IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Ana Hemani
02/04/2025

Outline

Executive Summary

- Summary of methodologies
  - Data Collection with API
  - Data Collection with Web Scraping
  - Data Wrangling
  - Exploratory Data Analysis with SQL
  - Exploratory Data Analysis with Data Visualization
  - Interactive Visual Analytics with Folium
  - Machine Learning Prediction
- Summary of all results
  - Exploratory Data Analysis result
  - Interactive analytics in screenshots
  - Predictive Analytics result

Introduction

- Project background and context

  SpaceX advertises Falcon 9 rocket launches on its website at a cost of $62 million, whereas other providers charge upwards of $165 million per launch. A significant portion of these savings comes from SpaceX's ability to reuse the first stage of the rocket. Therefore, predicting whether the first stage will successfully land can help determine the cost of a launch. This information is valuable for alternative companies looking to compete with SpaceX in the rocket launch market. The objective of this project is to develop a machine learning pipeline that predicts the successful landing of the first stage.

- Problems you want to find answers to
  - What factors influence the successful landing of a rocket?
  - The interplay of various features contributes to the likelihood of a successful landing.
  - What operating conditions must be met to ensure a reliable landing program?

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
    - Data was collected using SpaceX API and web scraping from Wikipedia.
- Perform data wrangling
    - One-hot encoding was applied to categorical features
- 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 was collected using numerous methods
  - Data collection was obtained using get request to the SpaceX API.
  - The response content was then decoded as a Json using .json() function call and then moved to Pandas dataframe using.json_normalized().
  - Data was then cleaned, checked for missing values, and filled in missing values as needed.
  - Next, we executed web scraping from Wikipedia for Falcon 9 launch records via BeautifulSoup.
  - Lastly, the objective was to extract the launch records as a HTML table, parse the table, and then convert it into a Pandas dataframe for future analysis.

Data Collection - SpaceX API

- Used the GET request to the SpaceX API to obtain data
- Cleaned the requested data
- Sorted out some basic data wrangling and formatting

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)
```

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

```
[12]: # Use json_normalize method to convert the json result into a dataframe

      # decode response content as json
      static_json_df = res.json()
```

```
[13]: # apply json_normalize
      data = pd.json_normalize(static_json_df)
```

Calculate below the mean for the `PayloadMass` using the `.mean()`. Then use the mean and the `.replace()` function to replace `np.nan` values in the data with the mean you calculated.

```
[28]: # Calculate the mean value of PayloadMass column
      PayloadMass = pd.DataFrame(data_falcon9['PayloadMass'].values.tolist()).mean(1)
      print(PayloadMass)
```

```
0    5919.165341
dtype: object
```

```
[32]: # Extract the first row's 'PayloadMass' as a list
      rows = data_falcon9.loc[0, 'PayloadMass']

      # Ensure it's a NumPy array
      rows = np.array(rows)

      # Reshape into a 2D format if necessary
      rows = rows.reshape(-1, rows.shape[-1]) if rows.ndim == 3 else rows.reshape(-1, 1)

      # Convert to a DataFrame
      df_rows = pd.DataFrame(rows)

      # Replace NaN values with a default value
      default_value = 0  # Set to a suitable replacement value
      df_rows.fillna(default_value, inplace=True)

      # Assign the updated values back to the DataFrame
      data_falcon9.at[0, 'PayloadMass'] = df_rows.values

      # Display the updated DataFrame
      display(data_falcon9)
```

Data Collection - Scraping

- Applied web scraping to web scrap Falcon 9 launch records via BeautifulSoup
- Parsed the table
- Converted the table into a Panda dataframe

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

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

```
[5]: 200
```

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

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

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

```
[7]: # Use soup.title attribute
     soup.title
```

```
[7]: <title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

Next, we just need to iterate through the `<th>` elements and apply the provided `extract_column_from_header()` to extract column name one by one

```
[10]: 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
```

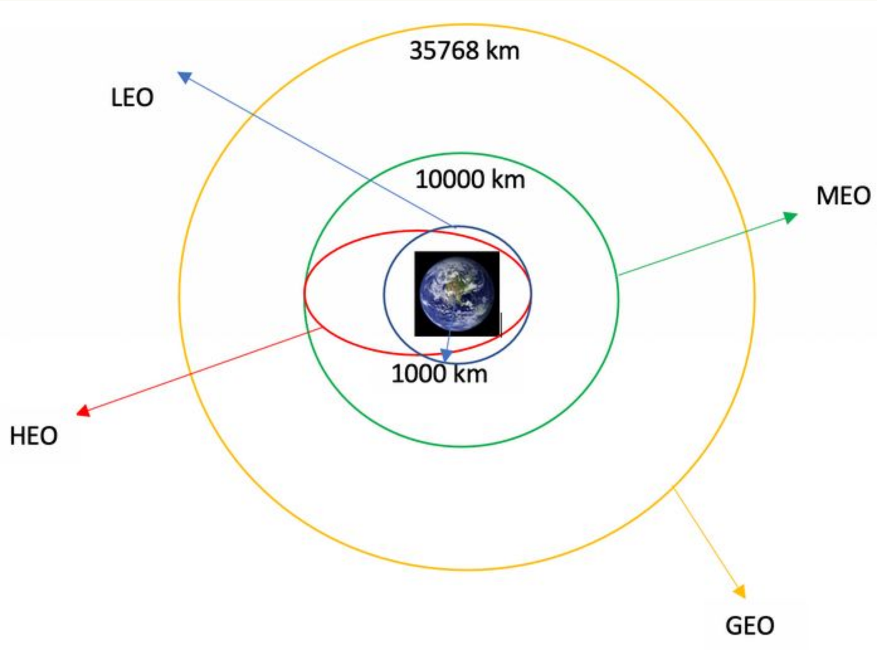
Check the extracted column names

```
[11]: print(column_names)
```

```
['Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome', 'Flight No.', 'Date and time ( )', 'L
aunch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome', 'Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass'
, 'Orbit', 'Customer', 'Launch outcome', 'Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome
', 'N/A', 'Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome', 'Flight No.', 'Date and time
( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome', 'Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Paylo
ad mass', 'Orbit', 'Customer', 'Launch outcome', 'FH 2', 'FH 3', 'Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Cu
stomer', 'Launch outcome', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome', 'Date and time ( )', 'Lau
nch site', 'Payload', 'Orbit', 'Customer', 'Date and time ( )', 'Launch site', 'Payload', 'Orbit', 'Customer', 'Date and time ( )', 'Launch site', 'Payloa
d', 'Orbit', 'Customer', 'Date and time ( )', 'Launch site', 'Payload', 'Orbit', 'Customer', 'Demonstrations', 'logistics', 'Crewed', 'Commercial satellit
es', 'Scientific satellites', 'Military satellites', 'Rideshares', 'Transporter', 'Bandwagon', 'Flight tests', 'Crewed', 'Commercial satellites', 'Current
', 'In development', 'Retired', 'Cancelled', 'Spacecraft', 'Cargo', 'Crewed', 'Test vehicles', 'Current', 'Retired', 'Unflown', 'Orbital', 'Atmospheric',
'Landing sites', 'Other facilities', 'Support', 'Contracts', 'R&D programs', 'Key people', 'Related', 'General', 'General', 'People', 'Vehicles', 'Launche
s by rocket type', 'Launches by spaceport', 'Agencies, companies and facilities', 'Other mission lists and timelines']
```

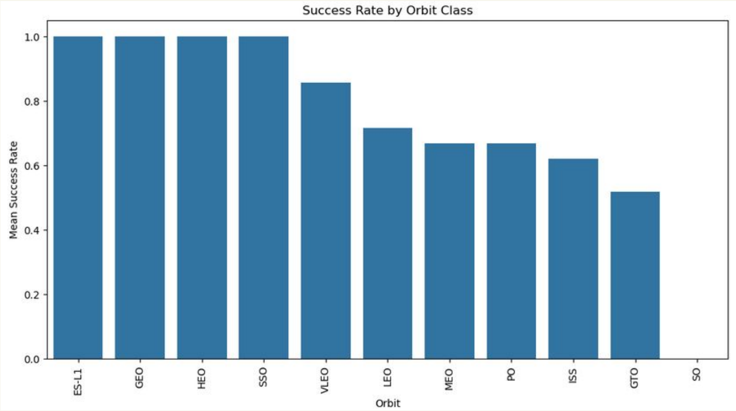
Data Wrangling

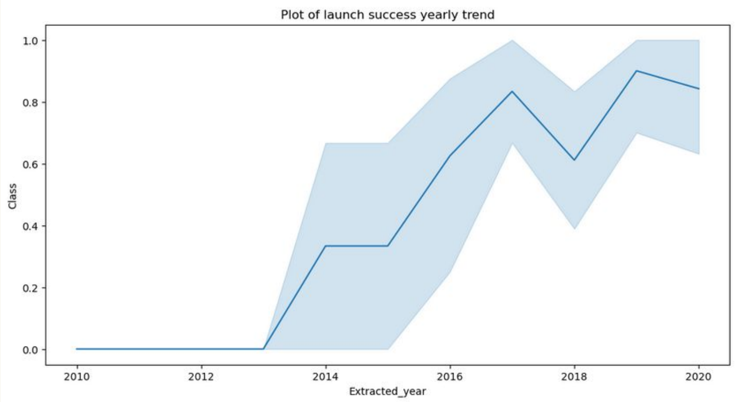
- Executed exploratory data analysis
- Defined the training labels
- Calculated the number of launches at each site and the number and occurrence of each orbit
- Created landing outcome label from outcome column
- Exported the results to CSV



EDA with Data Visualization

- Explored the data via envisioning the relationship between
  - Flight number and launch site
  - Payload and site
  - Success rate of each orbit type
  - Flight number and orbit type
  - The launch success yearly trend





EDA with SQL

- Loaded the SpaceX dataset into a PostgreSQL database without existing the Jupyter Notebook
- Applied EDA via SQL to gain insight from the data
- Wrote queries to find the following instances:
    - Names of the 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 v 1.1
    - Total number of successful and failed mission outcomes
    - Failed landing outcomes in drone ship, booster version, and launch the site names.

Build an Interactive Map with Folium

- Marked all launch sites
- 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 (success/failure) to class 0 and 1
    - I.e., 0 = failure and 1 = success
- Per the color-labeled marker clusters
    - Identified which launch sites have relatively high success rate
- Calculated the distances between a launch site to its proximities
- Answered some questions:
    - Are launch sites near railways, highways, and coastlines?
    - Do launch sites keep certain distance away from cities?

Build a Dashboard with Plotly Dash

- Built an interactive dashboard via Plotly Dash
- Plotted pie charts showing the total launches by a certain site
- Plot scatter graph indicating the relationship with Outcome and Payload Mass (kg) for the different booster version

Predictive Analysis (Classification)

- Loaded the data via NumPy and Pandas
- Transformed the data
- Split data into training and testing
- Built various machine learning models and tuned different hyperparameters via GridSearchCV
- Used accuracy as the metric for the model
- Improved the model using feature engineering and algorithm tuning
- Established the best performing classification model

Results

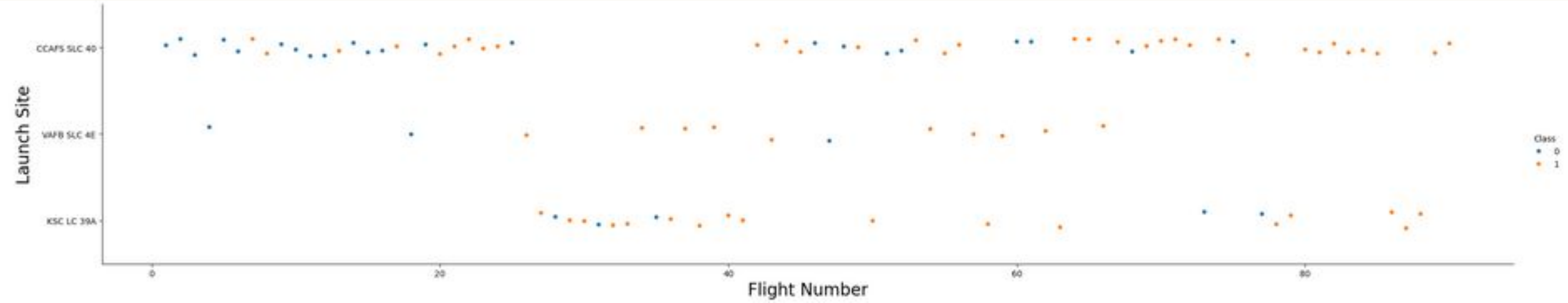
- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results

Section 2

Insights drawn from EDA

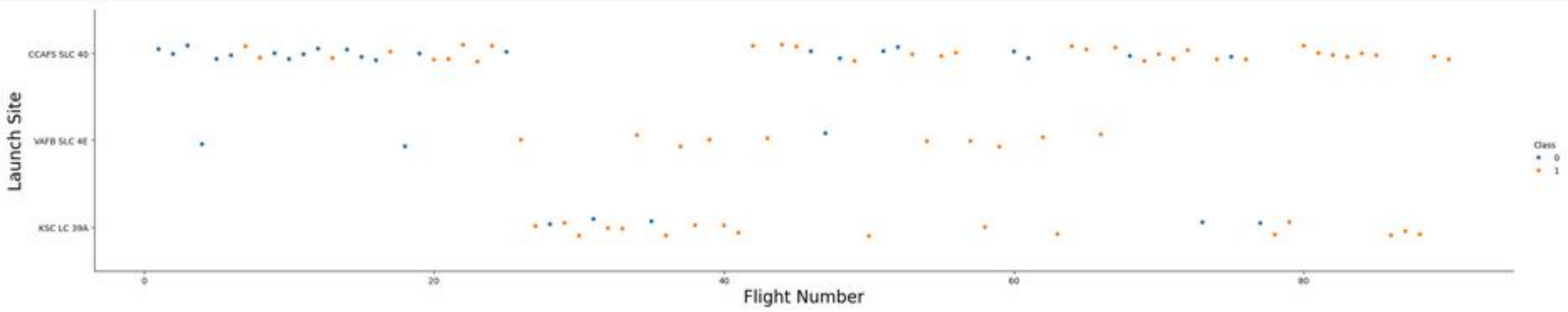
Flight Number v Launch Site

- Found that the larger the flight amount at a launch site, the greater the success rate



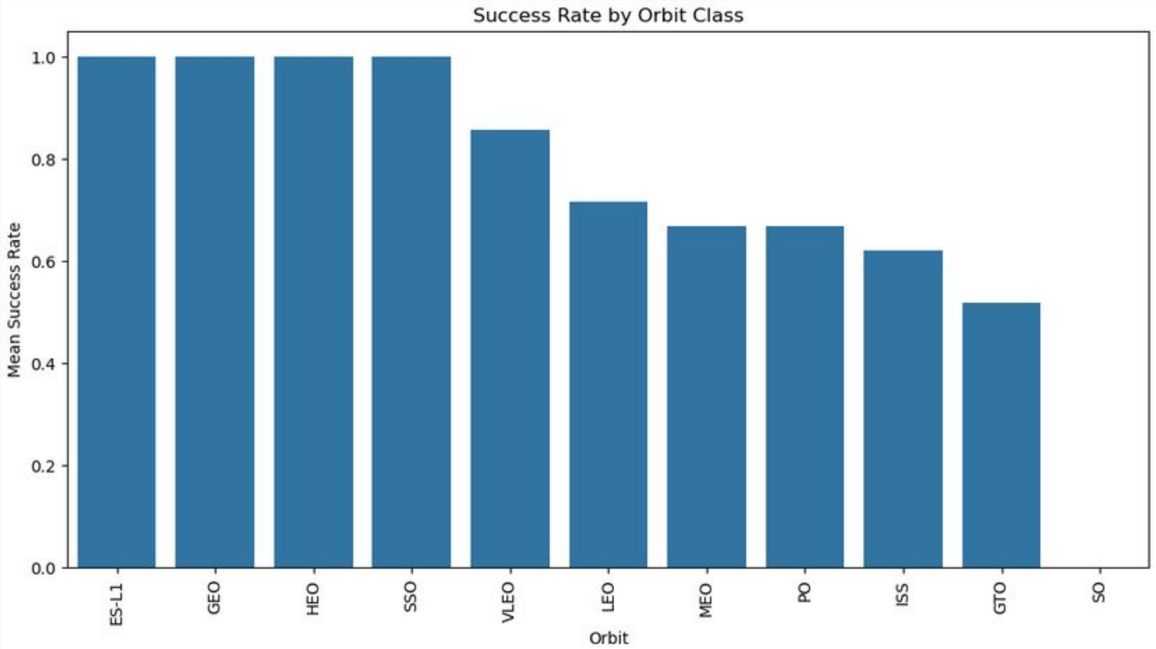
Payload v Launch Site

- Greater the payload mass for launch site CCAFS SLC 40, the higher the success rate for the rocket



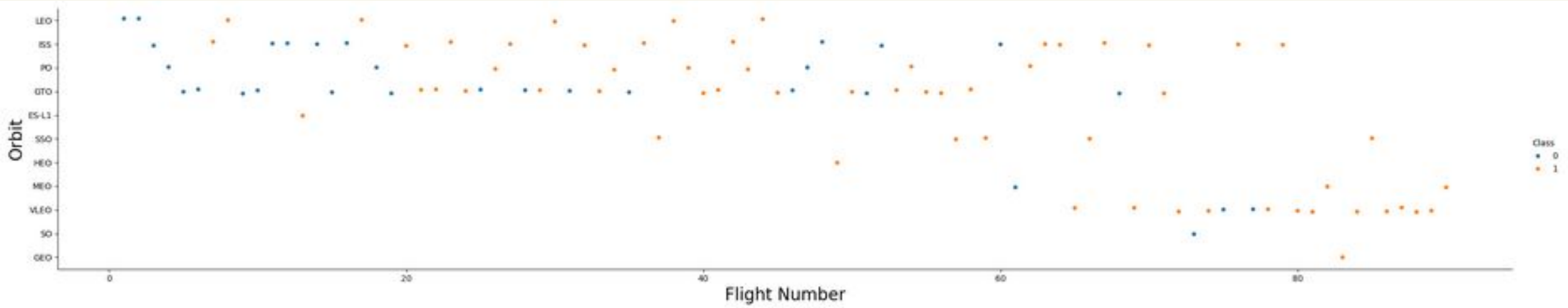
Success Rate v Orbit Type

- ES-L 1, GEO, HEO, SSO, VLEO had the most success rates



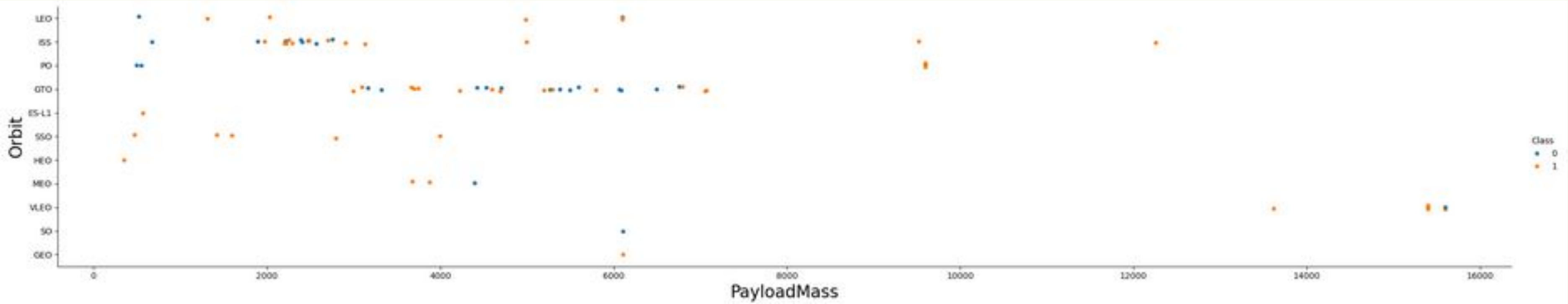
Flight Number v Orbit Type

- Observed in the LEO orbit, success is related to the number of flights
- GTO orbit shows no relationship between flight number and the orbit



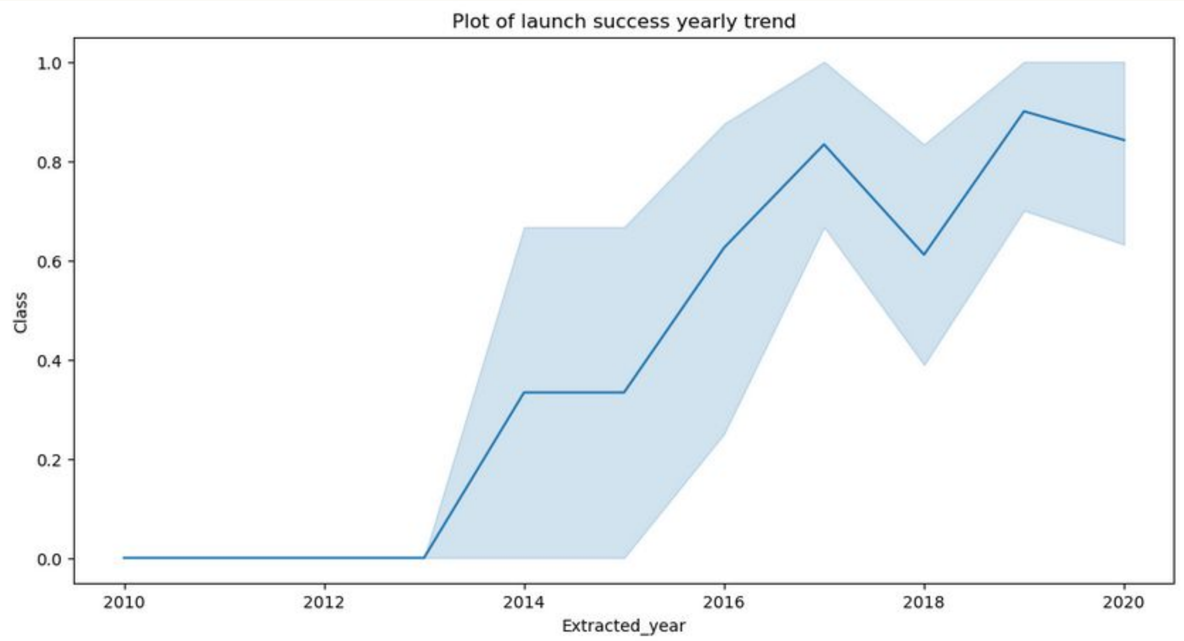
Payload v Orbit Type

- Observed the successful landing are more for PO, LEO, ISS orbits



Launch Success Yearly Trend

- Success rate since 2013 continued to increase until 2020



All Launch Site Names

- Used the keyword DISTINCT to display only distinctive launch sites from the SpaceX data

[10]:
```
task_1 = '''
        SELECT DISTINCT LaunchSite
        FROM SpaceX
'''
create_pandas_df(task_1, database=conn)
```

[10]:

	launchsite
0	KSC LC-39A
1	CCAFS LC-40
2	CCAFS SLC-40
3	VAFB SLC-4E

Launch Site Names Begin with ‘CCA’

- Displayed 5 records where launch sites start with ‘CCA’

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

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

[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

- Calculated the total payload carried by boosters from NASA as 45596

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

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

[12]:

	total_payloadmass
0	45596

Average Payload Mass by F9 v 1.1

- Calculated the average payload mass carried by booster version F9 v 1.1 as 2928.4

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

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

[13]:

	avg_payloadmass
0	2928.4

First Successful Ground Landing Date

- Observed that the dates of the first successful landing outcome on ground pad was December 22, 2015

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

Hint:Use min function

[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)
```

[14]:

	firstsuccessfull_landing_date
0	2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

- Used the WHERE clause to sift for boosters which have positively landed on drone ship
- Applied the AND condition to define successful landing with payload mass more 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

[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)
```

[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

- Used wildcard like '%' to sift for WHERE MissionOutcome was a success or failure

List the total number of successful and failure mission outcomes

```
[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:
```

[16]:

	failureoutcome
0	1

Boosters Carried Maximum Payload

- Defined the booster which has 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

[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)
```

[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

- Used a combination of the WHERE clause, LIKE, AND, and BETWEEN condition to sift for negative landing outcomes in drone ship, their booster versions, and launch sites names for 2015

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

[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)
```

[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 06/10/2010 and 03/20/2017

- Selected landing outcomes, the COUNT of landing outcomes from the data, and used the WHERE clause to sift for landing outcomes BETWEEN 06/04/2010 to 03/20/2010
- Applied the GROUP BY clause to group the landing outcomes and the ORDER BY clause to order the grouped landing outcomes in 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

[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)
```

[19]:

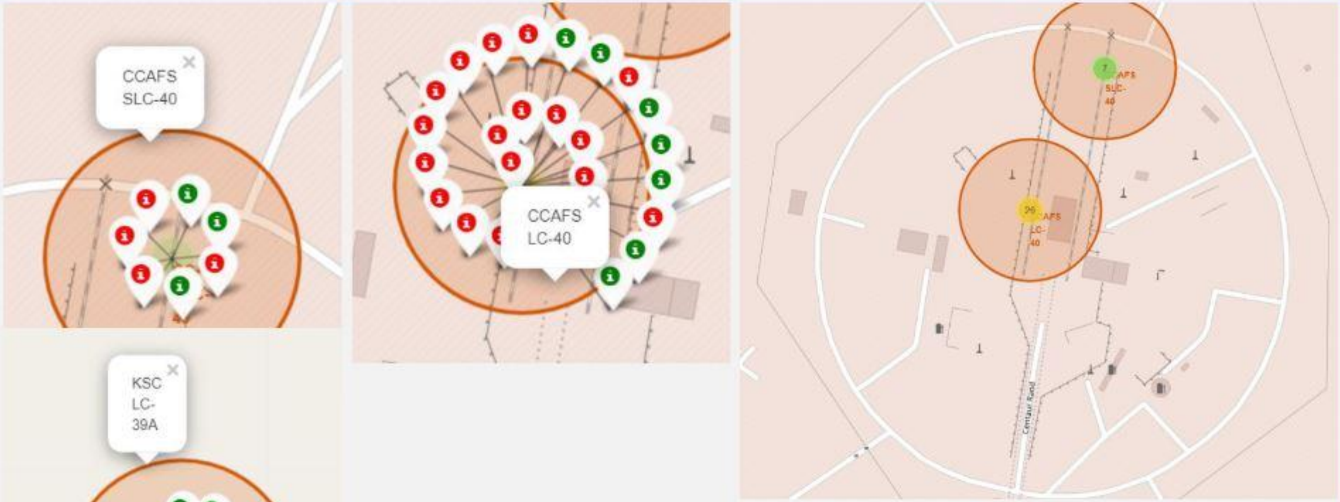
	landingoutcome	count
0	No attempt	10
1	Success (drone ship)	6
2	Failure (drone ship)	5
3	Success (ground pad)	5
4	Controlled (ocean)	3
5	Uncontrolled (ocean)	2
6	Precluded (drone ship)	1
7	Failure (parachute)	1

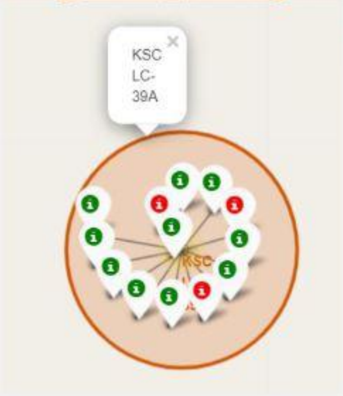
Section 3

Launch Sites Proximities Analysis

All launch sites globally

Markers indicating launch sites with colored labels



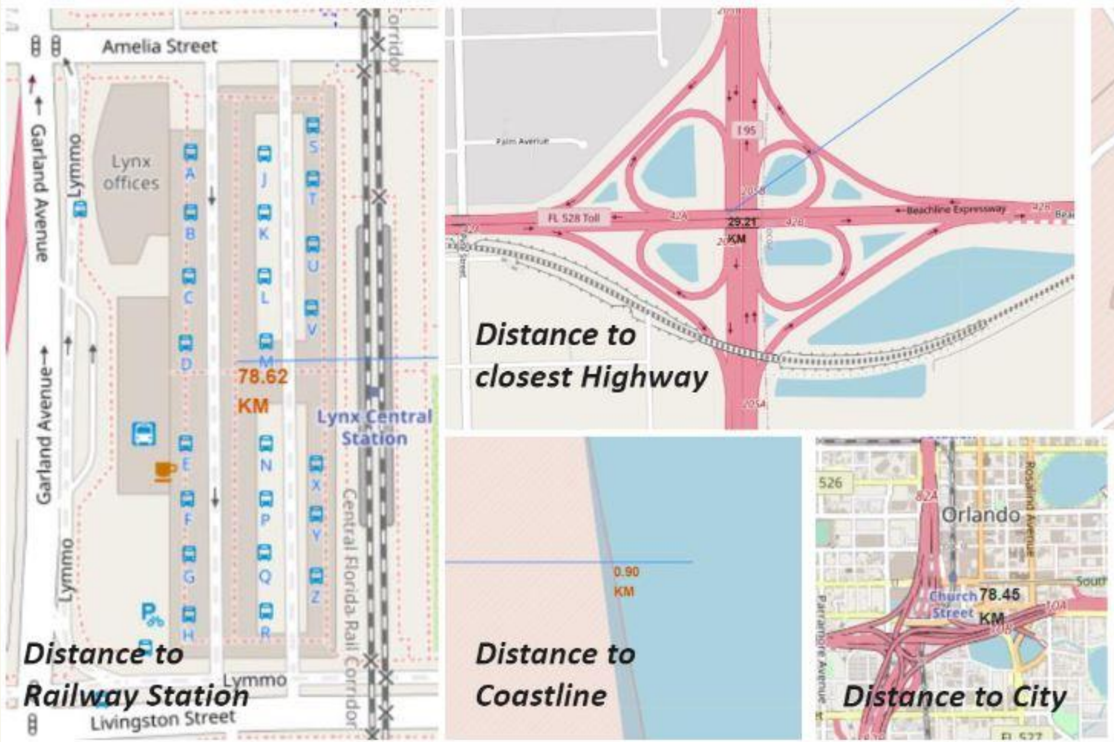


Florida Launch Sites

Green Marker shows successful Launches and Red Marker shows Failures

California Launch Site

Launch sites distance to landmarks

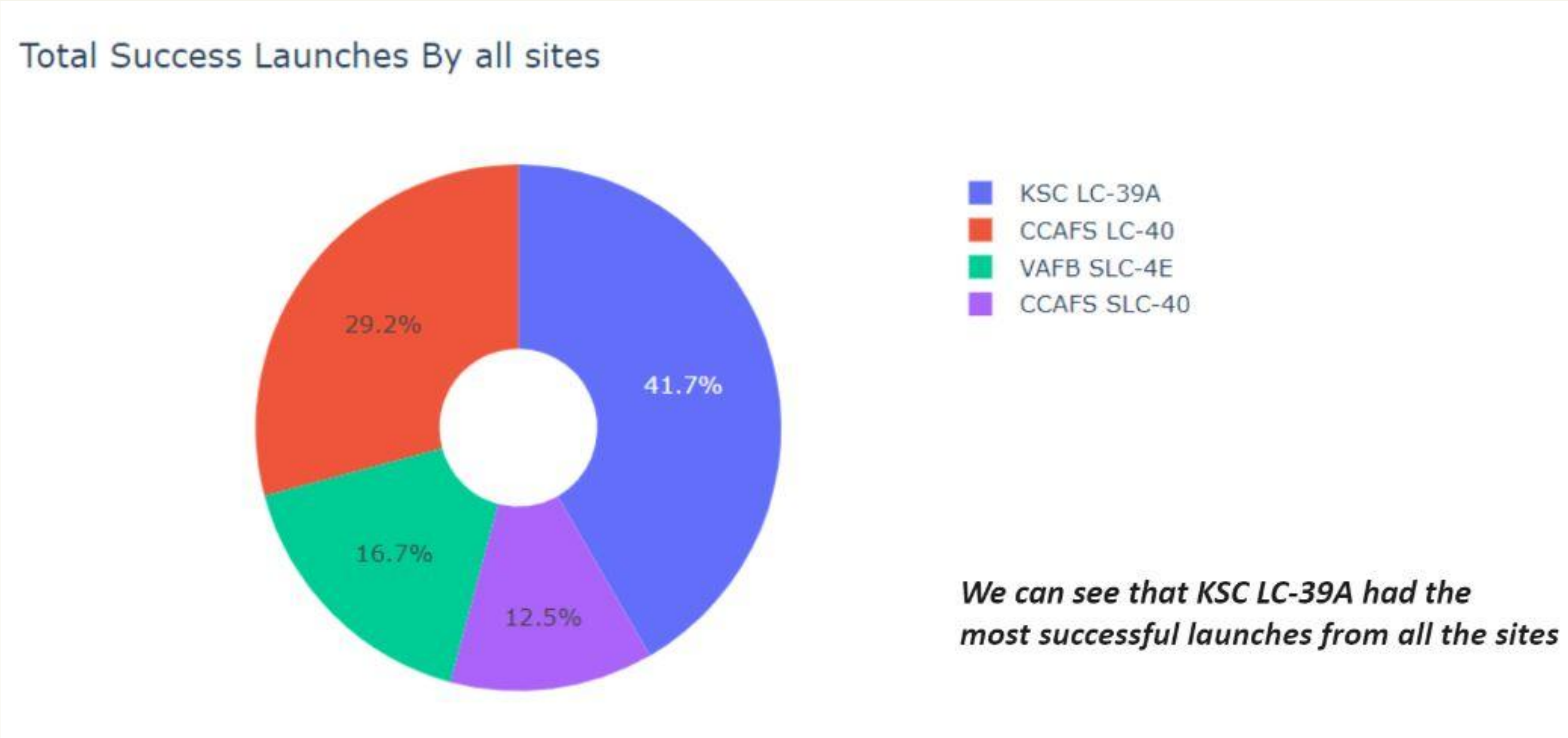


- Are launch sites in close proximity to railways? No
- Are launch sites in close proximity to highways? No
- Are launch sites in close proximity to coastline? Yes
- Do launch sites keep certain distance away from cities? Yes

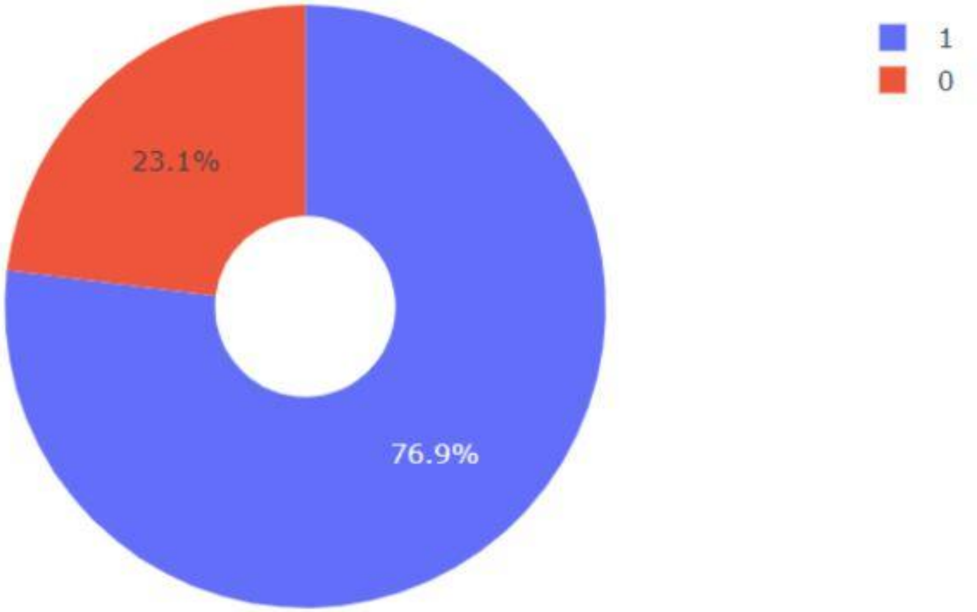
Section 4

Build a Dashboard with Plotly Dash

Pie chart indicating the success % achieved by each launch site



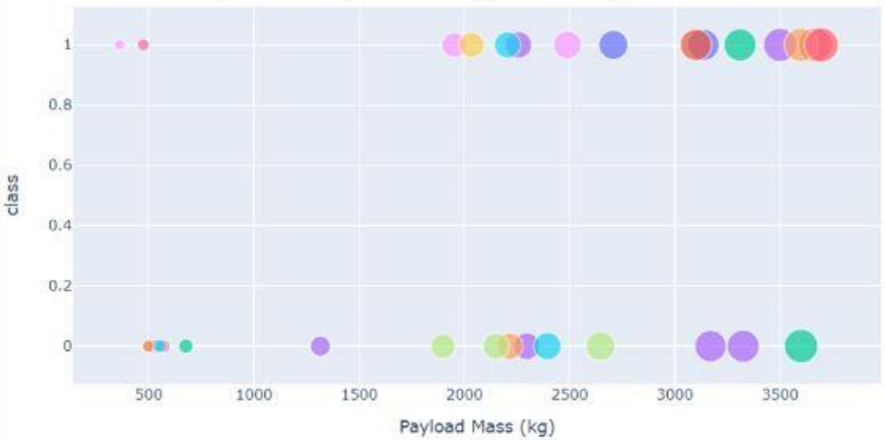
Pie chart indicating the launch site with the most launch success ratio



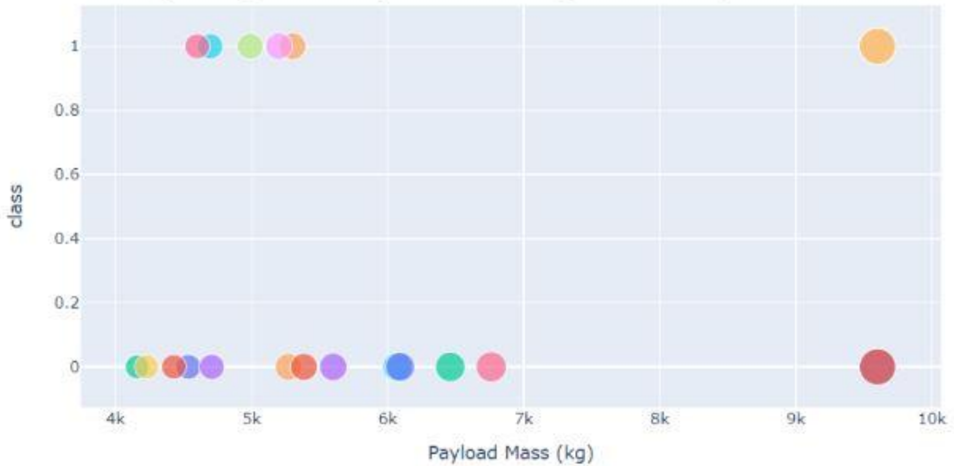
KSC LC-39A achieved a 76.9% success rate while getting a 23.1% failure rate

Scatter plot of Payload v Launch Outcome for all sites with various payload designated in the range slider

Low Weighted Payload 0kg – 4000kg



Heavy Weighted Payload 4000kg – 10000kg



We can see the success rates for low weighted payloads is higher than the heavy weighted payloads

Section 5

Predictive Analysis (Classification)

Classification Accuracy

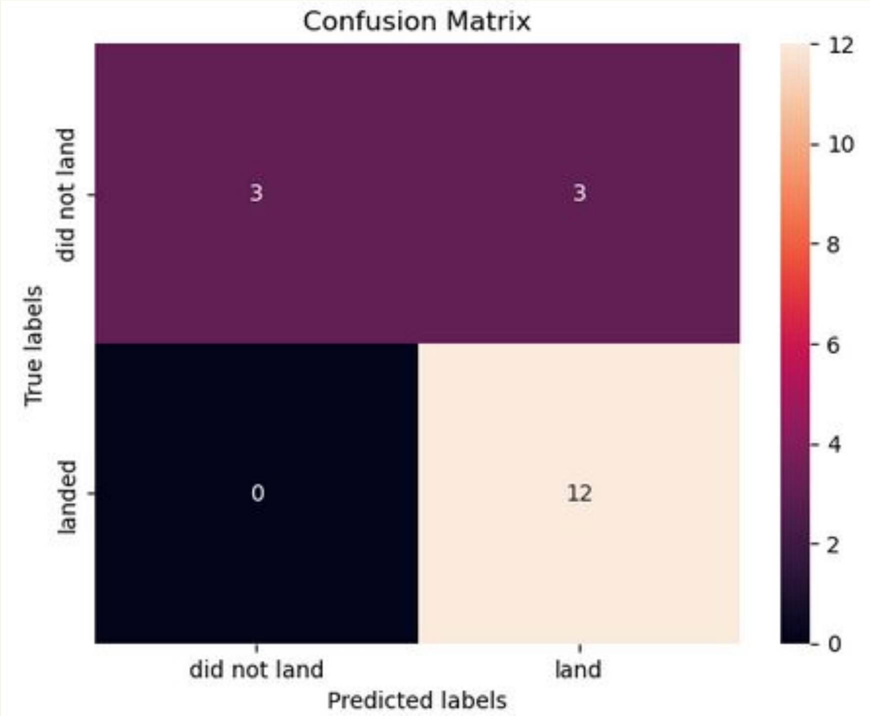
```
[25]: models = {'KNeighbors':knn_cv.best_score_,
                'DecisionTree':tree_cv.best_score_,
                'LogisticRegression':logreg_cv.best_score_,
                'SupportVector': svm_cv.best_score_}

      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': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 5, 'splitter': 'random'}
```

Confusion Matrix

- Decision tree classifier indicates 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

- The larger the flight amount as a launch site, the bigger the success rate at a launch site
- Launch success rate started to rise in 2013 until 2020
- Orbits ES-1, GEO, HEO, SSO, VLEO has the most success rate
- KSC LC-39A has the most successful launch of any sites
- Decision tree classifier is the finest machine learning algorithm for this task