
# Winning Space Race with Data Science

Ahmed Nasreldeen
16/10/2023

# Outline

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

# Executive Summary

## Summary of methodologies

This project follows these steps:

a. Data Collection.

b. Data Wrangling.

c. Exploratory Data Analysis.

d. Interactive Visual Analytics

e. Predictive Analysis

## Summary of all results

Project outputs:

1. Exploratory Data Analysis.
2. Geospatial Analysis.
3. Interactive Dashboard
4. Predictive Analysis Of Classification Models.

# Introduction

- SpaceX launches Falcon 9 rockets at a cost of around$62m. This is considerably cheaper than other providers (which usually cost upwards of $165m), and much of the savings are because SpaceX can land, and then re-use the first stage of the rocket

- if we can make predictions on whether the first stage will land, we can determine the cost of a launch, and use this information to assess whether or not an alternate company should bid and SpaceX for a rocket launch.

Section 1

# Methodology

# Methodology

## Executive Summary

**Data collection methodology:**

- Using SpaceX REST API to GET Requests.
- Web Scraping.

**Perform data wrangling**

- Using some helping methods like:
  - .fillna() :to fill nulls cells with measurements of central tendency.
  - .count_values() : to show frequency of event like:
    - Number of Launches on each site

**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-Proces the data
  - Split data
  - Find Hyper-Parameter Using Grid SearchCV
- Assessing Accuracy Of each Model.
- Confusion matrices for each classification algorithm

# Data Collection – SpaceX API

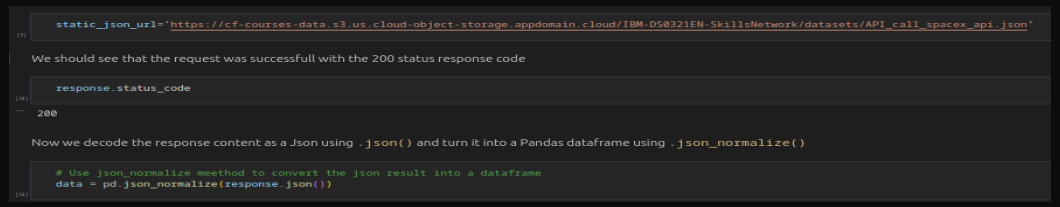
- Make GET Request to retrieve data From API
- Covert response to .json then to Pandas Data Frame.

- Using custom logic to clean the data
- Define lists to store data in
- Call custom function to fill these lists with retrieved data
- Use these lists as values in dictionary to build Dataset

- Create pandas data frame form dataset

- Filter dataframe to only include falcon9
- reset flightnumber column
- Replace null values with mean of columns

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




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

```
# 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()})
```



GitHub

# Data Collection - Scraping

Web Scraping to collect Falcon9 historical launch records From Wikipedia page

- Make GET Request to get HTML Page

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

- Create a Beautiful Soup Object From response
- Find all tables within html page

```
soup = BeautifulSoup(data, 'html5lib')
html_tables = soup.find_all('table')
```

- Collect all column header names from tables

```
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 is None and len(name) > 0):
        column_names.append(name)
```

- Use column names as key to build Dictionary
- use custom function to sparse all tables to fill dictionary values

```
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']=[]
```

- Convert Dictionary to Pandas DataFrame

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



GitHub

# Data Wrangling

- Context:
  - The SpaceX dataset contains several SpaceX 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:
    - Number of launches on each site
    - Number and occurrence of each orbit
    - Number and occurrence of landing outcome per orbit type

```
Apply value_counts() on column La
f['LaunchSite'].value_counts()

 SLC 40     55
C 39A       22
SLC 4E      13
 LaunchSite, dtype: int64
```

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

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

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

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

GitHub

# EDA with Data Visualization

| Scatter Charts | Bar Charts | Line Charts |
| --- | --- | --- |
| Scatter Chart Were produced to visualize the relationship between:<br>• Flight Number And Launch Site<br>• Payload And Launch Site<br>• Orbit Type And Flight Number<br>• Orbit Type And Payload | Bar Chart was produced to visualize the relationships between:<br>• Success Rate and Orbit Type | Line Charts were produced to visualize the relationships between:<br>• Success Rate and Year |
| Scatter Charts are useful to observe the relationships between to numeric variables | Bar Chart are used to compare a numerical variable to a categorical variable | Line Chart contain numerical values on both axes and are general used to show the changes of variable over time. |

GitHub

# EDA with SQL

To gather some information about dataset, some SQL queries were used

=> Queries like:

- Display the names of the unique launch sites in the space mission
- Display 5 records where launch sites begin with the string 'CCA'
- Display the total payload mass carried by boosters launched by NASA (CRS)
- Display the average payload mass carried by booster version F9 v1.1
- List the date when the first successful landing outcome on a ground pad was achieved
- List the names of the boosters which had success on a drone ship and a payload mass between 4000 and 6000 kg
- List the total number of successful and failed mission outcomes
- List the names of the booster versions which have carried the maximum payload mass
- List the failed landing outcomes on drone ships, their booster versions, and launch site names for 2015
- Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order

GitHub

# Build an Interactive Map with Folium

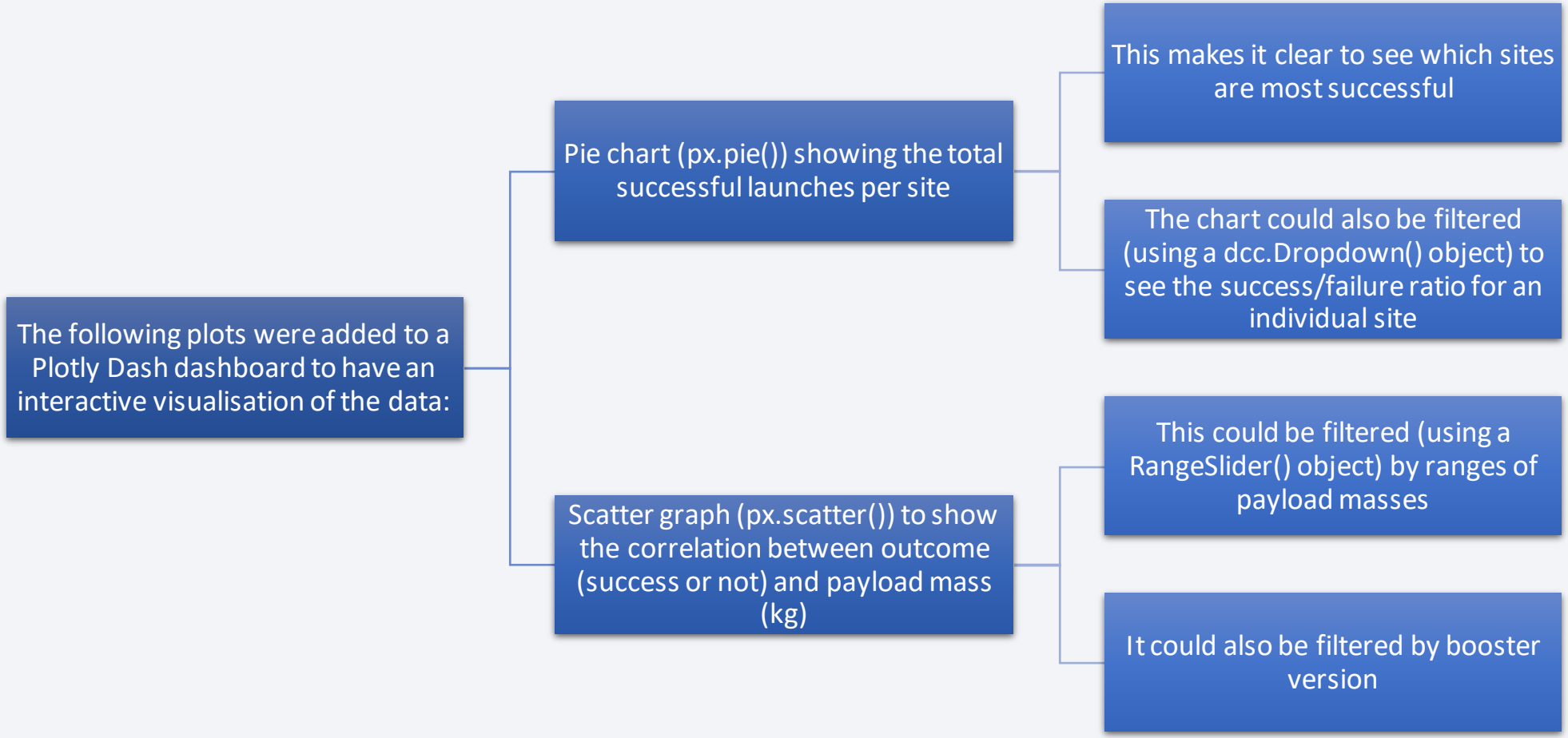
## The following steps were taken to visualize the launch data on an interactive map

1. **Mark all launch sites on a map**
   - Initialize the map using a folium Map object
   - Add folium circle and folium marker for each launch site on the map
2. **Mark the success/failed launches for each site on a map**
   - Assign color green for successful class and red for failed class to its marker
   - Since many launch sites have the same coordinates, it make sense to cluster them together.
   - To put the launches into clusters, add the folium.Marker to MarckerClusters object
3. **Calculate the distance between a launch site to it's 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.



GitHub

# Build a Dashboard with Plotly Dash






GitHub

# Predictive Analysis (Classification)

## Model Development

- Data set Preparation to model
  - Load dataset
  - Perform pre-processing to data
  - Split data into training and test data
  - Decide which ML algorithms are suitable
- For each chosen algorithm
  - Create dictionary parameter and GridSearchCV
  - Fit Object to Parameter
  - Use Training dataset to Train model

## Model Evaluation

For each chosen algorithm

- Using the output GridSearchCV Object:
  - Check the tuned hyperparameters
  - 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



GitHub

# Results

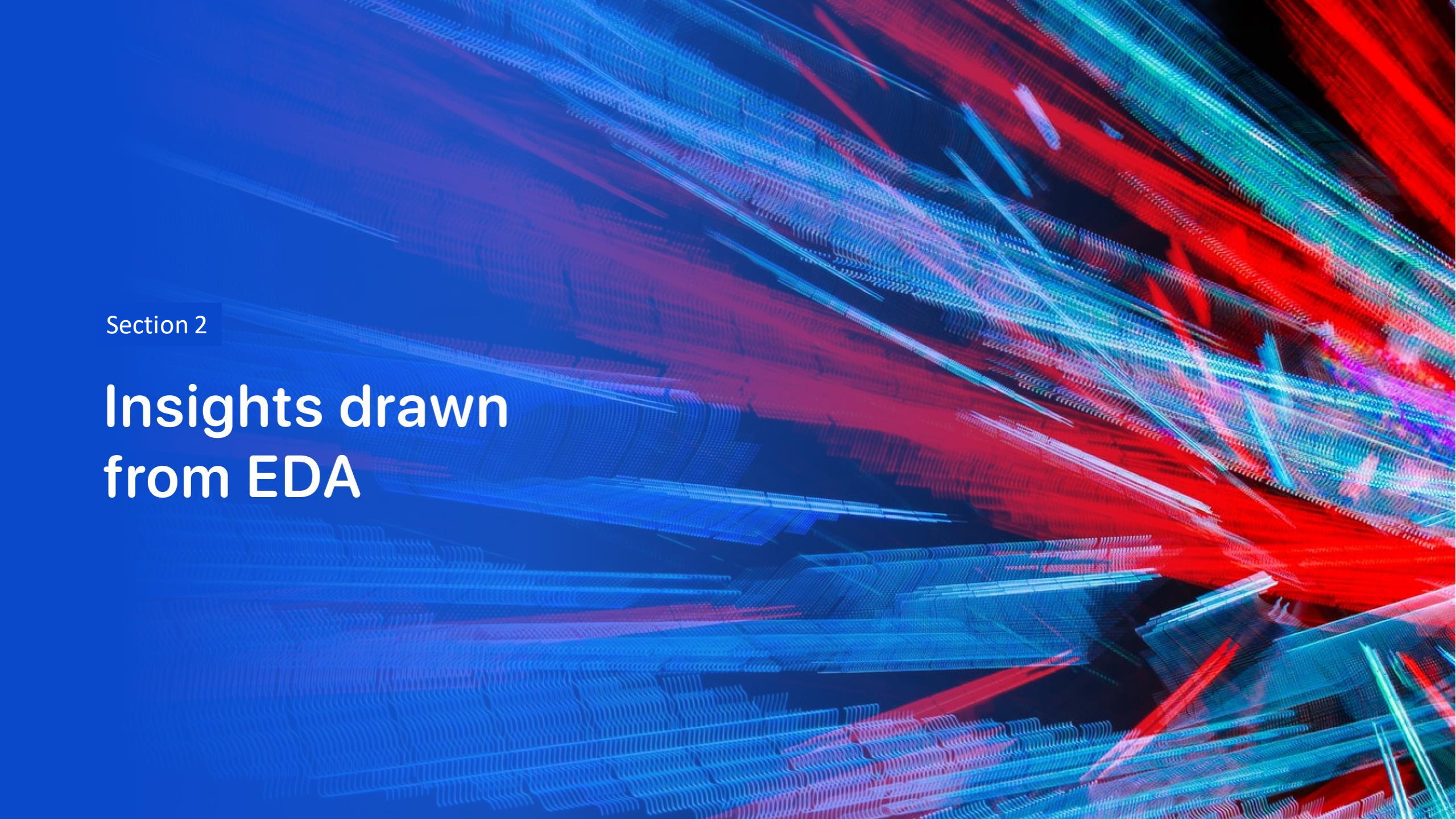
EXPLORATORY DATA ANALYSIS RESULTS

INTERACTIVE ANALYTICS DEMO IN SCREENSHOTS

PREDICTIVE ANALYSIS RESULTS

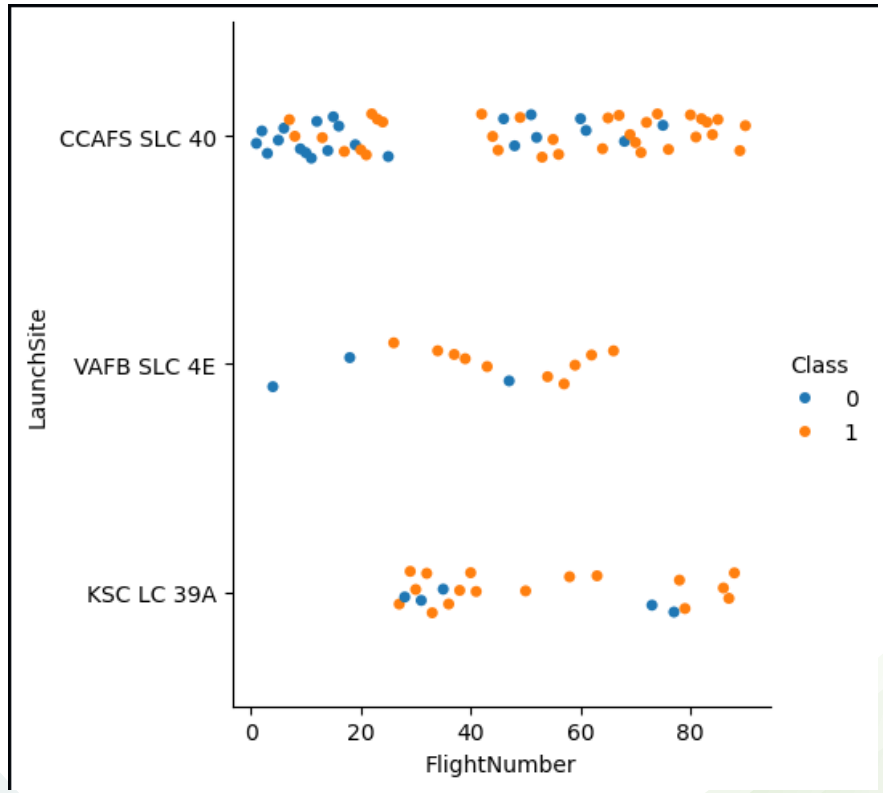
Section 2

# Insights drawn from EDA



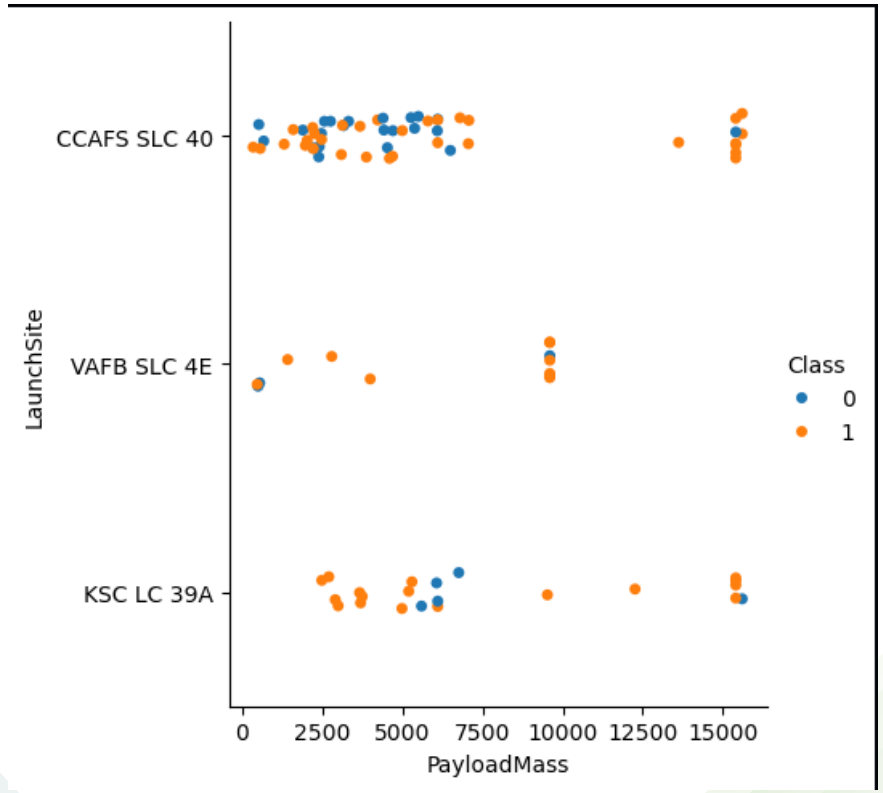
# Flight Number vs. Launch Site

- 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).



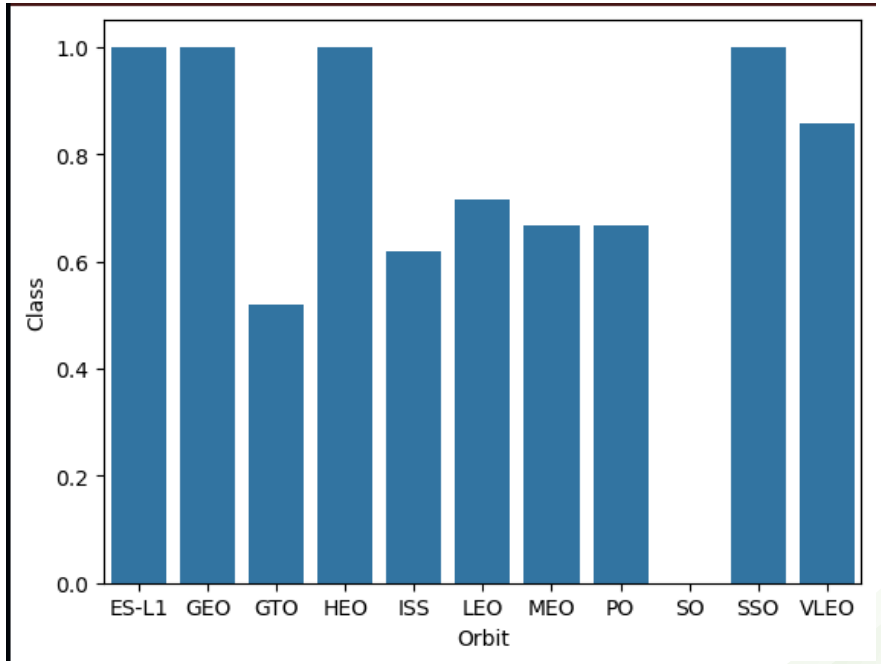
# Payload vs. Launch Site

- 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).



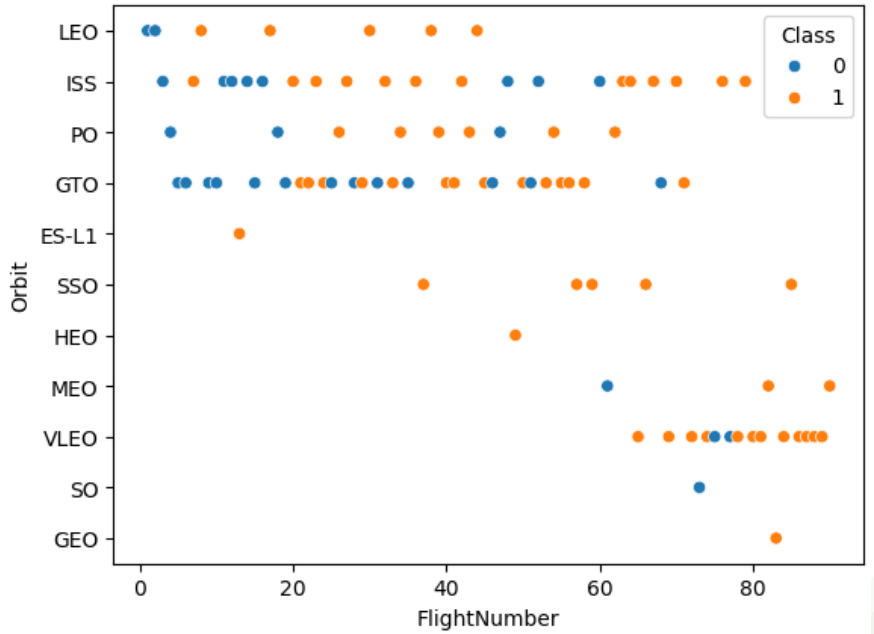
# Success Rate vs. Orbit Type

- 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

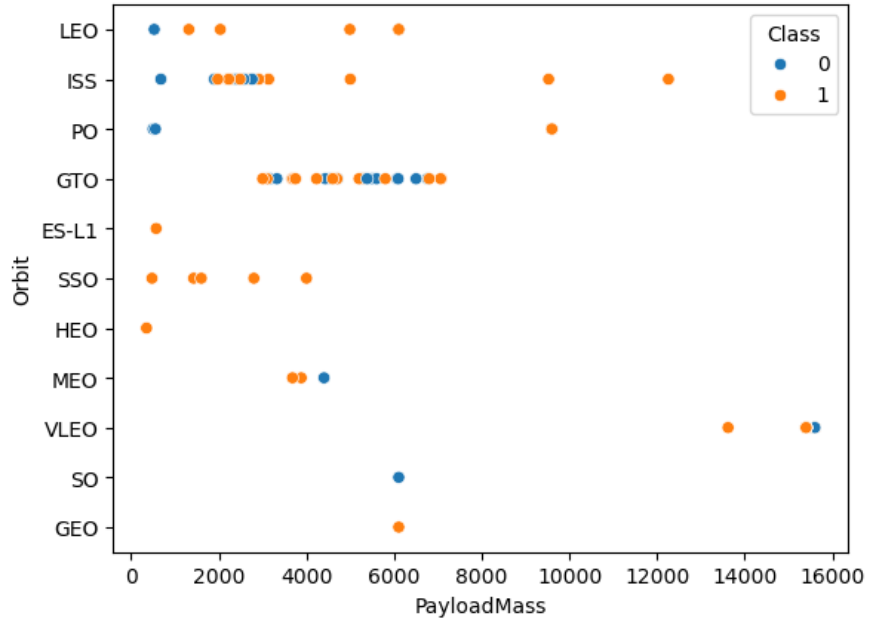
This scatter plot of Orbit Type vs. Flight number shows a few useful things that the previous plots did not, such as:




- 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

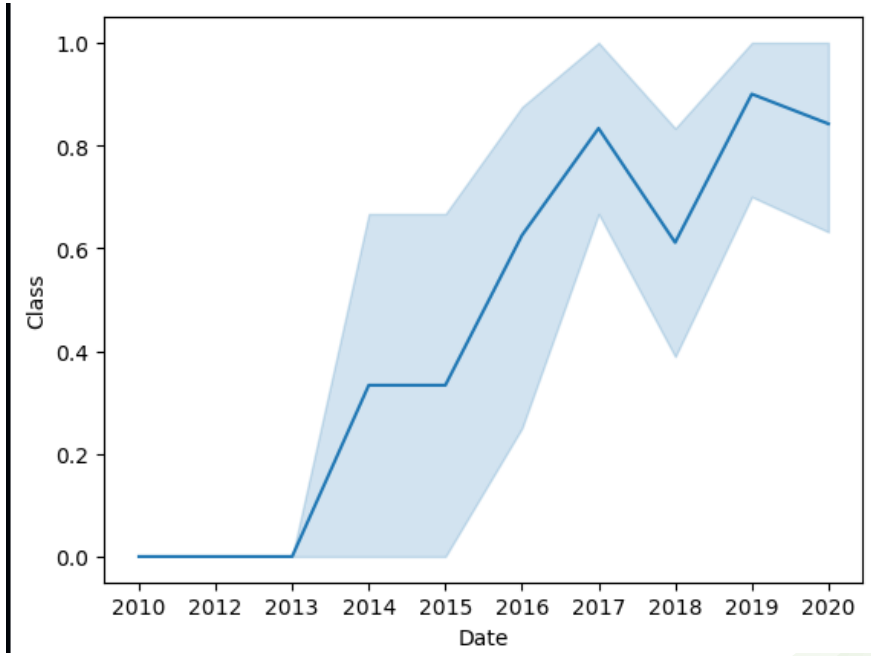
- This scatter plot of Orbit Type vs. Payload Mass shows that:
- The following orbit types have more success with heavy payloads:
  - PO
  - ISS
  - LEO
- For GTO, the relationship between payload mass and success rate is unclear.
- VLEO (Very Low Earth Orbit) launches are associated with heavier payloads,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

Display the names of the unique launch sites in the space mission

```
[19]: %sql SELECT DISTINCT LAUNCH_SITE FROM SPACEXTBL
```

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

[19]: ,,,,,,,,,,,,,,,,,,,

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

The word **DISTINCT** return only unique values from Launch Site column of the SPACEXTBL table.

# Launch Site Names Begin with 'CCA'

LIKE keyword is used with the wild card 'CAA%' to retrieve the values beginning with 'CCA'

LIMIT 5 fetch only five records

[25]:
```
%%sql
SELECT LAUNCH_SITE
FROM SPACEXTBL
WHERE LAUNCH_SITE LIKE 'CCA%' LIMIT 5;
```

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

[25]:

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

# Total Payload Mass

SUM keyword is used to calculate the values of 'PAYLOAD_MASS_KG_' column

With condition CUSTOMER is equal 'NASA (CRS)'

Task 3

Display the total payload mass carried by boosters launched by NASA (CRS)

[27]:
```
%%sql
SELECT SUM(PAYLOAD_MASS__KG_) AS Total_Payload
FROM SPACEXTBL
WHERE CUSTOMER == 'NASA (CRS)';
```

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

[27]:

| Total_Payload |
|---|
| 45596 |

## Task 4

Display average payload mass carried by booster version F9 v1.1

[29]:
```
%%sql
SELECT AVG(PAYLOAD_MASS__KG_) AS AVERAGE_PAYLOAD_MASS
FROM SPACEXTBL
WHERE Booster_Version == 'F9 v1.1';
```

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

[29]:

| AVERAGE_PAYLOAD_MASS |
|---|
| 2928.4 |

# Average Payload Mass by F9 v1.1

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.

[30]:
```
%%sql
SELECT MIN(Date) AS FIRST_SUCCESSFUL_GROUND_LANDING
FROM SPACEXTBL
WHERE Landing_Outcome == 'Success (ground pad)';
```

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

Done.
```

[30]:

| FIRST_SUCCESSFUL_GROUND_LANDING |
| --- |
| 2015-12-22 |

# First Successful Ground Landing Date

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.

List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000

[35]:
```
%%sql
SELECT Booster_Version
FROM SPACEXTBL
WHERE Landing_Outcome == 'Success (ground pad)' AND PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000;
```

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

Done.
```

[35]:

| Booster_Version |
|---|
| F9 FT B1032.1 |
| F9 B4 B1040.1 |
| F9 B4 B1043.1 |

# Successful Drone Ship Landing with Payload between 4000 and 6000

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.

List the total number of successful and failure mission outcomes

[42]:
```
%%sql
SELECT Mission_Outcome, COUNT(Mission_Outcome) AS Total_Number
FROM SPACEXTBL
GROUP BY Mission_Outcome;
```

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

Done.
```

[42]:

| Mission_Outcome | Total_Number |
|---|---|
| Failure (in flight) | 1 |
| Success | 98 |
| Success | 1 |
| Success (payload status unclear) | 1 |

# Total Number of Successful and Failure Mission Outcomes

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.

[47]:
```
%%sql
SELECT DISTINCT Booster_Version
FROM SPACEXTBL
WHERE PAYLOAD_MASS__KG_ == (
    SELECT MAX(PAYLOAD_MASS__KG_)
    FROM SPACEXTBL
)
```

```
 * 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 |

# Boosters Carried Maximum Payload

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.

List the records which will display the month names, failure landing_outcomes in drone ship ,booster versions, launch_site for the months in year 2015.

**Note: SQLLite does not support monthnames. So you need to use substr(Date, 6,2) as month to get the months and substr(Date,0,5)='2015' for year.**

[61]:
```
%%sql
SELECT substr(Date, 6, 2) AS Month, Booster_Version, Launch_Site
FROM SPACEXTBL
WHERE (Landing_Outcome = 'Failure (drone ship)') AND (substr(Date, 0,5) = '2015');
```

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

[61]:

| Month | Booster_Version | Launch_Site |
|---|---|---|
| 10 | F9 v1.1 B1012 | CCAFS LC-40 |
| 04 | F9 v1.1 B1015 | CCAFS LC-40 |

# 2015 Launch Records

The WHERE keyword is used to filter the results for only failed landing outcomes, AND only for the year of 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.

[52]:
```
%%sql
SELECT Landing_Outcome, COUNT(Landing_Outcome) AS TOTAL_NUMBER
FROM SPACEXTBL
WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY Landing_Outcome
ORDER BY TOTAL_NUMBER DESC;
```

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

Done.
```

52]: ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,

| Landing_Outcome | TOTAL_NUMBER |
|---|---|
| No attempt | 10 |
| Success (ground pad) | 5 |
| Success (drone ship) | 5 |
| Failure (drone ship) | 5 |
| Controlled (ocean) | 3 |
| Uncontrolled (ocean) | 2 |
| Precluded (drone ship) | 1 |
| Failure (parachute) | 1 |

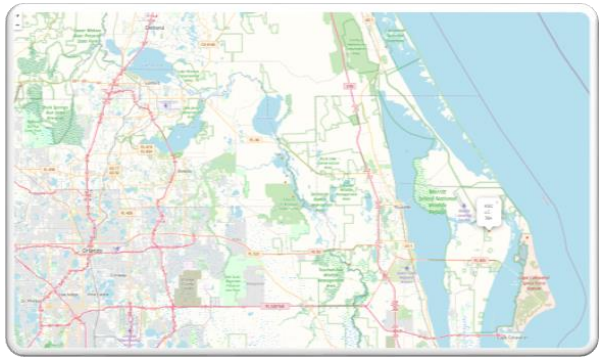
# Rank Landing Outcomes Between 2010-06-04 and 2017-03-20

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


Section 3
Launch Sites
Proximities Analysis

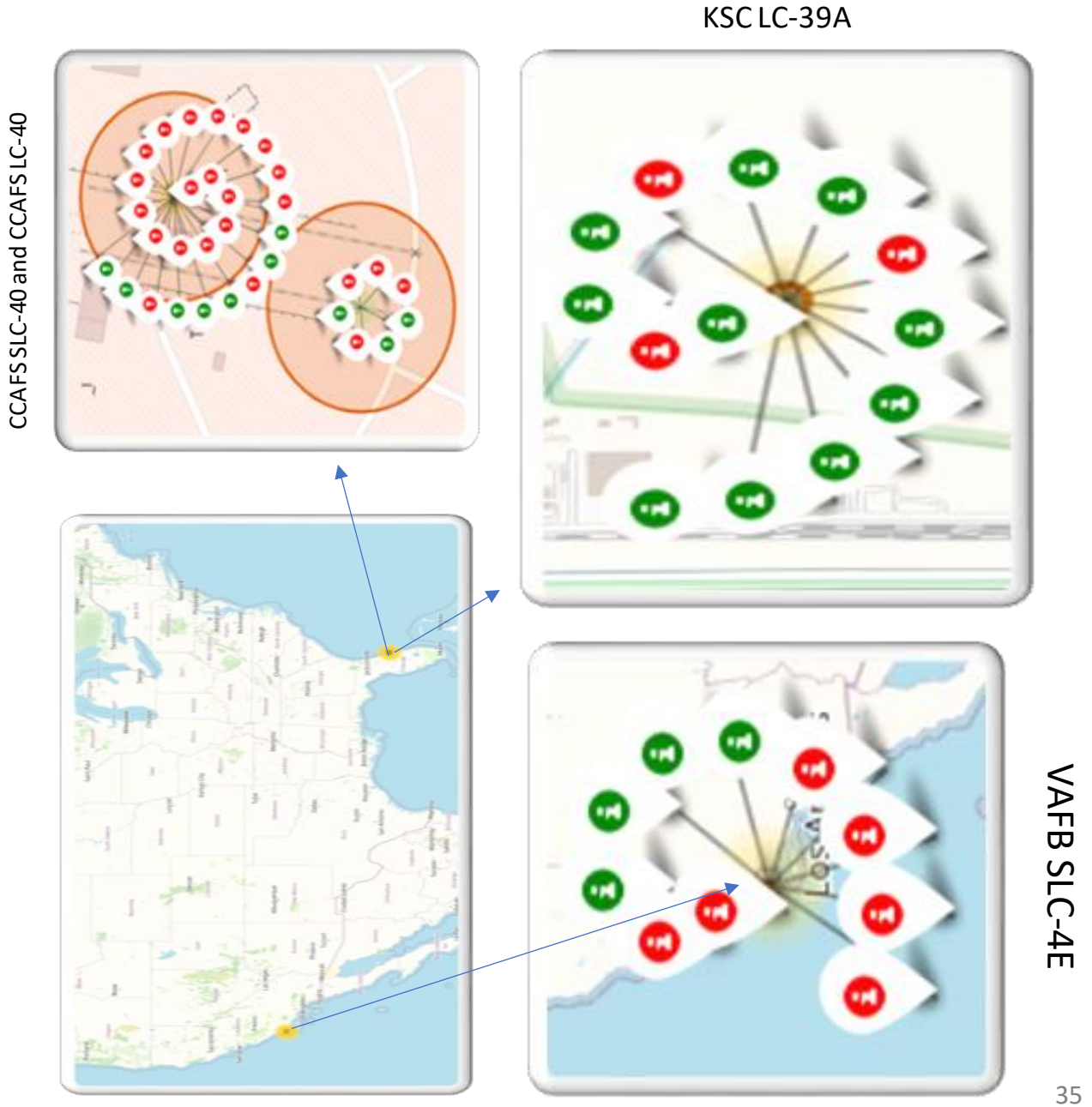
# All Launch Sites On A Map

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

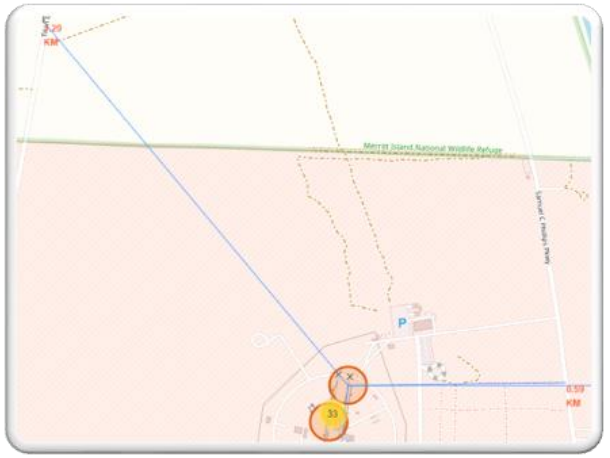
# Success/Failed Launches For Each Site

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



# 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?

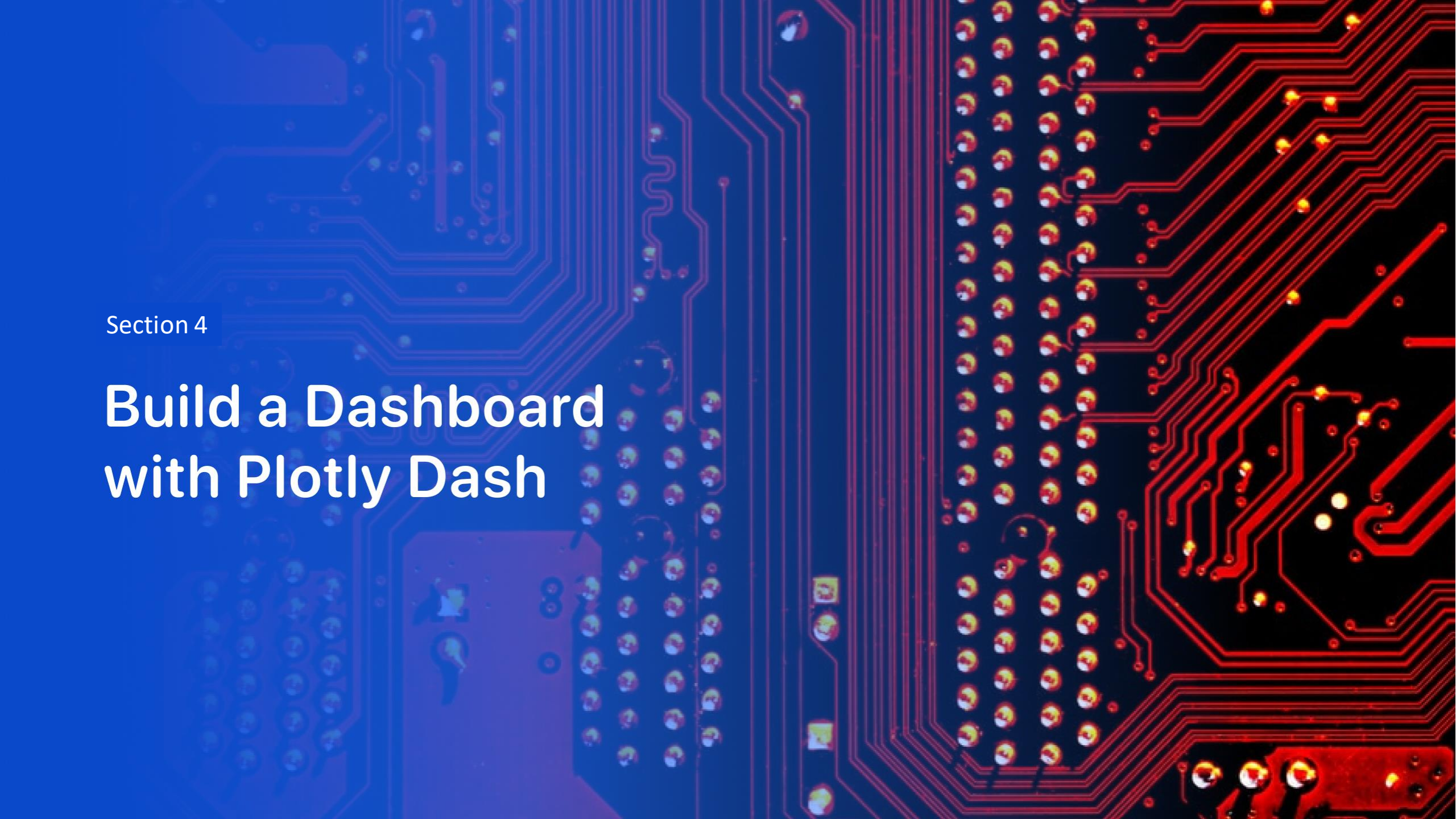
- YES. The nearest highway is only 0.59km away.

Are launch sites in close proximity to railways?

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

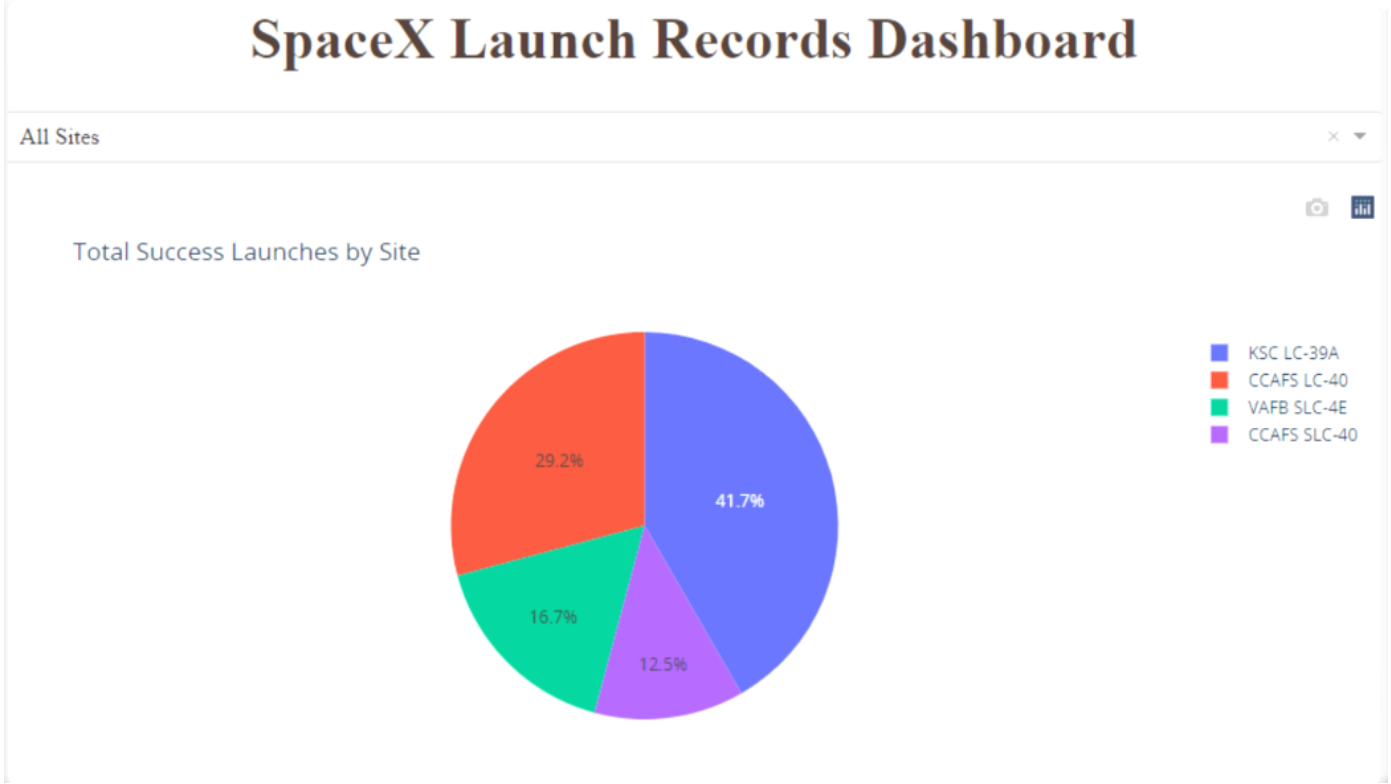
Do launch sites keep certain distance away from cities?

- YES. The nearest city is 51.74 km


Section 4
Build a Dashboard with Plotly Dash

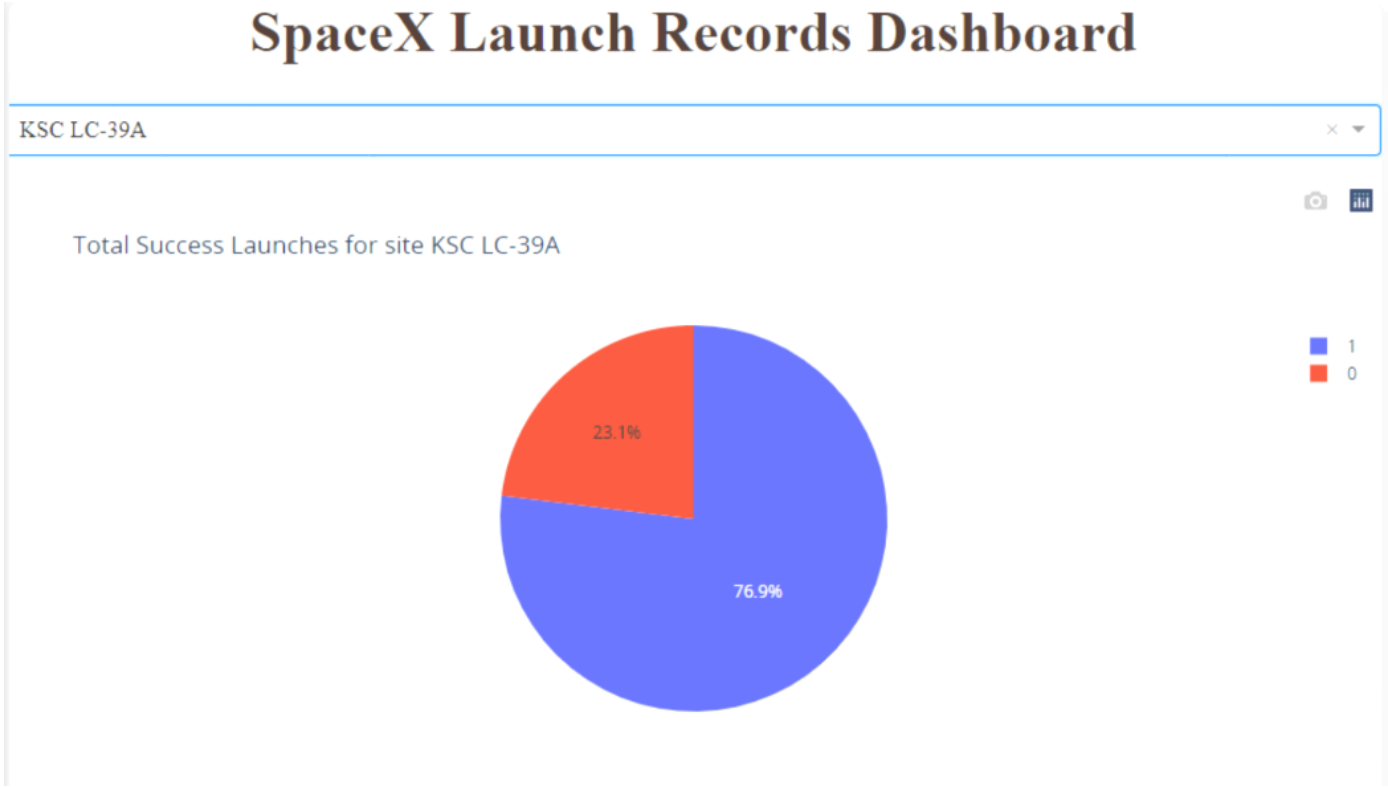
# LAUNCH SUCCESS COUNT FOR ALL SITES

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



# Highest Launch Success

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



# Payload vs. Launch Outcome scatter plot for all sites

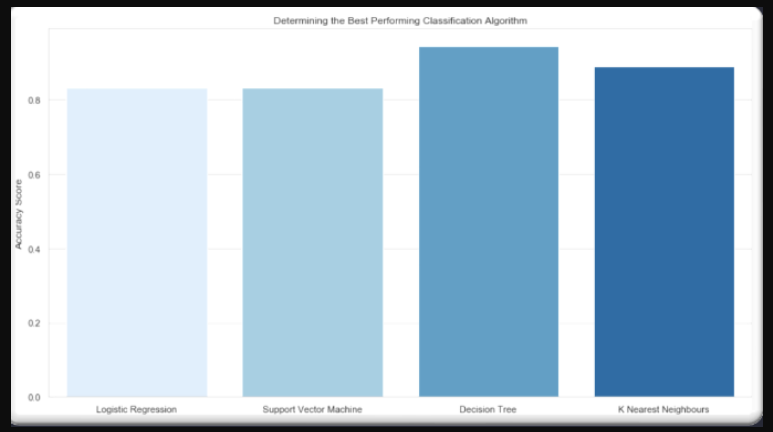
- Plotting the launch outcome vs. payload for all sites shows a gap around 4000 kg, so it makes sense to split the data into 2 ranges:
  - 0 – 4000 kg (low payloads)
  - 4000 – 10000 kg (massive payloads)
- From these 2 plots, it can be shown that the success for massive payloads is lower than that for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

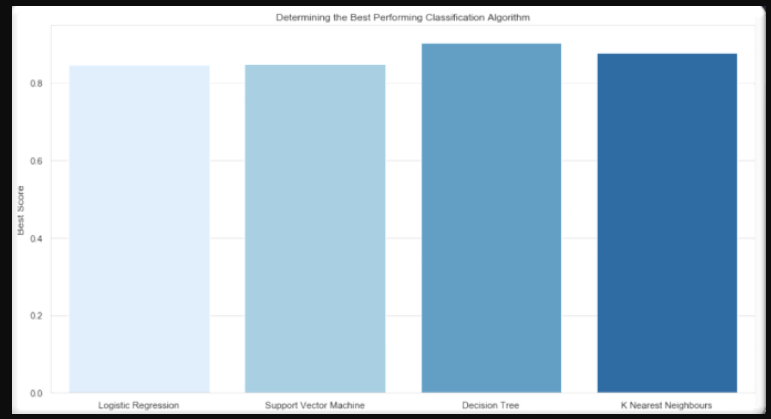
Section 5

# Predictive Analysis (Classification)

# Classification Accuracy

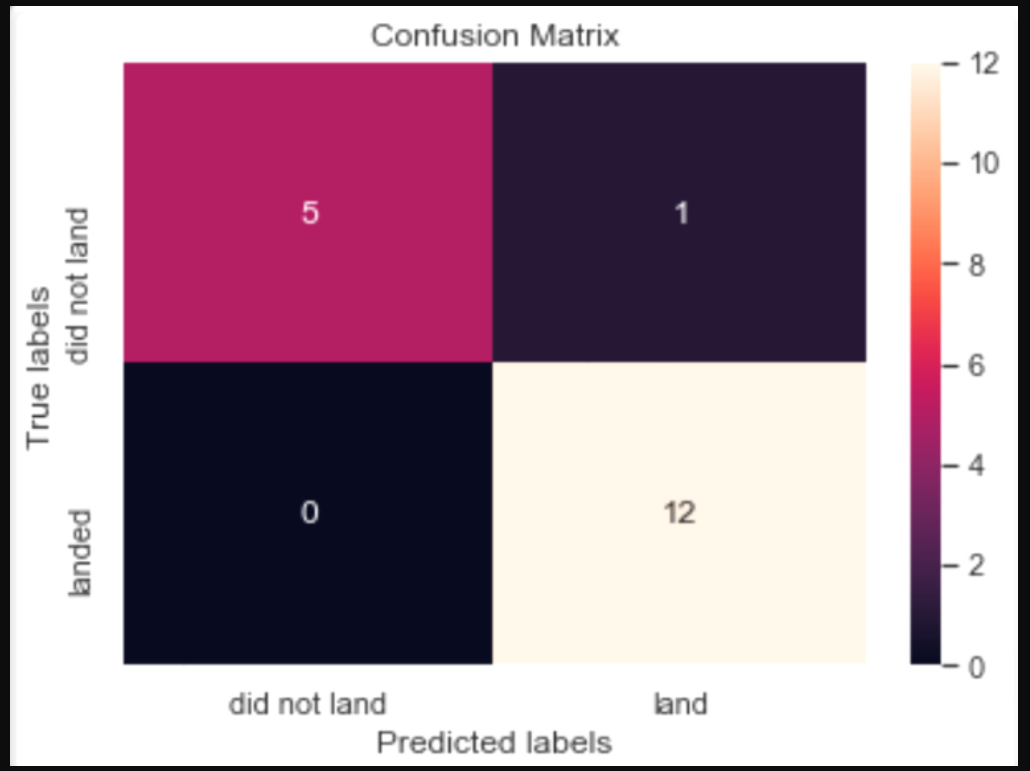
- Plotting the accuracy score and best score for each classification algorithm produces the following result:
  - The decision tree model has highest classification accuracy.
    - The accuracy score is 94.4%
    - The best score is 90.3%



# 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 result 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%.

# Appendix

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

[5]:
```
# 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'])
```

Now let's start requesting rocket launch data from SpaceX API with the following URL:

[6]:
```
spacex_url="https://api.spacexdata.com/v4/launches/past"
```

[7]:
```
response = requests.get(spacex_url)
```

Check the content of the response

[8]:
```
print(response.content)
```

## Import Libraries and Define Auxiliary Functions

We will import the following libraries into the lab

[1]:
```
# Requests allows us to make HTTP requests which we will use to get data from an API
import requests
# Pandas is a software library written for the Python programming language for data manipulation and analysis.
import pandas as pd
# NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a larg
import numpy as np
# Datetime is a library that allows us to represent dates
import datetime

# Setting this option will print all collumns of a dataframe
pd.set_option('display.max_columns', None)
# Setting this option will print all of the data in a feature
pd.set_option('display.max_colwidth', None)
```

Below we will define a series of helper functions that will help us use the API to extract information using identification numbers in the launch data.

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

[2]:
```
# 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.

[3]:
```
# 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.

[4]:
```
# 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'])
```

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, w

# Appendix

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

and we will provide some helper functions for you to process web scraped HTML table

[3]:
```
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()

    column_name = ' '.join(row.contents)

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

[13]:
```
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)

            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)

            # 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`
            try:
                customer = row[6].a.string
            except:
                customer = 'Various'
            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)
```

# Thank you!