IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Sai Charan Mekala
July 2023

Outline

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

Executive Summary

- Summary of methodologies

  This project follows these steps:

  - Data Collection
  - Data Wrangling
  - Exploratory Data Analysis
  - Interactive Visual Analytics
  - Predictive Analysis (Classification)

- Summary of all results

  This project produced the following outputs and visualizations:

  Exploratory Data Analysis (EDA) results

  Geospatial analytics

  Interactive dashboard

  Predictive analysis of classification models

Introduction

- Falcon 9 rocket launches by SpaceX cost about $62 million. This is significantly less expensive than other providers (which typically cost more than $165 million), and a large portion of the savings are attributable to SpaceX's ability to land and reuse the rocket's first stage.
- We can calculate the cost of a launch if we can determine whether the first stage will land or not.
- This project is focused on building a machine learning model to predict if the SpaceX Falcon 9 first stage will land successfully.

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - Using SpaceX REST API
  - Web Scraping
- Perform data wrangling
  - Using .value_count() method to determine the following:
    - Number of launches per each Launch sites
    - Number and occurrence of each Orbit
    - Number and occurrence of mission outcome per orbit type
  - Create a 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
  - Using Scikit-Learn to:
    - Pre-process (standardize) the data
    - Split the data into training and testing data using train_test_split
    - Train different classification models
    - Find hyperparameters using GridSearchCV
  - Plotting confusion matrices for each classification model
  - Assessing the accuracy of each classification model

Data Collection – SpaceX API

Using the SpaceX API to retrieve data about launches, including information about the rocket used, payload delivered, launch specifications, landing specifications, and landing outcome.

1. Retrieve data using get request from SpaceX API call and convert the response to pandas dataframe
2. Create a dictionary of required data
   - Clean the data and retrieve useful information (see Appendix)
   - Define a list and construct the dictionary using this list
3. Create Pandas dataframe using the dictionary
4. Filter the dataframe to only Falcon 9 launches
5. Dealing with missing values

1
```
spacex_url="https://api.spacexdata.com/v4/launches/past"

response = requests.get(spacex_url)

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

2
```
#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}
```

3
```
# Create a data from launch_dict
data = pd.DataFrame(launch_dict)
```

4
```
data_falcon9 = data[data['BoosterVersion'] != 'Falcon 1']
# Now that we have removed some values we should reset the FlgihtNumber column
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
```

5
```
# 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(np.nan, mean)
```

GitHub Link

Data Collection - Scraping

Using web scraping, historical Falcon 9 launch data was gathered from the List of Falcon 9 and Falcon Heavy launches page on Wikipedia.

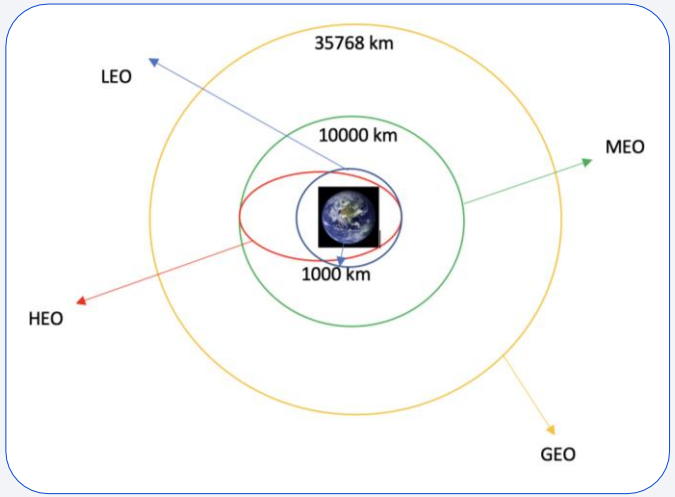
1. Using get request from url to parse the html page
2. Create BeautifulSoup object and extract all the tables
3. Use a custom method (see Appendix) to get all the column header names from the tables on the HTML page.
4. Create a dictionary using the column names as keys and fill the dictionary values (see Appendix)
5. Create the dataframe using the extracted launch_dict

1
```
static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"
response = requests.get(static_url)
```
2
```
soup = BeautifulSoup(response.text, 'html.parser')

    # Use soup.title attribute
    soup.title
```
MagicPython
```
<title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```
```
html_tables = soup.find_all('table')
```
3
```
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
table_headers = first_launch_table.find_all('th')

for row in table_headers :
    name = extract_column_from_header(row)
    if name is not None and len(name) > 0:
        column_names.append(name)
```
4
```
launch_dict = dict.fromkeys(column_names)
```
5
```
df=pd.DataFrame(launch_dict)
```


GitHub Link

Data Wrangling

- The LaunchSite column lists each of the several Space X launch facilities that are included in the SpaceX dataset.
- Each launch is intended for a certain orbit, some of which are seen in the image below. The Orbit column contains the orbit type.




```
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
```

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

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

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

```
Orbit
GTO      27
ISS      21
VLEO     14
PO        9
LEO       7
SSO       5
MEO       3
ES-L1     1
GEO       1
HEO       1
SO        1
dtype: int64
```

Initial Data Exploration:

Using the .value_counts() method to determine the following:

1. Number of launches on each site
2. Number and occurrence of each orbit
3. Number and occurrence of landing outcome per orbit type

GitHub Link

Data Wrangling

Context:

- The landing outcome is shown in the Outcome column:
  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 ASDS – the mission outcome was successfully landed to a drone ship
  6. False ASDS – the mission outcome was unsuccessfully landed to a drone ship.
  7. None ASDS and None None – these represent a failure to land.

Data Wrangling:

- It is more desirable to have a binary column, where the value is 1 or 0, signifying the success of the landing, to decide whether a booster will land successfully.
- This is done by:
  1. Defining a set of unsuccessful (bad) outcomes, bad_outcome
  2. Creating a list, landing_class, where the element is 0 if the corresponding row in Outcome is in the set bad_outcome, otherwise, it's 1.
  3. Create a Class column that contains the values from the list landing_class
  4. Export the DataFrame as a .csv file.

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

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

```
landing_class = df['Outcome'].apply(lambda x: 0 if x in bad_outcomes else 1)

landing_class.value_counts()
```

```
1    60
0    30
Name: Outcome, dtype: int64
```

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

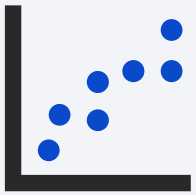
```
df.to_csv("dataset_part_2.csv", index=False)
```

GitHub Link

EDA with Data Visualization

SCATTER PLOTS

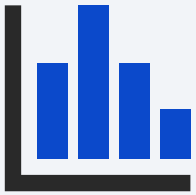
- Scatter plots were produced to visualize the relationships between:
  - Flight Number and Launch Site
  - Payload and Launch Site
  - Orbit Type and Flight Number
  - Payload and Orbit Type



Scatter plots are useful to observe relationships, or correlations, between two numeric variables.

BAR PLOTS

- A bar plot was produced to visualize the relationship between:
  - Success Rate and Orbit Type



Bar plots are used to compare a numerical value to a categorical variable. Horizontal or vertical bar charts can be used, depending on the size of the data.

LINE PLOTS

- Line plots were produced to visualize the relationships between:
  - Success Rate and Year (i.e. the launch success yearly trend)

Line plots contain numerical values on both axes and are generally used to show the change of a variable over time.

GitHub Link

EDA with SQL

- To gather some information about the dataset, some SQL queries were performed.
- The SQL queries performed on the data set were used to:

1. Display the names of the unique launch sites in the space mission
2. Display 5 records where launch sites begin with the string ‘CCA’
3. Display the total payload mass carried by boosters launched by NASA (CRS)
4. Display the average payload mass carried by booster version F9 v1.1
5. List the date when the first successful landing outcome on a ground pad was achieved
6. List the names of the boosters which had success on a drone ship and a payload mass between 4000 and 6000 kg
7. List the total number of successful and failed mission outcomes
8. List the names of the booster versions which have carried the maximum payload mass
9. List the failed landing outcomes on drone ships, their booster versions, and launch site names for 2015
10. 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

In order to display the launch data on an interactive map, the following steps were taken:

1. Mark all launch sites on a map
   - Initialise the map using a Folium Map object
   - Add a folium.Circle and folium.Marker on the launch map for each launch site

2. Mark the success/failed launches for each site on a map
   - It makes sense to group launches together because they all have the same coordinates.
   - Assign a marker color of green for successful (class = 1) and red for unsuccessful (class = 0) before clustering them.
   - Add a folium.Marker to the MarkerCluster() object for each launch in order to group the launches into clusters
   - Create an icon as a text label and set the icon_color to the previously chosen marker_color.

3. Calculate the distances between a launch site to its proximities
   - The Lat and Long can be used to calculate the distances between places in order to investigate the proximity of launch sites.
   - Create a folium.Marker object to show the distance.
   - Draw a folium.PolyLine in order to display the distance line between two points and add this to the map.

GitHub Link

Build a Dashboard with Plotly Dash

To create an interactive data visualization, the following plots were added to a Plotly Dash dashboard:

1. The total successful launches per site – Pie Chart
   - Create a pie chart using px.pie()
   - This makes it easy to determine which sites have the most successful launches.
   - In order to view the success/failure ratio for a certain site, the chart could also be filtered (using a dcc.Dropdown() object).

2. Correlation between outcome (success or not) and payload mass (kg) – Scatter Plot
   - Create a pie chart using px.scatter()
   - A RangeSlider() object was defined to filter with respect to range of payload masses.
   - It can also be filtered according to the booster version.

GitHub Link

Predictive Analysis (Classification)

Model Development

Prepare the data

- Pre-process and standardise
- Split the data using train_test_split()

Choose a model and train the model

- Create a GridSearchCV object with the dictionary of parameters
- Train the model using the training dataset

Model Evaluation

Obtain tuned hyper parameters(best_params)

Obtain the accuracy of the model (score and best_score)

Plot and examine the Confusion Matrix

Best Model

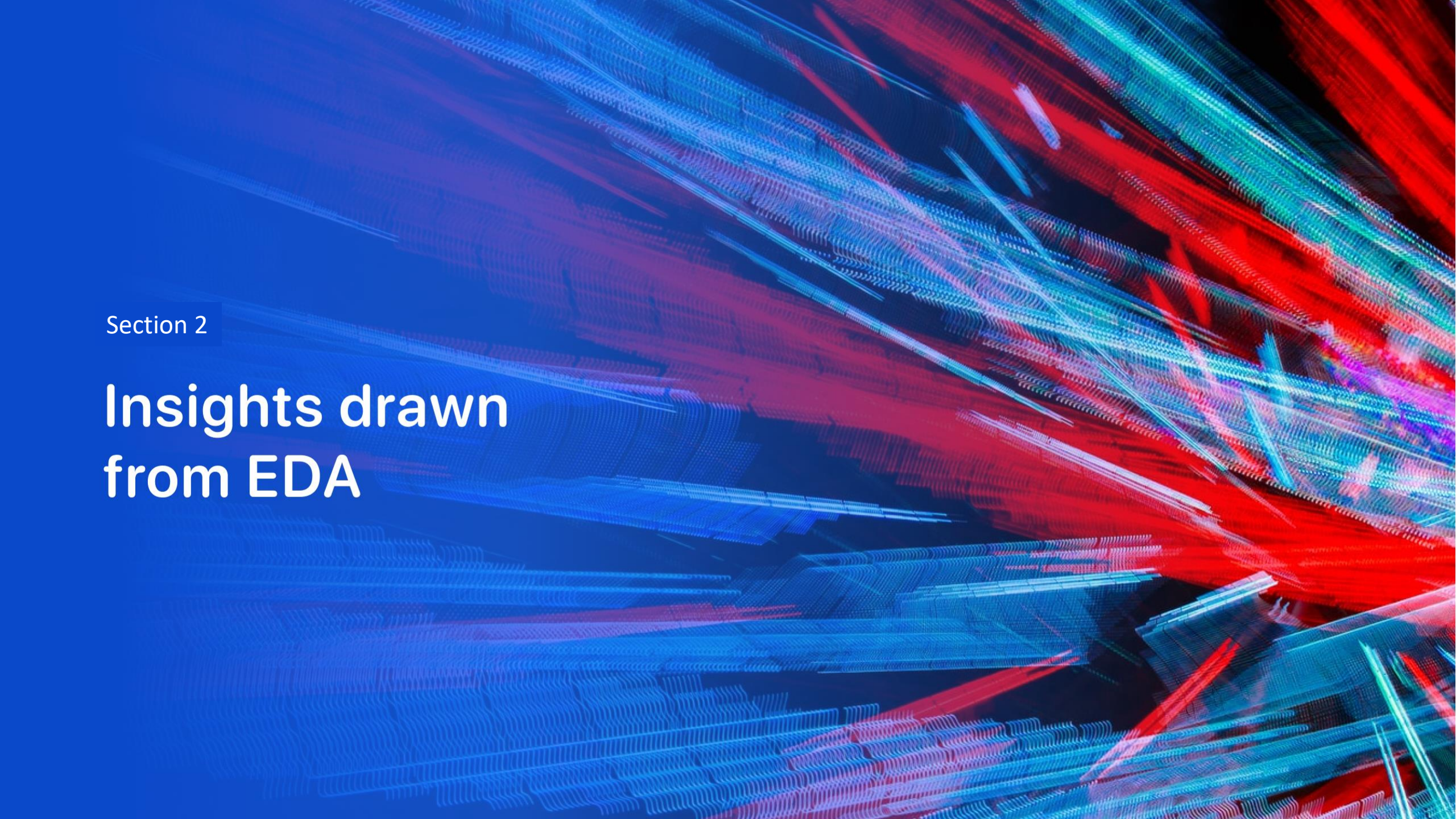
Review the accuracy on testing data and training data for each model

The model with highest accuracy is chosen as the best model

GitHub Link

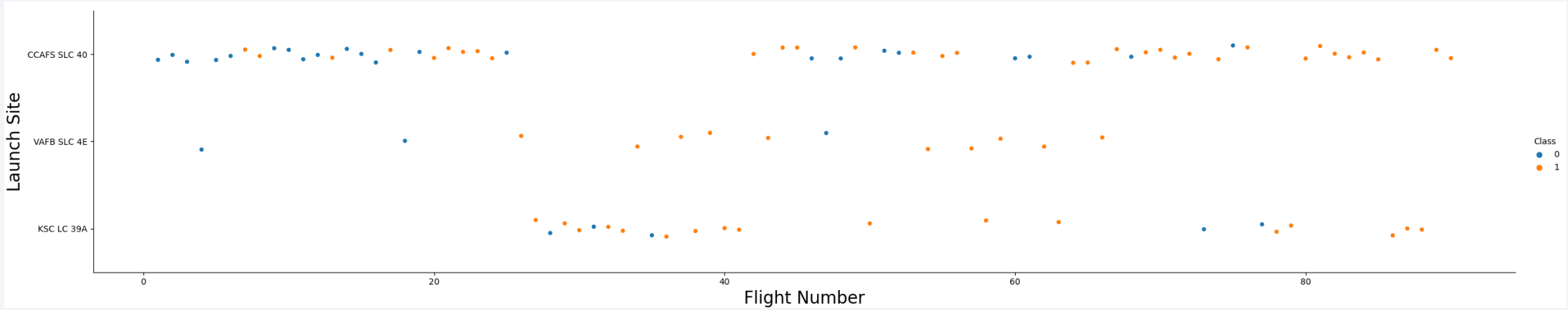
Results

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


Section 2
Insights drawn from EDA

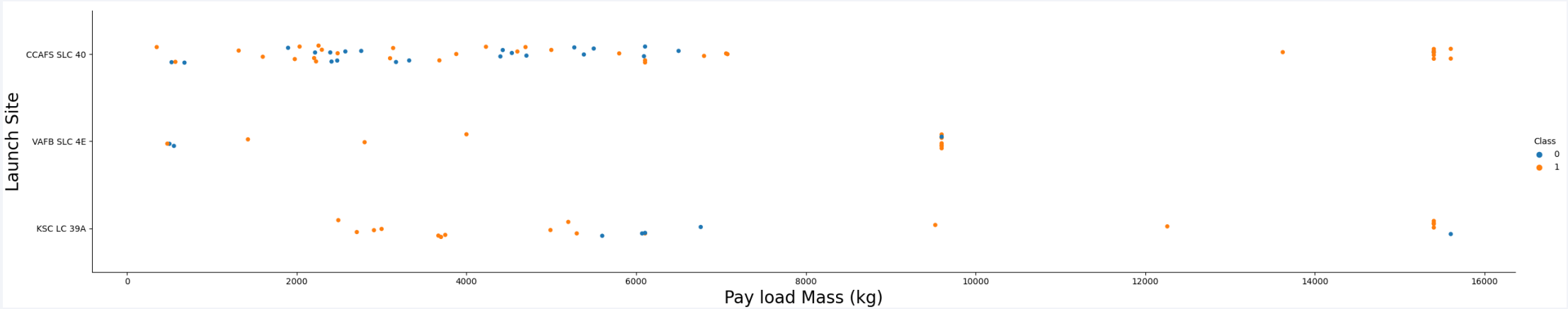
Flight Number vs. Launch Site

- Scatter plot of Flight Number vs. Launch Site shows that
  - The success rate at a launch site rises as the number of flights rises.
  - Launched from CCAFS SLC 40, the majority of the early flights (flight numbers 30) were mostly unsuccessful.
  - This pattern, that early flights were less successful, is also demonstrated by the flights from VAFB SLC 4E.
  - The launches from KSC LC 39A are more successful because no early flights were launched from there.
  - There are significantly more successful landings (Class = 1) above a flight number of about 30.



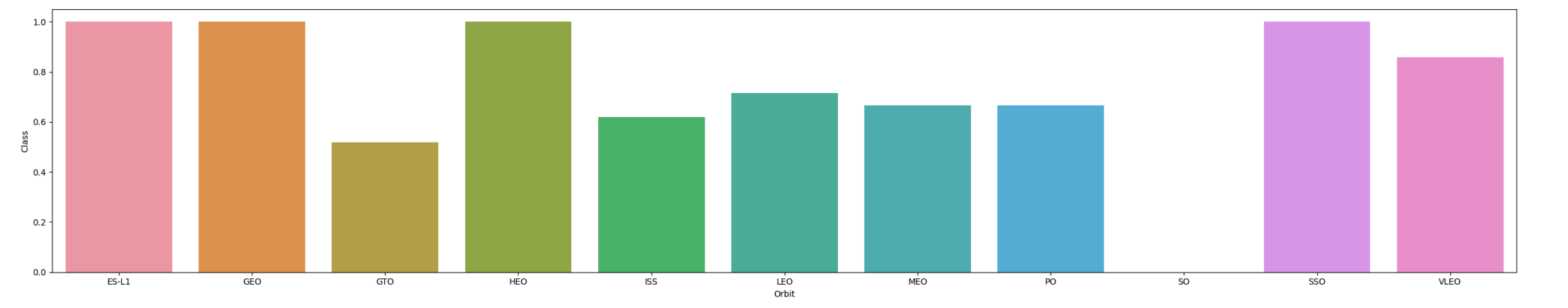
Payload vs. Launch Site

- Scatter plot of Payload vs. Launch Site shows that
  - There are extremely few failure landings over a payload mass of about 7000 kg, but there is also much less data for these heavier launches.
  - For a specific launch site, there is no obvious relationship between payload mass and success rate.
  - All sites launched a range of payload masses, with the majority of CCAFS SLC 40's launches (apart from a few outliers) having somewhat lighter payloads.



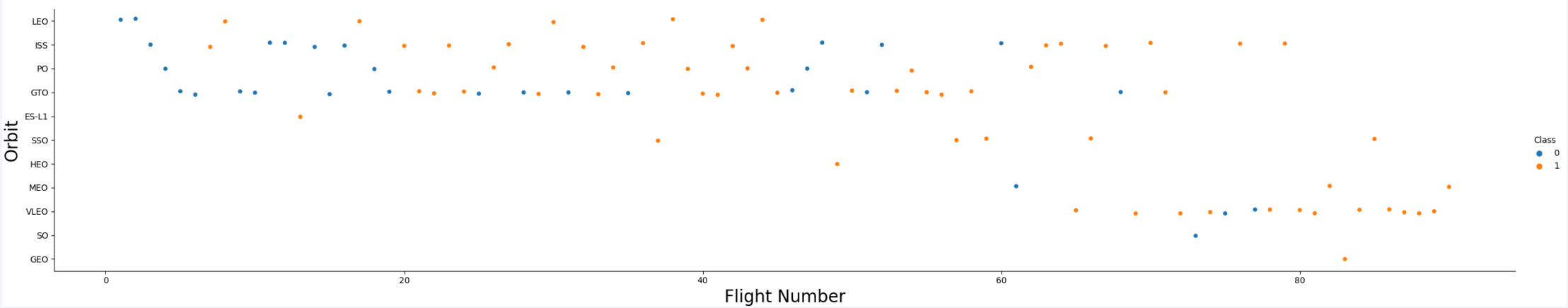
Success Rate vs. Orbit Type

- Bar chart for the success rate of each 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 0% success rate



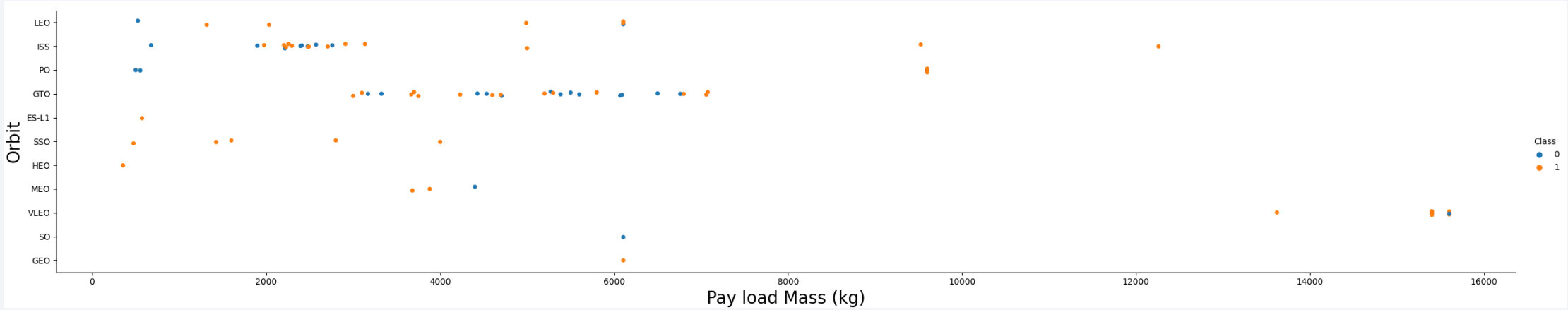
Flight Number vs. Orbit Type

- Scatter plot of Flight Number vs. Orbit Type shows that
  - One journey into each of the three orbits — GEO, HEO, and ES-L1 can account for their 100% success rates.
  - With 5 successful flights, SSO's 100% success rate is even more remarkable.
  - The Success Rate for GTO and Flight Number have little in common.
  - In general, the success rate rises as Flight Number does. Extreme cases of this include LEO, where only a small percentage of flights (early launches) resulted in unsuccessful landings.



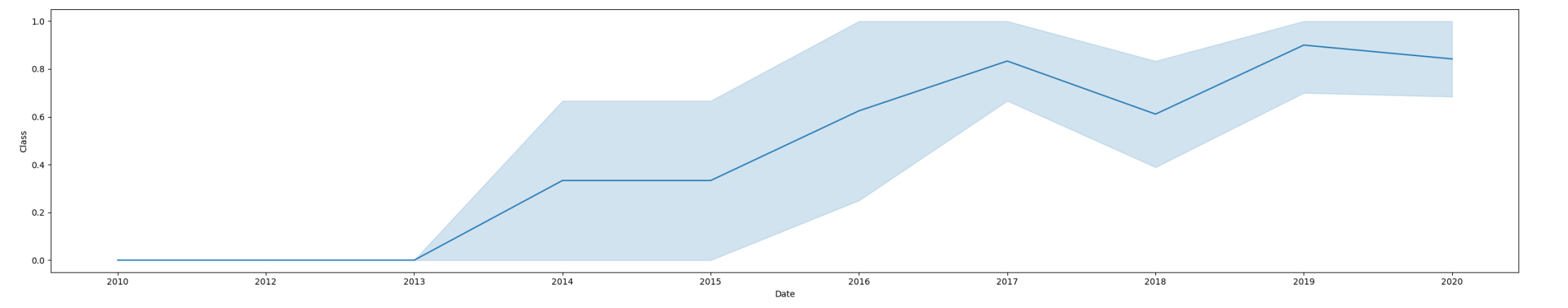
Payload vs. Orbit Type

- Scatter plot of Payload vs. Orbit Type shows that
  - With big payloads, the following orbital types perform better:
    - PO (even though there are few data points)
    - ISS
    - LEO
  - It is unknown for GTO what the link is between payload mass and success rate.
  - It makes sense that heavier payloads are sent into VLEO (Very Low Earth Orbit).



Launch Success Yearly Trend

- According to the line graph of the annual average success rate, from 2010 to 2013, there were no successful landings (since the success rate was 0).
- After 2013, despite slight declines in 2018 and 2020, the success rate typically rose.
- After 2016, the likelihood of success was always higher than 50%.



All Launch Site Names

- Find the names of the unique launch sites

```
%sql SELECT distinct(Launch_Site) FROM SPACEXTBL
```

MagicPython

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

Launch_Site
CCAFS LC-40
VAFB SLC-4E
KSC LC-39A
CCAFS SLC-40
None

The word UNIQUE returns only unique values from the LAUNCH_SITE column of the SPACEXTBL table.

Launch Site Names Begin with 'CCA'

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

```
%sql SELECT * FROM SPACEXTBL WHERE Launch_Site LIKE 'CCA%' LIMIT 5
```

MagicPython

* sqlite:///my_data1.db
Done.

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	C
06/04/2010	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0.0	LEO	
12/08/2010	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0.0	LEO (ISS)	
22/05/2012	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525.0	LEO (ISS)	
10/08/2012	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500.0	LEO (ISS)	
03/01/2013	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677.0	LEO (ISS)	

LIMIT 5 restricts the results to 5, LIKE keyword with wildcard 'CCA%' will find the results Launch_Site starting with CCA.

Total Payload Mass

- Calculate the total payload carried by boosters from NASA

```
%sql SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE Customer = 'NASA (CRS)'
```

MagicPython

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

SUM(PAYLOAD_MASS__KG_)
45596.0

The LAUNCH column is totaled using the SUM keyword, and the SUM keyword (and the corresponding condition) limits the results to boosters from NASA (CRS).

Average Payload Mass by F9 v1.1

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

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

MagicPython

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

avg(PAYLOAD_MASS__KG_)
2928.4

The PAYLOAD_MASS__KG_ column is averaged using the AVG keyword, and the WHERE keyword (and the corresponding condition) limits the results to only the F9 v1.1 booster version.

First Successful Ground Landing Date

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

```
%sql SELECT min(Date) FROM SPACEXTBL WHERE Landing_Outcome LIKE 'Success%' or 'Controlled%'
```

MagicPython

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

min(Date)
01/07/2020

The MIN keyword is used to determine the earliest date in the DATE column, and the WHERE keyword (together with its associated condition) narrows the search to only the successful landings at the 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

```
%sql SELECT Booster_Version FROM SPACEXTBL WHERE Landing_Outcome = 'Success (drone ship)' AND PAYLOAD_MASS__KG_ > 4000 and PAYLOAD_MASS__KG_ < 6000
```

MagicPython

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

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

Because the AND keyword is also used, the WHERE keyword is used to narrow the search to only those results that meet both parameters. With the BETWEEN keyword, you can pick from a range of 4000 < x < 6000 possible values.

Total Number of Successful and Failure Mission Outcomes

- Calculate the total number of successful and failure mission outcomes

```
%sql SELECT Mission_Outcome, count(Mission_Outcome) as Total FROM SPACEXTBL \
GROUP BY Mission_Outcome;
```

MagicPython

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

Mission_Outcome	Total
None	0
Failure (in flight)	1
Success	98
Success	1
Success (payload status unclear)	1

You may get a count of all mission results by using the COUNT keyword, and you can sort those results by mission result type using the GROUPBY keyword.

Boosters Carried Maximum Payload

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

```
%sql SELECT DISTINCT Booster_Version FROM SPACEXTBL \
WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTBL);
```

MagicPython

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

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

This makes use of a subquery. The WHERE condition relies on the maximum payload, which is found by the SELECT expression enclosed in brackets. Finally, unique versions of boosters are retrieved using the DISTINCT keyword.

2015 Launch Records

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

```
%sql SELECT substr(Date, 4, 2) as month, Landing_Outcome, Booster_Version, Launch_Site FROM SPACEXTBL \
WHERE substr(Date,7,4)='2015' and Landing_Outcome = 'Failure (drone ship)';
```

MagicPython

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

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

With the WHERE clause, we can restrict the output to only landing attempts that were unsuccessful AND to 2015 as the search year.

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

```
%sql SELECT Landing_Outcome, count(Landing_Outcome) as total_count FROM SPACEXTBL \
WHERE Date BETWEEN "04/06/2010" AND "20/03/2017" \
GROUP BY Landing_Outcome \
Order BY total_count DESC;
```

MagicPython

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

Landing_Outcome	total_count
Success	20
No attempt	9
Success (drone ship)	8
Success (ground pad)	7
Failure (drone ship)	3
Failure	3
Failure (parachute)	2
Controlled (ocean)	2
No attempt	1

When combined with BETWEEN, the WHERE keyword narrows search results to just those that fall within the range of the supplied dates. DESC is used to specify a descending order when using the keywords GROUP BY and ORDER BY to sort the results.

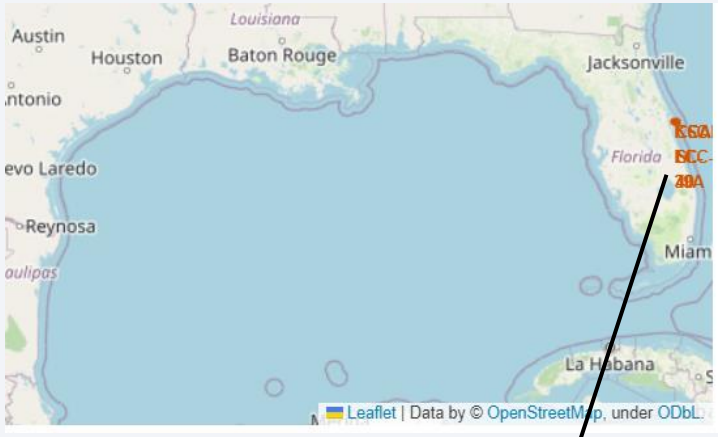
Section 3

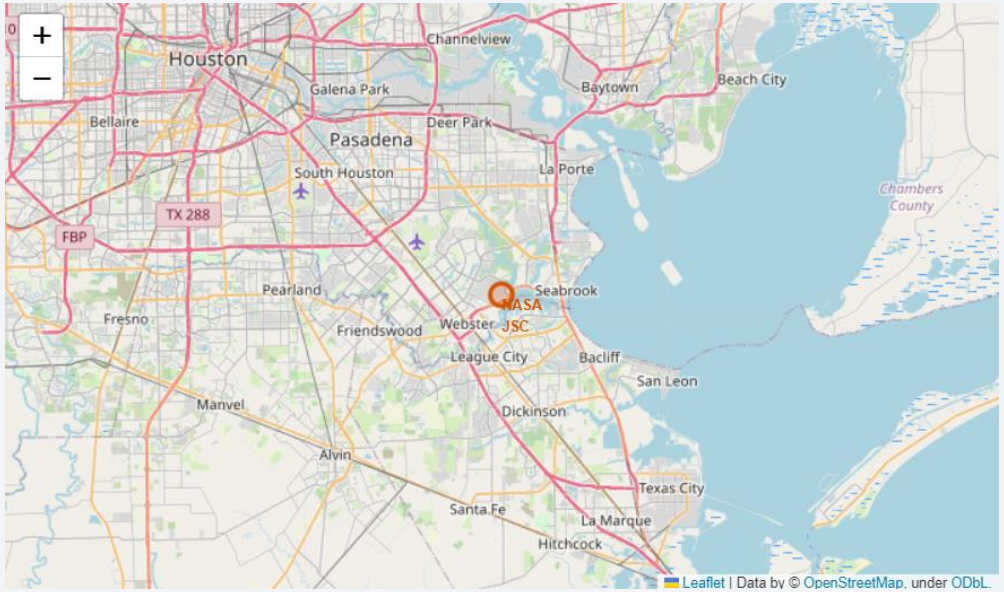
Launch Sites Proximities Analysis

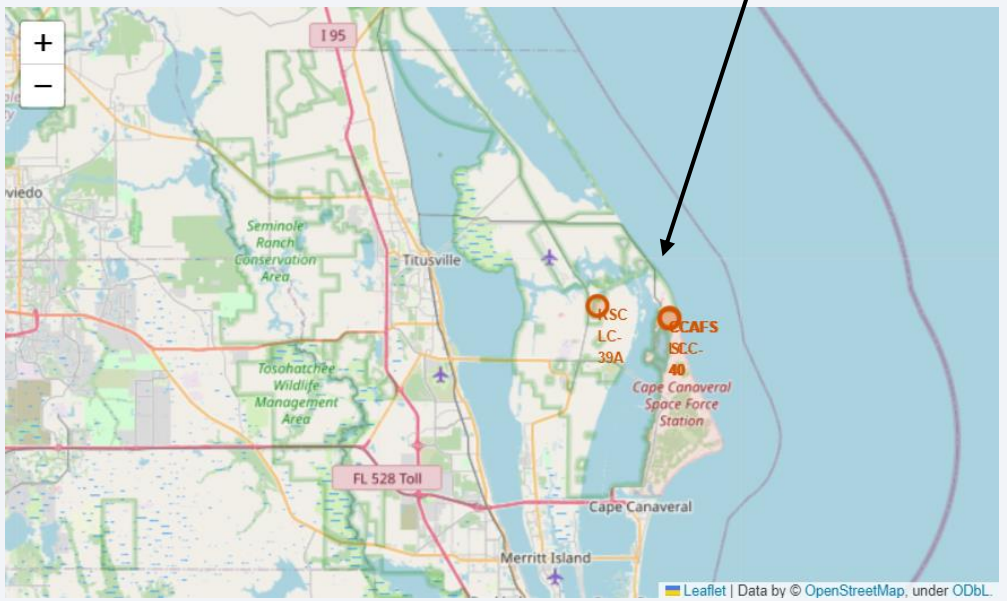
All launch sites on the map

Table of all launch sites

	Launch Site	Lat	Long
0	CCAFS LC-40	28.562302	-80.577356
1	CCAFS SLC-40	28.563197	-80.576820
2	KSC LC-39A	28.573255	-80.646895
3	VAFB SLC-4E	34.632834	-120.610745

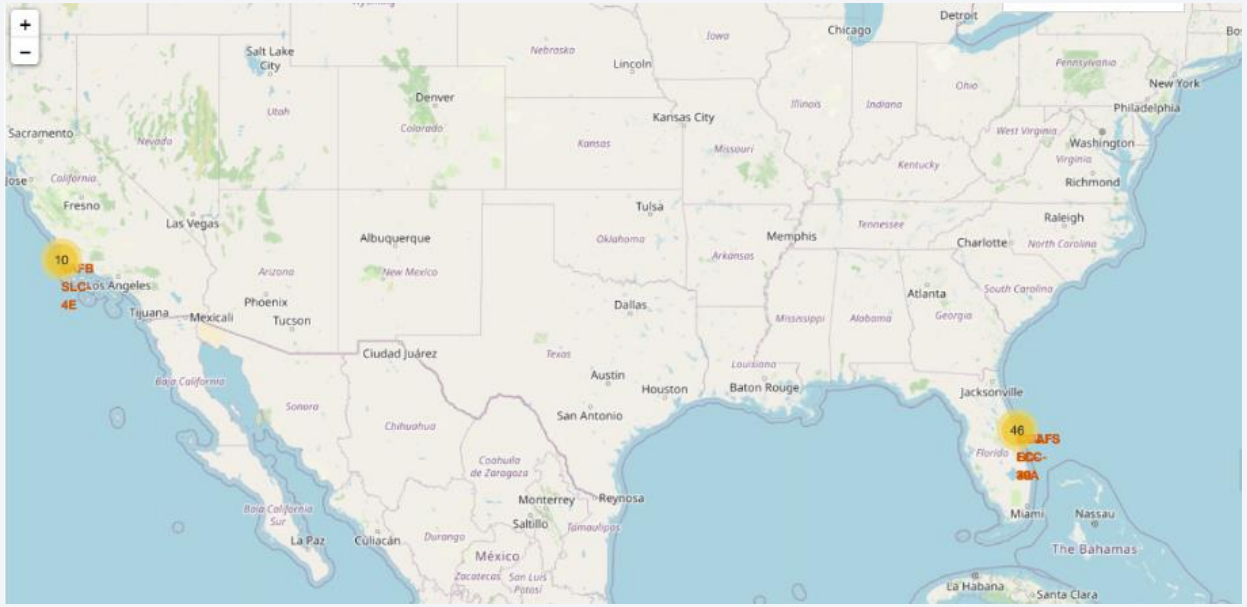


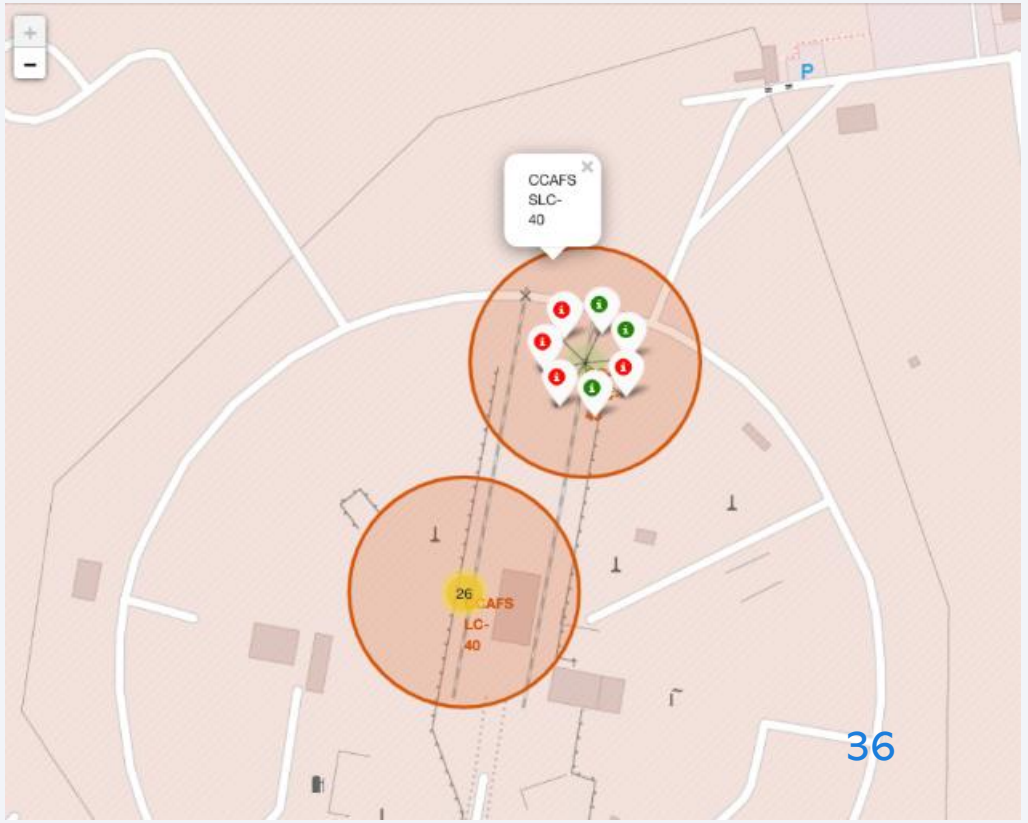




Success/Failed launches for each site

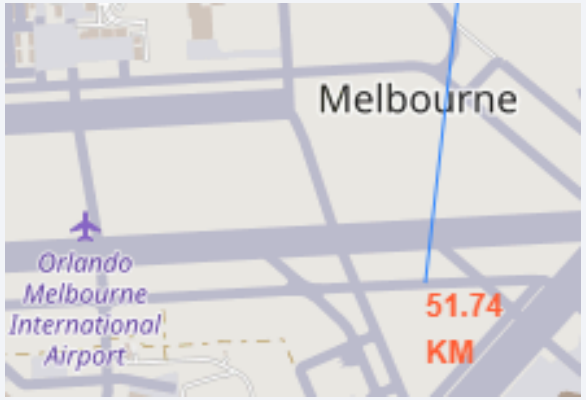
Launches have been grouped into clusters, and annotated with green icons for successful launches, and red icons for failed launches.





<Folium Map Screenshot 3>

Using the CCAFS SLC-40 launch site as an example site, we can understand more about the placement of launch sites.



Are launch sites in close proximity to railways?

- YES. The coastline is only 0.87 km due East.

Are launch sites in close proximity to highways?

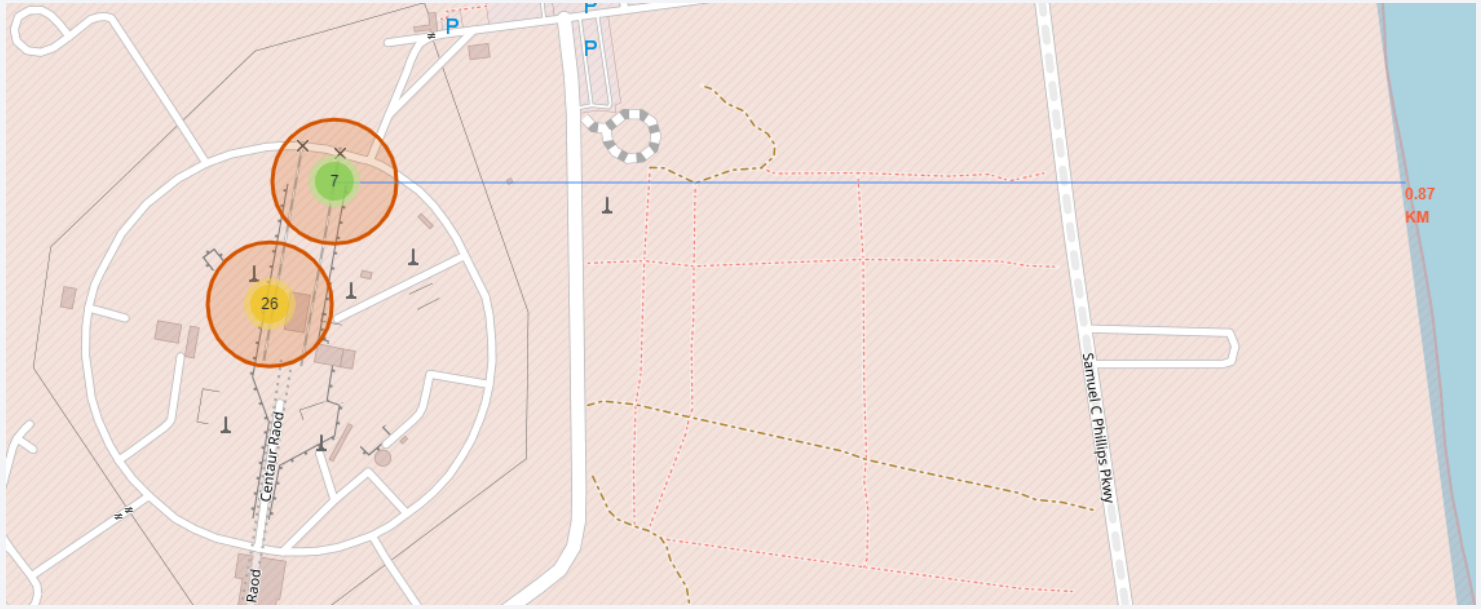
- YES. The nearest highway is only 0.59km away.

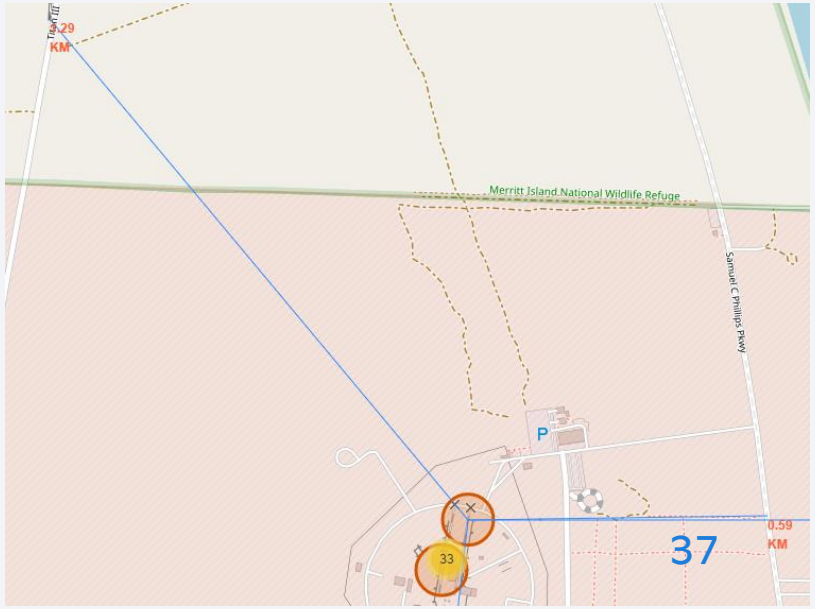
Are launch sites in close proximity to railways?

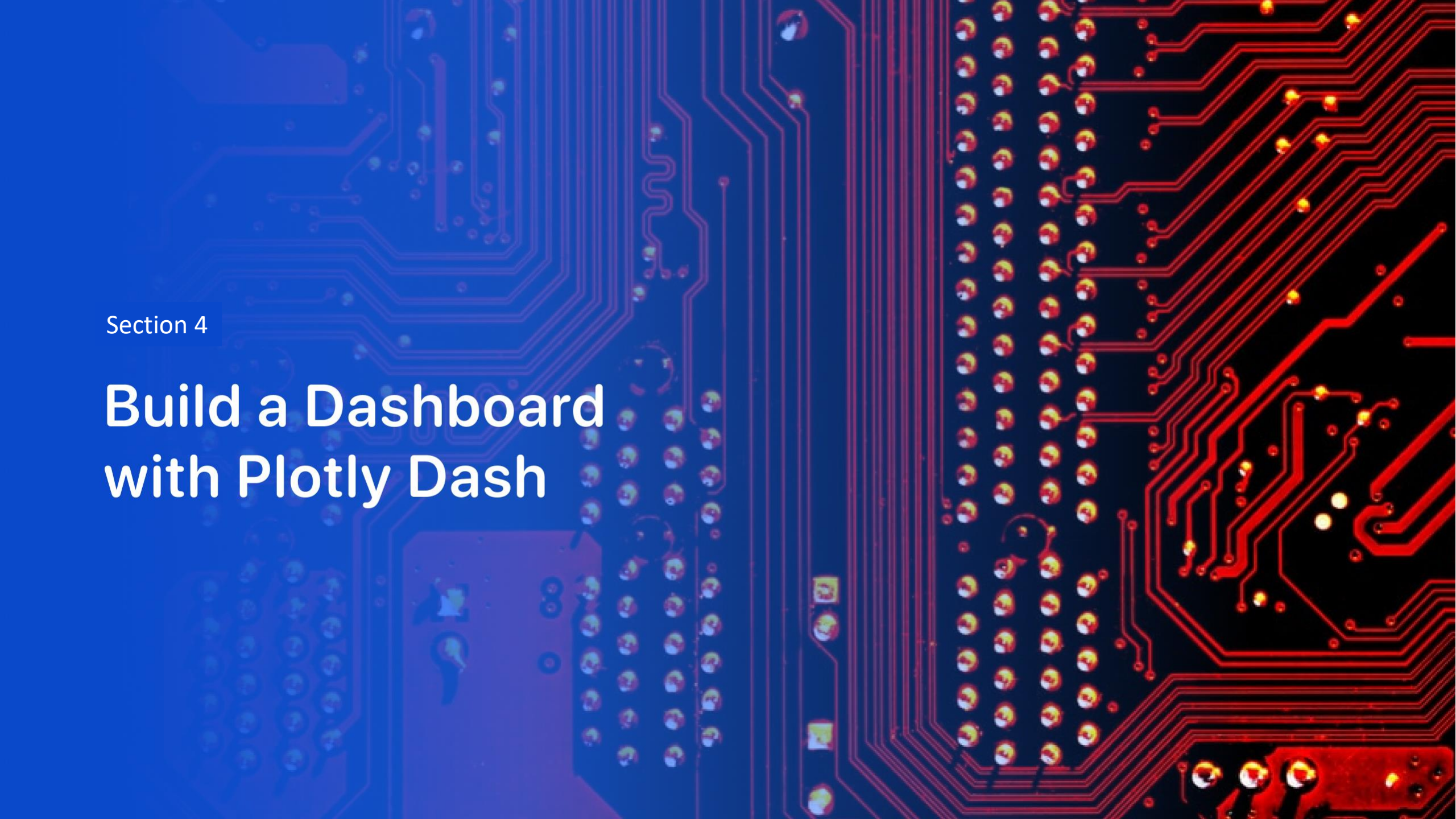
- YES. The nearest railway is only 1.29 km away.

Do launch sites keep certain distance away from cities?

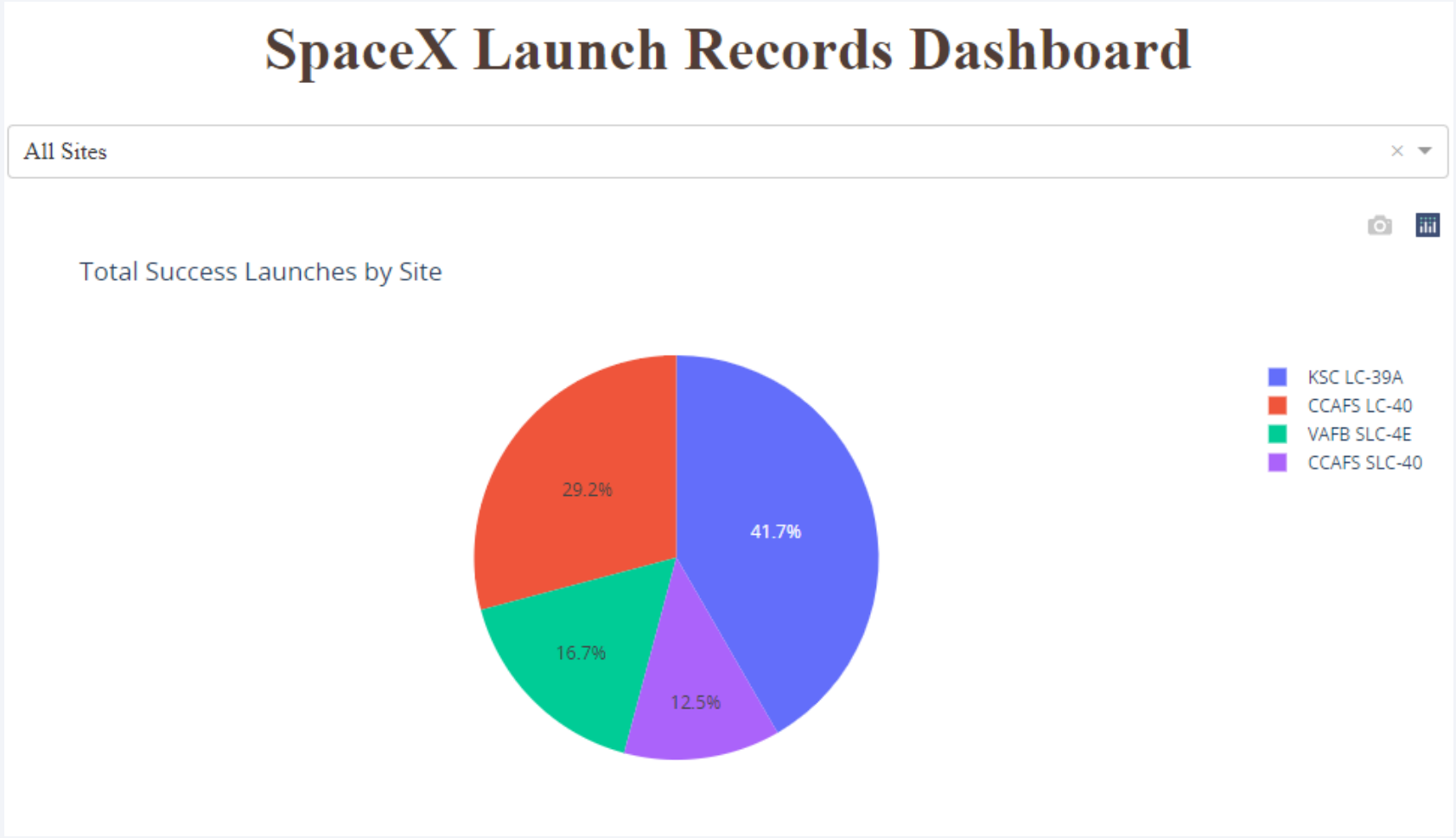
- YES. The nearest city is 51.74 km away.





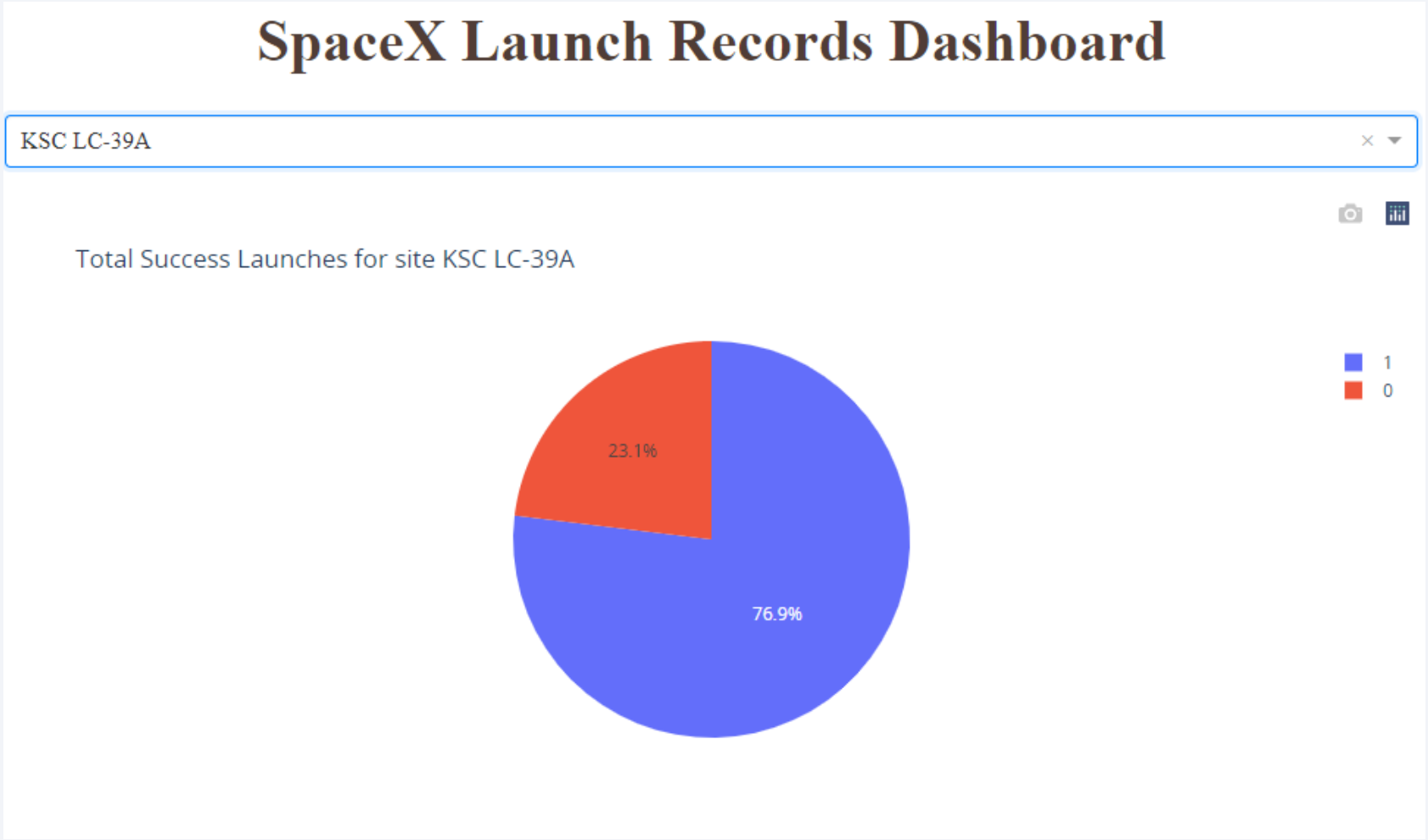

Section 4
Build a Dashboard with Plotly Dash

launch success count for all sites



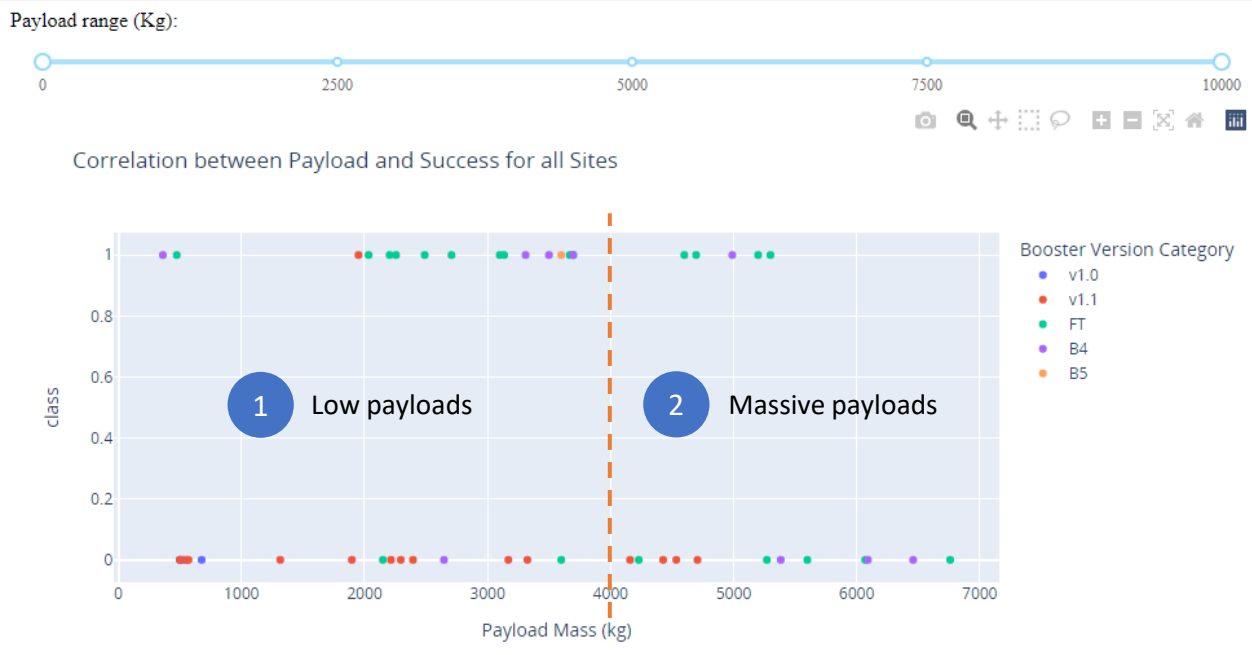
The launch site KSC LC-39 A had the most successful launches, with 41.7% of the total successful launches.

Pie chart for the launch site with highest launch success ratio

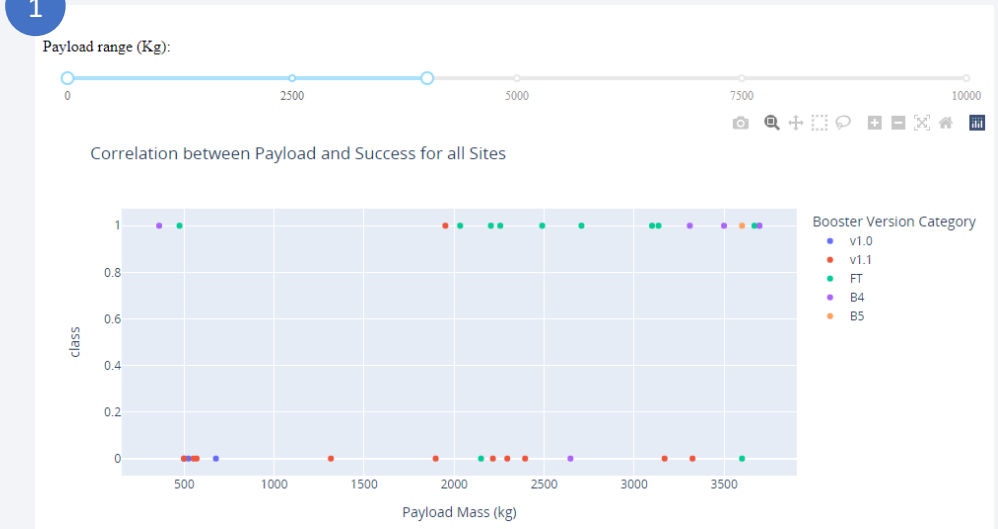


The launch site KSC LC-39 A also had the highest rate of successful launches, with a 76.9% success rate.

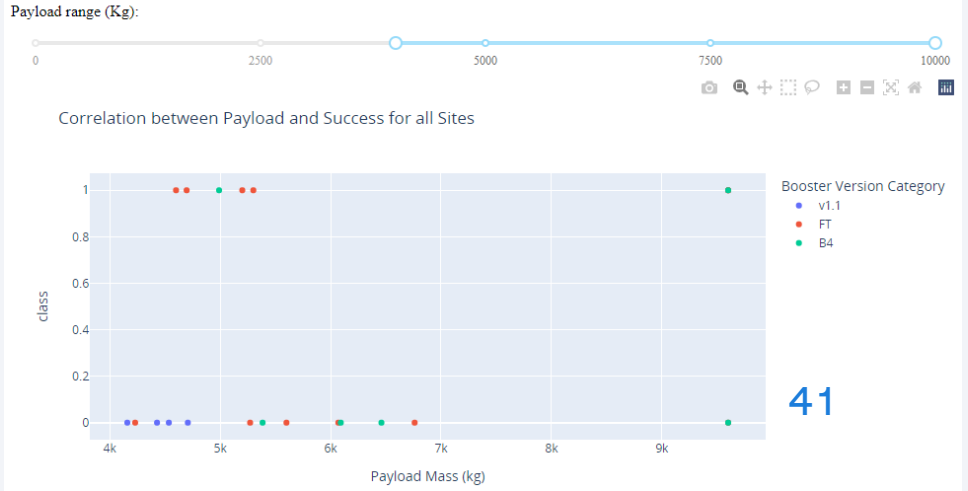
Launch Outcome VS. Payload scatter plot for all sites



1



2



- Plotting the launch outcome vs. payload for all sites shows a gap around 4000 kg, so it makes sense to split the data into 2 ranges:
  - 0 – 4000 kg (low payloads)
  - 4000 – 10000 kg (massive payloads)
- From these 2 plots, it can be shown that the success for massive payloads is lower than that for low payloads.
- It is also worth noting that some booster types (v1.0 and B5) have not been launched with massive payloads.

Note: $class \begin{cases} 0, Failure \\ 1, Success \end{cases}$

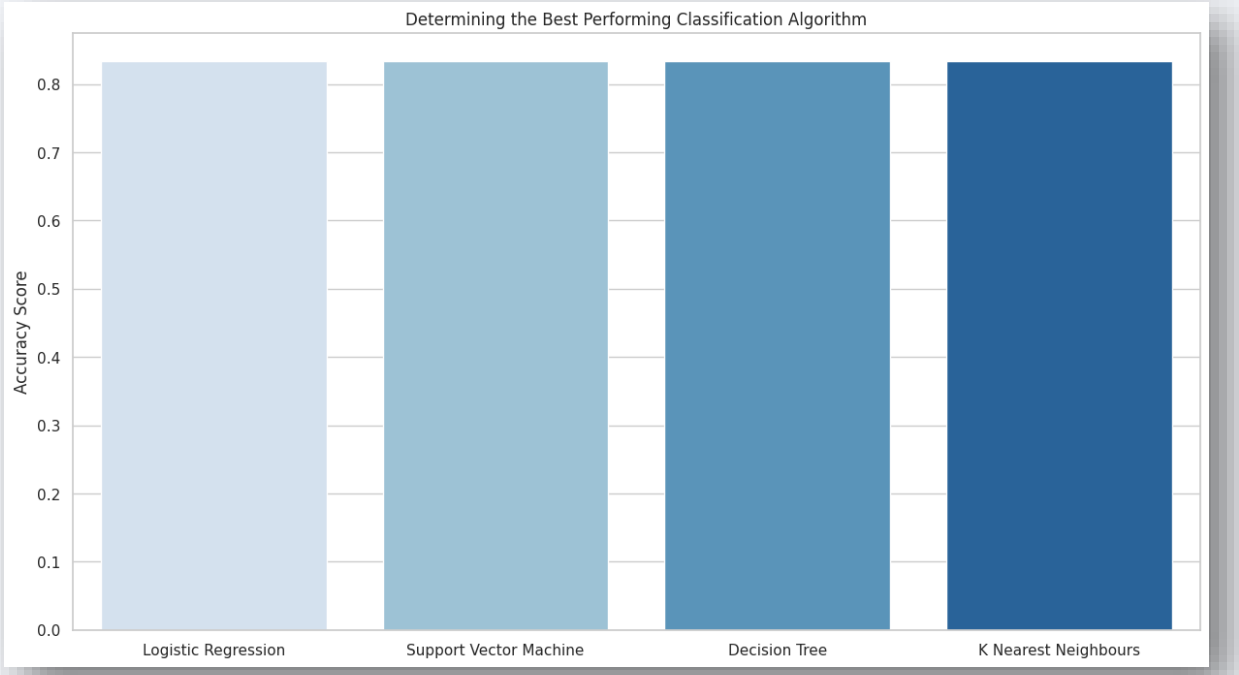
Section 5

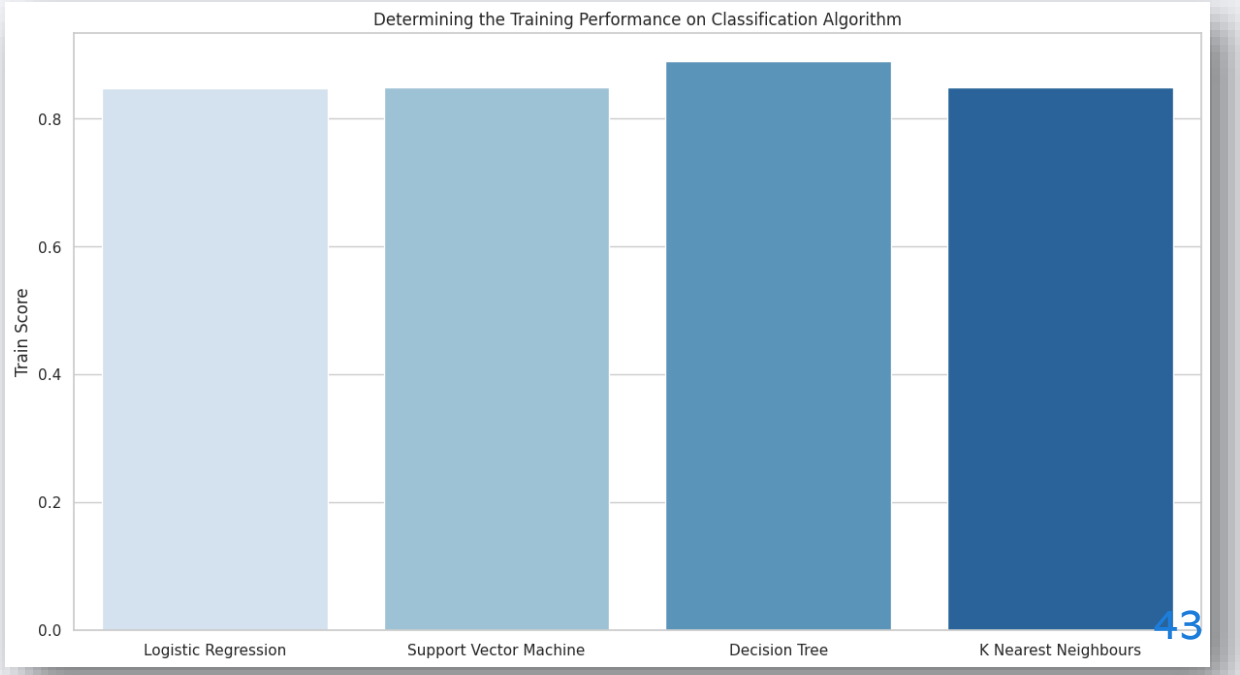
Predictive Analysis (Classification)

Classification Accuracy

- The following outcome is obtained by plotting the Accuracy Score and Best Score for each classification algorithm:
- The classification accuracy is highest with the Decision Tree model.
  - The Accuracy Score is 83.33%
  - The Best Score is 88.93%

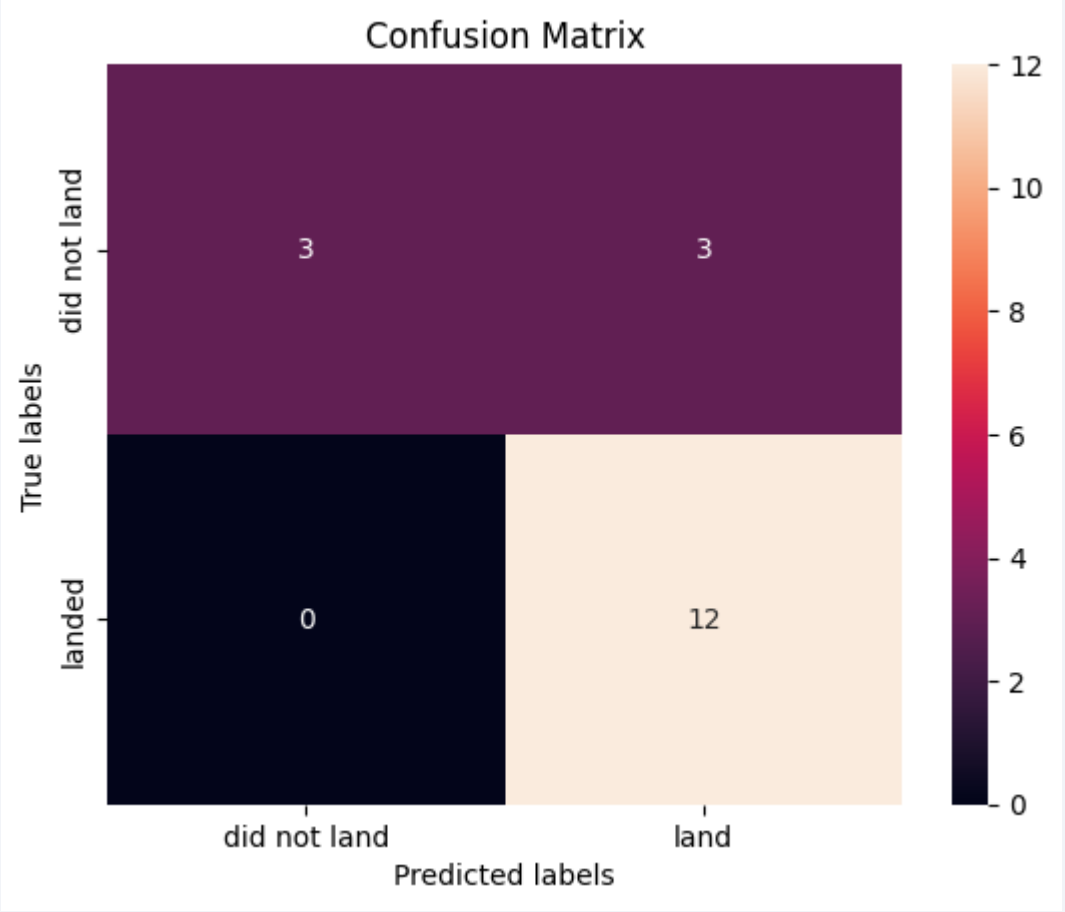
	Algorithm	Accuracy Score	Train Score
0	Logistic Regression	0.833333	0.846429
1	Support Vector Machine	0.833333	0.848214
2	Decision Tree	0.833333	0.889286
3	K Nearest Neighbours	0.833333	0.848214





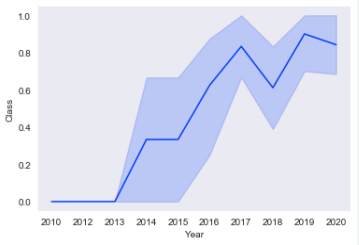
Confusion Matrix

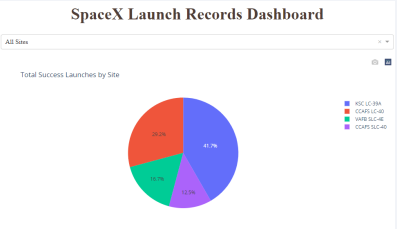
- The Decision Tree model, with an accuracy of 83.33%, is the best-performing classification model, as was previously demonstrated.
- The confusion matrix, which displays 3 out of a total of 18 results that were mistakenly categorized (a false positive, shown in the top-right corner), explains this.
- The remaining 15 results (3 didn't land, 12 did) are accurately categorized.

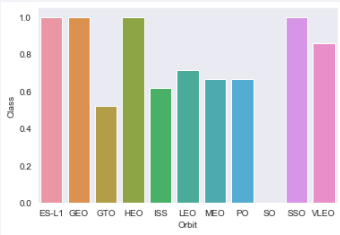


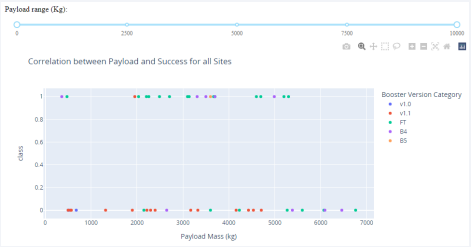
Conclusions

- As the number of flights increases, the rate of success at a launch site increases, with most early flights being unsuccessful. i.e., with more experience, the success rate increases.
  - Between 2010 and 2013, all landings were unsuccessful (as the success rate is 0).
  - After 2013, the success rate generally increased, despite small dips in 2018 and 2020.
  - After 2016, there was always a greater than 50% chance of success.
- Orbit types ES-L1, GEO, HEO, and SSO, have the highest (100%) success rate.
    - The 100% success rate of GEO, HEO, and ES-L1 orbits can be explained by only having 1 flight into the respective orbits.
    - The 100% success rate in SSO is more impressive, with 5 successful flights.
    - The orbit types PO, ISS, and LEO, have more success with heavy payloads:
    - VLEO (Very Low Earth Orbit) launches are associated with heavier payloads, which makes intuitive sense.
- 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 success for massive payloads (over 4000kg) is lower than that for low payloads.
- The best performing classification model is the Decision Tree model, with an accuracy of 83.33%.









Appendix – Data Collection – SpaceX API

- Custom code to clean the data

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

Appendix – Data Collection – SpaceX API

- Custom functions to retrieve the data

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

```
# Takes the dataset and uses the rocket column to call the API and append the data to the lis
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'])
```

MagicPython

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
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'])
```

MagicPython

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
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'])
```

MagicPython

From `cores` we would like to learn the outcome of the landing, the type of the landing, number of flights with that core, whether gridfins were used, wheter the core is reused, wheter legs were used, the landing pad used, the block of the core which is a number used to seperate version of cores, the number of times this specific core has been reused, and the serial of the core.

```
# Takes the dataset and uses the cores column to call the API and append the data to the list
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cores/"+core['core'])
                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'])
```

MagicPython

Appendix – Data Collection – Scraping

- Creating a launch_dict

```
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'] = []
```

MagicPython

```
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.`
            launch_dict['Flight No.'].append(flight_number)

            # print(flight_number)
            datatimelist = date_time(row[0])

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

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

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

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

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

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

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

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

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

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

Appendix – Data Collection – Scraping

- Custom functions to extract the data

```
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
```

Thank you!