IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Name: Cristina Ortega Trujillo
Date: October 30, 2024

CriselPy (Crisel Nublo 🪻)

Cristina (Crisel Nublo) Ortega | LinkedIn

Outline

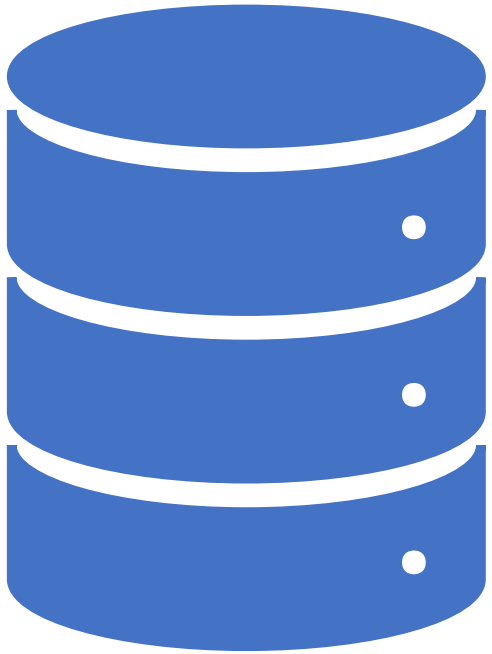
EXECUTIVE SUMMARY

INTRODUCTION

METHODOLOGY

CONCLUSION

Executive Summary



Summary of methodologies

- Data Collection through API
- Data Collection with Web Scraping
- Data Wrangling
- Exploratory Data Analysis with SQL
- Exploratory Data Analysis with Data Visualization
- Interactive Visual Analytics with Folium
- Machine Learning Prediction

Summary of all results

- Exploratory Data Analysis result
- Interactive analytics in screenshots
- Predictive Analytics result

Introduction

Project Background and Context

SpaceX markets Falcon 9 rocket launches on its website at a price of $62 million per launch, significantly lower than the $165 million charged by other providers. A large part of this cost advantage comes from SpaceX's ability to reuse the first stage of the rocket. As such, being able to predict whether the first stage will successfully land is essential to understanding the overall cost-effectiveness of a launch. This information is particularly valuable for other companies that might compete with SpaceX by submitting bids for rocket launches. The primary objective of this project is to develop a machine learning pipeline capable of predicting whether the first stage of a Falcon 9 rocket will land successfully.

Problems to Address

- Key Factors for Successful Landing: What are the critical variables that influence whether the rocket's first stage lands successfully?
- Feature Interactions: How do different factors interact with one another to affect the likelihood of a successful landing?
- Optimal Operating Conditions: What specific conditions need to be met to ensure that the rocket's landing program achieves consistent success?

By addressing these questions, the project aims to enhance the understanding of factors driving successful landings and support better decision-making in rocket launch operations.

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - **Import Libraries and Define Auxiliary Functions**
    - We imported necessary libraries such as **requests**, **pandas**, **numpy**, and **datetime** to handle data requests, manipulation, and analysis.
    - **Pandas options** were set to display all columns and data in a feature for better visibility.
  - **Define Helper Functions**

Several helper functions were defined to extract specific data from the **SpaceX API:**

- **getBoosterVersion(data):** Extracts the booster version using the **rocket** column.
- **getLaunchSite(data):** Extracts the launch site name, longitude, and latitude using the **launchpad** column.
- **getPayloadData(data):** Extracts the payload mass and orbit using the **payloads** column.
- **getCoreData(data):** Extracts various core-related data using the **cores** column.

- **Request and Parse SpaceX Launch Data**
  - The SpaceX API URL for past launches was defined.
  - A GET request was made to the SpaceX API to retrieve the data.
  - The content of the response was checked to ensure it was successful.
  - A static JSON URL was used for consistent results.
  - The response content was decoded as JSON and converted into a pandas dataframe for further analysis.

Methodology

Executive Summary

- Data collection methodology:
  - **Request and Parse the SpaceX Launch Data Using the GET Request**
    - The request was verified to be successful with a 200-status response code.
    - The JSON response was converted into a pandas dataframe.
    - The first 5 rows of the dataframe were printed to inspect the data.
    - The dataframe was subset to keep only relevant columns and filter out rows with multiple cores or payloads.
    - The **date_utc** column was converted to a datetime datatype and the date was extracted.
    - The dates of the launches were restricted to a specific range.
  - **Filter the Dataframe to Only Include Falcon 9 Launches**
    - The dataframe was filtered using the **BoosterVersion** column to keep only Falcon 9 launches.
    - The **FlightNumber** column was reset to ensure sequential numbering.
  - **Data Wrangling**
    - Missing values in the dataset were identified and handled.
    - **None** values in the **LandingPad** column were retained to represent when landing pads were not used.

Methodology

Executive Summary

- Data collection methodology:
  - **Dealing with Missing Values**
    - The mean for the **PayloadMass** column was calculated.
    - **np.nan** values in the **PayloadMass** column were replaced with the calculated mean.
    - It was verified that the number of missing values in the **PayloadMass** column was zero.
  - **Export Data**
    - The cleaned and filtered dataframe was exported to a CSV file for the next section of the project.
- Data Collection - Scraping:
  - **HTML Request:** Sent an HTTP GET request to the Wikipedia page.
  - **HTML Parsing:** Parsed the HTML content using BeautifulSoup.
  - **Table Identification:** Located the specific HTML table with launch records.
  - **Column Extraction:** Extracted column names from the table headers.
  - **Row Iteration:** Iterated through each row of the table to extract data.
  - **Data Cleaning:** Used helper functions to clean and format the extracted data.
  - **Data Structuring:** Organized the cleaned data into a dictionary.
  - **DataFrame Creation:** Converted the dictionary into a structured Pandas DataFrame.
  - **Data Export:** Exported the DataFrame to a CSV file for subsequent analysis

Methodology

Executive Summary

- Data collection Wrangling
  - **Import Libraries:**
    - Imported necessary libraries: **pandas** for data manipulation and **numpy** for numerical operations.
  - **Load Dataset:**
    - Loaded SpaceX dataset from a CSV file.
  - **Data Cleaning:**
    - Identified and calculated the percentage of missing values in each attribute.
    - Determined which columns are numerical and categorical.
  - **Exploratory Data Analysis:**
    - Calculated the number of launches at each site using the **value_counts()** method on the **LaunchSite** column.
    - Calculated the number and occurrence of each orbit using the **value_counts()** method on the **Orbit** column.
    - Calculated the number and occurrence of mission outcomes using the **value_counts()** method on the **Outcome** column.
  - **Create Landing Outcome Labels:**
    - Created a set of outcomes where the second stage did not land successfully.
    - Generated a list where each element is zero if the corresponding row in **Outcome** is in the set of bad outcomes; otherwise, it's one.
    - Assigned this list to the variable **landing_class**.
  - **Add Classification Variable:**
    - Added the **landing_class** variable to the dataset to represent the outcome of each launch (0 for unsuccessful, 1 for successful).
  - **Determine Success Rate:**
    - Calculated the success rate of the first stage landing.
  - **Export Processed Data:**
    - Exported the processed dataset to a CSV file for further analysis.

Methodology

Executive Summary

- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - How to build, tune, evaluate classification models

Data Collection

https://github.com/CriselPy/Spacex-launch-data/blob/main/Collecting%20the%20Data/jupyter-labs-spacex-data-collection-api-v2.ipynb

Import Libraries and Define Auxiliary Functions

- ✓ Import necessary libraries: **Requests, Pandas, Numpy, Datetime.**
- ✓ Set pandas options to display all columns and data in a feature.

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

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

Python

Data Collection

Define Helper Functions

1. **getBoosterVersion(data):** Uses the **rocket** column to call the API and append the booster version to a list.

```python
# Takes the dataset and uses the rocket column to call the API and append the data to the list
def getBoosterVersion(data):
    for x in data['rocket']:
        if x:
            response = requests.get("https://api.spacexdata.com/v4/rockets/"+str(x)).json()
            BoosterVersion.append(response['name'])
```

2. **getLaunchSite(data):** Uses the **launchpad** column to call the API and append the launch site name, longitude, and latitude to lists.

```python
# Takes the dataset and uses the launchpad column to call the API and append the data to the list
def getLaunchSite(data):
    for x in data['launchpad']:
        if x:
            response = requests.get("https://api.spacexdata.com/v4/launchpads/"+str(x)).json()
            Longitude.append(response['longitude'])
            Latitude.append(response['latitude'])
            LaunchSite.append(response['name'])
```

3. **getPayloadData(data):** Uses the **payloads** column to call the API and append the payload mass and orbit to lists.

```python
# Takes the dataset and uses the payloads column to call the API and append the data to the lists
def getPayloadData(data):
    for load in data['payloads']:
        if load:
            response = requests.get("https://api.spacexdata.com/v4/payloads/"+load).json()
            PayloadMass.append(response['mass_kg'])
            Orbit.append(response['orbit'])
```

4. **getCoreData(data):** Uses the **cores** column to call the API and append various core-related data to lists.

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

Request and Parse SpaceX Launch Data

Define the SpaceX API URL for past launches.

1

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

Make a GET request to the SpaceX API.

2

```python
response = requests.get(spacex_url)
```

Check the content of the response.

3

```python
print(response.content)
```

Use a static JSON URL for consistent results.

4

```python
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-Skills
```

Verify the request was successful with a 200-status response code.

5

```python
response = requests.get(static_json_url)
response.status_code
```

Convert the JSON response into a pandas dataframe.

6

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

Print the first 5 rows of the dataframe.

7

```python
# Get the head of the dataframe
data.head()
```

Convert **date_utc** to a datetime datatype and extract the date.

8

```python
# Lets take a subset of our dataframe keeping only the features we want and the flight number, and date_utc.
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]

# We will remove rows with multiple cores because those are falcon rockets with 2 extra rocket boosters and rows that have
data = data[data['cores'].map(len)==1]
data = data[data['payloads'].map(len)==1]

# Since payloads and cores are lists of size 1 we will also extract the single value in the list and replace the feature.
data['cores'] = data['cores'].map(lambda x : x[0])
data['payloads'] = data['payloads'].map(lambda x : x[0])

# We also want to convert the date_utc to a datetime datatype and then extracting the date leaving the time
data['date'] = pd.to_datetime(data['date_utc']).dt.date

# Using the date we will restrict the dates of the launches
data = data[data['date'] <= datetime.date(2020, 11, 13)]
```

Data Collection

Request and Parse SpaceX Launch Data

Subset the dataframe to keep only relevant columns and filter out rows with multiple cores or payloads.

9

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

Python

```python
BoosterVersion
```

Python

```python
# Call getBoosterVersion
getBoosterVersion(data)
```

Python

```python
BoosterVersion[0:5]
```

Python

```python
# Call getLaunchSite
getLaunchSite(data)
```

Python

```python
# Call getPayloadData
getPayloadData(data)
```

Python

```python
# Call getCoreData
getCoreData(data)
```

Python

Data Collection

Request and Parse SpaceX Launch Data

Restrict the dates of the launches and create a Dataframe.

10

```python
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}
```

Python

```python
# Create a data from launch_dict
launch_data_df = pd.DataFrame(launch_dict)
```

Python

```python
# Show the head of the dataframe
launch_data_df.head()
```

Python

Data Collection

Filter the dataframe to only include Falcon 9 launches

Filter the dataframe using the **BoosterVersion** column to keep only Falcon 9 launches.

```python
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = launch_data_df[launch_data_df['BoosterVersion'] != 'Falcon 1']
```

Python

Reset the FlightNumber column.

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

Python

Data Collection

Data Wrangling

- ✓ Identify and handle missing values in the dataset.
- ✓ Retain None values in the LandingPad column to represent when landing pads were not used.

```python
data_falcon9.isnull().sum()
```

```
FlightNumber      0
Date              0
BoosterVersion    0
PayloadMass       5
Orbit             0
LaunchSite        0
Outcome           0
Flights           0
GridFins          0
Reused            0
Legs              0
LandingPad       26
Block             0
ReusedCount       0
Serial            0
Longitude         0
Latitude          0
dtype: int64
```

Data Collection

Dealing with Missing Values

- ✓ Calculate the mean for the PayloadMass column.
- ✓ Replace **np.nan** values in the PayloadMass column with the calculated mean.
- ✓ Verify that the number of missing values in the PayloadMass column is zero.

```python
# Calculate the mean value of PayloadMass column
mean_payload_mass = data_falcon9['PayloadMass'].mean()
print("Mean PayloadMass:", mean_payload_mass)
# Replace the np.nan values with its mean value
data_falcon9.loc[data_falcon9['PayloadMass'].isnull(), 'PayloadMass'] = mean_payload_ma
missing_payload_mass_count = data_falcon9['PayloadMass'].isnull().sum()
print("Missing values in PayloadMass after replacement:", missing_payload_mass_count)
```

Python

```
Mean PayloadMass: 6123.547647058824
Missing values in PayloadMass after replacement: 0
```

Data Collection

[Spacex-launch-data/Collecting the Data/dataset_part_1.csv at main · CriselPy/Spacex-launch-data]

Export Data

Export the cleaned and filtered dataframe to a CSV file for the next section.

```
data_falcon9.to_csv('dataset_part_1.csv', index=False)
```

Python

Data Collection - Scraping

[Spacex-launch-data/Collecting the Data/jupyter-labs-webscraping.ipynb at main · CriselPy/Spacex-launch-data]

Import the necessary packages for web scraping and data manipulation.

Importing Packages

- Imported Packages:
  - ✓ **sys**: System-specific parameters and functions.
  - ✓ **requests:** Perform HTTP requests.
  - ✓ **BeautifulSoup:** Parse HTML and XML documents.
  - ✓ **re:** Regular expressions.
  - ✓ **unicodedata**: Unicode data manipulation.
  - ✓ **pandas:** Data manipulation and analysis.

```python
import sys

import requests
from bs4 import BeautifulSoup
import re
import unicodedata
import pandas as pd
```

Data Collection - Scraping

Define helper functions to process HTML table cells.

Helper Functions

- Functions
  - ✓ date_time(table_cells): Extracts date and time from a table cell.
  - ✓ booster_version(table_cells): Extracts the booster version from a table cell.
  - ✓ landing_status(table_cells): Extracts the landing status from a table cell.
  - ✓ get_mass(table_cells): Extracts the payload mass from a table cell.
  - ✓ extract_column_from_header (row): Extracts the column name from a table header cell.

```python
def date_time(table_cells):
    """
    This function returns the data and time from the HTML  table cell
    Input: the  element of a table data cell extracts extra row
    """
    return [data_time.strip() for data_time in list(table_cells.strings)][0:2]

def booster_version(table_cells):
    """
    This function returns the booster version from the HTML  table cell
    Input: the  element of a table data cell extracts extra row
    """
    out=''.join([booster_version for i,booster_version in enumerate( table_cells.strings) if i%2==0][0:-1])
    return out

def landing_status(table_cells):
    """
    This function returns the landing status from the HTML table cell
    Input: the  element of a table data cell extracts extra row
    """
    out=[i for i in table_cells.strings][0]
    return out


def get_mass(table_cells):
    mass=unicodedata.normalize("NFKD", table_cells.text).strip()
    if mass:
        mass.find("kg")
        new_mass=mass[0:mass.find("kg")+2]
    else:
        new_mass=0
    return new_mass

def extract_column_from_header(row):
    """
    This function returns the landing status from the HTML table cell
    Input: the  element of a table data cell extracts extra row
    """
    if (row.br):
        row.br.extract()
    if row.a:
        row.a.extract()
    if row.sup:
        row.sup.extract()

    colunm_name = ' '.join(row.contents)

    # Filter the digit and empty names
    if not(colunm_name.strip().isdigit()):
        colunm_name = colunm_name.strip()
        return colunm_name
```

Python

Data Collection - Scraping

Perform an HTTP GET request to obtain the Wikipedia HTML page with launch records.

Requesting the HTML Page

Request the HTML page using **requests.get()**.

1

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

```python
response = requests.get(static_url)

if response.status_code == 200:
    print("Request successful!")
else:
    print(f"Request failed with status code: {response.status_code}")
```

```
Request successful!
```

Create a **BeautifulSoup** object from the HTML response.

2

```python
soup = BeautifulSoup(response.text, "html.parser")
```

Verify the page title to ensure it has loaded correctly.

3

```python
print("Page Title:", soup.title.string)
```

```
Page Title: List of Falcon 9 and Falcon Heavy launches - Wikipedia
```

Data Collection - Scraping

Extracting Column Names

Extract the column names from the HTML table containing launch records

Find all tables on the HTML page

1

```python
html_tables = soup.find_all("table", "wikitable")
```

Identify the table containing the launch records.

2

```python
first_launch_table = html_tables[2]
print(first_launch_table)
```

Iterate through the <th> elements to extract column names using the **extract_column_from_header()** function and print.

3

```python
column_names = []

for header in first_launch_table.find_all("th"):  # Iterate over header cells
    name = extract_column_from_header(header)     # Use the provided function
    if name is not None and len(name) > 0:        # Filter out empty headers
        column_names.append(name)
```

```python
print(column_names)
```

```
['Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome']
```

Data Collection - Scraping

Create an empty dictionary with keys from the extracted column names.

Creating a Dictionary for Launch Data

Initialize the dictionary with empty lists for each column

```python
launch_dict= dict.fromkeys(column_names)

# Remove an irrelvant 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']=[]
```

Python

Remove irrelevant columns if necessary

Data Collection - Scraping

Filling the Dictionary with Launch Records

Iterate through the rows of the HTML table to extract launch data and fill the dictionary.

Iterate through the table rows

Check if the row contains a flight number

Extract data from each cell in the row and add it to the dictionary.

```python
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders collapsible")):
   # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number corresponding to launch a number
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
        else:
            flag=False
        #get table element
        row=rows.find_all('td')
        #if it is number save cells in a dictonary
        if flag:
            extracted_row += 1
            launch_dict['Flight No.'].append(flight_number)
            print(flight_number)

            datatimelist=date_time(row[0])

            date = datatimelist[0].strip(',')
            launch_dict['Date'].append(date)
            print(date)

            time = datatimelist[1]
            launch_dict['Time'].append(time)
            print(time)

            bv = booster_version(row[1])
            if not bv:
                bv = row[1].a.string
            launch_dict['Version Booster'].append(bv)
            print(bv)

            launch_site = row[2].a.string
            launch_dict['Launch site'].append(launch_site)
            print(launch_site)

            payload = row[3].a.string
            launch_dict['Payload'].append(payload)
            print(payload)

            payload_mass = get_mass(row[4])
            launch_dict['Payload mass'].append(payload_mass)
            print(payload_mass)

            orbit = row[5].a.string
            launch_dict['Orbit'].append(orbit)
            print(orbit)

            customer = row[6].a.string if row[6].a else None
            launch_dict['Customer'].append(customer)
            print(customer)

            launch_outcome = list(row[7].strings)[0]
            launch_dict['Launch outcome'].append(launch_outcome)
            print(launch_outcome)

            booster_landing = landing_status(row[8])
            launch_dict['Booster landing'].append(booster_landing)
            print(booster_landing)
```

Python

Data Collection - Scraping

[Spacex-launch-data/Collecting the Data/spacex_web_scraped.csv at main · CriselPy/Spacex-launch-data]

Create a Pandas DataFrame from the launch data dictionary.

Creating a DataFrame

Convert the dictionary into a Pandas DataFrame

```python
df= pd.DataFrame({ key:pd.Series(value) for key, value in launch_dict.items() })
```

Python

Export the DataFrame to a CSV file for further use

```python
df.to_csv('spacex_web_scraped.csv', index=False)
```

Python

Data Wrangling

[Spacex-launch-data/Data Wrangling/labs-jupyter-spacex-Data wrangling-v2.ipynb at main · CriselPy/Spacex-launch-data]

Import Libraries and Define Auxiliary Functions

1

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

Python

Data Analysis
Load Space X dataset, from last section

2

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

Python

Identify and calculate the percentage of the missing values in each attribute

3

```python
df.isnull().sum()/len(df)*100
```

Python

Identify which columns are numerical and categorical

4

```python
df.dtypes
```

Python

Data Wrangling

Calculate the number and occurrence of mission outcome of the orbits

Use the method `value_counts()` on the column `LaunchSite` to determine the number of launches on each site:

1

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

Python

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

Use the method `.value_counts()` to determine the number and occurrence of each orbit in the column `Orbit`

2

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

Python

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

Use the method `.value_counts()` on the column `Outcome` to determine the number of `landing_outcomes`. Then assign it to a variable landing_outcomes.

3

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

Python

```
for i,outcome in enumerate(landing_outcomes.keys()):
    print(i,outcome)
```

Python

```
0 True ASDS
1 None None
2 True RTLS
3 False ASDS
4 True Ocean
5 False Ocean
6 None ASDS
7 False RTLS
```

Data Wrangling

Create a landing outcome label from Outcome column

We create a set of outcomes where the second stage did not land successfully:

4

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

Python

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

We create a set of outcomes where the second stage did not land successfully:

1

```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
landing_class = df['Outcome'].apply(lambda x: 0 if x in bad_outcomes else 1).tolist()
```

Python

This variable will represent the classification variable that represents the outcome of each launch. If the value is zero, the first stage did not land successfully; one means the first stage landed successfully.

2

```
df['Class']=landing_class
df[['Class']].head(8)
```

Python

```
df.head(5)
```

Python

We can use the following line of code to determine the success rate:

3

```
df["Class"].mean()
```

Python

```
np.float64(0.6666666666666666)
```

Data Wrangling

[Spacex-launch-data/Data Wrangling/dataset_part_2.csv at main · CriselPy/Spacex-launch-data]

We can now export it to a CSV for the next section, but to make the answers consistent, in the next lab we will provide data in a pre-selected date range

```python
df.to_csv("dataset_part_2.csv", index=False)
```

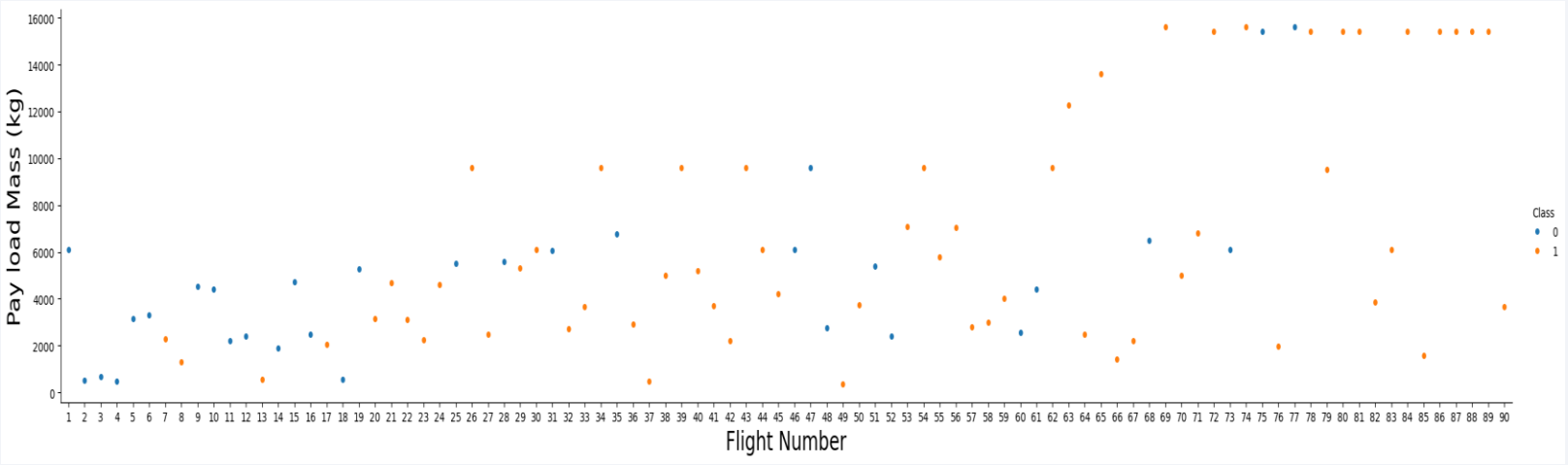
Python

EDA with Data Visualization

https://github.com/CriselPy/Spacex-launch-data/blob/main/Exploratory%20Analysis%20Using%20Pandas%20and%20Matplotlib/jupyter-labs-eda-dataviz-v2.ipynb

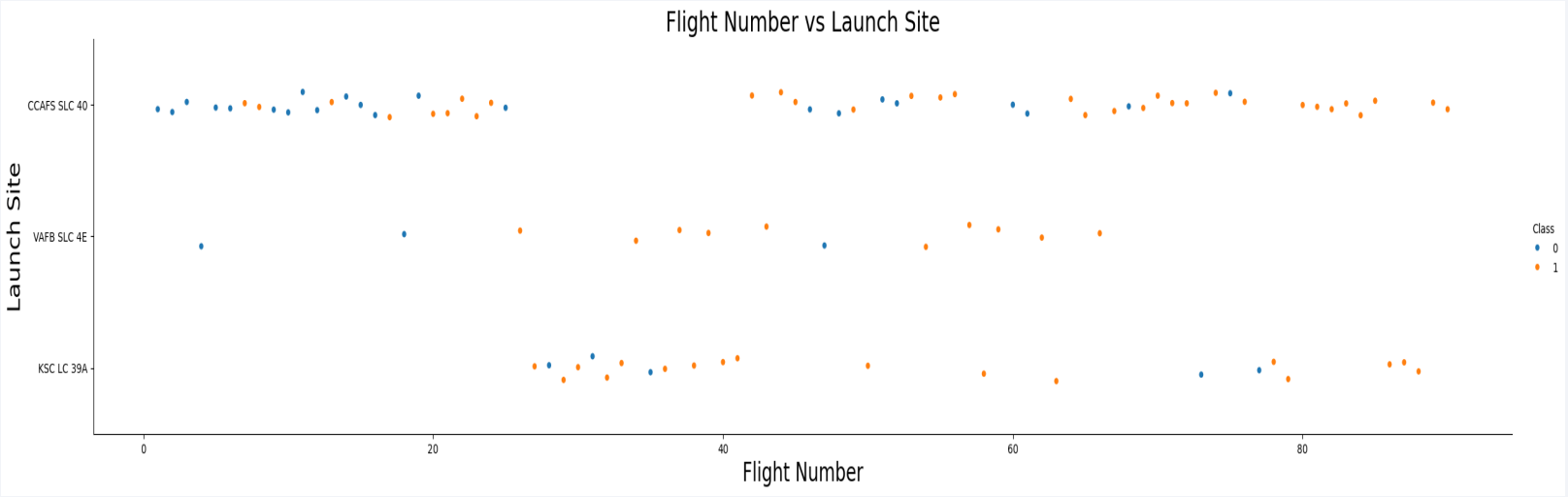
Flight Number vs. Payload Mass:

- **Chart Type:** Scatter plot with hue.
- **Purpose:** To observe the relationship between flight number, payload mass, and launch success. It helps to see if higher flight numbers and payload masses affect the likelihood of a successful landing.



Flight Number vs. Launch Site:

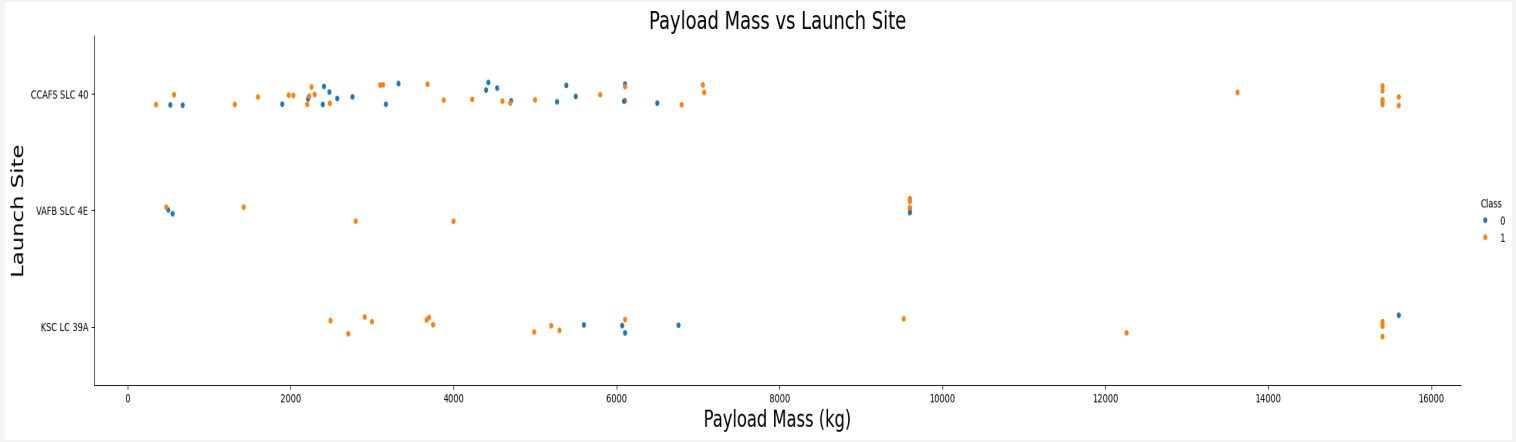
- **Chart Type:** Scatter plot with hue.
- **Purpose:** To visualize the relationship between flight number and launch site, and how it correlates with the success of the launch.



EDA with Data Visualization

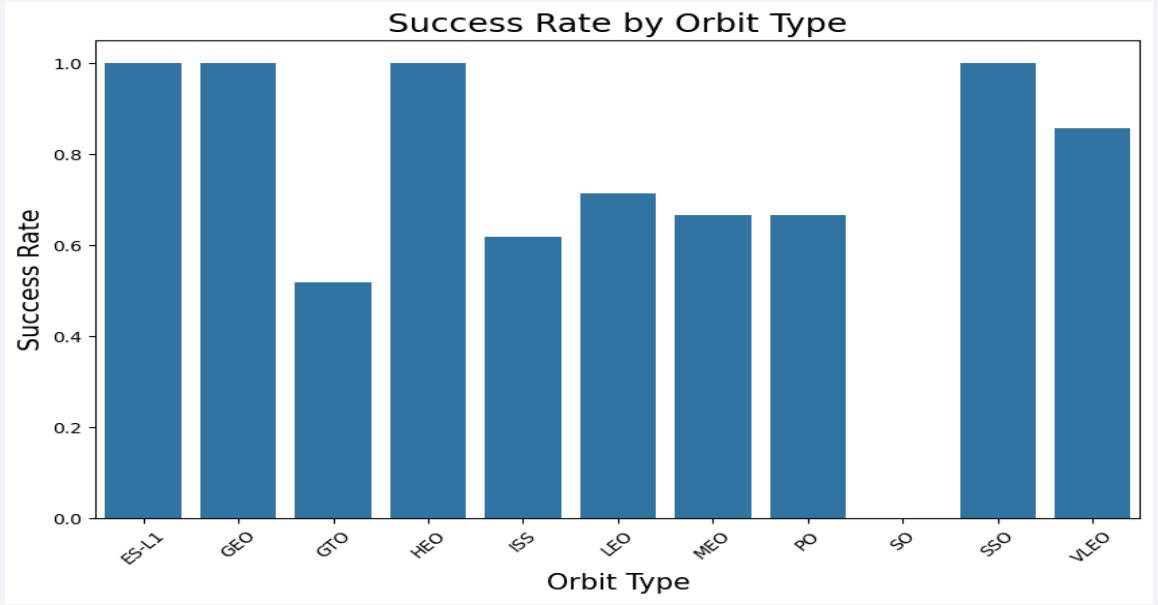
Payload Mass vs. Launch Site:

- **Chart Type**: Scatter plot with hue.
- **Purpose:** To examine the relationship between payload mass and launch site, and how it affects the success of the launch.



Success Rate by Orbit Type:

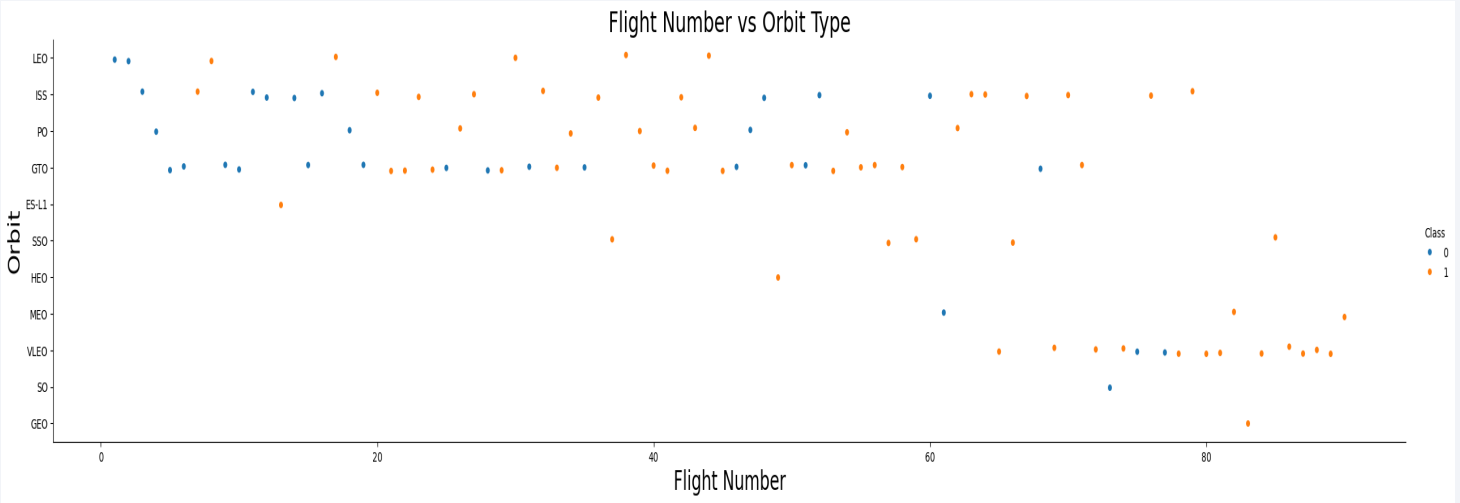
- **Chart Type:** Bar chart.
- **Purpose:** To visualize the success rate for each orbit type, helping to identify which orbits have higher success rates.



EDA with Data Visualization

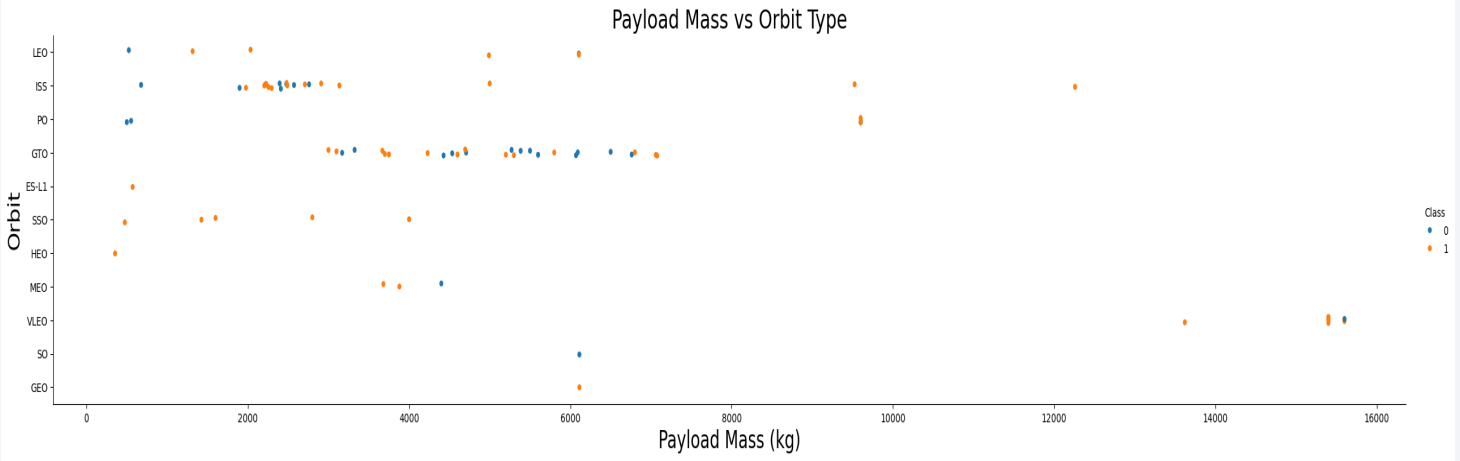
Flight Number vs. Orbit Type:

- Chart Type: Scatter plot with hue.
- Purpose: To explore the relationship between flight number and orbit type, and how it correlates with the success of the launch.



Payload Mass vs. Orbit Type:

- **Chart Type:** Scatter plot with hue.
- **Purpose:** To investigate the relationship between payload mass and orbit type, and how it affects the success of the launch.

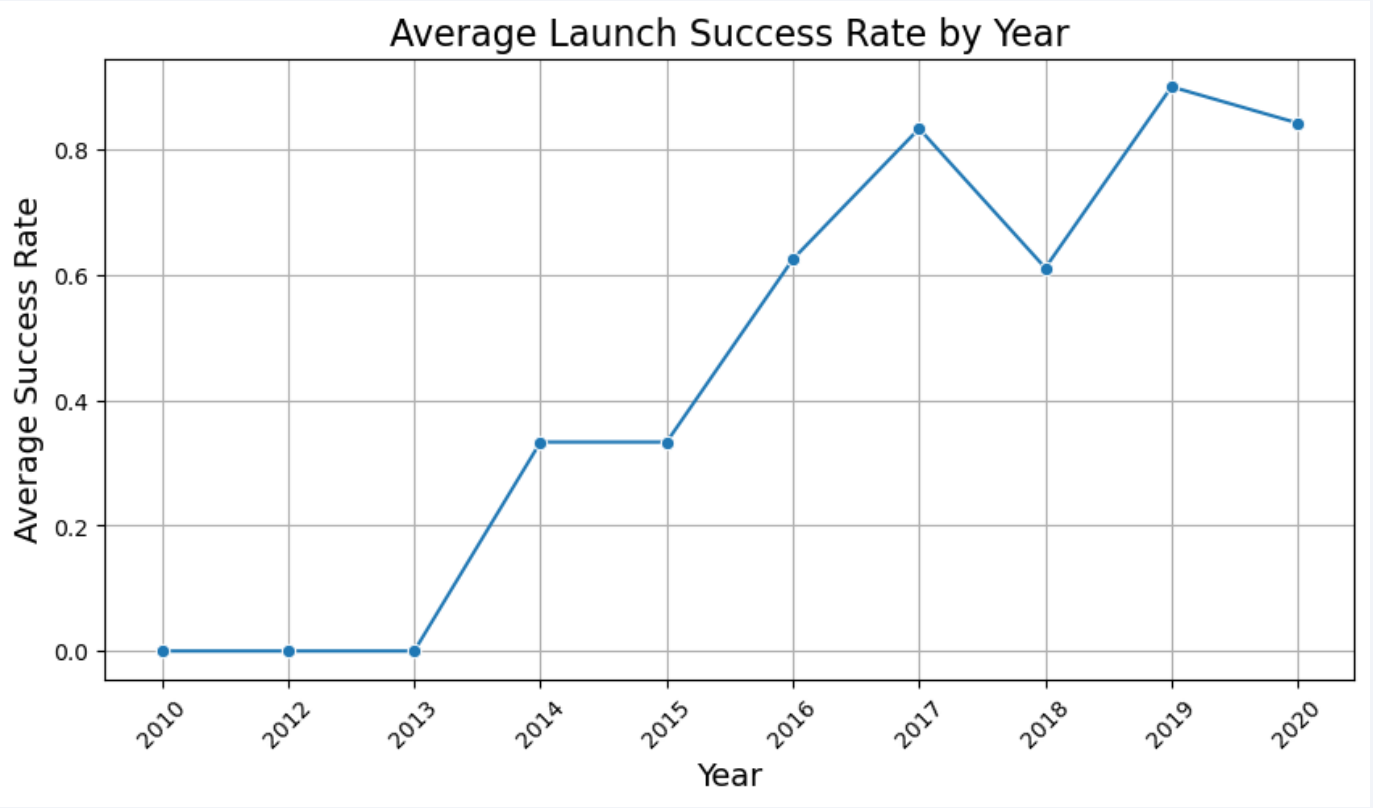


EDA with Data Visualization

Spacex-launch-data/Exploratory Analysis Using Pandas and Matplotlib/dataset_part_3.csv at main · CriselPy/Spacex-launch-data

Average Launch Success Rate by Year:

- Chart Type: Line chart.
- Purpose: To observe the trend of average launch success rates over the years, identifying any patterns or improvements over time.



EDA with SQL

https://github.com/CriselPy/Spacex-launch-data/blob/main/Exploratory%20Analysis%20Using%20SQL/jupyter-labs-eda-sql-coursera_sqllite.ipynb

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

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

Python

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

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

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

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

Python

```
* 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

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

```
%%sql
SELECT SUM("PAYLOAD_MASS__KG_") AS "Total_Payload_Mass"
FROM SPACEXTABLE
WHERE "Customer" = 'NASA (CRS)';
```

Python

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

Total_Payload_Mass
45596

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

```
%%sql
SELECT AVG("PAYLOAD_MASS__KG_") AS "Average_Payload_Mass"
FROM SPACEXTABLE
WHERE "Booster_Version" = 'F9 v1.1';
```

Python

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

Average_Payload_Mass
2928.4

EDA with SQL

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

```
%%sql
SELECT MIN("Date") AS "First_Successful_Landing_Date"
FROM SPACEXTABLE
WHERE "Landing_Outcome" = 'Success (ground pad)';
```

Python

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

First_Successful_Landing_Date
2015-12-22

List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000

```
%%sql
SELECT DISTINCT "Booster_Version"
FROM SPACEXTABLE
WHERE "Landing_Outcome" = 'Success (drone ship)'
  AND "PAYLOAD_MASS__KG_" > 4000
  AND "PAYLOAD_MASS__KG_" < 6000;
```

Python

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

Booster_Version
F9 FT B1022
F9 FT B1026
F9 FT B1021.2
F9 FT B1031.2

List the total number of successful and failure mission outcomes.

```
%%sql
SELECT "Landing_Outcome", COUNT(*) AS "Count"
FROM SPACEXTABLE
GROUP BY "Landing_Outcome";
```

Python

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

Landing_Outcome	Count
Controlled (ocean)	5
Failure	3
Failure (drone ship)	5
Failure (parachute)	2
No attempt	21
No attempt	1
Precluded (drone ship)	1
Success	38
Success (drone ship)	14
Success (ground pad)	9
Uncontrolled (ocean)	2

EDA with SQL

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

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

Python

```
* 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

List the records which display the month names, failure landing outcomes in drone ship, booster versions, and launch site for the months in the year 2015.

```
%%sql
SELECT
    CASE SUBSTR("Date", 6, 2)
        WHEN '01' THEN 'January'
        WHEN '02' THEN 'February'
        WHEN '03' THEN 'March'
        WHEN '04' THEN 'April'
        WHEN '05' THEN 'May'
        WHEN '06' THEN 'June'
        WHEN '07' THEN 'July'
        WHEN '08' THEN 'August'
        WHEN '09' THEN 'September'
        WHEN '10' THEN 'October'
        WHEN '11' THEN 'November'
        WHEN '12' THEN 'December'
    END AS Month,
    "Landing_Outcome",
    "Booster_Version",
    "Launch_Site"
FROM SPACEXTABLE
WHERE "Landing_Outcome" = 'Failure (drone ship)'
  AND SUBSTR("Date", 1, 4) = '2015';
```

Python

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

Month	Landing_Outcome	Booster_Version	Launch_Site
January	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
April	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

Rank the count of landing outcomes between the date 2010-06-04 and 2017-03-20, in descending order.

```
%%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;
```

Python

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

Landing_Outcome	Count
No attempt	10
Success (drone ship)	5
Failure (drone ship)	5
Success (ground pad)	3
Controlled (ocean)	3
Uncontrolled (ocean)	2
Failure (parachute)	2
Precluded (drone ship)	1

Build an Interactive Map with Folium

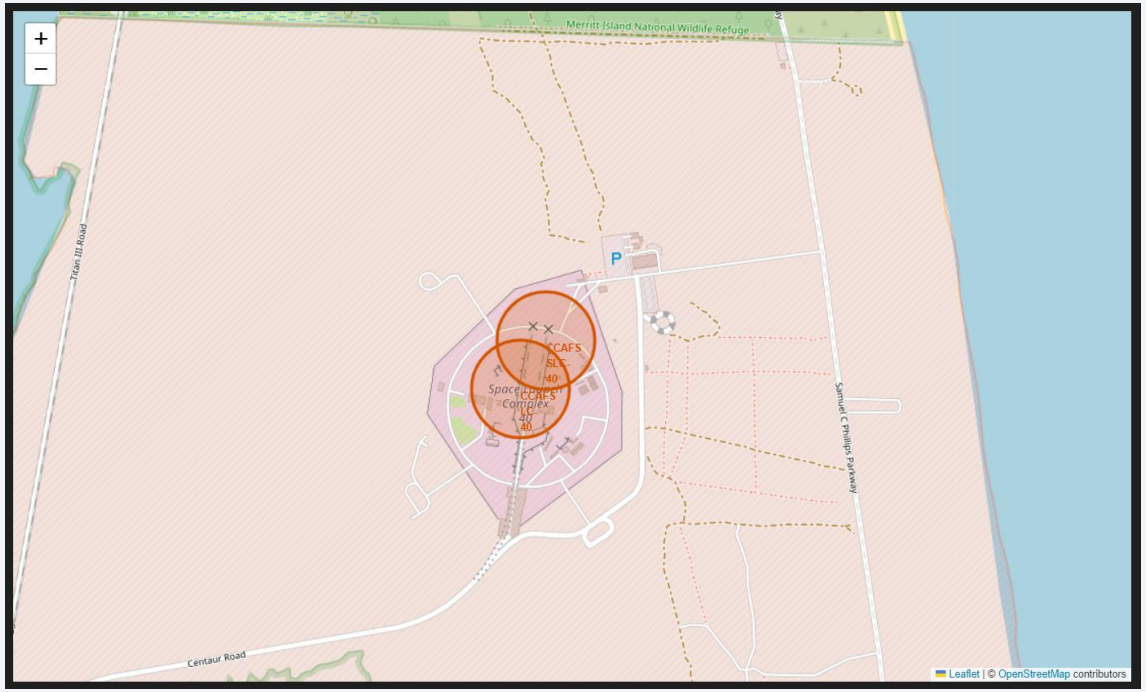
https://github.com/CriselPy/Spacex-launch-data/blob/main/Interactive%20Visual%20Analytics%20and%20Dashboard/lab-jupyter-launch-site-location-v2.ipynb

Markers and Circles for Launch Sites:

- **Markers:** Added markers for each launch site with a text label showing the site name.
- **Circles:** Added circles with a radius of 100 meters around each launch site to highlight their locations.

```python
# 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
for i in range (len(launch_sites_df.index)):
    coordinate = [launch_sites_df["Lat"][i], launch_sites_df["Long"][i]]
    circle = folium.Circle(coordinate, radius=100, color='#d35400', fill=True).add_child(folium.Popup(launch_sites_df["Launch Site"][i]))
    marker = 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>' % launch_sites_df["Launch Site"][i],
            )
        )
    site_map.add_child(circle)
    site_map.add_child(marker)

site_map
```

Python



Reason: To visually identify and label the locations of SpaceX launch sites on the map.

Build an Interactive Map with Folium

MarkerCluster for Launch Outcomes:

- **Markers:** Added markers for each launch record, colored green for successful launches and red for failed launches.

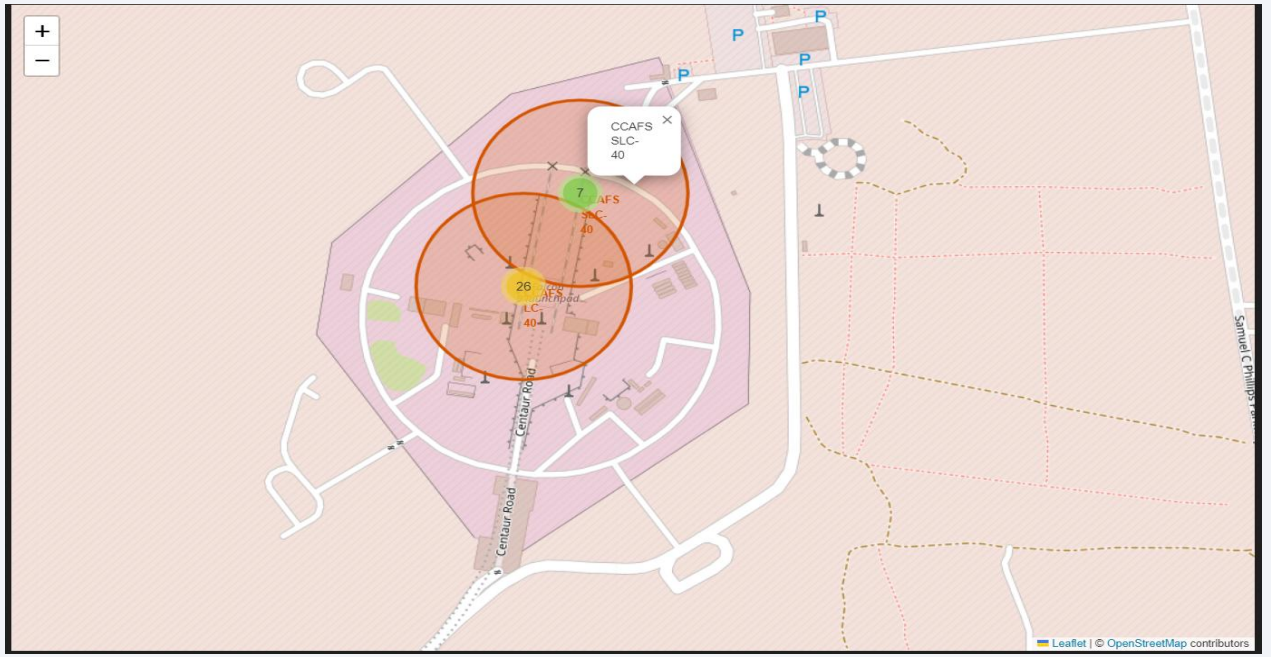
Reason: To visualize the success and failure rates of launches at each site, making it easier to identify patterns.

```python
# Function to assign color to launch outcome
def assign_marker_color(launch_outcome):
    if launch_outcome == 1:
        return 'green'
    else:
        return 'red'

spacex_df['marker_color'] = spacex_df['class'].apply(assign_marker_color)
spacex_df.tail(10)
```

Python

```python
# Add marker_cluster to current site_map
site_map.add_child(marker_cluster)

# for each row in spacex_df data frame
# create a Marker object with its coordinate
# and customize the Marker's icon property to indicate if this launch was successed or failed,
# e.g., icon=folium.Icon(color='white', icon_color=row['marker_color']
for index, record in spacex_df.iterrows():
    # TODO: Create and add a Marker cluster to the site map
    # marker = folium.Marker(...)
    marker = folium.Marker(
        location=[record['Lat'], record['Long']],  # Latitude and Longitude
        icon=folium.Icon(color='white', icon_color=record['marker_color'])  # Set the color based on outcome
    )
    marker_cluster.add_child(marker)

site_map
```

Python



Build an Interactive Map with Folium

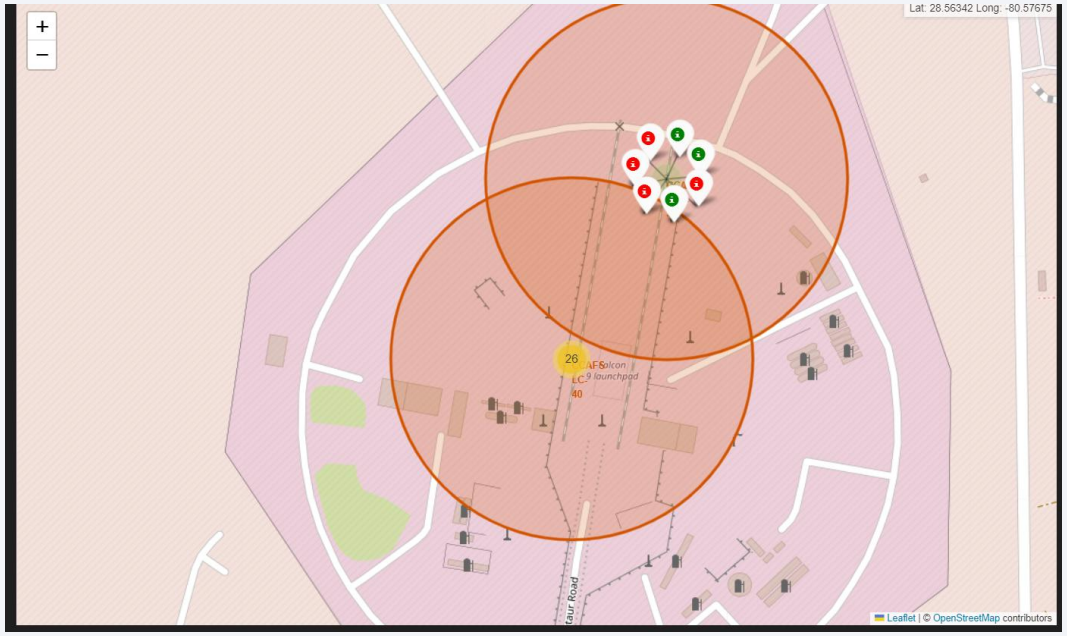
MousePosition Plugin:

- MousePosition: Added a plugin to display the coordinates of the mouse pointer on the map.

```python
# Add Mouse Position to get the coordinate (Lat, Long) for a mouse over on the map
formatter = "function(num) {return L.Util.formatNum(num, 5);};"
mouse_position = MousePosition(
    position='topright',
    separator=' Long: ',
    empty_string='NaN',
    lng_first=False,
    num_digits=20,
    prefix='Lat:',
    lat_formatter=formatter,
    lng_formatter=formatter,
)

site_map.add_child(mouse_position)
site_map
```

Python

Reason: To easily find and mark the coordinates of points of interest (e.g., coastline, railway, highway).

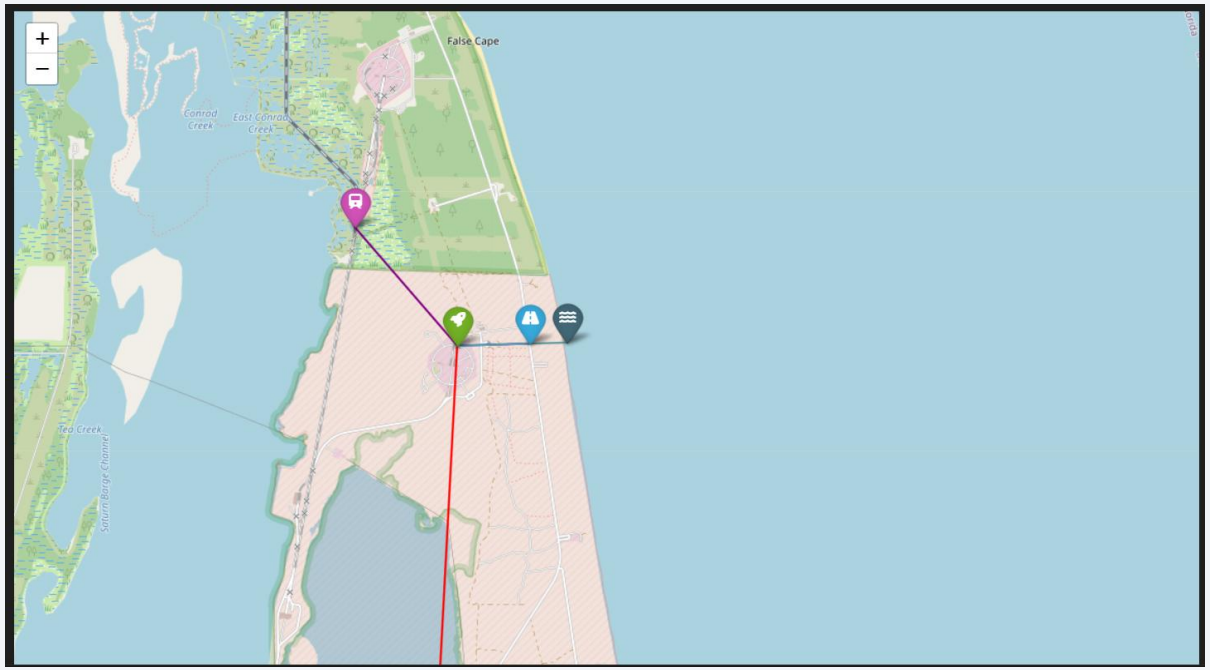


Build an Interactive Map with Folium

[Spacex-launch-data/Interactive Visual Analytics and Dashboard/spacex_launch_geo.csv at main · CriselPy/Spacex-launch-data]

Distance Markers and Lines:

- **Markers:** Added markers at points of interest (e.g., closest coastline, railway, highway, city) with labels showing the distance from the launch site.
- **PolyLines:** Added lines connecting the launch site to these points of interest.




```python
# Coordinates for the launch site and points of interest
launch_site_lat = 28.56341
launch_site_lon = -80.57679
highway_lat = 28.56357
highway_lon = -80.57081
city_lat = 28.07923
city_lon = -80.6051
railway_lat = 28.57221
railway_lon = -80.58528
coastline_lat = 28.56367
coastline_lon = -80.56772

# Function to calculate the distance between two geographical points
def calculate_distance(lat1, lon1, lat2, lon2):
    R = 6373.0  # Approximate Earth radius in km
    lat1, lon1, lat2, lon2 = map(radians, [lat1, lon1, lat2, lon2])
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))
    distance = R * c
    return distance

# Create a map centered on the launch site
site_map = folium.Map(location=[launch_site_lat, launch_site_lon], zoom_start=15)

# List of points of interest with names, colors, and icons
points_of_interest = [
    {"name": "Highway", "lat": highway_lat, "lon": highway_lon, "color": "blue", "icon": "road"},
    {"name": "City", "lat": city_lat, "lon": city_lon, "color": "red", "icon": "building"},
    {"name": "Railway", "lat": railway_lat, "lon": railway_lon, "color": "purple", "icon": "train"},
    {"name": "Coastline", "lat": coastline_lat, "lon": coastline_lon, "color": "cadetblue", "icon": "water"}
]

# Add markers and distance lines for each point of interest
for poi in points_of_interest:
    distance = calculate_distance(launch_site_lat, launch_site_lon, poi['lat'], poi['lon'])

    # Create a marker at the point of interest with an icon and color
    marker = folium.Marker(
        location=[poi['lat'], poi['lon']],
        popup=f"{poi['name']} - {distance:.2f} KM",
        icon=folium.Icon(color=poi['color'], icon=poi['icon'], prefix='fa')
    )
    site_map.add_child(marker)

    # Draw a line between the launch site and the point of interest
    line = folium.PolyLine(
        locations=[[launch_site_lat, launch_site_lon], [poi['lat'], poi['lon']]],
        weight=2,
        color=poi['color']
    )
    site_map.add_child(line)

# Add a marker for the launch site
launch_marker = folium.Marker(
    location=[launch_site_lat, launch_site_lon],
    popup="Launch Site",
    icon=folium.Icon(color='green', icon='rocket', prefix='fa')
)
site_map.add_child(launch_marker)

# Display the map
site_map
```

Reason: To analyze the proximity of launch sites to key infrastructure and geographical features, providing insights into site selection criteria.

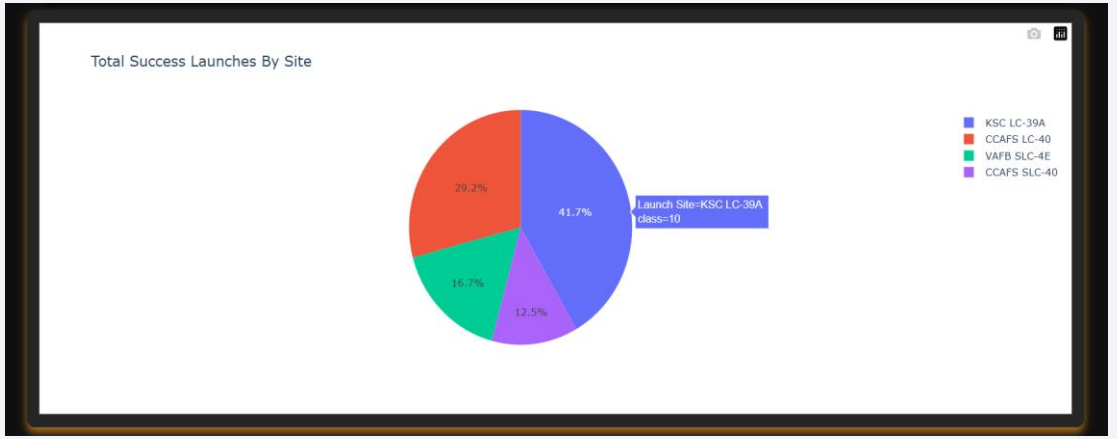
Build a Dashboard with Plotly Dash

https://github.com/CriselPy/Spacex-launch-data/blob/main/SpaceX%20Dash%20App/spacex_dash_app.py

Plots/Graphs

Payload Mass Distribution Plot:
- This plot likely visualizes the distribution of payload masses for SpaceX launches.
- It helps users understand the range and frequency of different payload masses.

Launch Success Rate by Site:
- This plot likely shows the success rate of launches for different launch sites.
- It provides insights into the performance of various launch sites.

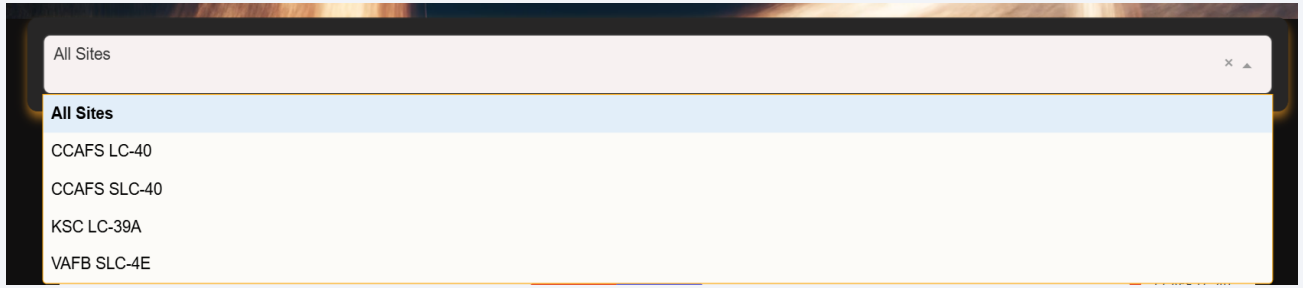


Correlation between Payload and Success:
- This plot likely examines the relationship between payload mass and launch success.
- It helps users analyze if heavier or lighter payloads have higher success rates.

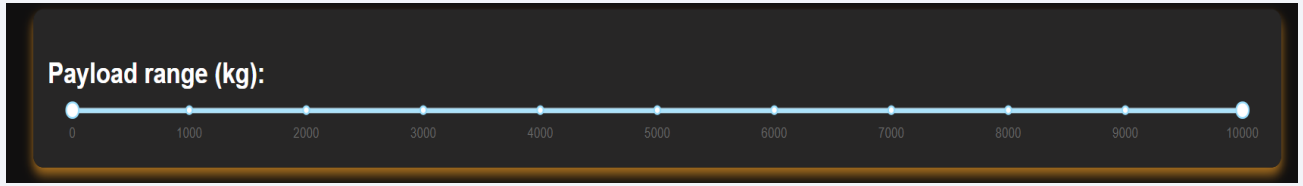
Build a Dashboard with Plotly Dash

Interactions

Dropdown for Launch Site Selection:
- Allows users to filter the data and plots based on the selected launch site.
- Provides a focused view of the performance and statistics for a specific site.



Slider for Payload Mass Range:
- Enables users to adjust the range of payload masses to be displayed in the plots.
- Helps in analyzing trends and patterns within a specific payload mass range.



Build a Dashboard with Plotly Dash

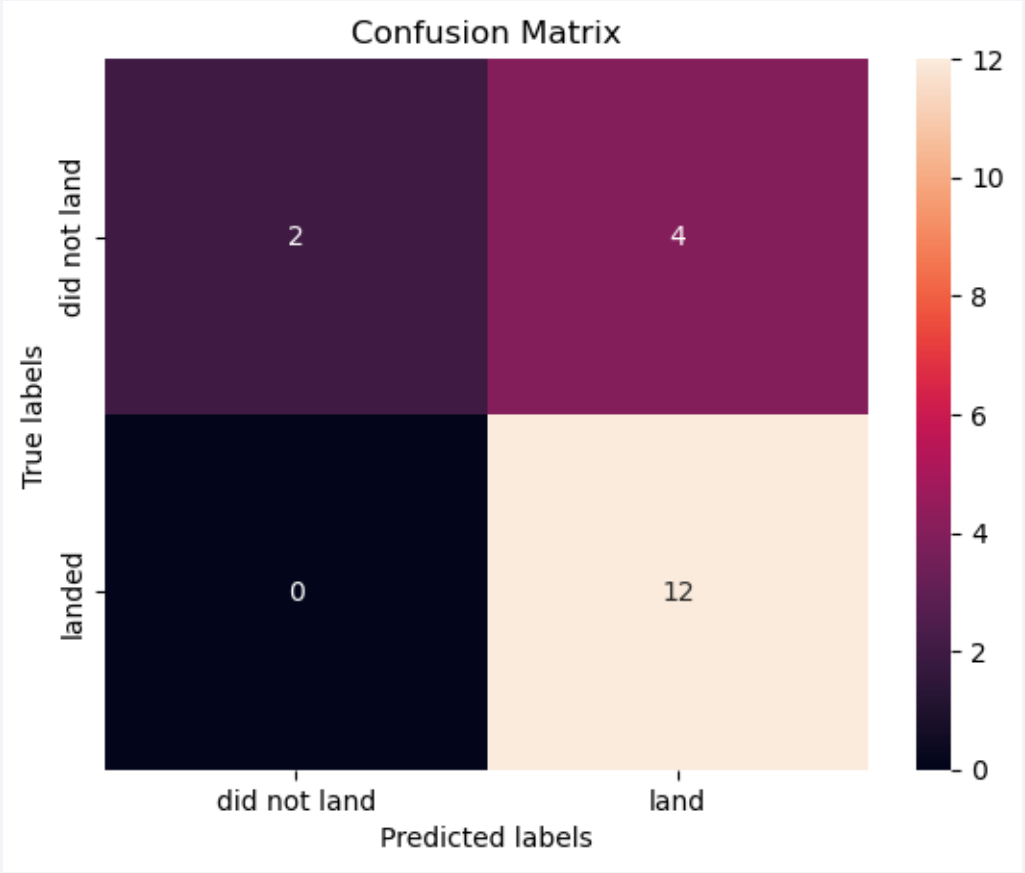
- Reasons for Adding These Plots and Interactions:
  - **Payload Mass Distribution Plot:** To give users an overview of the payload capacities handled by SpaceX.
  - **Launch Success Rate by Site:** To compare the performance of different launch sites and identify any site-specific issues or strengths.
  - **Correlation between Payload and Success:** To investigate if there is any significant impact of payload mass on the success of launches.
  - **Dropdown for Launch Site Selection:** To provide a customizable and interactive experience, allowing users to drill down into specific sites.
  - **Slider for Payload Mass Range:** To enable detailed analysis within user-defined payload mass ranges, making the dashboard more flexible and informative.

Predictive Analysis (Classification)

https://github.com/CriselPy/Spacex-launch-data/blob/main/Predictive%20Analysis%20(Classification)/SpaceX-Machine-Learning-Prediction-Part-5-v1.ipynb

Confusion Matrix

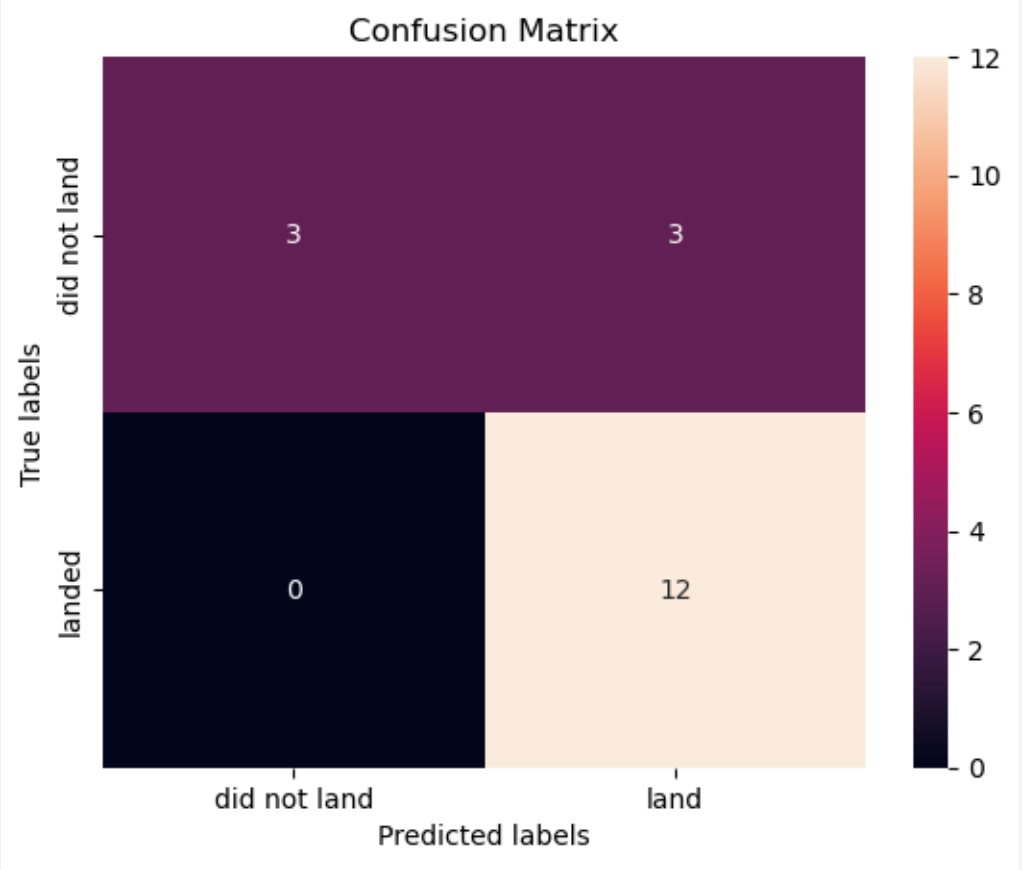
- **Description:** Shows the model's performance in terms of true positives, false positives, true negatives, and false negatives.
- **Key Points:**
  - **True Positives (TP):** Correct predictions of successful landing.
  - **False Positives (FP):** Incorrect predictions of successful landing.
  - **True Negatives (TN):** Correct predictions of no landing.
  - **False Negatives (FN):** Incorrect predictions of no landing.



Predictive Analysis (Classification)

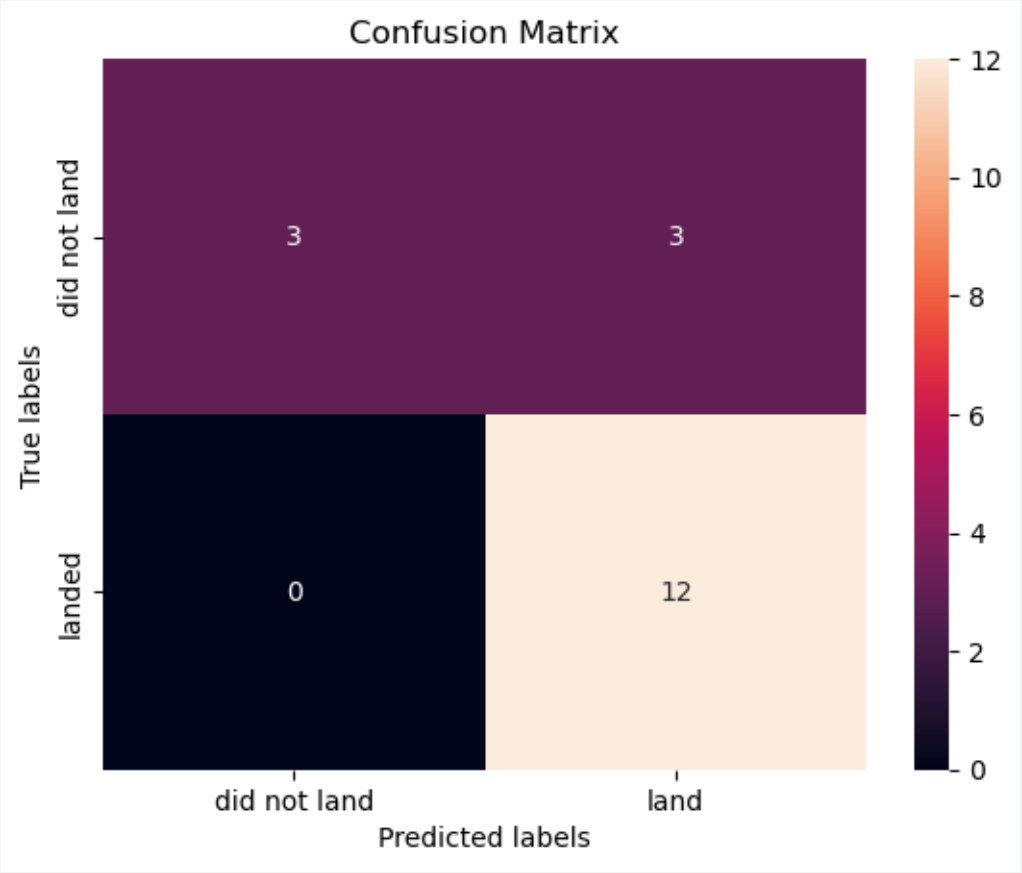
2. ROC Curve (Receiver Operating Characteristic)
Description: Shows the model's ability to distinguish between classes.
Key Points:

- **X-axis** (False Positive Rate): Rate of false positives.
- **Y-axis** (True Positive Rate): Rate of true positives.
- **AUC** (Area Under Curve): Area under the curve; the closer to 1, the better.



Predictive Analysis (Classification)

3. **Precision vs Parameters**
- **Description:** Shows how the model's precision varies with different parameter values.
- **Key Points:**
  - **X-Axis:** Parameter values (e.g., C in SVM).
  - **Y-Axis:** Model precisión.
  - **Curve**: Indicates the model's performance with different parameter configurations.



Predictive Analysis (Classification)

- We imported the dataset using numpy and pandas, processed it, and divided it into training and testing sets.
- We developed various machine learning models and adjusted their hyperparameters with GridSearchCV.
- We evaluated the models using accuracy as the performance metric and enhanced the results through feature engineering and tuning algorithms.
- Ultimately, we identified the classification model with the best performance.

Results

Exploratory Data Analysis Results

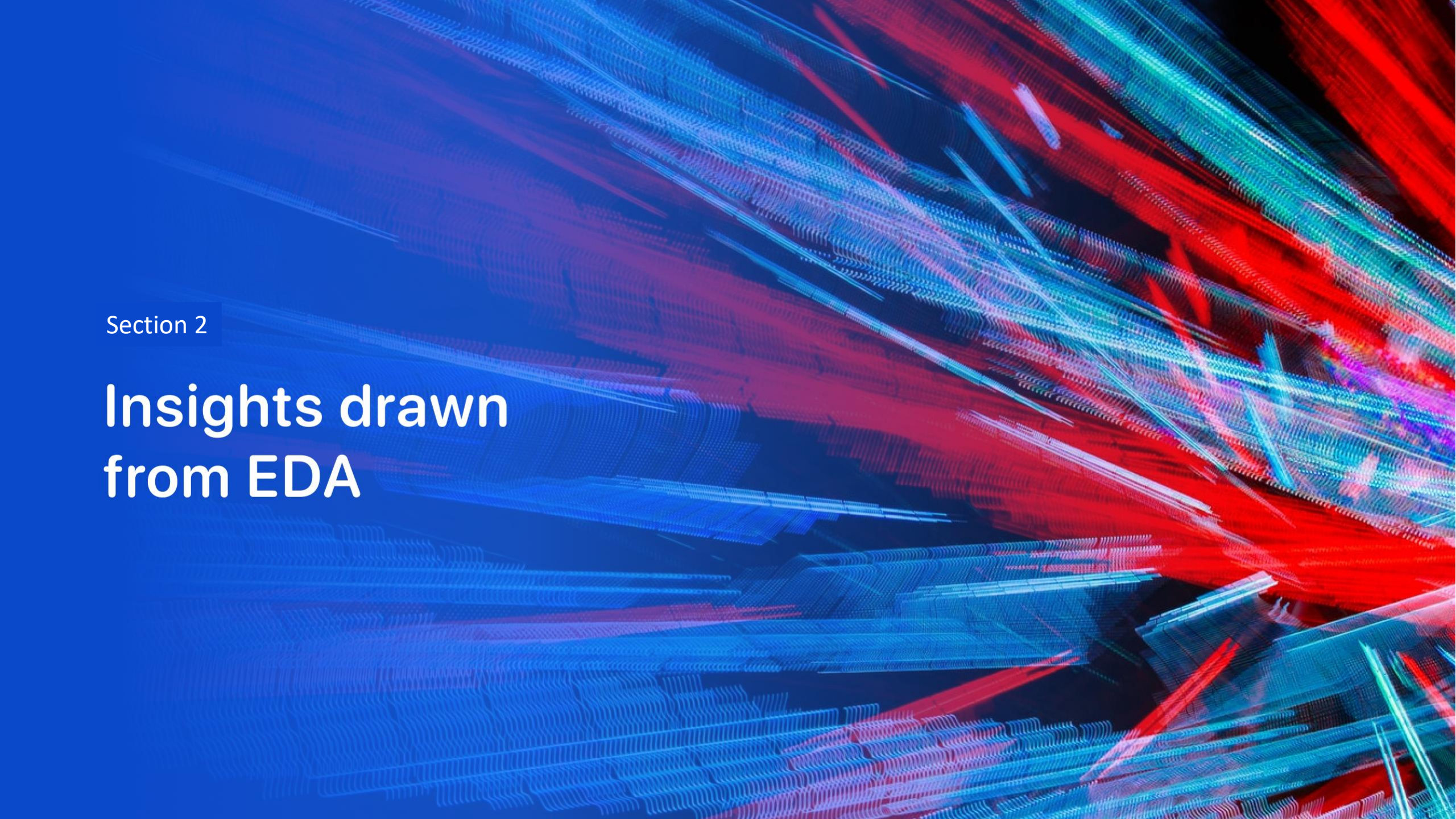
- Analyzed the relationship between flight number, payload mass, and launch success.
- Visualized launch success rates across different launch sites and orbit types.
- Observed trends in launch success rates over the years.

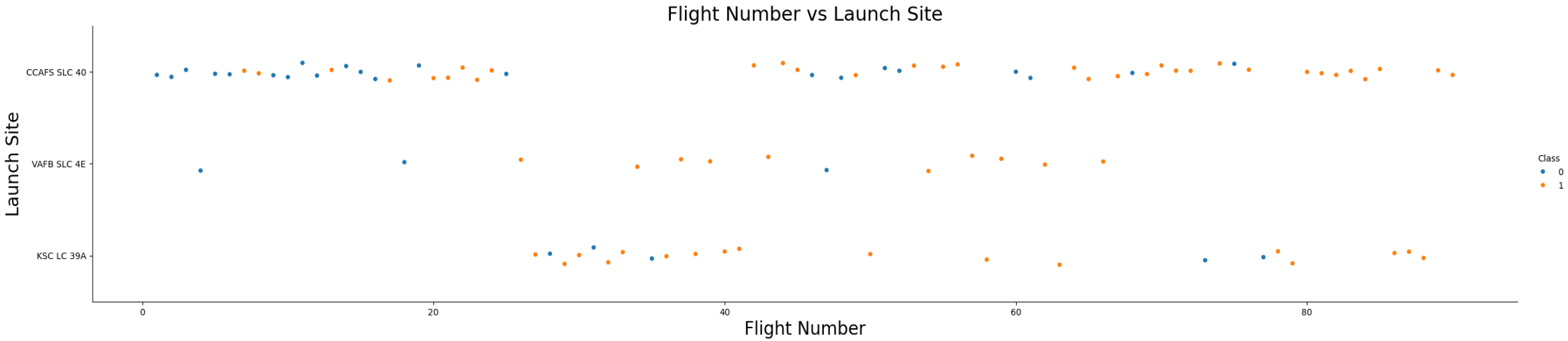
Interactive Analytics Demo in Screenshots

- Demonstrated interactive visualizations using seaborn and matplotlib.
- Included scatter plots, bar charts, and line charts to explore data relationships.

Predictive Analysis Results

- Engineered features for predictive modeling.
- Created dummy variables for categorical data.
- Prepared data for future predictive analysis tasks.


Section 2
Insights drawn from EDA



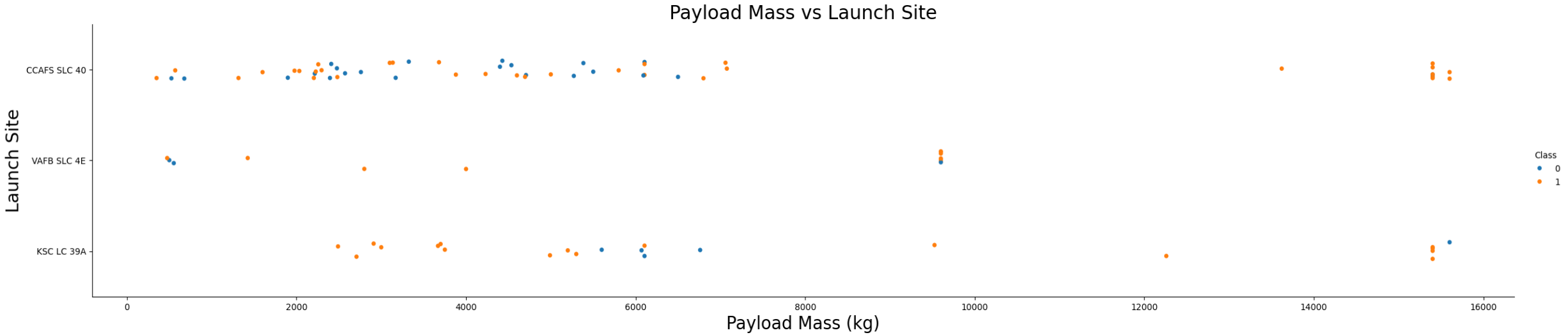
Flight Number vs. Launch Site

The scatter plot of Flight Number vs. Launch Site is used to visualize the relationship between the sequential flight numbers and the different launch sites.

- **X-axis (Flight Number):** This axis represents the flight number, which is a sequential identifier for each flight. It helps in understanding the order and frequency of the flights.
- **Y-axis (Launch Site):** This axis represents the launch sites. Each unique launch site is plotted as a distinct category on the y-axis.
- **Hue (Class):** The hue parameter is used to differentiate the success (Class=1) and failure (Class=0) of the launches. Different colors are used to represent successful and unsuccessful launches.

Interpretation

- **Clusters:** Points that are vertically aligned indicate multiple flights from the same launch site. This helps in identifying which launch sites are used more frequently.
- **Trends:** If there is a pattern in the distribution of points, it might indicate a preference or trend in using certain launch sites over time.
- **Success Rate:** By observing the colors (hue) of the points, you can infer the success rate of launches from different sites. For example, if a particular launch site has more points of the color representing success, it indicates a higher success rate for that site.

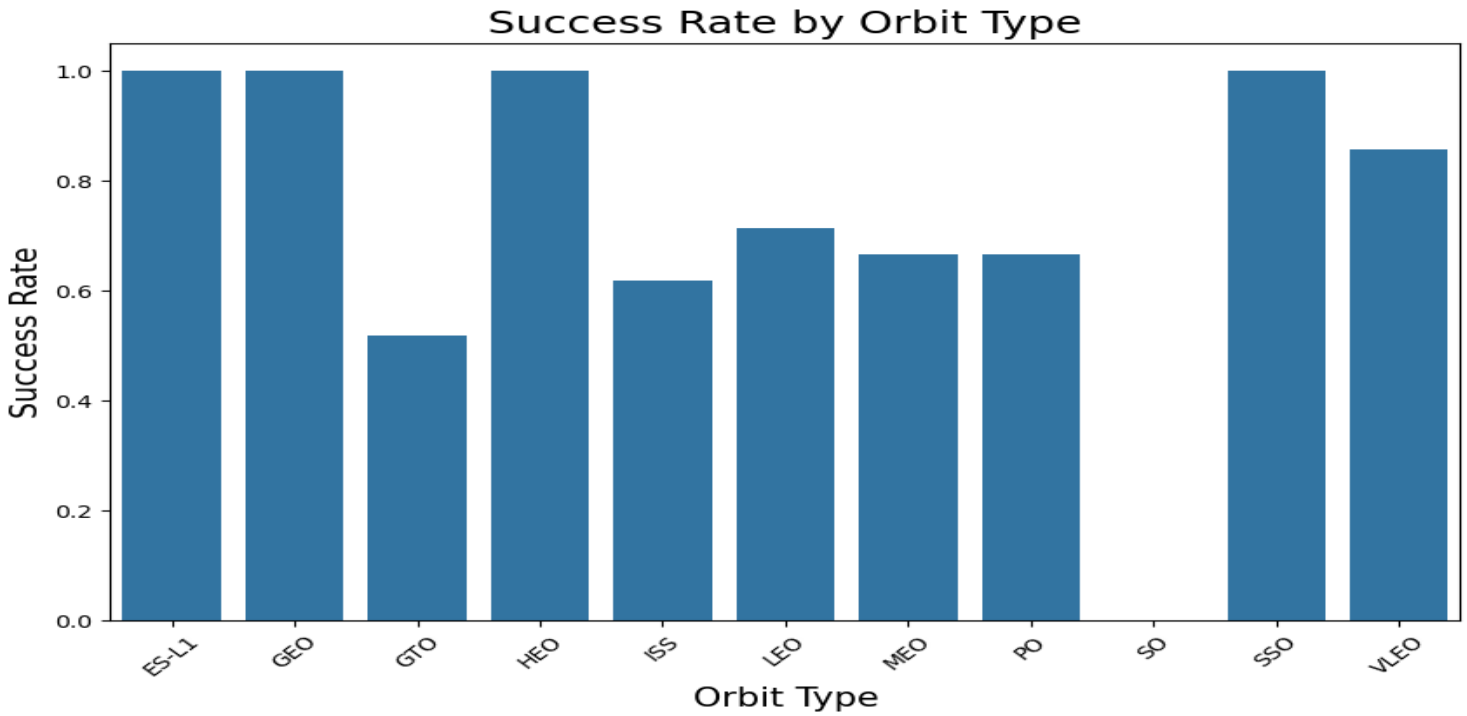


Payload vs. Launch Site

The scatter plot above shows the relationship between the payload mass and the launch site. Each point represents a launch, with the x-axis showing the payload mass in kilograms and the y-axis showing the launch site.

The hue indicates the class of the launch, where different colors represent different classes (e.g., successful or unsuccessful landings).

This visualization helps to identify any patterns or trends in the data, such as whether certain launch sites are associated with higher payloads or different success rates.

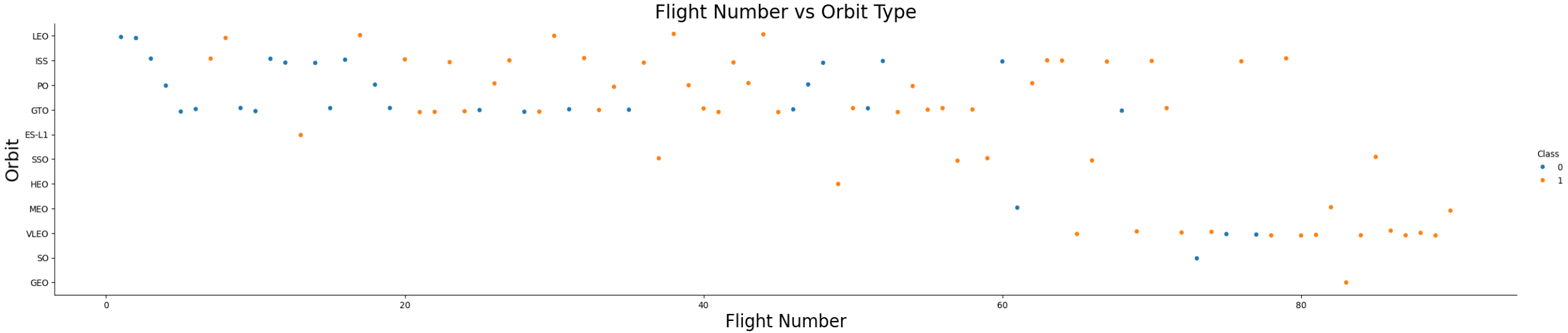


Success rate by Orbit type

The bar chart above shows the success rate of launches for each orbit type.

The x-axis represents the different orbit types, while the y-axis shows the success rate, which is the proportion of successful launches for each orbit type.

This chart helps to identify which orbit types have higher success rates and can provide valuable insights for future missions.

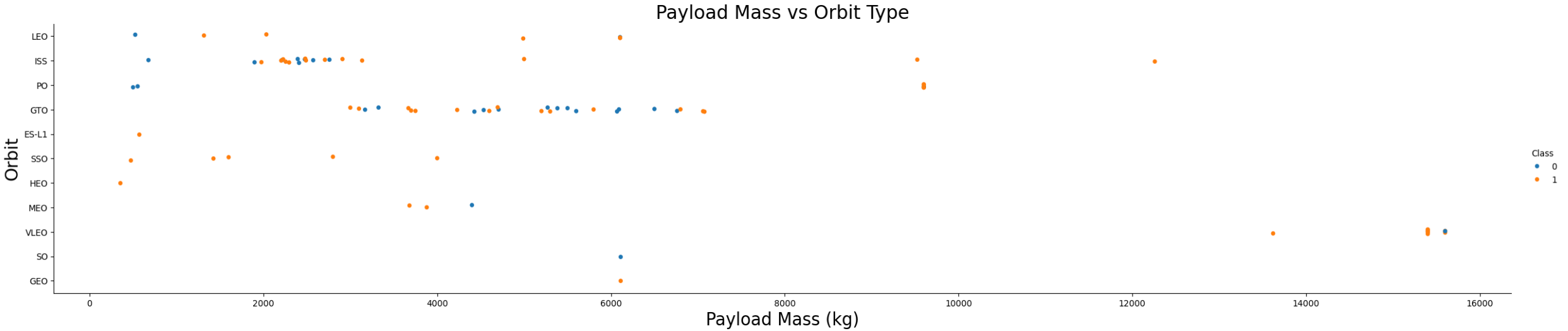


Flight Number vs. Orbit Type

The scatter plot above shows the relationship between the flight number and the orbit type. Each point represents a flight, with the x-axis showing the flight number and the y-axis showing the orbit type.

The hue indicates the class of the flight, where different colors represent different classes (e.g., successful or unsuccessful landings).

This visualization helps to identify any patterns or trends in the data, such as whether certain orbit types are associated with specific flight numbers or different success rates.

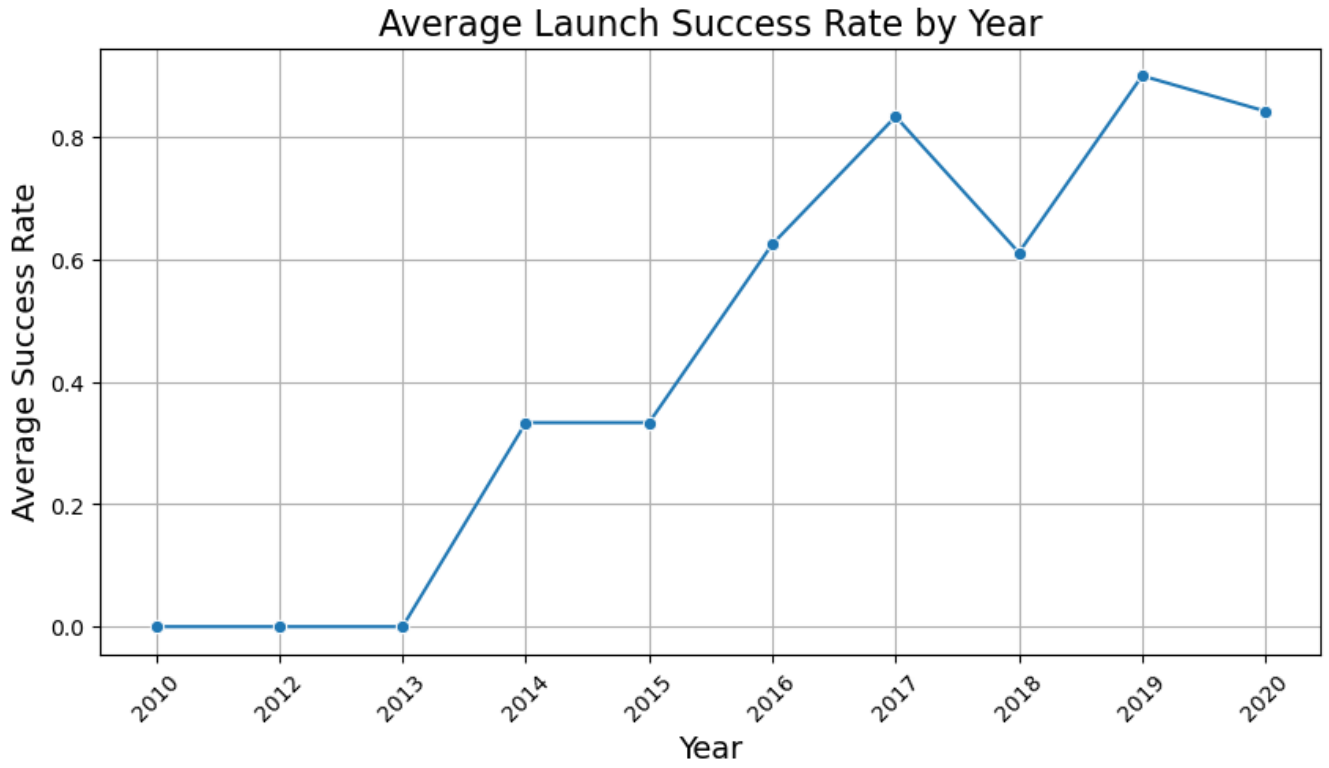


Payload vs. Orbit Type

The scatter plot above shows the relationship between the payload mass and the orbit type. Each point represents a launch, with the x-axis showing the payload mass in kilograms and the y-axis showing the orbit type.

The hue indicates the class of the launch, where different colors represent different classes (e.g., successful or unsuccessful landings).

This visualization helps to identify any patterns or trends in the data, such as whether certain orbit types are associated with higher payloads or different success rates.



Launch Success Yearly Trend

The line chart above shows the yearly average success rate of launches. The x-axis represents the year, while the y-axis shows the average success rate, which is the proportion of successful launches for each year.

Each point on the line represents the average success rate for that year.

This chart helps to identify trends over time, such as whether the success rate is improving, declining, or remaining stable.

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

Python

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

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

All Launch Site Names

This task involves querying the dataset to retrieve the distinct names of the launch sites used in SpaceX missions. The goal is to identify all the unique locations from which SpaceX has launched rockets.

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

Python

```
 * 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

Launch Site Names Begin with CCA

This task filters the dataset to find records where the launch site names start with CCA. It then displays the first 5 records that match this criterion. This helps in understanding the launches that took place at specific sites starting with CCA

```
%%sql
SELECT SUM("PAYLOAD_MASS__KG_") AS "Total_Payload_Mass"
FROM SPACEXTABLE
WHERE "Customer" = 'NASA (CRS)';
```

Python

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

Total_Payload_Mass
45596

Total Payload Mass

This task calculates the total payload mass for all missions where NASA (CRS) is listed as the customer. It sums up the payload masses of these missions to provide an aggregate value, giving insight into the total mass transported for NASA (CRS).

```sql
%%sql
SELECT AVG("PAYLOAD_MASS__KG_") AS "Average_Payload_Mass"
FROM SPACEXTABLE
WHERE "Booster_Version" = 'F9 v1.1';
```

Python

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

Average_Payload_Mass
2928.4

Average Payload Mass by F9 v1.1

This task computes the average payload mass for missions that used the booster version F9 v1.1. By calculating the mean value of the payload masses, it provides an understanding of the typical payload capacity for this specific booster version

```
%%sql
SELECT MIN("Date") AS "First_Successful_Landing_Date"
FROM SPACEXTABLE
WHERE "Landing_Outcome" = 'Success (ground pad)';
```

Python

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

First_Successful_Landing_Date
2015-12-22

First Successful Ground Landing Date

This task identifies the earliest date on which a successful landing on a ground pad was recorded. It uses the minimum function to find the first occurrence of a successful ground pad landing, highlighting a significant milestone in SpaceX's landing achievements.

```
%%sql
SELECT DISTINCT "Booster_Version"
FROM SPACEXTABLE
WHERE "Landing_Outcome" = 'Success (drone ship)'
  AND "PAYLOAD_MASS__KG_" > 4000
  AND "PAYLOAD_MASS__KG_" < 6000;
```

Python

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

Booster_Version
F9 FT B1022
F9 FT B1026
F9 FT B1021.2
F9 FT B1031.2

Successful Drone Ship Landing with Payload between 4000 and 6000

This task retrieves the names of booster versions that successfully landed on a drone ship and carried a payload mass between 4000 and 6000 kg. It filters the dataset based on these criteria to provide a list of boosters that meet both conditions.

Total Number of Successful and Failure Mission Outcomes

This task counts the number of missions that resulted in different landing outcomes, such as success or failure. It groups the data by landing outcome and provides a count for each category, offering a summary of mission success rates and failure rates.

```
%%sql
SELECT "Landing_Outcome", COUNT(*) AS "Count"
FROM SPACEXTABLE
GROUP BY "Landing_Outcome";
```

Python

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

Landing_Outcome	Count
Controlled (ocean)	5
Failure	3
Failure (drone ship)	5
Failure (parachute)	2
No attempt	21
No attempt	1
Precluded (drone ship)	1
Success	38
Success (drone ship)	14
Success (ground pad)	9
Uncontrolled (ocean)	2

Boosters Carried Maximum Payload

This task identifies the booster versions that carried the maximum payload mass. It uses a subquery to first determine the maximum payload mass and then retrieves the booster versions associated with that mass. This highlights the most capable boosters in terms of payload capacity.

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

Python

```
 * 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

2015 Launch Records

This task extracts records from the year 2015 where the landing outcome was a failure on a drone ship. It also converts the month from the date into a month name and includes the booster version and launch site in the output. This provides detailed information about specific failed missions in 2015.

```
%%sql
SELECT
    CASE SUBSTR("Date", 6, 2)
        WHEN '01' THEN 'January'
        WHEN '02' THEN 'February'
        WHEN '03' THEN 'March'
        WHEN '04' THEN 'April'
        WHEN '05' THEN 'May'
        WHEN '06' THEN 'June'
        WHEN '07' THEN 'July'
        WHEN '08' THEN 'August'
        WHEN '09' THEN 'September'
        WHEN '10' THEN 'October'
        WHEN '11' THEN 'November'
        WHEN '12' THEN 'December'
    END AS Month,
    "Landing_Outcome",
    "Booster_Version",
    "Launch_Site"
FROM SPACEXTABLE
WHERE "Landing_Outcome" = 'Failure (drone ship)'
  AND SUBSTR("Date", 1, 4) = '2015';
```

Python

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

Month	Landing_Outcome	Booster_Version	Launch_Site
January	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
April	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

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

This task counts the occurrences of different landing outcomes within the specified date range and ranks them in descending order. It provides a ranked list of landing outcomes, showing which outcomes were most and least common during this period.

```
%%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;
```

Python

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

Landing_Outcome	Count
No attempt	10
Success (drone ship)	5
Failure (drone ship)	5
Success (ground pad)	3
Controlled (ocean)	3
Uncontrolled (ocean)	2
Failure (parachute)	2
Precluded (drone ship)	1

Section 3

Launch Sites Proximities Analysis

SpaceX Launch Sites Locations on Global Map

Elements:

- The circles and markers indicate the locations of SpaceX launch sites.
- The names of the launch sites are displayed on the markers.

Findings:

- The launch sites are located in Florida and California.
- Proximity to the coast is evident, which is crucial for launch safety.

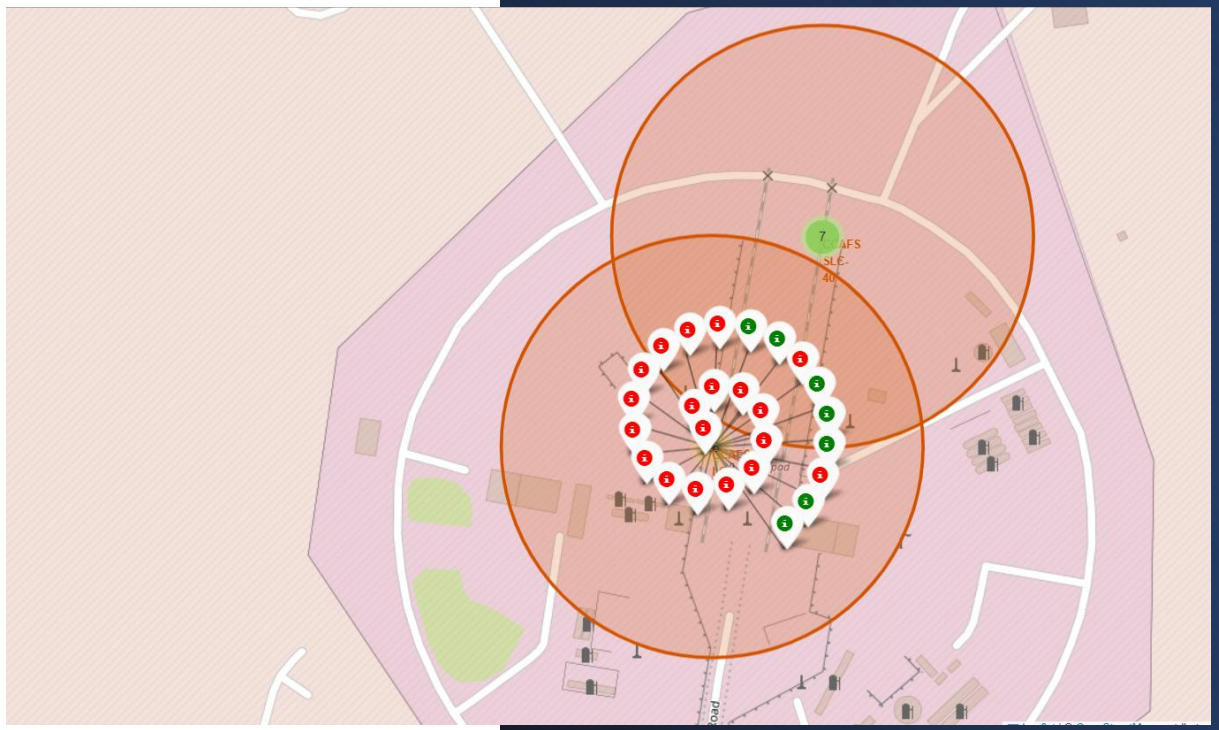
SpaceX Launch Outcomes Labeled by Color

Elements:

- Green markers indicate successful launches.
- Red markers indicate failed launches.

Findings:

- Most launches at the Florida sites have been successful.
- The visualization helps identify sites with high success rates.



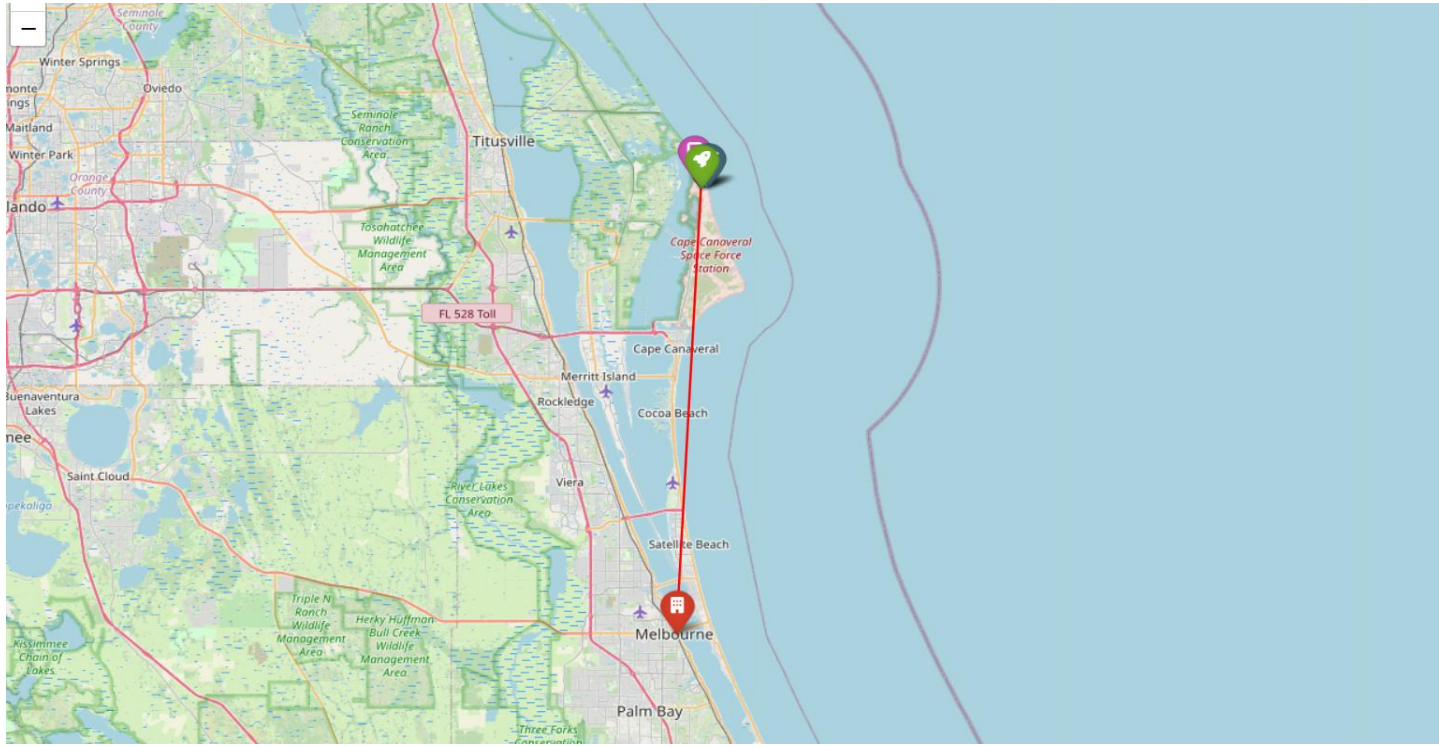
Proximity of Launch Site to Key Infrastructures

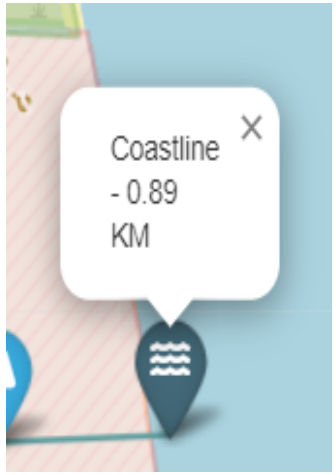
Elements:

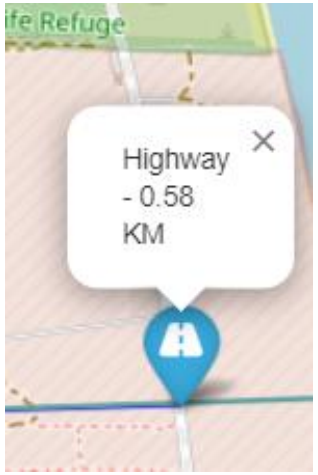
- The lines indicate the distance to the Infrastructures.
- Colored markers indicate proximity to highways, cities, and railways.

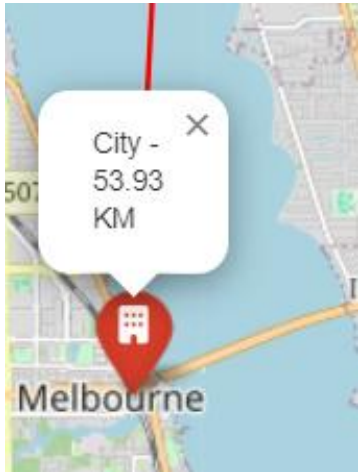
Findings:

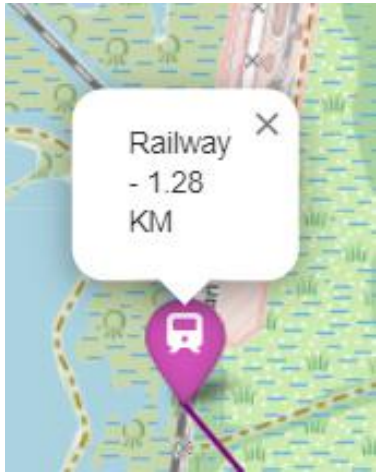
- The launch site is close to the coastline, which is crucial for safety.
- Proximity to highways and railways facilitates the transportation of equipment and materials.
- The distance to the city ensures the safety of populated areas.

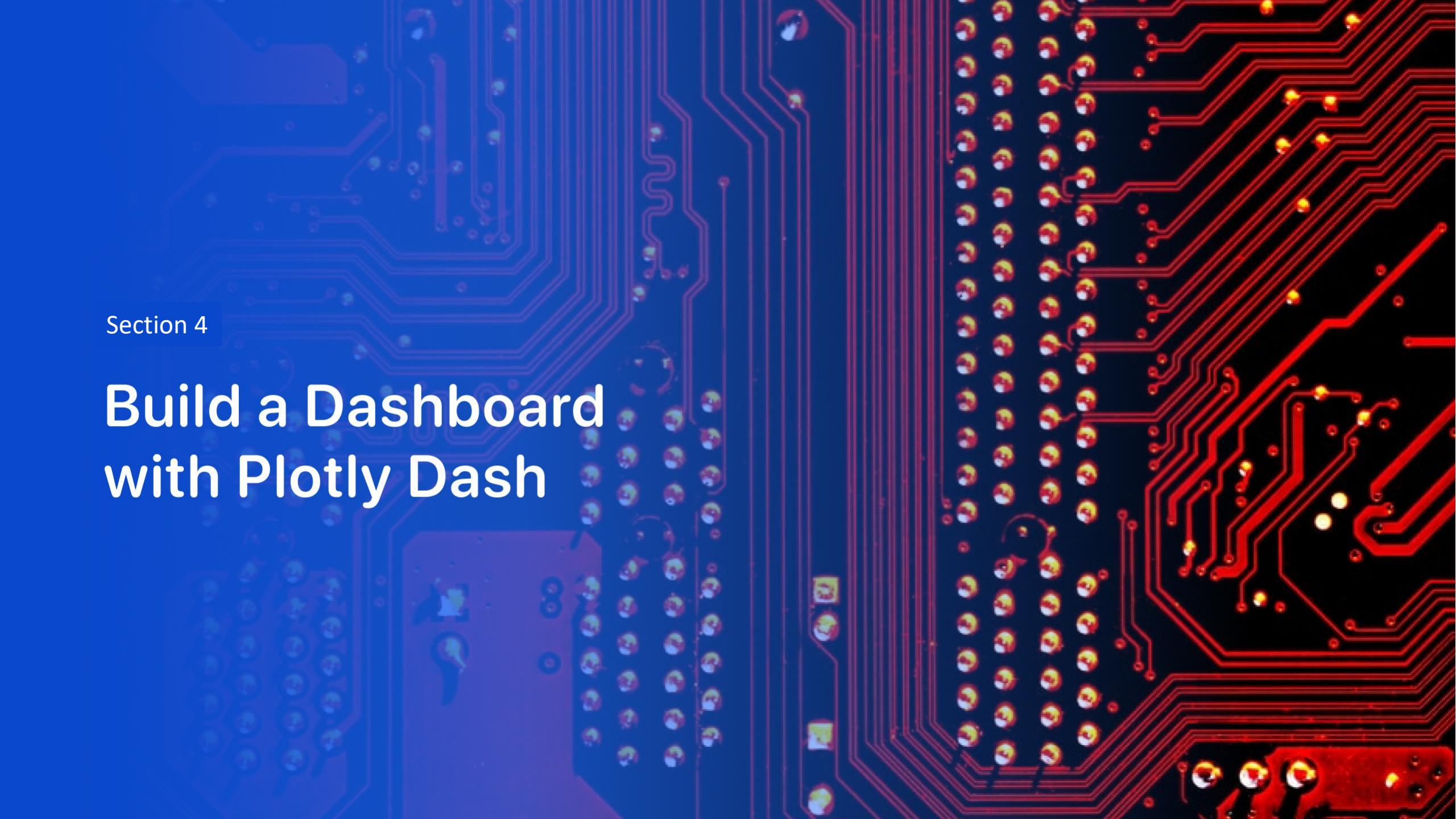




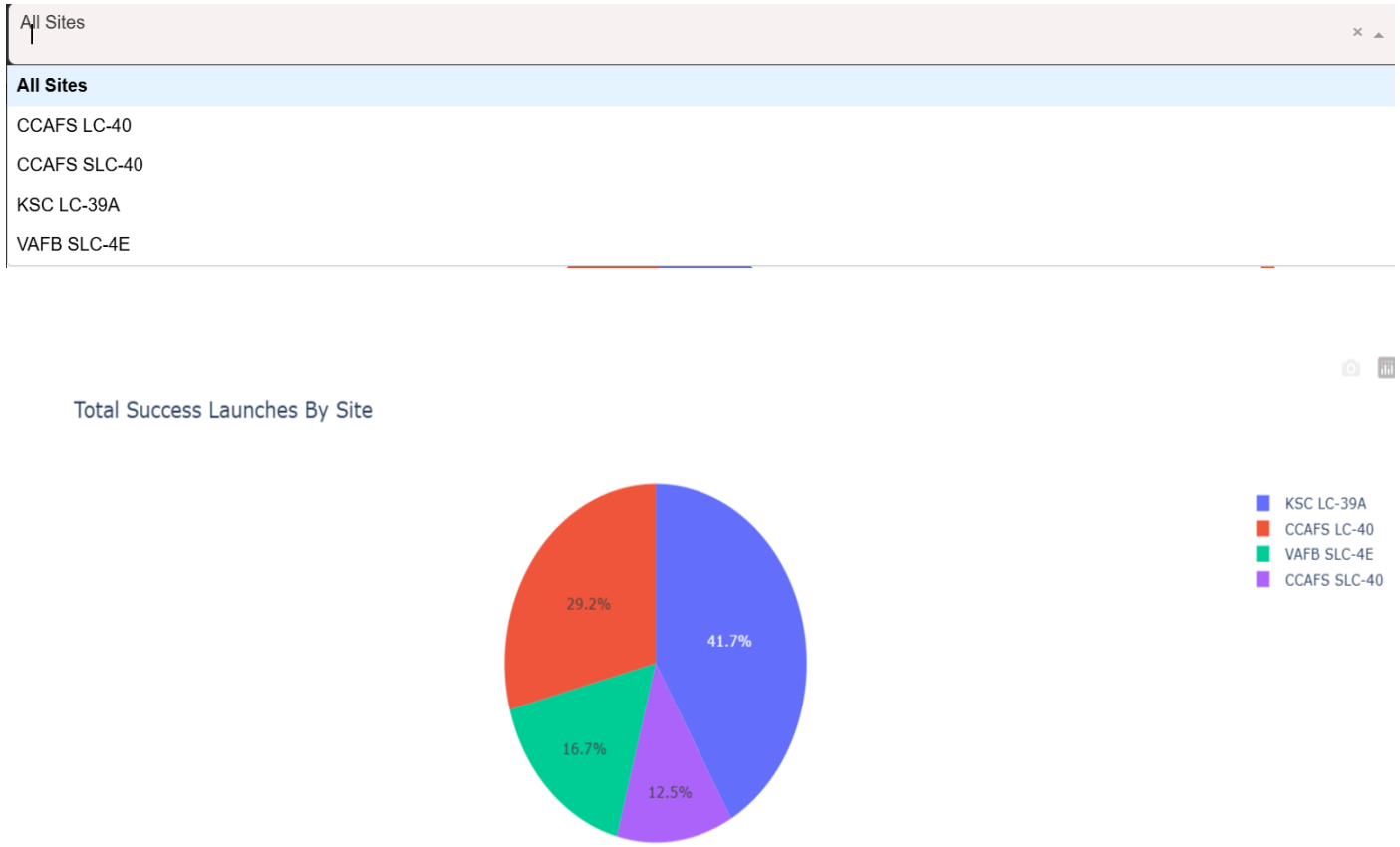







Section 4
Build a Dashboard with Plotly Dash

Launch Success Count for All Sites



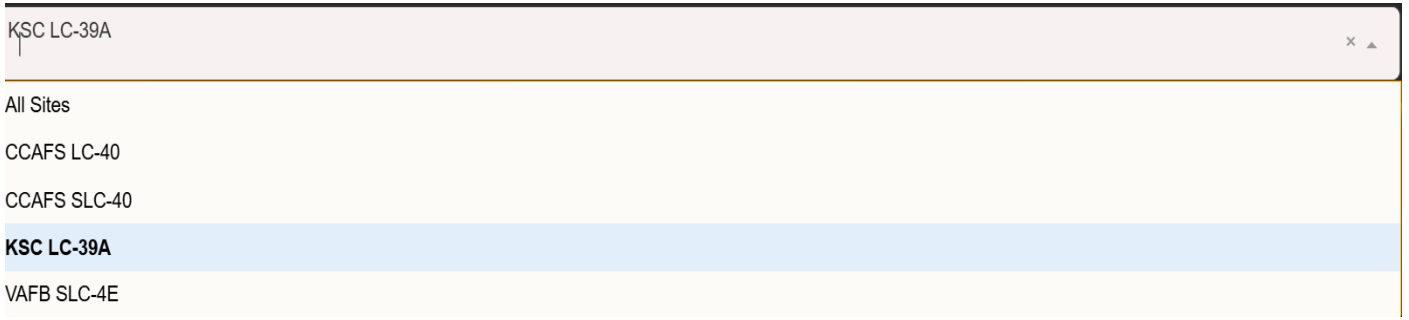
Elements:

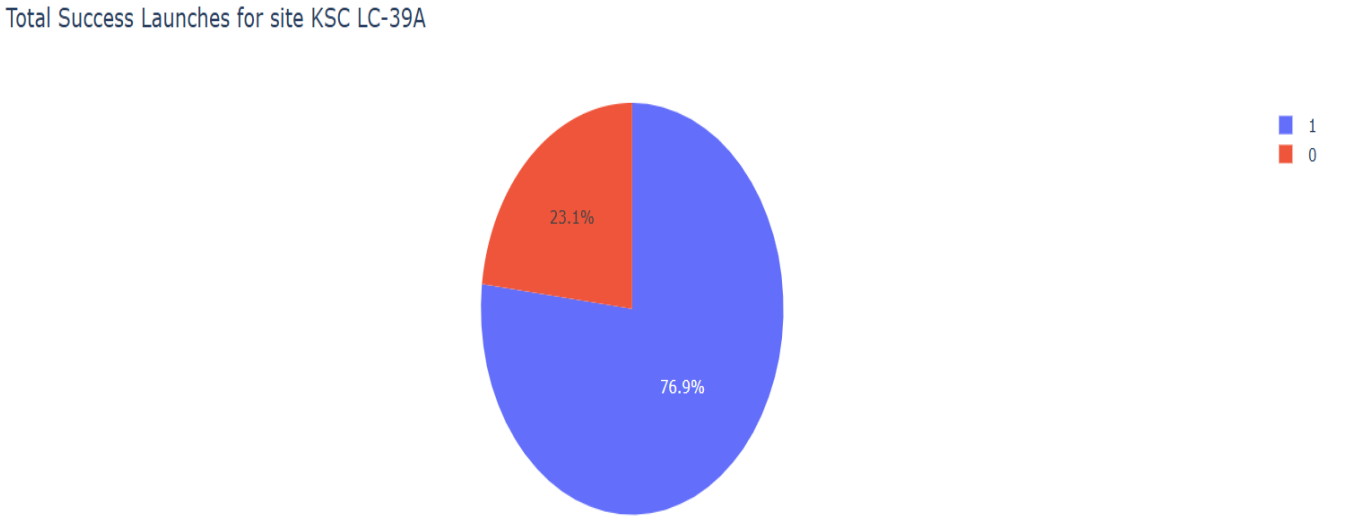
- **Pie Chart:** Visual representation of the total number of successful launches for each SpaceX launch site.
- **Legend:** Indicates the launch sites corresponding to each pie slice.

Findings:

- The pie chart shows the distribution of successful launches across different launch sites.
- The site with the largest pie slice has the highest number of successful launches.

Launch Success Ratio for the Site with Highest Success Rate





Elements:

- **Pie Chart:** Visual representation of the success ratio for the launch site with the highest success rate.
- **Legend:** Indicates the success and failure counts.

Findings:

- The pie chart shows the proportion of successful and failed launches for the specific site.
- The site with the highest success rate will have a larger proportion of successful launches compared to failures.

Payload vs. Launch Outcome for All Sites

Elements:

- **Scatter Plot:** Shows the correlation between payload mass and launch outcome for all sites
- **Color Coding:** Different colors represent different booster versions.
- **Axes**: X-axis represents payload mass (kg), Y-axis represents launch outcome (success/failure).

Findings:

- The scatter plots illustrate how payload mass affects the success rate of launches.
- Certain payload ranges may show higher success rates.
- Specific booster versions might be more successful with certain payload ranges.
- By analyzing these plots, one can identify the optimal payload ranges and booster versions for successful launches.

Section 5

Predictive Analysis (Classification)

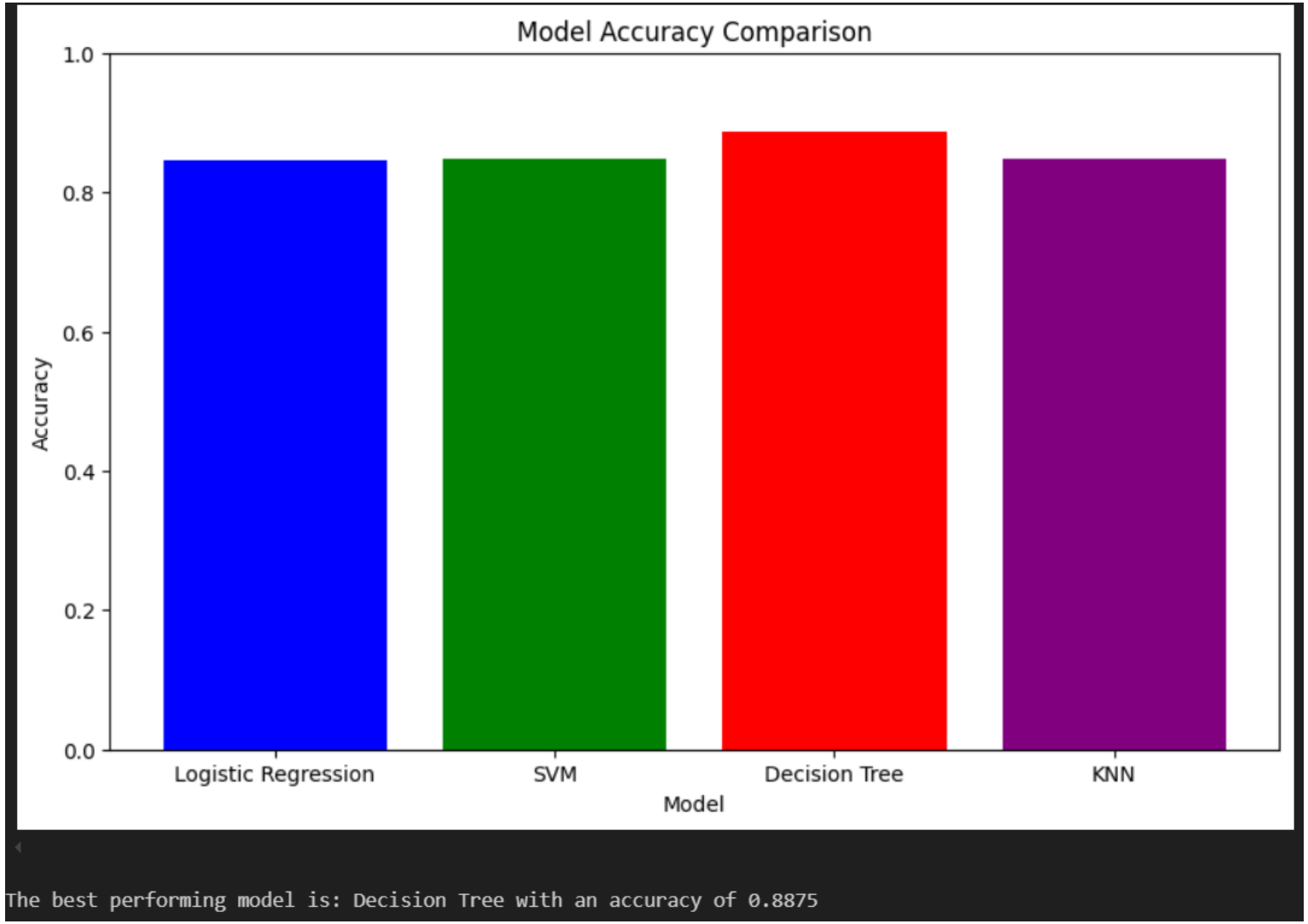
```python
import matplotlib.pyplot as plt

# Collect the accuracies of the models
models_accuracies = {
    'Logistic Regression': logreg_cv.best_score_,
    'SVM': svm_cv.best_score_,
    'Decision Tree': tree_cv.best_score_,
    'KNN': knn_cv.best_score_
}

# Create a bar chart
plt.figure(figsize=(10, 6))
plt.bar(models_accuracies.keys(), models_accuracies.values(), color=['blue', 'green', 'red', 'purple'])
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.title('Model Accuracy Comparison')
plt.ylim(0, 1)
plt.show()

# Identify the model with the highest accuracy
best_model = max(models_accuracies, key=models_accuracies.get)
print(f"The best performing model is: {best_model} with an accuracy of {models_accuracies[best_model]:.4f}")
```

✓ 0.0s Python

Classification Accuracy

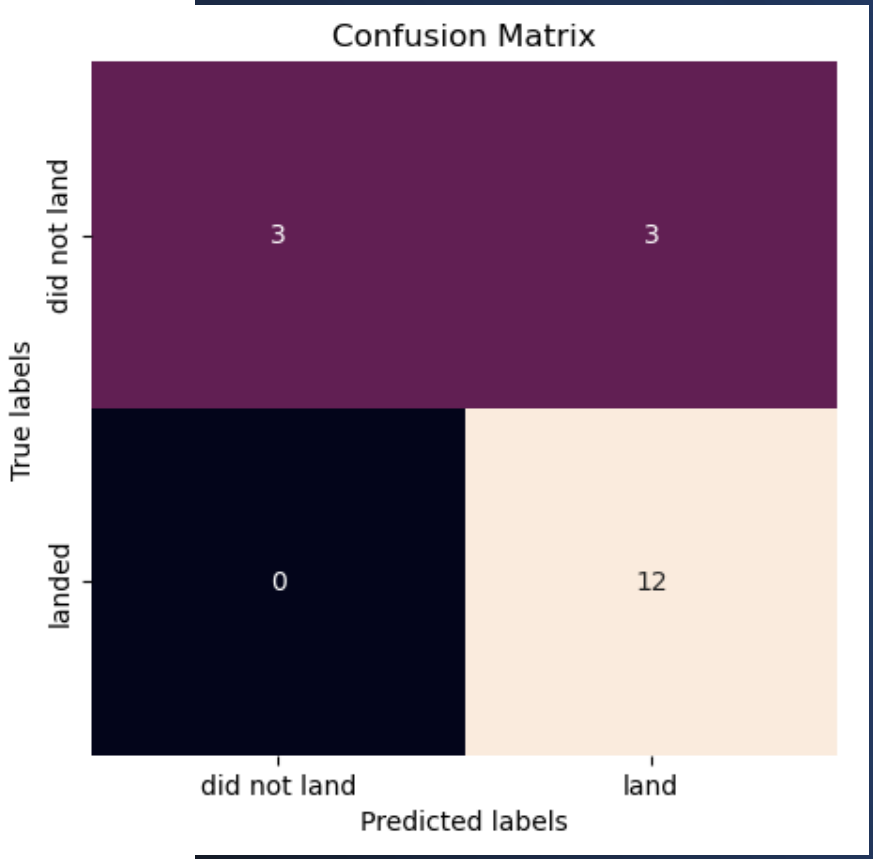
The bar chart visualization shows the accuracy of each classification model built. From the chart, we can see that the Decision Tree model has the highest accuracy. Specifically, the Decision Tree model achieved an accuracy of 0.8875, making it the best performing model among those evaluated. This means that the Decision Tree model correctly predicted the outcomes 88.75% of the time on the test data, outperforming the Logistic Regression, SVM, and KNN models.

Classification Accuracy



Confusion Matrix

The confusion matrix generated for the decision tree classifier provides a clear representation of the model's ability to correctly distinguish between the various categories it was trained to identify. It demonstrates that the classifier performs reasonably well in separating the classes. However, one of the most notable challenges observed is the occurrence of false positives. This refers to instances where the classifier mistakenly labels an unsuccessful landing as a successful one, which could lead to incorrect conclusions or actions based on the predictions. Addressing this issue is crucial to improving the overall accuracy and reliability of the classifier's performance.



Conclusions

Based on the analysis, we can draw the following conclusions:

- Launch sites with a higher volume of flights tend to have a correspondingly higher success rate, indicating a positive correlation between flight frequency and success.
- There has been a steady increase in the launch success rate starting from 2013, culminating in notable improvements by 2020.
- Among the various orbital destinations, ES-L1, GEO, HEO, SSO, and VLEO have demonstrated the highest success rates, highlighting these orbits as particularly reliable for missions.
- The Kennedy Space Center's Launch Complex 39A (KSC LC-39A) has outperformed other launch sites, recording the highest number of successful launches.
- When comparing machine learning models for this analysis, the decision tree classifier stands out as the most effective algorithm, delivering the best results for this task.

Thank you!