IBM

This is my capstone project from the IBM Data Science Professional Certificate. It involved applying **data science methodology**, various **Python libraries** (NumPy, Pandas, Matplotlib, Seaborn, Folium, Scikit-Learn), **SQL**, **machine learning**, and **report writing**.

The project aimed to predict the **first stage landing success rate** for the **SpaceX Falcon 9 rocket**. Key steps included **data collection** (from the SpaceX public API and Wikipedia), **cleaning**, **analysis**, and **visualization** (through static plots, interactive maps, and a dashboard).

I trained **Logistic Regression, SVM, Decision Tree, and KNN machine learning models** to predict future landing outcomes. **Logistic Regression, SVM, and KNN models performed equally well** on this dataset.

Humberto Hernandez Renteria

coursera

Report Structure

- Overview & Key Insights (Executive Summary)
- Background & Purpose (Introduction)
- Approach & Methods (Methodology)
- Findings & Analysis (Visual Analytics)
- Insights & Recommendations
- Supporting Documents

Overview & Key Insights

- **Methodology Overview**
  - Data Collection via API
  - Web Scraping Techniques
  - Data Wrangling & Cleaning
  - Exploratory Data Analysis (EDA) using SQL
  - Data Visualization for EDA
  - Interactive Geospatial Analysis with Folium
  - Predictive Modeling with Machine Learning
- **Results Summary**
  - Key Findings from Exploratory Analysis
  - Screenshots of Interactive Visualizations
  - Outcomes from Predictive Analytics

Background & Purpose

• **Project Background and Context**

SpaceX advertises Falcon 9 rocket launches at a cost of $62 million, while other providers charge upwards of $165 million per launch. A significant portion of the cost savings is due to SpaceX's ability to reuse the first stage of the rocket. Therefore, if we can predict whether the first stage will land successfully, we can estimate the overall cost of a launch. This insight can be valuable for competing companies aiming to bid against SpaceX for launch contracts. The primary objective of this project is to develop a machine learning pipeline capable of predicting the success of first-stage landings.

• **Key Research Questions**

- What factors influence the likelihood of a successful rocket landing?
- How do different features interact to impact landing success rates?
- What operational conditions are necessary to optimize the success of the landing program?

Approach & Methods

Executive Summary

- **Data Collection:** Acquired through the SpaceX API and web scraping from Wikipedia.
- **Data Preparation:** Applied data wrangling techniques, including one-hot encoding of categorical variables.
- **Exploratory Data Analysis (EDA):** Conducted using SQL queries and data visualizations to uncover patterns and insights.
- **Interactive Analytics:** Developed visual tools using Folium and Plotly Dash to enable dynamic exploration of the data.
- **Predictive Modeling:** Built and evaluated classification models to predict first-stage landing success.
- **Model Optimization:** Focused on tuning and assessing model performance to ensure accuracy and reliability.

Data Collection Process

- The data was gathered using multiple methods:
- Data was retrieved through GET requests to the SpaceX API.
- The API response was decoded as JSON using the .json() method and converted into a pandas DataFrame with json_normalize().
- The dataset was then cleaned by checking for and handling missing values appropriately.
- Additionally, web scraping was performed using BeautifulSoup to extract Falcon 9 launch records from Wikipedia.
- The objective was to retrieve the HTML launch table, parse its contents, and convert it into a pandas DataFrame for further analysis.

Space X API

We used GET requests to access data from the SpaceX API, followed by data cleaning, basic wrangling, and formatting to prepare it for analysis.

1. Get request for rocket launch data using API

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

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

2. Use json_normalize method to convert json result to dataframe

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

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

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

3. We then performed data cleaning and filling in the missing values

In [30]:
```
rows = data_falcon9['PayloadMass'].values.tolist()[0]

df_rows = pd.DataFrame(rows)
df_rows = df_rows.replace(np.nan, PayloadMass)

data_falcon9['PayloadMass'][0] = df_rows.values
data_falcon9
```

Web Scraping

We used web scraping with BeautifulSoup to extract Falcon 9 launch records from a webpage. The launch table was parsed and converted into a pandas DataFrame for analysis.

1. Apply HTTP Get method to request the Falcon 9 rocket launch page

In [4]:
```
static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"
```

In [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
```

Out[5]:
```
200
```

2. Create a BeautifulSoup object from the HTML response

In [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

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

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

3. Extract all column names from the HTML table header

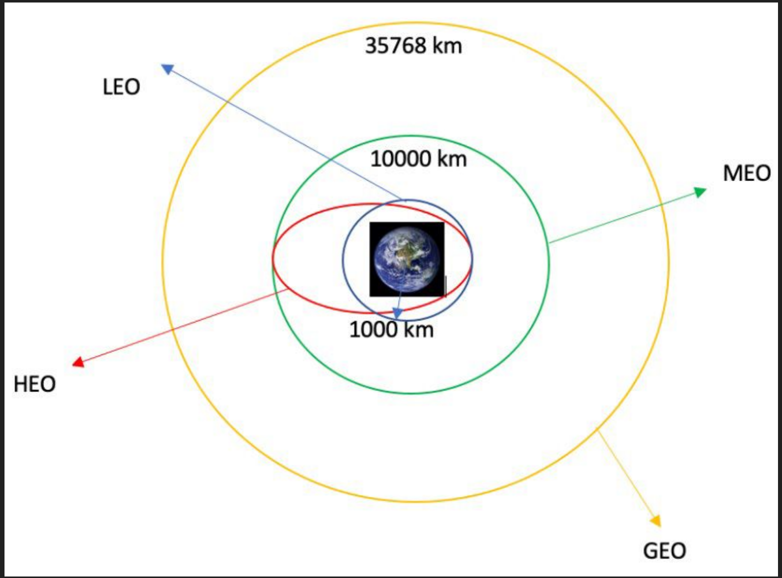
In [10]:
```
column_names = []

# 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
```

4. Create a dataframe by parsing the launch HTML tables
5. Export data to csv

Exploratory Data Analysis - Wrangling Data

We conducted exploratory data analysis (EDA) to better understand the dataset and define the training labels.

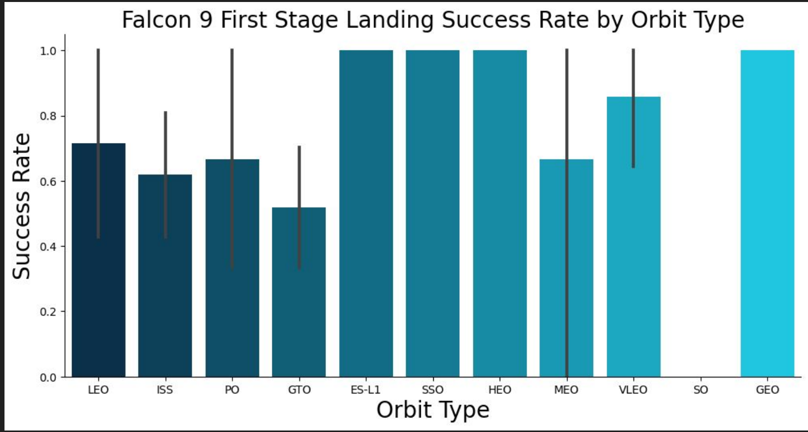
- We analyzed the number of launches by site and examined the frequency of each orbit type.
- A new landing outcome label was derived from the existing outcome column.
- The processed results were then exported to a CSV file for further use.

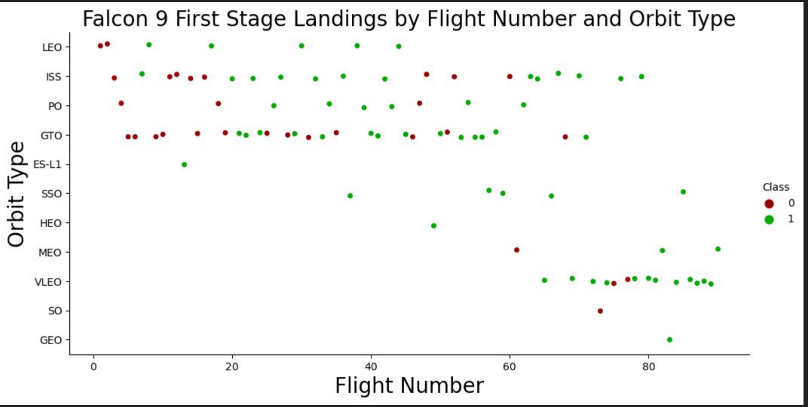


Exploratory Data Analysis - Visualizing Data

We explored the data by visualizing key relationships, including:

- Flight number versus launch site
- Payload versus launch site
- Success rates by orbit type
- Flight number versus orbit type
- Yearly trends in launch success





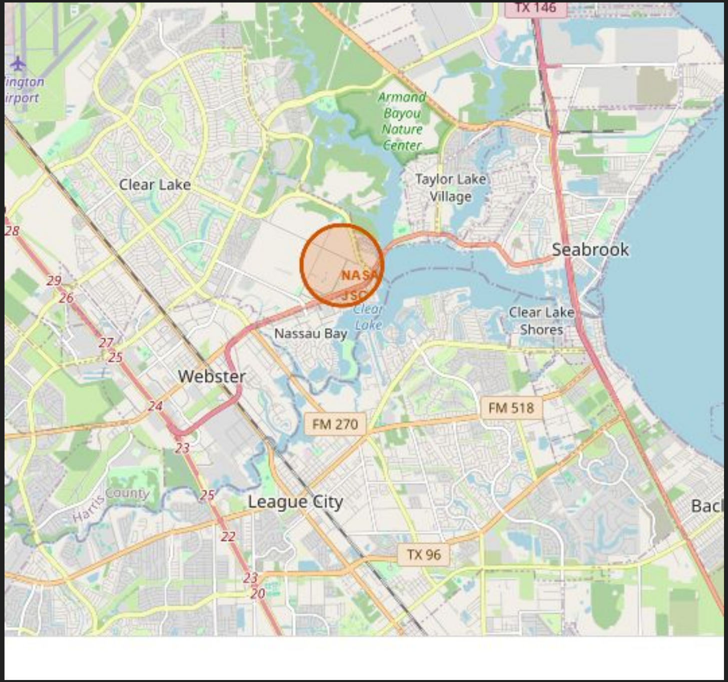
Exploratory Data Analysis - SQL

We loaded the SpaceX dataset into a PostgreSQL database directly from the Jupyter notebook.
Using SQL queries, we performed exploratory data analysis (EDA) to extract valuable insights, such as:

- Identifying unique launch site names involved in the missions.
- Calculating the total payload mass carried by boosters launched under NASA's CRS program.
- Determining the average payload mass for booster version F9 v1.1.
- Counting the total number of successful and failed mission outcomes.
- Analyzing failed landings on drone ships, including their booster versions and launch site locations.

Folium - Interactive Map

We visualized all launch sites on a Folium map, adding map elements such as markers, circles, and lines to represent the success or failure of each launch. Launch outcomes were categorized into two classes: 0 for failure and 1 for success. By using color-coded marker clusters, we were able to identify which launch sites demonstrated relatively high success rates. We also calculated the distances from each launch site to nearby features and addressed questions such as whether launch sites are located near railways, highways, or coastlines, and whether they maintain a certain distance from cities.

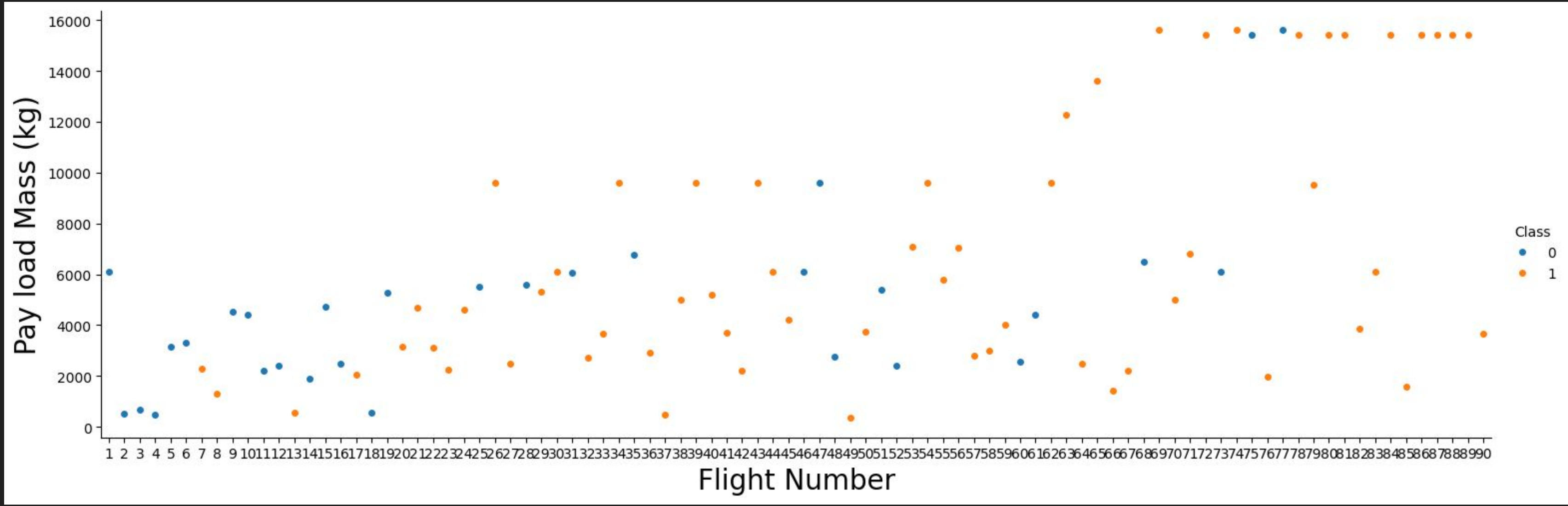


Classification - Predictive Analysis

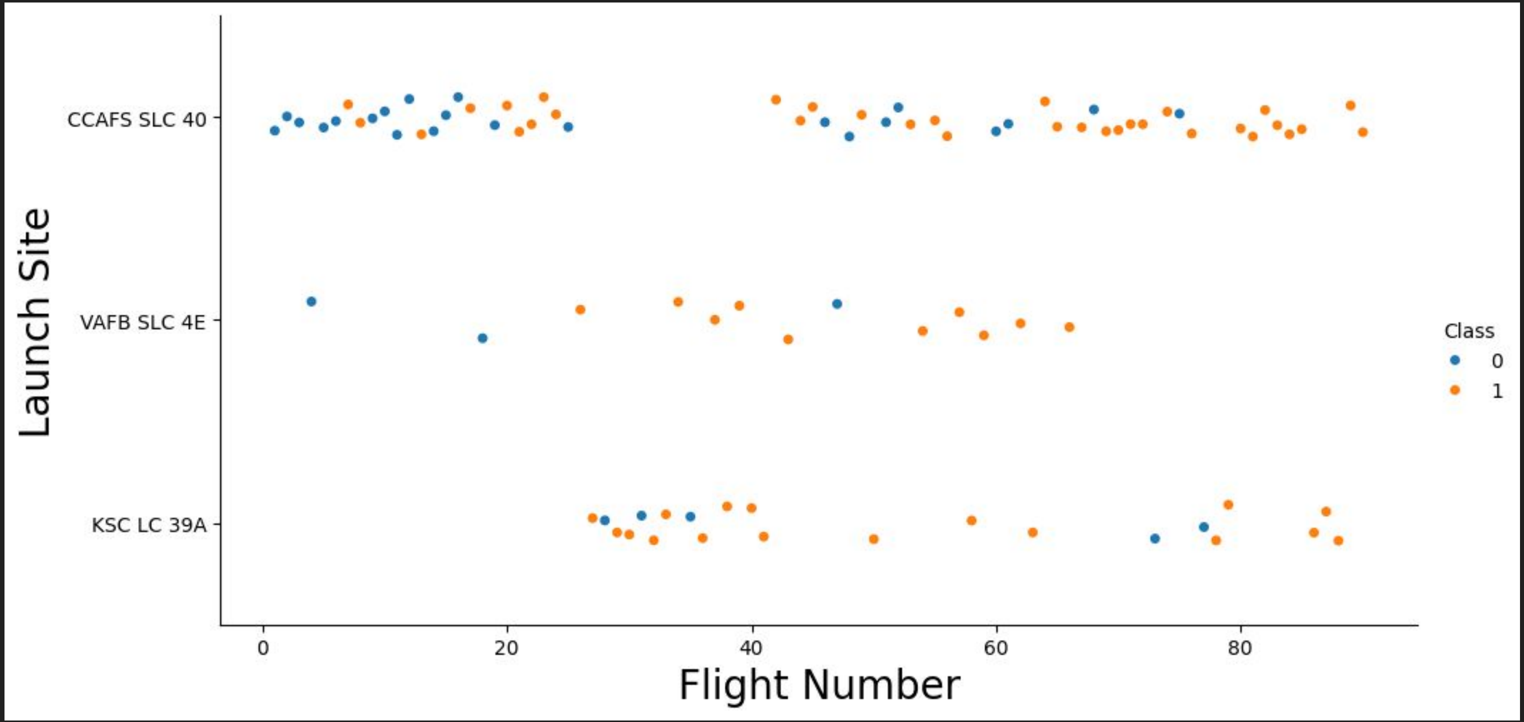
We loaded and transformed the data using NumPy and pandas, then split it into training and testing sets. We developed various machine learning models and fine-tuned their hyperparameters using GridSearchCV. Accuracy served as our primary evaluation metric, and we enhanced model performance through feature engineering and algorithm optimization. In the end, we identified the best-performing classification model.

Visualization Insights

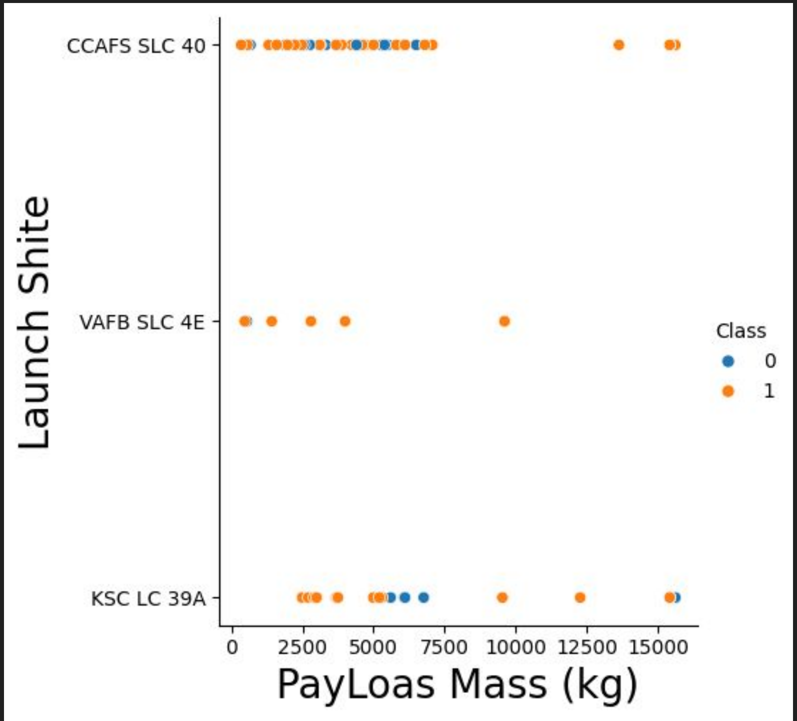
Perform exploratory Data Analysis and Feature Engineering using Pandas and Matplotlib. Exploratory Data Analysis and Preparing Data Feature Engineering

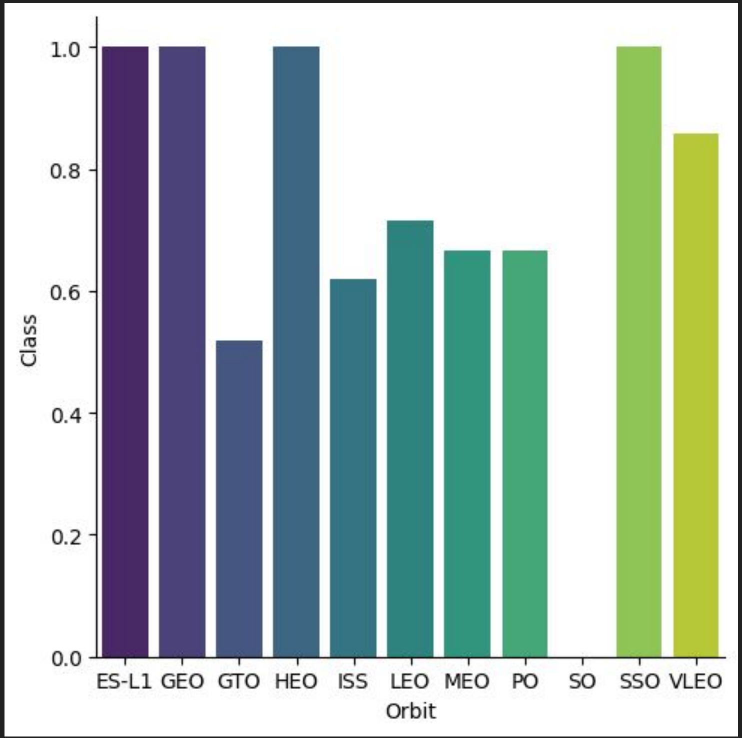


Launch Site VS Flight Number

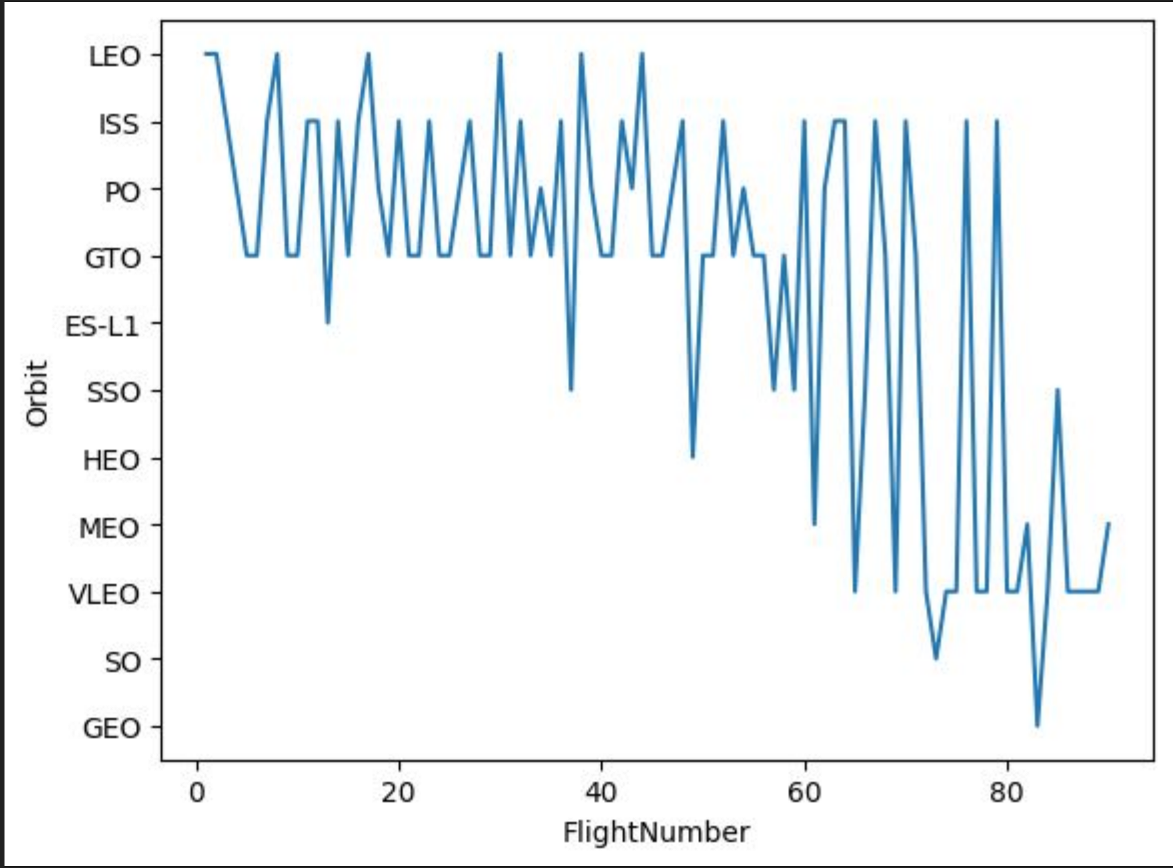


Launch Site VS PayLoad Mass - Class VS Orbit

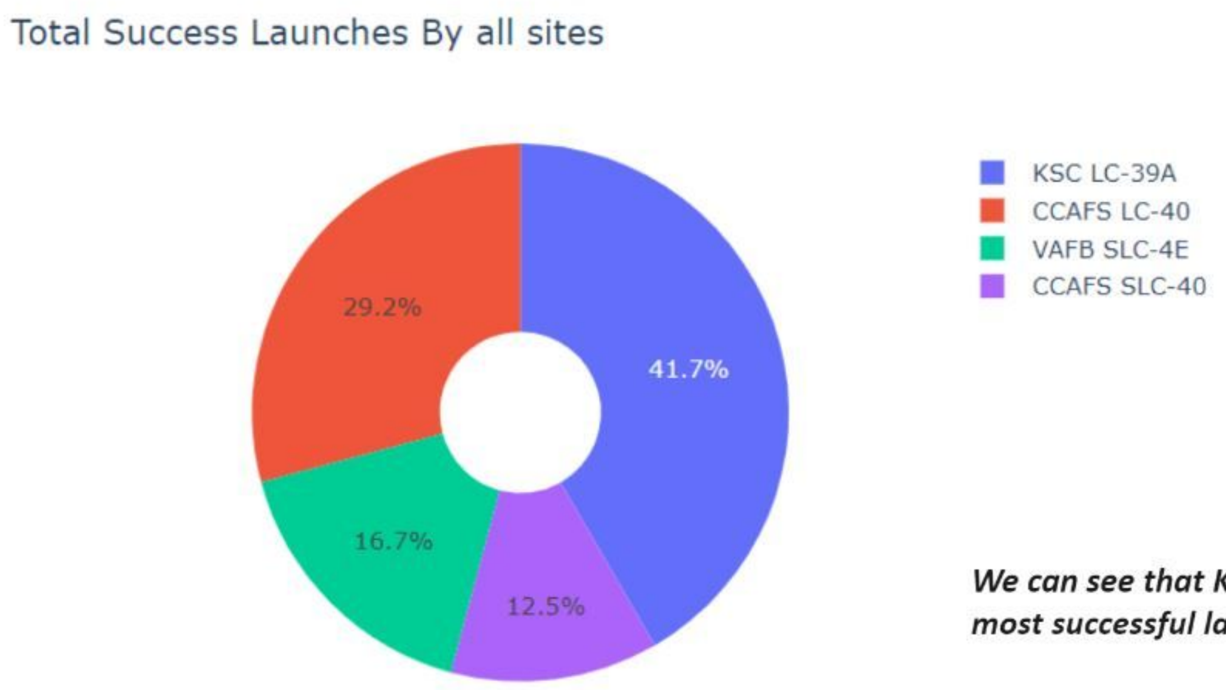




Orbit VS Flight Number

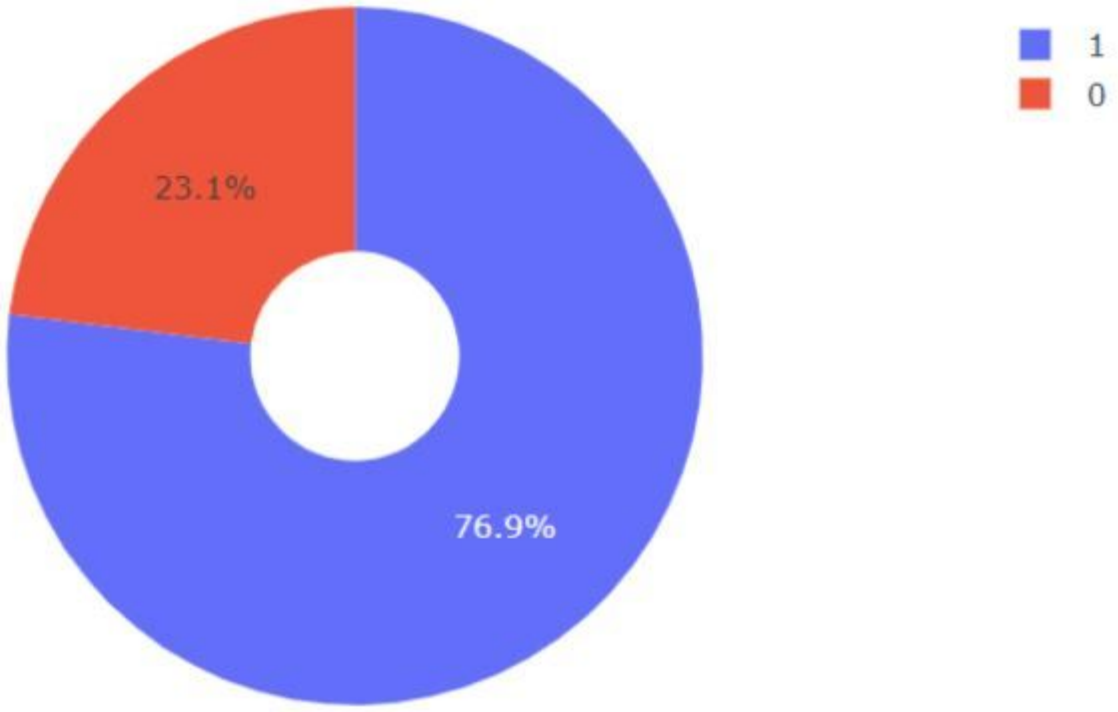


Dashboard with Plotly Dash



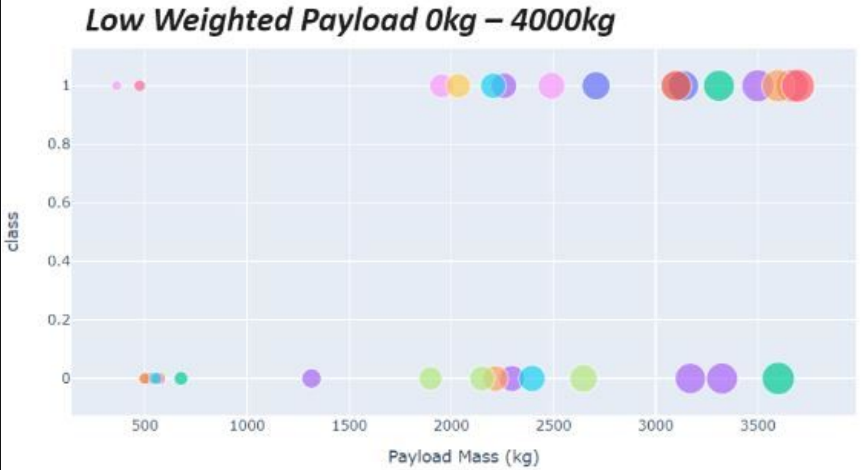
We can see that KSC LC-39A had the most successful launches from all the sites

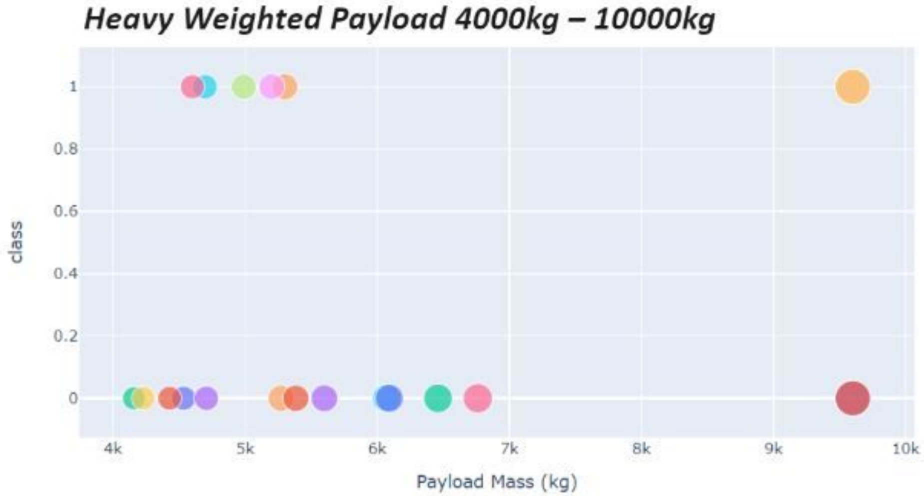
Dashboard with Plotly Dash



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

Dashboard with Plotly Dash





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

SQL Insights

We used the keyword **DISTINCT** to retrieve and display only the unique launch site names from the SpaceX dataset, ensuring that each launch site appeared only once in the results.

We calculated the total payload carried by NASA boosters to be 45,596 kg using the following SQL query.

Display the names of the unique launch sites in the space mission

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

Out[10]:

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

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

```
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

SQL Insights

We calculated the average payload mass carried by the booster version F9 v1.1 to be 2,928.4 kg.

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

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

The first successful landing on a ground pad was recorded on December 22, 2015, based on our analysis of the landing outcome dates.

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

SQL Insights

We used the WHERE clause to filter boosters that successfully landed on a drone ship, and applied the AND condition to further narrow down the results to those with a payload mass between 4,000 and 6,000 kg.

```
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

We used the wildcard % with the LIKE operator in the WHERE clause to filter records where the Mission Outcome indicated either a success or a failure.

List the total number of successful and failure mission outcomes

```
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

SQL Insights

We used the query above to display 5 records where the launch site names begin with CCA. In SQL, this is achieved using the LIKE operator with a wildcard. Specifically, the condition WHERE Launch_Site LIKE 'CCA%' filters the results to include only those records where the launch site name starts with "CCA". The % symbol acts as a wildcard, matching any sequence of characters that follow "CCA". To limit the output to just 5 records, the LIMIT 5 clause is added to the query.

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

SQL Insights

We identified the booster that carried the maximum payload by using a subquery with the MAX() function inside the WHERE clause. This allowed us to filter and display the booster(s) associated with the highest recorded payload.

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

SQL Insights

We ranked landing outcomes between 2010-06-04 and 2017-03-20 by selecting the landing outcome values along with their respective counts. Using the WHERE clause, we filtered the data to include only records within this date range. We then applied the GROUP BY clause to group the data by landing outcome and used the ORDER BY clause to sort the results in descending order based on the count of each outcome.

Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad))

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

Classification Results

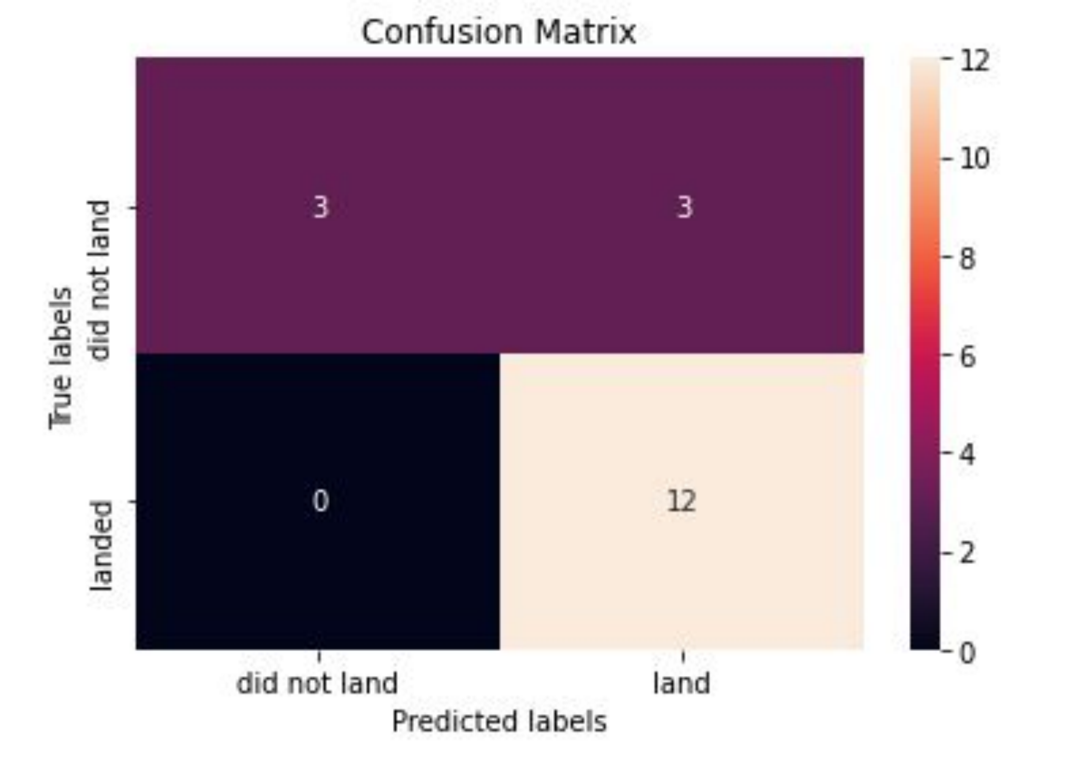
The decision tree classifier achieved the highest classification accuracy among all the models evaluated. Its ability to handle both numerical and categorical data, along with its interpretability and ease of visualization, made it a strong candidate for this classification task. By effectively capturing complex decision boundaries and requiring minimal data preprocessing, the decision tree outperformed other models in terms of predictive performance on our test dataset.

```
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 demonstrates that it can effectively differentiate between the classes. However, the primary issue lies in the number of false positives—cases where unsuccessful landings are incorrectly predicted as successful. This indicates a tendency of the model to overestimate positive outcomes.



Conclusions

We can conclude that launch sites with a higher number of flights tend to exhibit greater success rates. The overall success rate of launches showed a steady increase starting in 2013 and continued through 2020. Among the various orbits, ES-L1, GEO, HEO, SSO, and VLEO demonstrated the highest success rates. KSC LC-39A stood out as the launch site with the most successful missions. Additionally, the decision tree classifier proved to be the most effective machine learning algorithm for predicting launch outcomes in this analysis.