IBM Data Science Capstone Project

SpaceX Falcon 9 Landing Analysis

Aesha Darji

Agenda

- **Executive Summary**
- **Introduction**
- **Methodology**
- **Results**
- **Conclusion**

Executive Summary

Methodology

- Data on Falcon 9 first stage landings from 2010 to 2020 was gathered from the public SpaceX API (https://api.spacexdata.com/), Wikipedia (https://en.wikipedia.org/wiki/SpaceX), and additional datasets provided in the course.
- Data cleaning and wrangling focused on extracting landing outcome data to serve as the dependent variable for machine learning models.
- SQL queries and various data visualization techniques—including static plots, interactive maps, and an interactive dashboard—were used to analyze the dataset and derive insights.
- Predictive analysis was conducted using four machine learning models: Logistic Regression, Support Vector Machine (SVM), Decision Tree, and k-Nearest Neighbors (KNN).

Results

- The dataset includes details such as flight number, launch date, payload mass, orbit type, launch site, and mission outcome.
- Logistic Regression, SVM, and KNN demonstrated comparable performance in predicting landing outcomes.

Introduction

- SpaceX launches Falcon 9 rockets at approximately $62 million, significantly lower than other providers, which typically charge over $165 million. A major factor in these cost savings is SpaceX's ability to land and reuse the first stage of the rocket.
- By predicting whether the first stage will land successfully, we can estimate the cost per launch and use this insight to evaluate whether alternative companies should compete with SpaceX for rocket launches.
- The goal of this project is to predict the successful landing of the SpaceX Falcon 9 first stage.

Methodology

1. Data Collection

- Retrieving data from the SpaceX REST API using GET requests
- Web scraping to gather additional information

2. Data Wrangling

- Handling missing values using .fillna()
- Utilizing .value_counts() to analyze:
  - Launch counts per site
  - Frequency of each orbit type
  - Mission outcomes categorized by orbit type
- Creating a landing outcome label:
  - 0 → Booster did not land successfully
  - 1 → Booster landed successfully

3. Exploratory Data Analysis

- Using SQL queries to manipulate and analyze the SpaceX dataset
- Visualizing relationships and identifying patterns with Pandas and Matplotlib

4. Interactive Visual Analytics

- Conducting geospatial analysis with Folium
- Building an interactive dashboard using Plotly Dash

5. Data Modeling and Evaluation

- Leveraging Scikit-Learn for:

➢ Pre-processing (standardizing) the data

➢ Splitting the dataset into training and testing sets (train_test_split)

➢ Training multiple classification models

➢ Optimizing hyperparameters using GridSearchCV

- Generating confusion matrices for each model
- Evaluating the accuracy of all classification models

Data Collection – SpaceX REST API

Retrieving launch data using the SpaceX API, including details on the rocket used, payload delivered, launch parameters, landing specifications, and outcome.

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

response = requests.get(spacex_url)
```

```
# Use json_normalize method to convert the json result into a dataframe
data = pd.json_normalize(response.json())
```

- Create a GET response to the SpaceX REST API
- Convert the response to a .json file then to a Pandas DataFrame

```
#Global variables
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
ReusedCount = []
Serial = []
Longitude = []
Latitude = []
```

```
# Call getBoosterVersion
getBoosterVersion(data)

# Call getLaunchSite
getLaunchSite(data)

# Call getPayloadData
getPayloadData(data)

# Call getCoreData
getCoreData(data)
```

```
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}
```

- Use custom logic to clean the data
- Define lists for data to be stored in
- Call custom functions to retrieve data and fill the lists
- Use these lists as values in a dictionary and construct the dataset

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

- Create a Pandas DataFrame from the constructed dictionary dataset

```
data_falcon9 = df[df['BoosterVersion']!='Falcon 1']
```

```
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
```

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

- Filter the DataFrame to only include Falcon 9 launches
- Reset the FlightNumber column
- Replace missing values of PayloadMass with the mean PayloadMass value

Data Collection – Web Scraping

Extracting historical Falcon 9 launch data through web scraping from the Wikipedia page titled "List of Falcon 9 and Falcon Heavy launches."

```
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
response = requests.get(static_url)
# assign the response to a object
data = response.text
```

- Request the HTML page from the static URL
- Assign the response to an object

```
soup = BeautifulSoup(data, 'html5lib')
html_tables = soup.find_all('table')
```

- Create a BeautifulSoup object from the HTML response object
- Find all tables within the HTML page

```
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

for row in first_launch_table.find_all('th'):
    name = extract_column_from_header(row)
    if(name != None and len(name) > 0):
        column_names.append(name)
```

- Collect all column header names from the tables found within the HTML page

- Use the column names as keys in a dictionary
- Use custom functions and logic to parse all launch tables to fill the dictionary values

```
launch_dict= dict.fromkeys(column_names)

# Remove an irrelevant column
del launch_dict['Date and time ( )']

# Let's initial the launch_dict with each value to be an empty list
launch_dict['Flight No.'] = []
launch_dict['Launch site'] = []
launch_dict['Payload'] = []
launch_dict['Payload mass'] = []
launch_dict['Orbit'] = []
launch_dict['Customer'] = []
launch_dict['Launch outcome'] = []
# Added some new columns
launch_dict['Version Booster']=[]
launch_dict['Booster landing']=[]
launch_dict['Date']=[]
launch_dict['Time']=[]
```

- Convert the dictionary to a Pandas DataFrame ready for export

```
df = pd.DataFrame(launch_dict)
```

Data Wrangling

The SpaceX dataset contains information about several SpaceX launch facilities, recorded in the LaunchSite column. Each launch aims at a specific orbit, categorized in the Orbit column. The dataset also includes landing outcomes, shown in the Outcome column, which indicates whether a booster successfully landed.

Initial Data Exploration:

To analyze the dataset, the .value_counts() method is used to determine the following:

- Number of launches at each site.
- Number and occurrence of each orbit type.
- Number and occurrence of landing outcomes per orbit type.

```
# Apply value_counts() on column LaunchSite
df['LaunchSite'].value_counts()
```

```
CCAFS SLC 40    55
KSC LC 39A      22
VAFB SLC 4E     13
Name: LaunchSite, dtype: int64
```

```
# Apply value_counts on Orbit column
df['Orbit'].value_counts()
```

```
GTO      27
ISS      21
VLEO     14
PO        9
LEO       7
SSO       5
MEO       3
ES-L1     1
GEO       1
SO        1
HEO       1
Name: Orbit, dtype: int64
```

```
# landing_outcomes = values on Outcome column
landing_outcomes = df['Outcome'].value_counts()
landing_outcomes
```

```
True ASDS      41
None None      19
True RTLS      14
False ASDS      6
True Ocean      5
None ASDS       2
False Ocean     2
False RTLS      1
Name: Outcome, dtype: int64
```

Data Wrangling:

To determine whether a booster successfully landed, a binary column is created:

- 1 indicates a successful landing.
- 0 indicates an unsuccessful landing.

Steps to Create a Binary Landing Outcome Column:

1. Define a set of unsuccessful outcomes (bad_outcomes).
2. Create a list (landing_class) with 0 if the Outcome is in bad_outcomes, otherwise 1.
3. Add a new Class column containing values from landing_class.
4. Export the DataFrame as a .csv file.

```
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
bad_outcomes
```

```
{'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise

landing_class = []

for outcome in df['Outcome']:
    if outcome in bad_outcomes:
        landing_class.append(0)
    else:
        landing_class.append(1)
```

```
df['Class']=landing_class
```

```
df.to_csv("dataset_part\_2.csv", index=False)
```

EDA – Visualization Methodology

To explore relationships in the dataset, various visualization techniques are applied:

Scatter Charts

Scatter plots are created to examine correlations between:

- Flight Number and Launch Site
- Payload and Launch Site
- Orbit Type and Flight Number
- Payload and Orbit Type

Purpose: Scatter charts highlight relationships between two numerical variables.

Bar Charts

A bar chart is generated to analyze the relationship between:

- Success Rate and Orbit Type

Purpose: Bar charts compare numerical data across categorical variables and can be displayed either horizontally or vertically.

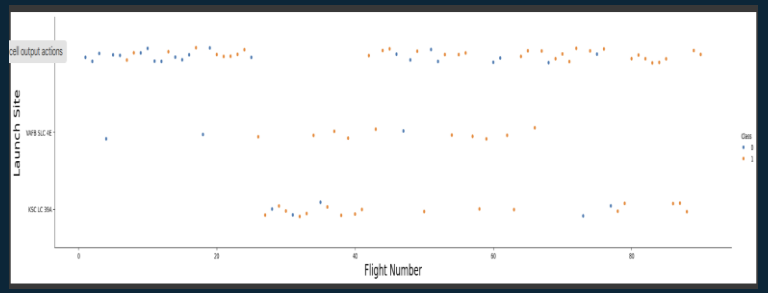
Line Charts

Line charts are used to observe trends over time, specifically:

- Success Rate per Year (showing annual launch success trends)

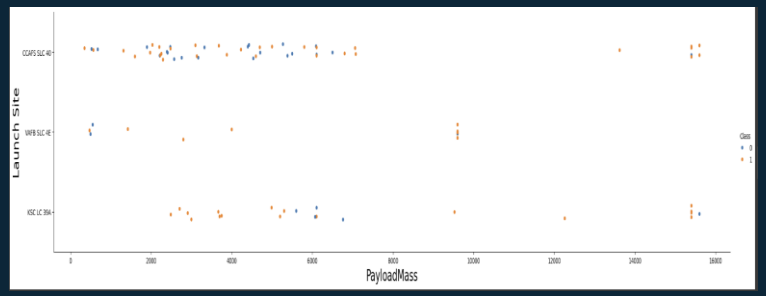
Purpose: Line charts help visualize changes in numerical values over time.

EDA – Visualization Results



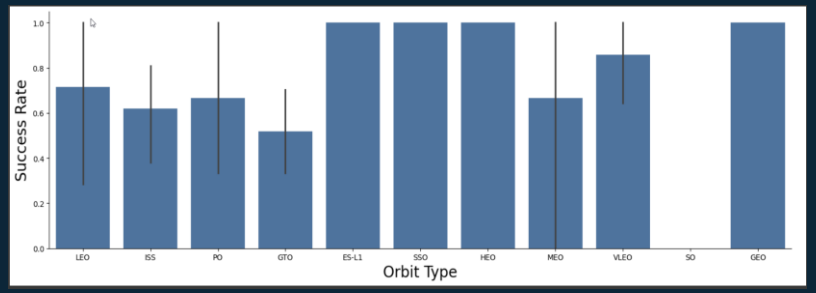
The scatter plot of Launch Site vs. Flight Number shows:

- Higher Flights, Higher Success – More flights lead to better success rates.
- Early Failures at CCAFS & VAFB – Flights below 30 had more failures.
- KSC LC 39A Had Better Success – No early launches, higher success rate.
- Success Improves After 30 – Flights above 30 had more successful landings.



The scatter plot of Launch Site vs. Payload Mass highlights the following findings:

- For payloads exceeding 7000 kg, unsuccessful landings are rare, though there is limited data available for these heavier launches.
- No distinct correlation exists between payload mass and landing success rate at any specific launch site.
- All launch sites handled a range of payload masses, with CCAFS SLC 40 primarily launching lighter payloads, though a few outliers exist.



The bar chart of Success Rate vs. Orbit Type indicates the following:

The orbits with a 100% success rate include:

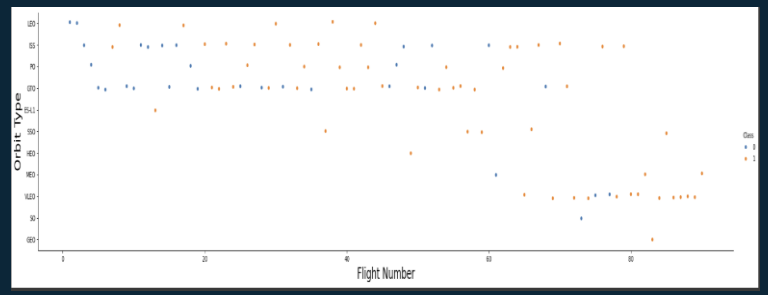
ES-L1 (Earth-Sun First Lagrangian Point)

- GEO (Geostationary Orbit)
- HEO (High Earth Orbit)
- SSO (Sun-synchronous Orbit)

The orbit with the lowest success rate (0%) is:

- SO (Heliocentric Orbit)

EDA – Visualization Results



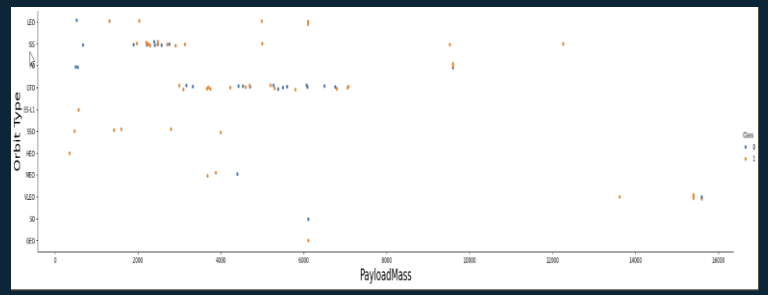
Orbit Type vs. Flight Number

100% success in GEO, HEO, and ES-L1 due to single flights.

SSO shows reliable success with five successful flights.

GTO shows no clear trend between flight number and success.

Higher flight numbers improve success, especially for LEO.



Orbit Type vs. Payload Mass

PO, ISS, and LEO succeed with heavier payloads.

GTO shows no clear link between payload mass and success.

VLEO launches are typically for heavier payloads.

Yearly Success Rate Trend

2010-2013: No successful landings.

Post-2013: Steady increase with dips in 2018 and 2020.

After 2016: Success rate consistently above 50%.

EDA – SQL

SQL Queries on the Dataset:

- Retrieve unique launch site names from space missions.
- Fetch 5 records where launch sites start with 'CCA’.
- Calculate the total payload mass from NASA (CRS) booster launches.
- Compute the average payload mass for booster version F9 v1.1.Identify the date of the first successful ground pad landing.
- List boosters with successful drone ship landings and payloads between 4000-6000 kg.
- Count the total successful and failed missions.
- Find booster versions that carried the highest payload mass.
- List failed drone ship landings with booster versions and launch sites for 2015.
- Rank landing outcomes (Failure/Success) between 2010-06-04 and 2017-03-20 in descending order.

EDA – SQL Results

1

```
%sql SELECT DISTINCT Launch_Site FROM SPACEXTABLE;
```

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

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

2

```
%sql SELECT * FROM SPACEXTABLE WHERE Launch_Site LIKE 'CCA%' LIMIT 5;
```

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

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	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	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

3

```
%sql SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEXTABLE WHERE Customer LIKE 'NASA (CRS)';
```

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

SUM(PAYLOAD_MASS__KG_)
45596

4

```
%sql SELECT AVG(PAYLOAD_MASS__KG_) FROM SPACEXTABLE WHERE Booster_Version LIKE 'F9 v1.1%';
```

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

AVG(PAYLOAD_MASS__KG_)
2534.6666666666665

5

```
%sql SELECT MIN(Date) FROM SPACEXTABLE WHERE "Landing _Outcome" = 'Success (ground pad)';
```

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

MIN(Date)
None

6

```
%sql SELECT Booster_Version FROM SPACEXTABLE WHERE "Landing _Outcome" = 'Success (drone ship)' AND PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000;
```

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

Booster_Version

7

```
%sql SELECT Mission_Outcome, COUNT(*) AS Total FROM SPACEXTABLE GROUP BY Mission_Outcome;
```

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

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

8

```
%sql SELECT Booster_Version FROM SPACEXTABLE WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTABLE);
```

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

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

9

```
%%sql
SELECT substr(Date, 6, 2) AS Month, "Landing _Outcome", Booster_Version, Launch_Site
FROM SPACEXTABLE
WHERE substr(Date, 1, 4) = '2015' AND "Landing _Outcome" LIKE 'Failure (drone ship)%';
```

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

Month	"Landing _Outcome"	Booster_Version	Launch_Site

10

```
%%sql
SELECT "Landing _Outcome", COUNT(*) AS Count
FROM SPACEXTABLE
WHERE Date BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY "Landing _Outcome"
ORDER BY Count DESC;
```

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

"Landing _Outcome"	Count
Landing _Outcome	31

Geospatial Analysis - Folium

Mapping Launch Sites and Analysis

Mark Launch Sites

- Initialize a Folium Map.
- Add folium.
- Circle and folium. Marker for each site.

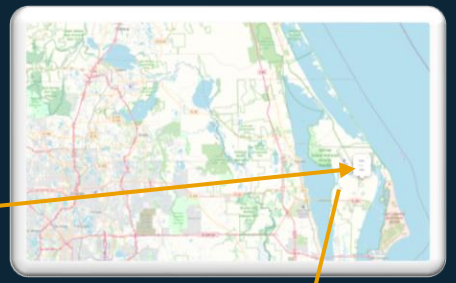
Mark Success/Failure of Launches

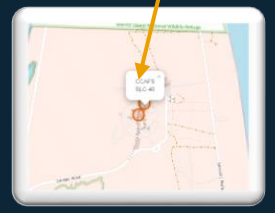
- Cluster launches with the same coordinates.
- Use green for success (class = 1) and red for failure (class = 0).Add folium.
- Marker to a MarkerCluster() and set icon_color accordingly.

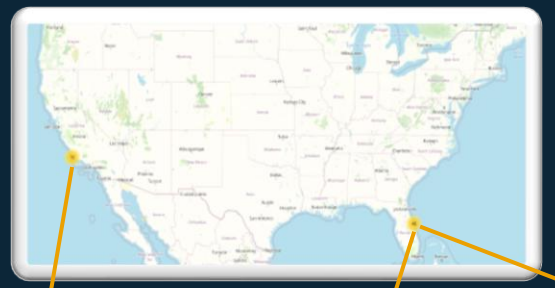
Calculate Distances

- Use Lat/Long values to compute distances.
- Mark points with folium.Marker and connect them using folium.PolyLine.

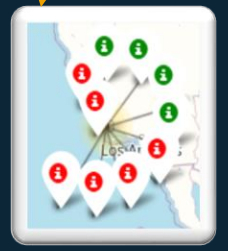
Folium - Results



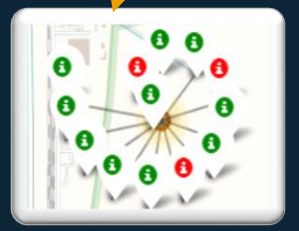




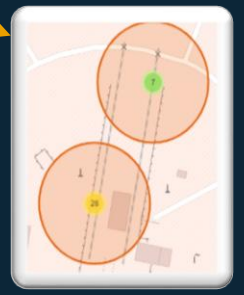
VAFB SLC-4E



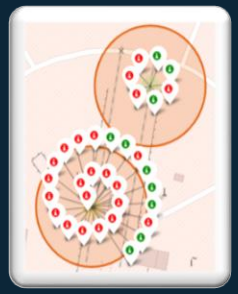
KSC LC-39A

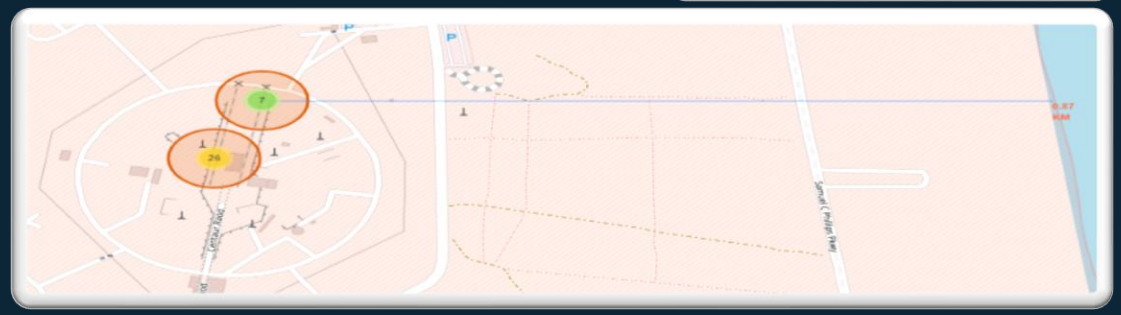


CCAFS SLC-40 and CCAFS LC-40



=





Plotly Dash Dashboard

Plotly Dash Visualizations

Pie Chart (px.pie())

- Shows total successful launches per site.
- Can be filtered with dcc.Dropdown() to view success/failure ratios.

Scatter Plot (px.scatter())

- Displays correlation between success and payload mass (kg).
- Can be filtered by payload range (RangeSlider()) and booster version.

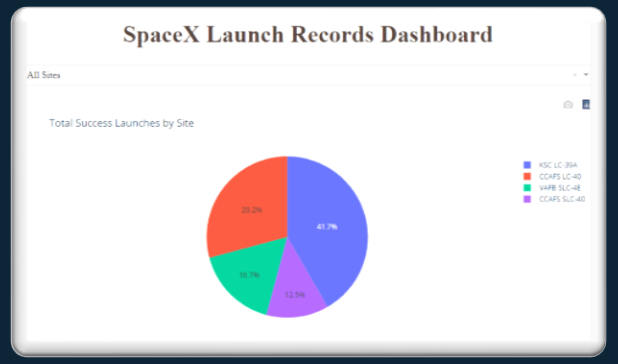
Launch Outcome vs. Payload

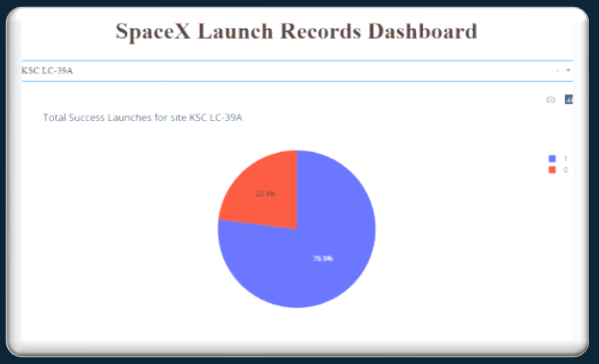
Data is split into two ranges:

- 0–4000 kg (low payloads)
- 4000–10,000 kg (massive payloads)

Success rates are lower for massive payloads.

Booster types v1.0 and B5 were not used for massive payloads.





Predictive Analysis

Model Development

Model Evaluation

Finding the Best Classification Model

To prepare the dataset for model development:

- Load dataset
- Perform necessary data transformations (standardise and pre-process)
- Split data into training and test data sets, using train_test_split()
- Decide which type of machine learning algorithms are most appropriate

For each chosen algorithm:

- Create a GridSearchCV object and a dictionary of parameters
- Fit the object to the parameters
- Use the training data set to train the model

- For each chosen algorithm:
  - Using the output GridSearchCV object:
    - Check the tuned hyperparameters (best_params_)
    - Check the accuracy (score and best_score_)
  - Plot and examine the Confusion Matrix

- Review the accuracy scores for all chosen algorithms
- The model with the highest accuracy score is determined as the best performing model

Predictive Analysis - Results

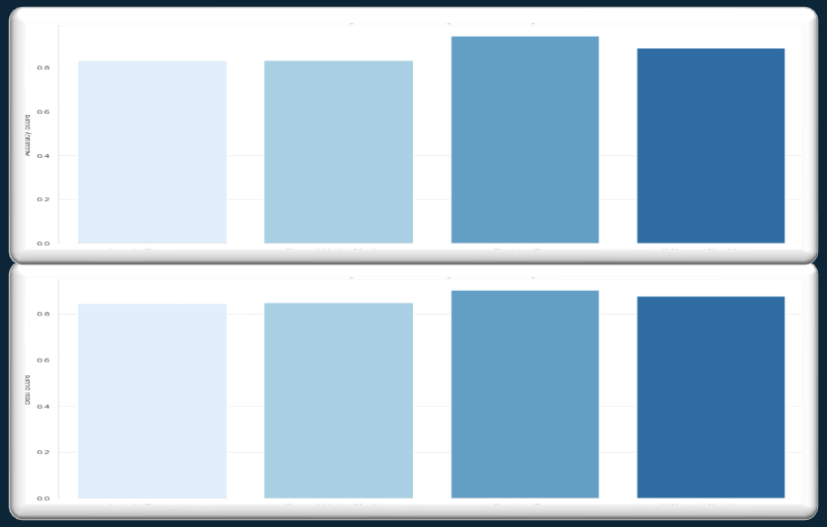
Classification Model Performance:

•Decision Tree achieved the highest accuracy.

•Accuracy Score: 94.44%

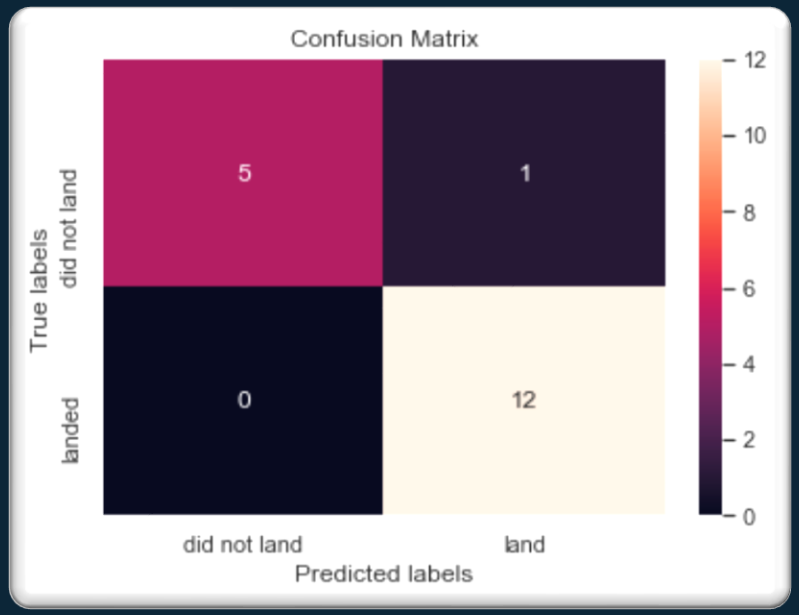
•Best Score: 90.36%

Algorithm	Accuracy Score	Best Score
Logistic Regression	0.833333	0.846429
Support Vector Machine	0.833333	0.848214
Decision Tree	0.944444	0.903571
K Nearest Neighbours	0.888889	0.876786



Decision Tree Model Performance:

- Best performing model with 94.44% accuracy.
- Confusion matrix: 1 misclassification (false positive) out of 18 results.
- Correct classifications: 5 did not land, 12 successfully landed.



Conclusions

Key Insights from Launch Data

Success Rate Over Time

- 2010-2013: No successful landings.
- Post-2013: Success rate improved, with minor dips in 2018 and 2020.
- After 2016: Success rate consistently above 50%.

Orbit Success Rates

- ES-L1, GEO, HEO, and SSO: 100% success rate.
- SSO's success is notable with 5 successful flights.
- PO, ISS, and LEO: More success with heavy payloads.
- VLEO launches are associated with heavier payloads.

Top Launch Site

- KSC LC-39A: Most successful site, 41.7% of total successes with a 76.9% success rate.

Payload Success

- Massive payloads (>4000kg) have lower success rates than lighter ones.

Best Classification Model

- Decision Tree Model had the highest accuracy at 94.44%.

Thank you