IBM Developer
SKILLS NETWORK

# Winning Space Race with Data Science

Iman Hemyari

11 July 2023

# Outline

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

# Executive Summary

- **Summary of methodologies**
  - SpaceX Data Collection using SpaceX API
  - SpaceX Data Collection with Web Scraping
  - SpaceX Data Wrangling
  - SpaceX Exploratory Data Analysis using SQL
  - Space-X EDA DataViz Using Python Pandas and Matplotlib
  - Space-X Launch Sites Analysis with Folium-Interactive Visual Analytics and Ploty Dash
  - SpaceX Machine Learning Landing Prediction
- **Summary of all results**
  - EDA results
  - Interactive Visual Analytics and Dashboards
  - Predictive Analysis(Classification)

# Introduction


- Project background and context

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

- Problems you want to find answers

  In this capstone, we will predict if the Falcon 9 first stage will land successfully using data from Falcon 9 rocket launches advertised on its website.

Section 1

# Methodology

# Methodology

## Executive Summary

- Data collection methodology:
  - Describe how data was collected
- Perform data wrangling
  - Describe how data was processed
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - How to build, tune, evaluate classification models

# Data Collection

- **Describe how data sets were collected**.
  - Data was first collected using SpaceX API (a RESTful API) by making a get request to the SpaceX API. This was done by first defining a series helper functions that would help in the use of the API to extract information using identification numbers in the launch data and then requesting rocket launch data from the SpaceX API url.
  - Finally to make the requested JSON results more consistent, the SpaceX launch data was requested and parsed using the GET request and then decoded the response content as a Json result which was then converted into a Pandas data frame.
  - Also performed web scraping to collect Falcon 9 historical launch records from a Wikipedia page titled List of Falcon 9 and Falcon Heavy launches of the launch records are stored in a HTML. Using BeautifulSoup and request Libraries, I extract the Falcon 9 launch HTML table records from the Wikipedia page, Parsed the table and converted it into a Pandas data frame
- **You need to present your data collection process use key phrases and flowcharts**

# Data Collection – SpaceX API

- Data collected using SpaceX API (a RESTful API) by making a get request to the SpaceX API then requested and parsed the SpaceX launch data using the GET request and decoded the response content as a Json result which was then converted into a Pandas data frame.
- https://github.com/imanhemyari/SpaceX/blob/main/labs-jupyter-spacex-Data%20wrangling.ipynb

Task 1: Request and parse the SpaceX launch data using the GET request

To make the requested JSON results more consistent, we will use the following static response object for this project:

```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_call_spacex_api.json'
```

We should see that the request was successfull with the 200 status response code

```
response.status_code
```

```
200
```

Now we decode the response content as a Json using `.json()` and turn it into a Pandas dataframe using `.json_normalize()`

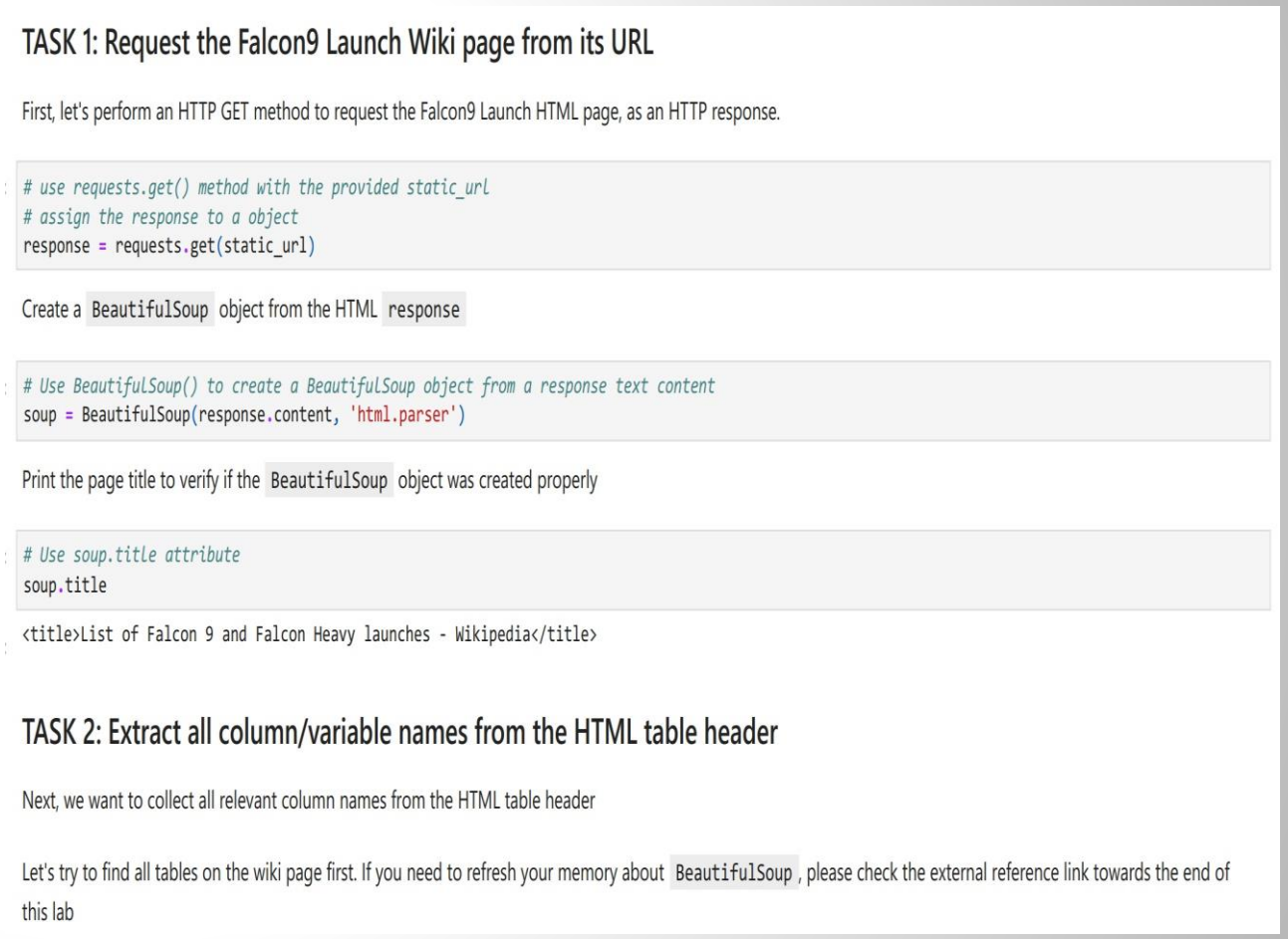
```
# Use json_normalize meethod to convert the json result into a dataframe
respjson = response.json()
data = pd.json_normalize(respjson)
```

Using the dataframe `data` print the first 5 rows

```
# Get the head of the dataframe
data.head()
```

# Data Collection - Scraping

- Performed web scraping to collect Falcon 9 historical launch records from a Wikipedia using BeautifulSoup and request, to extract the Falcon 9 launch records from HTML table of the Wikipedia page, then created a data frame by parsing the launch HTML.
- https://github.com/imanhemyari/SpaceX/blob/main/jupyter-labs-webscraping.ipynb



TASK 1: Request the Falcon9 Launch Wiki page from its URL

First, let's perform an HTTP GET method to request the Falcon9 Launch HTML page, as an HTTP response.

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

Create a `BeautifulSoup` object from the HTML `response`

```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(response.content, 'html.parser')
```

Print the page title to verify if the `BeautifulSoup` object was created properly

```
# Use soup.title attribute
soup.title
```

```
<title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

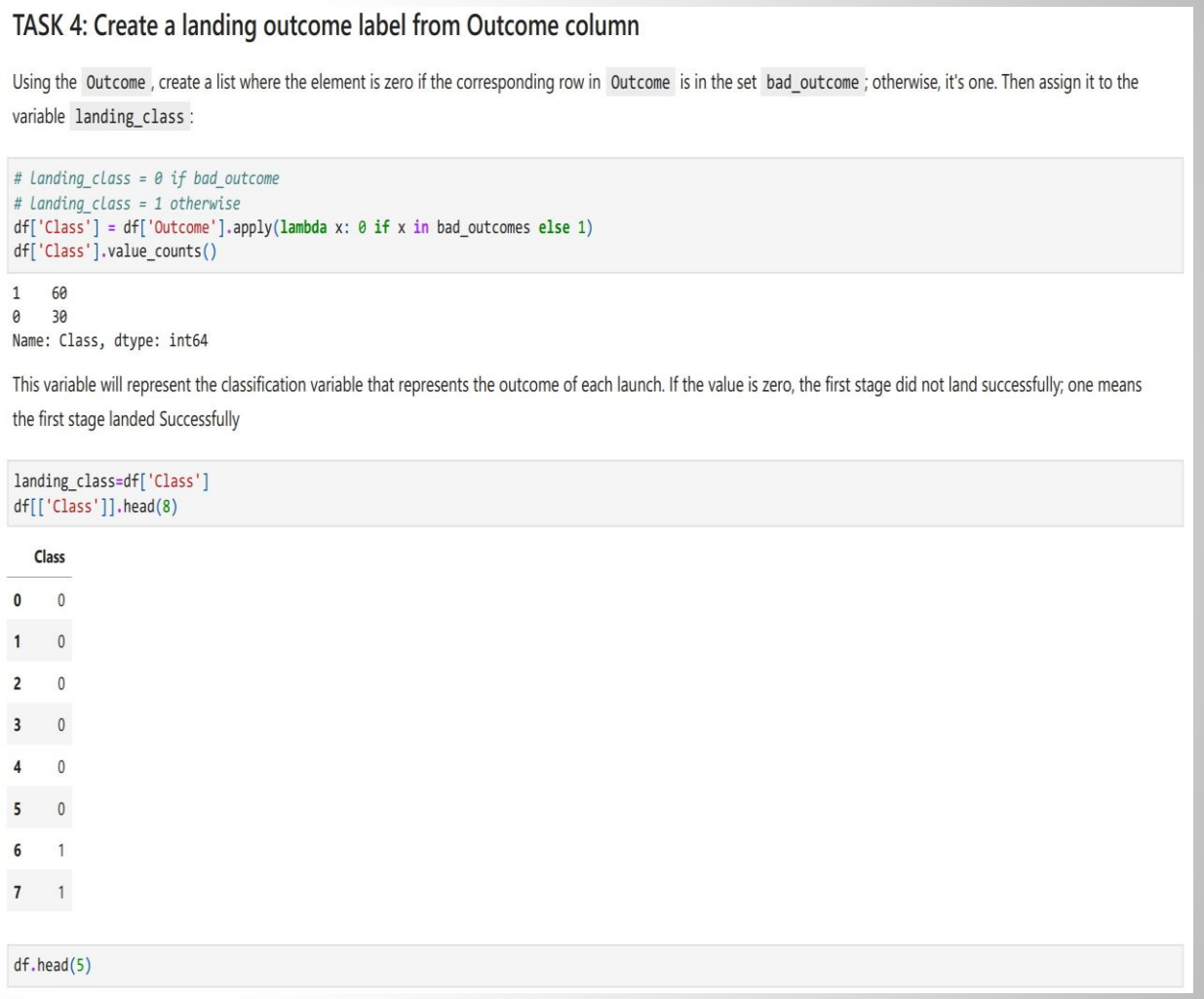
TASK 2: Extract all column/variable names from the HTML table header

Next, we want to collect all relevant column names from the HTML table header

Let's try to find all tables on the wiki page first. If you need to refresh your memory about `BeautifulSoup` , please check the external reference link towards the end of this lab

# Data Wrangling

- After obtaining and creating a Pandas DF from the collected data, data was filtered using theBoosterVersion column to only keep the Falcon 9 launches, then dealt with the missing data values in the LandingPad andPayloadMass, columns. For the PayloadMass missing data values were replaced using mean value of column. Also performed some Exploratory Data Analysis (EDA) to find some patterns in the data and determine what would be the label for training supervised models.
- https://github.com/imanhemyari/SpaceX/blob/main/labs-jupyter-spacex-Data%20wrangling.ipynb



TASK 4: Create a landing outcome label from Outcome column

Using the `Outcome`, create a list where the element is zero if the corresponding row in `Outcome` is in the set `bad_outcome`; otherwise, it's one. Then assign it to the variable `landing_class`:

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

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

This variable will represent the classification variable that represents the outcome of each launch. If the value is zero, the first stage did not land successfully; one means the first stage landed Successfully

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

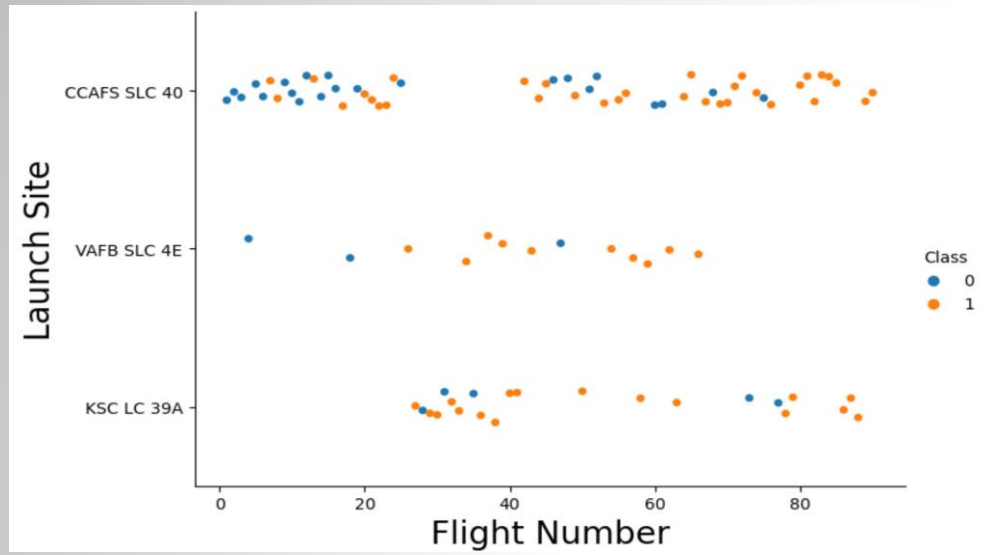
| | Class |
|---|---|
| 0 | 0 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 1 |
| 7 | 1 |

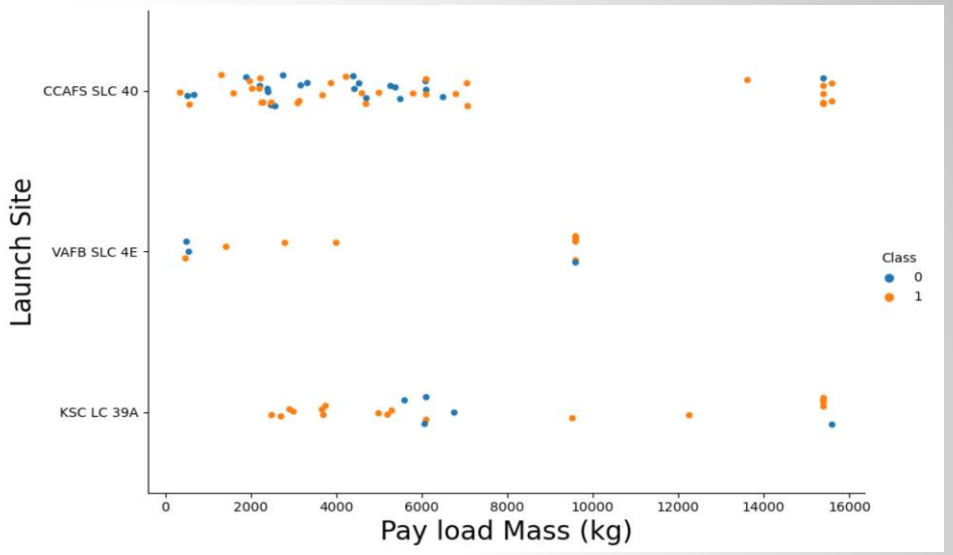
```
df.head(5)
```

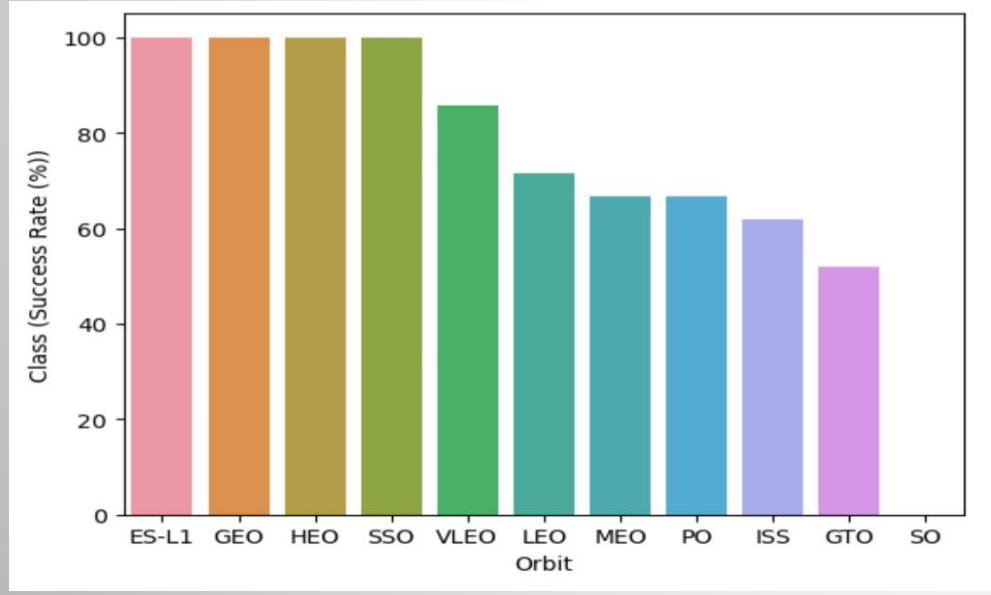
# EDA with Data Visualization

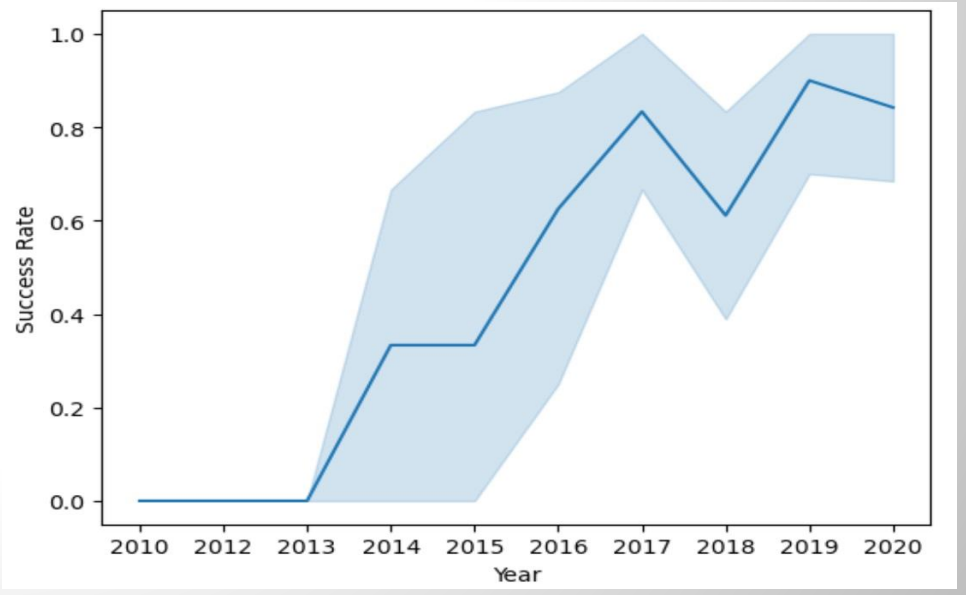
- Performed data Analysis and Feature Engineering using Pandas and Matplotlib.i.e.
  - Exploratory Data Analysis
  - Preparing Data Feature Engineering
- Used scatter plots to Visualize the relationship between Flight Number and Launch
- Site, Payload and Launch Site, Flight Number and Orbit type, Payload and Orbit type.
- Used Bar chart to Visualize the relationship between success rate of each orbit type
- Line plot to Visualize the launch success yearly trend.
- Here is the GitHub URL of your completed EDA with data visualization notebook.
- https://github.com/imanhemyari/SpaceX/blob/main/jupyter-labs-eda-dataviz.ipynb

# EDA with Data Visualization



# EDA with SQL

- **Task 1**

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

```
%sql SELECT DISTINCT LAUNCH_SITE as "Launch_Sites" FROM SPACEXTBL;
```

- **Task 2**

  **Display 5 records where launch sites begin with the string 'CCA'**

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

- **Task 3**
- **Display the total payload mass carried by boosters launched by NASA (CRS)**

```
%sql SELECT SUM(PAYLOAD_MASS__KG_) as "Total Payload Mass(Kgs)", Customer FROM 'SPACEXTBL' WHERE Customer = 'NA
```

- **Task 4**

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

```
%sql SELECT AVG(PAYLOAD_MASS__KG_) as "Payload Mass Kgs", Customer, Booster_Version FROM 'SPACEXTBL' WHERE Boos
```

# EDA with SQL

- **Task 5**

  **List the date when the first succesful landing outcome in ground pad was acheived.**

```
%sql SELECT MIN(DATE) FROM 'SPACEXTBL' WHERE "Landing _Outcome" = "Success (ground pad)";
```

- **Task 6**

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

```
%sql SELECT DISTINCT Booster_Version, Payload FROM SPACEXTBL WHERE "Landing _Outcome" = "Success (drone ship)"
```

- **Task 7**

  **List the total number of successful and failure mission outcomes**

```
%sql SELECT "Mission_Outcome", COUNT("Mission_Outcome") as Total FROM SPACEXTBL GROUP BY "Mission_Outcome";
```

- GitHub URL of your completed EDA with SQL notebook.

  **https://github.com/imanhemyari/SpaceX/blob/main/jupyter-lab-Space-X-EDA-Using%20SQL.ipynb**

# Build an Interactive Map with Folium

- The launch success rate may depend on many factors such as payload mass, orbit type, and so on. It may also depend on the location and proximities of a launch site, i.e., the initial position of rocket trajectories. Finding an optimal location for building a launch site certainly involves many factors and hopefully we could discover some of the factors by analyzing the existing launch site locations.
- Created folium map to marked all the launch sites, and created map objects such as markers, circles, lines to mark the success or failure of launches for each launch site.
- Add the GitHub URL of your completed interactive map with Folium map, as an external reference and peer-review purpose

  https://github.com/imanhemyari/SpaceX/blob/main/lab_jupyter_launch_site

  _location.ipynb

# Build a Dashboard with Plotly Dash

- **Built an interactive dashboard application with Plotly dash by:**

  - Adding a Launch Site Drop-down Input Component

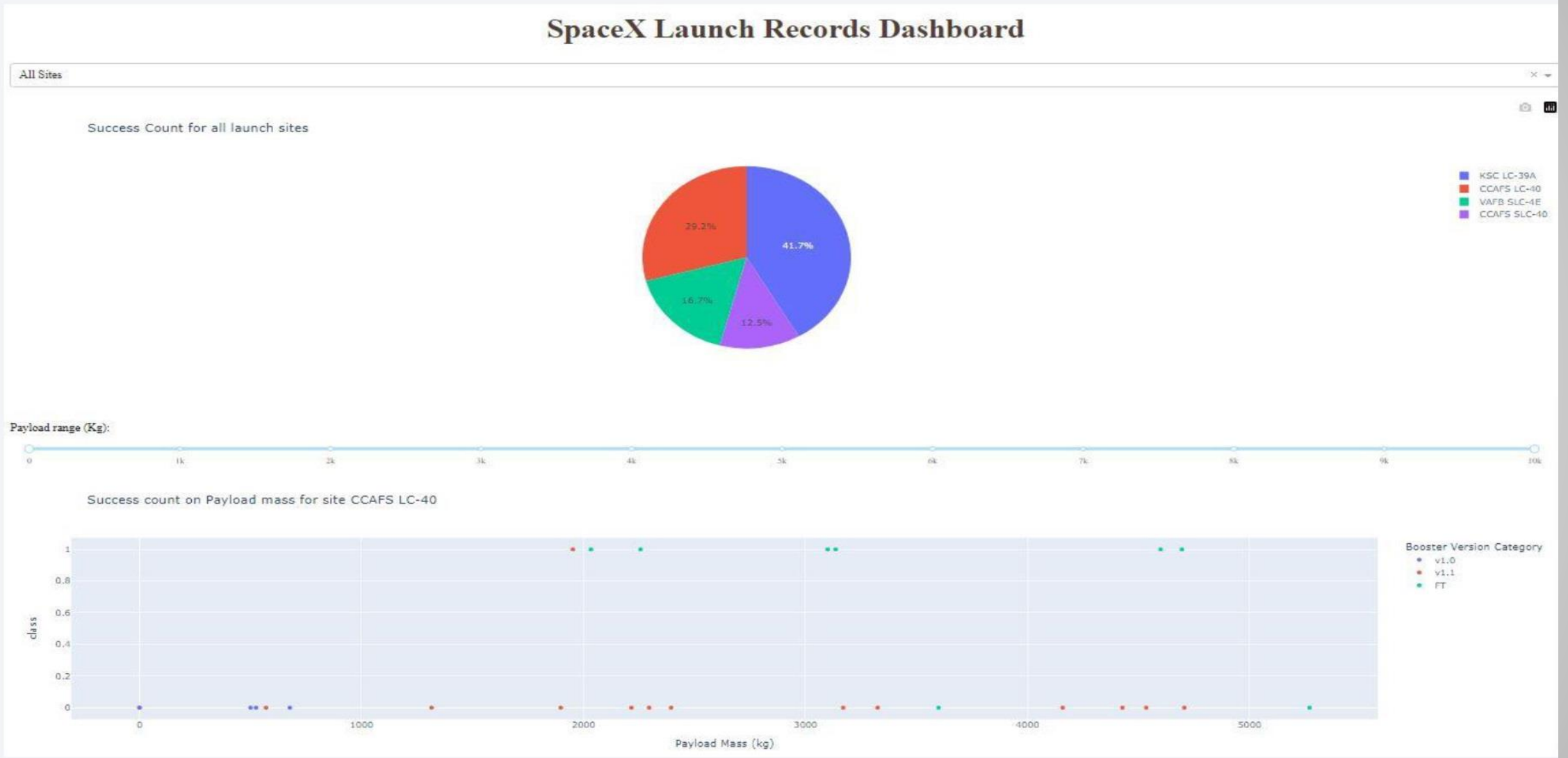
  - Adding a callback function to render success-pie-chart based on selected site dropdown

  - Adding a Range Slider to Select Payload

  - Addeng a callback function to render the success-payload-scatter-chart scatter plot

- **Add the GitHub URL of your completed Plotly Dash lab, as an external reference and peer-review purpose**
- https://github.com/imanhemyari/SpaceX/blob/main/Build%20an%20Interactive%20Dashboard%20with%20Ploty%20Dash%20-%20spacex_dash_app.py

# SpaceX Dash App



# Predictive Analysis (Classification)

- **Summarize how you built, evaluated, improved, and found the best performing classification model**
- **After loading the data as a Pandas Dataframe, I set out to perform exploratory Data Analysis and determine Training Labels by;**

i. creating a NumPy array from the column Class in data, by applying the method to_numpy() then assigned it to the variable Y as the outcome variable.

ii. Then standardized the feature dataset (x) by transforming it using preprocessing.StandardScaler() function from Sklearn.

iii. After which the data was split into training and testing sets using the function train_test_split from sklearn.model_selection with the test_size parameter set to 0.2 and random_state to 2.

# Predictive Analysis (Classification)

- In order to find the best ML model/ method that would performs best using the test data between SVM, Classification Trees, k nearest neighbors and Logistic Regression;

i. First created an object for each of the algorithms then created a GridSearchCV object and assigned them a set of parameters for each model.

ii. For each of the models under evaluation, the GridsearchCV object was created with cv=10, then fit the training data into the GridSearch object for each to Find best Hyperparameter.

iii. After fitting the training set, we output GridSearchCV object for each of the models, then displayed the best parameters using the data attribute best_params_ and the accuracy on the validation data using the data attribute best_score_.

iv. Finally using the method score to calculate the accuracy on the test data for each model and plotted a confussion matrix for each using the test and predicted outcomes.

# Predictive Analysis (Classification)

- In order to find the best ML model/ method that would performs best using the test data between SVM, Classification Trees, k nearest neighbors and Logistic Regression;

| Method | Test Data Accuracy |
|---|---|
| **Logistic_Reg** | 0.833333 |
| **SVM** | 0.833333 |
| **Decision Tree** | 0.833333 |
| **KNN** | 0.833333 |

*All the methods perform equally on the test data: i.e. They all have the same accuracy of 0.833333 on the test Data*

- Add the GitHub URL of your completed predictive analysis lab,

https://github.com/imanhemyari/SpaceX/blob/main/SpaceX_Machine_Learning_Prediction_part_5.ipynb

# Results

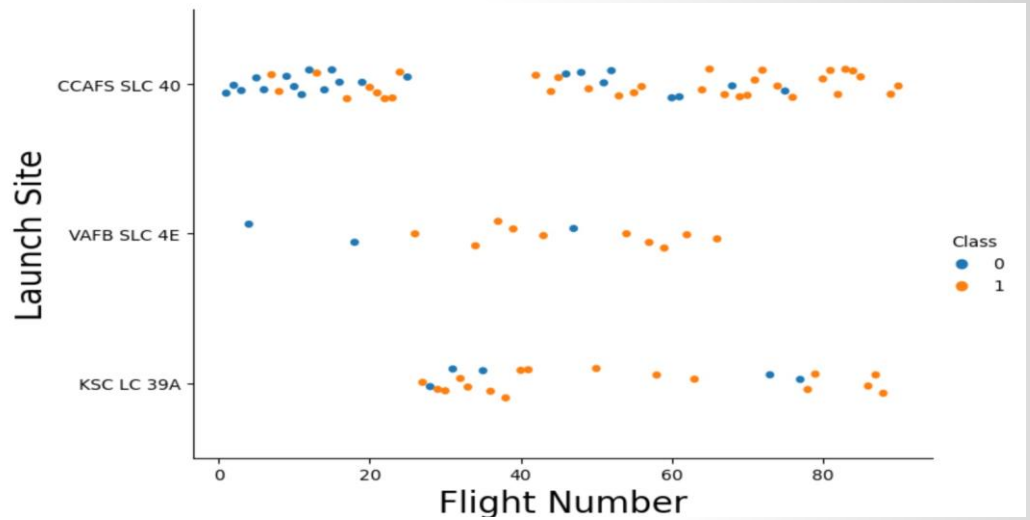
- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results

Section 2

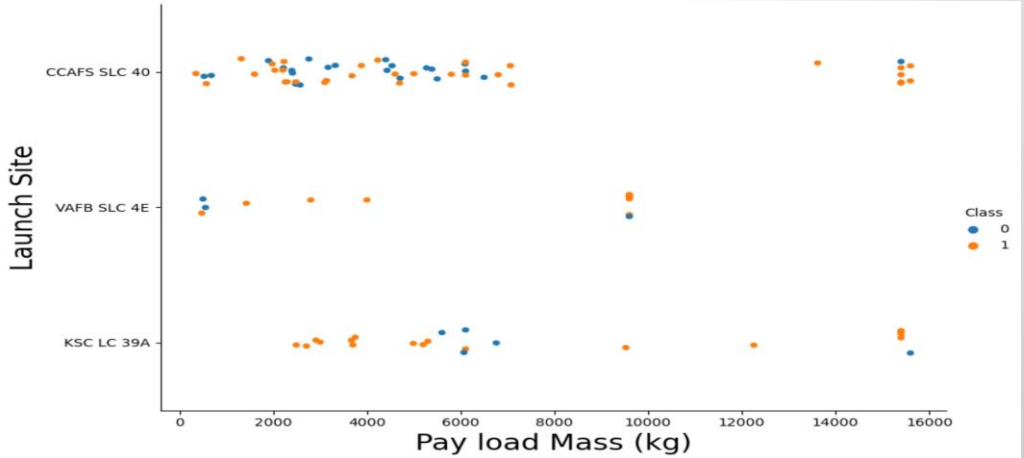
# Insights drawn from EDA

# Flight Number vs. Launch Site

- Show a scatter plot of Flight Number vs. Launch Site



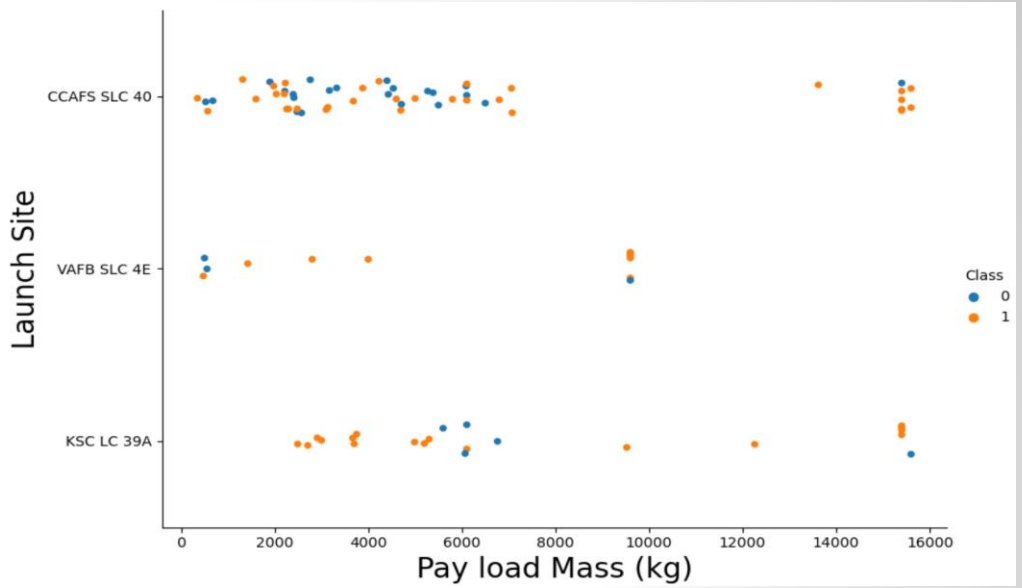

- Show the screenshot of the scatter plot with explanations



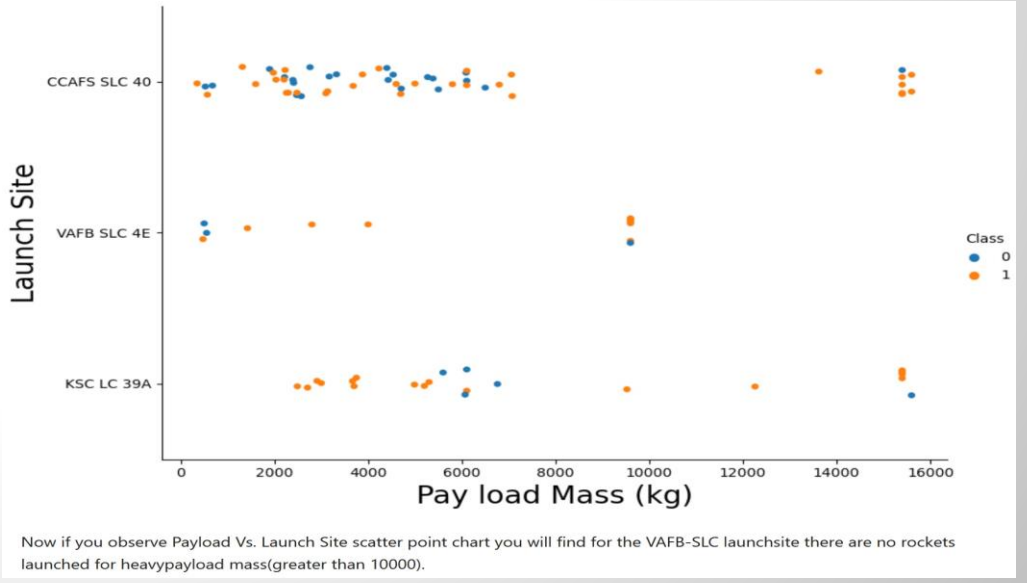

Now if you observe Payload Vs. Launch Site scatter point chart you will find for the VAFB-SLC launchsite there are no rockets launched for heavypayload mass(greater than 10000).

# Payload vs. Launch Site

- Show a scatter plot of Payload vs. Launch Site

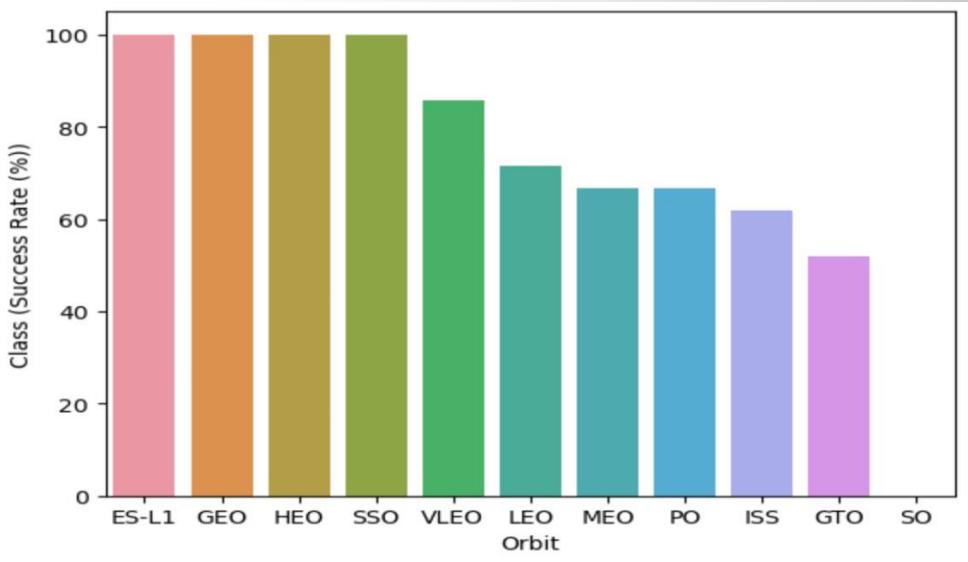



- Show the screenshot of the scatter plot with explanations

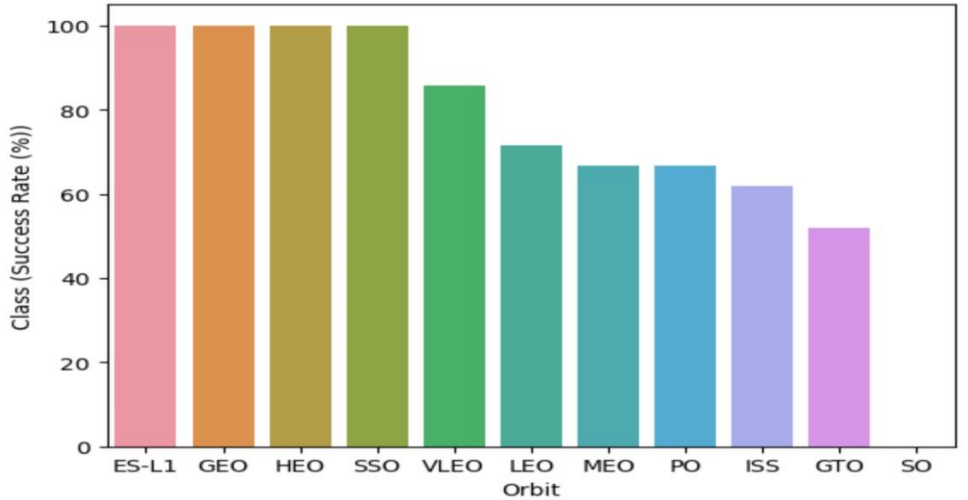


# Success Rate vs. Orbit Type

- Show a bar chart for the success rate of each orbit type

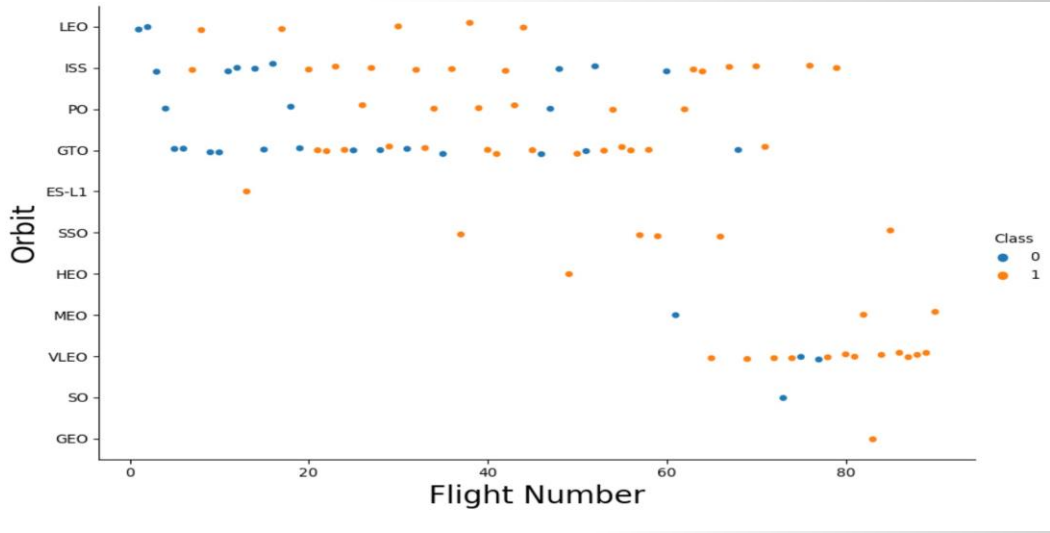



- Show the screenshot of the scatter plot with explanations

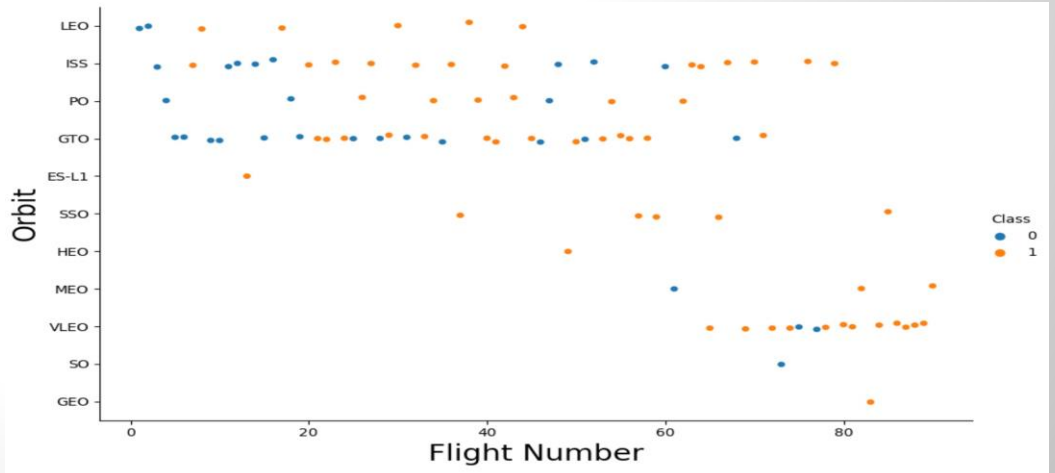


# Flight Number vs. Orbit Type

- Show a scatter point of Flight number vs. Orbit type



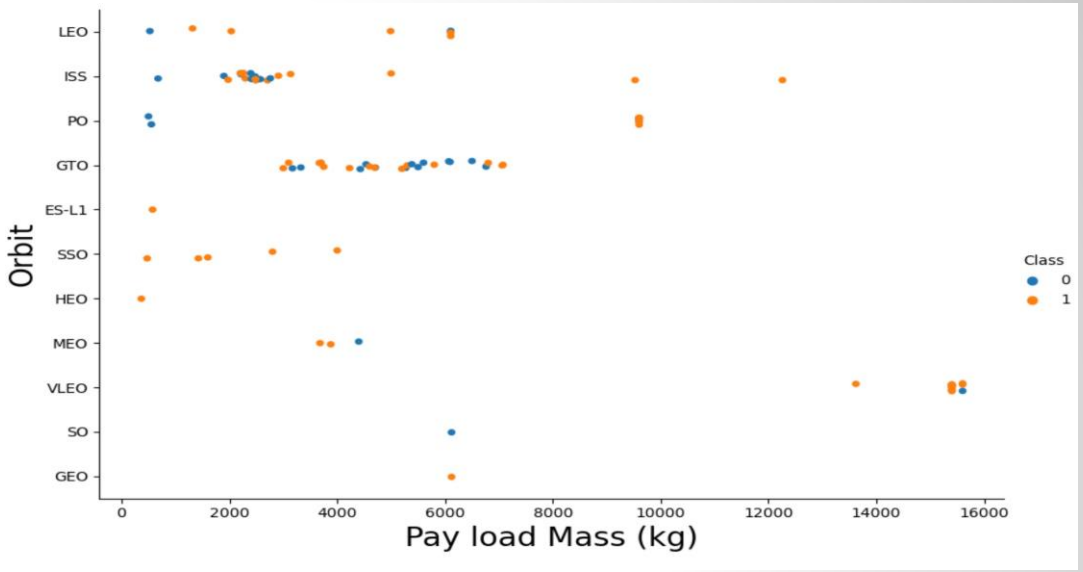

- Show the screenshot of the scatter plot with explanations



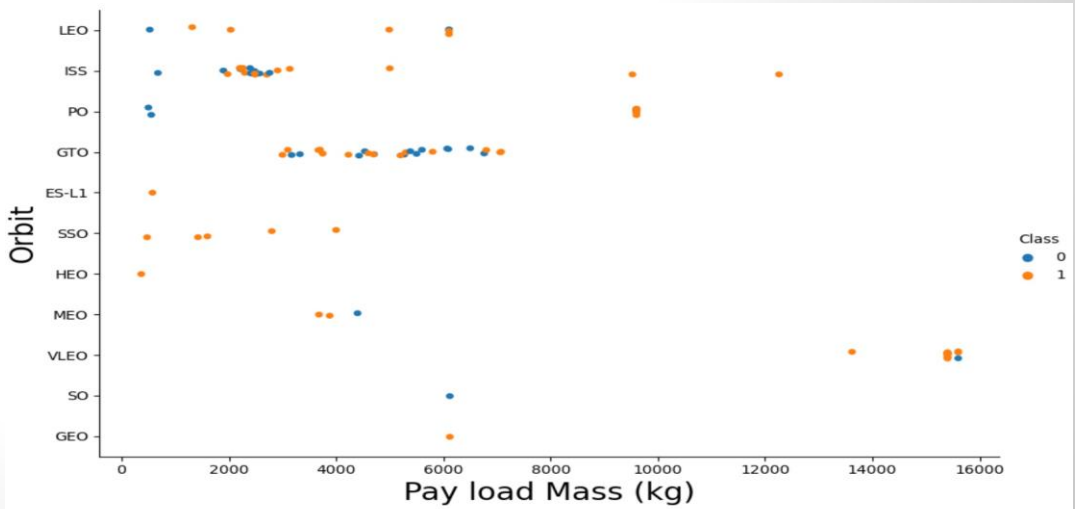

You should see that in the LEO orbit the Success appears related to the number of flights; on the other hand, there seems to be no relationship between flight number when in GTO orbit.

# Payload vs. Orbit Type

- Show a scatter point of payload vs. orbit type




- Show the screenshot of the scatter plot with explanations

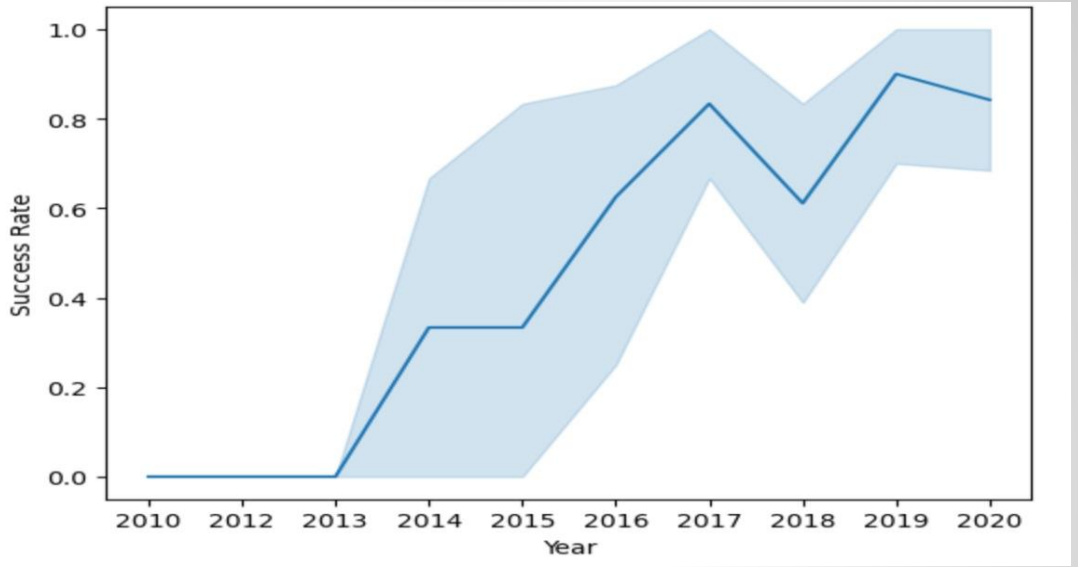



With heavy payloads the successful landing or positive landing rate are more for Polar,LEO and ISS.

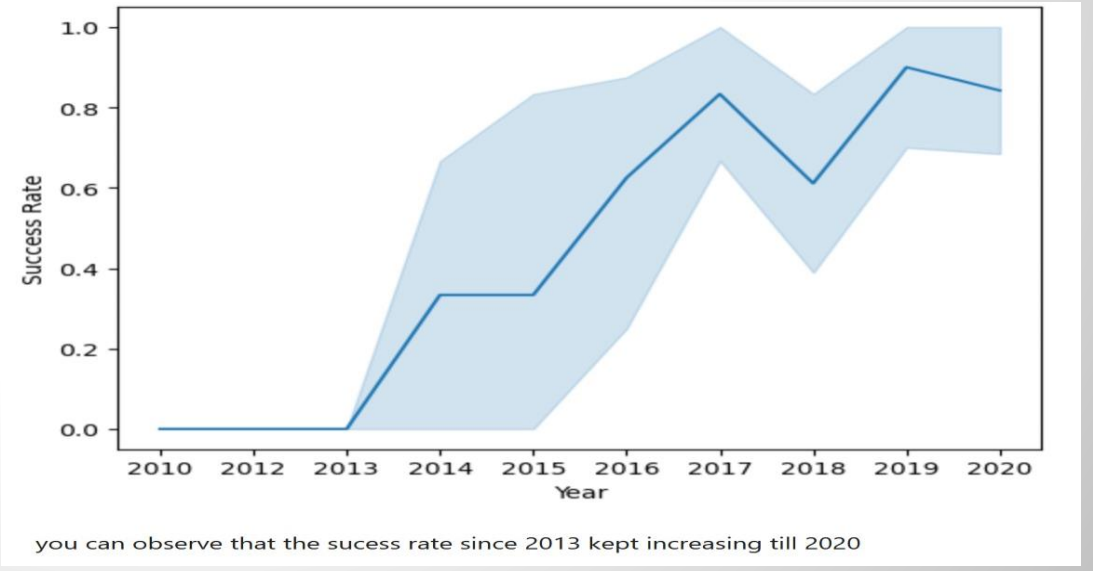
However for GTO we cannot distinguish this well as both positive landing rate and negative landing(unsuccessful mission) are both there here.

# Launch Success Yearly Trend

- Show a line chart of yearly average success rate




- Show the screenshot of the scatter plot with explanations



# All Launch Site Names

- Find the names of the unique launch sites

- Used 'SELECT DISTINCT' statement to return only the unique launch sites from the 'LAUNCH_SITE' column of the SPACEXTBL table

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

```
%sql SELECT DISTINCT LAUNCH_SITE as "Launch_Sites" FROM SPACEXTBL;
```

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

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

# Launch Site Names Begin with 'CCA'

- Find 5 records where launch sites begin with `CCA`
- Used ‘LIKE’ command with ‘%’ wildcard in ‘WHERE’ clause to select and display a table of all records where launch sites begin with the string 'CCA'

Display 5 records where launch sites begin with the string 'CCA'

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

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

| Date | Time (UTC) | Booster_Version | Launch_Site | Payload | PAYLOAD_MASS__KG_ | Orbit | Customer | Mission_Outcome |
|---|---|---|---|---|---|---|---|---|
| 04-06-2010 | 18:45:00 | F9 v1.0 B0003 | CCAFS LC-40 | Dragon Spacecraft Qualification Unit | 0 | LEO | SpaceX | Success |
| 08-12-2010 | 15:43:00 | F9 v1.0 B0004 | CCAFS LC-40 | Dragon demo flight C1, two CubeSats, barrel of Brouere cheese | 0 | LEO (ISS) | NASA (COTS) NRO | Success |
| 22-05-2012 | 07:44:00 | F9 v1.0 B0005 | CCAFS LC-40 | Dragon demo flight C2 | 525 | LEO (ISS) | NASA (COTS) | Success |
| 08-10-2012 | 00:35:00 | F9 v1.0 B0006 | CCAFS LC-40 | SpaceX CRS-1 | 500 | LEO (ISS) | NASA (CRS) | Success |
| 01-03-2013 | 15:10:00 | F9 v1.0 B0007 | CCAFS LC-40 | SpaceX CRS-2 | 677 | LEO (ISS) | NASA (CRS) | Success |

# Total Payload Mass

- Calculate the total payload carried by boosters from NASA
- Used the ‘SUM()’ function to return and display the total sum of ‘PAYLOAD_MASS_KG’ column for Customer ‘NASA(CRS)’

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

```
%sql SELECT SUM(PAYLOAD_MASS__KG_) as "Total Payload Mass(Kgs)", Customer FROM 'SPACEXTBL' WHERE Customer = 'NA
```

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

| Total Payload Mass(Kgs) | Customer |
|---|---|
| 45596 | NASA (CRS) |

# Average Payload Mass by F9 v1.1

- Calculate the average payload mass carried by booster version F9 v1.1
- Used the 'AVG()' function to return and display the average payload mass carried by booster version F9 v1.1

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

```
%sql SELECT AVG(PAYLOAD_MASS__KG_) as "Payload Mass Kgs", Customer, Booster_Version FROM 'SPACEXTBL' WHERE Boos
```

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

| Payload Mass Kgs | Customer | Booster_Version |
|---|---|---|
| 2534.6666666666665 | MDA | F9 v1.1 B1003 |

# First Successful Ground Landing Date

- Find the dates of the first successful landing outcome on ground pad
- Used the ‘MIN()’ function to return and display the first (oldest) date when first successful landing outcome on ground pad ‘Success (ground pad)’happened.

List the date when the first succesful landing outcome in ground pad was acheived.

*Hint:Use min function*

```
%sql SELECT MIN(DATE) FROM 'SPACEXTBL' WHERE "Landing _Outcome" = "Success (ground pad)";
```

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

| MIN(DATE) |
| --- |
| 01-05-2017 |

# 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
- Used 'Select Distinct' statement to return and list the 'unique' names of boosters with operators >4000 and <6000 to only list booster with payloads between 4000-6000 with landing outcome of 'Success (drone ship)'.

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

```
# %sql SELECT * FROM 'SPACEXTBL'
```

```
%sql SELECT DISTINCT Booster_Version, Payload FROM SPACEXTBL WHERE "Landing _Outcome" = "Success (drone ship)"
```

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

| Booster_Version | Payload |
|---|---|
| F9 FT B1022 | JCSAT-14 |
| F9 FT B1026 | JCSAT-16 |
| F9 FT B1021.2 | SES-10 |
| F9 FT B1031.2 | SES-11 / EchoStar 105 |

# Total Number of Successful and Failure Mission Outcomes

- Calculate the total number of successful and failure mission outcomes
- Used the ‘COUNT()’ together with the ‘GROUP BY’ statement to return total number of missions outcomes

List the total number of successful and failure mission outcomes

```
%sql SELECT "Mission_Outcome", COUNT("Mission_Outcome") as Total FROM SPACEXTBL GROUP BY "Mission_Outcome";
```

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

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

# Boosters Carried Maximum Payload

- List the names of the booster which have carried the maximum payload mass
- Using a Subquerry to return and pass the Max payload and used it list all the boosters that have carried the Max payload of 15600kgs

List the names of the booster_versions which have carried the maximum payload mass. Use a subquery

```
%sql SELECT "Booster_Version",Payload, "PAYLOAD_MASS__KG_" FROM SPACEXTBL WHERE "PAYLOAD_MASS__KG_" = (SELECT M
```

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

| Booster_Version | Payload | PAYLOAD_MASS__KG_ |
|---|---|---|
| F9 B5 B1048.4 | Starlink 1 v1.0, SpaceX CRS-19 | 15600 |
| F9 B5 B1049.4 | Starlink 2 v1.0, Crew Dragon in-flight abort test | 15600 |
| F9 B5 B1051.3 | Starlink 3 v1.0, Starlink 4 v1.0 | 15600 |
| F9 B5 B1056.4 | Starlink 4 v1.0, SpaceX CRS-20 | 15600 |
| F9 B5 B1048.5 | Starlink 5 v1.0, Starlink 6 v1.0 | 15600 |
| F9 B5 B1051.4 | Starlink 6 v1.0, Crew Dragon Demo-2 | 15600 |
| F9 B5 B1049.5 | Starlink 7 v1.0, Starlink 8 v1.0 | 15600 |
| F9 B5 B1060.2 | Starlink 11 v1.0, Starlink 12 v1.0 | 15600 |
| F9 B5 B1058.3 | Starlink 12 v1.0, Starlink 13 v1.0 | 15600 |
| F9 B5 B1051.6 | Starlink 13 v1.0, Starlink 14 v1.0 | 15600 |
| F9 B5 B1060.3 | Starlink 14 v1.0, GPS III-04 | 15600 |
| F9 B5 B1049.7 | Starlink 15 v1.0, SpaceX CRS-21 | 15600 |

# 2015 Launch Records

- List the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015
- Used the 'subsrt()' in the select statement to get the month and year from the date column where substr(Date,7,4)='2015' for year and Landing_outcome was 'Failure (drone ship') and return the records matching the filter.

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, 4, 2) as month to get the months and substr(Date,7,4)='2015' for year.**

```
%sql SELECT substr(Date,7,4), substr(Date, 4, 2),"Booster_Version", "Launch_Site", Payload, "PAYLOAD_MASS__KG_"
```

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

| substr(Date,7,4) | substr(Date, 4, 2) | Booster_Version | Launch_Site | Payload | PAYLOAD_MASS__KG_ | Mission_Outcome | Landing _Outcome |
|---|---|---|---|---|---|---|---|
| 2015 | 01 | F9 v1.1 B1012 | CCAFS LC-40 | SpaceX CRS-5 | 2395 | Success | Failure (drone ship) |
| 2015 | 04 | F9 v1.1 B1015 | CCAFS LC-40 | SpaceX CRS-6 | 1898 | Success | Failure (drone ship) |

# 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

Rank the count of successful landing_outcomes between the date 04-06-2010 and 20-03-2017 in descending order.

```
%sql SELECT * FROM SPACEXTBL WHERE "Landing _Outcome" LIKE 'Success%' AND (Date BETWEEN '04-06-2010' AND '20-03
```

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

| Date | Time (UTC) | Booster_Version | Launch_Site | Payload | PAYLOAD_MASS__KG_ | Orbit | Customer | Mission_C |
|---|---|---|---|---|---|---|---|---|
| 19-02-2017 | 14:39:00 | F9 FT B1031.1 | KSC LC-39A | SpaceX CRS-10 | 2490 | LEO (ISS) | NASA (CRS) | |
| 18-10-2020 | 12:25:57 | F9 B5 B1051.6 | KSC LC-39A | Starlink 13 v1.0, Starlink 14 v1.0 | 15600 | LEO | SpaceX | |
| 18-08-2020 | 14:31:00 | F9 B5 B1049.6 | CCAFS SLC-40 | Starlink 10 v1.0, SkySat-19, -20, -21, SAOCOM 1B | 15440 | LEO | SpaceX, Planet Labs, PlanetIQ | |
| 18-07-2016 | 04:45:00 | F9 FT B1025.1 | CCAFS LC-40 | SpaceX CRS-9 | 2257 | LEO (ISS) | NASA (CRS) | |
| 18-04-2018 | 22:51:00 | F9 B4 B1045.1 | CCAFS SLC-40 | Transiting Exoplanet Survey Satellite (TESS) | 362 | HEO | NASA (LSP) | |
| 17-12-2019 | 00:10:00 | F9 B5 B1056.3 | CCAFS SLC-40 | JCSat-18 / Kacific 1, Starlink 2 v1.0 | 6956 | GTO | Sky Perfect JSAT, Kacific 1 | |
| 16-11-2020 | 00:27:00 | F9 B5B1061.1 | KSC LC-39A | Crew-1, Sentinel-6 Michael Freilich | 12500 | LEO (ISS) | NASA (CCP) | |

Section 3

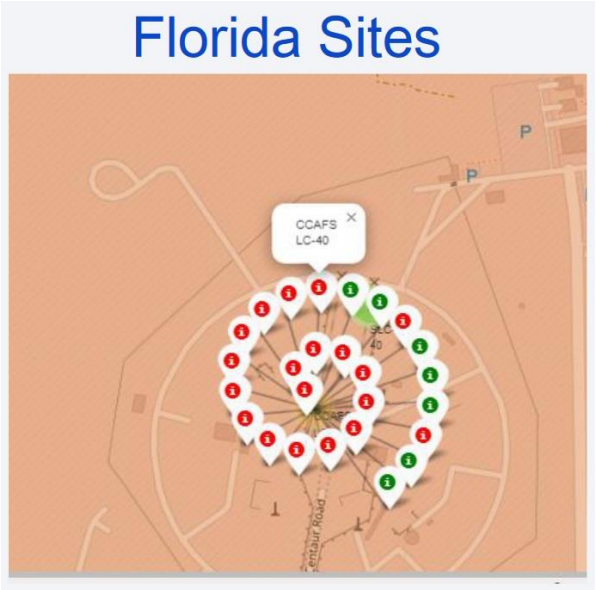
# Launch Sites Proximities Analysis

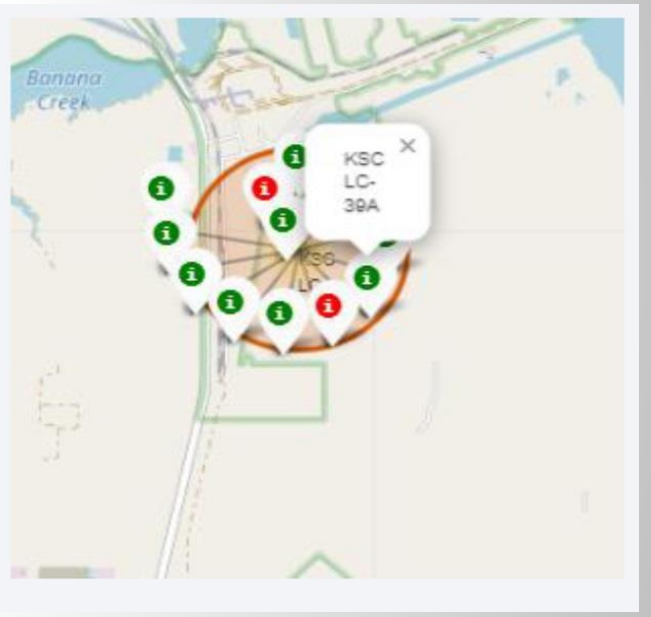
# Markers of all launch sites on global map

- All launch sites are in proximity to the Equator, (located southwards of the US map). Also all the laumch sites are in very close proximity to the coast.

# Launch outcomes for each site on the map With Color Markers

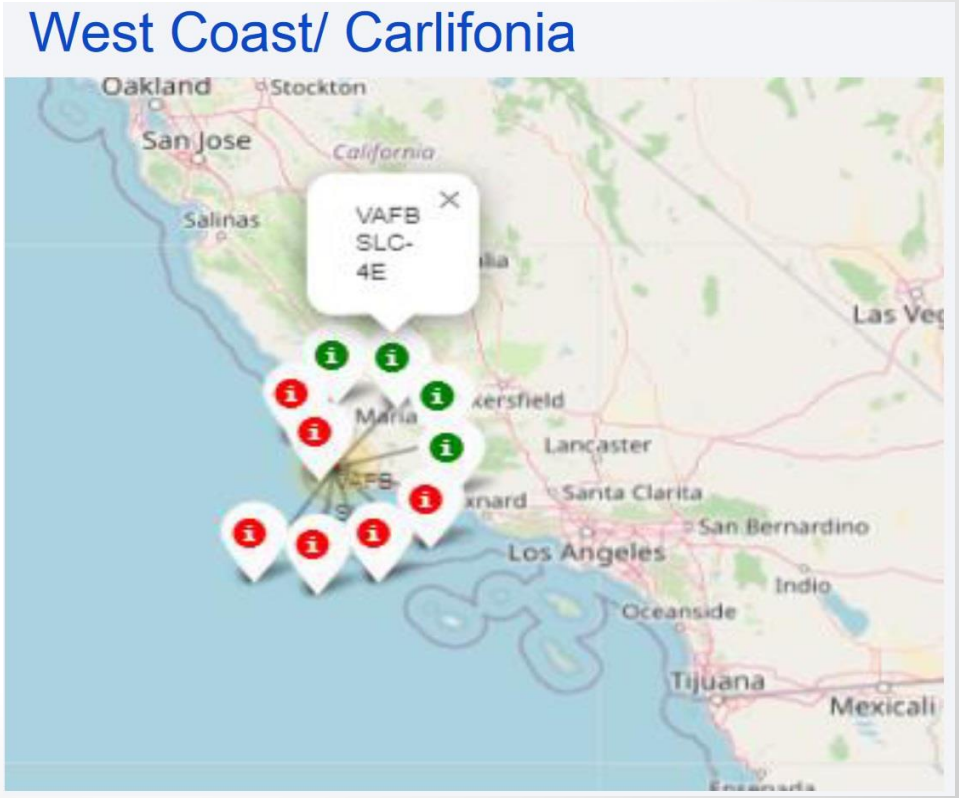
- In the Eastern coast (Florida) Launch site KSC LC-39A has relatively high success rates compared to CCAFS SLC-40 & CCAFS LC-40.

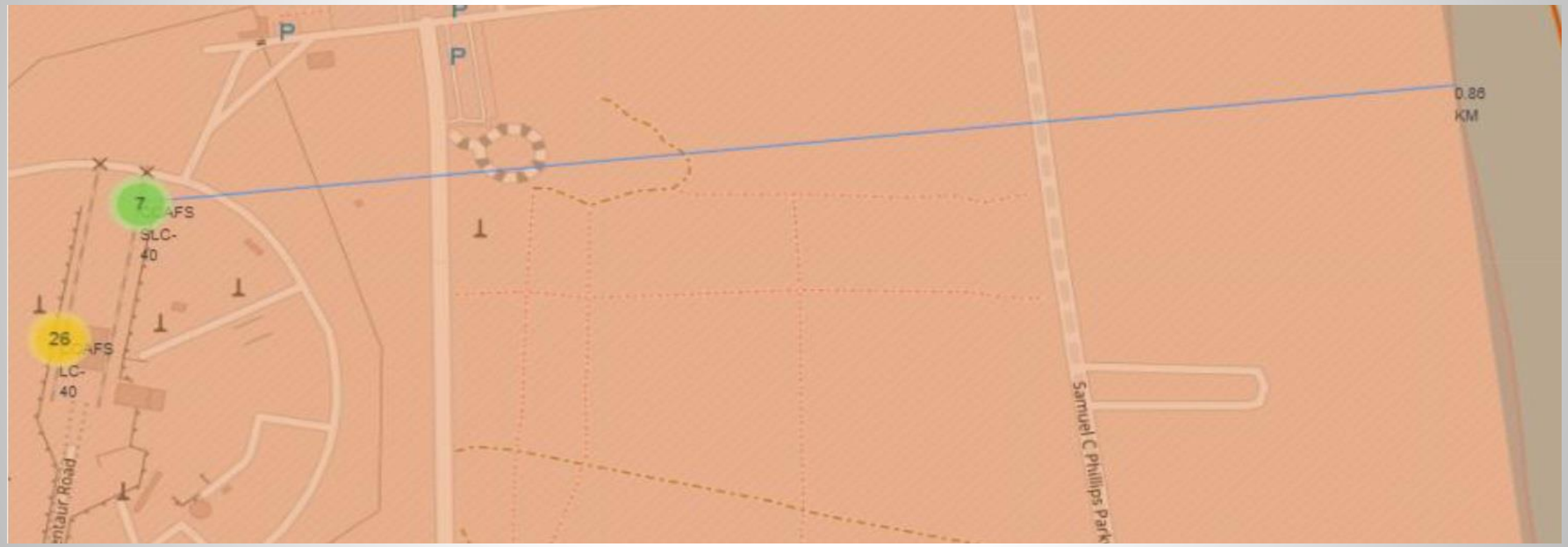
# Launch outcomes for each site on the map With Color Markers

- In the West Coast (California) Launch site VAFB SLC-4E has relatively lower success rates 4/10 compared to KSC LC-39A launch site in the Eastern Coast of Florida.

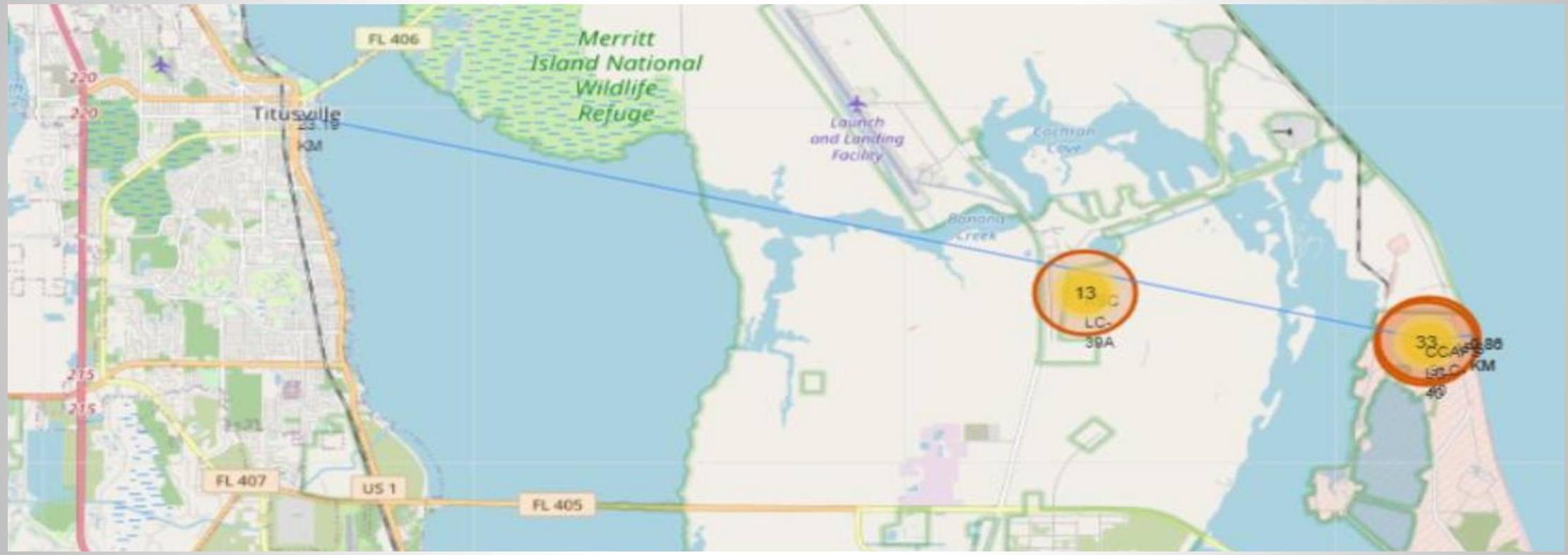
# Distances between a launch site to its proximities

- Launch site CCAFS SLC-40 proximity to coastline is 0.86km



# Distances between a launch site to its proximities

- Launch site CCAFS SLC-40 closest to highway (Washington Avenue) is 23.19km

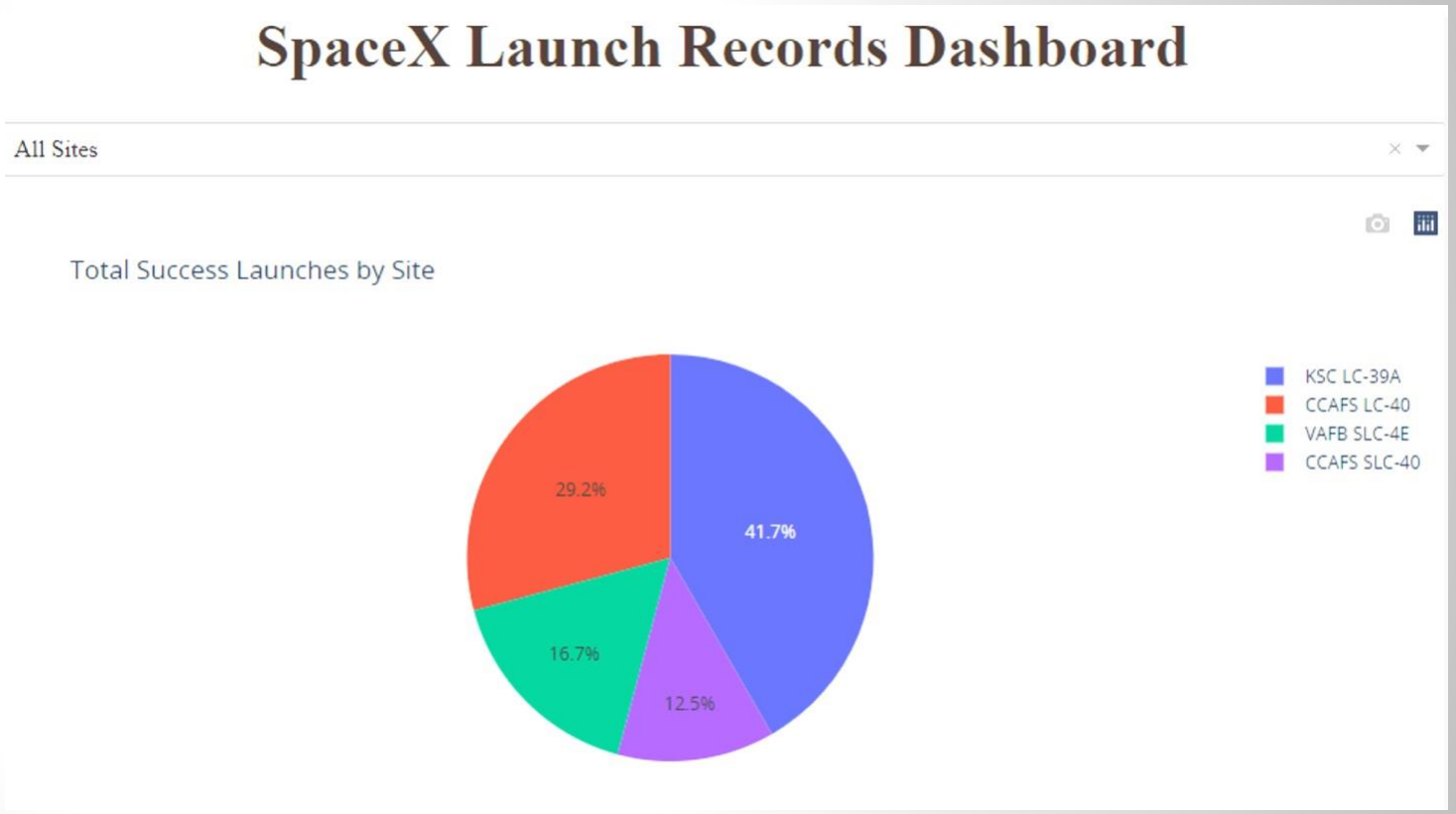


Section 4

# Build a Dashboard with Plotly Dash

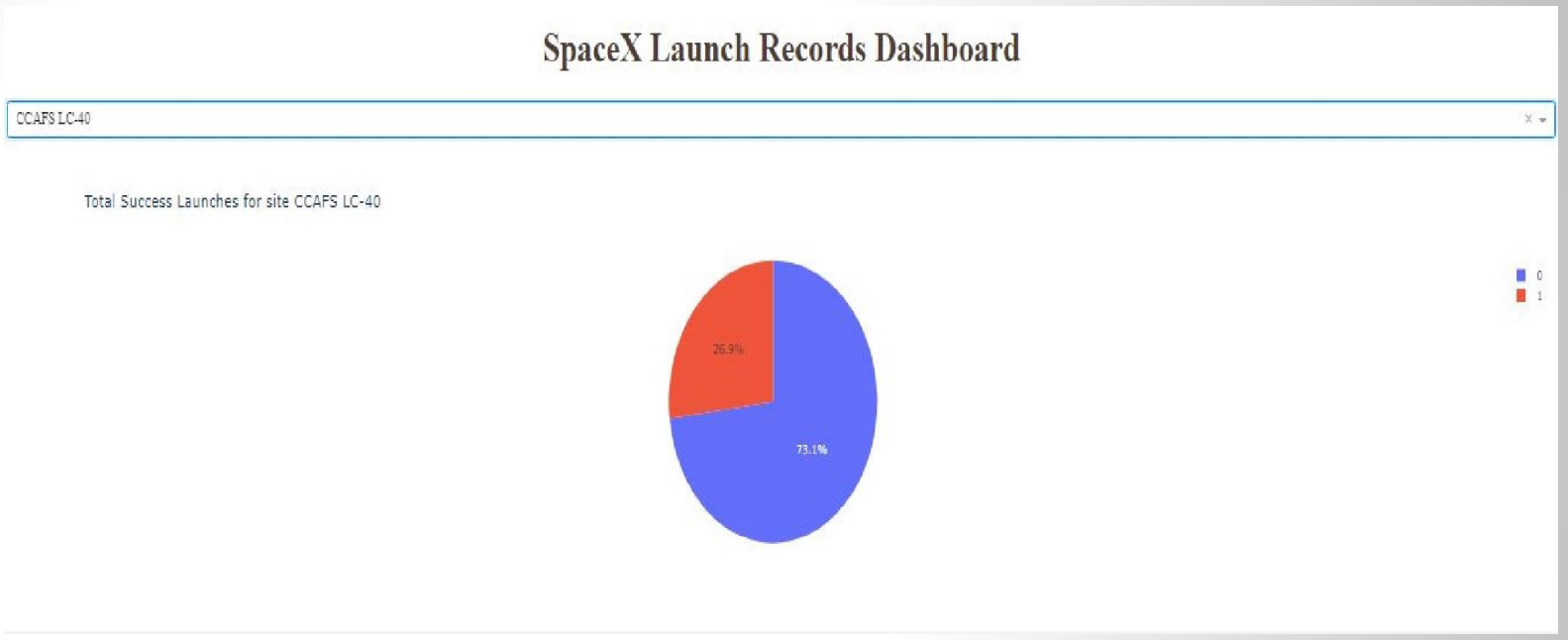
# Pie-Chart for launch success count for all sites

- Launch site KSC LC-39A has the highest launch success rate at 42% followed by CCAFS LC-40 at 29%, VAFB SLC-4E at 17% and lastly launch site CCAFS SLC-40 with a success rate of 13%



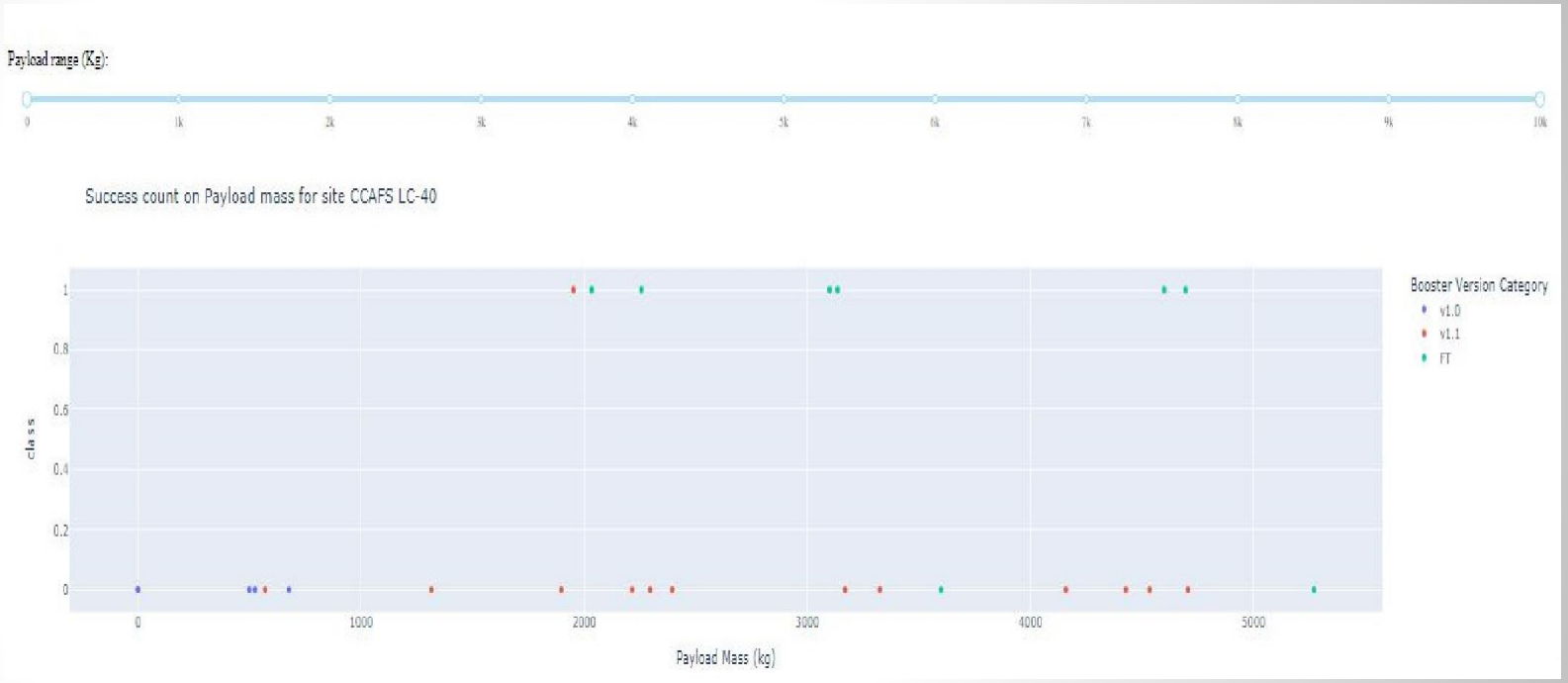
# Pie chart for the launch site with 2nd highest launch success ratio

- Launch site CCAFS LC-40 had the 2nd highest success ratio of 73% success against 27% failed launches



# Payload vs. Launch Outcome scatter plot for all sites

- For Launch site CCAFS LC-40 the booster version FT has the largest success rate from a payload mass of >2000kg



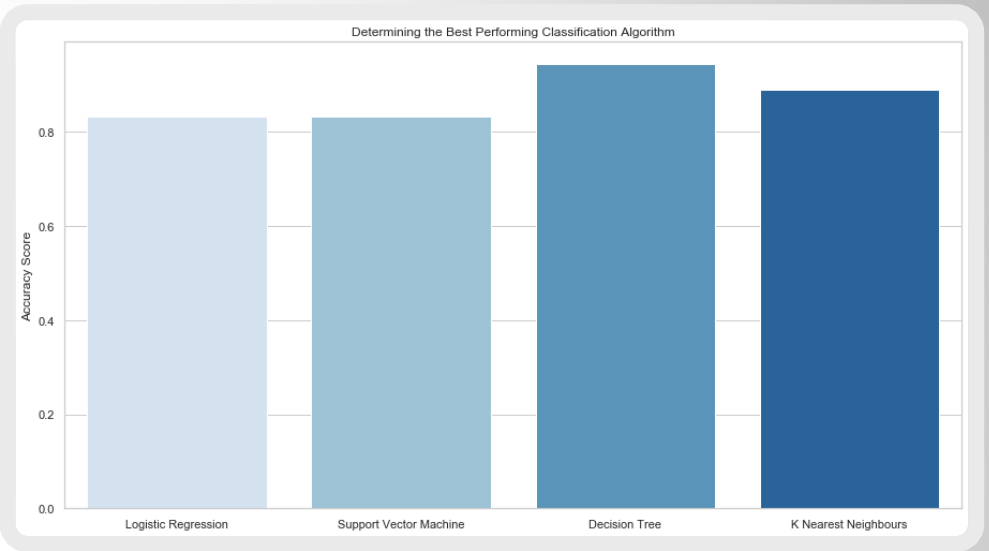
Section 5

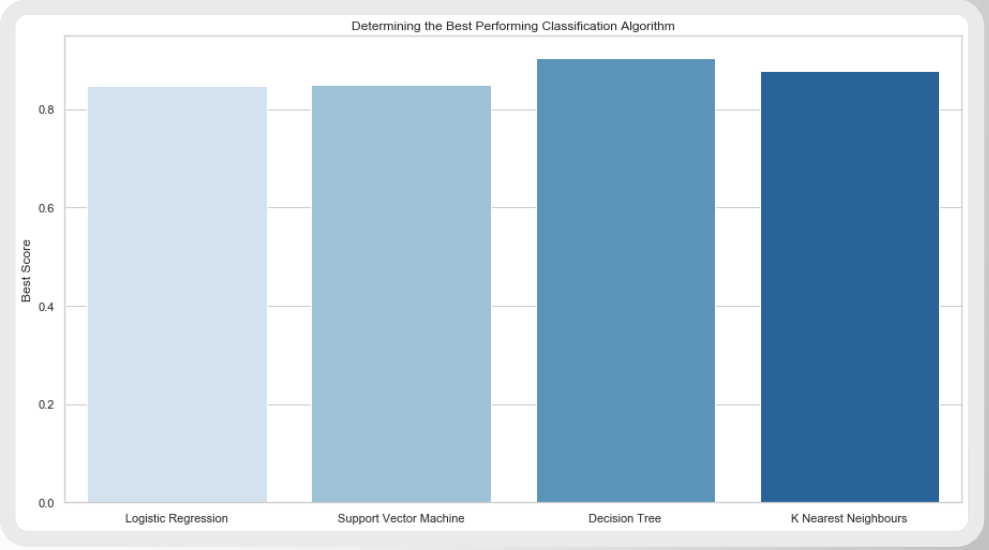
# Predictive Analysis (Classification)

# Classification Accuracy

- Plotting the Accuracy Score and Best Score for each classification algorithm produces the following result:
- The Decision Tree model has the highest classification accuracy
  - The Accuracy Score is 94.44%
  - The Best Score is 90.36%

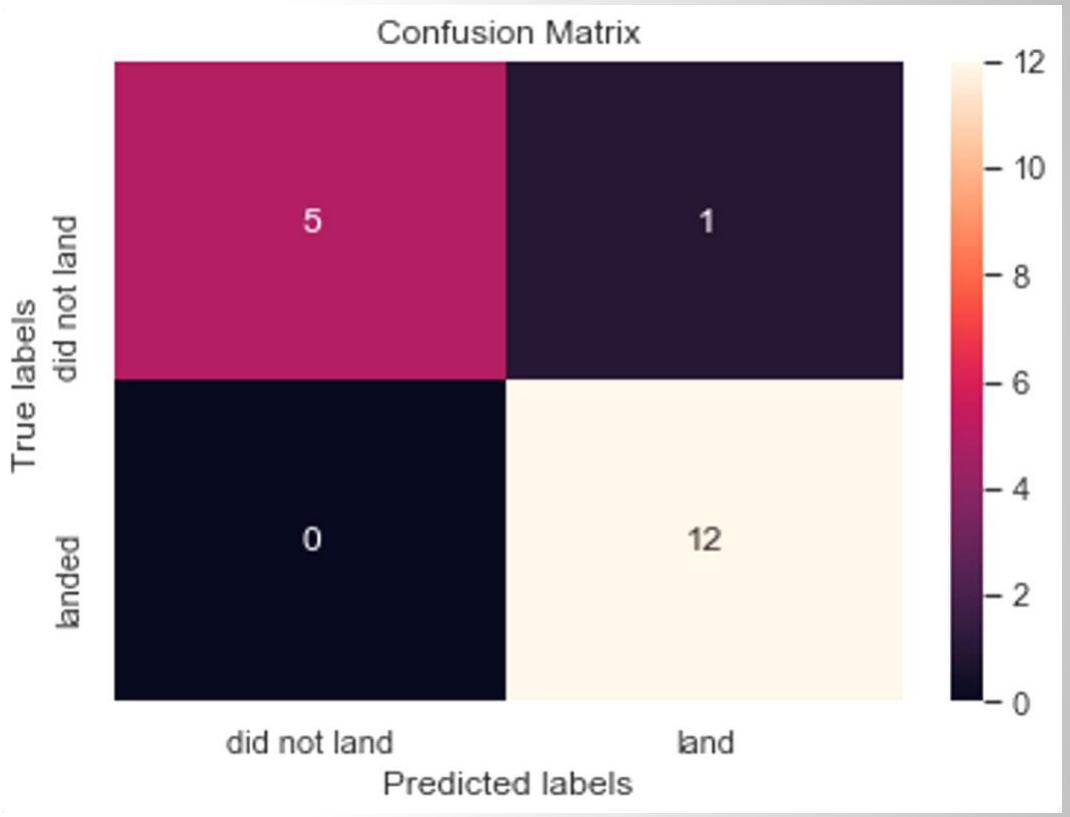
| | Algorithm | Accuracy Score | Best Score |
|---|---|---|---|
| 0 | Logistic Regression | 0.833333 | 0.846429 |
| 1 | Support Vector Machine | 0.833333 | 0.848214 |
| 2 | Decision Tree | 0.944444 | 0.903571 |
| 3 | K Nearest Neighbours | 0.888889 | 0.876786 |



# 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%.
- With heavy payloads the successful landing or positive landing rate are more for Polar,LEO and ISS. However for GTO we cannot distinguish this well as both positive landing rate and negative landing(unsuccessful mission) are both there here

# Appendix

- DATA COLLECTION – space x REST api
- Custom functions to retrieve the required information
- Custom logic to clean the data

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'])
```

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'])
```

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'])
```

- DATA COLLECTION – WEB SCRAPING
- Custom functions for web scraping
- Custom logic to fill up the launch_dict values with values from the launch tables

```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
```

# Thank you!