IBM Developer
SKILLS NETWORK

IBM DATA SCIENCE CAPSTONE PROJECT

Winning Space Race with Data Science

- Space X Falcon 9 Landing Analysis

Balkiss BARAGH

13/05/2024

Outline

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

Executive Summary

❑ Summary of methodologies

- Data Collection using web scraping and SpaceX API;
- Data Wrangling
- Exploratory Data Analysis (EDA)
- Data visualization and interactive visual analytics;
- Predictive analysis (classification).

❑ Summary of all results

- Exploratory Data Analysis (EDA) results
- Geospatial analytics
- Interactive dashboard
- Predictive analysis using classification models

Introduction

Project Background and Context

This project centers on SpaceX, the renowned aerospace company established by Elon Musk. Our objective is to examine data pertaining to SpaceX launches, investigating success rates, payload trends, rocket performance, landing results, launch site selections, and temporal patterns.

Problems

Launch Success: Which factors influence the success of SpaceX launches?

Payload Analysis: How does payload mass impact the successful landing across different orbits?

Rocket Performance: What insights can be gained about the reliability and reusability of SpaceX rockets?

Landing Outcomes: What patterns emerge in the results of first stage landings?

Launch Site Assessment: How do the choices of launch sites affect mission success?

Temporal Trends: Are there significant trends or seasonal patterns in SpaceX's launch history?

Section 1

Methodology

Methodology

Executive Summary

1. Data collection methodology:
   - Using the SpaceX REST API: https://api.spacexdata.com/v4/rockets/
   - Web Scraping: https://en.wikipedia.org/wiki/List_of_Falcon_9_and_Falcon_Heavy_launches
2. Perform data wrangling
   - Remove missing values using the .fillna()
   - Using the .value_counts() method to determine the following:

     a - Number of launches on each site ; b - Number and occurrence of each orbit

     and c - Number and occurrence of mission outcome per orbit type
   - Determining labels for training the supervised classification as follows:

     '0' for unsuccessful landing and '1' for successful landing

Methodology

Executive Summary

3. Perform exploratory data analysis (EDA) using visualization and SQL

   - Using SQL queries in order to manage and evaluate the SpaceX dataset
   - Using Pandas and Matplotlib to prepare visualizations showing relationships between all the studied variables, and thus helps in determining patterns

4. Perform interactive visual analytics using Folium and Plotly Dash

   - Folium was used to get geospatial analytics
   - Plotly Dash used to create interactive dashboard

Methodology

Executive Summary

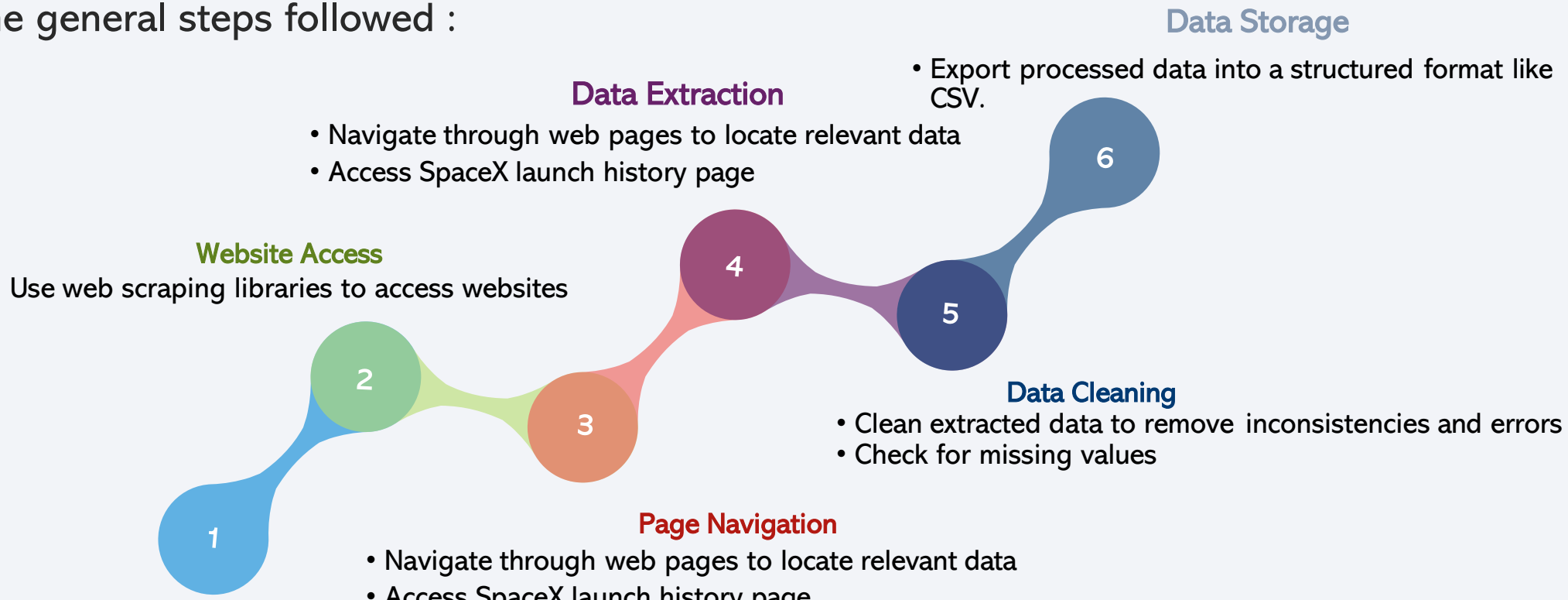
5. Perform predictive analysis using classification models

- Pre-process (standardize) the data using StandardScaler()
- Split the data into training and testing data using train_test_split
- Train different classification models (logistic regression, support vector machine, Decision Tree and KNN)
- Find hyperparameters using GridSearchCV
- Plotting confusion matrices for each classification model
- Assessing the accuracy of each classification model

Data Collection

- Data Collection was done through two methods: SPACE X REST API and Web Scraping

These are the general steps followed :

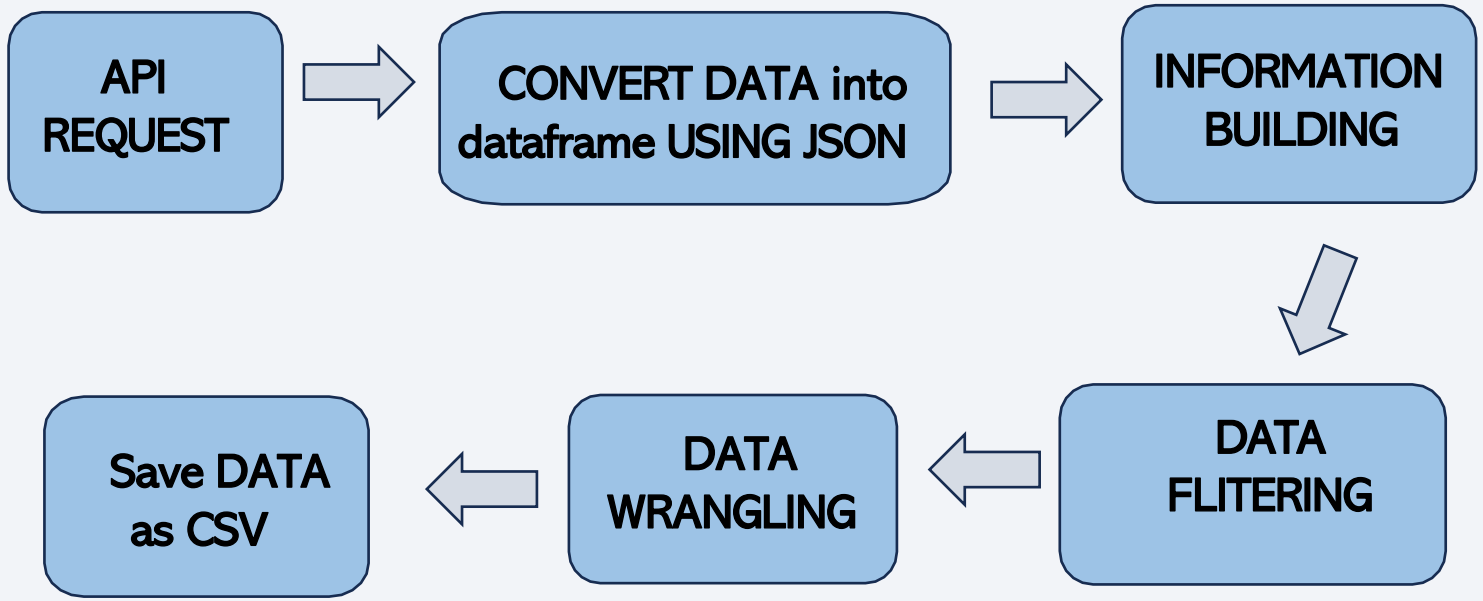


I'll go through the details of each one of them !

Data Collection – SpaceX API

- **API Request:** Request launch data from SpaceX API using the GET request.
- **Convert data into dataframe:** Use json_normalize meethod to convert the json result into a dataframe
- **Data Cleanup:** Clean the obtained data to remove missing values from PlayloadMass (but missing values for LandingPad remained).
- **Extracted Information :**

```
FlightNumber
Date
BoosterVersion
PayloadMass
Orbit
LaunchSite
Outcome
Flights
GridFins
Reused
Legs
LandingPad
Block
ReusedCount
Serial
Longitude
Latitude
```



- **Data Filtering:** Retained only Falcon 9 launches.
- **Data Wrangling:** Imputed missing Payload Mass values with the mean.
- **Export data to CSV**

Github - Link

Data Collection – SpaceX API

1.API Request and read response into DF | 2. Information building | 3. Call helper functions to get relevant data | 4. Construct data using dictionary | 5. Convert Dict to Dataframe and filter for Falcon9 launches | 6. DATA WRANGLING and sabe data to CSV

1. API request to launch data from SpaceX API, normalize data and convert it to dataframe using json :

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

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

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

2. Choose global variable lists that Will be stored in the final dataset for later use

```
#Global variables
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
ReusedCount = []
Serial = []
Longitude = []
Latitude = []
```

3. Call helper functions to get relevant data (see appendix)
   - getBoosterVersion(data)
   - getLaunchSite(data)
   - getPayloadData(data)
   - getCoreData(data)

4. Construct dataset from received data and then combine columns into a dictionary:

```
launch_dict = {'FlightNumber': list(data['flight_number']),
'Date': list(data['date']),
'BoosterVersion':BoosterVersion,
'PayloadMass':PayloadMass,
'Orbit':Orbit,
'LaunchSite':LaunchSite,
'Outcome':Outcome,
'Flights':Flights,
'GridFins':GridFins,
'Reused':Reused,
'Legs':Legs,
'LandingPad':LandingPad,
'Block':Block,
'ReusedCount':ReusedCount,
'Serial':Serial,
'Longitude': Longitude,
'Latitude': Latitude}
```

5. Create Dataframe from dictionary and filter it to keep only the Falcon9 launches:

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

```
# Hint data['BoosterVersion']!='Falcon 1'
# Since in the BoosterVersion we have ['Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 9']
# so we will drop Falcon 1 to keep only the desired one
data_falcon9 = df_launch[df_launch['BoosterVersion']!= 'Falcon 1']
```

6. DATA Wrangling and save the final dataframe as CSV:

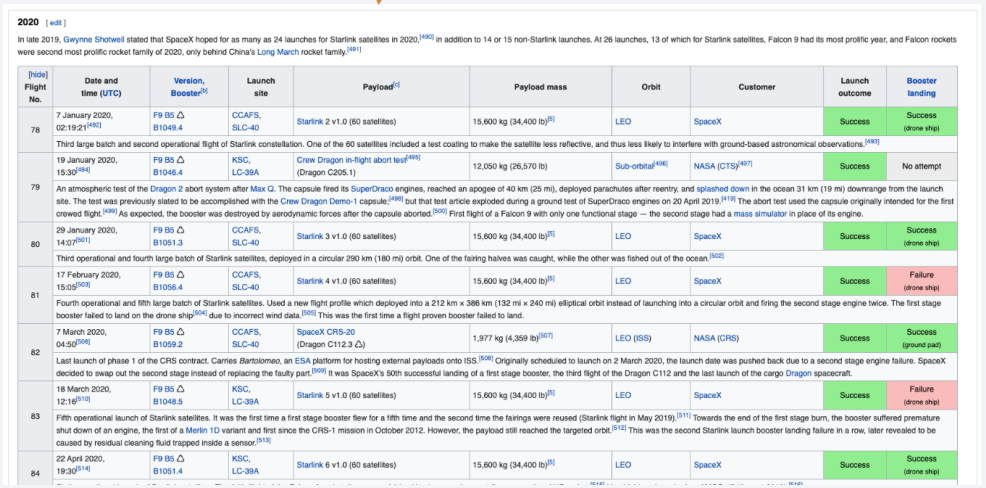
```
# Calculate the mean value of PayloadMass column
payload_mass_mean = data_falcon9['PayloadMass'].mean()

# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'].replace(np.nan, payload_mass_mean, inplace = True)
data_falcon9.isnull().sum()
```

```
data_falcon9.to_csv('dataset_part_1.csv', index=False)
```

Data Collection - Scraping

- **Objective:** Web scrape Falcon 9 launch records from Wikipedia link.

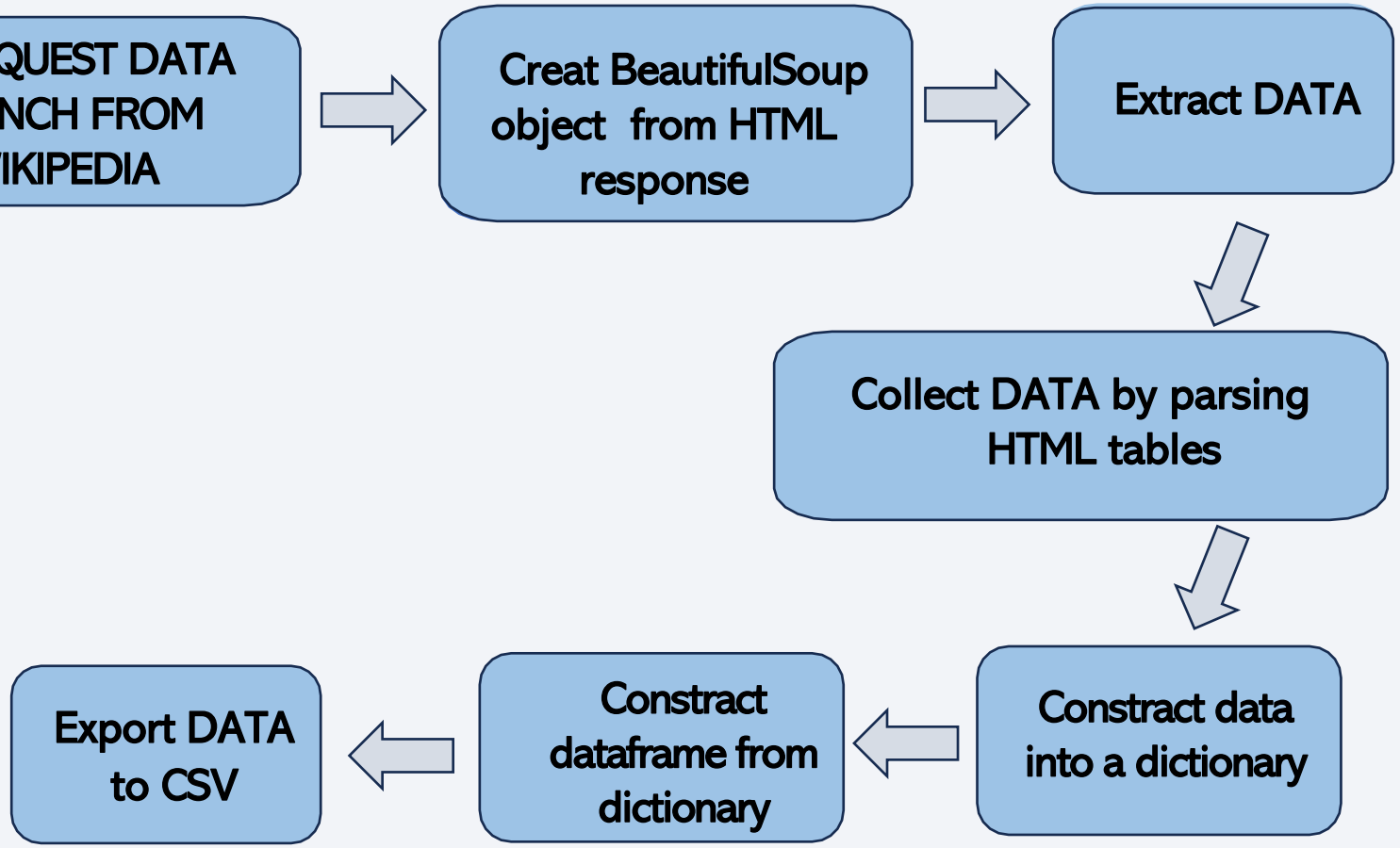


2020 [edit]

In late 2019, Gwynne Shotwell stated that SpaceX hoped for as many as 24 launches for Starlink satellites in 2020,[490] in addition to 14 or 15 non-Starlink launches. At 26 launches, 13 of which for Starlink satellites, Falcon 9 had its most prolific year, and Falcon rockets were second most prolific rocket family of 2020, only behind China's Long March rocket family.[491]

[hide] Flight No.	Date and time (UTC)	Version, Booster[N]	Launch site	Payload[c]	Payload mass	Orbit	Customer	Launch outcome	Booster landing
78	7 January 2020, 02:19:21[490]	F9 B5 △ B1049.4	CCAFS, SLC-40	Starlink 2 v1.0 (60 satellites)	15,600 kg (34,400 lb)[5]	LEO	SpaceX	Success	Success (drone ship)
	Third large batch and second operational flight of Starlink constellation. One of the 60 satellites included a test coating to make the satellite less reflective, and thus less likely to interfere with ground-based astronomical observations.[493]								
79	19 January 2020, 15:30[494]	F9 B5 △ B1046.4	KSC, LC-39A	Crew Dragon in-flight abort test[495] (Dragon C205.1)	12,050 kg (26,570 lb)	Sub-orbital[496]	NASA (CTS)[497]	Success	No attempt
	An atmospheric test of the Dragon 2 abort system after Max Q. The capsule fired its SuperDraco engines, reached an apogee of 40 km (25 mi), deployed parachutes after reentry, and splashed down in the ocean 31 km (19 mi) downrange from the launch site. The test was previously slated to be accomplished with the Crew Dragon Demo-1 capsule,[498] but that test article exploded during a ground test of SuperDraco engines on 20 April 2019.[419] The abort test used the capsule originally intended for the first crewed flight.[499] As expected, the booster was destroyed by aerodynamic forces after the capsule aborted.[500] First flight of a Falcon 9 with only one functional stage — the second stage had a mass simulator in place of its engine.								
80	29 January 2020, 14:07[501]	F9 B5 △ B1051.3	CCAFS, SLC-40	Starlink 3 v1.0 (60 satellites)	15,600 kg (34,400 lb)[5]	LEO	SpaceX	Success	Success (drone ship)
	Third operational and fourth large batch of Starlink satellites, deployed in a circular 290 km (180 mi) orbit. One of the fairing halves was caught, while the other was fished out of the ocean.[502]								
81	17 February 2020, 15:05[503]	F9 B5 △ B1056.4	CCAFS, SLC-40	Starlink 4 v1.0 (60 satellites)	15,600 kg (34,400 lb)[5]	LEO	SpaceX	Success	Failure (drone ship)
	Fourth operational and fifth large batch of Starlink satellites. Used a new flight profile which deployed into a 212 km × 386 km (132 mi × 240 mi) elliptical orbit instead of launching into a circular orbit and firing the second stage engine twice. The first stage booster failed to land on the drone ship[504] due to incorrect wind data.[505] This was the first time a flight proven booster failed to land.								
82	7 March 2020, 04:50[506]	F9 B5 △ B1059.2	CCAFS, SLC-40	SpaceX CRS-20 (Dragon C112.3 △)	1,977 kg (4,359 lb)[507]	LEO (ISS)	NASA (CRS)	Success	Success (ground pad)
	Last launch of phase 1 of the CRS contract. Carries *Bartolomeo*, an ESA platform for hosting external payloads onto ISS.[508] Originally scheduled to launch on 2 March 2020, the launch date was pushed back due to a second stage engine failure. SpaceX decided to swap out the second stage instead of replacing the faulty part.[509] It was SpaceX's 50th successful landing of a first stage booster, the third flight of the Dragon C112 and the last launch of the cargo Dragon spacecraft.								
83	18 March 2020, 12:16[510]	F9 B5 △ B1048.5	KSC, LC-39A	Starlink 5 v1.0 (60 satellites)	15,600 kg (34,400 lb)[5]	LEO	SpaceX	Success	Failure (drone ship)
	Fifth operational launch of Starlink satellites. It was the first time a first stage booster flew for a fifth time and the second time the fairings were reused (Starlink flight in May 2019).[511] Towards the end of the first stage burn, the booster suffered premature shut down of an engine, the first of a Merlin 1D variant and first since the CRS-1 mission in October 2012. However, the payload still reached the targeted orbit.[512] This was the second Starlink launch booster landing failure in a row, later revealed to be caused by residual cleaning fluid trapped inside a sensor.[513]								
84	22 April 2020, 19:30[514]	F9 B5 △ B1051.4	KSC, LC-39A	Starlink 6 v1.0 (60 satellites)	15,600 kg (34,400 lb)[5]	LEO	SpaceX	Success	Success (drone ship)

- **Tools Used:** BeautifulSoup, Requests, Pandas.



Github - Link

Data Collection - Scraping

1. Perform HTTP GET to request HTML page
2. Create Beautiful Soap object
3. Extract all variable names from HTML table header
4. Create Dictionary with keys from extracted variable names
5. Call helper functions to store launch records in the created dictionary
6. Convert Dictionary to Dataframe and sabe it to CSV

1. Create API GET method to request Falcon9 launch HTML page

```
static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"
```

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

2. Create Beautiful Soap object

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

3. Find all the tables on the Wiki page and extract relevant variable names from the HTML table header

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

```
column_names = []

# Apply find_all() function with `th` element on first_launch_table
# Iterate each th element and apply the provided extract_column_from_header() to get a column name
# Append the Non-empty column name (`if name is not None and len(name) > 0`) into a list called column_names
for element in first_launch_table.find_all('th'):
    name = extract_column_from_header(element)
    if name is not None and len(name) > 0:
        column_names.append(name)
```

4. Create Dictionary with keys from extracted variable names:

```
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']=[]
```

5. Fill up the created dictionary using helper functions to parse HTML data (see appendix)

```
def date_time(table_cells):
def booster_version(table_cells):
def landing_status(table_cells):
def get_mass(table_cells):
```

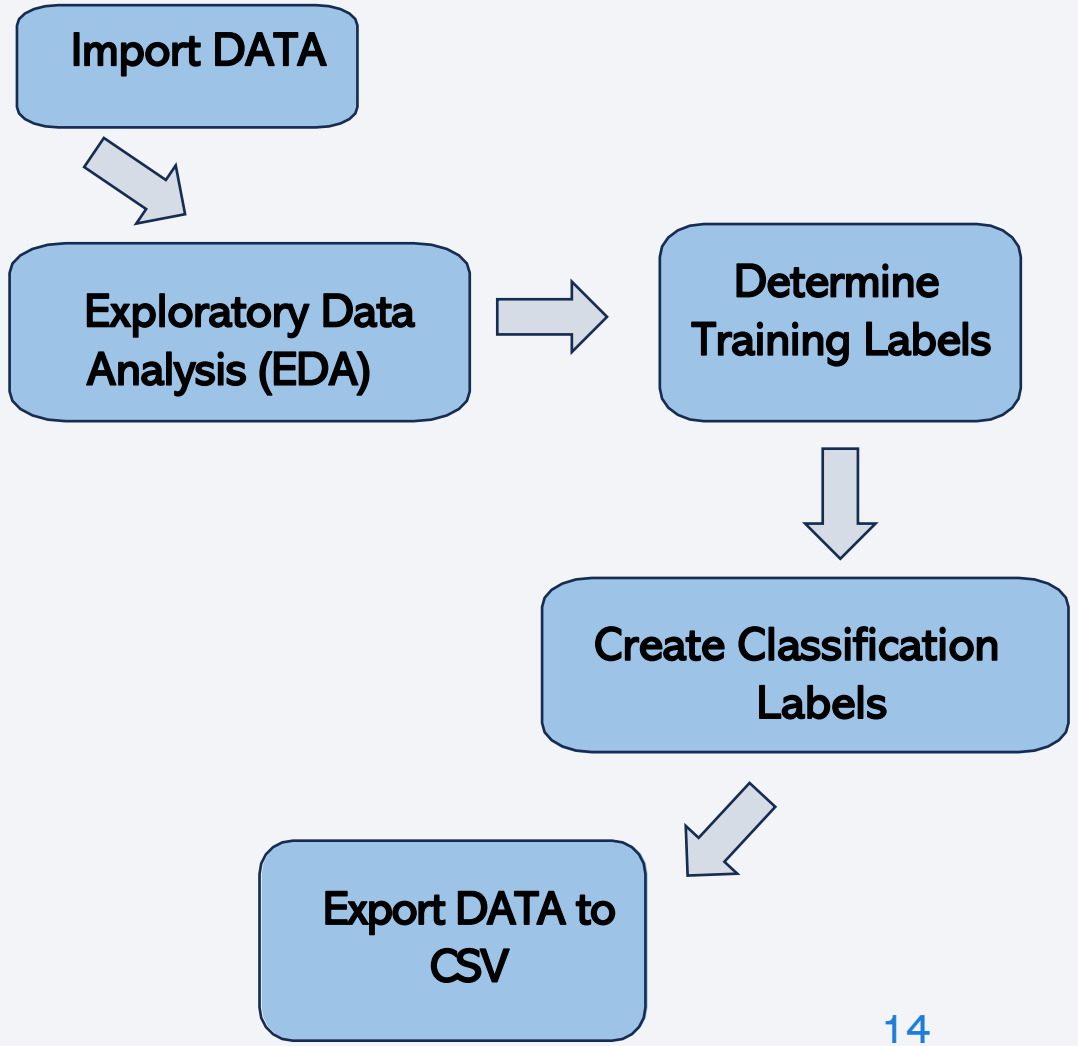
6. Convert launch_dict to Dataframe and save it to CSV:

```
df= pd.DataFrame({ key:pd.Series(value) for key, value in launch_dict.items() })
```

```
df.to_csv('spacex_web_scraped.csv', index=False)
```

Data Wrangling

- **Import DATA:**
  - Import data from the following CSV file containing SpaceX Falcon 9 launch data: https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_1.csv
- **Exploratory Data Analysis (EDA):**
  - Determine patterns and missing values.
  - Identify various attributes Using the .value_counts() method :
    - Number of launches on each site
    - Number and occurrence of each orbit
    - Number and occurrence of landing outcome per orbit type
- **Analyzing Orbit Types:**
  - Identify mission outcomes (landing results) and their frequencies.
  - Create a set of unsuccessful outcomes (bad_outcomes).
- **Creating Classification Labels:**
  - Generate a classification variable (landing_class) where 0 represents unsuccessful landings and 1 represents successful landings.
- **Export DATA:**
  - Export the dataframe to a CSV file (dataset_part_2.csv).



Github - Link

Data Wrangling

1. Load dataset in to Dataframe
2. Find patterns in data
3. Create Classification Labels
4. Export Data to CSV

1. Load SpaceX dataset (csv) in to a Dataframe

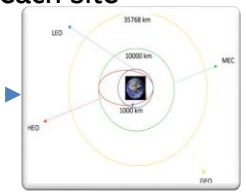
```
df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_1.csv")
```

2. Find data patterns:

- Calculate the number of launches on each site

```
df['LaunchSite'].value_counts()
```

```
CCAFS SLC 40    55
KSC LC 39A      22
VAFB SLC 4E     13
```



- Calculate the number and occurrence of each orbit

```
df['Orbit'].value_counts()
```

```
GTO     27
ISS     21
VLEO    14
PO       9
LEO      7
SSO      5
MEO      3
ES-L1    1
HEO      1
SO       1
GEO      1
```

- Calculate number/occurrence of mission outcomes per orbit type (see appendix)

```
landing_outcomes = df['Outcome'].value_counts()
```

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

```
0 True ASDS
1 None None
2 True RTLS
3 False ASDS
4 True Ocean
5 False Ocean
6 None ASDS
7 False RTLS
```

3. Create a landing outcome label from Outcome column in the Dataframe

- Launch outcome unsuccessfully landed

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

```
{'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

- Create a landing outcome label from Outcome column

```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
landing_class = []
for i in df['Outcome']:
    if i in bad_outcomes:
        landing_class.append(0)
    else:
        landing_class.append(1)
```

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

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

4. Export DATA to CSV:

```
df.to_csv("dataset_part_2.csv", index=False)
```

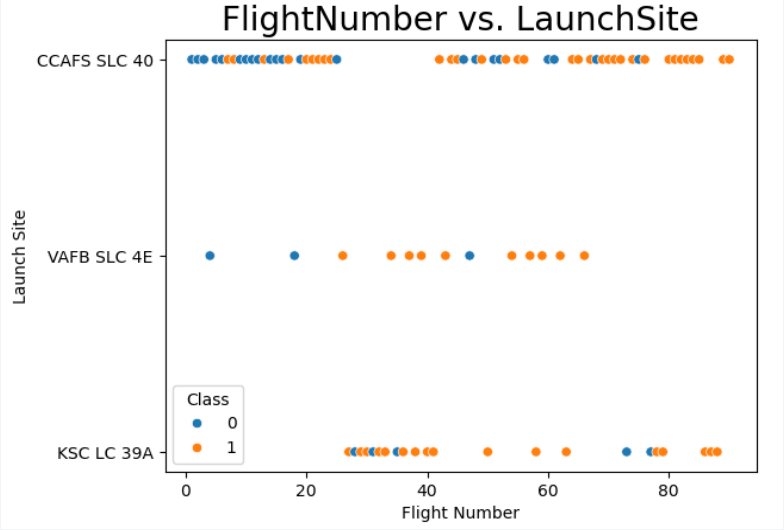
EDA with Data Visualization

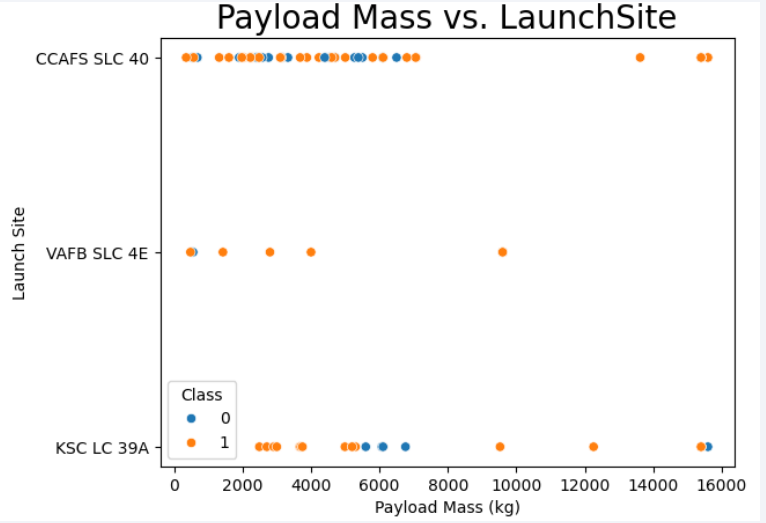
Github - Link

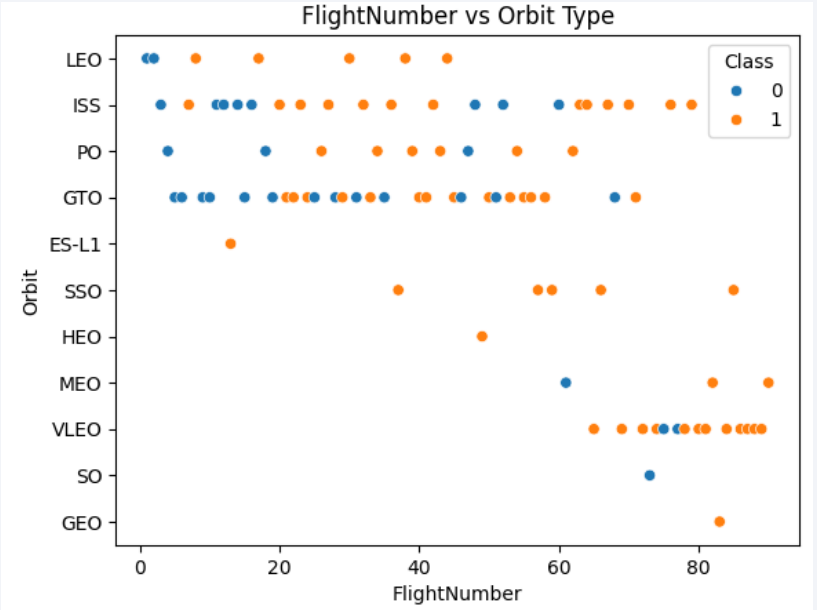
1. SCATTER CHARTS

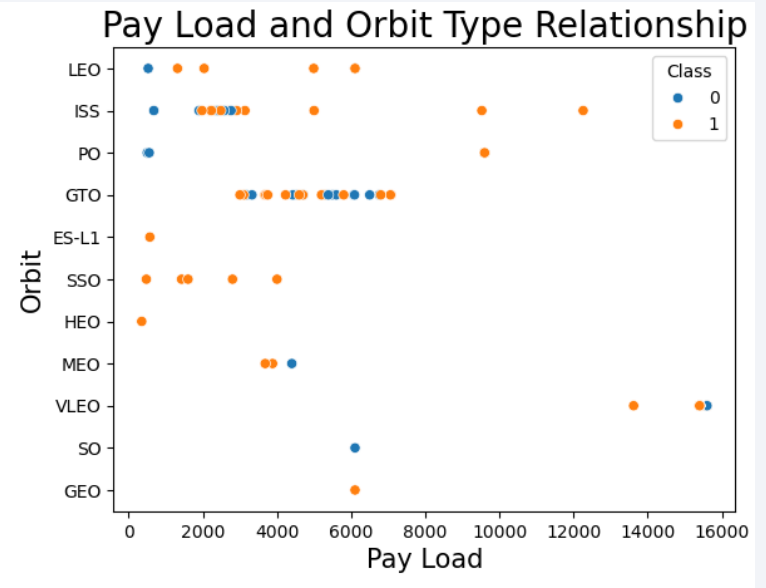
Scatter charts were produced to visualize the relationships between:

- Flight Number and Launch Site
- Payload and Launch Site
- Orbit Type and Flight Number
- Payload and Orbit Type









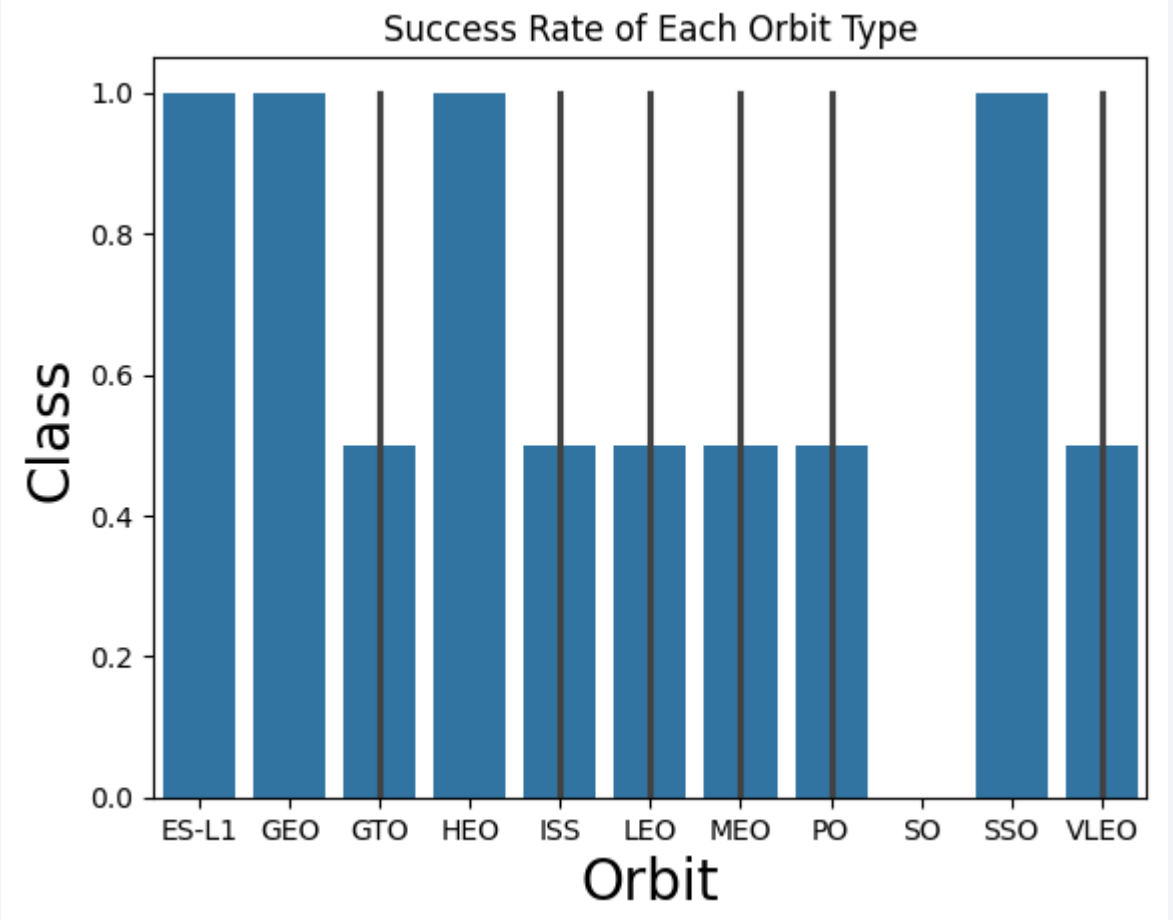
EDA with Data Visualization

Github - Link

2. BAR CHART

A bar chart was produced to visualize the relationship between:

- Success Rate and Orbit Type



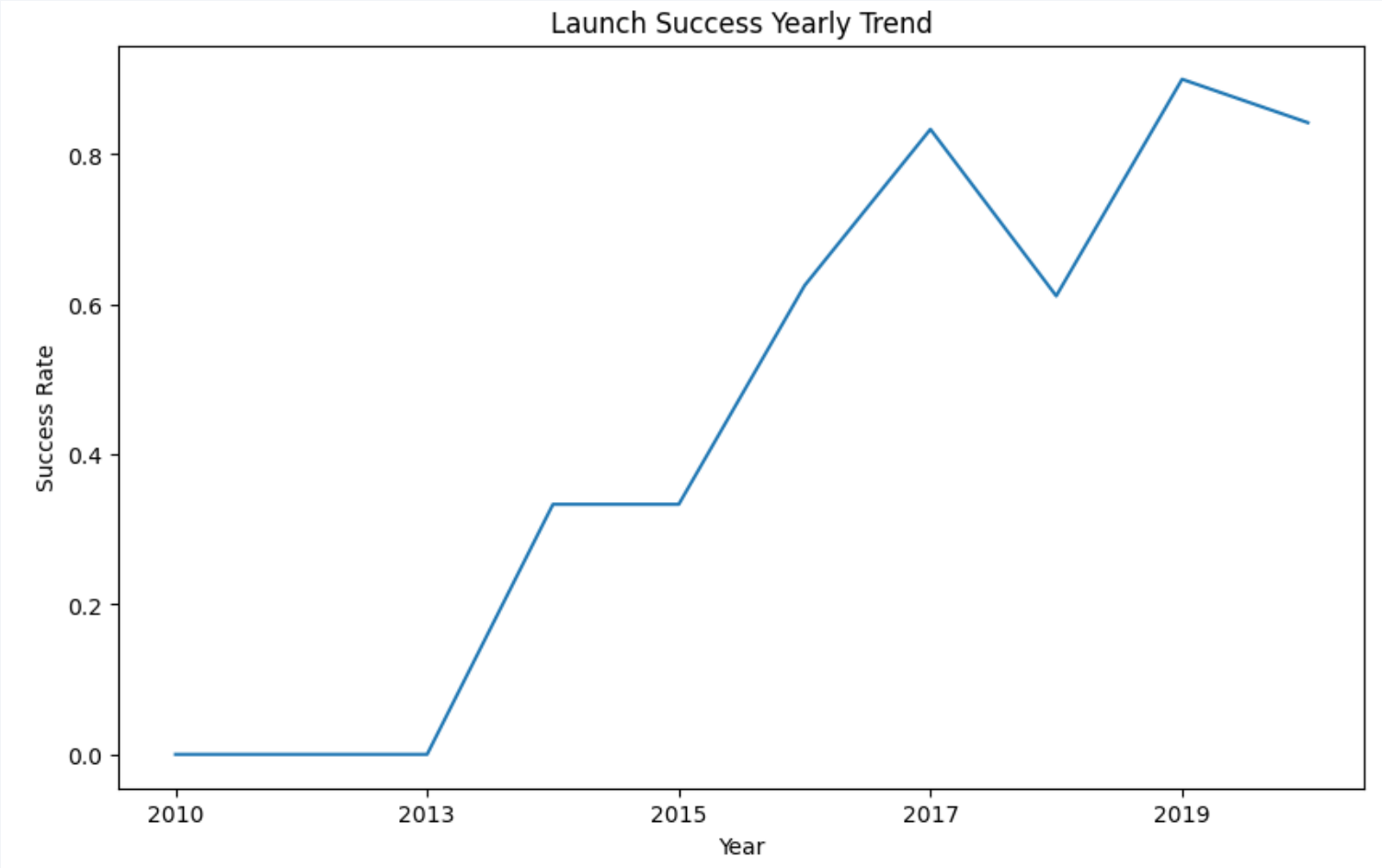
EDA with Data Visualization

Github - Link

3. LINE CHARTS

Line charts were produced to visualize the relationships between:

- Success Rate and Year



EDA with SQL

Github - Link

SQL queries were used to gather the following information about the collected dataset:

1. Display the names of the unique launch sites in the space mission
2. Display 5 records where launch sites begin with the string 'CCA'
3. Display the total payload mass carried by boosters launched by NASA (CRS)
4. Display the average payload mass carried by booster version F9 v1.1
5. List the date when the first successful landing outcome on a ground pad was achieved
6. List the names of the boosters which had success on a drone ship and a payload mass between 4000 and 6000 kg
6. List the total number of successful and failed mission outcomes
7. List the names of the booster versions which have carried the maximum payload mass
8. List the failed landing outcomes on drone ships, their booster versions, and launch site names for 2015
9. 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

1

```
%%sql
select distinct Launch_Site from spacextbl
```

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

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

3

```
%%sql
select sum(PAYLOAD_MASS__KG_) from spacextbl where Customer = 'NASA (CRS)'
```

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

sum(PAYLOAD_MASS__KG_)
45596

4

```
%%sql
select avg(PAYLOAD_MASS__KG_) from spacextbl where Booster_Version LIKE 'F9 v1.1';
```

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

avg(PAYLOAD_MASS__KG_)
2928.4

8

```
%%sql
select Booster_Version, PAYLOAD_MASS__KG_ from spacextbl where PAYLOAD_MASS__KG_ = (select max(PAYLOAD_MASS__KG_) from spacextbl);
```

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

Booster_Version	PAYLOAD_MASS__KG_
F9 B5 B1048.4	15600
F9 B5 B1049.4	15600
F9 B5 B1051.3	15600
F9 B5 B1056.4	15600
F9 B5 B1048.5	15600
F9 B5 B1051.4	15600
F9 B5 B1049.5	15600
F9 B5 B1060.2	15600
F9 B5 B1058.3	15600
F9 B5 B1051.6	15600
F9 B5 B1060.3	15600
F9 B5 B1049.7	15600

2

```
%%sql
select * from spacextbl where Launch_Site LIKE 'CCA%' limit 5;
```

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

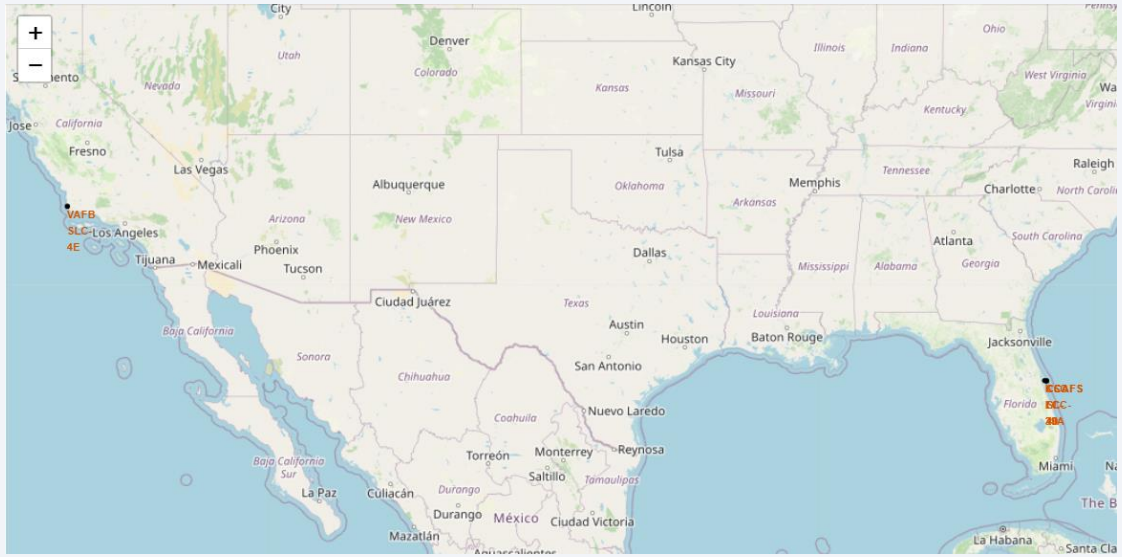
Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Github - Link

Build an Interactive Map with Folium

- Building an interavtive Map with folium allowed to gather information about the location of launch sites (with Successful and Unsuccessful landing). In order to do so I followed these steps:

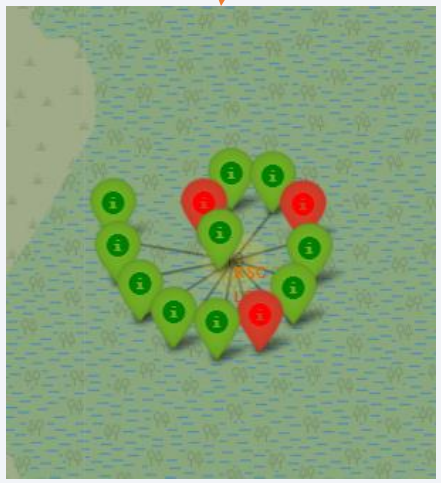
- Mark all launch sites on the map. This allowed to visually see the launch sites on the map.
  - Add 'folium.circle' and 'folium.marker' to highlight circle area with a text label over each launch site.

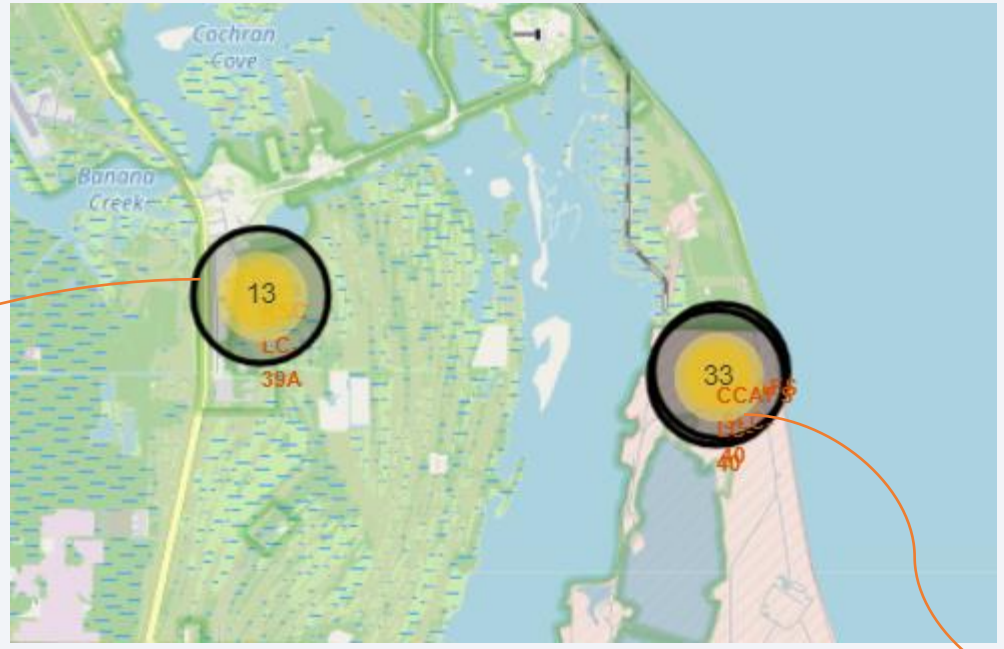


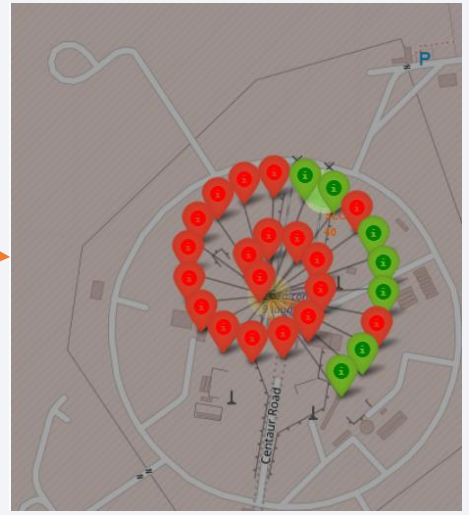
Build an Interactive Map with Folium

Github - Link

- Add a 'MarkerCluster()' to show launch success (green) and failure (red) markers for each launch site.



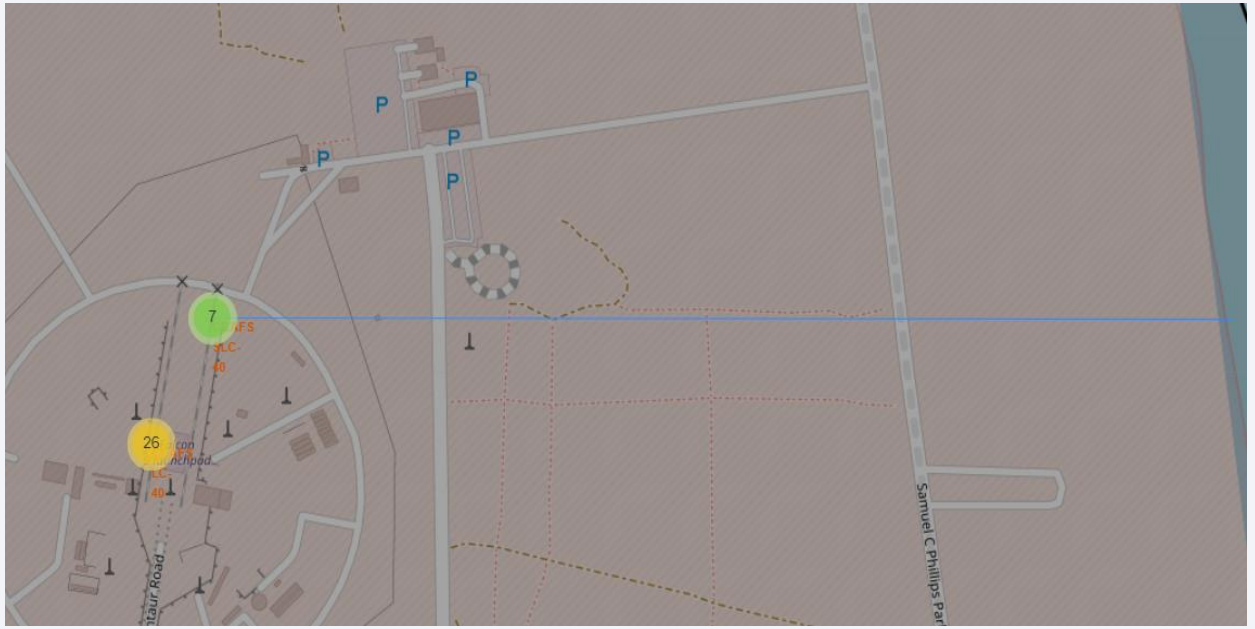




Build an Interactive Map with Folium

Github - Link

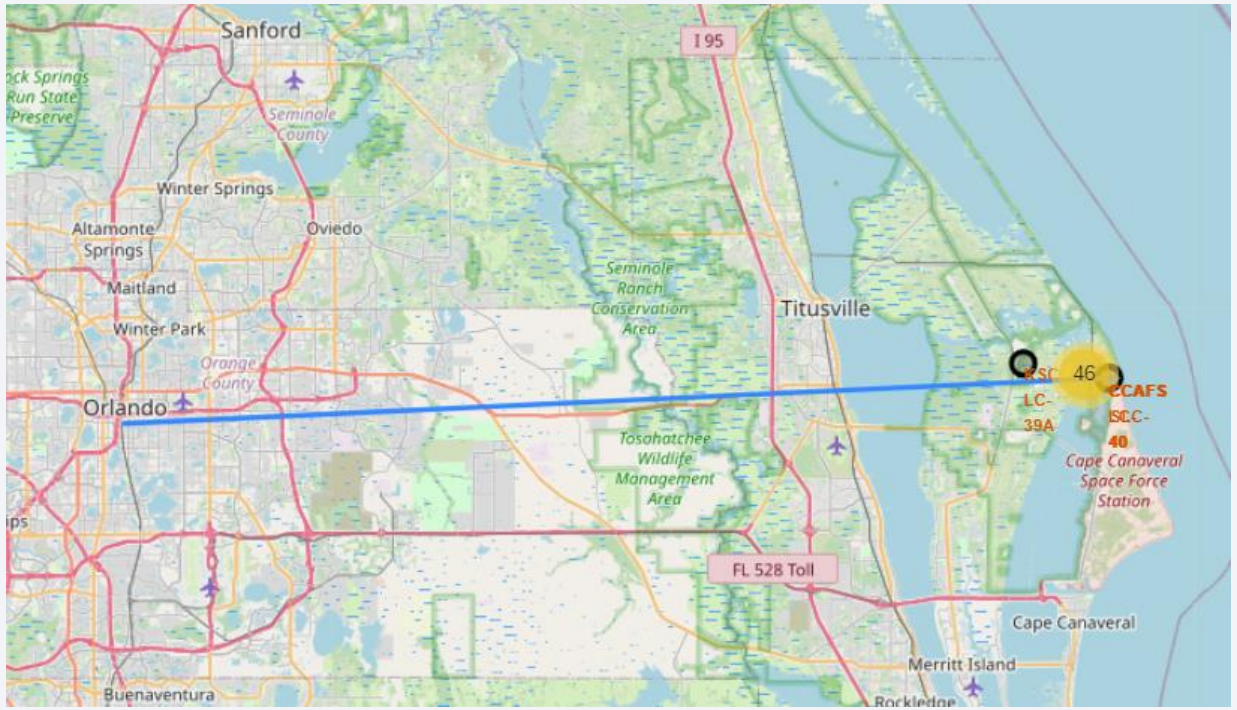
- Calculate distances between a launch site to its proximities (e.g., coastline, railroad, highway, city)
  - Incorporate 'MousePosition()' to capture the coordinates of the mouse position over a specific point on the map.
  - Use 'folium.Marker()' to display the distance (in kilometers) from a point on the map, such as a coastline, railroad, highway, or city.
  - Implement 'folium.Polyline()' to draw a line connecting the point on the map to the launch site.
  - Repeat the steps above to add markers and draw lines between the launch sites and nearby features such as coastlines, railroads, highways, and cities.



Build an Interactive Map with Folium

Github - Link

- These interactive Maps answered the following questions:
  - Are launch sites in close proximity to railways? YES
  - Are launch sites in close proximity to highways? YES
  - Are launch sites in close proximity to coastline? YES
  - Do launch sites keep certain distance away from cities? YES



Build a Dashboard with Plotly Dash

Github - Link

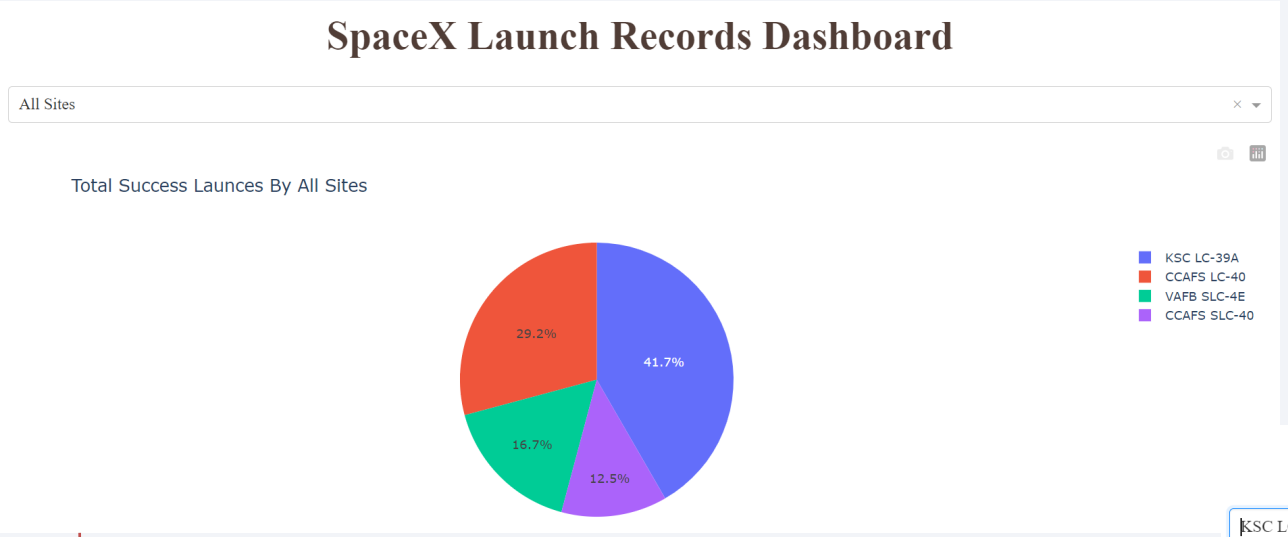
Dashboard was built with Plotly Dash :

1. Pie chart (px.pie()) shows the total successful launches per site
   - Show which sites are most successful
   - The dcc.Dropdown()object was used to filter success or failure ratio for an individual site

2. Scatter graph (px.scatter()) shows the correlation between landing outcome (success landing or not) and payload mass (kg)
   - RangeSlider()object was used to show graph by ranges of payload masses
   - Booster version was also used to flter.

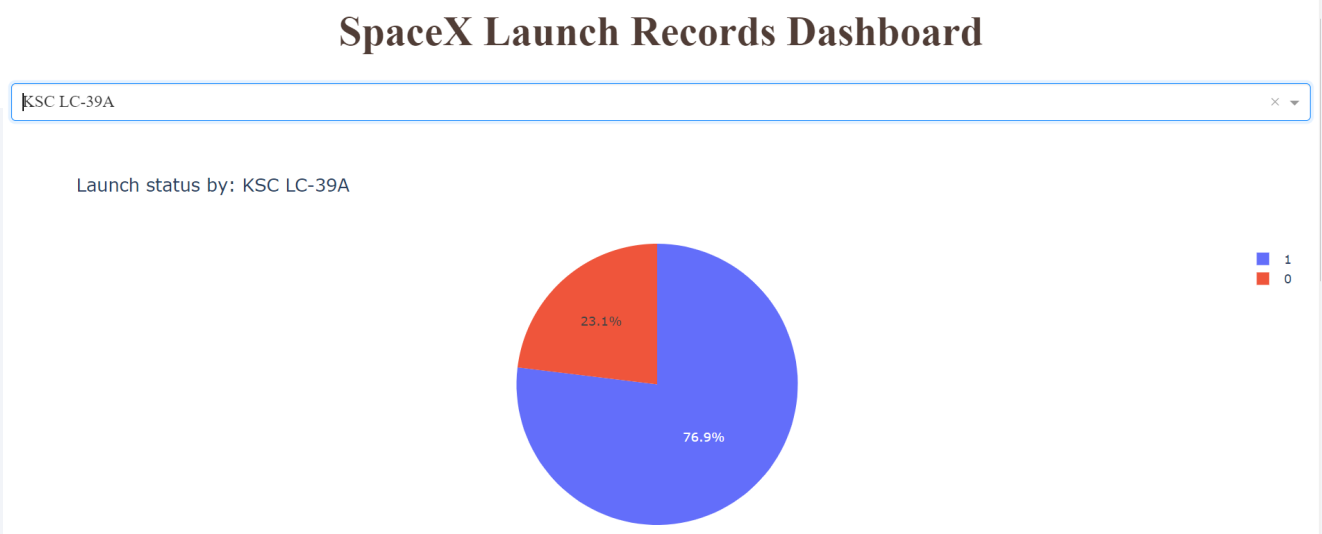
Build a Dashboard with Plotly Dash

Github - Link



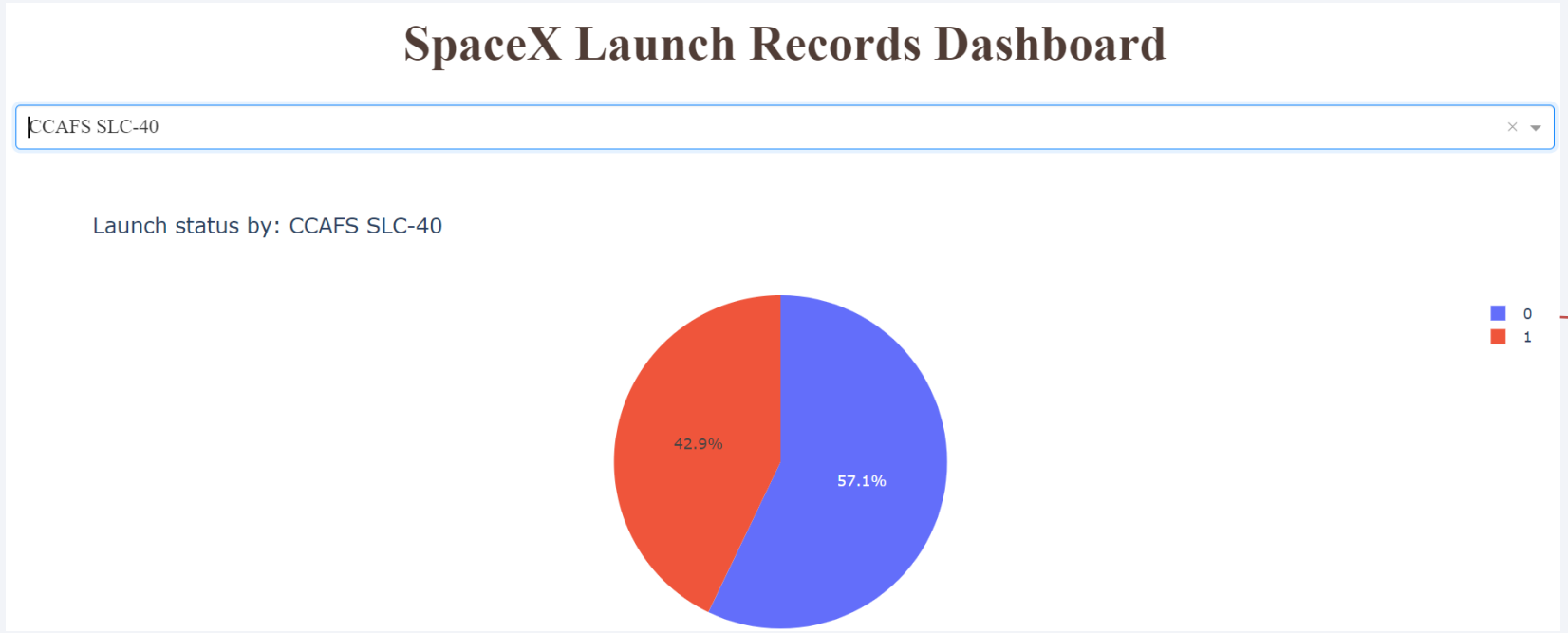
- KSC LC-39A Launch Site has the highest launch success rate :
  - ➢ Launch success rate is 76.9%
  - ➢ Launch failure rate is 23.1%

- Launch Site 'KSC LC-39A' has the highest launch success rate
- Launch Site 'CCAFS SLC- 40' has the lowest launch success rate



Build a Dashboard with Plotly Dash

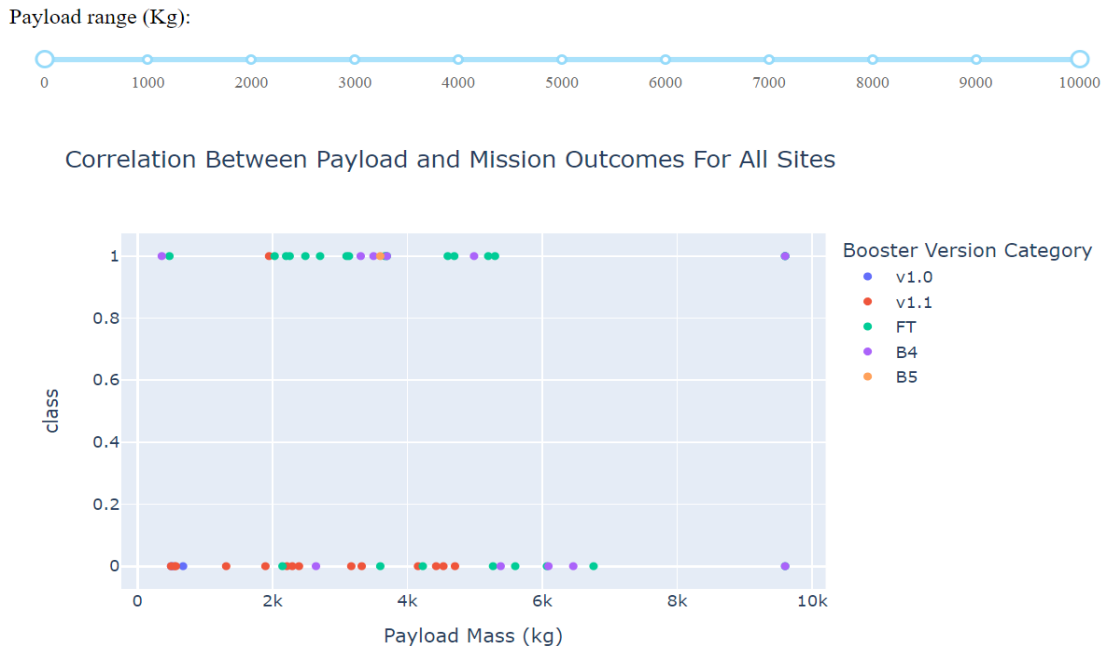
Github - Link

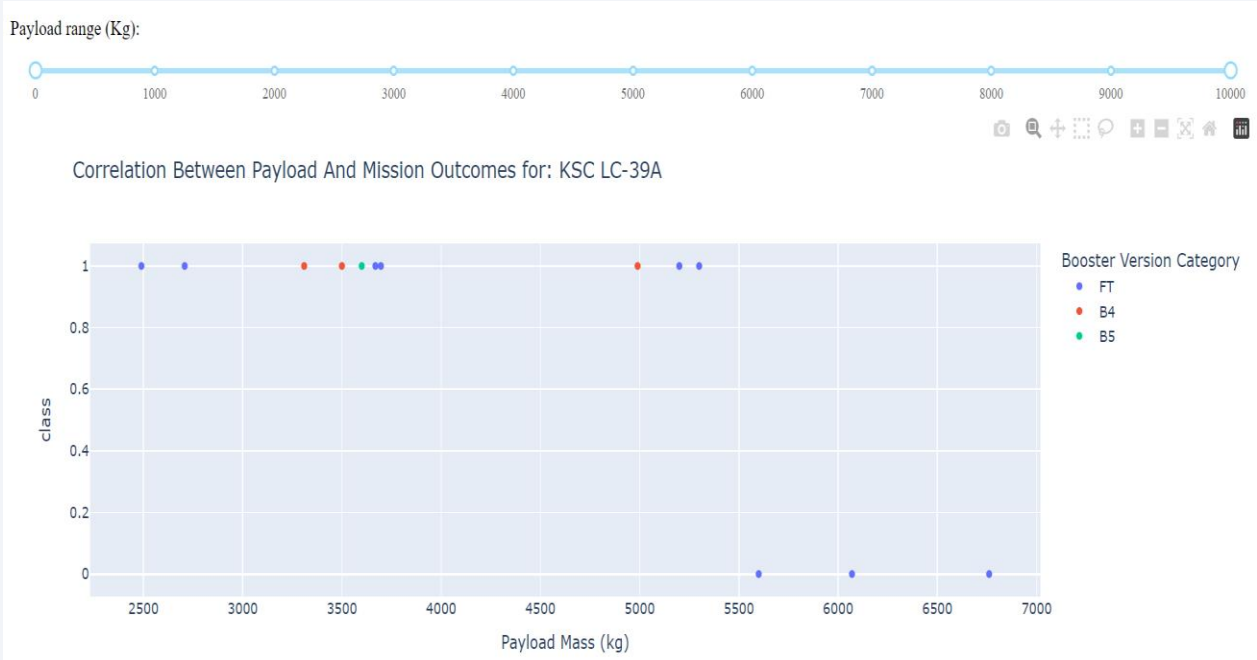


- CCAFS SLC-40 Launch Site has the lowest launch success rate :
  - Launch success rate is 42.9%
  - Launch failure rate is 57.1%

Build a Dashboard with Plotly Dash

Github - Link



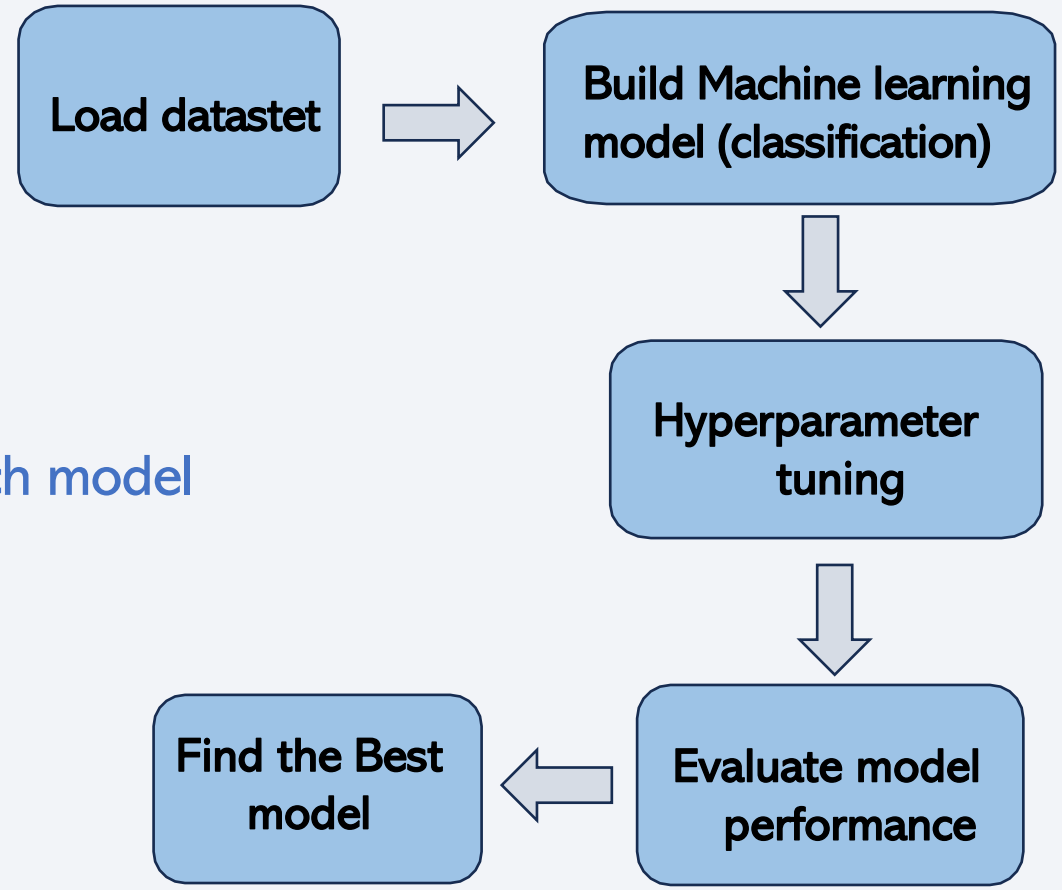


- Payload vs. Launch Outcome Scatter Plot for All Sites and for KSC LC-39A

Predictive Analysis (Classification)

Github - Link

- Build Machine learning model
  - Create columna for the classification class
  - Standarize the data
  - Train-test Split the data into training and testing datasets
  - Build 4 classification models (logistic regression, support vector machine (SVM), decision tree and k nearest neighbors (KNN))
- Hyperparameter tuning to get the best parameters for each model Using GridSearchCV.
- Evaluate model performance
  - Measure Accuracy for each model for training and testing sets
  - Calculate confusion matrix
- Find Best machine learning model



Predictive Analysis (Classification)

Github - Link

1. Read dataset from the two datasets and create a 'Class' array using numpy
2. Standardize the data
3. Train/Test Split data in to training and testing data sets
4. Define machine learning Models
5. Find the best performing Model

1. Load dataset (dataset_part_2.csv and dataset_part_3.csv) and create NumPy array from the column class in data

```
URL1 = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_2.csv"
resp1 = await fetch(URL1)
text1 = io.BytesIO((await resp1.arrayBuffer()).to_py())
data = pd.read_csv(text1)
```

```
URL2 = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_3.csv'
resp2 = await fetch(URL2)
text2 = io.BytesIO((await resp2.arrayBuffer()).to_py())
X = pd.read_csv(text2)
```

```
Y = data['Class'].to_numpy()
```

2. Standardize data in X then reassign to variable X using transform

```
# students get this
transform = preprocessing.StandardScaler()
```

```
X= transform.fit(X).transform(X)
```

3. Train/test split X and Y in to training and test data sets.

```
# Split data for training and testing data sets
X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size=0.2, random_state=2)
print ('Train set:', X_train.shape,  Y_train.shape)
print ('Test set:', X_test.shape,  Y_test.shape)
```

3. Define machine learning models (logistic regression, support vector machine (SVM), decision tree and k nearest neighbors (KNN)) and hyperparameter tuning:

- Create machine learning model object and then create a GridSearchCV object to choose best parameters
- Find and display best hyperparameters and accuracy score
- Check the accuracy on the test data by creating a confusion matrix

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

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

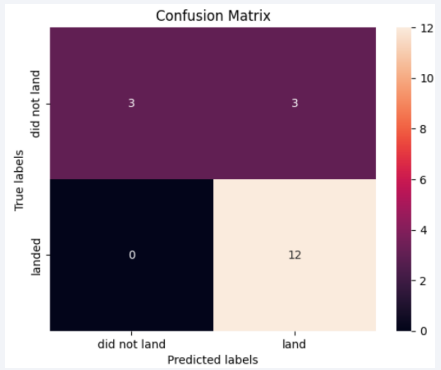
```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

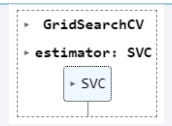
```
tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8464285714285713
```

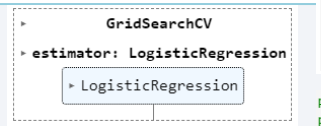
```
print("Test data accuracy score using Logistic Regression: ", logreg_cv.score(X_test, Y_test))
```

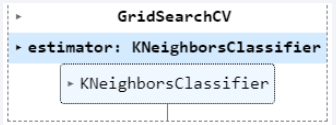
```
Test data accuracy score using Logistic Regression:  0.8333333333333334
```

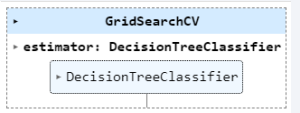
```
yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```











4. Find the best performing model

```
# Create a Data Frame for algorithm type and respective best scores
Model_Performance = pd.DataFrame({'Machine Learning Model': ['Logistic Regression', 'SVM','Decision Tree','KNN'],
'Accuracy Score': [logreg_cv.best_score_, svm_cv.best_score_, tree_cv.best_score_, knn_cv.best_score_],
'Test Data Accuracy Score': [logreg_cv.score(X_test, Y_test), svm_cv.score(X_test, Y_test), tree_cv.score(X_test, Y_test), knn_cv.score(X_test, Y_test)]})
```

```
# Sort best machine learning model
Model_Performance.sort_values(['Accuracy Score'], ascending = False, inplace=True)
# reset the index
Model_Performance.reset_index(drop=True, inplace=True)
```

	Machine Learning Model	Accuracy Score	Test Data Accuracy Score
0	Decision Tree	0.887500	0.888889
1	KNN	0.848214	0.833333
2	SVM	0.848214	0.833333
3	Logistic Regression	0.846429	0.833333

Results

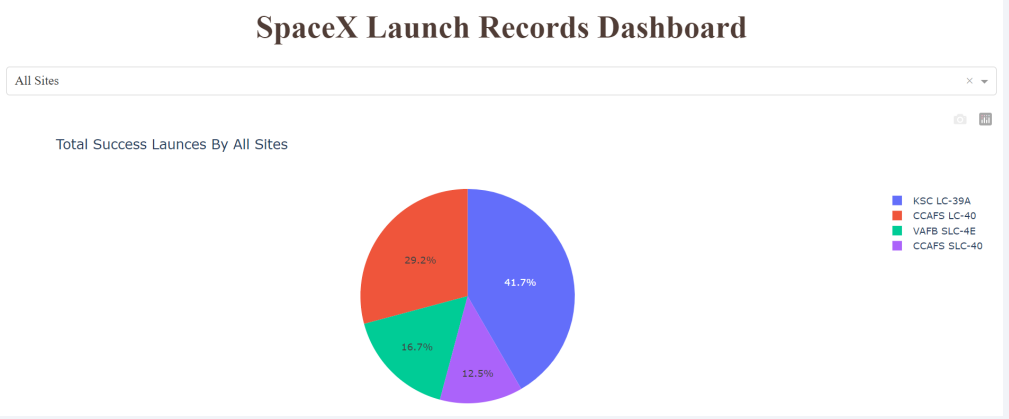
- Exploratory data analysis results
  - Space X uses 4 different launch sites.
  - Lighter payloads have higher performance (successful landing) compared to heavier ones.
  - As years go on, SpaceX launch is more prone to have successful landing. This is promising for a flawless launches over time.
  - 4 Orbits has 100% success landing rate

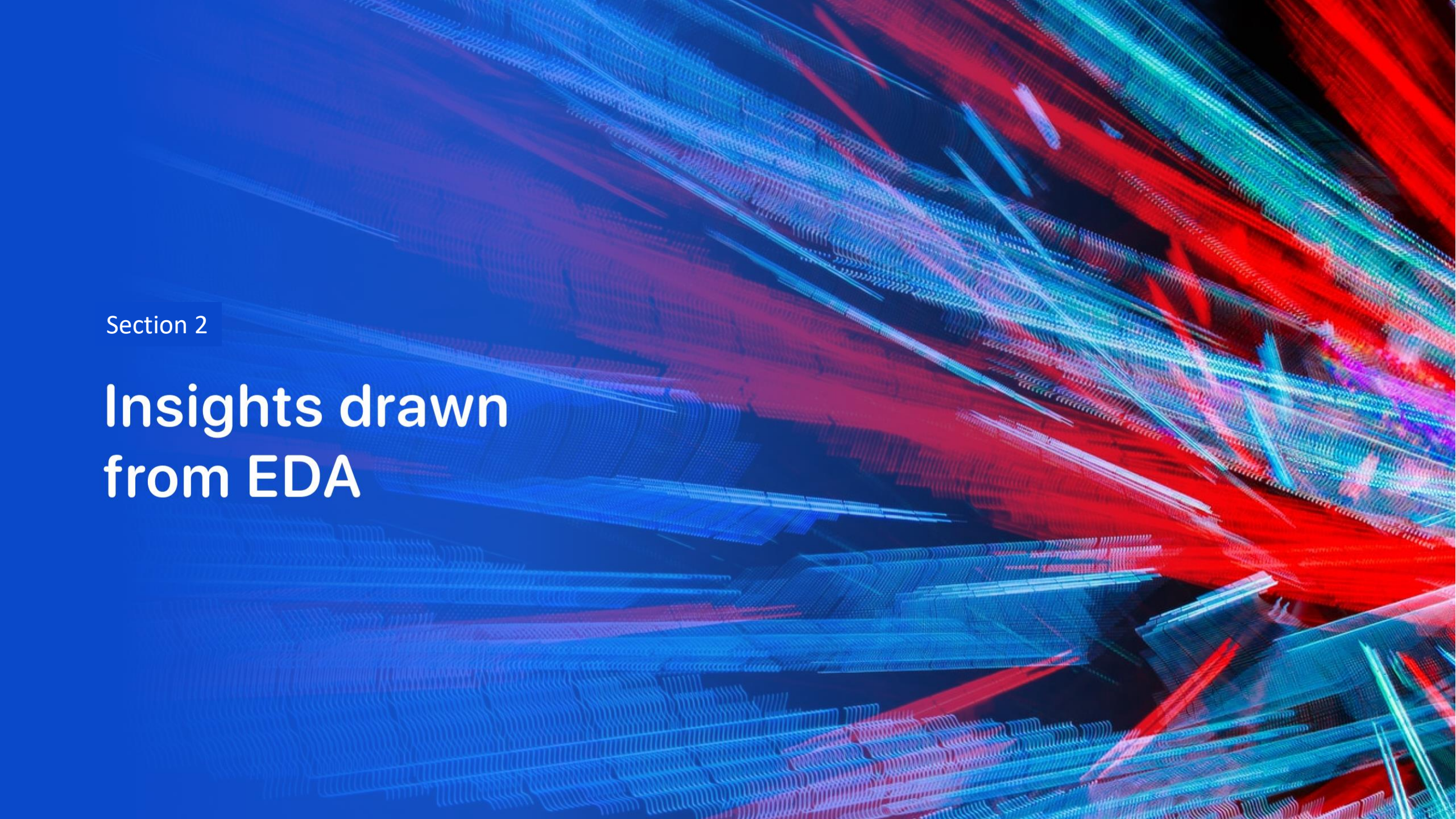
1. Orbits with 100% success rate are:
   - ES-L1
   - GEO
   - HEO
   - SSO
2. Orbits with 0% success rate are:
   - SO
3. Orbits with 50% success rate are:
   - GTO
   - ISS
   - LEO
   - MEO
   - PO
   - VLEO

- Interactive analytics demo in screenshots
  - Launch Complex 39A at Kennedy Apace Center has the highest number of successful launches compared ro other launch site.

- Predictive analysis results
  - Decision Tree model is the best classification model to predict successful landing.

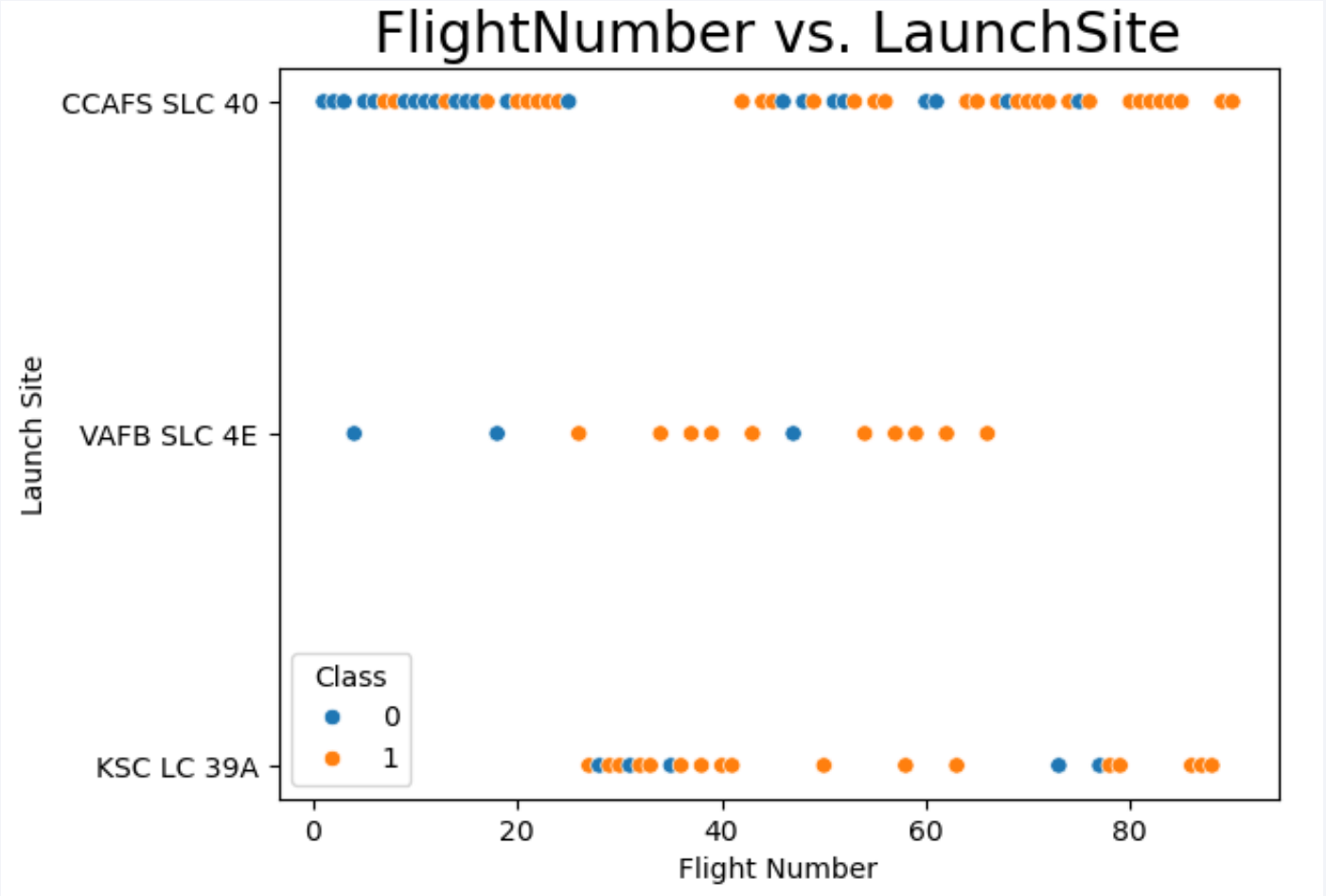
	Machine Learning Model	Accuracy Score	Test Data Accuracy Score
0	Decision Tree	0.887500	0.888889
1	KNN	0.848214	0.833333
2	SVM	0.848214	0.833333
3	Logistic Regression	0.846429	0.833333




Section 2
Insights drawn from EDA

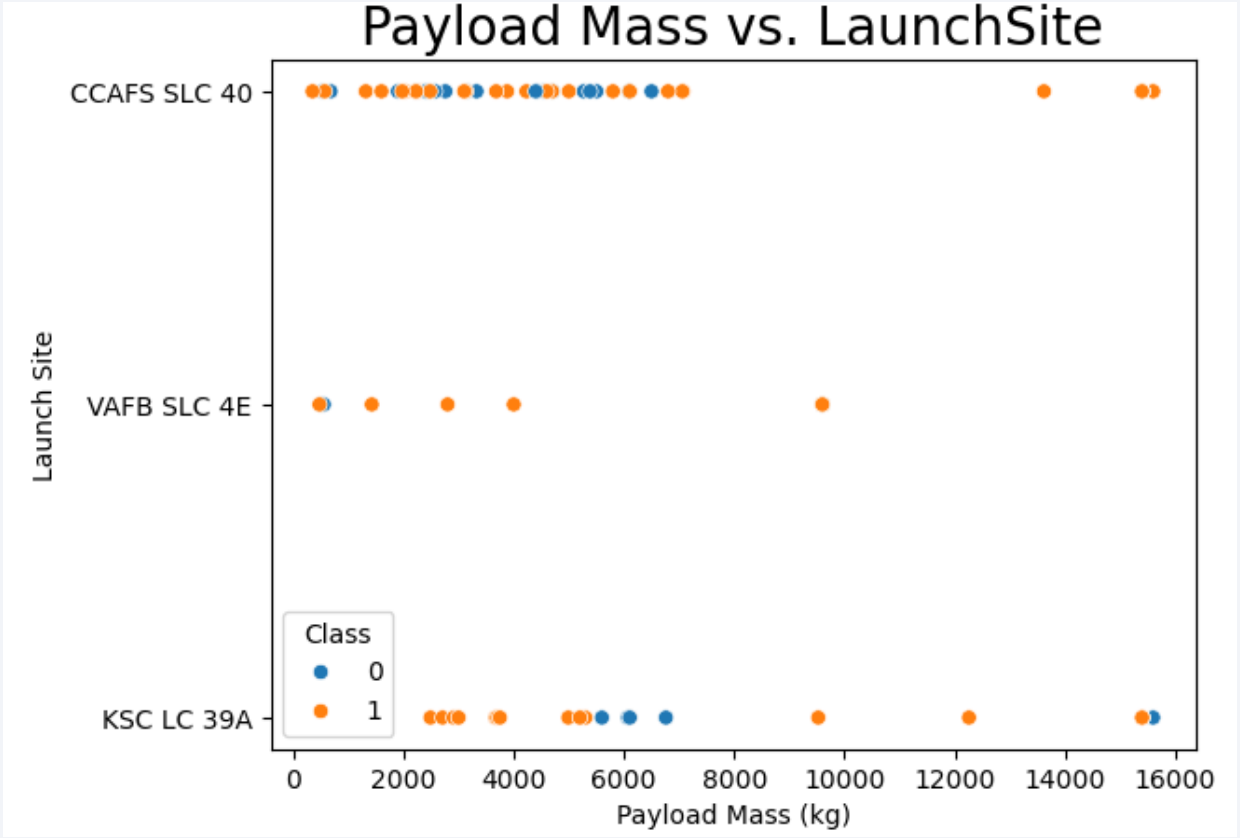
Flight Number vs. Launch Site

- Scatter plot of Flight Number vs. Launch Site
- Success rates (Class=1) increases as the number of flights increase



Payload vs. Launch Site

- Scatter plot of Payload mass vs. Launch Site
  - For the launch site ‘VAFB SLC 4E’, all the launched rockets playloads are greater than 10k kg
  - From the scatter plot, no clear correlation appears between launch site and payload mass

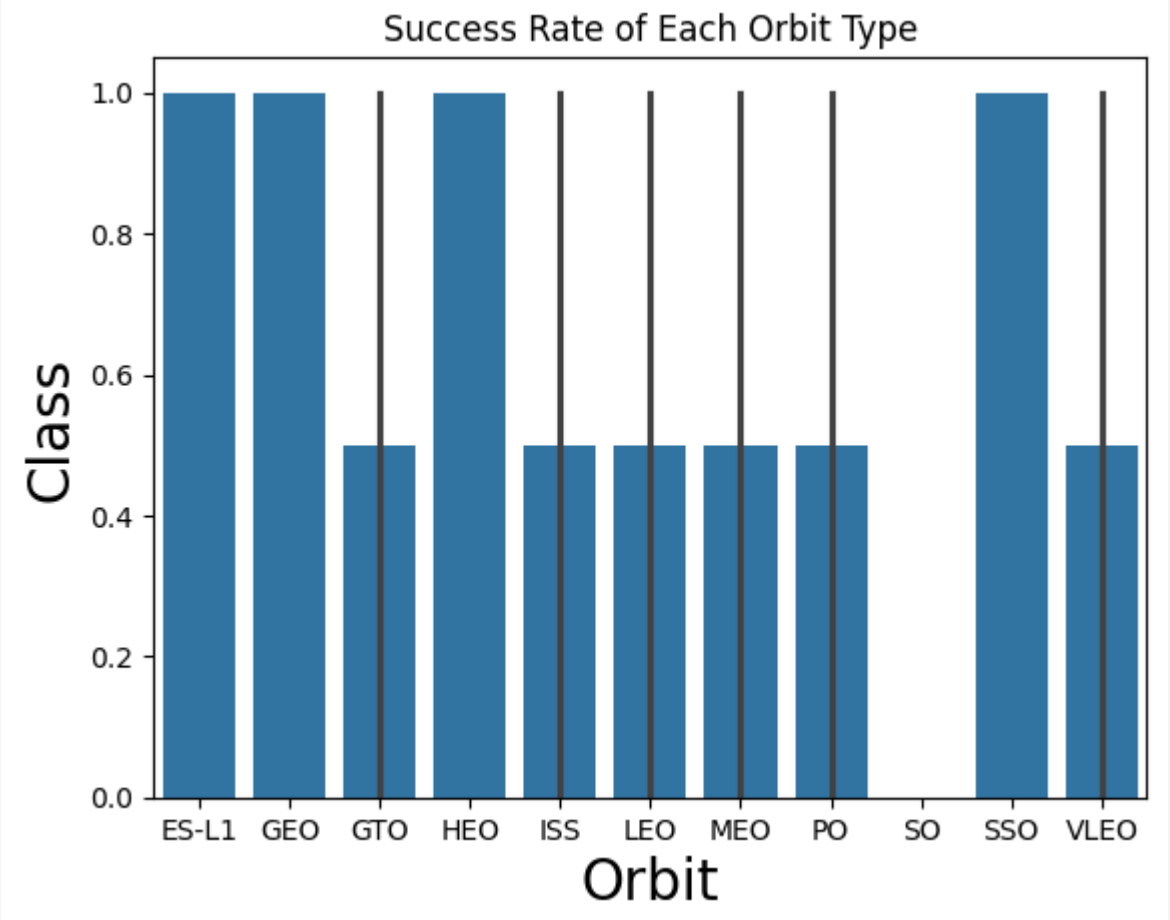


Success Rate vs. Orbit Type

- Bar chart for the success rate of each orbit type

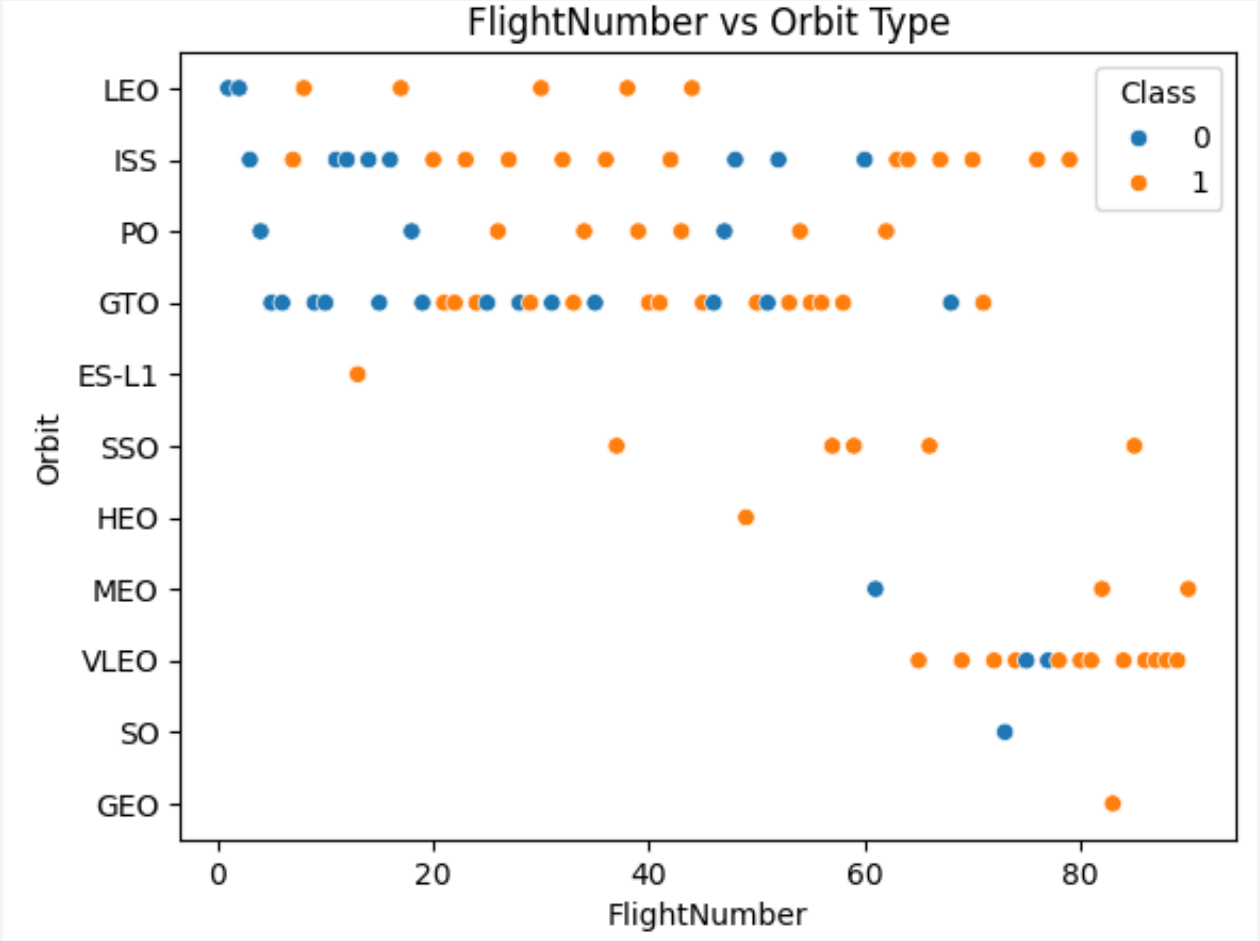
Bar plot analysis:

1. Orbits with 100% success rate are:
   - ES-L1
   - GEO
   - HEO
   - SSO
2. Orbits with 0% success rate are:
   - SO
3. Orbits with 50% success rate are:
   - GTO
   - ISS
   - LEO
   - MEO
   - PO
   - VLEO



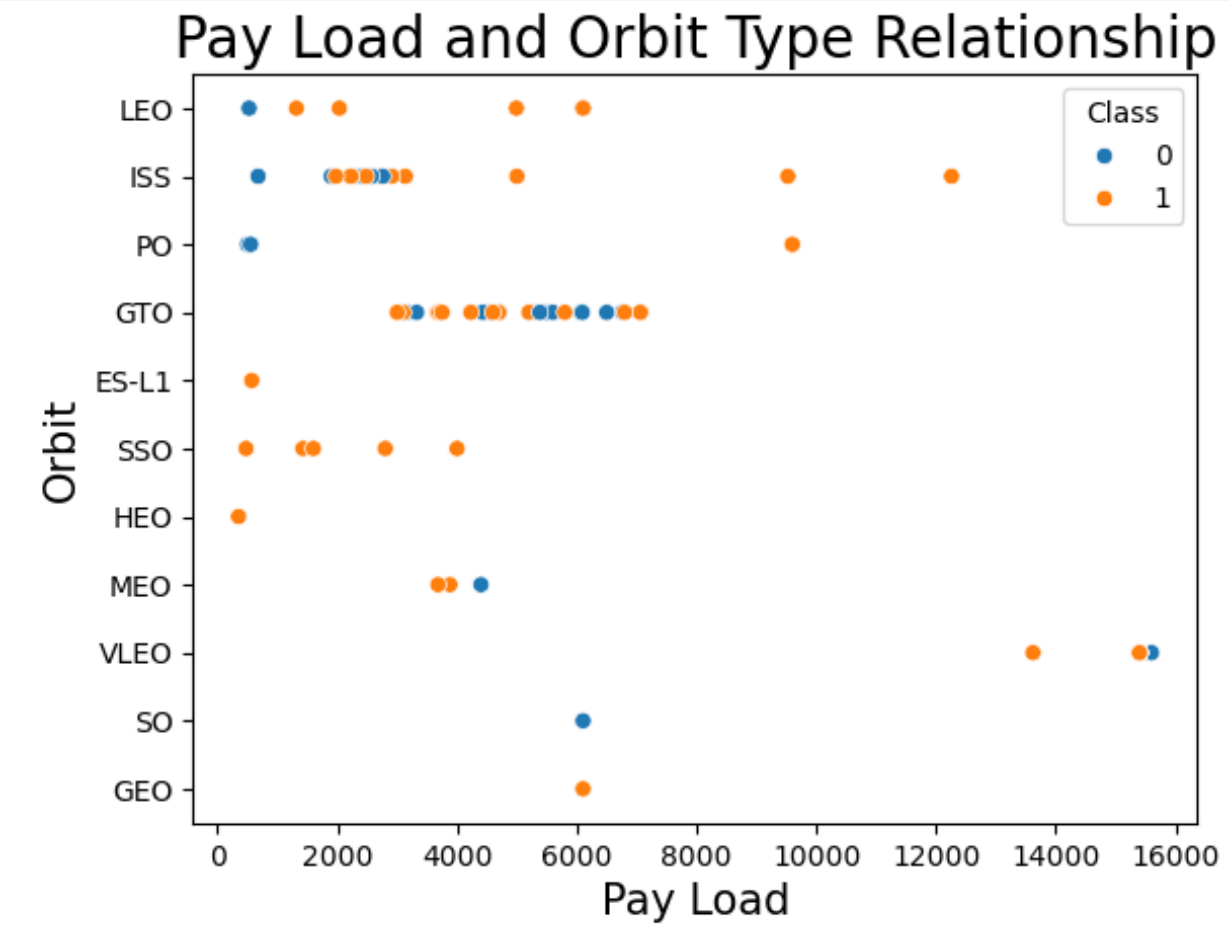
Flight Number vs. Orbit Type

- Scatter point of Flight number vs. Orbit type
- For LEO orbit the Success appears related to the number of flights: For flight numbers higher than 10 it's a success
- No relationship between flight number and successful landing for GTO orbit.



Payload vs. Orbit Type

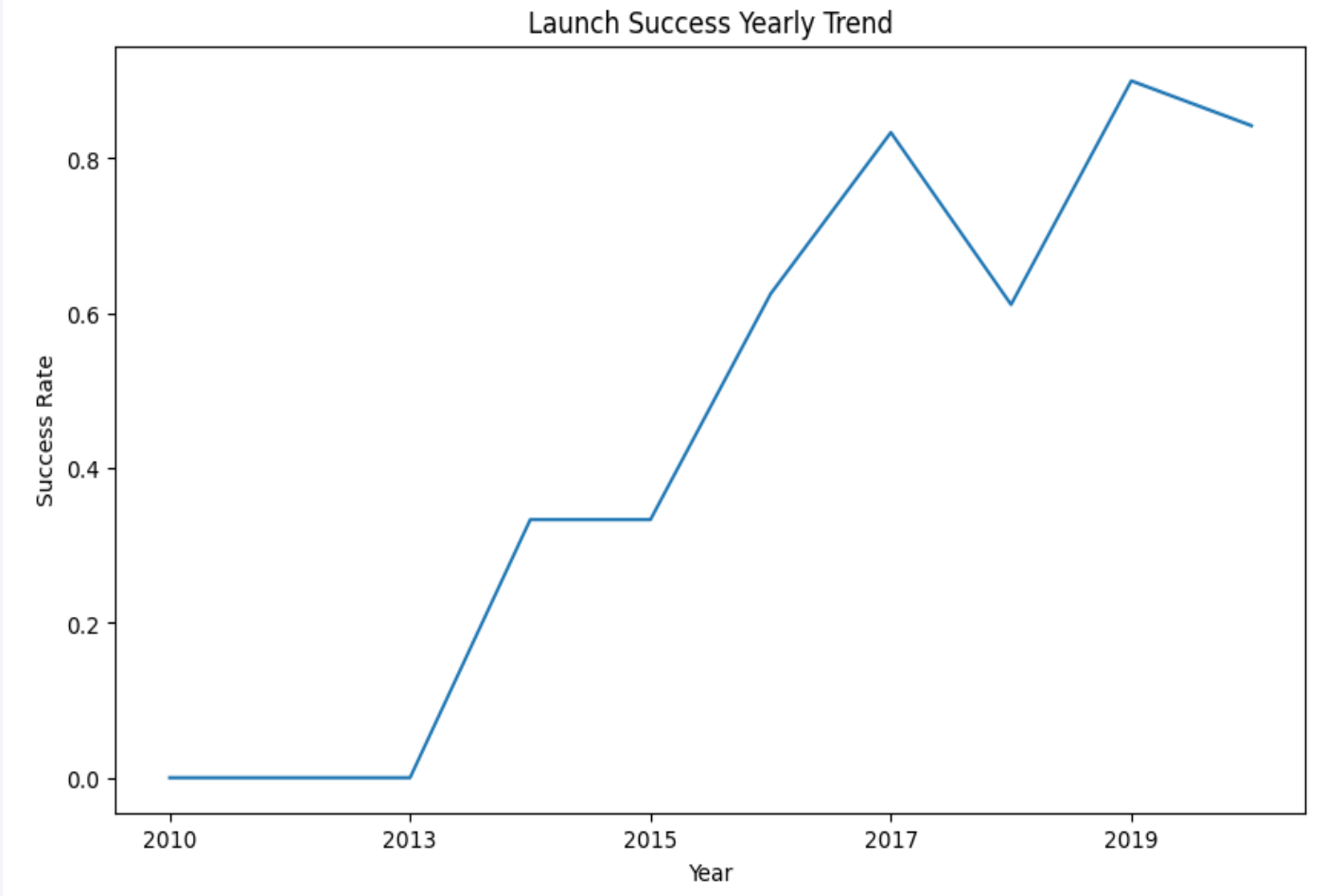
- Scatter point of payload vs. orbit type
- With heavy payloads the successful landing rate are more observed for Polar,LEO and ISS orbits.
- For GTO no relation between playload and successful landing seems to be clear.



Launch Success Yearly Trend

- Line chart of yearly average success rate

- Success rate kept increasing since 2013 till 2020



All Launch Site Names

- Find the names of the unique launch sites

  - Explanation:

    - There are 4 unique launch sites

```
[9]: %%sql

     select distinct Launch_Site from spacextbl

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

[9]:

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

Launch Site Names Begin with 'CCA'

- Find 5 records where launch sites begin with `CCA`

```
%%sql

select * from spacextbl where Launch_Site LIKE 'CCA%' limit 5;
```

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

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Total Payload Mass

- Calculate the total payload carried by boosters from NASA

```
%%sql
select sum(PAYLOAD_MASS__KG_) from spacextbl where Customer = 'NASA (CRS)'
```

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

sum(PAYLOAD_MASS__KG_)
45596

Average Payload Mass by F9 v1.1

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

```
%%sql

select avg(PAYLOAD_MASS__KG_) from spacextbl where Booster_Version LIKE 'F9 v1.1';
```

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

avg(PAYLOAD_MASS__KG_)
2928.4

Total Number of Successful and Failure Mission Outcomes

- Calculate the total number of successful and failure mission outcomes

```
%%sql

select Mission_Outcome, count(Mission_Outcome) as counts from spacextbl group by Mission_Outcome;
```

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

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

Boosters Carried Maximum Payload

- List the names of the booster which have carried the maximum payload mass

```
%%sql

select Booster_Version, PAYLOAD_MASS__KG_ from spacextbl where PAYLOAD_MASS__KG_ = (select max(PAYLOAD_MASS__KG_) from spacextbl);
```

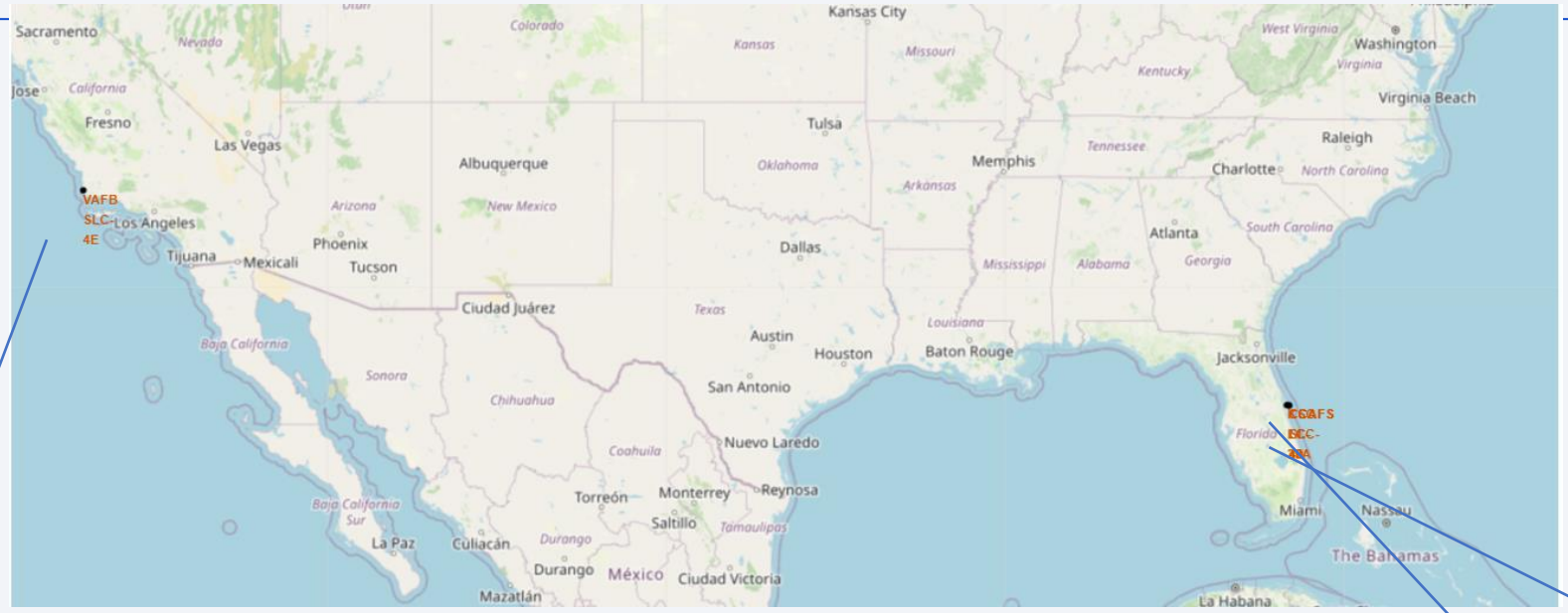
```
 * sqlite:///my_data1.db
Done.
```

Booster_Version	PAYLOAD_MASS__KG_
F9 B5 B1048.4	15600
F9 B5 B1049.4	15600
F9 B5 B1051.3	15600
F9 B5 B1056.4	15600
F9 B5 B1048.5	15600
F9 B5 B1051.4	15600
F9 B5 B1049.5	15600
F9 B5 B1060.2	15600
F9 B5 B1058.3	15600
F9 B5 B1051.6	15600
F9 B5 B1060.3	15600
F9 B5 B1049.7	15600

Section 3

Launch Sites Proximities Analysis

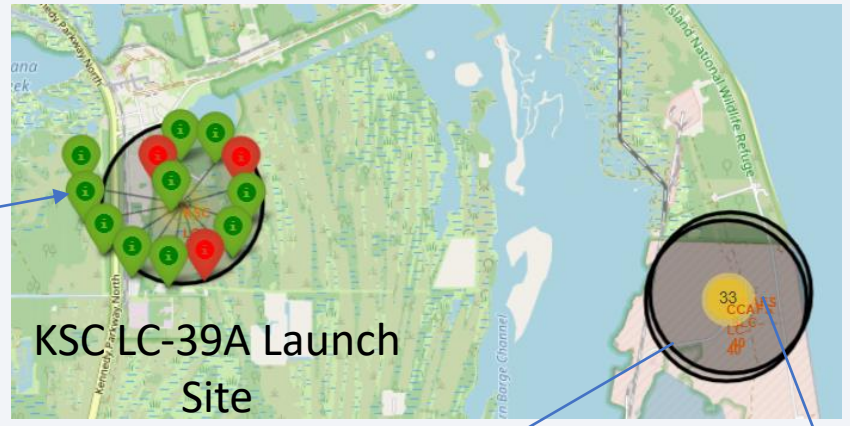
SpaceX Falcon9 - Launch Sites Map

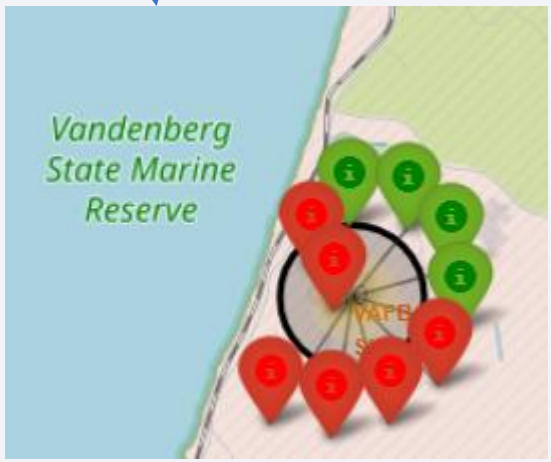


This map shows 4 launch sites:

- VAFB SLC-4E (CA)
- CCAFS LC-40 (FL)
- KSC LC-39A (FL)
- CCAFS SLC-40 (FL)

SpaceX Falon9 – Success/Failed Launch Map for all Launch Sites

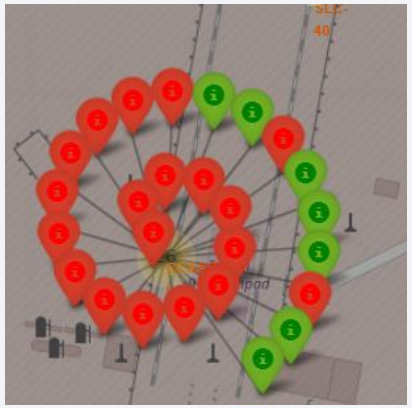




VAFB Launch Site

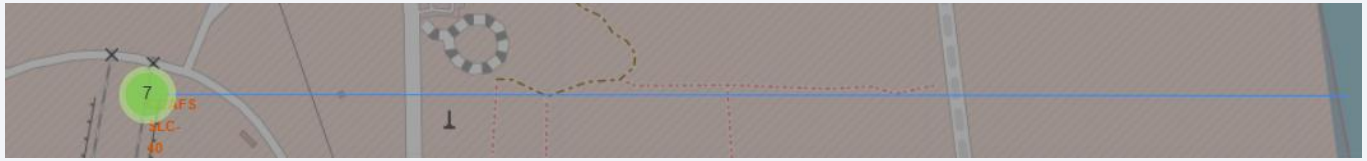
- By looking at all 4 launch sites in the map, KSC LC-39A Launch Site has the greatest number of successful launches.

CCAFS SLC-40 Launch Site

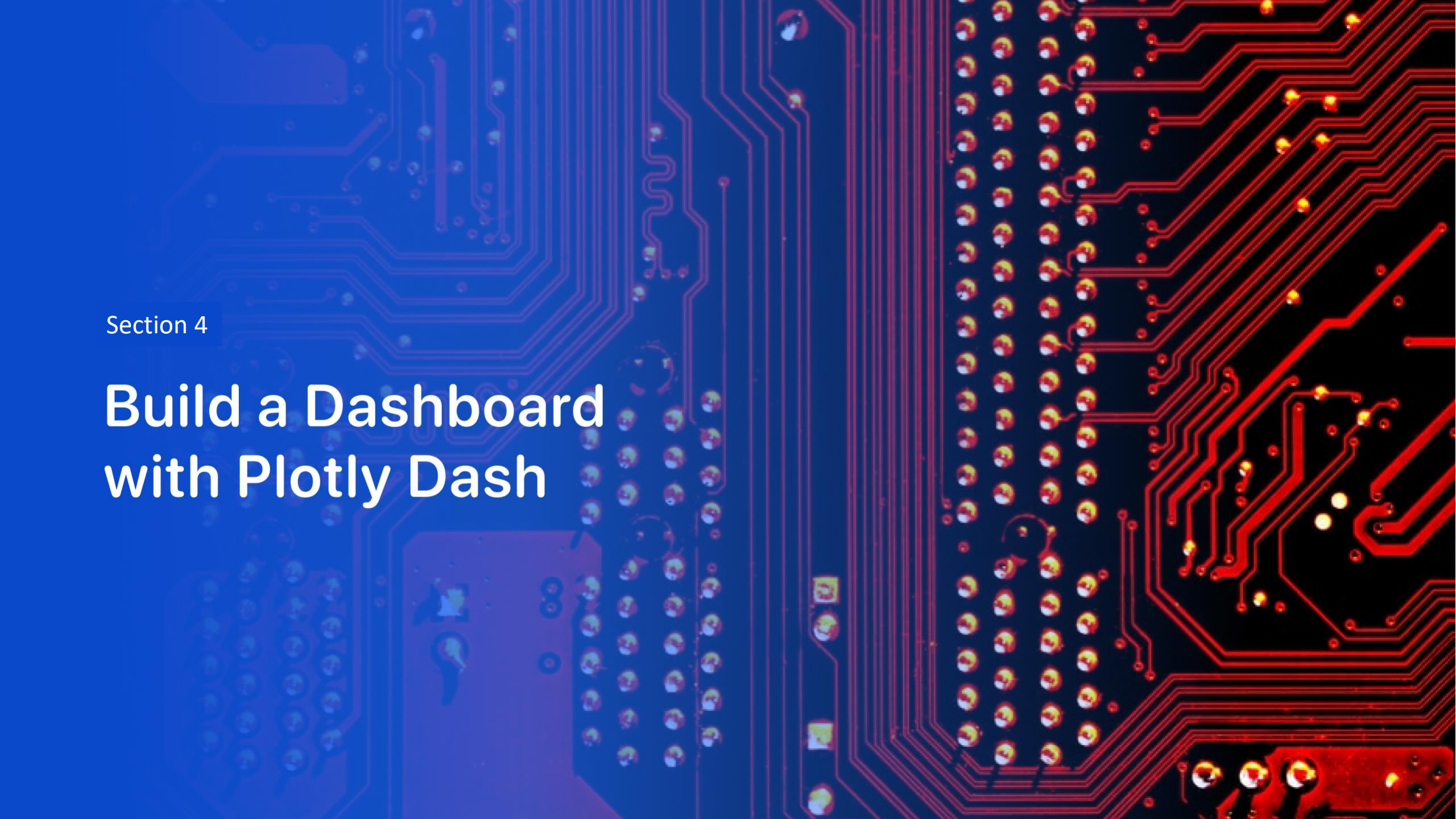


CCAFS LC-40 Launch Site

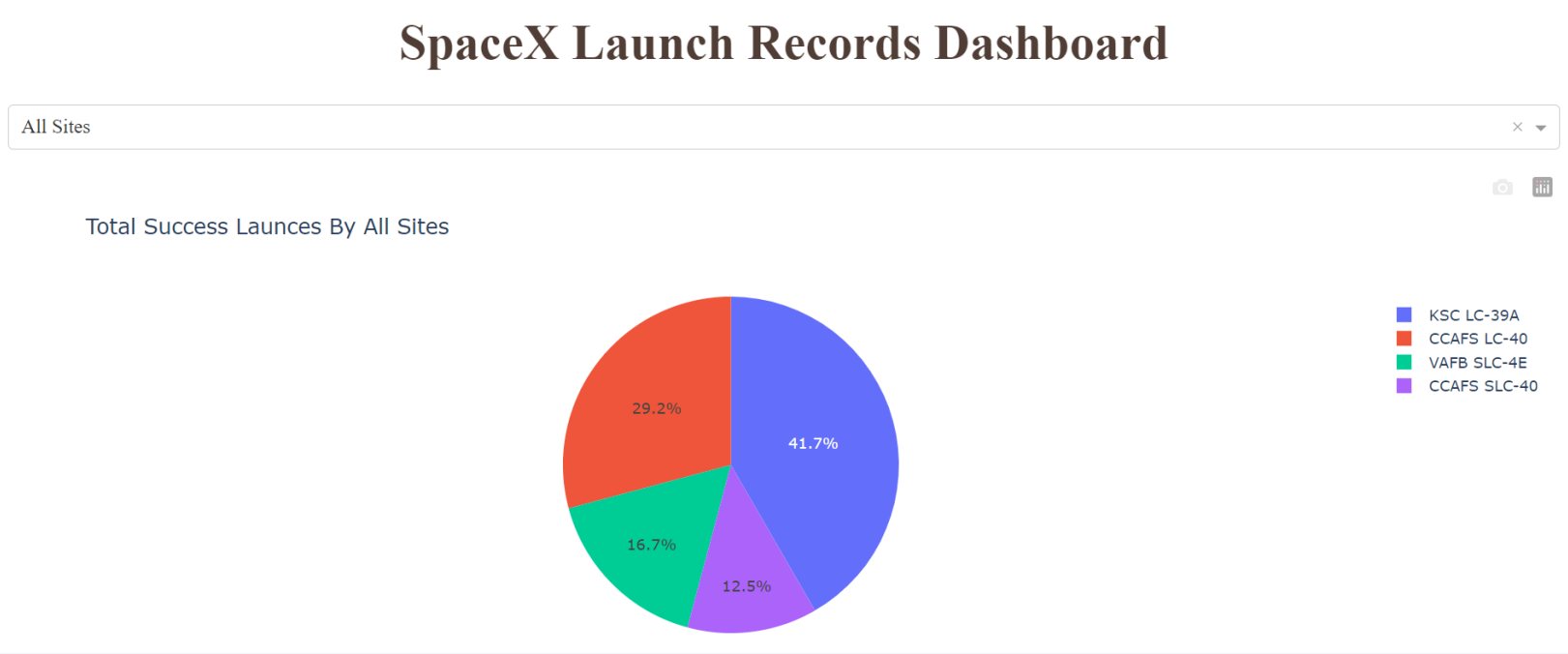
SpaceX Falcon9 – Launch Site to proximity Distance Map



Cities are located far away from the Launch Sites chosen by SPACEX in order to minimize the impact of any accidental events (like crushing) on the population living near by and also on the infrastructures. For this purpose, Launch Sites are strategically located near the coastline, railroad, and highways in order to provide easy access to resources.


Section 4
Build a Dashboard with Plotly Dash

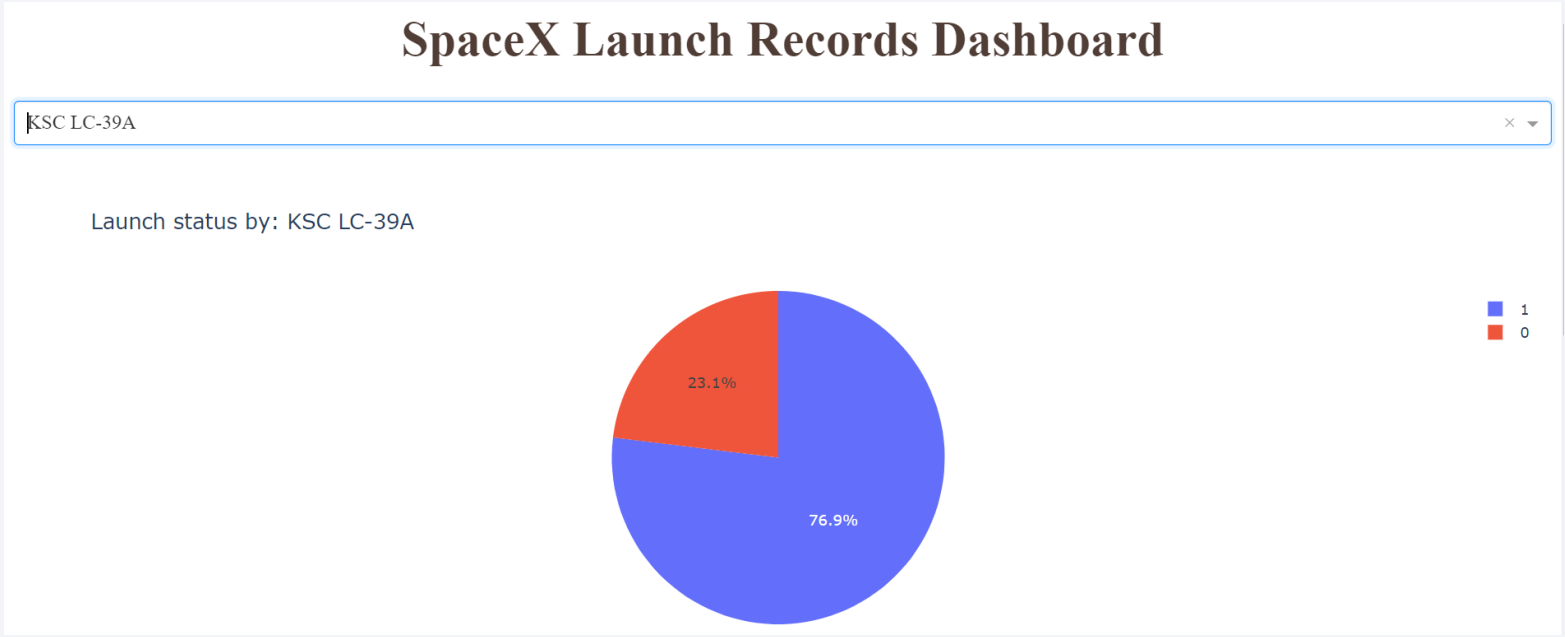
Launch Success Counts For All Sites



As said before :

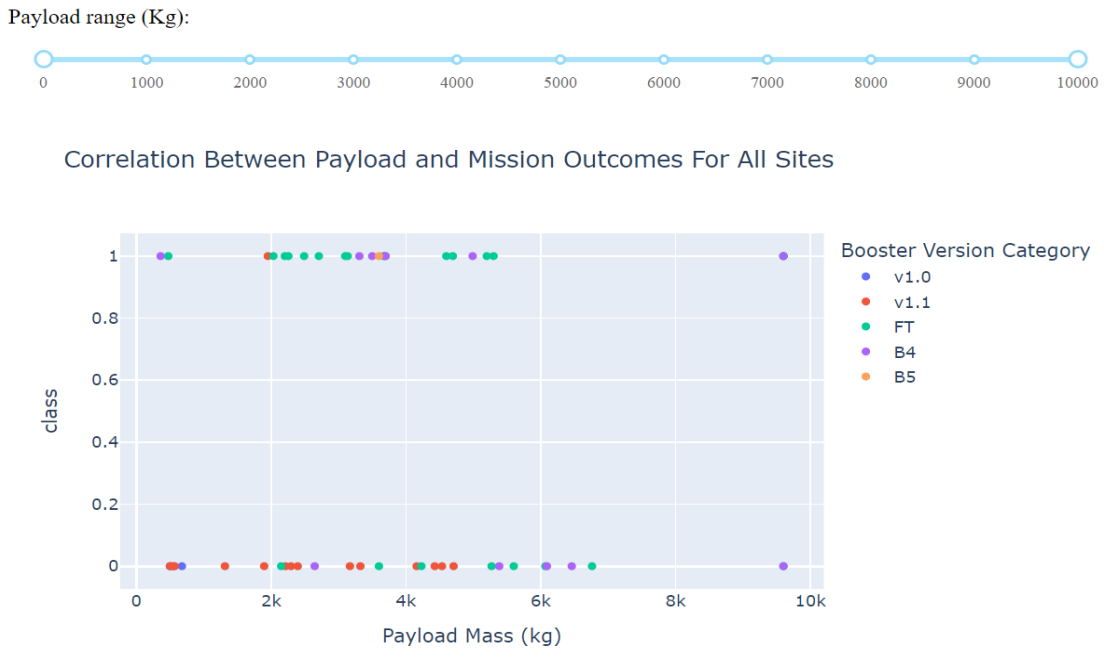
- Launch Site 'KSC LC-39A' has the highest launch success rate
- Launch Site 'CCAFS SLC- 40' has the lowest launch success rate

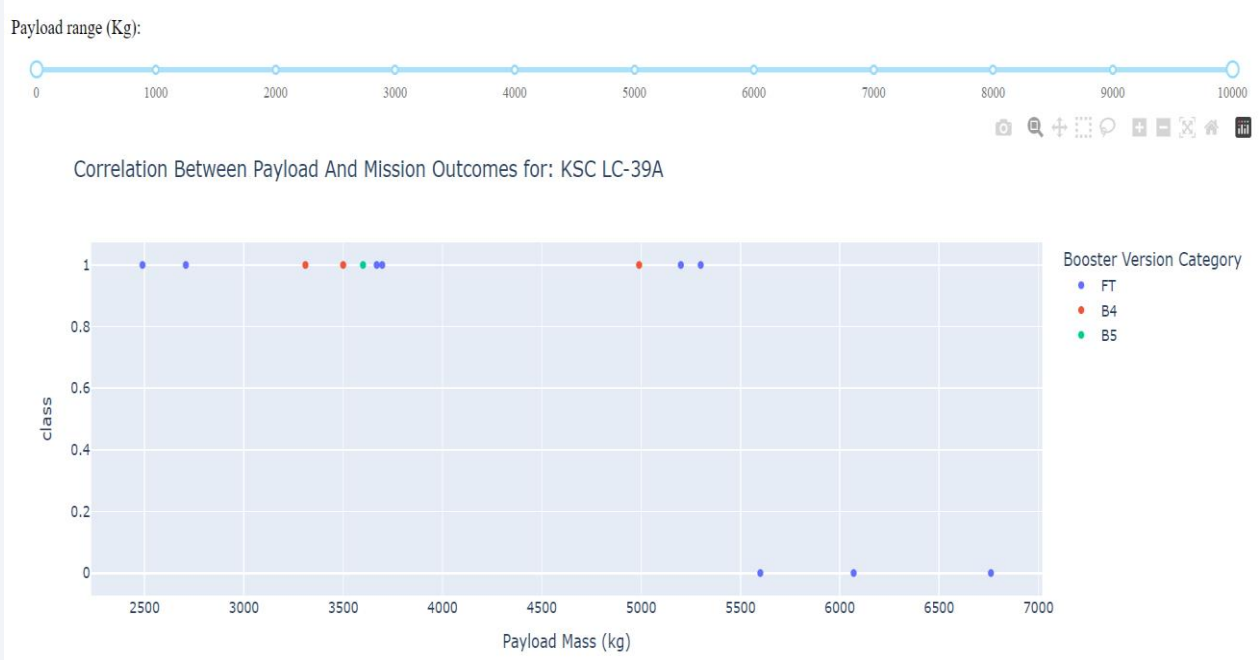
Launch Site with Highest Launch Success Ratio



- KSC LC-39A Launch Site has the highest launch success rate :
  - ➢ Launch success rate is 76.9%
  - ➢ Launch failure rate is 23.1%

Payload vs. Launch Outcome Scatter Plot for All Sites and for KSC LC-39A





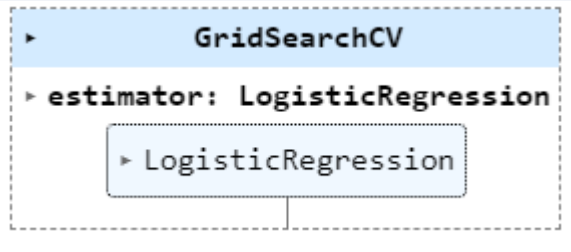
- Most successful launches were when the payload range from 2k to 6k.
- Booster version category ‘FT’ has the most successful launches in the playload range (2k -6k)
- B4 is the only booster with a success launch when payload is greater than 6k

Section 5

Predictive Analysis (Classification)

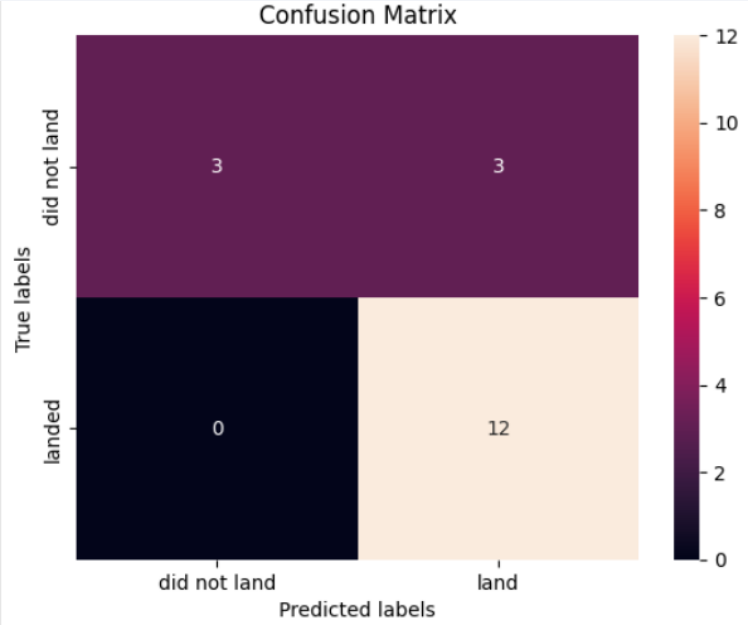
Classification Accuracy

1. Logistic regression (LR)




```
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
```




```
print("Test data accuracy score using Logistic Regression: ", logreg_cv.score(X_test, Y_test))
```

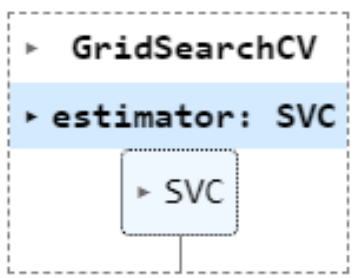
```
Test data accuracy score using Logistic Regression:  0.8333333333333334
```

Accuracy :

- **Training set : 84.64%**
- **Test set: 83.33%**

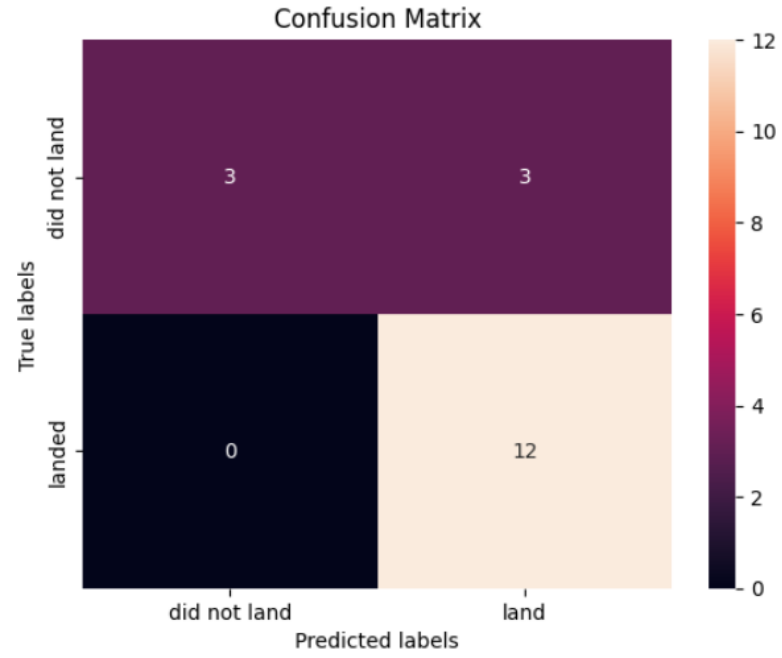
Classification Accuracy

2. support vector machine (SVM)




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

```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
accuracy : 0.8482142857142856
```




```
print("Test data accuracy score using support vector machine: ", svm_cv.score(X_test, Y_test))
```

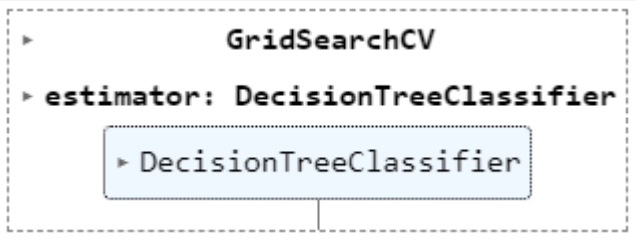
```
Test data accuracy score using support vector machine:  0.8333333333333334
```

Accuracy :

- **Training set : 84.82%**
- **Test set: 83.33%**

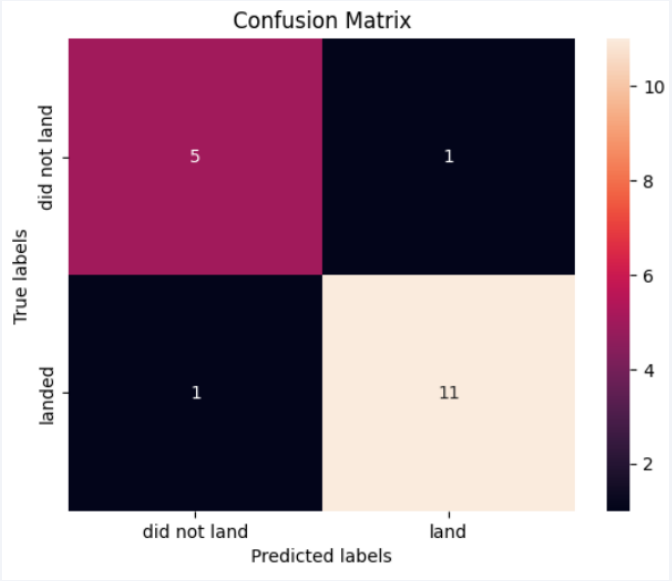
Classification Accuracy

3. Decision Tree




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

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 4, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 5, 'splitter': 'r
andom'}
accuracy : 0.8875
```




```
print("Test data accuracy score using decision tree classifier : ", tree_cv.score(X_test, Y_test))
```

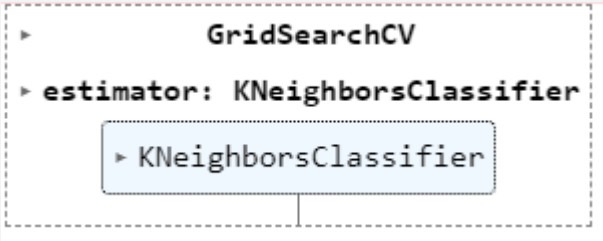
```
Test data accuracy score using decision tree classifier :  0.8888888888888888
```

Accuracy :

- **Training set : 88.75%**
- **Test set: 88.88%**

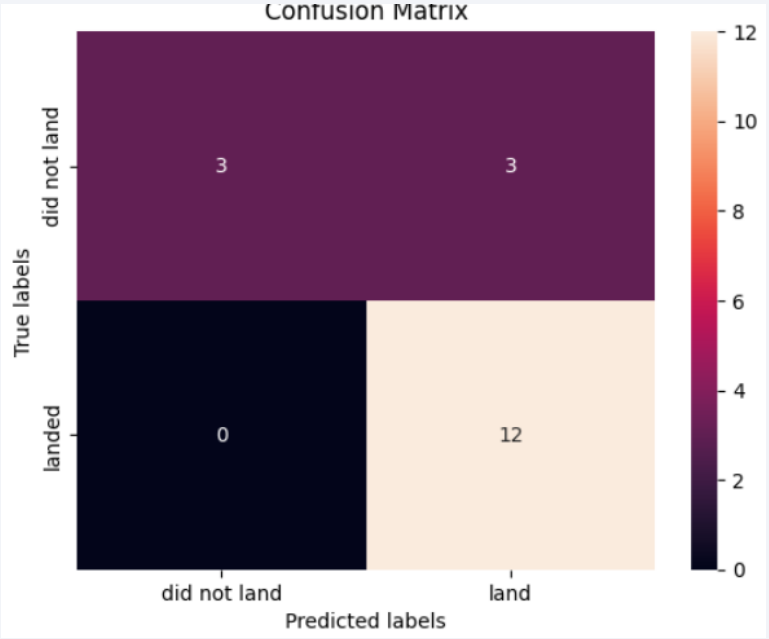
Classification Accuracy

4. k nearest neighbors (KNN)




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

```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}
accuracy : 0.8482142857142858
```




```
print("Test data accuracy score using decision tree classifier : ", knn_cv.score(X_test, Y_test))
```

```
Test data accuracy score using decision tree classifier :  0.8333333333333334
```

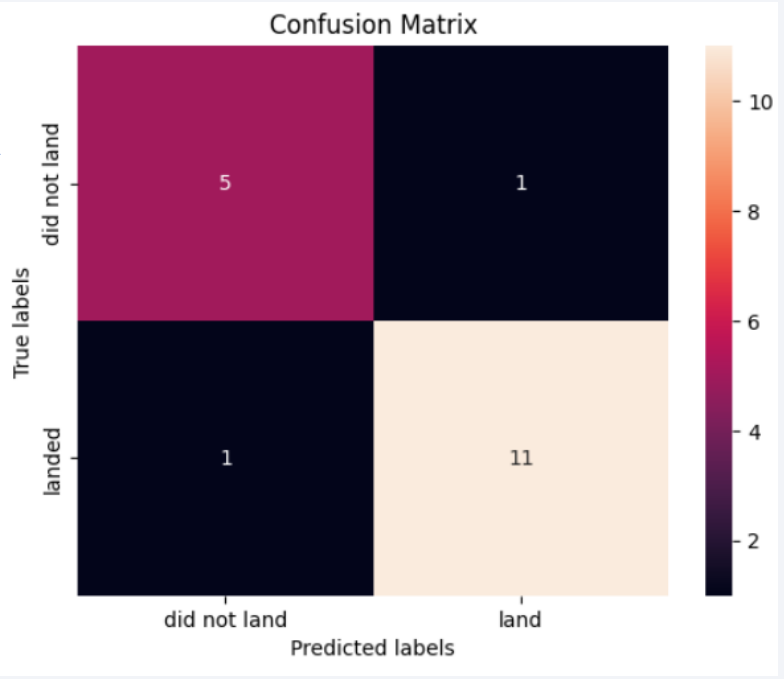
Accuracy :

- **Training set : 84.82%**
- **Test set: 83.33%**

Best Machine learning Model

- The best performing machine learning for this case study is Decision Tree with an accuracy of 88.89% (test dataset)

	Machine Learning Model	Accuracy Score	Test Data Accuracy Score
0	Decision Tree	0.887500	0.888889
1	KNN	0.848214	0.833333
2	SVM	0.848214	0.833333
3	Logistic Regression	0.846429	0.833333



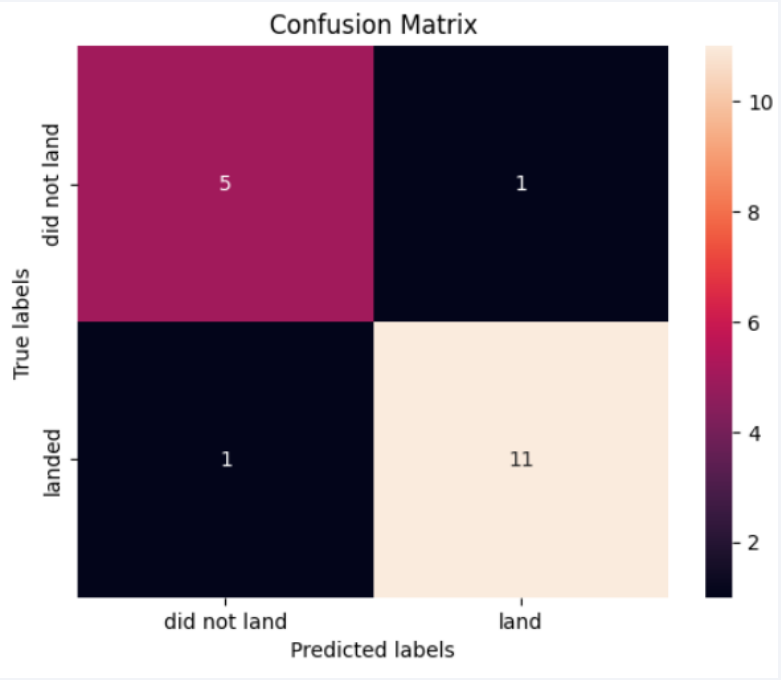
Conclusions

- In this project, we established a comprehensive analysis of SpaceX's launch data, aiming to gain insights into the company's launch history, success rates, and payload trends.
- Even though Success rates appears to go up as Payload increases for certain orbits but there is no clear correlation between Payload mass and success rates.
- Launch success rate kept increasing from 2013 up to 2020.
- Launch Site ‘KSC LC-39A’ has the highest launch success rate and Launch Site ‘CCAFS SLC-40’ has the lowest one.
- Orbits ES-L1, GEO, HEO, and SSO have the highest launch success rates and orbit GTO the lowest.
- Lunch sites are located strategically away from the cities and are particularly closer to coastline, railroads, and highways.

Conclusions

- The best performing Machine Learning Classification Model is the Decision Tree with an accuracy of about 88.89 %.

	Machine Learning Model	Accuracy Score	Test Data Accuracy Score
0	Decision Tree	0.887500	0.888889
1	KNN	0.848214	0.833333
2	SVM	0.848214	0.833333
3	Logistic Regression	0.846429	0.833333



Appendix

- Helper functions for SPACEX API scraping

```
def getBoosterVersion(data):
    for x in data['rocket']:
       if x:
        response = requests.get("https://api.spacexdata.com/v4/rockets/"+str(x)).json()
        BoosterVersion.append(response['name'])
```

```
def getLaunchSite(data):
    for x in data['launchpad']:
       if x:
        response = requests.get("https://api.spacexdata.com/v4/launchpads/"+str(x)).json()
        Longitude.append(response['longitude'])
        Latitude.append(response['latitude'])
        LaunchSite.append(response['name'])
```

```
def getPayloadData(data):
    for load in data['payloads']:
       if load:
        response = requests.get("https://api.spacexdata.com/v4/payloads/"+load).json()
        PayloadMass.append(response['mass_kg'])
        Orbit.append(response['orbit'])
```

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

Appendix

- Helper functions for webscraping from wikipedia

```
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders collapsible")):
   # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number corresponding to launch a number
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
        else:
            flag=False
        #get table element
        row=rows.find_all('td')
        #if it is number save cells in a dictonary
        if flag:
            extracted_row += 1
            # Flight Number value
            # TODO: Append the flight_number into launch_dict with key `Flight No.`
            #print(flight_number)
            launch_dict['Flight No.'].append(flight_number)
            datatimelist=date_time(row[0])
```

```
            # Date value
            # TODO: Append the date into launch_dict with key `Date`
            date = datatimelist[0].strip(',')
            launch_dict['Date'].append(date)
            #print(date)
```

```
            # Time value
            # TODO: Append the time into launch_dict with key `Time`
            time = datatimelist[1]
            launch_dict['Time'].append(time)
            #print(time)
```

```
            # Booster version
            # TODO: Append the bv into launch_dict with key `Version Booster`
            bv=booster_version(row[1])
            if not(bv):
                bv=row[1].a.string
            print(bv)
            launch_dict['Version Booster'].append(bv)
```

```
            # Launch Site
            # TODO: Append the bv into launch_dict with key `Launch Site`
            launch_site = row[2].a.string
            launch_dict['Launch site'].append(launch_site)
            #print(launch_site)
```

```
            # Payload
            # TODO: Append the payload into launch_dict with key `Payload`
            payload = row[3].a.string
            launch_dict['Payload'].append(payload)
            #print(payload)
```

```
            # Payload Mass
            # TODO: Append the payload_mass into launch_dict with key `Payload mass`
            payload_mass = get_mass(row[4])
            launch_dict['Payload mass'].append(payload_mass)
            #print(payload)
```

```
            # Orbit
            # TODO: Append the orbit into launch_dict with key `Orbit`
            orbit = row[5].a.string
            launch_dict['Orbit'].append(orbit)
            #print(orbit)
```

```
            # Customer
            # TODO: Append the customer into launch_dict with key `Customer`
            try:
                customer = row[6].a.string
            except:
                customer = 'Various'
            launch_dict['Customer'].append(customer)
```

```
            #print(customer)
```

```
            # Launch outcome
            # TODO: Append the launch_outcome into launch_dict with key `launch_outcome`
            launch_outcome = list(row[7].strings)[0]
            launch_dict['Launch outcome'].append(launch_outcome)
            #print(launch_outcome)
```

```
            # Booster landing
            # TODO: Append the launch_outcome into launch_dict with key `Booster_landing`
            booster_landing = landing_status(row[8])
            launch_dict['Booster landing'].append(booster_landing)
            #print(booster_landing)
```

Appendix

- Meaning of each landing_outcome

`True Ocean` means the mission outcome was successfully landed to a specific region of the ocean while `False Ocean` means the mission outcome was unsuccessfully landed to a specific region of the ocean. `True RTLS` means the mission outcome was successfully landed to a ground pad `False RTLS` means the mission outcome was unsuccessfully landed to a ground pad. `True ASDS` means the mission outcome was successfully landed to a drone ship `False ASDS` means the mission outcome was unsuccessfully landed to a drone ship. `None ASDS` and `None None` these represent a failure to land.

Thank you!