IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Nauman Rasheed

16th May 2025

Outline

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

Executive Summary

Summary of methodologies

- Data collection
- Data wrangling
- Exploratory data analysis
- Data visualization
- Visual analytics
- Predictive analysis

Summary of all results

- Exploratory data analysis results
- Geospatial analysis results
- Interactive dashboard
- Predictive analysis results

Introduction

- Project background and context
  - Space X is able to launch rockets at a significantly cheaper cost (USD 62 million) than other providers (USD 165/- million)
  - This can be attributed to the recovery and reuse of the first stage of the Falcon 9 rockets
  - By determining if the first stage of the Falcon 9 rockers will land, we can determine the cost of the launch.
  - This will allow an alternate company to determine if they should bid against Space X for a rocket launch
- Problems you want to find answers
  - What factors (location, parts, orbit etc.) determine if the first stage of the Falcon 9 rocket will land?
  - Which predictive model best classifies successful and unsuccessful recovery of the first stage of Falcon 9 rockets?

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - Data collection using API
  - Data collection using web scrapping
- Perform data wrangling
  - Remove null values
  - Determine distinct values for launch sites and orbit to determine:
    - Number of launches per site
    - Number and occurrence of each orbit
    - Number and occurrence of mission outcome of the orbits
  - Landing outcome label from Outcome column
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - Preprocess the data
  - Split the data into training and test set
  - Train different classification models
  - Hyper tune the parameters to find the best ones
  - Determine the accuracy of the model and make the confusion matrix

Data Collection

- We will use the Space X Rest API to retrieve the data about Flight Number, Date, Booster Version (rocket type), Payload mass, Orbit, Launch Site, Outcome, Number of Flights, Parts reused and type of Landing Pad used. We will then convert this data into a pandas dataframe
- Then we will use Data Collection using Web Scrapping to retrieve information from Space X Wikipedia page about Launch Sites, Payload type, Customer and Landing Outcome

Data Collection – SpaceX API

- Make a get request to SpaceX Rest API
- Convert the response to a json file
- Convert the json file to a pandas dataframe
- Make lists for the columns in the data
- Use predefined functions **(see appendix)** to retrieve data and fill the lists
- Use the lists as values in the dictionary to construct the dataset
- Create a pandas dataframe from the dictionary dataset
- Filter the data to include only Falcon 9 launches
- Reset the Flight Number column
- Replace missing values of PayloadMass column with the mean of that column
- GitHub URL: https://github.com/Nauman89/IBM-Data-Science-Certification-Capstone-Project/blob/5c2d346f293deefc44403d4fbd172d65252af234/1.Data%20Collection%20with%20API.ipynb

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

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

In [16]:
# Use json_normalize meethod to convert the json result into a dataframe
data = pd.json_normalize(response.json())

In [20]:
#Global variables
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
ReusedCount = []
Serial = []
Longitude = []
Latitude = []

n [22]:
# Call getBoosterVersion
getBoosterVersion(data)

In [24]:
# Call getLaunchSite
getLaunchSite(data)

In [25]:
# Call getPayloadData
getPayloadData(data)

In [26]:
# Call getCoreData
getCoreData(data)

In [27]:
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}

In [71]:
# Create a data from launch_dict
df = pd.DataFrame.from_dict(launch_dict)

In [73]:
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = df[df['BoosterVersion'] != 'Falcon 1']
data_falcon9.head()

In [74]:
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9

In [77]:
# Calculate the mean value of PayloadMass column
mean = data_falcon9['PayloadMass'].mean()

# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'] = data_falcon9['PayloadMass'].replace(to_replace = np.nan , value = mean)
```

Data Collection - Scraping

- We use Web Scraping to collect Falcon 9 historical launch records from a Wikipedia page
- Steps:
    1. Request the HTML page from a Static URL and assign the response to an object
    2. Create a BeautifulSoup object from the HTML response object. Then find all the tables in the HTML page
    3. Collect all the column header names
    4. Use columns names as keys in a dictionary and use custom functions to parse all launch tables to the dictionary
    5. Convert the dictionary to a Pandas DataFrame ready for export

- Github link: https://github.com/Nauman89/IBM-Data-Science-Certification-Capstone-Project/blob/5c2d346f293deefc44403d4fbd172d65252af234/2.Data%20Collection%20Using%20Web%20Scraping.ipynb

In [4]:
```
static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"
```

In [8]:
```
# use requests.get() method with the provided static_url
# assign the response to a object
response = requests.get(static_url)
```

In [11]:
```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
html_content = response.text
soup = BeautifulSoup(html_content, 'html.parser')
```

In [13]:
```
# 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')
```

In [17]:
```
column_names = []

# Apply find_all() function with `th` element on first_launch_table
th_elements = first_launch_table.find_all('th')

# Iterate each th element and apply the provided extract_column_from_header() to get a column name
for th in th_elements:
    name = extract_column_from_header(th)

# Append the Non-empty column name (`if name is not None and len(name) > 0`) into a list called column_names
    if name is not None and len(name) > 0:
        column_names.append(name)
```

In [19]:
```
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'] = []
```

In [25]:
```
df= pd.DataFrame({ key:pd.Series(value) for key, value in launch_dict.items() })
```

Data Wrangling

- Using value_counts() we determine:
  1. Number of launches on each site
  2. Number and occurrence of each orbit
  3. Number and occurrence of landing outcome per orbit type

- Understand the context of the dependent variable i.e. Outcome:
  1. True Ocean: the mission outcome was successfully landed to a specific region of the Ocean
  2. False Ocean: the mission outcome was unsuccessfully landed to a specific region of the Ocean
  3. True RTLS: the mission outcome was successfully landed to a ground pad
  4. False RTLS: the mission outcome was unsuccessfully landed to a ground pad
  5. True ADS: the mission outcome was successfully landed on a drone ship
  6. False ADS: the mission outcome was successfully landed on a drone ship

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

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

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

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

```
landing_outcomes
```

```
Outcome
True ASDS      41
None None      19
True RTLS      14
False ASDS      6
True Ocean      5
False Ocean     2
None ASDS       2
False RTLS      1
```

Data Wrangling

- We need to convert these outcomes to binary variables where 1 represents successful landing and 0 represents unsuccessful landing:
  1. Define a set of unsuccessful outcomes in the variable 'bad_outcome'
  2. Create a list 'landing_class' where the element is 0 if the corresponding row in Outcome is in the 'bad_outcome'
  3. Create a 'Class' column that contains the values from the list 'landing_class'
  4. Export the DataFrame as a .csv file

  Github link: https://github.com/Nauman89/IBM-Data-Science-Certification-Capstone-Project/blob/5c2d346f293deefc44403d4fbd172d65252af234/3.Data%20Wrangling.ipynb

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

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

landing_class = [0 if outcome in bad_outcomes else 1 for outcome in df['Outcome']]
```

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

EDA with Data Visualization

- Scatter plots to visualize the correlation between:
  - ‘Flight Number’ and ‘Launch Site’
  - ‘Payload’ and ‘Launch Site’
  - ‘Orbit Type’ and ‘Flight Number’
  - ‘Payload’ and ‘Ortbit Type’
- Bar chart to observes the relationship between ‘Success Rate' and ‘Orbit Type’
- Line chart to see the change in ‘Success Rate’ over the years
- Git hub link: https://github.com/Nauman89/IBM-Data-Science-Certification-Capstone-Project/blob/5c2d346f293deefc44403d4fbd172d65252af234/5.EDA%20with%20Data%20Visualization.ipynb

EDA with SQL

- Using SQL queries, we:
  - Display the names of the unique launch sites
  - Display 5 records where launch site begins with ‘CAA’
  - Display the total payload mass carried by NASA (CRS)
  - Display average payload mass carried by booster version F9 v1.1
  - List the date of the first successful landing outcome on ground pad
  - List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000
  - List the total number of successful and failure mission outcomes
  - List all the booster_versions that have carried the maximum payload mass.
  - List the records which will display the month names, failure landing_outcomes in drone ship ,booster versions, launch_site for the months in year 2015
  - 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.
- GitHub link: https://github.com/Nauman89/IBM-Data-Science-Certification-Capstone-Project/blob/5c2d346f293deefc44403d4fbd172d65252af234/4.EDA%20with%20SQL.ipynb

Build an Interactive Map with Folium

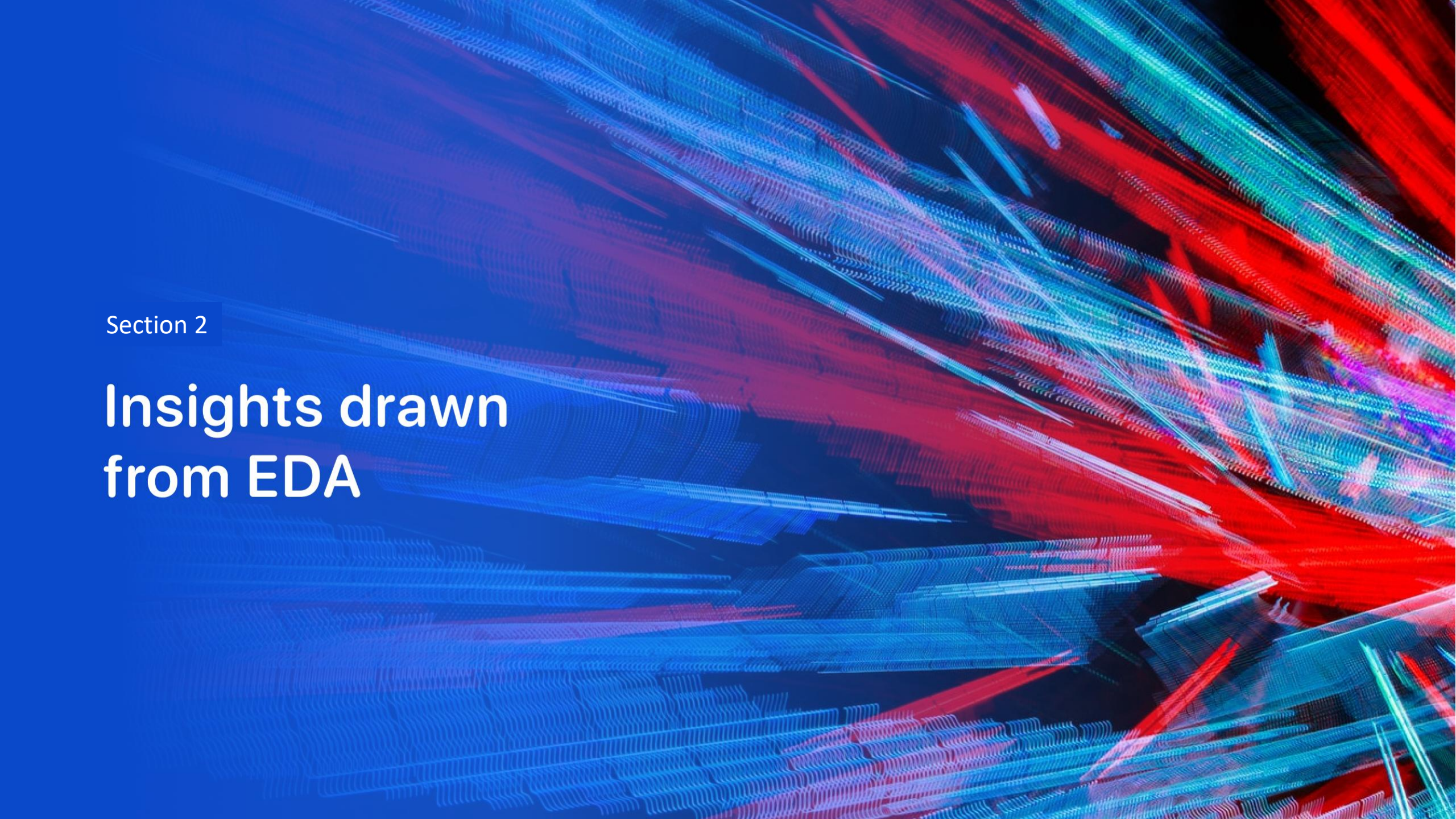
- Mark all launch sites on a map:
  - Initialize the map using a Folium Map object
  - Add a folium.Circle and folium.Marker for each launch site on the launch map
- Mark successful and failed launches for each site on the map:
  - Since most sites have the same coordinates, it makes sense to cluster them
  - Before clustering, we assign green marker colour successful launch and red marker colur to an unsuccessful launch
  - We add folium.Marker to MarkerCluster() object to put launches into clusters
  - Create and icon as a text label, assigning the icon_colour as the marker_colour
- Calculate the distances between the launch site to its proximities:
  - Using Latitude and Longitude values of the launch sites, we can determine proximities of the launch sites
  - Create a folium.Marker object to show the distance between the launch sites
  - Create a folium.Polyline and add it to the map to display the distance between launch sites
- GitHub link: https://github.com/Nauman89/IBM-Data-Science-Certification-Capstone-Project/blob/5c2d346f293deefc44403d4fbd172d65252af234/6.Visual%20Analytics%20with%20Folium.ipynb

Build a Dashboard with Plotly Dash

- Create a pie chart, showing the total successful launchers per site to:
  - Identify the most successful sites
  - Use filters to see success/failure ratios at each site
- Create a scatter plot to show the correlation outcome and payload mass. We can fiter the data by:
  - Ranges of payload mass
  - Booster versions

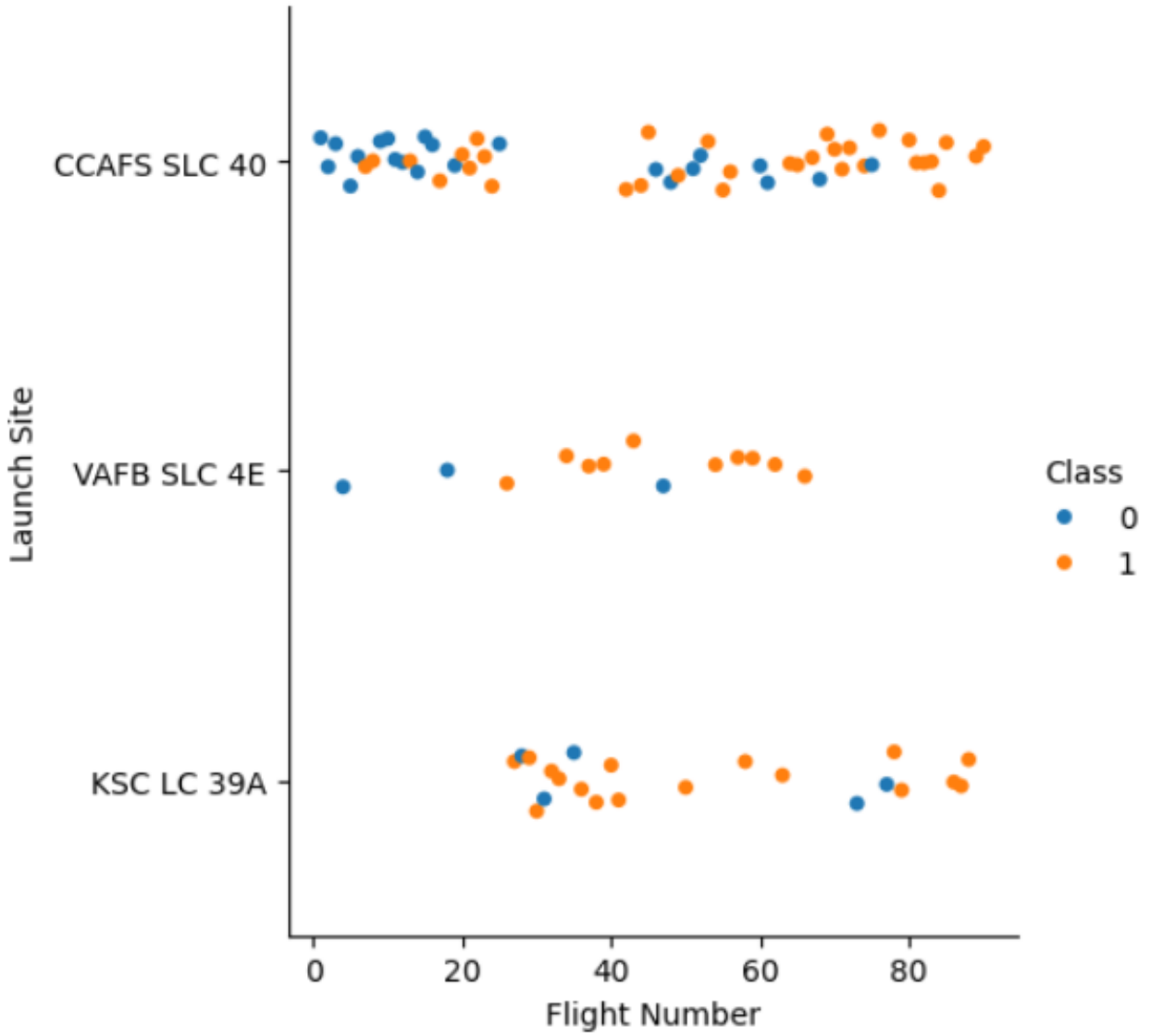
Predictive Analysis (Classification)

- Model development:
    1. Load dataset
    2. Preprocess the data
    3. Split the data into training and test sets
    4. Choose the appropriate machine learning algorithm
    5. Create a GridSearch object and a dictionary of parameters
    6. Fit the object to the parameters
    7. Use the training data set to train the model
- Model Evaluation:
    1. Use GridSearch object check tuned hyperparameters and the accuracy socres
    2. Plot and examine the Confusion Matrix
- Finding the best classification model:
    1. The model with the highest accuracy score is the best performing model


Section 2
Insights drawn from EDA

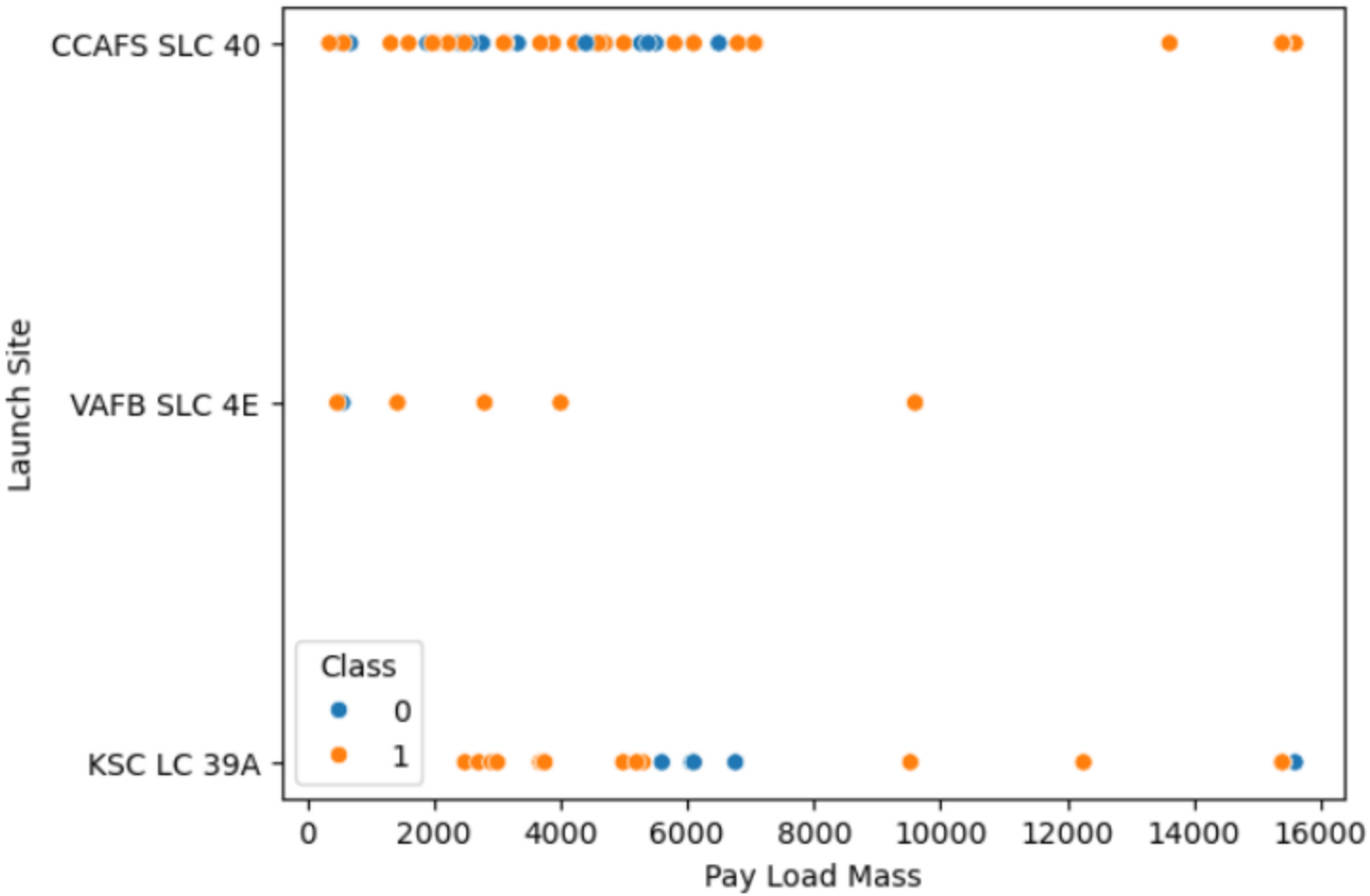
Flight Number vs. Launch Site

- As the number of flights increases, the rate of success at a launch site increase.
- Early flights from CCAFS SLC 40 and VAFB SLC 4E were generally unsuccessful.
- After flight number 30, flights have been more successful from each site



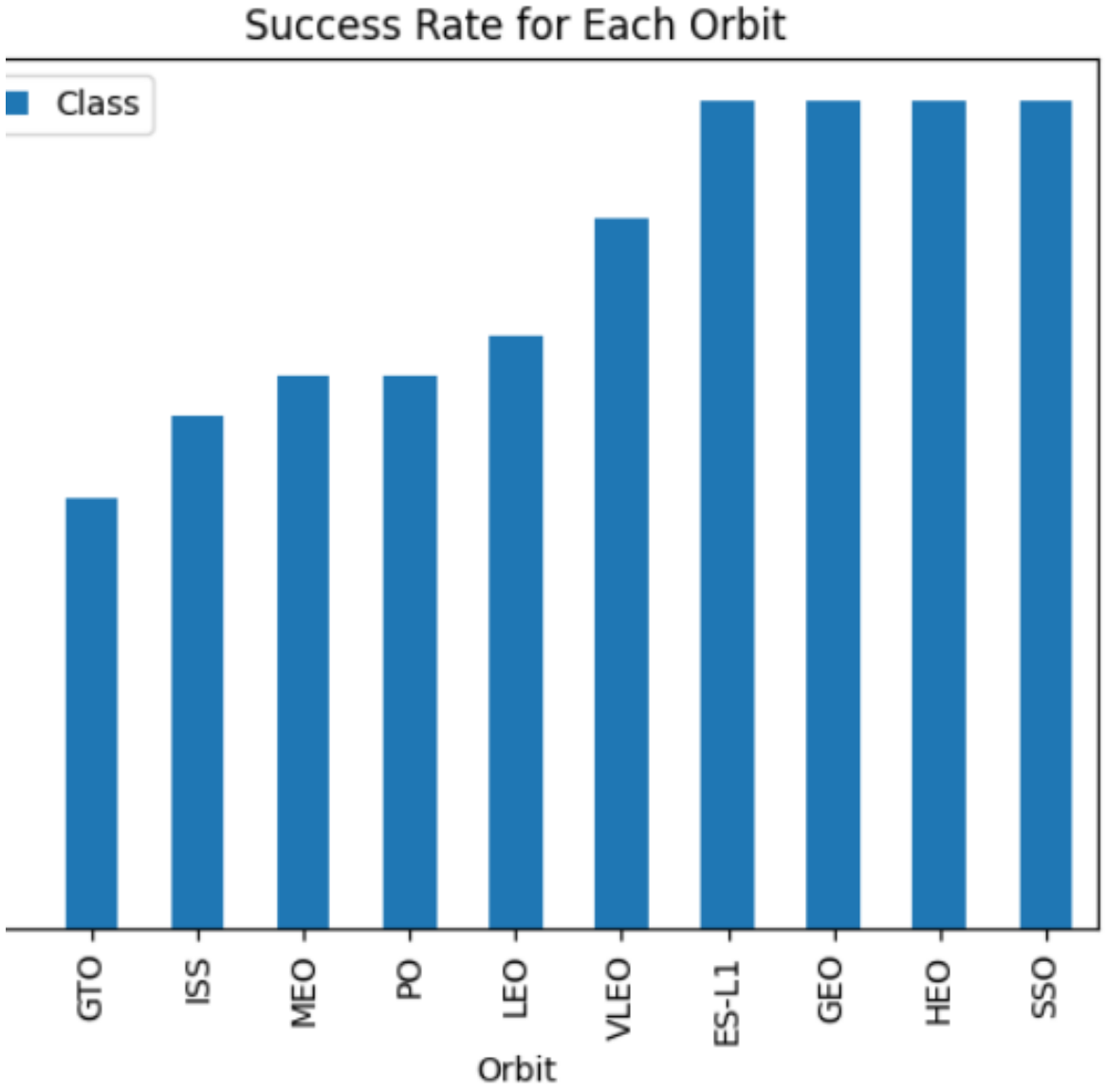
Payload vs. Launch Site

- Flights with heavier payloads (above 7000 kgs) are tend to be more successful, however, we have less data for such flights
- There is no clear correlation between payload mass and launch site
- All sites launched a variety of payloads



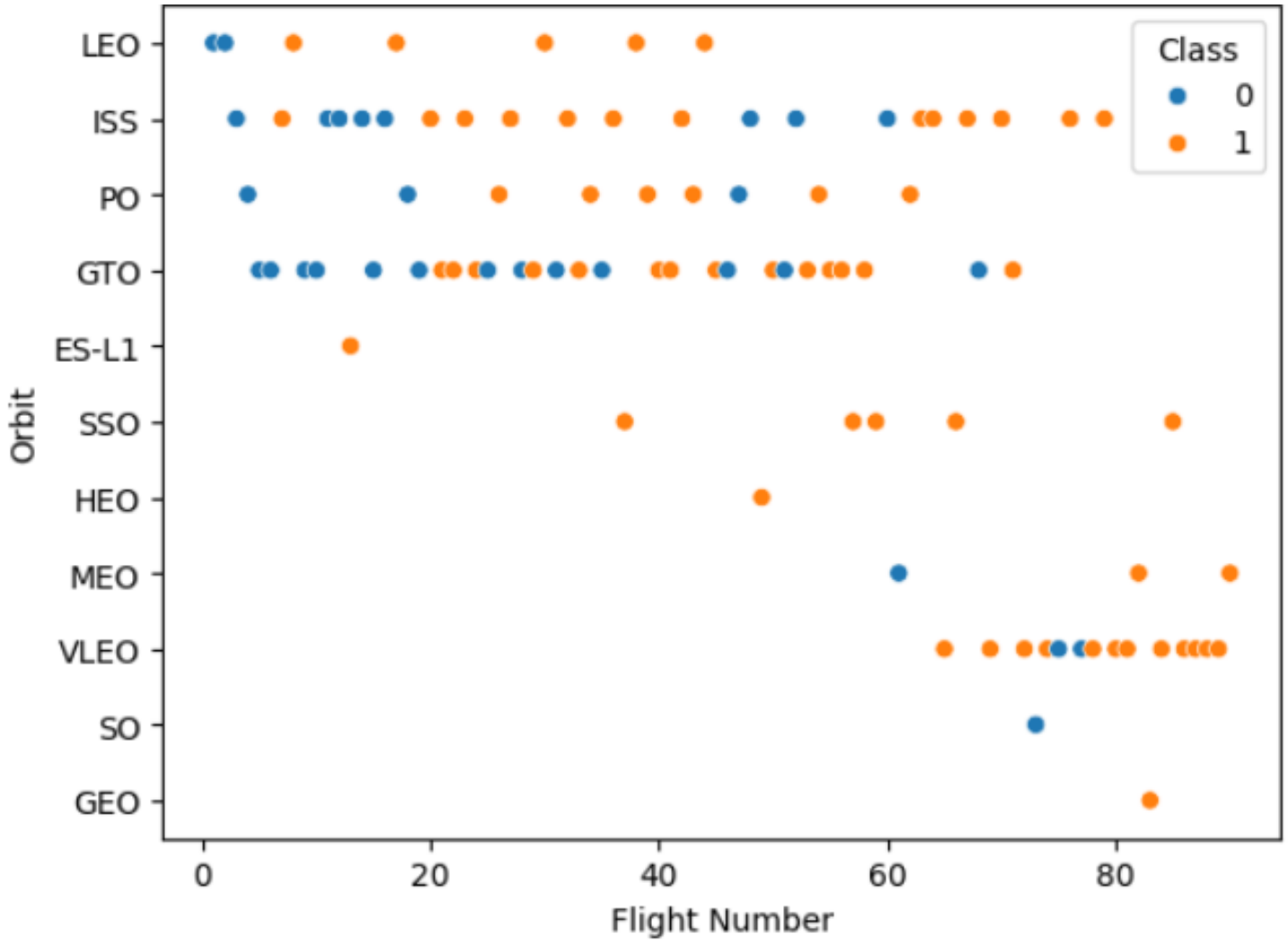
Success Rate vs. Orbit Type

- SO (Heliocentric Orbit) success rate is the lowest (0%)
- ES-L1 (Earth-Sun First Lagrangian Point), GEO (Geostationay Orbit), HEO (High Earth Orbit) and SSO (Sun-Syndrome Orbit) have 100% success rate



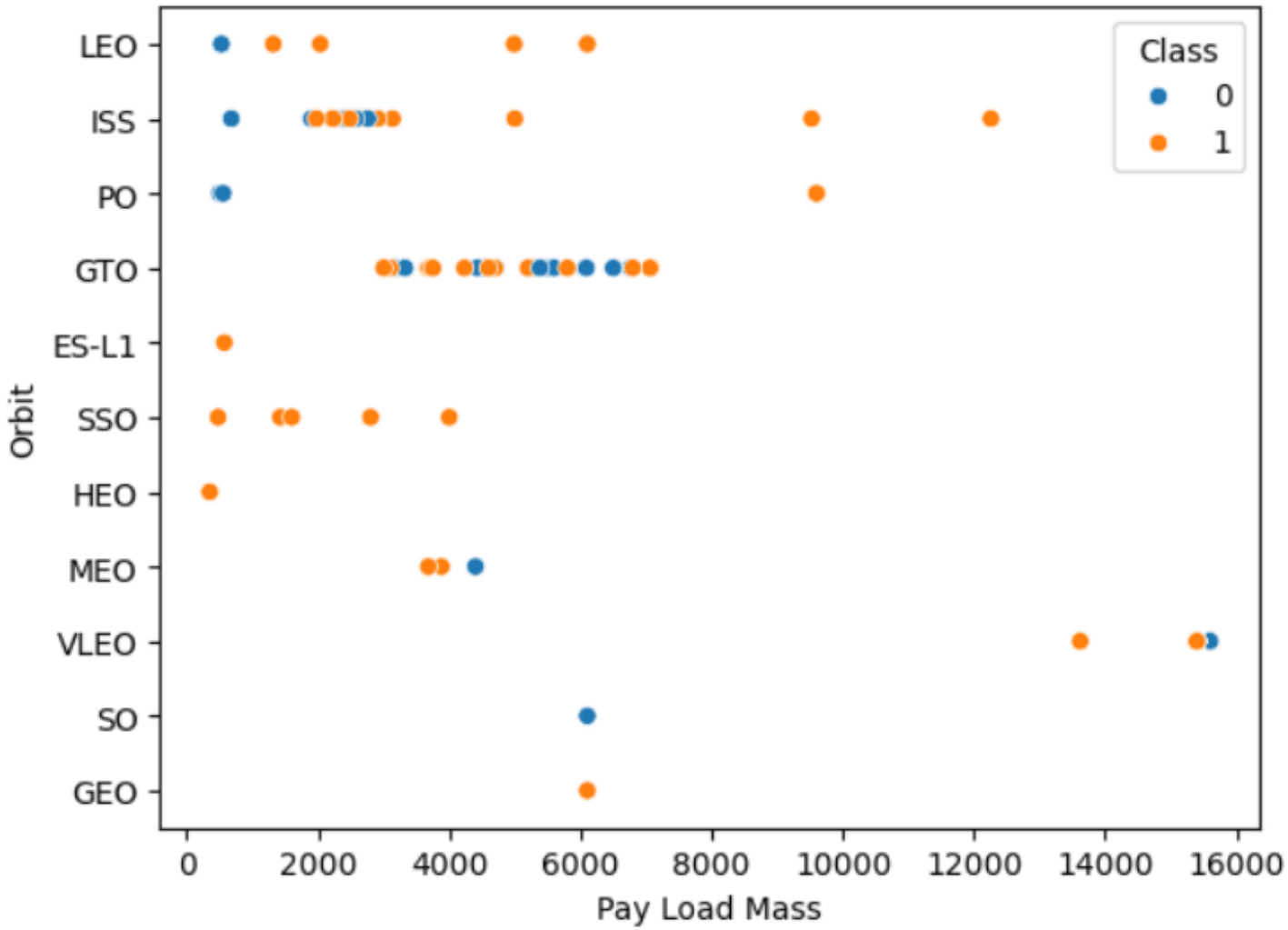
Flight Number vs. Orbit Type

- ES L1, HEO, GEO have just one flight which explains their 100% success rate.
- However, SSO has a 100% success rate with 5 flights
- As number of flights increase, the success rate increases, with the exception of GTO which shows no clear correlation with Flight Number



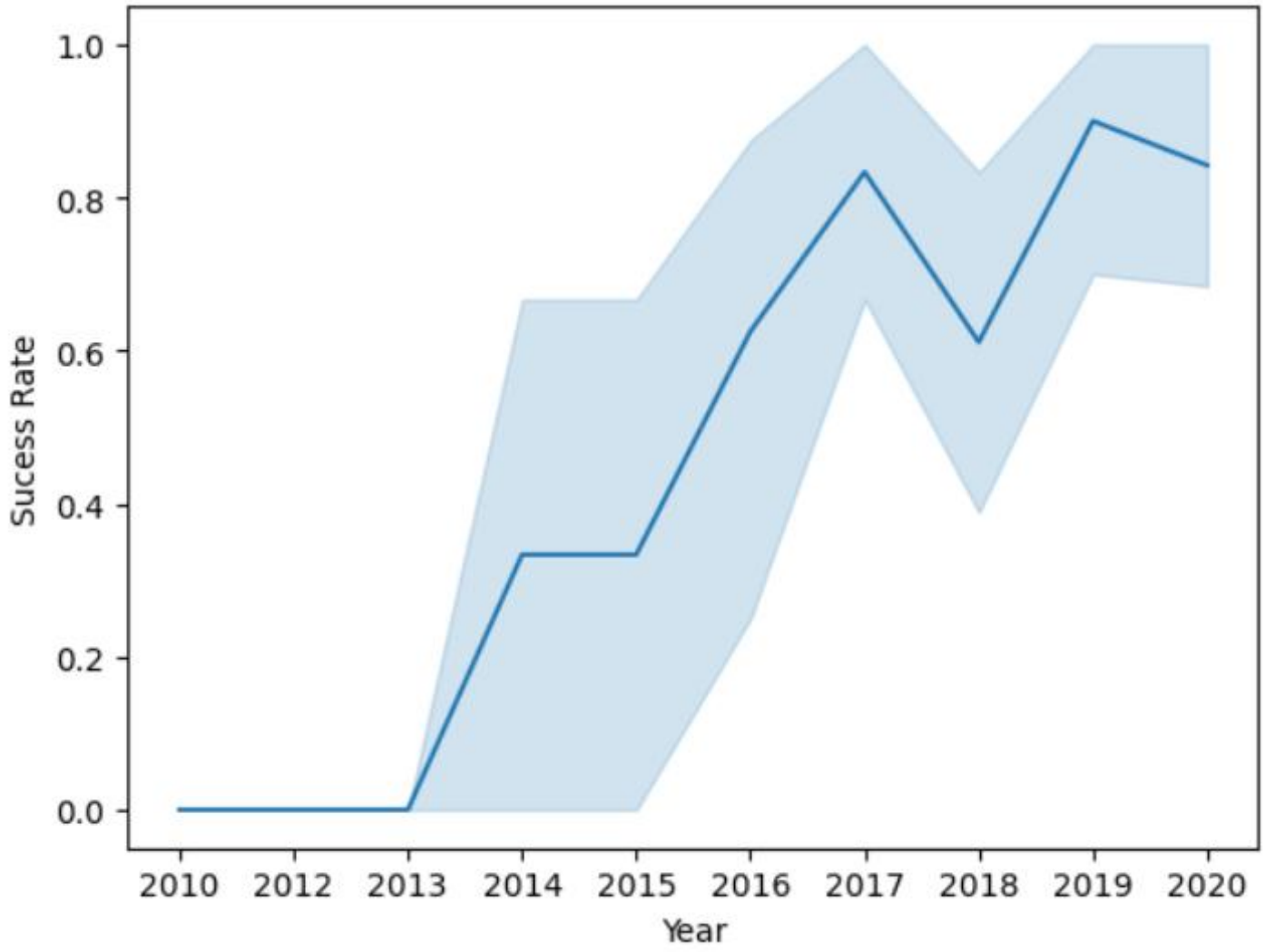
Payload vs. Orbit Type

- There is no clear relationship between payload and orbit type for GTO orbit
- LEO, ISS, PO are more successful at higher payloads, though we only have two observations for PO
- Flights in SO orbit have all been successful at lower payloads
- VLEO orbit only has trips in very high payloads



Launch Success Yearly Trend

- Since 2013, launch success rate has shown an overall increasing trend, with a minor dip observed from 2017 to 2018.
- Launch success rate has been above 50% since 2016.



All Launch Site Names

- Launch site names:
  1. CCAFS LC-40
  2. VSFB SLC-4E
  3. KSC LC-39A
  4. CCAFS SLC-4A

```
%sql SELECT Distinct Launch_Site From SPACEXTBL;
```

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

- Using 'DISTINCT' returns unique values from the Launch_Site column column of the SPACEXTBL table

Launch Site Names Begin with 'CCA'

```
%sql SELECT * From SPACEXTBL Where Launch_Site like 'CCA%' Limit 5
```

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outc
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (paracl
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 (paracl
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No att
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No att
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No att

- Using ‘like’ keyword ‘CCA%’ wildcard retrieves strings beginning with ‘CCA.’ Also, ‘Limit 5’ returns only 5 results

Total Payload Mass

```
%sql SELECT sum(PAYLOAD_MASS__KG_) from SPACEXTBL Where Customer = 'NASA (CRS)'
```

sum(PAYLOAD_MASS__KG_)
45596

- The ‘sum’ keyword is used to calculate the sum of the ‘PAYLOAD_MASS__KG_’ column from the SAPCEXTBL table. The ‘where’ keyword is used to filter out results to show the Customer ‘NASA (CRS)’

Average Payload Mass by F9 v1.1

```
%sql SELECT avg(PAYLOAD_MASS__KG_) from SPACEXTBL Where Booster_Version = 'F9 v1.1'
```

avg(PAYLOAD_MASS__KG_)
2928.4

- The ‘avg’ keyword calculates the average of the ‘PAYLOAD_MASS__KG_’ column from the ‘SPACEXTBL’ and the ‘where’ keyword filters out results to show only the ones with ‘Booster_Version = F9 v1.1’

First Successful Ground Landing Date

```
%sql Select min(Date) From SPACEXTBL Where Landing_Outcome = 'Success (ground pad)'
```

min(Date)
2015-12-22

- The ‘Where’ keyword filters the data to show the results with a successful ground pad landing
- The ‘min’ keyword fetches the minimum value or the first date from the ‘Date’ column in the ‘SPACEXTBL’ table after applying the abovesaid filter

Successful Drone Ship Landing with Payload between 4000 and 6000

```
%sql Select Distinct Booster_Version from SPACEXTBL Where Landing_Outcome = 'Success (drone ship)' andPAYLOAD_MASS__KG_ between 4000 and 6000
```

Booster_Version
F9 FT B1022
F9 FT B1026
F9 FT B1021.2
F9 FT B1031.2

- The ‘Where’ + ‘and’ keywords filter the data to show results where there was a successful drone ship landing of boosters carrying payload mass between 4000 kg to 6000 kg
- The ‘Distinct’ keyword selects unique values from the ‘Booster_Version’ column of the ‘SPACEXTBL’ table

Total Number of Successful and Failure Mission Outcomes

```
%sql Select substr(Mission_Outcome, 1, 7) as Mission_Outcome, count(*) from SPACEXTBL group by 1
```

Mission_Outcome	count(*)
Failure	1
Success	100

- The ‘substr’ keyword extracts a string from the Mission_Outcome’ Column with a length of 7.
- The ‘as’ keyword changes the column name to ‘Mission_Outcome’
- The ‘count’ keyword counts the total number of mission outcomes
- The ‘group by 1’ argument groups the data by the first column, i.e. Mission_Outcome

Boosters Carried Maximum Payload

```
%sql Select Distinct Booster_Version from SPACEXTBL Where PAYLOAD_MASS__KG_ = (select max(PAYLOAD_MASS__KG_) from SPACEXTBL)
```

- The ‘select’ statement in parentheses is a sub query that selects the maximum payload from the ‘PAYLOAD_MASS__KG_’ column
- The ‘Where’ keyword filters out the data the show the results in which the max payload is carried
- The ‘Distinct’ keyword selects unique results from the ‘Booster_Version’ column in the ‘SPACEXTBL’ table

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

2015 Launch Records

```
%sql Select substr (Date, 6, 2) as Month, Landing_Outcome, Booster_Version, Launch_Site From SPACEXTBL
```

```
Where substr(Date, 0, 5) = '2015' and landing_outcome = 'Failure (drone ship)'
```

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

- The ‘substr’ keyword extracts a string of length 2 from the 6th position in the ‘Date’ column of the ‘SPACEXTBL’ table; i.e. the Month. The ‘as Month’ argument returns the said result in Month column
- The combination of the ‘Where’ and ‘substr’ keywords filters the data to show all results that had a failed drone ship landing in the year 2015

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

```
%sql Select Landing_Outcome, count(*) from SPACEXTBL where Date between '2010-06-04' and '2017-03-20'
group by Landing_Outcome order by 2 Desc
```

- The combination of the 'Where' and 'between' keywords filters the data to show results within the specified dates
- The 'count(*)' keywords counts total number of occurrences
- The 'group by' keyword groups the results by 'Landing_Outcome' and the 'order by 2 Desc' argument arranges the data in descending order based on the count(*) column

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

Section 3

Launch Sites Proximities Analysis

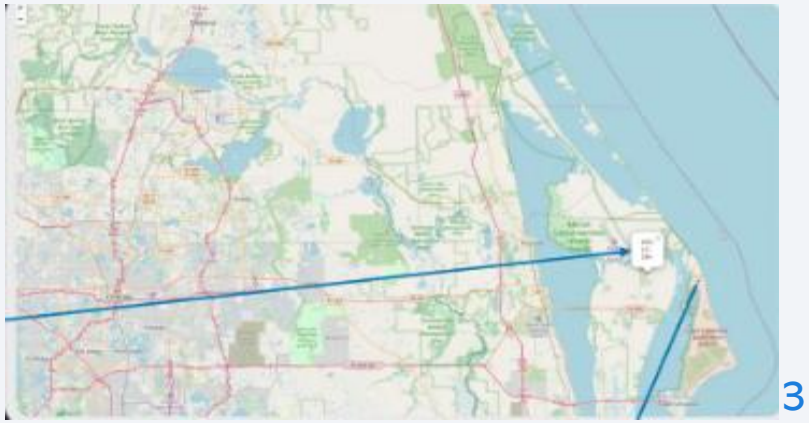
Launch Site Locations

- From the maps it is apparent that all Space X launch sites are near the coast of USA. Three of the sites are in Florida and one is in California. Proximity to the coast minimizes the risk of debris falling of exploding near the population.

- All launch site are also situated near the equator line so that the earths rotation can give a boost to the rockets and reduce fuel consumption

VAFB SLC-4E

KSC LC 39-A



CCAFS SLC-40

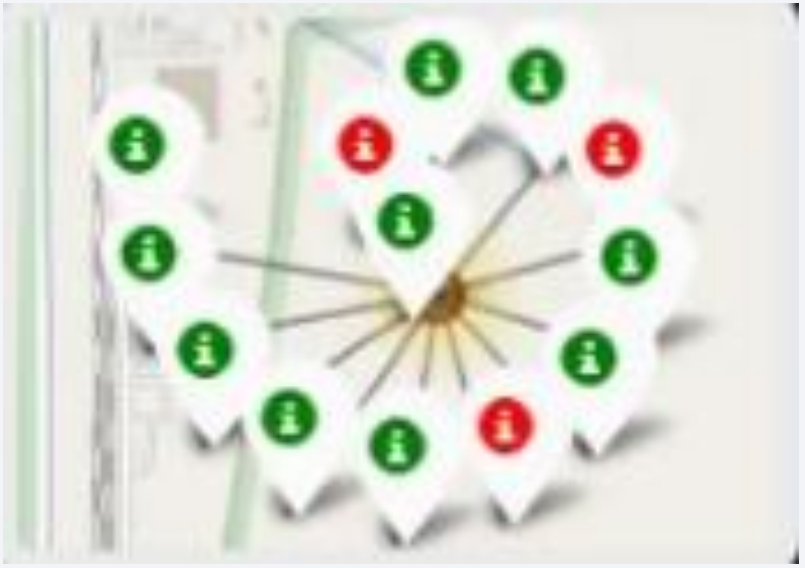
CCAFS LC-40

Successful/Failed Landings

- Launches have been grouped into clusters based on launch site
- They have been annotated with green icons for successful landings and red icons for unsuccessful landings

VAFB SLC-4E

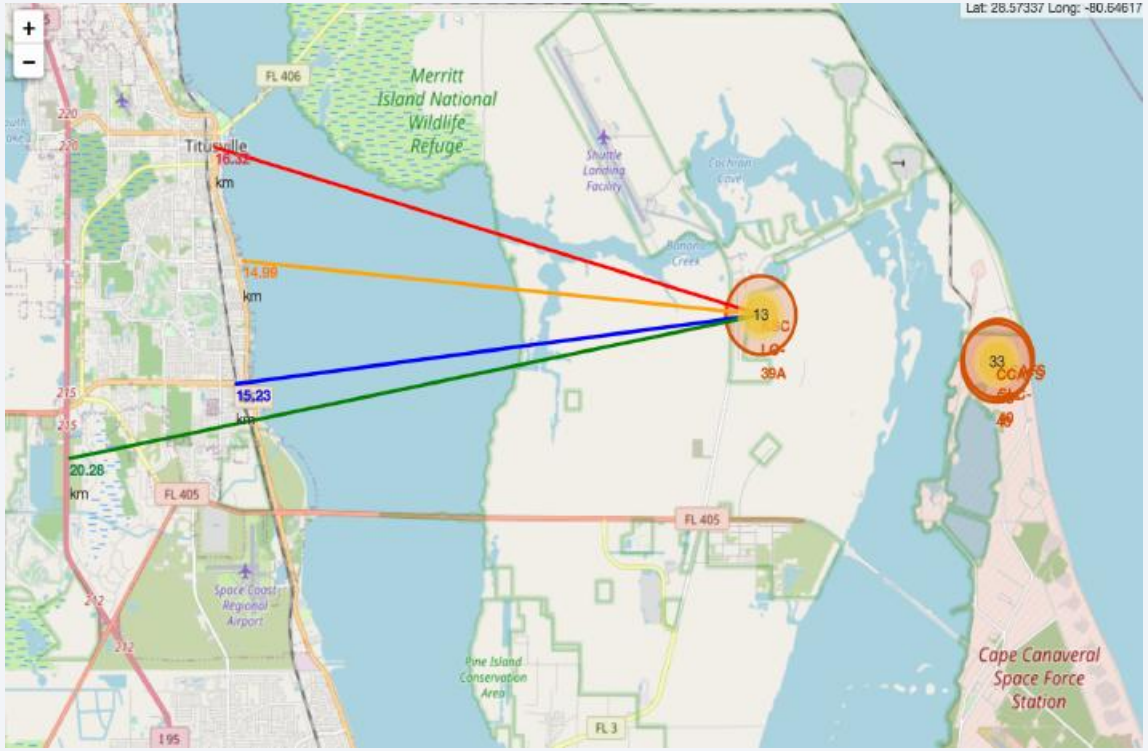
KSC LC-39A

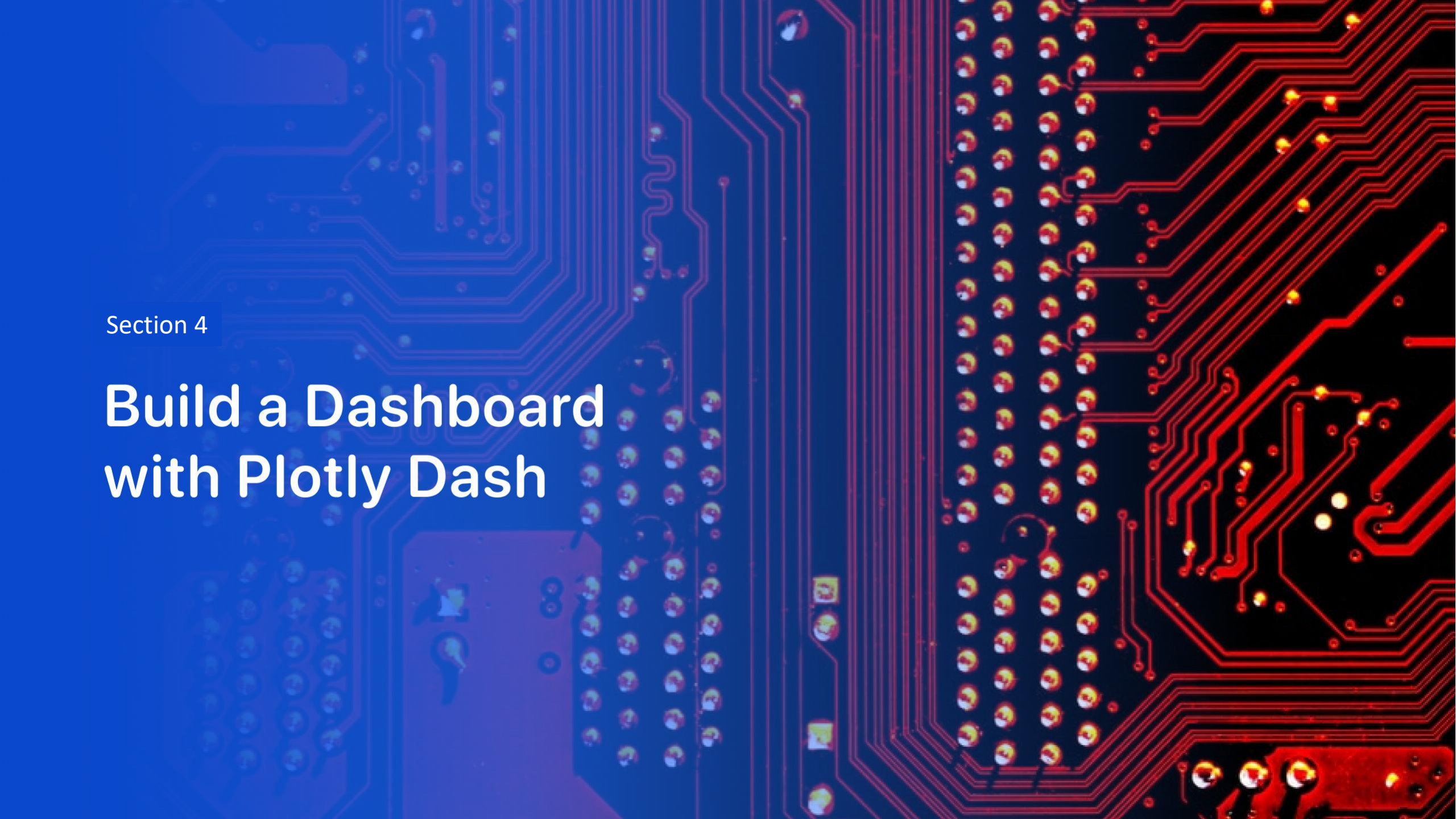


CCAFS SLC -40 & CCAFS LC-40

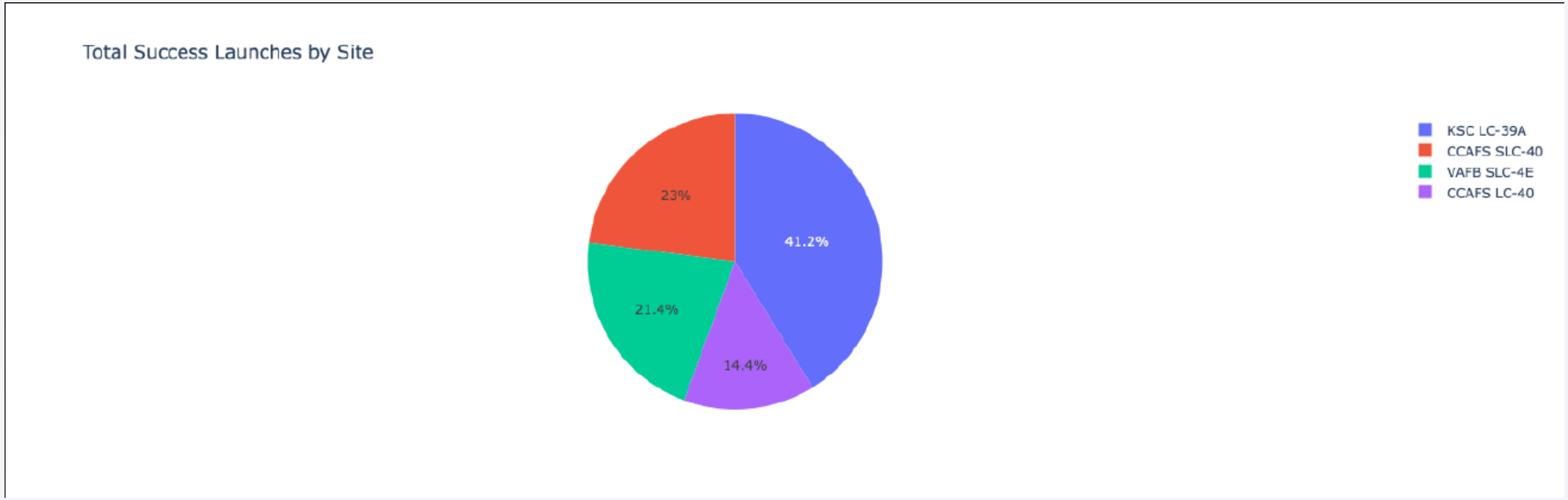
Distance of KSC LC-39A Launch Site to its Proximities

- This launch site is:
  - close to the railway (15.23 km)
  - close to the highway (20.28 km)
  - close to the coastline (14.99 km)
  - close to the city of Titusville (16.32 km)
- The proximity to the above-said points of interest means that rocket debris could be a source of hazard for public infrastructure and the population of Titusville.



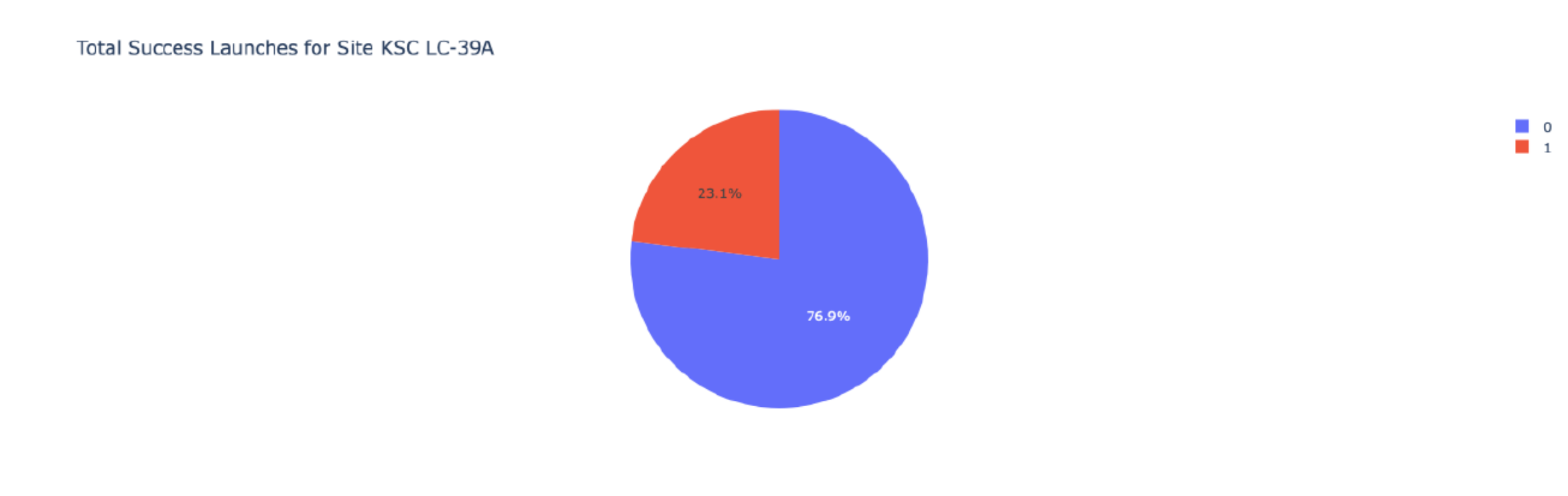

Section 4
Build a Dashboard with Plotly Dash

Launch success rate for all sites



- KSC LC-39A has the highest success rate (41.2%) among all the launch sites
- Show CCAFS LC-40 has the lowest success rate (14.4%)

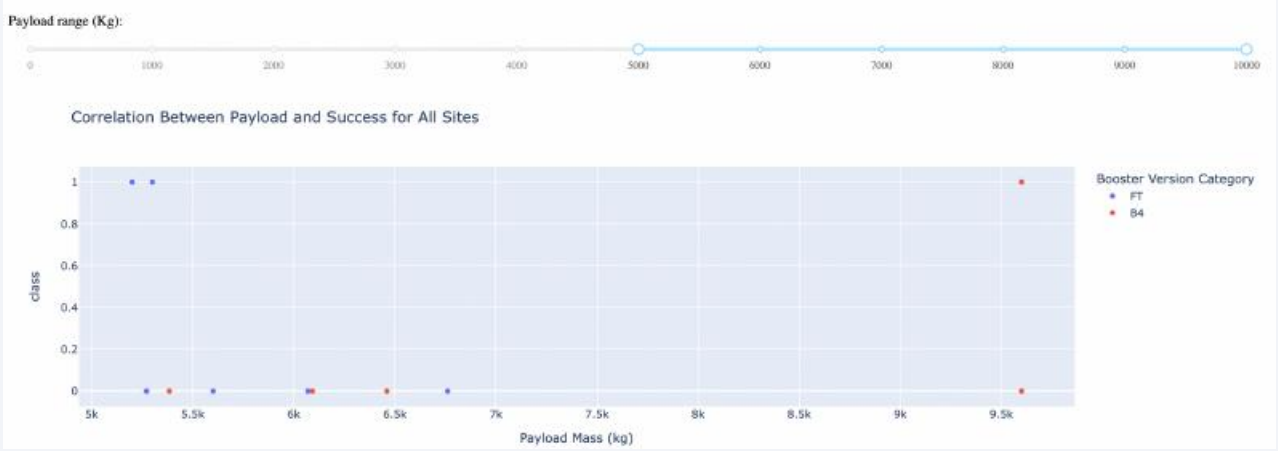
Launch success rate for KCS LC-39A



- KSC LC-39A has the highest launch success rate (76.9%)

Payload Mass vs Launch Outcome for all sites

- The success rate is higher with lighter payloads than with heavier payloads



Section 5

Predictive Analysis (Classification)

Classification Accuracy

- We fit the data using the following models:
    1. Linear Regression (LR)
    2. Support Vector Machine (SVM)
    3. Decision Tree
    4. K Nearest Neighbors (KNN)

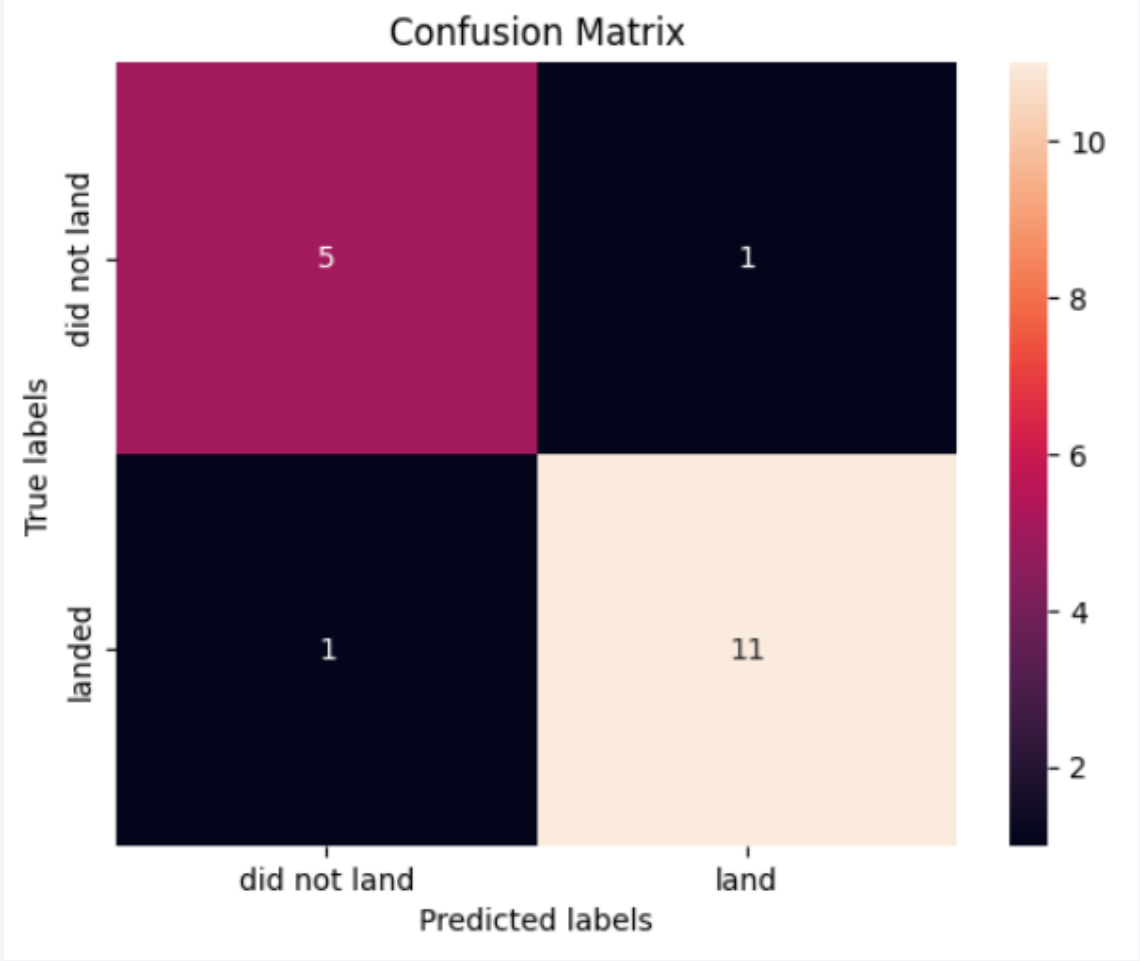
```
print('LR Accuracy:', '{:.2%}'.format(logreg_accuracy))
print( 'SVM Accuracy:', '{:.2%}'.format(svm_accuracy))
print('Decision Tree Accuracy:', '{:.2%}'.format(tree_accuracy))
print('KNN Accuracy:', '{:.2%}'.format(knn_accuracy))
```

```
LR Accuracy: 83.33%
SVM Accuracy: 83.33%
Decision Tree Accuracy: 88.89%
KNN Accuracy: 83.33%
```

- The Decision Tree model gives the highest classification accuracy of 88.89%

Confusion Matrix

- We can see that the Decision Tree model gives us:
    1. True positives: 5
    2. False positives: 1
    3. False negatives: 1
    4. True negatives: 11
- 16 results have been correctly identified (11 rockets landed and 5 did not land)
- We have 1 false positive which means that the model classified a flight that did not land as landed
- We also have 1 false negative, which shows that the model classified a flight that landed as ‘did not land’



Conclusions

As the number of flights has increased, the rate of success at launch sites has also gone up. From 2016 onwards, launch success rate has been above 50%.

Orbit types ES-L1, GEO, HEO and SSO have 100% success rate. While ES-L1, GEO and HEO only have one flight, SSO has had 5 flights in total, all of which have been successful.

The launch site KSC LC-39A has the highest successful launches, accounting for 41.2% of the total successful launches.

Heavier payloads tend to have lower success rate that lighter payloads

The best performing classification model is the Decision Tree Model, with an accuracy of 88.89%

Appendix

- Custom functions to retrieve data from API

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

In [2]:
```
# Takes the dataset and uses the rocket column to call the API and append the data to the list
def getBoosterVersion(data):
    for x in data['rocket']:
       if x:
        response = requests.get("https://api.spacexdata.com/v4/rockets/"+str(x)).json()
        BoosterVersion.append(response['name'])
```

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

In [3]:
```
# Takes the dataset and uses the launchpad column to call the API and append the data to the list
def getLaunchSite(data):
    for x in data['launchpad']:
       if x:
         response = requests.get("https://api.spacexdata.com/v4/launchpads/"+str(x)).json()
         Longitude.append(response['longitude'])
         Latitude.append(response['latitude'])
         LaunchSite.append(response['name'])
```

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

In [4]:
```
# Takes the dataset and uses the payloads column to call the API and append the data to the lists
def getPayloadData(data):
    for load in data['payloads']:
       if load:
        response = requests.get("https://api.spacexdata.com/v4/payloads/"+load).json()
        PayloadMass.append(response['mass_kg'])
        Orbit.append(response['orbit'])
```

Appendix

- Custom functions to fill in lists

In [5]:
```
# Takes the dataset and uses the cores column to call the API and append the data to the lists
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cores/"+core['core']).json()
                Block.append(response['block'])
                ReusedCount.append(response['reuse_count'])
                Serial.append(response['serial'])
            else:
                Block.append(None)
                ReusedCount.append(None)
                Serial.append(None)
            Outcome.append(str(core['landing_success'])+' '+str(core['landing_type']))
            Flights.append(core['flight'])
            GridFins.append(core['gridfins'])
            Reused.append(core['reused'])
            Legs.append(core['legs'])
            LandingPad.append(core['landpad'])
```

In [18]:
```
# Lets take a subset of our dataframe keeping only the features we want and the flight number, and date_utc.
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]

# We will remove rows with multiple cores because those are falcon rockets with 2 extra rocket boosters and rows that have m
data = data[data['cores'].map(len)==1]
data = data[data['payloads'].map(len)==1]

# Since payloads and cores are lists of size 1 we will also extract the single value in the list and replace the feature.
data['cores'] = data['cores'].map(lambda x : x[0])
data['payloads'] = data['payloads'].map(lambda x : x[0])

# We also want to convert the date_utc to a datetime datatype and then extracting the date leaving the time
data['date'] = pd.to_datetime(data['date_utc']).dt.date

# Using the date we will restrict the dates of the launches
data = data[data['date'] <= datetime.date(2020, 11, 13)]
```

Thank you!