Winning Space Race with Data Science

Gregory A. Moore
November 12th, 2022

Outline

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

Executive Summary

For the purpose of building a prediction model to predict if the SpaceX Falcon 9 (first stage) will land successfully, based on knowledge of past launches, several data analysis data collection and data wrangling techniques were put into play to create a usable Falcon 9 dataset.

To collect the needed data, the SpaceX API was used to return public SpaceX data in the form of a JSON object. The Python **BeautifulSoup** package was then used to web scrape HTML tables from the SpaceX Wikipedia page. Both of these actions had specific steps to target individual data points while using Python built functions and to parse the Falcon 9 records from the full population of SpaceX launches. Additional steps had to be carried out to handle missing and/or incomplete data.

After having a solid starting point, in terms of data, it became possible to start analyzing the information using the Python libraries **Pandas** and **NumPy**. The most important aspect of this first analysis was to sort through the eight different Falcon 9 landing outcome possibility categories and then create a new "Class" categorical value where a 1 represented a successful landing and a 0 represented a failure. This newly created value was then added to the main data frame so that it could be used as the prediction variable in the created data models.

Executive Summary (continued)

The next phase of the EDA (Exploratory Data Analysis) came by way of creating a **SQLite** database so that drilled-in explorations could be made within the dataset using SQL queries. This gave us insight on distinct launch sites, average payloads, successful launch outcome ratios, and many other significant facts about the Falcon 9 missions.

After using SQL to gain insight into the Falcon 9 dataset, the **Matplotlib** plotting library and the **Seaborn** visualizations library were used to extend the exploratory data analysis phase of the project by creating color coded scatter plots, bar charts, and line charts. This quickly provided information on which collected data points have potential to have direct correlation to successful or failed launches. Payload Mass (kg), Launch Sites, Orbit type, and the year of the individual launches were heavily explored.

A continuation of using visualizations to study the Falcon 9 data occurred by building interactive maps using **Folium** to take advantage of available geospatial data points collected throughout this process. Maps were created to show exact locations of the Falcon 9 launch sites, markers of each site's successes and failures, and exact distances between the launch sites and other relevant locations in the area (e.g. cities, coasts, highways, railroads, etc.). After this, the Python **Plotly Dash** library package was used to create a dashboard application. This app made use of a dropdown picker methodto allow individual launch sites to be selected and a range slider for Payload Mass (Kg) ranges to be quickly toggled. All of which allowed for a quick way to see success ratios by location, Payload Mass (Kg), and booster versions.

Executive Summary (continued)

All of the exploratory actions briefly explained above were for the sole purpose of creating the most accurate prediction model possible. The insight gained during these exercises allowed for the data to be manipulated in a way that allowed it to be plugged into the various created machine learning objects. The first step in this process was to standardize the data so that it could be plugged into the various created models. After this the **Sklearn** *train_test_split()* methodwas used to randomly split the data into a TRAIN dataset and a smaller TEST dataset. The TRAIN dataset was used to create the data models and then the TEST dataset was used to gauge the accuracy of the created objects.

Each model used a **Sklearn** *GrideSearchCV* functionality to locate the best parameters for the created model and then each model's accuracy was tested using the *best_score_* data attribute. After these steps were completed, the TEST datasets were used to determine how the created models interacted with new data and a confusion matrix was used to quickly visualize how well the model handled the TEST data.

While trying to create the most accurate model possible, while using the SpaceX dataset's TRAIN split, a baseline Logistic Regression model was created and scored. This first model was then followed by Support Vector Machine, Decision Tree, and K-Nearest Neighbor model methodologies.

Executive Summary (continued)

After each model was created, the TEST split data was plugged into the four models and a final R-Squared (R^2) value was calculated. While each model's accuracy varied while using the TRAIN data, the final accuracy score using the TEST data was consistent across all four model types with each producing an R^2 rating of 0.83334.

Though this 0.83334 accuracy score is a respectable value for the created models, it is extremely important to remember that the source data used to create the models, before the TRAIN/TEST splits, had less than one hundred rows. Without a larger sample size, it is difficult to determine if overfitting occurred while creating the models with the TRAIN dataset.

Introduction

Project background

When things go as expected, SpaceX's Falcon 9 rockets can be launched for an estimated sixty two million dollars ($62M). This cost is close to one hundred million dollars ($100M) cheaper than SpaceX's competition and this fact makes it very difficult for other space transportation companies to compete when submitting bids to private corporations and governments. As SpaceY is new to the privatized rocket industry, it is clear that we will need to be strategic when competing with SpaceX and to do that we will need to use every resource available.

One readily available resource is SpaceX's overconfidence. SpaceX launch data and launch costs are widely available and this availability presents an opportunity for its competitors to explore ways to compete with the Falcon 9 rocket.

Introduction (continued)

Problem Statement

The difference between SpaceX and other rocket manufactures is that the Falcon 9's first stage can safely return to the earth after the second stage separates with the payload. This allows for SpaceX to reuse the first stage and it allows them to have quicker turnarounds between launches.

All of these cost saving traits are dependent on the successful landing of the Falcon 9's first stage and the topics covered in this data findings report will concentrate on building a data model to accurately predict if a launch will land successfully or if SpaceX will have to cover the cost of repairing or fully replacing a Falcon 9 rocket.

Section 1

Methodology

Methodology

- Data collection methodology:
  - Requested rocket launch data via the SpaceX API
  - Web scraping of the SpaceX Wikipedia page (HTML tables)
- Perform data wrangling
  - Locate and mitigate missing values
  - Create "Outcome" binary categorical variable
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - Manipulate, Standardize, and Split (TRAIN & TEST) the data
  - Fit and Score the best parameters

Data Collection - SpaceX API

- Used the REQUESTS Python library to make HTTP requests to the SpaceX API (api.spacexdata.com/v4/) to target specific information in the following SpaceX API endpoints:
  - /launches/past
  - /rockets
  - /launchpads
  - /payloads
  - /cores
- The response to the *"api.spacexdata.com/v4/launches/past"* API endpoint returned a massive JSON object

Data Collection - SpaceX API

- Used the PANDAS, NUMPY, and DATETIME Python libraries to Normalize the JSON response into a Pandas dataframe and to manipulate the data.

- Removed all columns from the created dataframe except for 'rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc'

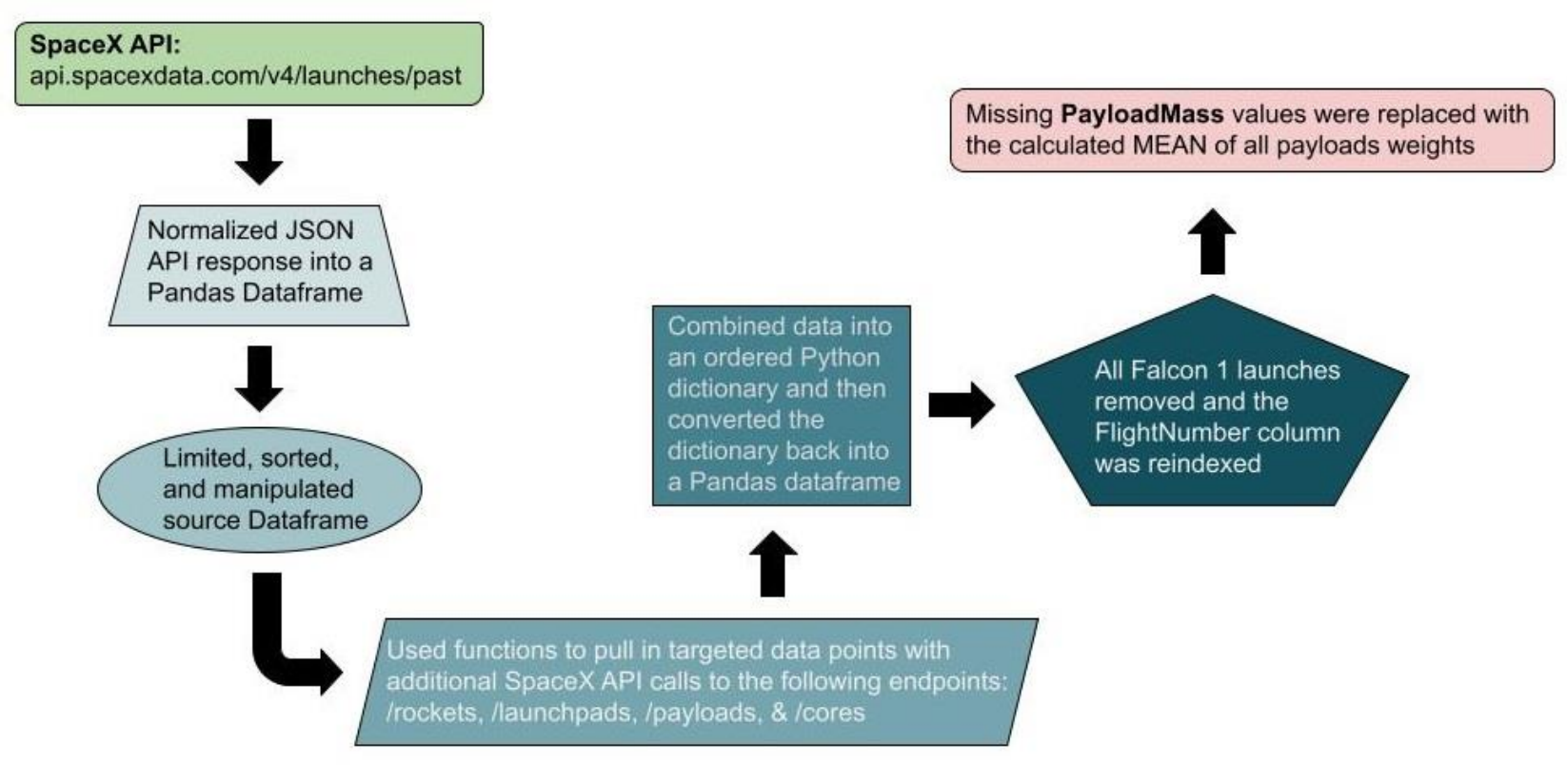
- Removed launches where multiple cores were used, converted nested lists to single entries, converted datetime datatype to date only (YYYY-MM-DD), and limited launch dates to be less than or equal to 2020-11-13.

Data Collection - SpaceX API

- Used created Functions, ID values within the dataframe, and targeted API calls to pull additional information for the collected dataframe.
  - This data was then combined into an ordered Python dictionary.
- The dictionary was then converted back into a Pandas dataframe
- All Falcon 1 launches were removed and the FlightNumber column was reindexed to account for the removed rows.
- Missing PayloadMass values (5 total) were then located and dealt with accordingly replacing the NULLS with the calculated MEAN of all play load weights

Data Collection – SpaceX API

- GitHub URL: SpaceX API notebook & SpaceX API Python File



Note: See Appendix A for a data sample from this step

Data Collection – Web Scraping

- Used the REQUESTS Python library to send an HTTP GET requests to the "*List of Falcon 9 and Falcon Heavy launches*" Wikipedia page
  - https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922
- Used the *BeautifulSoup()* methodto create a structured BeautifulSoup object from the response of the GET call.
- Located all tables within the BeautifulSoup object using the *FindAll("table")* command

Data Collection – Web Scraping

- Iterated through the BeautifulSoup object to extract the column headers by targeting the HTML "th" tags with the *extract_column_from_header()* function
  - This step also included removing any columns without a value (i.e. NULLS) or empty spaces

- While using the extracted column headers and the *dict.fromkeys()* function, an empty Python dictionary was created using the column names as the dictionary's keys. This step also included removing the "Date and time (UTC)" column and the addition of the following new columns.
  - Version Booster
  - Booster landing
  - Date
  - Time

Data Collection – Web Scraping

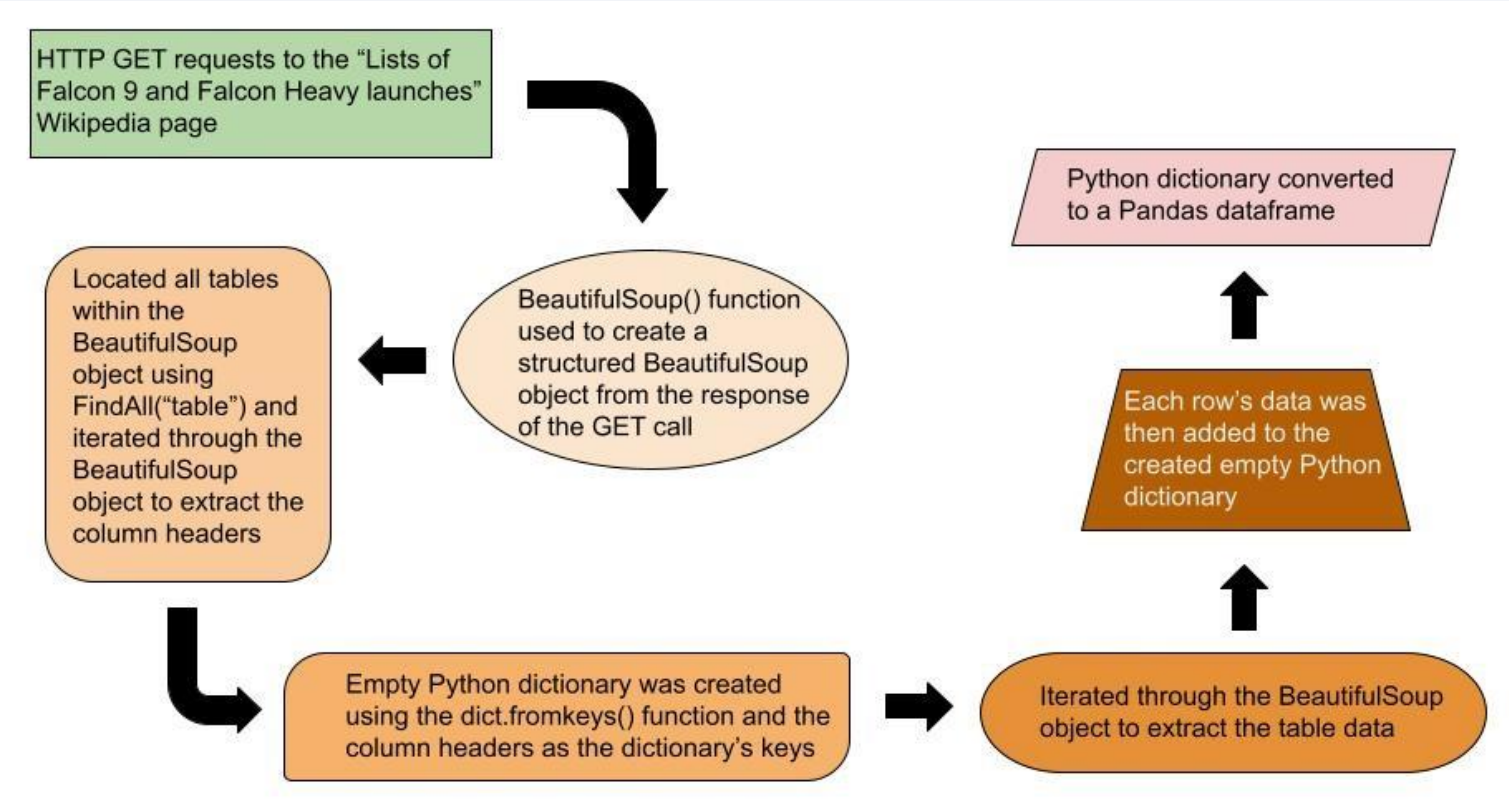
- Iterated through the BeautifulSoup object to extract each row from all of the collected tables in the HTTP GET request's response.

- Each row's data was then added to the created empty Python dictionary while using the table's column headers to match the keys within the dictionary.
  - Five prebuilt functions were used in during this step.
  - These functions helped target the data and to put it into a more friendly format

- The now populated Python dictionary was then converted to a Pandas dataframe that contained 121 rows with 11 columns

Data Collection – Web Scraping

- GitHub URL: Web Scraping notebook



Note: See Appendix B for a data sample from this step

Data Wrangling

- Used the Pandas library to load the dataset created during the SpaceX API data collection step into a Pandas dataframe.

- Calculated the percentage of missing values and determined the data type for each column (e.g. int64, float64, bool, etc.).

- Used the Pandas *value_counts()* methodto get the count total for each LaunchSite, Orbit, and Outcome (LaunchSite example below)

```
# Apply value_counts() on column LaunchSite

df.LaunchSite.value_counts()
```

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

Data Wrangling

- Loaded the *df.Outcome.value_counts()* results into a new **landing_outcomes** variable.

- Created a **bad_outcomes** variable by assigning each **landing_outcomes** category a numerical "key" value and then targeting the unsuccessful landings by targeting the following key categories [1,3,5,6,7]:

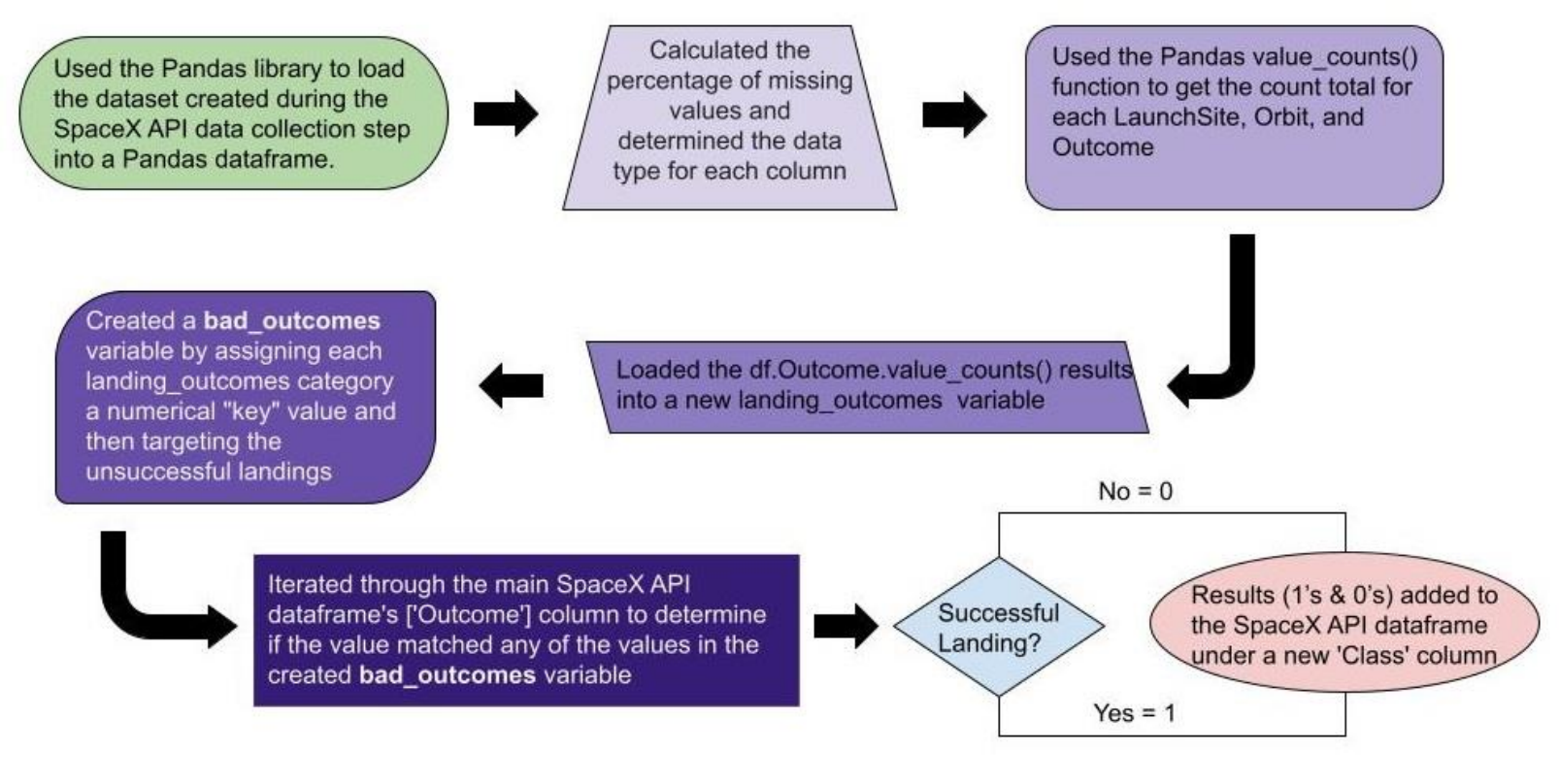
0 True ASDS
1 None None
2 True RTLS
3 False ASDS
4 True Ocean
5 False Ocean
6 None ASDS
7 False RTLS

Data Wrangling

- Iterated through the main SpaceX API dataframe's ['Outcome'] column to determine if the value matched any of the values in the created **bad_outcomes** variable.
  - A new **landing_class** variable was created by assigning a '0' if a match was made with the collected categories in the **bad_outcomes** variable.
  - For Outcomes not in the **bad_outcomes** variable, a value of '1' was assigned.
- The results collected in the **landing_class** variable was then added to the SpaceX API dataframe under a new 'Class' column.
- Using this new 'Class' column, determining the landing success rate for the dataset was now easily calculated by way of the *mean()* function.

Data Wrangling

- GitHub URL: [Data Wrangling notebook]



Note: See Appendix C for data sample from this step

EDA with SQL

- Using the Pandas, iPython-SQL, and SQLite3 Python libraries, a tablespace named "SPACEXTBL" was created using a version of the web scrapped dataset (Wikipedia).
  - This step effectively created a temporary table named SPACEXTBL that could be queried directly in a Jupyter notebook using standard SQL syntax.
- As launch site location was a major component in every SpaceX launch, the first query run was done using a DISTINCT in the SELECT clause and the statement produced four locations (results below):

```
res = cur.execute(
    """
    SELECT Distinct Launch_Site
    FROM SPACEXTBL
    """)
res.fetchall()
```

```
[('CCAFS LC-40',), ('VAFB SLC-4E',), ('KSC LC-39A',), ('CCAFS SLC-40',)]
```

EDA with SQL

The next area of interests made use of SUM() and AVG() functions while targeting the **PayloadMass** values for specific scenarios.

In order to get a basic understanding of SpaceX's relationship with its clients, a statement was created to targeted the total **PayloadMass** (Kilograms) for one of its most well-known customers: *NASA*.

```
res = cur.execute(
    """
    SELECT SUM(PAYLOAD_MASS__KG_)
    FROM SPACEXTBL
    WHERE Customer = 'NASA (CRS)'
    """)
res.fetchall()
```

```
[(45596,)]
```

The next statement was run to help get a better understanding of the average **PayloadMass** (Kilograms) for a Booster Version. For this statement, an earlier booster version was selected.

```
res = cur.execute(
    """
    SELECT AVG(PAYLOAD_MASS__KG_)
    FROM SPACEXTBL
    WHERE Booster_Version = 'F9 v1.1'
    """)
res.fetchall()
```

```
[(2928.4,)]
```

EDA with SQL

- The next statement was to discover when the first successful Ground Pad landing occurred. This first required converting the **Date** value from a basic text field into a usable date field. This action was done using SQLite's *date()* methodand a the aggregate method*min()*.

```
res = cur.execute(
    """
    select MIN(date(substr(Date, 7, 4) || '-' || substr(Date, 4, 2) || '-' || substr(Date, 1, 2)))
    from SPACEXTBL
    WHERE "Landing _Outcome" = 'Success (ground pad)'
    """)
res.fetchall()
```

```
[('2015-12-22',)]
```

EDA with SQL

- It was then decided to explore the **Mission_Outcome** field to gauge the total counts of successful and failed mission outcomes. This was done by using a *Count()* methodwhile using a "Grouping By" clause to count each Mission_Outcome category.
  - The output produced by the below statement is somewhat confusing. The data clearly indicates that not all **Landing_Outcomes** were successful, but SpaceX clearly viewed 100 launches to be productive missions. One would likely need to understand each mission's true purpose to use this data effectively.

```
res = cur.execute(
    """
    SELECT TRIM(Mission_Outcome), Count(Mission_Outcome)
    FROM SPACEXTBL
    Group by TRIM(Mission_Outcome)
    """)
res.fetchall()
```

```
[('Failure (in flight)', 1),
 ('Success', 99),
 ('Success (payload status unclear)', 1)]
```

EDA with SQL

- The last significant SQL statement run was to determine number of successful **Landing_Outcomes** between a specific date range: June 4th, 2010 – March 20th, 2017
  - This statement's output shows there were only 8 (out of 31) total successful landings during this span.

```
res = cur.execute(
    """
    SELECT
    "Landing _Outcome",
    Count("Landing _Outcome") as LandingCount,
    RANK () OVER (
        PARTITION BY "Landing _Outcome"
        ORDER BY Date DESC
    ) LengthRank
    FROM
    SPACEXTBL
    WHERE 1=1
    AND "Landing _Outcome" like '%Success%'
    AND date(substr(Date, 7, 4) || '-' || substr(Date, 4, 2) || '-' || substr(Date, 1, 2))
              BETWEEN DATE('2010-06-04') AND DATE('2017-03-20')
    GROUP BY "Landing _Outcome"
    Order by LandingCount desc
    """)
res.fetchall()
```

```
[('Success (drone ship)', 5, 1), ('Success (ground pad)', 3, 1)]
```

EDA with SQL

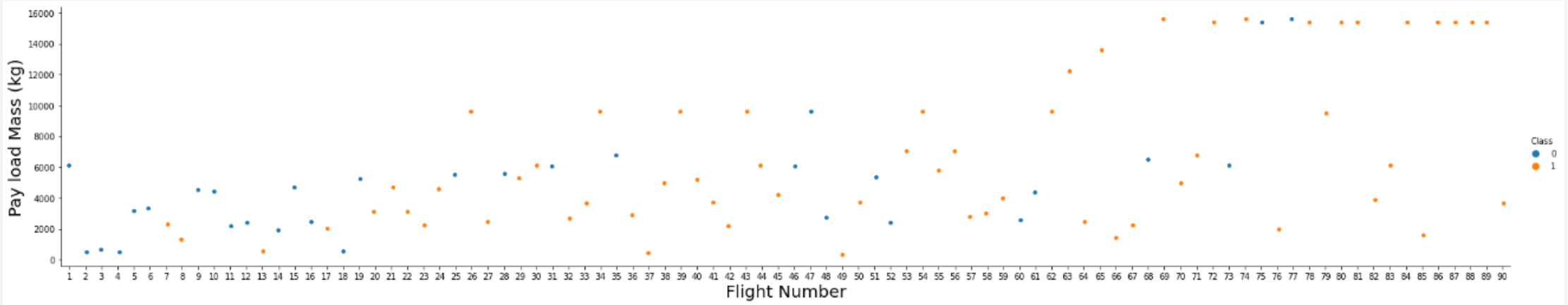
- GitHub URL: SQLite notebook

- **Note:** There were several other less significant SQL statements run during this exploratory data analysis (EDA) phase. These additional statements, and their results, can be found in Appendix D

EDA with Data Visualization

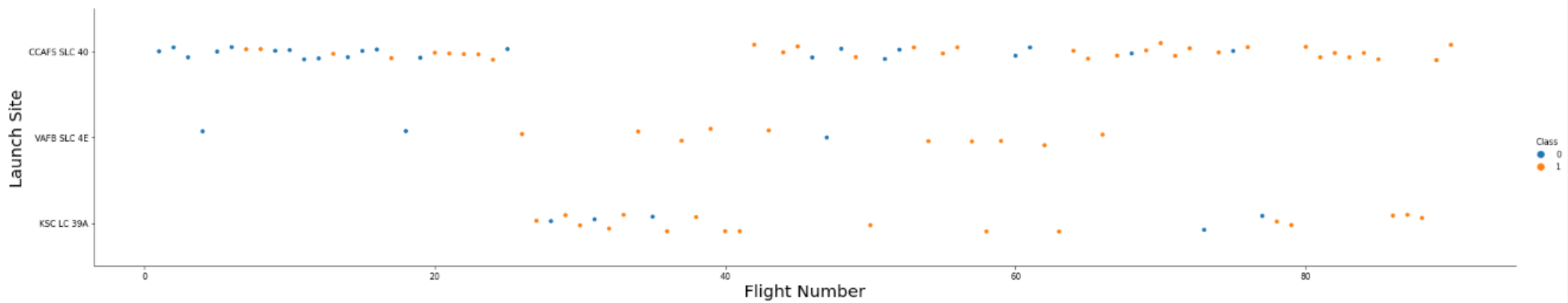
Note: *All steps done in this Data Visualization section were done while working within the latest SpaceX API dataset (i.e. contains the newly added "Class" column).*

- Used Python Matplotlib and Seaborn libraries to create a scatterplot of **PayloadMass** by **FlightNumber** while using the **Class** as a color indicator of the first stage successfully landing.
    - As the **FlightNumber** mirrors the ascending order launch **Date** value captured for each launch, this visualization provides a timeline of launches without fixating on the actual date of the launch.



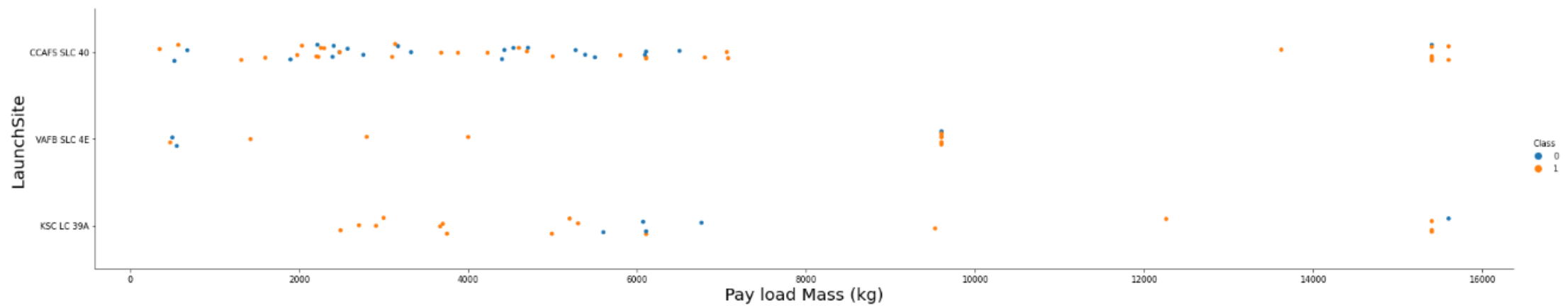
EDA with Data Visualization

- The next created a scatterplot also used **FlightNumber**, while still using the **Class** as a color indicator, but **LaunchSite** replaced **PayloadMass** used for the Y-axis.
  - This visualization helps to conceptualize the success rates for the three launch sites while maintaining the timeline structure of the previous plot.
  - It also helped demonstrate that site *CCAFS SLC 40* had the most launch attempts of the three sites and that this was especially true in the beginning of the SpaceX program.



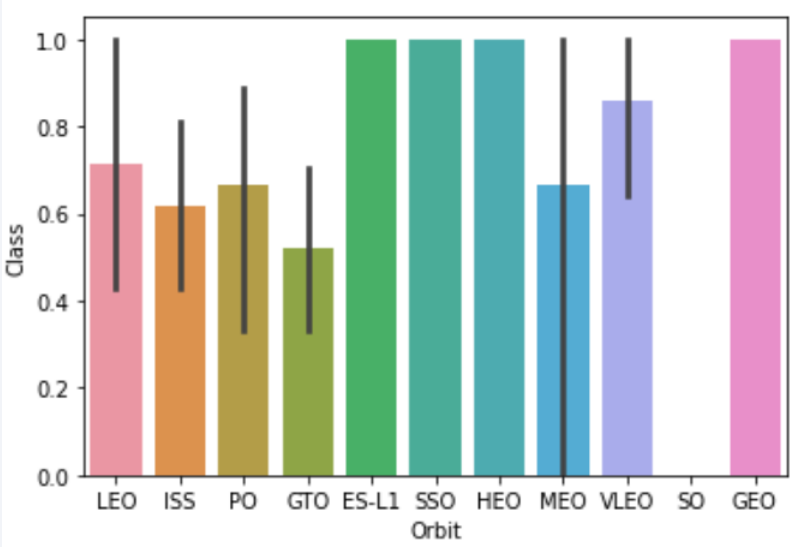
EDA with Data Visualization

- Using insights gained in the previous visualizations, the next scatterplot created used **LaunchSite** for the Y-axis and **PayloadMass** used for the X-axis.
  - This plot not only showed successful payloads attempted at each site (by size), but it also showed that launch site *VAFB SLC 4E* did not have any launches with a **PayloadMass** value above 10,000 kg
  - More importantly, this plot also shows that there were only three unsuccessful landings, out of while having a **PlayloadMass** value greater than 7,000 kg.



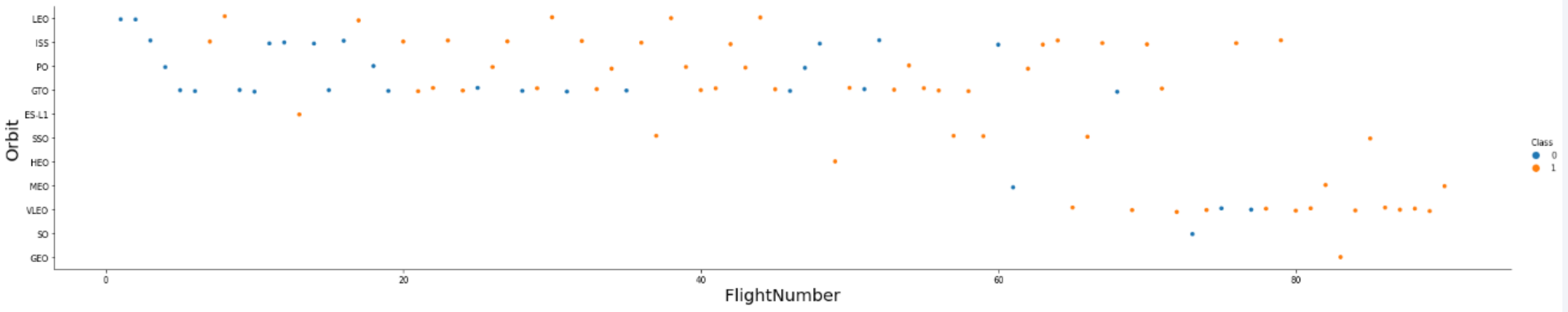
EDA with Data Visualization

- After exploring launch sites and payloads, the **Orbit** became the next area of study.
- As **Orbit** had eleven categorical classes, the first look at this data point was done while using a bar chart.
  - This visualization shows success averages per Orbit, but it lacks a sense of scale and clarity. Without any count totals, it is impossible to discern the significance of ES-L1, SSO, HEO, GEO's 100% success rates or SO's 0% average.



EDA with Data Visualization

- In order to answer some of the **Orbit** related questions not answered by the previous bar chart, **FlightNumber** was added back into the equation in a scatter plot.
  - Here we see that ES-L1, SSO, HEO, GEO & SO's success rates were indeed bloated by the low number of attempts
  - We can also see that MEO also had a low number of launches



EDA with Data Visualization

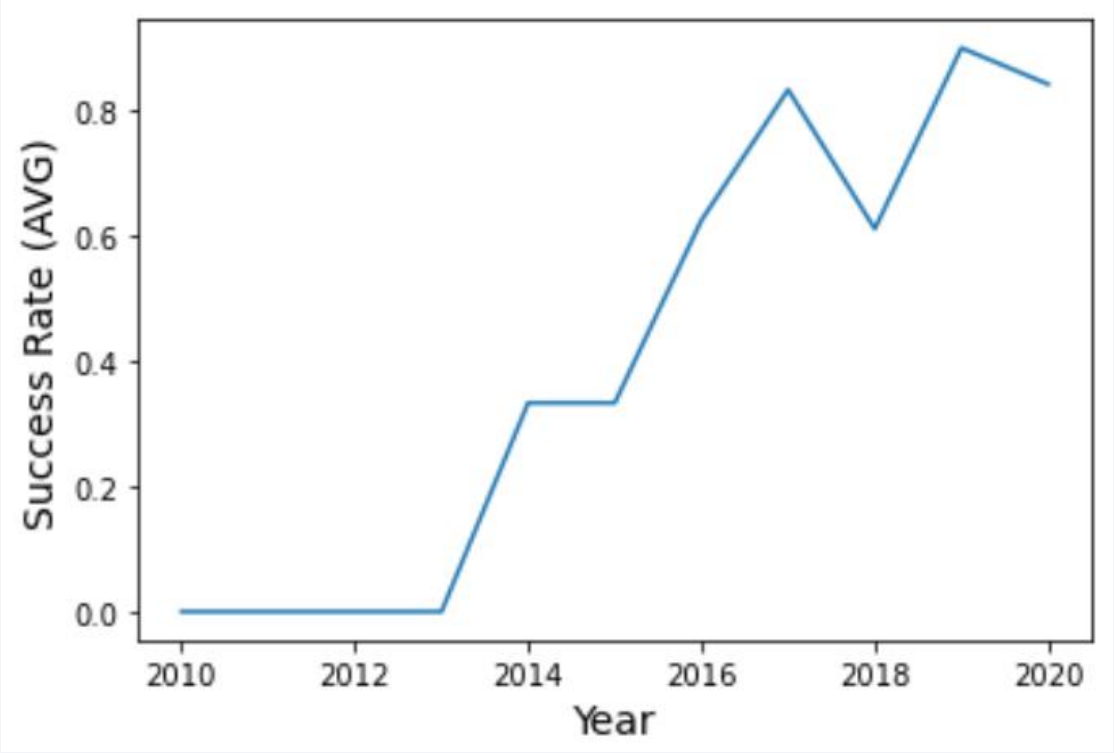
- In an attempt to tie all of the success rate relationships together (e.g. **FlightNumber**, **PayloadMass**, **LaunchSite**, and **Orbit**) a final scatter plot was created using **Orbit** for the Y-axis and **PayloadMass** as the X-axis while still using **Class** as the color indicator.

- This plot did show some landing rate improvements for some of the individual **Orbit** categories, but the graph did not consider the obvious pattern that SpaceX's successful landing rates increased throughout the launch **Date** and/or **FlightNumber** progression.

Note: Please see Appendix D for the scatter plot graph described on this slide.

EDA with Data Visualization

- To explore the success rate, as it relates to the passage of time, a line chart was created while using the **Year** (X-axis) and **Class** (Y-axis).
  - **Year** value was derived via the Pandas *DatetimeIndex().year* methodwhile targeting the **Date** column.
  - The plotted Class value was average value for the whole year: 2010 – 2020
  - A *groupby()* methodwas used to create the **Year** bins.



EDA with Data Visualization

- GitHub URL: Data Visualization notebook

- **Note:** See Appendix E for data sample from this step

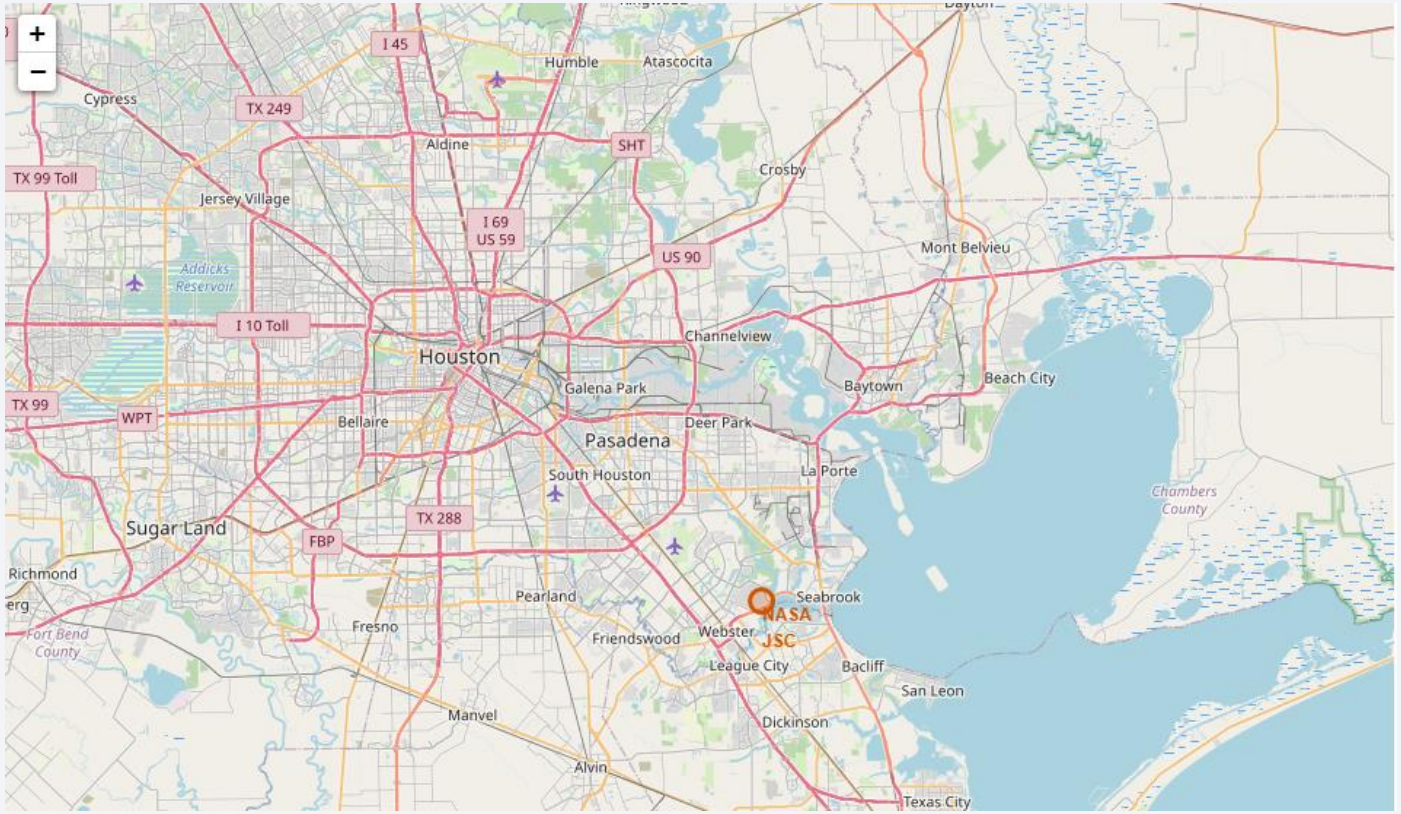
Build an Interactive Map with Folium

- Using the Pandas, Folium, and wget, Python libraries, several interactive maps were created using the geospatial launch data collected in during the web-scrapping.
  - Additional Folium Plugins used: MarkerCluster, MousePosition, and DivIcon
- The first step was to create a dataframe using a select group of columns from the imported SpaceX dataset. For this, only **Launch Site**, **Lat**(Latitude), **Long**(Longitude), and **Class** were selected.
- Before mapping the individual launch sites, a baseline Folium map needed to be created. To do this, a starting point set of Latitude and Longitude coordinates were needed. As this this whole exercise is studying rocket launches to space, the NASA Johnson Space Center Latitude and Longitude were Plugged into the *folium.Map()* function.

```
# Start location is NASA Johnson Space Center
nasa_coordinate = [29.559684888503615, -95.0830971930759]
site_map = folium.Map(location=nasa_coordinate, zoom_start=10)
```

Build an Interactive Map with Folium

- This step included adding a *folium.Circle()* as a location marker and it also included a matching marker label of 'NASA JSC'.



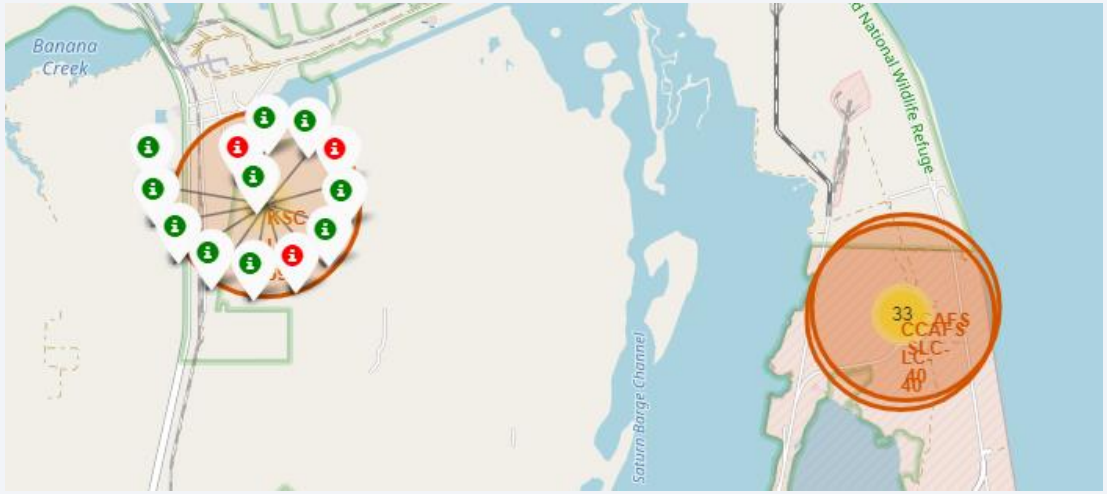
Build an Interactive Map with Folium

- The next step was to cycle through each distinct Launch Site and add them to the map using the coordinates in the created dataframe. Marker icons were also used to mark the locations.

Build an Interactive Map with Folium

- As many of the launch sites have the same coordinates, the *MarkerCluster()* Folium plugin was used to add clarity to each individual launch site.

- This was done by adding a new column to the created dataframe named **marker_color** and this column was populated by iterating through the **Class** value for each row and assigning a color based on landing outcome: Class = 1 will be assigned the color green and Class = 0 will be assigned the color red.



Build an Interactive Map with Folium

- In order to study the proximity between a SpaceX launch site and nearby population centers, a methodwas created to calculate the distance between two coordinates.
- To accurately determine the distance, a means was needed to select a specific location on a map to pull the exact coordinates of that exact spot. To accomplish this, the Folium *MousePosition()* plugin was included in the created Folium map to display the location's coordinates in the top-right hand corner of the map.

Build an Interactive Map with Folium

- Using the distance the *MousePosition()* functionality, it was now possible to hover a mouse cursor over any location on the map to record the posted coordinates.

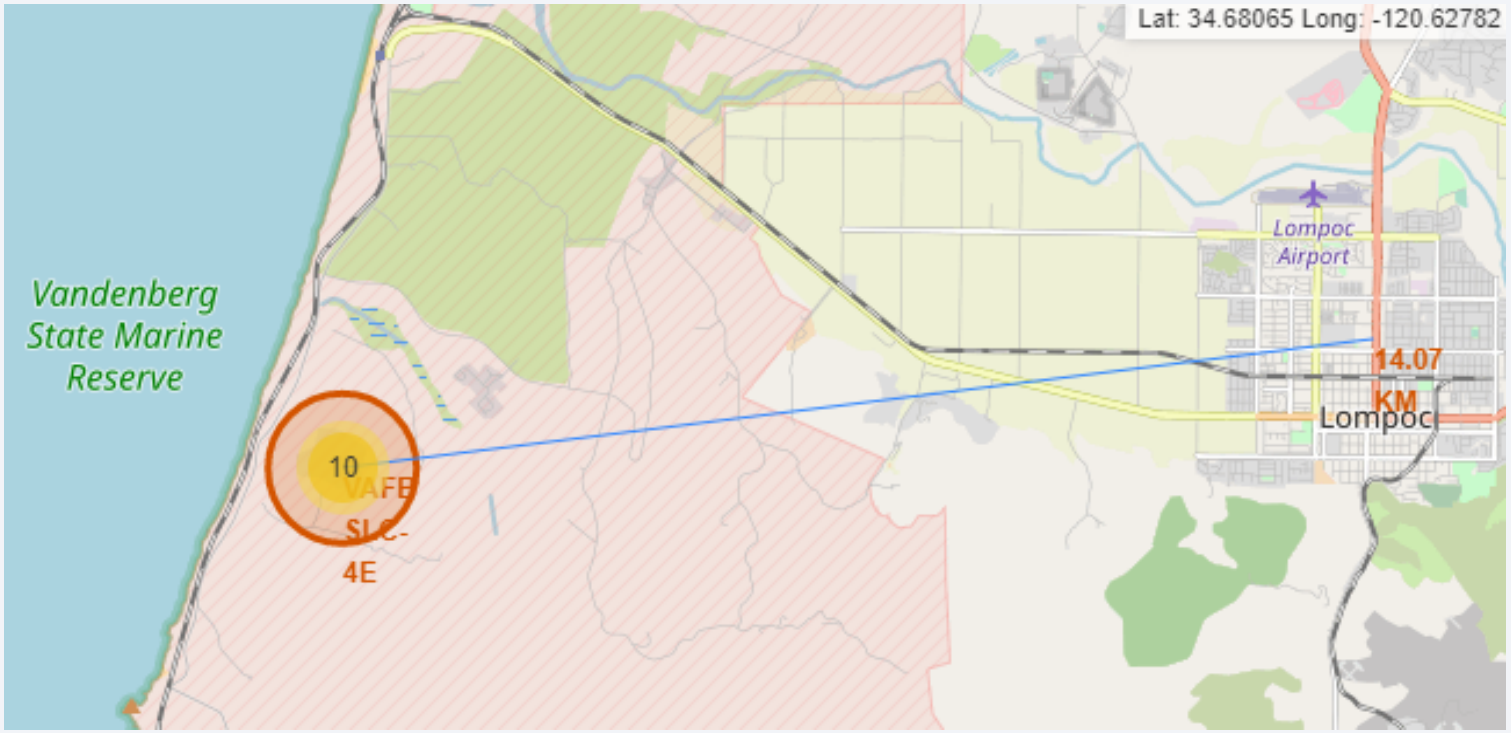
- This information could then be plugged into the distances formula, along with the starting point coordinates of a SpaceX launch site, to calculate the distance between the point points of insterest (in KM).

- This calculation was then saved as a variable and it, along with a plotted line, was then added to the Folium map as a child object.

- **Note:** Example of the above actions can be seen on the next slide

Build an Interactive Map with Folium

- GitHub URL: [Folium Map notebook]



- **Note:** Folium maps cannot be seen directly in GitHub. The screenshots taken in this section were all taken while viewing the Jupyter notebook in Anaconda

Build a Dashboard with Plotly Dash

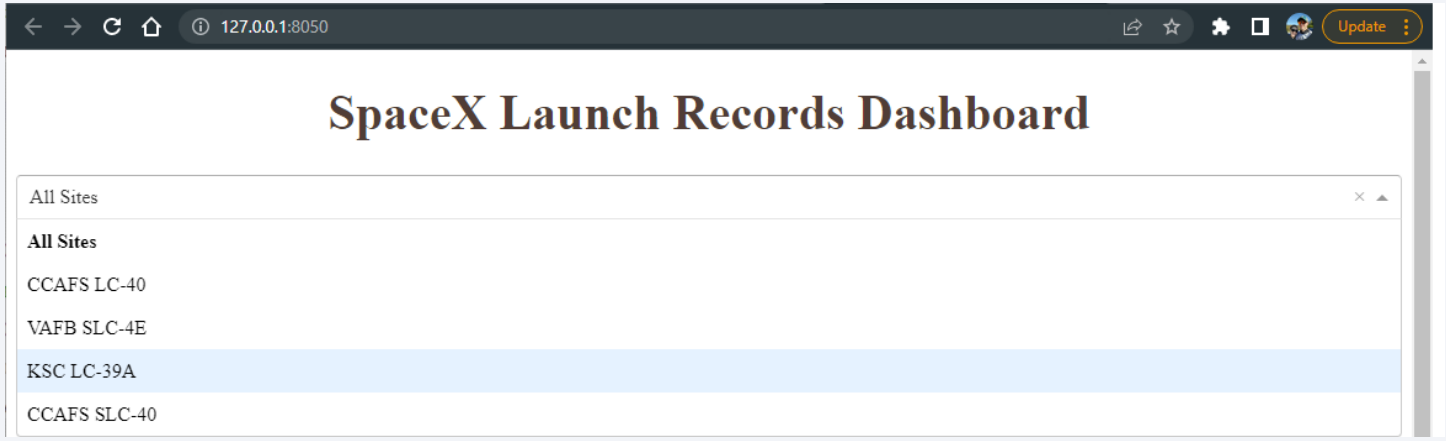
- Using the Pandas, Dash, Dash_HTML_Components, and Plotly Python libraries, an interactive dashboard app was created to quickly drill through launch parameters of interest.
  - The following Dash modules were also utilized: DCC and Dash.Dependencies (Input & Output)

- The created app had five sections:
  - Dashboard Title
  - Dropdown picker to allow the user to select ALL or select just one of the SpaceX launch sites
  - Interactive Pie Chart
  - PayloadMass (Kg) slider
  - Interactive Scatter Chart

Build a Dashboard with Plotly Dash

- **Dashboard Title** – Made use of the *dash_html_components.H1()* component, above first HTML division (div) container, to display the dashboard's title 'SpaceX Launch Records Dashboard'.

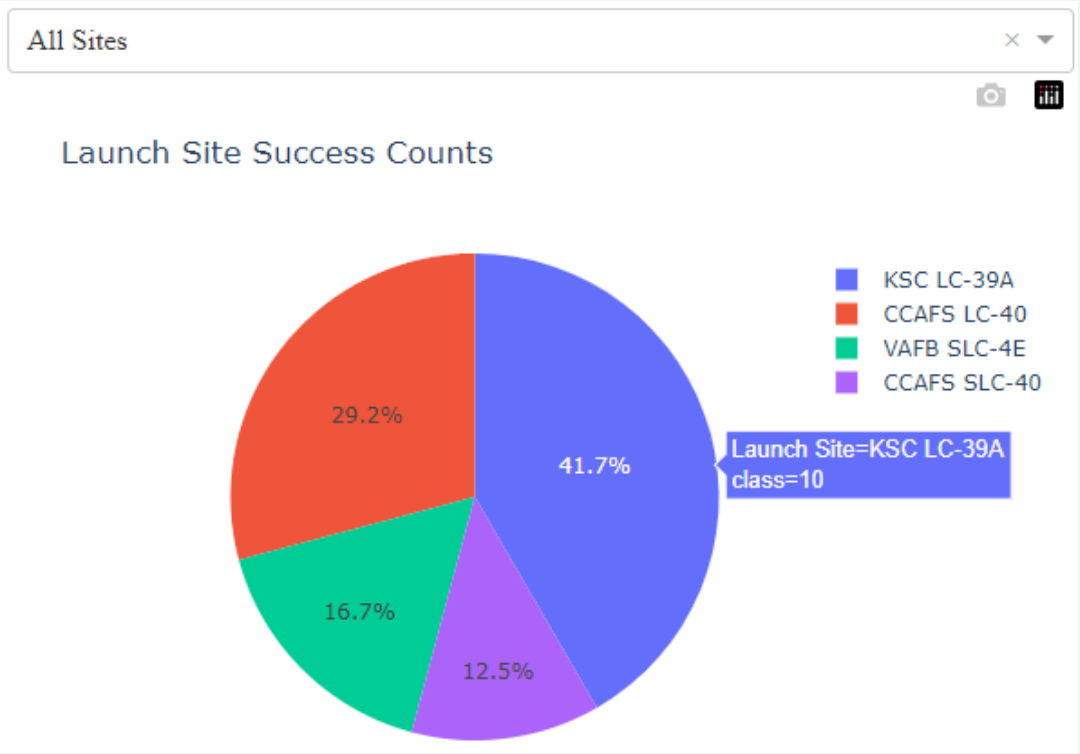
- **Dropdown Picker** – The dropdown picker allows for the user to limit the dashboard's results by individual Launch Sites. The default value for this parameter was set to **All Sites**.



Build a Dashboard with Plotly Dash

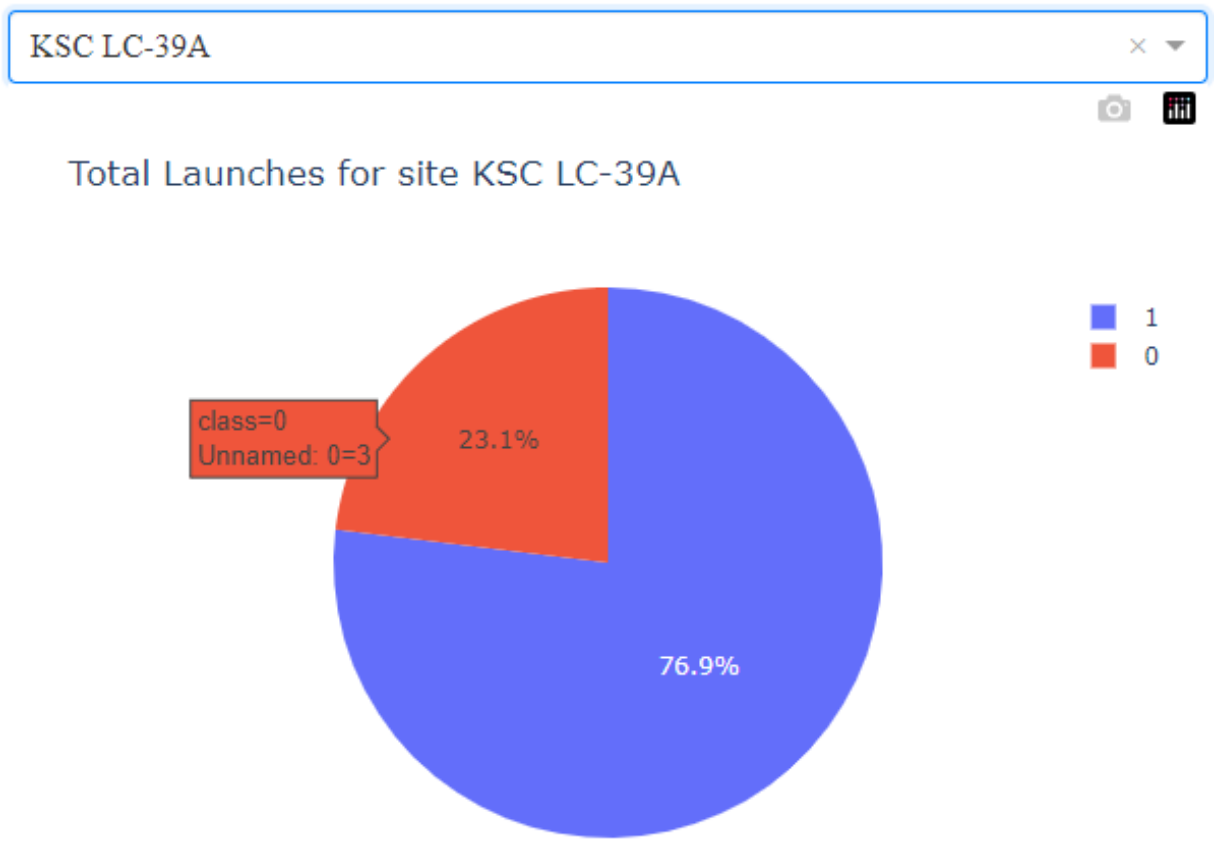
- **Interactive Pie Chart** – The pie chart was designed to take the input from the Dropdown Picker and then display the Success-to-Failure ratio for successful landings (note: Class = 1 indicates a successful landing).

  - When **All Sites** is selected in the dropdown picker, the pie chart will show the four landing site's percentage of the total count of successful landings.
  - Hovering over any section of the Pie Chart will show that site's total count of successes.
  - Note: Clicking a launch site in the legend will remove it from consideration (see Appendix F.



Build a Dashboard with Plotly Dash

- When individual sites are selected with the dropdown picker, only the selected launch site's success rates are shown.



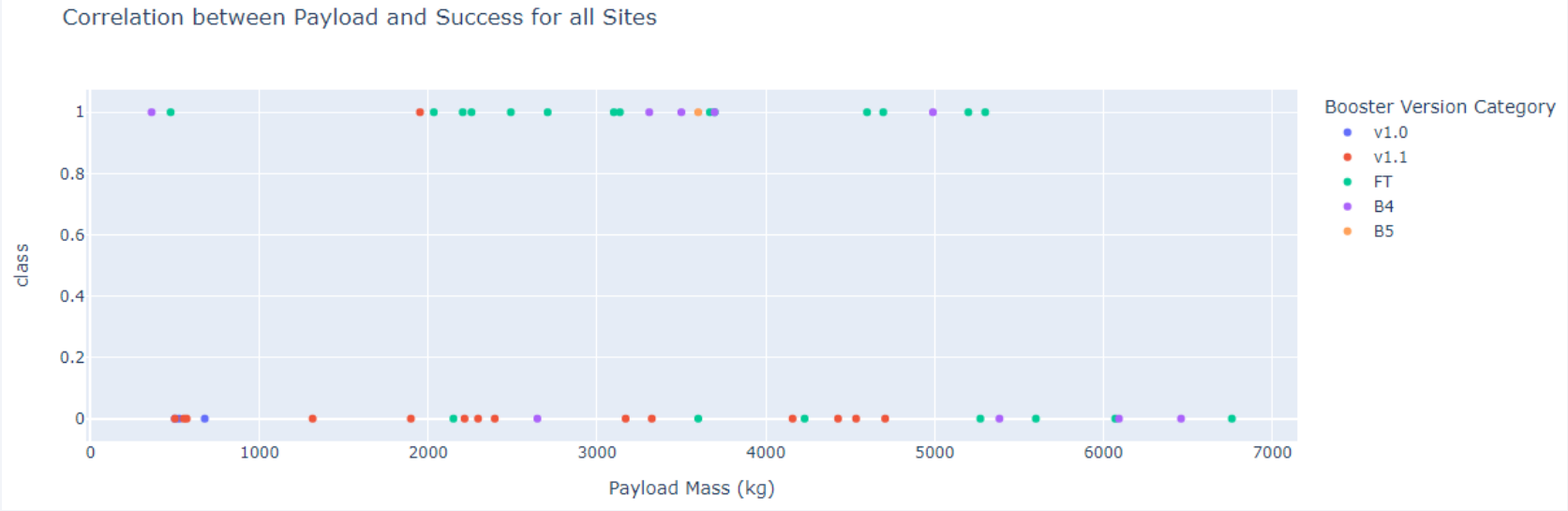
Build a Dashboard with Plotly Dash

- **Interactive PayloadMass (Kg) range slider** – This horizontal range slider was created using a range between 0 Kg and 10,000 Kg, with individual selection steps (I.e. Marks) every 1,000 Kg.
  - To make the slider easier to process (visually), the Marks were labeled at intervals of 2000 Kg.

- Pandas was used to calculate the *max()* and *min()* values of the PayloadMass (Kg) column and these values were saved as individual variables.
  - These minimum and maximum values were then used as the starting range default value in the range slider.

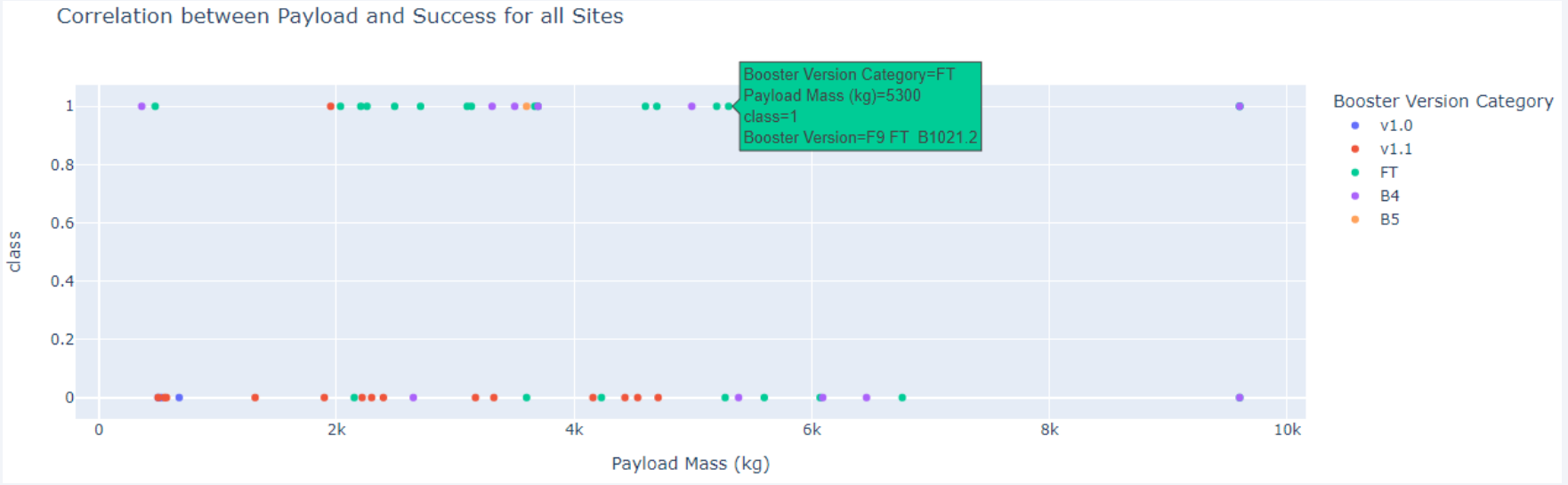
Build a Dashboard with Plotly Dash

- **Interactive Scatterplot Chart** – This scatterplot was designed to accept both the input from the PayloadMass (Kg) range slider and the Launch Site dropdown picker.
- The **Class** column was used as the Y-axis while **Payload Mass (Kg)** made up the X-axis.
  - To add more depth to the chart, **Booster Version Category** was used as the color indicator for each plotted launch.



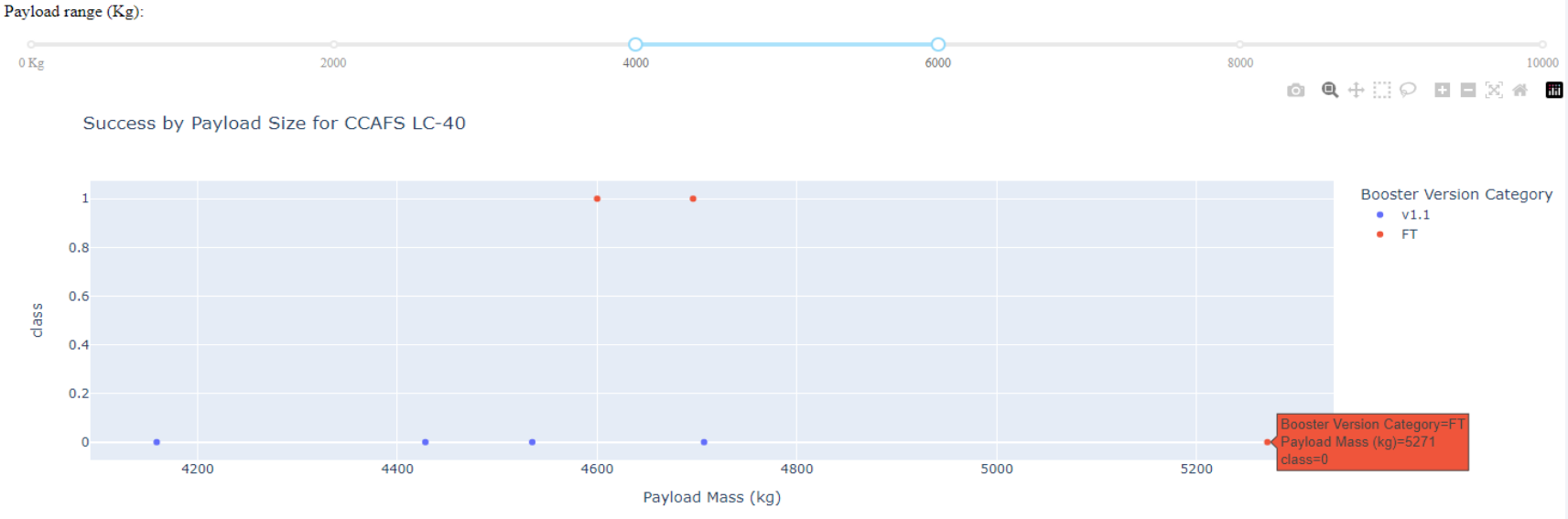
Build a Dashboard with Plotly Dash

- Like the pie chart, hovering over points of interest allowed the user to see that row's data, as it relates to the scatterplot.



Build a Dashboard with Plotly Dash

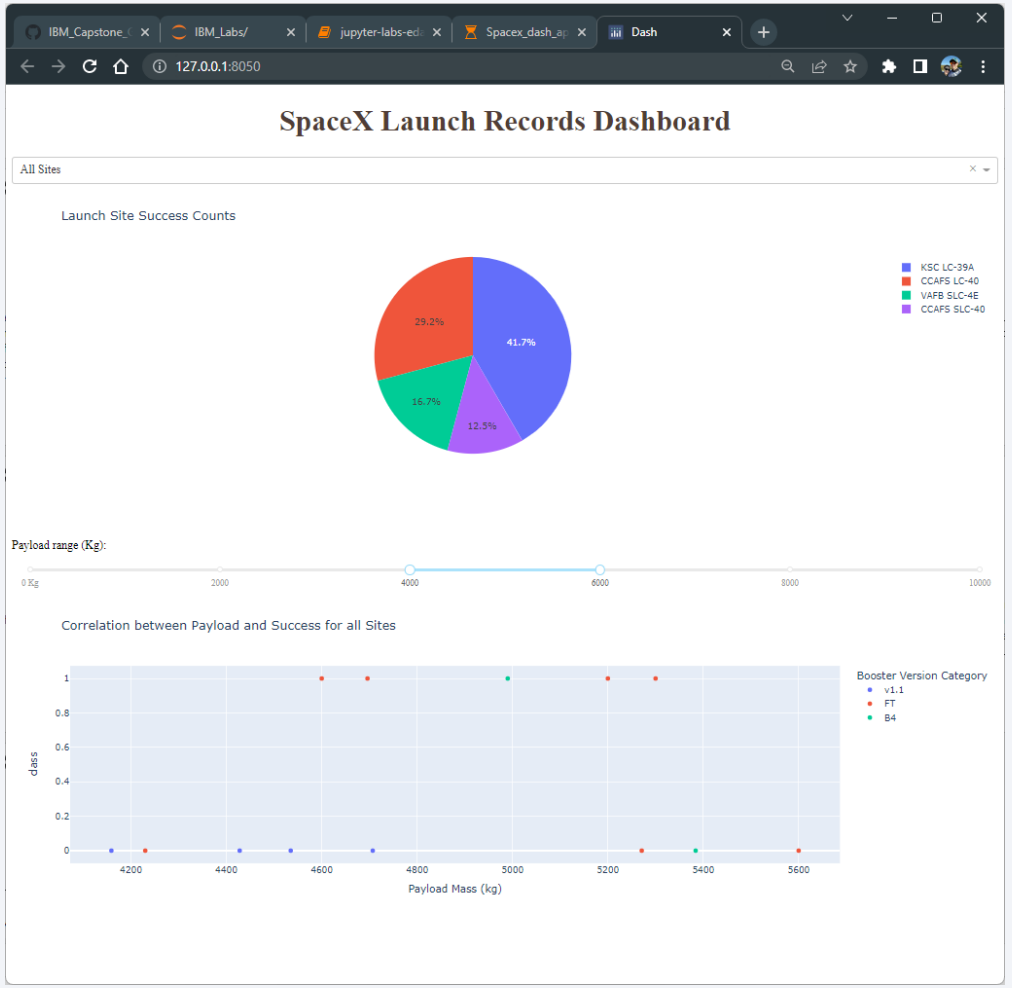
- Being able to accept the filtering parameters from the dropdown picker and the range slider, the created scatterplot can give ANY user the ability to instantly drill into the SpaceX dataset
  - For example, with just three mouse clicks on the dashboard, the scatterplot could show a user the success rate and booster version category for all launches that left the **CCAFS LC-40** launch site with a total payload mass between **4,000 Kg** and **6,000 Kg**.



Build a Dashboard with Plotly Dash

- To the right you can see the created dashboard in its entirety.

- GitHub URL: [Plotly Dashboard notebook]



Predictive Analysis (Classification)

- Using the Pandas, NumPy, Seaborn, Matplotlib, and Sklearn Python libraries, the two SpaceX datasets were manipulated for the purpose of building the most accurate data model possible while trying to predict Falcon 9 successful landings.
  - Additional Sklearn Plugins used: preprocessing, train_test_split, GridSearchCV, LogisticRegression, SVC, DecisionTreeClassifier and KNeighborsClassifier

- The following steps were actioned to complete this task:
  - Data manipulation
  - Random data splitting into TRAIN and TEST datasets
  - Data Model creation
  - Data Model accuracy review

Predictive Analysis (Classification)

Data manipulation – A step was taken at the end of the EDA with Data Visualization data exploration process that took four categorical columns from the SpaceX API dataset and converted them to Binary values (1 = True & 0 = False).

- A new dataframe was created only selecting twelve of the SpaceX columns and in this new dataframe, **Orbit**, **LaunchSite**, **LandingPad**, and **Serial** were put through the Pandas *getdummies()* function.

  - This "**OneHotEncoder**" step created a new column for every category within each of these columns and it assigned a 1 or a 0 for each row based on the original value for that row.

  - **Note:** there would only be a single 1 value in each category grouping (per row).

Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	Orbit_LEO	Orbit_MEO	Orbit_PO	Orbit_SO	Orbit_SSO	Orbit_VLEO
0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0

Predictive Analysis (Classification)

- The dataframe was then put through the Pandas *astype('float64')* method and this step converted the entire data set in to the Float64 datatype. It should be noted that this step also converted the Boolean TRUE/FALSE values into 1's and 0's (respectively).
- This converted data was then imported as a dataframe named '**X**' and it was further processed by putting the dataframe through the Sklearn *preprocessing.StandardScaler()* function.
- This step was done to put all variables in the '**X**' dataset at an equal footing with each other by removing the mean of each value while also transforming things in a way to make the standard deviation of the distribution equal to 1.
  - This action reduces potential bias of the different levels of measure while using the different datapoints in the data models
  - **For example**: Binary (1's and 0's) columns verses PayloadMass values measured in the thousands

Predictive Analysis (Classification)

- The web scrapped data was also imported and the derived successful sanding **Class** column was put into a dataframe named '**Y**'

- The 'Y' data frame was then converted to a NumPy array via the to_numpy() function.
  - This step was done to convert the tabular dataframe into an indexed array
  - **Note:** Efficiency was also a motivator this conversion. As per the *Sklearn* Python library's project website (scikit-learn.org), *Sklearn* "assumes the data is an NumPy array" when interacting with any dataset.

    While Sklearn can interact cleanly with Pandas dataframes, it is best practice to reduce unnecessary overhead while doing any machine learning activities.

Predictive Analysis (Classification)

- **Data manipulation** - The *train_test_split()* method (Sklearn) was used to randomly split the 90 rows of data contained within the 'X' and 'Y' arrays into four distinct variables and these variables were grouped by TRAIN and TEST designations

  - The X_train and the Y_train pair each had **72** indexed rows and they will be used to generate the data models.

  - The X_test and Y_test set contained **18** indexed rows and they will be used to test the created data model's accuracy once they are create (mocked new data).

Predictive Analysis (Classification)

- **Data Model creation** – Using individual parameters as starting points, the following data model methodologies were fitted while using the TRAIN split of the SpaceX data.
  - Logistic Regression
  - Support Vector Machine (SVM)
  - Decision Tree
  - K-Nearest Neighbors (KNN)

- Using Sklearn's *GridSearchCV()* functionally, each model was created to locate and use the best tuned hyperparameters to locate the most accurate model possible.

Predictive Analysis (Classification)

- To minimize over-fitting, each model was run using 10 as its k-fold cross-validation parameter value.

- After each model was fitted, while using the before mentioned hyperparameters, the below *GridSearchCV()* attributes were used to help evaluate the created object.

  - ***best_params_:*** Used to see which parameters were used to create the best model

  - ***best_score_:*** Used to display the score of the most accurate set of parameter used in the ten corss-validation folds.

Predictive Analysis (Classification)

- See below for the accuracy score (*best_score_*) for each fitted data model while using the TRAIN SpaceX dataset.
  - Logistic Regression: `0.8464`
  - Support Vector Machine (SVM): `0.8482`
  - Decision Tree: `0.8892`
  - K-Nearest Neighbors (KNN): `0.8482`

- Each model scored similarly well, but the **Decision Tree** model was the clear winner in terms accuracy.

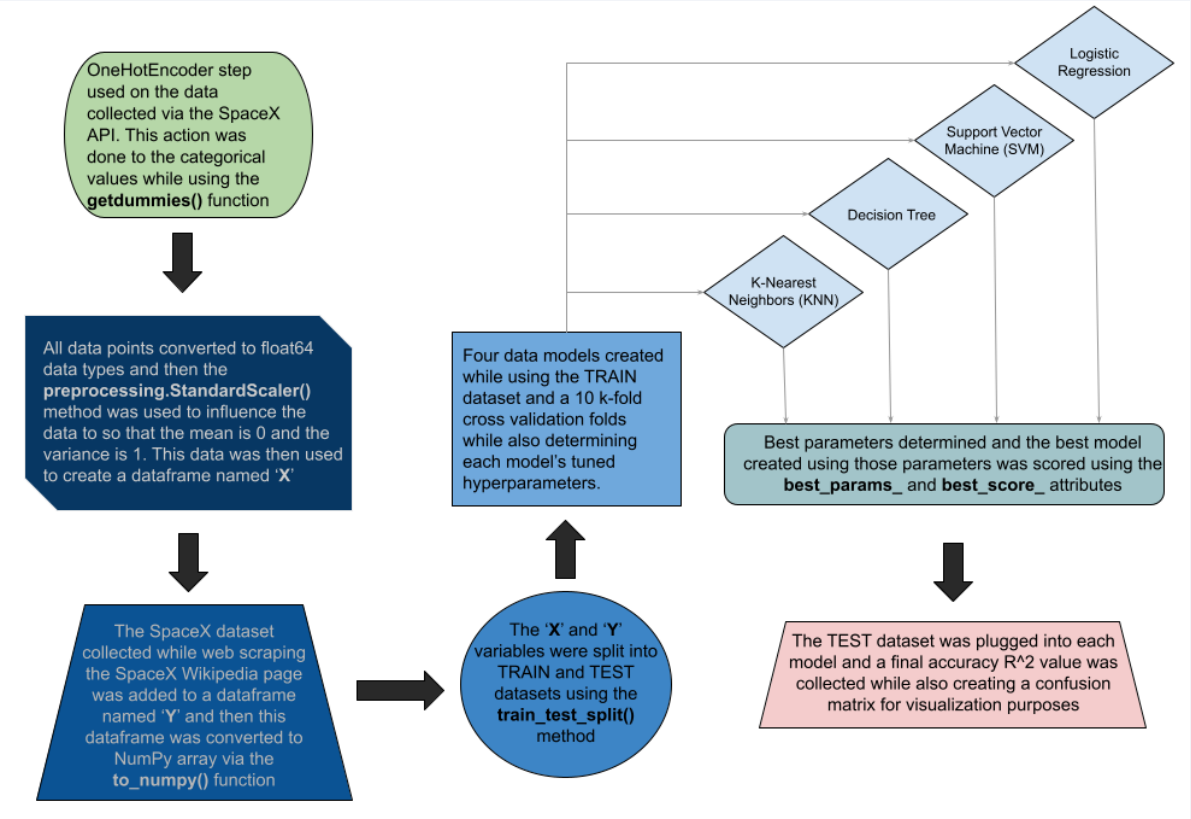
Predictive Analysis (Classification)

- **Data Model accuracy** – Having data models with high accuracy scores was the first important hurdle in this exercise, but it was not the most important step in the process. The true purpose of a data model is to accurately predict while using new data and to simulate new data, the TEST split '**X**' and '**Y**' variables were run through each created object.
  - The K-fold cross-validation mimics this step while trying to find the most accurate model, but that alone does not mitigate the possibility of creating an overfitted model.

- Each model's interaction with the TEST split was scored (R^2) while using Sklearn's *score()* method and then, to help visualize the results, a created methodwas used to plot a Sklearn confusion matrix
  - The confusion matrix was utilized to see false positives and false negatives.

Predictive Analysis (Classification)

- GitHub URL: [Machine Learning Prediction notebook]



Results

Final results while attempting to predict SpaceX's Falcon 9 successful landings.

- **Logistic Regression:**

  - R^2 = 0.8334

```
logreg_cv.score(X_test, Y_test)
```

```
0.8333333333333334
```

  - Confusion Matrix

Results – Predicting SpaceX's Falcon 9 successful landings

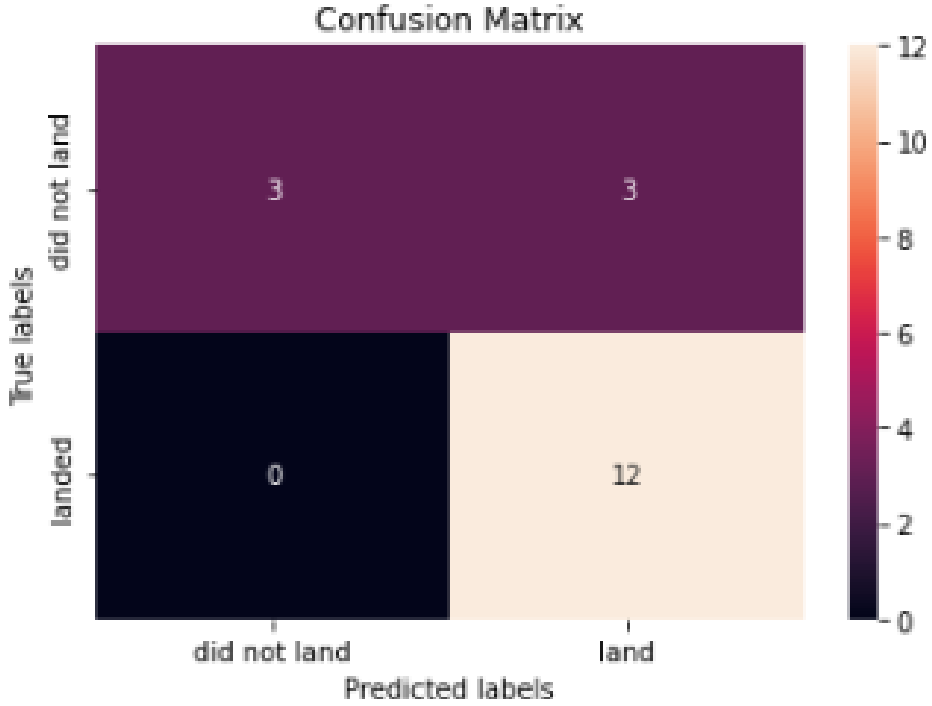
Logistic Regression:

- R^2 = 0.83334

```
logreg_cv.score(X_test, Y_test)
```

```
0.8333333333333334
```

```
yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



Results – Predicting SpaceX's Falcon 9 successful landings

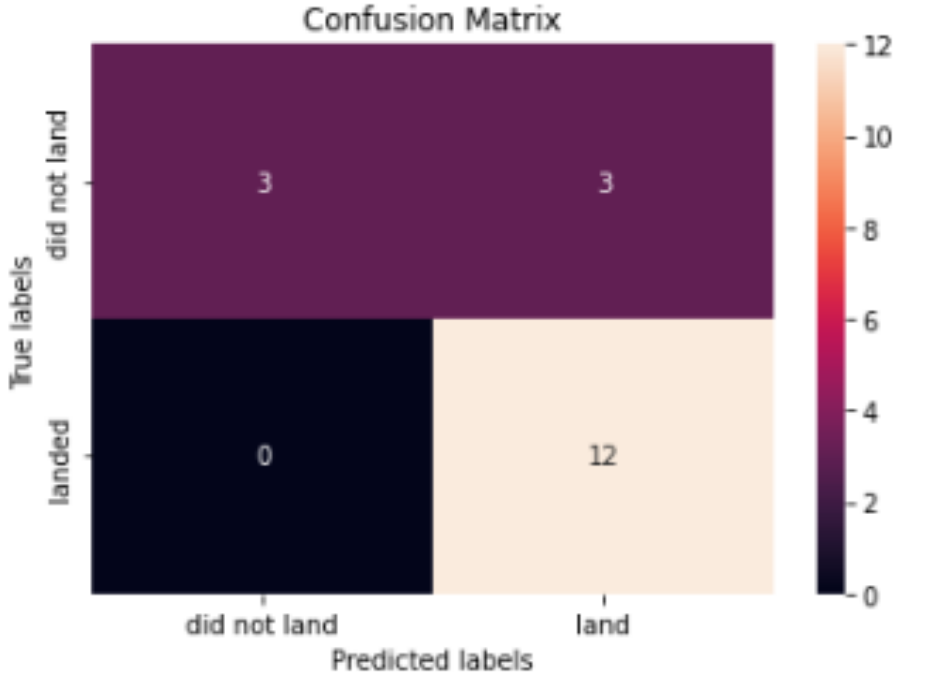
Support Vector Machine (SVM):

- R^2 = 0.83334

```
svm_cv.score(X_test, Y_test)
```

```
0.8333333333333334
```

```
yhat1=svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat1)
```



Results – Predicting SpaceX's Falcon 9 successful landings

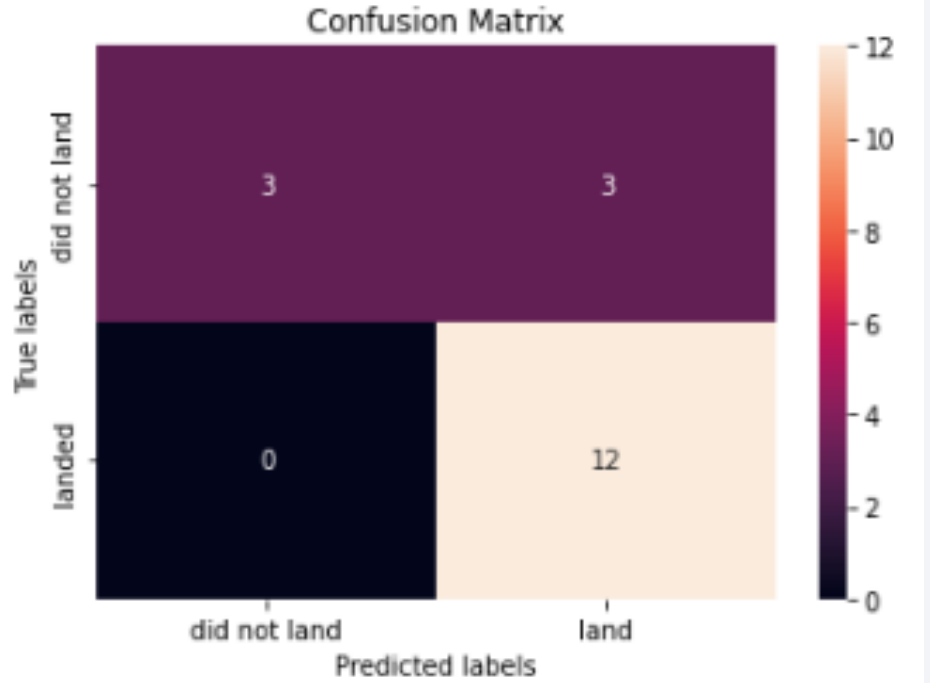
Decision Tree

- R^2 = 0.83334

```
tree_cv.score(X_test, Y_test)
```

```
0.8333333333333334
```

```
yhat2 = tree_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat2)
```



Results – Predicting SpaceX's Falcon 9 successful landings

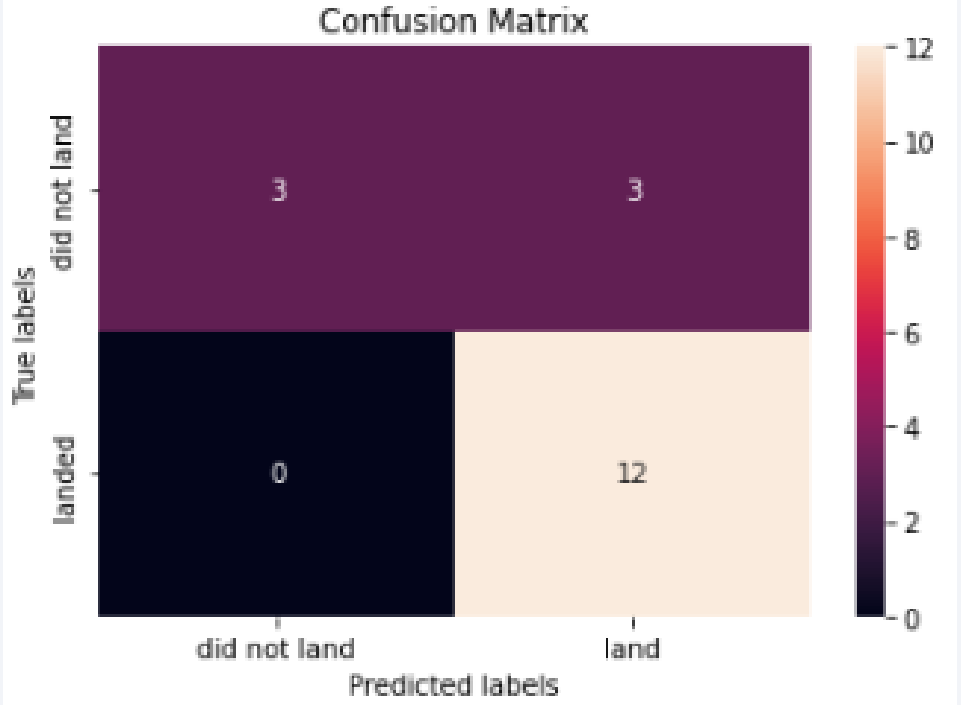
Nearest Neighbors (KNN)

- R^2 = 0.83334

```
knn_cv.score(X_test, Y_test)
```

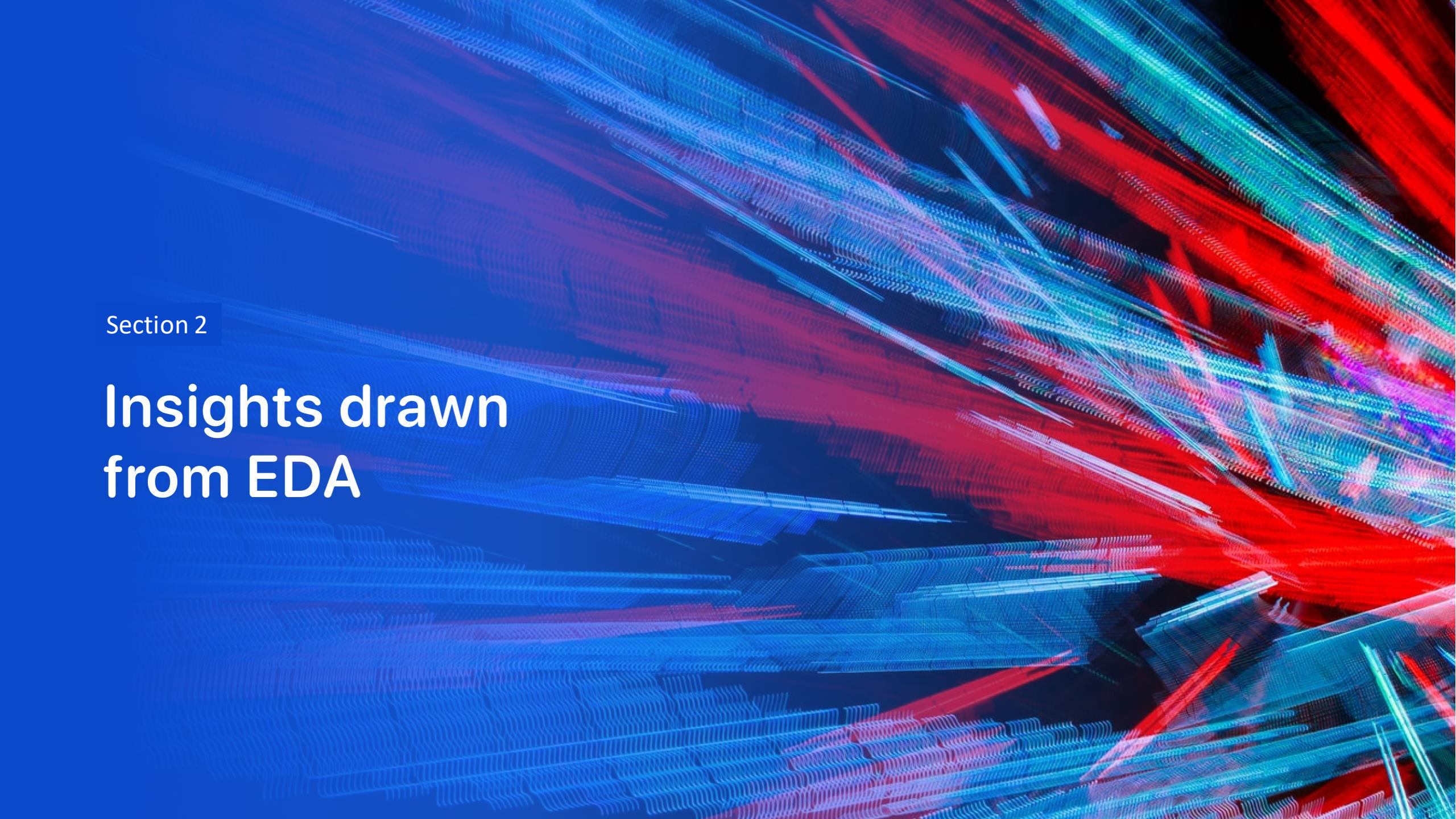
```
0.8333333333333334
```

```
yhat3 = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat3)
```



Section 2

Insights drawn from EDA



*** Note to Reader ***

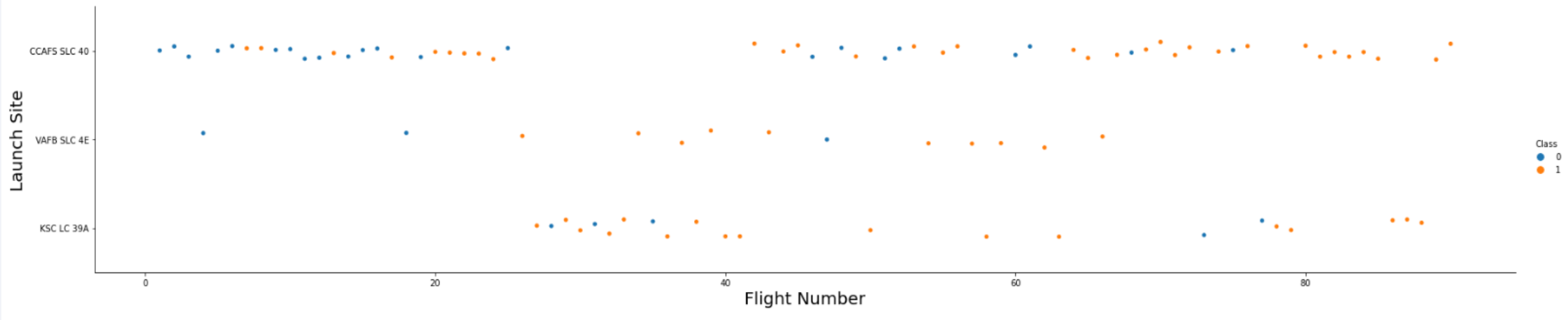
The content contained in slides 69 - 101 relate directly to requirements set by the IBM Data Science Professional Certificate program (Coursera).

Most all screenshots and descriptions will have been displayed in earlier slides.

It should also be noted that the datasets varied from module to module and that there were observed differences (ex. Number of Launch Sites). One would surmise this was Coursera's only method to help identify or prevent plagiarism.

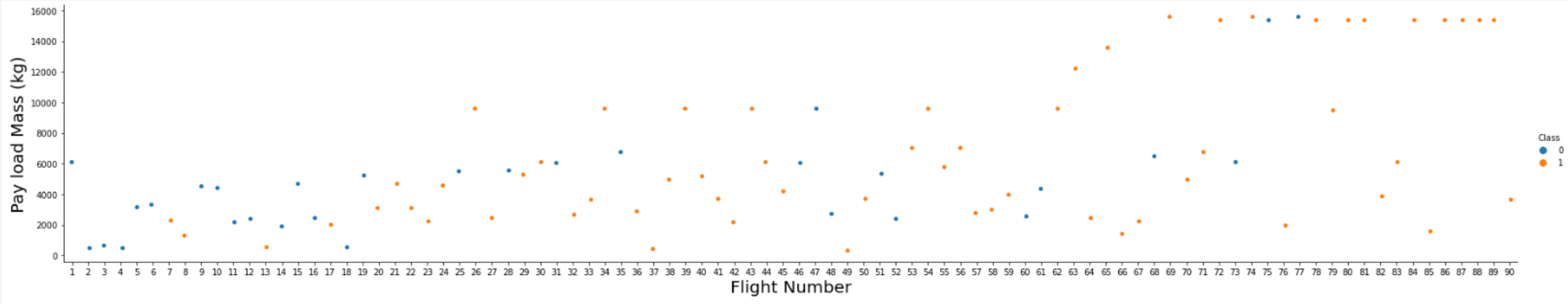
Flight Number vs. Launch Site

- The below scatterplot shows the success/failure of each Falcon 9 landing by Launch Site. This plot clearly show that CCAFS SLC 40 had the most SpaceX launches and that success rates increase over time.



Payload vs. Launch Site

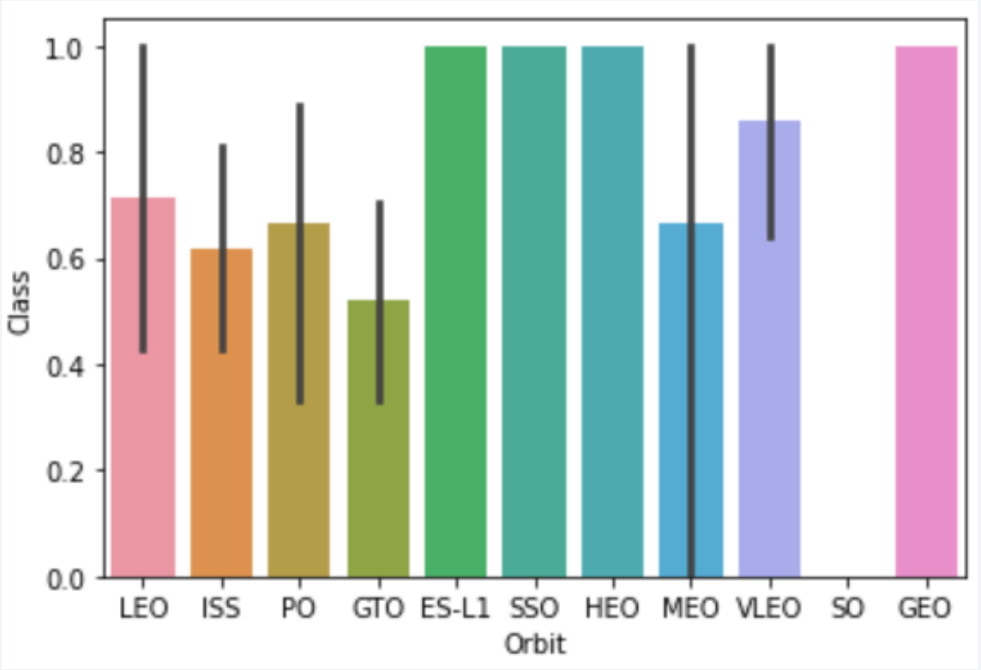
- Now we see the Payload Mass success rates and the recorded MIN/MAX range for all flights. This plot also shows the vast majority of launches were below 8,000 Kg.
- This plot also shows that there were only three failed landings for launches that had a Payload Mass above this 8,000 Kg mark.



Success Rate vs. Orbit Type

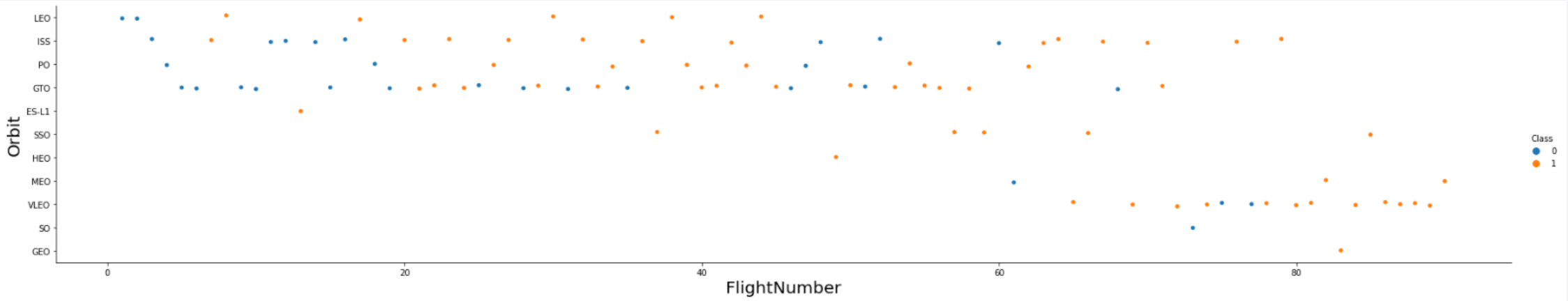
- The bar chart to the right shows the average success rate for each of the orbit types attempted by SpaceX's Falcon 9 rocket.

- This plot is somewhat meaningless as it does not show the total count of launches per orbit. Without this knowledge, one cannot tell is ES-L1, SSO, HEO, or GEO had 1 or 1000 successful landings. Same is true for SO's unsuccessful attempts. The percentile black line may offer clues, but this is still an unreliable indicator of success.



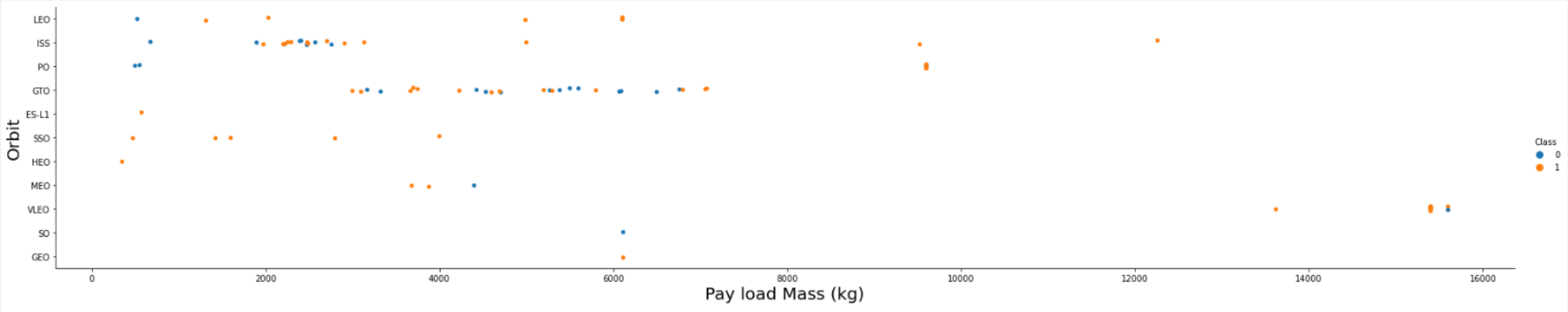
Flight Number vs. Orbit Type

- The scatterplot below fills in all of the gaps for the previous slide's bar chart. Here we see that HEO and GEO orbit types did have 100% success rates, but they only had a single attempt each.



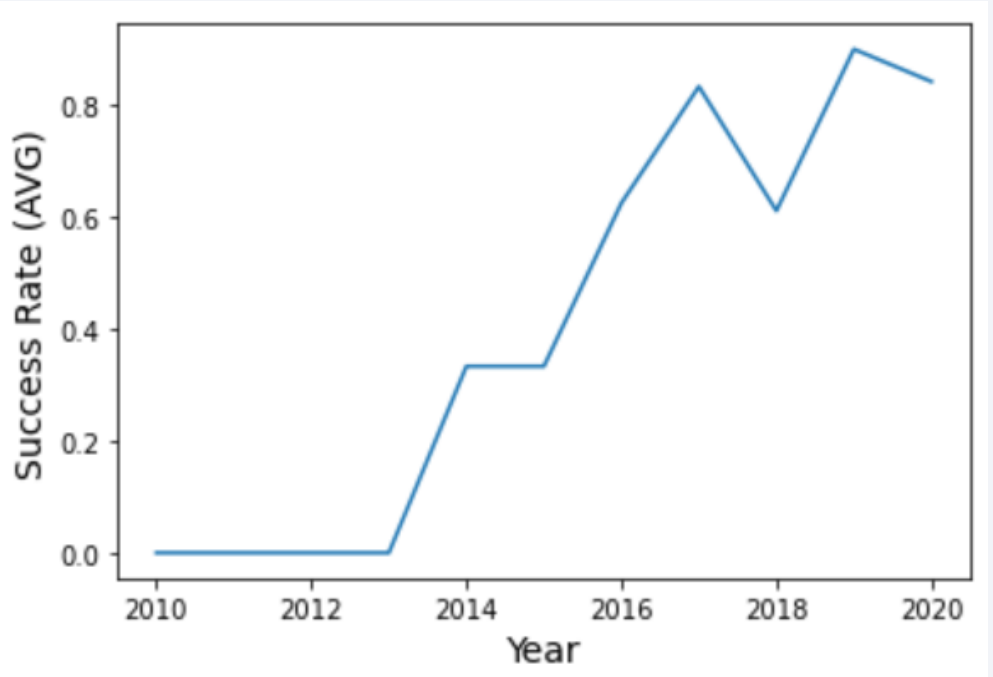
Payload vs. Orbit Type

- Here we see a scatterplot of successes and failures for Payload Masses by Orbit type. This plot reinforces the earlier observation about the success rates for Payload Mass values over 8,000 Kg. It also shows a populated grouping of ISS Orbits between 2,000 Kg and 3,000 Kg, and it shows all GTO attempts ranged from 3,000 Kg and 7,000 Kg.



Launch Success Yearly Trend

- The earlier scatterplots that used Flight Numbers indicated that success rates for the Falcon 9 rockets.
- When the averages were plotted, by year, this observation was mostly reinforced (as shown by the line chart to the right).
- The dip in 2018 may need to be researched.



All Launch Site Names

- While using SQLite to query the SpaceX dataset, a **DISTINCT** used in the **SELECT** clause provide a list of all of the Falcon 9 Launch Sites.
  - CCAFS LC-40, VAFB SLC-4E, KSC LC-39A, CCAFS SLC-40

```
res = cur.execute(
    """
    SELECT Distinct Launch_Site
    FROM SPACEXTBL
    """)
res.fetchall()
```

```
[('CCAFS LC-40',), ('VAFB SLC-4E',), ('KSC LC-39A',), ('CCAFS SLC-40',)]
```

Launch Site Names Begin with 'CCA'

- A **LIKE** conditional and a '%' wildcard was used to find Launch Sites that started with CCA. The **LIMIT 5** logic was added to ensure the query only returned five results.

```
res = cur.execute(
    """
    SELECT *
    FROM SPACEXTBL
    WHERE Launch_Site like 'CCA%'
    LIMIT 5

    """)
res.fetchall()
```

```
[('04-06-2010',
  '18:45:00',
  'F9 v1.0  B0003',
  'CCAFS LC-40',
  'Dragon Spacecraft Qualification Unit',
  0,
  'LEO',
  'SpaceX',
  'Success',
  'Failure (parachute)'),
 ('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',
  'Failure (parachute)'),
 ('08-10-2012',
  '00:35:00',
  'F9 v1.0  B0006',
  'CCAFS LC-40',
  'SpaceX CRS-1',
  500,
  'LEO (ISS)',
  'NASA (CRS)',
  'Success',
  'No attempt'),
 ('22-05-2012',
  '07:44:00',
  'F9 v1.0  B0005',
  'CCAFS LC-40',
  'Dragon demo flight C2',
  525,
  'LEO (ISS)',
  'NASA (COTS)',
  'Success',
  'No attempt'),
 ('01-03-2013',
  '15:10:00',
  'F9 v1.0  B0007',
  'CCAFS LC-40',
  'SpaceX CRS-2',
  677,
  'LEO (ISS)',
  'NASA (CRS)',
  'Success',
  'No attempt')]
```

Total Payload Mass

- A **SUM()** aggregate was used on the Payload_Mass__KG_ column while targeting SpaceX's customer '*NASA*'.
- This query produced a value of 45,596 Kg as the total amount carried by the Falcon 9 rocket.

```
res = cur.execute(
    """
    SELECT SUM(PAYLOAD_MASS__KG_)
    FROM SPACEXTBL
    WHERE Customer = 'NASA (CRS)'
    """)
res.fetchall()
```

```
[(45596,)]
```

Average Payload Mass by F9 v1.1

- A similar aggregate function was used on the Payload_Mass__KG_ column, but this query used the **AVG()** function to get the mean weight carried by the 'F9 v1.1' Booster.
- On average, this booster carried 2,928.4 Kg.

```
res = cur.execute(
    """
    SELECT AVG(PAYLOAD_MASS__KG_)
    FROM SPACEXTBL
    WHERE Booster_Version = 'F9 v1.1'
    """)
res.fetchall()
```

```
[(2928.4,)]
```

First Successful Ground Landing Date

- SQLite does not have a clean "To_Date" String-to-Date function, so a substring was needed to take advantage of the available **DATE()** function while using the **MIN()** aggerate function to query the first successful Ground Pad landing.
- Double quotes were needed in the **WHERE** Clause to account for the Landing Outcome column having a white space before the underscore (_)
- December 22nd, 2015 was the first successful ground pad landing for SpaceX

```
res = cur.execute(
    """
    select MIN(date(substr(Date, 7, 4) || '-' || substr(Date, 4, 2) || '-' || substr(Date, 1, 2)))
    from SPACEXTBL
    WHERE "Landing _Outcome" = 'Success (ground pad)'
    """)
res.fetchall()
```

```
[('2015-12-22',)]
```

Successful Drone Ship Landing with Payload between 4000 and 6000

- The below query was used to pull the boosters which have successfully landed on drone ship while also having a Payload Mass greater than 4,000 Kg but less than 6,000 Kg.
- This action was done by using a **BETWEEN** operator in the **WHERE** clause and the end result was the F9 FT B1022, F9 FT B1026, F9 FT B1021.2, F9 FT B1031.2 boosters.

```
res = cur.execute(
    """
    SELECT Booster_Version
    FROM SPACEXTBL
    WHERE "Landing _Outcome" = 'Success (drone ship)'
      AND PAYLOAD_MASS__KG_ between 4000 and 6000
    """)
res.fetchall()
```

```
[('F9 FT B1022',), ('F9 FT B1026',), ('F9 FT  B1021.2',), ('F9 FT  B1031.2',)]
```

Total Number of Successful and Failure Mission Outcomes

- In order to get the total number of both successful and failed Misson Outcomes, the **COUNT()** aggregate function was combined with a **GROUP BY** clause.
- This query produced a count of **99** 'Success', **1** 'Success (payload status unclear)', and **1** Failure (inflight)'

```
res = cur.execute(
    """
    SELECT TRIM(Mission_Outcome), Count(Mission_Outcome)
    FROM SPACEXTBL
    Group by TRIM(Mission_Outcome)
    """)
res.fetchall()
```

```
[('Failure (in flight)', 1),
 ('Success', 99),
 ('Success (payload status unclear)', 1)]
```

Boosters Carried Maximum Payload

- To capture the booster versions used to transport the maximum payload mass, a **DISTINCT** was used in the **SELECT** clause while using an **IN** operator and a nested sub-query using a **MAX()** aggregate to match things to the maximum payload mass value in the dataset (the results of this query can be seen below).

```
res = cur.execute(
    """
    SELECT Distinct Booster_Version
    FROM SPACEXTBL
    WHERE PAYLOAD_MASS__KG_ in (select MAX(PAYLOAD_MASS__KG_) FROM SPACEXTBL)
    """)
res.fetchall()
```

```
[('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 ',)]
```

2015 Launch Records

- In order to find the 2015 landing outcomes, booster versions, the month of the launch, and launch sites while using a drone ship, a **LIKE** operator was used with a '**%**' around the words "drone" & "ship".
- It was also necessary to pull a **SUBSTR()** of the Date value to get the month and to use 2015 as a conditional in the **WHERE** clause.

```
res = cur.execute(
    """
    SELECT substr(Date, 4, 2) as "Month",
    "Landing _Outcome" as "Landing_Outcome",
    Booster_Version,
    Launch_Site
    FROM SPACEXTBL
    Where substr(Date,7,4)='2015'
       AND "Landing _Outcome" Like '%drone%ship%'
    """)
res.fetchall()
```

```
[('01', 'Failure (drone ship)', 'F9 v1.1 B1012', 'CCAFS LC-40'),
 ('04', 'Failure (drone ship)', 'F9 v1.1 B1015', 'CCAFS LC-40'),
 ('06', 'Precluded (drone ship)', 'F9 v1.1 B1018', 'CCAFS LC-40')]
```

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

- I am not sure why, but we were asked to RANK, in a descending order, to number of landing outcomes between 2010-06-04 and 2017-03-20.
- This was done using the following techniques:
  - **COUNT()** aggregate and a **GROUP BY** clause on the Landing Outcome column
  - **RANK() OVER PARTICTION** ranking function that grouped by the Landing Outcome and then **ORDER BY** the Date column in a descending order.
  - **LIKE** operator in the **WHERE** clause while using a '**%**' around the word "Success".
  - **BETWEEN** operator to target any Date value between June 4th, 2010 and March 20th, 2017.
- This query only produced the following two records in the following order: Success (drone ship, 5 & Success (ground pad), 3
  - The query used to capture these results can be seen on the next slide.

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

```
res = cur.execute(
    """
    SELECT
    "Landing _Outcome",
    Count("Landing _Outcome") as LandingCount,
    RANK () OVER (
        PARTITION BY "Landing _Outcome"
        ORDER BY Date DESC
    ) LengthRank
    FROM
    SPACEXTBL
    WHERE 1=1
    AND "Landing _Outcome" like '%Success%'
    AND date(substr(Date, 7, 4) || '-' || substr(Date, 4, 2) || '-' || substr(Date, 1, 2))
                BETWEEN DATE('2010-06-04') AND DATE('2017-03-20')
    GROUP BY "Landing _Outcome"
    Order by LandingCount desc
    """)
res.fetchall()
```

```
[('Success (drone ship)', 5, 1), ('Success (ground pad)', 3, 1)]
```


Section 3
Launch Sites
Proximities Analysis

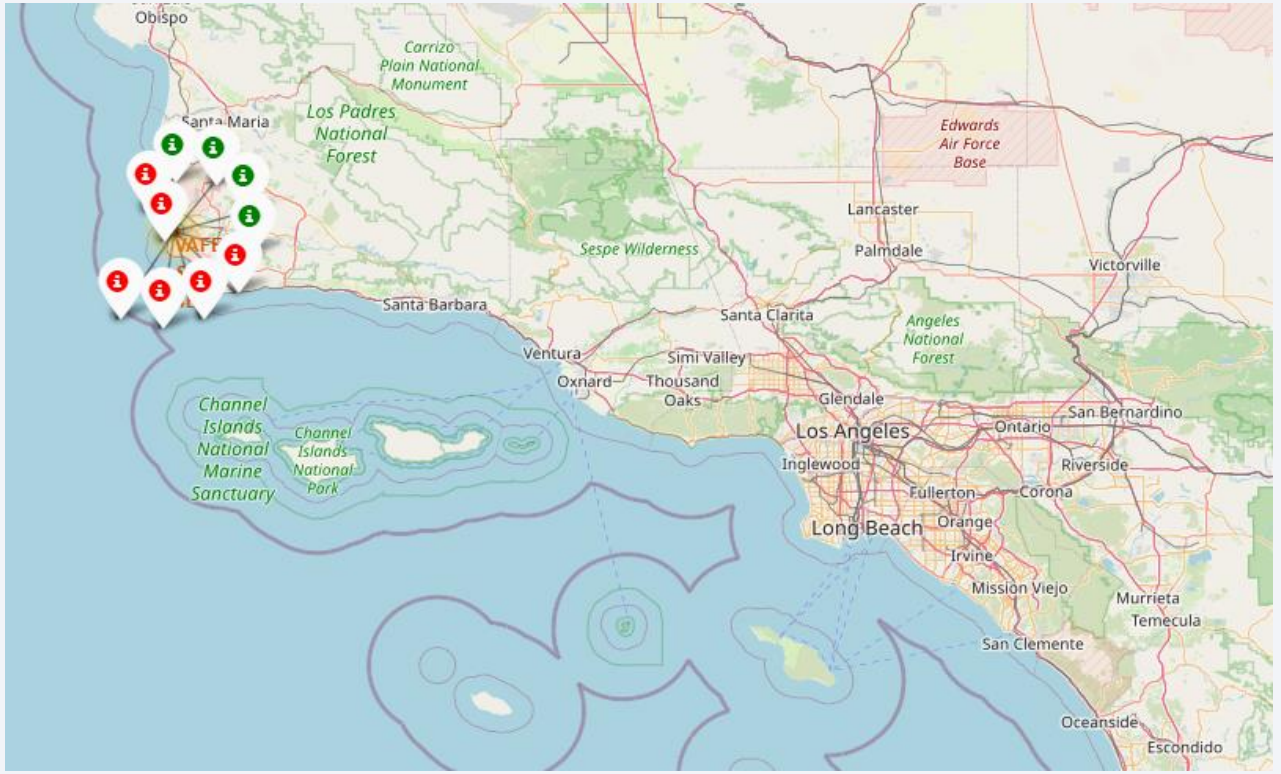
SpaceX Falcon 9 Launch Sites: CA and FL

- On the next slide there are four SpaceX launch sites are marked via the **folium.Circle()** function along with the site's name as the circle's label.
- There is one location in California (CA) and there are three in Florida (FL).
- The three locations in Florida are in a very close to each other and this is the explanation of the muddled location labels while viewing the map further out.
- Below is how these locations are marked while zoomed closer.

SpaceX Falcon 9 Launch Sites: CA and FL

Successes and Failures

- When the Class value for successful launches was added by way of the **folium.map.Marker()** function, a user can drill into the map to see each launch sites successes and failures (**GREEN** for Success & **RED** for Failure).



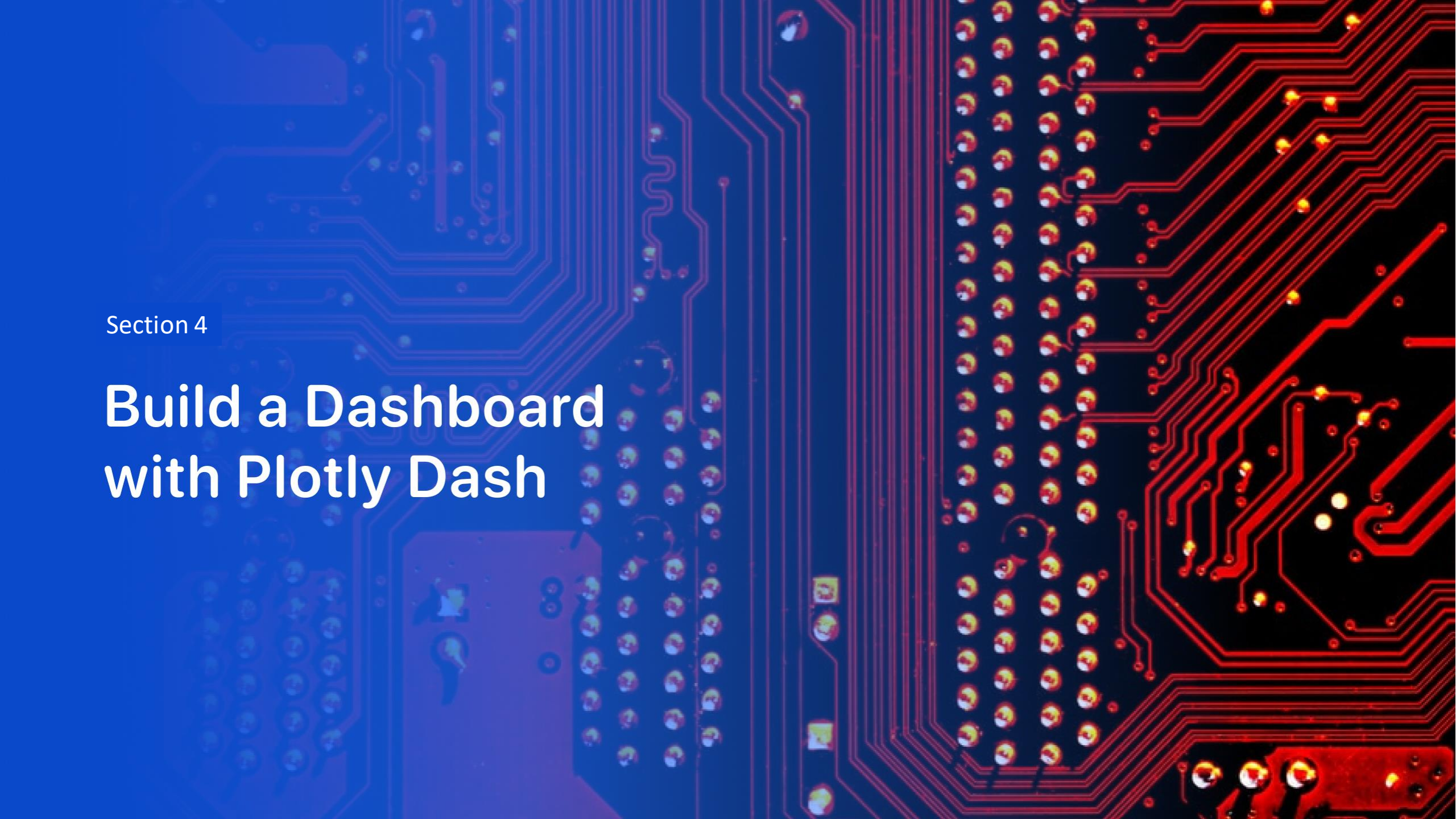
Launch Site Proximity to Population Centers

- Using Folium's **MousePosition()** plugin, it was possible to see the Latitude (Lat) and Longitude (Long) of any location on the map by hovering over the spot with a mouse cursor.
- With the coordinates of the Launch Sites as the starting point, the 'Lat' and 'Long' values of a specific location could then be plugged into a pre-made function to calculate the distance between two points (in KM).
- The same coordinates could be then added to the **PolyLine()** plugin to plot a **Blue** line between the two points with the measured distance being added to the end of the plotted line as a **Red** label.

Note: An example of the above steps can be seen on the next slide. Please pay special attention to the 'Lat' and 'Long' values in the top right hand corner of the map.

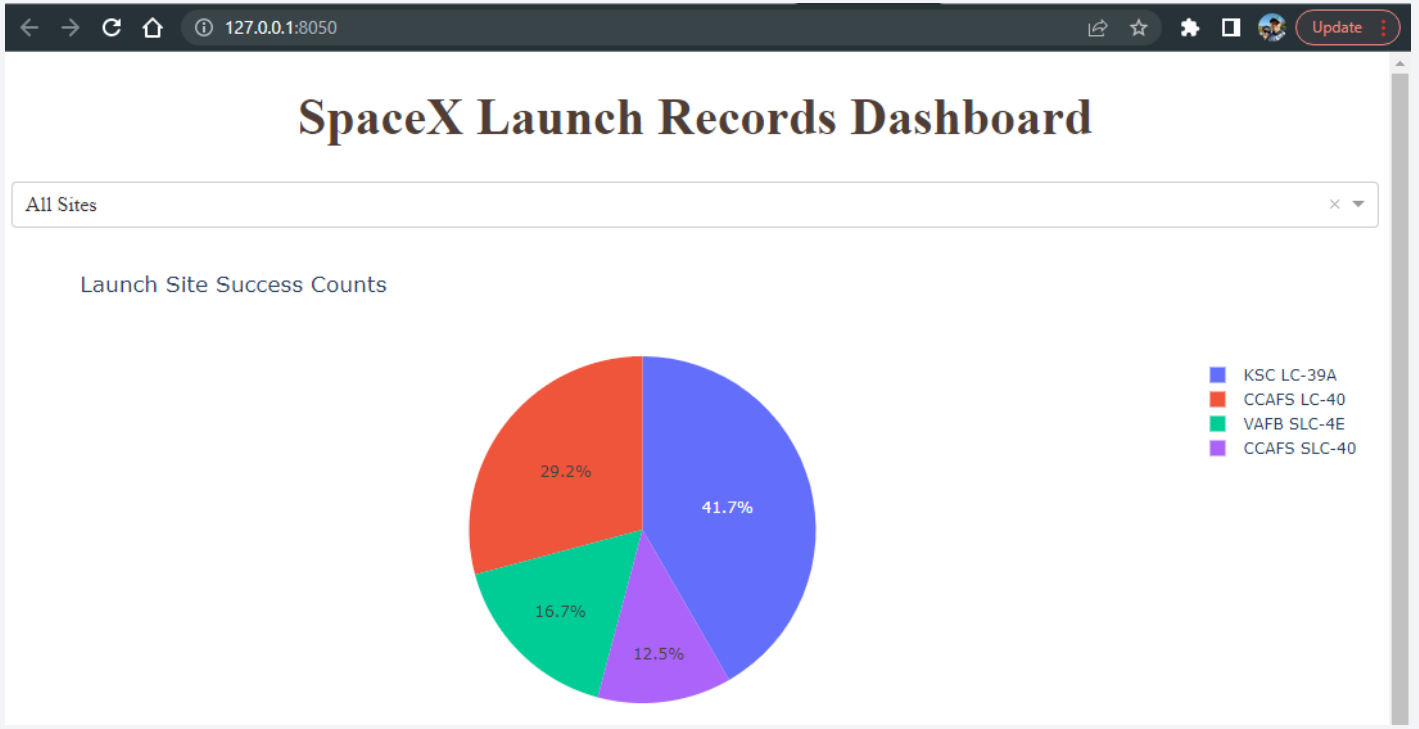
Launch Site Proximity to Population Centers

- Launch Site VAFB SLC-4E's distance to the city of *Lompoc, California* is **14.07 KM** and this city's coordinates can be seen in the top right hand corner of the screenshot.


Section 4
Build a Dashboard with Plotly Dash

Dashboard: All Sites Pie Chart

- The below dashboard screenshot represents **ALL** launch sites success rates percentage for **ALL** successful flights (in total). Here we see that KSC LC-39A was the starting point for **41.7%** of SpaceX successful Falcon 9 landings.

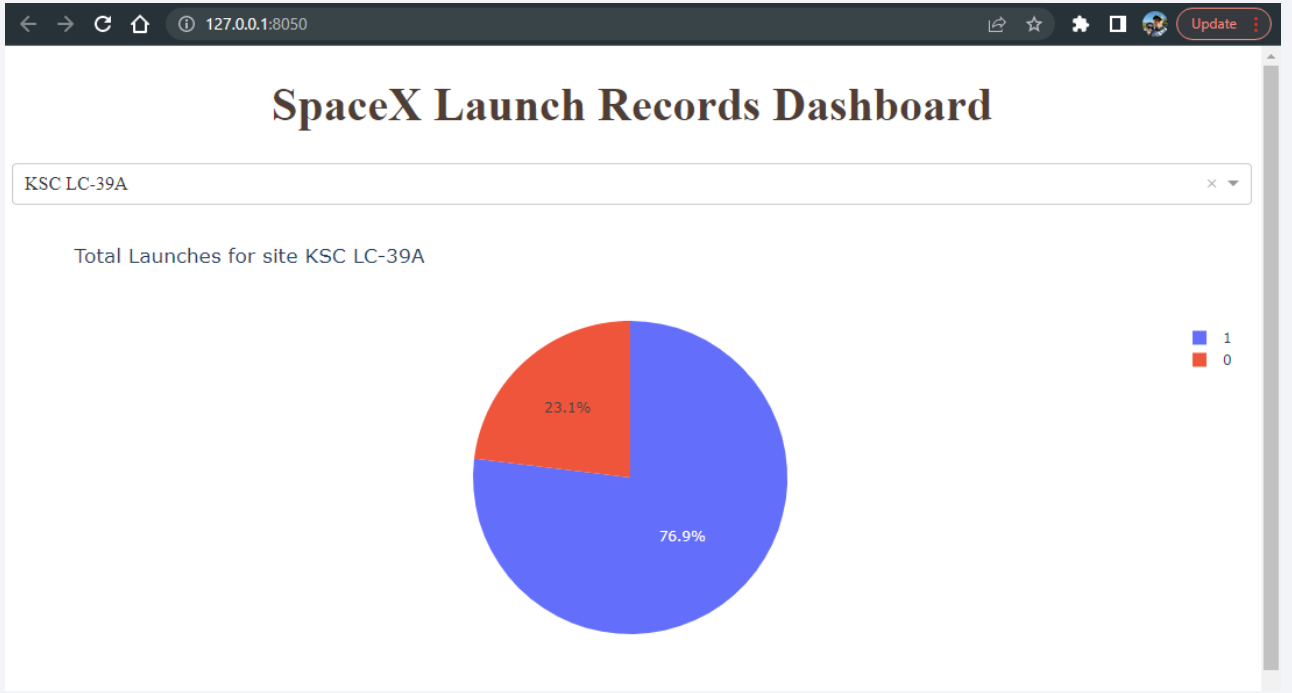


Dashboard: Highest Percentage Launch Site

- When a user runs through the results for each individual Launch Site, while using the shown dropdown picker, they are able to see the success/failure ratio for each location.

- This course of action shows that the KSC LC-39A launch site not only made up **41.7%** of all successful landings, but it also had the highest individual success rate.

- This site successfully landed 10 out of its 13 attempts (**76.9%**).

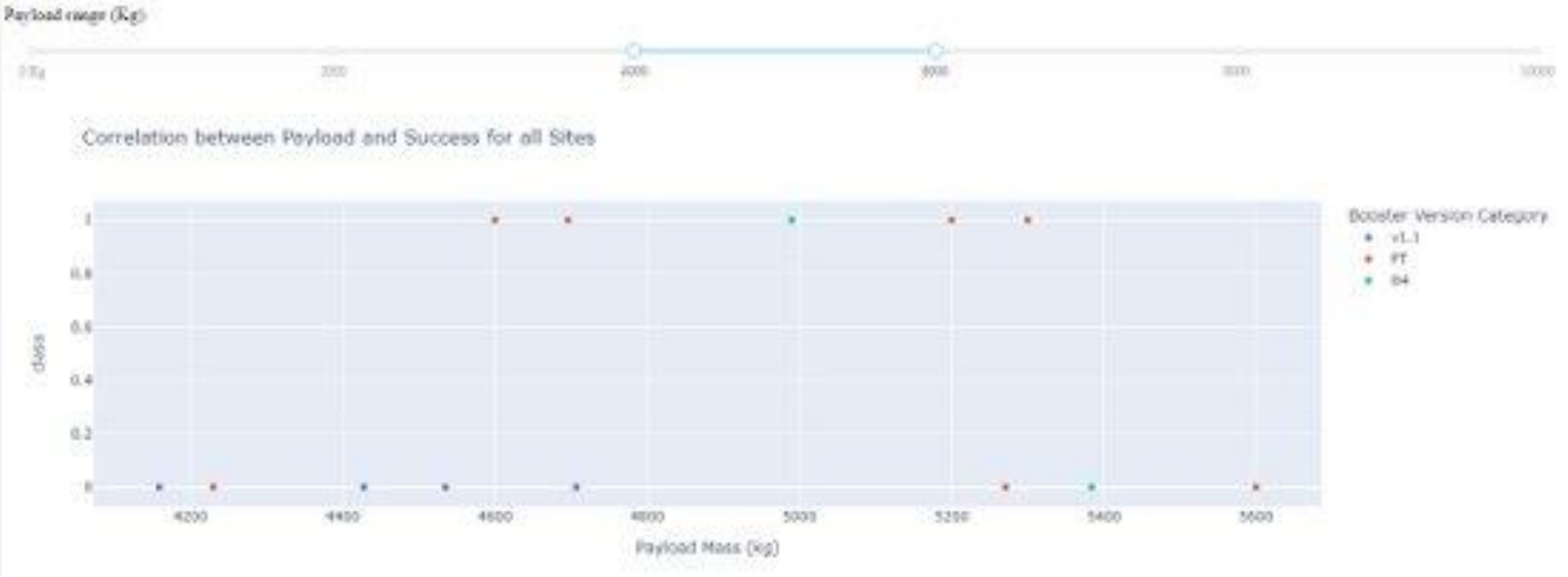


Dashboard: Payload Mass Success by Booster

- The second portion of the created dashboard was an interactive scatterplot that allow for the use of a range slider to select different Payload Mass ranges. This scatterplot also received input from the Launch Site dropdown picker.

Dashboard: Payload Mass Success by Booster

- The below screenshot tshows the results when the Payload Mass ranger slider was adjusted to show launches between 4,000 Kg and 6,000 Kg.
- Here we see that the **FT** Booster version had the most successful landings at this range while **v1.1** booster failed to record a landing.



Dashboard: Payload Mass Success by Booster

- The next Payload Mass range of interest was the **6,000 Kg** and **10,000 Kg** grouping.
- This showed that only the **B4** booster recorded a successful landing in this range and that there were limited attempts made for payload masses above **9,000 Kg.**

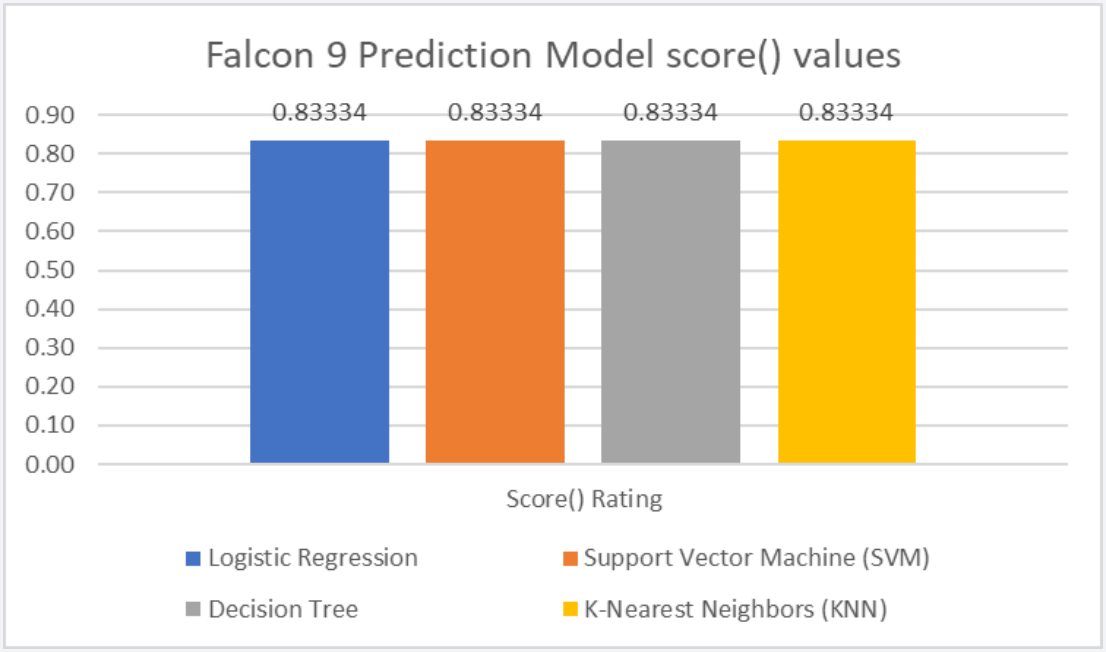
Section 5

Predictive Analysis (Classification)

Classification Accuracy

- While using the **score()** function to gauge the accuracy of each created data model interaction with the TEST split of the SpaceX data, the **Logistic Regression**, **Support Vector Machine (SVM)**, **Decision Tree**, and **K-Nearest Neighbors (KNN)** models each produced the same R^2 score of **0.83334**

- As we were instructed to pick the most accurate model for these remaining slides, the decision was made to display the **Decision Tree** model's results.

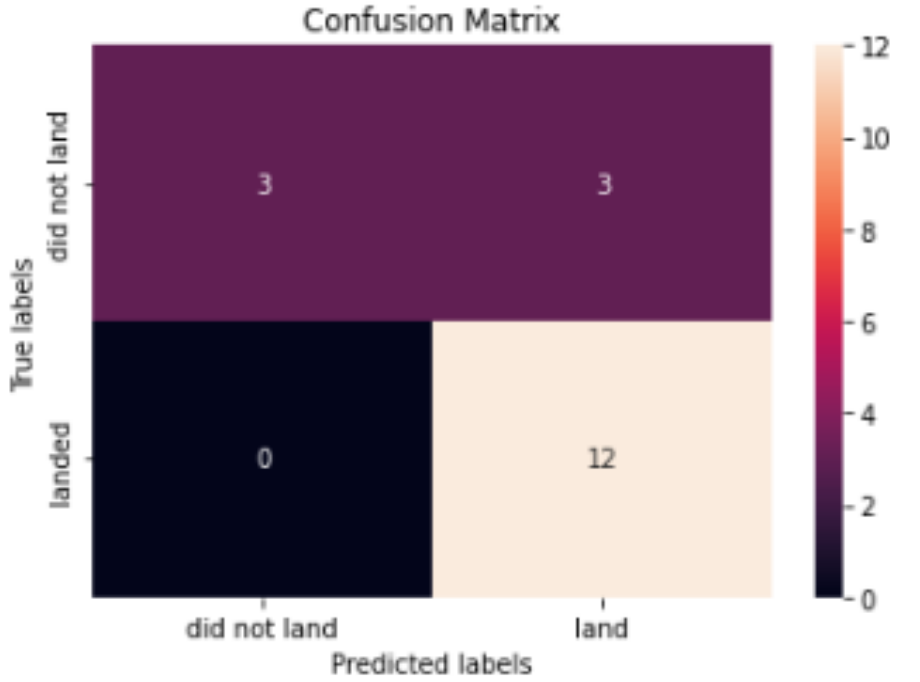


Confusion Matrix

- The **Decsion Tree's** accuracy score was noticeably higher while fitting the TRAIN data set (**0.8892**), therefor it made sense to select this model as having the most potential of the four created objects.

- Since the **TEST** accuracy scores were all equal, one can assume overfitting was not the cause of the higher score while using the **TRAIN** dataset.

```
yhat2 = tree_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat2)
```



Results, Conclusion, & Appendix

Results

- As the four R^2 scores and the confusion matrixes show, each data model technique produced the same exact results while attempting to predict Falcon 9 successful landings while using the TEST datasets.
  - R^2 = 0.83334
  - Accurately predicting **15** of the launches
  - Inaccurately predicting that **3** of the successful landings did not land
- The conformity of the four model's results is likely related to the small size of the SpaceX dataset(s) and also because Python's SKlearn library makes obtaining the most efficient hyperparameters relatively painless and easy.

Conclusions

- The purpose of this SpaceX data exploration was to build an accurate data model to predict Falcon 9 successful landings and that undertaking has been fruitful. Four different data models were created using the available data and each one had an accuracy score rating over **.80** in their final evaluation.

- There were other insights gained while exploring the collected data and these discoveries could then be combined with the created data models to find a niche area to compete within the private space industry sector.

  - The vast majority of SpaceX's launches had payloads below **8,000 Kg**
  - SpaceX mostly had rockets delivering payloads in a select few Orbit types: **GTO**, **VLEO**, **ISS**, **PO**, & **LEO**
  - SpaceY could aggressively pursue Orbit bids when targeting **GOE**, **SO**, **MEO**, **HEO**, and **ES-L1**

Conclusions

- It should be noted that the Falcon 9 data sample was extremely small and that the dependent variable (i.e. Successful Landing = Yes/No) only had two possible outcomes. SpaceY should not be using the data models as the sole deciding point to make million dollar decisions.

- Another important statistic gained through the exploratory data analysis is that SpaceX 's total success rate for successful landings steadily rose from 2013 to 2020. There was a slight dip in 2018 in their success rate, but overall SpaceX's prowess at rocket launches appears to be getting more solid as they gain experience.

- The gained insights and learned tendencies for the SpaceX Falcon 9 program could be skewed due to failures in the beginning of SpaceX's entry into the rocket and space launch industry.

- It is very possible SpaceX's ever increasing success rates for its launches could render everything in this data analysis writeup moot.

Appendix A

FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude
1	6/4/2010	Falcon 9	6123.547647	LEO	CCSFS SLC 40	None None	1	FALSE	FALSE	FALSE		1	0	B0003	-80.577366	28.5618571
2	5/22/2012	Falcon 9	525	LEO	CCSFS SLC 40	None None	1	FALSE	FALSE	FALSE		1	0	B0005	-80.577366	28.5618571
3	3/1/2013	Falcon 9	677	ISS	CCSFS SLC 40	None None	1	FALSE	FALSE	FALSE		1	0	B0007	-80.577366	28.5618571
4	9/29/2013	Falcon 9	500	PO	VAFB SLC 4E	False Ocean	1	FALSE	FALSE	FALSE		1	0	B1003	-120.610829	34.632093
5	12/3/2013	Falcon 9	3170	GTO	CCSFS SLC 40	None None	1	FALSE	FALSE	FALSE		1	0	B1004	-80.577366	28.5618571
6	1/6/2014	Falcon 9	3325	GTO	CCSFS SLC 40	None None	1	FALSE	FALSE	FALSE		1	0	B1005	-80.577366	28.5618571
7	4/18/2014	Falcon 9	2296	ISS	CCSFS SLC 40	True Ocean	1	FALSE	FALSE	TRUE		1	0	B1006	-80.577366	28.5618571
8	7/14/2014	Falcon 9	1316	LEO	CCSFS SLC 40	True Ocean	1	FALSE	FALSE	TRUE		1	0	B1007	-80.577366	28.5618571
9	8/5/2014	Falcon 9	4535	GTO	CCSFS SLC 40	None None	1	FALSE	FALSE	FALSE		1	0	B1008	-80.577366	28.5618571
10	9/7/2014	Falcon 9	4428	GTO	CCSFS SLC 40	None None	1	FALSE	FALSE	FALSE		1	0	B1011	-80.577366	28.5618571
11	9/21/2014	Falcon 9	2216	ISS	CCSFS SLC 40	False Ocean	1	FALSE	FALSE	FALSE		1	0	B1010	-80.577366	28.5618571
12	1/10/2015	Falcon 9	2395	ISS	CCSFS SLC 40	False ASDS	1	TRUE	FALSE	TRUE	5e9e3032383ecb761634e7cb	1	0	B1012	-80.577366	28.5618571
13	2/11/2015	Falcon 9	570	ES-L1	CCSFS SLC 40	True Ocean	1	TRUE	FALSE	TRUE		1	0	B1013	-80.577366	28.5618571
14	4/14/2015	Falcon 9	1898	ISS	CCSFS SLC 40	False ASDS	1	TRUE	FALSE	TRUE	5e9e3032383ecb761634e7cb	1	0	B1015	-80.577366	28.5618571
15	4/27/2015	Falcon 9	4707	GTO	CCSFS SLC 40	None None	1	FALSE	FALSE	FALSE		1	0	B1016	-80.577366	28.5618571
16	6/28/2015	Falcon 9	2477	ISS	CCSFS SLC 40	None ASDS	1	TRUE	FALSE	TRUE	5e9e3032383ecb6bb234e7ca	1	0	B1018	-80.577366	28.5618571
17	12/22/2015	Falcon 9	2034	LEO	CCSFS SLC 40	True RTLS	1	TRUE	FALSE	TRUE	5e9e3032383ecb267a34e7c7	1	0	B1019	-80.577366	28.5618571
18	1/17/2016	Falcon 9	553	PO	VAFB SLC 4E	False ASDS	1	TRUE	FALSE	TRUE	5e9e3033383ecbb9e534e7cc	1	0	B1017	-120.610829	34.632093
19	3/4/2016	Falcon 9	5271	GTO	CCSFS SLC 40	False ASDS	1	TRUE	FALSE	TRUE	5e9e3032383ecb6bb234e7ca	1	0	B1020	-80.577366	28.5618571

Appendix B

Flight No.	Launch site	Payload	Payload mass	Orbit	Customer	Launch outcome	Version Booster	Booster landing	Date	Time
1	CCAFS	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX		F9 v1.0B0003.1	Failure	4-Jun-10	18:45
2	CCAFS	Dragon	0	LEO	NASA	Success	F9 v1.0B0004.1	Failure	8-Dec-10	15:43
3	CCAFS	Dragon	525 kg	LEO	NASA	Success	F9 v1.0B0005.1		22-May-12	7:44
4	CCAFS	SpaceX CRS-1	4,700 kg	LEO	NASA		F9 v1.0B0006.1	No attempt	8-Oct-12	0:35
5	CCAFS	SpaceX CRS-2	4,877 kg	LEO	NASA		F9 v1.0B0007.1		1-Mar-13	15:10
6	VAFB	CASSIOPE	500 kg	Polar orbit	MDA	Success	F9 v1.1B1003	Uncontrolled	29-Sep-13	16:00
7	CCAFS	SES-8	3,170 kg	GTO	SES	Success	F9 v1.1	No attempt	3-Dec-13	22:41
8	CCAFS	Thaicom 6	3,325 kg	GTO	Thaicom	Success	F9 v1.1	No attempt	6-Jan-14	22:06
9	Cape Canaveral	SpaceX CRS-3	2,296 kg	LEO	NASA		F9 v1.1	Controlled	18-Apr-14	19:25
10	Cape Canaveral	Orbcomm-OG2	1,316 kg	LEO	Orbcomm	Success	F9 v1.1	Controlled	14-Jul-14	15:15
11	Cape Canaveral	AsiaSat 8	4,535 kg	GTO	AsiaSat	Success	F9 v1.1	No attempt	5-Aug-14	8:00
12	Cape Canaveral	AsiaSat 6	4,428 kg	GTO	AsiaSat	Success	F9 v1.1		7-Sep-14	5:00
13	Cape Canaveral	SpaceX CRS-4	2,216 kg	LEO	NASA	Success	F9 v1.1	Uncontrolled	21-Sep-14	5:52
14	Cape Canaveral	SpaceX CRS-5	2,395 kg	LEO	NASA	Success	F9 v1.1	Failure	10-Jan-15	9:47
15	Cape Canaveral	DSCOVR	570 kg	HEO	USAF		F9 v1.1	Controlled	11-Feb-15	23:03
16	Cape Canaveral	ABS-3A	4,159 kg	GTO	ABS		F9 v1.1	No attempt	2-Mar-15	3:50
17	Cape Canaveral	SpaceX CRS-6	1,898 kg	LEO	NASA		F9 v1.1	Failure	14-Apr-15	20:10
18	Cape Canaveral	TÃ¼rkmenÃ„lem 52Â°E / MonacoSAT	4,707 kg	GTO			F9 v1.1	No attempt	27-Apr-15	23:03
19	Cape Canaveral	SpaceX CRS-7	1,952 kg	LEO	NASA	Failure	F9 v1.1	Precluded	28-Jun-15	14:21

Appendix C

FlightNumber	Date	BoosterVe	PayloadM	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCo	Serial	Longitude	Latitude	Class
1	6/4/2010	Falcon 9	6104.959	LEO	CCAFS SLC 40	None None	1	FALSE	FALSE	FALSE		1	0	B0003	-80.5774	28.56186	0
2	5/22/2012	Falcon 9	525	LEO	CCAFS SLC 40	None None	1	FALSE	FALSE	FALSE		1	0	B0005	-80.5774	28.56186	0
3	3/1/2013	Falcon 9	677	ISS	CCAFS SLC 40	None None	1	FALSE	FALSE	FALSE		1	0	B0007	-80.5774	28.56186	0
4	9/29/2013	Falcon 9	500	PO	VAFB SLC 4E	False Ocean	1	FALSE	FALSE	FALSE		1	0	B1003	-120.611	34.63209	0
5	12/3/2013	Falcon 9	3170	GTO	CCAFS SLC 40	None None	1	FALSE	FALSE	FALSE		1	0	B1004	-80.5774	28.56186	0
6	1/6/2014	Falcon 9	3325	GTO	CCAFS SLC 40	None None	1	FALSE	FALSE	FALSE		1	0	B1005	-80.5774	28.56186	0
7	4/18/2014	Falcon 9	2296	ISS	CCAFS SLC 40	True Ocean	1	FALSE	FALSE	TRUE		1	0	B1006	-80.5774	28.56186	1
8	7/14/2014	Falcon 9	1316	LEO	CCAFS SLC 40	True Ocean	1	FALSE	FALSE	TRUE		1	0	B1007	-80.5774	28.56186	1
9	8/5/2014	Falcon 9	4535	GTO	CCAFS SLC 40	None None	1	FALSE	FALSE	FALSE		1	0	B1008	-80.5774	28.56186	0
10	9/7/2014	Falcon 9	4428	GTO	CCAFS SLC 40	None None	1	FALSE	FALSE	FALSE		1	0	B1011	-80.5774	28.56186	0
11	9/21/2014	Falcon 9	2216	ISS	CCAFS SLC 40	False Ocean	1	FALSE	FALSE	FALSE		1	0	B1010	-80.5774	28.56186	0
12	1/10/2015	Falcon 9	2395	ISS	CCAFS SLC 40	False ASDS	1	TRUE	FALSE	TRUE	5e9e3032383ecb761634e7cb	1	0	B1012	-80.5774	28.56186	0
13	2/11/2015	Falcon 9	570	ES-L1	CCAFS SLC 40	True Ocean	1	TRUE	FALSE	TRUE		1	0	B1013	-80.5774	28.56186	1
14	4/14/2015	Falcon 9	1898	ISS	CCAFS SLC 40	False ASDS	1	TRUE	FALSE	TRUE	5e9e3032383ecb761634e7cb	1	0	B1015	-80.5774	28.56186	0
15	4/27/2015	Falcon 9	4707	GTO	CCAFS SLC 40	None None	1	FALSE	FALSE	FALSE		1	0	B1016	-80.5774	28.56186	0
16	6/28/2015	Falcon 9	2477	ISS	CCAFS SLC 40	None ASDS	1	TRUE	FALSE	TRUE	5e9e3032383ecb6bb234e7ca	1	0	B1018	-80.5774	28.56186	0
17	12/22/2015	Falcon 9	2034	LEO	CCAFS SLC 40	True RTLS	1	TRUE	FALSE	TRUE	5e9e3032383ecb267a34e7c7	1	0	B1019	-80.5774	28.56186	1
18	1/17/2016	Falcon 9	553	PO	VAFB SLC 4E	False ASDS	1	TRUE	FALSE	TRUE	5e9e3033383ecbb9e534e7cc	1	0	B1017	-120.611	34.63209	0
19	3/4/2016	Falcon 9	5271	GTO	CCAFS SLC 40	False ASDS	1	TRUE	FALSE	TRUE	5e9e3032383ecb6bb234e7ca	1	0	B1020	-80.5774	28.56186	0

Appendix D

- List the names of the boosters that had successful Drone Ship landing while having a PayloadMass value between 4K and 6K Kilograms

```
res = cur.execute(
    """
    SELECT Booster_Version
    FROM SPACEXTBL
    WHERE "Landing _Outcome" = 'Success (drone ship)'
      AND PAYLOAD_MASS__KG_ between 4000 and 6000
    """)
res.fetchall()
```

```
[('F9 FT B1022',), ('F9 FT B1026',), ('F9 FT  B1021.2',), ('F9 FT  B1031.2',)]
```

Appendix D (continued)

- For launches in 2015, list the month, booster version, launch site and the landing outcomes while using a drone ship.

```
# Quotes needed around "Landing _Outcome" due
# to the whitespace before the underscore
res = cur.execute(
    """
    SELECT substr(Date, 4, 2) as "Month",
    "Landing _Outcome" as "Landing_Outcome",
    Booster_Version,
    Launch_Site
    FROM SPACEXTBL
    Where substr(Date,7,4)='2015'
       AND "Landing _Outcome" Like '%drone%ship%'
    """)
res.fetchall()
```

```
[('01', 'Failure (drone ship)', 'F9 v1.1 B1012', 'CCAFS LC-40'),
 ('04', 'Failure (drone ship)', 'F9 v1.1 B1015', 'CCAFS LC-40'),
 ('06', 'Precluded (drone ship)', 'F9 v1.1 B1018', 'CCAFS LC-40')]
```

Appendix D (continued)

- The booster versions that have carried the maximum payload mass.

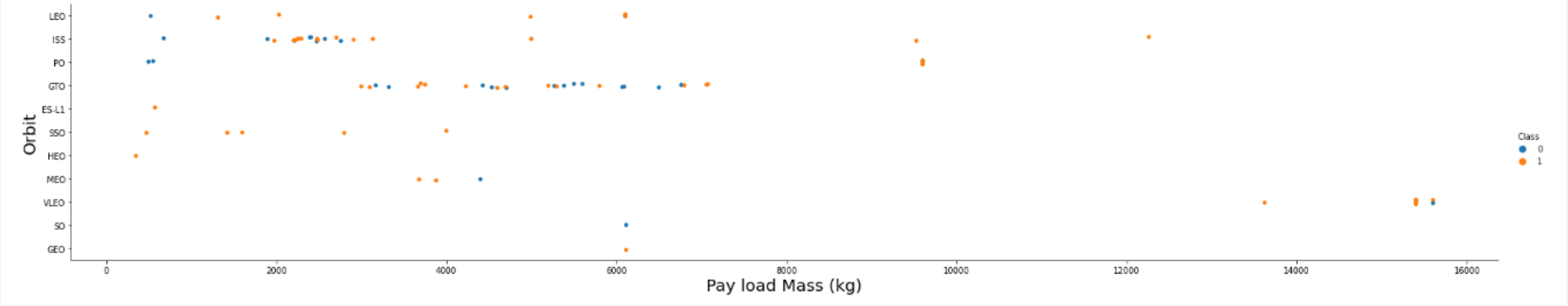
```
res = cur.execute(
    """
    SELECT Distinct Booster_Version
    FROM SPACEXTBL
    WHERE PAYLOAD_MASS__KG_ in (select MAX(PAYLOAD_MASS__KG_) FROM SPACEXTBL)
    """)
res.fetchall()
```

```
[('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 ',)]
```

Appendix E

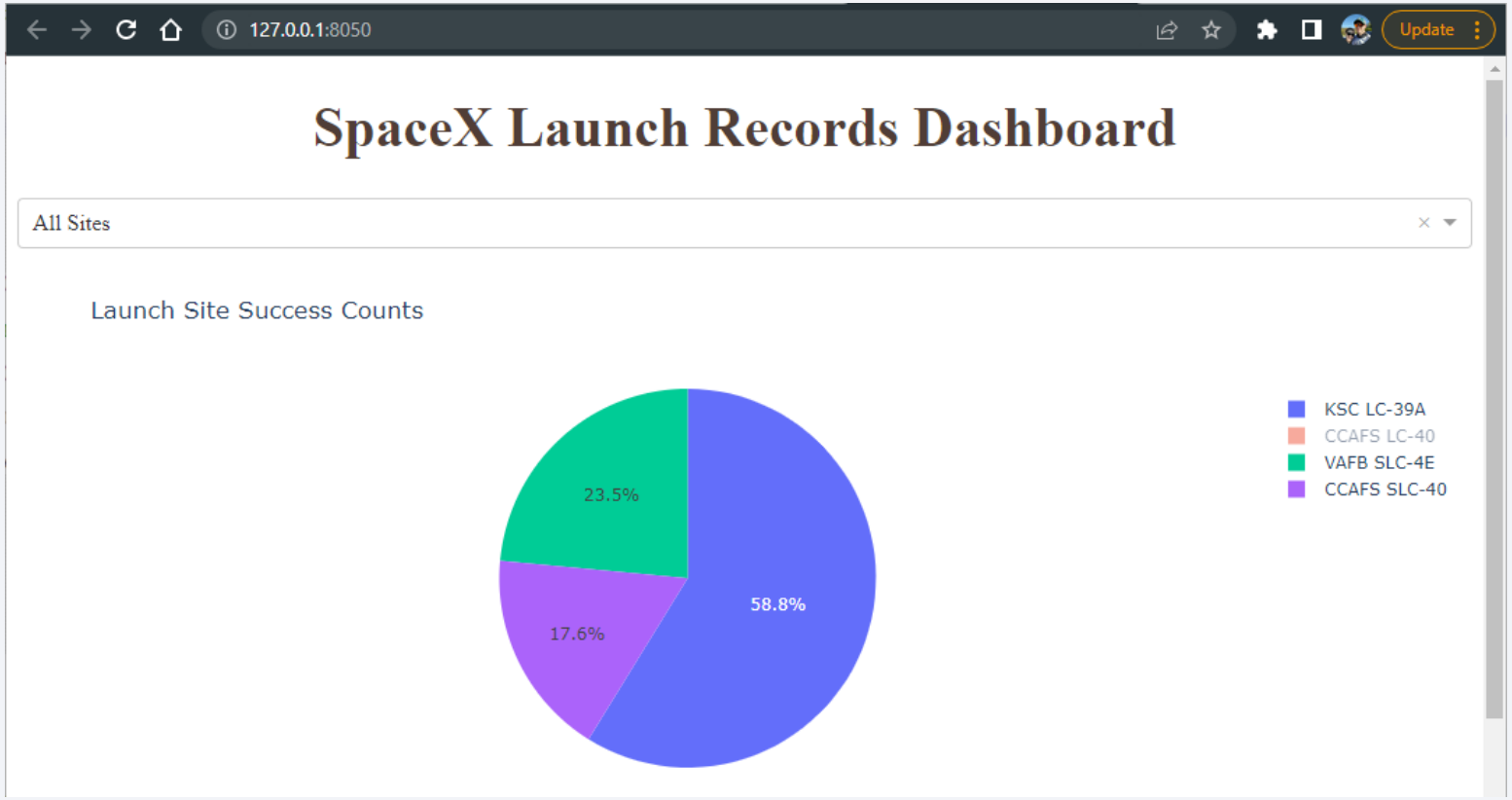
```
# Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, and hue to be th

sns.catplot(y="Orbit", x="PayloadMass", hue="Class", data=df, aspect = 5)
plt.ylabel("Orbit",fontsize=20)
plt.xlabel("Pay load Mass (kg)",fontsize=20)
plt.show()
```



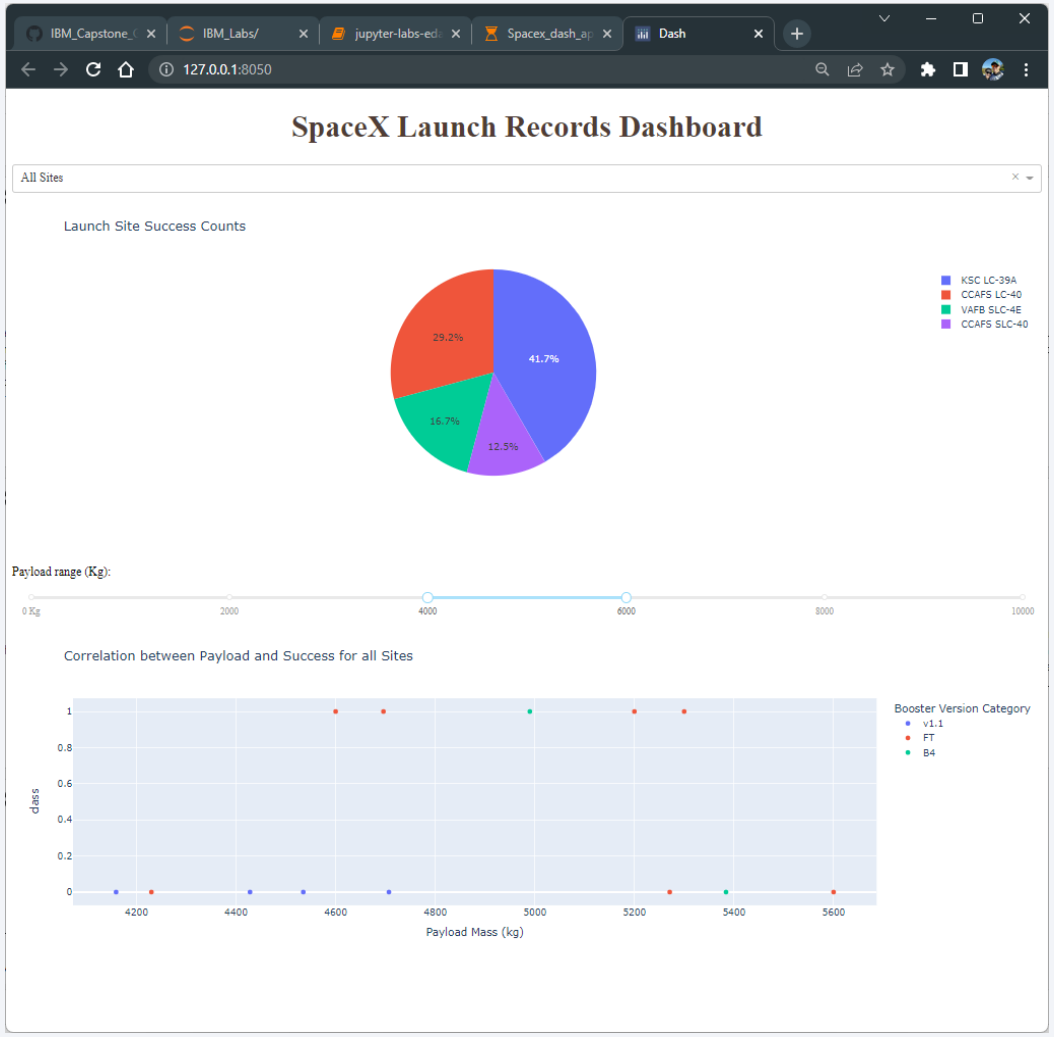
Appendix F

- CCADS LC-40 removed from the total All Sites list by clicking on it in the Legend .



Appendix G

- Screenshot of the full dashboard



Thank you!