IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Themístocles Negreiros
December 30, 2022

Outline

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

Executive Summary

- **Summary of methodologies**

  This project has 3 main parts:

- **Part 1 : Data Collection and Data Wrangling**
  - Data Collection with REST API
  - Data Collection with Web Scraping
  - Data Wrangling
- **Part 2: Exploratory Data Analysis (EDA)**
  - Exploratory Analysis using SQL
  - Exploratory Analysis using Pandas and Matplotlib
  - Interactive Visual Analytics and Dashboard
- **Part 3: Predictive Analysis (Classification)**
  - Create and find the best Machine Learning Model

Executive Summary

- **Summary of all results**

This project has 4 main results:

- Cleaned data for analysis
- Exploratory Data Analysis Insights
- Interactive Visual Analytics and Dashboard
- Predictive Analysis of Classification models

Introduction

Project background and context:

- SpaceX advertises Falcon 9 rocket launches on its website, with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is because SpaceX can reuse the first stage. Therefore, if we can determine if the first stage will land, we can determine the cost of a launch. This information can be used if an alternate company wants to bid against SpaceX for a rocket launch.

Aim of the Project :

- The aim of this project is to predict if the Falcon 9 first stage will land successfully.

Section 1

Methodology

Methodology

- **Executive Summary**
- **Data collection methodology:**
  - SpaceX REST API
  - Web Scraping using BeautifulSoup
- **Data Wrangling**
  - Using **.mean( )** method to get mean value of a column
  - Using **.fillna( )** method to replace missing values with the mean value
  - Using **.value_counts( )** method to:
    - Calculate the number of launches on each site
    - Calculate the number and occurrence of each orbit
    - Calculate the number and occurrence of mission outcome per orbit type
  - Create a landing outcome label from Outcome column
    - Labeling **0** when the booster land fail
    - Labeling **1** when the booster land successfully

Methodology

- **Executive Summary**
- **Exploratory Data Analysis (EDA)**
  - Using **% sql** magic to do SQL queries on Spacex launch data
  - Using **Pandas API** to manipulate the data
  - Using **Matplolib** to plot and visualize relationship between variables and get insights from data
- **Interactive Visual Analytics**
  - Geospatial analytics using Folium
  - Using **Plolty Dash** to create interactive dashboard
- **Predictive Analysis using classification models**
  - Using Scikit-learn API to:
    - Pre-process the data
    - Split the data into train and test datasets
    - Create the models
    - Evaluate the models

Data Collection – REST API

Data collection steps:

1. Collecting data using GET request to the SpaceX REST API

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

response = requests.get(spacex_url)
```

2. Convert response to a dataframe using Pandas .json_normalize( ) method

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

3. Subset the Dataframe with only columns of interest

```
# Lets take a subset of our dataframe keeping only the features we want and the flight number, and
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 boos
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 an
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
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)]
```

4. Get data using functions and set it to empty lists and convert it to a dictionary

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

```
# Call getBoosterVersion
getBoosterVersion(data)

# Call getLaunchSite
getLaunchSite(data)

# Call getPayloadData
getPayloadData(data)

# Call getCoreData
getCoreData(data)
```

```
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. Convert dictionary to a dataframe and filter only Falcon 9 launches

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

```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = df_launch[df_launch['BoosterVersion']!='Falcon 1']
data_falcon9
```

6. Reset the FlightNumber column and replace NaN values with mean

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

```
# Calculate the mean value of PayloadMass column
mean = data_falcon9['PayloadMass'].mean()
mean
# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'].fillna(value= mean, inplace=True)
```

GitHub link

Data Collection – Web Scraping

Web Scraping steps:

1. **Request the Falcon9 Launch Wiki page from its URL**

```
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
response = requests.get(static_url)
# assign the response to a object
response.content
```

2. **Create a BeautifulSoup object from HTML response**

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

3. **Get all tables on BeautifulSoup object using .find_all ( )**

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

4. **Extract all column/variable names from the HTML table header**

```
# Let's print the third table and check its content
first_launch_table = html_tables[2]
print(first launch 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 i in first_launch_table.find_all('th'):
    name =extract_column_from_header(i)
    if name != None and len(name) > 0:
        column_names.append(name)
```

GitHub link

Data Collection – Web Scraping

Web Scraping steps:

5. Create an empty dictionary with keys from the extracted column names and use a function to parse launch record

```
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']=[]
```

6. Create a DataFrame from the dictionary

```
df=pd.DataFrame(launch_dict)
```

GitHub link

Data Wrangling

Data Wrangling steps:

1. **Using .value_counts( ) method to:**

Calculate the number of launches on each site

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

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

Calculate the number and occurrence of each orbit

```
# Apply value_counts on Orbit column
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
Name: Orbit, dtype: int64
```

Calculate the number and occurrence of mission outcome per orbit type

```
# landing_outcomes = values on Outcome column
landing_outcomes = df['Outcome'].value_counts()
landing_outcomes
```

```
True ASDS      41
None None      19
True RTLS      14
False ASDS      6
True Ocean      5
False Ocean     2
None ASDS       2
False RTLS      1
Name: Outcome, dtype: int64
```

GitHub link

2. **Get the index of rows with False values. False = fail landing**

```
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. **Set bad outcomes to bad_outcomes variable using index filtering**

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

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

4. **Create a landing outcome label from Outcome column**

```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
#create a list
landing_class = []

for outcome in df['Outcome']:
    if outcome in bad_outcomes:
        landing_class.append(0)
    else:
        landing_class.append(1)
print(landing_class)
```

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

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

EDA with Data visualization

Data visualization steps:

1. **Import python libraries**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

2. **Using seaborn and matplolib.pyplot to create plots to::**

Visualize the relationship between Flight Number and Launch Site (using sns.catplot ( ))

```
### TASK 1: Visualize the relationship between Flight Number and Launch Site
sns.catplot(data=df, x='FlightNumber', y='LaunchSite', hue='Class')
plt.xlabel('Flight Number')
plt.ylabel('Launch Site')
plt.show()
```

Visualize the relationship between Payload and Launch Site (using sns.catplot ( ))

```
### TASK 2: Visualize the relationship between Payload and Launch Site
sns.catplot(data=df, x='PayloadMass', y='LaunchSite', hue='Class')
plt.xlabel('Pay load Mass (kg)')
plt.ylabel('Launch Site')
plt.show()
```

Create a new dataframe with mean values using .groupby( ) method and plot a Bar plot (sns.barplot( ))

Visualize the relationship between success rate of each orbit type

```
### TASK 3: Visualize the relationship between success rate of each orbit ty
df_orbit = df.groupby('Orbit').mean().reset_index()

sns.barplot(data=df_orbit, x='Orbit', y='Class')
plt.show()
```

Visualize the relationship between FlightNumber and Orbit type (using sns.scatterplot ( ))

```
### TASK 4: Visualize the relationship between FlightNumber and Orbit type
sns.scatterplot(data=df, x='FlightNumber', y='Orbit', hue='Class')
plt.show()
```

Visualize the relationship between Payload and Orbit type (using sns.scatterplot ( ))

```
### TASK 5: Visualize the relationship between Payload and Orbit type
sns.scatterplot(data=df, x='PayloadMass', y='Orbit', hue='Class')
plt.show()
```

GitHub link

EDA with Data visualization

Data visualization steps:

3. Extract years from the date using a function

```
# A function to Extract years from the date
year=[]
def Extract_year():
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year
Extract_year()
df['Date'] = year
df.head()
```

4. Plot a line chart with the extracted year vs success rate

```
# Plot a line chart with x axis to be the extracted year and y axis to be the success rate
plt.figure(figsize=(15,8))
sns.lineplot(data=df, x='Date', y='Class')
plt.show()
```

GitHub link

EDA with SQL

Using SQL queries to:

- Display the names of the unique launch sites in the space mission.
- Display 5 records where launch sites begin with the string 'CCA’.
- Display the total payload mass carried by boosters launched by NASA (CRS).
- Display average payload mass carried by booster version F9 v1.1.
- List the date when the first successful landing outcome in ground pad was achieved.
- List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000.
- List the total number of successful and failure mission outcomes.
- List the names of the booster versions which 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 successful landing outcomes between the date 04-06-2010 and 20-03-2017 in descending order.

GitHub link

Build an Interactive Map with Folium

Creating interactive map steps:

1. Select relevant sub-columns

```
# Select relevant sub-columns: `Launch Site`, `Lat(Latitude)`, `Long(Longitude)`, `class`
spacex_df = spacex_df[['Launch Site', 'Lat', 'Long', 'class']]
launch_sites_df = spacex_df.groupby(['Launch Site'], as_index=False).first()
launch_sites_df = launch_sites_df[['Launch Site', 'Lat', 'Long']]
launch_sites_df
```

2. Mark all launch sites on a map using folium.Circle( )

```
# Initial the map
site_map = folium.Map(location=nasa_coordinate, zoom_start=5)
# For each launch site, add a Circle object based on its coordinate (Lat, Long) values. In addition, add Launc
# iterate over launch sites and extract latitude, longitude and launch sites
for i, row in launch_sites_df.iterrows():
    lat  = row[1]
    long = row[2]
    site = row[0]

    # create circle object
    circle = folium.Circle([lat, long], radius=1000, color='#d35400', fill=True).add_child(folium.Popup(site))

    # add circle to site_map
    site_map.add_child(circle)


site_map
```

3. Mark the success/failed launches for each site on the map using folium.Marker( )

Assign a different color to failure and success outcomes

```
# Function to assign color to launch outcome
def assign_marker_color(launch_outcome):
    if launch_outcome == 1:
        return 'green'
    else:
        return 'red'

spacex_df['marker_color'] = spacex_df['class'].apply(assign_marker_color)
spacex_df.tail(10)
```

Create a marker on map using folium.Marker( )

```
# Add marker_cluster to current site_map
site_map.add_child(marker_cluster)

# for each row in spacex_df data frame
# create a Marker object with its coordinate
# and customize the Marker's icon property to indicate if this launch was successed or failed,
# e.g., icon=folium.Icon(color='white', icon_color=row['marker_color']
for index, record in spacex_df.iterrows():
    lat  = record[1]
    long = record[2]
    site = record[0]

    #Create marker object with its coordinate
    marker = folium.Marker([lat, long],
        # Create an icon to indicate if this launch was successed or failed
        icon=folium.Icon(color='white', icon_color=record['marker_color'])
    )
    # add marker to site_map
    marker_cluster.add_child(marker)

site_map
```

GitHub link

Build an Interactive Map with Folium

Creating interactive map steps:

4. **Calculate the distances between a launch site to its proximities**

```
# find coordinate of the closet coastline
launch_site_lat = 28.56319
launch_site_long = -80.57683
launch_site_coordinates = [launch_site_lat, launch_site_long]

coastline_lat = launch_site_lat #same latitude, so the line is directly east
coastline_long = -80.56794
coastline_coordinates = [coastline_lat, coastline_long]

distance_coastline = calculate_distance(launch_site_lat, launch_site_long, coastline_lat, coastline_long)
```

```
#define the distance marker (which will be shown at the coastline)
distance_marker = folium.Marker(
    coastline_coordinates,
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='%s ' % "{:10.2f} KM".format(distance_coastline),
        )
    )
#add the distance marker to the map and show the map
site_map.add_child(distance_marker)
site_map
```

```
# Create a `folium.PolyLine` object using the coastline coordinates and launch site coordinate
#define the distance line (between the coastline and the launch site)
distance_line=folium.PolyLine(
    locations=[launch_site_coordinates, coastline_coordinates],
    weight=1
)

#add the distance line to the map and show the map
site_map.add_child(distance_line)
site_map
```

GitHub link

Build a Dashboard with Plotly Dash

Tasks to Create an interactive dashboard :

TASK 1: Add a dropdown list to enable Launch Site selection

```
# TASK 1: Add a dropdown list to enable Launch Site selection
# The default select value is for ALL sites
dcc.Dropdown(id='site-dropdown',
             options = launch_sites,
             value='All Sites',
             placeholder="Launch Site",
             searchable=True
             ),
```

TASK 2: Add a pie chart to show the total successful launches count for all sites

```
html.Div(dcc.Graph(id='success-pie-chart')),
html.Br(),
```

```
# TASK 2:
# Add a callback function for `site-dropdown` as input, `success-pie-chart` as output
@app.callback(Output(component_id='success-pie-chart', component_property='figure'),
              Input(component_id='site-dropdown', component_property='value'))

def get_pie_chart(entered_site):
    filtered_df = spacex_df[spacex_df['Launch Site'] == entered_site]
    if entered_site == 'All Sites':
        fig = px.pie(spacex_df, values='class', names='Launch Site', title='Total Success Launches by Site')
        return fig
    else:
        # return the outcomes pie chart for a selected site
        site_df = filtered_df.groupby(['Launch Site', 'class']).size().reset_index(name='class count')
        fig = px.pie(site_df,values='class count',names='class',title=f'Total Success Launches for site {entered_site}')
        return fig
```

TASK 3: Add a slider to select payload range

```
# TASK 3: Add a slider to select payload range
dcc.RangeSlider(id='payload-slider',
                min=0, max=10000, step=1000,
                marks={0: '0', 2500: '2500', 5000: '5000', 7500: '7500', 10000: '10000'},
                value=[min_payload, max_payload]),
```

TASK 4: Add a scatter chart to show the correlation between payload and launch success

```
# TASK 4: Add a scatter chart to show the correlation bet
html.Div(dcc.Graph(id='success-payload-scatter-chart')),
])
```

```
# Add a callback function for `site-dropdown` and `payload-slider` as inputs, `success-payload-sc
@app.callback(Output(component_id='success-payload-scatter-chart', component_property='figure'),
              [Input(component_id='site-dropdown', component_property='value'),
               Input(component_id='payload-slider', component_property='value')])

def get_scatter_chart(entered_site, payload_slider):
    low, high = payload_slider
    slide=(spacex_df['Payload Mass (kg)'] > low) & (spacex_df['Payload Mass (kg)'] < high)
    dropdown_scatter=spacex_df[slide]

    #If All sites are selected, render a scatter plot to display all values for variables Payload
    #Point colour is set to the booster version category
    if entered_site == 'All Sites':
        fig = px.scatter(
            dropdown_scatter, x='Payload Mass (kg)', y='class',
            hover_data=['Booster Version'],
            color='Booster Version Category',
            title='Correlation between Payload and Success for all Sites')
        return fig
    else:
    #If a specific site is selected, filter the spacex_df dataframe first, then render a scatter
        dropdown_scatter = dropdown_scatter[spacex_df['Launch Site'] == entered_site]
        fig=px.scatter(
            dropdown_scatter,x='Payload Mass (kg)', y='class',
            hover_data=['Booster Version'],
            color='Booster Version Category',
            title = f'Success by Payload Size for site {entered_site}')
        return fig
```

GitHub link

Predictive Analysis (Classification)

1. Preparing data to train model:

Standardize the data using preprocessing.StandardScaler()

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

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

Split train and test datasets using train_test_split( )

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
```

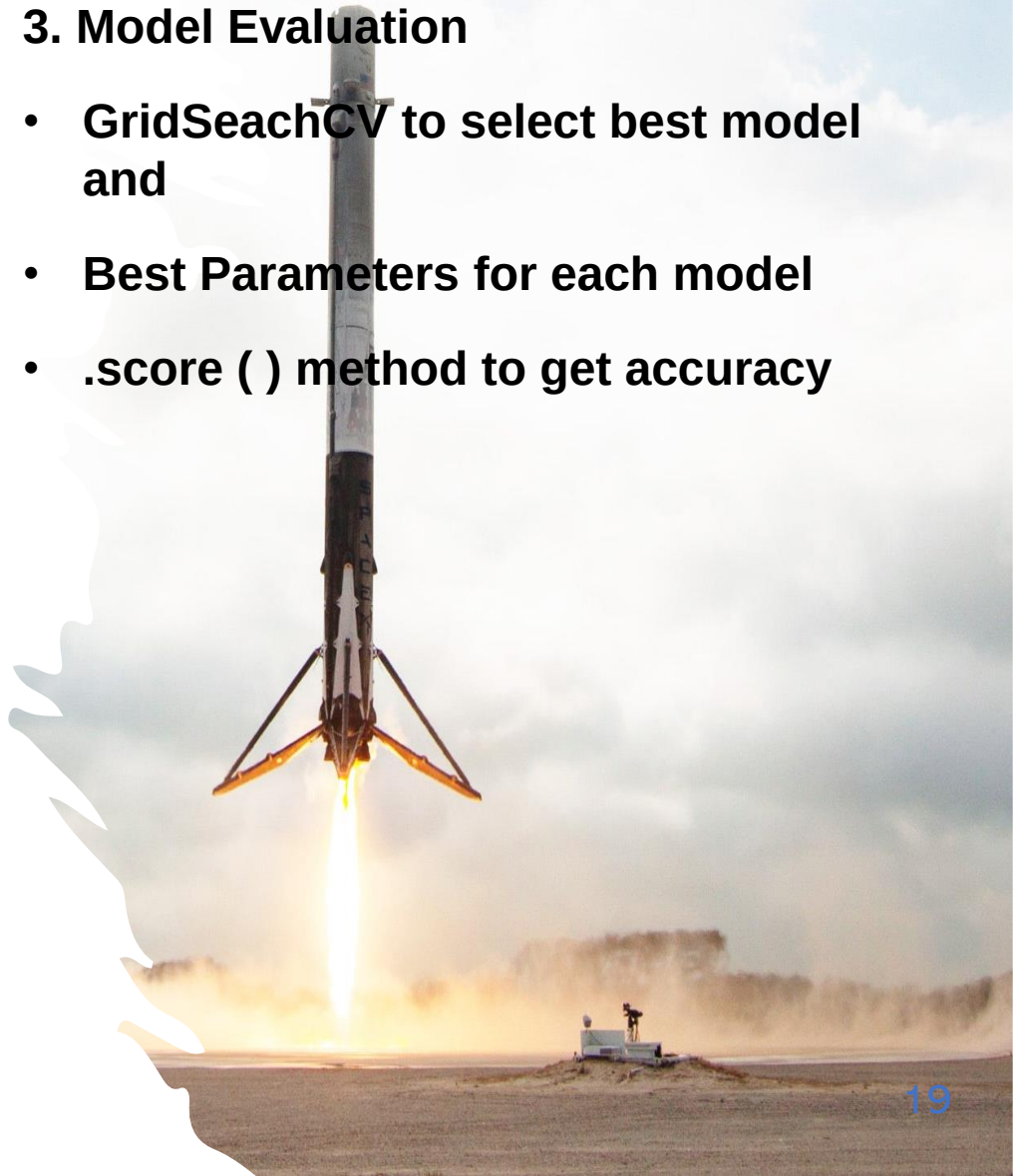
2. Training the models

- **Algorithms tested:**
  - **Logistic Regression**
  - **Suport Verctor Machine (SVM**
  - **Decision Tree**
  - **K nearest Neighbors (KNN)**

GitHub link

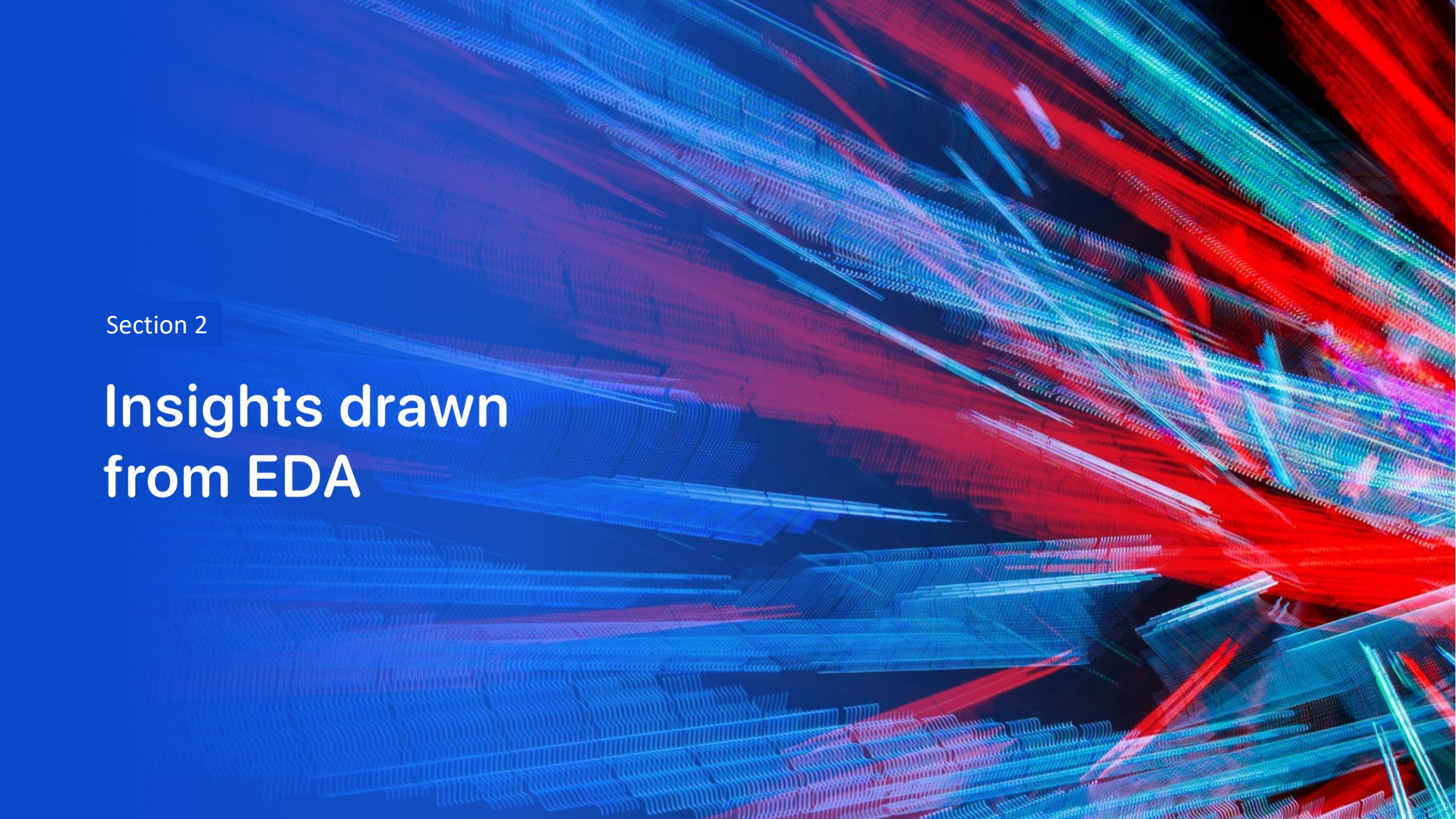
3. Model Evaluation

- **GridSeachCV to select best model and**
- **Best Parameters for each model**
- **.score ( ) method to get accuracy**



Results

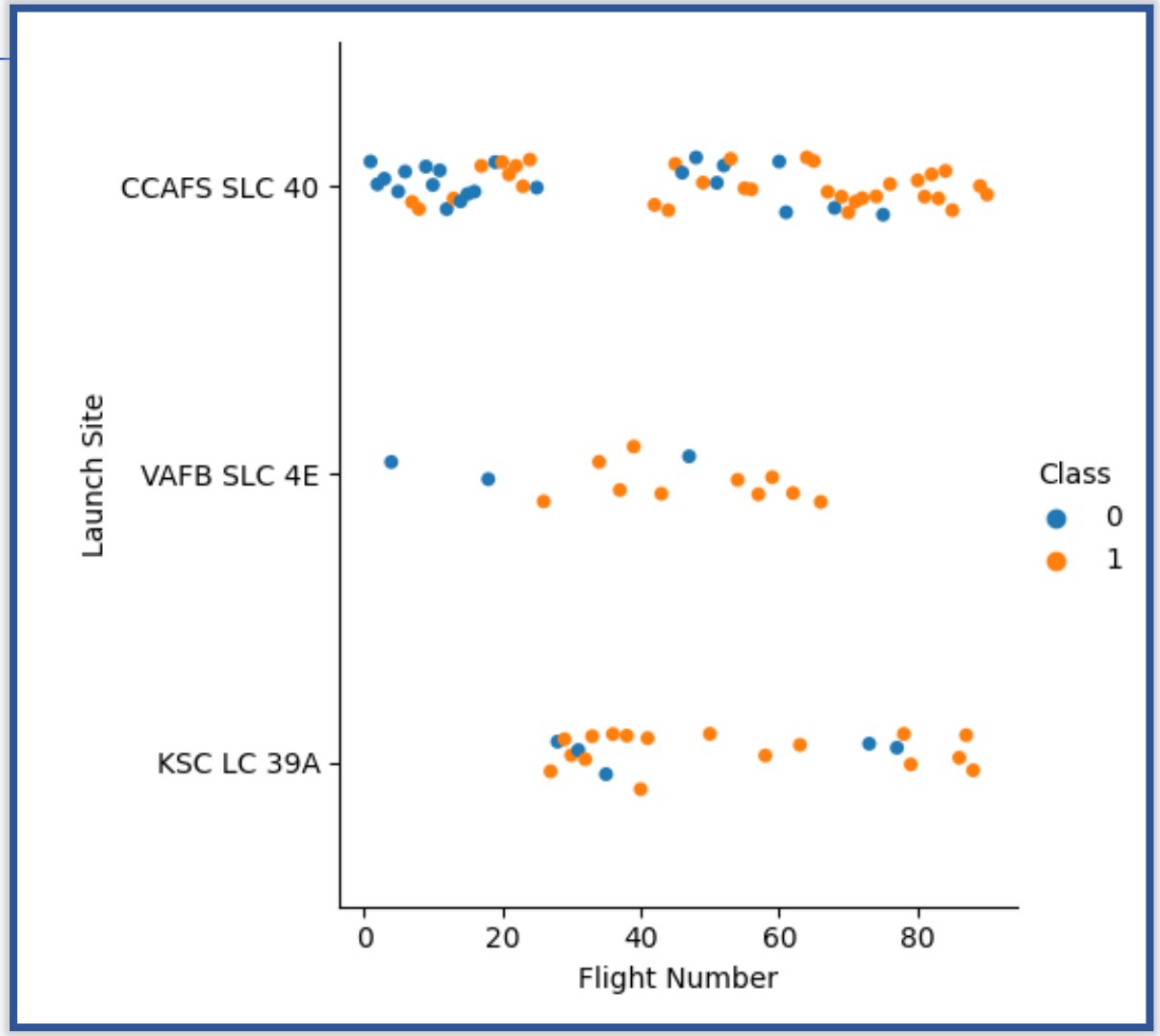
- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results


Section 2
Insights drawn from EDA

FLIGHT NUMBER VS. LAUNCH SITE

Insights from scatter plot analysis:

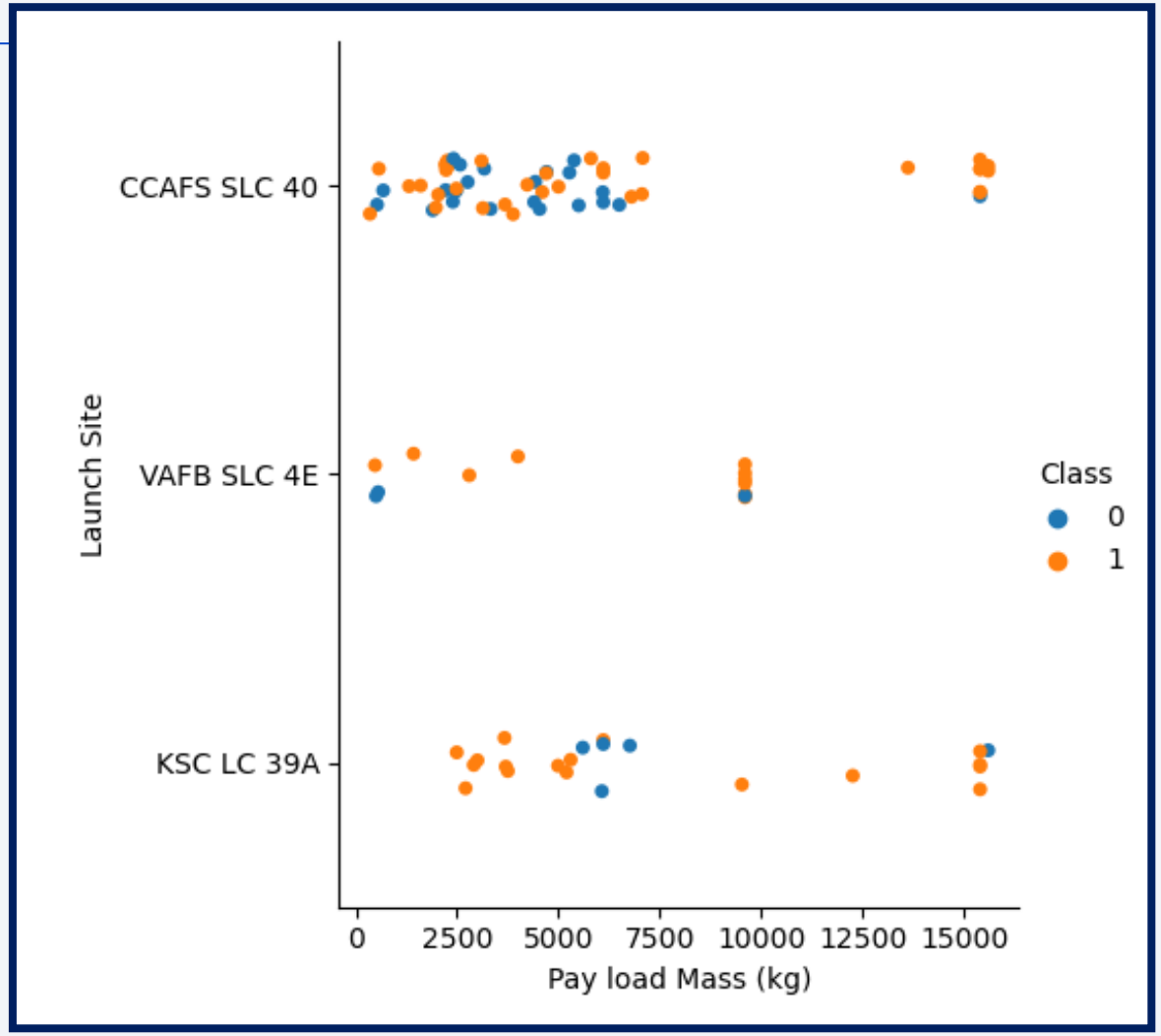
- **Success rate increases with number of flight > 30**
- **CCAFS SLC 40 has more flights and was used since the beginning of flights**
- **KSC LC 39A started to be used later after the 30Th flight**
- **VAFB SLC 4E has few flights compared to the others**



PAYLOAD VS. LAUNCH SITE

Insights from Scatter plot analysis:

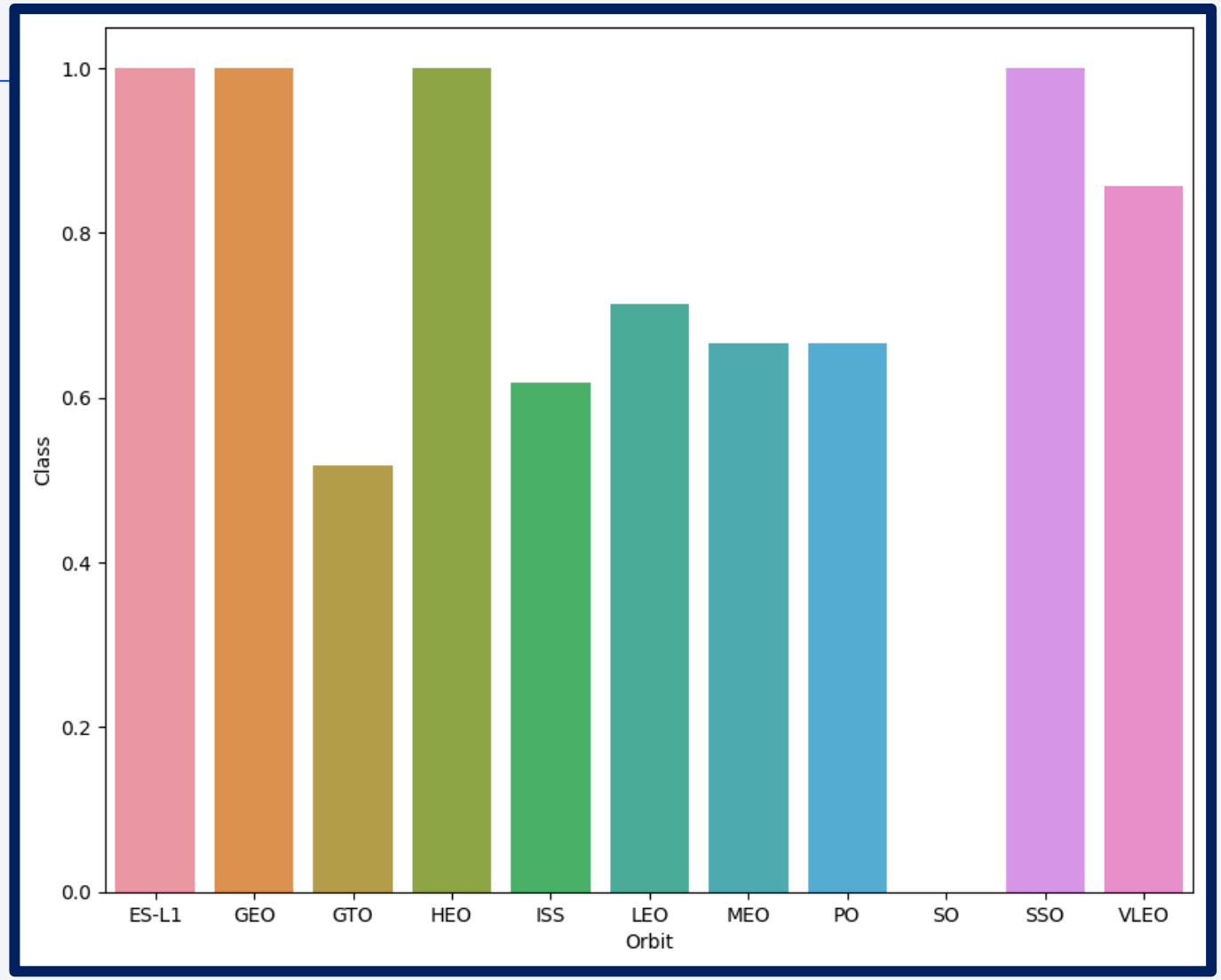
- Most of flights had a Pay load Mass < 7500 Kg
- KSC LC 39A has successful rate with Pay Load Mass between 2500 and 7500 Kg
- VAFB SLC 4E has flights with Pay load Mass < 10000 Kg
- Pay load Mass > 10000 seems to have higher success rate



SUCCESS RATE VS. ORBIT TYPE

This bar plot shows:

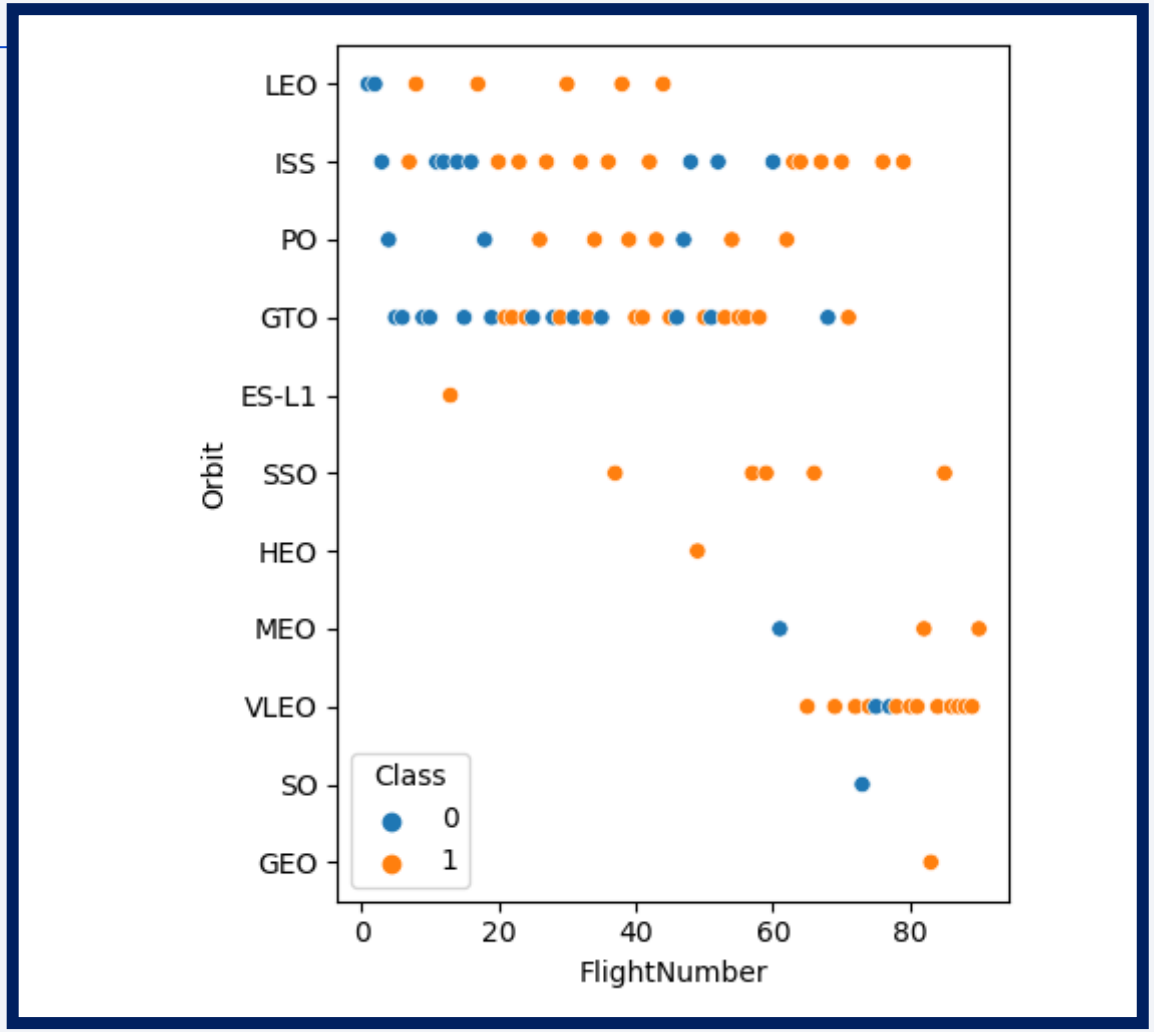
- Orbits with highers success rate (100 %):
  - ES-L1 (Earth-Sun First Lagrangian Point)
  - GEO (Geostationary Orbit)
  - HEO (High Earth Orbit)
  - SSO (Sun-synchronous Orbit)
- Orbit with lowest success rate (0 %):
  - SO (Heliocentric Orbit)



FLIGHT NUMBER VS. ORBIT TYPE

Insights from scatter plot analysis:

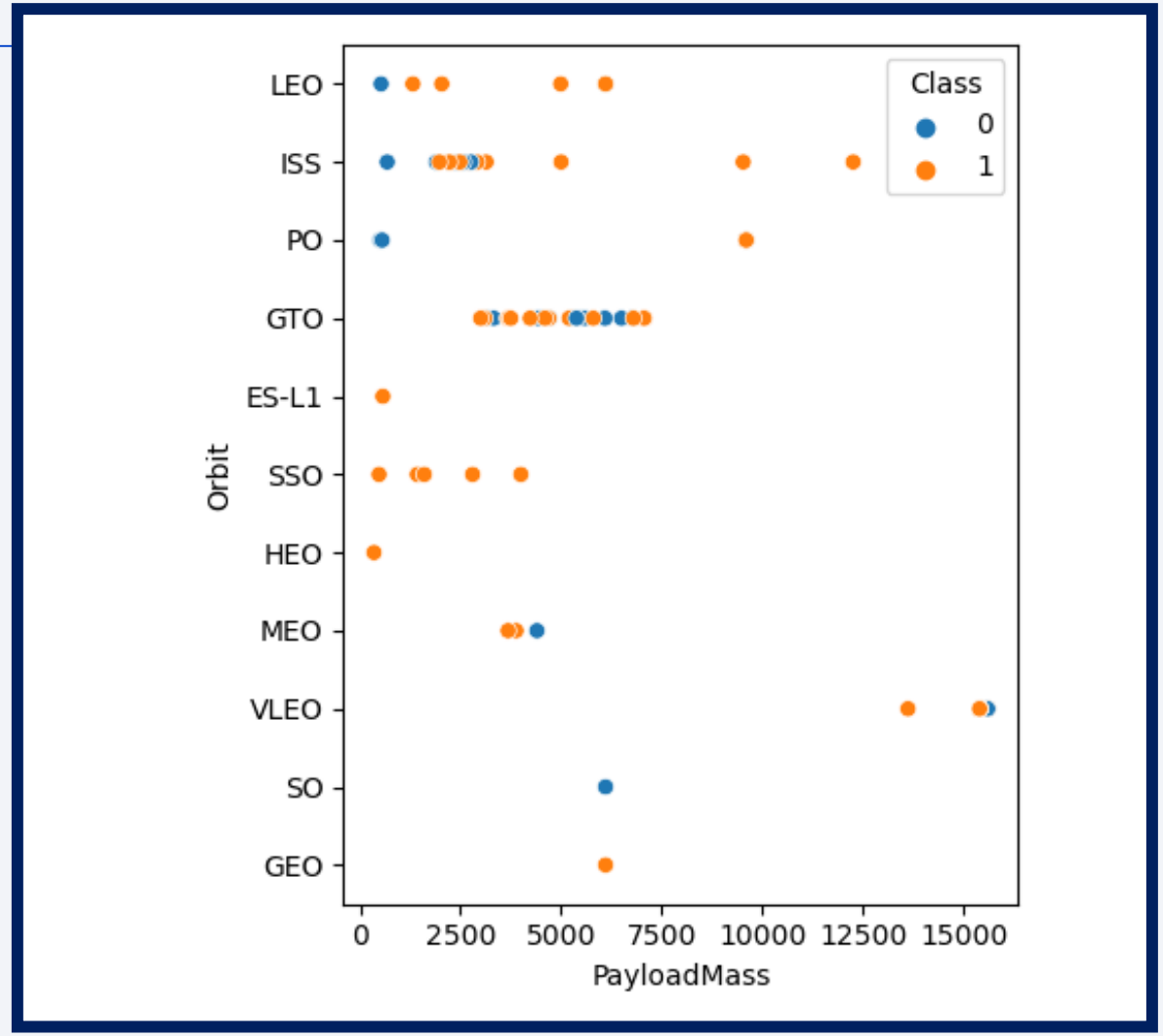
- **VLEO has flight after flight number > 60. Most of them successful**
- **ES-L1, HEO and has GEO just one flight. A successful one.**
- **SSQ has just 5 flights. All of them successful**
- **SO has just one flight. A failure one**



PAYLOAD VS. ORBIT TYPE

Insights from scatter plot analysis:

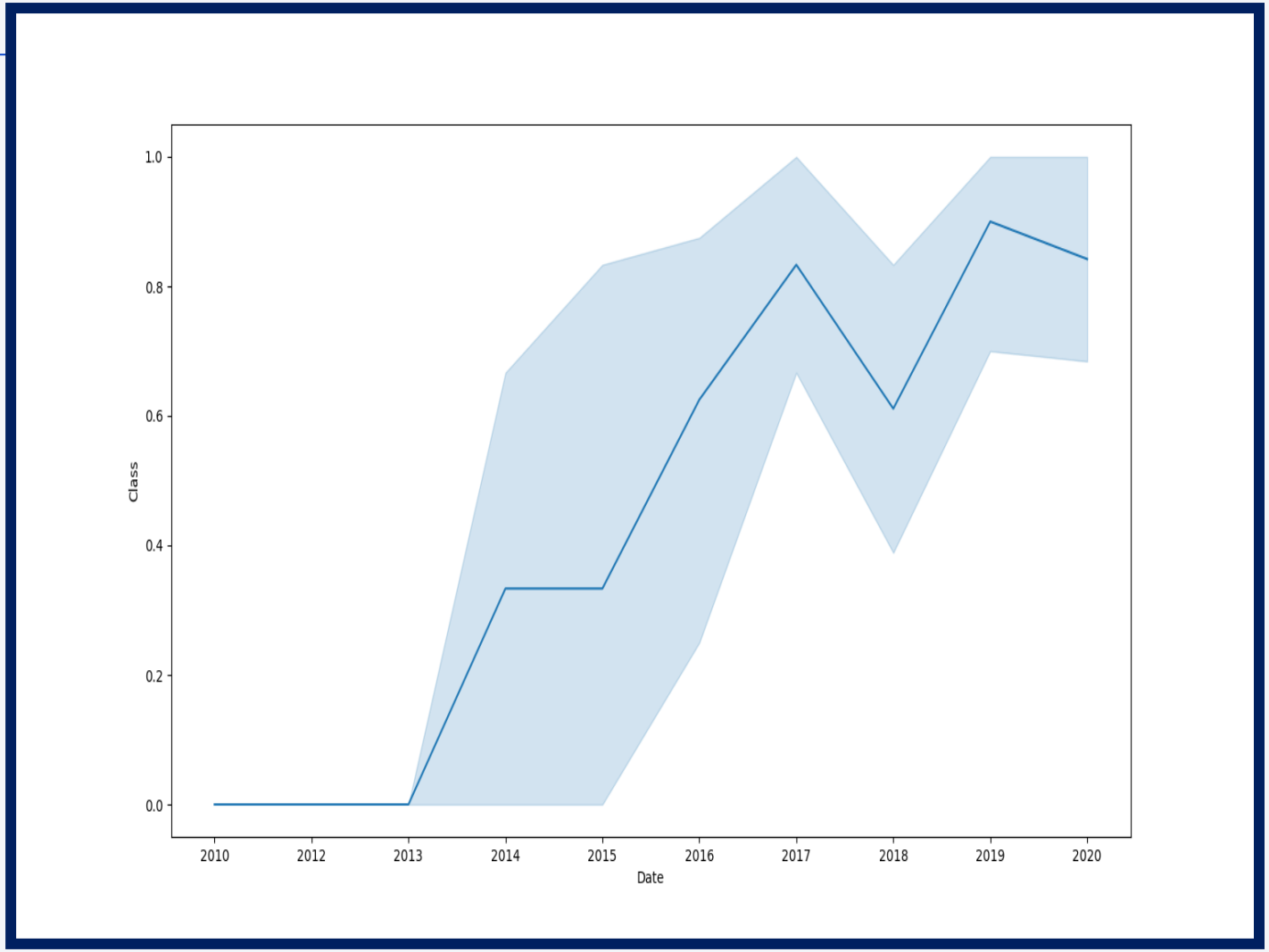
- **VLEO has flights with Pay Load Mass > 12500 Kg. Most of them successful**
- **ISS, PO and VLEO has flight with Pay Load Mass > 7500 Kg**
- **ES-L1, SSO and HEO has all their flights successful with Pay Load Mass < 5000 Kg**
- **ISS has the large range of Pay Load Mass between 0 and 12500 Kg. Most of them successful**



LAUNCH SUCCESS YEARLY TREND

The line chart shows:

- Between 2010 and 2013, all landing fail
- After 2013, successful landing increases until almost 40 %
- Between 2015 and 2017, it increases a lot the rate of success being over 80 %
- Between 2017 and 2018, landing success decrease to 60 %
- After 2019, landing success increase to Its higher rate (about 90 %)



ALL LAUNCH SITE NAMES

Find the names of the unique launch sites

In [9]:
```
%sql SELECT DISTINCT(launch_site) FROM spacextbl;
```

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

Out[9]:

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

The Command DISTINCT returns unique values of column Launch_Site from Spacextbl Table.

LAUNCH SITE NAMES BEGIN WITH 'CCA'

Find 5 records where launch sites begin with `CCA`

```
Entrée [8]: %sql SELECT launch_site FROM spacextbl WHERE launch_site LIKE 'CCA%' LIMIT 5;
 * sqlite:///my_data1.db
Done.
```

Out[8]:

Launch_Site
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40

The command WHERE combined with LIKE `CCA` retrieve values of launch_site column that begins with CCA. The command LIMIT 5 display just first five records.

TOTAL PAYLOAD MASS

Calculate the total payload carried by boosters from NASA

```
In [22]: %sql SELECT SUM(PAYLOAD_MASS__KG_) AS TOTAL_PAYLOAD_MASS FROM spacextbl WHERE customer = 'NASA (CRS)';
 * sqlite:///my_data1.db
Done.
Out[22]: TOTAL_PAYLOAD_MASS
         45596
```

The command SUM on columns PAYLOAD_MASS__KG_ combined with condition WHERE customer = 'NASA (CRS)' retrieve the sum of pay load mass where costumer is NASA (CRS).

AVERAGE PAYLOAD MASS BY F9 V1.1

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

```
Entrée [69]: %sql SELECT AVG(PAYLOAD_MASS__KG_) AS AVG_PAYLOAD_MASS FROM spacextbl WHERE Booster_Version = 'F9 v1.1';
              * sqlite:///my_data1.db
             Done.
Out[69]:  AVG_PAYLOAD_MASS
                    2928.4
```

The command AVG on PAYLOAD_MASS__ KG column combined with condition WHERE Booster_version = 'F9 v1.1' retrieve average payload mass carried by booster version F9 v1.1.

FIRST SUCCESSFUL GROUND LANDING DATE

Find the dates of the first successful landing outcome on ground pad

```
Entrée [62]: %sql SELECT MIN(Date) AS FIRST_LANDING FROM spacextbl WHERE [landing _outcome] = 'Success (ground pad)';
              * sqlite:///my_data1.db
             Done.
Out[62]:  FIRST_LANDING
               01-05-2017
```

The command MIN on column Date combined with condition WHERE landing _outcome = 'Success (ground pad)' retrieve dates of the first successful landing outcome on ground pad.

SUCCESSFUL DRONE SHIP LANDING WITH PAYLOAD BETWEEN 4000 AND 6000

List the names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000

```
Entrée [63]: %sql SELECT Booster_Version FROM spacextbl \
             WHERE ([landing _outcome] = 'Success (drone ship)') AND (PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000);
              * sqlite:///my_data1.db
             Done.
```

Out[63]:

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

The command SELECT on Booster_Version column combined with conditions using WHERE, AND, BETWEEN retrieve boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000.

TOTAL NUMBER OF SUCCESSFUL AND FAILURE MISSION OUTCOMES

Calculate the total number of successful and failure mission outcomes

In [41]:
```
%sql SELECT mission_outcome, COUNT(mission_outcome) AS TOTAL_NUMBER FROM spacextbl GROUP BY mission_outcome;
```

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

Out[41]:

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

The command COUNT on mission_outcome column combined with GROUP BY retrieve the count of mission outcome grouped by mission outcome.

BOOSTERS CARRIED MAXIMUM PAYLOAD

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

```
In [46]: %sql SELECT DISTINCT(booster_version) FROM spacextbl WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM spacextbl);
 * sqlite:///my_data1.db
Done.
```

Out[46]:

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

The command DISTINCT retrieve unique values of booster_version column combined with a condition subquery with the MAX command retrieve unique values which have the maximum Pay load Mass.

2015 LAUNCH RECORDS

List the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015

```
Entrée [68]: %sql SELECT booster_version, launch_site FROM spacextbl \
             WHERE ([landing _outcome] = 'Failure (drone ship)') AND (substr(Date,7,4) = '2015');

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

Out[68]:

Booster_Version	Launch_Site
F9 v1.1 B1012	CCAFS LC-40
F9 v1.1 B1015	CCAFS LC-40

The SELECT command to select 2 columns combined with a condition using WHERE, AND to retrieve data which Lading failure in drone ship in the year of 2015.

RANK LANDING OUTCOMES BETWEEN 2010-06-04 AND 2017-03-20

Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order

```
Entrée [67]: %sql SELECT [landing _outcome], COUNT([landing _outcome]) AS TOTAL_NUMBER FROM spacextbl \
             WHERE DATE BETWEEN '04-06-2010' AND '20-03-2017' GROUP BY  [landing _outcome] ORDER BY TOTAL_NUMBER DESC;
              * sqlite:///my_data1.db
             Done.
```

Out[67]:

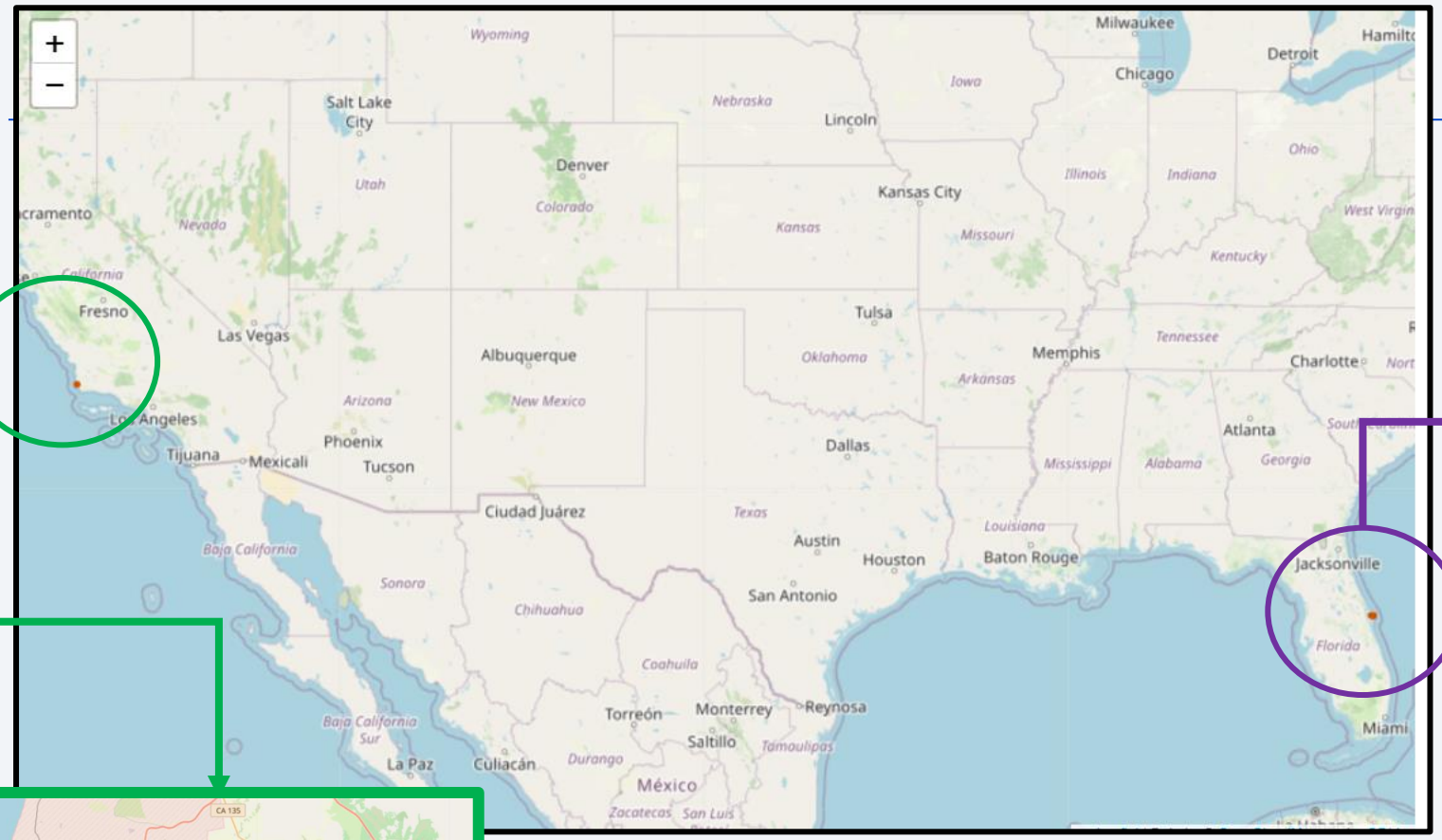
Landing _Outcome	TOTAL_NUMBER
Success	20
No attempt	10
Success (drone ship)	8
Success (ground pad)	6
Failure (drone ship)	4
Failure	3
Controlled (ocean)	3
Failure (parachute)	2
No attempt	1

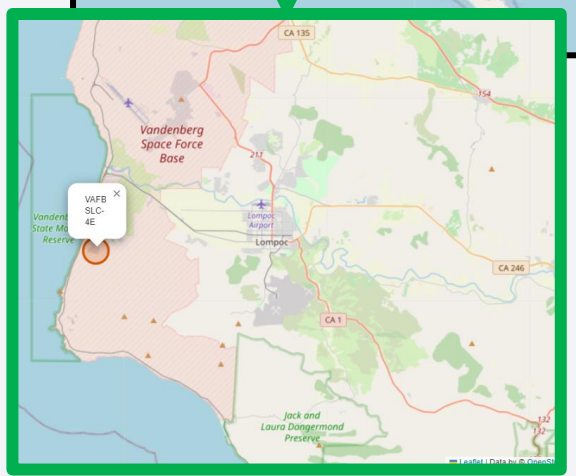
Using the COUNT command to count of landing outcomes combined with condition WHERE retrieve data between the date 2010-06-04 and 2017-03-20 (using BETWEEN) combined with GROUP BY and ORDER BY to retrieve Grouped by landing outcome in descending order.

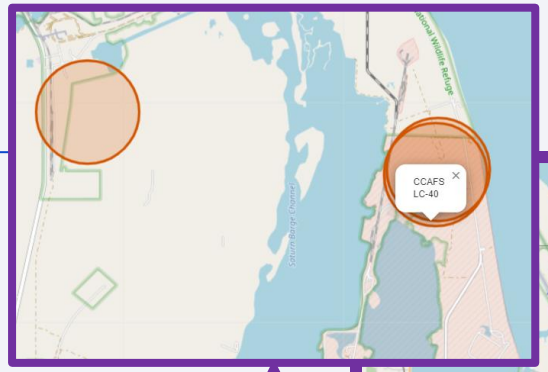
Section 3

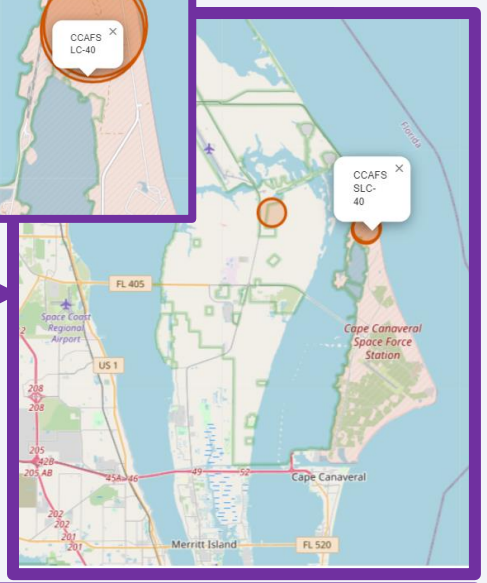
Launch Sites Proximities Analysis

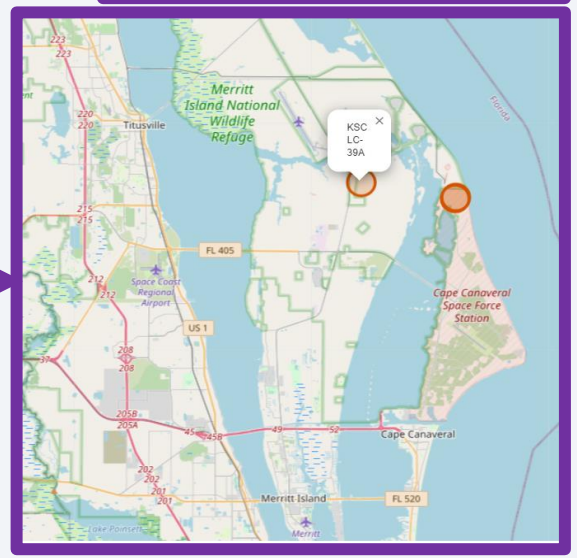
SPACEX LAUNCH SITE'S LOCATIONS ANALYSIS WITH FOLIUM





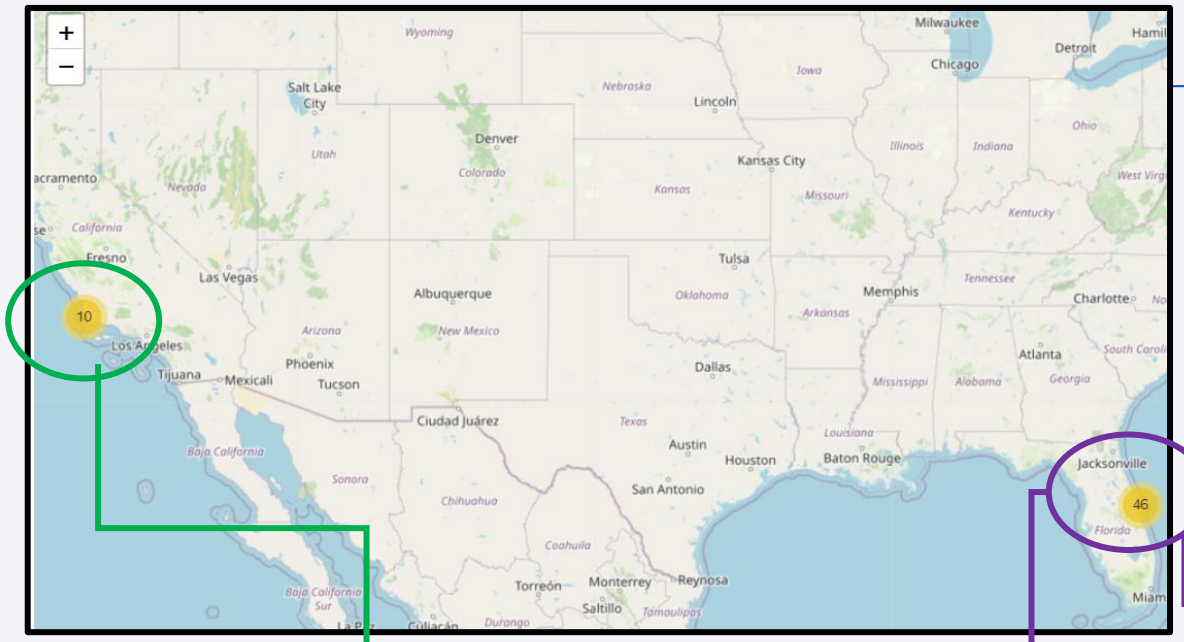




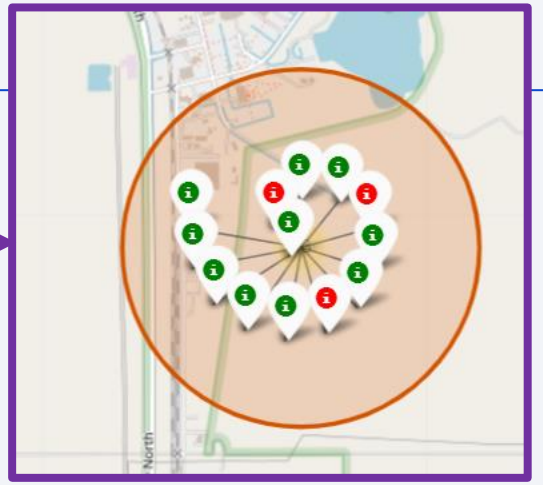


All launch sites is located at coast area. 3 in coast of Florida and 1 in coast of California.

SPACEX LAUNCH SITE'S LOCATIONS ANALYSIS WITH FOLIUM

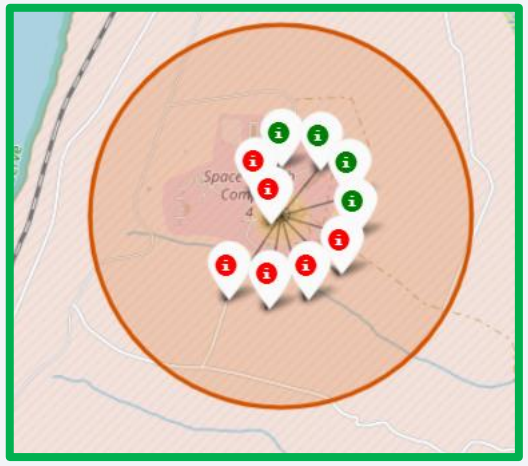


KSC LC-39A



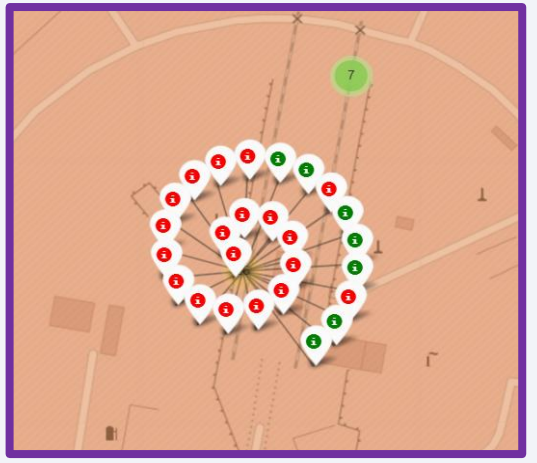
Marks were added to launch sites with green icons representing successful launches, and red icons representing failed launches.

VAFB SLC-44E

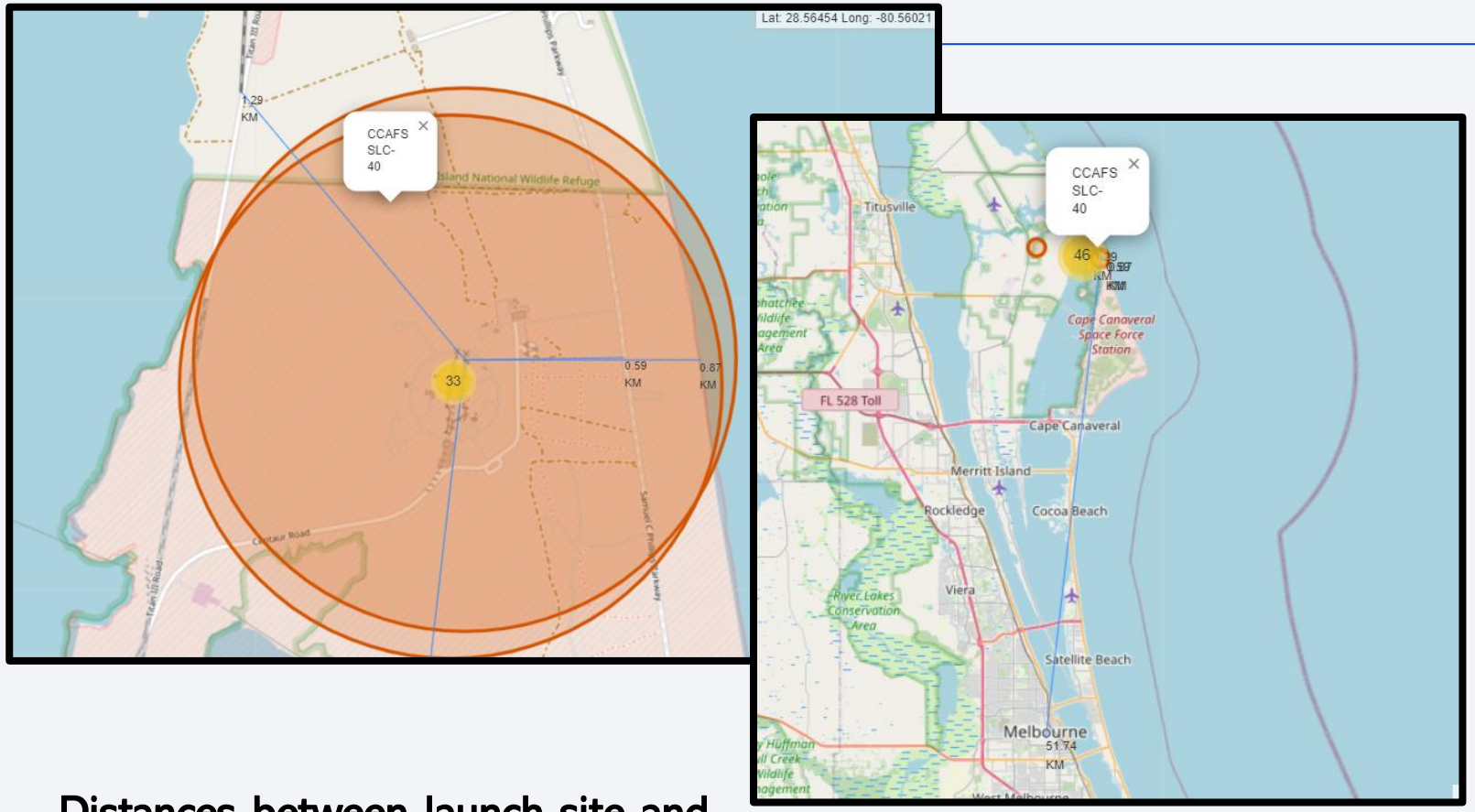


CCAFS SLC-40

CCAFS LC-40



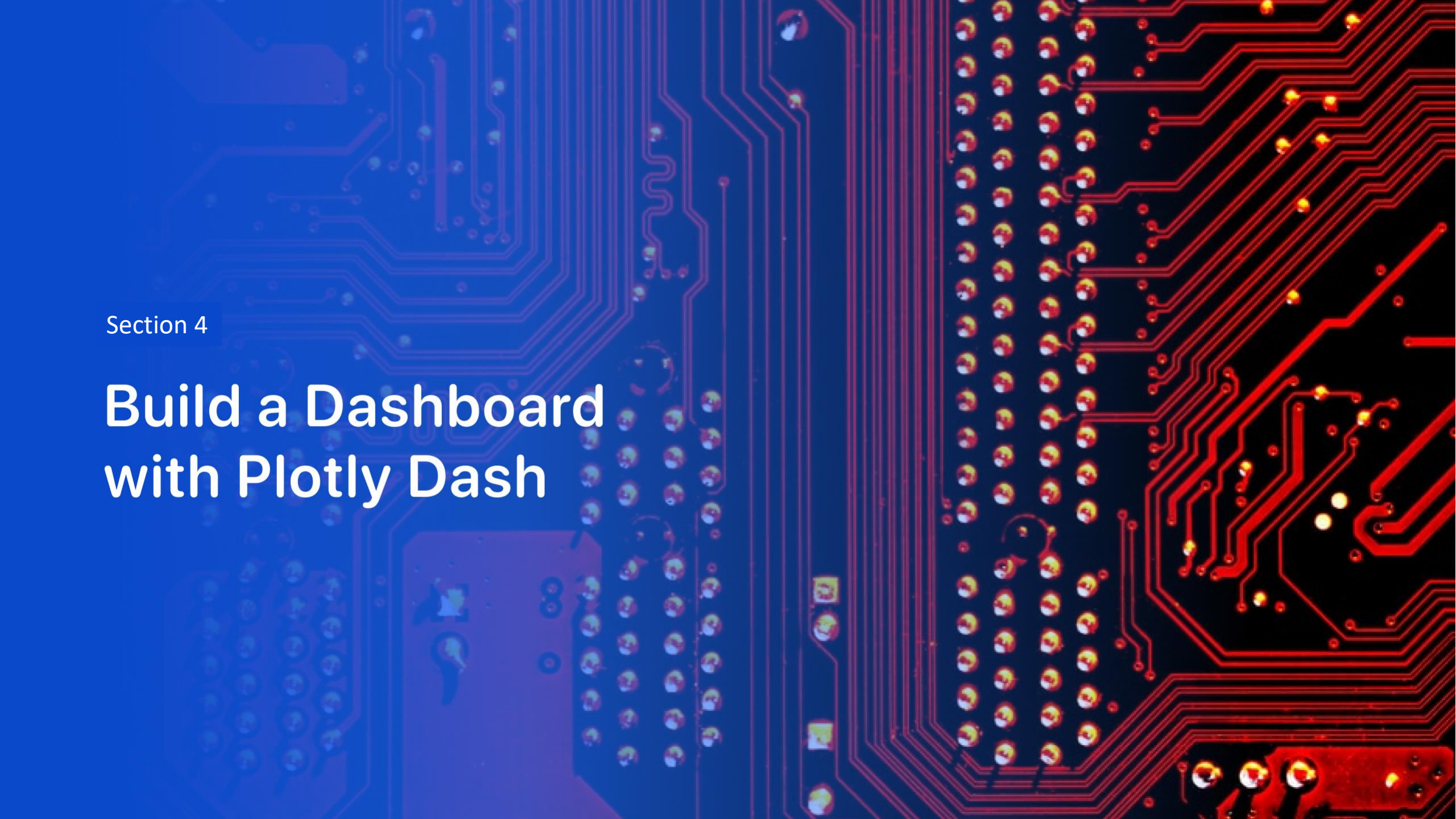
SPACEX LAUNCH SITE'S LOCATIONS ANALYSIS WITH FOLIUM



Distances between launch site and some places were measured. We could use it to answer the questions.

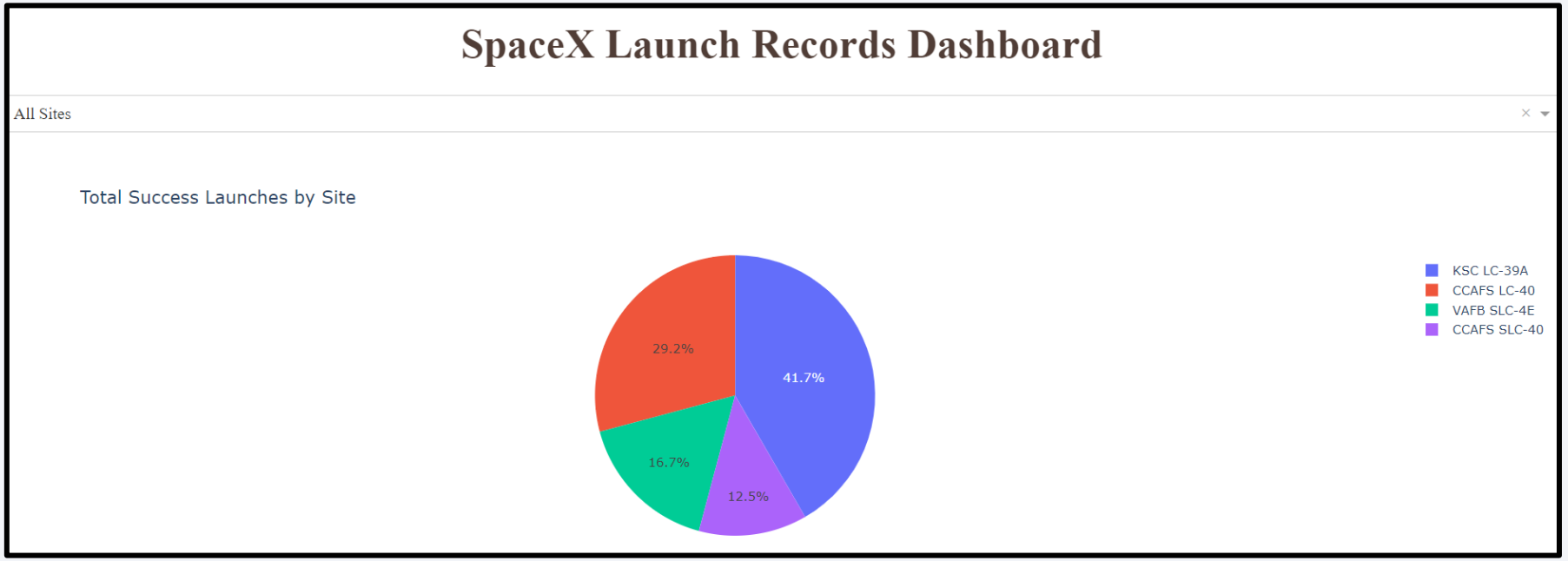
Questions:

1. **Are launch sites in close proximity to railways?**
   Yes. Distance is just 0.59 Km.
2. Are launch sites in close proximity to highways?
   Yes. Distance is 1.29 Km.
3. Are launch sites in close proximity to coastline?
   Yes. Distance is just 0.87 Km.
4. Do launch sites keep certain distance away from cities?
   Yes. Distance is 51,74 Km to nearest city.


Section 4
Build a Dashboard with Plotly Dash

INTERACTIVE DASHBOARD WITH PLOTLY AND DASH

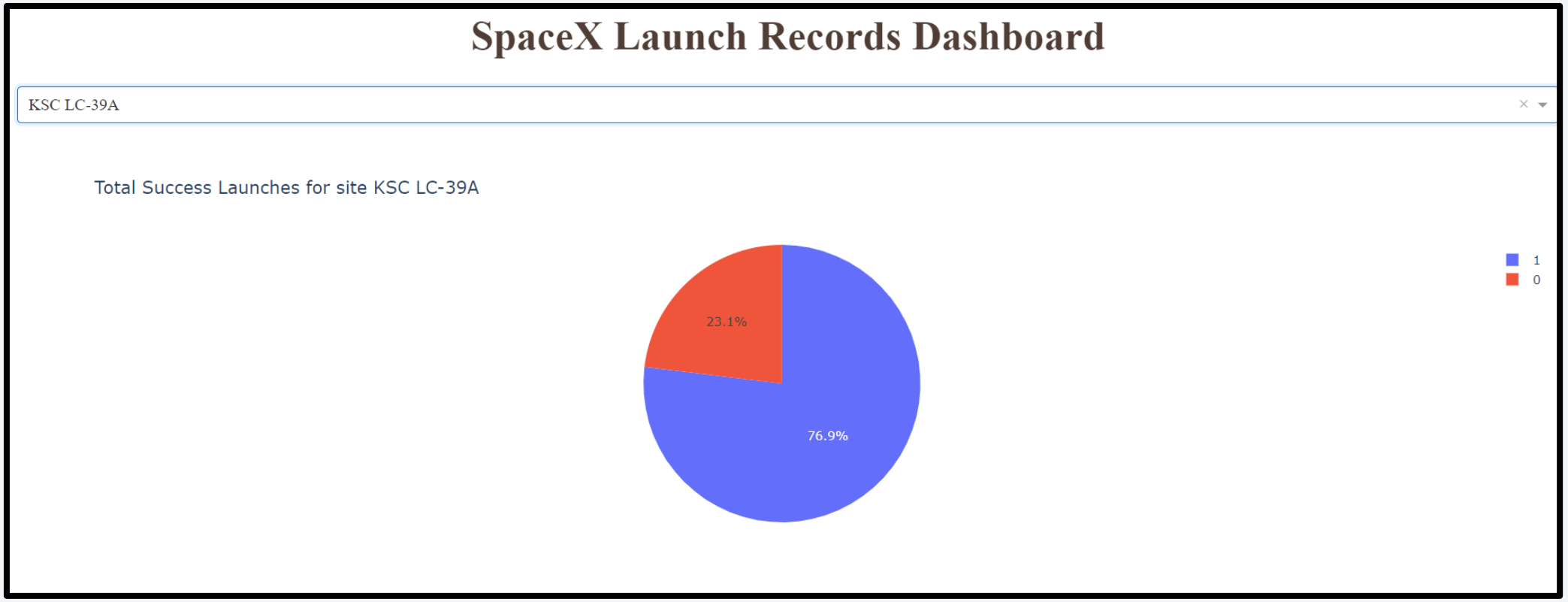
Piechart showing launch success count for all sites



We can see that KSC LC 39A has the highest rate of success with 41.7%, followed by CCAFS LC-40 with 29.2%.

INTERACTIVE DASHBOARD WITH PLOTLY AND DASH

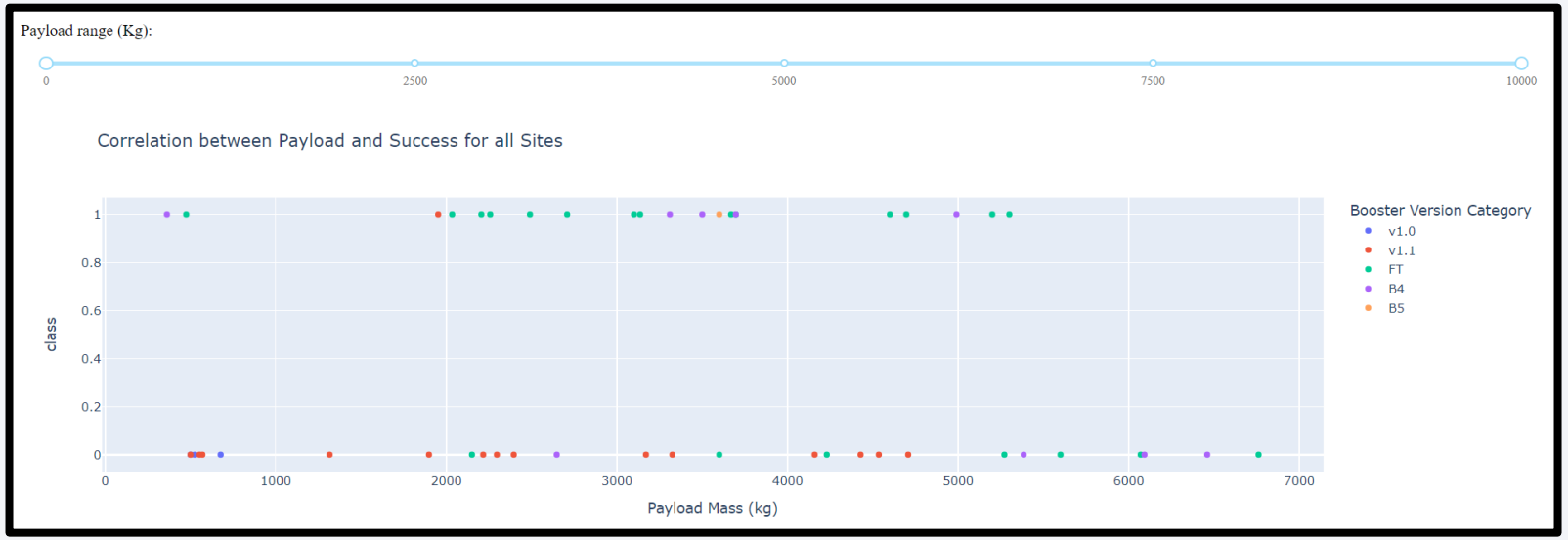
This Piechart shows the launch site with highest launch success ratio



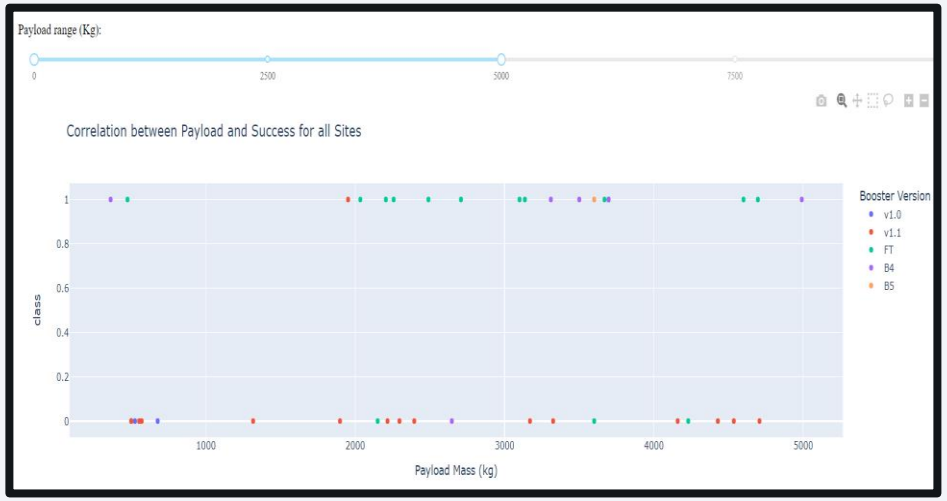
1 label is for success launches
0 label is for for fail launches

INTERACTIVE DASHBOARD WITH PLOTLY AND DASH

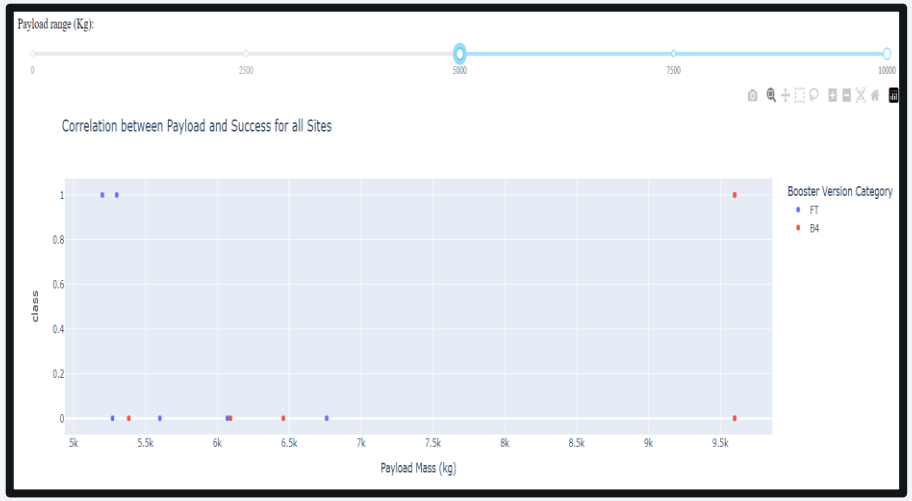
Payload vs. Launch Outcome scatter plot for all sites, with different payload selected in the range slider



Success rate increases with lower pay load mass is. All boosters version were uses to launch Pay load with mass < 5000 Kg. Only Two Booster versions category (FT and B4) were used to launch pay load mass > 5000.



Low Pay load Mass

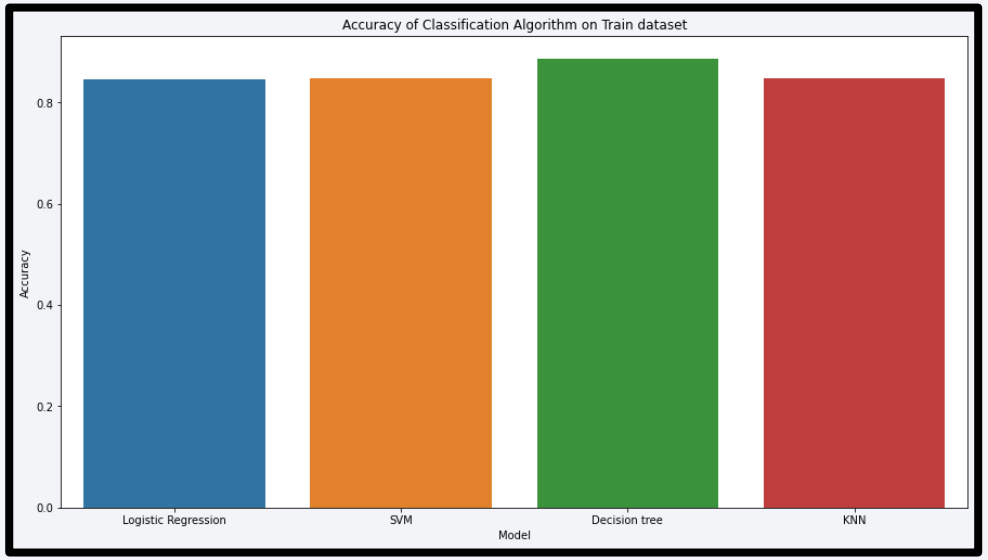


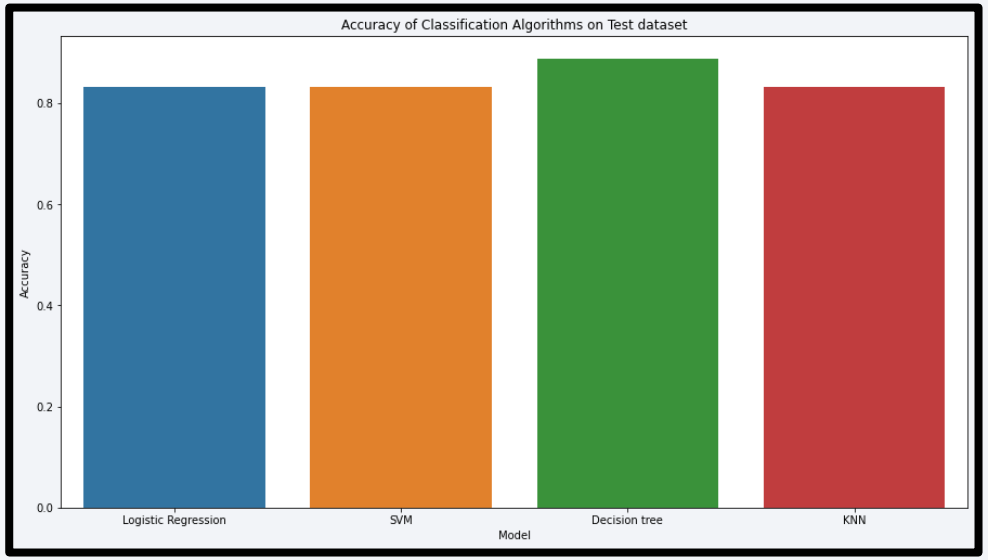
Higher Pay load Mass

Section 5

Predictive Analysis (Classification)

CLASSIFICATION ACCURACY

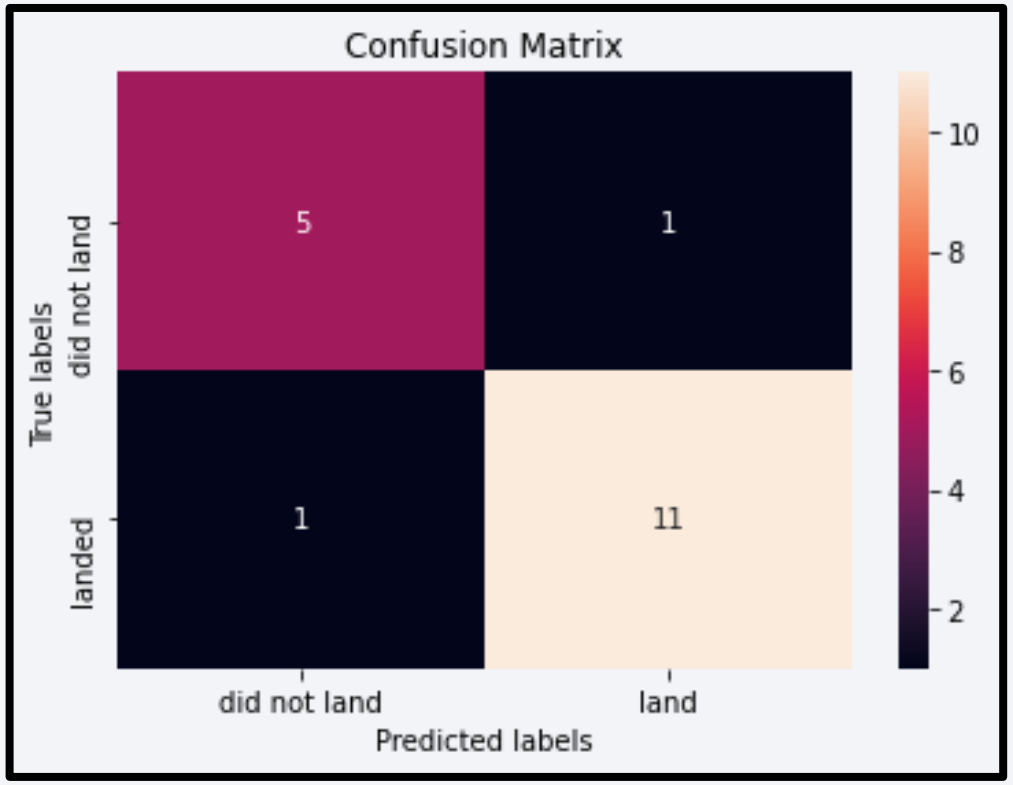




Decision tree has the best accuracy score (88.75 %) on train data and Test data (88,88 %). The accuracy score on Test data does not decreases compared to the train data. This shows how the algorithm can generalize the data.

CONFUSION MATRIX

Confusion matrix of the best performing model (Decision Tree)



Confusion matrix show how accurate is the Decision Tree model. For label "Landing" 11 of 12 were labeled correctly and for label "did not land" 5 of 6 were labeled correctly. This shows that model can predict both classes with a good accuracy.

CONCLUSIONS

- The launches success rate increased after flight number 30.
- CCAFS SLC 40 was the launch site more used, and It was used since the beginning of flights.
- KSC LC 39A had the highest rate of success with 41.7%, followed by CCAFS LC-40 with 29.2%.
- Launches with Pay Load Mass < 5000 Kg had more success rate.
- Launches between 2010 and 2013 all failed.
- Launches success started to increase after 2013 and get the highest success rate in 2019 with 90% success rate.
- ES-L1, GEO, HEO, and SSO orbits had the highest success rate (100%).
- SO (Heliocentric Orbit) had 0% of success rate.
- Decision Tree model was the best performing classification with an accuracy of 88,88%.

APPENDIX

DATA COLLECTION – SPACE X REST API

- **CUSTOMIZED FUNCTIONS**

```
# 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.

```
# 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.

```
# 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'])
```

```
# 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'])
```

APPENDIX

DATA COLLECTION – WEB SCRAPING - CUSTOMIZED FUNCTIONS AND LOGIC ITERATION LOOPS

```
def date_time(table_cells):
    """
    This function returns the data and time from the HTML  table cell
    Input: the  element of a table data cell extracts extra row
    """
    return [data_time.strip() for data_time in list(table_cells.strings)][0:2]

def booster_version(table_cells):
    """
    This function returns the booster version from the HTML  table cell
    Input: the  element of a table data cell extracts extra row
    """
    out=''.join([booster_version for i,booster_version in enumerate( table_cells.strings) if i%2==0][0:-1])
    return out

def landing_status(table_cells):
    """
    This function returns the landing status from the HTML table cell
    Input: the  element of a table data cell extracts extra row
    """
    out=[i for i in table_cells.strings][0]
    return out


def get_mass(table_cells):
    mass=unicodedata.normalize("NFKD", table_cells.text).strip()
    if mass:
        mass.find("kg")
        new_mass=mass[0:mass.find("kg")+2]
    else:
        new_mass=0
    return new_mass


def extract_column_from_header(row):
    """
    This function returns the landing status from the HTML table cell
    Input: the  element of a table data cell extracts extra row
    """
    if (row.br):
        row.br.extract()
    if row.a:
        row.a.extract()
    if row.sup:
        row.sup.extract()

    colunm_name = ' '.join(row.contents)

    # Filter the digit and empty names
    if not(colunm_name.strip().isdigit()):
        colunm_name = colunm_name.strip()
        return colunm_name
```

```
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)
            print('Flight number {}'.format(flight_number))

            datatimelist=date_time(row[0])

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

            # Time value
            # TODO: Append the time into launch_dict with key `Time`
            time = datatimelist[1]
            print('Time: {}'.format(time))
            launch_dict['Time'].append(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('Version Booster: {}'.format(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
            print('Launch Site: {}'.format(launch_site))
            launch_dict['Launch Site'].append(launch_site)

            # Payload
            # TODO: Append the payload into launch_dict with key `Payload`
            payload = row[3].a.string
            print('Payload: {}'.format(payload))
            launch_dict['Payload'].append(payload)
            # Payload Mass
            # TODO: Append the payload_mass into launch_dict with key `Payload mass`
            payload_mass = get_mass(row[4])
            print('Payload mass: {}'.format(payload_mass))
            launch_dict['Payload mass'].append(payload_mass)

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

            # Customer
            # TODO: Append the customer into launch_dict with key `Customer`
            if row[6].a != None:
                customer = row[6].a.string
            else:
                customer = 'None'
            print('Customer: {}'.format(customer))
            launch_dict['Customer'].append(customer)

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

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

Thank you!