Winning Space Race with Data Science

Magnus
27/06/2023

Outline

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

Executive Summary

Methodology

- Data Collection (API & Web Scraping)
- Data Wrangling
- Exploratory Data Analysis (SQL, Pandas, Matplotlib)
- Interactive Visual Analytics Folium & Plotly Dash)
- Predictive Analysis (Scikit-Learn, confusion matrices, performance assessment of classification models)

Summary of Key Results

- As the number of flights increase, the rate of success at a launch site increases.
- Orbit types ES-L1, GEO, HEO, and SSO, have the highest success rate (100%)
- 41.7% of the successful launches has taken place at Launch site KSC LC-39 A. This site also has the highest rate of successful launches (76.9%)
- The success for payloads over 4,000 kg is lower than that for lower-weight payloads.
- The best performing classification model is the Decision Tree model, with an accuracy of 94.44%.

Github project file: https://github.com/MaggiePI/datascience/tree/main/IBM%20Datacience%20Professional%20Certificate%20Captsone

Introduction

- Launching a Space X Falcon 9 rocket costs around $62m whereas launches by other providers cost upwards of $165m. Space X's savings largely stems from the company's ability to land, and then re-use the first stage of the rocket.
- This project seeks to assess the probability that SpaceX's Falcon 9 first stage will land successfully
- By understanding this probability, we can determine the cost of a launch, and use this information to assess pricing for Space Y launches.

Section 1

Methodology

Methodology Summary

Data collection:

- Making GET requests to the SpaceX REST API
- Web Scraping

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 the following:
  - 0 when the booster did not land successfully
  - 1 when the booster did land successfully

Exploratory data analysis (EDA) using visualization and SQL:

- Using SQL queries to manipulate and evaluate the SpaceX dataset
- Using Pandas and Matplotlib to visualize relationships between variables, and determine patterns

Interactive visual analytics (Folium and Plotly Dash):

- Geospatial analytics using Folium
- Creating an interactive dashboard using Plotly Dash

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.

Github: https://tinyurl.com/bdzyeubj

1

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

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

3

- Create a Pandas DataFrame from the constructed dictionary dataset

4

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

Data Collection – Web Scraping

Web scraping to collect Falcon 9 historical launch records from a Wikipedia page titled List of Falcon 9 and Falcon Heavy launches.

GitHub: https://tinyurl.com/bdcwrp2f

1
- Request the HTML page from the static URL
- Assign the response to an object

2
- Create a BeautifulSoup object from the HTML response object
- Find all tables within the HTML page

3
- Collect all column header names from the tables found within the HTML page

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

5
- Convert the dictionary to a Pandas DataFrame ready for export

Data Wrangling

Context:

- The SpaceX dataset contains several Space X launch facilities, and each location is in the LaunchSite column.
- Each launch aims to a dedicated orbit, and some of the common orbit types are shown in the figure below. The orbit type 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

Data Wrangling (Continued)

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:

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.

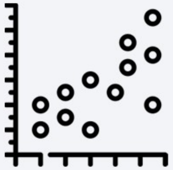
Github: https://tinyurl.com/3a8uvmby

EDA with Data Visualization

SCATTER CHARTS

Scatter charts 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

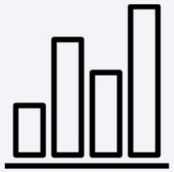


Scatter charts are useful to observe relationships, or correlations, between two numeric variables.

BAR CHART

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 axis, and are generally used to show the change of a variable over time.

EDA with SQL

Summary of performed SQL queries:

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: https://tinyurl.com/mrx86mxj

Geospatial Analysis with Folium

The following steps were taken to visualize the launch data on an interactive map:

1. 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

2. 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.

3. 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.

Build a Dashboard with Plotly Dash

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

1. Pie chart (`px.pie()`) showing the total successful launches per site
   - 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

2. Scatter graph (`px.scatter()`) to show the correlation between outcome (success or not) and 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)

The following steps were taking 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

Identifying 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

Interactive Analytics

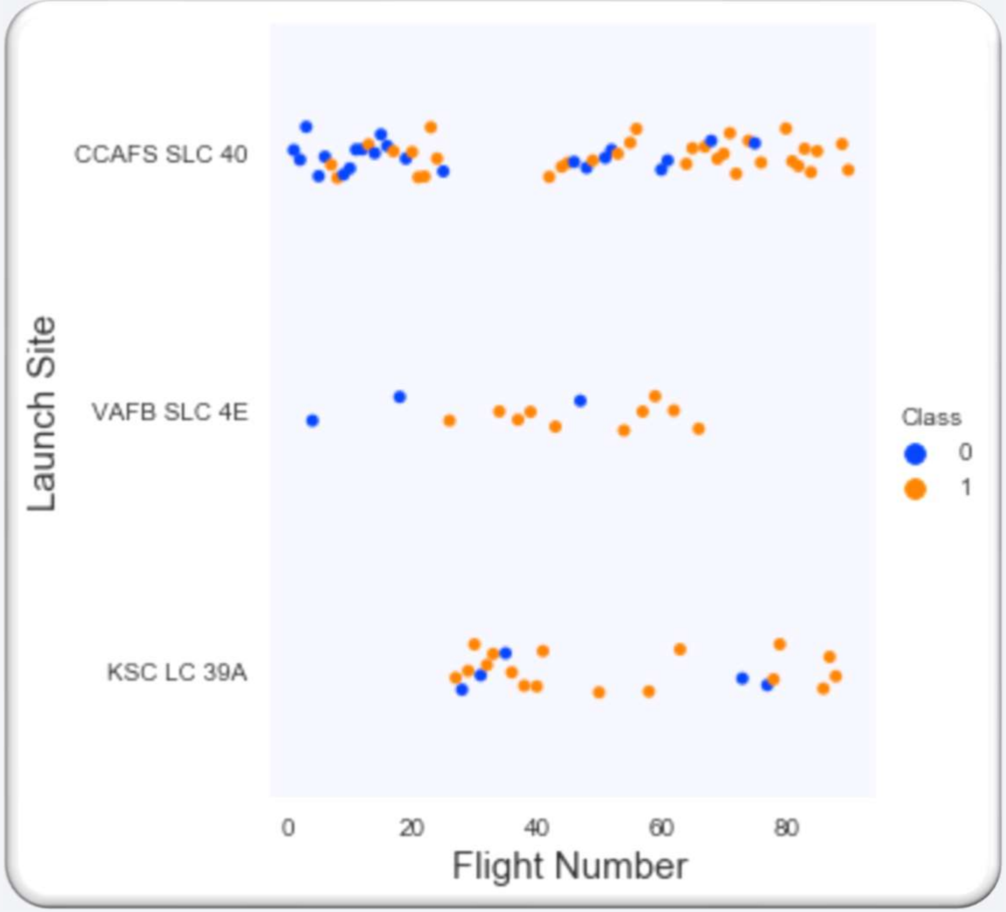
Predictive Analysis

Section 2

Insights drawn from EDA

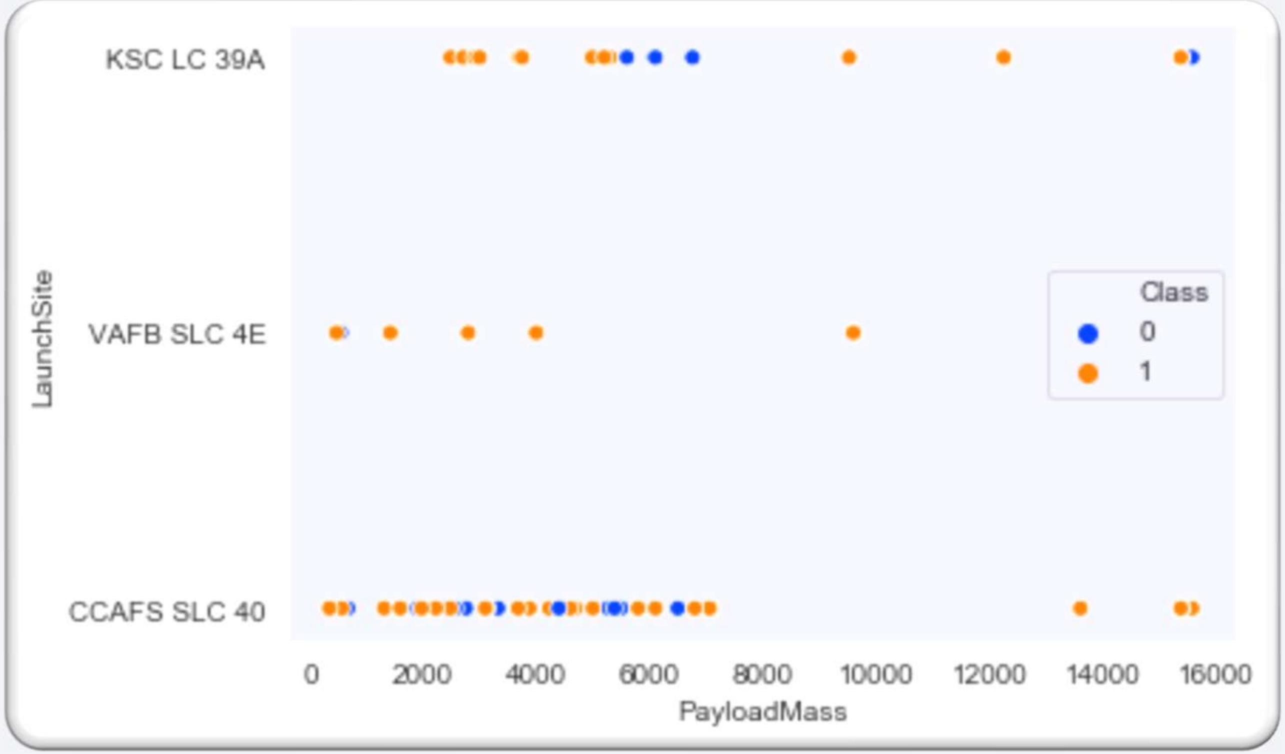
Flight Number vs. Launch Site

- 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).



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).



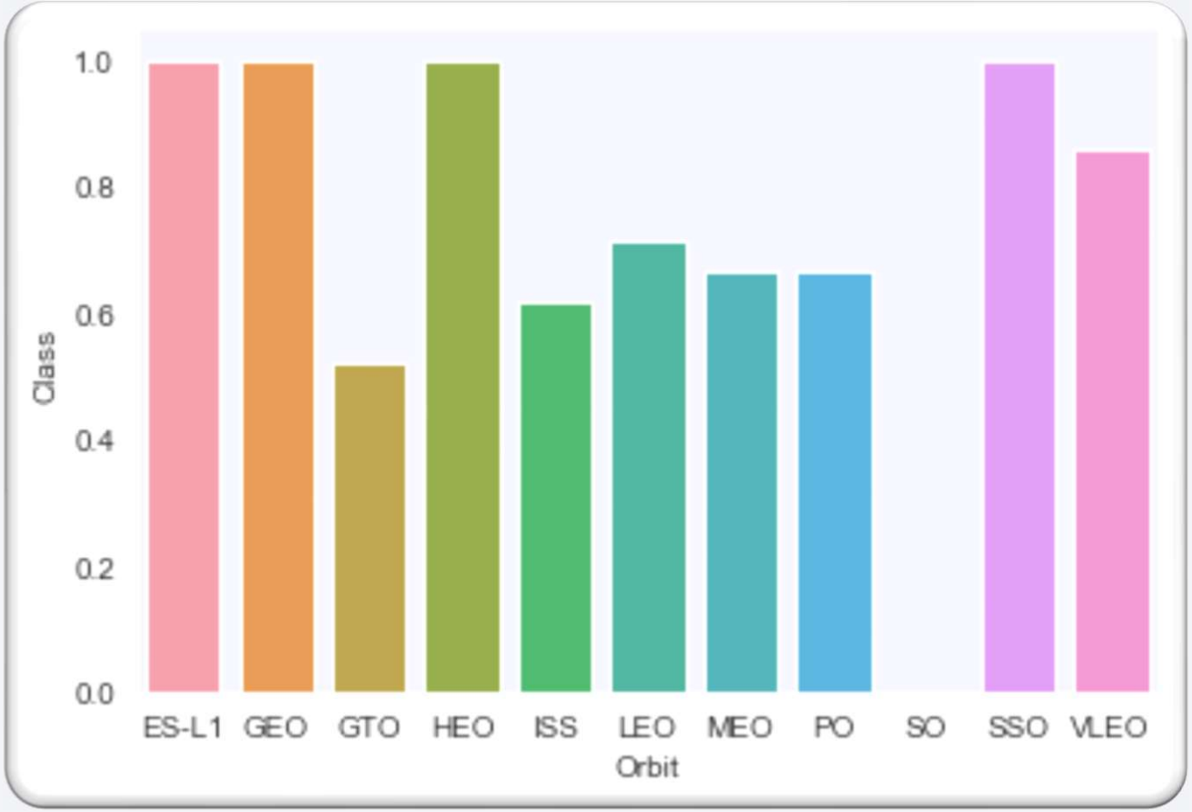
Success Rate vs. Orbit Type

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:

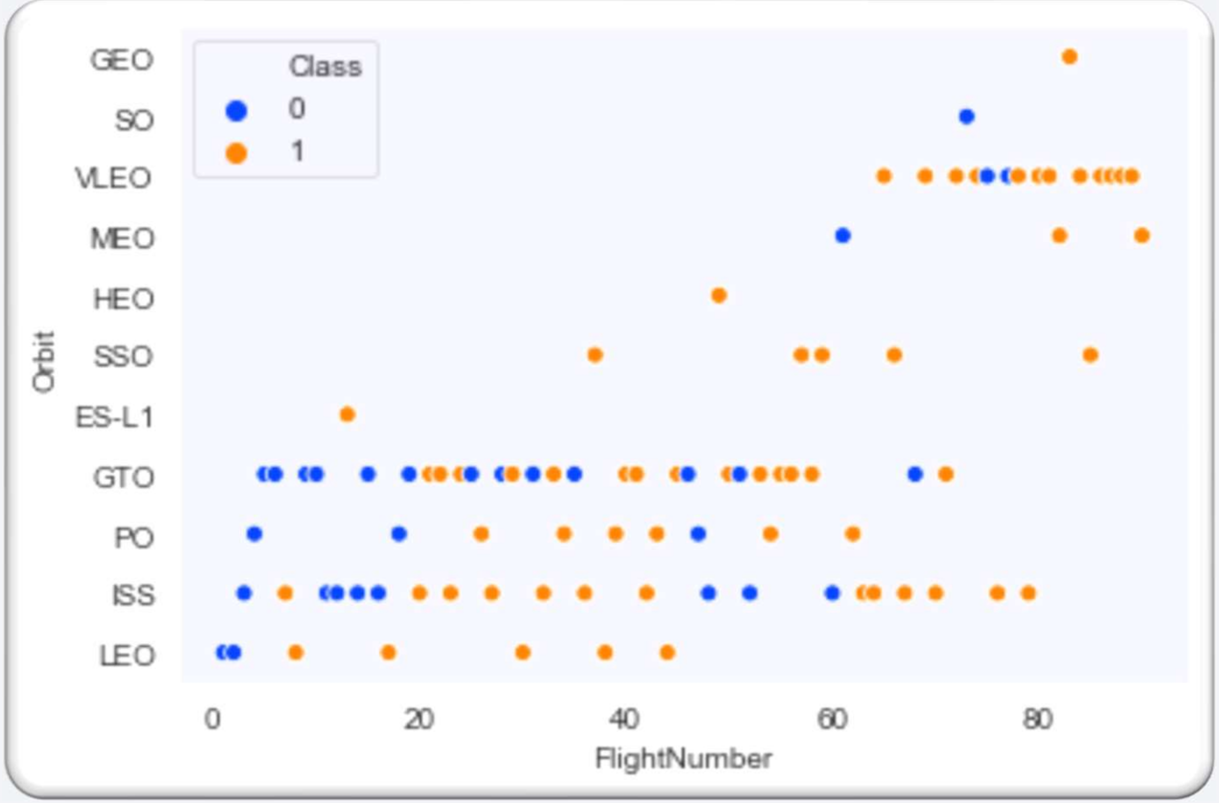
- SO (Heliocentric Orbit)



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).



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

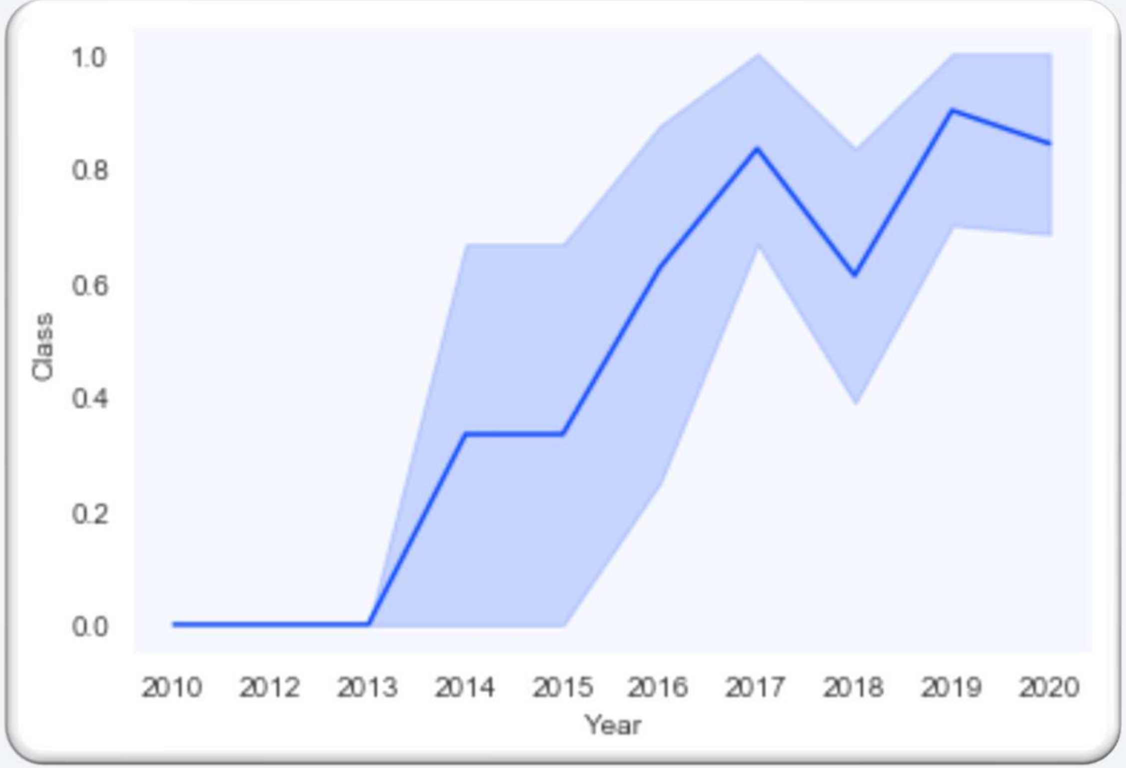
- For GTO, the relationship between payload mass and success rate is unclear.

- VLEO (Very Low Earth Orbit) launches are associated with heavier payloads, which makes intuitive sense.

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.



All Launch Site Names

- Find the names of the unique launch sites.

```
%sql SELECT DISTINCT(LAUNCH_SITE) FROM SPACEXTBL;
```

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

The word DISTINCT 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 LAUNCH_SITE FROM SPACEXTBL WHERE LAUNCH_SITE LIKE 'CCA%' LIMIT 5;
```

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

- `LIMIT 5` fetches only 5 records, and the `LIKE` keyword is used with the wild card `‘CCA%’` to retrieve string values beginning with ‘CCA’.

Total Payload Mass

- Calculate the total payload carried by boosters from NASA.

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

TOTAL_PAYLOAD_MASS
45596.0

- The SUM keyword is used to calculate the total of the LAUNCH column, and the SUM keyword (and the associated condition) filters the results to only 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_) AS AVERAGE_PAYLOAD_MASS FROM SPACEXTBL WHERE BOOSTER_VERSION = 'F9 v1.1';
```

AVERAGE_PAYLOAD_MASS
2928.4

- The AVG keyword is used to calculate the average of the PAYLOAD_MASS__KG_ column, and the WHERE keyword (and the associated condition) filters 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) AS First_Successful_Landing FROM SPACEXTBL WHERE LANDING_OUTCOME = 'Success (ground pad)';
```

First_Successful_Landing
22/12/2015

- The MIN keyword is used to calculate the minimum of the DATE column, i.e. the first date, and the WHERE keyword (and the associated condition) filters the results to only the successful ground pad landings.

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_ BETWEEN 4000 AND 6000);
```

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

- The WHERE keyword is used to filter the results to include only those that satisfy both conditions in the brackets (as the AND keyword is also used). The BETWEEN keyword allows for 4000 < x < 6000 values to be selected.

Total Number of Successful and Failure Mission Outcomes

- Calculate the total number of successful and failure mission outcome.

```
%sql SELECT MISSION_OUTCOME, COUNT(MISSION_OUTCOME) AS TOTAL_NUMBER FROM SPACEXTBL GROUP BY MISSION_OUTCOME;
```

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

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

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);
```

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

- A subquery is used here. The `SELECT` statement within the brackets finds the maximum payload, and this value is used in the `WHERE` condition. The `DISTINCT` keyword is then used to retrieve only distinct /unique booster versions.

2015 Launch Records

- List the failed landing outcomes for drone ships in 2016 along with their booster versions, and launch site names.

```
%sql SELECT substr(Date,4,2) as month, DATE,BOOSTER_VERSION, LAUNCH_SITE, [Landing_Outcome] FROM SPACEXTBL where [Landing_Outcome] = 'Failure (drone ship)' and substr(Date,7,4)='2015';
```

month	Date	Booster_Version	Launch_Site	Landing_Outcome
10	01/10/2015	F9 v1.1 B1012	CCAFS LC-40	Failure (drone ship)
04	14/04/2015	F9 v1.1 B1015	CCAFS LC-40	Failure (drone ship)

- The WHERE keyword is used to filter the results for only failed landing outcomes, AND to limit results to 2015.

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

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

```
%sql SELECT [Landing_Outcome], count(*) as count_outcomes FROM SPACEXTBL WHERE DATE between '04-06-2010' and '20-03-2017' group by [Landing_Outcome] order by count_outcomes DESC;
```

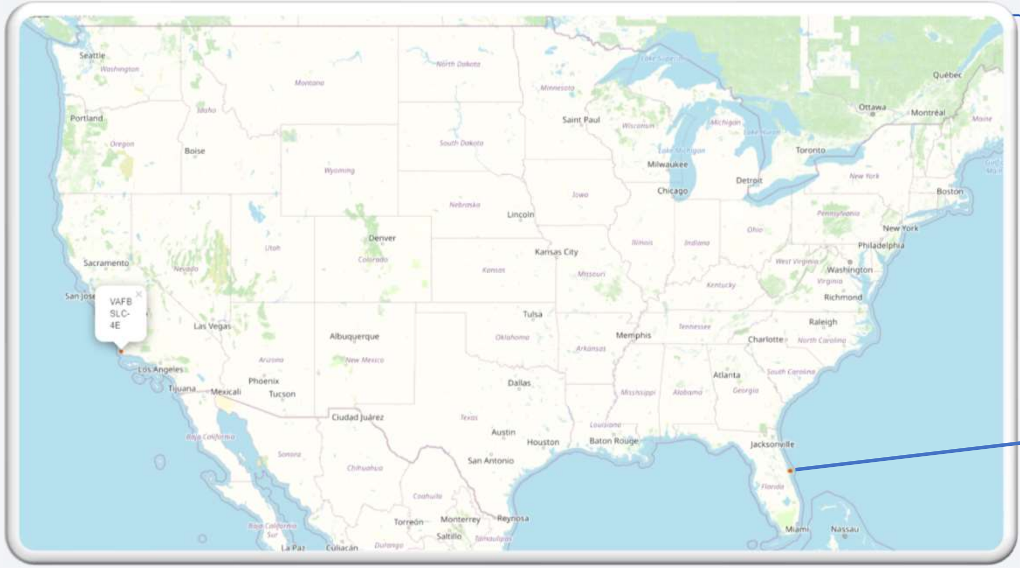
The WHERE keyword is used with the BETWEEN keyword to filter the results to dates only within those specified. The results are then grouped and ordered, using the keywords GROUP BY and ORDER BY, respectively, where DESC is used to specify the descending order.

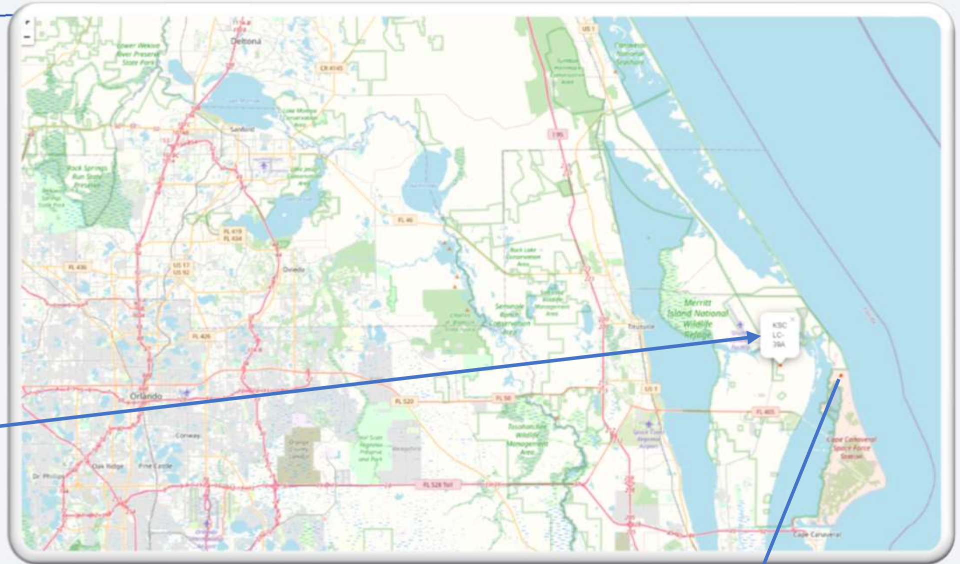
Landing_Outcome	count_outcomes
Success	20
No attempt	10
Success (drone ship)	8
Success (ground pad)	7
Failure (drone ship)	3
Failure	3
Failure (parachute)	2
Controlled (ocean)	2
No attempt	1

Section 3

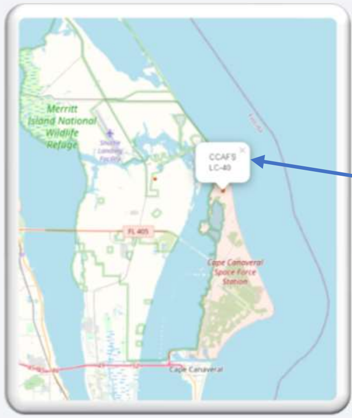
Launch Sites Proximities Analysis

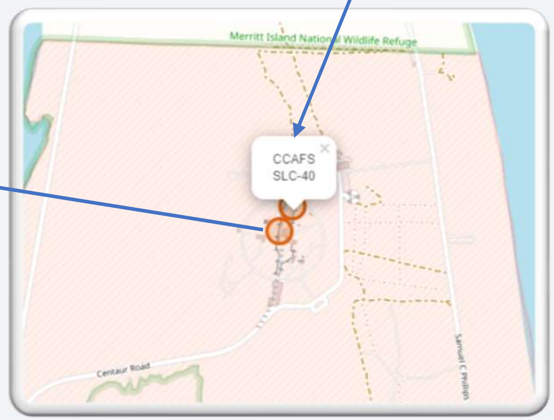
Map View of All Launch Sites



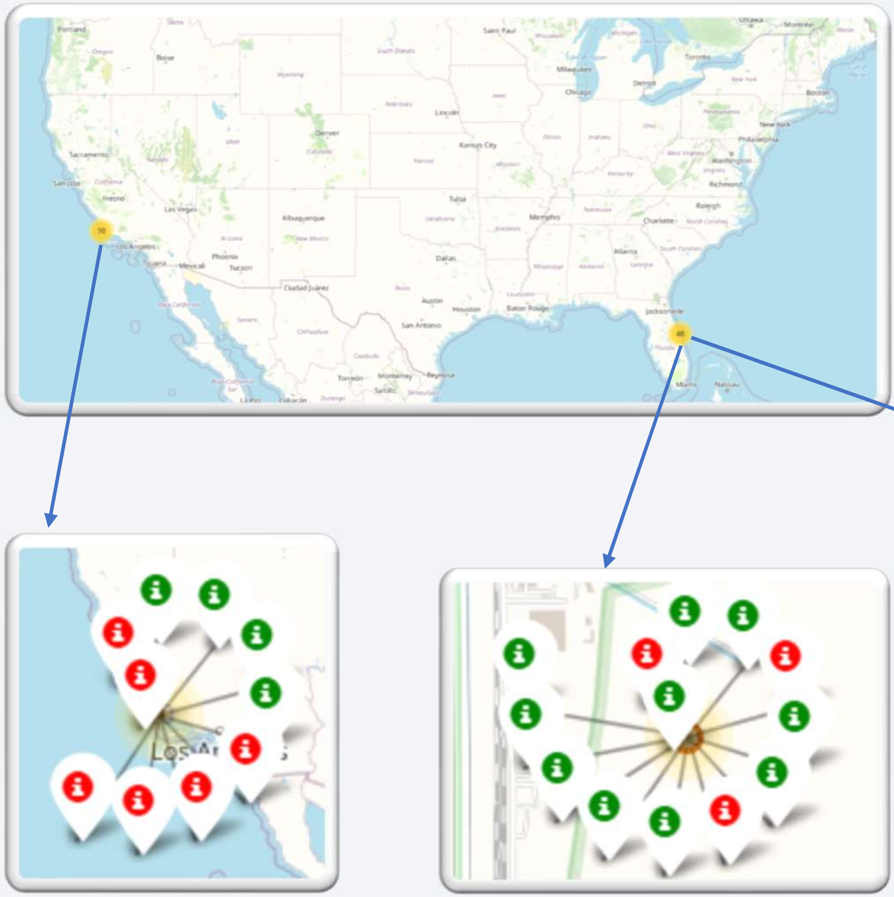


- All SpaceX launch sites are on coasts of the United States of America, specifically Florida and California.

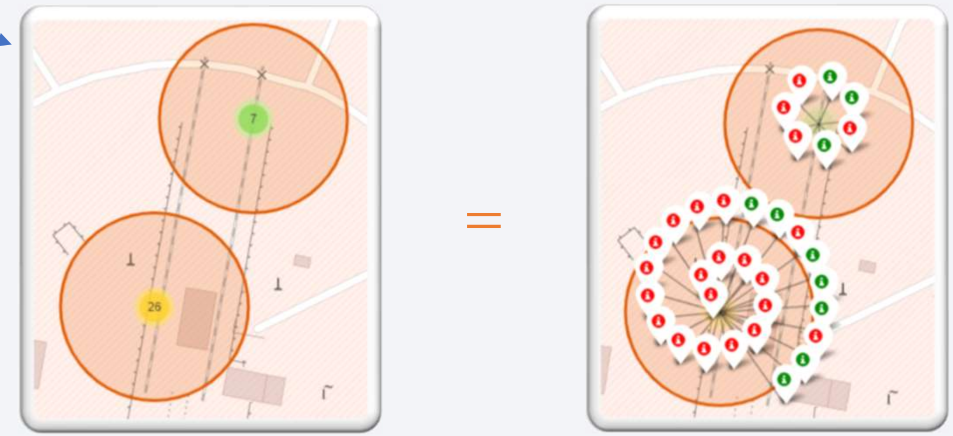




Successful/Failed Launches For Each Site

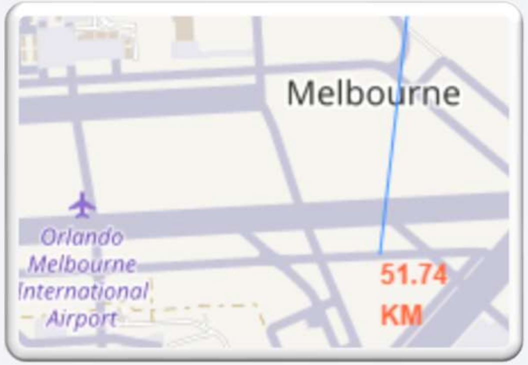


- Launches have been grouped into clusters, and annotated with green icons for successful launches, and red icons for failed launches.



Proximity of Launch Sites to Other Points of Interest

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?

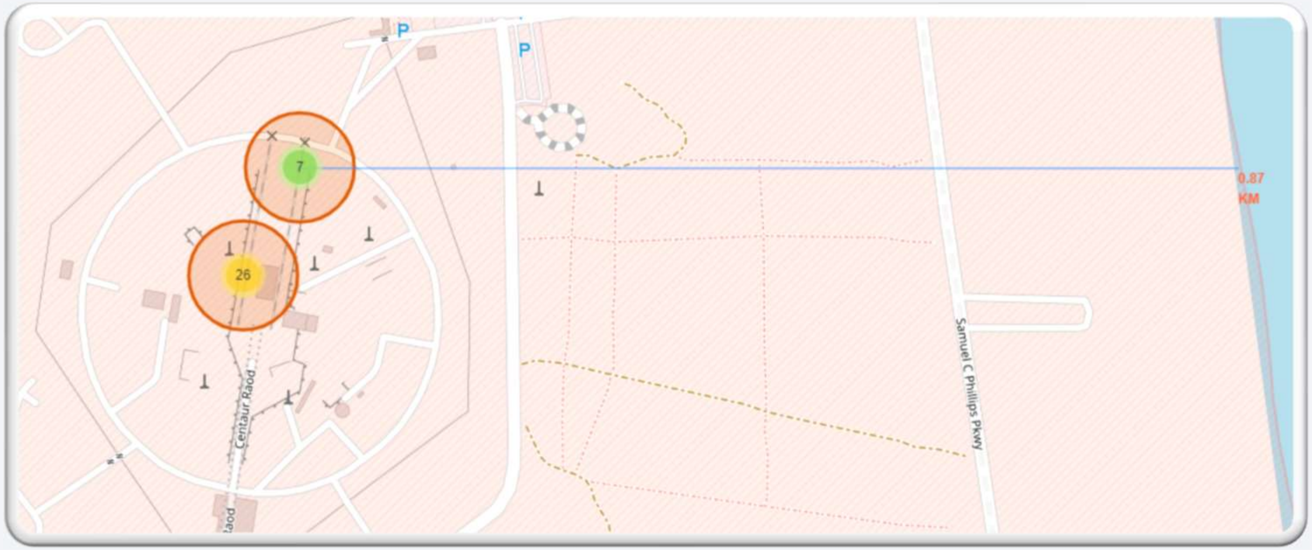
- YES. The nearest highway is only 0.59km away.

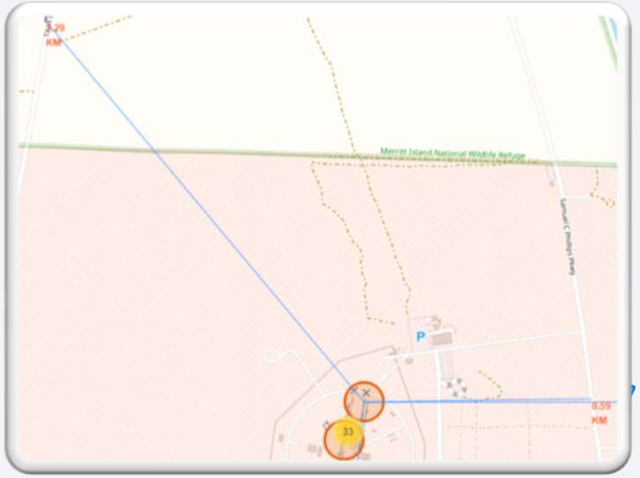
Are launch sites in close proximity to railways?

- 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.

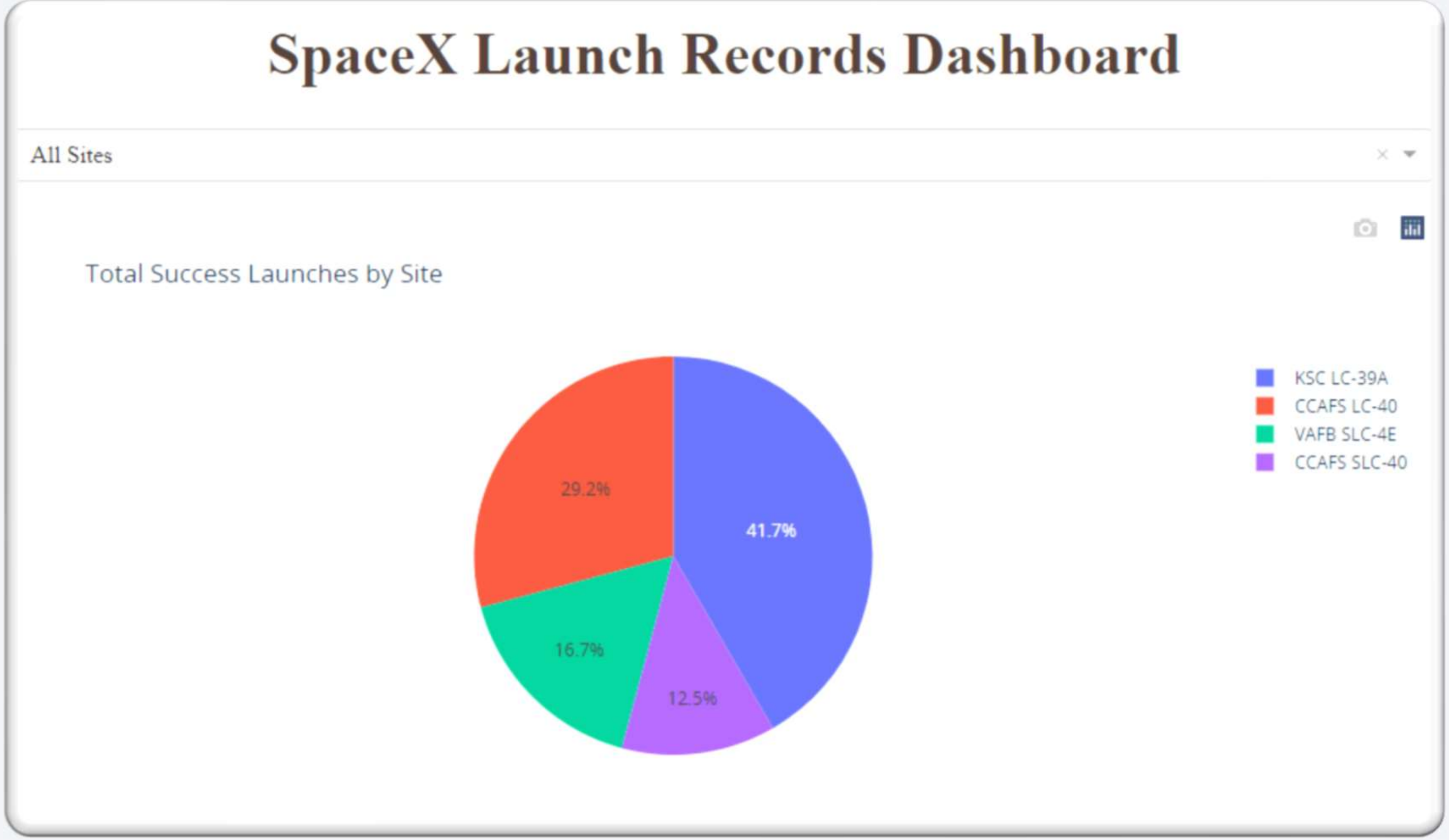




Section 4

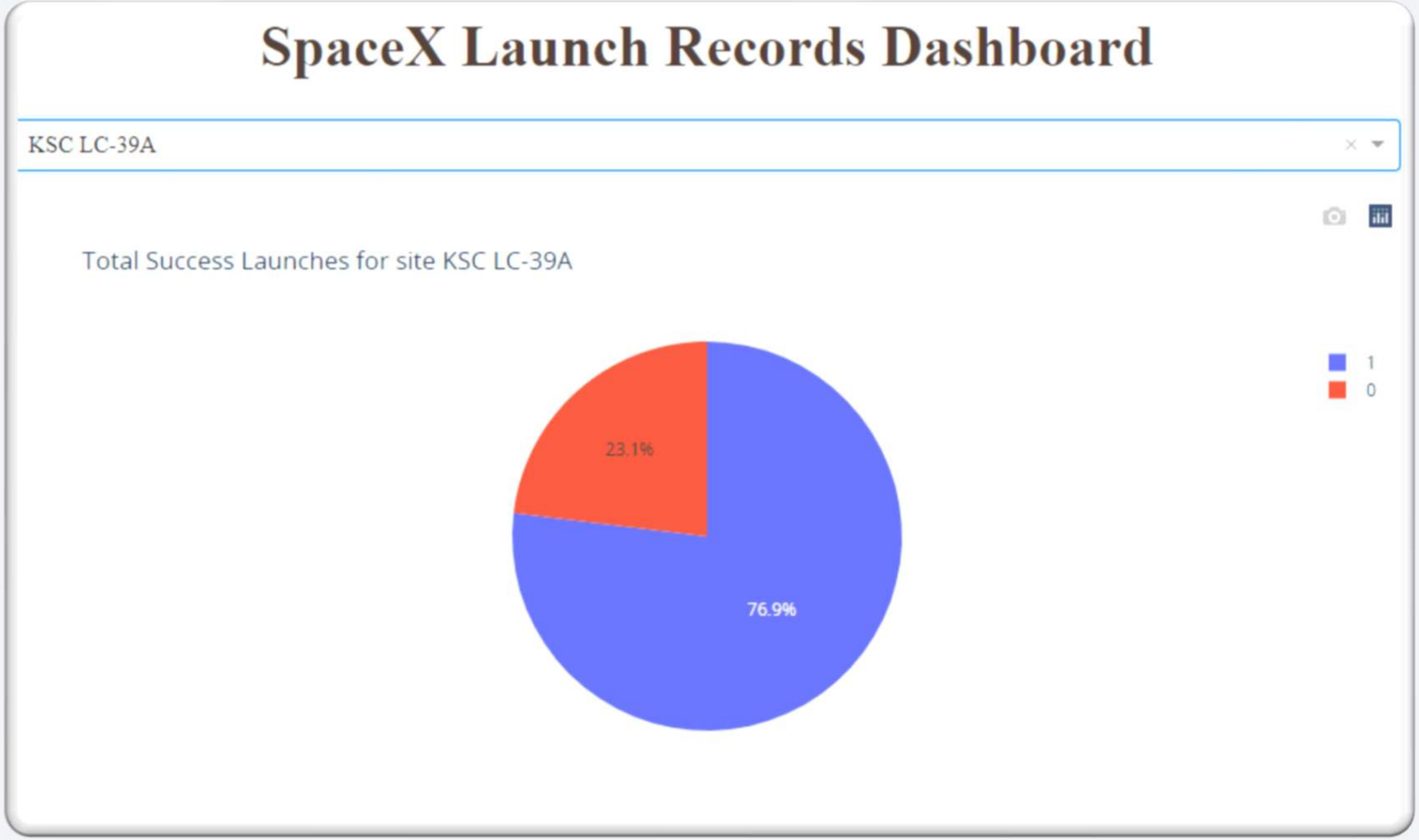
Build a Dashboard with Plotly Dash

Count of Successful Launches – All Sites



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

Launch Site With The 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

- 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 (small payloads)
  - 4000 – 10000 kg (massive payloads)
- These two plots show that the success for massive payloads is lower than for small payloads.
- 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)

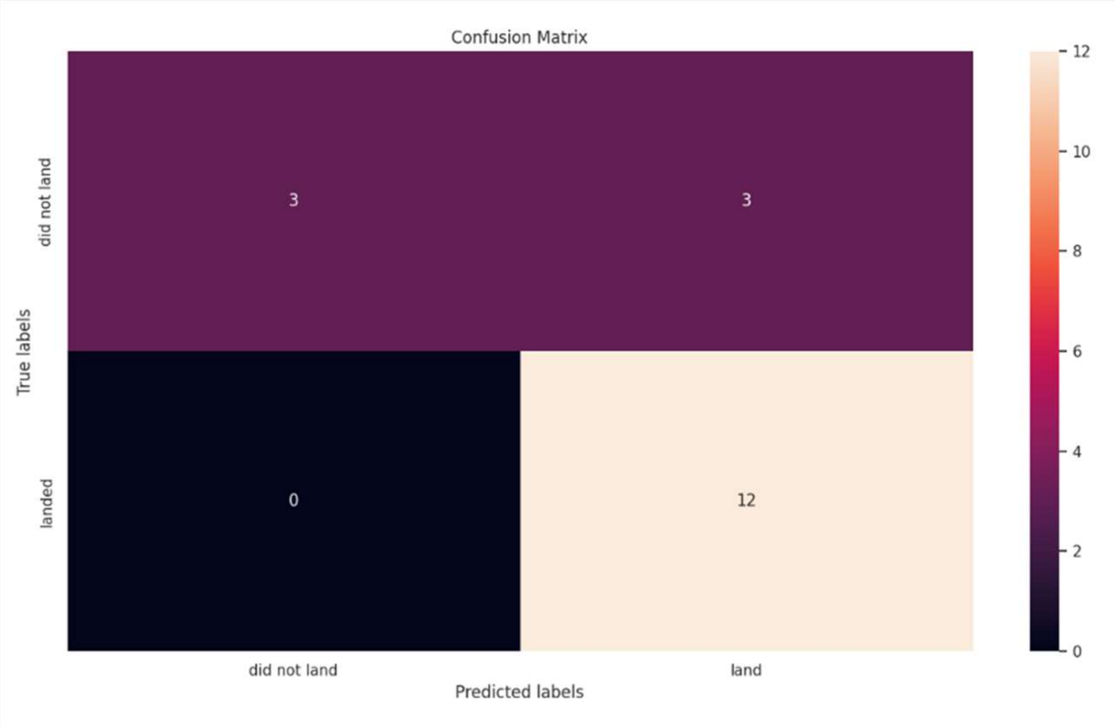
Classification Accuracy

Plotting the Accuracy Score and Best Score for each classification algorithm produces the following result:

	Algorithm	Accuracy Score	Best Score
0	Logistic Regression	0.833333	0.846429
1	Support Vector Machine	0.833333	0.848214
2	Decision Tree	0.833333	0.916071
3	K Nearest Neighbours	0.722222	0.876786

- The Decision Tree model has the highest classification accuracy
  - The Accuracy Score is 83.33%
  - The Best Score is 91.61%

Confusion Matrix



- As shown previously, best performing classification model is the Decision Tree model, with an accuracy of 91.61%.
- This is explained by the confusion matrix, which shows only 3 out of 18 total results classified incorrectly (falses positive, shown in the top-right corner).
- The other 15 results are correctly classified (3 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 lighter payloads**.
- **The best performing classification model is the Decision Tree model, with an accuracy of 94.44%.**

Appendix

REST API

- Custom functions to retrieve the required information
- Custom logic to clean the data

```
# 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)]
```

Appendix

REST API

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 list
def getBoosterVersion(data):
    for x in data['rocket']:
        if x:
            response = requests.get("https://api.spacexdata.com/v4/rockets/"+str(x)).json()
            BoosterVersion.append(response['name'])
```

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

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

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

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

Appendix

REST API

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

Appendix

Web Scraping

- Custom functions for web scraping
- Custom logic to fill up the launch_dict values with values from the launch tables

```python
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')
```

```python
# if number save cells in a dict
        if flag:
            extracted_row += 1
            # Flight Number value
            # TODO: Append the flight_number into launch_dict with key `Flight No.`
            # print(flight_number)
            launch_dict["Flight No."].append(flight_number)

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

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

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

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

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

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

            launch_dict['Customer'].append(customer)

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

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

Web Scraping

```python
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

- Include any relevant assets like Python code snippets, SQL queries, charts, Notebook outputs, or data sets that you may have created during this project

Thank you!