IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Dauren Yeleukenov

March 10th, 2025.

Outline

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

Executive Summary

Project Overview

- This project analyzes historical SpaceX data to predict whether Falcon 9 first-stage boosters will land successfully. Since SpaceX's cost advantage largely comes from reusing these boosters, accurately predicting landing outcomes has significant financial implications.

Methodology Used

- The project followed a comprehensive data science workflow:

  Data Acquisition: Combined SpaceX API data (via GET requests) with Wikipedia data (via web scraping with BeautifulSoup)

  Data Processing: Cleaned missing values and encoded categorical variables, creating a binary classification target for landing success

  Analysis: Used SQL queries and various visualization tools (Matplotlib, Seaborn, Folium, Plotly Dash) to identify patterns in launch outcomes

  Machine Learning: Built and compared four classification models (Logistic Regression, SVM, Decision Trees, KNN)

  Key Findings

- All models achieved approximately 83% accuracy, with the Decision Tree model performing best (88.93% validation accuracy). These predictions help improve cost estimation, resource planning, and risk assessment for future launches.

Significance

- This project demonstrates how data science techniques can enhance the economics of space exploration by helping predict and improve reusable rocket landing success rates.

Introduction

- SpaceX has transformed the space sector through its economical Falcon 9 launches. The key to their cost advantage lies in their ability to reuse the first stage of the rocket.
- This project sought to develop a machine learning system that could forecast first stage landing outcomes, an essential factor in minimizing launch expenses.
- By identifying the variables that affect landing success, we can assist competitor companies in challenging SpaceX's market position.

Section 1

Methodology

Methodology

- Data collection methodology applied:
  - We collected the data from the SpaceX API using get requests.
  - Further, the response was decoded into JSON format and converted to a pandas DataFrame for analysis.
- Data wrangling
  - Cleaned the data by checking for and filling in missing values.
  - Applied one-hot encoding to categorical features.
- Exploratory data analysis (EDA) using visualization and SQL
- Interactive visual analytics using Folium and Plotly Dash
- Using classification models conducted predictive analysis

Data Collection

- Data collection began by utilizing the SpaceX API (a RESTful API) through GET requests. This process involved creating several helper functions to extract information from the API using identification numbers found in the launch data. To ensure consistency in the JSON results, the SpaceX launch data was requested, parsed via GET requests, and then decoded as JSON before being converted into a Pandas dataframe.
- Additionally, web scraping techniques were employed to gather historical Falcon 9 launch records from a Wikipedia page titled "List of Falcon 9 and Falcon Heavy launches." Since these records were stored in HTML format, BeautifulSoup and request libraries were used to extract the Falcon 9 launch HTML table records from the Wikipedia page. These records were then parsed and transformed into a Pandas dataframe for further analysis. Describe how data sets were collected.

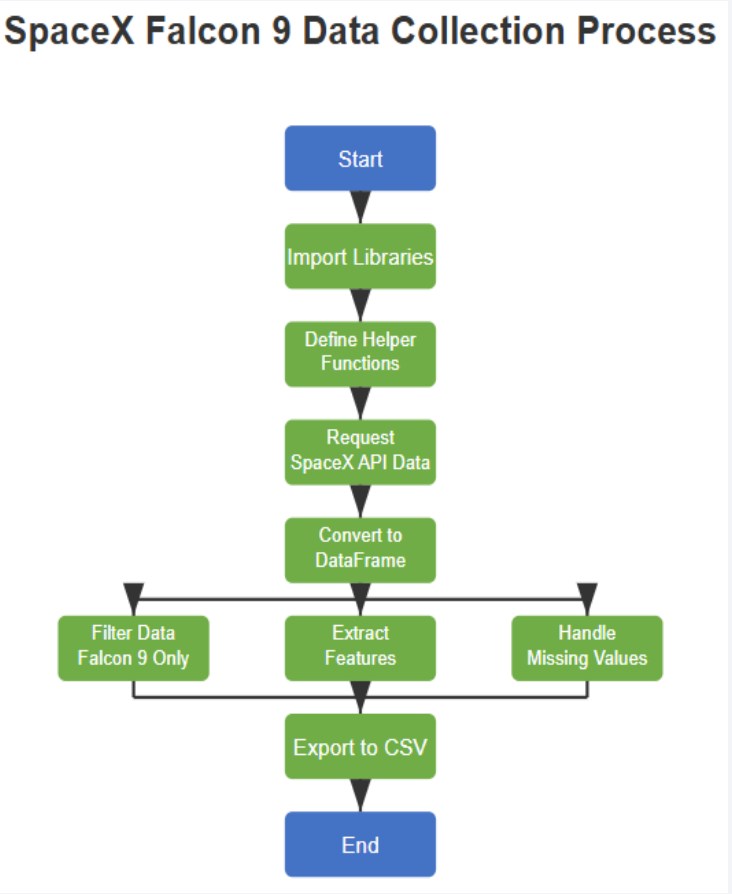
Data Collection – SpaceX API

The code in the Jupyter notebook demonstrates how to gather and process SpaceX launch data using their API. The process begins by importing essential libraries and defining helper functions to navigate the API's response structure. The workflow continues with sending GET requests to the SpaceX API to retrieve launch records, which are then decoded from JSON format into a structured pandas DataFrame.

The notebook focuses specifically on Falcon 9 launches by filtering the dataset and extracting critical features including booster information, payload details, launch site coordinates, and landing outcomes.

After handling missing values through mean imputation, the cleaned dataset is exported to CSV format, creating a foundation for developing predictive models to determine the likelihood of successful Falcon 9 first stage landings - a key factor in SpaceX's cost-efficient launch strategy.

https://github.com/daureny/IBM_course/blob/main/jupyter-labs-spacex-data-collection-api%20(1).ipynb

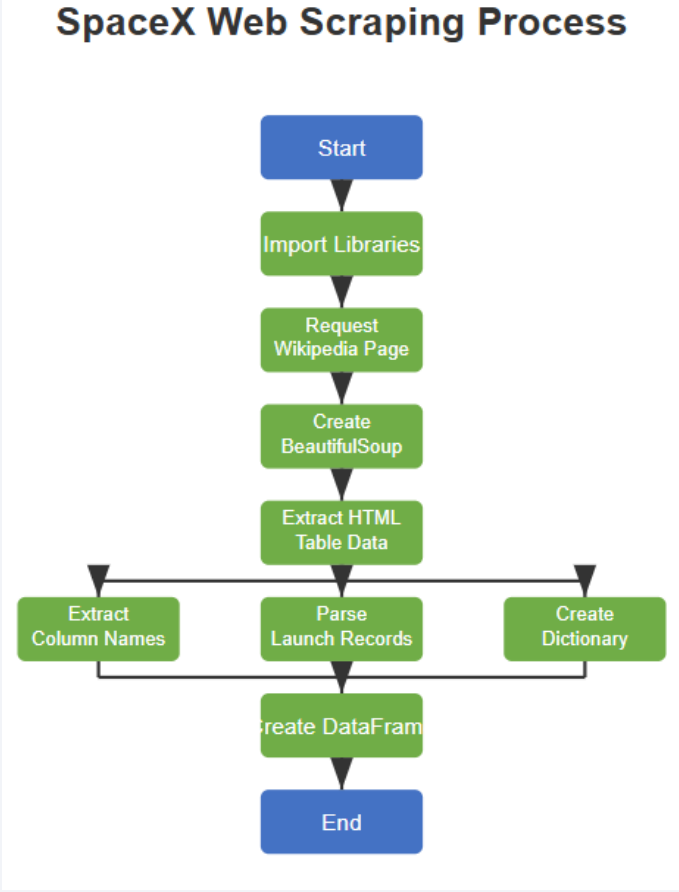


Data Collection - Scraping

The SpaceX web scraping process extracts Falcon 9 launch data from Wikipedia using BeautifulSoup and requests libraries. After connecting to a specific version of the Wikipedia page, the notebook parses HTML content to locate launch record tables. It systematically extracts column names from table headers and processes each row to collect flight numbers, dates, booster versions, launch sites, payload details, and landing outcomes.

All extracted data is organized into a dictionary with appropriate keys, then converted to a pandas DataFrame for subsequent analysis. This structured approach transforms unstructured web content into an organized dataset suitable for predicting Falcon 9 first stage landing success.

https://github.com/daureny/IBM_course/blob/main/jupyter-labs-webscraping%20(1).ipynb



Data Wrangling

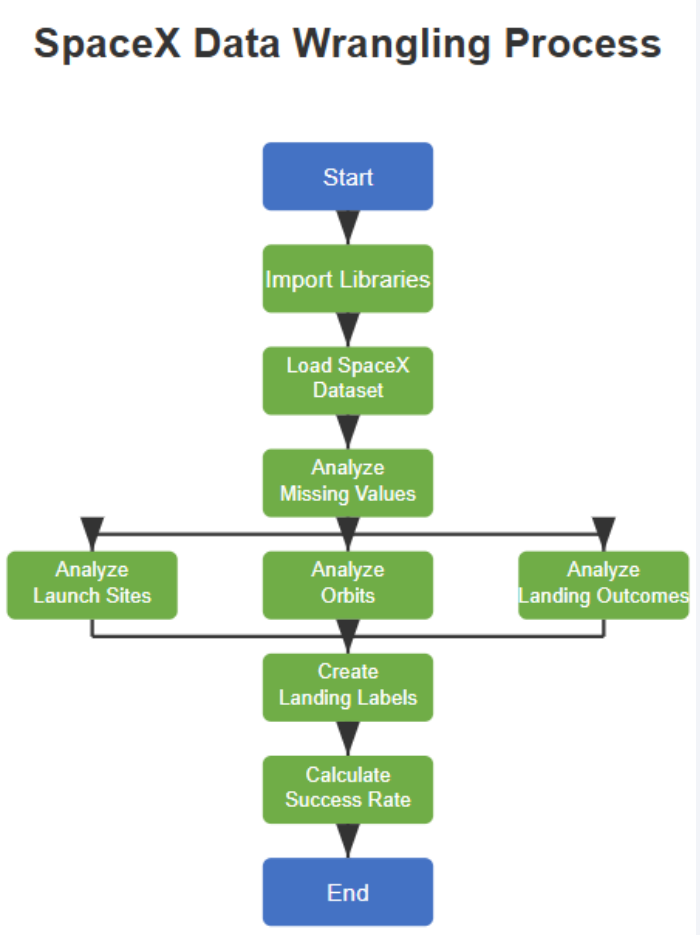
The third part focuses on data wrangling and exploratory data analysis to prepare for the Falcon 9 landing prediction model. This essential preparatory phase transforms raw data into meaningful training labels by examining landing outcomes.

The process begins by importing pandas and numpy libraries before loading the SpaceX dataset collected in the previous steps. The notebook first analyzes missing values, determining that only the LandingPad column has missing data (approximately 29%). It then progresses through systematic exploratory analysis of three key areas: launch sites (with CCAFS SLC 40 being the most frequent), orbit types (with GTO, ISS, and VLEO being the most common), and landing outcomes.

The critical transformation occurs when analyzing the landing outcomes column, which contains values like "True ASDS" (successful drone ship landing), "False Ocean" (unsuccessful ocean landing), and "None None" (no landing attempt). These diverse outcomes are converted into a binary classification variable where 1 represents successful landings and 0 represents unsuccessful ones. This conversion creates the target variable for subsequent machine learning models.

The notebook concludes by calculating the overall success rate of Falcon 9 first stage landings, revealing that approximately 67% of the missions achieved successful landings. This processed dataset with clearly defined labels forms the foundation for predictive modeling in later stages of the project.

https://github.com/daureny/IBM_course/blob/main/labs-jupyter-spacex-Data%20wrangling%20(1).ipynb



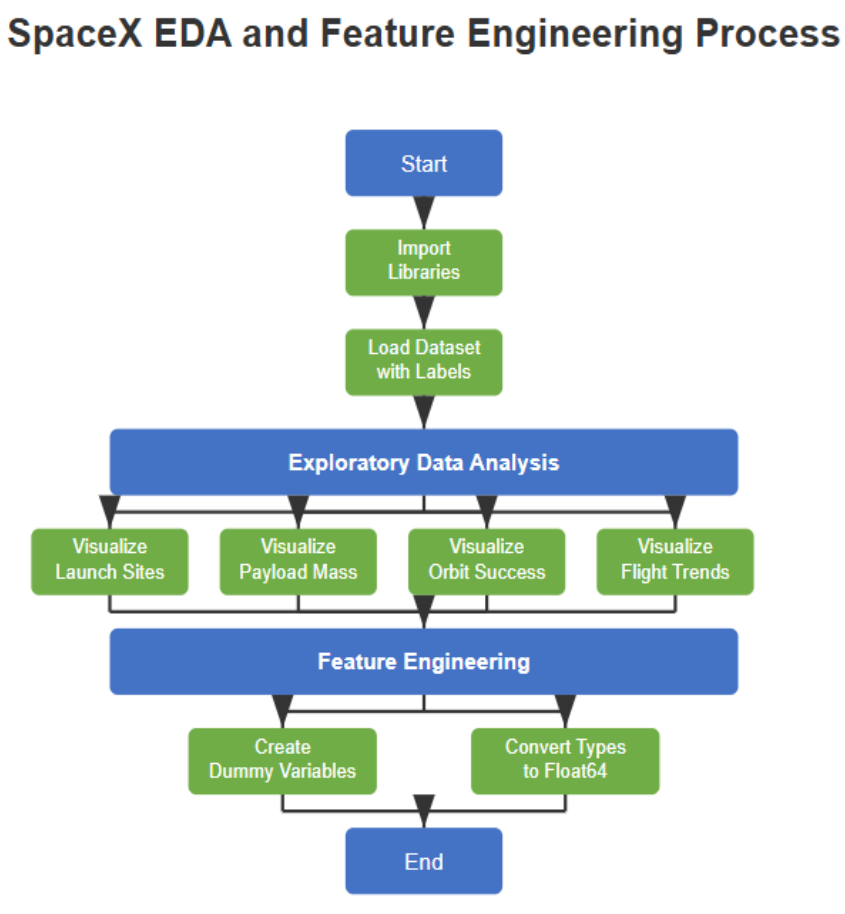
EDA with Data Visualization

The fourth Jupyter notebook in the SpaceX Falcon 9 First Stage Landing Prediction project focuses on exploring the previously labeled dataset and preparing features for machine learning. This critical stage provides visual insights into variables affecting landing success and structures the data for predictive modeling. The process begins by importing essential libraries (pandas, numpy, matplotlib, and seaborn) and loading the labeled dataset created in the previous data wrangling step. This dataset already contains the crucial "Class" column where 1 represents successful landings and 0 represents unsuccessful ones.

In the exploratory data analysis phase, the notebook systematically investigates relationships between various factors and landing success through multiple visualizations: 1) Flight number and payload mass plotted together reveal that more recent flights had higher success rates, irrespective of payload mass, 2) Analysis of launch sites shows each site's landing success pattern over time, 3) Visualization of payload mass by launch site indicates that certain sites accommodate heavier payloads, 4) Orbit-specific success rates demonstrate varying levels of difficulty in achieving successful landings for different orbit types, 5) A yearly trend analysis shows a clear improvement in success rates over time as SpaceX refined their technology

The feature engineering section transforms the analyzed data into a format suitable for machine learning: categorical variables (such as Orbit, LaunchSite, LandingPad, and Serial) are converted to numeric values using one-hot encoding. All variables are converted to float64 data type for consistency in model training

https://github.com/daureny/IBM_course/blob/main/edadataviz.ipynb



EDA with SQL

The fifth Jupyter notebook in the SpaceX project focuses on utilizing SQL queries to analyze launch data and extract meaningful insights. This approach allows for structured data manipulation and sophisticated analysis through SQL's querying capabilities.

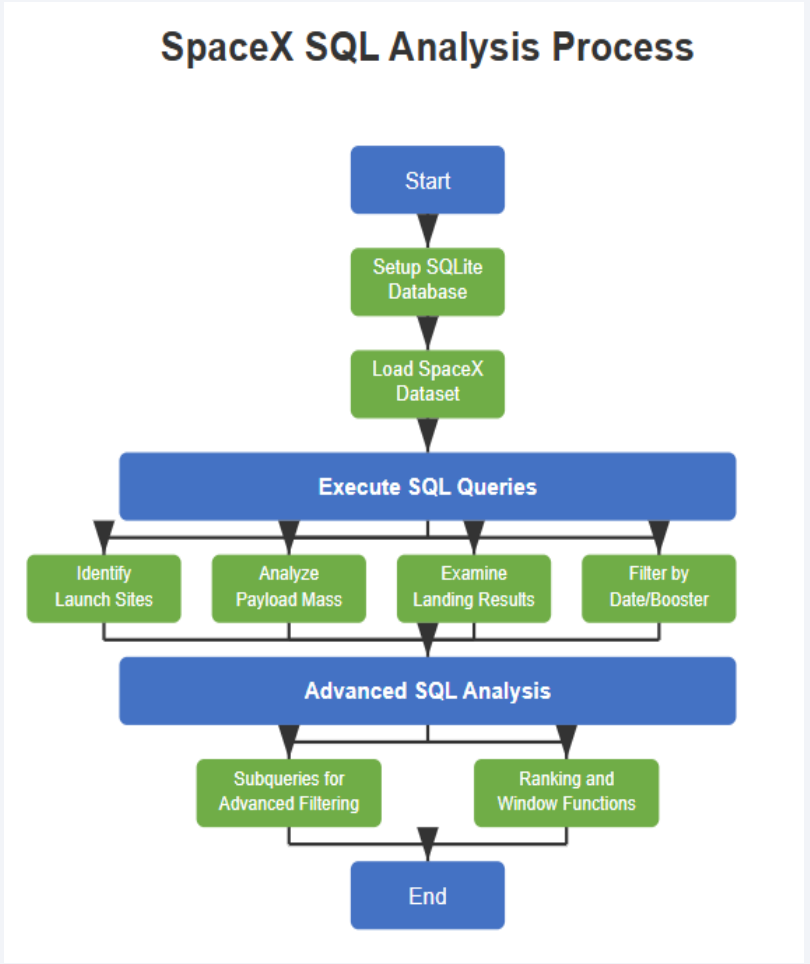
The process begins with the initial setup of an SQLite database environment, importing necessary libraries like sqlite3 and pandas. The SpaceX dataset is then downloaded from a provided URL and loaded into a database table, creating a foundation for SQL-based analysis.

Once the data is properly structured in the database, the notebook demonstrates a progressive series of SQL queries that explore different aspects of the SpaceX launch data:

1. Basic queries identify unique launch sites and extract specific records matching criteria like launch sites beginning with "CCA"
2. Aggregation queries calculate total payload mass for NASA missions and average payload mass for specific booster versions
3. Date-based queries find significant milestones such as the first successful ground landing
4. Complex filtering extracts records of boosters that successfully landed on drone ships with specific payload mass ranges
5. Advanced analysis uses SQL window functions to rank landing outcomes and subqueries to find boosters carrying maximum payload

The SQL approach provides a powerful way to interact with the data, allowing for precise filtering, aggregation, and analysis. This methodology complements the previous data processing and visualization steps by enabling targeted extraction of information that answers specific business questions about SpaceX's launch operations and landing success rates.

https://github.com/daureny/IBM_course/blob/main/jupyter-labs-eda-sql-coursera_sqllite.ipynb



Build an Interactive Map with Folium

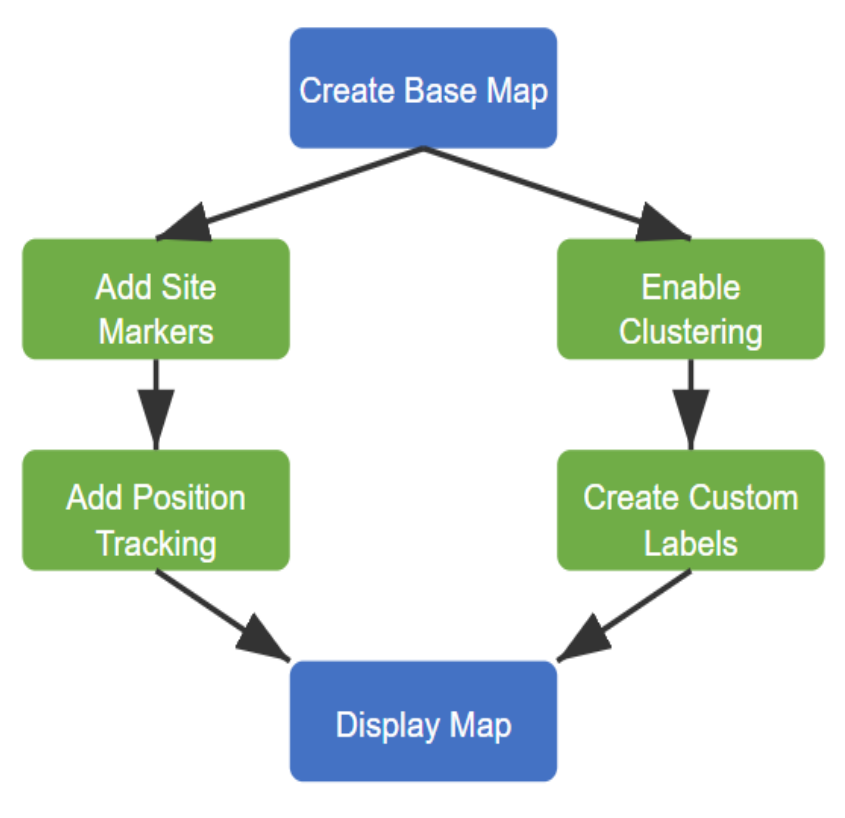
The notebook creates an interactive Folium map to visualize the locations of SpaceX launch sites. To ensure clarity and usability, several key elements are incorporated.

First, **Markers** are added to pinpoint the exact locations of each launch site. Since multiple sites are included, **Marker clusters** are used to group them together, reducing clutter and making the map more navigable. This allows users to zoom in and explore individual locations with ease.

To enhance interactivity, the **Mouse Position plugin** is enabled, displaying real-time latitude and longitude coordinates as users hover over different areas of the map. This feature provides precise location details and improves usability.

Additionally, **Custom labels** are incorporated using Folium's `DivIcon` feature, ensuring that each launch site is clearly named and identifiable without requiring users to click on individual markers.

Together, these elements create an intuitive and informative map, allowing for seamless exploration of SpaceX's launch sites.



https://github.com/daureny/IBM_course/blob/main/lab_jupyter_launch_site_location.ipynb

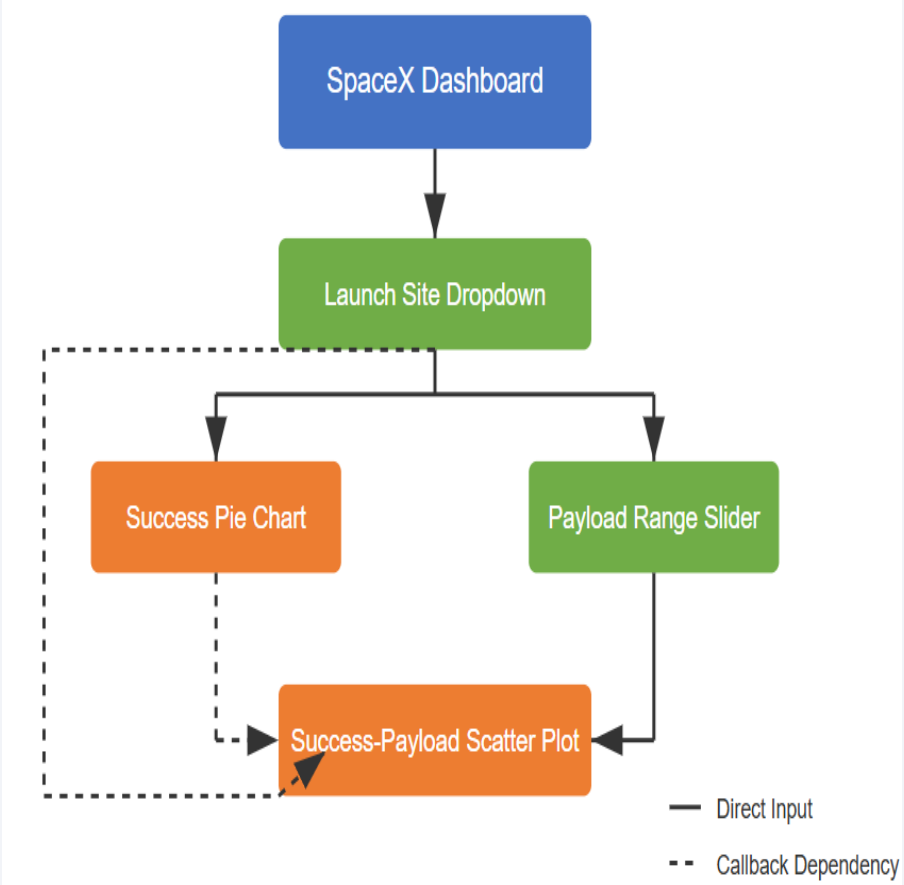
Build a Dashboard with Plotly Dash

The dashboard features four key interactive elements:

- Site Selection Dropdown - allowing users to filter between all launch sites or a specific location
- Success Rate Pie Chart - showing either success distribution across sites or success/failure ratio for a selected site
- Payload Range Slider - for filtering data by payload mass
- Success-Payload Scatter Chart - displaying the relationship between payload mass and mission outcomes, color-coded by booster version

These visualizations enable users to identify patterns in launch success rates across different sites, examine how payload mass affects mission outcomes, and evaluate performance differences between booster versions. The interactive filtering capabilities support both broad analysis and focused investigation of specific factors affecting SpaceX launch success.

https://github.com/daureny/IBM_course/blob/main/dash.py

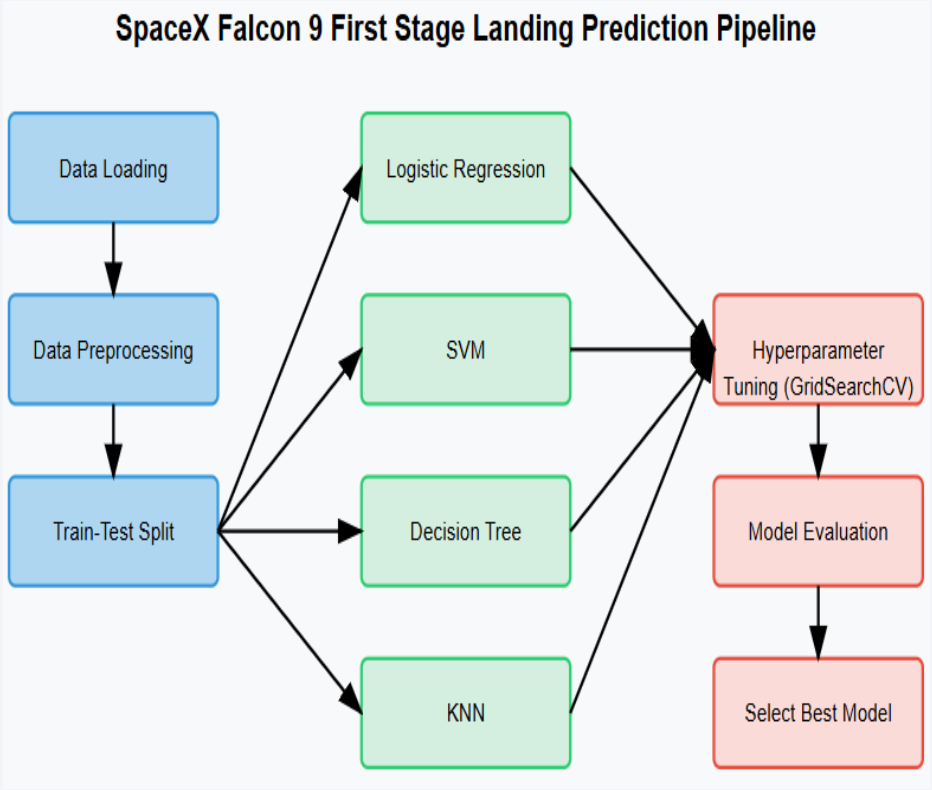


Predictive Analysis (Classification)

This notebook represents the last part of a machine learning project focused on predicting whether SpaceX Falcon 9 first stages will land successfully after launch. The project has significant business implications since SpaceX's ability to reuse first stages reduces launch costs from about $165 million to $62 million.

- **Key Components**loading and preprocessing from previous stages
- Feature standardization and train-test splitting
- Implementation of multiple classification algorithms with hyperparameter tuning:
  - Logistic Regression
  - Support Vector Machines (SVM)
  - Decision Trees
  - K-Nearest Neighbors (KNN)
- Model evaluation using accuracy metrics and confusion matrices
- Determining the best performing model for landing prediction
- Data

https://github.com/daureny/IBM_course/blob/main/SpaceX_Machine%20Learning%20Prediction_Part_5.ipynb

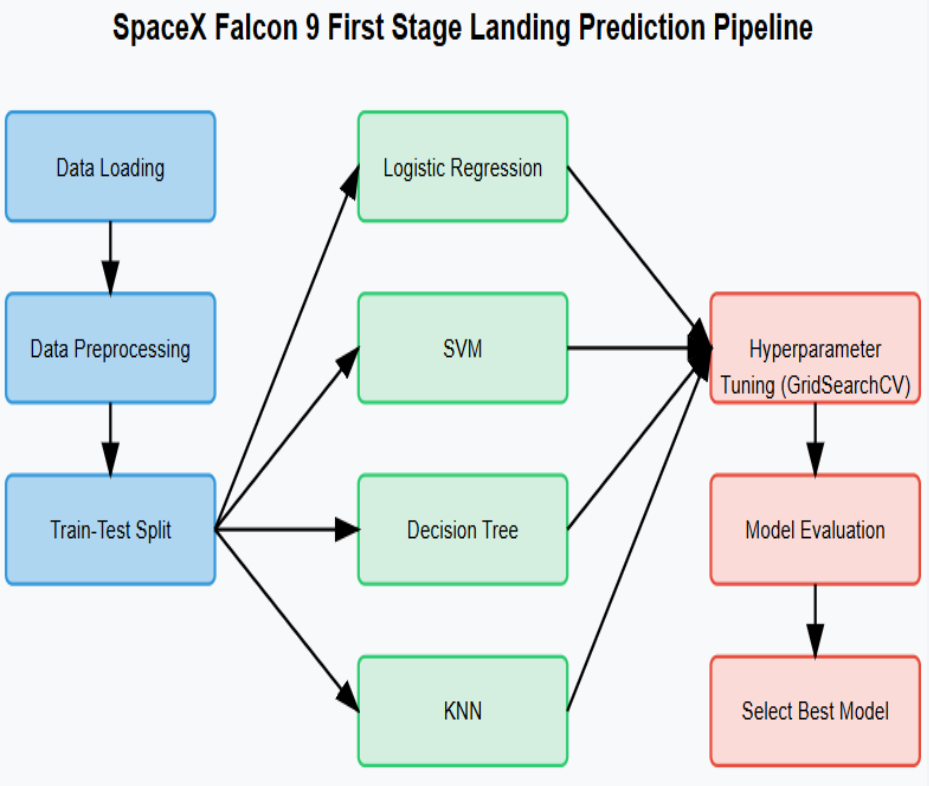


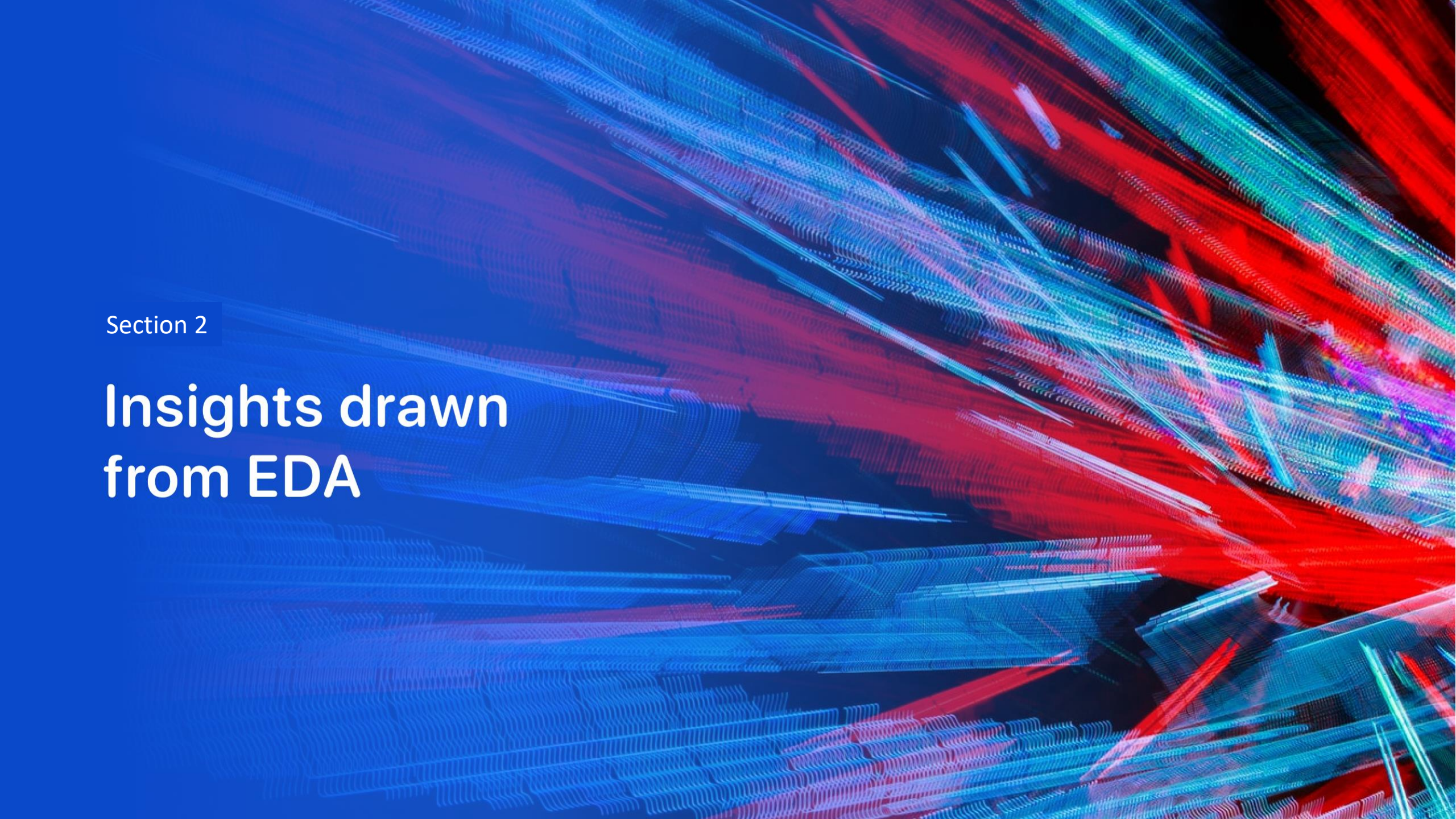
Predictive Analysis (cont)

Business Context

This project has significant practical applications with the following results:

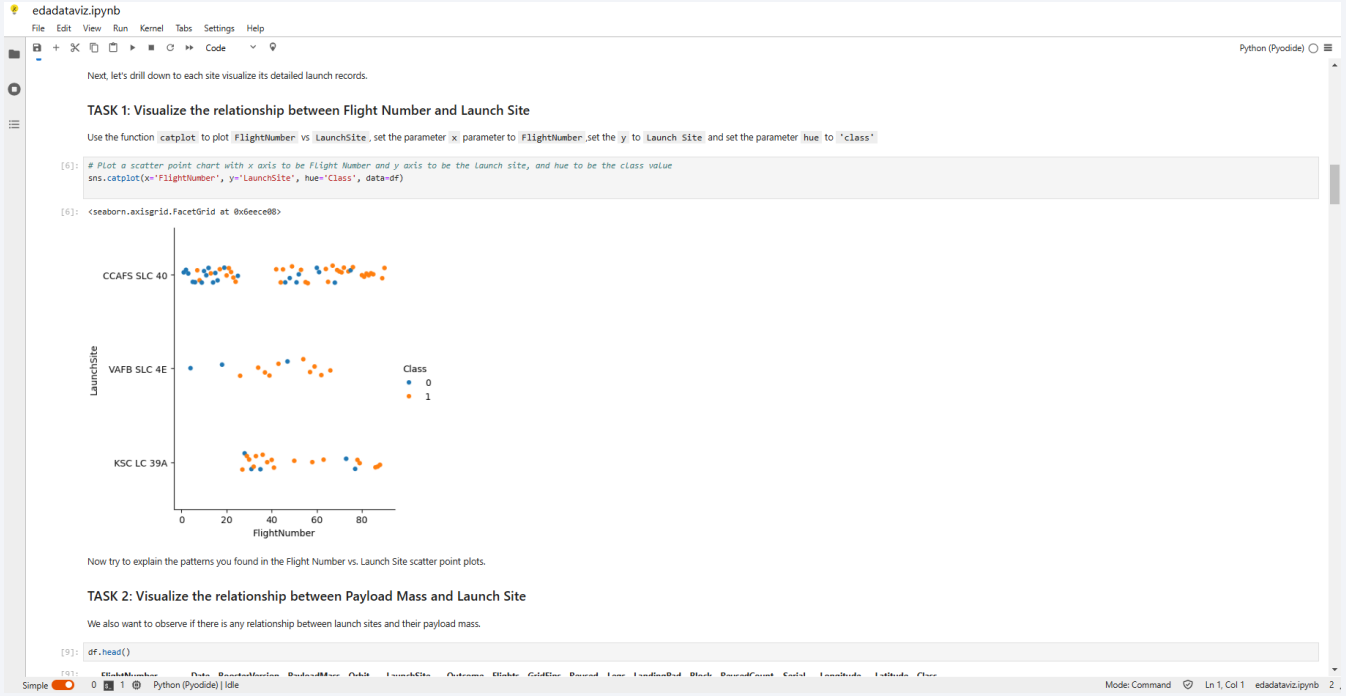
- SpaceX can reuse Falcon 9 first stages, reducing the cost of launches from approximately $165 million to $62 million
- Companies competing with SpaceX need to estimate costs accurately
- Predicting landing success helps in determining when launches will be more cost-efficient
- Companies bidding against SpaceX can use this model to make more competitive pricing decisions
- The machine learning pipeline systematically evaluates multiple classification models to determine which algorithm best predicts whether a first stage will land successfully based on launch parameters and conditions.



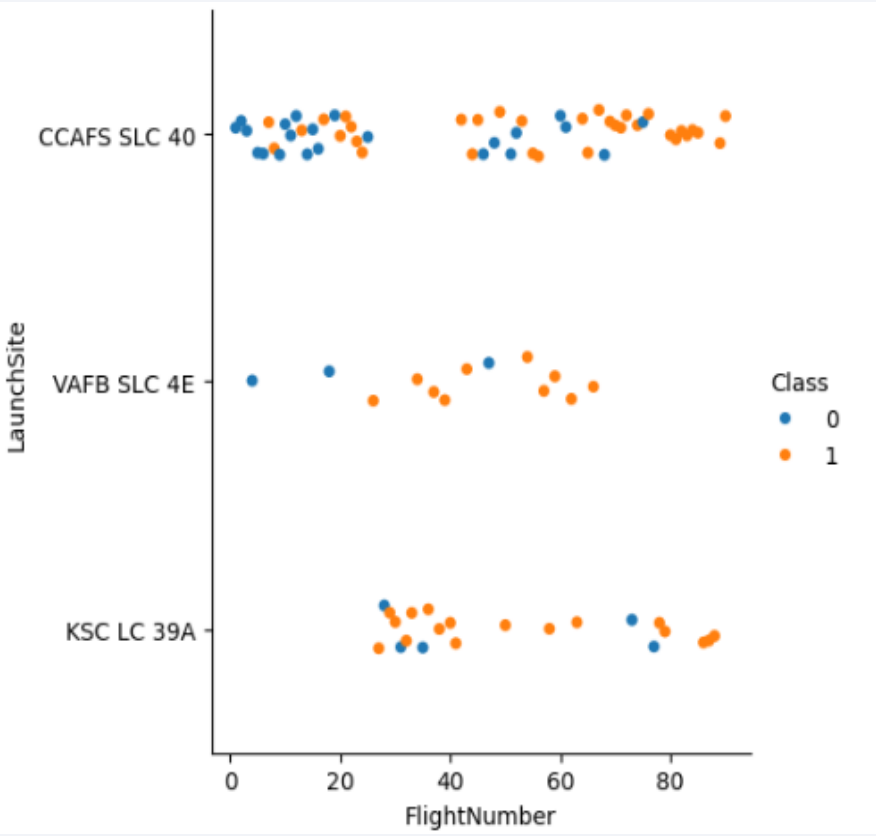

Section 2
Insights drawn from EDA

Flight Number vs. Launch Site

The screenshot of the scatter plot with explanations



Scatter plot of Flight Number vs. Launch Site



Here we can see that the majority of launches were done on CCAFS launch sites. Also share of fails is larger vs the VAFB and KSC launch sites.

Payload vs. Launch Site

The screenshot of the scatter plot with explanations

edadataviz.ipynb

File Edit View Run Kernel Tabs Settings Help

Code

Now try to explain the patterns you found in the Flight Number vs. Launch Site scatter point plots.

TASK 2: Visualize the relationship between Payload Mass and Launch Site

We also want to observe if there is any relationship between launch sites and their payload mass.

[9]:
```
df.head()
```

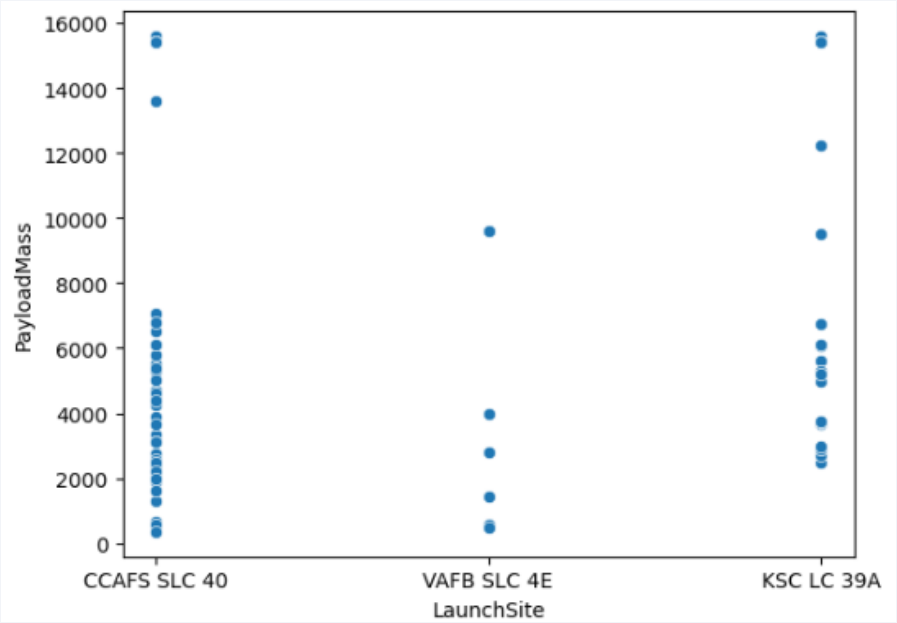
[9]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577366	28.561857	0
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366	28.561857	0
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.577366	28.561857	0
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	0	B1003	-120.610829	34.632093	0
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561857	0

[10]:
```
# 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.scatterplot(x='LaunchSite', y='PayloadMass', data=df)
```

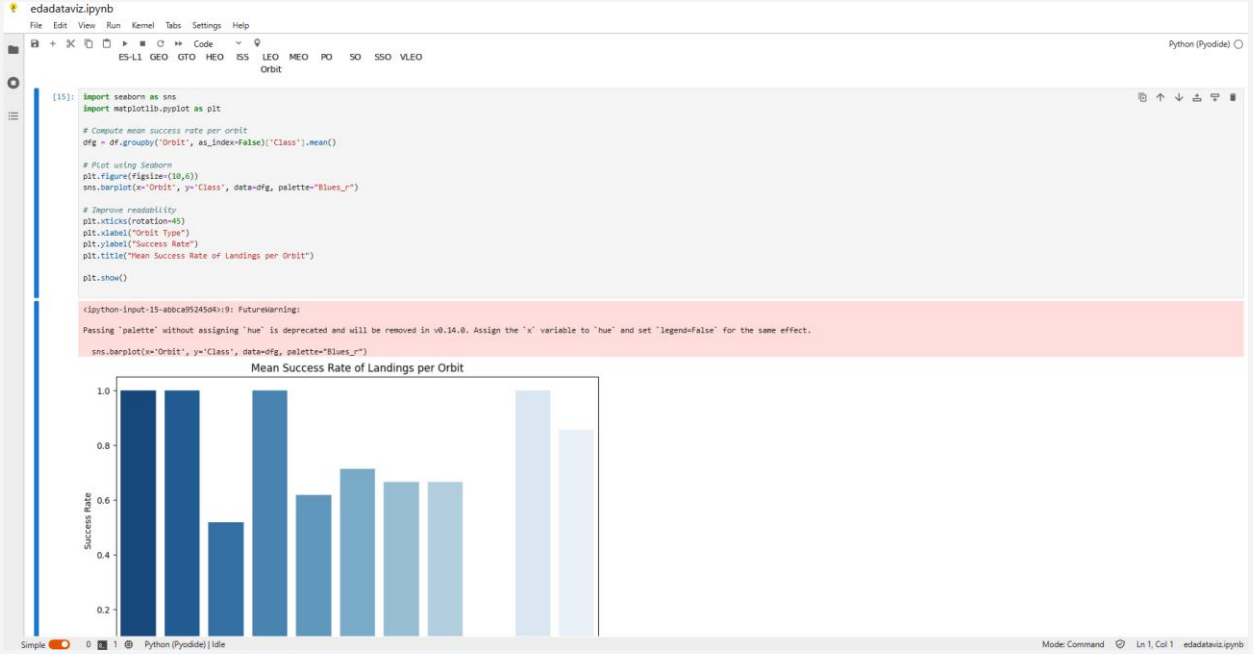
Heavier payloads were launched from KSC launch site.

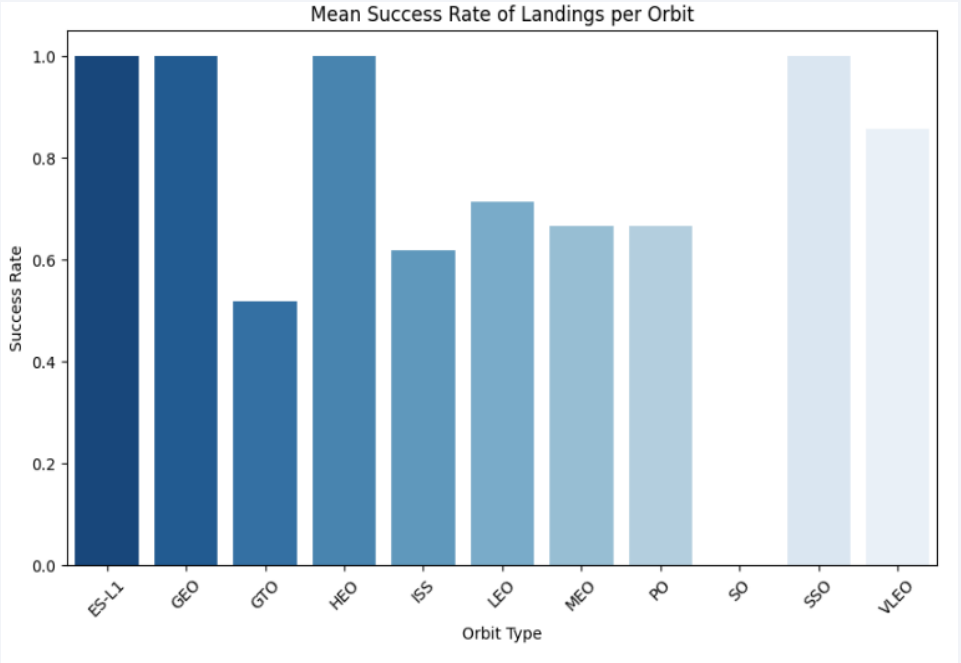
Scatter plot of Payload vs. Launch Site



Success Rate vs. Orbit Type

The screenshot of the plot with explanations

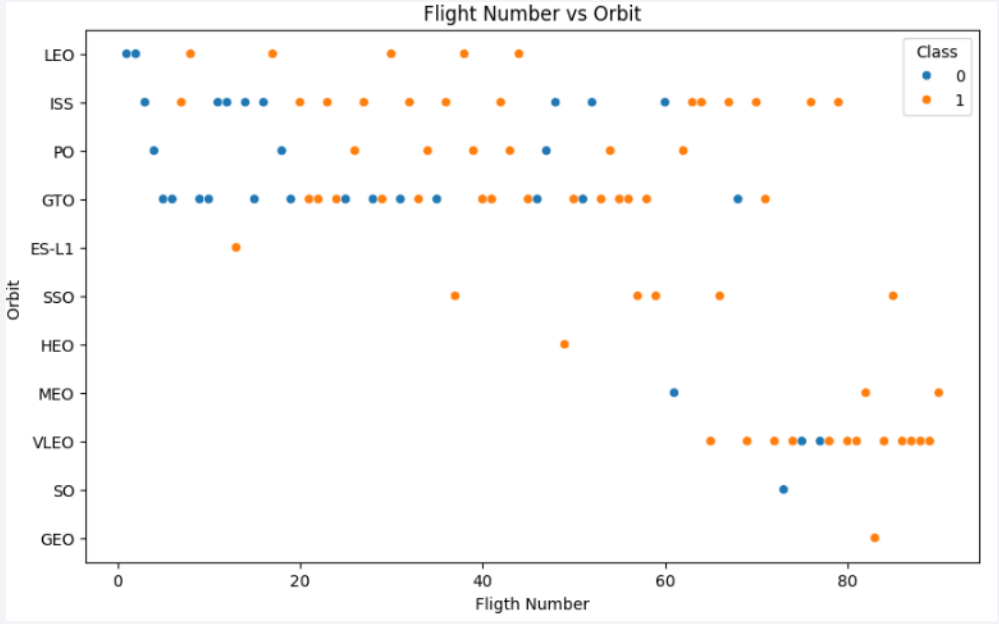




Orbits with least successful launches where GTO, ISS, LEO, MEO, PO.

Flight Number vs. Orbit Type

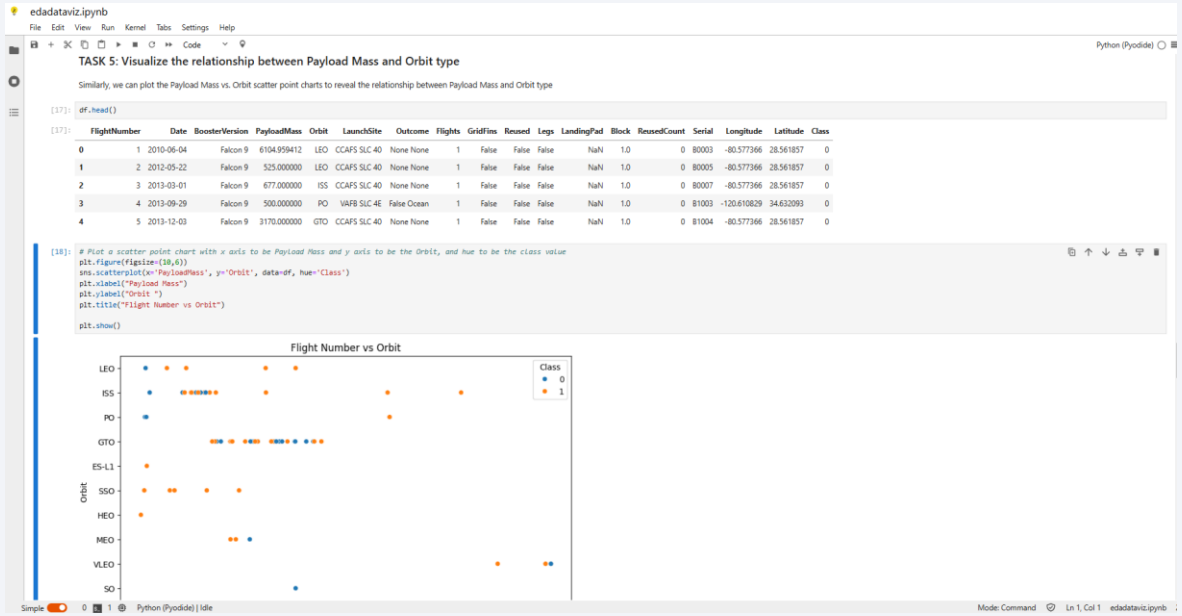
Show the screenshot of the scatter plot

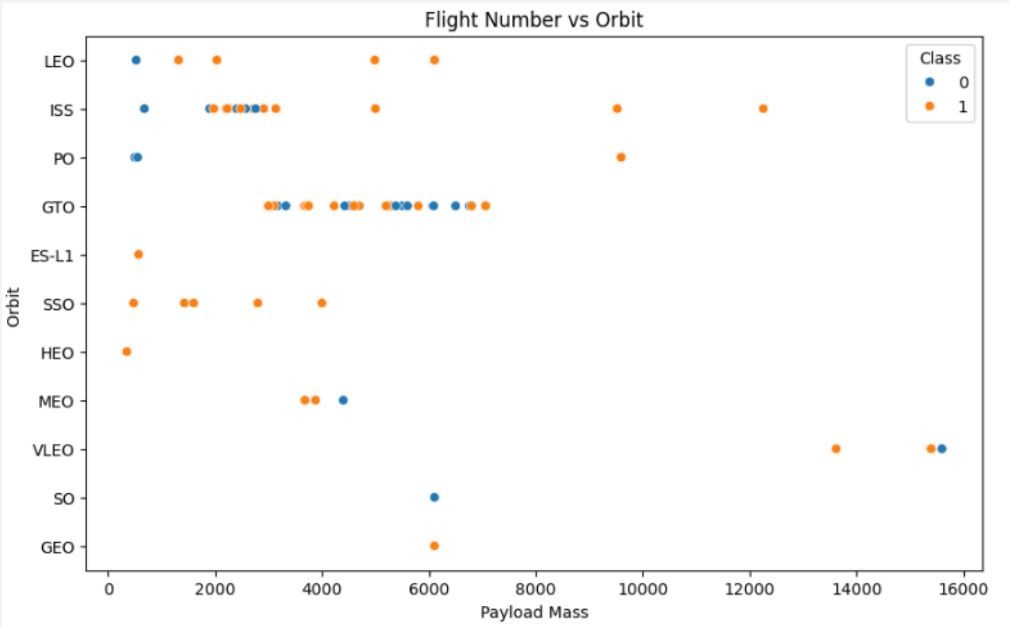


Relatively many flights were done for ISS, GTO and VLEO orbits. Out of which GTO seems to have the largest number of failed launches.

Payload vs. Orbit Type

Show the screenshot of the scatter plot with explanations





Higher payloads seem to have more successful launches. GTO and ISS launches which are least successful operate with lower payloads (mostly lower than 8000 Kg).

Launch Success Yearly Trend

Show a line chart of yearly average success rate

edadataviz.ipynb

File Edit View Run Kernel Tabs Settings Help

Code — Python (Pyodide)

[19]:
```
# A function to Extract years from the date
year=[]
def Extract_year():
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year
Extract_year()
df['Date'] = year
df.head()
```

[19]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
0	1	2010	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577366	28.561857	0
1	2	2012	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366	28.561857	0
2	3	2013	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.577366	28.561857	0
3	4	2013	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	0	B1003	-120.610829	34.632093	0
4	5	2013	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561857	0

[22]:
```
dfg = df.groupby('Date', as_index=False)['Class'].mean()
dfg.head()
```

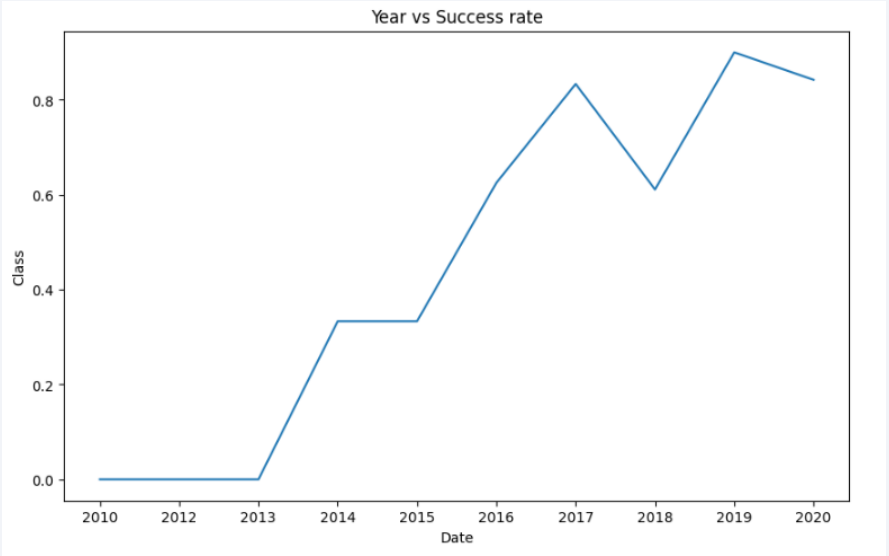
[22]:

	Date	Class
0	2010	0.000000
1	2012	0.000000
2	2013	0.000000
3	2014	0.333333
4	2015	0.333333

[25]:
```
# Plot a line chart with x axis to be the extracted year and y axis to be the success rate
plt.figure(figsize=(10,6))
sns.lineplot(x='Date', y='Class', data=dfg)
plt.xlabel("Date")
plt.ylabel("Class")
plt.title("Year vs Success rate")

plt.show()
```

Year vs Success rate

Simple — Python (Pyodide) | Idle — Mode: Command — Ln 1, Col 1 — edadataviz.ipynb



Success rate shows positive and relatively stable growth over the 2010 – 2020 period.

All Launch Site Names

- The names of the unique launch sites:
  - 'CCAFS LC-40',
  - 'VAFB SLC-4E',
  - 'KSC LC-39A',
  - 'CCAFS SLC-40'
- Here we can see that there are actually four unique launch sites.

Task 1

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

```
statement = 'select distinct "Launch_Site" from SPACEXTABLE'
cur.execute(statement)
output_all=cur.fetchall()
for row_all in output_all:
    print(row_all)
```

```
('CCAFS LC-40',)
('VAFB SLC-4E',)
('KSC LC-39A',)
('CCAFS SLC-40',)
```

Launch Site Names Begin with 'CCA'

Find 5 records where launch sites begin with `CCA`

Task 2

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

```
statement = 'select * from SPACEXTABLE where "Launch_Site" like "CCA%" limit 5'
cur.execute(statement)
output_all=cur.fetchall()
for row_all in output_all:
    print(row_all)
```

```
('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')
```

With the aid of the above query, we can obtain the list of first five records for the launch site, the name of which starts with 'CCA'.

Total Payload Mass

Calculate the total payload carried by boosters from NASA

Task 3

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

```
statement = 'select sum("PAYLOAD_MASS__KG_") from SPACEXTABLE where "Customer" = "NASA (CRS)"'
cur.execute(statement)
output_all=cur.fetchall()
for row_all in output_all:
    print(row_all)
```

```
(45596,)
```

The total payload carried by boosters, where customer was NASA is 45596 Kg.

Average Payload Mass by F9 v1.1

Calculate the average payload mass carried by booster version F9 v1.1

Task 4

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

```
statement = 'select avg("PAYLOAD_MASS__KG_") from SPACEXTABLE where "Booster_Version" = "F9 v1.1"'
cur.execute(statement)
output_all=cur.fetchall()
for row_all in output_all:
    print(row_all)
```

```
(2928.4,)
```

The average payload mass carried by booster version F9 v1.1 is 2928 Kg.

First Successful Ground Landing Date

Find the dates of the first successful landing outcome on ground pad

Task 5

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

Hint:Use min function

```
statement = 'select min("Date") from SPACEXTABLE where "Landing_Outcome" = "Success"'
cur.execute(statement)
output_all=cur.fetchall()
for row_all in output_all:
    print(row_all)
```

```
('2018-07-22',)
```

The first successful landing on ground pad was on 2018-07-22.

Successful Drone Ship Landing with Payload between 4000 and 6000

List the names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000

```
: statement = 'SELECT "Booster_Version" FROM SPACEXTABLE WHERE "Landing_Outcome" = \'Success (drone ship)\' AND "PAYLOAD_MASS__KG_" BETWEEN 4000 AND 6000'
  cur.execute(statement)
  output_all = cur.fetchall()
  for row_all in output_all:
      print(row_all)
```

```
('F9 FT B1022',)
('F9 FT B1026',)
('F9 FT  B1021.2',)
('F9 FT  B1031.2',)
```

We can see the particular names of boosters with mass in range 4000 and 6000 Kg.

Total Number of Successful and Failure Mission Outcomes

Calculate the total number of successful and failure mission outcomes

Task 7

List the total number of successful and failure mission outcomes

```
# statement = 'select count("Landing_Outcome") from SPACEXTABLE where "Landing_Outcome" like \'Suc%\' or \'Fail\' group by \'Landing_Outcome\''

statement = 'select "Landing_Outcome", count("Landing_Outcome") from SPACEXTABLE where "Landing_Outcome" like "Suc%" or "Landing_Outcome" like "Fail%" group by "Landing_Outcome"'
cur.execute(statement)
output_all=cur.fetchall()
for row_all in output_all:
    print(row_all)
```

```
('Failure', 3)
('Failure (drone ship)', 5)
('Failure (parachute)', 2)
('Success', 38)
('Success (drone ship)', 14)
('Success (ground pad)', 9)
```

Here we can see the Failure vs Success mission outcomes in numbers also specified by type of landing.

Boosters Carried Maximum Payload

List the names of the booster which have carried the maximum payload mass

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

```
statement = 'select "Booster_Version" from SPACEXTABLE where "Payload_Mass_Kg" = (select max("Payload_Mass_Kg") from SPACEXTABLE)'
cur.execute(statement)
output_all=cur.fetchall()
for row_all in output_all:
    print(row_all)
```

```
('F9 v1.0  B0003',)
('F9 v1.0  B0004',)
('F9 v1.0  B0005',)
('F9 v1.0  B0006',)
('F9 v1.0  B0007',)
('F9 v1.1  B1003',)
('F9 v1.1',)
('F9 v1.1',)
('F9 v1.1',)
('F9 v1.1',)
('F9 v1.1',)
('F9 v1.1 B1011',)
('F9 v1.1 B1010',)
('F9 v1.1 B1012',)
('F9 v1.1 B1013',)
('F9 v1.1 B1014',)
('F9 v1.1 B1015',)
('F9 v1.1 B1016',)
```

Here we can see the names of boosters that have carried maximum payloads.

2015 Launch Records

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

Task 9

List the records which will display the month names, failure landing_outcomes in drone ship ,booster versions, launch_site for the months in year 2015.

Note: SQLLite does not support monthnames. So you need to use substr(Date, 6,2) as month to get the months and substr(Date,0,5)='2015' for year.

```
statement = '''
SELECT
    CASE
        WHEN SUBSTR("Date", 6, 2) = '01' THEN 'January'
        WHEN SUBSTR("Date", 6, 2) = '02' THEN 'February'
        WHEN SUBSTR("Date", 6, 2) = '03' THEN 'March'
        WHEN SUBSTR("Date", 6, 2) = '04' THEN 'April'
        WHEN SUBSTR("Date", 6, 2) = '05' THEN 'May'
        WHEN SUBSTR("Date", 6, 2) = '06' THEN 'June'
        WHEN SUBSTR("Date", 6, 2) = '07' THEN 'July'
        WHEN SUBSTR("Date", 6, 2) = '08' THEN 'August'
        WHEN SUBSTR("Date", 6, 2) = '09' THEN 'September'
        WHEN SUBSTR("Date", 6, 2) = '10' THEN 'October'
        WHEN SUBSTR("Date", 6, 2) = '11' THEN 'November'
        WHEN SUBSTR("Date", 6, 2) = '12' THEN 'December'
    END AS Month_Name,
    "Landing_Outcome",
    "Booster_Version",
    "Launch_Site"
FROM SPACEXTABLE
WHERE SUBSTR("Date", 0, 5) = '2015'
AND "Landing_Outcome" LIKE 'Failure (drone ship)';
'''

cur.execute(statement)
output_all = cur.fetchall()

for row_all in output_all:
    print(row_all)
```

```
('January', 'Failure (drone ship)', 'F9 v1.1 B1012', 'CCAFS LC-40')
('April', 'Failure (drone ship)', 'F9 v1.1 B1015', 'CCAFS LC-40')
```

Here we can see the missions with names of boosters, their launch sites, and a month in 2015, which were a failure.

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

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

Task 10

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.

```
statement = '''
SELECT
    "Landing_Outcome",
    COUNT(*) AS Outcome_Count,
    RANK() OVER (ORDER BY COUNT(*) DESC) AS Rank
FROM SPACEXTABLE
WHERE "Date" BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY "Landing_Outcome"
ORDER BY Outcome_Count DESC;
'''

cur.execute(statement)
output_all = cur.fetchall()

for row_all in output_all:
    print(row_all)
```

```
('No attempt', 10, 1)
('Success (drone ship)', 5, 2)
('Failure (drone ship)', 5, 2)
('Success (ground pad)', 3, 4)
('Controlled (ocean)', 3, 4)
('Uncontrolled (ocean)', 2, 6)
('Failure (parachute)', 2, 6)
('Precluded (drone ship)', 1, 8)
```

As a result of the query we have got a 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.

Section 3

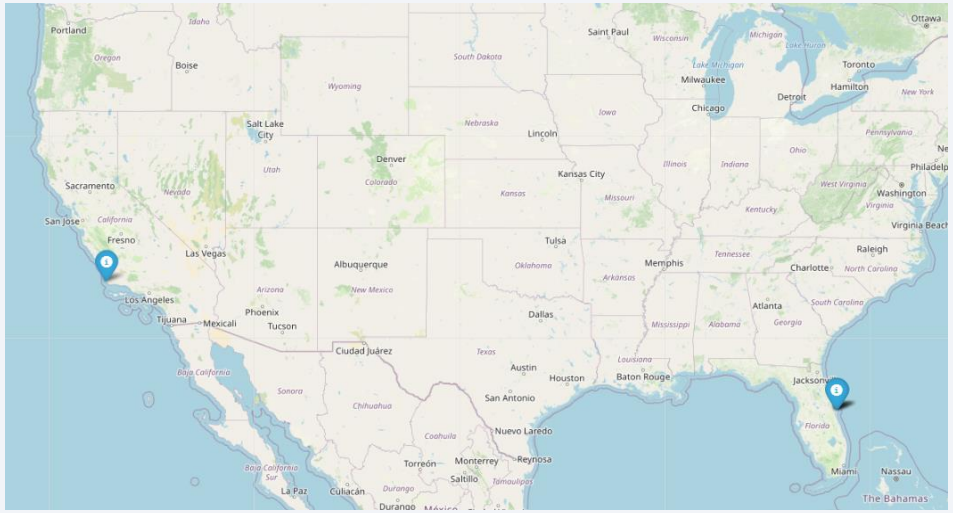
Launch Sites Proximities Analysis

Map of all launch sites

These are coordinates for launch sites:

	Launch Site	Lat	Long
0	CCAFS LC-40	28.562302	-80.577356
1	CCAFS SLC-40	28.563197	-80.576820
2	KSC LC-39A	28.573255	-80.646895
3	VAFB SLC-4E	34.632834	-120.610745

The resulting map shows us two locations for launch sites:



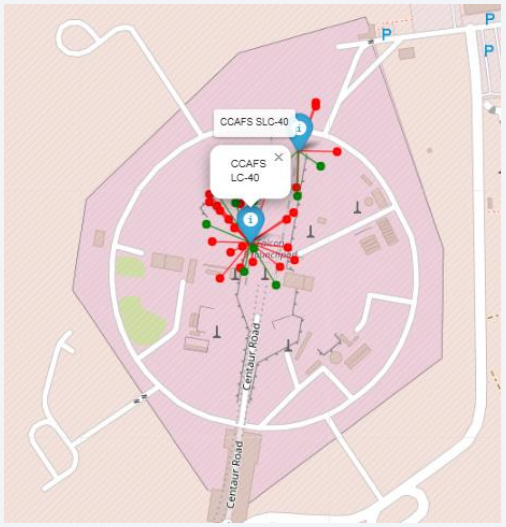
To understand visually where the launch site are, we perform the following code using Folium

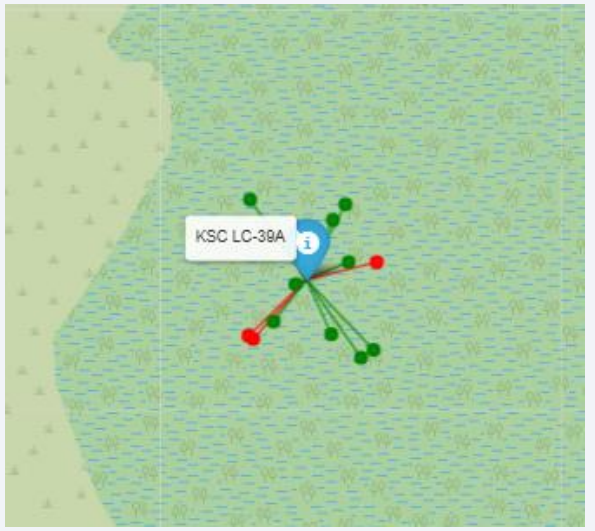
```
# Initial the map
site_map = folium.Map(location=nasa_coordinate, zoom_start=5)
# For each launch site, add a Circle object based on its coordinate (Lat, Long) values. In addition, add Launch site name as a popup label

folium.Circle(coordinate, radius=1000, color='#000000', fill=True).add_child(folium.Popup(...))
folium.map.Marker(coordinate, icon=DivIcon(icon_size=(20,20),icon_anchor=(0,0), html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % 'label', ))
```

```
# Define coordinates for each launch site
launch_sites = {
    "CCAFS LC-40": (28.562302, -80.577356),
    "CCAFS SLC-40": (28.563197, -80.576820),
    "KSC LC-39A": (28.573255, -80.646895),
    "VAFB SLC-4E": (34.632834, -120.610745)
}

# Define map centered around the first launch site
map_center = [28.562302, -80.577356]
launch_map = folium.Map(location=map_center, zoom_start=5)

# Add markers for each launch site with names as labels
for site, (lat, lon) in launch_sites.items():
    folium.Marker(
        location=[lat, lon],
        popup=folium.Popup(site, parse_html=True),
        tooltip=site,  # Tooltip shows name on hover
        icon=folium.Icon(color="blue", icon="info-sign")
    ).add_to(launch_map)

# Display the map
launch_map
```

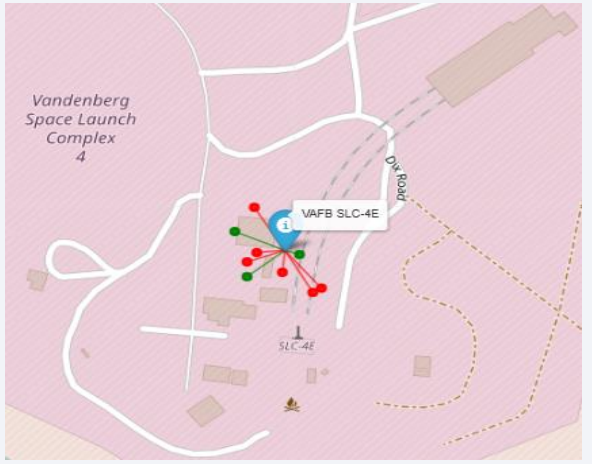
The success/failed launches for each site on the map

- Here we can see the visualized data on success rate at each launch site.
- The most successful launch site is KSC LC-39A
- The least successful launch site is CCAFS LC-40



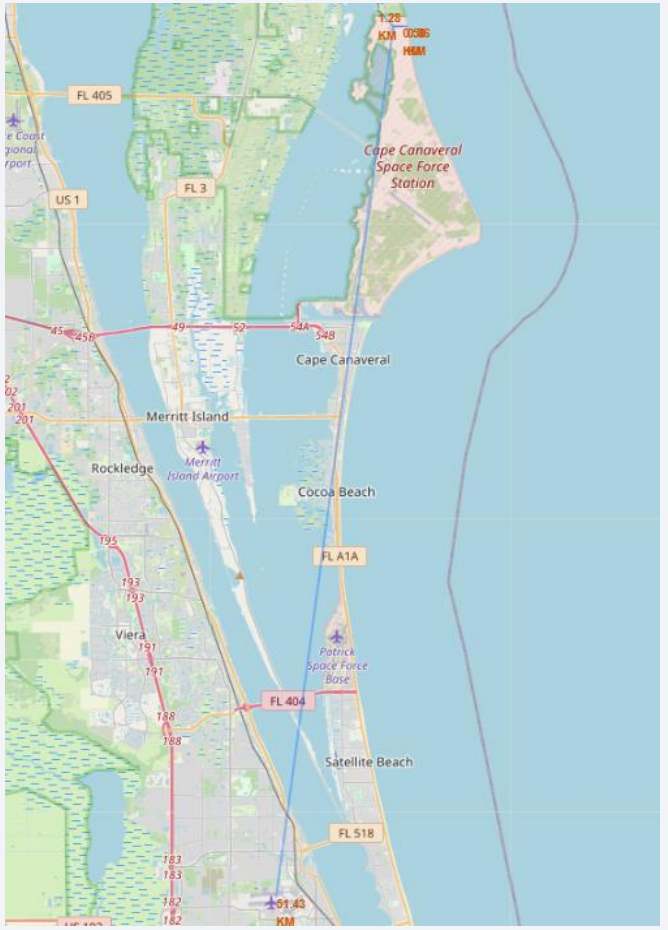


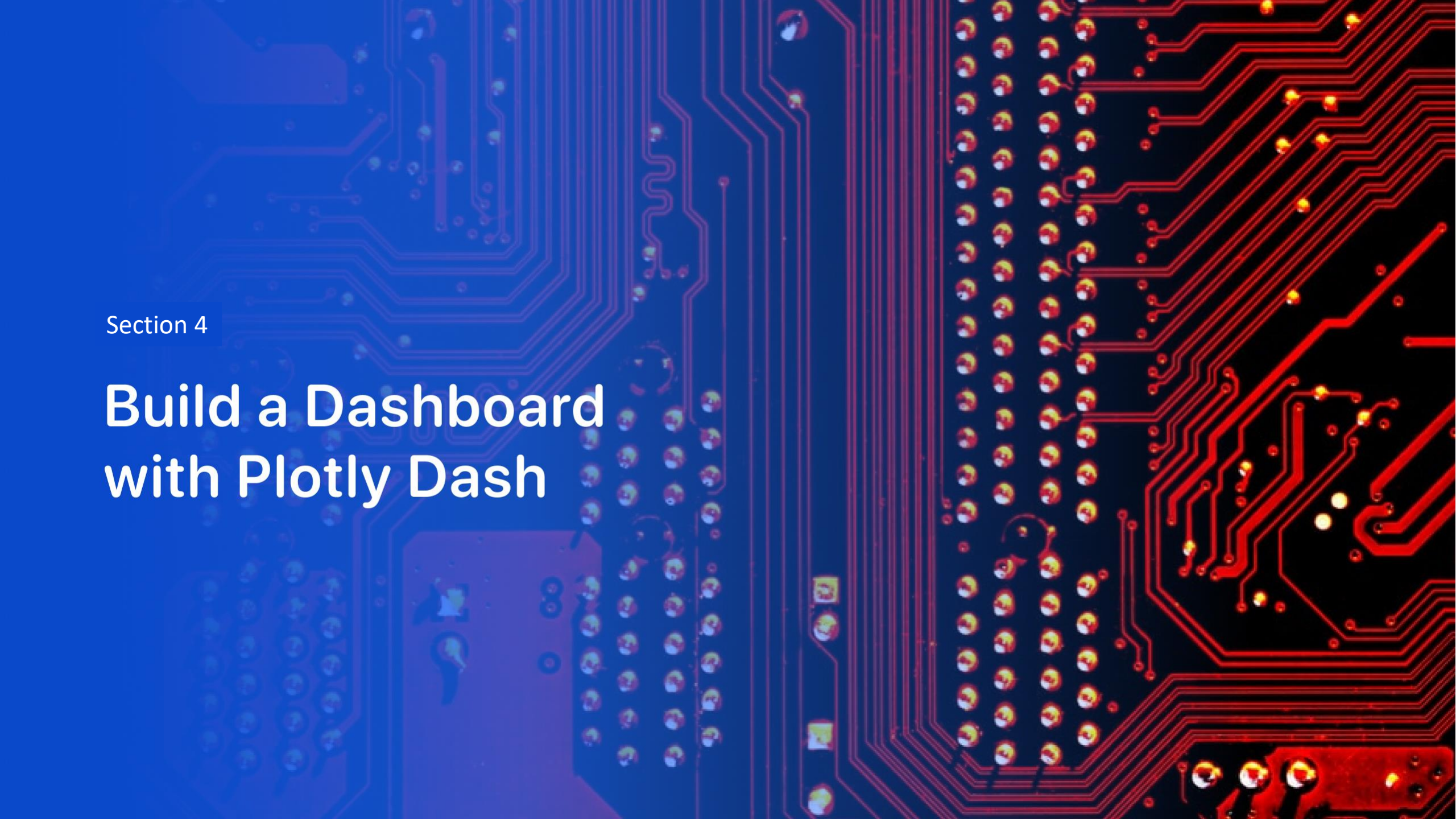
	Launch Site	Success Rate
2	KSC LC-39A	0.769231
1	CCAFS SLC-40	0.428571
3	VAFB SLC-4E	0.400000
0	CCAFS LC-40	0.269231



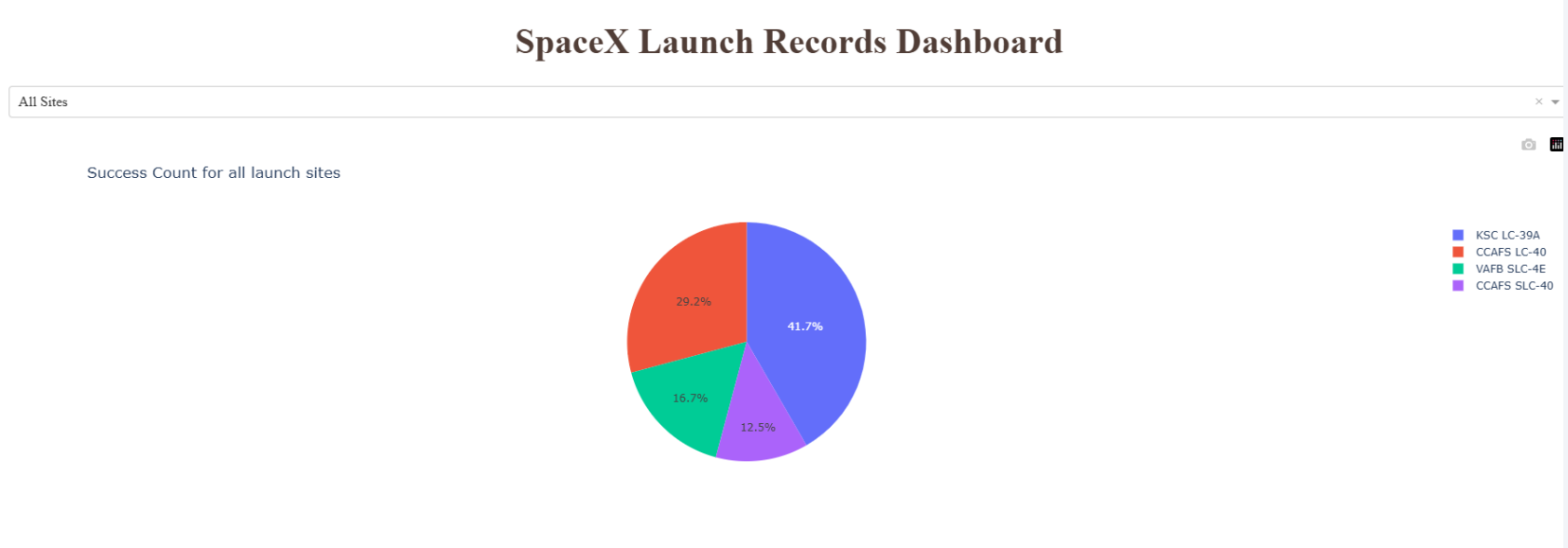
Proximity of the launch site to railway, highway, coastline

- Launch sites are strategically positioned near the equator to leverage Earth's approximately 30km/sec eastward rotation, significantly reducing the fuel needed to achieve orbit.
- They're also deliberately located along coastlines, enabling spacecraft to travel over water during launch. This provides crucial safety advantages: it creates an emergency water landing option during aborts and reduces the risk to populated areas from falling debris.
- These facilities maintain good transportation infrastructure connections. Their proximity to highways facilitates the movement of personnel and equipment, while railway access enables the transportation of large, heavy components essential for launch operations.
- Importantly, launch sites are established at a safe distance from urban centers, minimizing potential hazards to densely populated communities while still maintaining the necessary logistical access for operations.



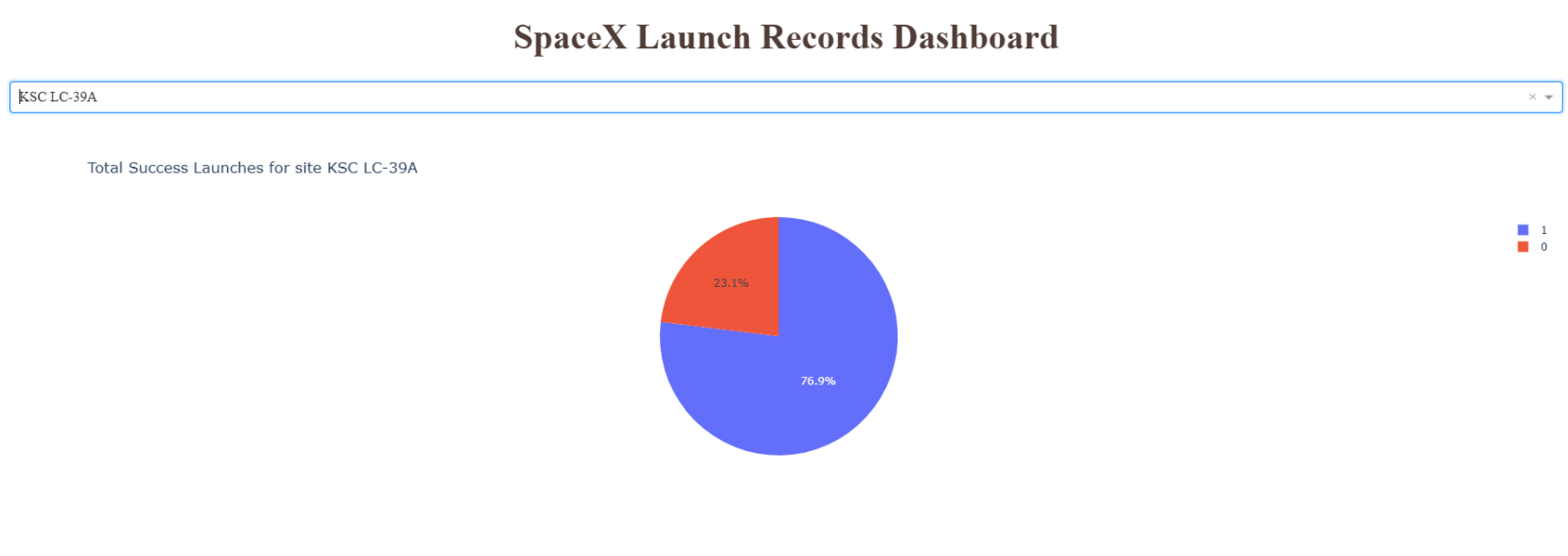

Section 4
Build a Dashboard with Plotly Dash

Dashboard - most successful launch site



Now presenting the data as a dashboard, we can assure ourselves that the most successful launch site is KSC LC-39A, also success shares for other launch sites are presented.

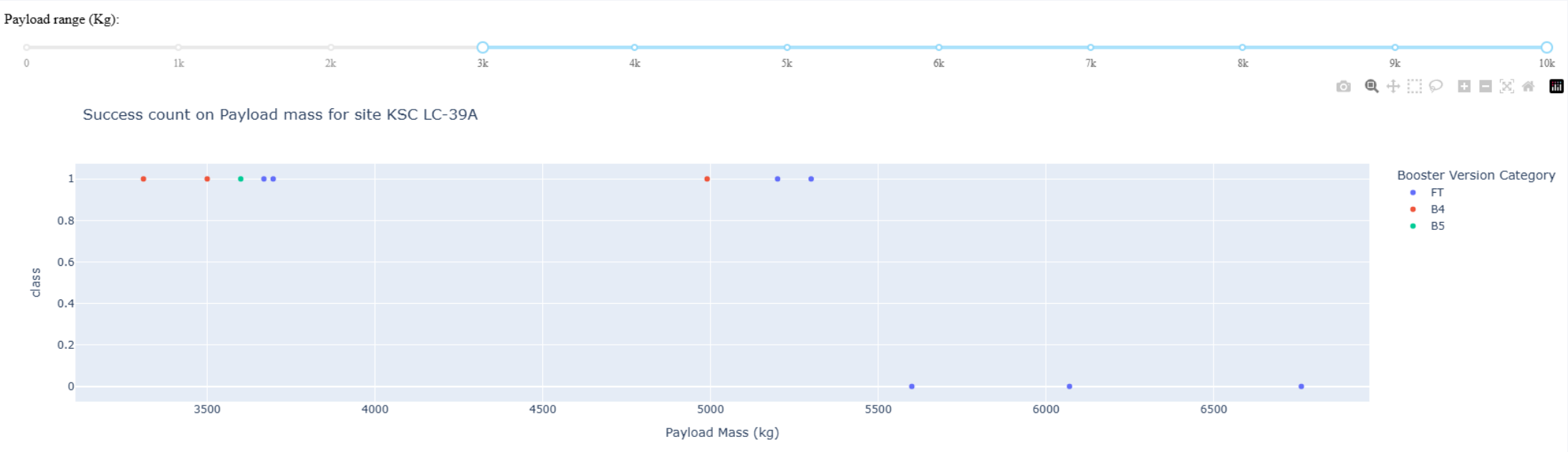
Highest success ratio



Here we can see the screenshot of a pie chart for the launch site with highest launch success ratio. We can choose other launch sites as well.

Success count vs payload mass

We continue with the screenshot of Payload vs. Launch Outcome scatter plot for all sites, with different payload selected in the range slider. We can gather interactively various information. For example, we can see that the most successful booster is B4 for the payload range 3k – 10k Kg.



Section 5

Predictive Analysis (Classification)

Predictive analysis context

Business Context: SpaceX can reuse Falcon 9 first stages, reducing launch costs from ~$165M to ~$62M

Objective: Predict whether the first stage will land successfully

Application: Competitive cost estimation for companies bidding against SpaceX

Approach: Machine learning classification to predict landing success based on launch

Results – Model Comparison

Performance Metrics

Model	Training Accuracy	Test Accuracy
Logistic Regression	84.64%	83.33%
SVM	84.82%	83.33%
Decision Tree	88.93%	83.33%
KNN	84.82%	83.33%

Confusion Matrix Analysis:

- All models showed similar patterns
- 12 True Positives (correctly predicted landing)
- 3 False Positives (predicted landing when actually didn't land)
- Consistent prediction behavior across all models

Analysis of the models

Key Findings

- All models achieved identical test accuracy (83.33%)
- Decision Tree showed higher training accuracy but same test performance
- Model complexity did not improve generalization
- Feature standardization was critical for model performance

Thoughts for improvement

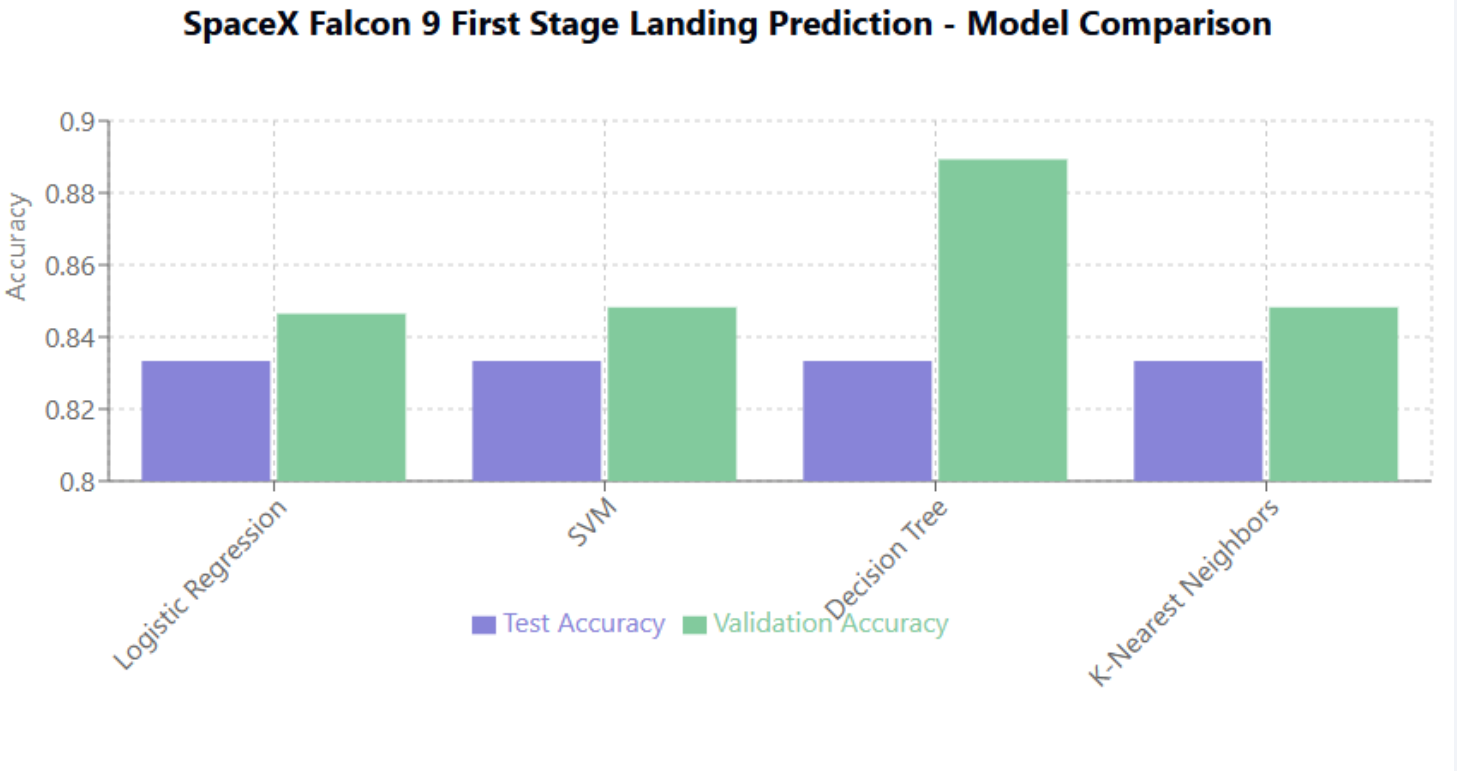
- Focus on feature engineering could potentially improve results
- Ensemble methods might provide additional improvements
- Collecting more data samples could boost model reliability

Business Application

- The model can reliably predict landing success in ~83% of cases
- May be useful for cost estimation and competitive bidding
- Provides strategic advantage when competing with SpaceX

Classification Accuracy

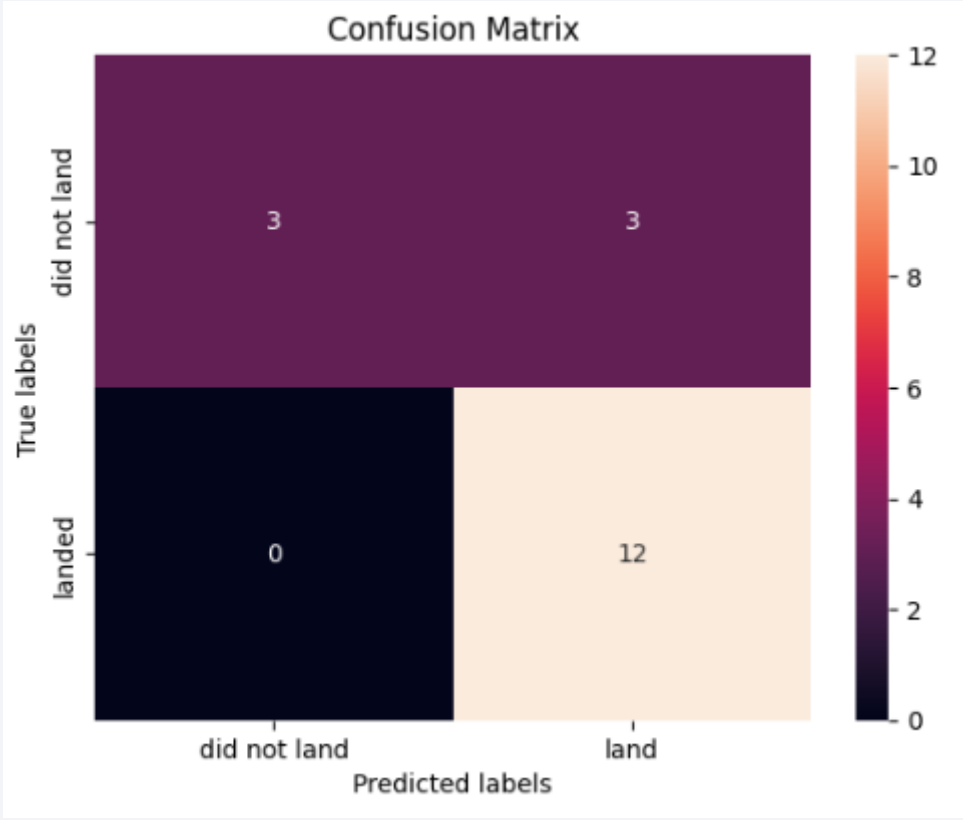
- All models show identical test accuracy of 83.33% on the unseen test data.
- The Decision Tree model has the highest validation accuracy at 88.93%.
- The Decision Tree model outperforms other models during validation, showing better ability to learn from the training data.
- Despite differences in validation accuracy, all models perform equivalently on the test set, suggesting they've found similar decision boundaries for this problem.



Confusion Matrix

The confusion matrix shows how the Decision Tree model performed on the 18 test samples:

- **True Positives (TP) = 12:** The model correctly predicted 12 instances where the first stage landed successfully.
- **True Negatives (TN) = 3:** The model correctly predicted 3 instances where the first stage did not land successfully.
- **False Positives (FP) = 3:** The model incorrectly predicted successful landings in 3 cases where the first stage actually failed to land.
- **False Negatives (FN) = 0:** The model never failed to predict a successful landing when one occurred.



Confusion Matrix

Model Behavior

- Tends to be optimistic in its predictions
- Perfect at detecting actual successful landings
- Sometimes predicts success when failures actually occur
- Never misses a successful landing opportunity
- Half of the actual landing failures are misclassified

Key Performance Metrics

- Overall Accuracy: 83.33% (15/18)
- Precision for landing success: 80% (12/15)
- Recall for landing success: 100% (12/12)
- Precision for landing failure: 100% (3/3)
- Recall for landing failure: 50% (3/6)
- F1 Score: 0.889

Model Summary

- The Decision Tree model achieves good overall performance with 83.33% accuracy on the test set. Its standout characteristic is perfect recall for successful landings, never missing an opportunity when a landing will succeed. The model's errors are exclusively false positives - optimistically predicting successful landings in some cases where failures actually occur.
- This behavior makes the model particularly valuable for applications where identifying all potential successful landing opportunities is the priority, even at the cost of occasional overoptimism about landing success.

Conclusions

To conclude our analysis, the model's prediction pattern has several important business implications:

- **Cost Estimation:** May occasionally underestimate costs by predicting successful landings (and thus first stage recovery) when failures will occur, hence need more data to improve the model.
- **Resource Allocation:** Organizations might sometimes be underprepared for landing failures
- **Competitive Intelligence:** For SpaceX competitors, this model might lead to underestimating operational costs in some scenarios
- **Launch Opportunity Recognition:** The model does not seem to miss conditions favorable for successful landing
- **Risk Management:** Better suited for scenarios where missing a successful landing opportunity is more costly than occasionally predicting success incorrectly

Thank you!

Appendix –Parameters of the models

Machine Learning Pipeline

1. **Data Loading & Preprocessing**: Standardization of features
2. **Train-Test Split**: 80% training, 20% testing data
3. **Model Training**: Four classification algorithms with hyperparameter tuning
4. **Model Evaluation**: Cross-validation and test set performance assessment
5. **Model Selection**: Based on accuracy and generalization ability

Logistic Regression

Model: `LogisticRegression()` from scikit-learn
Hyperparameters tuned:
  - C: [0.01, 0.1, 1] (regularization strength)
  - penalty: ['l2'] (Ridge regularization)
  - solver: ['lbfgs'] (optimization algorithm)
Best parameters: C=0.01, penalty='l2', solver='lbfgs'

Performance
- Training accuracy: 84.64%
- Test accuracy: 83.33%

Characteristics
- Linear model with sigmoid function
- Interpretable coefficients
- L2 regularization prevents overfitting

SVM

Model: `SVC()` from scikit-learn
Hyperparameters tuned:
  - kernel: ['linear', 'rbf', 'poly', 'sigmoid']
  - C: [0.001 to 1000] (regularization parameter)
  - gamma: [0.001 to 1000] (kernel coefficient)
Best parameters: C=1.0, gamma=0.0316, kernel='sigmoid'

Performance
- Training accuracy: 84.82%
- Test accuracy: 83.33%

Characteristics
- Finds optimal hyperplane to separate classes
- Effective in high-dimensional spaces
- Sigmoid kernel provided best performance

Decision Tree Classifier

Model: `DecisionTreeClassifier()` from scikit-learn

Hyperparameters tuned:
- criterion: ['gini', 'entropy'] (split quality measure)
- splitter: ['best', 'random'] (split strategy)
- max_depth: [2-18] (maximum tree depth)
- max_features, min_samples_leaf, min_samples_split

Best parameters: criterion='gini', max_depth=18, max_features='sqrt', min_samples_leaf=4, min_samples_split=10, splitter='random'

Performance
Training accuracy: 88.93% (highest of all models)
Test accuracy: 83.33%

Characteristics
- Non-parametric, captures non-linear relationships
- Higher training accuracy suggests slight overfitting

K-Nearest Neighbors

Model: `KNeighborsClassifier()` from scikit-learn

Hyperparameters tuned:
- n_neighbors: [1-10] (number of neighbors)
- algorithm: ['auto', 'ball_tree', 'kd_tree', 'brute']
- p: [1, 2] (distance metric: Manhattan or Euclidean)

Best parameters: algorithm='auto', n_neighbors=10, p=1 (Manhattan distance)

Performance
- Training accuracy: 84.82%
- Test accuracy: 83.33%

Characteristics
- Non-parametric, instance-based learning
- Manhattan distance preferred over Euclidean
- No explicit training phase