IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

<THULANI BABELI>
<02/05/2023>

Outline

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

Executive Summary

- Summary of methodologies
  - Employed methodologies include data collection using API and Web Scraping
  - Use of Data Wrangling
  - Exploratory Data analysis using SQL and data visualizations; Interactive Visual analytics using Folium
  - Prediction using machine learning classification models
- Summary of all results
  - Results from exploratory data analysis
  - Visuals from interactive analytics
  - Results from predictive analysis

Introduction

- Project background and context

  Space X’s Falcon 9 rocket launches are substantially more cost effective than those of competitors. Much of the savings comes from the fact that Space X can reuse the first stage. We can therefore deduce the cost of a launch if we can determine if the first stage will land.

  The purpose of this analysis is to create a data driven machine learning pipeline to predict if the first stage will land successfully.

- These are the issues we want to find and answers we wish to answer.
  - The factors that make the rocket land successfully.
  - Various variables that determine the successful landing
  - Operating conditions that make successful landing

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - Data collection was collected through SpaceX REST API and through Web Scraping from Wikipedia
- Perform data wrangling
  - One-hot encoding was applied to data fields
- Perform exploratory data analysis (EDA) using visualization and SQL
  - Scatter and Bar plots to understand data patterns
- Perform interactive visual analytics using Folium and Plotly Dash
  - Visual analytics using Folium and Plotly Dash Visualisations
- Perform predictive analysis using classification models
  - Building and evaluations of classification models

Data Collection

- Describe how data sets were collected.
  - Data was collected from SpaceX API and through web scrapping was from Wikipedia.
  - Data was collected from SpaceX API and was converted into a dataframe using pandas library and web scrapping was from Wikipedia was performed.
  - Data wrangling was performed to fill missing values and the dataframe filtered for Falcon 9 rockets

Data Collection – SpaceX API

- A screenshot of a code used to collect data.
- The file can be accessed from the following link:
- https://github.com/tbabeli/dsLearning/blob/main/Final_Assignment.ipynb

github.com/tbabeli/dsLearning/blob/main/Final_Assignment.ipynb

```
        Serial.append(None)
    Outcome.append(str(core['landing_success'])+' '+str(core['landing_type']))
    Flights.append(core['flight'])
    GridFins.append(core['gridfins'])
    Reused.append(core['reused'])
    Legs.append(core['legs'])
    LandingPad.append(core['landpad'])
```

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

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

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

Check the content of the response

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

b'[{"fairings":{"reused":false,"recovery_attempt":false,"recovered":false,"ships":[]},"links":{"patch":{"small":"https://images2.imgbox.com/94/f2/NN6Ph45r_o.png","large":"https://images2.imgbox.com/5b/02/QcxHUb5V_o.png"},"reddit":{"campaign":null,"launch":null,"media":null,"recovery":null},"flickr":{"small":[],"original":[]},"presskit":null,"webcast":"https://www.youtube.com/watch?v=0a_00nJ_Y88","youtube_id":"0a_00nJ_Y88","article":"https://www.s

Data Collection - Scraping

- Web Scraping using Beautiful Soup
- The table was parsed and converted into a pandas data frame
- File can be accessed from the following link:
- https://github.com/tbabeli/dsLearning/blob/main/Webscraping_Assignment.ipynb

In [5]:
```python
static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"
```

Next, request the HTML page from the above URL and get a `response` object

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

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

In [6]:
```python
# use requests.get() method with the provided static_url
data = requests.get(static_url).text
# assign the response to a object
```

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

In [7]:
```python
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(data,"html.parser")
```

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

In [8]:
```python
# Use soup.title attribute
print(soup.title)
```

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

Data Wrangling

- Exploratory data analysis was done and training labels determined
- Landing outcomes labels were created
- https://github.com/tbabeli/dsLearning/blob/main/Data_Wrangling_Assignment.ipynb

In [2]:
```python
# Pandas is a software library written for the Python programming language for data manipulation and analysis.
import pandas as pd
#NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collectio
import numpy as np
```

Data Analysis

Load Space X dataset, from last section.

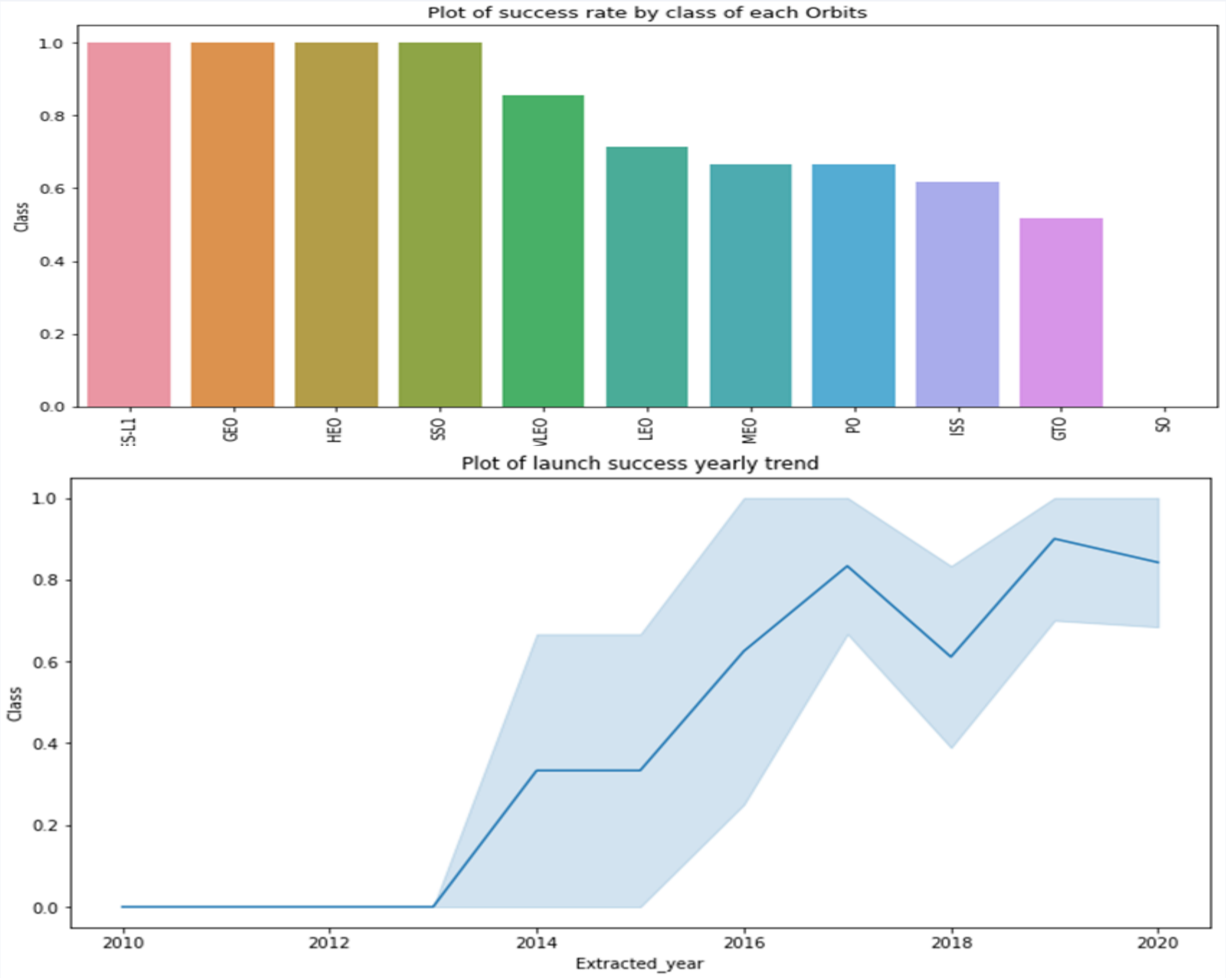
In [3]:
```python
df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_1.csv")
df.head(10)
```

Out[3]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Lon
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.5
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.5
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.5

EDA with Data Visualization

- Data was expored 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.
- https://github.com/tbabeli/dsLearning/blob/main/Data%20Visualization.ipynb



EDA with SQL

- SQL Query Summary
  - The names of unique launch sites in the space mission.
  - The total payload mass carried by boosters launched by NASA (CRS)
  - The 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.

- https://github.com/tbabeli/dsLearning/blob/main/SQL.ipynb

Build an Interactive Map with Folium

- We 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.
- We assigned the feature launch outcomes (failure or success) to class 0 and 1 respectively.
- Using the color-labeled marker clusters, we identified launch sites with relatively high success rate.
- We calculated the distances between a launch site to its proximities.
- https://github.com/tbabeli/dsLearning/blob/main/Interactive_Visualization.ipynb

Build a Dashboard with Plotly Dash

- We built an interactive dashboard with Plotly dash
- We plotted pie charts showing the total launches by a certain sites
- We plotted scatter graph showing the relationship with Outcome and Payload Mass (Kg) for the different booster version.
- Add the GitHub URL of your completed Plotly Dash lab, as an external reference and peer-review purpose

Predictive Analysis (Classification)

- We loaded the data using numpy and pandas, transformed the data, split our data into training and testing.
- We built different machine learning models and tune different hyperparameters using GridSearchCV.
- We used accuracy as the metric for our model, improved the model using feature engineering and algorithm tuning.
- We determined the best performing classification model.
- https://github.com/tbabeli/dsLearning/blob/main/Predictive_Analysis.ipynb

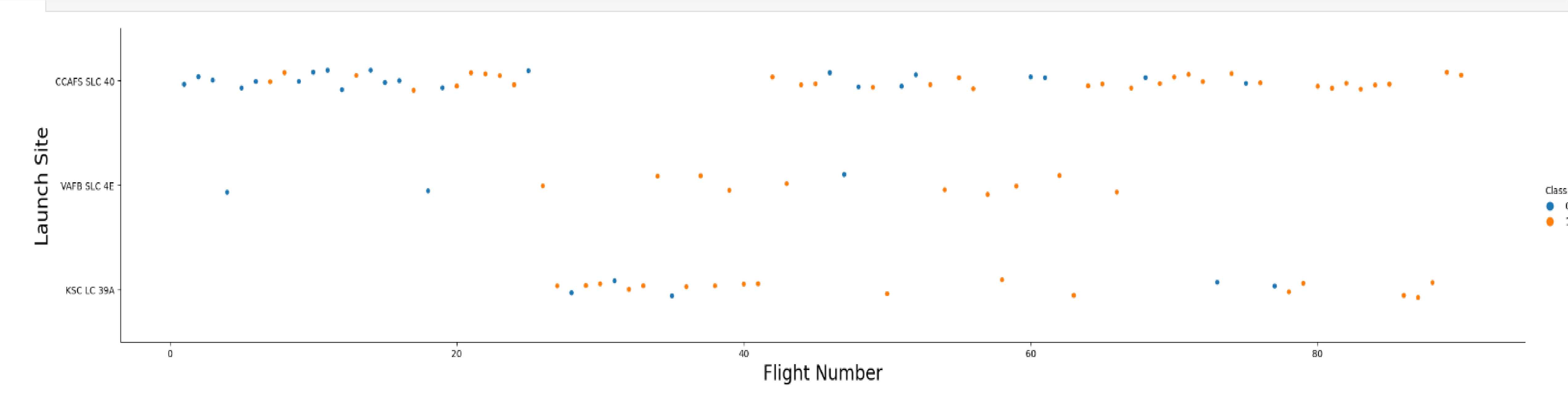
Results

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

Section 2

Insights drawn from EDA

Flight Number vs. Launch Site

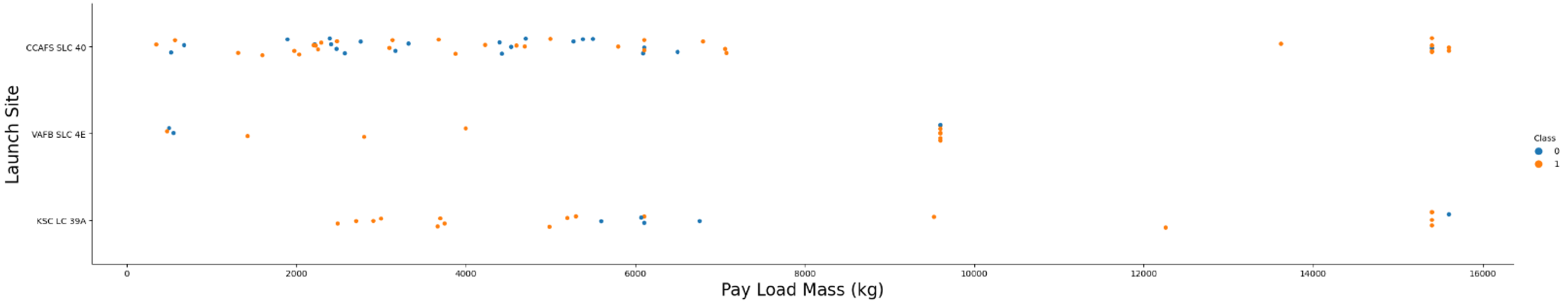


- For the given launch sites, higher flight numbers increase the success rate of the launch as can be seen from the plot illustrated above.

Payload vs. Launch Site

In [8]:

```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch site, and hue to be the class value
sns.catplot(y="LaunchSite", x="PayloadMass", hue="Class", data=df, aspect = 5)
plt.xlabel("Pay Load Mass (kg)",fontsize=20)
plt.ylabel("Launch Site",fontsize=20)
plt.show()
```

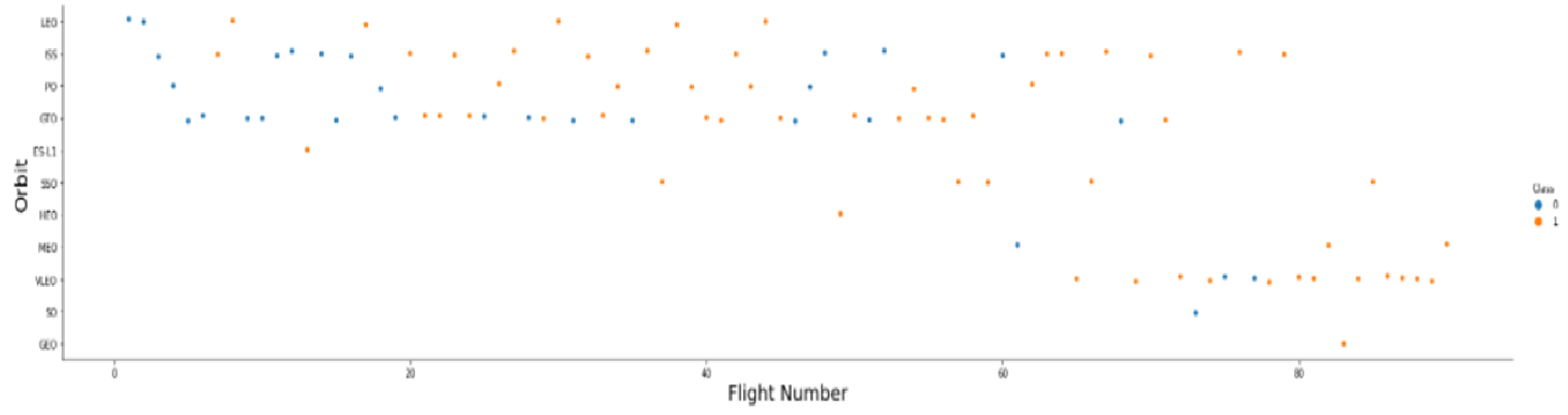


- Greater payload leads to higher success rate of the launch as it can be seen from the scatter plot above.

Success Rate vs. Orbit Type

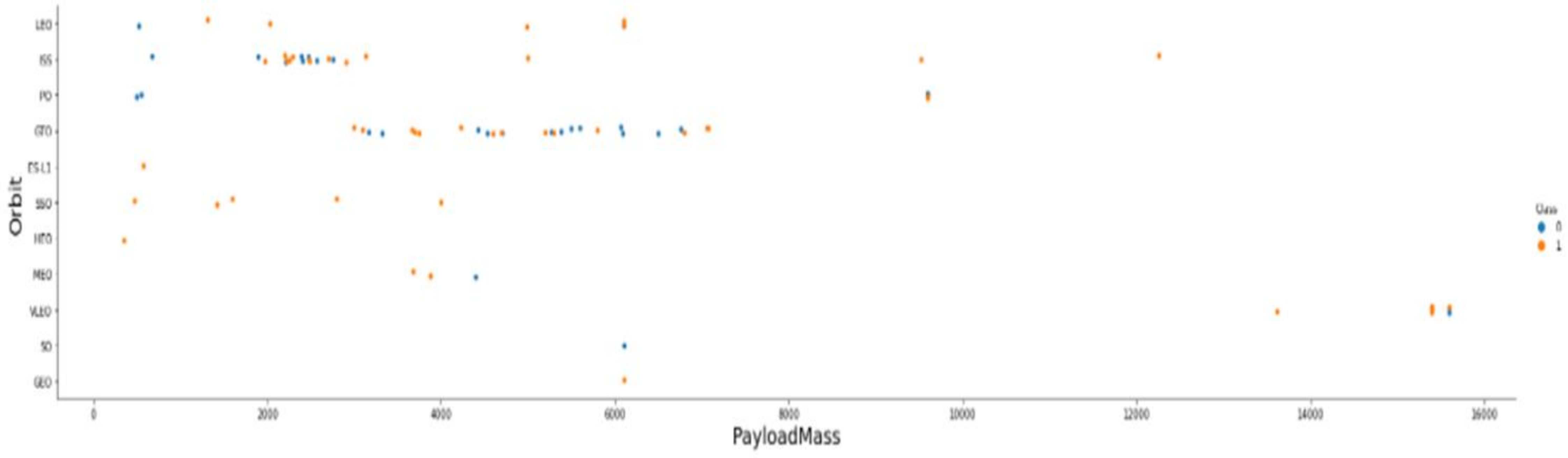
- ES-L1, GEO, HEO, SSO, VLEO are orbit types which had the most success rate as can be seen from the bar chart above.

Flight Number vs. Orbit Type



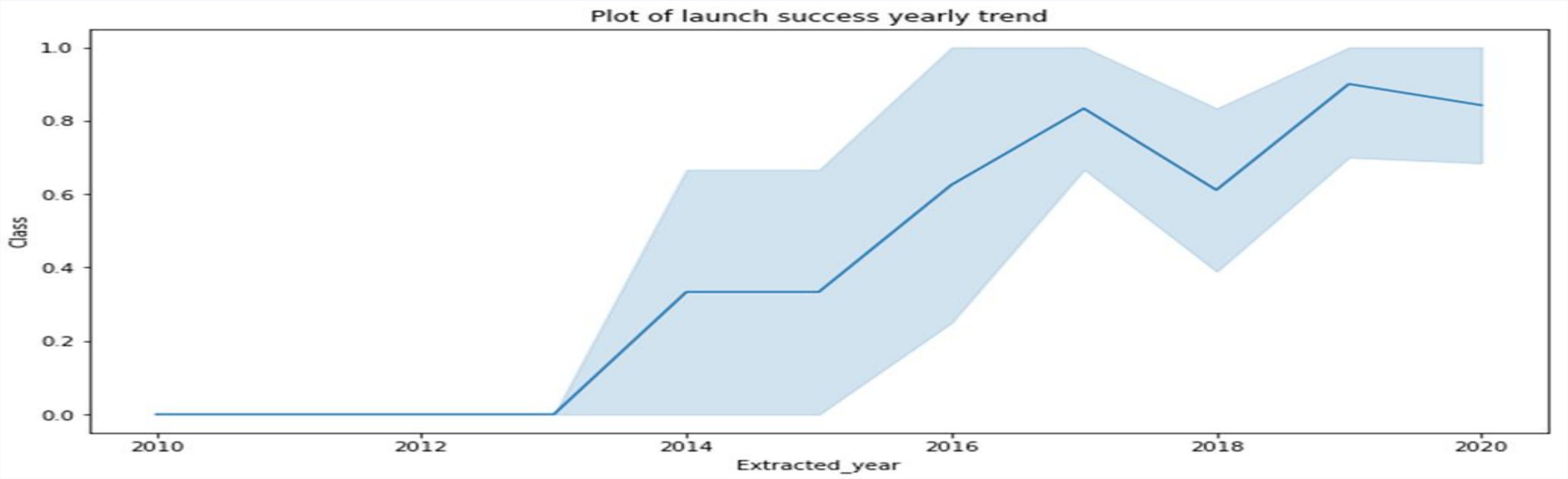
- For the LEO orbit, success increases with the number of flights as can be seen from the scatterplot above.
- For the GTO orbit, no relationship between the number of flights and success rate as can be seen from the scatterplot above.

Payload vs. Orbit Type



- Heavy payloads have a negative influence on MEO, GTO and VLEO orbits but positive influence on PO, LEO and ISS orbits. This is illustrated on the plot above.

Launch Success Yearly Trend



- Success rate constant between 2010 and 2013, but generally on an upward trend from 2013 to 2020 as can be seen from the plot, with the highest success rate seemingly in 2019.

All Launch Site Names

- The names of the Launch sites listed below drawn using the sql query.

Out[6]:

launch_site
CCAFS LC-40
CCAFS SLC-40
CCAFSSLC-40
KSC LC-39A
VAFB SLC-4E

Launch Site Names Begin with 'CCA'

- This query returns the records where launch sites begin with `CCA`in all launch sites available.

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

In [11]:
```
task_2 = '''
        SELECT *
        FROM SpaceX
        WHERE LaunchSite LIKE 'CCA%'
        LIMIT 5
        '''
create_pandas_df(task_2, database=conn)
```

Out[11]:

	date	time	boosterversion	launchsite	payload	payloadmasskg	orbit	customer	missionoutcome	landingoutcome
0	2010-04-06	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
1	2010-08-12	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of...	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2	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
3	2012-08-10	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
4	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

- Total payload carried by boosters from NASA is $45596

In [12]:

```
task_3 = '''
        SELECT SUM(PayloadMassKG) AS Total_PayloadMass
        FROM SpaceX
        WHERE Customer LIKE 'NASA (CRS)'
        '''
create_pandas_df(task_3, database=conn)
```

Out[12]:

	total_payloadmass
0	45596

Average Payload Mass by F9 v1.1

- The average payload mass carried by booster version F9 v1.1 is 2928.4 as can be seen from the results of the sql query.

In [13]:
```
task_4 = '''
        SELECT AVG(PayloadMassKG) AS Avg_PayloadMass
        FROM SpaceX
        WHERE BoosterVersion = 'F9 v1.1'
        '''
create_pandas_df(task_4, database=conn)
```

Out[13]:

	avg_payloadmass
0	2928.4

First Successful Ground Landing Date

- The date for the first successful landing outcome on ground pad was 22nd December 2015 as can be seen below from the results of the sql query

In [14]:

```
task_5 = '''
        SELECT MIN(Date) AS FirstSuccessfull_landing_date
        FROM SpaceX
        WHERE LandingOutcome LIKE 'Success (ground pad)'
        '''
create_pandas_df(task_5, database=conn)
```

Out[14]:

	firstsuccessfull_landing_date
0	2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

- The names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000: these are illustrated below with the sql query.

In [15]:

```
task_6 = '''
        SELECT BoosterVersion
        FROM SpaceX
        WHERE LandingOutcome = 'Success (drone ship)'
            AND PayloadMassKG > 4000
            AND PayloadMassKG < 6000
        '''
create_pandas_df(task_6, database=conn)
```

Out[15]:

	boosterversion
0	F9 FT B1022
1	F9 FT B1026
2	F9 FT B1021.2
3	F9 FT B1031.2

Total Number of Successful and Failure Mission Outcomes

- The total number of successful and failure mission outcomes is illustrated below using the following lines of sql code.

```
In [16]:  task_7a = '''
                  SELECT COUNT(MissionOutcome) AS SuccessOutcome
                  FROM SpaceX
                  WHERE MissionOutcome LIKE 'Success%'
                  '''

          task_7b = '''
                  SELECT COUNT(MissionOutcome) AS FailureOutcome
                  FROM SpaceX
                  WHERE MissionOutcome LIKE 'Failure%'
                  '''
          print('The total number of successful mission outcome is:')
          display(create_pandas_df(task_7a, database=conn))
          print()
          print('The total number of failed mission outcome is:')
          create_pandas_df(task_7b, database=conn)
```

The total number of successful mission outcome is:

	successoutcome
0	100

The total number of failed mission outcome is:

Out[16]:

	failureoutcome
0	1

Boosters Carried Maximum Payload

- The names of the booster which have carried the maximum payload mass, these were drawn using the below lines of sql code:

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

In [17]:
```
task_8 = '''
        SELECT BoosterVersion, PayloadMassKG
        FROM SpaceX
        WHERE PayloadMassKG = (
                                SELECT MAX(PayloadMassKG)
                                FROM SpaceX
                                )
        ORDER BY BoosterVersion
        '''
create_pandas_df(task_8, database=conn)
```

Out[17]:

	boosterversion	payloadmasskg
0	F9 B5 B1048.4	15600
1	F9 B5 B1048.5	15600
2	F9 B5 B1049.4	15600
3	F9 B5 B1049.5	15600
4	F9 B5 B1049.7	15600
5	F9 B5 B1051.3	15600
6	F9 B5 B1051.4	15600
7	F9 B5 B1051.6	15600
8	F9 B5 B1056.4	15600
9	F9 B5 B1058.3	15600
10	F9 B5 B1060.2	15600
11	F9 B5 B1060.3	15600

2015 Launch Records

- The List of the failed landing outcomes in drone ship, their booster versions, and launch site names for in year 2015, drawn using the sql code lines below

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

In [18]:

```
task_9 = '''
        SELECT BoosterVersion, LaunchSite, LandingOutcome
        FROM SpaceX
        WHERE LandingOutcome LIKE 'Failure (drone ship)'
            AND Date BETWEEN '2015-01-01' AND '2015-12-31'
        '''
create_pandas_df(task_9, database=conn)
```

Out[18]:

	boosterversion	launchsite	landingoutcome
0	F9 v1.1 B1012	CCAFS LC-40	Failure (drone ship)
1	F9 v1.1 B1015	CCAFS LC-40	Failure (drone ship)

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

- Rank of 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. These were found using the sql code below:

```
In [19]:  task_10 = '''
                  SELECT LandingOutcome, COUNT(LandingOutcome)
                  FROM SpaceX
                  WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20'
                  GROUP BY LandingOutcome
                  ORDER BY COUNT(LandingOutcome) DESC
                  '''
          create_pandas_df(task_10, database=conn)
```

Out[19]:

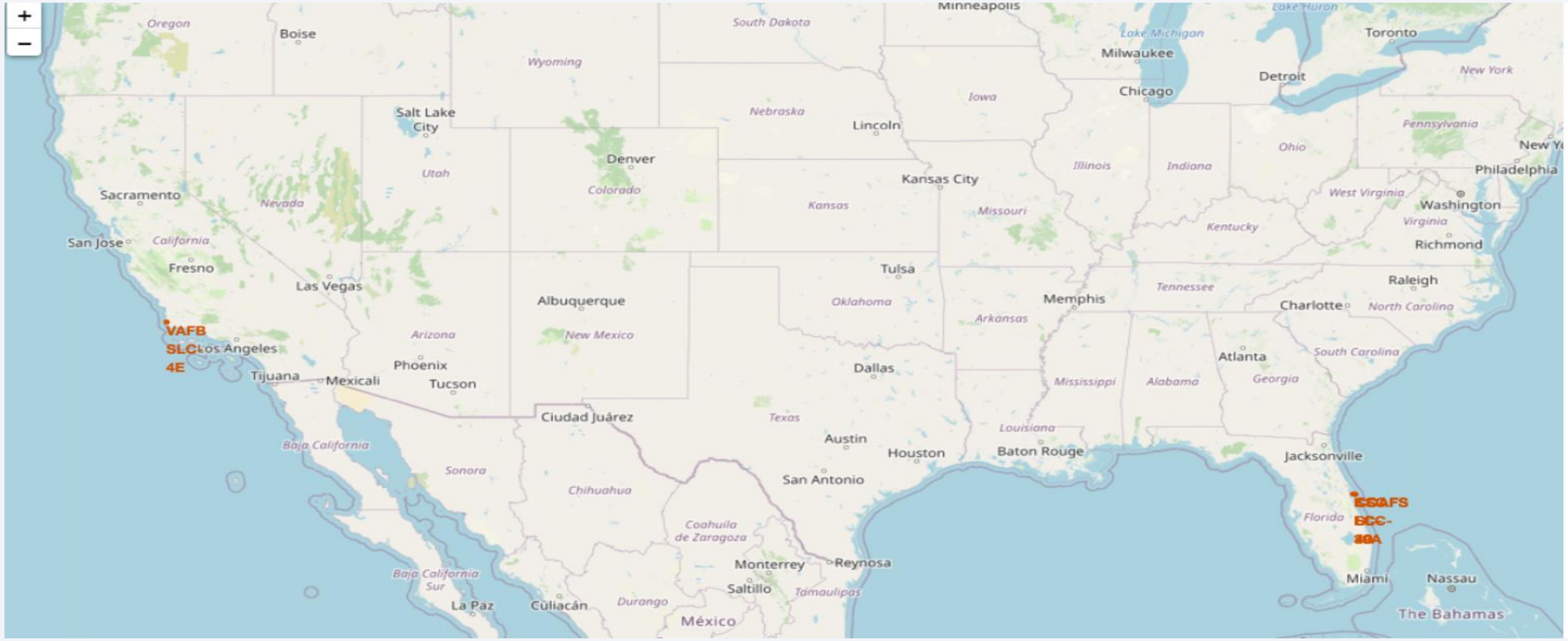
	landingoutcome	count
0	No attempt	10
1	Success (drone ship)	6
2	Failure (drone ship)	5
3	Success (ground pad)	5
4	Controlled (ocean)	3
5	Uncontrolled (ocean)	2
6	Precluded (drone ship)	1
7	Failure (parachute)	1

Section 3

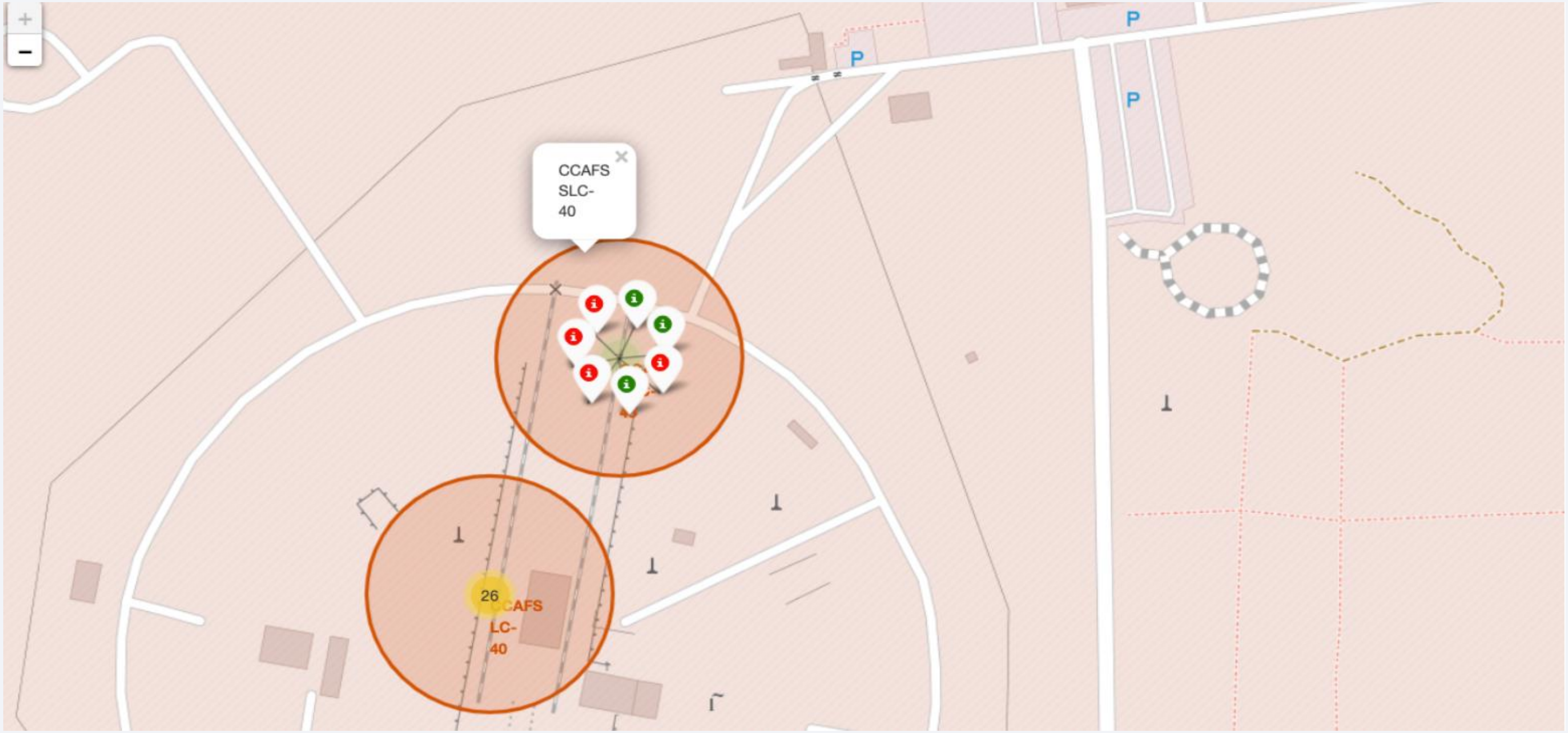
Launch Sites Proximities Analysis

Launch Sites on the Map

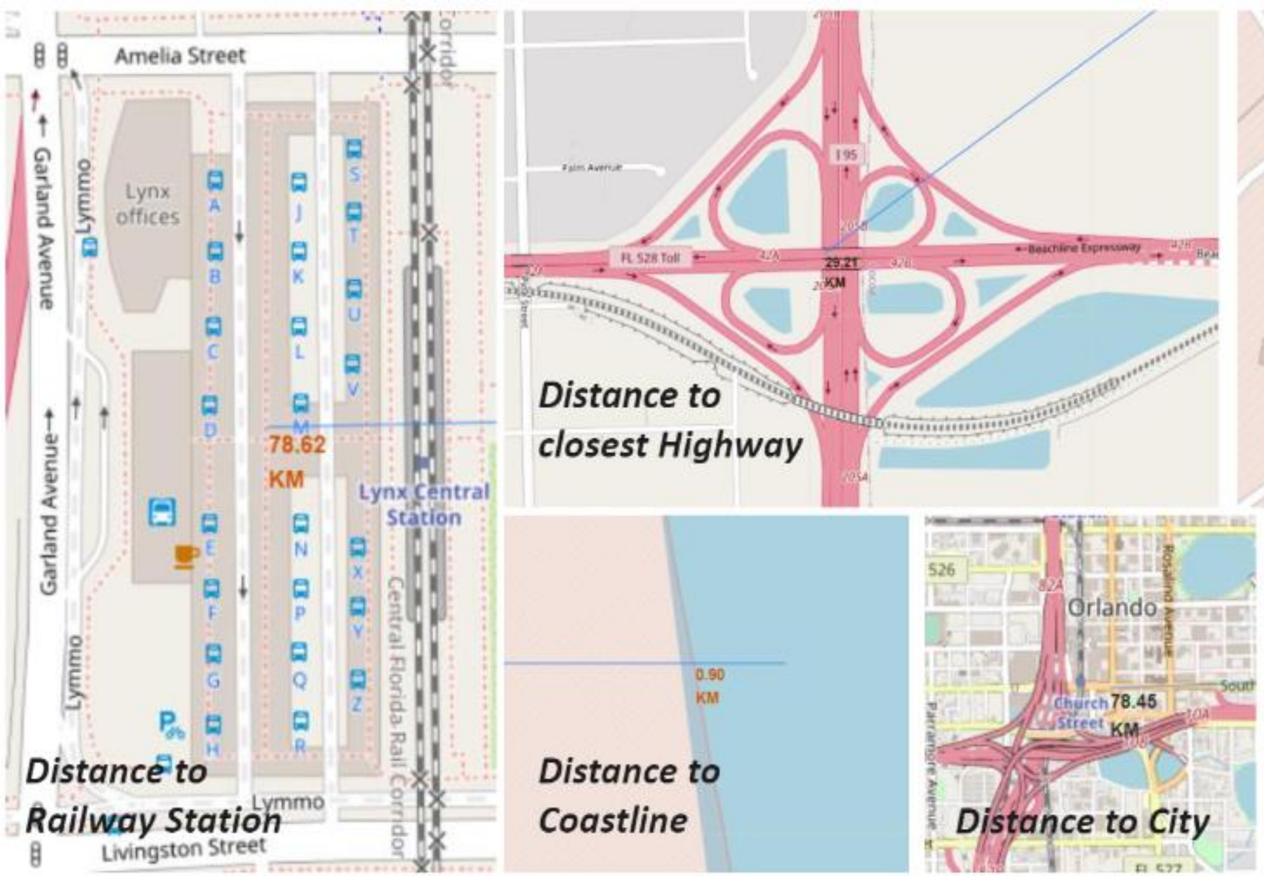
- Launch Sites are near the coastal areas of the United States, in the west and in the east.



Markers showing Launch Sites



Launch Sites proximity to landmarks

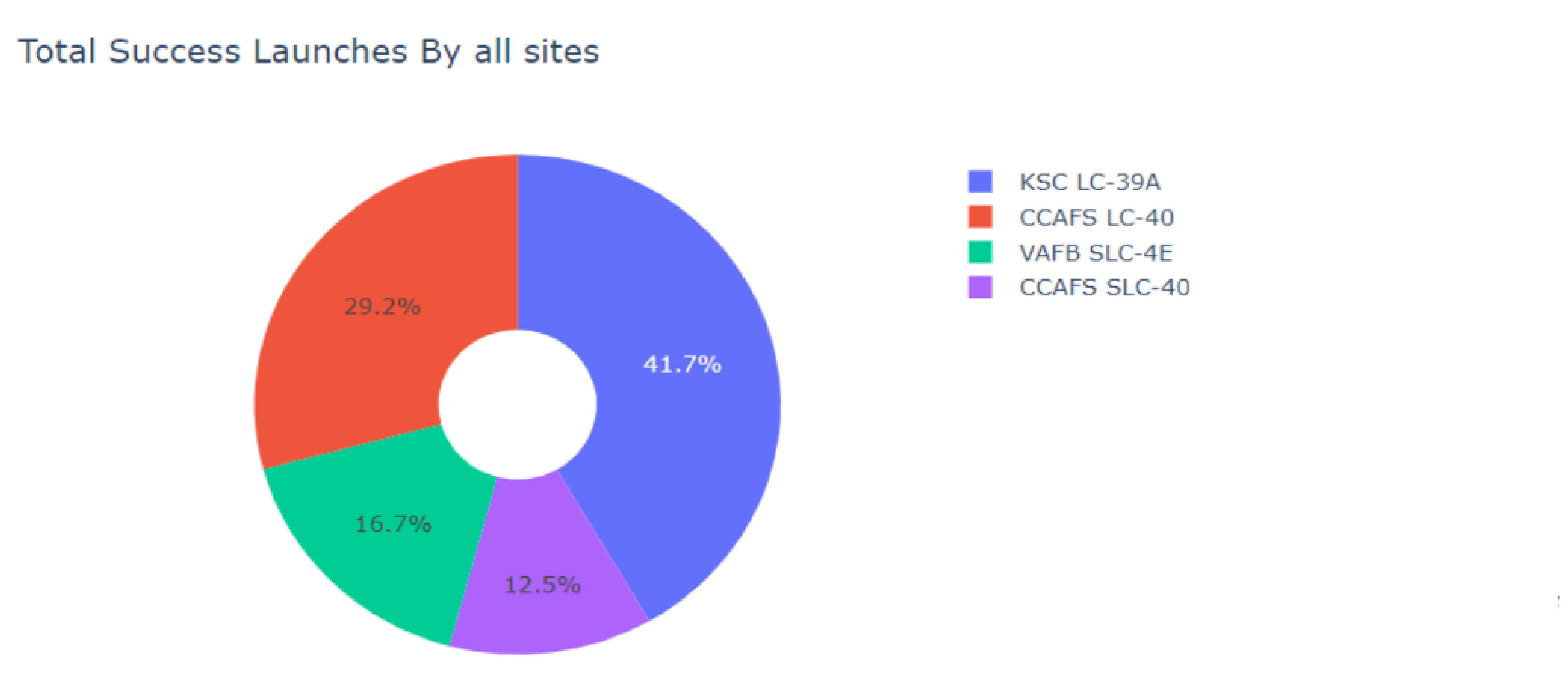


- Are launch sites in close proximity to railways? No
- Are launch sites in close proximity to highways? No
- Are launch sites in close proximity to coastline? Yes
- Do launch sites keep certain distance away from cities? Yes

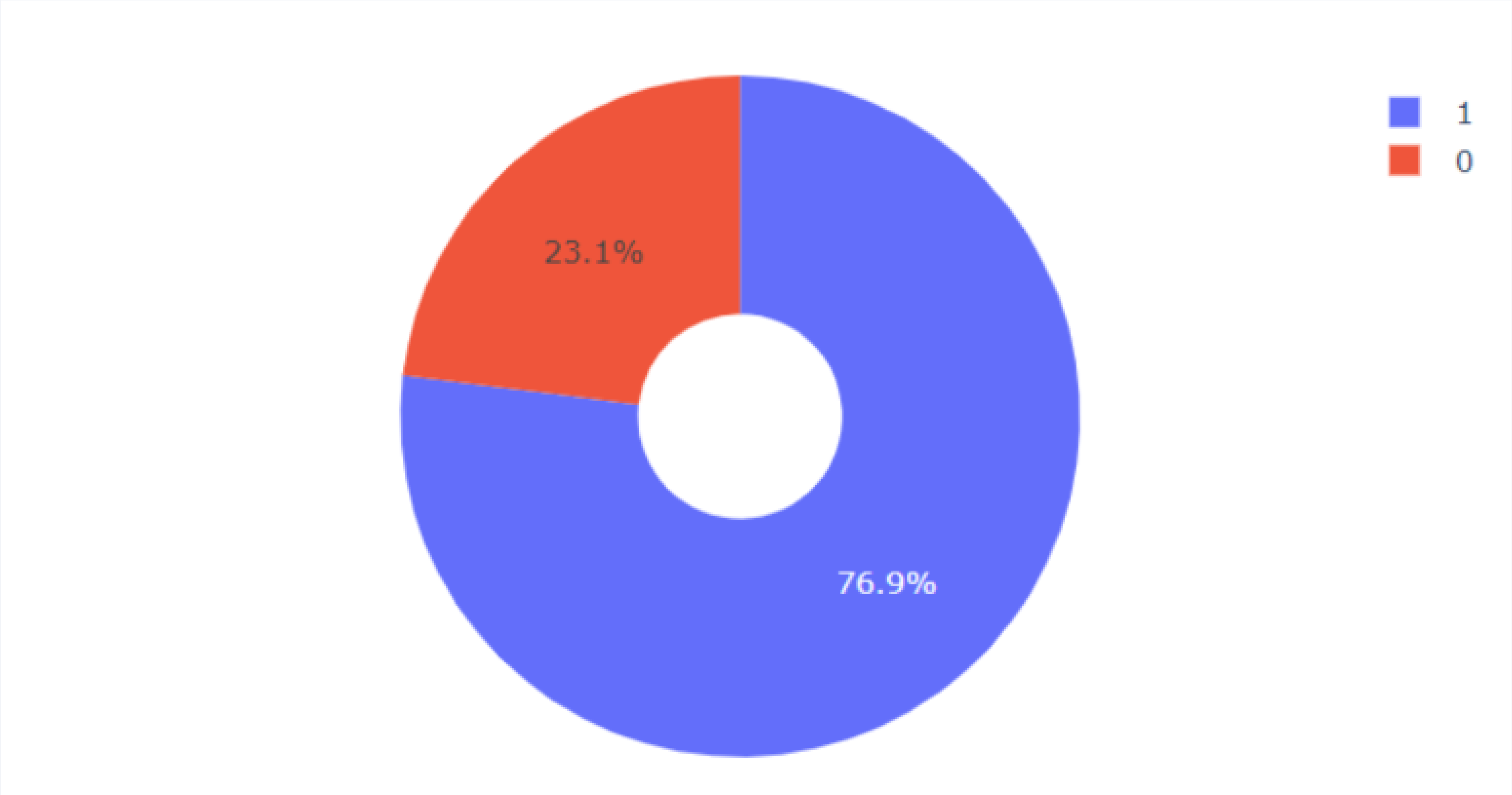
Section 4

Build a Dashboard with Plotly Dash

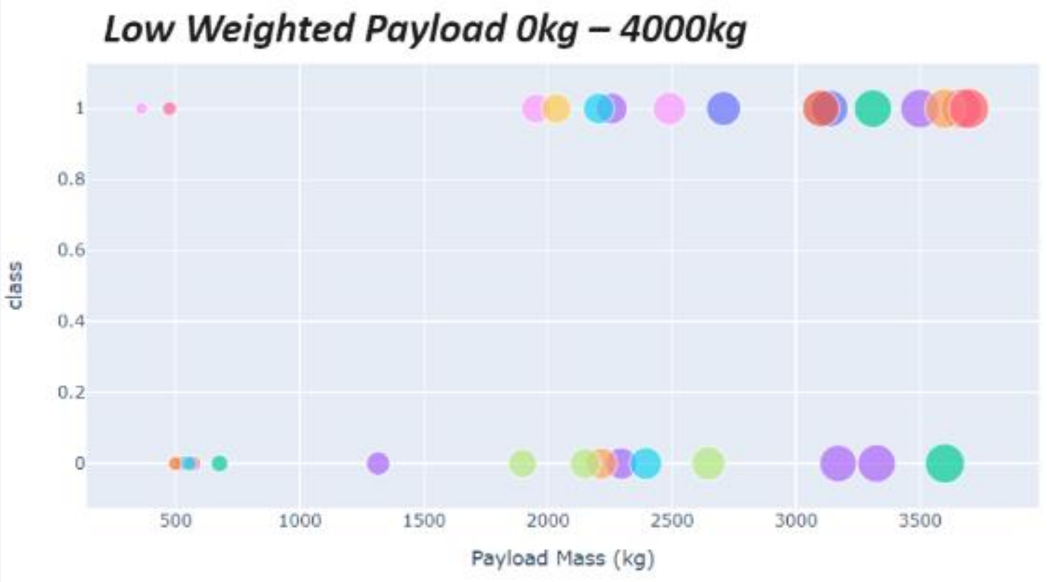
Launch Sites portion of successful; launches

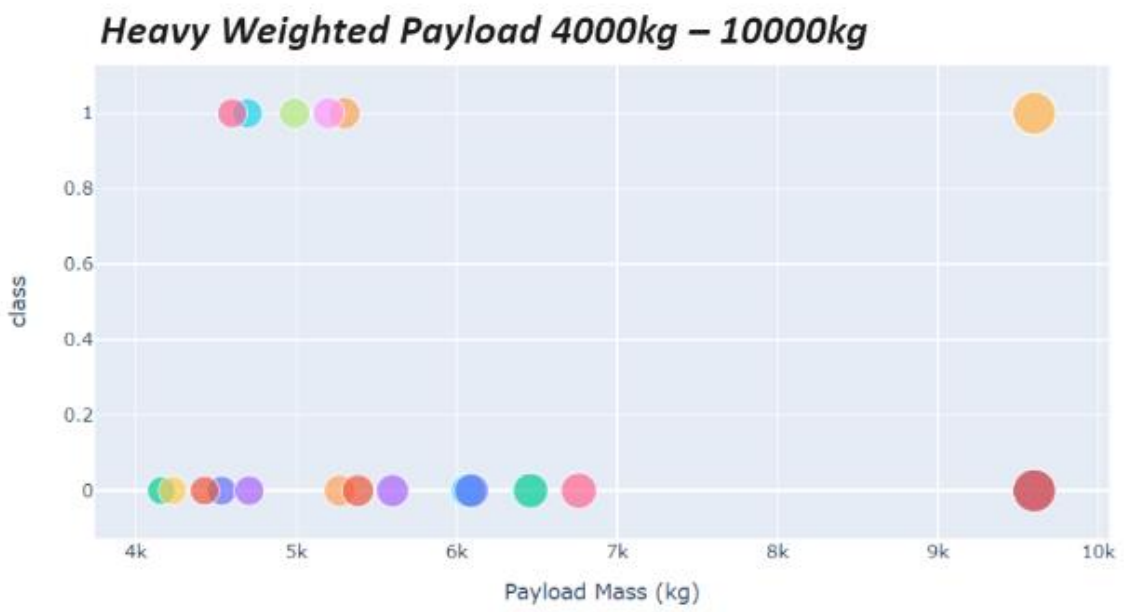


Launch Site with the highest success rate: KSC LC-39A



Low and heavy weighted payload against launch outcome





Section 5

Predictive Analysis (Classification)

Classification Accuracy

Decision Tree has the best Classification Accuracy, as can be seen below using the code shown

TASK 12

Find the method performs best:

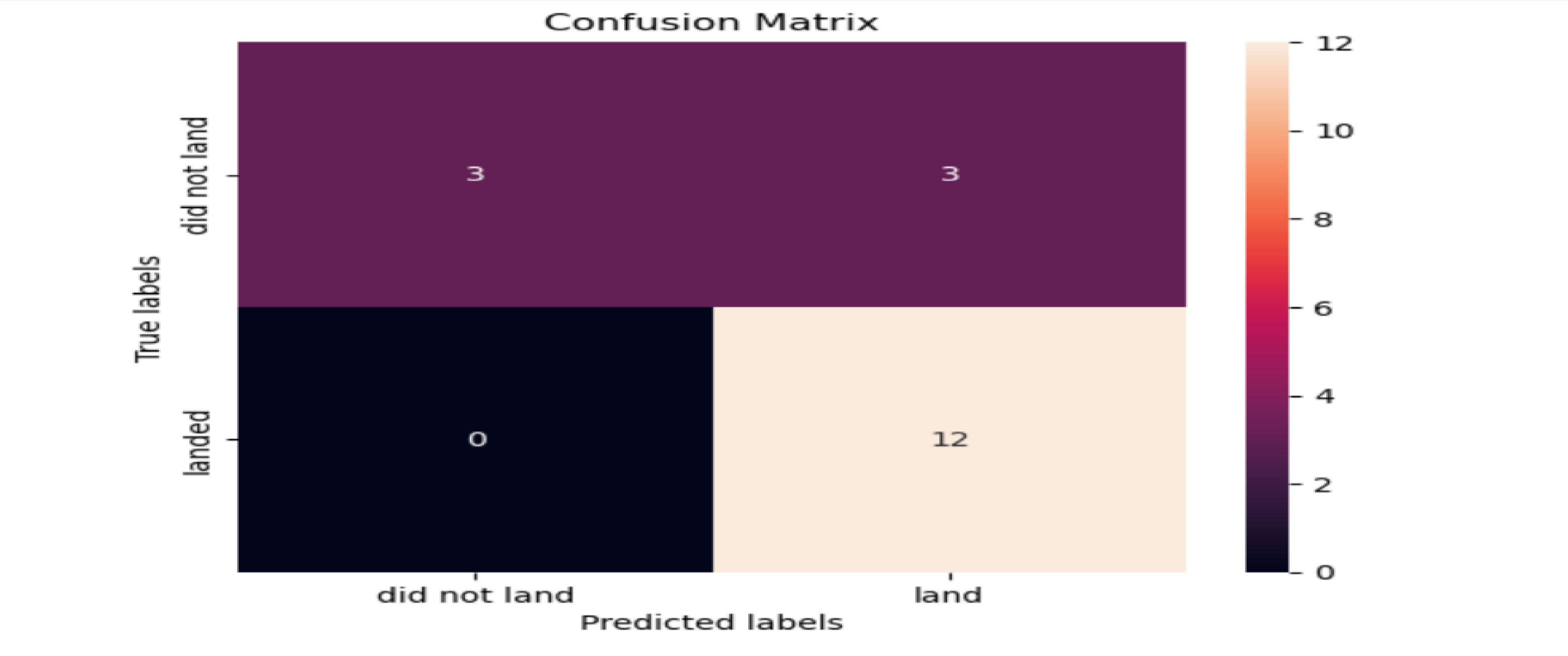
In [35]:

```
print('Accuracy for Logistics Regression method:', logreg_cv.score(X_test, Y_test))
print( 'Accuracy for Support Vector Machine method:', svm_cv.score(X_test, Y_test))
print('Accuracy for Decision tree method:', tree_cv.score(X_test, Y_test))
print('Accuracy for K nearst neighbors method:', knn_cv.score(X_test, Y_test))
```

```
Accuracy for Logistics Regression method: 0.8333333333333334
Accuracy for Support Vector Machine method: 0.8333333333333334
Accuracy for Decision tree method: 0.8888888888888888
Accuracy for K nearst neighbors method: 0.8333333333333334
```

Confusion Matrix

- The confusion matrix of the best performing model with an explanation - the major problem is the false positives .i.e., unsuccessful landing marked as successful landing by the classifier.



Conclusions

- Over time, success rates for Space X's launches seem to be improving
- The following orbits the the highest success rates: ES-L1, GEO, HEO and SSO
- KSC LC-39A had the most successful launches of any sites.
- For this data, The Decision tree classifier is the best machine learning algorithm.

Appendix

- # Requests allows us to make HTTP requests which we will use to get data from an API
- import requests
- # Pandas is a software library written for the Python programming language for data manipulation and analysis.
- import pandas as pd
- # NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays
- import numpy as np
- # Datetime is a library that allows us to represent dates
- import datetime

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

```
# 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’])
```

```
# 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'])
```

```
# 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’])

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

```
# 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’])

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

```
# 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'])

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

```
# 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’])

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

```
# 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’])

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

```
# 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’])

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

```
# 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’])

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

```
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}
```

Thank you!