IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Gene Henry Gueco
Sep. 24, 2023

Outline

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

Executive Summary

- Summary of methodologies
  - Data Collection through API and Web Scraping
  - Data Wrangling
  - Exploratory Analysis with SQL and Data Visualization
  - Interactive Visual Analytics
  - Machine Learning
- Summary of all results
  - Exploratory results
  - Screenshots for Interactive Visual Analytics
  - Prediction results from Machine Learning

Introduction

- Project background and context

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

- Problems you want to find answers
  - Factors to determine if the rocket will land successfully.
  - Landing Success Rates
  - Proper predictions for conditions to maximize the success rates of landing outcomes.

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - SpaceX API and web scraping.
- Perform data wrangling
  - Exploratory data analysis and determining training labels
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - How to build, tune, evaluate classification models

Data Collection

- Data collection methods:
  - Using get request to the SpaceX API.
  - Decoding the response as Json using .json() function call and turn it into a pandas dataframe using .json_normalize().
  - Data Cleaning with identifying missing values and populating them if necessary.
  - Web scraping using BeautifulSoup.
  - Extract records as HTML and parse it to Pandas dataframe.

Data Collection – SpaceX API

- Using the get request to the SpaceX API to collect data, clean and did some basic data wrangling and formatting.

- Github URL:

  https://github.com/GeneGueco/DS-Coursera-Capstone/blob/main/Data%20Collection%20API.ipynb

1. Use get request

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

```
response = requests.get(spacex_url)
```

2. Use json_normalize

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

3. Clean and fill Null values

```
# Calculate the mean value of PayloadMass column
mean = data_falcon9['PayloadMass'].mean()
# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'] = data_falcon9['PayloadMass'].fillna(mean)
```

Data Collection - Scraping

- Web scraping Falcon 9 launches with BeautifulSoup and parsed it convert to pandas dataframe.
- Github URL:

  https://github.com/GeneGueco/DS-Coursera-Capstone/blob/main/Data%20Collection%20with%20Web%20Scraping.ipynb

1. Request the HTML page for URL and get a response object

```
static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"
```

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

2. Create a BeautifulSoup object from the HTML response

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

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

3. Extract column names

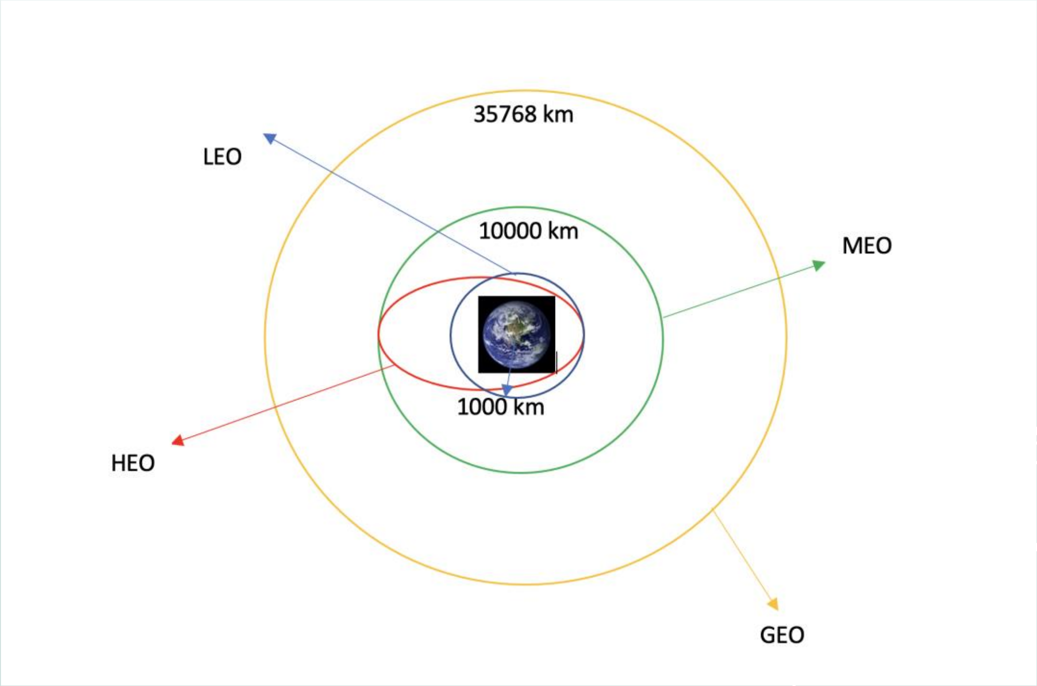
```
column_names = []

# Apply find_all() function with `th` element on first_launch_table
# Iterate each th element and apply the provided extract_column_from_header() to get a column name
# Append the Non-empty column name (`if name is not None and len(name) > 0`) into a list called column_names

element = soup.find_all('th')
for row in range(len(element)):
    try:
        name = extract_column_from_header(element[row])
        if (name is not None and len(name) > 0):
            column_names.append(name)
    except:
        pass
```

4. Create dataframe

Data Wrangling

- Performed exploratory Data Analysis and determined Training Labels.
- Calculated the number of launches on each site.
- Calculated the number and occurrence of each orbit.
- Calculate the number and occurrence of mission outcome per orbit type.
- Created a landing outcome label
- Github URL:

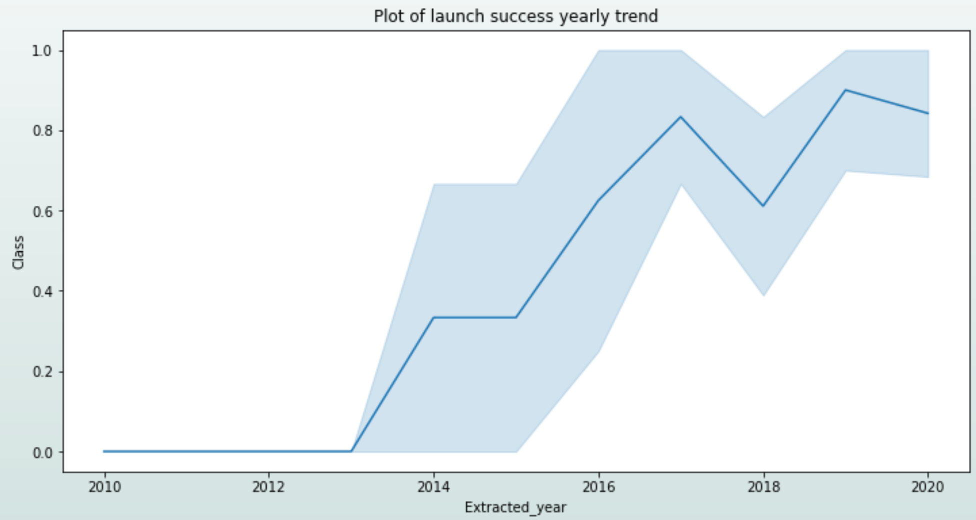
  https://github.com/GeneGueco/DS-Coursera-Capstone/blob/main/Data%20Wrangling.ipynb



EDA with Data Visualization

- We explored the data by visualizing the relationship between flight number and launch Site, payload and launch site, success rate of each orbit type, flight number and orbit type, the launch success yearly trend.
- Github URL:

  https://github.com/GeneGueco/DS-Coursera-Capstone/blob/main/EDA%20with%20Data%20Visualization.ipynb



EDA with SQL

- We loaded the SpaceX dataset into a PostgreSQL database without leaving the jupyter notebook.
- We applied EDA with SQL to get insight from the data. We wrote queries to find out for instance:
  - Unique launch sites in the space mission
  - Total payload mass carried by boosters launched by NASA (CRS)
  - Average payload mass carried by booster version F9 v1.1
  - The total number of successful and failure mission outcomes
  - The failed landing outcomes in drone ship, their booster version and launch site names.
  - Rank the landing outcomes:

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(LANDING_OUTCOME) as COUNT FROM SPACEXTBL WHERE "DATE" BETWEEN '2010-06-04' and '2017-03-20' GROUP BY LANDING_OUTCOME  ORDER BY COUNT(LANDING_OUTCOME) DESC
```

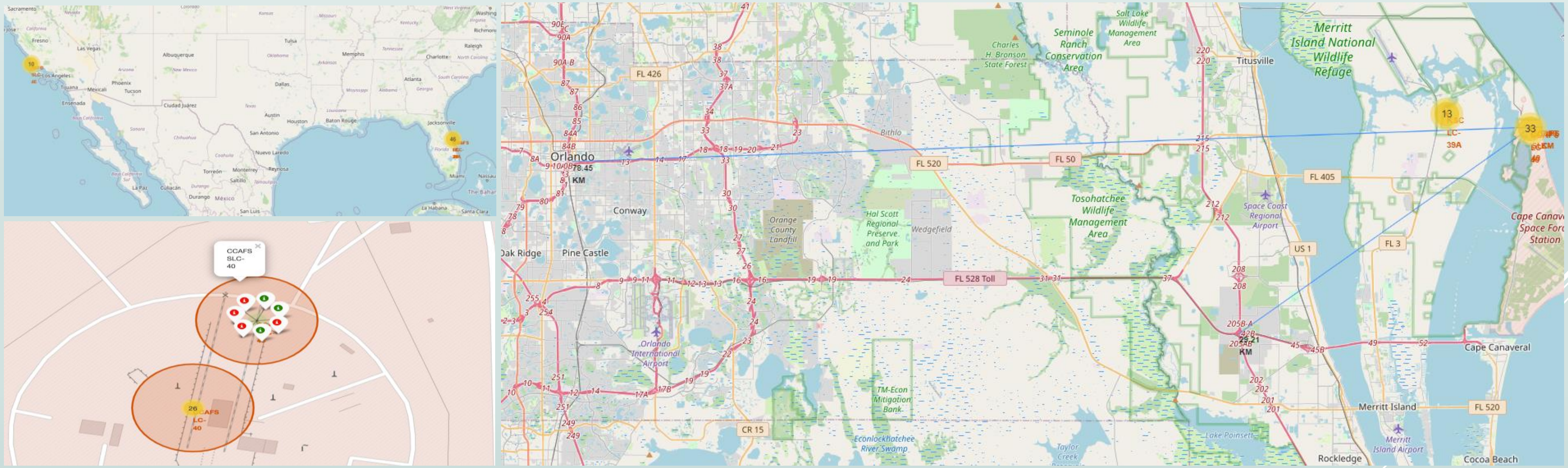
```
 * sqlite:///my_data1.db
Done.
```

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

- Github URL:
  - https://github.com/GeneGueco/DS-Coursera-Capstone/blob/main/EDA%20with%20SQL%20lab.ipynb

Build an Interactive Map with Folium

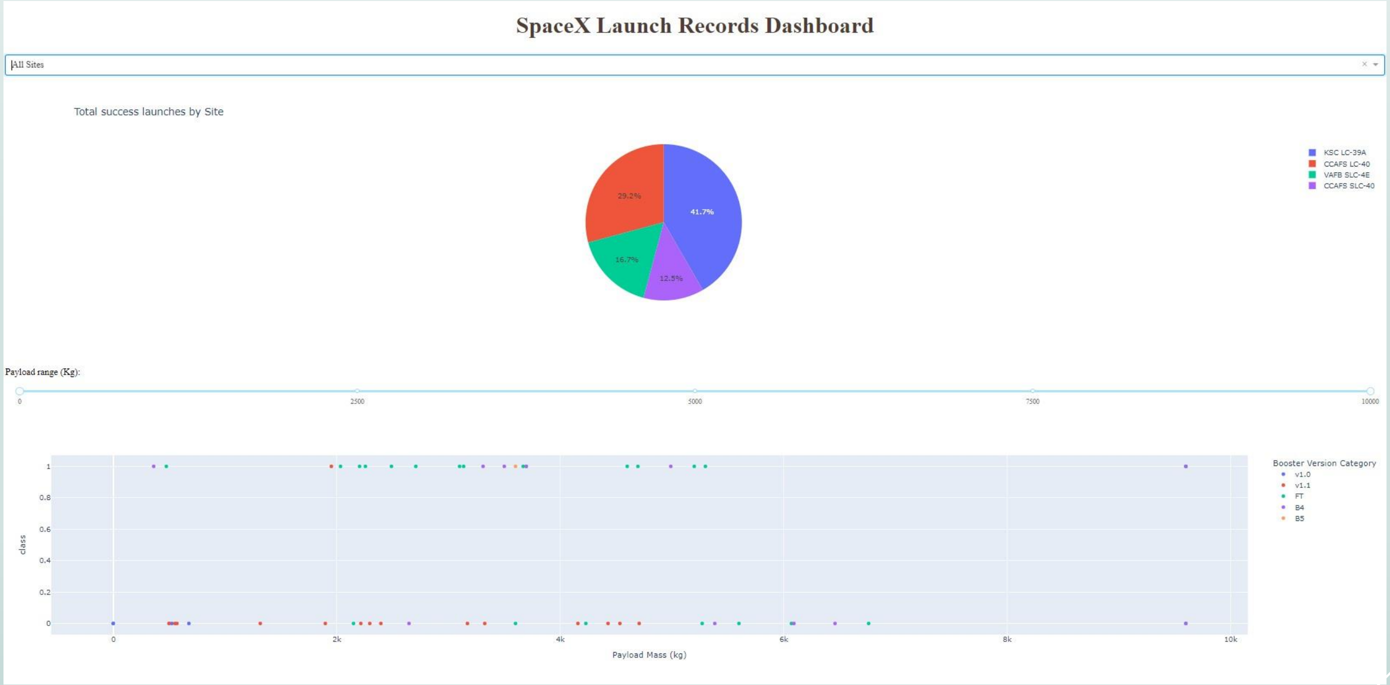
- Marked all launch sites, and added map objects such as markers, circles, lines to mark the success or failure of launches for each site on the folium map.
- Assigned the feature launch outcomes (failure or success) to class 0 and 1.i.e., 0 for failure, and 1 for success.
- Using the color-labeled marker clusters, we identified which launch sites have relatively high success rate.
- Calculated the distances between a launch site to its proximities. We answered some question for instance:
  - *Are launch sites near railways, highways and coastlines.*
  - *Do launch sites keep certain distance away from cities.*



- Github URL: https://github.com/GeneGueco/DS-Coursera-Capstone/blob/main/Interactive%20Visual%20Analytics%20with%20Folium.ipynb

Build a Dashboard with Plotly Dash

- Built an interactive dashboard with Plotly dash
- Plotted pie charts showing the total launches by a certain sites
- Plotted scatter graph showing the relationship with Outcome and Payload Mass (Kg) for the different booster version.



- Github URL: https://github.com/GeneGueco/DS-Coursera-Capstone/blob/main/spacex_dash_app_gene.py

Predictive Analysis (Classification)

- Loaded the data using numpy and pandas, transformed the data, split our data into training and testing.
- Built different machine learning models and tune different hyperparameters using GridSearchCV.
- Used accuracy as the metric for our model, improved the model using feature engineering and algorithm tuning.
- Found the best performing classification model.
- Github URL:

  https://github.com/GeneGueco/DS-Coursera-Capstone/blob/main/Machine%20Learning%20Prediction.ipynb

Results

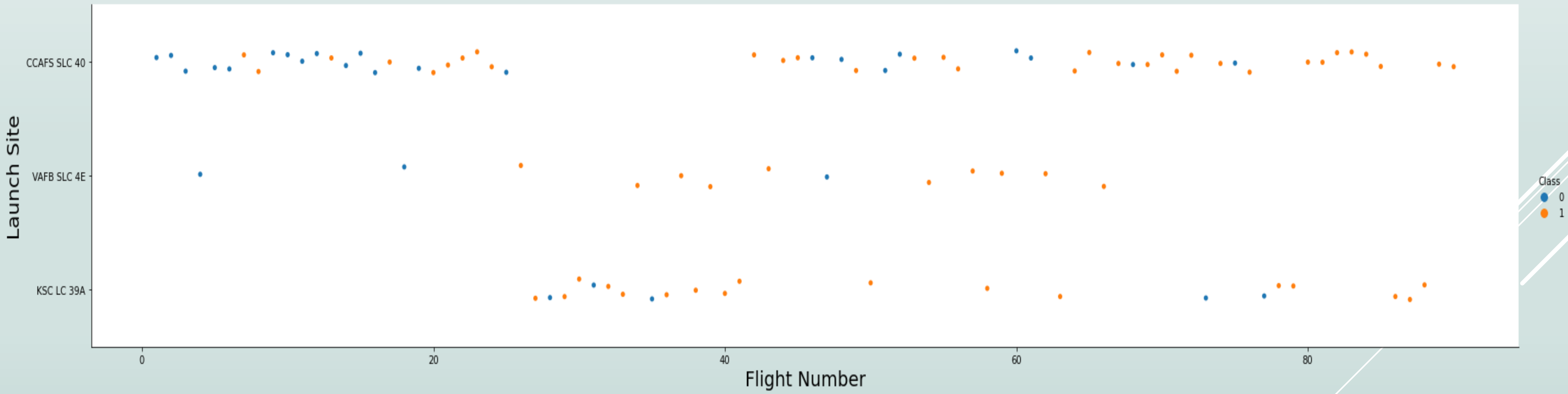
- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results

Section 2

Insights drawn from EDA

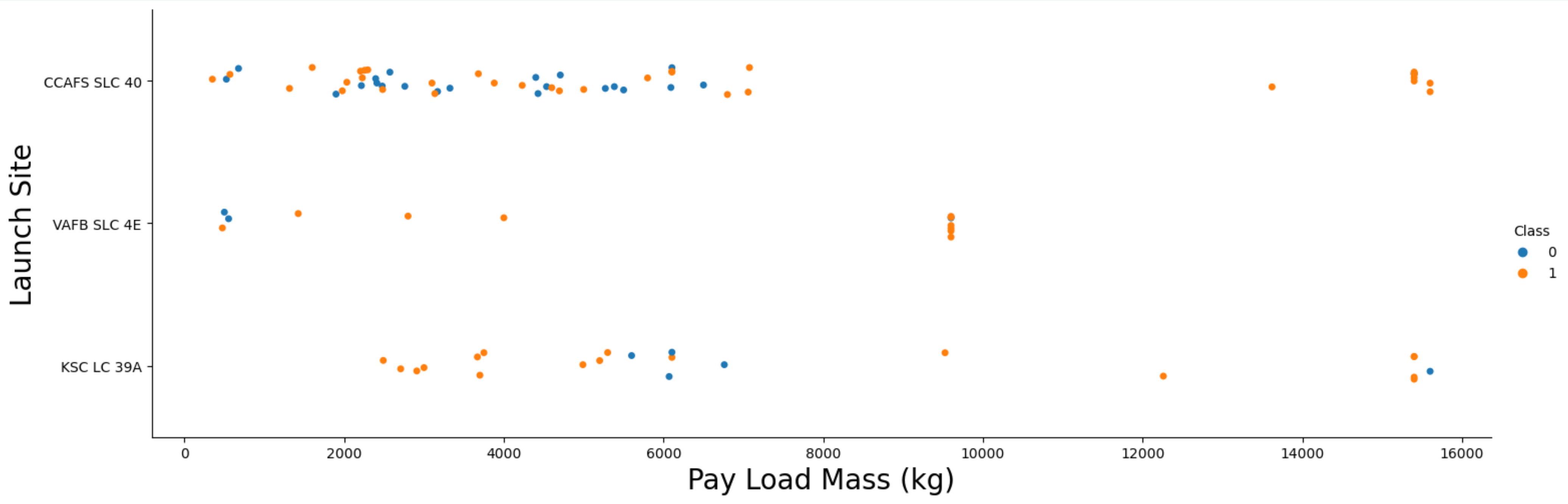
Flight Number vs. Launch Site

- From the plot, we found that the larger the flight amount at a launch site, the greater the success rate at a launch site.



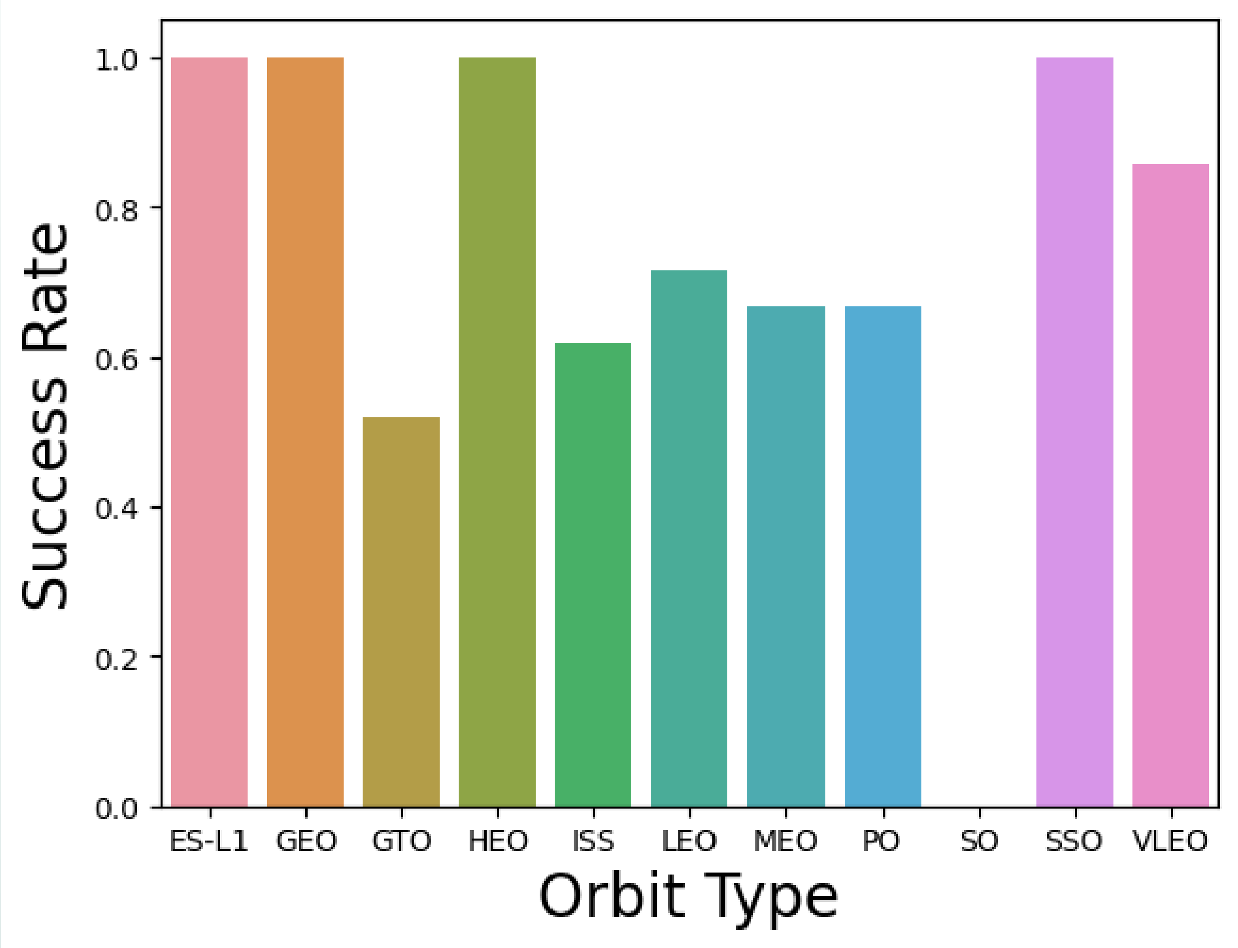
Payload vs. Launch Site

- If you observe Payload Vs. Launch Site scatter point chart you will find for the VAFB-SLC launch site there are no rockets launched for heavy payload mass(greater than 10000)



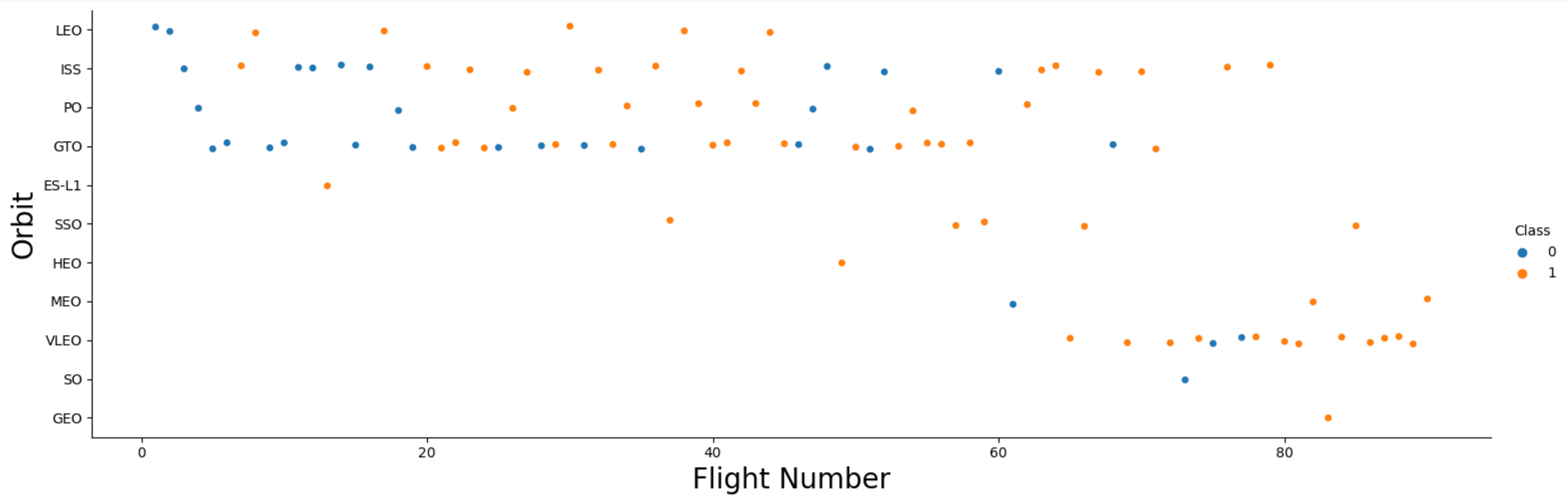
Success Rate vs. Orbit Type

- From the plot, we can see that ES-L1, GEO, HEO, SSO had the most success rate.



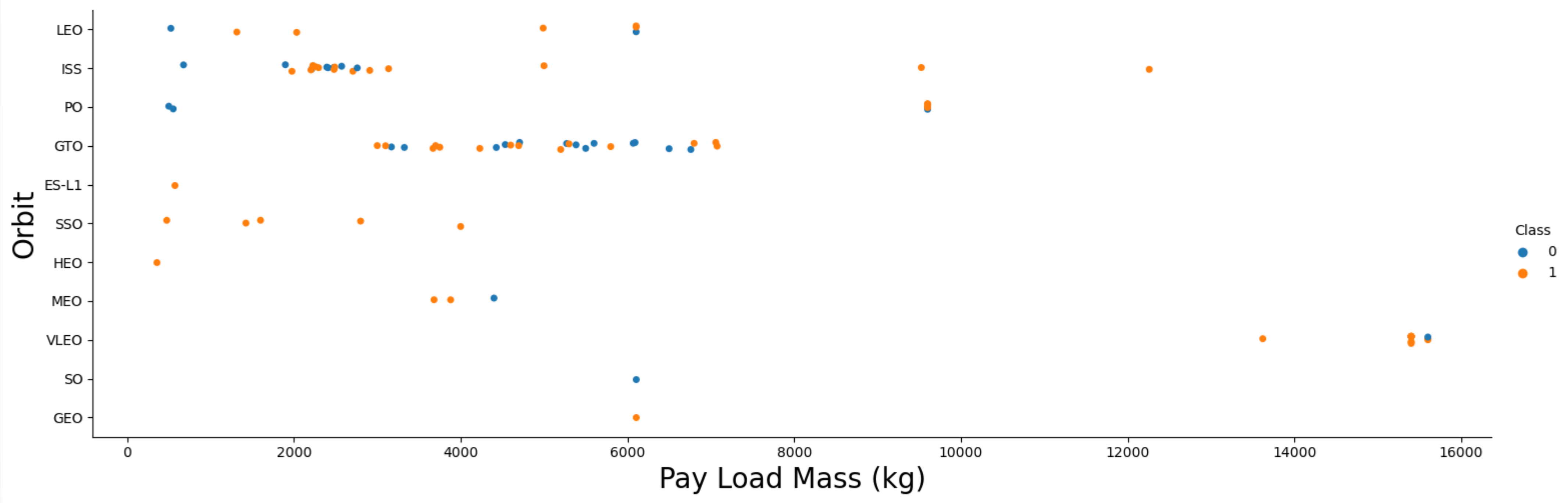
Flight Number vs. Orbit Type

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



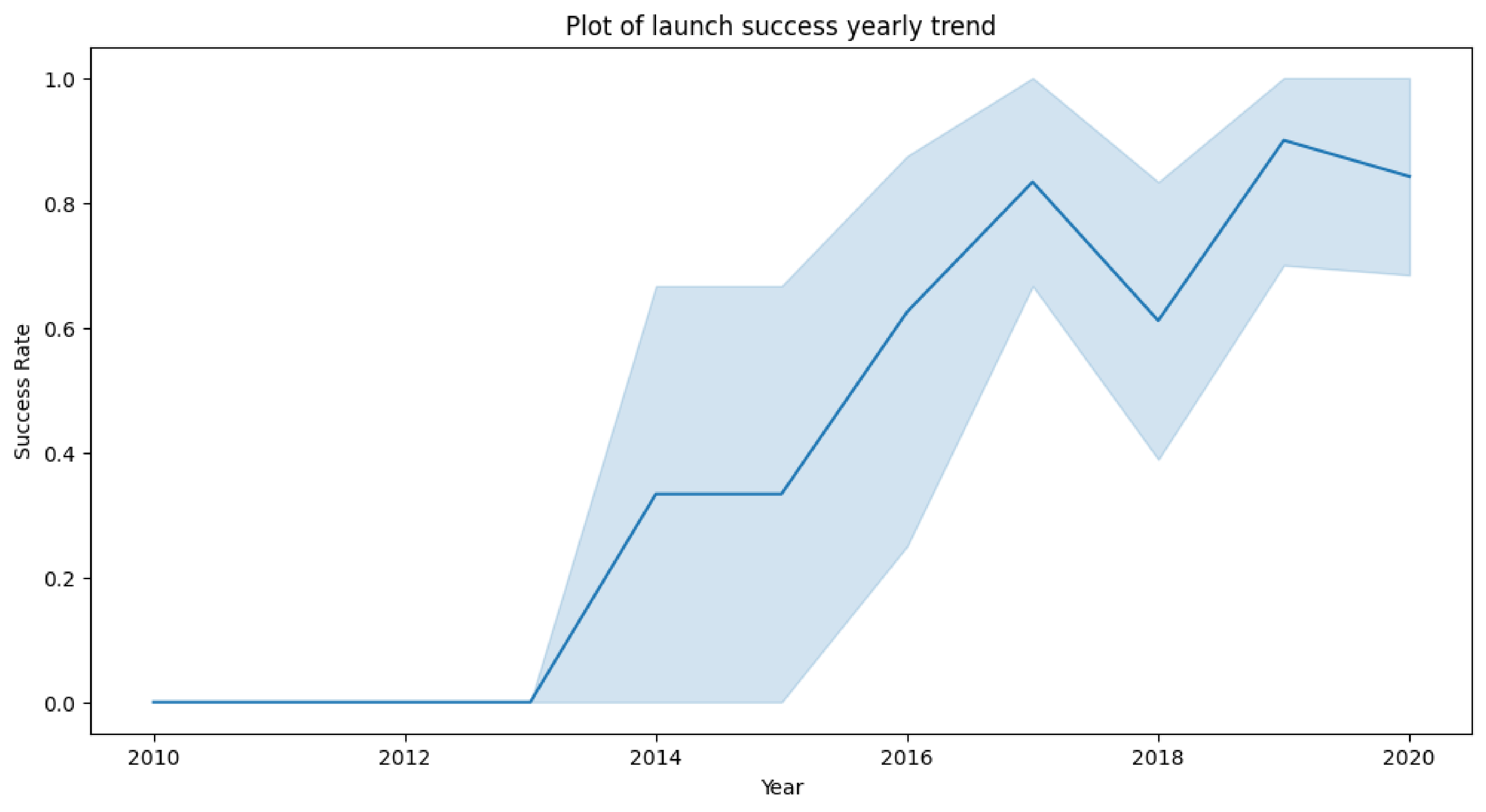
Payload vs. Orbit Type

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



Launch Success Yearly Trend

- From the plot, we can see that the success rate since 2013 kept on increasing until 2020



All Launch Site Names

- Unique launch site names:
  - CCAFS LC-40
  - VAFB SLC-4E
  - KSC LC-39A
  - CCAFS SLC-40
- Using **Distinct**, we can show unique names on the LAUNCH_SITE column.

```
%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 **Like** operator in the where clause, we can eliminate entries that are not starting with CCA.
- Also used **Limit** clause to get desired query output
  - e.g. *LAUNCH_SITE LIKE ‘CCA%’*

In [9]:
```
%sql SELECT * FROM SPACEXTBL WHERE LAUNCH_SITE LIKE 'CCA%' LIMIT 5
```

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

Out[9]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2010-04-06	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-08-12	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)
2012-05-22	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-08-10	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-01-03	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Total Payload Mass

- Using **SUM** function, calculate the total payload carried by boosters from NASA as 45596 using the query below

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

In [11]:
```
%sql SELECT SUM(PAYLOAD_MASS__KG_) as PAYLOAD_SUM FROM SPACEXTBL WHERE CUSTOMER='NASA (CRS)'
```

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

Out[11]:

PAYLOAD_SUM
45596

Average Payload Mass by F9 v1.1

- Using **AVG** function, calculate the average payload mass carried by booster version F9 v1.1 as 2928.4

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

In [13]:
```
%sql SELECT AVG(PAYLOAD_MASS__KG_) as AVG_PAYLOAD FROM SPACEXTBL WHERE BOOSTER_VERSION='F9 v1.1'
```

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

Out[13]:

AVG_PAYLOAD
2928.4

First Successful Ground Landing Date

- We can observe that the exact date for the first successful landing outcome on ground pad was achieved at the 22nd of December 2015.
- Using the **MIN** function, we are able to output the first date in the column.

```
In [15]: %sql SELECT min(DATE) as FIRST_LANDING FROM SPACEXTBL WHERE LANDING_OUTCOME='Success (ground pad)'

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

Out[15]:

FIRST_LANDING
2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

- Used the **WHERE** clause to filter the boosters which have successfully landed on drone ship and applied the **BETWEEN** and **AND** operator to determine successful landing with payload mass greater than 4000 but less than 6000.

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

```
[17]: %sql SELECT BOOSTER_VERSION FROM SPACEXTBL WHERE PAYLOAD_MASS__KG_ between 4000 and 6000 AND LANDING_OUTCOME='Success (drone ship)'
```

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

[17]:

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

Total Number of Successful and Failure Mission Outcomes

- Using the **Like** operator and **Or** operator in the **Where** clause to filter out the results while using the **Count** function to get the total of mission outcomes based on criteria.

List the total number of successful and failure mission outcomes

```
In [18]: %sql SELECT COUNT(*) FROM SPACEXTBL WHERE MISSION_OUTCOME LIKE '%Success%' OR MISSION_OUTCOME LIKE '%Failure%'
```

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

Out[18]:

COUNT(*)
101

- We can also adjust the query to show each type of mission outcome and their count result

List the total number of successful and failure mission outcomes

```
[10]: %sql SELECT MISSION_OUTCOME, COUNT(MISSION_OUTCOME) as OUTCOME_COUNT FROM SPACEXTBL WHERE MISSION_OUTCOME LIKE '%Success%' OR MISSION_OUTCOME LIKE '%Failure%' group by MISSION_OUTCOME
```

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

[10]:

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

Boosters Carried Maximum Payload

- We determined the booster that have carried the maximum payload using a subquery in the **WHERE** clause and the **MAX** function.

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

In [19]:
```
%sql SELECT BOOSTER_VERSION FROM SPACEXTBL WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTBL)
```

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

Out[19]:

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

- By using **SUBSTR** function, we acquired the year and month from the **DATE** column.
- Also filtered the Landing_Outcome by ‘Failure (drone ship)’ and used the **Like** operator for the desired year.

```
%sql SELECT substr(Date,1,4) as YEAR, substr(Date, 6, 2) as MONTH,  LANDING_OUTCOME, BOOSTER_VERSION, LAUNCH_SITE FROM SPACEXTBL WHERE LANDING_OUTCOME = 'Failure (drone ship)' AND "DATE" LIKE '%2015%'
```

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

YEAR	MONTH	Landing_Outcome	Booster_Version	Launch_Site
2015	10	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
2015	04	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

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

- Used **Count** function to get the total of landing outcomes.
- Used **Between** since we are trying to find results in a date range.
- Used **Group By** to get same results per category and **Order By** to rank by descending order

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(LANDING_OUTCOME) as COUNT FROM SPACEXTBL WHERE "DATE" BETWEEN '2010-06-04' and '2017-03-20' GROUP BY LANDING_OUTCOME  ORDER BY COUNT(LANDING_OUTCOME) DESC
```

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

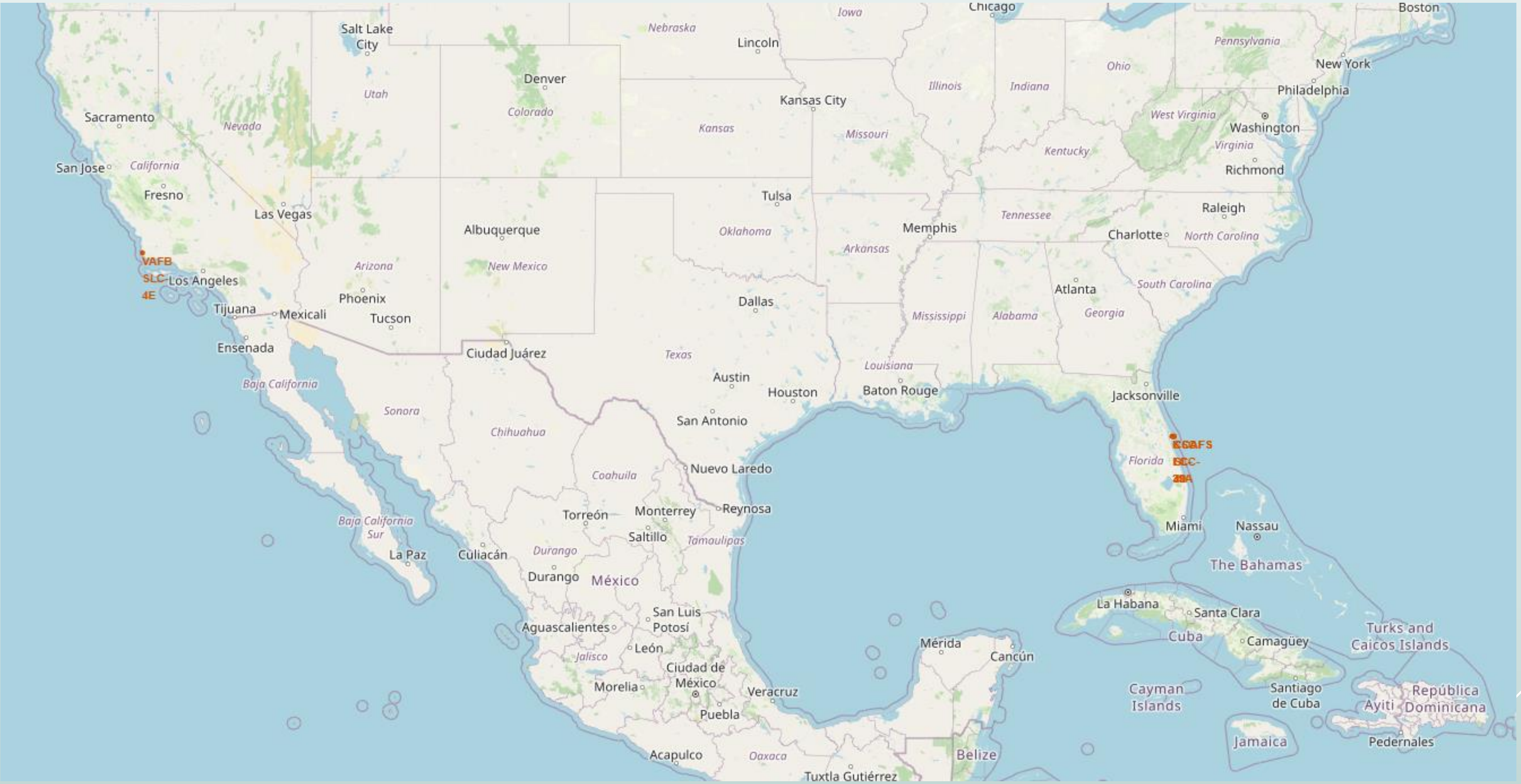
Landing_Outcome	COUNT
No attempt	10
Success (ground pad)	5
Success (drone ship)	5
Failure (drone ship)	5
Controlled (ocean)	3
Uncontrolled (ocean)	2
Precluded (drone ship)	1
Failure (parachute)	1

Section 3

Launch Sites Proximities Analysis

Marking all launch sites on a map

- As observed from the map, the SpaceX launch sites are in the United States of America and are near the coast line.



Colored Markers showing Launch sites and Outcomes

Legend:

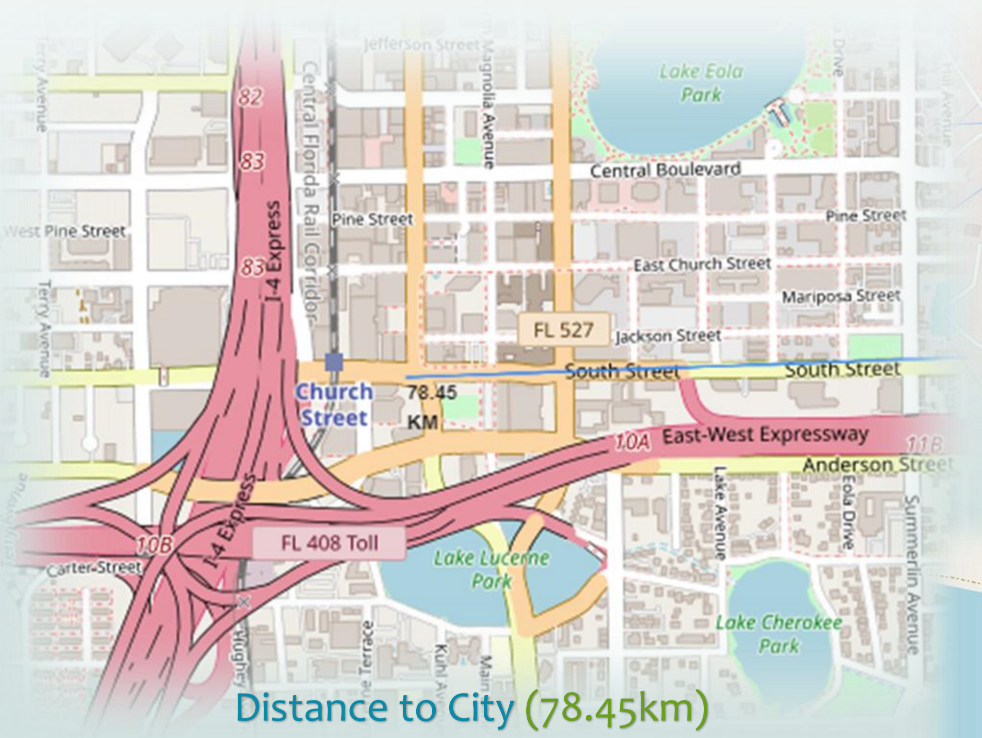
Green Marker – Success

Red Marker - Failure

California Launch Sites

Florida Launch Sites

Distance to Landmarks



Distance to City (78.45km)

Distance to Coastline (0.9km)

Distance to Railway (1.54km)

Distance to Highway (29.21km)

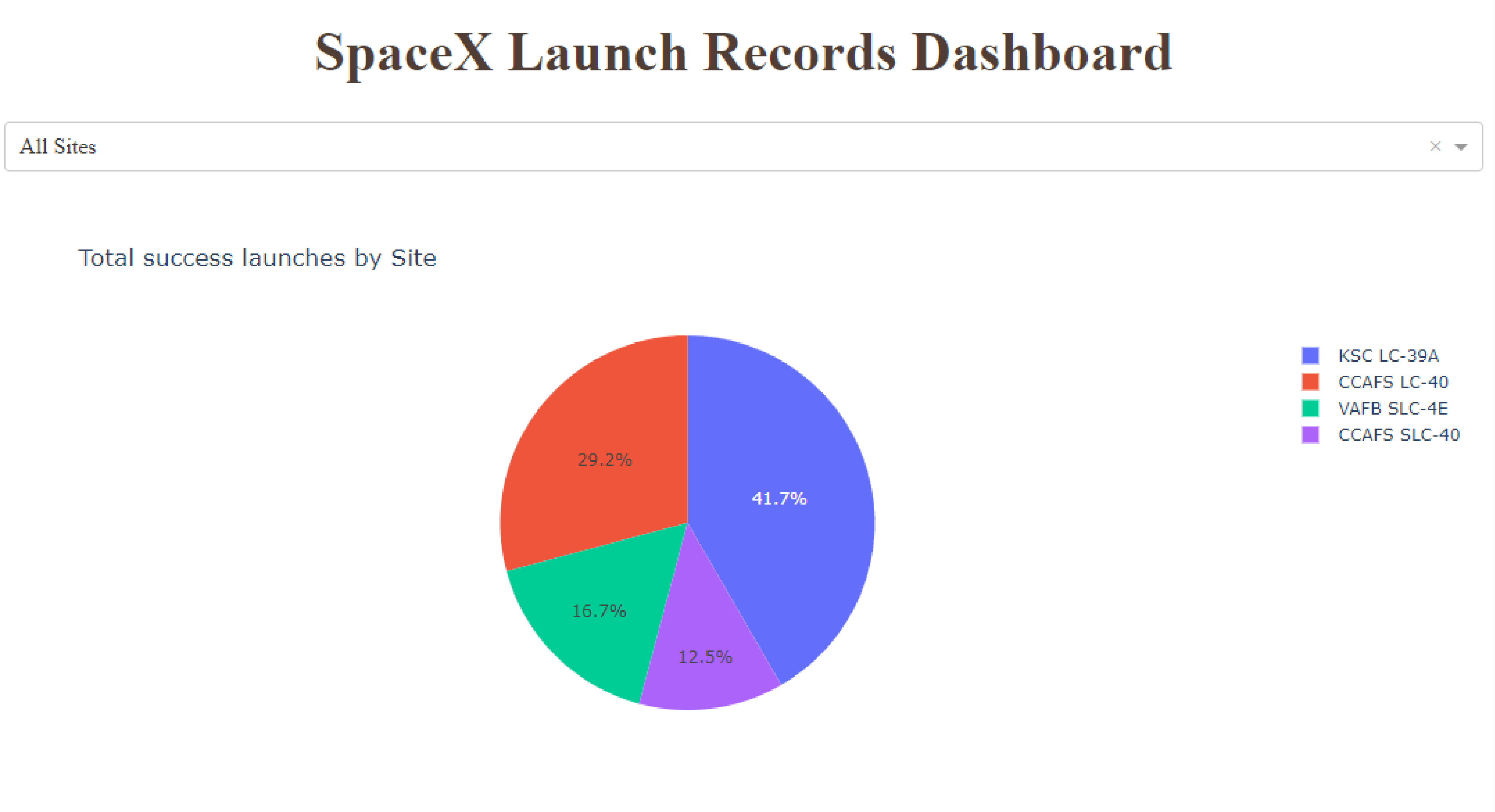
Are launch sites in close proximity to landmarks?

- Railways - Yes
- Highways – No
- Coastline – Yes
- Cities - No

Section 4

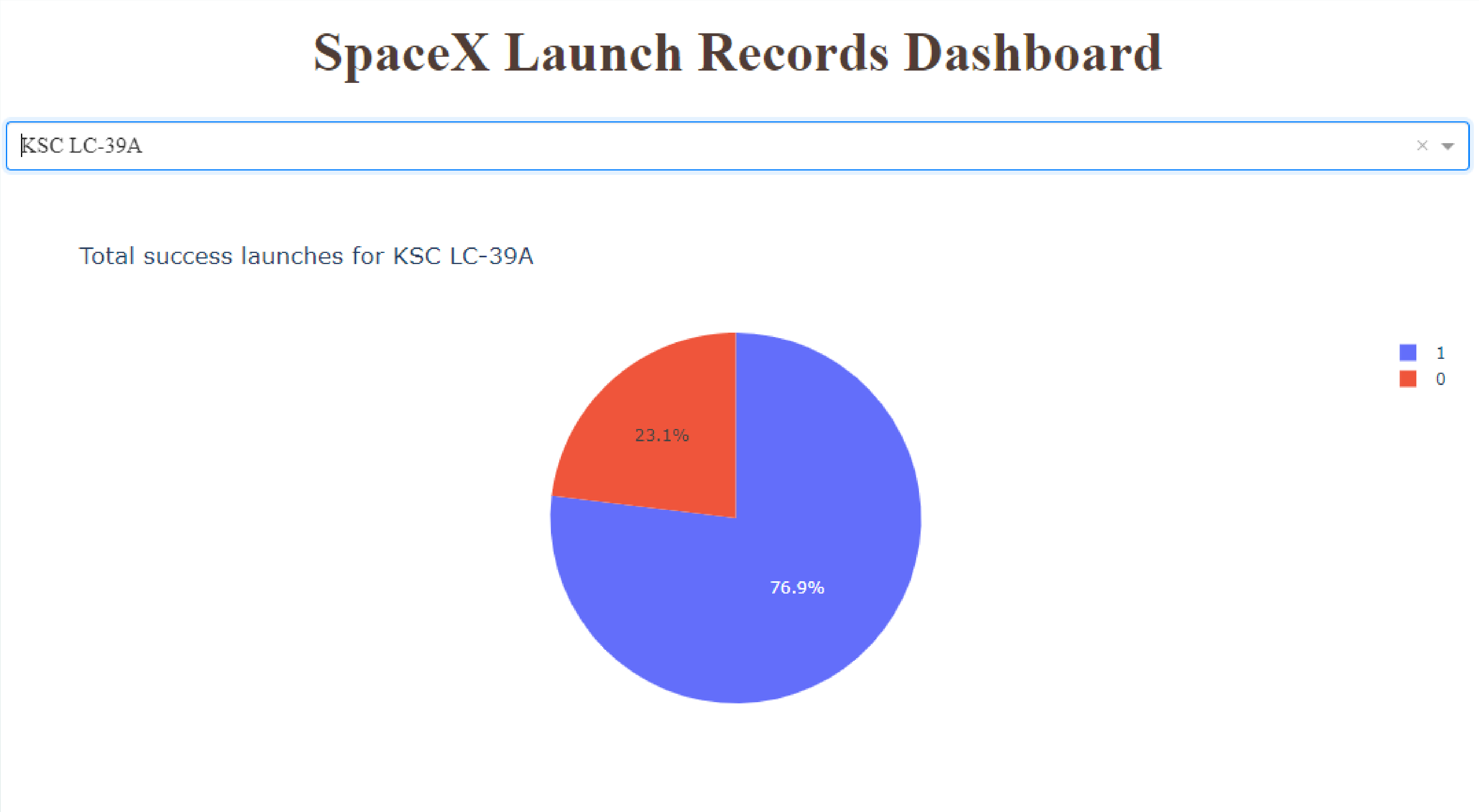
Build a Dashboard with Plotly Dash

Pie chart success rates by each launch site



- As seen on the dashboard, KSC LC-39A has the most successful launches from all the sites

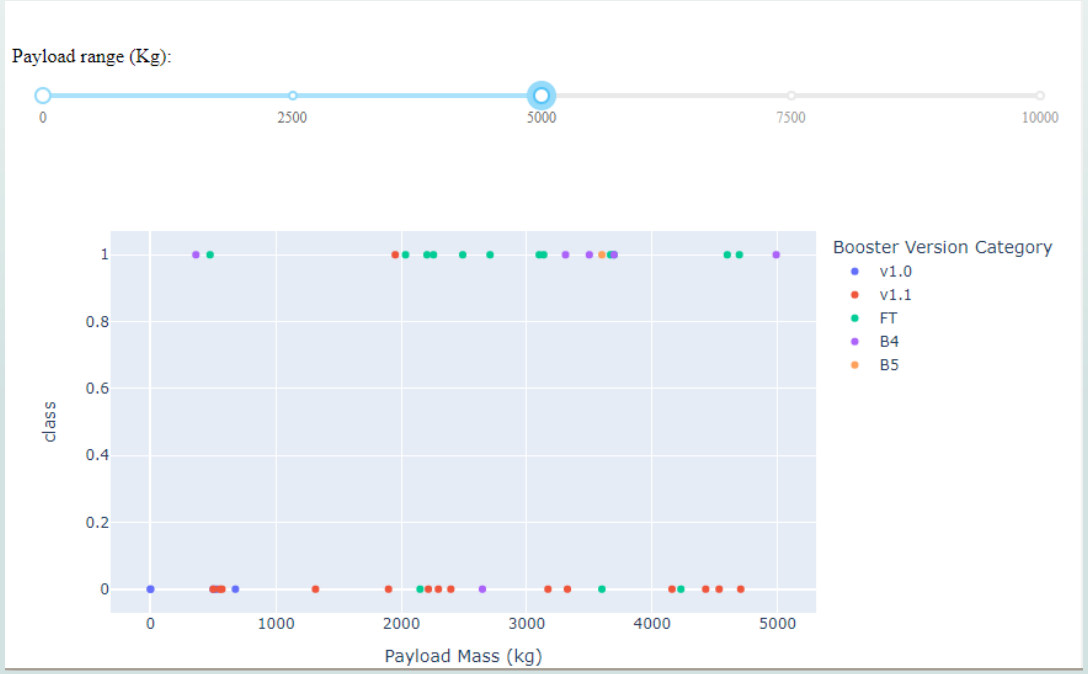
Pie chart showing the Launch site with the highest success ratio



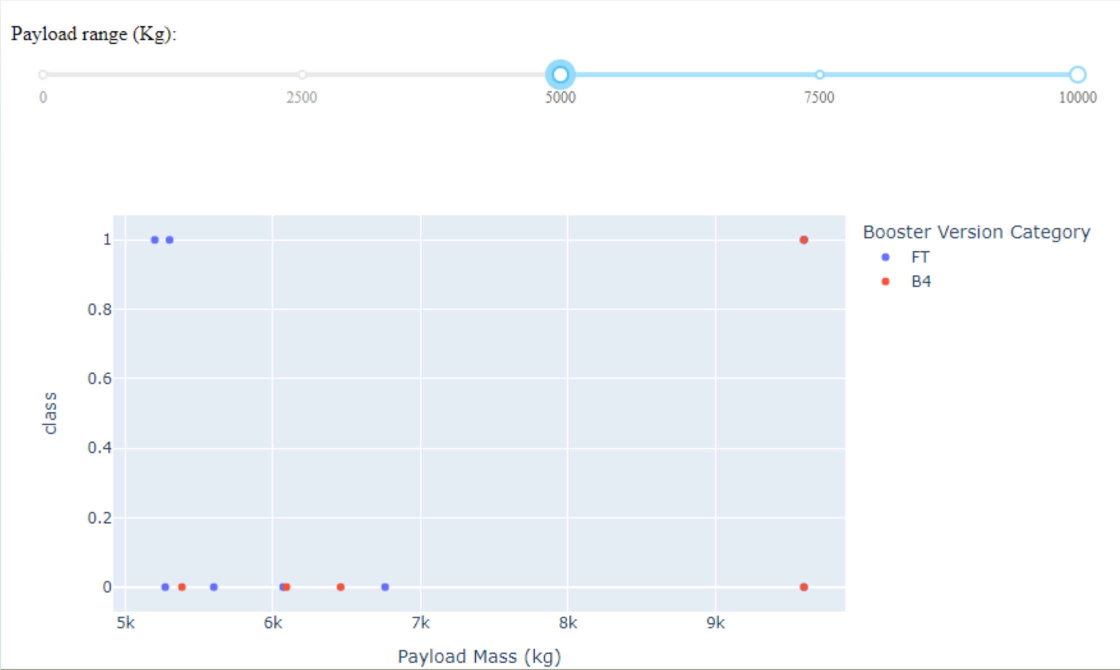
- KSC LC-39A achieved a 76.9% success rate while only having a 23.1% failure rate.

Payload vs Launch Outcome Scatter plot for all sites using different payload range

Low to Mid Payload 0kg – 5000kg



Mid to High Payload 5000kg – 10000kg



- From the images above, we can see that the success rates for lower payloads is greater than in higher payloads

Section 5

Predictive Analysis (Classification)

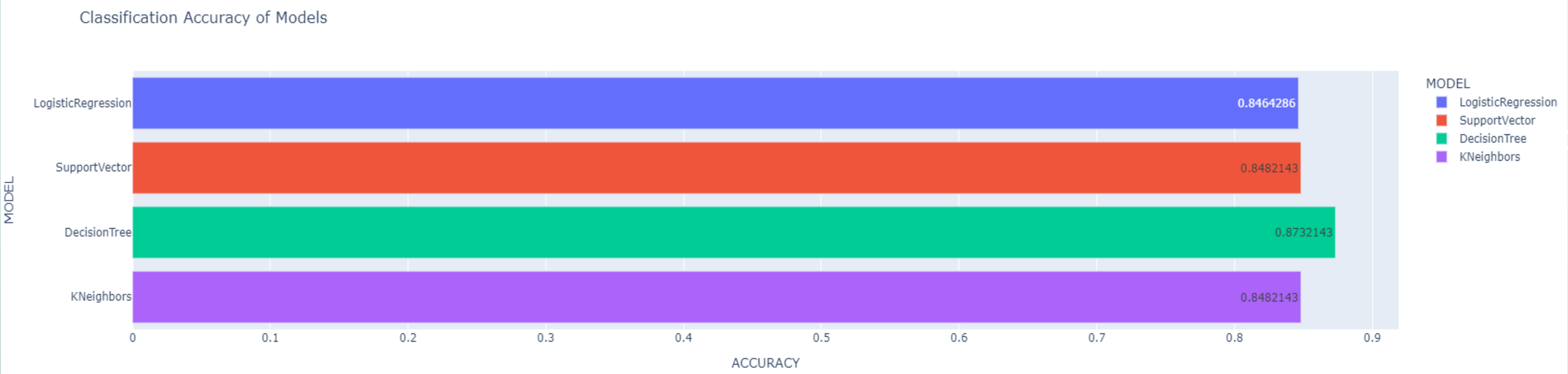
Classification Accuracy

- The Decision Tree Model has the best accuracy with a score of 0.8732142857142856

Find the method performs best:

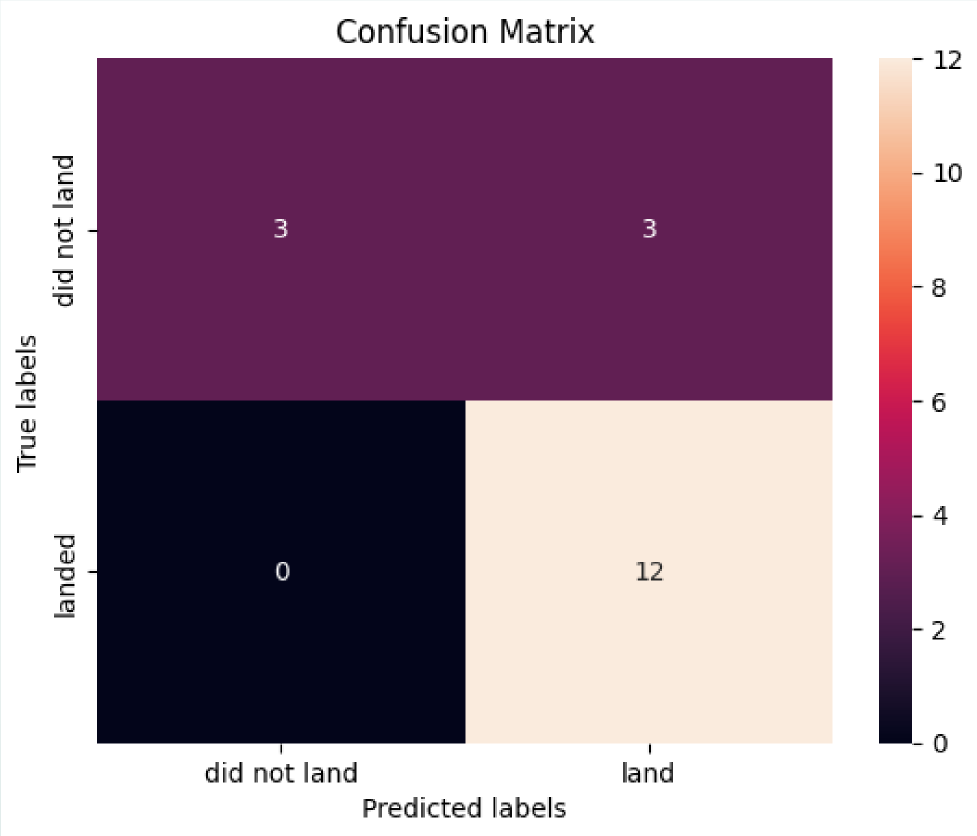
```
models = {'KNeighbors':knn_cv.best_score_,
              'DecisionTree':tree_cv.best_score_,
              'LogisticRegression':logreg_cv.best_score_,
              'SupportVector': svm_cv.best_score_}

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 for the decision tree classifier shows that the classifier can distinguish between the different classes. The major problem is the false positives .i.e., unsuccessful landing marked as successful landing by the classifier.



Conclusions

- Launch success rates started to increase in **2013** up to **2020**
- The larger the flight amount at a launch site, the higher the success rate it has.
- Orbits **ES-L1**, **GEO**, **HEO** and **SSO** had the most success rate.
- **KSC LC-39A** had the most successful launches of any sites.
- The **Decision Tree** classifier is the best machine learning algorithm for this task.

Appendix

- Here is the code for creating the simple bar chart for classification's model accuracy

```
data = {'MODEL': ['LogisticRegression', 'SupportVector', 'DecisionTree', 'KNeighbors'],
      'ACCURACY': [0.8464285714285713, 0.8482142857142856, 0.8732142857142856, 0.8482142857142858]}

df = pd.DataFrame(data)

fig = px.bar(df, x = "ACCURACY", y = "MODEL",
             title= "Classification Accuracy of Models",
             color= 'MODEL',
             text_auto= True)
fig.show()
```

Thank you!