IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Phone Sett Paing
12/11/2022

Outline

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

Executive Summary

Summary of methodologies

The main steps in this project are:

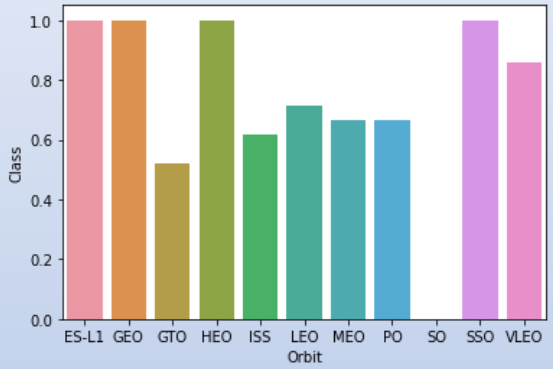
- Data Collection
- Data Wrangling
- Exploratory Data Analysis (EDA)
- Build Interactive Dashboard
- Predictive Analysis (Classification)

Summary of all results

The Outputs and Visualizations of this project:

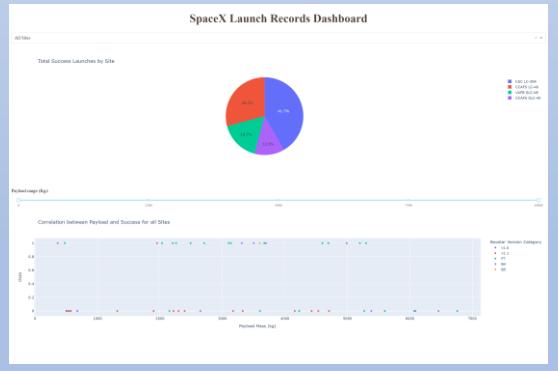
1. Exploratory Data Analysis (EDA)
2. Geospatial analytics
3. Interactive dashboard
4. Classification models

1. Success Rate of each Orbit Type

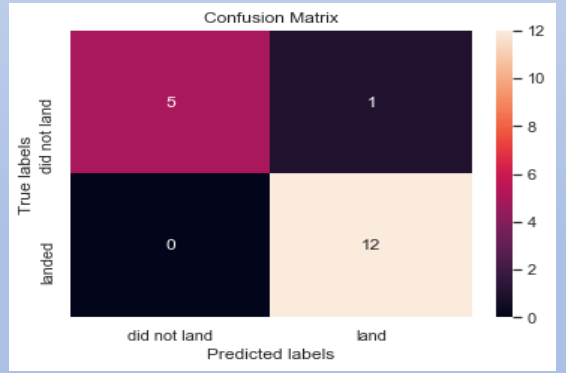


2. Success/Failed Launches for each sites

3. Interactive Dashboard



4. Decision Tree Confusion Matrix



Introduction

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

- The main problems that we are trying to answer is :

  - What factors influence SpaceX Falcon 9 launch success?
  - Will the first stage of the rocket land successfully?

Section 1

Methodology

Methodology

Executive Summary

- **Data collection methodology:**
  - Making GET requests to SpaceX REST API
  - Web Scraping with BeautifulSoup
- **Perform data wrangling:**
  - Find missing values
  - Replace missing values with mean value of the column
  - Used One Hot Encoding to transform categorical variables into factors then into integers
  - Classified data into new column ‘Class’ as 1 for Successful landing or 0 for failed landing
- **Perform exploratory data analysis (EDA) using visualization and SQL**
- **Perform interactive visual analytics using Folium and Plotly Dash**
- **Perform predictive analysis using classification models:**
  - Built and tuned multiple classification models to predict landing success
  - Used Grid Search and Cross Validation to find the best model parameters for each model tested (Logistic, SVM, Decision Tree, and KNN)
  - Split Data into testing and training to test model accuracy resilience on Out of Sample Data
  - Selected top performing Model on both testing and training set based on accuracy of predictions

Data Collection – SpaceX API

1. Make a GET response to the API and convert the response to a .json file then to a Pandas DataFrame
2. Clean the data and extract desired information using custom functions to stored it in define lists
3. Assign cleaned data to a new data frame using dictionary of lists
4. Filter the data frame to only include Falcon 9 launches and clean the data frame to replace missing values of PayloadMass with the mean PayloadMass value, then export it to csv

1

```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.a
```

```
response.status_code
```

```
response = requests.get(static_json_url)
data=pd.json_normalize(response.json())
```

2

```
# Lets take a subset of our dataframe keeping only the features we want and the flight
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
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
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 d
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)]
```

```
#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)
```

3

```
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}
```

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

4

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

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

```
# Replace the np.nan values with its mean value
data_falcon9.fillna(value={'PayloadMass':mean_PayloadMass}, inplace = True)
```

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

GitHub Link

Data Collection - Scraping

1. Get Response from HTML and create Soup object
2. Find all tables within the HTML page and collect all column header names from the tables
3. Create a Dictionary and extract table data to dictionary
4. Convert the dictionary to a Pandas DataFrame and export it to csv

1

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

```
# Use BeautifulSoup() to create a Beauti
soup =BeautifulSoup(data, 'html5lib')
```

2

```
# Use the find_all function in the Beau
# Assign the result to a list called `h
html_tables = soup.find_all('table')
```

```
column_names = []

# Apply find_all() function with `th` element
# Iterate each th element and apply the provi
# Append the Non-empty column name (`if name

for row in first_launch_table.find_all('th'):
    name = extract_column_from_header(row)
    if (name != None and len(name) > 0):
        column_names.append(name)
```

3

```
launch_dict= dict.fromkeys(column_names)

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

# Let's initial the launch_dict with eac
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']=[]
```

```
extracted_row = 0
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders collapsible")):
    for rows in table.find_all("tr"):
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
        else:
            flag=False
        row=rows.find_all('td')
        if flag:
            extracted_row += 1

            launch_dict['Flight No.'].append(flight_number)
            print('Flight No.: ',flight_number)
            datatimelist=date_time(row[0])

            date = datatimelist[0].strip(',')
            launch_dict['Date'].append(date)
            print('Date: ', date)
```

4

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

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

GitHub Link

Data Wrangling

Context:

- Some of the important attributes in the dataset:
  - Orbit
  - LaunchSite
  - Outcome
- LaunchSite contains the different launch sites:
  - Vandenberg AFB Space Launch
  - Kennedy Space Center
  - CCAFS SLC 40
- Orbit contains the different orbits of the payload and each launch aims to an dedicated orbit:
  - LEO: Low Earth Orbit
  - GTO A geosynchronous orbit
- Outcome indicates if the first stage successfully landed. There are 8 of them:
  - True Ocean – the mission outcome was successfully landed to a specific region of the ocean
  - False Ocean – the mission outcome was unsuccessfully landed to a specific region of the ocean
  - True RTLS – the mission outcome was successfully landed to a ground pad
  - False RTLS – the mission outcome was unsuccessfully landed to a ground pad
  - True ASDS – the mission outcome was successfully landed to a drone ship
  - False ASDS – the mission outcome was unsuccessfully landed to a drone ship
  - None ASDS and None None – these represent a failure to land.
- To determine whether a booster will successfully land, it is best to have a binary column based on Outcome column, i.e., where the value of 1 or 0, representing the success of the landing.

Data Wrangling

1. Load the dataset and determine the:
   - Number of launches on each site
   - Number and occurrence of each orbit
   - Number and occurrence of mission outcome per orbit type
2. Defining a set of unsuccessfully landed Outcome, bad_outcomes
3. Creating a list, landing_class, where the element is 0 if the corresponding row in Outcome is in the set bad_outcomes, otherwise, it's 1.
4. Create a Class column that contains the values from the list landing_class and export the DataFrame as a .csv file

1

```
df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.a
df.head(10)
```

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

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

```
# 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
```

2

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

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

3

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

4

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

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

GitHub Link

EDA with Data Visualization

- Scatter charts were plotted to visualize the relationships between:
  - Flight Number and Launch Site
  - Payload and Launch Site
  - Flight Number and Orbit Type
  - Payload and Orbit Type
- Because Scatter charts are useful to observe relationships, or correlations, between two numeric variables.
- A bar chart was plotted to visualize the success rate of each orbit type
- Because Bar charts are used to compare a numerical value to a categorical variable.
- Line charts were plotted to visualize the launch success yearly trend
- Because Line charts contain numerical values on both axes, and are generally used to show the change of a variable over time.

GitHub Link

EDA with SQL

The SQL queries performed:

- 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 the average payload mass carried by booster version F9 v1.1
- List the date when the first successful landing outcome on a ground pad was achieved
- List the names of the boosters which had success on a drone ship and a payload mass between 4000 and 6000 kg
- List the total number of successful and failed mission outcomes
- List the names of the booster versions which have carried the maximum payload mass
- List the failed landing outcomes on drone ships, their booster versions, and launch site names for 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

Build an Interactive Map with Folium

- Mark all Launch Sites on the map by using their Latitude and Longitude Coordinates
- Mark the success/failed launches for each site on a map by assigning a marker colour of successful (class = 1) as green, and failed (class = 0) as red
- Used lines and points to measure (via Haversine's Distance Formula) and label the minimum distances of the launch sites to:
  - Cities
  - Highways
  - Coastlines
  - Railways
- They are used because launch success rate may also depend on the location and proximities of a launch site, i.e., the initial position of rocket trajectories. Finding an optimal location for building a launch site certainly involves many factors and hopefully we could discover some of the factors by analysing the existing launch site locations.
- 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? About 51.74 km

GitHub Link

Build a Dashboard with Plotly Dash

The following plots were added to a Plotly Dash dashboard to have an interactive visualisation of the data:

1. Pie chart showing the Total Success Launches by Site
   - To see which sites are most successful
   - This chart could also be filtered to see the success/failure ratio for an individual site

2. Scatter graph to show the Correlation between Payload and Success for all Sites
   - This graph could also be filtered by ranges of payload masses
   - It could also be filtered by booster version

GitHub Link

Predictive Analysis (Classification)

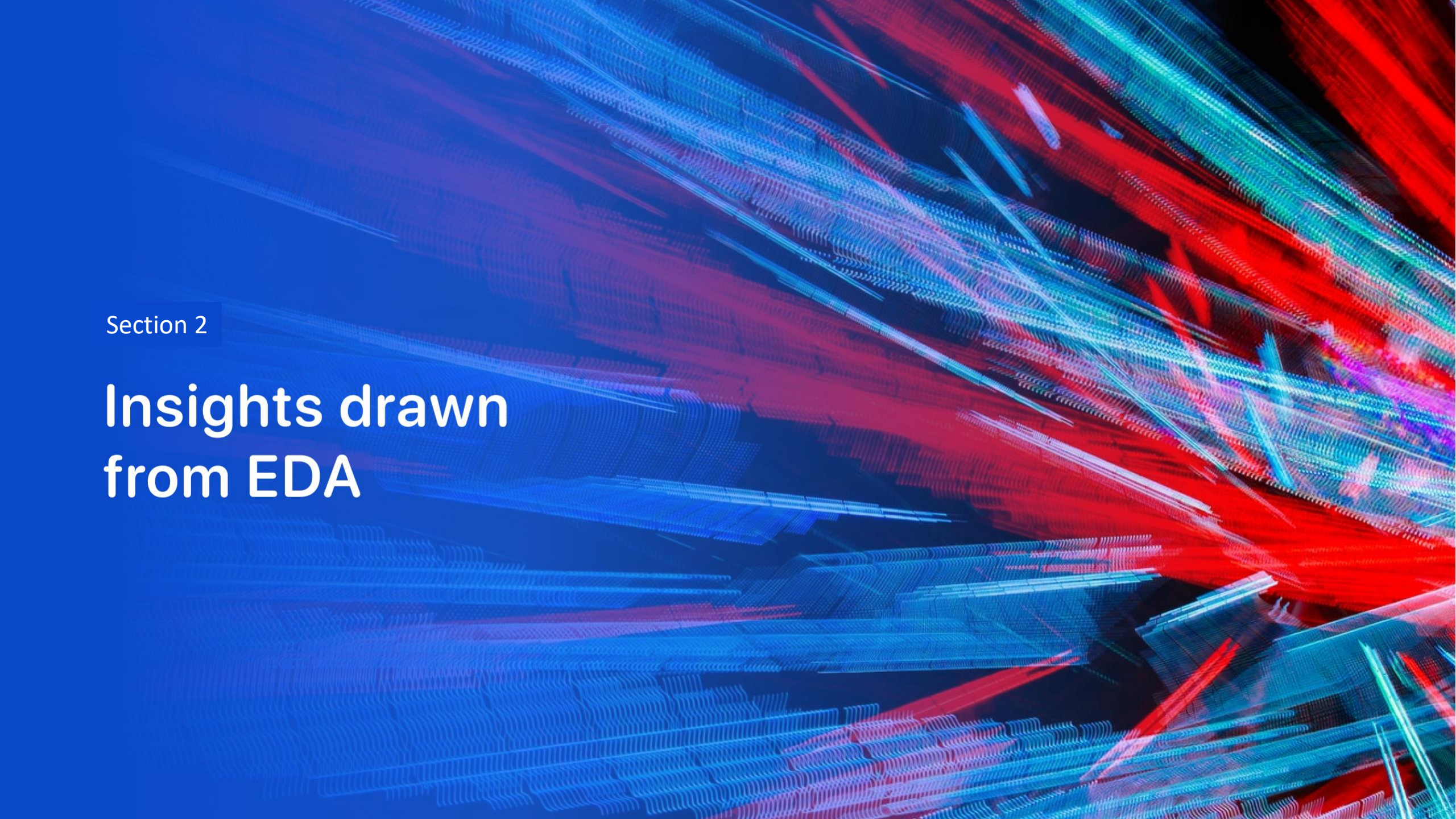
The following steps were taking to develop, evaluate, and find the best performing classification model:

- Building Model:
  - Transform data to Scale the columns
  - Split Data into Testing and Training Sets
  - Selected machine learning algorithms to use for classification (KNN, Decision Tree, SVM, Logistic Regression)
  - Use Grid Search and Cross Validation to find best tuning parameters for each model fitting on training sets
- Evaluating Model:
  - Check accuracy of each model on training and testing sets
  - Plot Confusion Matrix
- Improving Model:
  - Feature Engineering
  - Algorithm tuning
- Selecting the best performing classifier:
  - Model with the best accuracy score on test set is the best model. If there is a tie, check accuracy on training set as well

GitHub Link

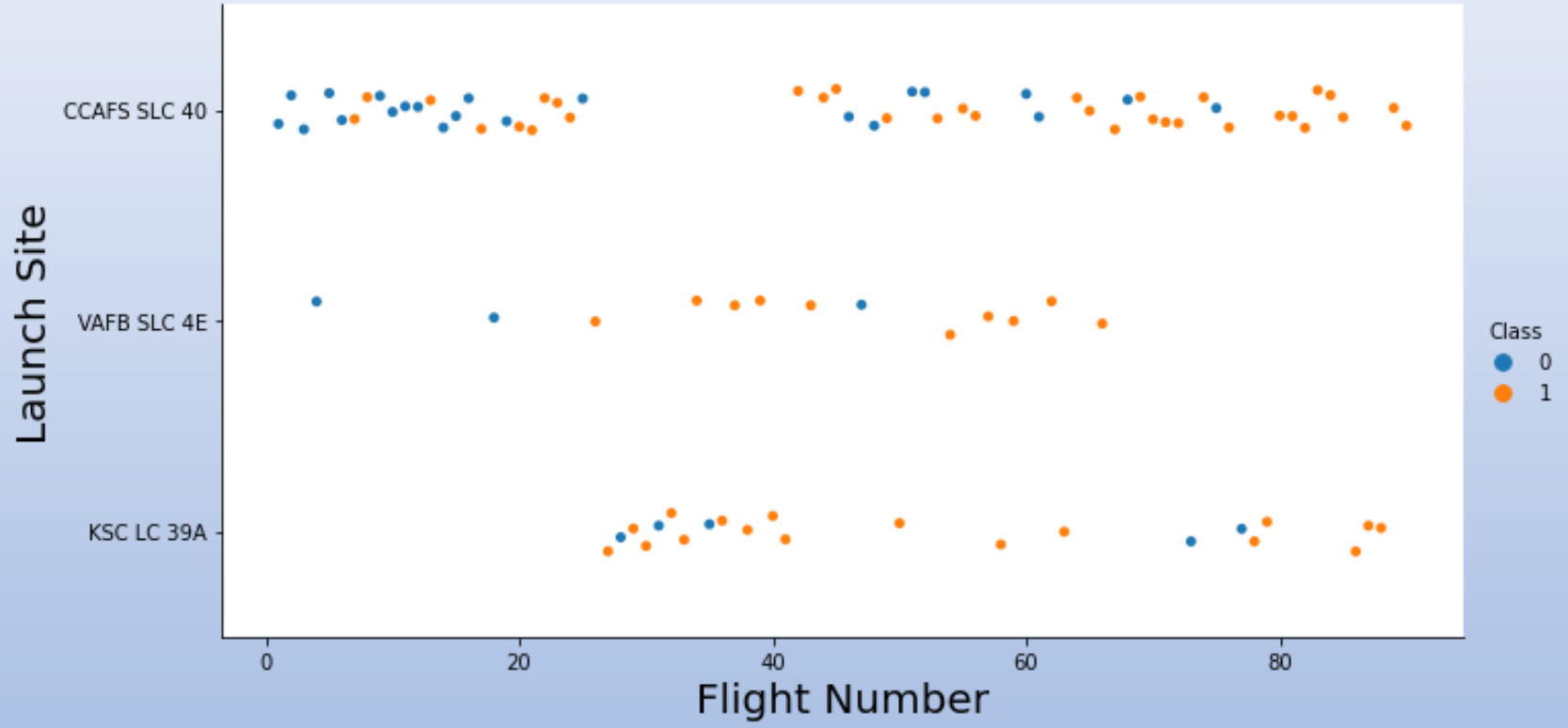
Results

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


Section 2
Insights drawn from EDA

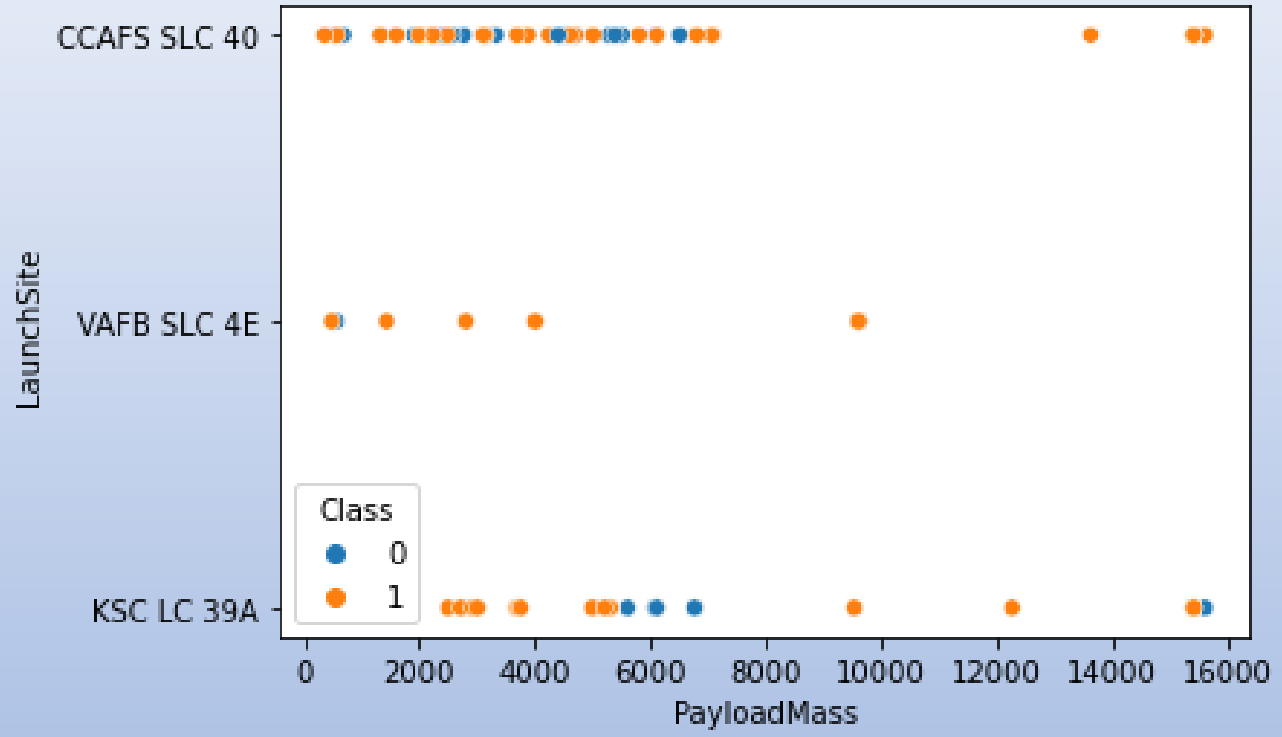
Flight Number vs. Launch Site

- As the number of flights increases, the rate of success at a launch site increases.
- Most of early flights were generally unsuccessful.
- Most of the flights were launch from CCAFS SLC 40.
- No early flights were launched from KSC LC 39A.



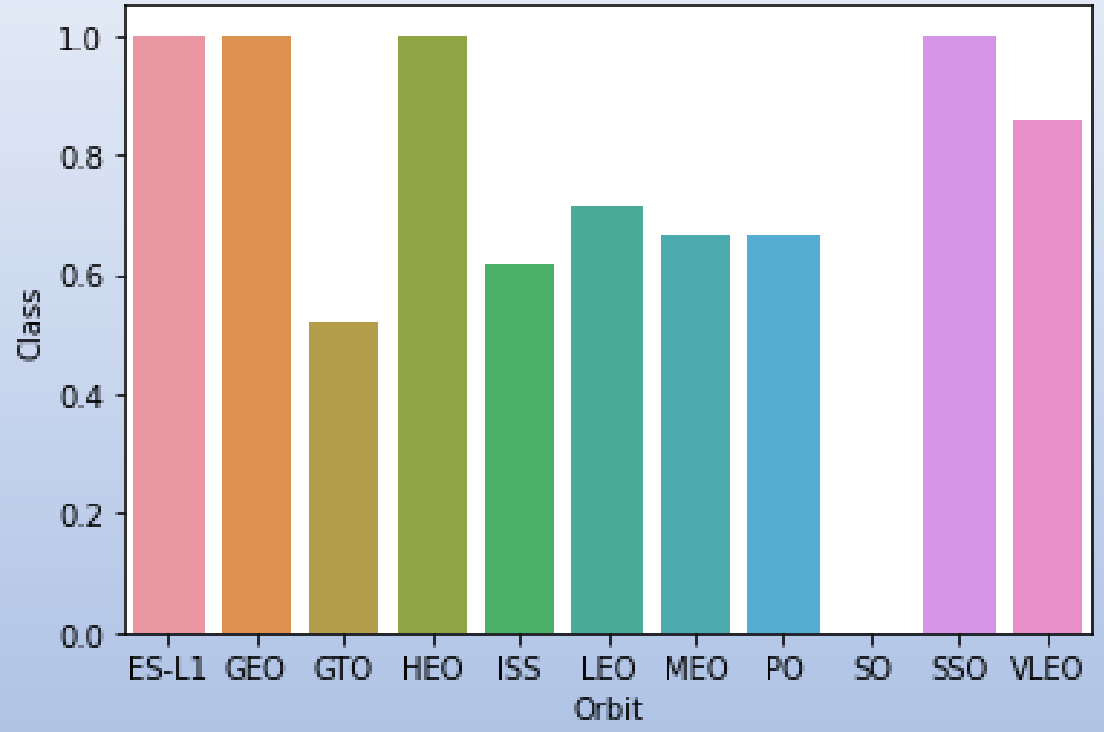
Payload vs. Launch Site

- There are no rockets launched for heavy payload mass(greater than 10000) from VAFB-SLC launch site.
- There is no clear correlation between payload mass and success rate for a given launch site.
- Most of the launches from CCAFS SLC 40 are lighter payloads.



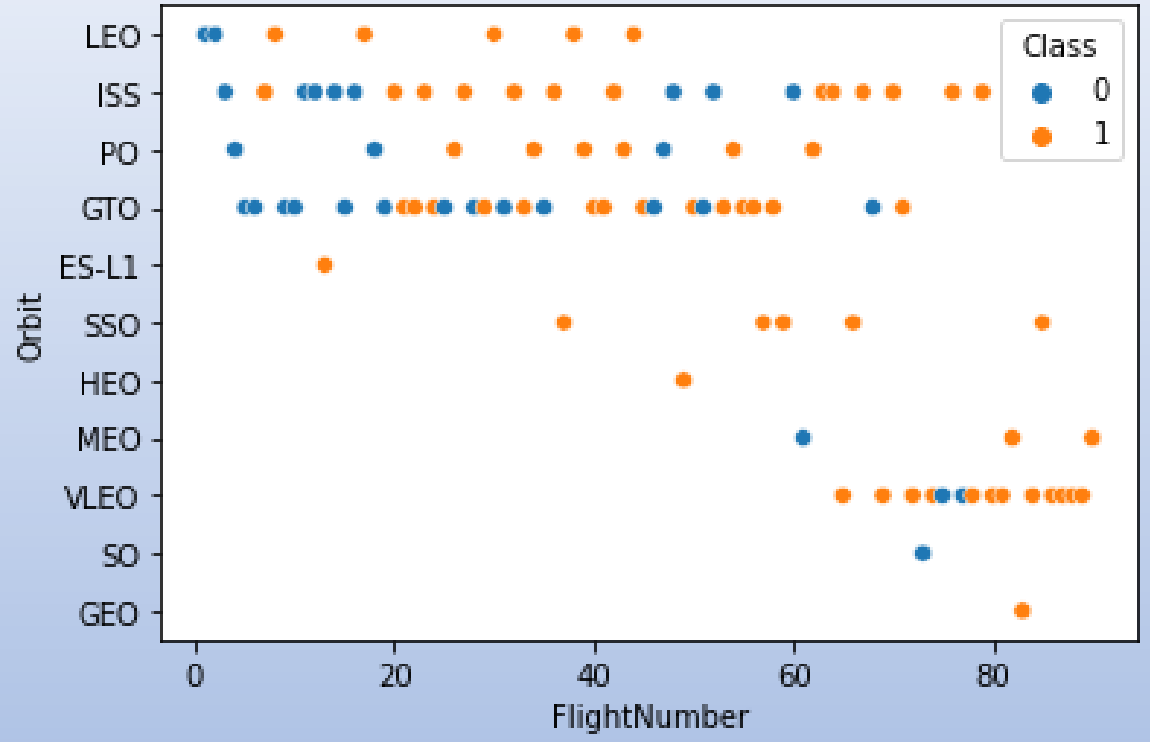
Success Rate vs. Orbit Type

- The following orbits have the highest (100%) success rate:
  - ES-L1 (Earth-Sun First Lagrangian Point)
  - GEO (Geostationary Orbit)
  - HEO (High Earth Orbit)
  - SSO (Sun-synchronous Orbit)
- SO (Heliocentric Orbit) has the lowest (0%) success rate.



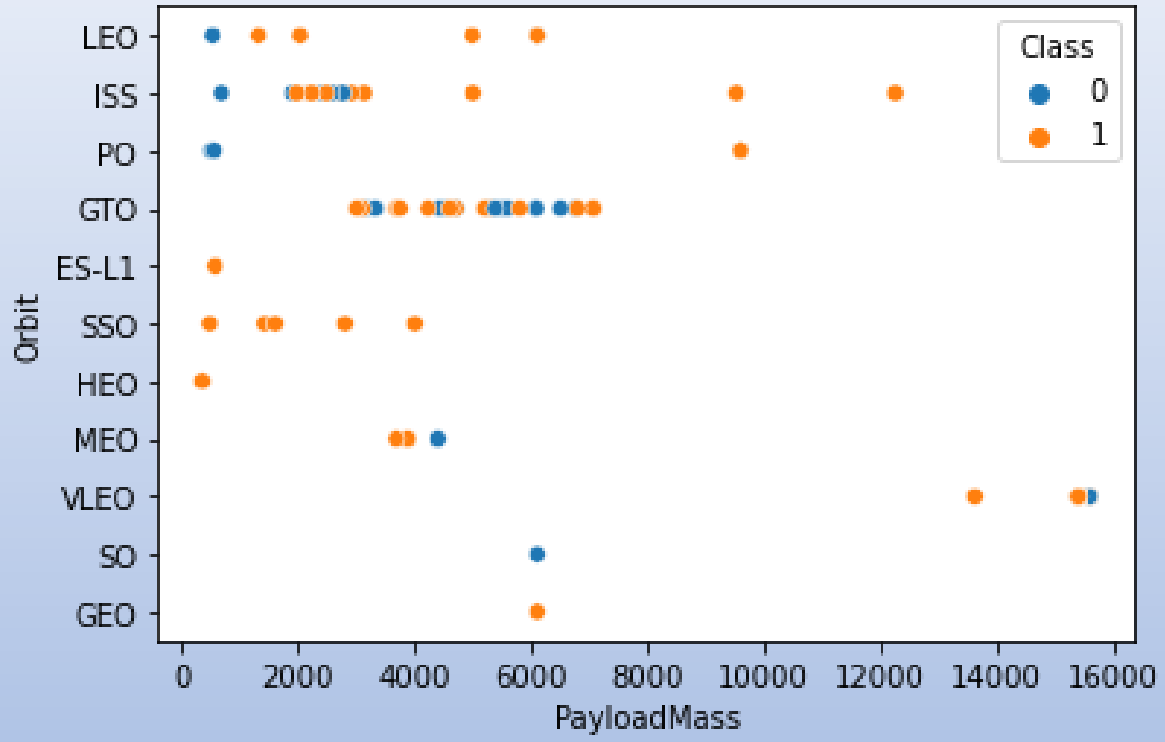
Flight Number vs. Orbit Type

- The 100% success rate of GEO, HEO, and ES - L1 orbits can be explained by only having 1 flight into the respective orbits.
- Compare to them, the 100% success rate in SSO is much more impressive, with 5 successful flights.
- Early flights tends to have lower success rate compare to later flights
- But for GTO, there is little relationship between Flight Number and Success Rate.



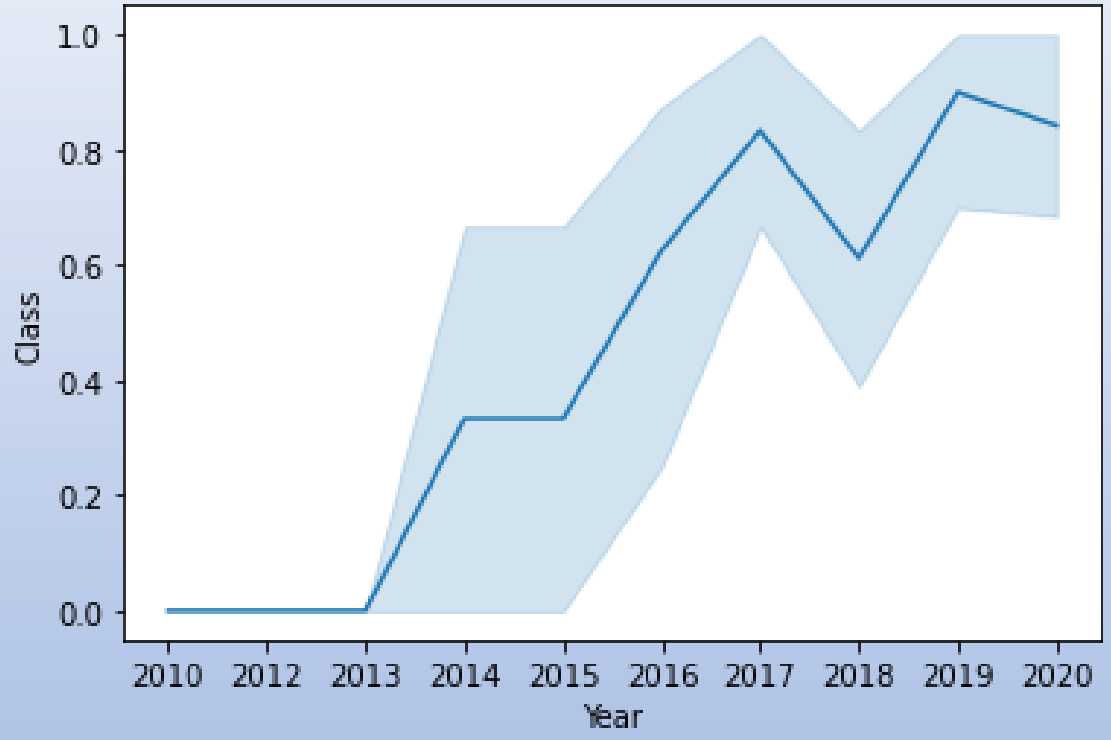
Payload vs. Orbit Type

- The 100% success rate of ES-L1, GEO, HEO, SSO were only with lower payload mass, below 7,000 kg.
- Compare to them, ISS, PO, VLEO have more success with heavy payload mass, above 1,000.
- For GTO, the relationship between payload mass and success rate is unclear.



Launch Success Yearly Trend

- We can see that there were no successful landings prior to 2013.
- After 2013, the success rate generally increased, despite small dips in 2018 and 2020.
- But there was always a greater than 50% chance of success after 2016.



All Launch Site Names

```
cur.execute('SELECT DISTINCT "Launch_Site" FROM SPACEXTBL')
rows = cur.fetchall()
for row in rows:
    print(row)
```

```
('CCAFS LC-40',)
('VAFB SLC-4E',)
('KSC LC-39A',)
('CCAFS SLC-40',)
```

- Find the names of the unique launch sites.
- The word DISTINCT returns only unique values from the Launch_Site column of the SPACEXTBL table.

Launch Site Names Begin with 'CCA'

```
cur.execute('SELECT * FROM SPACEXTBL WHERE "Launch_Site" LIKE "CCA%" LIMIT 5')
rows = cur.fetchall()
for row in rows:
    print(row)
```

```
('04-06-2010', '18:45:00', 'F9 v1.0  B0003', 'CCAFS LC-40', 'Dragon Spacecraft Qualification Unit', 0, 'LEO', 'SpaceX', 'Success', 'Failure (parachute)')
('08-12-2010', '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)')
('22-05-2012', '07:44:00', 'F9 v1.0  B0005', 'CCAFS LC-40', 'Dragon demo flight C2', 525, 'LEO (ISS)', 'NASA (COTS)', 'Success', 'No attempt')
('08-10-2012', '00:35:00', 'F9 v1.0  B0006', 'CCAFS LC-40', 'SpaceX CRS-1', 500, 'LEO (ISS)', 'NASA (CRS)', 'Success', 'No attempt')
('01-03-2013', '15:10:00', 'F9 v1.0  B0007', 'CCAFS LC-40', 'SpaceX CRS-2', 677, 'LEO (ISS)', 'NASA (CRS)', 'Success', 'No attempt')
```

- Find 5 records where launch sites begin with ‘CCA’.
- LIKE keyword is used with the wild card ‘CCA%’ to retrieve string values beginning with ‘CCA’.
- LIMIT 5 fetches only 5 records

Total Payload Mass

```
cur.execute('SELECT SUM("PAYLOAD_MASS__KG_"), Customer \
FROM SPACEXTBL WHERE "Customer"="NASA (CRS)"')
total_payload_mass = cur.fetchone()
print(total_payload_mass)
```

```
(45596, 'NASA (CRS)')
```

- Calculate the total payload carried by boosters from NASA
- The SUM keyword is used to calculate the total of the PAYLOAD_MASS__KG_ column
- Filter using WHERE keyword to only sum PAYLOAD_MASS__KG_ column with boosters from NASA (CRS)

Average Payload Mass by F9 v1.1

```
cur.execute('SELECT AVG("PAYLOAD_MASS__KG_"), Booster_Version FROM SPACEXTBL \
WHERE "Booster_Version"="F9 v1.1"')
average_payload_mass = cur.fetchone()
print(average_payload_mass)
```

```
(2928.4, 'F9 v1.1')
```

- Calculate the average payload mass carried by booster version F9 v1.1
- The AVG keyword is used to calculate the average of the PAYLOAD_MASS__KG_ column
- Filter using WHERE keyword to only average PAYLOAD_MASS__KG_ column with the F9 v1.1 booster version.

First Successful Ground Landing Date

```
cur.execute('SELECT MIN(Date), "Landing _Outcome" FROM SPACEXTBL \
WHERE "Landing _Outcome"="Success (ground pad)"')
date = cur.fetchone()
print(date)
```

```
('01-05-2017', 'Success (ground pad)')
```

- Find the dates of the first successful landing outcome on ground pad
- The MIN keyword is used to calculate the minimum of the DATE column, i.e. the first date
- Filter using WHERE keyword to only show minimum date of the successful ground pad landings

Successful Drone Ship Landing with Payload between 4000 and 6000

```
cur.execute('SELECT "Booster_Version", "Landing _Outcome" FROM SPACEXTBL \
WHERE ("Landing _Outcome"="Success (drone ship)") \
AND ("PAYLOAD_MASS__KG_" BETWEEN 4000 AND 6000)')
rows = cur.fetchall()
for row in rows:
    print(row)
```

```
('F9 FT B1022', 'Success (drone ship)')
('F9 FT B1026', 'Success (drone ship)')
('F9 FT  B1021.2', 'Success (drone ship)')
('F9 FT  B1031.2', 'Success (drone ship)')
```

- List the names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000.
- WHERE keyword is used to filter the results to only include, those which have successfully landed on drone ship
- The BETWEEN keyword allows for 4000 < payload mass < 6000 values to be selected.

Total Number of Successful and Failure Mission Outcomes

```
cur.execute('SELECT "Mission_Outcome", COUNT("Mission_Outcome") \
FROM SPACEXTBL GROUP BY "Mission_Outcome"')
rows = cur.fetchall()
for row in rows:
    print(row)
```

```
('Failure (in flight)', 1)
('Success', 98)
('Success ', 1)
('Success (payload status unclear)', 1)
```

- Calculate the total number of successful and failure mission outcomes
- The COUNT keyword is used to calculate the total number of mission outcomes
- The GROUPBY keyword is also used to group these results by the type of mission outcome

Boosters Carried Maximum Payload

```
cur.execute('SELECT DISTINCT("Booster_Version"), "PAYLOAD_MASS__KG_" FROM SPACEXTBL \
WHERE "PAYLOAD_MASS__KG_" = (SELECT MAX("PAYLOAD_MASS__KG_") FROM SPACEXTBL)')
rows = cur.fetchall()
for row in rows:
    print(row)
```

```
('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)
```

- List the names of the booster which have carried the maximum payload mass
- A subquery is used here. The SELECT statement within the brackets finds the maximum payload
- This value is used in the WHERE condition.
- The DISTINCT keyword is then used to retrieve only distinct /unique booster versions.

2015 Launch Records

```
cur.execute('SELECT SUBSTR(Date,4,2), "Landing _Outcome", "Booster_Version", "Launch_Site" FROM SPACEXTBL \
WHERE (SUBSTR(Date,7,4)="2015") AND ("Landing _Outcome"="Failure (drone ship)")')
rows=cur.fetchall()
for row in rows:
    print(row)
```

```
('01', 'Failure (drone ship)', 'F9 v1.1 B1012', 'CCAFS LC-40')
('04', 'Failure (drone ship)', 'F9 v1.1 B1015', 'CCAFS LC-40')
```

- List the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015
- SQLLite does not support monthnames. So substr(Date, 4, 2) is used as month to get the months and substr(Date,7,4) = '2015' for year
- The WHERE keyword is used to filter the results for only for the year of 2015, AND only failed landing outcomes.

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

```
cur.execute('SELECT "Landing _Outcome", COUNT("Landing _Outcome") FROM SPACEXTBL \
WHERE ((substr(DATE,7,4)||substr(DATE,4,2)||substr(DATE,1,2)) BETWEEN "20100604" AND "20170320") \
GROUP BY "Landing _Outcome" \
ORDER BY COUNT("Landing _Outcome") DESC')
rows = cur.fetchall()
for row in rows:
    print(row)
```

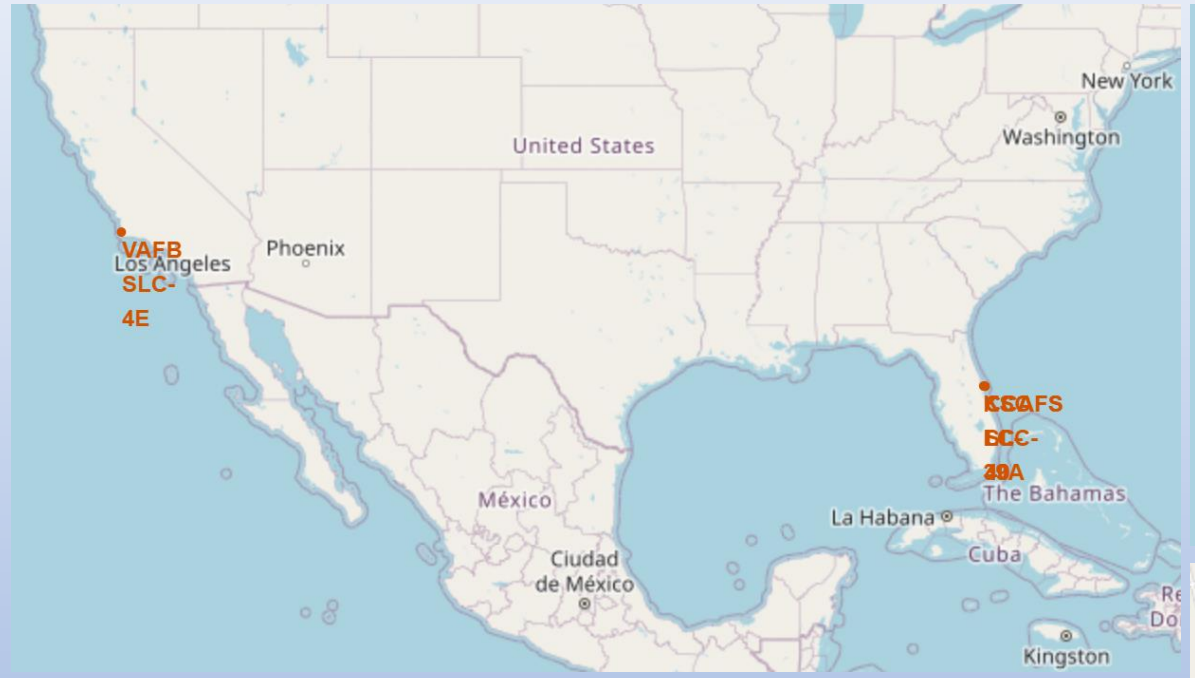
```
('No attempt', 10)
('Success (drone ship)', 5)
('Failure (drone ship)', 5)
('Success (ground pad)', 3)
('Controlled (ocean)', 3)
('Uncontrolled (ocean)', 2)
('Failure (parachute)', 2)
('Precluded (drone ship)', 1)
```

- 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
- The WHERE keyword is used with the BETWEEN keyword to filter the results to dates only within those specified.
- The results are then grouped and ordered, using the keywords GROUP BY and ORDER BY, respectively, where DESC is used to specify the descending order

Section 3

Launch Sites Proximities Analysis

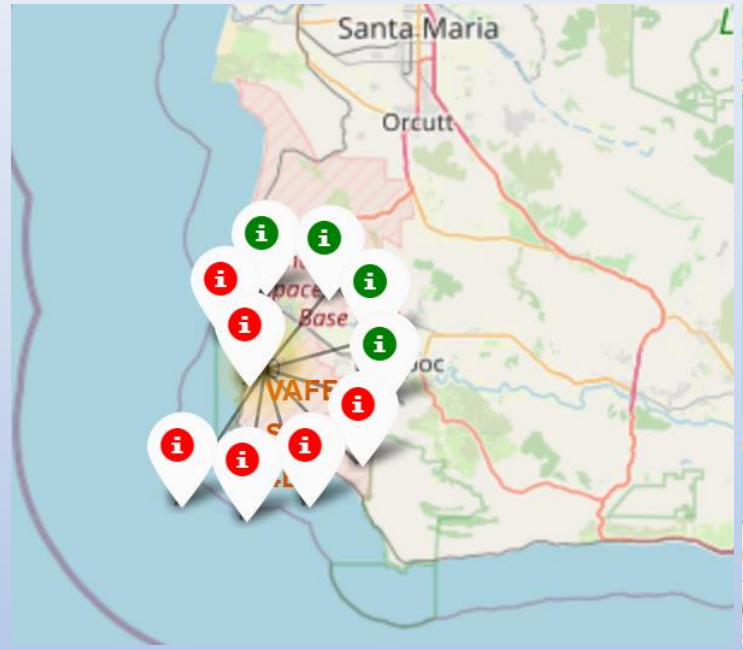
All Launch Sites On A Map

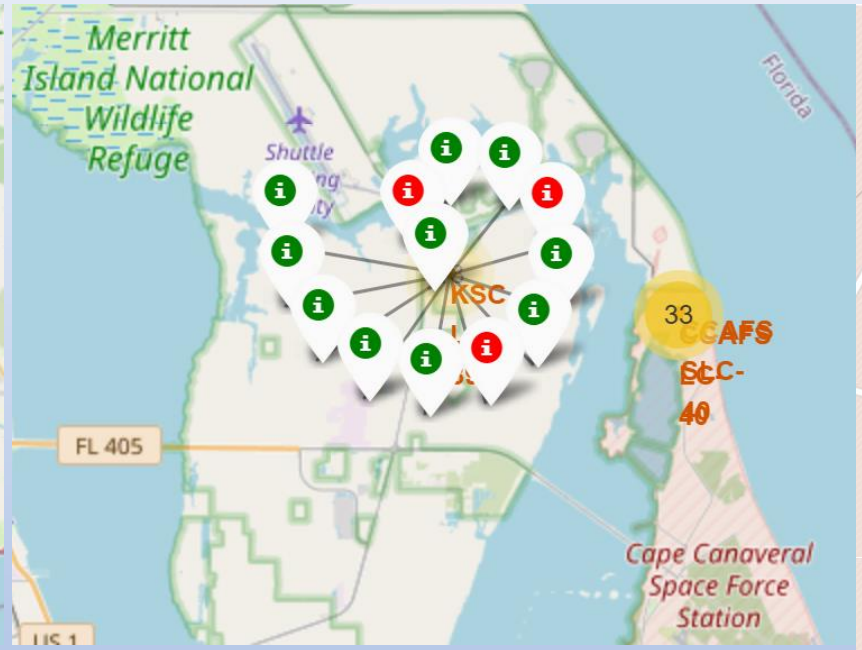


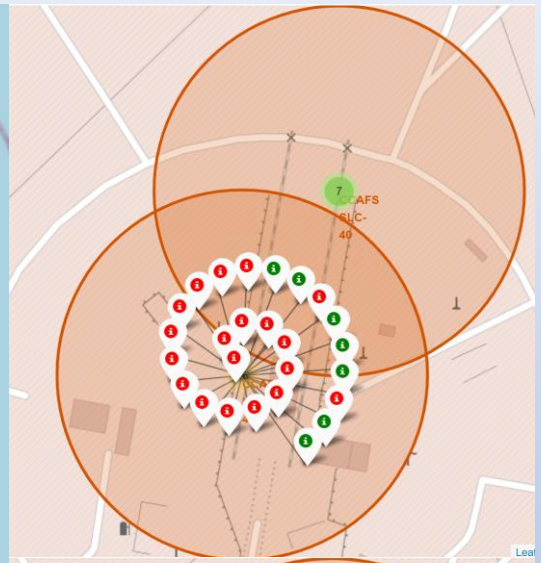
There are 4 total SpaceX Launch Sites:

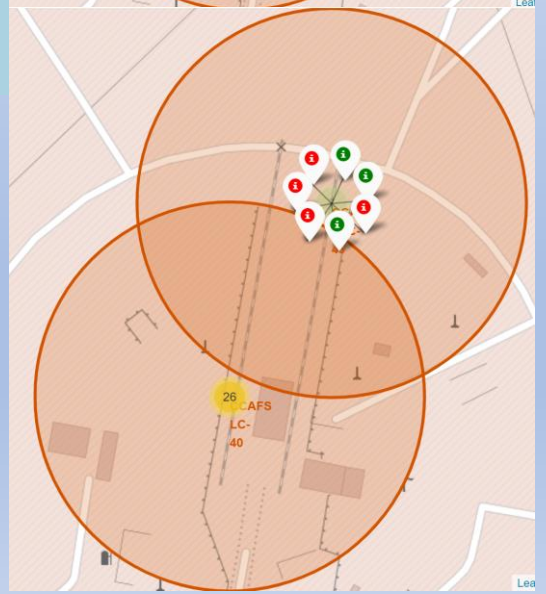
- 1 is Located on the West Coast in California
- 3 are located on the East Coast in Florida

Success or Failed Launches For Each Site on A Map







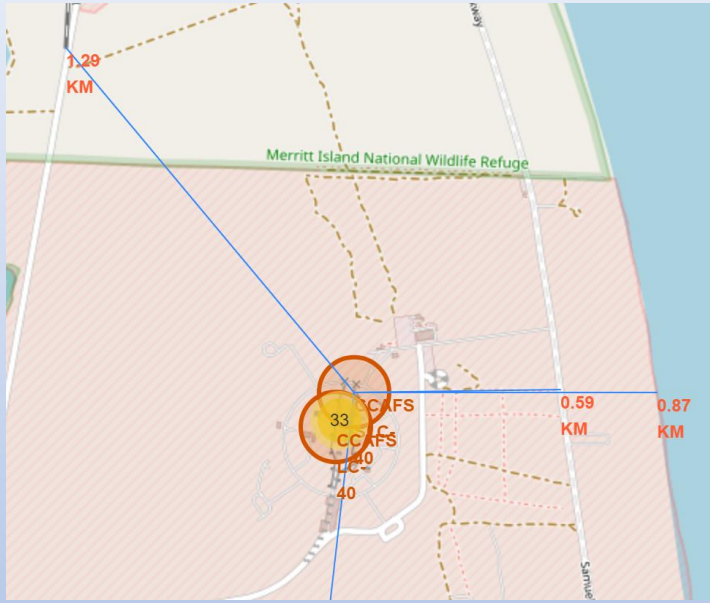


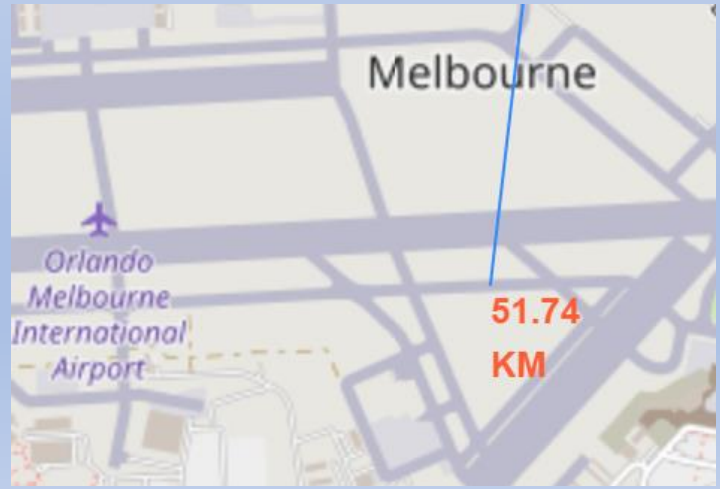
Launches have been grouped into clusters, and annotated with green icons for successful launches, and red icons for failed launches.

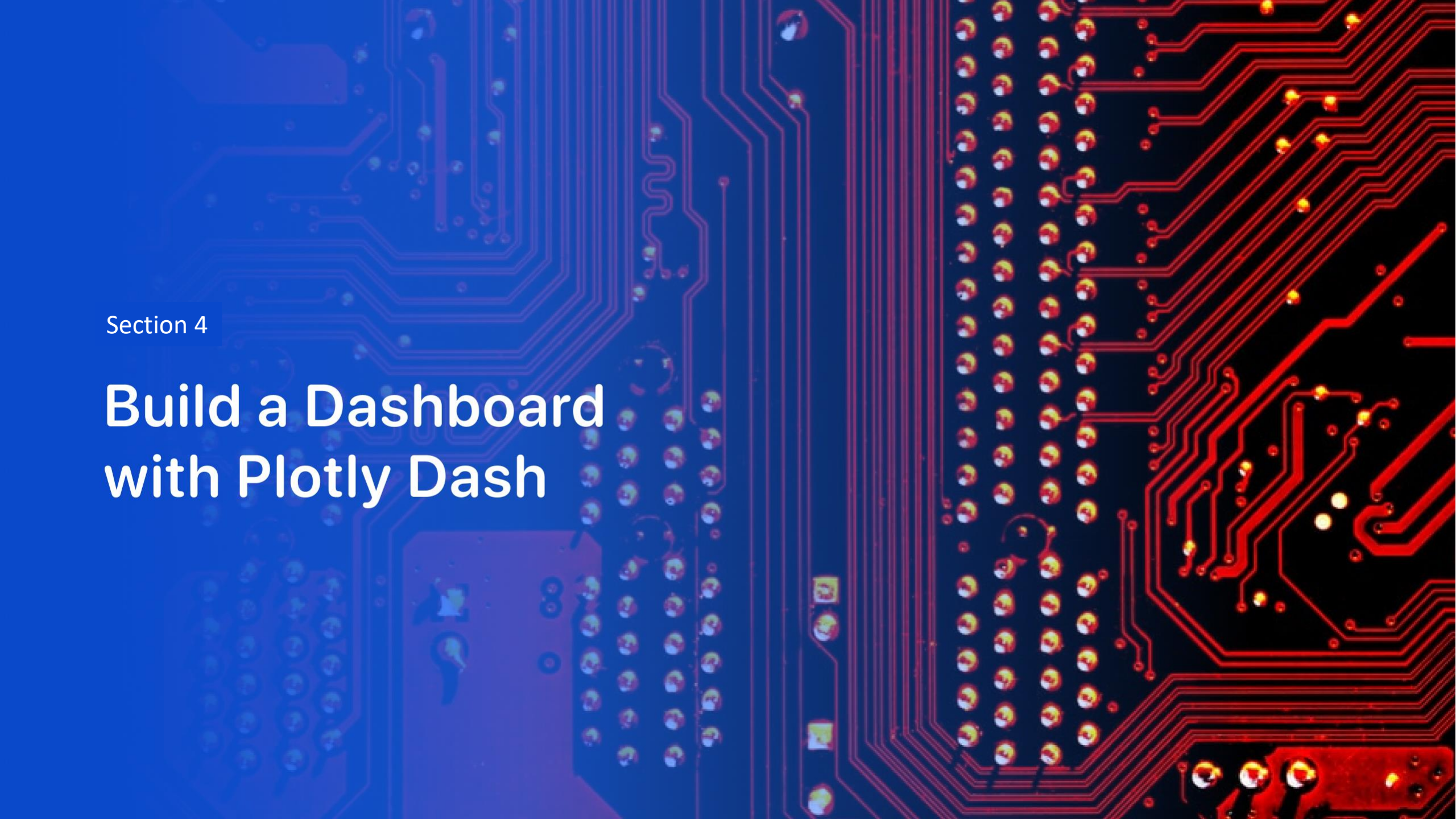
Distances Between A Launch Site To Its Proximities

Using the CCAFS SLC-40 launch site as an example site, we can answer the following questions easily:

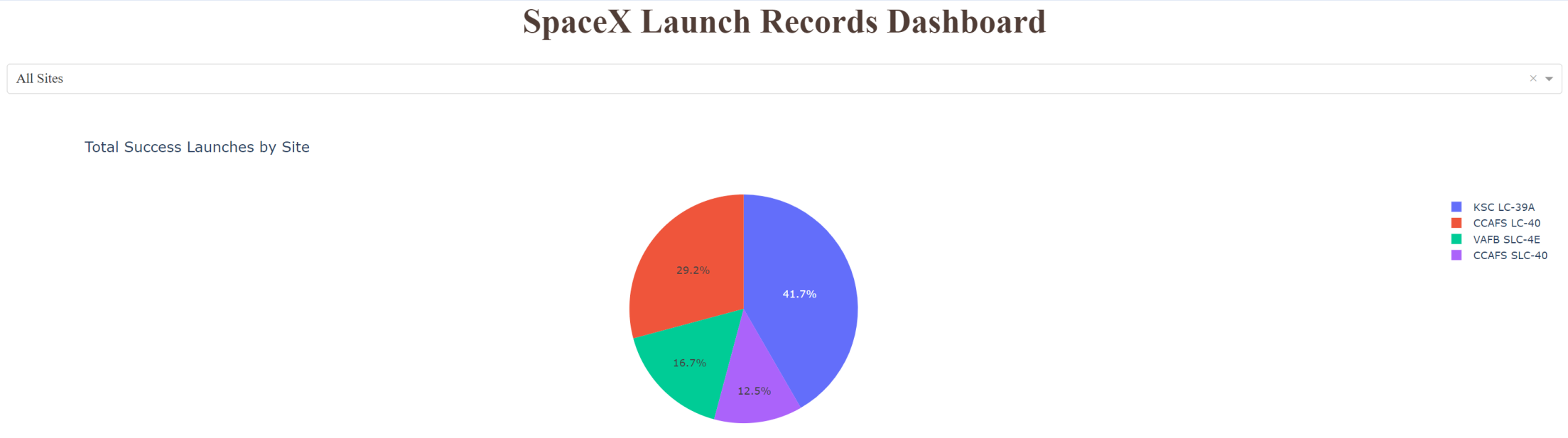
- Are launch sites in close proximity to railways?
  - YES. The nearest railway is only 1.29 km away.
- Are launch sites in close proximity to highways?
  - YES. The nearest highway is only 0.59km away.
- Are launch sites in close proximity to coastline?
  - YES. The coastline is only 0.87 km due East
- Do launch sites keep certain distance away from cities?
  - YES. The nearest city is 51.74 km away.





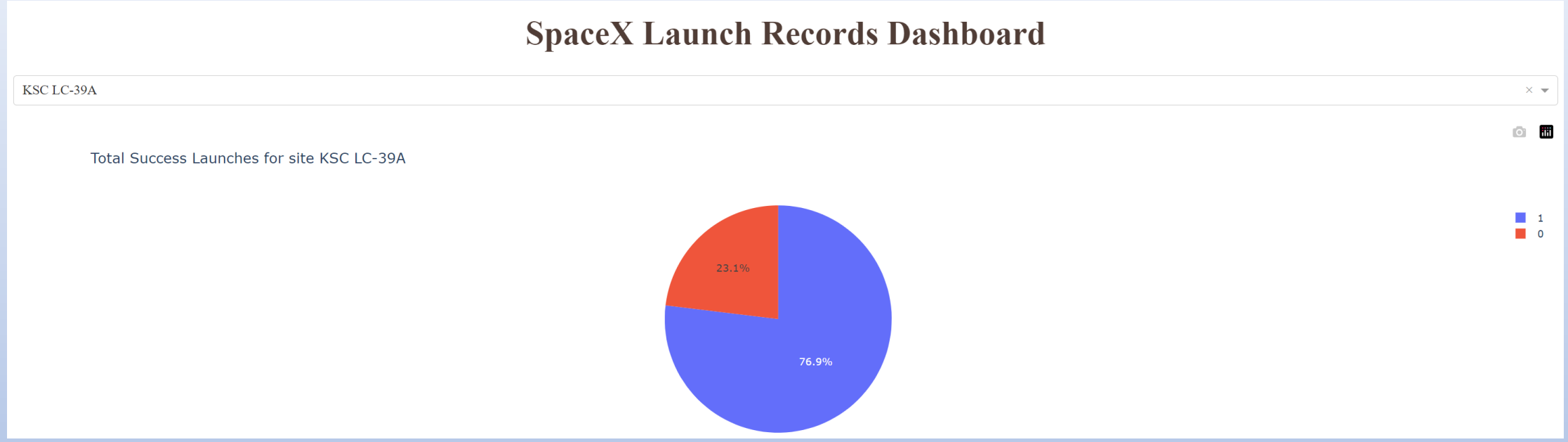

Section 4
Build a Dashboard with Plotly Dash

Launch Success Count For All Sites



- The launch site KSC LC-39 A had the most successful launches, with 41.7% of the total successful launches.

Launch Site With Highest Launch Success Ratio



- The KSC LC-39A Launch Site also has the highest probability of success per launch
- 76.9% of all launches at the KSC LC-39A Site Land Successfully
- 23.1% of all launches at the KSC LC-39A Site Fail to Land

Payload vs Launch Outcome Scatter Plot For All Sites

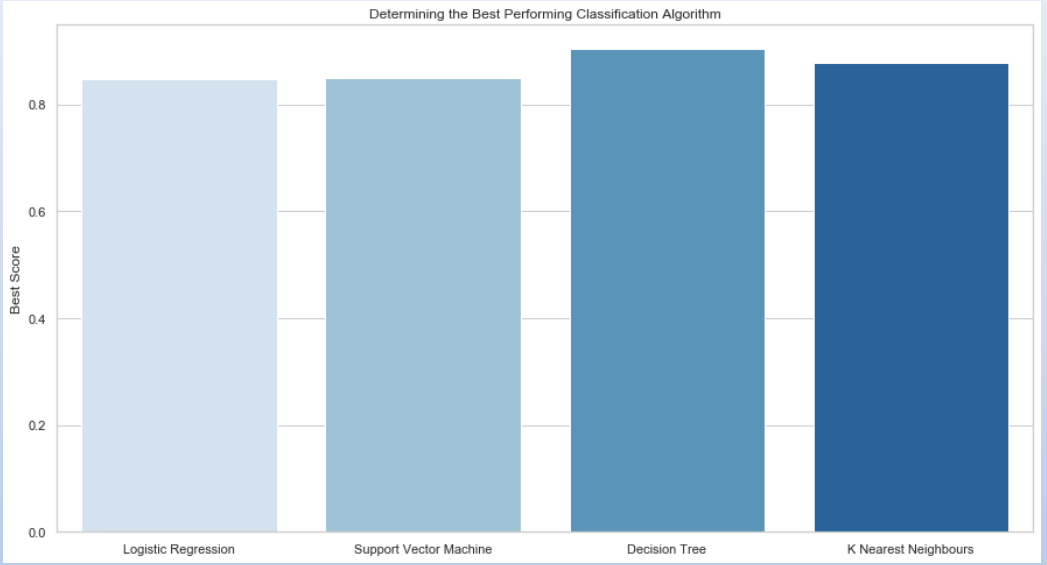
- Split the Plot into 2 ranges:
  - 0 – 5000 kg (low payloads)
  - 5000 – 10000 kg (massive payloads)
- Payloads under 5000kgs have a higher probability of success than payloads over 5000kgs
- Booster Versions FT and B4 are the only boosters to carry payloads over 5000kgs

Section 5

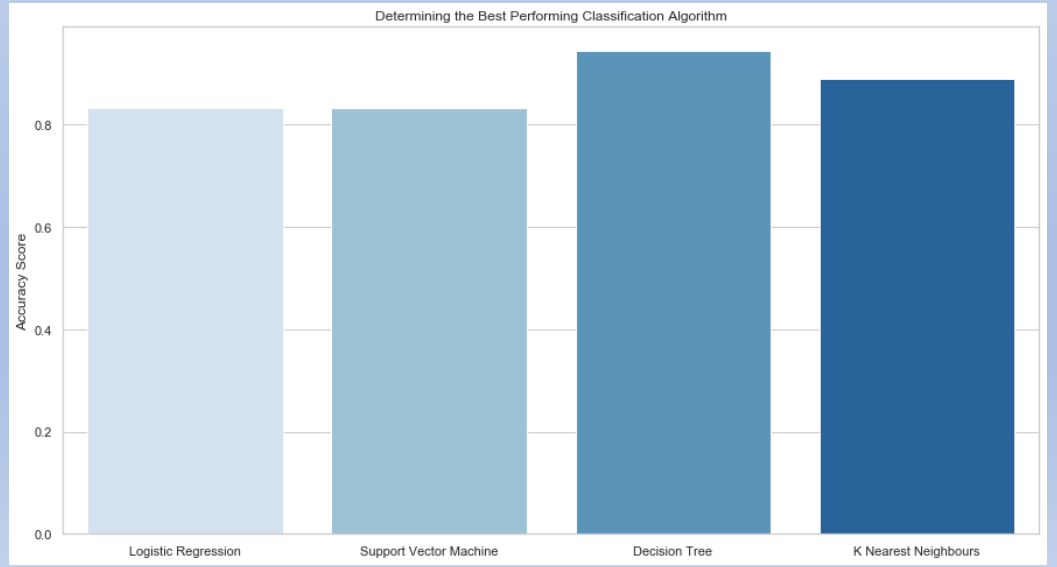
Predictive Analysis (Classification)

Classification Accuracy

	Algorithm	Accuracy Score	Best Score
0	Logistic Regression	0.833333	0.846429
1	Support Vector Machine	0.833333	0.848214
2	Decision Tree	0.944444	0.903571
3	K Nearest Neighbours	0.888889	0.876786

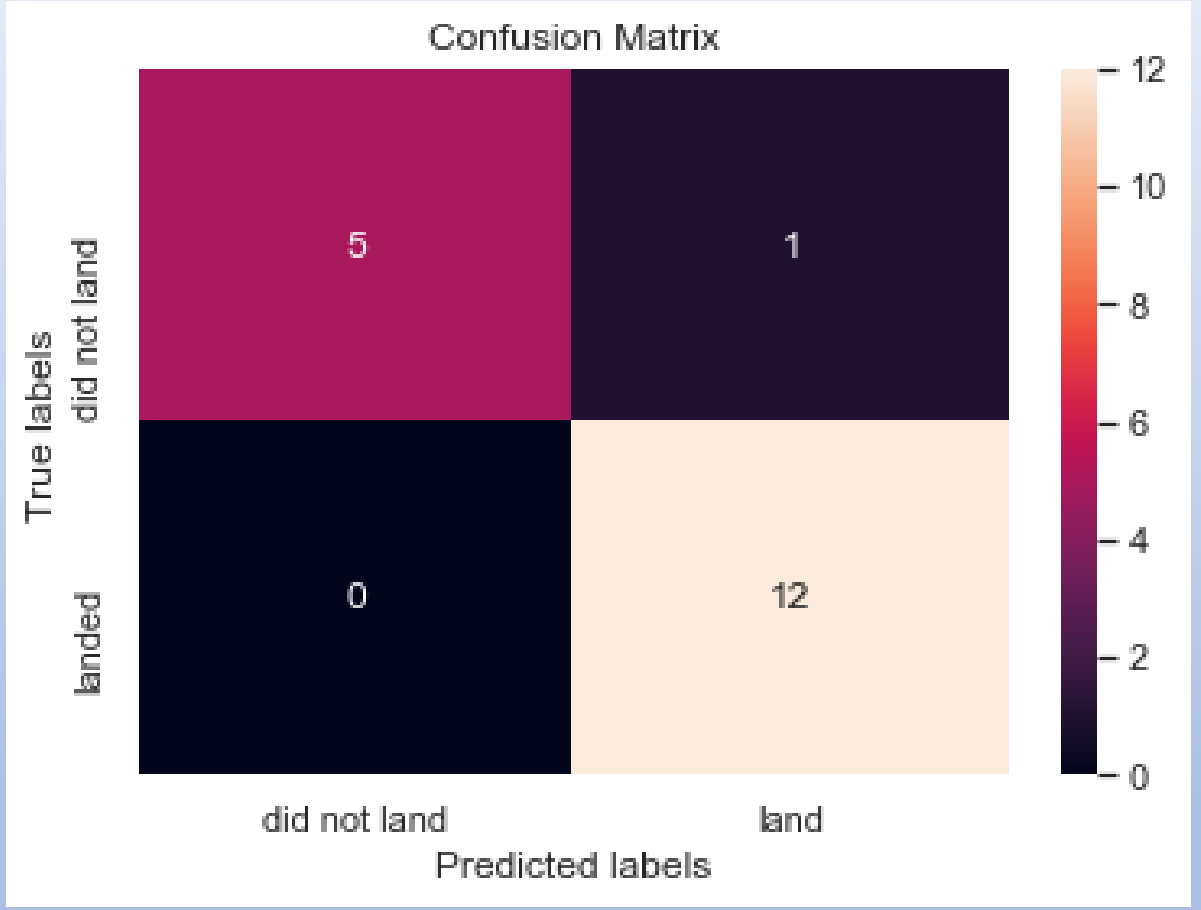


- The Decision Tree model has the highest classification accuracy:
  - The Accuracy Score is 94.44%
  - The Best Score is 90.36%



Confusion Matrix

- The best performing classification model is the Decision Tree model, with an accuracy of 94.44%.
- This is explained by the confusion matrix which correctly predict that 5 of the 6 testing points that fail to land correctly and also correctly predict that all 12 rockets will land.
- Accuracy = (5+12)/(5+1+12+0) = 94.44%



Conclusions

- As the number of flights increases, the rate of success at a launch site increases.
- Orbit types ES-L1, GEO, HEO, and SSO, have the highest (100%) success rate.
- SSO has 100% success rate with 5 successful flights, compare to it, ES-L1, GEO, HEO are with 1 successful flights only.
- The 100% success rate of ES-L1, GEO, HEO, SSO were only with lower payload mass, below 7,000 kg.
- Low weight payloads are more likely to land successfully than heavy payloads
- Early flights tends to have lower success rate compare to later flights.
- After 2013, the success rate generally increased, despite small dips in 2018 and 2020.
- The launch site KSC LC-39 A had the most successful launches, with 41.7% of the total successful launches, and also the highest rate of successful launches, with a 76.9% success rate.
- The best performing classification model is the Decision Tree model, with an accuracy of 94.44%.

Appendix

- Full Code Snippet for slide 8, Number 3 (extract table data to dictionary)

```
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 No.: ',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)
            print('Date: ', date)

            # Time value
            # TODO: Append the time into launch_dict with key `Time`
            time = datatimelist[1]
            launch_dict['Time'].append(time)
            #print(time)
            print('Time: ', 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
            launch_dict['Version Booster'].append(bv)
            print('Version Booster: ', 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)
            print('Launch site: ', 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)
            print('Payload: ', 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)
print('Payload mass: ', payload_mass)

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

# Customer
# TODO: Append the customer into launch_dict with key `Customer`
if row[6].a != None:
    customer = row[6].a.string
else:
    customer = row[6].text.strip()
launch_dict['Customer'].append(customer)
#print(customer)
print('Customer: ', 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)
print('Launch outcome: ', 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)
print('Booster landing: ', booster_landing)
```

Thank you!