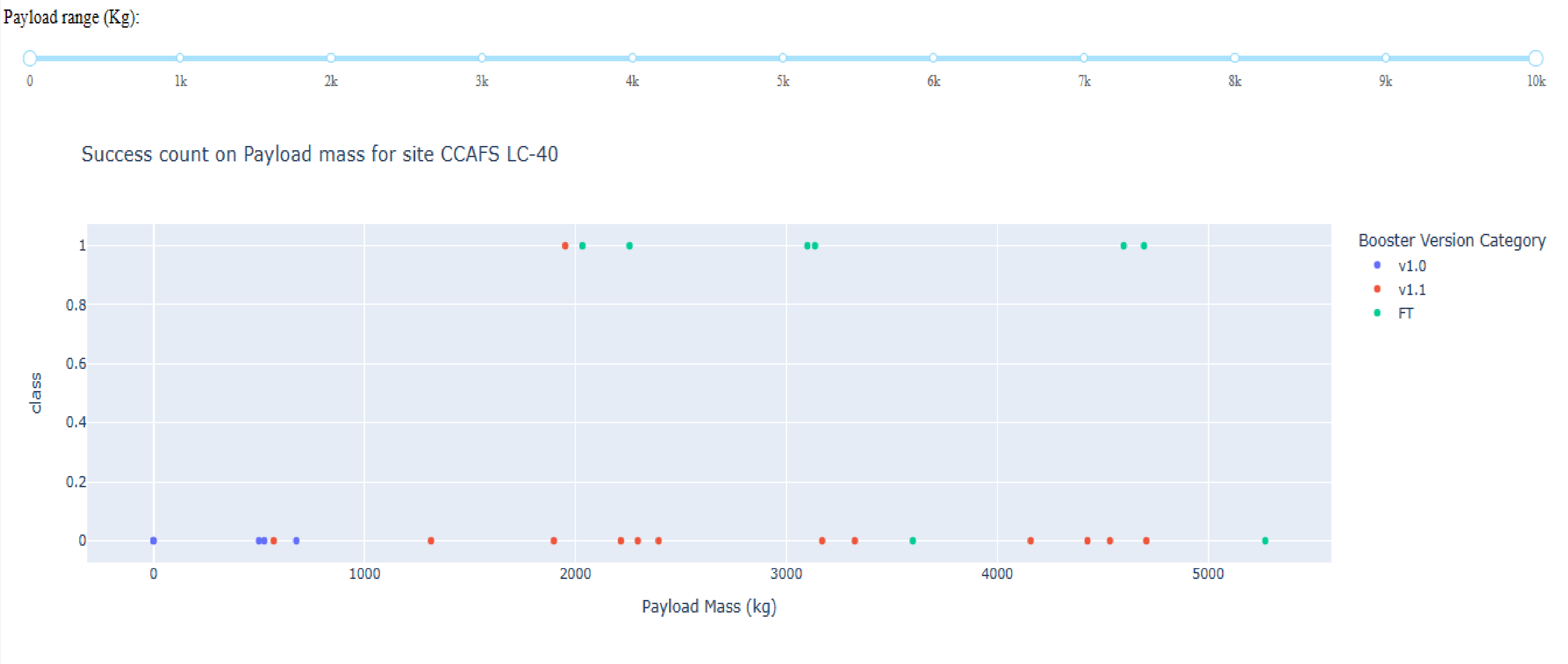
Pie chart for launch site 2nd highest launch success ratio

Launch site CCAFS LC-40 had the Second highest success ratio of 73% success against 27% failed launches



Scatter plot for Payload vs Launch outcome – all sites

For Launch site CCAFS LC-40 the booster version FT has the largest success rate from a payload mass of >2000kg



Section 5

Predictive Analysis (Classification)

Classification Accuracy

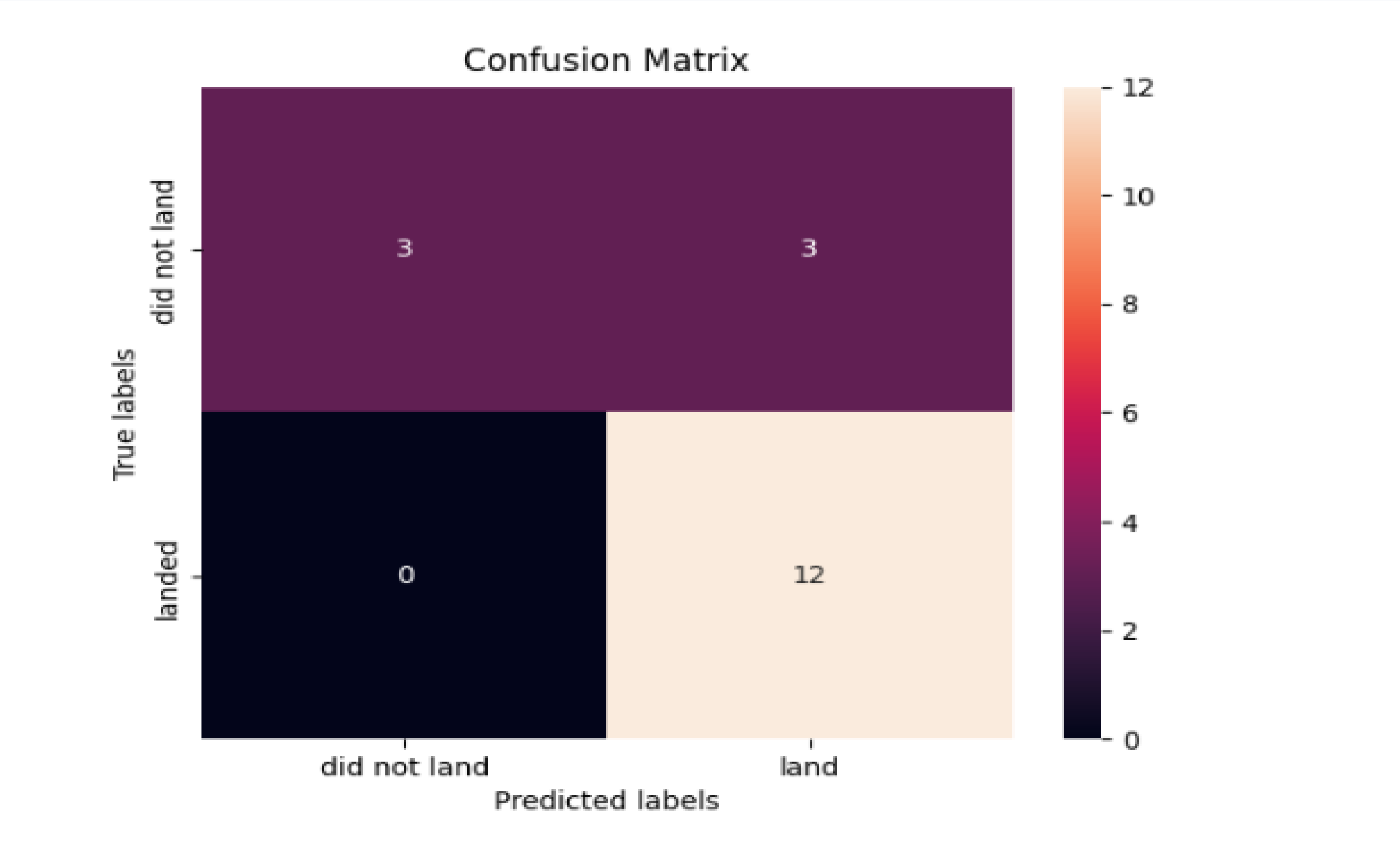
Out[33]:

	0
Method	Test Data Accuracy
Logistic_Reg	0.833333
SVM	0.833333
Decision Tree	0.833333
KNN	0.833333

All the method have same accuracy 0.833 on test data. So, all the method perform equally on test data.

Confusion Matrix

- All four classification models shared the same confusion matrices and were equally capable of differentiating between the various classes.
- False positives for all the models are the main issue.



Conclusions

- Success rates vary between launch locations. The success percentage for CCAFS LC-40 is 60%, compared to 77% for KSC LC-39A and VAFB SLC 4E.
- We may infer that the success rate rises as the number of flights rises at each of the three launch sites. For instance, the VAFB SLC 4E launch site had a 100% success rate following Flight 50. After the 80th flight, both KSC LC 39A and CCAFS SLC 40 achieved 100% success rates.
- There are no rockets launched for big payload mass (more than 10,000) from the VAFB-SLC launch site, according to the Payload Vs. Launch Site scatter point chart.
- The most successful orbits are ES-L1, GEO, HEO, and SSO, with SO orbit having the least successful orbits at about 50%. The success rate of Orbit SO is zero.
- For Polar, LEO, and ISS, the successful landing or positive landing rate is higher while carrying heavier payloads. However, for GTO, it is difficult to make this distinction because both positive and negative landing rates (missions that fail) are present.
- And lastly, from 2013 until 2020, the success rate continued to rise.

Thank you!