Winning Space Race with Data Science

IBM Applied Data Science Capstone Project

Gayan Abeynanda
10-19-2022

Outline

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

Executive Summary

- Summary of methodologies:

  This project is comprised of

  - Data Collection
  - Data Wrangling
  - Exploratory Data Analysis
  - Interactive Visual Analytics
  - Predictive Analysis through Machine Learning Models

- Summary of all results

  This project resulted in the following analysis and outputs:

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

Introduction

- Project background and context
  - Space X launches Falcon 9 rockets at a much lower cost compare to competitors ($62m in comparison to $165m or above).
  - Much of the savings come from Space X's ability to safely land and then re-use stage I of the rocket.
  - A new rocket company - Space Y plans to compete with Space X
  - If we can predict whether the first stage of a given Space X - Falcon 9 launch will land, this information can be used to determine the average cost of such a launch.

- Problems for which answers are sought
  - This project aims to predict if the first stage of a given Space X - Falcon 9 rocket launch will land successfully.

Section 1

Methodology

Methodology: Executive Summary

1. Data Collection
   - Making GET requests to the SpaceX REST API
   - Web Scraping
2. Data Wrangling
   - Using the .fillna() method to remove NaN values
   - Using the .value_counts() method to determine the following:
     - Number of launches on each site
     - Number and occurrence of each orbit
     - Number and occurrence of mission outcome per orbit type
   - Creating a landing outcome label that shows:
     - 0 when the booster did not land successfully
     - 1 when the booster did land successfully
3. Exploratory Data Analysis
   - Using SQL queries to manipulate and evaluate the SpaceX dataset
   - Using Pandas and Matplotlib to visualize relationships between variables, and determine patterns
4. Interactive Visual Analytics
   - Geospatial analytics using Folium
   - Creating an interactive dashboard using Plotly Dash
5. Data Modelling and Evaluation
   - 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

Methodology: Data Collection

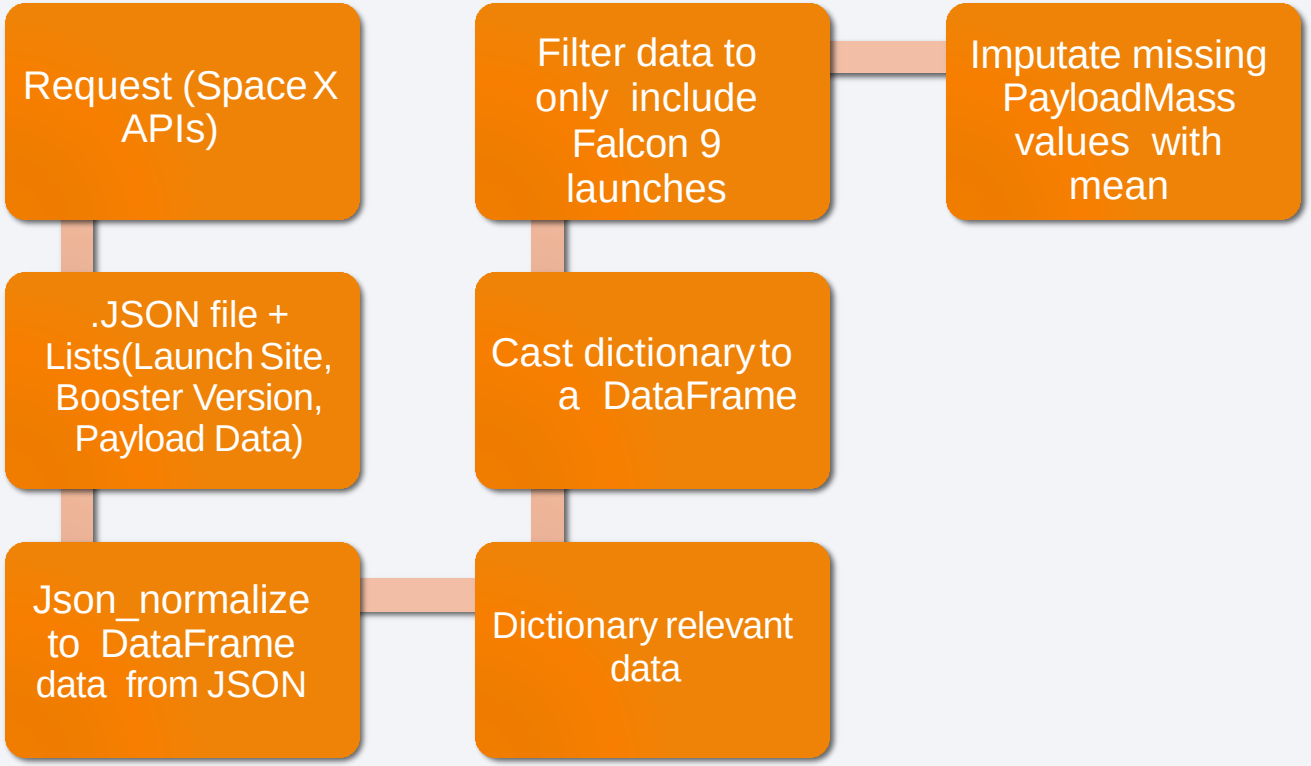
- Data collection phase comprised of:
  - Making GET requests to collect data from the SpaceX REST API
  - Web scraping data from a table in Space X's Falcon 9 and Falcon Heavy launches wiki page.

- Space X API Data Columns:
  - FlightNumber, Date, BoosterVersion, PayloadMass, Orbit, LaunchSite, Outcome, Flights, GridFins, Reused, Legs, LandingPad, Block, ReusedCount, Serial, Longitude, Latitude

- Webscraped Data Columns from Wikipedia:
  - Flight No., Launch site, Payload, PayloadMass, Orbit, Customer, Launch outcome, Version Booster, Booster landing, Date, Time

Methodology: Data Collection – Space-X 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.
- GitHub URL: https://github.com/gsabeynanda/IBM-DS-Prof-Cert-Capstone/blob/main/jupyter-labs-spacex-data-collection-api.ipynb



Methodology: Data Collection – Space-X API

Step 1

- Make a GET response to the SpaceX REST API
- Convert the response to a .json file then to a Pandas DataFrame

Step 2

- Use custom logic to clean the data (see Appendix)
- Define lists for data to be stored in
- Call custom functions (see Appendix) to retrieve data and fill the lists
- Use these lists as values in a dictionary and construct the dataset

Step 3

- Create a Pandas DataFrame from the constructed dictionary dataset

Step 4

- Filter the DataFrame to only include Falcon 9 launches
- Reset the FlightNumber column
- Replace missing values of PayloadMass with the mean PayloadMass value

Methodology: Data Collection – Space-X API

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

response = requests.get(spacex_url)
```

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

```
# 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}
```

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

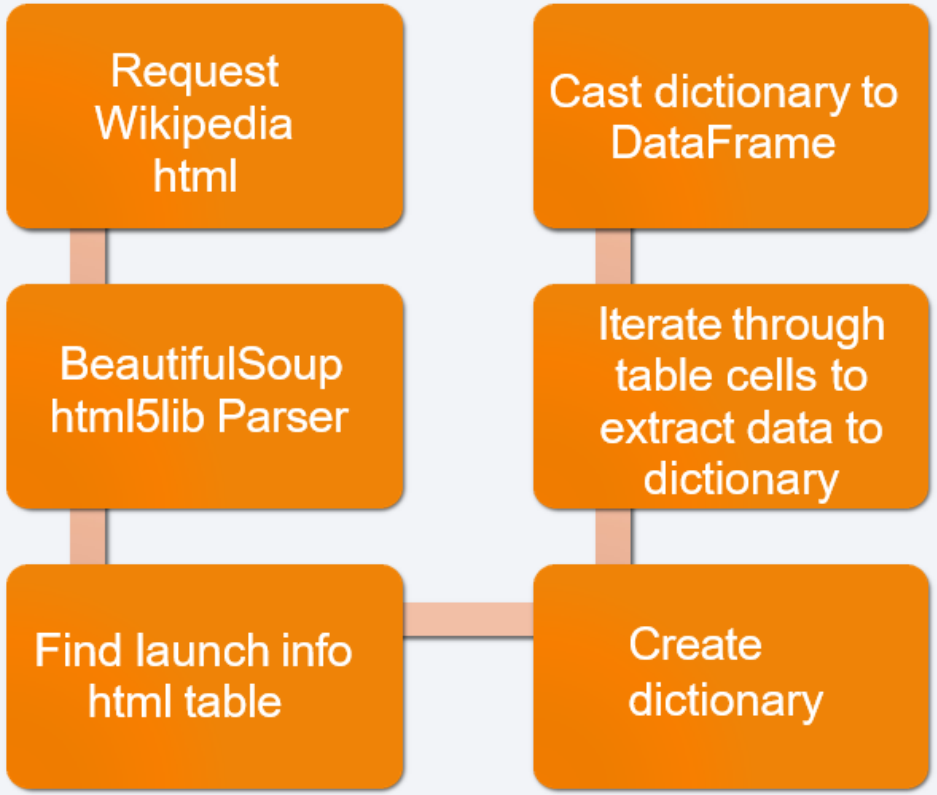
```
data_falcon9 = df[df['BoosterVersion']!='Falcon 1']
```

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

```
# Calculate the mean value of PayloadMass column and Replace the np.nan values with its mean value
data_falcon9 = data_falcon9.fillna(value={'PayloadMass': data_falcon9['PayloadMass'].mean()})
```

Methodology: Data Collection – Web scraping

- Using web scraping to collect Falcon 9 historical launch records from a Wikipedia page titled “List of Falcon 9 and Falcon Heavy launches”.
- GitHub URL: https://github.com/gsabeynanda/IBM-DS-Prof-Cert-Capstone/blob/main/jupyter-labs-webscraping.ipynb



Methodology: Data Collection – Web scraping

Step 1

- Request the HTML page from the static URL
- Assign the response to an object

Step 2

- Create a BeautifulSoup object from the HTML response object
- Find all tables within the HTML page

Step 3

- Collect all column header names from the tables found within the HTML page

Step 4

- Use the column names as keys in a dictionary
- Use custom functions and logic to parse all launch tables (see Appendix) to fill the dictionary values

Step 5

- Convert the dictionary to a Pandas DataFrame ready for export

Methodology: Data Collection – Web scraping

```
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
data = response.text
```

```
soup = BeautifulSoup(data, 'html5lib')
```

```
html_tables = soup.find_all('table')
```

```
column_names = []

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

for 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)
```

```
launch_dict= dict.fromkeys(column_names)

# Remove an irrelevant 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']=[]
```

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

Methodology: Data wrangling

- Data wrangling/manipulation phase comprised of:
  - Exploratory Data Analysis
    Initial Exploration of data in the data frame to understand the required manipulation to get the data in to the form we need.
  - Determine Training Labels
    Creation of a stage I landing outcome column that had a binary representation of success or failure to land.
- GitHub URL: https://github.com/gsabeynanda/IBM-DS-Prof-Cert-Capstone/blob/main/labs-jupyter-spacex-Data%20wrangling.ipynb

Methodology: Data wrangling – Data exploration

➢Context:

- The SpaceX dataset contains several Space X launch facilities, and each location is in the LaunchSite column.
- Each launch is assigned a designated orbit (see appendix), and this orbit type assigned for each launch is in the Orbit column.

➢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

Methodology: Data wrangling – Training labels

➢ Context:

- The landing outcome is shown in the Outcome column:
  - 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.

➢ Data wrangling:

- To determine whether a booster will successfully land, it is best to have a binary column, i.e., where the value is 1 or 0, representing the success of the landing.
- This is done by:
  - Defining a set of unsuccessful (bad) outcomes, `bad_outcome`
  - 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.
  - Create a `Class` column that contains the values from the list `landing_class`
  - Export the DataFrame as a .csv file.

EDA with Data Visualization

- Exploratory data analysis (EDA) using data visualizations phase comprised of:
  - Scatter plots
    To visualize several data relationships.
  - Bar charts
    To visualize several the relationships between Success Rate and Orbit Type.
  - Line charts
    To visualize several the relationships between Success Rate and Year.

- GitHub URL: https://github.com/gsabeynanda/IBM-DS-Prof-Cert-Capstone/blob/main/jupyter-labs-eda-dataviz.ipynb

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

BAR CHARTS

A bar chart was produced to visualize the relationship between:

- Success Rate and Orbit Type

➢ Bar charts 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 CHARTS

Line charts were produced to visualize the relationships between:

- Success Rate and Year (i.e. the launch success yearly trend)

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

Exploratory data analysis (EDA) with SQL

- Exploratory data analysis (EDA) using SQL phase comprised of:
  - Loading data set into IBM DB2.
  - SQL queries made using python to get a better understanding of the dataset.
  - Queries for information about launch site names, mission outcomes, various pay load sizes of customers and booster versions, and landing outcomes

- GitHub URL: https://github.com/gsabeynanda/IBM-DS-Prof-Cert-Capstone/blob/main/jupyter-labs-eda-sql-coursera_sqllite.ipynb

Exploratory data analysis (EDA) with SQL

- The SQL queries performed on the data set were used 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 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.

Geospatial Analysis - Interactive Map with Folium

- Visual Geospatial analysis using Folium phase comprised of:
  - Mark all launch sites on a map.
  - Mark the success or failed status for each launches from each site on a map.
  - Calculate the distances from a launch site to its important proximities

- GitHub URL: https://github.com/gsabeynanda/IBM-DS-Prof-Cert-Capstone/blob/main/lab_jupyter_launch_site_location.ipynb

Geospatial Analysis - Interactive Map with Folium

- The steps taken to visualize the launch data on an interactive map were as follows:
  - Mark all launch sites on a map
    - Initialise the map using a Folium Map object
    - Add a folium.Circle and folium.Marker for each launch site on the launch map
  - Mark the success/failed launches for each site on a map
    - As many launches have the same coordinates, it makes sense to cluster them together.
    - Before clustering them, assign a marker colour of successful (class = 1) as green, and failed (class = 0) as red.
    - To put the launches into clusters, for each launch, add a folium.Marker to the MarkerCluster() object.
    - Create an icon as a text label, assigning the icon_color as the marker_colour determined previously.
  - Calculate the distances between a launch site to its proximities
    - To explore the proximities of launch sites, calculations of distances between points can be made using the Lat and Long values.
    - After marking a point using the Lat and Long values, create a folium.Marker object to show the distance.
    - To display the distance line between two points, draw a folium.PolyLine and add this to the map.

Interactive Dashboard Visualization - Using Plotly Dash

- Interactive Dashboard Visualization with Plotly Dash phase comprised of:
  - Pie chart showing the successful and failed launch percentages per site with a drop down many to toggle between sites
  - Scatter plot to show the correlation between outcome (success or failure) and payload mass facilitated with a slider to choose min/max for payload mass.

- GitHub URL: https://github.com/gsabeynanda/IBM-DS-Prof-Cert-Capstone/blob/main/spacex_dash_app.py

Interactive Dashboard Visualization - Using Plotly Dash

- The following plots were added to a Plotly Dash dashboard to have an interactive visualisation of the data:
  - Pie chart (`px.pie()`) showing the successful and failed launch percentages per site with a drop down many to toggle between sites
    - This makes it clear to see which sites are most successful
    - The chart could also be filtered (using a `dcc.Dropdown()` object) to see the success/failure ratio for an individual site
  - Scatter plot to show the correlation between outcome (success or failure) and payload mass facilitated with a slider to choose min/max for payload mass (kg).
    - This could be filtered (using a `RangeSlider()` object) by ranges of payload masses
    - It could also be filtered by booster version

Predictive Analysis (Classification)

- Predictive Analysis (Classification) phase comprised of:
  - Developing several types of machine learning models.
  - Created a training/test split of the dataset.
  - Each classification model was trained and fitted.
  - Each classification model was evaluated so as to choose the best model for our prediction purpose.

- GitHub URL: https://github.com/gsabeynanda/IBM-DS-Prof-Cert-Capstone/blob/main/SpaceX_Machine%20Learning%20Prediction_Part_5.ipynb

Predictive Analysis (Classification)

➢ Steps taken to develop, evaluate, and find the best performing classification model:

Model Development

- To prepare the dataset for model development:
  - Load dataset
  - Perform necessary data transformations (standardise and pre-process)
  - Split data into training and test data sets, using `train_test_split()`
  - Decide which type of machine learning algorithms are most appropriate
- For each chosen algorithm:
  - Create a `GridSearchCV` object and a dictionary of parameters
  - Fit the object to the parameters
  - Use the training data set to train the model

Model Evaluation

- For each chosen algorithm:
  - Using the output GridSearchCV object:
    - Check the tuned hyperparameters (`best_params_`)
    - Check the accuracy (`score` and `best_score_`)
  - Plot and examine the Confusion Matrix

Finding the Best Model

- Review the accuracy scores for all chosen algorithms
- The model with the highest accuracy score is determined as the best performing model

Results

Exploratory Data Analysis

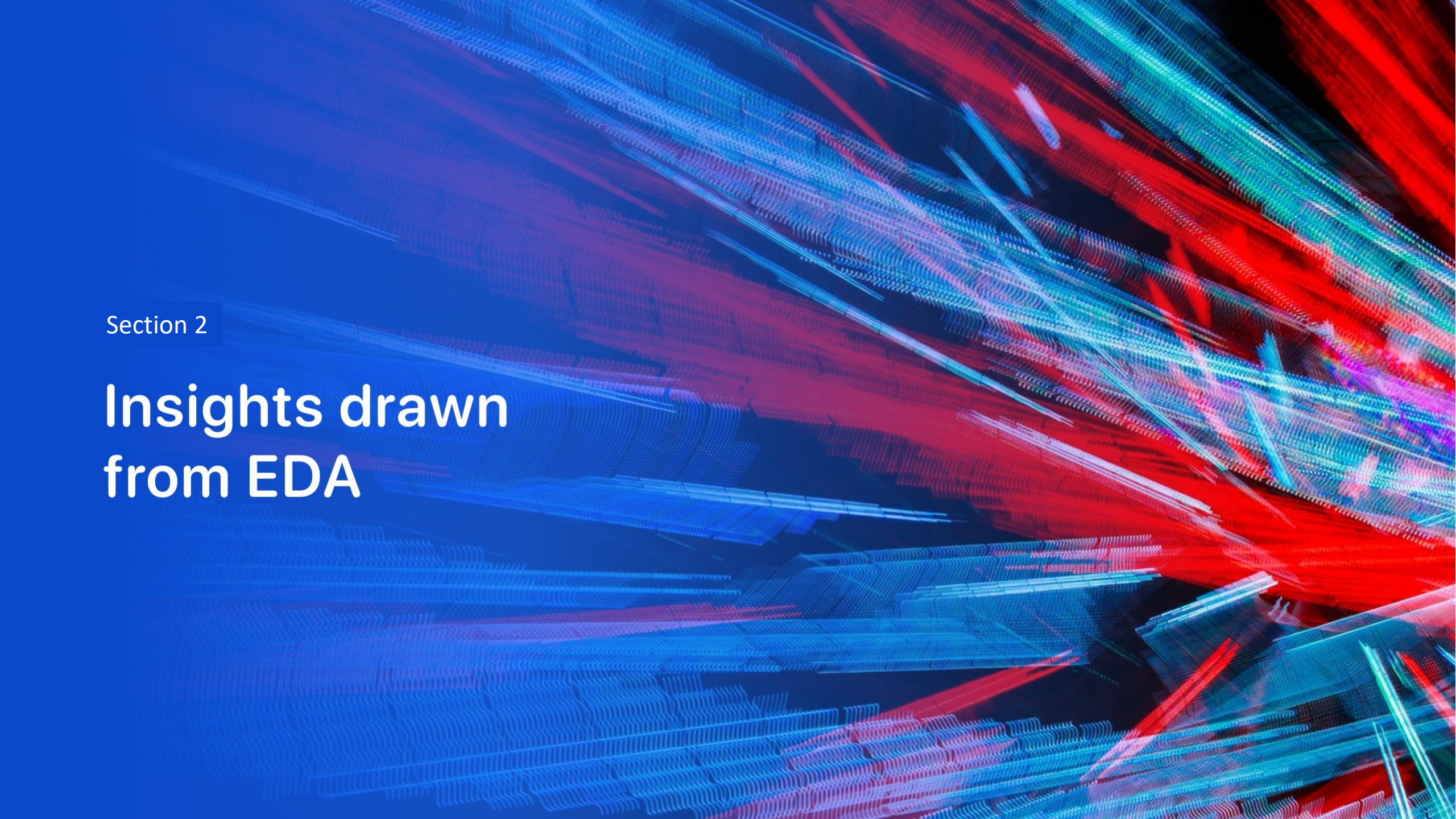
- Exploratory data analysis (EDA) using data visualizations
- Exploratory data analysis (EDA) using SQL

Interactive Analytics

- Interactive data analysis using Dashboard Visualization with Plotly Dash
- Interactive data analysis using Geospatial Visualization with Folium

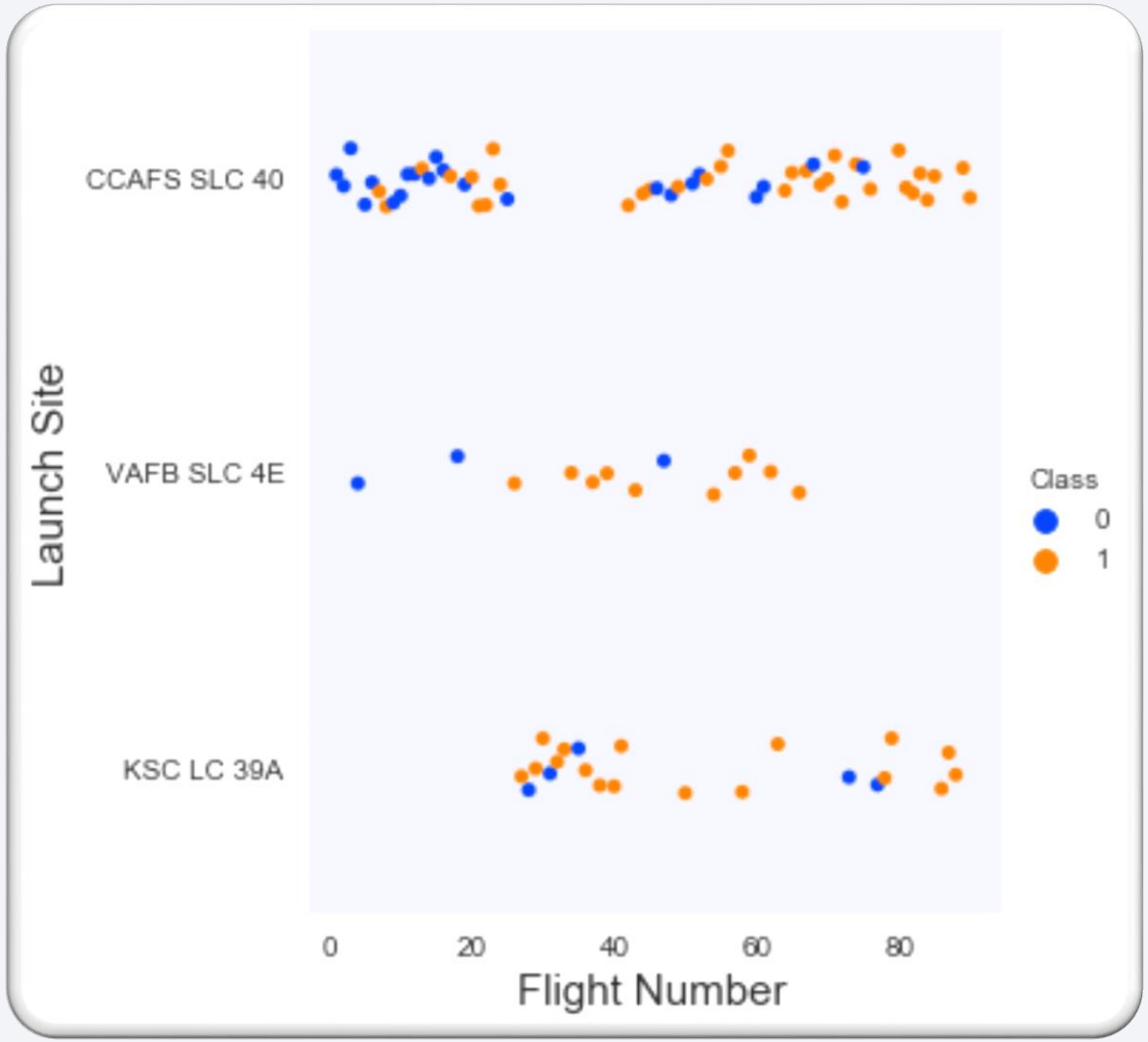
Predictive Analysis

- Model Evaluation
- The best model for the situation


Section 2
Insights drawn from EDA

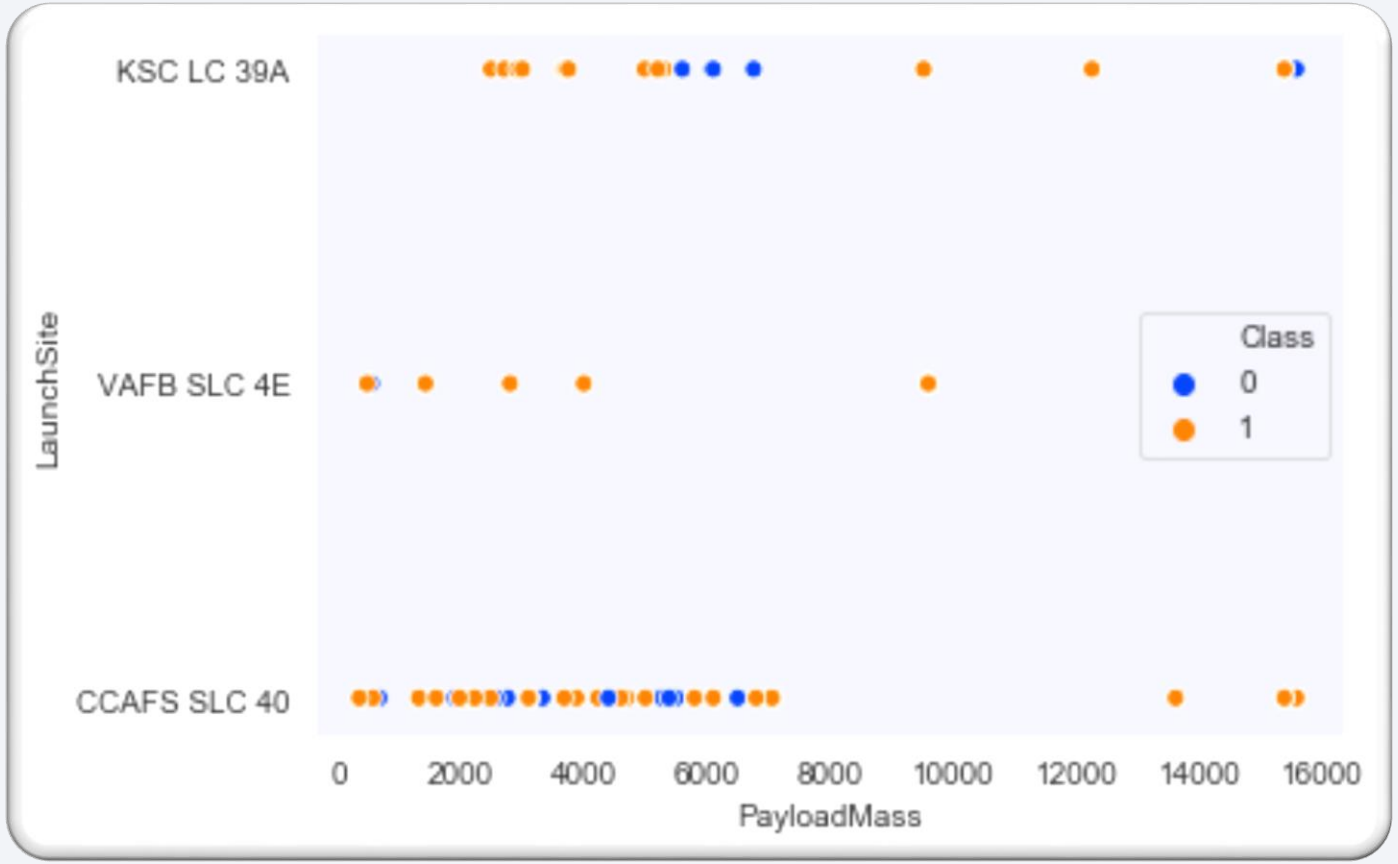
Data Visualization - Flight Number vs. Launch Site

- The scatter plot of Launch Site vs. Flight Number shows that:
  - As the number of flights increases, the rate of success at a launch site increases.
  - Most of the early flights (flight numbers < 30) were launched from CCAFS SLC 40, and were generally unsuccessful.
  - The flights from VAFB SLC 4E also show this trend, that earlier flights were less successful.
  - No early flights were launched from KSC LC 39A, so the launches from this site are more successful.
  - Above a flight number of around 30, there are significantly more successful landings (Class = 1).



Data Visualization - Payload vs. Launch Site

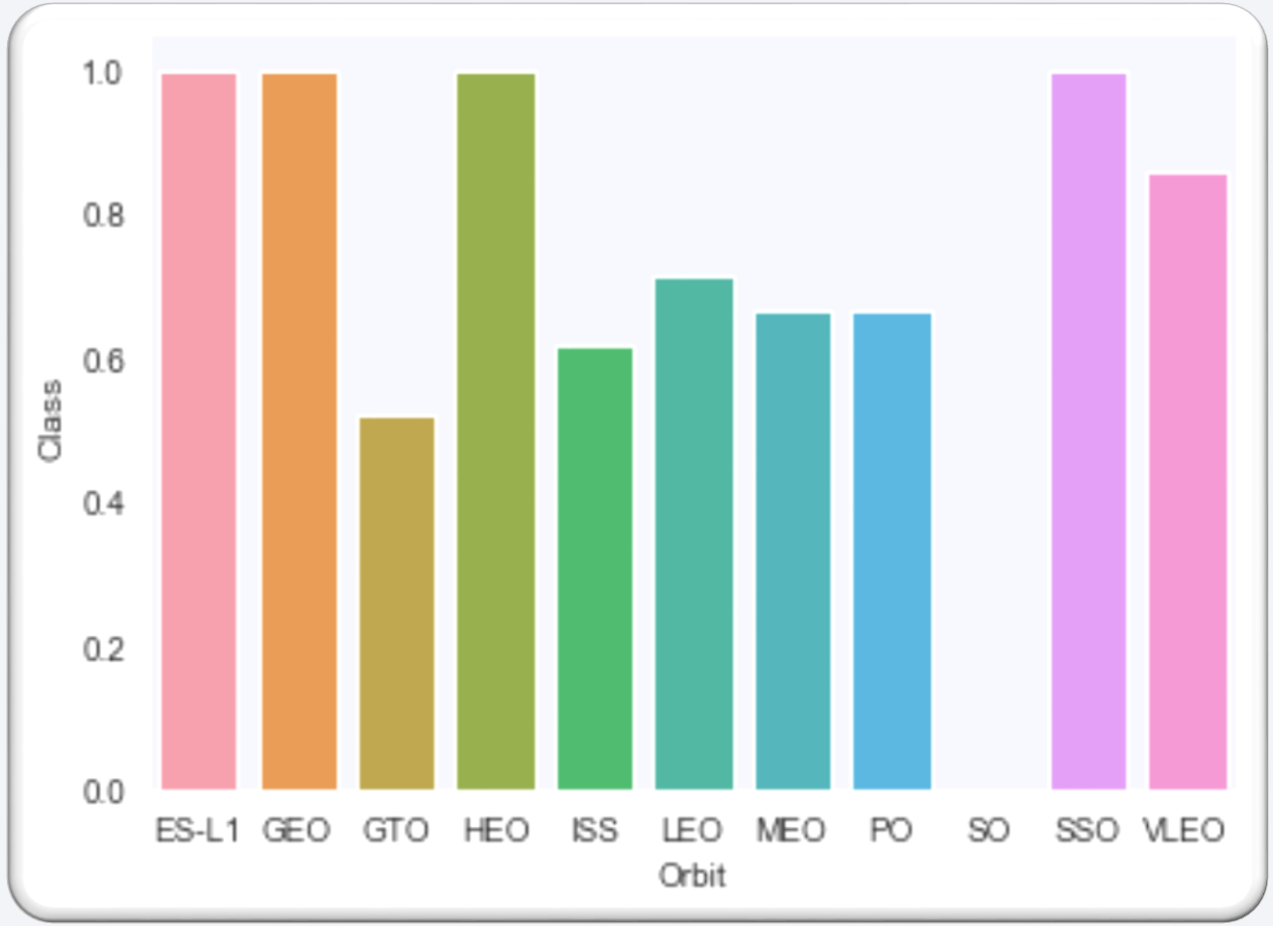
- The scatter plot of Launch Site vs. Payload Mass shows that:
  - Above a payload mass of around 7000 kg, there are very few unsuccessful landings, but there is also far less data for these heavier launches.
  - There is no clear correlation between payload mass and success rate for a given launch site.
  - All sites launched a variety of payload masses, with most of the launches from CCAFS SLC 40 being comparatively lighter payloads (with some outliers).



Data Visualization - Success Rate vs. Orbit Type

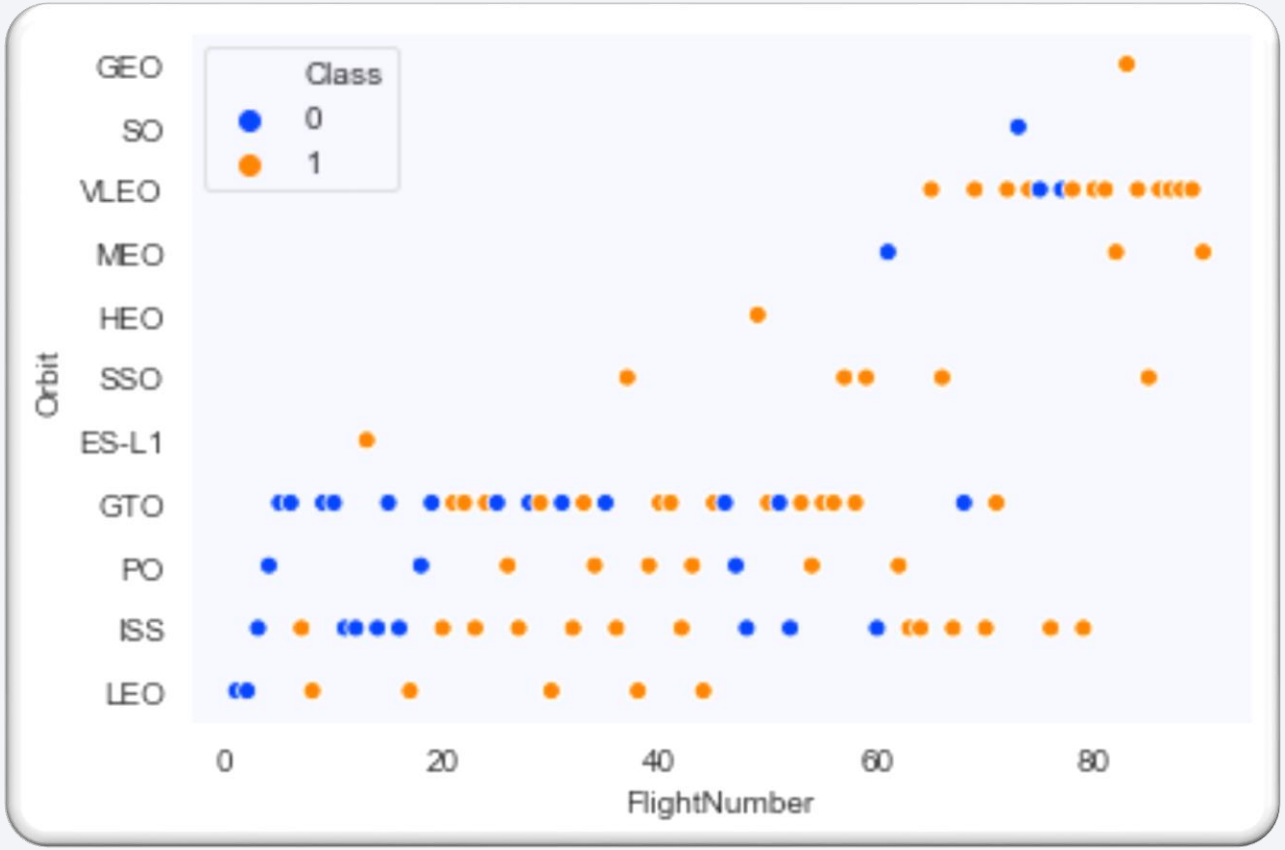
- The bar chart of Success Rate vs. Orbit Type shows that the following orbits have the highest (100%) success rate:
  - ES-L1 (Earth-Sun First Lagrangian Point)
  - GEO (Geostationary Orbit)
  - HEO (High Earth Orbit)
  - SSO (Sun-synchronous Orbit)

- The orbit with the lowest (0%) success rate is:
  - SO (Heliocentric Orbit)



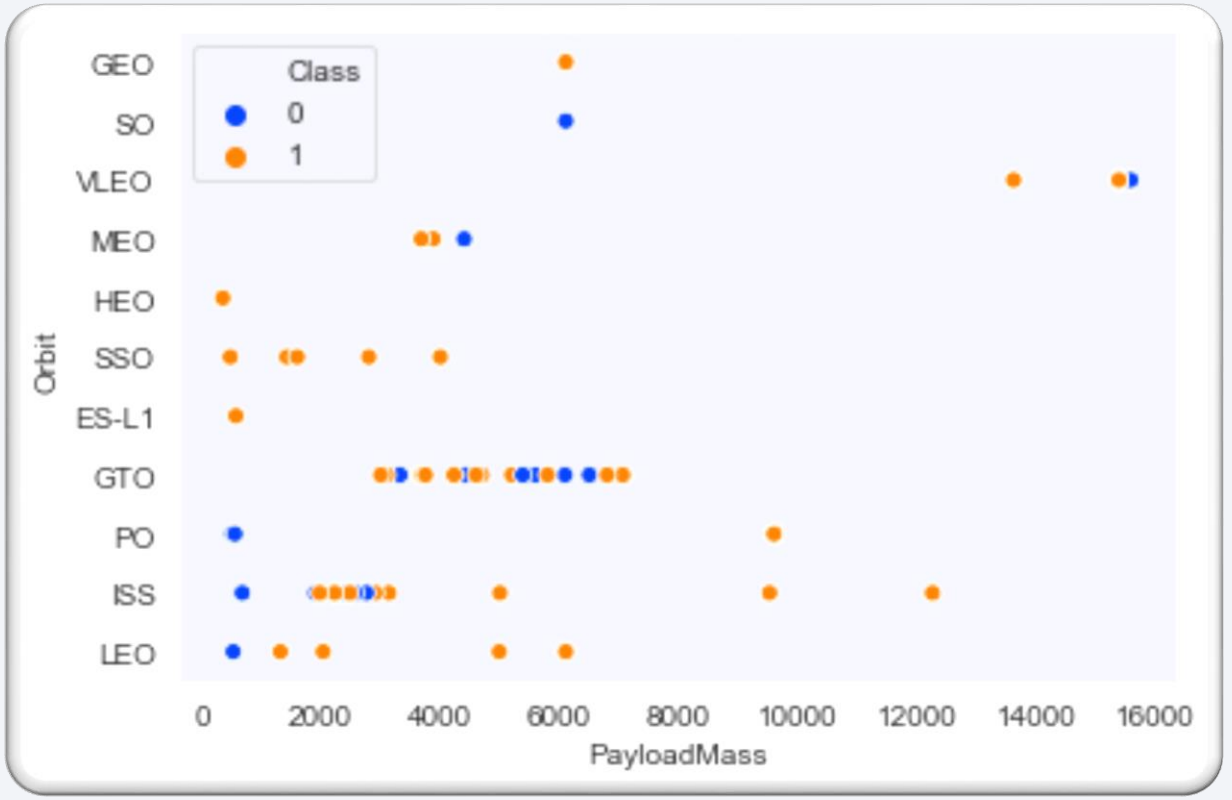
Data Visualization - Flight Number vs. Orbit Type

- This scatter plot of Orbit Type vs. Flight number shows a few useful things that the previous plots did not, such as:
  - 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.
  - There is little relationship between Flight Number and Success Rate for GTO.
  - Generally, as Flight Number increases, the success rate increases. This is most extreme for LEO, where unsuccessful landings only occurred for the low flight numbers (early launches).



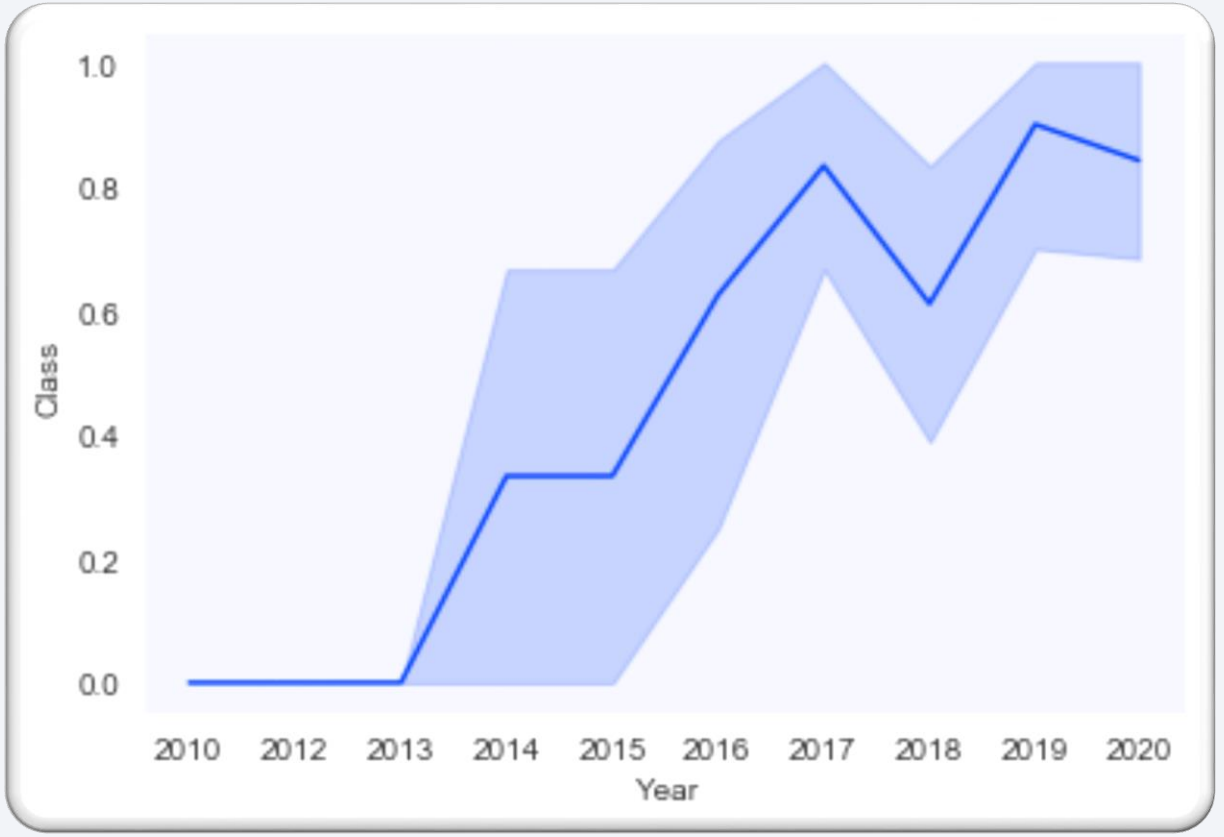
Data Visualization - Payload vs. Orbit Type

- This scatter plot of Orbit Type vs. Payload Mass shows that:
  - The following orbit types have more success with heavy payloads:
    - PO (although the number of data points is small)
    - ISS
    - LEO
  - For GTO, the relationship between payload mass and success rate is unclear.
  - VLEO (Very Low Earth Orbit) launches are associated with heavier payloads, which makes intuitive sense.



Data Visualization - Launch Success Yearly Trend

- The line chart of yearly average success rate shows that:
  - 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.



EDA with SQL - All Launch Site Names

```
In [4]: %%sql
        SELECT UNIQUE LAUNCH_SITE
        FROM SPACEXDATASET;

         * ibm_db_sa://ftb12020:***@0c77d6f2
        Done.
```

Out[4]:

launch_site
CCAFS LC-40
CCAFS SLC-40
CCAFSSLC-40
KSC LC-39A
VAFB SLC-4E

- Query unique launch site names from database.
- CCAFS SLC-40 and CCAFSSLC-40 likely both represent the same launch site with CCAFSSLC-40 being a data entry error.
- Likely only four distinct launch sites:

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

  CCAFS LC-40

- The SQL command `UNIQUE` returns only unique values from the `LAUNCH_SITE` column of the `SPACEXTBL` table.

Launch Site Names Begin with 'CCA'

In [5]:
```
%%sql
SELECT *
FROM SPACEXDATASET
WHERE LAUNCH_SITE LIKE 'CCA%'
LIMIT 5;
```

```
 * ibm_db_sa://ftb12020:***@0c77d6f2-5da9-48a9-81f8-86b520b87518.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31198/bludb
Done.
```

Out[5]:

DATE	time__utc_	booster_version	launch_site	payload	payload_mass__kg_	orbit	customer	mission_outcome	landing__outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2012-05-22	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

First five entries in database with Launch Site name beginning with CCA

Total Payload Mass

```
%%sql
SELECT SUM(PAYLOAD_MASS__KG_) AS SUM_PAYLOAD_MASS_KG
FROM SPACEXDATASET
WHERE CUSTOMER = 'NASA (CRS)';
```

```
 * ibm_db_sa://ftb12020:***@0c77d6f2-5da9-48a9-81f8-86
Done.
```

sum_payload_mass_kg
45596

- This query sums the total payload mass in kg where NASA was the customer.
- CRS stands for Commercial Resupply Services which indicates that these payloads were sent to the International Space Station (ISS).

Average Payload Mass by F9 v1.1

```
%%sql
SELECT AVG(PAYLOAD_MASS__KG_) AS AVG_PAYLOAD_MASS_KG
FROM SPACEXDATASET
WHERE booster_version = 'F9 v1.1'
```

```
 * ibm_db_sa://ftb12020:***@0c77d6f2-5da9-48a9-81f8-8
Done.
```

avg_payload_mass_kg
2928

- This query calculates the average payload mass or launches which used booster version F9 v1.1
- Average payload mass of F9 1.1 is on the low end of our payload mass range

First Successful Ground Landing Date

```
%%sql
SELECT MIN(DATE) AS FIRST_SUCCESS
FROM SPACEXDATASET
WHERE landing__outcome = 'Success (ground pad)';
```

```
 * ibm_db_sa://ftb12020:***@0c77d6f2-5da9-48a9-81
Done.
```

first_success
2015-12-22

- This query returns the first successful ground pad landing date.
- First ground pad landing wasn't until the end of 2015.
- Successful landings in general appear starting 2014.

Successful Drone Ship Landing with Payload between 4000 and 6000

```
%%sql
SELECT booster_version
FROM SPACEXDATASET
WHERE landing__outcome = 'Success (drone ship)' AND payload_mass__kg_ BETWEEN 4001 AND 5999;
```

```
 * ibm_db_sa://ftb12020:***@0c77d6f2-5da9-48a9-81f8-86b520b87518.bs2io90l08kqb1od8lcg.database
Done.
```

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

- This query returns the four booster versions that had successful drone ship landings with a payload mass between 4000 and 6000.

Total Number of Successful and Failure Mission Outcomes

```
%%sql
SELECT mission_outcome, COUNT(*) AS no_outcome
FROM SPACEXDATASET
GROUP BY mission_outcome;
```

```
 * ibm_db_sa://ftb12020:***@0c77d6f2-5da9-48a9-
Done.
```

mission_outcome	no_outcome
Failure (in flight)	1
Success	99
Success (payload status unclear)	1

- This query returns a count of each mission outcome.
- SpaceX appears to achieve its mission outcome nearly 99% of the time.
- This means that most of the landing failures are intended.
- Interestingly, one launch has an unclear payload status and unfortunately one failed in flight.

Boosters Carried Maximum Payload

```
%%sql
SELECT booster_version, PAYLOAD_MASS__KG_
FROM SPACEXDATASET
WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXDATASET);
```

```
 * ibm_db_sa://ftb12020:***@0c77d6f2-5da9-48a9-81f8-86b520b87518.bs2io90l08kqb1
Done.
```

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

- This query returns the booster versions that carried the highest payload mass of 15600 kg.
- These booster versions are very similar and all are of the F9 B5 B10xx.x variety.
- This likely indicates payload mass correlates with the booster version that is used.

2015 Launch Records

```
%%sql
SELECT MONTHNAME(DATE) AS MONTH, landing__outcome, booster_version, PAYLOAD_MASS__KG_, launch_site
FROM SPACEXDATASET
WHERE landing__outcome = 'Failure (drone ship)' AND YEAR(DATE) = 2015;
```

```
 * ibm_db_sa://ftb12020:***@0c77d6f2-5da9-48a9-81f8-86b520b87518.bs2io90l08kqb1od8lcg.databases.app
Done.
```

MONTH	landing__outcome	booster_version	payload_mass__kg_	launch_site
January	Failure (drone ship)	F9 v1.1 B1012	2395	CCAFS LC-40
April	Failure (drone ship)	F9 v1.1 B1015	1898	CCAFS LC-40

- This query returns the Month, Landing Outcome, Booster Version, Payload Mass (kg), and Launch site of 2015 launches where stage 1 failed to land on a drone ship.
- There were two such occurrences.

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

```
%%sql
SELECT landing__outcome, COUNT(*) AS no_outcome
FROM SPACEXDATASET
WHERE landing__outcome LIKE 'Succes%' AND DATE BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY landing__outcome
ORDER BY no_outcome DESC;
```

```
 * ibm_db_sa://ftb12020:***@0c77d6f2-5da9-48a9-81f8-86b520b87518.bs2io90l08kqb1od8lcg
Done.
```

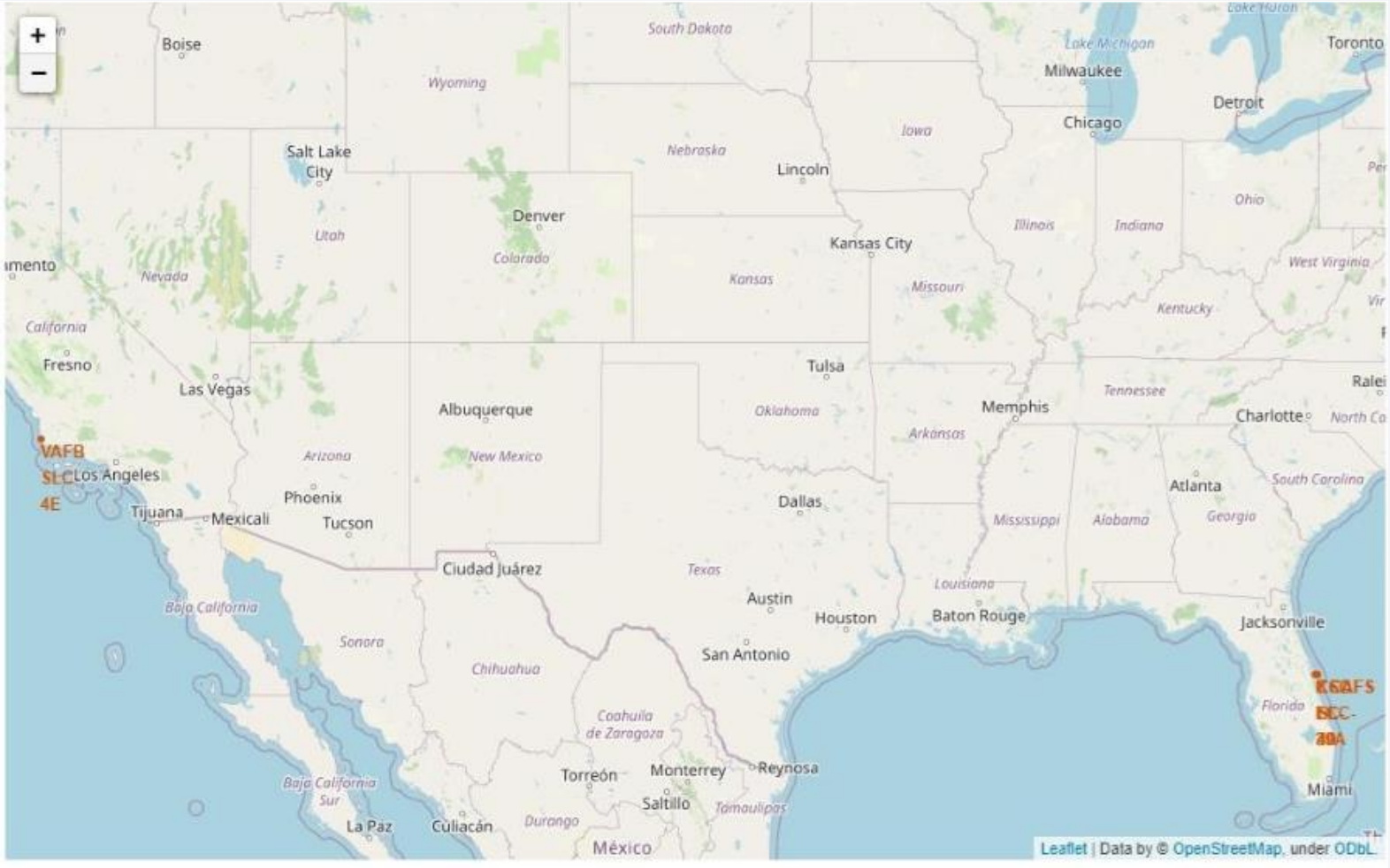
landing__outcome	no_outcome
Success (drone ship)	5
Success (ground pad)	3

- This query returns a list of successful landings and between 2010-06-04 and 2017-03-20 inclusively.
- There are two types of successful landing outcomes: drone ship and ground pad landings.
- There were 8 successful landings in total during this time period

Section 3

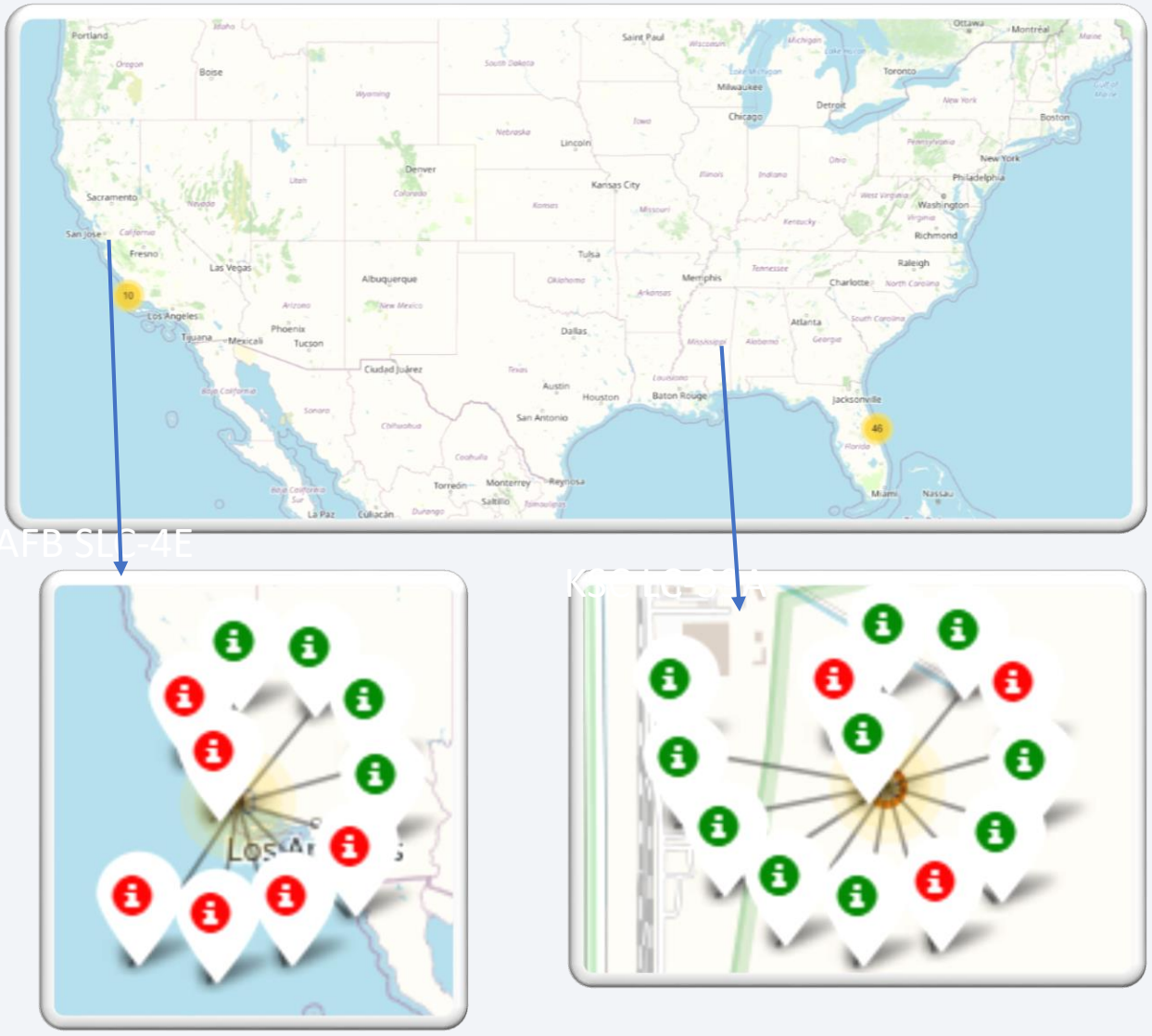
Launch Sites Proximities Analysis

Folium Interactive Map – Launch site locations



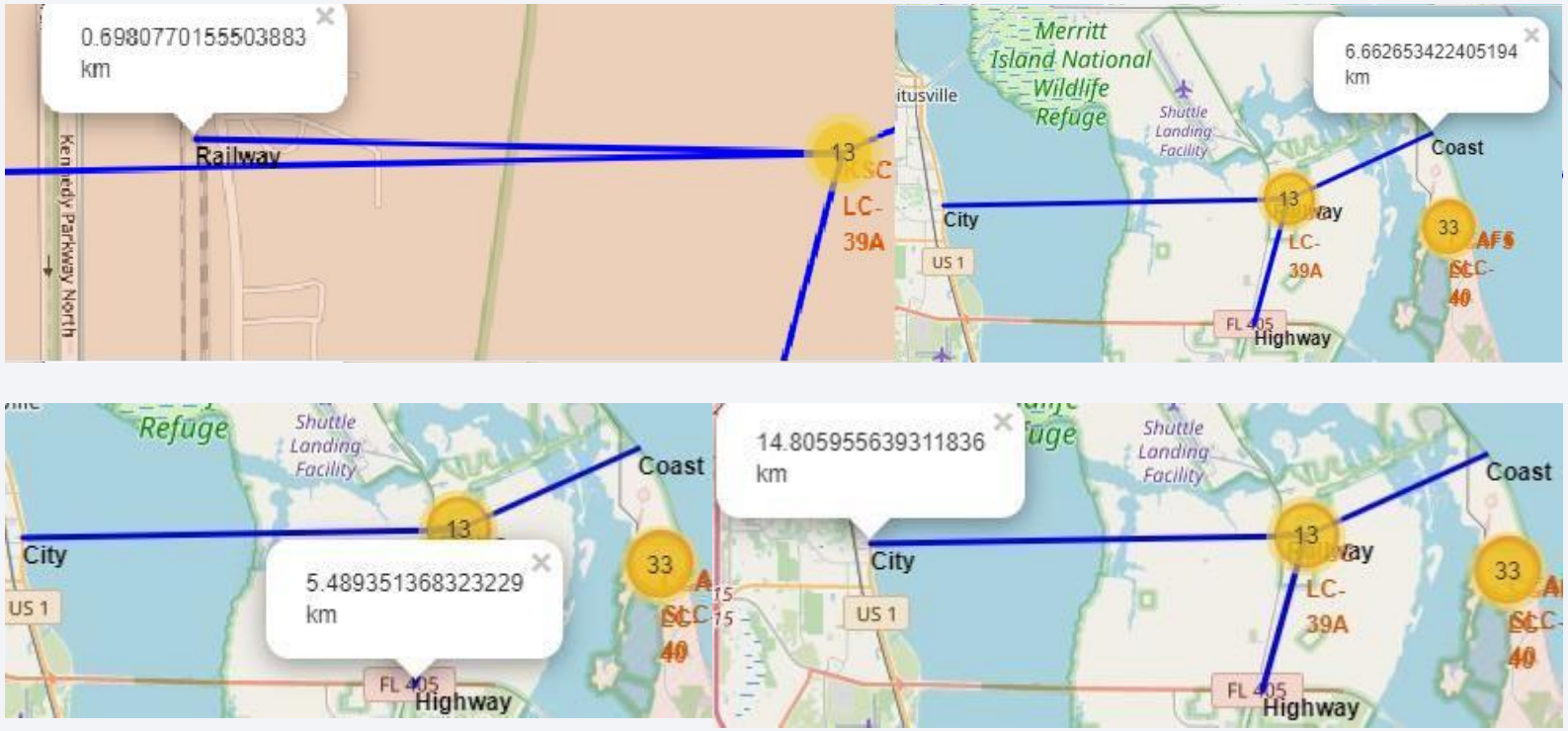
- Launch sites are only found in the two states Florida and California.
- Each launch site is in close proximity to the ocean.
- Each launch site is not too far away from the equator.

Folium Map – Outcomes of all launches from each site



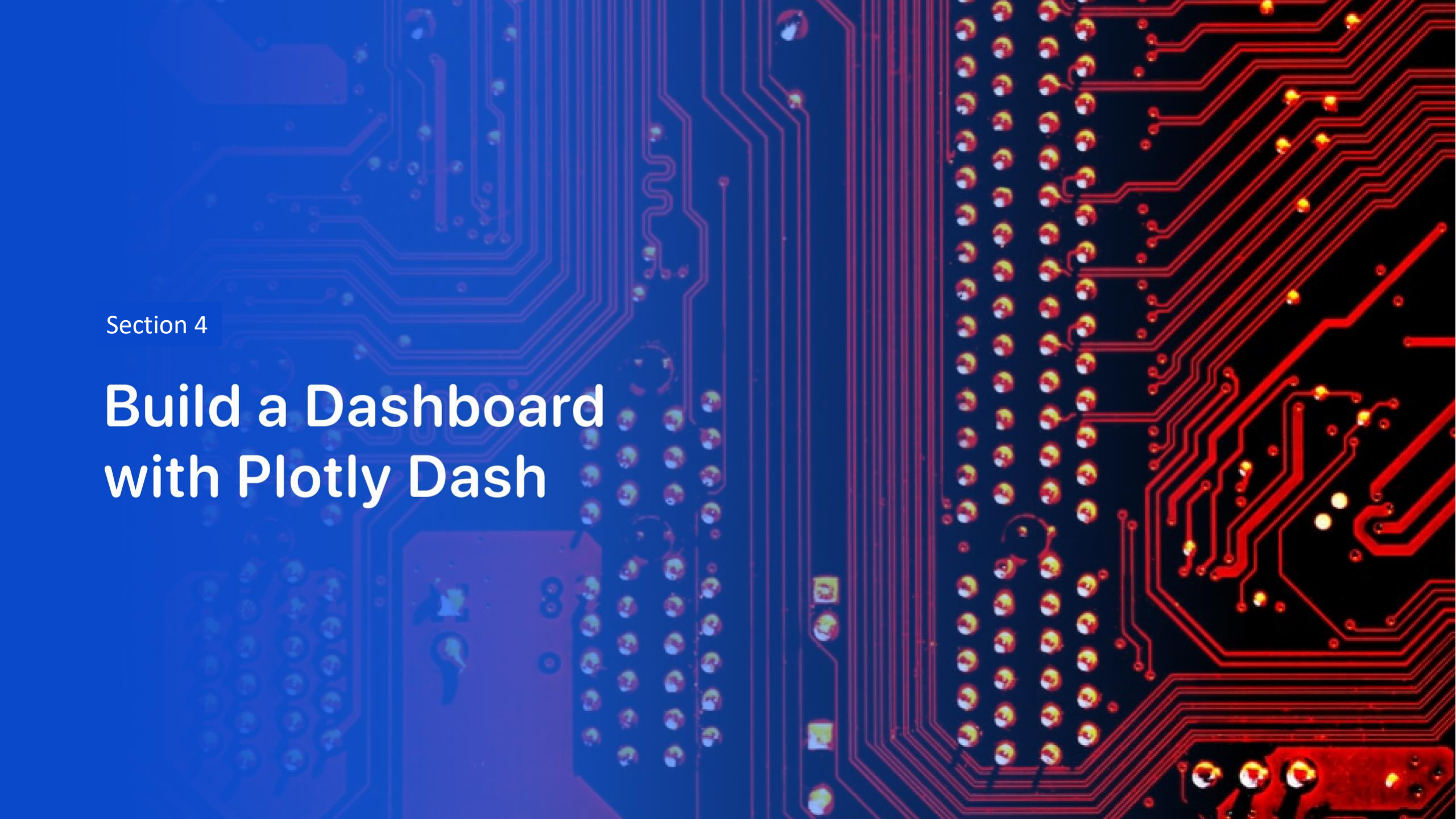
- Launches have been grouped into clusters (all launches from a given site makes one cluster).
- Each launch marker is annotated with green icons if a successful launch, and red icons for failed launches.

Folium Map – Proximity from each site to important locations

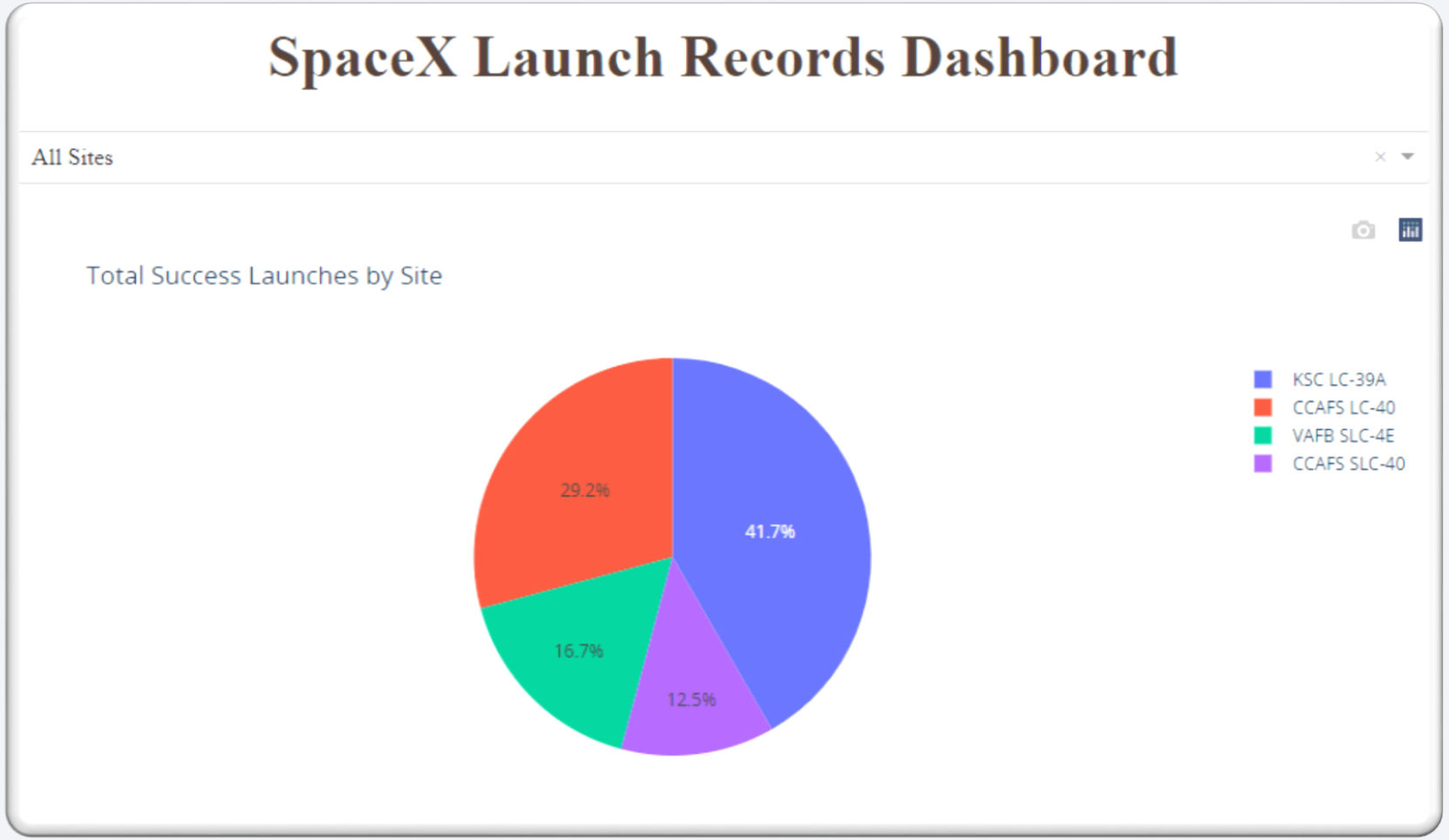


- Launch sites are in close proximity to railways.
- Launch sites are in close proximity to highways.
- Launch sites are in close proximity to the ocean.
- Launch sites are a certain distance away from cities.

Using KSC LC-39A as an example, launch sites are seen to be very close to railways and to highways for human and supply transportation requirements. Launch sites are also close to coasts and relatively far from cities so that launch failures will be crashing on either deserted areas or the ocean.

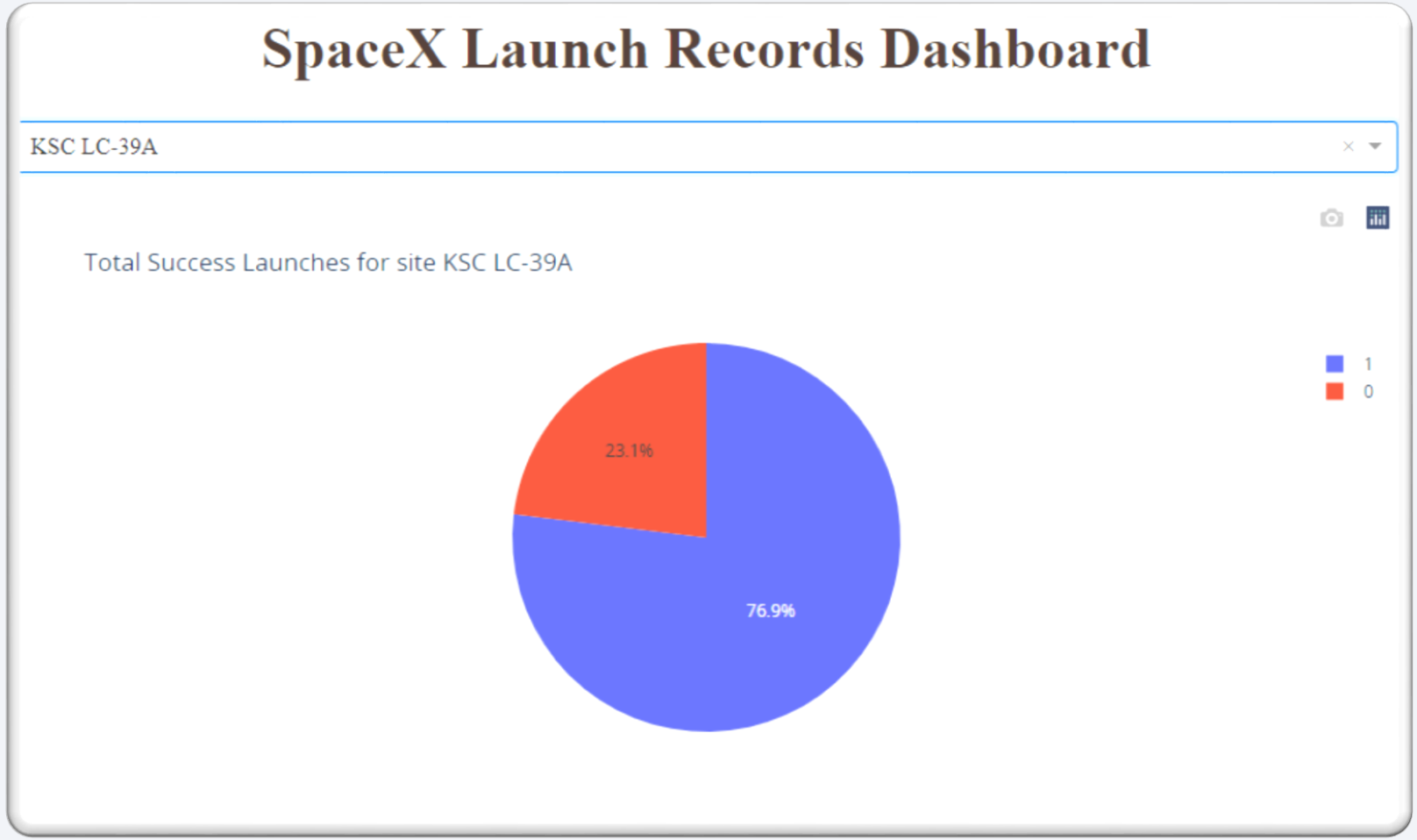

Section 4
Build a Dashboard with Plotly Dash

Interactive Dashboard – Success ratio of all sites



- The launch site KSC LC-39 A had the highest success rate for launches, with 41.7%.
- The launch site CCAFS SLC-40 had the lowest success rate for launches, with 12.5%.

Interactive Dashboard – Site with the highest Success ratio



- The launch site KSC LC-39 A had the highest success ratio among all sites
- The probability that a launch from this site will be successful is a high 76.9%.

Interactive Dashboard – Launch Outcome scatter plot for all sites

- 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.

Interactive Dashboard – Launch Outcome scatter plot for all sites

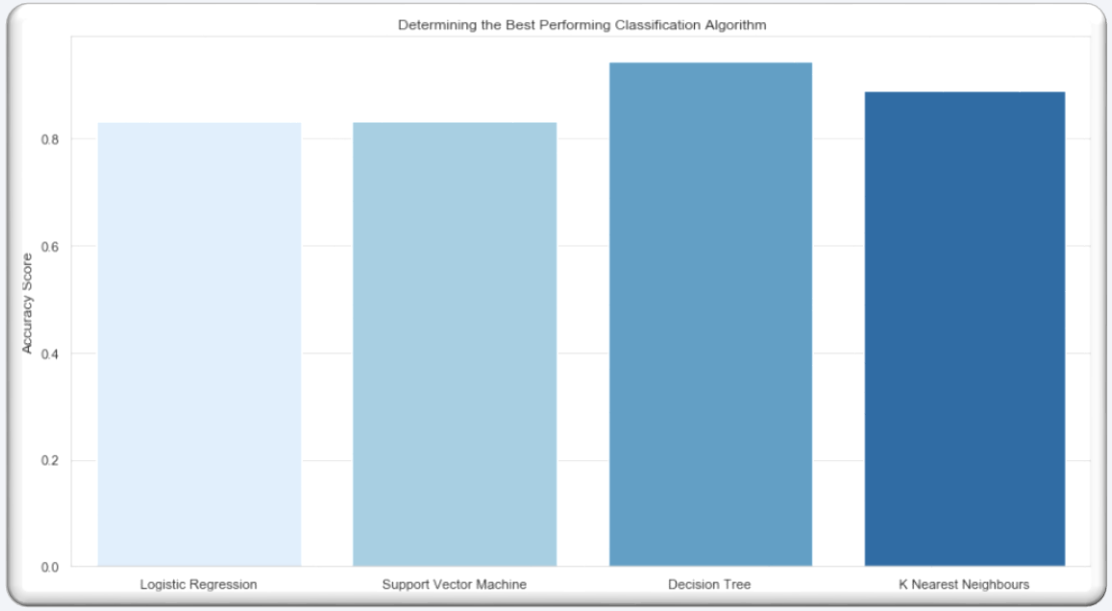
- The booster version with the highest success rate is gathered to be FT.
- The booster version with the lowest success rate is gathered to be v1.1.

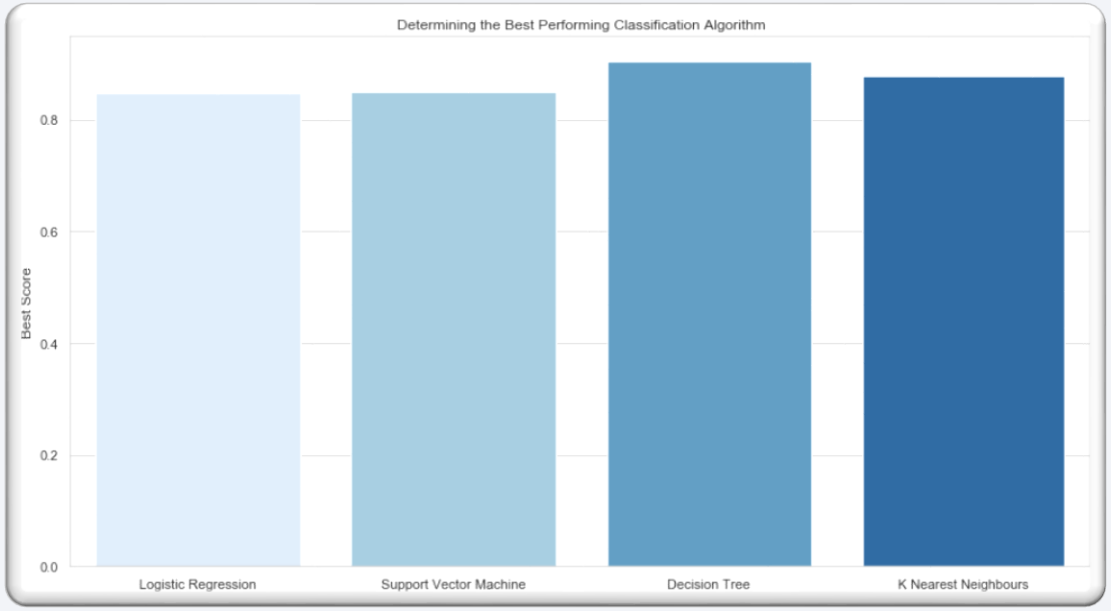
- It is also worth noting that some booster types (v1.0 and B5) have not been launched with massive payloads.

Section 5

Predictive Analysis (Classification)

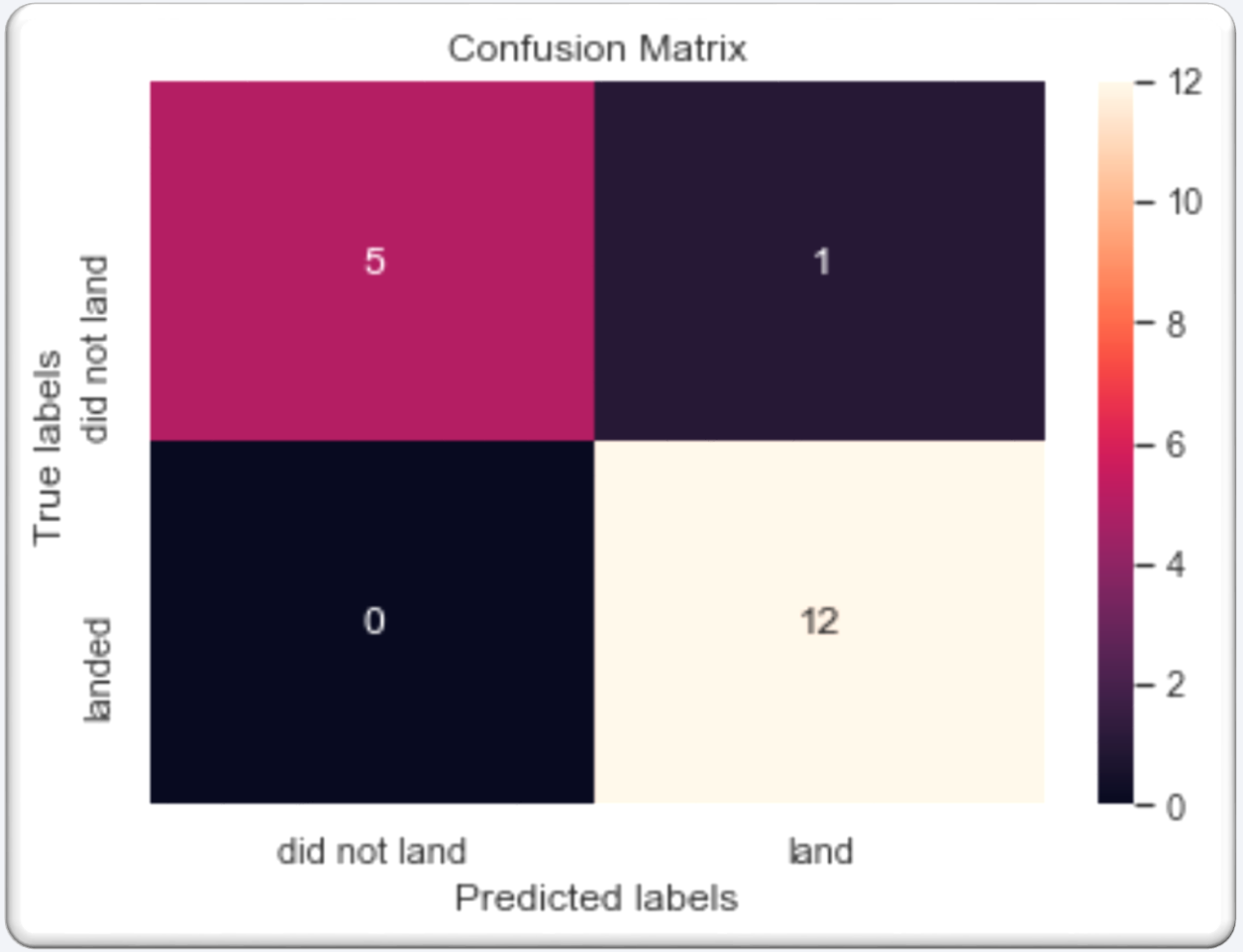
Classification Accuracy





- Plotting the Accuracy Score and Best Score for each classification algorithm we observe easily that:
  - The Decision Tree model has the highest classification accuracy
    - The Accuracy Score is 94.44%
    - The Best Score is 90.36%

Confusion Matrix



- As shown previously, best performing classification model is the Decision Tree model, with an accuracy of 94.44%.
- This is explained by the confusion matrix, which shows only 1 out of 18 total results classified incorrectly (a false positive, shown in the top-right corner).
- The other 17 results are correctly classified (5 did not land, 12 did land).

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 94.44%.

Acknowledgements

- The time and effort put in to making this course a delight to learn by all the instructors is acknowledged and applauded.

- Special Thanks to All Instructors:
  - https://www.coursera.org/professional-certificates/ibm-data-science?#instructors
  - Rav Ahuja, Alex Aklson, Aije Egwaikhide, Svetlana Levitan,
    Romeo Kienzler, Polong Lin, Joseph Santarcangelo, Azim Hirjani,
    Hima Vasudevan, Saishruthi Swaminathan, Saeed Aghabozorgi, Yan Luo

Appendix

✓ Custom functions to retrieve the required information and code to clean the data

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

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

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

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

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

Python

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, whether the core is reused, whether legs were used, the landing pad used, the block of the core (which is a number used to separate versions of cores), the number of times this specific core has been reused, and the serial of the core.

```python
# 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'])
```

Python

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

```python
# 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']:
        response = requests.get("https://api.spacexdata.com/v4/rockets/"+str(x)).json()
        BoosterVersion.append(response['name'])
```

Python

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

```python
# 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']:
        response = requests.get("https://api.spacexdata.com/v4/launchpads/"+str(x)).json()
        Longitude.append(response['longitude'])
        Latitude.append(response['latitude'])
        LaunchSite.append(response['name'])
```

Python

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

```python
# 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']:
        response = requests.get("https://api.spacexdata.com/v4/payloads/"+load).json()
        PayloadMass.append(response['mass_kg'])
        Orbit.append(response['orbit'])
```

Python

Appendix

✓ Custom functions to retrieve and store relevant information from web scraping

```
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
```

Appendix

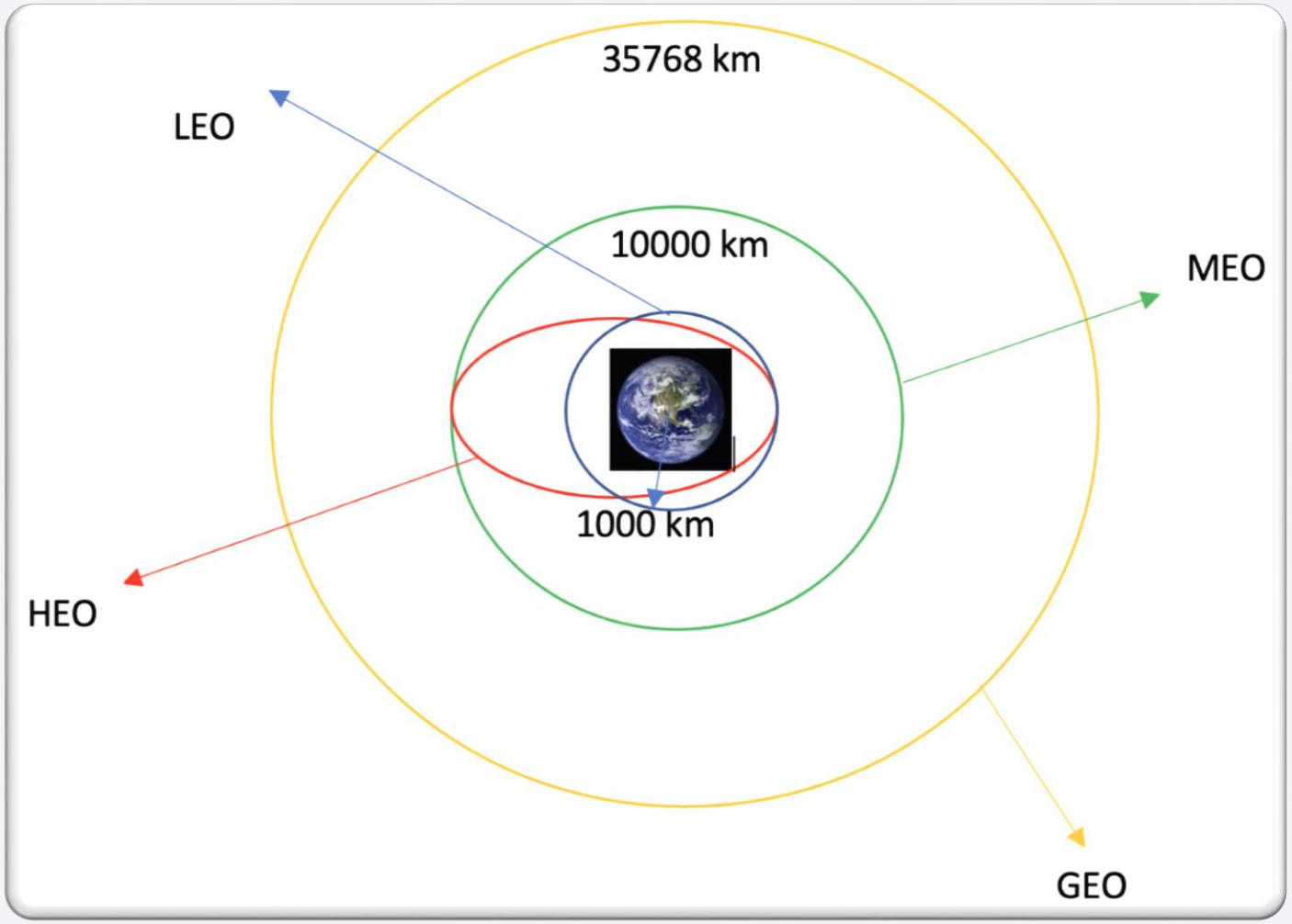
✓ Code to fill up the launch_dict values with values from the launch tables

```
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
            # Append the flight_number into launch_dict with key `Flight No.`
            launch_dict["Flight No."].append(flight_number)

            # Date value
            #Append the date into launch_dict with key `Date`
            datatimelist=date_time(row[0])
            date = datatimelist[0].strip(',')
            launch_dict["Date"].append(date)

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

            # Booster version
            #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)
```

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

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

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

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

            # Customer
            #Append the customer into launch_dict with key `Customer`
            if row[6].a != None:
                customer = row[6].a.string
            else:
                customer = 'None'

            launch_dict['Customer'].append(customer)

            # Launch outcome
            #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)

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


print(f"Flight Number: {flight_number}, Date: {date}, Time: {time} \n \
Booster Version {bv}, Launch Site: {launch_site} \n \
Payload: {payload}, Orbit: {orbit} \n \
Customer: {customer}, Launch Outcome: {launch_outcome}\
Booster Landing: {booster_landing} \n \
*** ")
```

Appendix

✓ Widely used orbit types for SPACE – X rockets



Thank you!