Winning Space Race with Data Science

Lei Wu

20240221

Outline

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

Executive Summary

- Summary of methodologies
  - SpaceX Data Collection Using SpaceX API
  - Collecting Data by Web Scraping from Wikipedia
  - Data Wrangling for Suitable Labels Supervised Models
  - EDA Using SQL
  - Data Visualization with Pandas and Matplotlib
  - Launch Sites Analysis with Folium and Ploty Dash
  - Machine Learning for Landing Success Prediction
- Summary of all results
  - EDA Results
  - Interactive Visuals and Dashboards
  - Prediction Results

Introduction

- Project background and context

- SpaceX advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upwards 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. Spaces X's Falcon 9 launch like regular rockets. To help us understand the scale of the Falcon 9, we are going to use these diagrams from Forest Katsch, at zlsadesign.com.

- Problems you want to find answers

- In this capstone, we will predict if the Falcon 9 first stage will land successfully.

Section 1

Methodology

Methodology

- Executive Summary
- Data collection methodology:
  - We will discuss how data was collected
- Perform data wrangling
  - We will 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

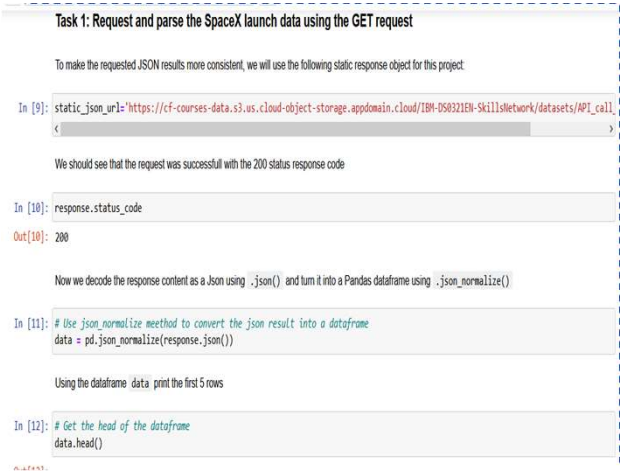
Data Collection

- Describe how data sets were collected.
- The process starts by making a GET request to the SpaceX API to retrieve past launch data. The obtained JSON response is converted into a Pandas dataframe for further processing. Columns relevant to the analysis, such as rocket, payloads, launchpad, and cores, are extracted from the dataframe.
- Subsequently, data cleaning steps are performed to filter out rows with multiple cores or payloads and convert date formats. Additionally, certain columns are filtered based on a specified date range.
- Global variables are initialized to store the extracted data globally. Functions are then applied to populate these variables with relevant information retrieved from the API.
- Finally, a dictionary containing the extracted data is created, which is then converted into a Pandas dataframe. Further data cleaning is performed, including filtering out specific rocket versions (e.g., Falcon 1), handling missing values by replacing them with the mean, and adjusting flight numbers for Falcon 9 rockets.
- The resulting dataframe contains organized data on SpaceX launches, ready for analysis.

Data Collection – SpaceX API

The process starts by making a GET request to the SpaceX API to retrieve past launch data. The obtained JSON response is converted into a Pandas dataframe for further processing. Columns relevant to the analysis, such as rocket, payloads, launchpad, and cores, are extracted from the dataframe.

Add the GitHub URL of the completed SpaceX API calls notebook (https://github.com/Dadaranger/IBM-Final-Project-for-Peer-Review/blob/main/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 [9]: static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_call_
```

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

```
In [10]: response.status_code
Out[10]: 200
```

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

```
In [11]: # Use json_normalize meethod to convert the json result into a dataframe
         data = pd.json_normalize(response.json())
```

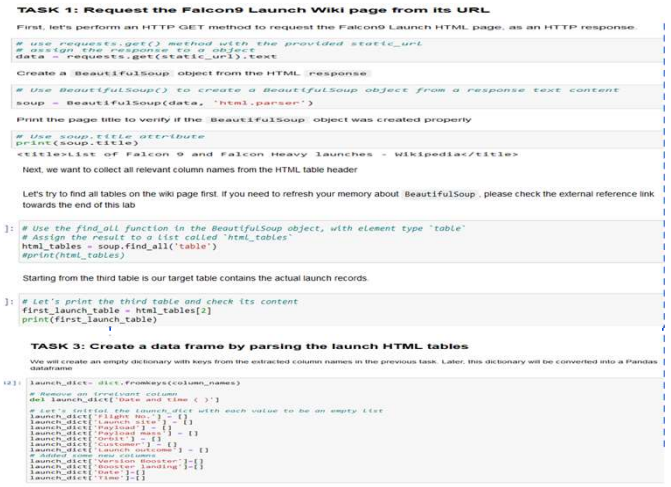
Using the dataframe `data` print the first 5 rows

```
In [12]: # Get the head of the dataframe
         data.head()
```

Data Collection - Scraping

Collected Falcon 9 historical launching records by scraping Wikipedia using BS4 and request. Parsed the data to a Dataframe.

https://github.com/Dadaranger/IBM-Final-Project-for-Peer-Review/blob/main/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.

```
# use requests.get() method with the provided static_url
# assign the response to a object
data = requests.get(static_url).text
```

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

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

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

```
# Use soup.title attribute
print(soup.title)
```

```
<title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

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

```
# 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')
#print(html_tables)
```

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)
```

TASK 3: Create a data frame by parsing the launch HTML tables

We will create an empty dictionary with keys from the extracted column names in the previous task. Later, this dictionary will be converted into a Pandas dataframe

```
launch_dict= dict.fromkeys(column_names)

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

# Let's initial the launch_dict with each value to be an empty list
launch_dict['Flight No.'] = []
launch_dict['Launch site'] = []
launch_dict['Payload'] = []
launch_dict['Payload mass'] = []
launch_dict['Orbit'] = []
launch_dict['Customer'] = []
launch_dict['Launch outcome'] = []
# Added some new columns
launch_dict['Version Booster']=[]
launch_dict['Booster landing']=[]
launch_dict['Date']=[]
launch_dict['Time']=[]
```

Data Wrangling

- Data Loading and Inspection: The dataset is loaded from an online CSV file, and the first 10 rows are displayed. The script checks for missing values and calculates the percentage of missing values in each column. Additionally, it examines the data types of each column.
- Exploring Data Distribution: The script analyzes the distribution of values in the "LaunchSite" and "Orbit" columns using the value_counts() function. It also investigates the frequency of different outcomes in the "Outcome" column.
- Processing Data: Based on observed outcomes, the script identifies "bad outcomes" by selecting specific indices from the outcome values. It then assigns a binary class label (0 or 1) to each outcome: 0 indicates a bad outcome, while 1 indicates otherwise. This information is stored in a new column named "Class".
- Analyzing Data: The script calculates the mean of the "Class" column to determine the overall proportion of successful outcomes in the dataset.
- Saving Processed Dataset: Finally, the processed dataset, including the added "Class" column, is saved to a new CSV file named "dataset_part_2.csv", excluding the index column.

TASK 1: Calculate the number of launches on each site

The data contains several Space X launch facilities: Cape Canaveral Space Launch Complex 40 **VAFB SLC 4E** , Vandenberg Air Force Base Space Launch Complex 4E **(SLC-4E)**, Kennedy Space Center Launch Complex 39A **KSC LC 39A** .The location of each Launch is placed in the column `LaunchSite`

Next, let's see the number of launches for each site.

Use the method `value_counts()` on the column `LaunchSite` to determine the number of launches on each site:

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

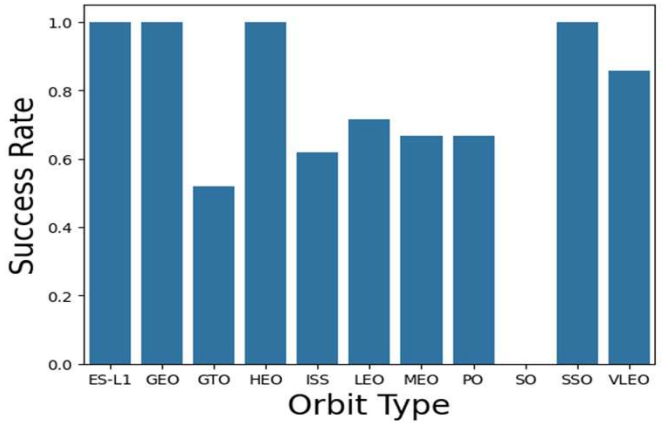
```
CCAFS SLC 40    55
KSC LC 39A      22
VAFB SLC 4E     13
Name: LaunchSite, dtype: int64
```

https://github.com/Dadaranger/IBM-Final-Project-for-Peer-Review/blob/main/3.Spacex-Data%20wrangling.ipynb

EDA with Data Visualization

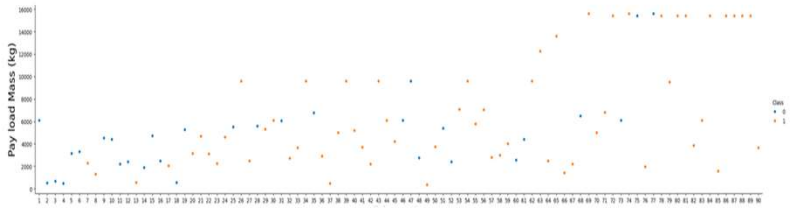
- Utilized feature engineering from Pandas and Matplotlib libraries
- Conducted EDA
- Bar chart for relationship between success and orbit type

- https://github.com/Dadaranger/IBM-Final-Project-for-Peer-Review/blob/main/5.Spacex-eda-dataviz.ipynb.jupyterlite.ipynb



Analyze the ploted bar chart try to find which orbits have high sucess rate.

```
sns.catplot(y="PayloadMass", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Pay load Mass (kg)",fontsize=20)
plt.show()
```



Task 7

List the total number of successful and failure mission outcomes

```
q = pd.read_sql("select substr(Mission_Outcome, 1, 7) as Mission_Outcome, count(*) from SPACEXTABLE group by 1", con)
q
```

	Mission_Outcome	count(*)
0	Failure	1
1	Success	100

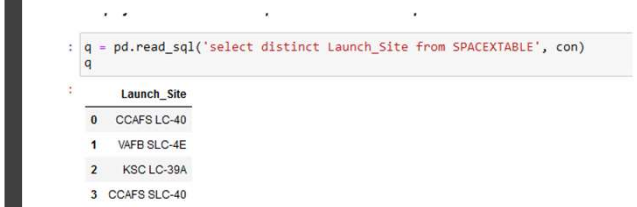
```
q = pd.read_sql("SELECT COUNT(Mission_Outcome) AS SuccessOutcome FROM SPACEXTABLE WHERE Mission_Outcome LIKE 'Success%'", co
q
```

	SuccessOutcome
0	100

Task 8

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

```
q = pd.read_sql("select distinct Booster_Version from SPACEXTABLE where PAYLOAD_MASS__KG_ = (select max(PAYLOAD_MASS__KG_) f
q
```



```
q = pd.read_sql('select distinct Launch_Site from SPACEXTABLE', con)
q
```

	Launch_Site
0	CCAFS LC-40
1	VAFB SLC-4E
2	KSC LC-39A
3	CCAFS SLC-40

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

```
q = pd.read_sql("select * from SPACEXTABLE where Launch_Site like 'CCA%' limit 5", con)
q
```

Task 3

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

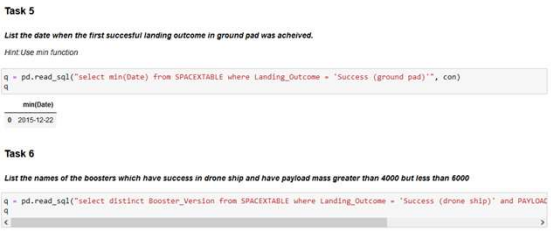
```
q = pd.read_sql("select sum(PAYLOAD_MASS__KG_) from SPACEXTABLE where Customer = 'NASA (CRS)' ", con)
q
```

	sum(PAYLOAD_MASS__KG_)
0	45596

Task 4

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

```
q = pd.read_sql("select avg(PAYLOAD_MASS__KG_) from SPACEXTABLE where Booster_Version = 'F9 v1.1'", con)
q
```



Task 5

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

Hint Use min function

```
q = pd.read_sql("select min(Date) from SPACEXTABLE where Landing_Outcome = 'Success (ground pad)'", con)
q
```

	min(Date)
0	2015-12-22

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

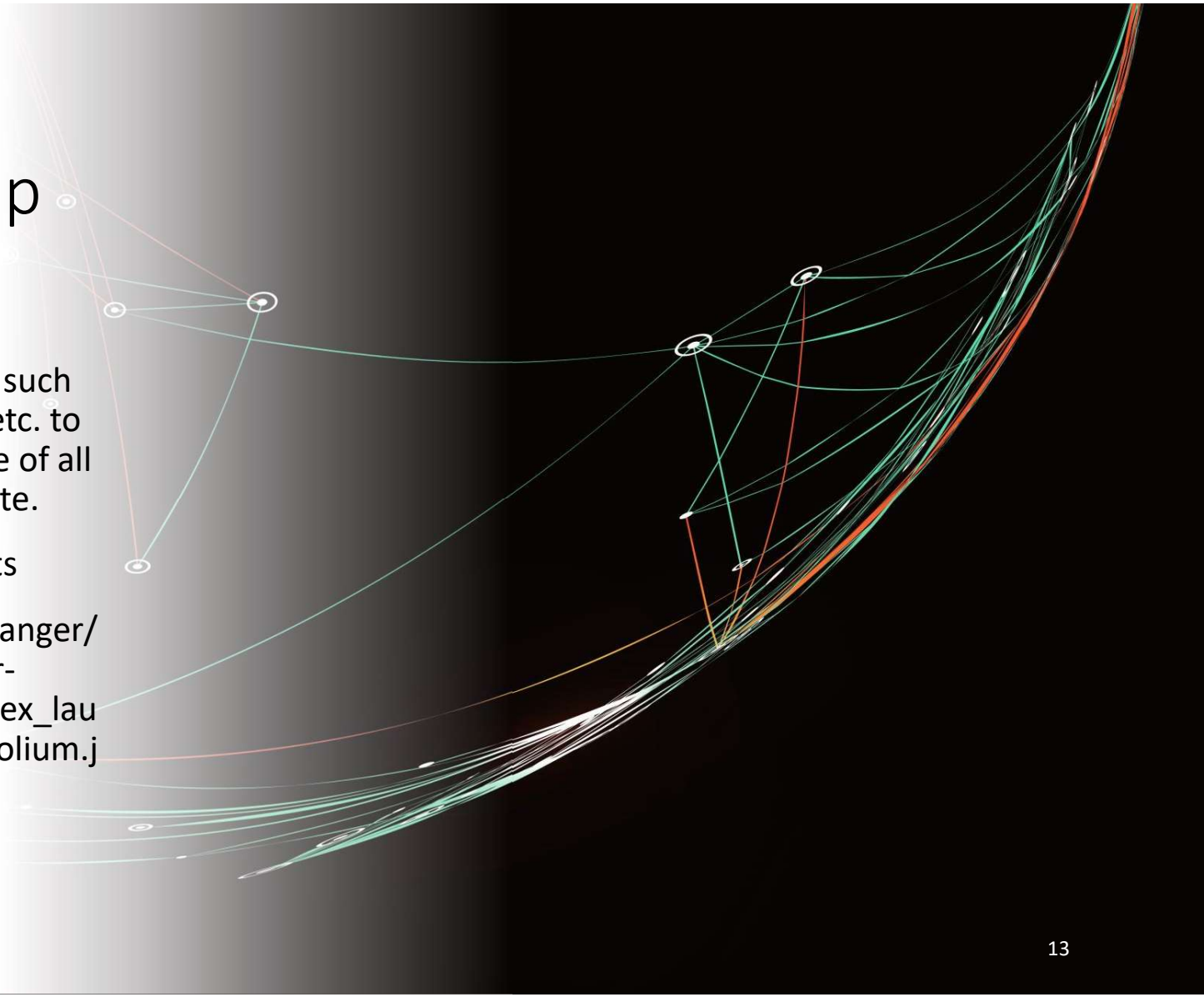
```
q = pd.read_sql("select distinct Booster_Version from SPACEXTABLE where Landing_Outcome = 'Success (drone ship)' and PAYLOAD
q
```

EDA with SQL

- https://github.com/Dadaranger/IBM-Final-Project-for-Peer-Review/blob/main/4.Spacex-eda-sql-coursera_sqllite.ipynb

Build an Interactive Map with Folium

- Using Folium map objects such as markers, circles, lines, etc. to mark the success or failure of all launches at each launch site.
- Per requirements and hints
- https://github.com/Dadaranger/IBM-Final-Project-for-Peer-Review/blob/main/6.Spacex_launch_site_location_with_Folium.jupyterlite.ipynb

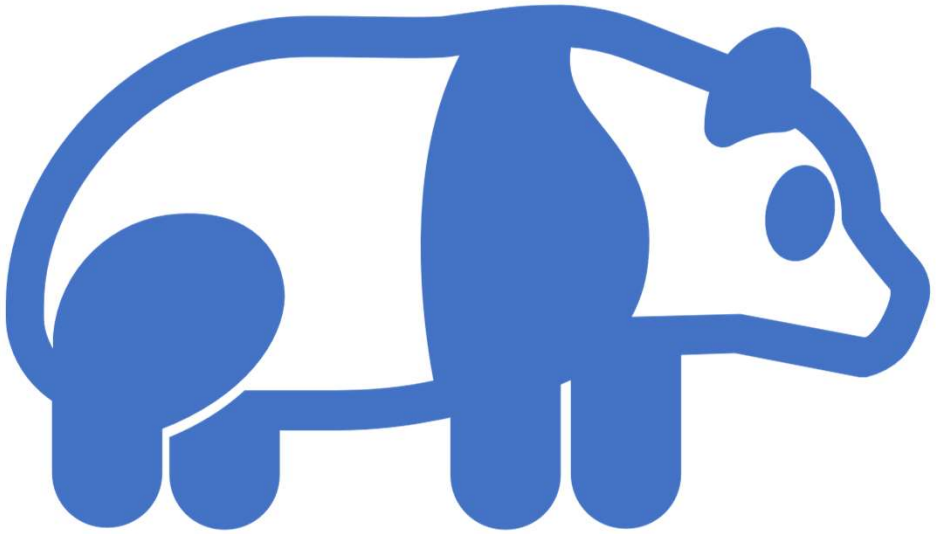


Build a Dashboard with Plotly Dash

- Performed the following actions:
  - Added Launch Site drop-down component.
  - Added callback function to render the pie-chart per each site drop-down.
  - Added slider to select Payload
  - Added callback function to render the payload to success scatter plot.
- https://github.com/Dadaranger/IBM-Final-Project-for-Peer-Review/blob/main/7.Spacex_Interactive_Dashboard_App.py

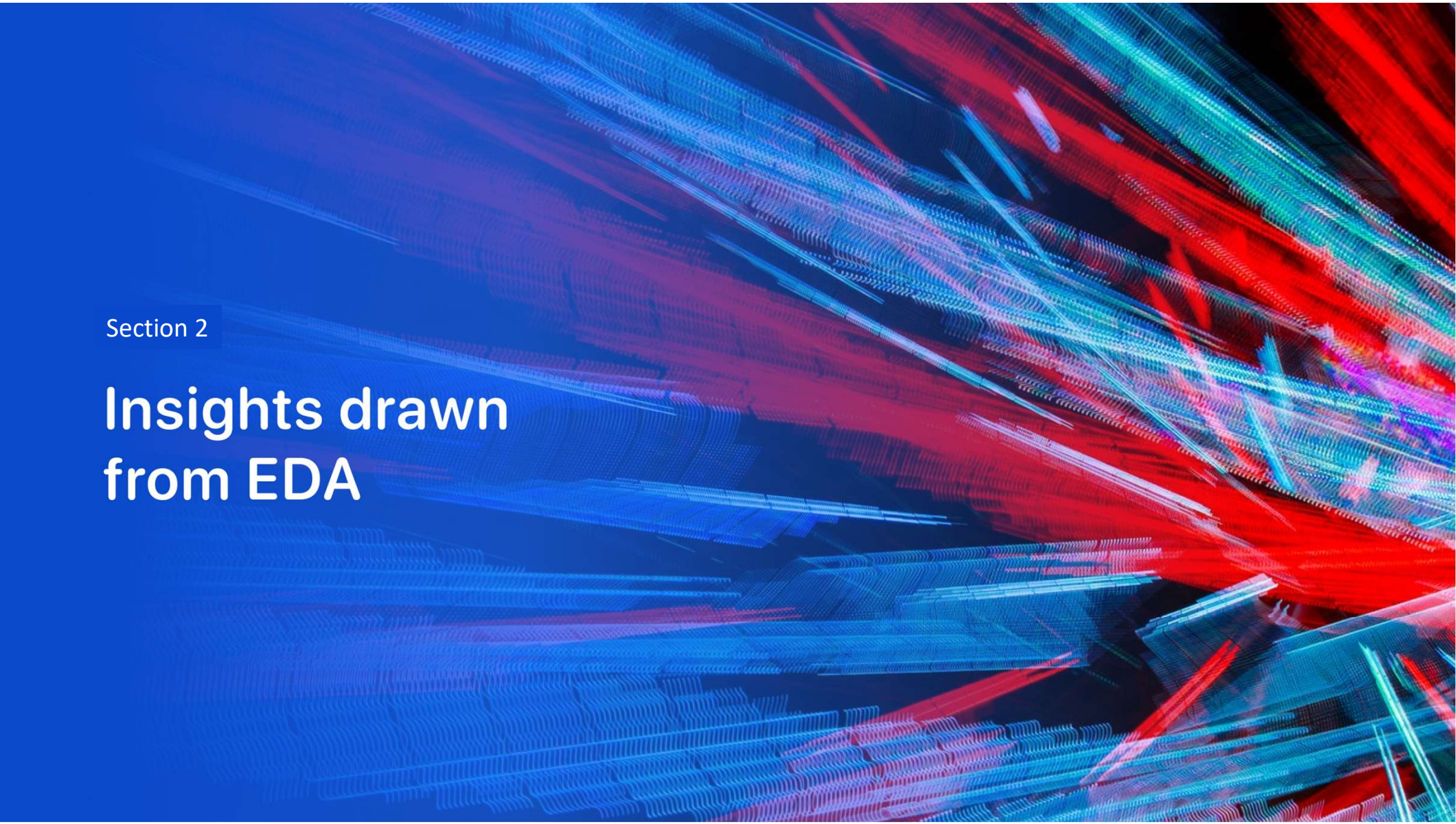
Predictive Analysis (Classification)

- Importing Libraries: The script imports essential libraries such as Pandas, NumPy, Matplotlib, Seaborn, and scikit-learn modules for data manipulation, visualization, and machine learning algorithms.
- Fetching Data: Data is retrieved from an online source using the requests module. The script fetches data from two URLs and converts them into Pandas DataFrames for further processing.
- Data Preprocessing: The fetched data is preprocessed using standard scaling to standardize features.
- Data Splitting: The preprocessed data is split into training and testing sets using the train_test_split function from scikit-learn. This step is crucial for evaluating the performance of machine learning models.
- Model Training and Evaluation: Logistic Regression is chosen as the classification algorithm. GridSearchCV is employed to perform hyperparameter tuning, which helps in finding the optimal parameters for the logistic regression model. The best parameters and accuracy scores are displayed.
- Model Performance Evaluation: The accuracy of the model on the test data is calculated and displayed. Additionally, a confusion matrix is plotted to visualize the performance of the model in predicting outcomes.
- https://github.com/Dadaranger/IBM-Final-Project-for-Peer-Review/blob/main/8.Spacex_Mashine_Learning.ipynb



Results

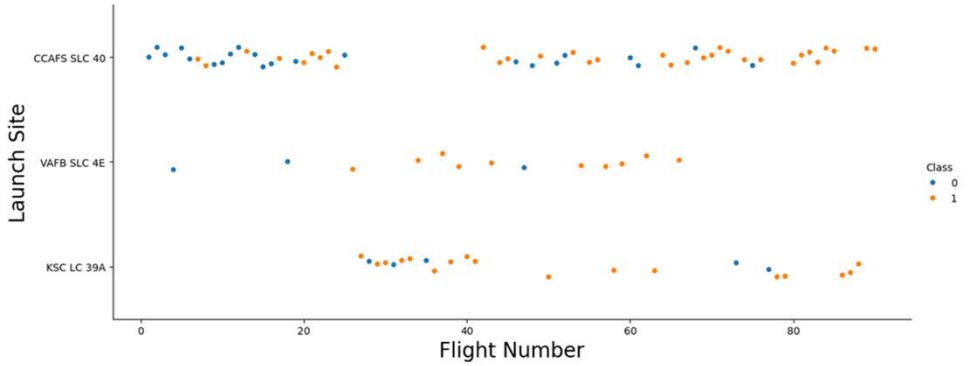
- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results



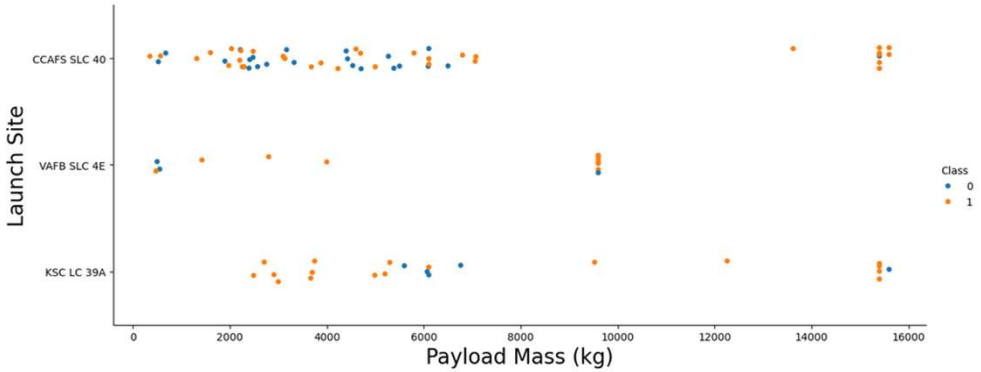
Section 2

Insights drawn from EDA

Flight Number vs. Launch Site

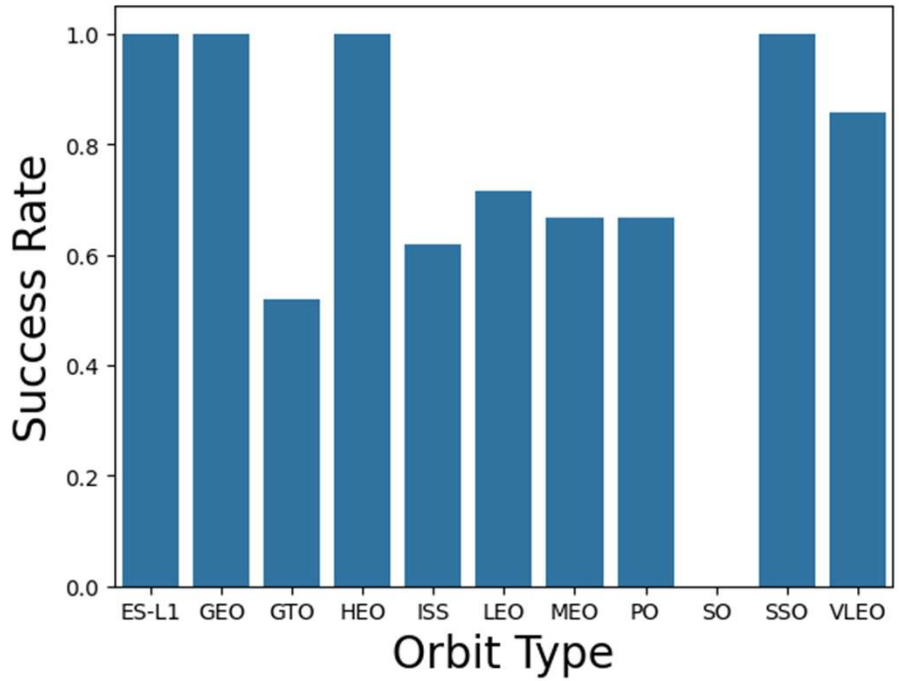


Payload vs. Launch Site



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).

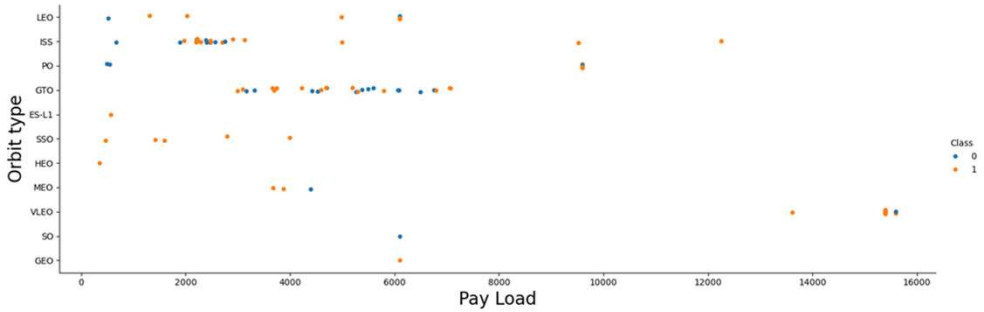
Success Rate vs. Orbit Type



Analyze the ploted bar chart try to find which orbits have high sucess rate.

Flight Number vs. Orbit Type

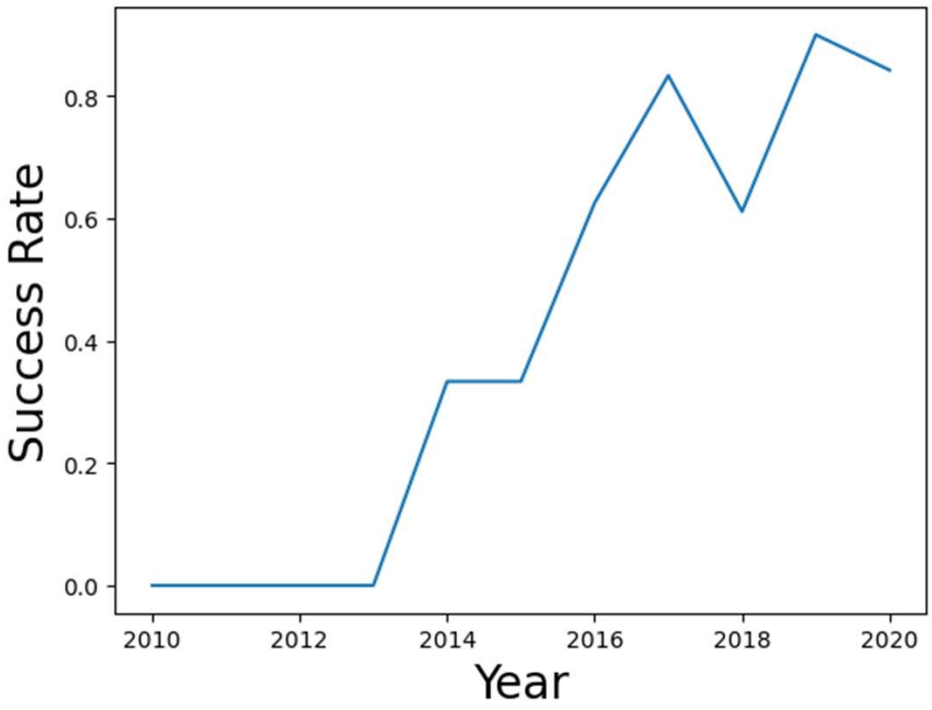
Payload vs. Orbit Type



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 sucess rate since 2013 kept increasing till 2020

All Launch Site Names

```
q = pd.read_sql('select distinct Launch_Site from SPACEXTABLE', con)
q
```

	Launch_Site
0	CCAFS LC-40
1	VAFB SLC-4E
2	KSC LC-39A
3	CCAFS SLC-40

Launch Site Names Begin with 'CCA'

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

```
q = pd.read_sql("select * from SPACEXTABLE where Launch_Site like 'CCA%' limit 5", con)
q
```

	Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
0	2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
1	2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of...	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2	2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
3	2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
4	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)

```
q = pd.read_sql("select sum(PAYLOAD_MASS__KG_) from SPACEXTABLE where Customer = 'NASA (CRS)' ", con)
q
```

	sum(PAYLOAD_MASS__KG_)
0	45596

Average Payload Mass by F9 v1.1

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

```
q = pd.read_sql("select avg(PAYLOAD_MASS__KG_) from SPACEXTABLE where Booster_Version = 'F9 v1.1'", con)
q
```

	avg(PAYLOAD_MASS__KG_)
0	2928.4

First Successful Ground Landing Date

```
q = pd.read_sql("select min(Date) from SPACEXTABLE where Landing_Outcome = 'Success (ground pad)'", con)
q
```

	min(Date)
0	2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

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

```
: q = pd.read_sql("select distinct Booster_Version from SPACEXTABLE where Landing_Outcome =
                  'Success (drone ship)' and PAYLOAD_MASS__KG_ between 4000 and 6000", con)
  q
```

:

	Booster_Version
0	F9 FT B1022
1	F9 FT B1026
2	F9 FT B1021.2
3	F9 FT B1031.2

Total Number of Successful and Failure Mission Outcomes

List the total number of successful and failure mission outcomes

```
q = pd.read_sql("select substr(Mission_Outcome, 1, 7) as Mission_Outcome, count(*) from SPACEXTABLE group by 1", con)
q
```

	Mission_Outcome	count(*)
0	Failure	1
1	Success	100

Boosters Carried Maximum Payload

Task 8

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

```
]: q = pd.read_sql("select distinct Booster_Version from SPACEXTABLE where PAYLOAD_MASS__KG_ =
                    (select max(PAYLOAD_MASS__KG_) from SPACEXTABLE)", con)
   q
```

]:

	Booster_Version
0	F9 B5 B1048.4
1	F9 B5 B1049.4
2	F9 B5 B1051.3
3	F9 B5 B1056.4
4	F9 B5 B1048.5
5	F9 B5 B1051.4
6	F9 B5 B1049.5
7	F9 B5 B1060.2
8	F9 B5 B1058.3
9	F9 B5 B1051.6
10	F9 B5 B1060.3

2015 Launch Records

Note: SQLLite does not support monthnames. So you need to use substr(Date, 6,2) as month to get the months and substr(Date,0,5)='2015' for year.

```
]: q = pd.read_sql("select distinct Landing_Outcome, Booster_Version, Launch_Site from SPACEXTABLE
                    where Landing_Outcome ='Failure (drone ship)' and Date Between '2015-01-01' and '2015-12-31'", con)
   q
```

]:

	Landing_Outcome	Booster_Version	Launch_Site
0	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
1	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

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

```
]: q = pd.read_sql("select Landing_Outcome, count(*) from SPACEXTABLE
                    where Date between '2011-06-04' and '2017-03-20' group by Landing_Outcome order by 2 desc", con)
   q
```

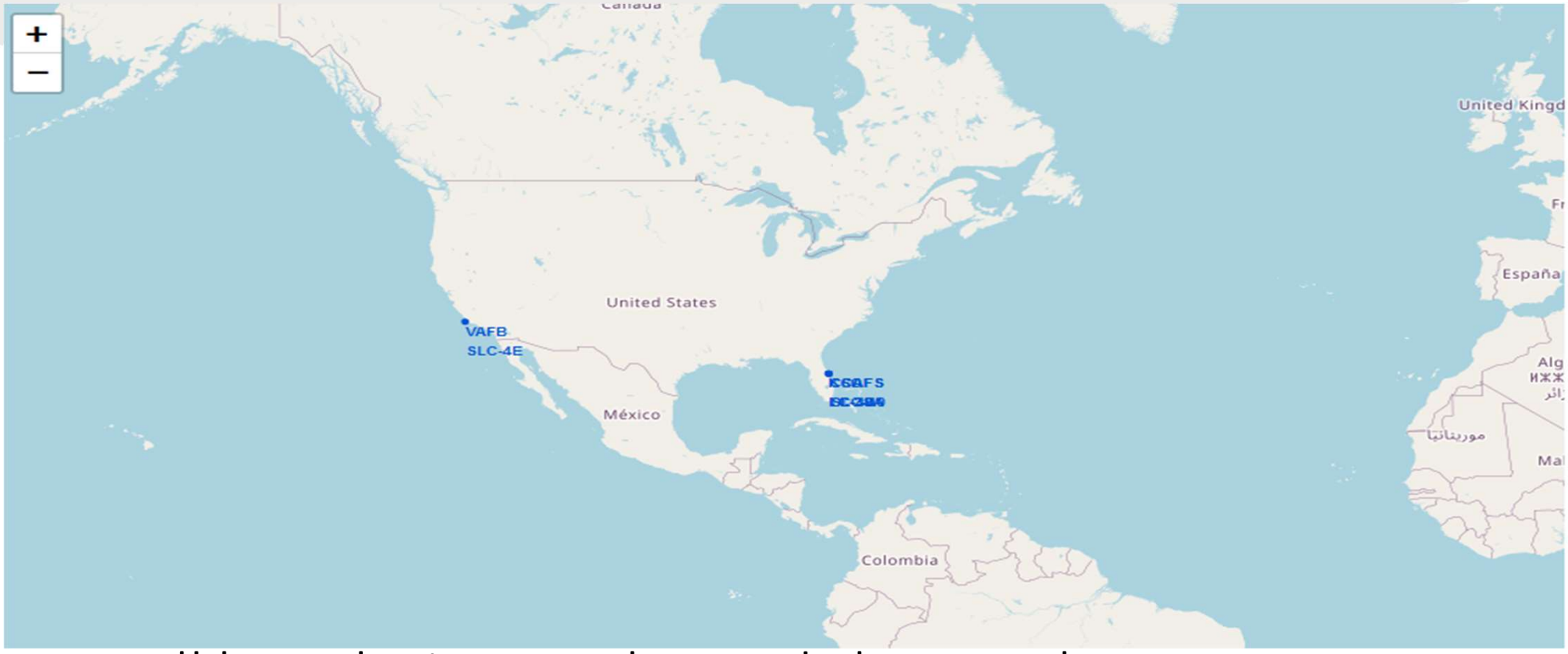
]:

	Landing_Outcome	count(*)
0	No attempt	10
1	Success (drone ship)	5
2	Failure (drone ship)	5
3	Success (ground pad)	3
4	Controlled (ocean)	3
5	Uncontrolled (ocean)	2
6	Precluded (drone ship)	1

Section 3

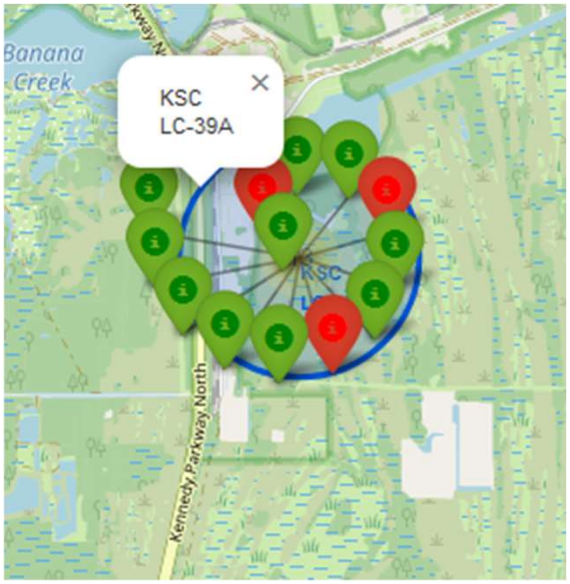
Launch Sites Proximities Analysis

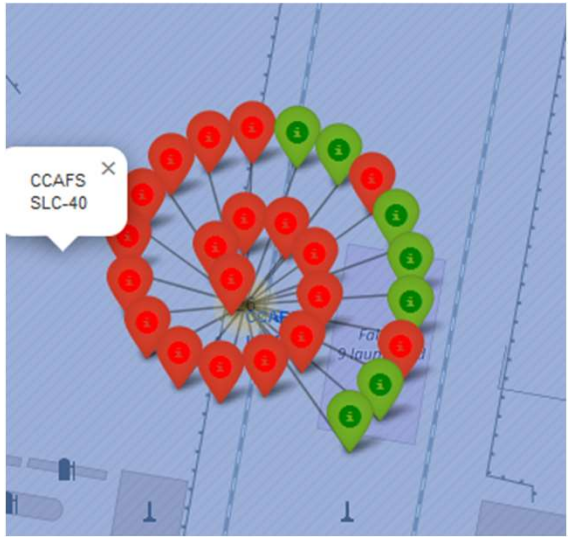
Marker of Launch Sites

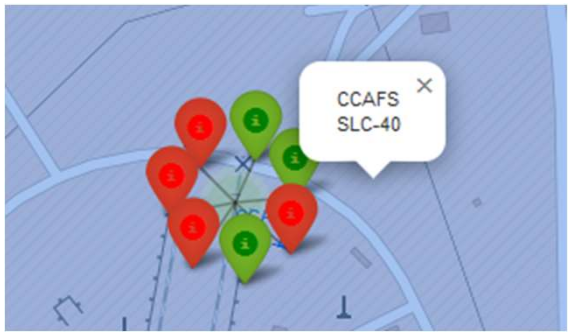


All launch sites are located close to the coast

Florida launching Sites

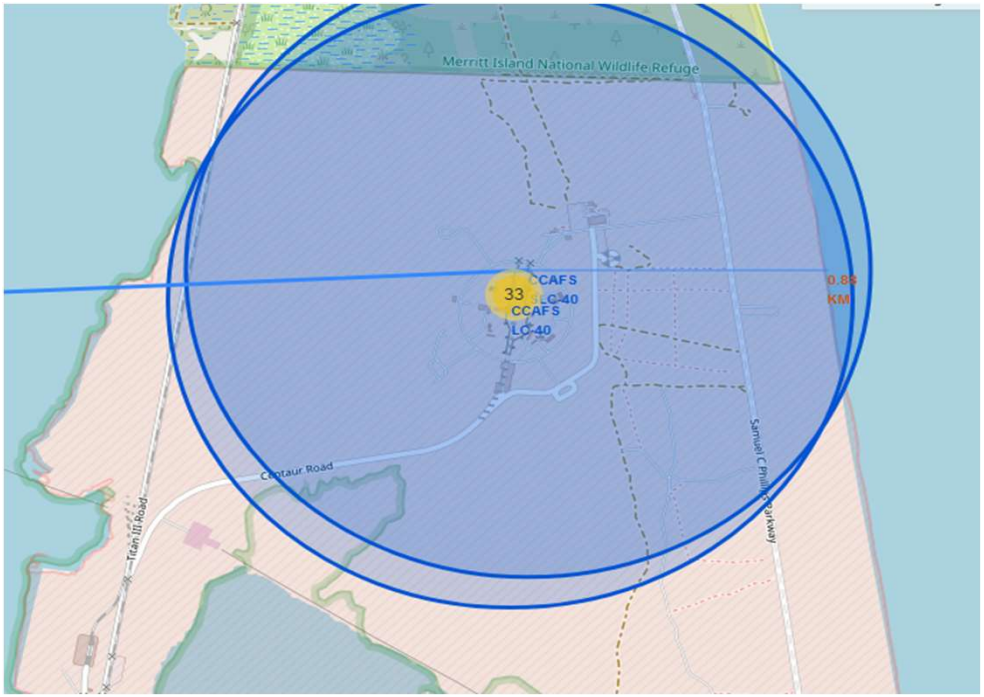




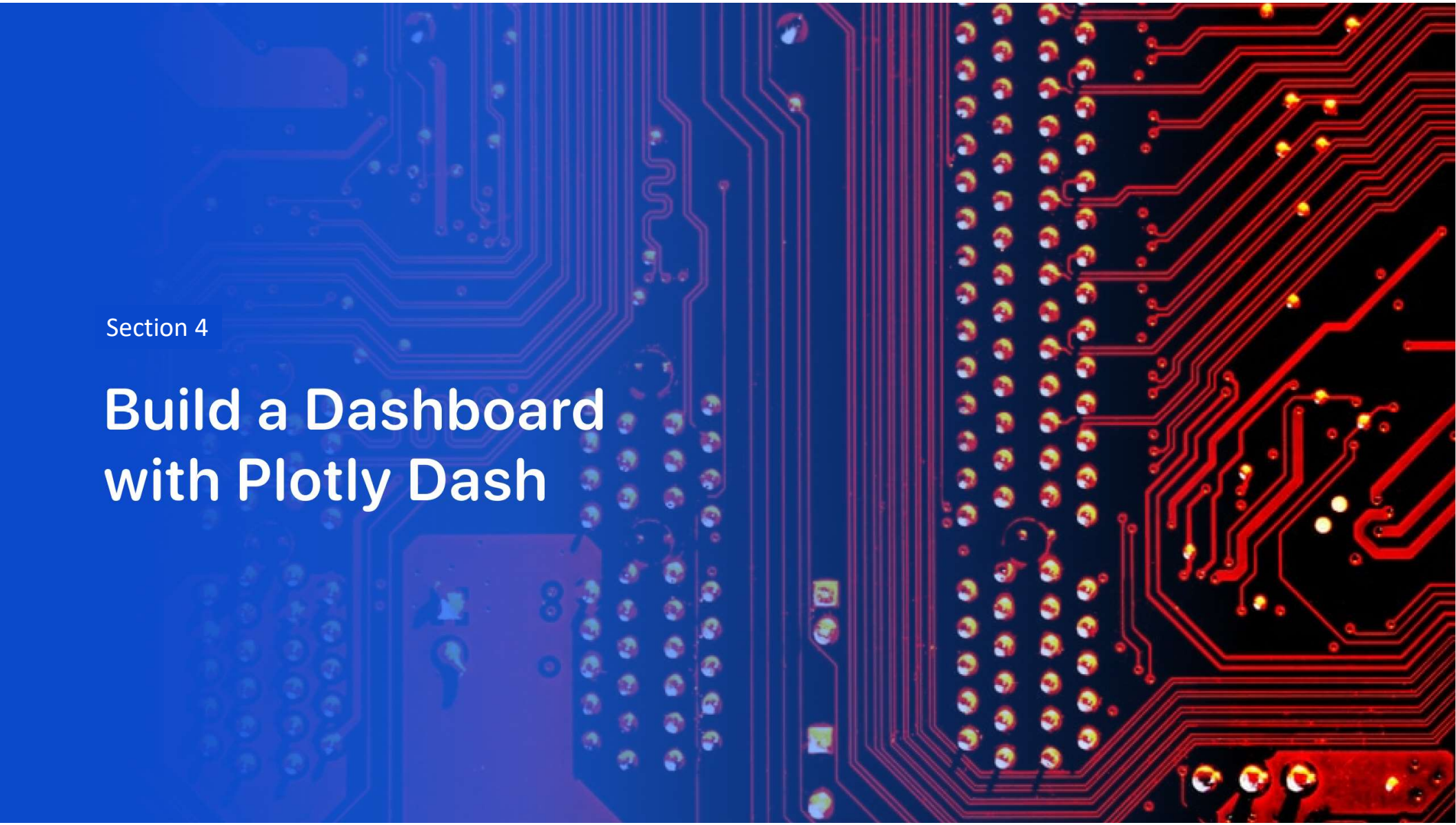


KSC LC-39A has higher success rate comparing to other two sites.

CCAFSLC-40 Launching Site



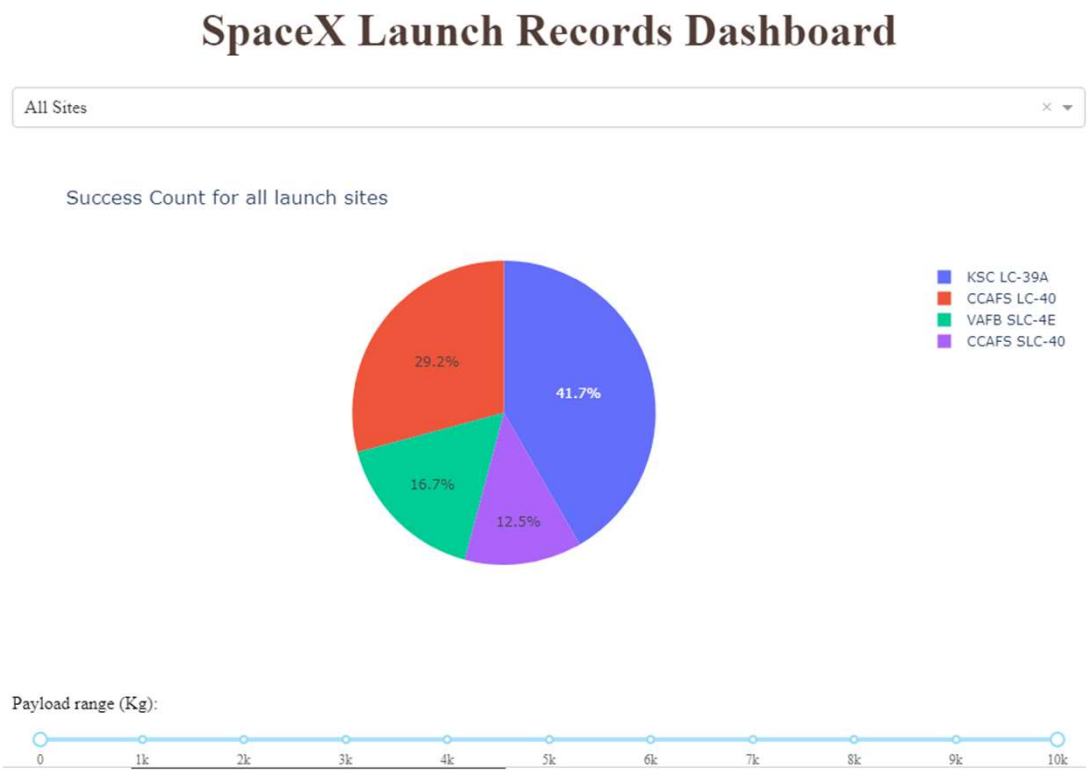
CCAFSLC-40 Launching Site proximately 0.86km to the coastline.



Section 4

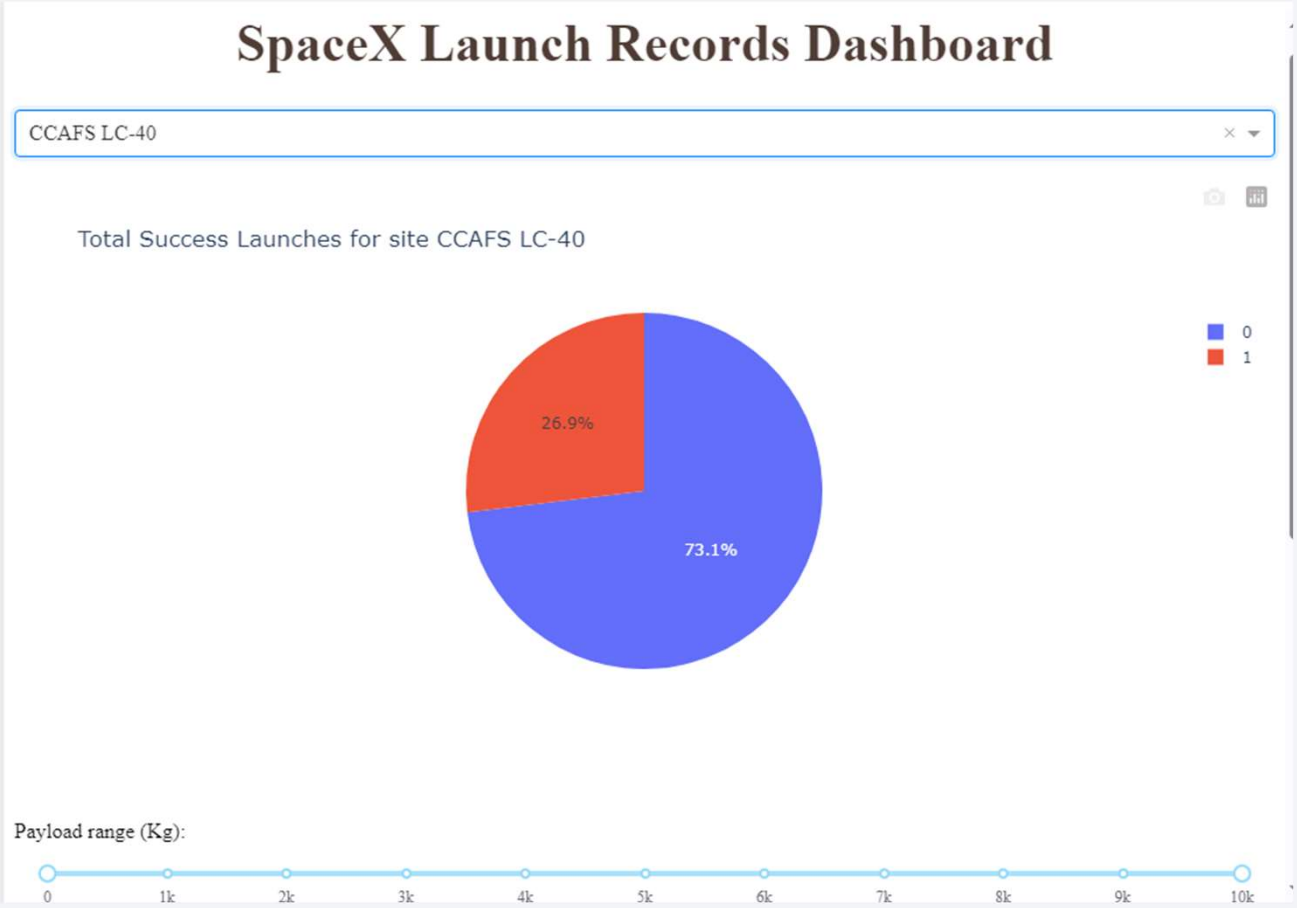
Build a Dashboard with Plotly Dash

Pie Chart for Launching Success Counts



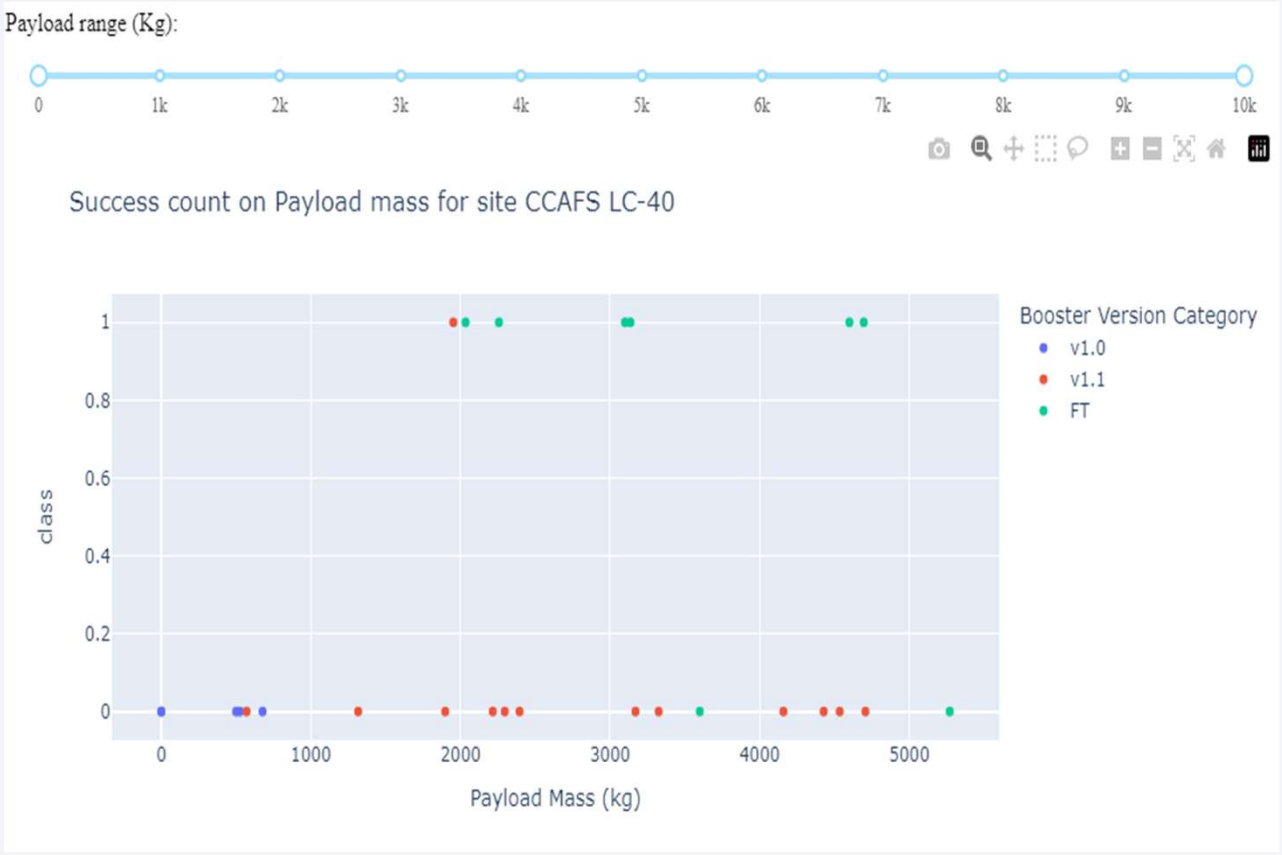
All sites in the chart

Highest Launch Site



CCAFS LC-40 with highest ratio of 73.1%

Success Count on Payload Mass for All Sites



Booster Verstion FT has the highest success rate from mass over 2000kg

Section 5

Predictive Analysis (Classification)

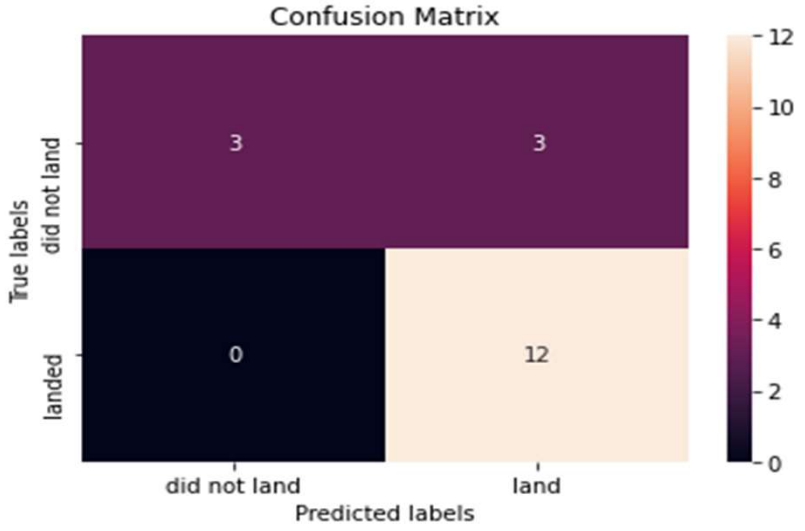
Classification Accuracy

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

They all performed the same.

Confusion Matrix

```
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
plt.show()
```



All 4 different classifcations models generated the same confusion matrixes which also reflected on the previous slides that all have the same accuracy

Conclusions

- The success rates vary across different launch sites, with CCAFS LC-40 demonstrating the highest success rate at 77%.
- As the flight numbers increase across the three launch sites, the success rates also tend to rise. Notably, in Low Earth Orbit (LEO), the success rate seems to correlate with the number of flights, while in Geostationary Transfer Orbit (GTO), there appears to be no discernible relationship between flight number and success rate. However, it's worth noting that overall success rates have been consistently increasing since 2013.

Thank you!