IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Lidiia Khashina
22/11/2024

Outline

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

Executive Summary

- Summary of methodologies

  - Data Collection using SpaceX API
  - Data Collection with Web Scraping
  - Data Wrangling
  - Exploratory Data Analysis using SQL
  - Data Visualization with Python Pandas and Matplotlib
  - Visual Analytics with Folium and Ploty Dash
  - Machine Learning Landing Prediction with Classifiaction

- Summary of all results

  Decision Tree has better accuracy than SVM, KNN and Logistic Regression

Introduction

In this capstone, we will predict if the Falcon 9 first stage will land successfully. SpaceX advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is because SpaceX can reuse the first stage. Therefore if we can determine if the first stage will land, we can determine the cost of a launch. This information can be used if an alternate company wants to bid against SpaceX for a rocket launch. In this lab, you will collect and make sure the data is in the correct format from an API. The following is an example of a successful and launch.

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 collected using the SpaceX API, which is a RESTful API. This was accomplished by making a GET request to the SpaceX API after defining several helper functions. These functions facilitated the extraction of information from the launch data using identification numbers.

To ensure consistency in the requested JSON results, we queried the SpaceX launch data through a GET request, decoded the response content into JSON format, and then converted the data into a Pandas DataFrame.

In addition to using the API, web scraping was performed to gather historical launch records for Falcon 9 from a Wikipedia page titled "List of Falcon 9 and Falcon Heavy launches." The launch records were stored in an HTML table. Using the BeautifulSoup and requests libraries, I extracted the Falcon 9 launch data from the HTML table, parsed it, and then converted it into a Pandas DataFrame.

Data Collection – SpaceX API

Data was collected using the SpaceX API, a RESTful API, by making a GET request to retrieve information about SpaceX launches. The response was parsed and the content was decoded into JSON format. This data was then converted into a Pandas DataFrame for further analysis.

https://github.com/adatheleader/falcon9/blob/main/jupyter-labs-spacex-data-collection-api-v2.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:

[21]:
```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_call_spacex_api.js
```

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

[23]:
```
response.status_code
```

[23]:
```
200
```

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

[25]:
```
# Use json_normalize meethod to convert the json result into a dataframe
respjson = response.json()
data = pd.json_normalize(respjson)
```

Using the dataframe `data` print the first 5 rows

[27]:
```
# Get the head of the dataframe
data.head()
```

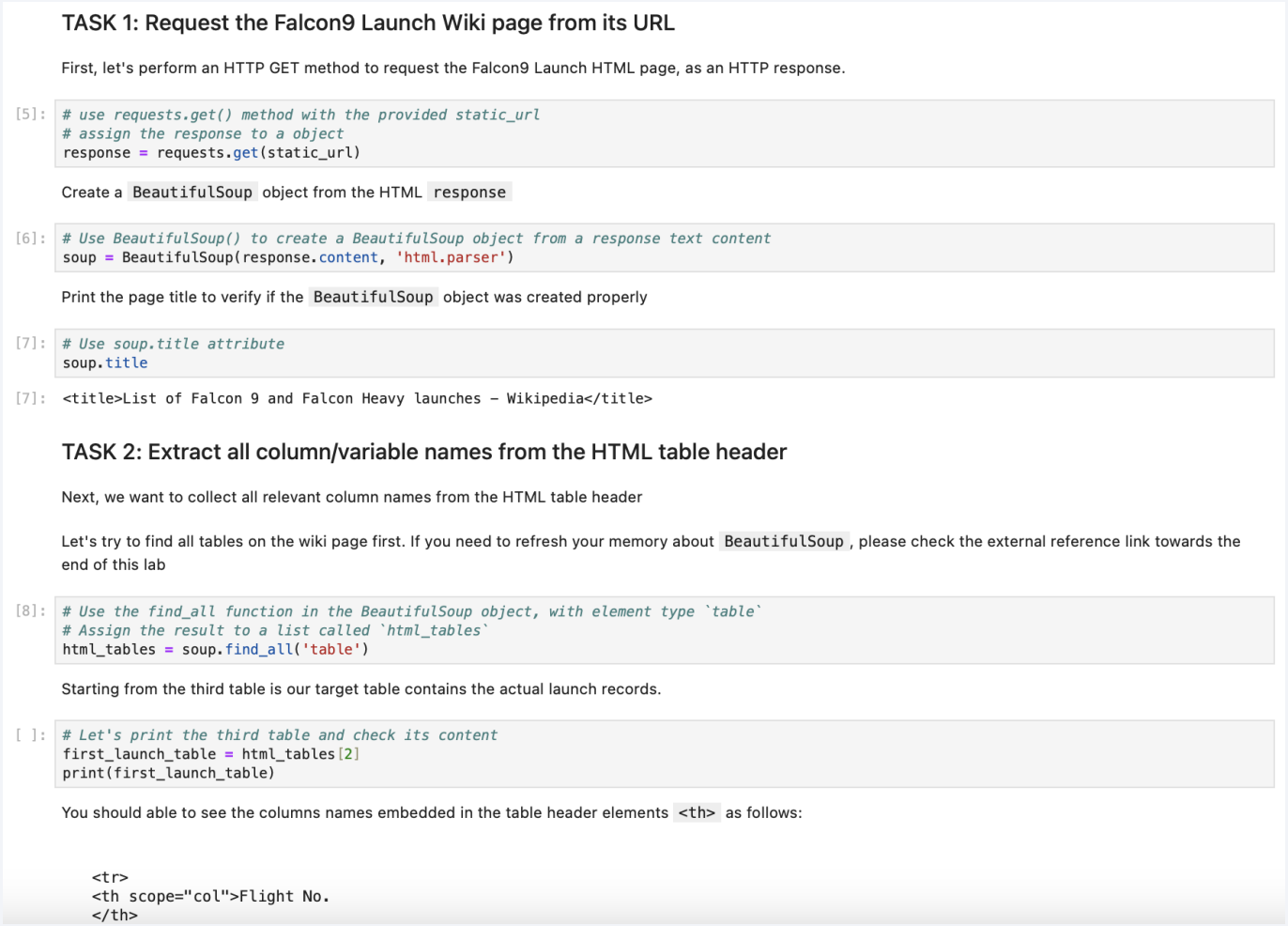
[27]:

	static_fire_date_utc	static_fire_date_unix	net	window	rocket	success	failures	details	crew	ships	capsules	
0	2006-03-17T00:00:00.000Z	1.142554e+09	False	0.0	5e9d0d95eda69955f709d1eb	False	[{'time': 33, 'altitude': None, 'reason': 'merlin engine failure'}]	Engine failure at 33 seconds and loss of vehicle	[]	[]	[]	[5eb0e4b5b6c

Data Collection - Scraping

I performed web scraping to collect historical launch records of the Falcon 9 from Wikipedia. Using BeautifulSoup and the Requests library, I extracted the launch records from the HTML table on the Wikipedia page and then created a data frame by parsing the launch data from the HTML.

https://github.com/adatheleader/falcon9/blob/main/jupyter-labs-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.

[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`

[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

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

[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

[8]:
```
# 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.

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

You should able to see the columns names embedded in the table header elements `<th>` as follows:

```
<tr>
<th scope="col">Flight No.
</th>
```

Data Wrangling

After obtaining and creating a Pandas DataFrame from the collected data, I filtered the data using the BoosterVersion column to keep only the Falcon 9 launches. Next, I addressed the missing values in the LandingPad and PayloadMass columns. For the PayloadMass, I replaced the missing values with the mean of the column. Additionally, I conducted Exploratory Data Analysis (EDA) to identify patterns in the data and to determine a suitable label for training supervised models.

https://github.com/adatheleader/falcon9/blob/main/labs-jupyter-spacex-Data%20wrangling.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`:

[11]:
```
# 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()
```

[11]:
```
Class
1    60
0    30
Name: count, 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

[13]:
```
landing_class=df['Class']
df[['Class']].head(8)
```

[13]:

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

[14]:
```
df.head(5)
```

[14]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Seri
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B00(
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B00(

EDA with Data Visualization

I conducted data analysis and feature engineering using Pandas and Matplotlib. Specifically, I utilized scatter plots to visualize the relationships between flight number and launch site, payload and launch site, flight number and orbit type, as well as payload and orbit type. Additionally, I employed bar charts to illustrate the success rates of each orbit type and created a line plot to display the yearly trend in launch success.

https://github.com/adatheleader/falcon9/blob/main/jupyter-labs-eda-dataviz-v2.ipynb

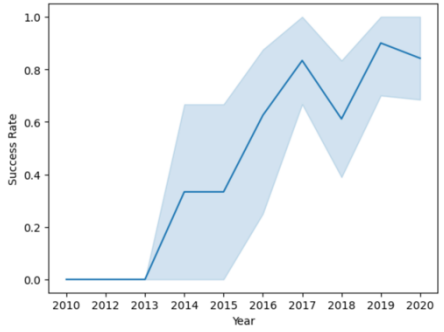
TASK 1: Visualize the relationship between Flight Number and Launch Site

Use the function `catplot` to plot `FlightNumber` vs `LaunchSite`, set the parameter `x` parameter to `FlightNumber`,set the `y` to `Launch Site` and set the parameter `hue` to `'class'`

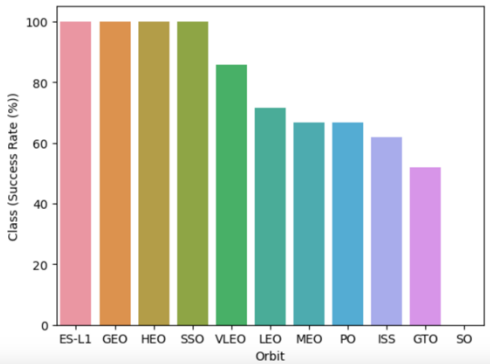
```
[40]: # Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to be the class value
sns.catplot(y="LaunchSite", x="FlightNumber", hue="Class", data=df, aspect = 2.0 ,height=5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Launch Site",fontsize=20)
plt.show()
```

```
/Users/lydiakhashina/anaconda3/lib/python3.11/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```


```
[32]: # Plot a line chart with x axis to be the extracted year and y axis to be the success rate
sns.lineplot(data=df, x="Date", y="Class")
plt.xlabel("Year")
plt.ylabel("Success Rate")
plt.show()
```




```
[20]: # HINT use groupby method on Orbit column and get the mean of Class column
sr_df = df.groupby('Orbit')['Class'].mean().reset_index().sort_values(by='Class', ascending=False)
sr_df['Class'] = sr_df['Class'] * 100
sns.barplot(data=sr_df, x='Orbit', y='Class')
plt.xlabel('Orbit')
plt.ylabel('Class (Success Rate (%))')
plt.show()
```



EDA with SQL

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

File display

```
%sql SELECT DISTINCT LAUNCH_SITE as "Launch_Sites" FROM SPACEXTBL;
```

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

Out[20]:

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

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

In [22]:

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

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

Out[22]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (p
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 (p
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	N
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	N
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	N

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

[24]:

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

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

[24]:

Total Payload Mass(Kgs)	Customer
45596	NASA (CRS)

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

[26]:

```
%sql SELECT AVG(PAYLOAD_MASS__KG_) as "Payload Mass Kgs", Customer, Booster_Version FROM 'SPACEXTBL' WHERE Booster_Version LIKE 'F9 v1.1%';
```

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

[26]:

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

- List the date when the first successful landing outcome in ground pad was achieved

[36]:

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

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

[36]:

MIN(DATE)
2015-12-22

https://github.com/adatheleader/falcon9/blob/main/jupyter-labs-eda-sql-coursera_sqllite.ipynb

EDA with SQL (continued)

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

```
[33]: %sql SELECT DISTINCT Booster_Version, Payload FROM SPACEXTBL WHERE "Landing_Outcome" = "Success (drone ship)" AND PAYLOAD_MASS__KG_ > 4000 AND
```

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

[33]:

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

- List the total number of successful and failure mission outcomes

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

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

[42]:

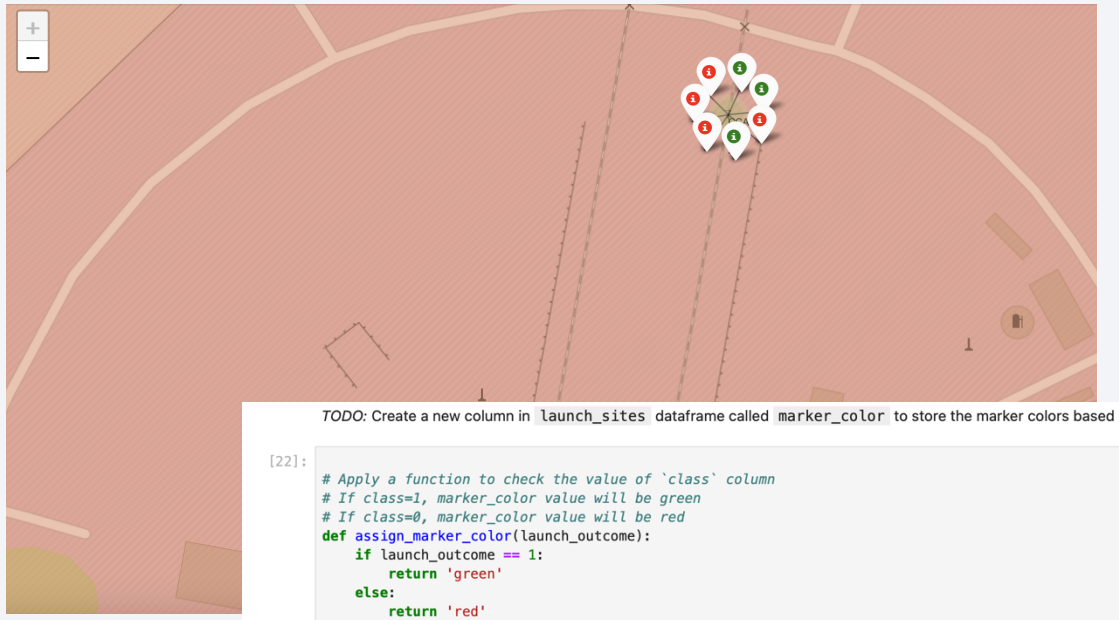
Mission_Outcome	Total
Failure (in flight)	1
Success	98
Success	1
Success (payload status unclear)	1

https://github.com/adatheleader/falcon9/blob/main/jupyter-labs-eda-sql-coursera_sqllite.ipynb

Build an Interactive Map with Folium

Created a Folium map to mark all the launch sites and to create map objects such as markers, circles, and lines to indicate the success or failure of launches for each launch site. A launch outcome dataset was created with values for failure (0) and success (1).

https://github.com/adathelea der/falcon9/blob/main/lab -jupyter-launch-site-location-v2.ipynb



TODO: Create a new column in `launch_sites` dataframe called `marker_color` to store the marker colors based on the `class` value

[22]:

```
# Apply a function to check the value of `class` column
# If class=1, marker_color value will be green
# If class=0, marker_color value will be red
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)
```

[22]:

	Launch Site	Lat	Long	class	marker_color
46	KSC LC-39A	28.573255	-80.646895	1	green
47	KSC LC-39A	28.573255	-80.646895	1	green
48	KSC LC-39A	28.573255	-80.646895	1	green
49	CCAFS SLC-40	28.563197	-80.576820	1	green
50	CCAFS SLC-40	28.563197	-80.576820	1	green
51	CCAFS SLC-40	28.563197	-80.576820	0	red
52	CCAFS SLC-40	28.563197	-80.576820	0	red
53	CCAFS SLC-40	28.563197	-80.576820	0	red
54	CCAFS SLC-40	28.563197	-80.576820	1	green
55	CCAFS SLC-40	28.563197	-80.576820	0	red

Build a Dashboard with Plotly Dash

I developed an interactive dashboard application using Plotly Dash by implementing the following features:

1. Added a drop-down input component to select a launch site.

2. Created a callback function that displays a success pie chart based on the selected launch site.

3. Incorporated a range slider to select the payload.

4. Developed a callback function to render a scatter plot of success versus payload.

```
    # TASK 1: Add a dropdown list to enable Launch Site selection
    # The default select value is for ALL sites
    # dcc.Dropdown(id='site-dropdown',...)
    dcc.Dropdown(id='site-dropdown',
                 options=[
                         {'label': 'All Sites', 'value': 'ALL'},
                         {'label': 'CCAFS LC-40', 'value': 'CCAFS LC-40'},
                         {'label': 'VAFB SLC-4E', 'value': 'VAFB SLC-4E'},
                         {'label': 'KSC LC-39A', 'value': 'KSC LC-39A'},
                         {'label': 'CCAFS SLC-40', 'value': 'CCAFS SLC-40'}
                         ],
                 value='ALL',
                 placeholder='Select a Launch Site here',
                 searchable=True
                 # style={'width':'80%','padding':'3px','font-size':'20px','text-align-last':'center'}
                 ),
    html.Br(),
```

```
# Add a callback function for `site-dropdown` as input, `success-pie-chart` as output
@app.callback(Output(component_id='success-pie-chart', component_property='figure'),
              Input(component_id='site-dropdown', component_property='value'))
def get_pie_chart(entered_site):
    filtered_df = spacex_df
    if entered_site == 'ALL':
        fig = px.pie(filtered_df, values='class',
        names='Launch Site',
        title='Success Count for all launch sites')
        return fig
    else:
        # return the outcomes piechart for a selected site
        filtered_df=spacex_df[spacex_df['Launch Site']== entered_site]
        filtered_df=filtered_df.groupby(['Launch Site','class']).size().reset_index(name='class count')
        fig=px.pie(filtered_df,values='class count',names='class',title=f"Total Success Launches for site {entered_site}")
        return fig
```

https://github.com/adatheleader/falcon9/blob/main/spacex_dash_app.py

Predictive Analysis (Classification)

After loading the data as a Pandas DataFrame, I began conducting Exploratory Data Analysis and determining the training labels. I created a NumPy array from the 'Class' column in the data by applying the numpy() method and assigned it to the variable Y, which serves as the outcome variable.

Next, I standardized the feature dataset (X) by transforming it using the `preprocessing.StandardScaler()` function from Scikit-learn. After standardization, I split the data into training and testing sets using the `train_test_split` function from `sklearn.model_selection`, with the `test_size` parameter set to 0.2 and `random_state` set to 2.

To identify the best machine learning model that would perform best on the test data, I evaluated several algorithms: Support Vector Machines (SVM), Classification Trees, k-Nearest Neighbors, and Logistic Regression. For each algorithm, I created an object and then instantiated a `GridSearchCV` object, assigning a set of parameters for each model.

For each model under evaluation, I created the `GridSearchCV` object with `cv=10` and fit the training data into it to find the best hyperparameters. Once I fitted the training set, I output the `GridSearchCV` object for each model. I then displayed the best parameters using the `best_params_` attribute and the accuracy on the validation data using the `best_score` attribute.

Finally, I used the `score` method to calculate the accuracy on the test data for each model and plotted a confusion matrix for each one, comparing the test outcomes with the predicted results.

https://github.com/adatheleader/falcon9/blob/main/SpaceX-Machine-Learning-Prediction-Part-5-v1.ipynb

File display

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

In [25]:

```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()
logreg_cv = GridSearchCV(lr, parameters, cv=10)
logreg_cv.fit(X_train, Y_train)
```

Out[25]:

File display

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

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

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 [27]:

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

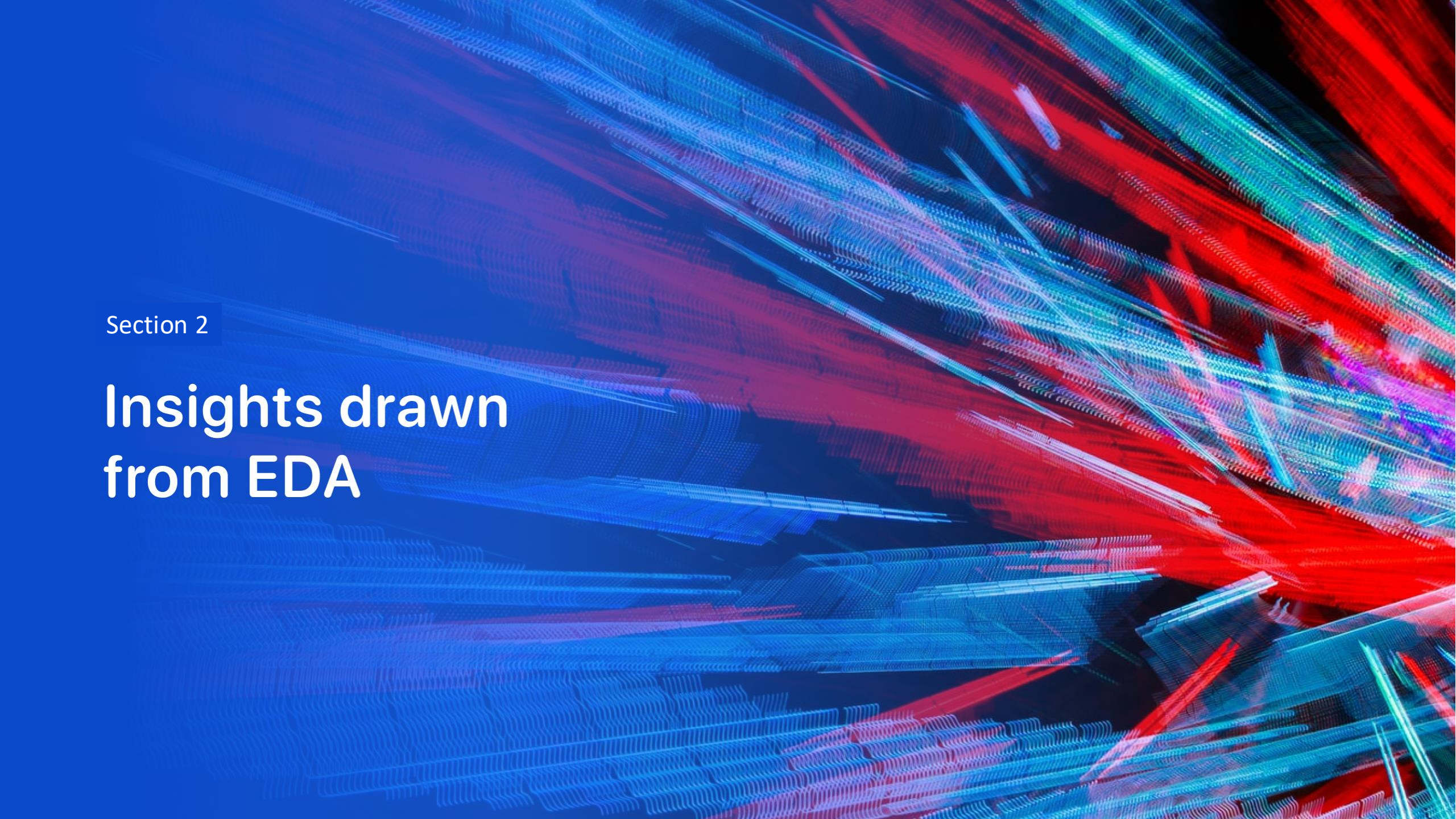
```
tuned hpyerparameters :(best parameters)  {'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8196428571428571
```

File display

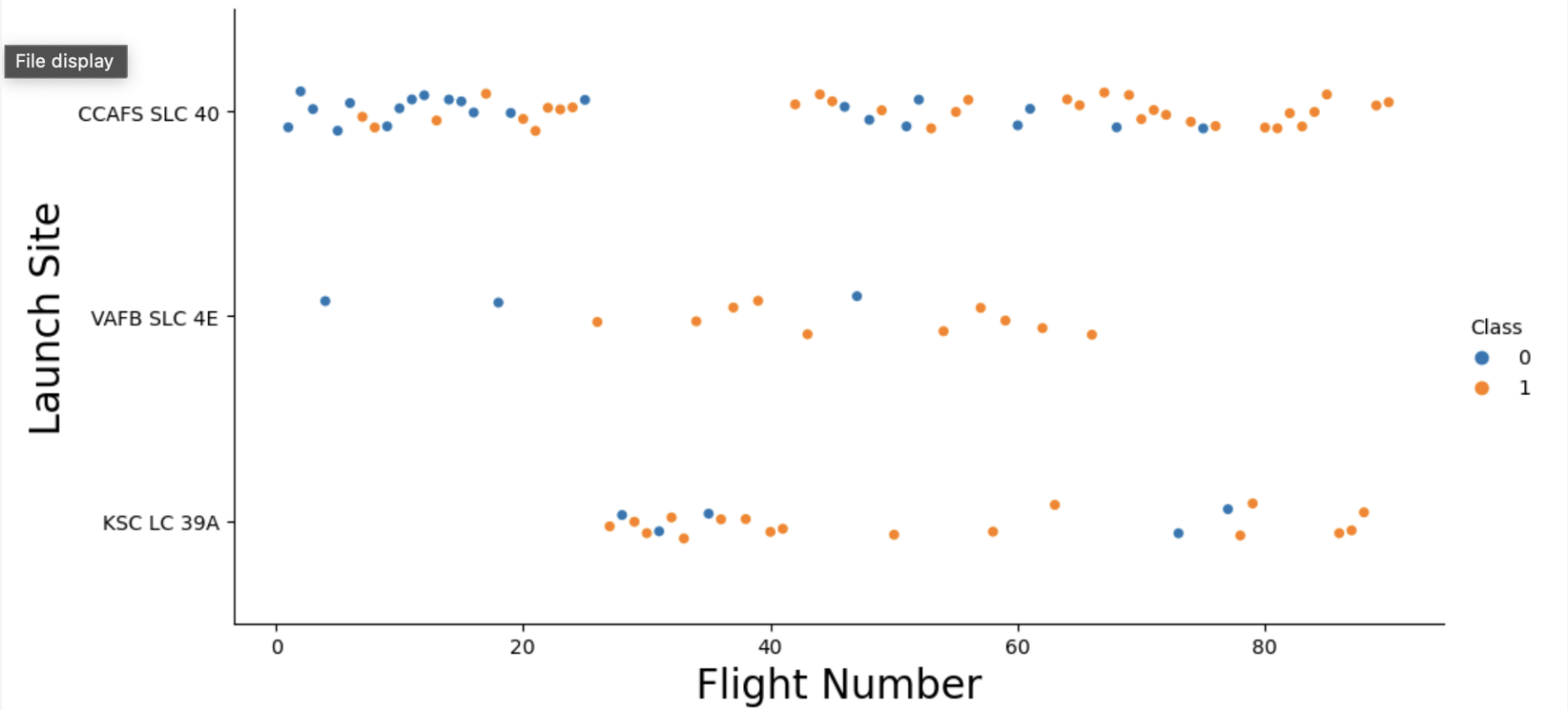
```
parameters = {'criterion': ['gini', 'entropy'],
     'splitter': ['best', 'random'],
     'max_depth': [2*n for n in range(1,10)],
     'max_features': ['auto', 'sqrt'],
     'min_samples_leaf': [1, 2, 4],
     'min_samples_split': [2, 5, 10]}

tree = DecisionTreeClassifier()
```

In [35]:

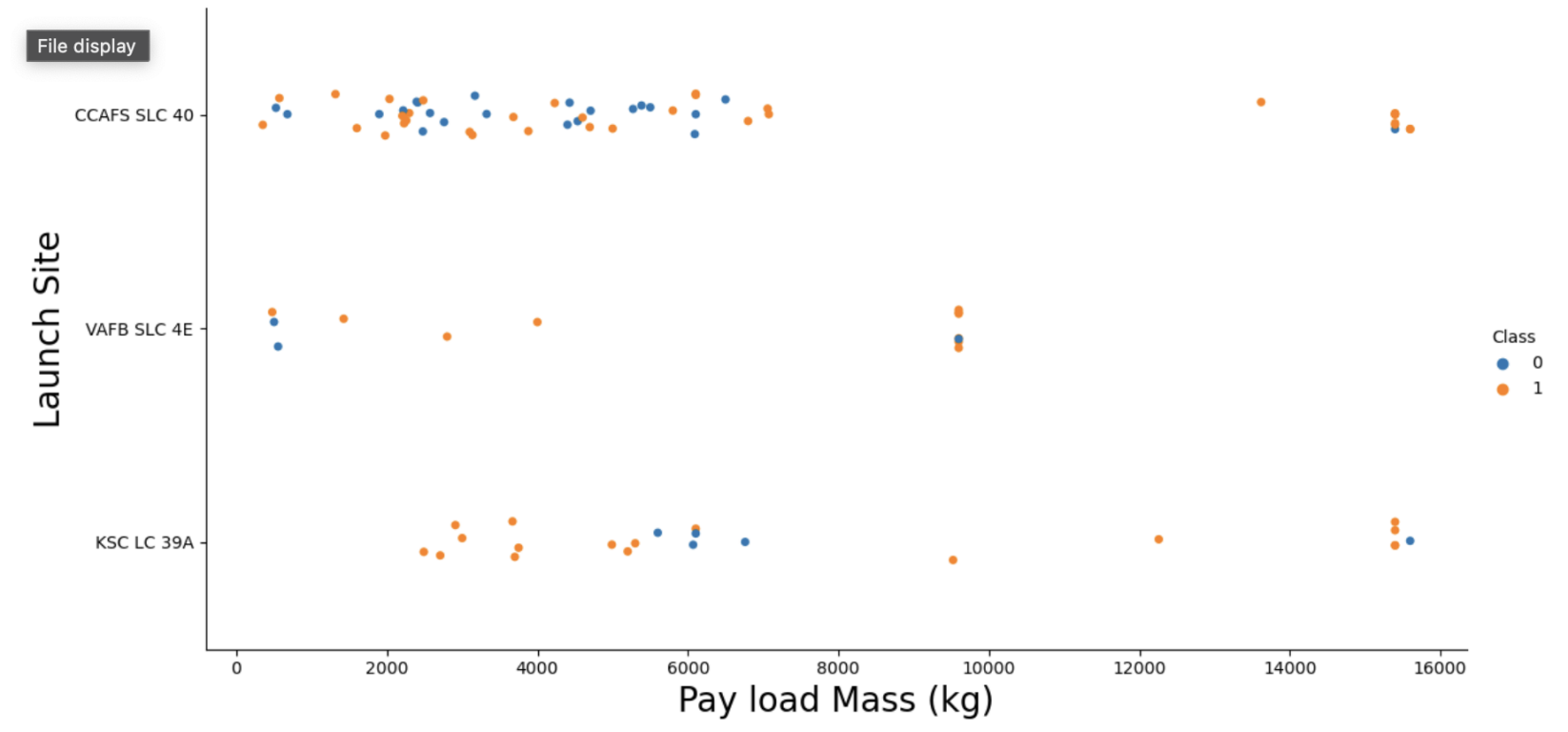
```
# Instantiate the GridSearchCV object: tree_cv
tree_cv = GridSearchCV(tree, parameters, cv=10)

# Fit it to the data
tree_cv.fit(X_train, Y_train)
```


Section 2
Insights drawn from EDA

Flight Number vs. Launch Site

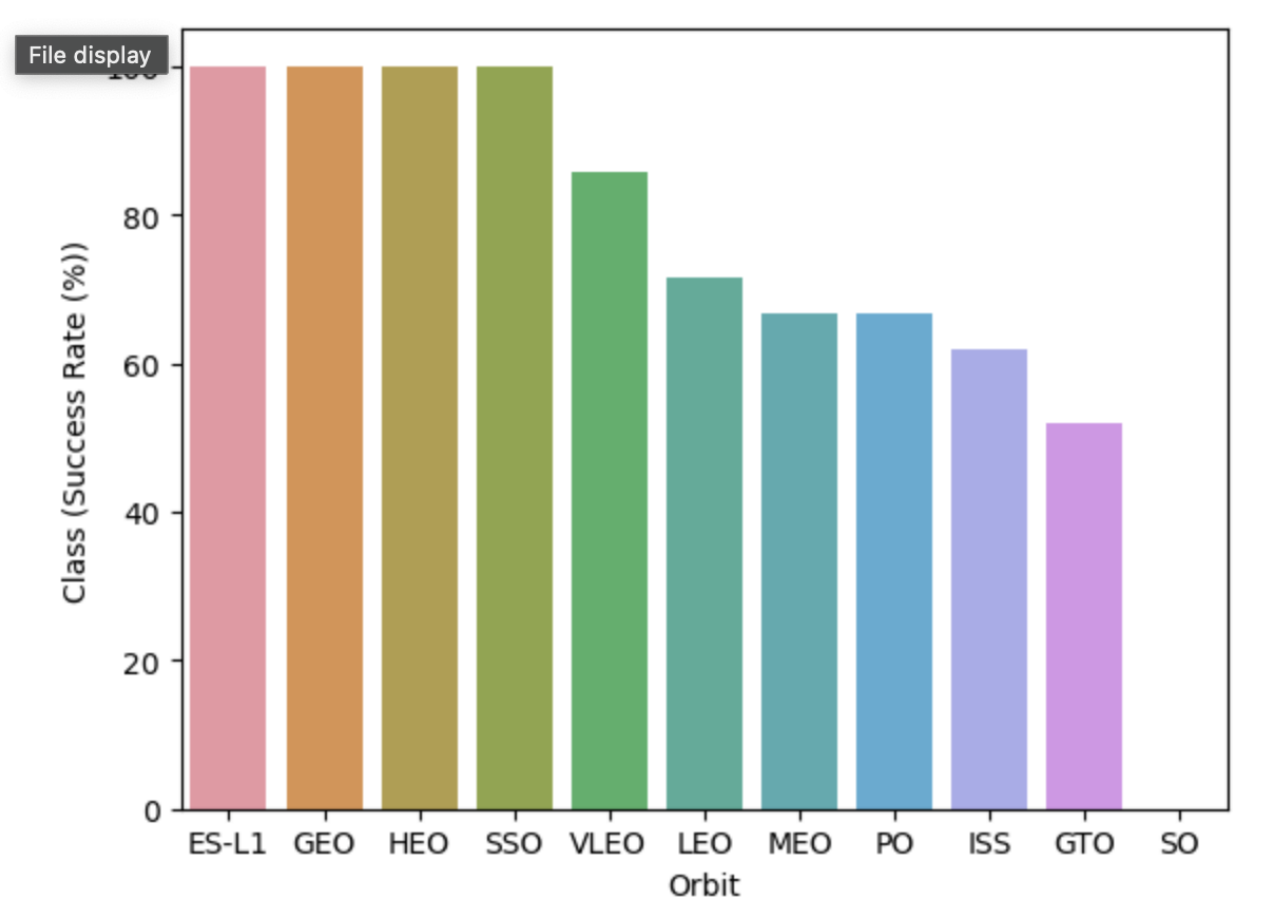


Payload vs. Launch Site

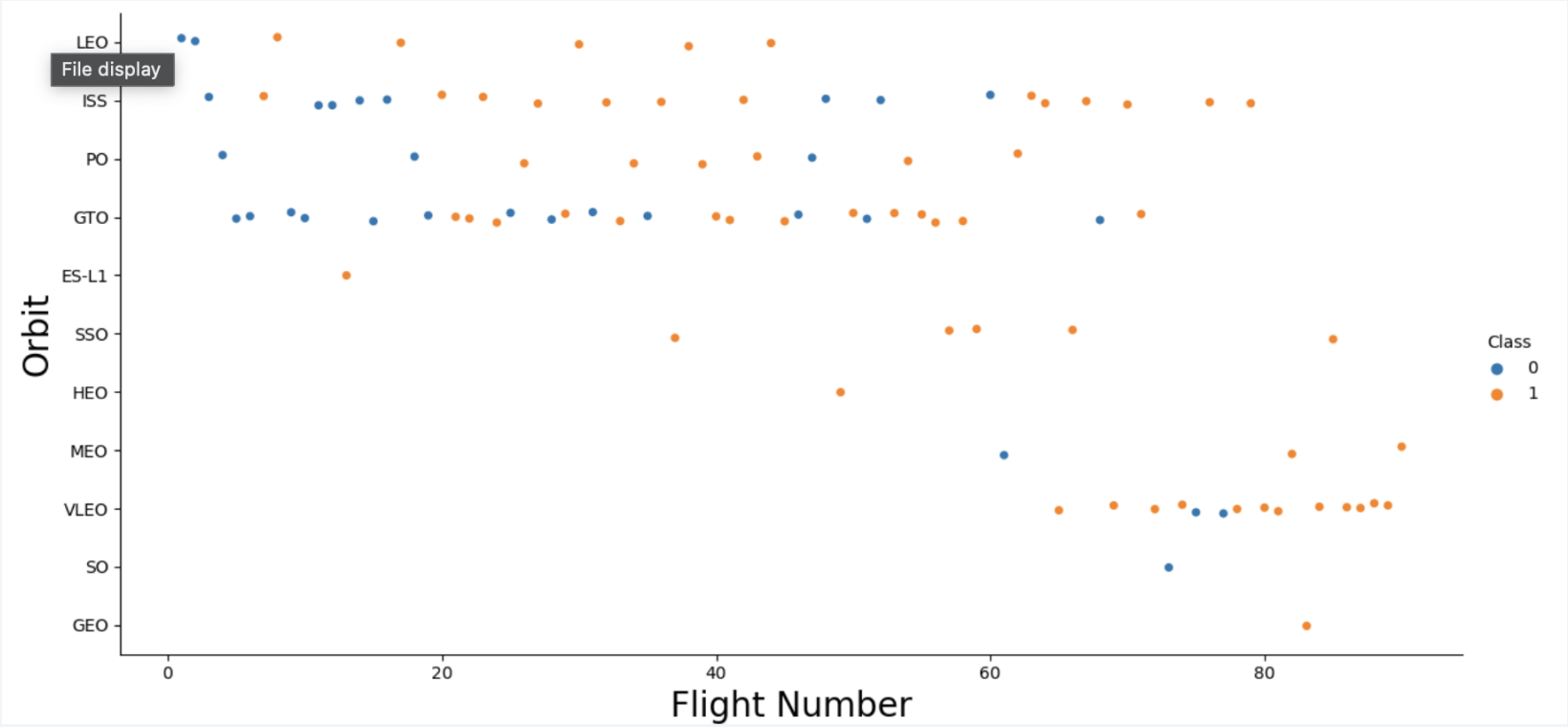


Now if you observe Payload Vs. Launch Site scatter point chart you will find for the VAFB-SLC launchsite there are no rockets launched for heavypayload mass(greater than 10000).

Success Rate vs. Orbit Type

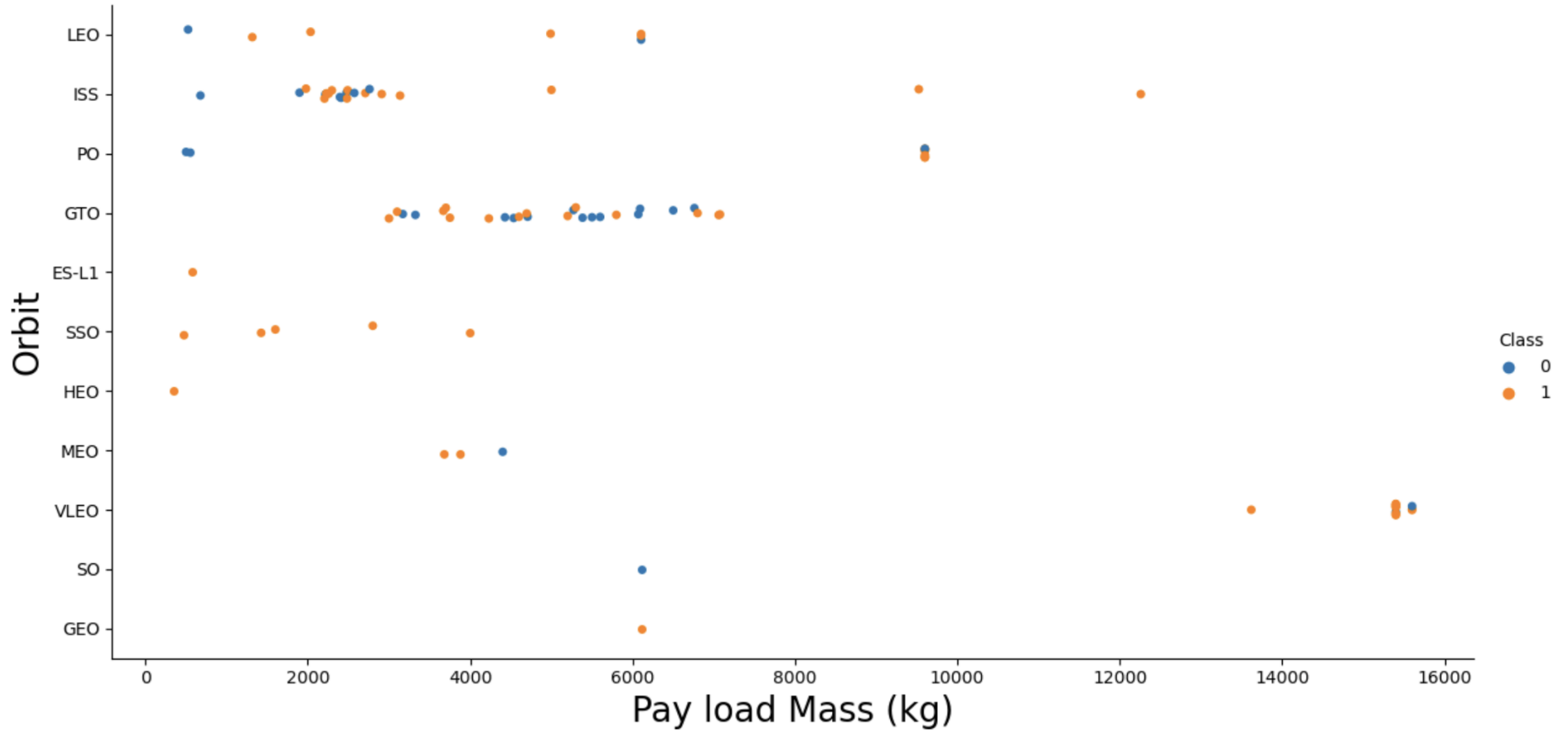


Flight Number vs. Orbit Type



You should 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.

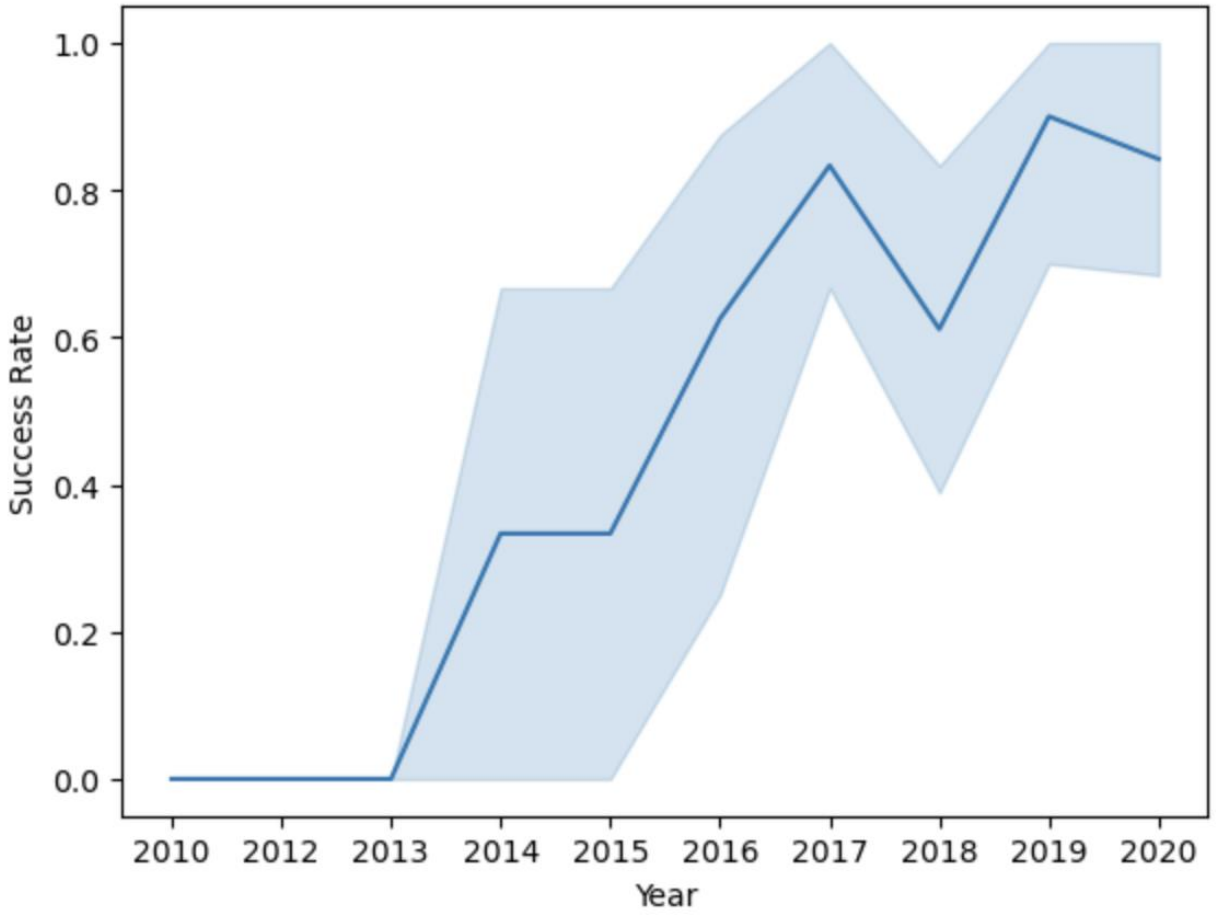
Payload vs. Orbit Type



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



You can observe that the success rate since 2013 kept increasing till 2017 (stable in 2014) and after 2015 it started increasing.

All Launch Site Names

File display

```
%sql SELECT DISTINCT LAUNCH_SITE as "Launch_Sites" FROM SPACEXTBL;
```

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

Out[20]:

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

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

Launch Site Names Begin with 'CCA'

In [22]:
```
%sql SELECT * FROM 'SPACEXTBL' WHERE Launch_Site LIKE 'CCA%' LIMIT 5;
```

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

Out[22]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (p
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 (p
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	N
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	N
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	N

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

Total Payload Mass

```
[24]: %sql SELECT SUM(PAYLOAD_MASS__KG_) as "Total Payload Mass(Kgs)", Customer FROM 'SPACEXTBL' WHERE Customer = 'NASA (CRS)';
 * sqlite:///my_data1.db
Done.
```

[24]:

Total Payload Mass(Kgs)	Customer
45596	NASA (CRS)

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

Average Payload Mass by F9 v1.1

```
[26]: %sql SELECT AVG(PAYLOAD_MASS__KG_) as "Payload Mass Kgs", Customer, Booster_Version FROM 'SPACEXTBL' WHERE Booster_Version LIKE 'F9 v1.1%';
 * sqlite:///my_data1.db
Done.
```

[26]:

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

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

First Successful Ground Landing Date

```
[36]: %sql SELECT MIN(DATE) FROM 'SPACEXTBL' WHERE "Landing_Outcome" = "Success (ground pad)";
 * sqlite:///my_data1.db
Done.
[36]: MIN(DATE)
      2015-12-22
```

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

Successful Drone Ship Landing with Payload between 4000 and 6000

```
[33]: %sql SELECT DISTINCT Booster_Version, Payload FROM SPACEXTBL WHERE "Landing_Outcome" = "Success (drone ship)" AND PAYLOAD_MASS__KG_ > 4000 AND
```

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

[33]:

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

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

Total Number of Successful and Failure Mission Outcomes

```
[42]: %sql SELECT "Mission_Outcome", COUNT("Mission_Outcome") as Total FROM SPACEXTBL GROUP BY "Mission_Outcome";
 * sqlite:///my_data1.db
Done.
```

[42]:

Mission_Outcome	Total
Failure (in flight)	1
Success	98
Success	1
Success (payload status unclear)	1

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

Boosters Carried Maximum Payload

```
[44]: %sql SELECT "Booster_Version",Payload, "PAYLOAD_MASS__KG_" FROM SPACEXTBL WHERE "PAYLOAD_MASS__KG_" = (SELECT MAX("PAYLOAD_MASS__KG_") FROM SP
 * sqlite:///my_data1.db
Done.
```

[44]:

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

Using a Subquerry to return and pass the Max payload and used it list all the boosters that have carried the Max payload of 15600kgs

2015 Launch Records

```
[46]: %sql SELECT substr(Date,0,5), substr(Date, 6, 2),"Booster_Version", "Launch_Site", Payload, "PAYLOAD_MASS__KG_", "Mission_Outcome", "Landing_O
```

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

[46]:

substr(Date,0,5)	substr(Date, 6, 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)

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

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

```
[65]: %sql SELECT Landing_Outcome, COUNT(*) AS NUMBERS FROM SPACEXTBL WHERE DATE>'2010-06-04' AND  DATE < '2017-03-20' GROUP BY landing_outcome ORDE
```

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

[65]:

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

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

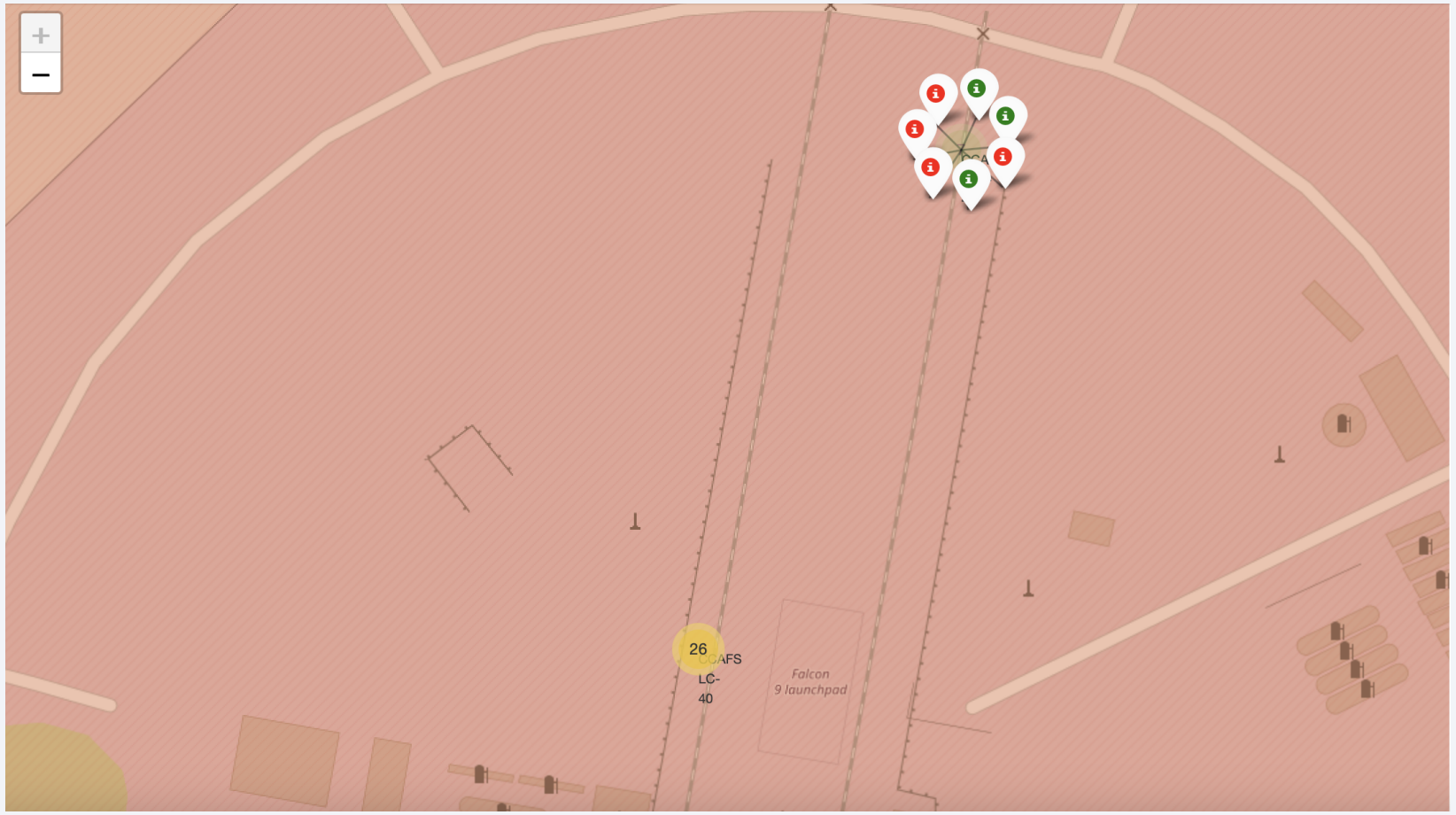
Section 3

Launch Sites Proximities Analysis

Mark all launch sites on a map

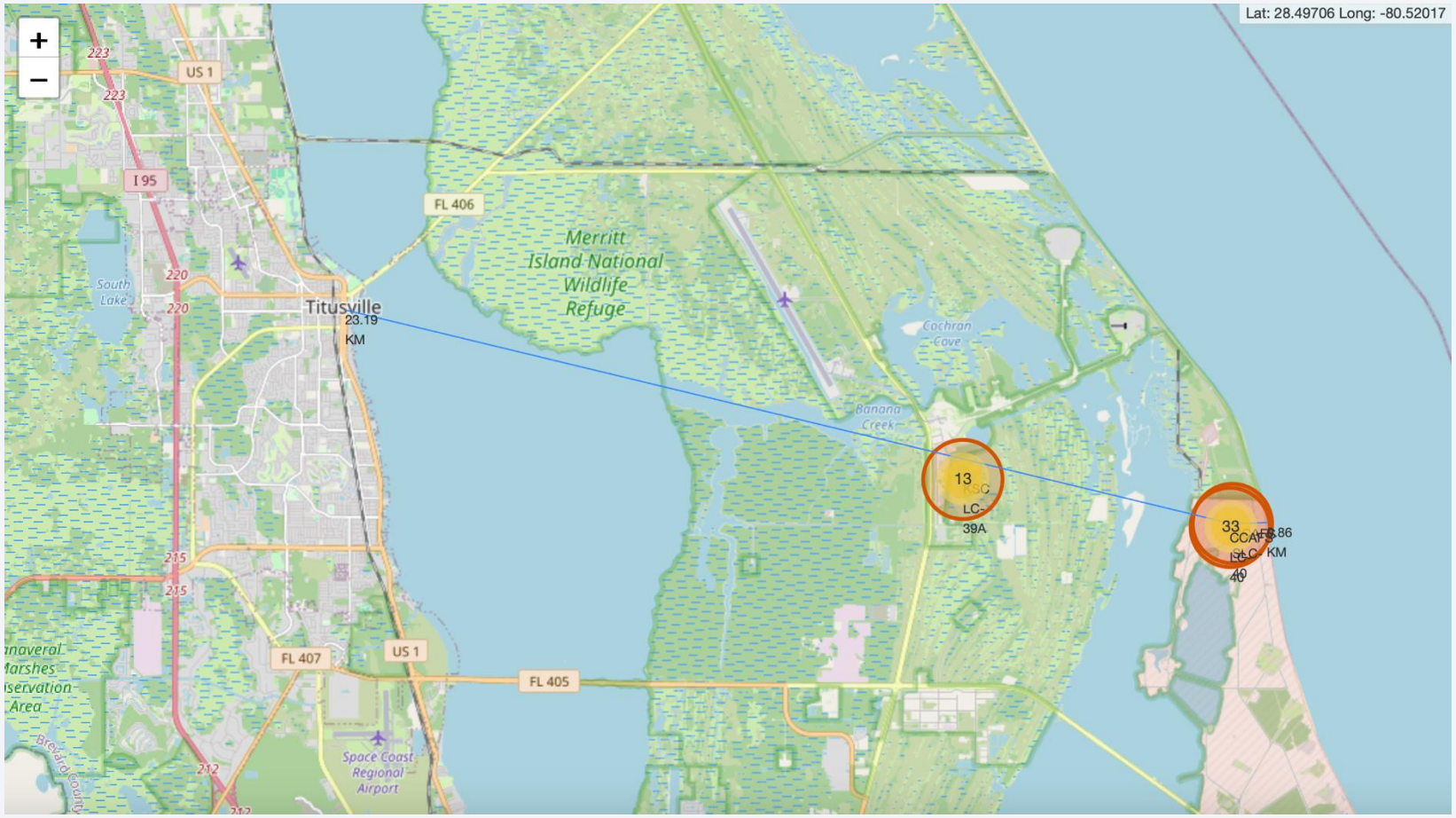
All launch sites are in proximity to the Equator, (located southwards of the US map). Also, all the launch sites are in very close proximity to the coast.

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

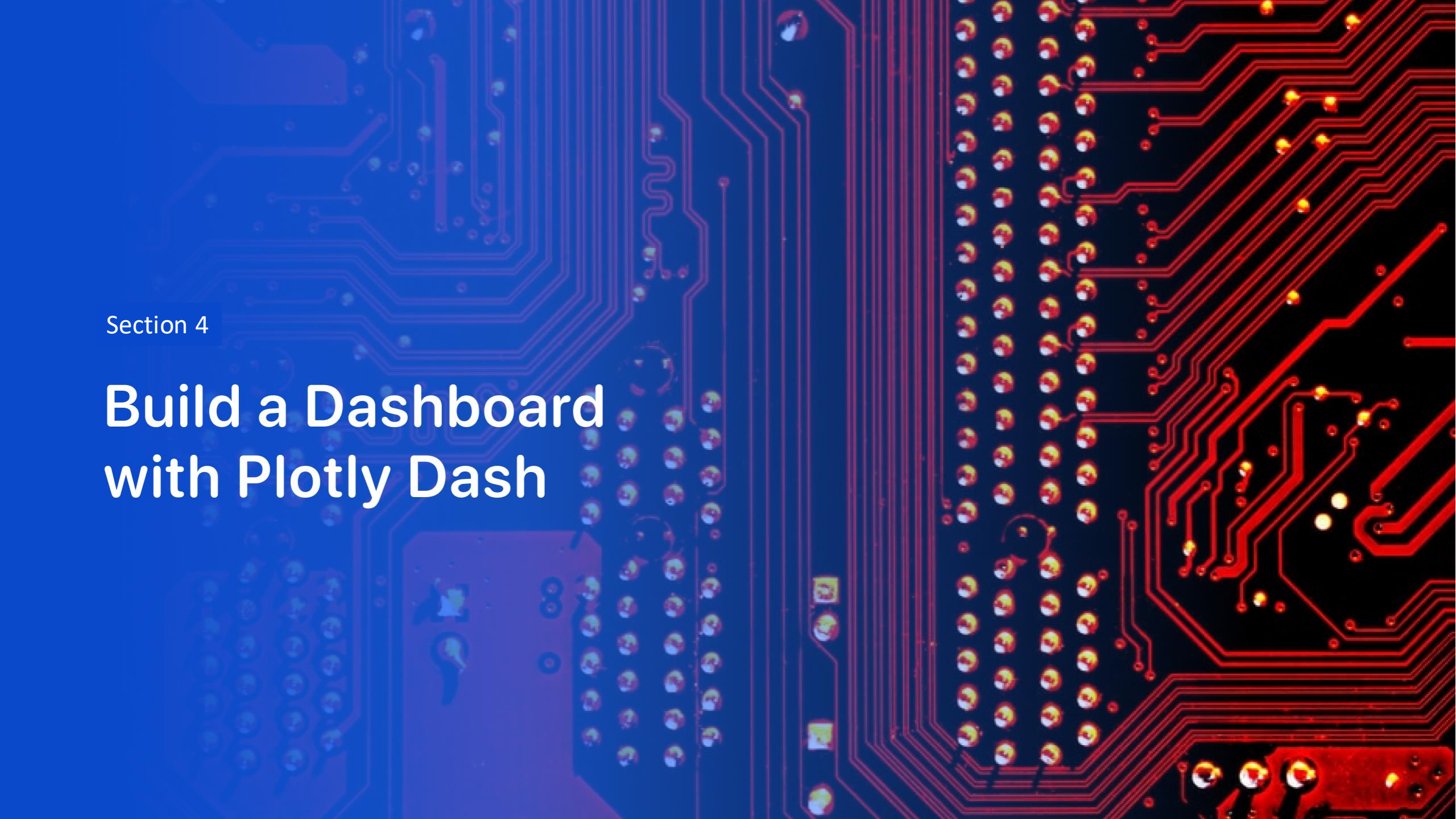


In the Eastern coast (Florida) Launch site KSC LC-39A has relatively high success rates compared to CCAFS SLC-40 & CCAFS LC-40.

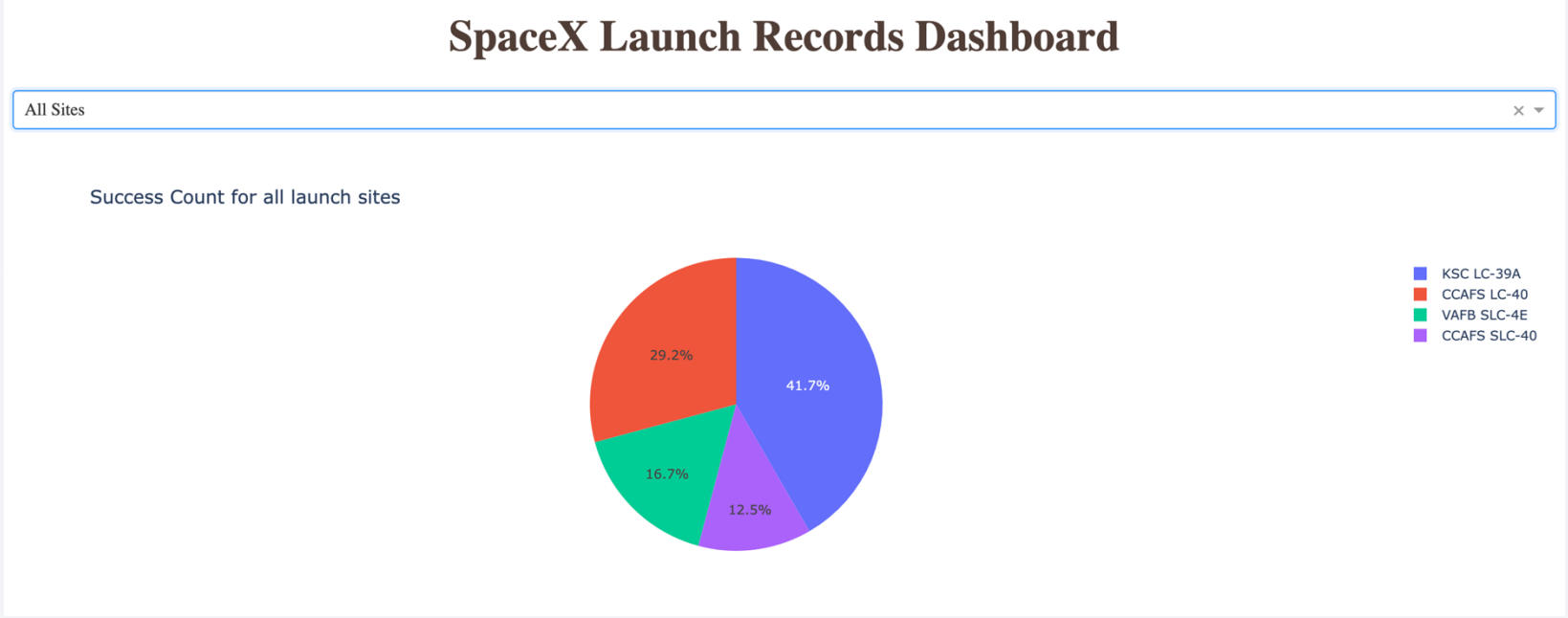
Calculate the distances between a launch site to its proximities



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

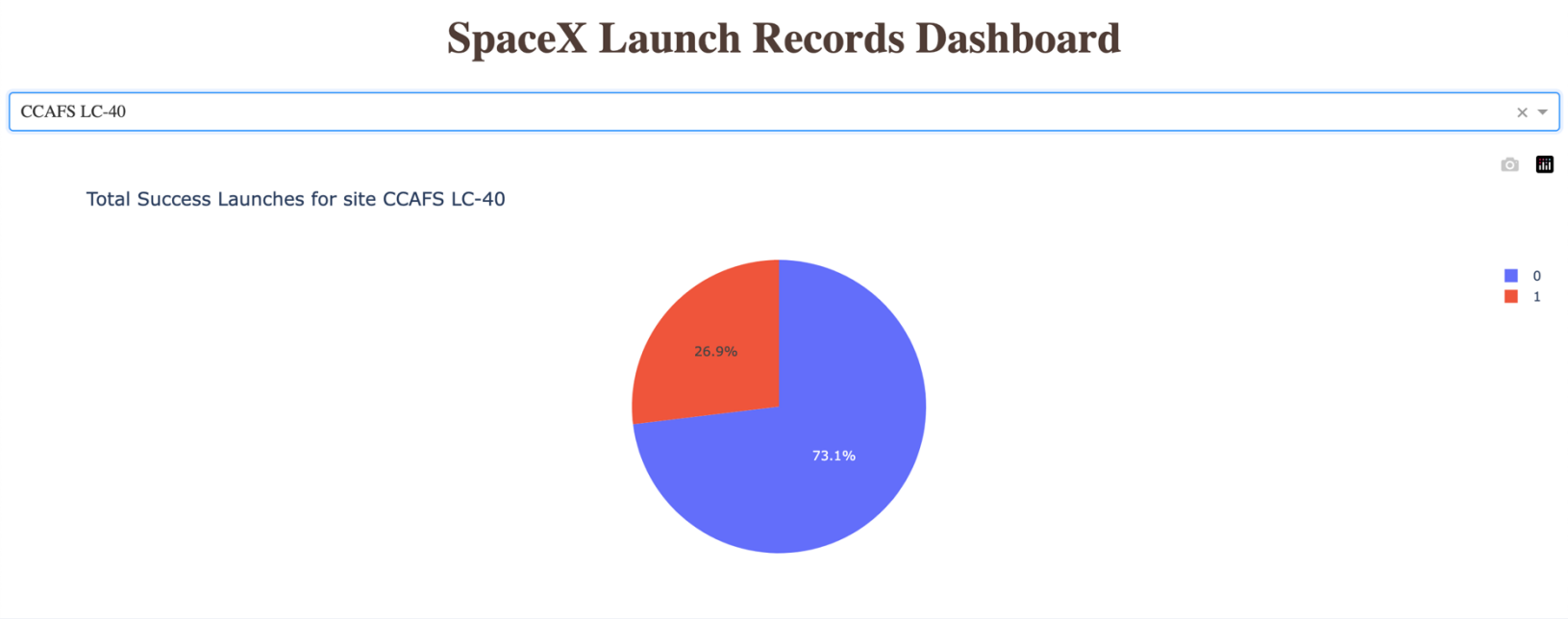

Section 4
Build a Dashboard with Plotly Dash

Launch success count for all sites, in a piechart



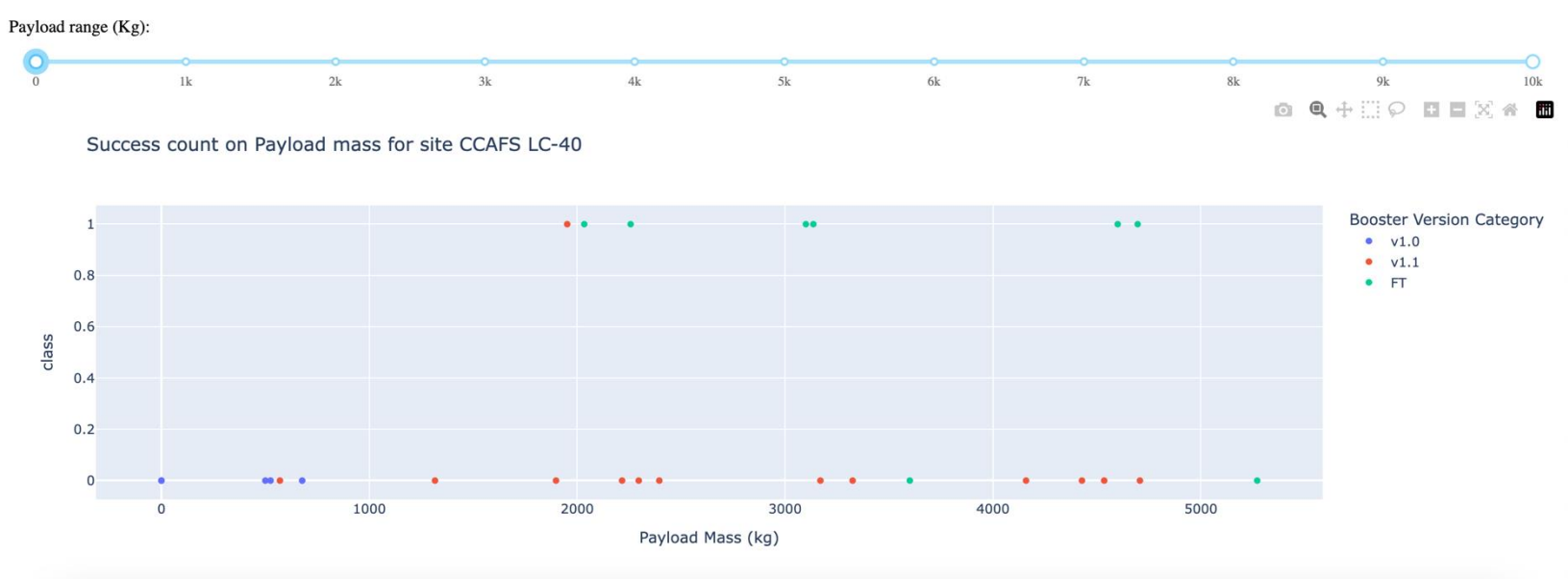
Launch site KSC LC-39A has the highest launch success rate at 42% followed by CCAFS LC-40 at 29%, VAFB SLC-4E at 17% and lastly launch site CCAFS SLC-40 with a success rate of 13%

The piechart for the launch site with highest launch success ratio



Launch site CCAFS LC-40 had the 2nd highest success ratio of 73% success against 27% failed launches

Payload vs. Launch Outcome scatter plot



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

Section 5

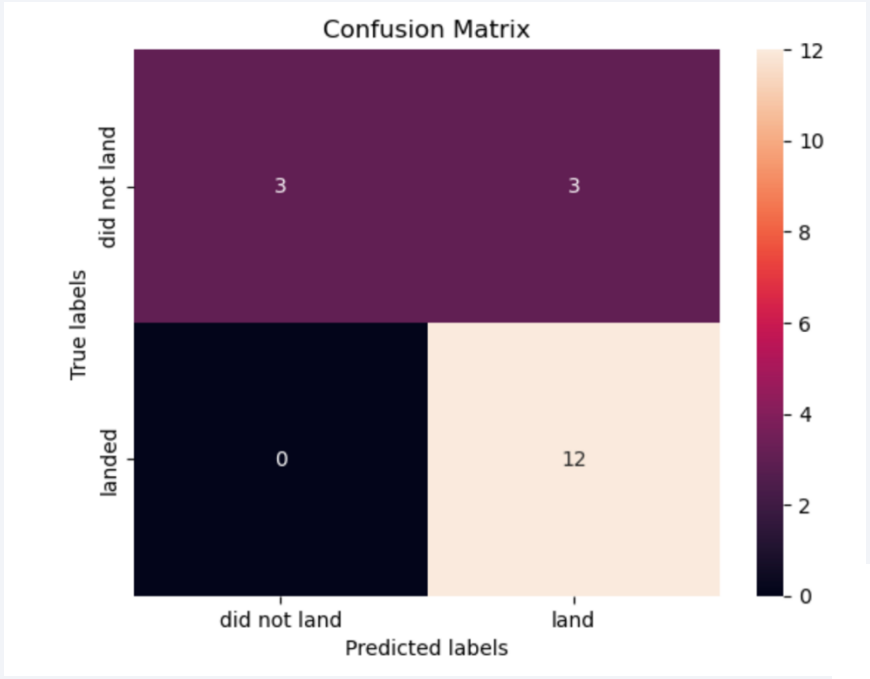
Predictive Analysis (Classification)

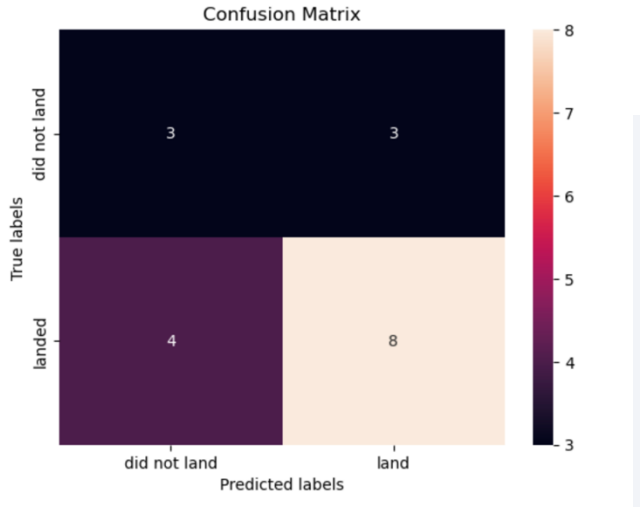
Classification Accuracy

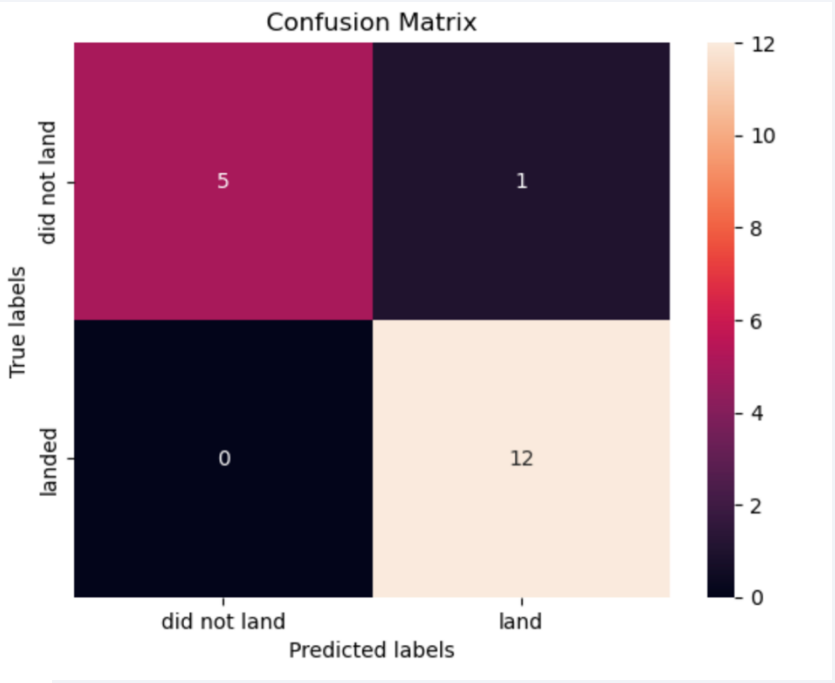
[53]:

	0
Method	Test Data Accuracy
Logistic_Reg	0.833333
Decision Tree	0.944444
KNN	0.611111

Confusion Matrix







Conclusions

- Different launch sites have varying success rates. For instance, CCAFS LC-40 has a success rate of 60%, while KSC LC-39A and VAFB SLC-4E both have a success rate of 77%.
- We can deduce that, as the number of flights increases at each of the three launch sites, so does the success rate. For example, the success rate for the VAFB SLC-4E launch site reaches 100% after the 50th flight. Both KSC LC-39A and CCAFS LC-40 achieve a 100% success rate after their 80th flight.
- If you observe the Payload vs. Launch Site scatter plot, you'll find that there are no rockets launched for heavy payloads (greater than 10000) from the VAFB SLC-4E launch site.
- The orbits ES-L1, GEO, HEO, and SSO exhibit the highest success rates at 100%, while the SO orbit has the lowest success rate at approximately 50%, with a few instances of 0% success.
- In LEO orbit, success appears to be related to the number of flights; conversely, there seems to be no relationship between the flight number and success rates in GTO orbit.
- For heavy payloads, successful landing rates are higher for Polar, LEO, and ISS orbits. However, in GTO orbit, we cannot clearly distinguish between positive landing rates and negative landings (unsuccessful missions).
- Finally, the overall success rate has been steadily increasing since 2013, reaching its peak in 2020.

Thank you!