IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Tony Primerano
09/08/2023

Outline

Executive Summary

Summary of methodologies

This project followed the steps for analysis listed below:

- Data Collection
- Data Wrangling
- Exploratory Analysis - SQL
- Interactive Visual Analytics - Folium, Plotly
- Predictive Analysis - Classification

Summary of all results

The following visualizations and outputs were produced:

- Exploratory Data Analysis results
- Geospatial Data
- Interactive Dashboard
- Predictive Analysis of Classification Models

Introduction

SpaceX launches the Falcon9 rocket for an estimated cost of $62m per launch compared to $165m per launch from other industry providers. These savings are due to SpaceX’s ability to land and re-use the first stage of the rocket for subsequent launches.

If we are able to make predictions regarding the first stage’s ability to land, we can determine the cost per launch and assess whether SpaceX or an alternate company should bid for a rocket launch.

This project will ultimately predict if the Falcon9 first stage from SpaceX will land successfully.

Section 1

Methodology

Methodology

Executive Summary

Data collection methodology:

- GET requests were made to the SpaceX REST API
- Web Scraping from Wikipedia

Perform data wrangling:

- One-hot encoding was applied to categorical features
- .fillna() used to remove NaN values
- Landing outcome label was created to show:
  - 0 for unsuccessful landings
  - 1 for successful landings
- .value_counts() was used to determine:
  - number of launches for each site
  - number of occurrence for each orbit
  - occurrence and number of outcomes for each orbit

Exploratory Data Analysis (EDA) using visualization and SQL:

- Using SQL queries, the data was manipulated and evaluated
- Pandas and Matplotlib were used to visualize relationships and determine patterns

Perform interactive visual analytics using Folium and Plotly Dash:

- Geospatial Analytics using Folium
- Interactive dashboard created using Plotly Dash

Perform predictive analysis using classification models:

- Data was standardized
- Train_test_split used to split data into train and test sets
- Trained different classification models on the data
- Found hyperparameters using GridsearchCV
- Confusion matrices for each model were plotted
- Accuracy of each model was assessed

Data Collection - SpaceX REST API

The SpaceX REST API was used to retrieve data pertaining to launches of the Falcon9 rocket, including the rocket used, payload delivered, launch specifications, and landing outcomes.

1. GET requests were made to the API
2. Responses were decoded as a .JSON file and turned into a dataframe using Pandas
3. Data was cleaned and missing values were filled
4. Web Scraping through BeautifulSoup was used to obtain launch records for Falcon9

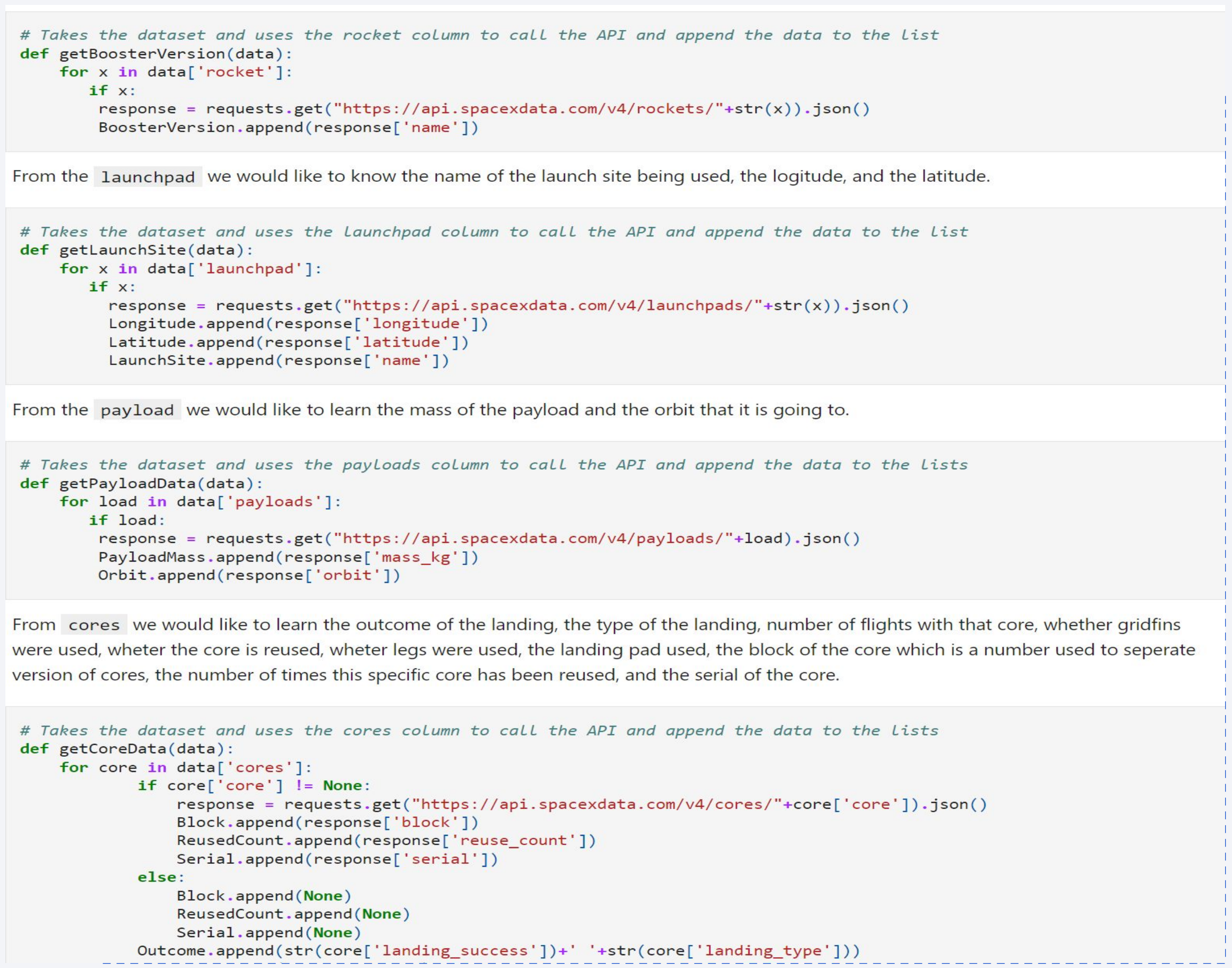
Objective: extract launch records as HTML table, parse the table and convert it to a Pandas dataframe for future analysis

Data Collection – SpaceX API

- The code used can be viewed at: https://github.com/Tonyprimo/Capstone/blob/c7d133f0d26e9c577bdce2a99f5266c00be232b4/Collecting%20the%20Data/jupyter-labs-spacex-data-collection-api.ipynb

Steps:

1. Request launch data from SpaceX API
2. Decode response as .JSON file
3. Request information about the launches using custom functions
4. Create dictionary for data
5. Create dataframe
6. Filter dataframe to include only Falcon9
7. Replace missing values for payload mass with calculated mean
8. Export the data to .CSV file



```
# Takes the dataset and uses the rocket column to call the API and append the data to the list
def getBoosterVersion(data):
    for x in data['rocket']:
       if x:
        response = requests.get("https://api.spacexdata.com/v4/rockets/"+str(x)).json()
        BoosterVersion.append(response['name'])
```

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

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

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

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

From `cores` we would like to learn the outcome of the landing, the type of the landing, number of flights with that core, whether gridfins were used, wheter the core is reused, wheter legs were used, the landing pad used, the block of the core which is a number used to seperate version of cores, the number of times this specific core has been reused, and the serial of the core.

```
# Takes the dataset and uses the cores column to call the API and append the data to the lists
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cores/"+core['core']).json()
                Block.append(response['block'])
                ReusedCount.append(response['reuse_count'])
                Serial.append(response['serial'])
            else:
                Block.append(None)
                ReusedCount.append(None)
                Serial.append(None)
            Outcome.append(str(core['landing_success'])+' '+str(core['landing_type']))
```

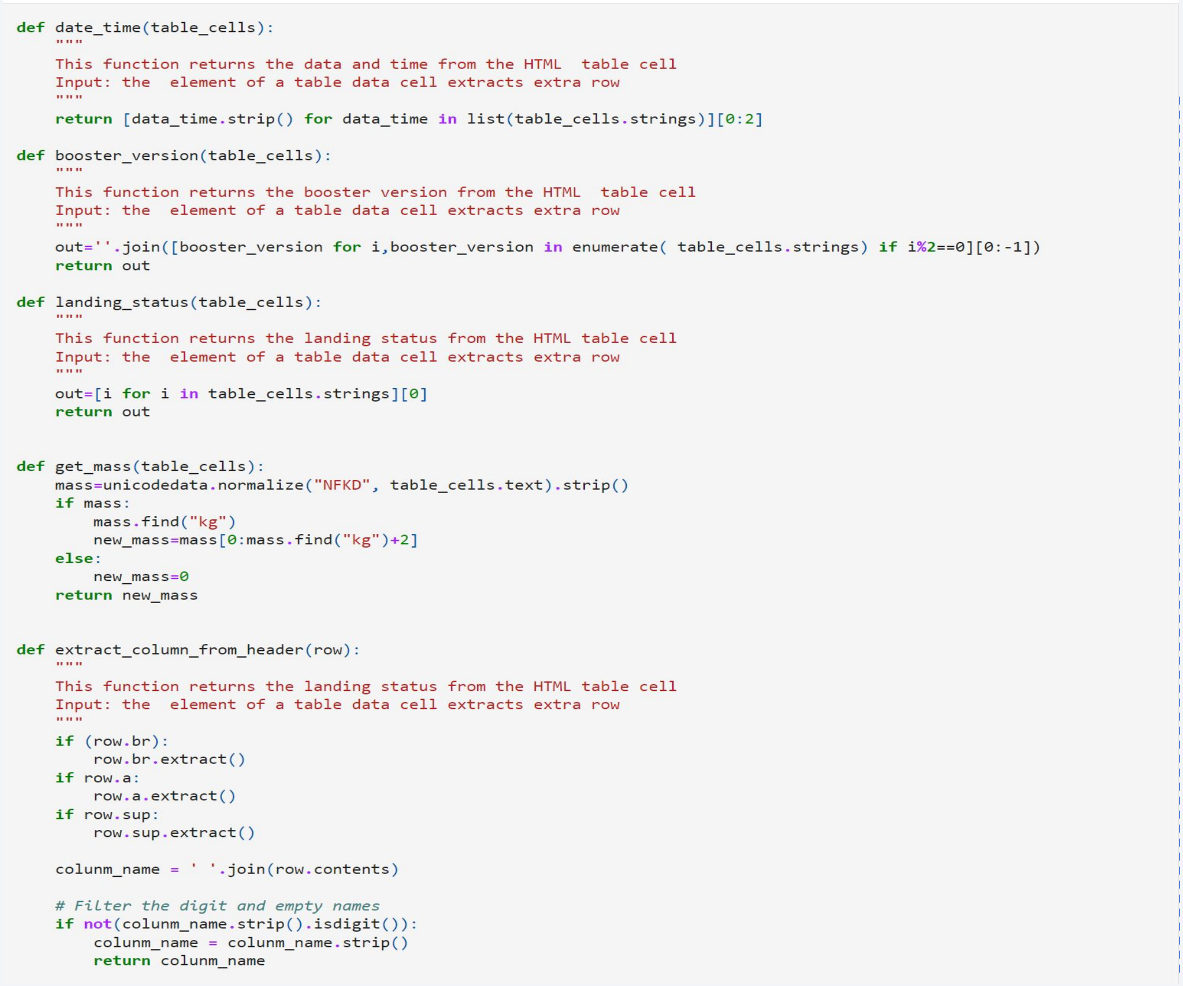
Data Collection - Web Scraping

The code used can be viewed at:

https://github.com/Tonyprimo/Capstone/blob/c7d133f0d26e9c577bdce2a99f5266c00be232b4/Collecting%20the%20Data/jupyter-labs-webscraping.ipynb

Steps:

1. Request Falcon9 launch data from Wikipedia
2. Create BeautifulSoup object from HTML response
3. Extract column names from the table header
4. Parse HTML table to collect data
5. Create dictionary from the data
6. Create dataframe from the dictionary
7. Export the data to .CSV file



```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
```

Data Wrangling

The code can be viewed at:
https://github.com/Tonyprimo/Capstone/blob/c7d133f0d26e9c577bdce2a99f5266c00be232b4/Data%20Wrangling/IBM-DS0321EN-SkillsNetwork_labs_module_1_L3_labs-jupyter-spacex-data_wrangling_jupyterlite.jupyterlite.ipynb

Steps:

1. perform EDA and determine labels for data
2. Calculate the number of launches for each site, number of occurrences of orbit, and occurrence of mission outcome per orbit.
3. create a binary landing outcome column
4. export data to .CSV file

Landing Outcomes:

- landing was not always successful
- True Ocean: outcome was successful landing to specific region of the ocean
- False Ocean: outcome was failed landing to specific region of the ocean
- True RTLS: outcome was successful landing on ground pad
- False RTLS: outcome was failed landing on ground pad
- True ASDS: outcome was successful landing on a drone ship
- False ASDS: outcome was a failed landing on a drone ship

Outcomes were converted into 0 for failed landings and 1 for successful landings

EDA with Data Visualization

Code can be viewed at:

https://github.com/Tonyprimo/Capstone/blob/c7d133f0d26e9c577bdce2a99f5266c00be232b4/Exploratory%20Analysis/IBM-DS0321EN-SkillsNetwork_labs_module_2_jupyter-labs-eda-dataviz.ipynb.jupyterlite.ipynb

Charts:

- Flight Number vs Payload
- Flight Number vs Launch Site
- Payload Mass (kg) vs Launch Site
- Payload Mass (kg) vs Orbit Type

Analysis:

- Relationships were viewed using scatterplots in order to determine if the variables would be useful for machine learning model.
- Comparisons were shown using bar charts for categorical data to determine presence of relationships among the categories and a measured value.

EDA with SQL

Code can be viewed at:
https://github.com/Tonyprimo/Capstone/blob/c7d133f0d26e9c577bdce2a99f5266c00be232b4/Exploratory%20Analysis/jupyter-labs-eda-sql-coursera_sqllite.ipynb

Queries used:

Display:

- Names of unique launch sites
- 5 record where the launch site name begins with “CCA”
- Total payload mass carried by boosters from NASA (CRS)
- Average payload mass carried by booster version F9 v1.1

List:

- first successful landing on ground pad date
- Names of boosters from successful landings on drone ship with masses between 4,000-6,000 kg
- Total successful landings on ground pad
- Names of booster versions carrying max payload
- Failed outcomes on drone ships with their respective booster versions and launch sites for the year 2015
- Count of landing outcomes from June 2010 to March 2017 in descending order

Build an Interactive Map with Folium

Code can be viewed at:
https://github.com/Tonyprimo/Capstone/blob/c7d133f0d26e9c577bdce2a99f5266c00be232b4/Interactive%20Visual%20Analytics/IBM-DS0321EN-SkillsNetwork_labs_module_3_lab_jupyter_launch_site_location.jupyterlite.ipynb

Launch Site Markers:

- Blue circle indicates NASA Johnson Space Center with a pop up for its name using latitude and longitude for the center
- Red circles indicate all launch site coordinates with a pop up for its respective name using latitude and longitude

Colored Markers for Launch Outcomes

- Colored markers were added for each launch outcome for each launch site. Green markers were used to indicate successful outcomes while red markers were used to indicate failed outcomes. This was used to aid in determining which launch site has the highest success rate.

Distance Between Launch Site to Proximities:

- colored lines show the distance between launch site CCAFS SLC-40 and its proximity to the nearest Coastline, highway,city, and railway system.

Build a Dashboard with Plotly Dash

Pie Charts: Built to show the successful launches per launch site

- Drop downs were created to be able to filter results for the successes and failures at each site.

Scatter plot: Built to show relationships between Payload Mass (kg) and outcome of the launch.

- Range sliders were created to filter results by payload.
- An additional filter was created to filter by booster version.

Code can be viewed at:

https://github.com/Tonyprimo/Capstone/blob/c7d133f0d26e9c577bdce2a99f5266c00be232b4/Interactive%20Visual%20Analytics/spacex_dash_app.py

Predictive Analysis (Classification)

1. Model Development:
   - To prepare the dataset for the model, the data was standardized and pre-processed, split into training and testing data using train_test_split(), and then used to determine which algorithm was the best fit for the model.
   - For each algorithm, GridSearchCV was used to create objects and parameters, the objects were then fitted to parameters, and the training data was used to train the model.
2. Model Evaluation:
   - Using the output from each GridSearchCV, the tuned hyper parameters were checked using best_params_, accuracy was checked using score and best_score_, and a confusion matrix was plotted.
3. Best Classification model:
   - accuracy scores for all algorithms were reviewed and the best model was determined by the algorithm with the highest accuracy score to be the best performing model.

Code can be viewed at:

https://github.com/Tonyprimo/Capstone/blob/16c45dc7b73b79a2bb7f8e5b8b874cfccfd14615/predictive%20analysis/IBM-DS0321EN-SkillsNetwork_labs_module_4_SpaceX_Machine_Learning_Prediction_Part_5.jupyterlite.ipynb

Results

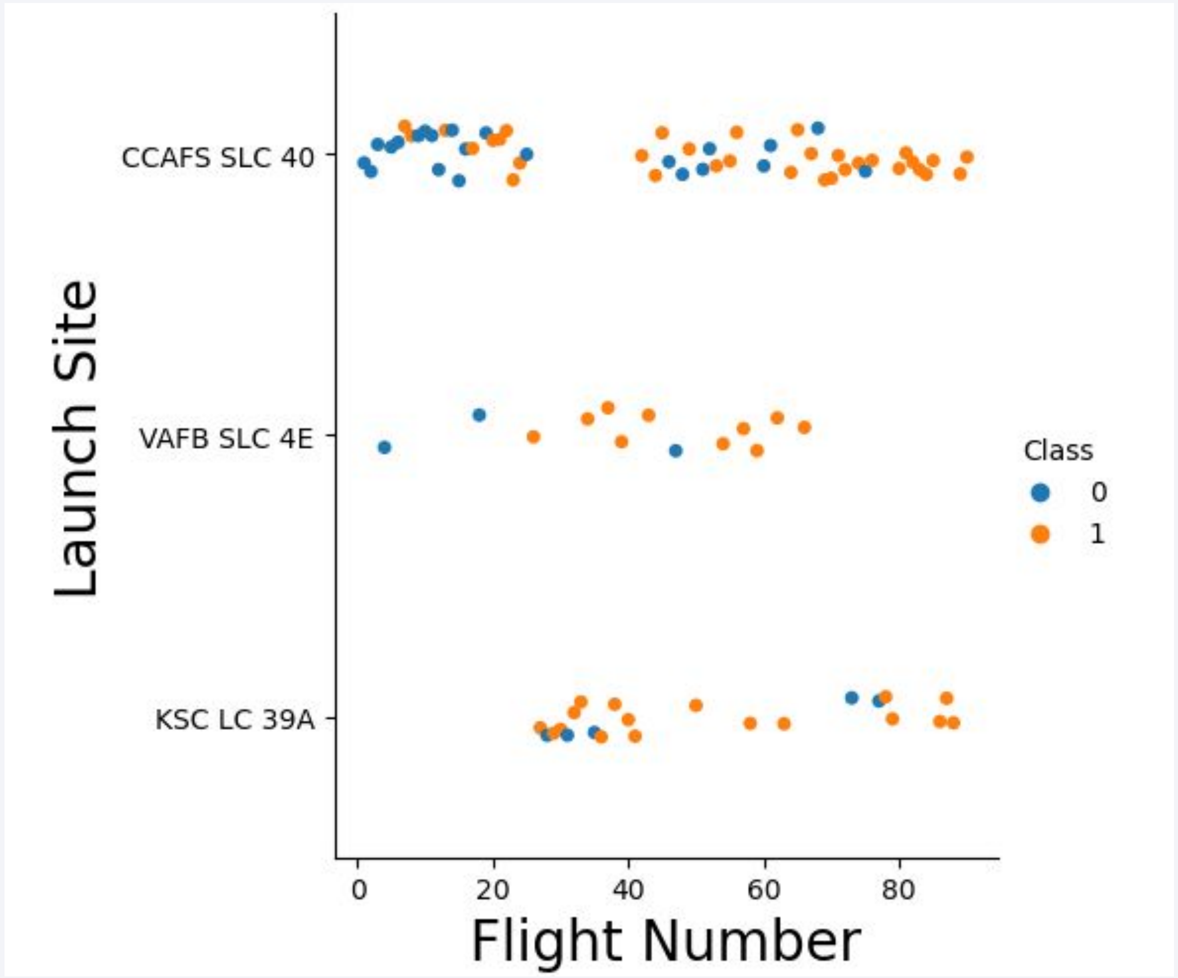
- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results

Section 2

Insights drawn from EDA

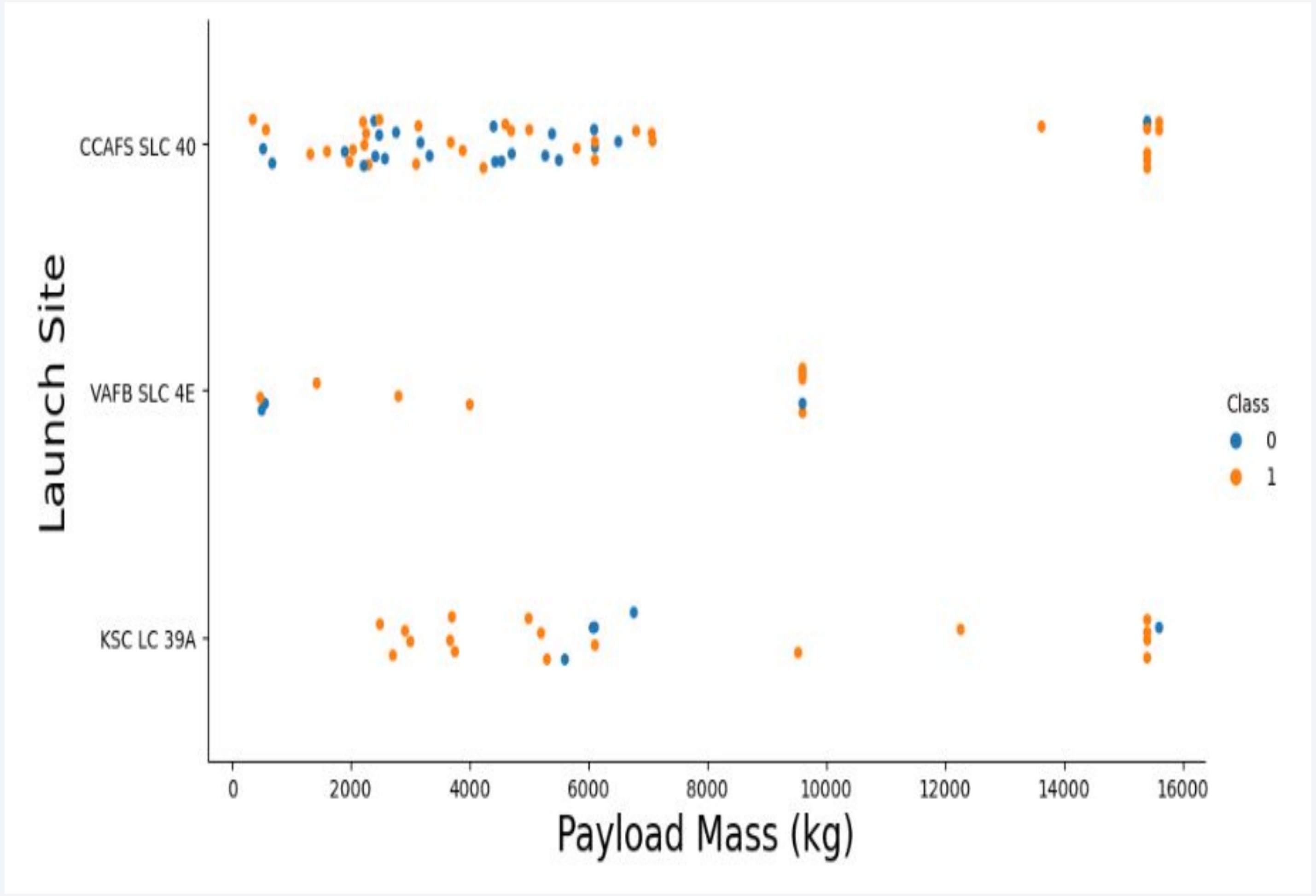
Flight Number vs. Launch Site

1. Early stage flights (flight numbers < 30) launched from CCAFS SSLC40 were unsuccessful.
2. VAFB SLC4E showed the same pattern with early stage flights, but then quickly tended to higher success rates after 30.
3. No early flight stages were conducted at KSC LC 39A, but most flights were successful.
4. After 30 flights, the number of successful launches increased significantly.



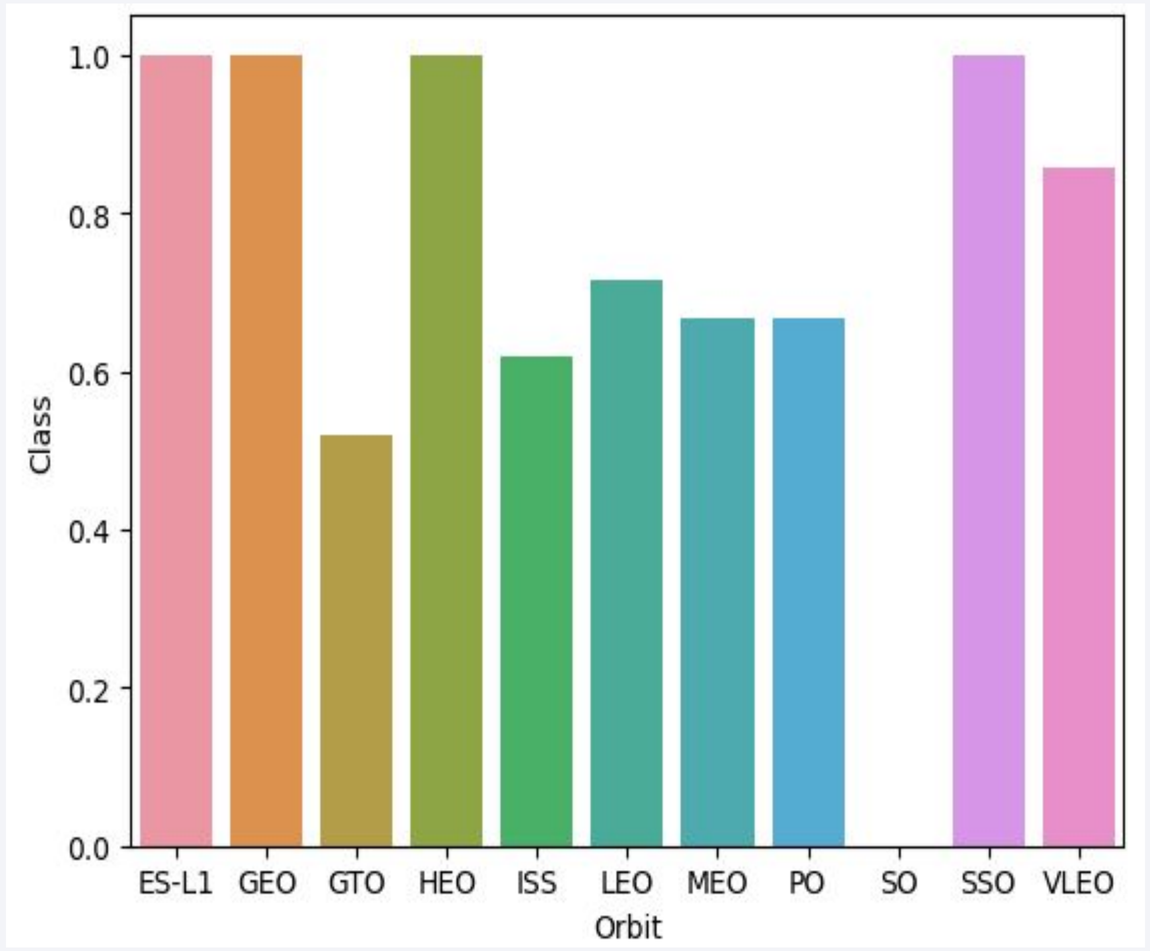
Payload vs. Launch Site

1. When payload was > 7000kg, success rate decreased.
2. No correlation was shown between payload and success rate for each site.
3. most launches were conducted with a payload < 7000kg.



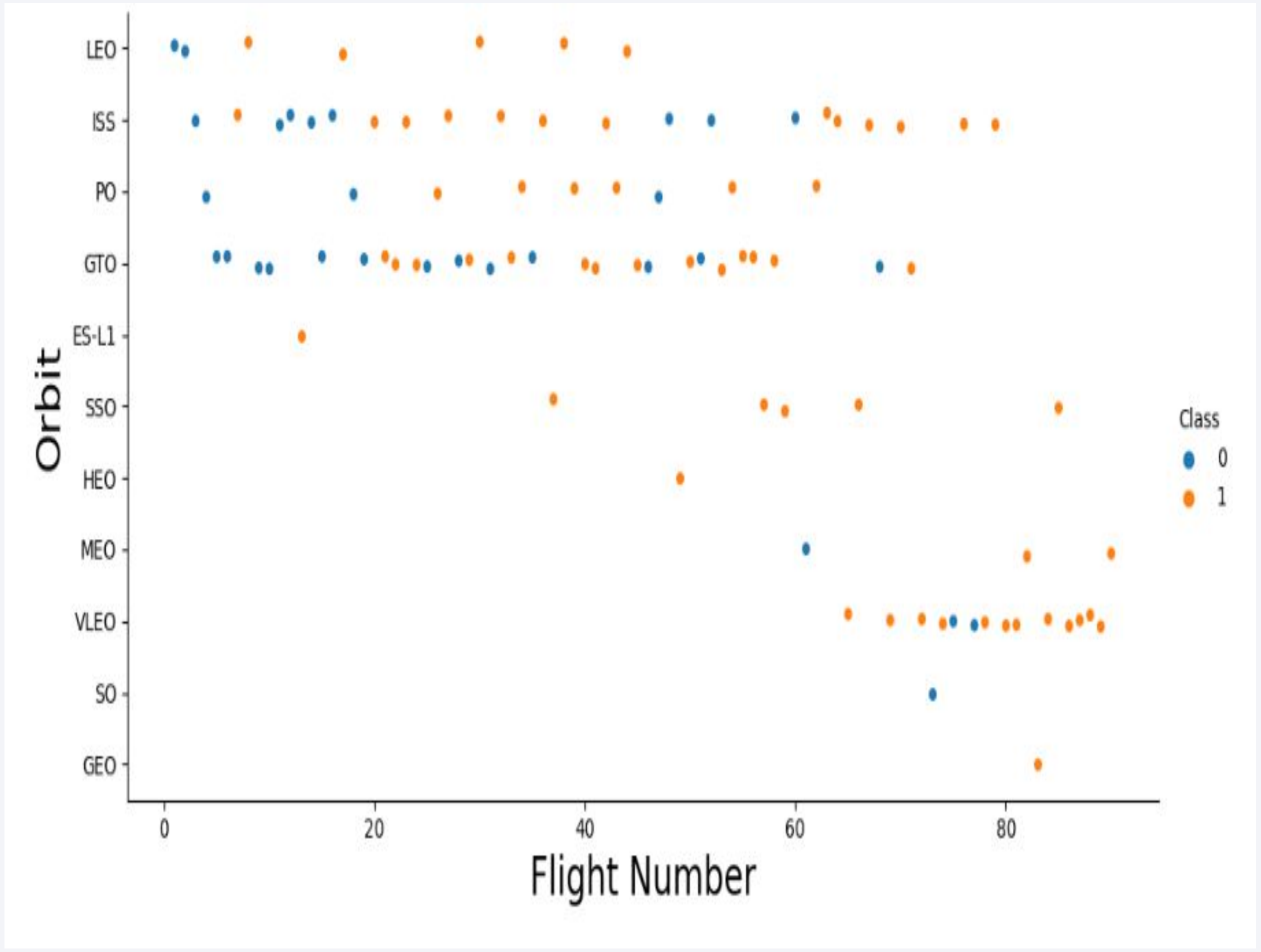
Success Rate vs. Orbit Type

1. 100% success rate shown for Earth-Sun First Lagrangian point (ES-L1), Geostationary Orbit (Geo), High Earth Orbit (HEO), and Sun Synchronous Orbit (SSO).
2. 0% success rate for Heliocentric Orbit (SO).



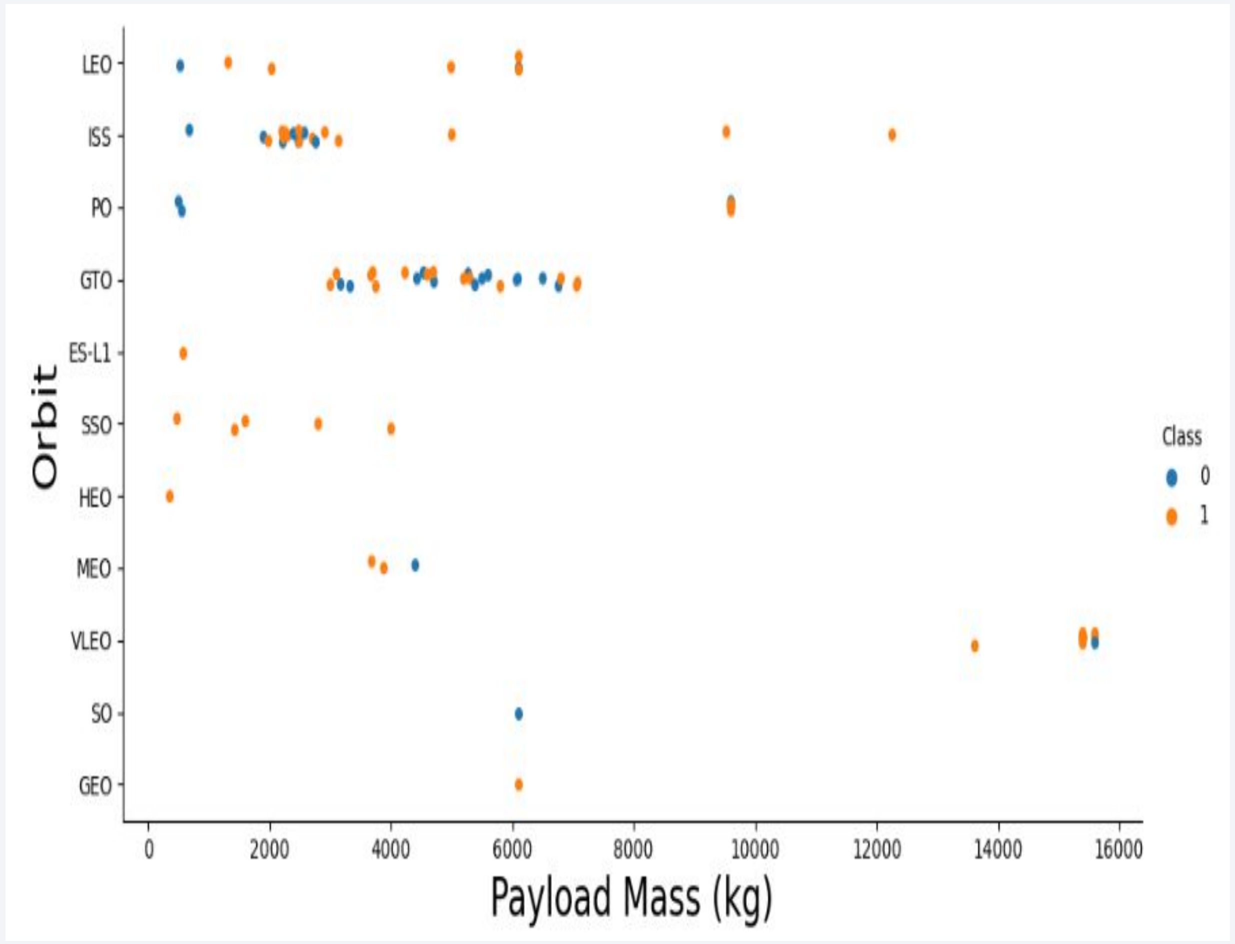
Flight Number vs. Orbit Type

1. GEO, HEO, ES-L1 had only 1 flight each, but had 100% success.
2. SSO had 5 flights with 100% success.
3. There was little to no relationship shown between flight number and success rate for GTO.
4. LEO only failed during low flight numbers.
5. As flight numbers increased, success rates increased (with exception of GTO).



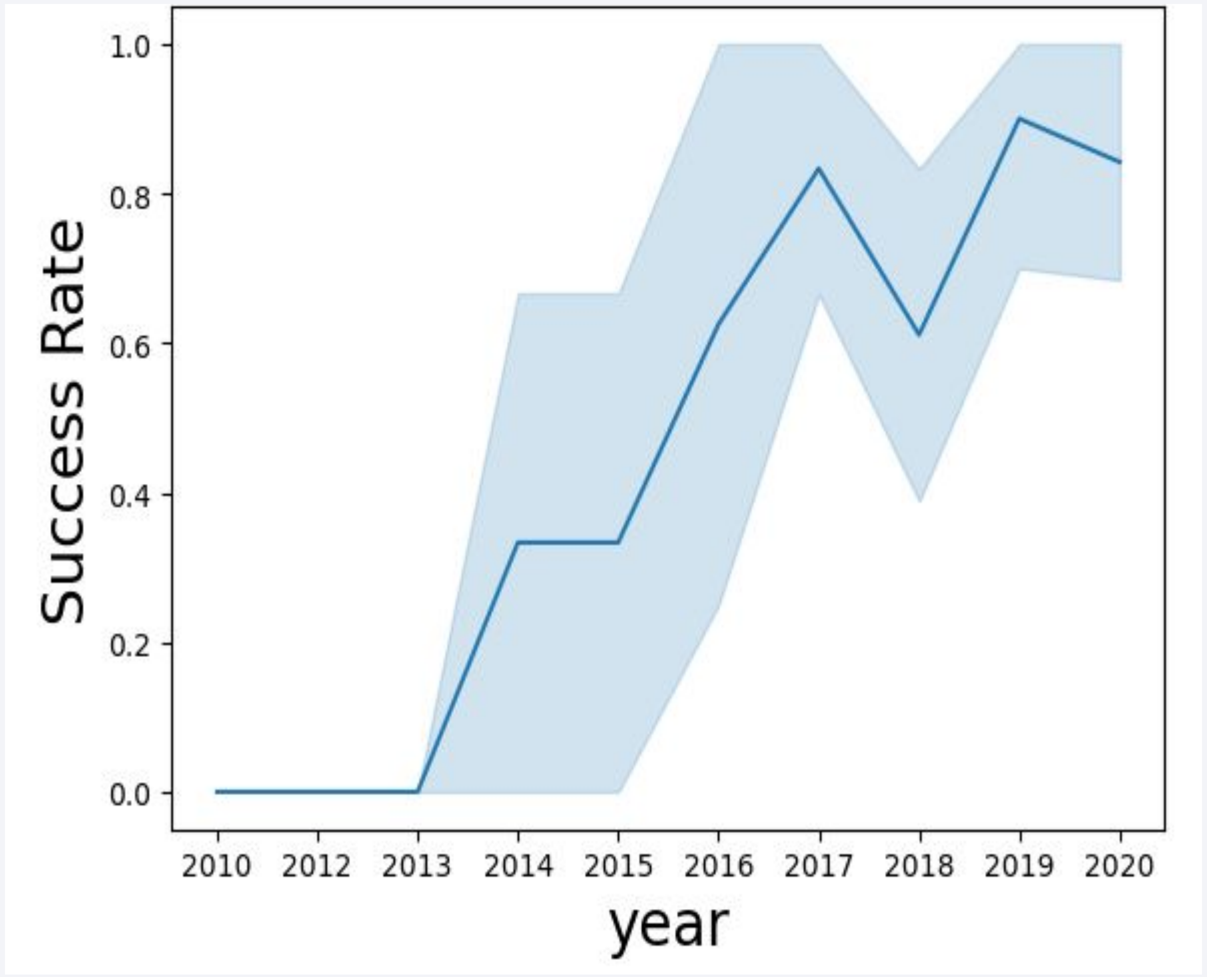
Payload vs. Orbit Type

1. PO,ISS,LEO showed an increase in success rate as payload increased.
2. VLEO was generally associated with higher payloads.
3. There is little to no relationship between payload and success for GTO.



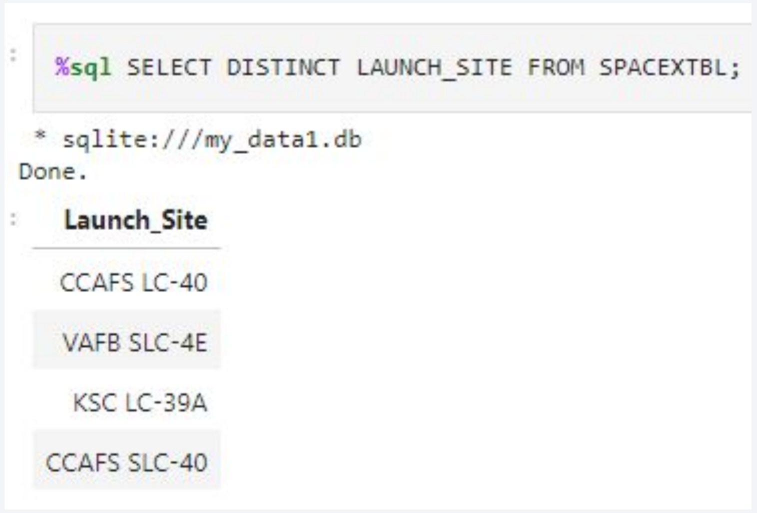
Launch Success Yearly Trend

1. In 2010 - 2013, there was 0 successful launches.
2. After 2013, the success rate was generally increasing.
3. There were small dips in the success rate in 2018 and 2020.
4. After 2016, the success rate was constantly over 50%.



All Launch Site Names

Using the SELECT DISTINCT clause, unique launch site names were found from the launch_site column of the SPACEXTBL table.

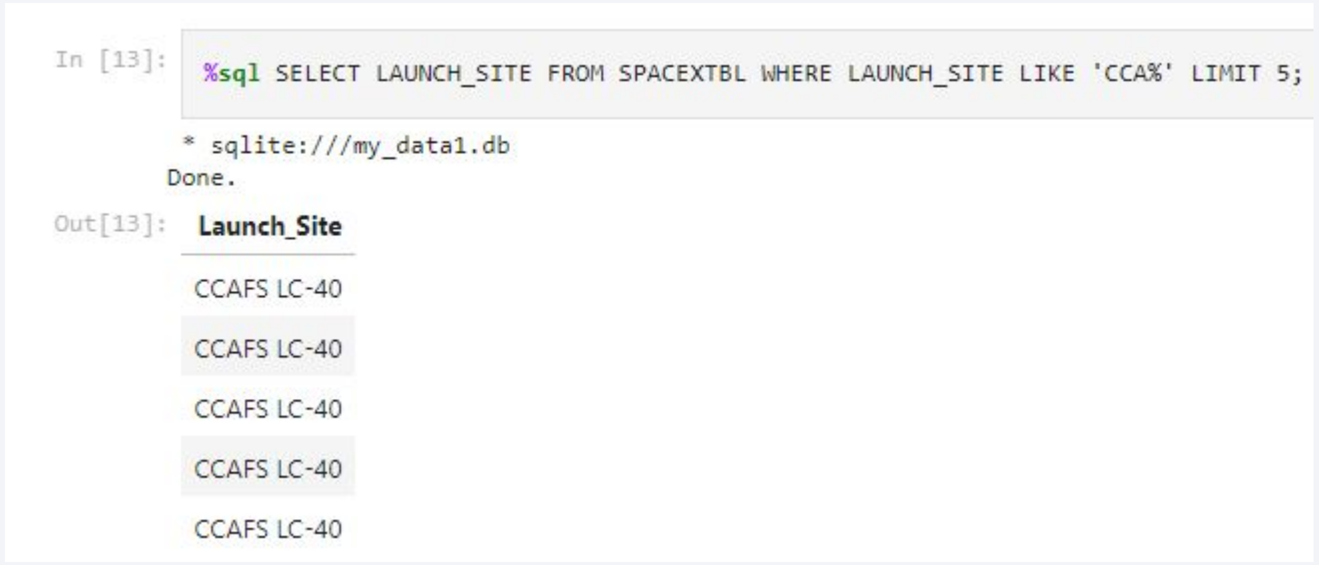


```
%sql SELECT DISTINCT LAUNCH_SITE FROM SPACEXTBL;

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

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

Launch Site Names Begin with 'CCA'

- Using the LIMIT 5 clause, we filtered the data to show only 5 results.
- Using the LIKE clause and the % placeholder symbol, we were able to find results from the data that began with CCA regardless of the ending characters.

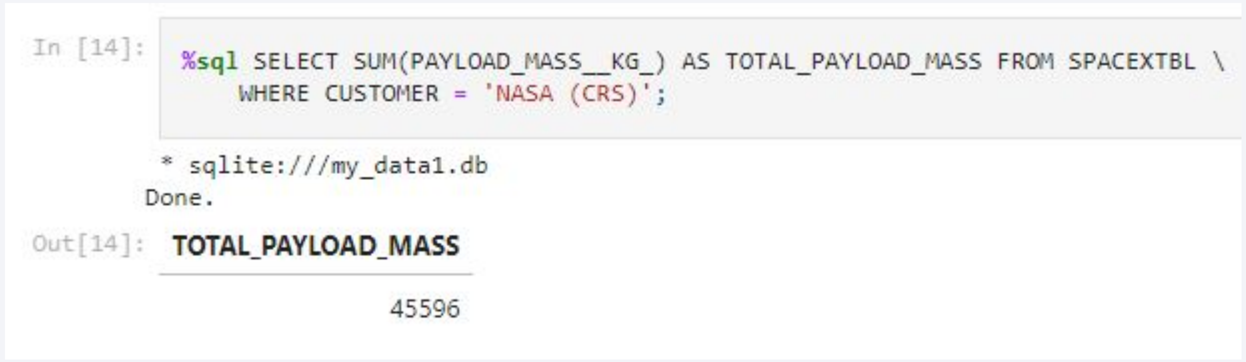


```
In [13]: %sql SELECT LAUNCH_SITE FROM SPACEXTBL WHERE LAUNCH_SITE LIKE 'CCA%' LIMIT 5;

 * sqlite:///my_data1.db
Done.

Out[13]:
```

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

Total Payload Mass

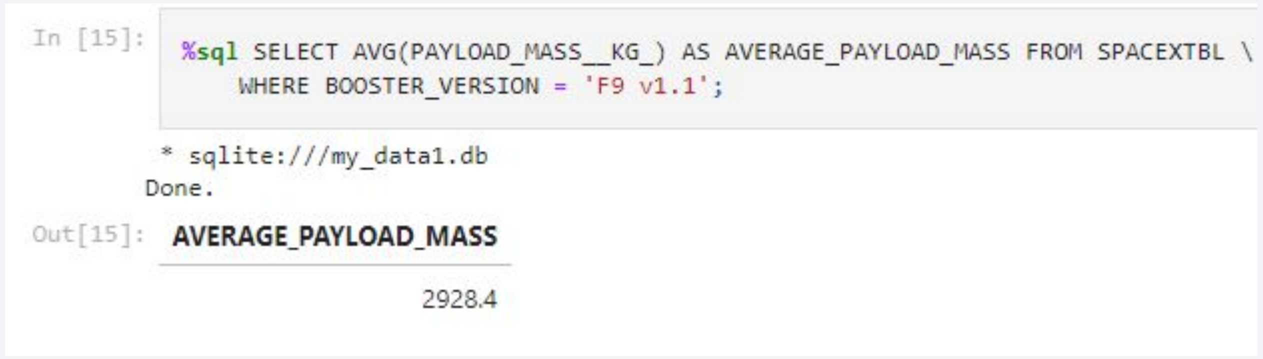
- Using the SUM function, we calculated the sum of payload masses and saved it as tital_payload_mass.
- Using the WHERE clause, we filtered the data to only be calculated if the customer was NASA (CRS).



```
In [14]: %sql SELECT SUM(PAYLOAD_MASS__KG_) AS TOTAL_PAYLOAD_MASS FROM SPACEXTBL \
             WHERE CUSTOMER = 'NASA (CRS)';

          * sqlite:///my_data1.db
         Done.

Out[14]: TOTAL_PAYLOAD_MASS
                      45596
```

Average Payload Mass by F9 v1.1

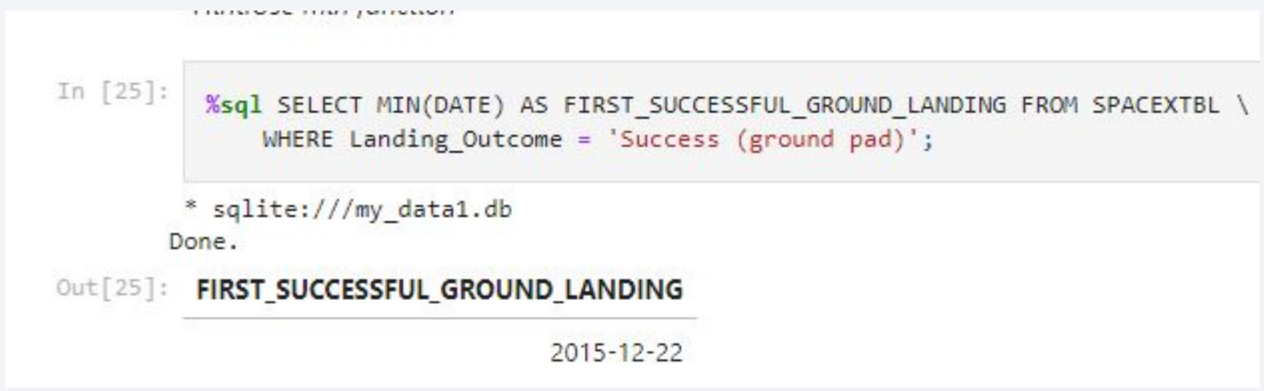
- Using the AVG function, we calculated the average payload mass and saved it in the column average_payload_mass.
- Using the WHERE clause, we filtered the data to only be calculated if the booster version was F9 v1.1.



```
In [15]:  %sql SELECT AVG(PAYLOAD_MASS__KG_) AS AVERAGE_PAYLOAD_MASS FROM SPACEXTBL \
              WHERE BOOSTER_VERSION = 'F9 v1.1';

           * sqlite:///my_data1.db
          Done.
Out[15]:  AVERAGE_PAYLOAD_MASS
                        2928.4
```

First Successful Ground Landing Date

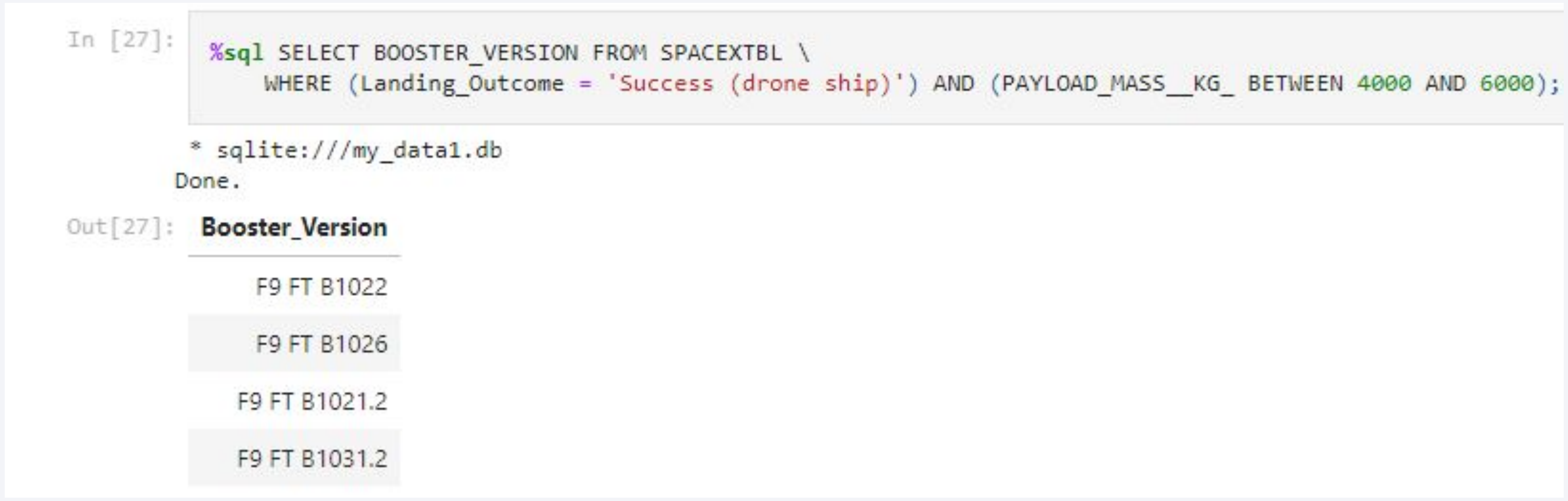
- The MIN function was used to find the first date (minimum date).
- The WHERE clause was used to filter the data to only include the landing outcomes Success (ground pad).



```
In [25]:  %sql SELECT MIN(DATE) AS FIRST_SUCCESSFUL_GROUND_LANDING FROM SPACEXTBL \
              WHERE Landing_Outcome = 'Success (ground pad)';

           * sqlite:///my_data1.db
          Done.

Out[25]:  FIRST_SUCCESSFUL_GROUND_LANDING
                                2015-12-22
```

Successful Drone Ship Landing with Payload between 4000 and 6000

- The booster versions were selected using the SELECT clause.
- the WHERE clause filtered the data to only include results that were success (ground pad) AND had a payload between 4000kg - 6000kg.



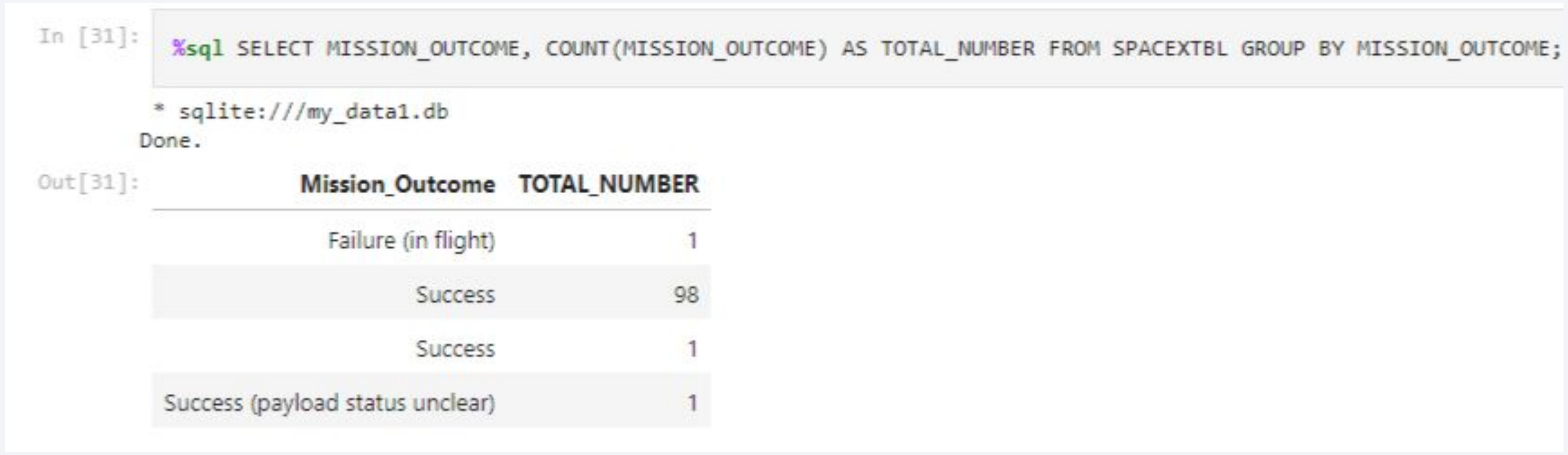
```
In [27]: %sql SELECT BOOSTER_VERSION FROM SPACEXTBL \
             WHERE (Landing_Outcome = 'Success (drone ship)') AND (PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000);

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

Out[27]:

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

Total Number of Successful and Failure Mission Outcomes

- The COUNT function was used to total the number of mission outcomes.
- The GROUP BY clause was used to sort the counted data by mission outcomes



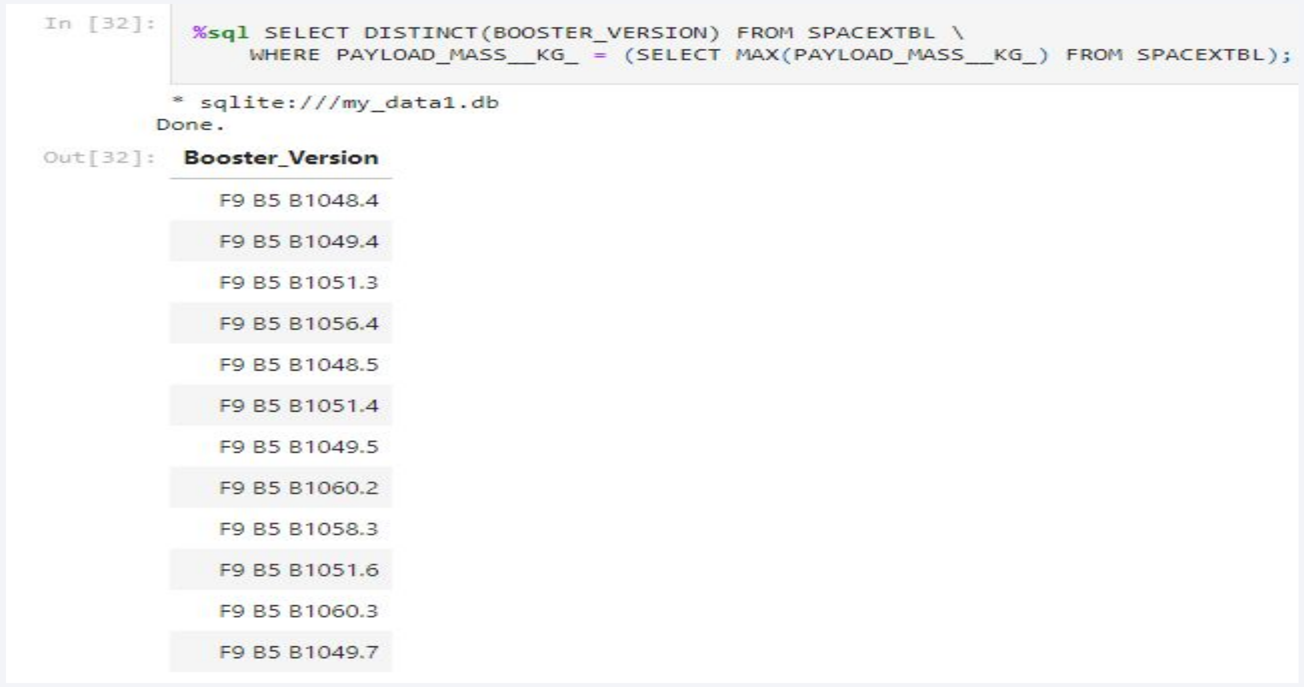
```
In [31]: %sql SELECT MISSION_OUTCOME, COUNT(MISSION_OUTCOME) AS TOTAL_NUMBER FROM SPACEXTBL GROUP BY MISSION_OUTCOME;

 * sqlite:///my_data1.db
Done.
Out[31]:
```

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

Boosters Carried Maximum Payload

- Boosters that carried the max payload:

F9 B5 B1048.4, F9 B5 B1048.5, F9 B5 B1049.4, F9 B5 B1049.5, F9 B5 B1049.7, F9 B5 B1051.3, F9 B5 B1051.4, F9 B5 B1051.6, F9 B5 B1056.4, F9 B5 B1058.3, F9 B5 B1060.2, F9, B5 B1060.3.



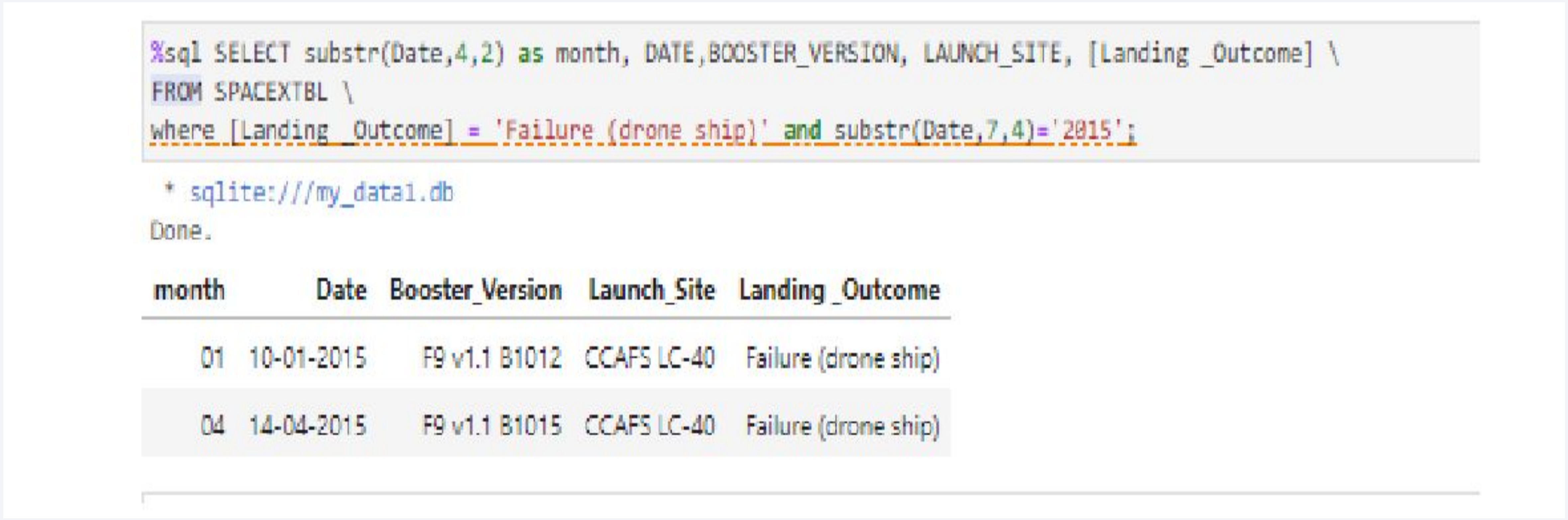
```
In [32]:  %sql SELECT DISTINCT(BOOSTER_VERSION) FROM SPACEXTBL \
              WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTBL);

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

Out[32]:

Booster_Version
F9 B5 B1048.4
F9 B5 B1049.4
F9 B5 B1051.3
F9 B5 B1056.4
F9 B5 B1048.5
F9 B5 B1051.4
F9 B5 B1049.5
F9 B5 B1060.2
F9 B5 B1058.3
F9 B5 B1051.6
F9 B5 B1060.3
F9 B5 B1049.7

2015 Launch Records

List the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015



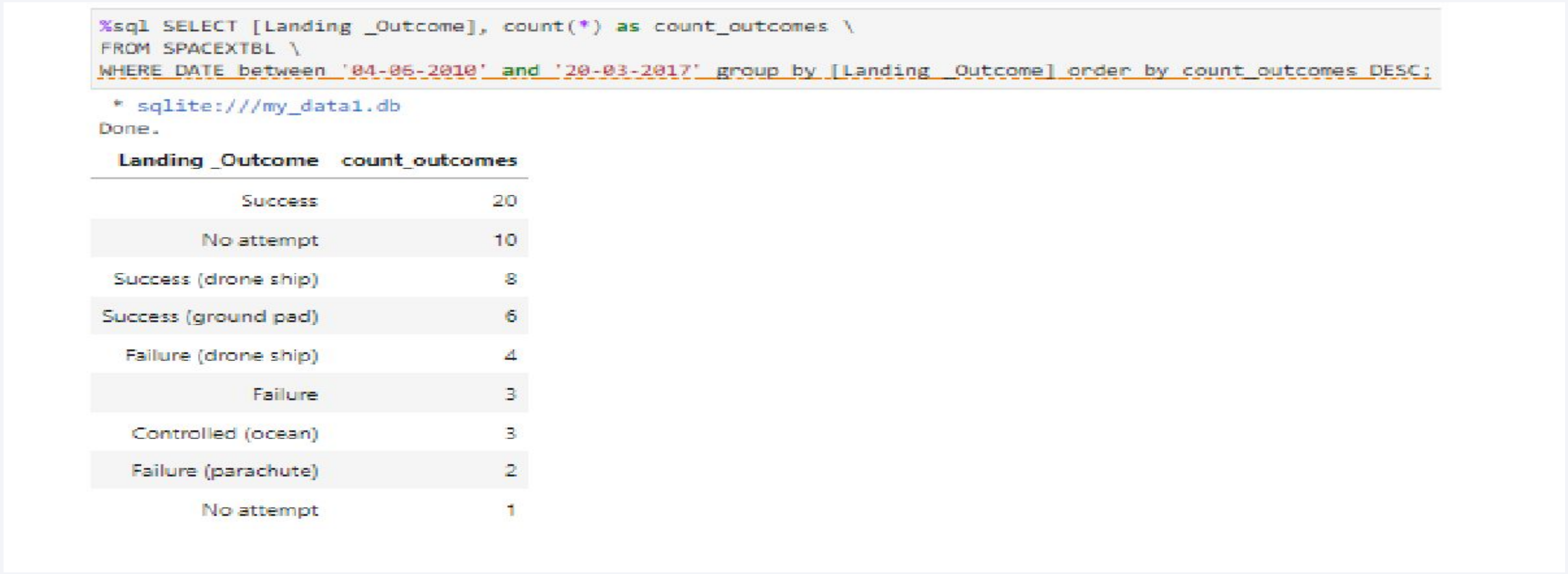
```
%sql SELECT substr(Date,4,2) as month, DATE,BOOSTER_VERSION, LAUNCH_SITE, [Landing _Outcome] \
FROM SPACEXTBL \
where [Landing _Outcome] = 'Failure (drone ship)' and substr(Date,7,4)='2015';
```

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

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

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

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



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

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

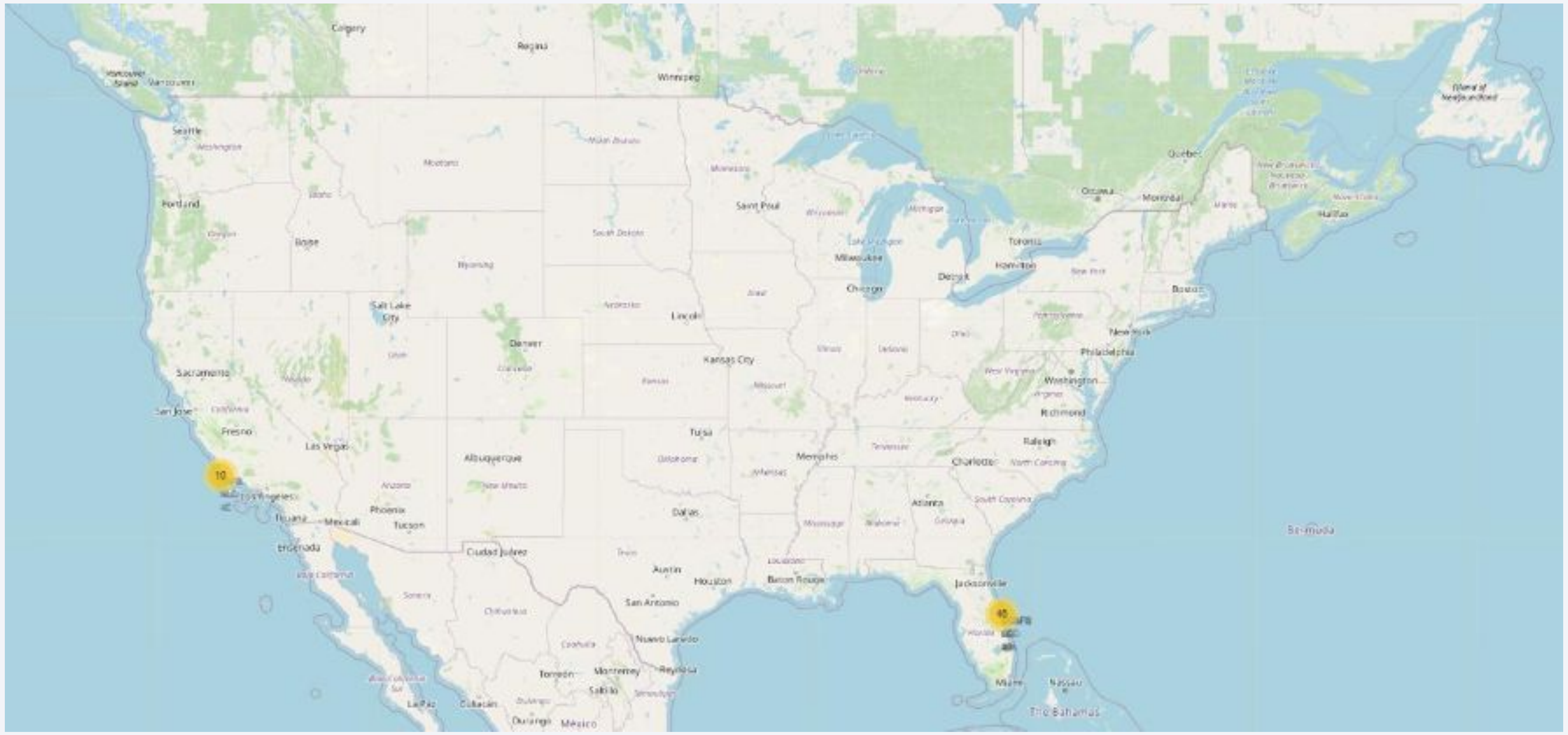
Landing _Outcome	count_outcomes
Success	20
No attempt	10
Success (drone ship)	8
Success (ground pad)	6
Failure (drone ship)	4
Failure	3
Controlled (ocean)	3
Failure (parachute)	2
No attempt	1

Section 3

Launch Sites Proximities Analysis

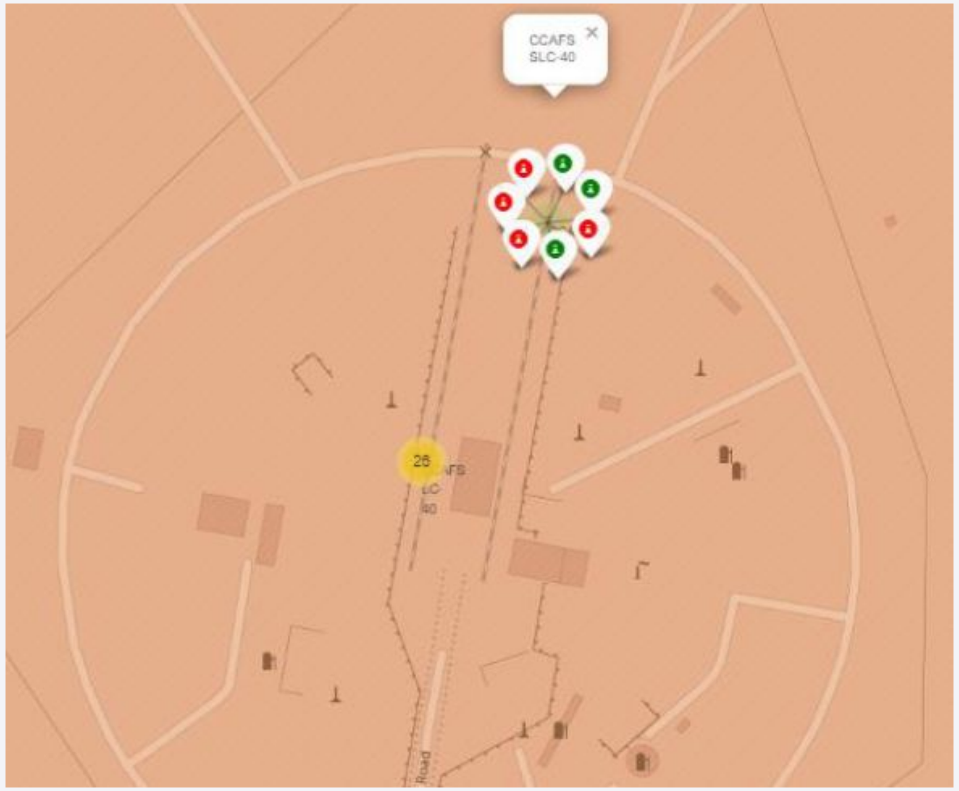
All Launch Sites on a Map

All launch sites are located on the coast of the U.S. specifically around Florida and California.



Failed and Successful launches at each site

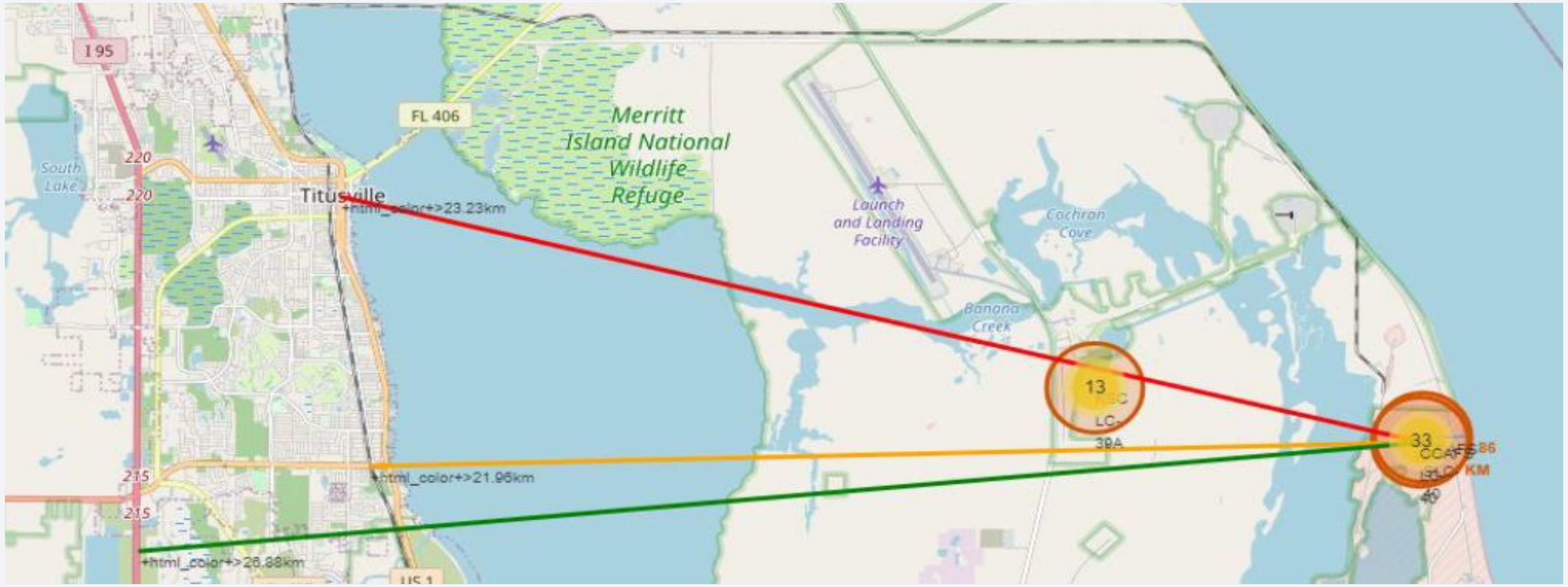
Launches were grouped into clusters at each sight and colored red for failed launches and green for successful launches accordingly.

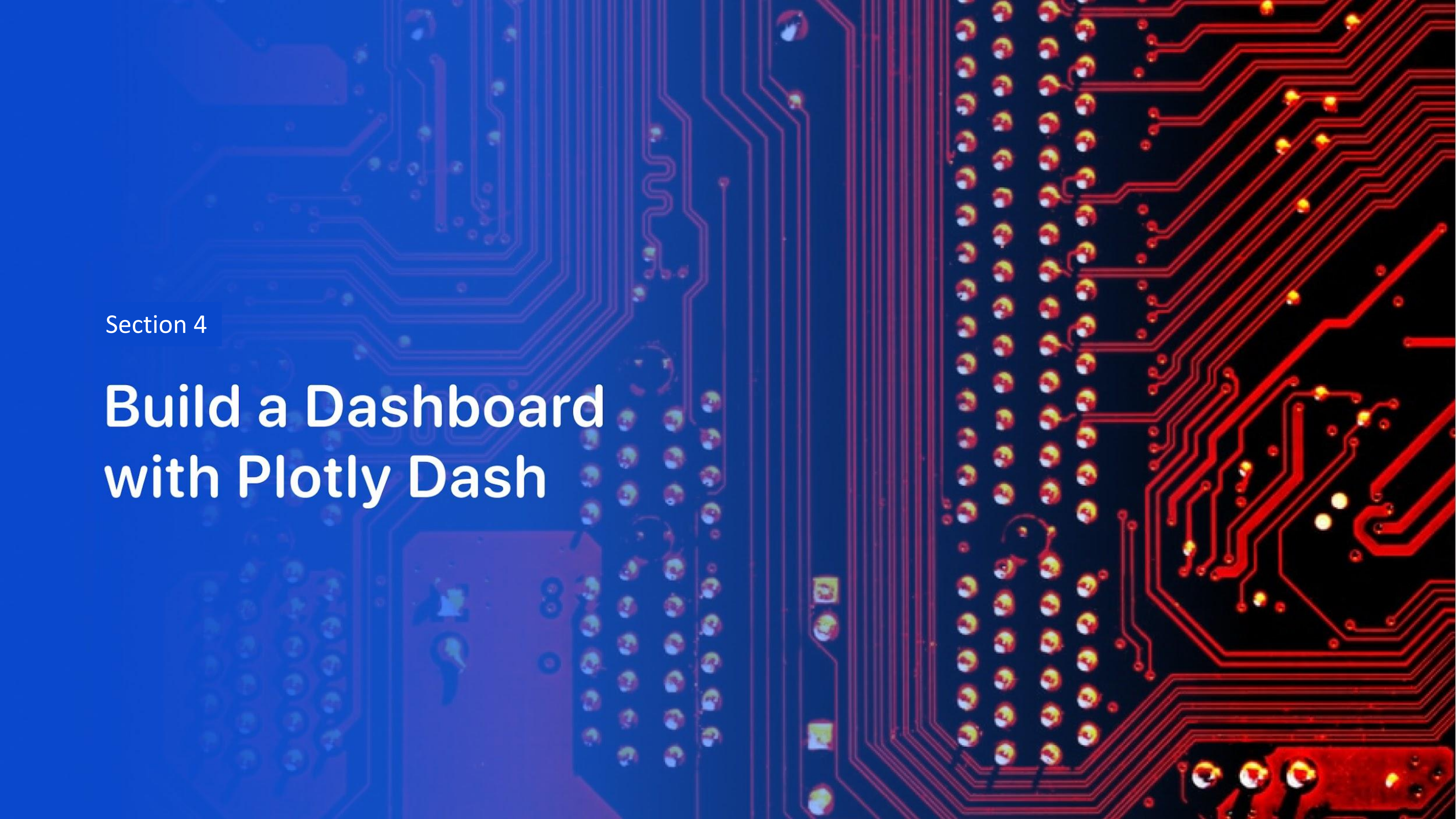


Launch Site Proximity to Other Interest Points

CCAFS SLC-40 launch site:

- Coastline is 0.87kn due east
- highway is 0.59km away
- railway is 1.29km away
- nearest city is 51.74km away.



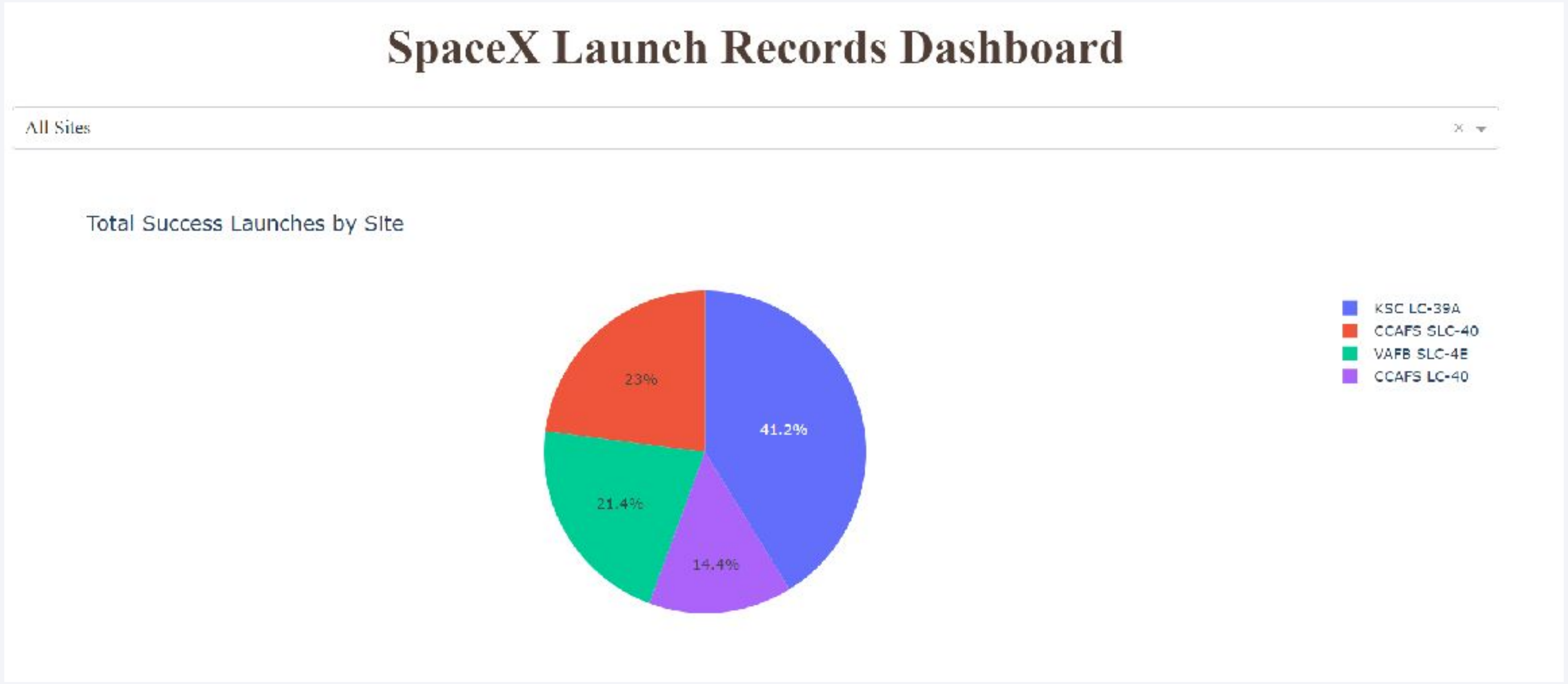


Section 4

Build a Dashboard with Plotly Dash

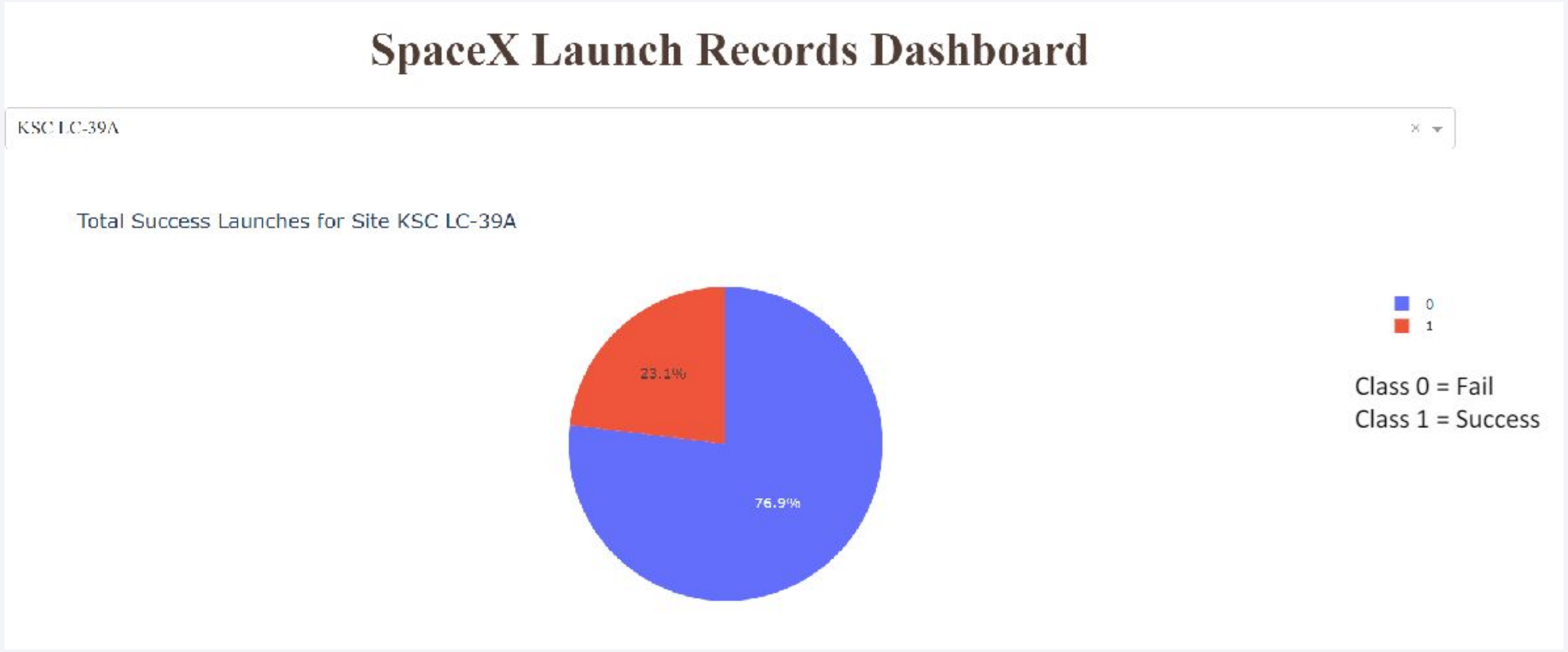
Launch Success Rate for Each Site

KSC LC-39A has the highest success rate with 41.7% of all successful launches.



Breakdown of Launches at KSC LC-39A

76.9% of launches at KSC LC-39A were successful whereas 23.1% failed giving KSC LC-39A the highest success rate.


SpaceX Launch Records Dashboard

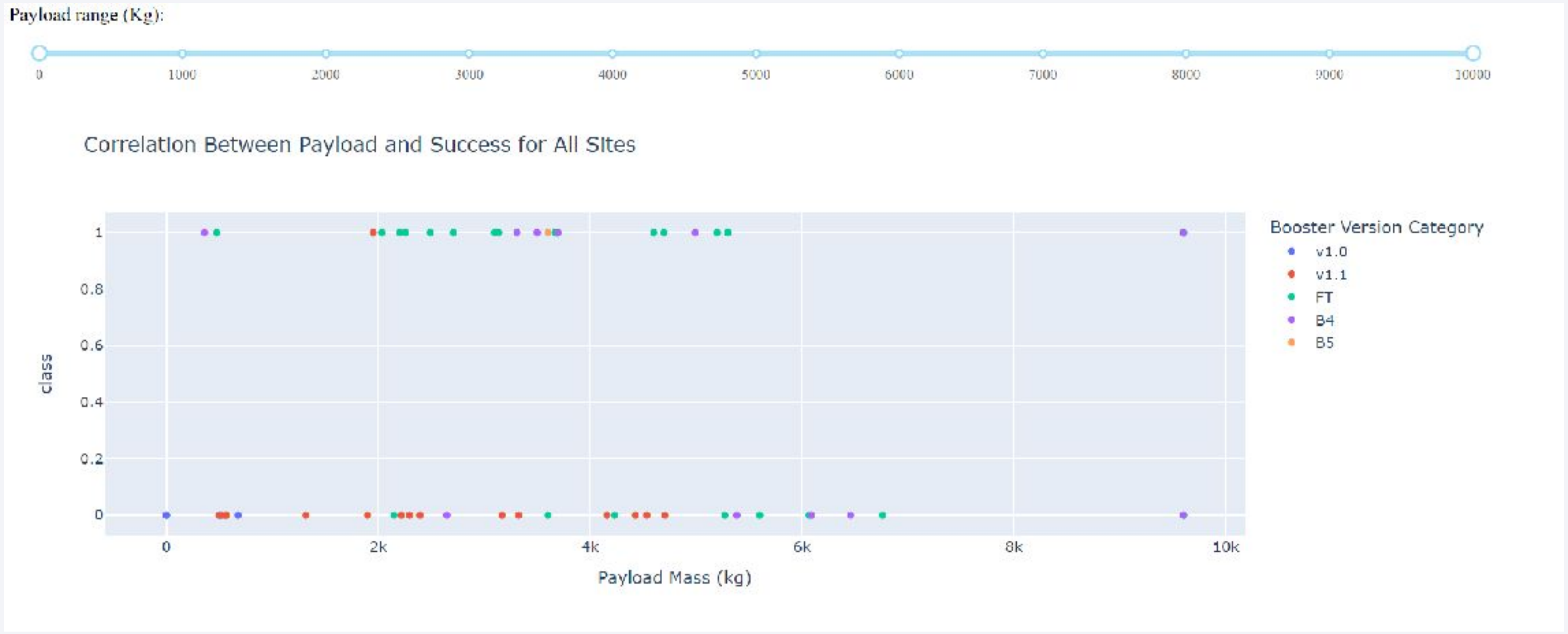
KSC LC-39A

Total Success Launches for Site KSC LC-39A

Class 0 = Fail
Class 1 = Success

Launch Outcome and Payload for All Sites

Lower payloads had greater success than heavier payloads.
Some boosters were not launches with heavier payloads.

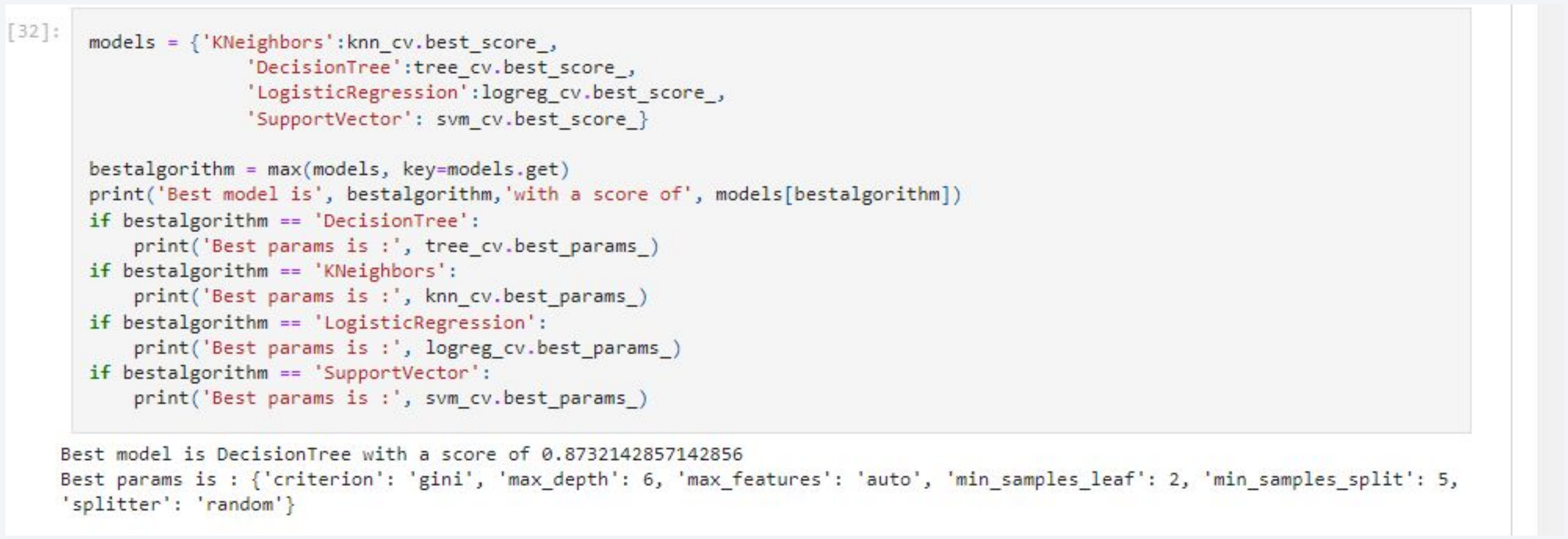


Section 5

Predictive Analysis (Classification)

Classification Accuracy

The decision tree method was the best classification model with
an 87.32% accuracy score.

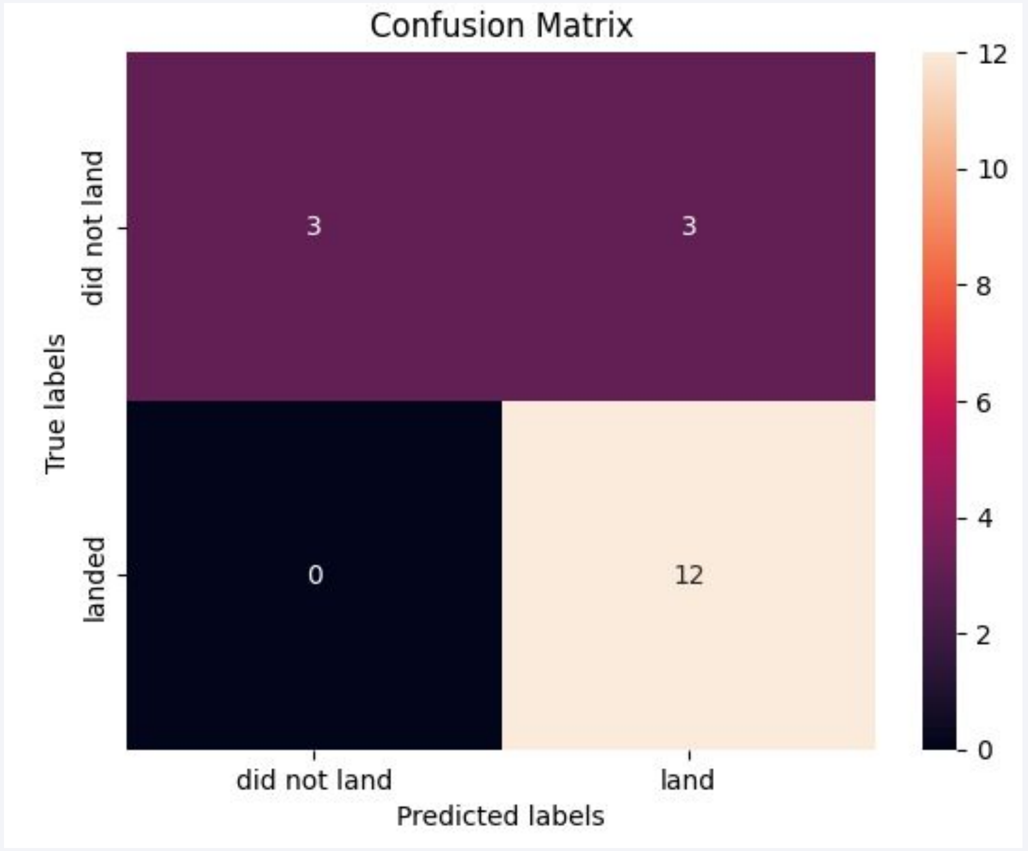


```
[32]:  models = {'KNeighbors':knn_cv.best_score_,
                     'DecisionTree':tree_cv.best_score_,
                     'LogisticRegression':logreg_cv.best_score_,
                     'SupportVector': svm_cv.best_score_}

       bestalgorithm = max(models, key=models.get)
       print('Best model is', bestalgorithm,'with a score of', models[bestalgorithm])
       if bestalgorithm == 'DecisionTree':
           print('Best params is :', tree_cv.best_params_)
       if bestalgorithm == 'KNeighbors':
           print('Best params is :', knn_cv.best_params_)
       if bestalgorithm == 'LogisticRegression':
           print('Best params is :', logreg_cv.best_params_)
       if bestalgorithm == 'SupportVector':
           print('Best params is :', svm_cv.best_params_)
```

```
Best model is DecisionTree with a score of 0.8732142857142856
Best params is : {'criterion': 'gini', 'max_depth': 6, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_samples_split': 5,
'splitter': 'random'}
```

Confusion Matrix

The confusion matrix shows 15 correctly identified outcomes and 3 incorrectly identified outcome resulting in a 83.33% accuracy.

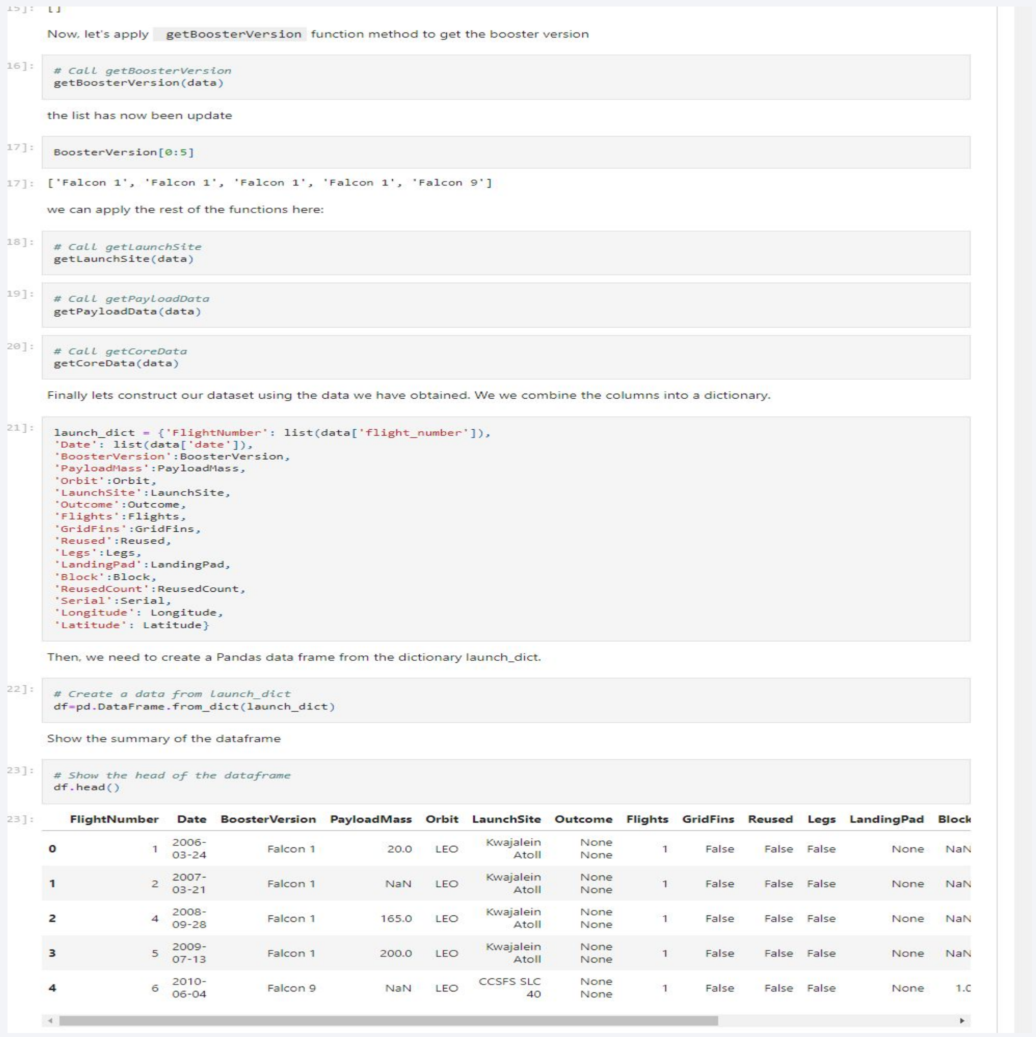


Conclusions

- As the number of flights increase, the success rates increased.
- 2010 - 2013: 0% success rate, after 2013: generally increasing success rate, after 2016: over 50% success rate.
- Orbit types ES-L1, GEO, HEO, SSO had the highest success rate. SSO had the most successful launches.
- Orbit types PO, ISS, LEO, VLEO had the most successful launches with heavy payloads.
- KSC LC-39A was the most successful launch site.
- Lower payloads had a higher success rate than heavy payloads, but also had more launches than higher payloads.

Appendix- Space X REST API & Web Scraping

REST API



```
15]: []
```

Now, let's apply `getBoosterVersion` function method to get the booster version

```
16]: # Call getBoosterVersion
     getBoosterVersion(data)
```

the list has now been update

```
17]: BoosterVersion[0:5]

17]: ['Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 9']
```

we can apply the rest of the functions here:

```
18]: # Call getLaunchSite
     getLaunchSite(data)

19]: # Call getPayloadData
     getPayloadData(data)

20]: # Call getCoreData
     getCoreData(data)
```

Finally lets construct our dataset using the data we have obtained. We we combine the columns into a dictionary.

```
21]: launch_dict = {'FlightNumber': list(data['flight_number']),
     'Date': list(data['date']),
     'BoosterVersion':BoosterVersion,
     'PayloadMass':PayloadMass,
     'Orbit':Orbit,
     'LaunchSite':LaunchSite,
     'Outcome':Outcome,
     'Flights':Flights,
     'GridFins':GridFins,
     'Reused':Reused,
     'Legs':Legs,
     'LandingPad':LandingPad,
     'Block':Block,
     'ReusedCount':ReusedCount,
     'Serial':Serial,
     'Longitude': Longitude,
     'Latitude': Latitude}
```

Then, we need to create a Pandas data frame from the dictionary launch_dict.

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

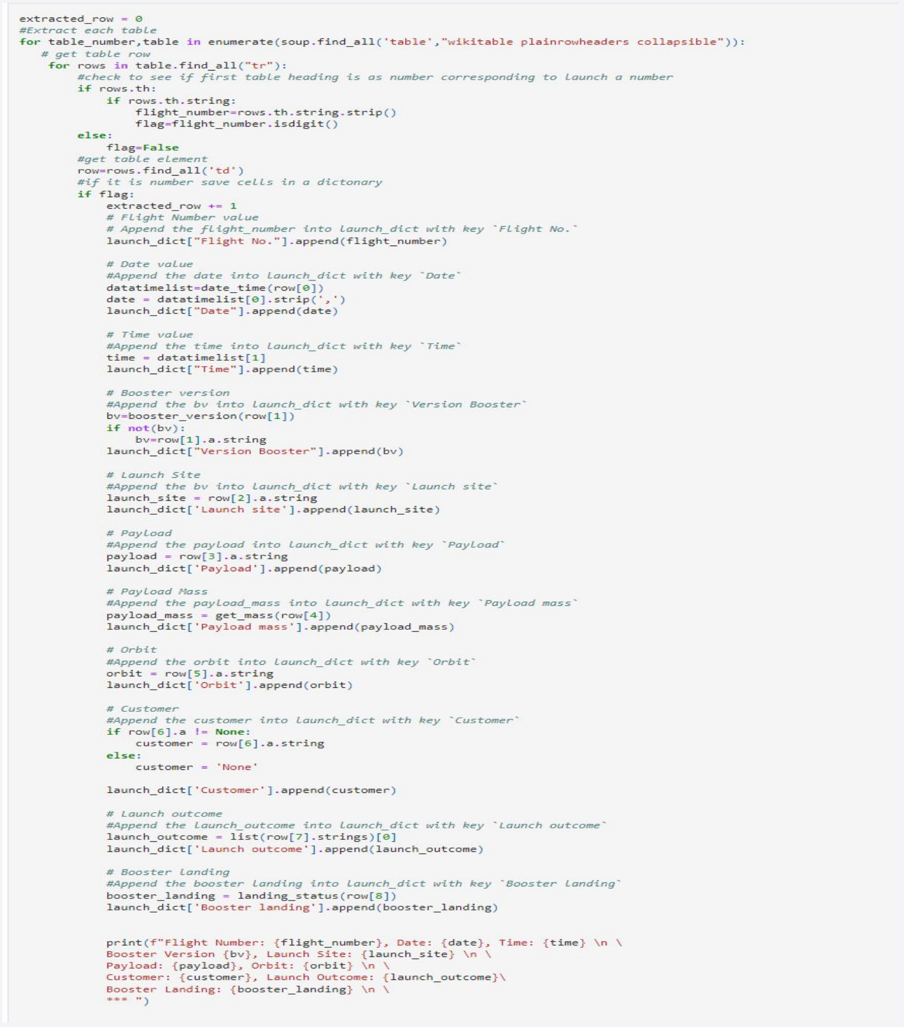
Show the summary of the dataframe

```
23]: # Show the head of the dataframe
     df.head()
```

23]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block
0	1	2006-03-24	Falcon 1	20.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN
1	2	2007-03-21	Falcon 1	NaN	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN
2	4	2008-09-28	Falcon 1	165.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN
3	5	2009-07-13	Falcon 1	200.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN
4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.

Web Scraping



```
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders collapsible")):
   # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number corresponding to launch a number
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
        else:
            flag=False
        #get table element
        row=rows.find_all('td')
        #if it is number save cells in a dictonary
        if flag:
            extracted_row += 1
            # Flight Number value
            # Append the flight_number into launch_dict with key `Flight No.`
            launch_dict["Flight No."].append(flight_number)

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

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

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

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

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

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

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

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

            launch_dict['Customer'].append(customer)

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

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


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

Thank you!