IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Lawrence Chinn
18 Nov 2024

Outline

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

Executive Summary

Summary of methodologies

1. Data collection methodology
    - Request from SpaceX API and web scraping from Wikipedia of Falcon Launches (https://en.wikipedia.org/wiki/List_of_Falcon_9_and_Falcon_Heavy_launches)
2. Perform data wrangling
    - The two data sets were cleaned, transformed and integrated into a single set for quality control before EDA
3. Perform exploratory data analysis (EDA) using visualization and SQL
4. Perform interactive visual analytics using Folium and Plotly Dash
5. Perform predictive analysis using classification models
    - Build and assess multiple training models

Summary of all results

1. SpaceX developed a degree of expertise over time (2010-2020) allowed them to achieve a relative high level of reliability when launching their Falcon F9 rockets
2. However, the Falcon F9 is limited by its payload restriction and some orbits (e.g. GTO) remain a challenge.

Introduction

Background

In the span of 10 years, SpaceX's accomplishments include

- Sending a spacecraft to the International Space Station
- Establishing Starlink, a satellite-based network providing Internet access
- Manned missions to Space.

For SpaceX, the rocket launches are relatively inexpensive. SpaceX advertises Falcon 9 rocket launches on its website for $62 million, whereas other providers advertise cost upwards of $165 million. Much of the savings is because, unlike other providers, SpaceX can reuse the first stage. However, this is not always the case. Sometimes, it will crash and is sacrificed due to the mission parameters, e.g. payload, orbit, and customer.

Problem

Accurately determine the successful recovery of the Falcon 9 booster to gain some insight for the actual cost of a rocket launch.

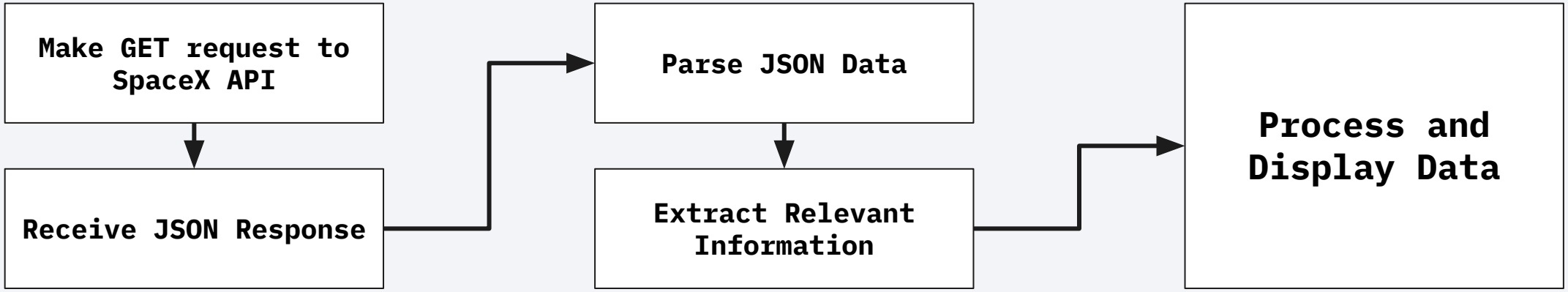
Section 1

Methodology

Methodology

- Data collection methodology
  - Request from SpaceX API and web scraping from Wikipedia of Falcon Launches (https://en.wikipedia.org/wiki/List_of_Falcon_9_and_Falcon_Heavy_launches)
- Perform data wrangling
  - The two data sets were cleaned, transformed and integrated into a single set for quality control before EDA
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - Build and assess multiple training models

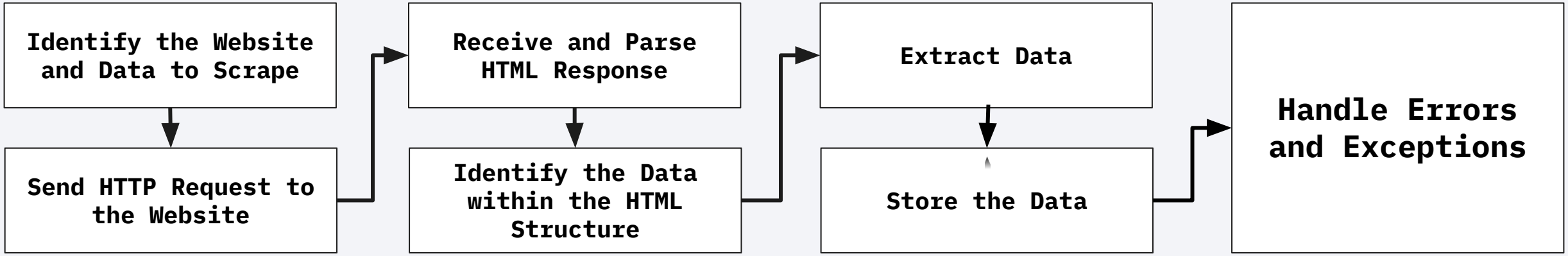
Data Collection - SpaceX API



GitHub URL:

https://github.com/blarghwtfbbq/Capstone-_SpaceX/blob/c454b1b52469f4981f2c4ec047159cff5944be75/jupyter-labs-spacex-data-collection-api%20Mod%201A.ipynb

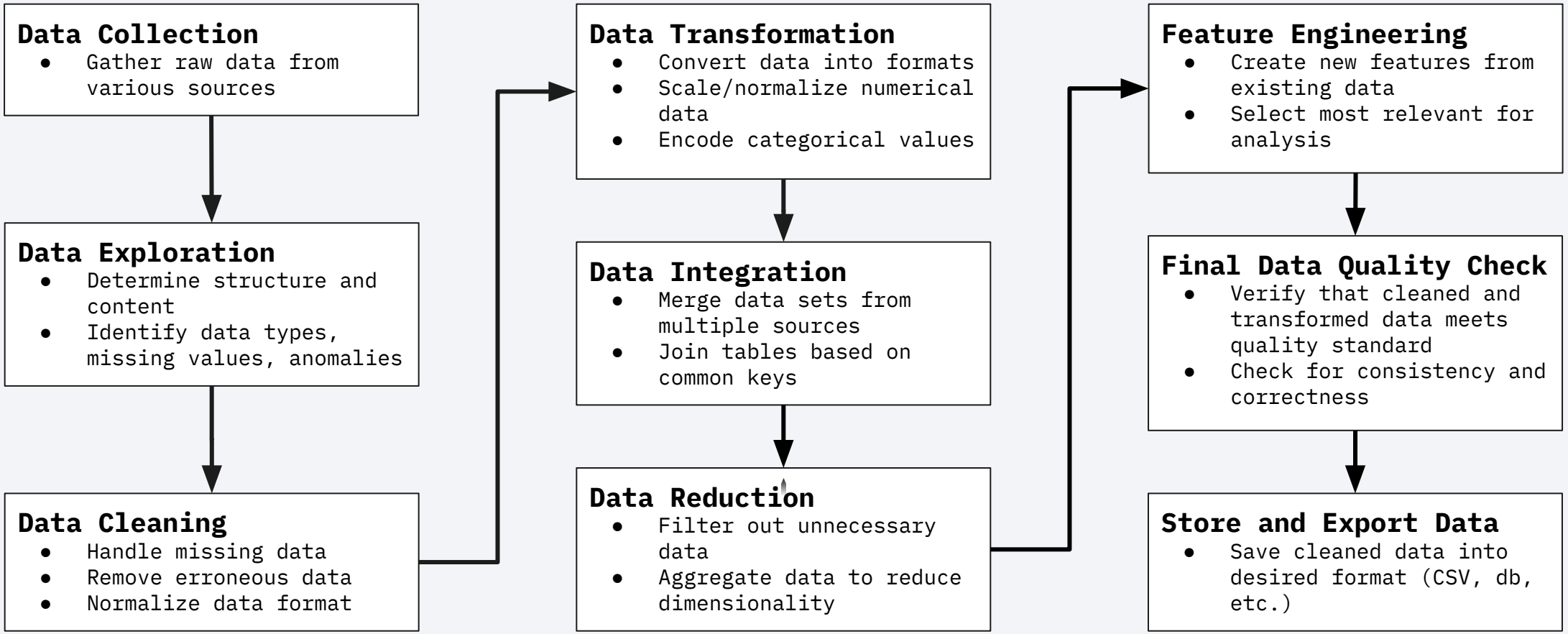
Data Collection - Scraping



GitHub URL:

https://github.com/blarghwtfbbq/Capstone-_SpaceX/blob/c454b1b52469f4981f2c4ec047159cff5944be75/jupyter-labs-webscraping%20Mod%201B.ipynb

Data Wrangling



GitHub URL:

https://github.com/blarghwtfbbq/Capstone-_SpaceX/blob/c454b1b52469f4981f2c4ec047159cff5944be75/labs-jupyter-spacex-Data%20wrangling%20Mod%201C.ipynb

EDA with Data Visualization

Visualizations

- Scatter plot: Flight number vs Launch Site
- Scatter plot: Payload Mass vs Launch Site
- Bar chart: Success rate by Orbit Type
- Scatter plot: Flight Number vs Orbit Type
- Scatter plot: Payload Type vs Orbit Type
- Line plot: Yearly Launch Success Rate Trend

GitHub URL:

https://github.com/blarghwtfbbq/Capstone-_SpaceX/blob/c454b1b52469f4981f2c4ec047159cff5944be75/edadataviz%20Mod%202B.ipynb

EDA with SQL

SQL Queries

- Names of unique launch sites
- Five records where launch site begins with “CCA”
- Total payload mass carried by boosters launched by NASA (CRS)
- Average payload mass carried by booster version F9 v1.1
- Date when first successful landing outcome in ground pad was achieved
- List names of boosters which have success in drone ship and have payload greater than 4000 kg but less than 6000 kg
- List total number of successful and failure mission outcomes
- List names of the booster version which carried the maximum payload mass (subquery)
- List records which display month names, failure outcomes in drone ship, booster versions, launch site for the months in the year 2015
- Rank the count landing outcomes between 2010-06-04 and 2017-03-20 in descending order

GitHub URL:

https://github.com/blarghwtfbbq/Capstone-_SpaceX/blob/c454b1b52469f4981f2c4ec047159cff5944be75/jupyter-labs-eda-sql-coursera_sqllite%20Mod%202A.ipynb

Build an Interactive Map with Folium

Map Features

1. Markers to identify locations of SpaceX launch sites across the U.S.
2. Sub-markers for the number of launches tat are color-coded to indicate success or failure
3. Distance calculations to indicate proximity to support infrastructure and populated areas

GitHub URL:

https://github.com/blarghwtfbbq/Capstone-_SpaceX/blob/c454b1b52469f4981f2c4ec047159cff5944be75/lab_jupyter_launch_site_location%20Mod%203A.ipynb

Build a Dashboard with Plotly Dash

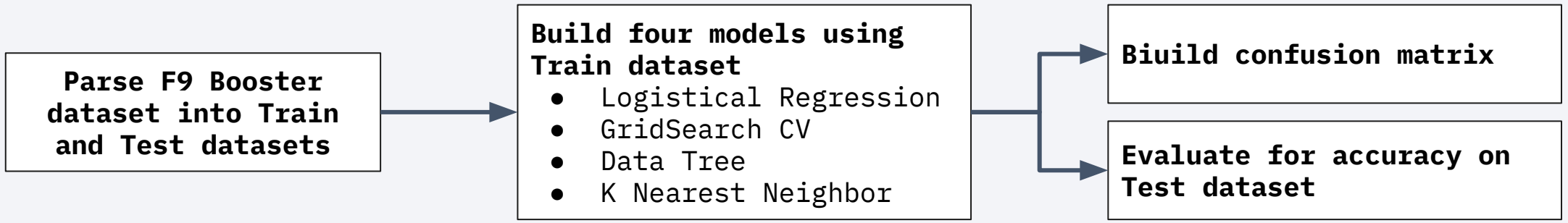
Dashboard Features

1. Range slider: Allows selection of payload mass that range from 0 kg up to 10,000 kg by increments of 2500 kg
2. Pie Chart callback: Updates based upon selected launch site
3. Scatter Chart callback: Updates scatter chart based upon selected launch site and payload range
    a. Correlate launch success and payload mass to color-labeled dots representing booster version category

GitHub URL:

https://github.com/blarghwtfbbq/Capstone-_SpaceX/blob/c454b1b52469f4981f2c4ec047159cff5944be75/spacex_dash_app.py

Predictive Analysis (Classification)



GitHub URL:

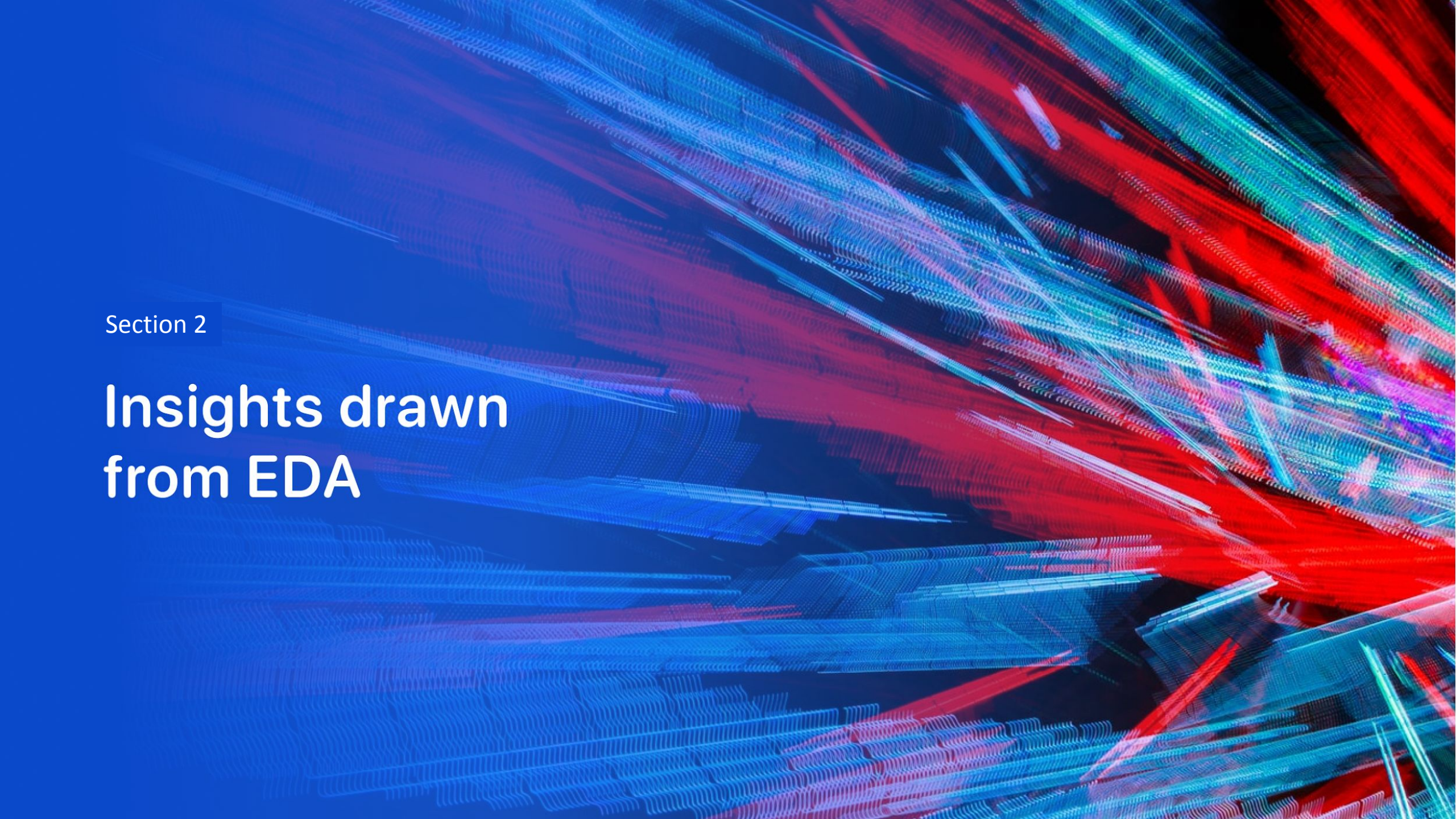
https://github.com/blarghwtfbbq/Capstone-_SpaceX/blob/c56ee2e1c810bb89c5e34fcc983d683616332543/SpaceX_Machine_Learning_Prediction_Part_5_v1.ipynb

Results

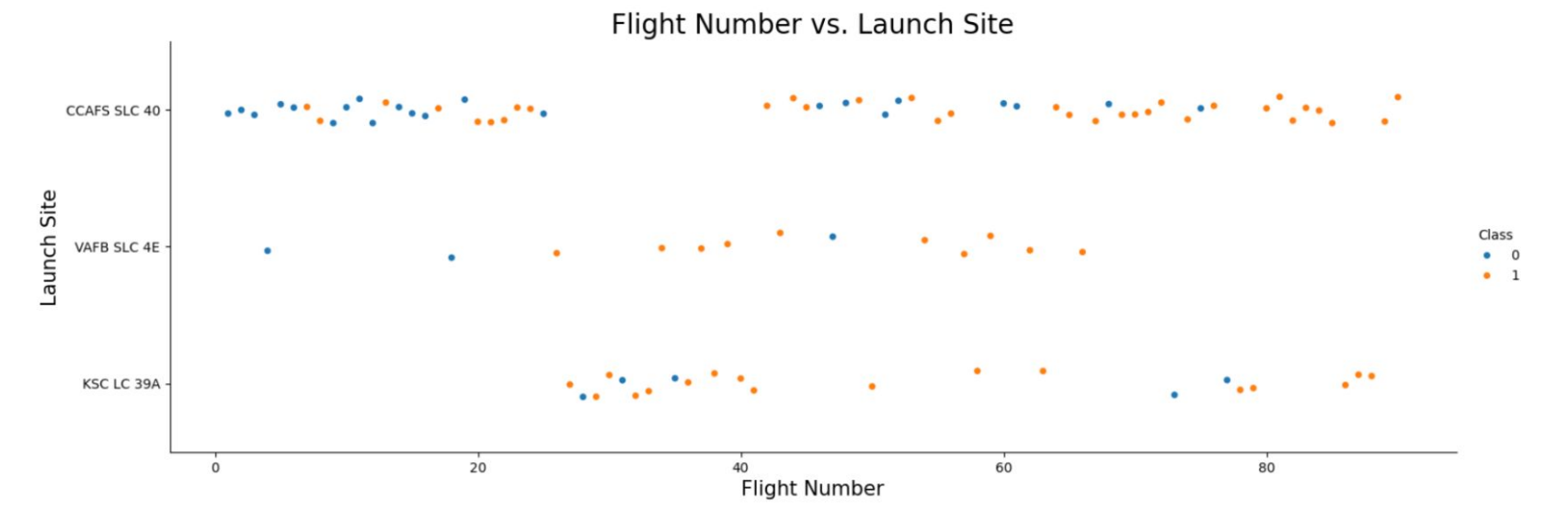
- Exploratory results suggest that SpaceX developed and maintain a high degree of expertise in an iterative fashion allowing them to maintain a relatively stable level of successful launches between 2010 and 2020.
- Interactive analytics demo in screenshots
- Predictive analysis results

Section 2

Insights drawn from EDA

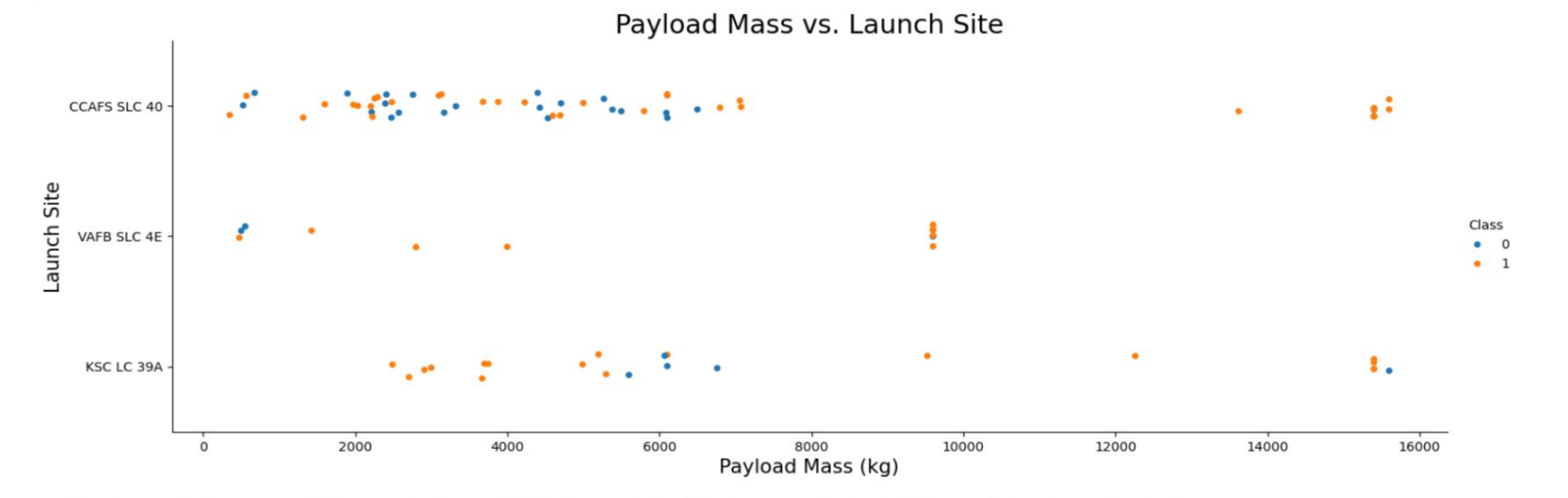


Flight Number vs. Launch Site



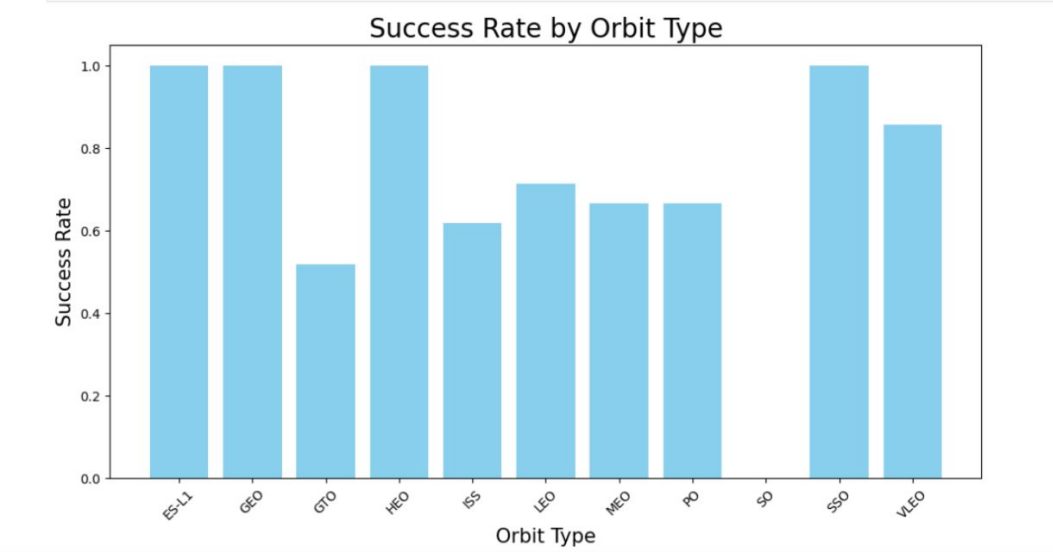
1. CCAFS SLC-40 may appear more frequently indicating it may be designated as a primary location for launches
2. Shift from failure to success with increasing flight number may be attributed improvement in technology, experience, or strategies
3. Greater success observed at CCAFS SLC-40 and KSC LC-39A may be due to factors such as site conditions, technology compatibility, or mission-specific complexities

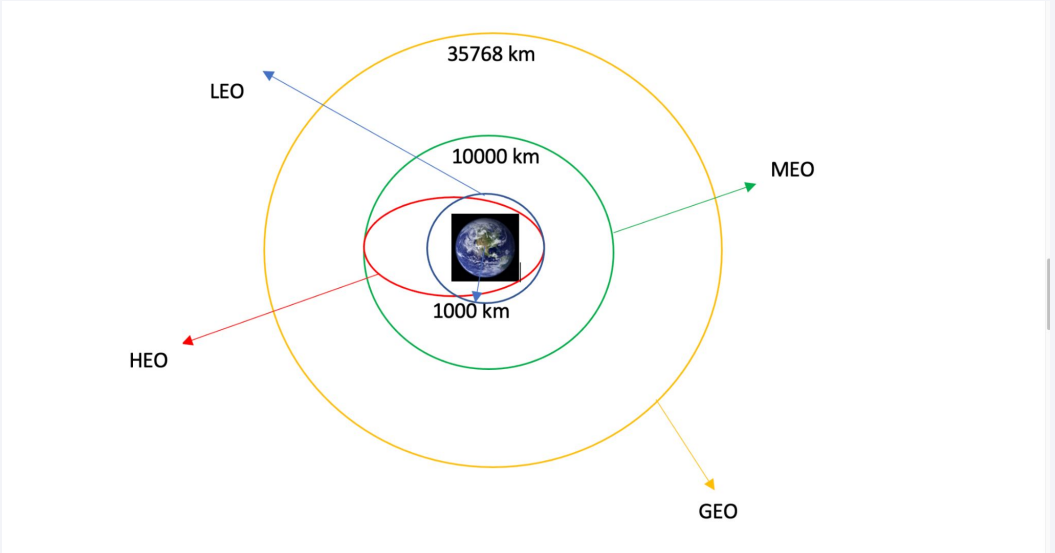
Payload vs. Launch Site



1. CCAFS SLC-40 may appear more frequently indicating it may be designated as a primary location for launches
2. Success pattern
   - No rockets launched from VAFB-SLC launch site with a heavy payload (greater than 10,000 kg)
   - Certain site may exhibit consistent success for specific payload ranges due to optimal conditions or better technology
3. Key infrastructure or procedural strengths at those locations may play a role in successful launches with heavy payloads at CCAFS SLC-40 and KSC LC-39A

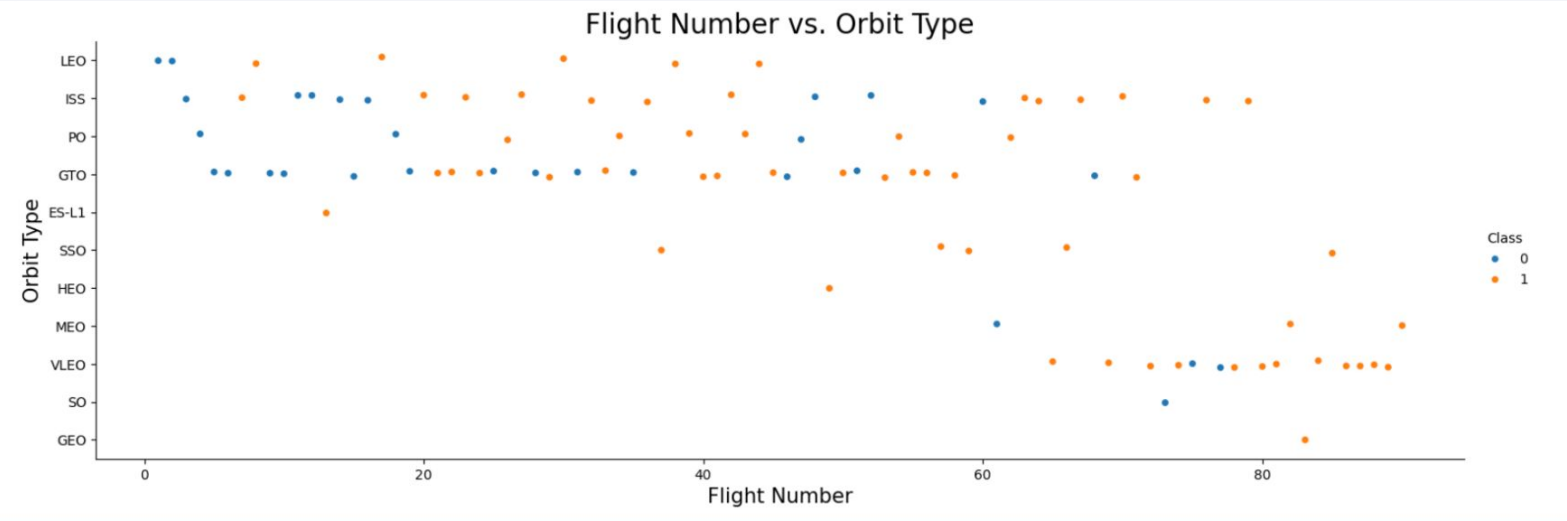
Success Rate vs. Orbit Type





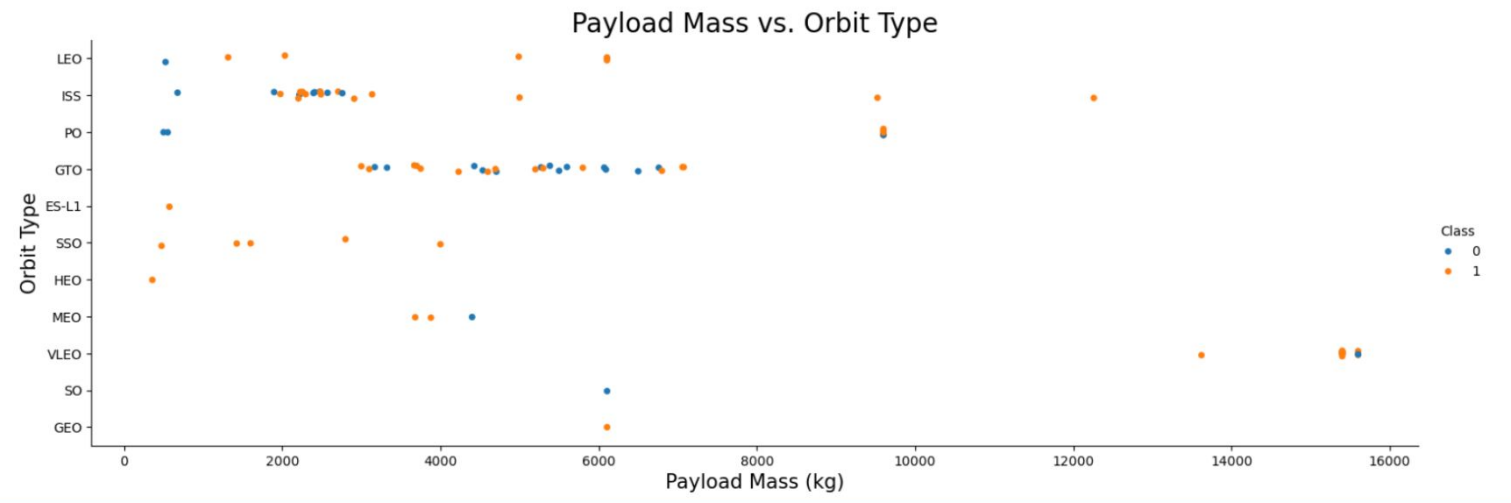
1. Higher success rates observed for launch attempts to the GEO, HEO, and VLEO
2. Lower success rates observed for launch attempts to GTO, LEO, and MEO
3. Generally, there seems to be a higher success rate for higher orbit attempts greater than 10,000 km
    - Further investigation is required to identify the challenges to successfully achieve LEO to MEO range as well as the failure to target GTO

Flight Number vs. Orbit Type



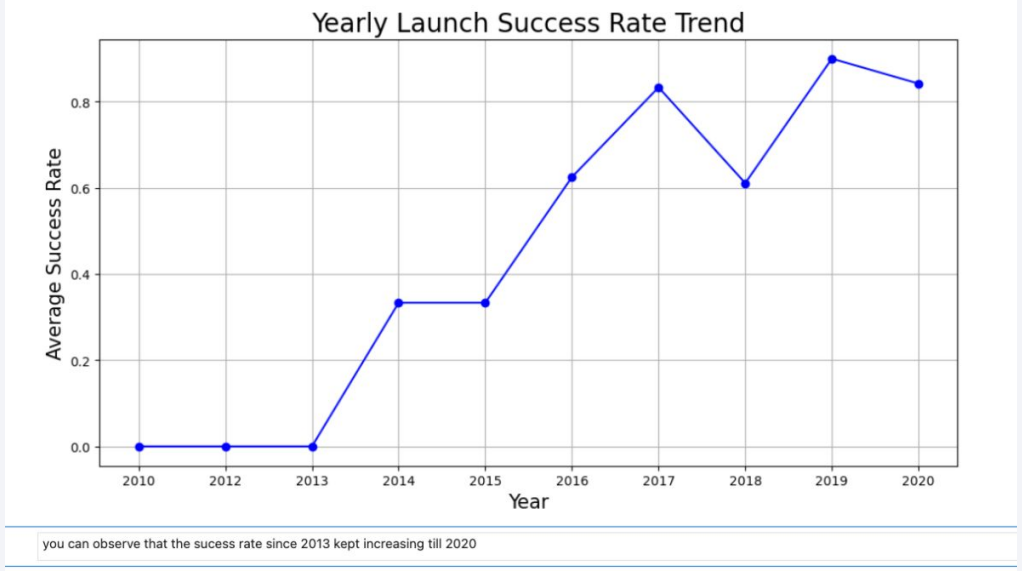
1. Success rate for LEO may improve with higher flight numbers, e.g. flight experience
2. Other factors, not related to flight numbers, have an impact upon the success for GTO

Payload vs. Orbit Type



1. Success was observed for LEO, ISS, and PO for heavy payloads
2. For GTO, the outcome is a mixture of success and failure across a range of payload sizes confounding any conclusion

Launch Success Yearly Trend



1. Positive trend in success rate with some yearly fluctuations possibly due to challenges, e.g. technical issues, adverse weather conditions
2. Pronounced improvement reflects its strive to continually learn and enhance reliability leading to a stabilized high level of success

SpaceX made significant improvement with its launch success rate over time. This can be attributed to a continuous effort to learn, innovate, and improve their launch systems

All Launch Site Names

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

```
# Write the SQL query to get unique launch sites
query = 'SELECT DISTINCT "Launch_Site" FROM SPACEXTBL'

# Execute the query and load the results into a DataFrame
unique_launch_sites = pd.read_sql_query(query, conn)

# Display the results
print(unique_launch_sites)
```

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

```
         Date Time (UTC) Booster_Version  Launch_Site  \
0  2010-06-04   18:45:00  F9 v1.0  B0003  CCAFS LC-40
1  2010-12-08   15:43:00  F9 v1.0  B0004  CCAFS LC-40
2  2012-05-22    7:44:00  F9 v1.0  B0005  CCAFS LC-40
3  2012-10-08    0:35:00  F9 v1.0  B0006  CCAFS LC-40
4  2013-03-01   15:10:00  F9 v1.0  B0007  CCAFS LC-40

                                             Payload  PAYLOAD_MASS__KG_  \
0              Dragon Spacecraft Qualification Unit                  0
1  Dragon demo flight C1, two CubeSats, barrel of...                  0
2                            Dragon demo flight C2                525
3                                      SpaceX CRS-1                500
4                                      SpaceX CRS-2                677

       Orbit         Customer Mission_Outcome      Landing_Outcome
0        LEO           SpaceX         Success  Failure (parachute)
1  LEO (ISS)  NASA (COTS) NRO         Success  Failure (parachute)
2  LEO (ISS)      NASA (COTS)         Success           No attempt
3  LEO (ISS)       NASA (CRS)         Success           No attempt
4  LEO (ISS)       NASA (CRS)         Success           No attempt
Index(['Date', 'Time (UTC)', 'Booster_Version', 'Launch_Site', 'Payload',
       'PAYLOAD_MASS__KG_', 'Orbit', 'Customer', 'Mission_Outcome',
       'Landing_Outcome'],
      dtype='object')
```

Launch Site Names Begin with 'CCA'

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

```
# SQL query to get 5 records where launch sites begin with 'CCA'
query = '''
SELECT *
FROM SPACEXTBL
WHERE "Launch_Site" LIKE 'CCA%'
LIMIT 5;
'''

# Execute the query and load the results into a DataFrame
cca_launch_sites = pd.read_sql_query(query, conn)

# Display the results
print(cca_launch_sites)
```

```
         Date Time (UTC) Booster_Version  Launch_Site  \
0  2010-06-04   18:45:00  F9 v1.0  B0003  CCAFS LC-40
1  2010-12-08   15:43:00  F9 v1.0  B0004  CCAFS LC-40
2  2012-05-22    7:44:00  F9 v1.0  B0005  CCAFS LC-40
3  2012-10-08    0:35:00  F9 v1.0  B0006  CCAFS LC-40
4  2013-03-01   15:10:00  F9 v1.0  B0007  CCAFS LC-40

                                             Payload  PAYLOAD_MASS__KG_  \
0                    Dragon Spacecraft Qualification Unit                 0
1  Dragon demo flight C1, two CubeSats, barrel of...                 0
2                                Dragon demo flight C2                525
3                                          SpaceX CRS-1                500
4                                          SpaceX CRS-2                677

       Orbit         Customer Mission_Outcome      Landing_Outcome
0        LEO           SpaceX         Success  Failure (parachute)
1  LEO (ISS)  NASA (COTS) NRO         Success  Failure (parachute)
2  LEO (ISS)      NASA (COTS)         Success           No attempt
3  LEO (ISS)       NASA (CRS)         Success           No attempt
4  LEO (ISS)       NASA (CRS)         Success           No attempt
```

Total Payload Mass

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

```
# SQL query to get the total payload mass carried by boosters launched by NASA (CRS)
query = '''
SELECT SUM("PAYLOAD_MASS__KG_") AS Total_Payload_Mass
FROM SPACEXTBL
WHERE "Customer" = 'NASA (CRS)';
'''

# Execute the query and load the results into a DataFrame
total_payload_mass = pd.read_sql_query(query, conn)

# Display the results
print(total_payload_mass)
```

```
   Total_Payload_Mass
0               45596
```

Average Payload Mass by F9 v1.1

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

```
# SQL query to get the average payload mass carried by booster version F9 v1.1
query = '''
SELECT AVG("PAYLOAD_MASS__KG_") AS Average_Payload_Mass
FROM SPACEXTBL
WHERE "Booster_Version" = 'F9 v1.1';
'''

# Execute the query and load the results into a DataFrame
average_payload_mass = pd.read_sql_query(query, conn)

# Display the results
print(average_payload_mass)
```

```
   Average_Payload_Mass
0                2928.4
```

First Successful Ground Landing Date

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

Hint:Use min function

```
# SQL query to find the date of the first successful landing on a ground pad
query = '''
SELECT MIN("Date") AS First_Successful_Landing_Date
FROM SPACEXTBL
WHERE "Landing_Outcome" = 'Success (ground pad)';
'''

# Execute the query and load the results into a DataFrame
first_successful_landing_date = pd.read_sql_query(query, conn)

# Display the results
print(first_successful_landing_date)
```

```
  First_Successful_Landing_Date
0                    2015-12-22
```

Successful Drone Ship Landing with Payload between 4000 and 6000

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

```
# SQL query to list booster versions with specific criteria
query = '''
SELECT "Booster_Version"
FROM SPACEXTBL
WHERE "Landing_Outcome" = 'Success (drone ship)'
AND "PAYLOAD_MASS__KG_" > 4000
AND "PAYLOAD_MASS__KG_" < 6000;
'''

# Execute the query and load the results into a DataFrame
boosters_success_drone_ship = pd.read_sql_query(query, conn)

# Display the results
print(boosters_success_drone_ship)
```

```
  Booster_Version
0     F9 FT B1022
1     F9 FT B1026
2  F9 FT  B1021.2
3  F9 FT  B1031.2
```

Total Number of Successful and Failure Mission Outcomes

List the total number of successful and failure mission outcomes

```
# SQL query to get the total number of successful and failure mission outcomes
query = '''
SELECT "Mission_Outcome", COUNT(*) AS Outcome_Count
FROM SPACEXTBL
GROUP BY "Mission_Outcome";
'''

# Execute the query and load the results into a DataFrame
mission_outcomes = pd.read_sql_query(query, conn)

# Display the results
print(mission_outcomes)
```

```
                  Mission_Outcome  Outcome_Count
0              Failure (in flight)              1
1                         Success             98
2                         Success              1
3  Success (payload status unclear)              1
```

Boosters Carried Maximum Payload

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

```
# SQL query to list booster versions with the maximum payload mass using a subquery
query = '''
SELECT "Booster_Version"
FROM SPACEXTBL
WHERE "PAYLOAD_MASS__KG_" = (
    SELECT MAX("PAYLOAD_MASS__KG_")
    FROM SPACEXTBL
);
'''

# Execute the query and load the results into a DataFrame
boosters_max_payload = pd.read_sql_query(query, conn)

# Display the results
print(boosters_max_payload)
```

```
   Booster_Version
0    F9 B5 B1048.4
1    F9 B5 B1049.4
2    F9 B5 B1051.3
3    F9 B5 B1056.4
4    F9 B5 B1048.5
5    F9 B5 B1051.4
6    F9 B5 B1049.5
7   F9 B5 B1060.2
8   F9 B5 B1058.3
9    F9 B5 B1051.6
10   F9 B5 B1060.3
11  F9 B5 B1049.7
```

2015 Launch Records

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.

```
# SQL query to get records with specific criteria
query = '''
SELECT
    SUBSTR("Date", 6, 2) AS Month,
    "Landing_Outcome",
    "Booster_Version",
    "Launch_Site"
FROM SPACEXTBL
WHERE
    "Landing_Outcome" = 'Failure (drone ship)'
    AND SUBSTR("Date", 0, 5) = '2015';
'''

# Execute the query and load the results into a DataFrame
failure_landings_2015 = pd.read_sql_query(query, conn)

# Display the results
print(failure_landings_2015)
```

```
  Month       Landing_Outcome Booster_Version  Launch_Site
0    01  Failure (drone ship)   F9 v1.1 B1012  CCAFS LC-40
1    04  Failure (drone ship)   F9 v1.1 B1015  CCAFS LC-40
```

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

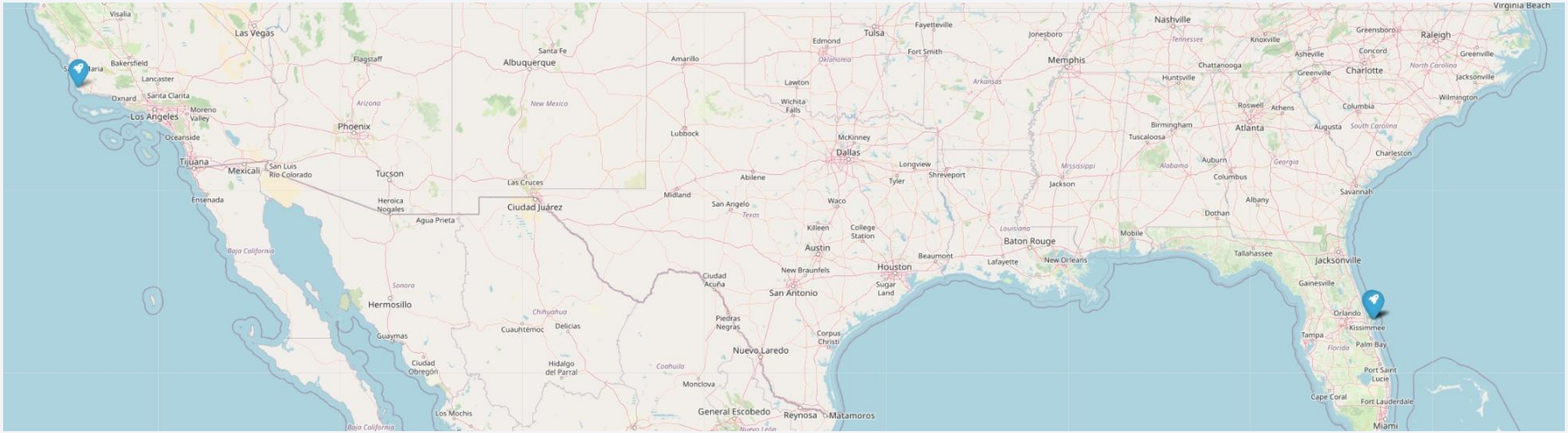
Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order.

```
# SQL query to rank the count of landing outcomes between the specified dates in descending order
query = '''
SELECT "Landing_Outcome", COUNT(*) AS Outcome_Count
FROM SPACEXTBL
WHERE "Date" BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY "Landing_Outcome"
ORDER BY Outcome_Count DESC;
'''

# Execute the query and load the results into a DataFrame
landing_outcomes_ranked = pd.read_sql_query(query, conn)

# Display the results
print(landing_outcomes_ranked)
```

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


Section 3
Launch Sites
Proximities Analysis

Location of SpaceX Launch Sites



1. Markers indicate that the SpaceX launch sites were located in California and Florida

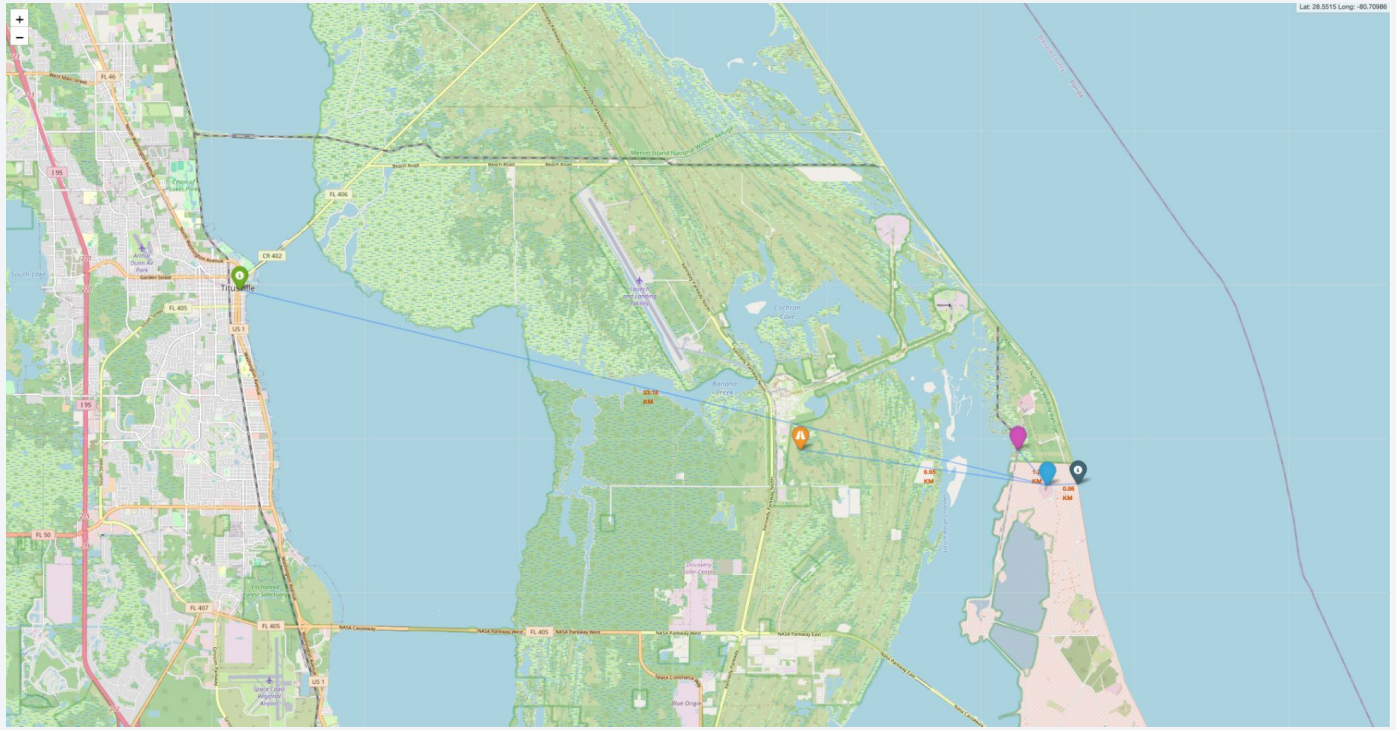
Florida Launch sites

CCAFS LC-40

CCAFS SLC-40

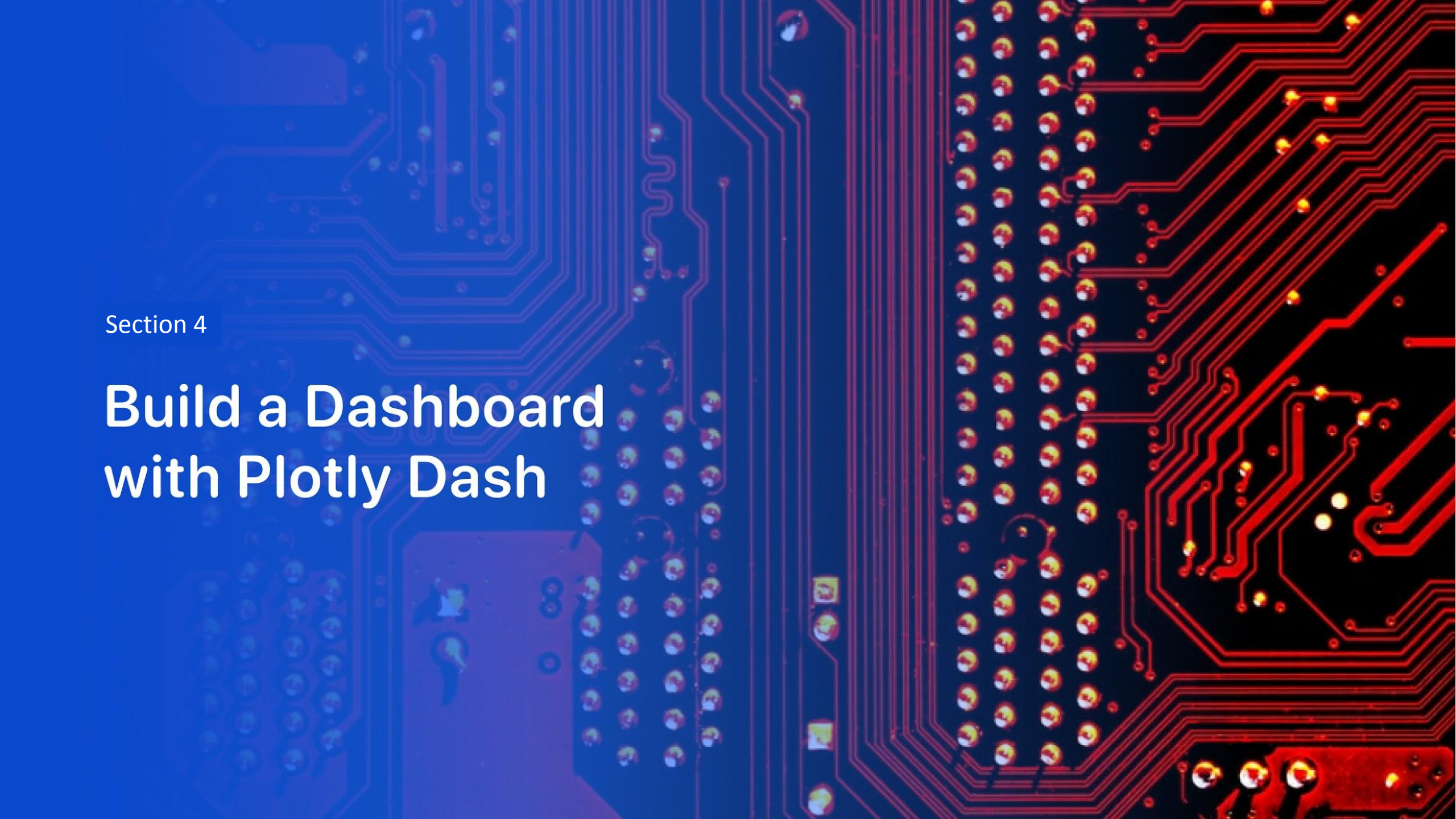
1. These two launch sites, CCAFS LC-40 and CCAFS SLC-40, in Florida are in close proximity to each other as shown on the map
2. The individual launch sites are enclosed by their respective circles
3. The success (green) or failure (red) of a launch is annotated by individual mark ups for each launch site

What is located near the launch site?

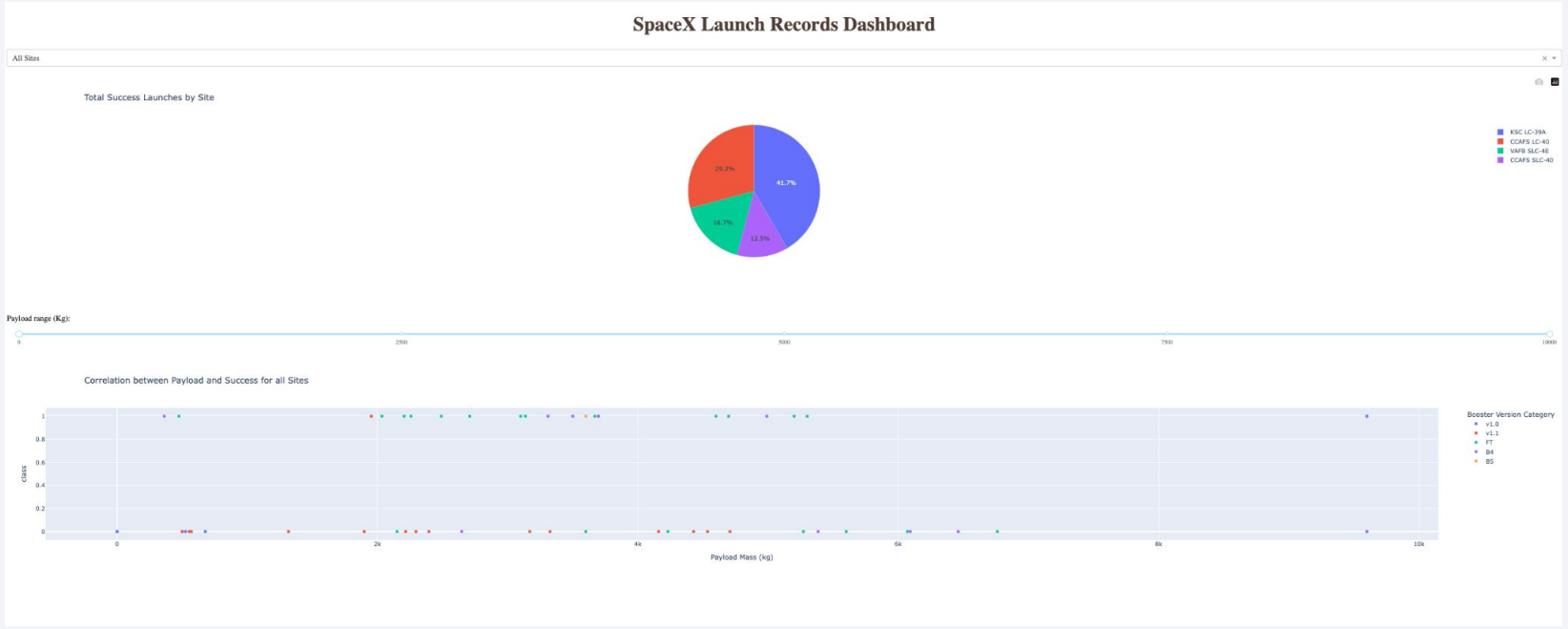


```
Distance between the launch site and the closest coastline point: 0.86 km
Distance between the launch site and Titusville, FL: 23.18 km
Distance between the launch site and the FEC Railway: 1.29 km
Distance between the launch site and the nearest highway: 6.95 km
```

1. Proximity to railroads and highways
   - Strategic placement facilitates the transportation of heavy equipment and materials required for launches
   - Efficient logistics and operational Support
2. Proximity to coastlines
   - Launching over the ocean reduces risk to populated areas and allow for safer disposal of rocket stages and other debris
3. Distance from cities
   - Maintaining a safe distance is crucial for minimize risk to Potential human life and infrastructure
   - Mitigates potential hazards during launch operation


Section 4
Build a Dashboard with Plotly Dash

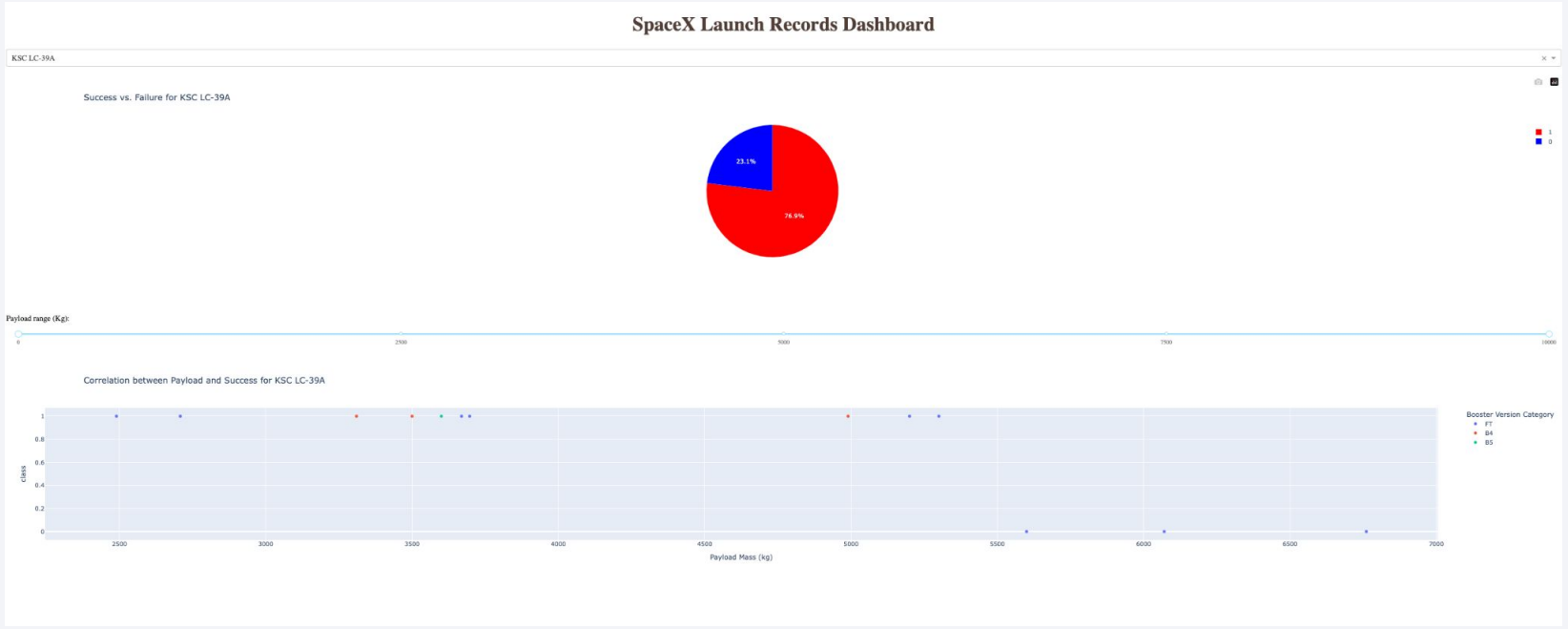
Successful Launches by Site



Dashboard Features

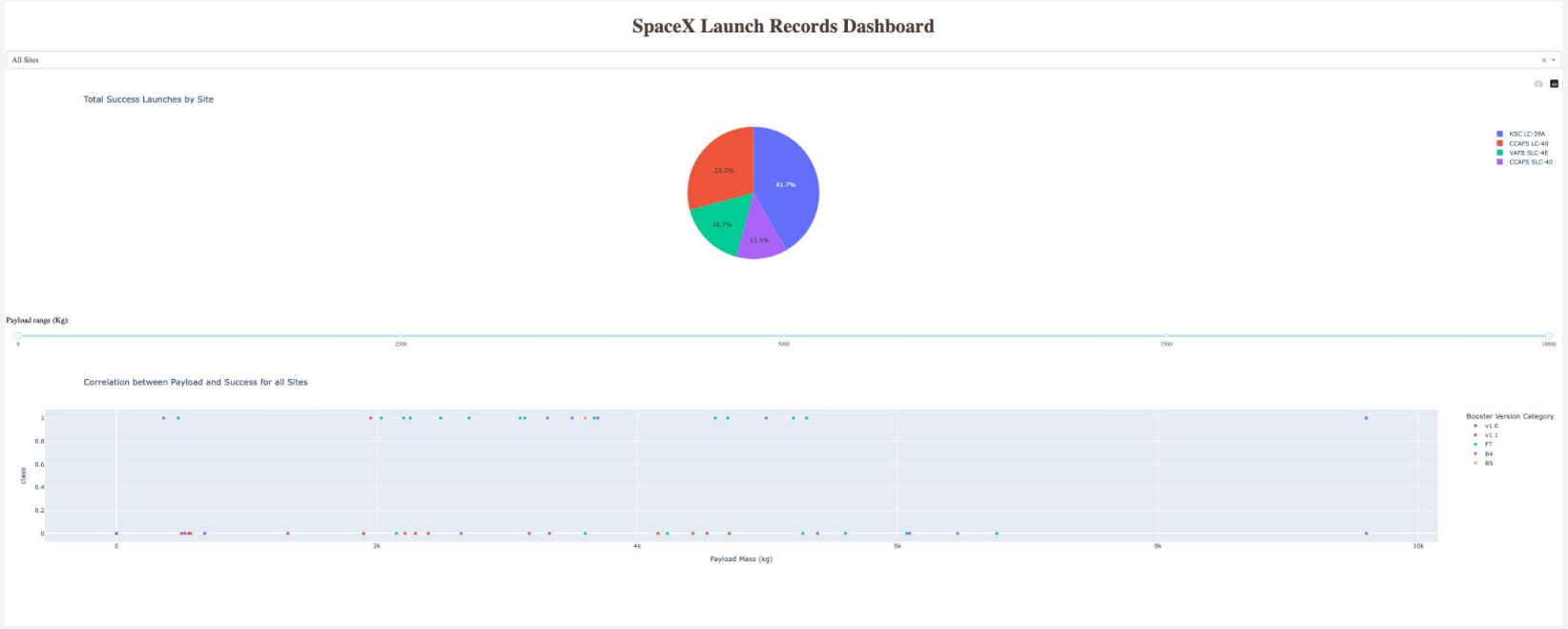
1. Range slider: Allows selection of payload mass that range from 0 kg up to 10,000 kg by increments of 2500 kg
2. Pie Chart callback: Updates based upon selected launch site
3. Scatter Chart callback: Updates scatter chart based upon selected launch site and payload range
    a. Correlate launch success and payload mass to color-labeled dots representing booster version category

Highest Launch Success Ratio



1. KSC LC-39A has the highest launch success rate of 76.9% out of 13 attempts
2. Payloads to 5500 kg were successful whereas the three attempts at higher payloads (5500 - 7000 kg) failed
3. Makes up 41.7% of the total successful launches for SpaceX
4. Launches from KSC LC-39A contribute to 41.7% of the total successful launches for SpaceX

Successful Launch of Payloads

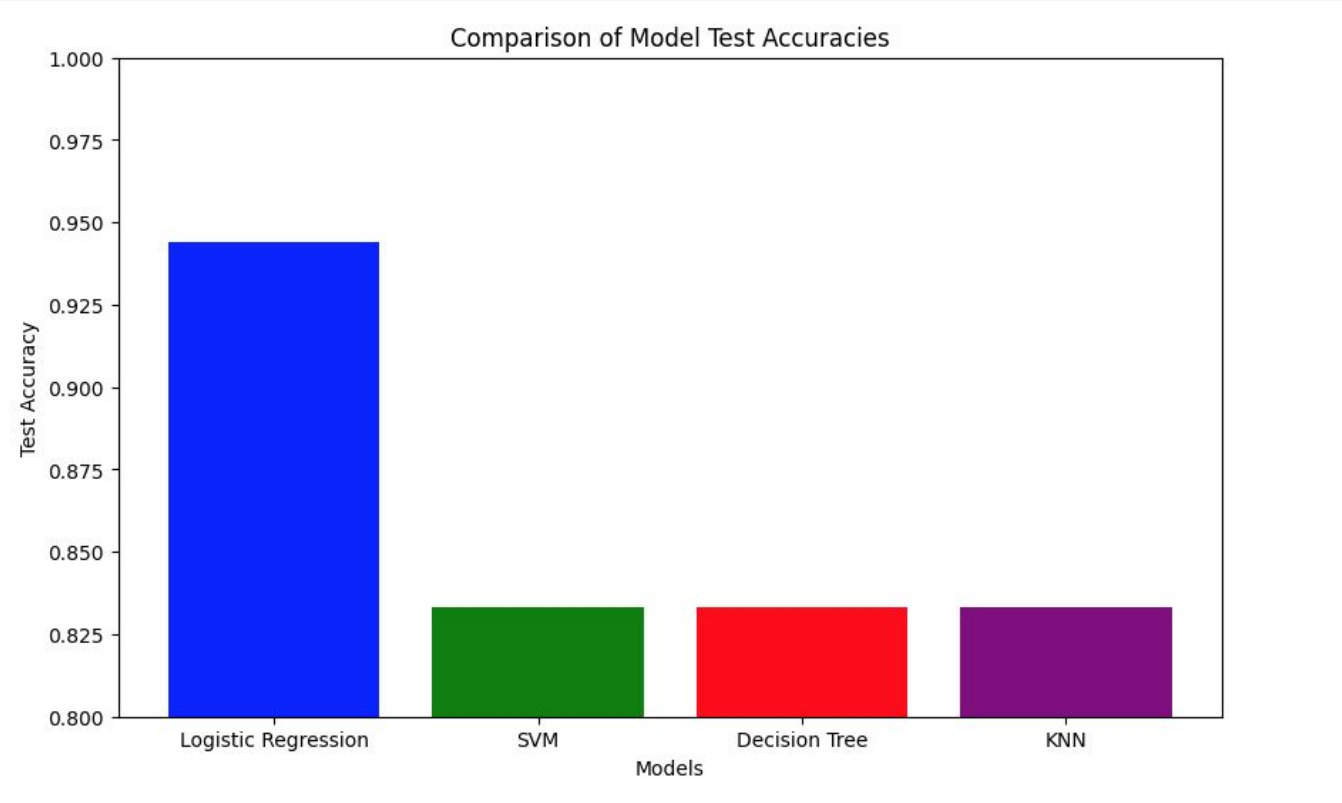


1. Highest rate of successful launches were with payloads in the 2000 - 4000 kg range
2. Least successful launches were with payloads greater than 6000 kg
3. Booster F9 FT was the most successful followed by F9 v1.1

Section 5

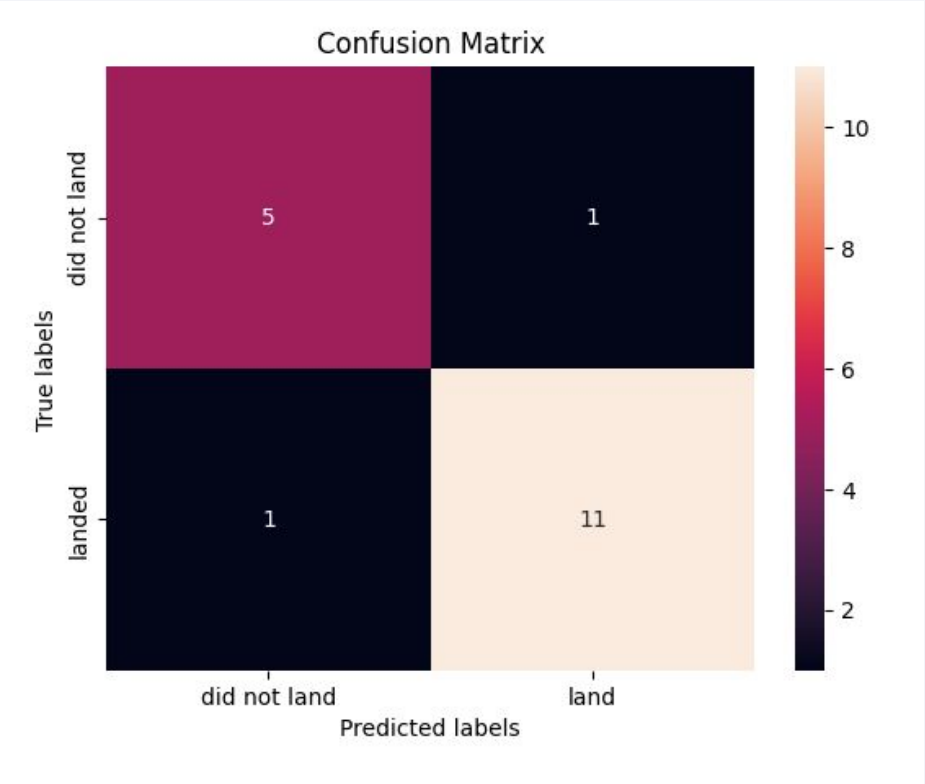
Predictive Analysis (Classification)

Classification Accuracy



1. Best accuracy for classification exhibited by Logistical Regression

Confusion Matrix



1. Logistical Regression demonstrated the highest accuracy (0.944) with fewer false positives and appears to be the most reliable based upon the confusion matrix
2. Logistical Regression is simpler and more interpretable

Conclusions

1. SpaceX was able to achieve a relative stable degree of success in their rocket launches through continually learning and innovating during the first 10 years of existence (2010-2020)
2. Although the frequency of recovery for the Falcon F9 first stage booster is positive, there is room for improvement in achieving certain orbits with the limited payload capacity
3. To be competitive, any newly formed entity needs to be conscientious that there exists a threshold of learning through execution that must be navigated in order to achieve success comparable to SpaceX.

Appendix

Appendix - Data Collection

```python
# Define the URL
static_json_url = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_call_spacex_api.json'

# Perform the GET request to fetch the data
response = requests.get(static_json_url)pd.json_normalize

# Decode the response content as JSON
json_data = response.json()

# Convert the JSON response to a Pandas dataframe
data = pd.json_normalize(json_data)

# Take a subset of the dataframe
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]

# Remove rows with multiple cores and payloads
data = data[data['cores'].map(len) == 1]
data = data[data['payloads'].map(len) == 1]

# Extract the single value from lists
data['cores'] = data['cores'].map(lambda x: x[0])
data['payloads'] = data['payloads'].map(lambda x: x[0])

# Convert date_utc to datetime and extract the date
data['date'] = pd.to_datetime(data['date_utc']).dt.date

# Restrict the dates of the launches
import datetime
data = data[data['date'] <= datetime.date(2020, 11, 13)]

# Display the modified dataframe
print(data.head())
```

```
                     rocket                  payloads  \
0  5e9d0d95eda69955f709d1eb  5eb0e4b5b6c3bb0006eeb1e1
1  5e9d0d95eda69955f709d1eb  5eb0e4b6b6c3bb0006eeb1e2
3  5e9d0d95eda69955f709d1eb  5eb0e4b7b6c3bb0006eeb1e5
4  5e9d0d95eda69955f709d1eb  5eb0e4b7b6c3bb0006eeb1e6
5  5e9d0d95eda69973a809d1ec  5eb0e4b7b6c3bb0006eeb1e7

                  launchpad  \
0  5e9e4502f5090995de566f86
1  5e9e4502f5090995de566f86
3  5e9e4502f5090995de566f86
4  5e9e4502f5090995de566f86
5  5e9e4501f509094ba4566f84

                                                                                                                                                                                                    cores  \
0  {'core': '5e9e289df35918033d3b2623', 'flight': 1, 'gridfins': False, 'legs': False, 'reused': False, 'landing_attempt': False, 'landing_success': None, 'landing_type': None, 'landpad': None}
1  {'core': '5e9e289ef35918416a3b2624', 'flight': 1, 'gridfins': False, 'legs': False, 'reused': False, 'landing_attempt': False, 'landing_success': None, 'landing_type': None, 'landpad': None}
3  {'core': '5e9e289ef3591855dc3b2626', 'flight': 1, 'gridfins': False, 'legs': False, 'reused': False, 'landing_attempt': False, 'landing_success': None, 'landing_type': None, 'landpad': None}
4  {'core': '5e9e289ef359184f103b2627', 'flight': 1, 'gridfins': False, 'legs': False, 'reused': False, 'landing_attempt': False, 'landing_success': None, 'landing_type': None, 'landpad': None}
5  {'core': '5e9e289ef359185f2b3b2628', 'flight': 1, 'gridfins': False, 'legs': False, 'reused': False, 'landing_attempt': False, 'landing_success': None, 'landing_type': None, 'landpad': None}

   flight_number                  date_utc        date
0              1  2006-03-24T22:30:00.000Z  2006-03-24
1              2  2007-03-21T01:10:00.000Z  2007-03-21
3              4  2008-09-28T23:15:00.000Z  2008-09-28
4              5  2009-07-13T03:35:00.000Z  2009-07-13
5              6  2010-06-04T18:45:00.000Z  2010-06-04
```

Then, we need to create a Pandas data frame from the dictionary launch_dict.

```python
# Create a data from launch_dict
import pandas as pd

# Sample dictionary containing the launch data
launch_dict = {
    'booster_version': BoosterVersion,
    'payload_mass': PayloadMass,
    'payload_orbit': Orbit,
    'launch_site': LaunchSite,
    'longitude': Longitude,
    'latitude': Latitude,
    'landing_outcome': Outcome,
    'landing_type': LandingPad,  # Assuming 'landpad' is the landing type here
    'core_flight': Flights,
    'gridfins_used': GridFins,
    'core_reused': Reused,
    'legs_used': Legs,
    'landing_pad': LandingPad,
    'core_block': Block,
    'times_reused': ReusedCount,
    'core_serial': Serial
}

# Create a new Pandas dataframe from the dictionary
launch_df = pd.DataFrame(launch_dict)

# Display the first few rows of the new dataframe
print(launch_df.head())
```

```
  booster_version  payload_mass payload_orbit      launch_site   longitude  \
0        Falcon 1          20.0           LEO  Kwajalein Atoll  167.743129
1        Falcon 1           NaN           LEO  Kwajalein Atoll  167.743129
2        Falcon 1         165.0           LEO  Kwajalein Atoll  167.743129
3        Falcon 1         200.0           LEO  Kwajalein Atoll  167.743129
4        Falcon 9           NaN           LEO      CCSFS SLC 40  -80.577366

    latitude landing_outcome landing_type  core_flight  gridfins_used  \
0   9.047721       None None         None            1          False
1   9.047721       None None         None            1          False
2   9.047721       None None         None            1          False
3   9.047721       None None         None            1          False
4  28.561857       None None         None            1          False

   core_reused  legs_used landing_pad  core_block  times_reused core_serial
0        False      False        None         NaN             0    Merlin1A
1        False      False        None         NaN             0    Merlin2A
2        False      False        None         NaN             0    Merlin2C
3        False      False        None         NaN             0    Merlin3C
4        False      False        None         1.0             0       B0003
```

Show the summary of the dataframe

```python
# Show the head of the dataframe
print(launch_df.head())
```

```
  booster_version  payload_mass payload_orbit      launch_site   longitude  \
0        Falcon 1          20.0           LEO  Kwajalein Atoll  167.743129
1        Falcon 1           NaN           LEO  Kwajalein Atoll  167.743129
2        Falcon 1         165.0           LEO  Kwajalein Atoll  167.743129
3        Falcon 1         200.0           LEO  Kwajalein Atoll  167.743129
4        Falcon 9           NaN           LEO      CCSFS SLC 40  -80.577366

    latitude landing_outcome landing_type  core_flight  gridfins_used  \
0   9.047721       None None         None            1          False
1   9.047721       None None         None            1          False
2   9.047721       None None         None            1          False
3   9.047721       None None         None            1          False
4  28.561857       None None         None            1          False

   core_reused  legs_used landing_pad  core_block  times_reused core_serial
0        False      False        None         NaN             0    Merlin1A
1        False      False        None         NaN             0    Merlin2A
2        False      False        None         NaN             0    Merlin2C
3        False      False        None         NaN             0    Merlin3C
4        False      False        None         1.0             0       B0003
```

```python
# Filter the dataframe to only include Falcon 9 launches using .loc[]
data_falcon9 = launch_df.loc[launch_df['booster_version'] != 'Falcon 1']

# Display the first few rows of the filtered dataframe
print(data_falcon9.head())
```

```
  booster_version  payload_mass payload_orbit   launch_site   longitude  \
4        Falcon 9           NaN           LEO  CCSFS SLC 40  -80.577366
5        Falcon 9         525.0           LEO  CCSFS SLC 40  -80.577366
6        Falcon 9         677.0           ISS  CCSFS SLC 40  -80.577366
7        Falcon 9         500.0            PO   VAFB SLC 4E -120.610829
8        Falcon 9        3170.0           GTO  CCSFS SLC 40  -80.577366

    latitude landing_outcome landing_type  core_flight  gridfins_used  \
4  28.561857       None None         None            1          False
5  28.561857       None None         None            1          False
6  28.561857       None None         None            1          False
7  34.632093     False Ocean         None            1          False
8  28.561857       None None         None            1          False

   core_reused  legs_used landing_pad  core_block  times_reused core_serial
4        False      False        None         1.0             0       B0003
5        False      False        None         1.0             0       B0005
6        False      False        None         1.0             0       B0007
7        False      False        None         1.0             0       B1003
8        False      False        None         1.0             0       B1004
```

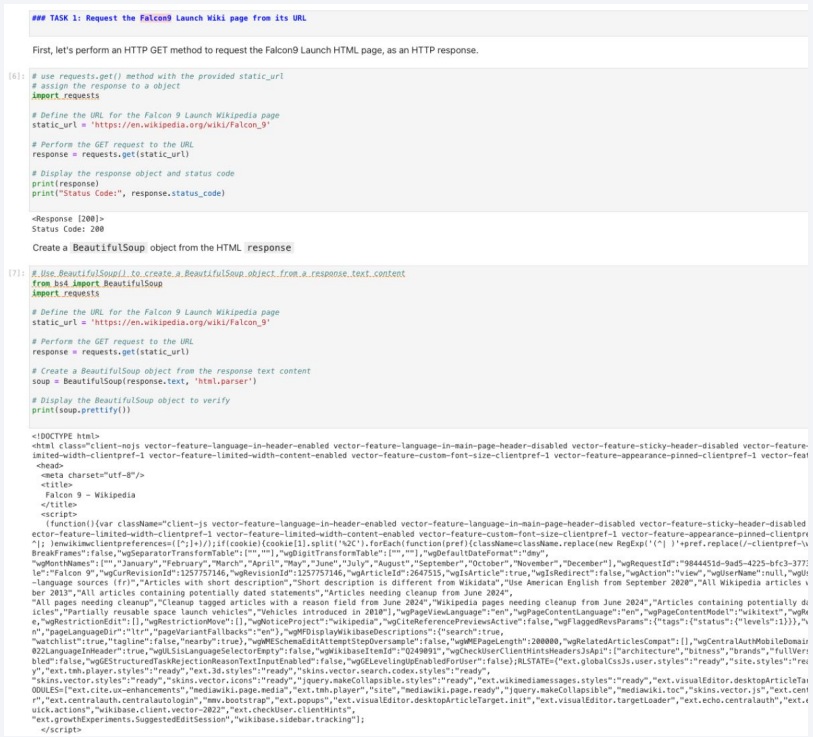
Calculate below the mean for the `PayloadMass` using the `.mean()`. Then use the mean and the `.replace()`

```python
# Calculate the mean value of PayloadMass column

# Replace the np.nan values with its mean value

import numpy as np

# Calculate the mean value of the PayloadMass column
mean_payload_mass = data_falcon9['payload_mass'].mean()

# Replace the np.nan values with the mean value using .loc[]
data_falcon9.loc[:, 'payload_mass'] = data_falcon9['payload_mass'].replace(np.nan, mean_payload_mass)

# Display the first few rows of the updated dataframe
print(data_falcon9.head())
data_falcon9.isnull().sum()
```

```
  booster_version  payload_mass payload_orbit   launch_site   longitude  \
4        Falcon 9   6123.547647           LEO  CCSFS SLC 40  -80.577366
5        Falcon 9    525.000000           LEO  CCSFS SLC 40  -80.577366
6        Falcon 9    677.000000           ISS  CCSFS SLC 40  -80.577366
7        Falcon 9    500.000000            PO   VAFB SLC 4E -120.610829
8        Falcon 9   3170.000000           GTO  CCSFS SLC 40  -80.577366

    latitude landing_outcome landing_type  core_flight  gridfins_used  \
4  28.561857       None None         None            1          False
5  28.561857       None None         None            1          False
6  28.561857       None None         None            1          False
7  34.632093     False Ocean         None            1          False
8  28.561857       None None         None            1          False

   core_reused  legs_used landing_pad  core_block  times_reused core_serial
4        False      False        None         1.0             0       B0003
5        False      False        None         1.0             0       B0005
6        False      False        None         1.0             0       B0007
7        False      False        None         1.0             0       B1003
8        False      False        None         1.0             0       B1004
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/pandas/core/indexing.py:1773: SettingWithCop
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html
  self._setitem_single_column(ilocs[0], value, pi)
booster_version     0
payload_mass        0
payload_orbit       0
launch_site         0
longitude           0
latitude            0
landing_outcome     0
landing_type       26
core_flight         0
gridfins_used       0
core_reused         0
legs_used           0
landing_pad        26
core_block          0
times_reused        0
core_serial         0
dtype: int64
```

You should see the number of missing values of the `PayLoadMass` change to zero.

Appendix - Web Scraping



TASK 1: Request the Falcon9 Launch Wiki page from its URL

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

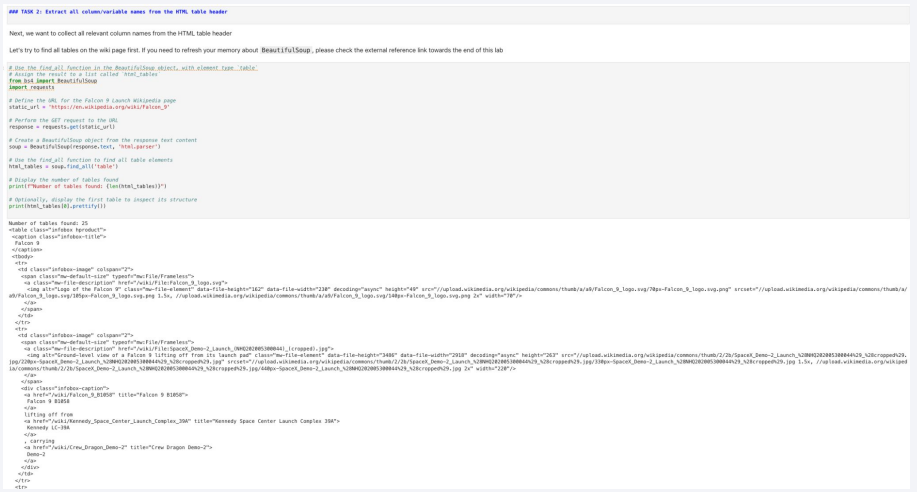
[6]:
```
# use requests.get() method with the provided static_url
# assign the response to a object
import requests

# Define the URL for the Falcon 9 Launch Wikipedia page
static_url = 'https://en.wikipedia.org/wiki/Falcon_9'

# Perform the GET request to the URL
response = requests.get(static_url)

# Display the response object and status code
print(response)
print("Status Code:", response.status_code)
```

```
<Response [200]>
Status Code: 200
```

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

[7]:
```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
from bs4 import BeautifulSoup
import requests

# Define the URL for the Falcon 9 Launch Wikipedia page
static_url = 'https://en.wikipedia.org/wiki/Falcon_9'

# Perform the GET request to the URL
response = requests.get(static_url)

# Create a BeautifulSoup object from the response text content
soup = BeautifulSoup(response.text, 'html.parser')

# Display the BeautifulSoup object to verify
print(soup.prettify())
```

```
<!DOCTYPE html>
<html class="client-nojs vector-feature-language-in-header-enabled vector-feature-language-in-main-page-header-disabled vector-feature-sticky-header-disabled vector-feature-
limited-width-clientpref-1 vector-feature-limited-width-content-enabled vector-feature-custom-font-size-clientpref-1 vector-feature-appearance-pinned-clientpref-1 vector-fea
 <head>
  <meta charset="utf-8"/>
  <title>
   Falcon 9 - Wikipedia
  </title>
  <script>
   (function(){var className="client-js vector-feature-language-in-header-enabled vector-feature-language-in-main-page-header-disabled vector-feature-sticky-header-disabled
ector-feature-limited-width-clientpref-1 vector-feature-limited-width-content-enabled vector-feature-custom-font-size-clientpref-1 vector-feature-appearance-pinned-clientpr
"[ ]enwikimwclientpreferences=([^;]+)/);if(cookie){cookie[1].split('%2C').forEach(function(pref){className=className.replace(new RegExp('(^| )'+pref.replace(/-/g,"\\-")+
BreakFrames":false,"wgSeparatorTransformTable":["",""],"wgDigitTransformTable":["",""],"wgDefaultDateFormat":"dmy",
"wgMonthNames":["","January","February","March","April","May","June","July","August","September","October","November","December"],"wgRequestId":"9844451d-9ad5-4225-bfc3-377
le":"Falcon 9","wgCurRevisionId":1257757146,"wgRevisionId":1257757146,"wgArticleId":2647515,"wgIsArticle":true,"wgIsRedirect":false,"wgAction":"view","wgUserName":null,"wgU
-language sources (fr)","Articles with short description","Short description is different from Wikidata","Use American English from September 2020","All Wikipedia articles
ber 2013","All articles containing potentially dated statements","Articles needing cleanup from June 2024",
"All pages needing cleanup","Cleanup tagged articles with a reason field from June 2024","Wikipedia pages needing cleanup from June 2024","Articles containing potentially d
icles","Partially reusable space launch vehicles","Vehicles introduced in 2010"],"wgPageViewLanguage":"en","wgPageContentLanguage":"en","wgPageContentModel":"wikitext","wgR
e,"wgRestrictionEdit":[],"wgRestrictionMove":[],"wgNoticeProject":"wikipedia","wgCiteReferencePreviewsActive":false,"wgFlaggedRevsParams":{"tags":{"status":{"levels":1}}},"
n","pageLanguageDir":"ltr","pageVariantFallbacks":"en"},"wgMFDisplayWikibaseDescriptions":{"search":true,
"watchlist":true,"tagline":false,"nearby":true},"wgWMESchemaEditAttemptStepOversample":false,"wgWMEPageLength":200000,"wgRelatedArticlesCompat":[],"wgCentralAuthMobileDomain
022LanguageInHeader":true,"wgULSisLanguageSelectorEmpty":false,"wgWikibaseItemId":"Q249091","wgCheckUserClientHintsHeadersJsApi":["architecture","bitness","brands","fullVer
bled":false,"wgGEStructuredTaskRejectionReasonTextInputEnabled":false,"wgGELevelingUpEnabledForUser":false};RLSTATE={"ext.globalCssJs.user.styles":"ready","site.styles":"re
y","ext.tmh.player.styles":"ready","ext.3d.styles":"ready","skins.vector.search.codex.styles":"ready",
"skins.vector.styles":"ready","skins.vector.icons":"ready","jquery.makeCollapsible.styles":"ready","ext.wikimediamessages.styles":"ready","ext.visualEditor.desktopArticleTa
ODULES=["ext.cite.ux-enhancements","mediawiki.page.media","ext.tmh.player","site","mediawiki.page.ready","jquery.makeCollapsible","mediawiki.toc","skins.vector.js","ext.cen
r","ext.centralauth.centralautologin","mmv.bootstrap","ext.popups","ext.visualEditor.desktopArticleTarget.init","ext.visualEditor.targetLoader","ext.echo.centralauth","ext.e
uick.actions","wikibase.client.vector-2022","ext.checkUser.clientHints",
"ext.growthExperiments.SuggestedEditSession","wikibase.sidebar.tracking"];
  </script>
```



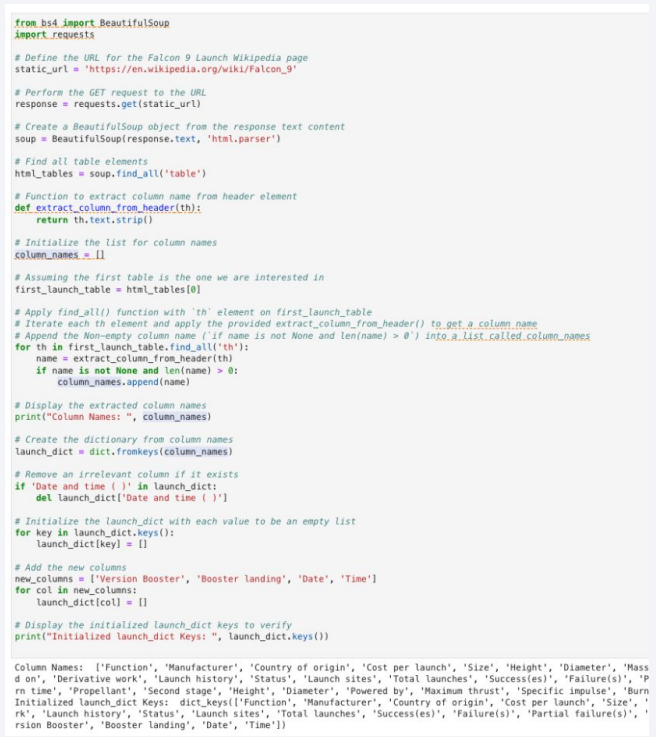
TASK 2: Extract all column/variable names from the HTML table header

Next, we want to collect all relevant column names from the HTML table header

Let's try to find all tables on the wiki page first. If you need to refresh your memory about `BeautifulSoup`, please check the external reference link towards the end of this lab

```
# Use the find_all function in the BeautifulSoup object, with element type `table`
# Assign the result to a list called `html_tables`
from bs4 import BeautifulSoup
import requests

# Define the URL for the Falcon 9 Launch Wikipedia page
static_url = "https://en.wikipedia.org/wiki/Falcon_9"

# Perform the GET request to the URL
response = requests.get(static_url)

# Create a BeautifulSoup object from the response text content
soup = BeautifulSoup(response.text, 'html.parser')

# Use the find_all function to find all table elements
html_tables = soup.find_all('table')

# Display the number of tables found
print(f"Number of tables found: {len(html_tables)}")

# Optionally, display the first table to inspect its structure
print(html_tables[0].prettify())
```

```
Number of tables found: 25
<table class="infobox hproduct">
 <caption class="infobox-title">
  Falcon 9
 </caption>
 <tbody>
  <tr>
   <td class="infobox-image" colspan="2">
    <span class="mw-default-size" typeof="mw:File/Frameless">
     <a class="mw-file-description" href="/wiki/File:Falcon_9_logo.svg">
      <img alt="Logo of the Falcon 9" class="mw-file-element" data-file-height="162" data-file-width="230" decoding="async" height="49" src="//upload.wikimedia.org/wikipedia/commons/thumb/a/ab/Falcon_9_logo.svg/70px-Falcon_9_logo.svg.png" srcset="//upload.wikimedia.org/wikipedia/commons/thumb/a/
ab/Falcon_9_logo.svg/105px-Falcon_9_logo.svg.png 1.5x, //upload.wikimedia.org/wikipedia/commons/thumb/a/ab/Falcon_9_logo.svg/140px-Falcon_9_logo.svg.png 2x" width="70" />
     </a>
    </span>
   </td>
  </tr>
  <tr>
   <td class="infobox-image" colspan="2">
    <span class="mw-default-size" typeof="mw:File/Frameless">
     <a class="mw-file-description" href="/wiki/File:SpaceX_Demo-2_Launch_(KHQ20200530004A4)_(cropped).jpg">
      <img alt="Ground-level view of a Falcon 9 lifting off from its launch pad" class="mw-file-element" data-file-height="3400" data-file-width="2918" decoding="async" height="263" src="//upload.wikimedia.org/wikipedia/commons/thumb/2/2b/SpaceX_Demo-2_Launch_%28KHQ20200530004A4%29_%28cropped%29.
jpg/220px-SpaceX_Demo-2_Launch_%28KHQ20200530004A4%29_%28cropped%29.jpg" srcset="//upload.wikimedia.org/wikipedia/commons/thumb/2/2b/SpaceX_Demo-2_Launch_%28KHQ20200530004A4%29_%28cropped%29.jpg/330px-SpaceX_Demo-2_Launch_%28KHQ20200530004A4%29_%28cropped%29.jpg 1.5x, //upload.wikimedia.org/wikiped
ia/commons/thumb/2/2b/SpaceX_Demo-2_Launch_%28KHQ20200530004A4%29_%28cropped%29.jpg/440px-SpaceX_Demo-2_Launch_%28KHQ20200530004A4%29_%28cropped%29.jpg 2x" width="220" />
     </a>
    </span>
    <div>
     <div class="infobox-caption">
      <a href="/wiki/Falcon_9_B1058" title="Falcon 9 B1058">
       Falcon 9 B1058
      </a>
      lifting off from
      <a href="/wiki/Kennedy_Space_Center_Launch_Complex_39A" title="Kennedy Space Center Launch Complex 39A">
       Kennedy LC-39A
      </a>
      , carrying
      <a href="/wiki/Crew_Dragon_Demo-2" title="Crew Dragon Demo-2">
       Demo-2
      </a>
     </div>
    </td>
   </tr>
   <tr>
```



```
from bs4 import BeautifulSoup
import requests

# Define the URL for the Falcon 9 Launch Wikipedia page
static_url = 'https://en.wikipedia.org/wiki/Falcon_9'

# Perform the GET request to the URL
response = requests.get(static_url)

# Create a BeautifulSoup object from the response text content
soup = BeautifulSoup(response.text, 'html.parser')

# Find all table elements
html_tables = soup.find_all('table')

# Function to extract column name from header element
def extract_column_from_header(th):
    return th.text.strip()

# Initialize the list for column names
column_names = []

# Assuming the first table is the one we are interested in
first_launch_table = html_tables[0]

# Apply find_all() function with `th` element on first_launch_table
# Iterate each th element and apply the provided extract_column_from_header() to get a column name
# Append the Non-empty column name (`if name is not None and len(name) > 0`) into a list called column_names
for th in first_launch_table.find_all('th'):
    name = extract_column_from_header(th)
    if name is not None and len(name) > 0:
        column_names.append(name)

# Display the extracted column names
print("Column Names: ", column_names)

# Create the dictionary from column names
launch_dict = dict.fromkeys(column_names)

# Remove an irrelevant column if it exists
if 'Date and time ( )' in launch_dict:
    del launch_dict['Date and time ( )']

# Initialize the launch_dict with each value to be an empty list
for key in launch_dict.keys():
    launch_dict[key] = []

# Add the new columns
new_columns = ['Version Booster', 'Booster landing', 'Date', 'Time']
for col in new_columns:
    launch_dict[col] = []

# Display the initialized launch_dict keys to verify
print("Initialized launch_dict Keys: ", launch_dict.keys())
```

```
Column Names:  ['Function', 'Manufacturer', 'Country of origin', 'Cost per launch', 'Size', 'Height', 'Diameter', 'Mass
d on', 'Derivative work', 'Launch history', 'Status', 'Launch sites', 'Total launches', 'Success(es)', 'Failure(s)', 'P
rn time', 'Propellant', 'Second stage', 'Height', 'Diameter', 'Powered by', 'Maximum thrust', 'Specific impulse', 'Burn
Initialized launch_dict Keys:  dict_keys(['Function', 'Manufacturer', 'Country of origin', 'Cost per launch', 'Size', '
rk', 'Launch history', 'Status', 'Launch sites', 'Total launches', 'Success(es)', 'Failure(s)', 'Partial failure(s)', '
rsion Booster', 'Booster landing', 'Date', 'Time'])
```

Appendix - Data Wrangling

TASK 1: Calculate the number of launches on each site

The data contains several Space X launch facilities: Cape Canaveral Space Launch Complex 40 **VAFB SLC 4E** , Vandenberg Air Force Base Space Launch Complex 4E **(SLC-4E)**, Kennedy Space Center Launch Complex 39A **KSC LC 39A** .The location of each Launch Is placed in the column `LaunchSite`

Next, let's see the number of launches for each site.

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

```
# Print the column names of the DataFrame to verify the correct names
print(df.columns)

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

# Display the number of launches for each site
print(launch_site_counts)
```

```
Index(['FlightNumber', 'Date', 'BoosterVersion', 'PayloadMass', 'Orbit',
       'LaunchSite', 'Outcome', 'Flights', 'GridFins', 'Reused', 'Legs',
       'LandingPad', 'Block', 'ReusedCount', 'Serial', 'Longitude',
       'Latitude'],
      dtype='object')
LaunchSite
CCAFS SLC 40    55
KSC LC 39A      22
VAFB SLC 4E     13
Name: count, dtype: int64
```

Each launch aims to an dedicated orbit, and here are some common orbit types:

TASK 2: Calculate the number and occurrence of each orbit

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

```
# Print the column names of the DataFrame to verify the correct names
print(df.columns)

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

# Display the number and occurrence for each orbit
print(orbit_counts)
```

```
Index(['FlightNumber', 'Date', 'BoosterVersion', 'PayloadMass', 'Orbit',
       'LaunchSite', 'Outcome', 'Flights', 'GridFins', 'Reused', 'Legs',
       'LandingPad', 'Block', 'ReusedCount', 'Serial', 'Longitude',
       'Latitude'],
      dtype='object')
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
```

TASK 4: Create a landing outcome label from Outcome column

Using the `Outcome` , create a list where the element is zero if the corresponding row in `Outcome` is in the set `bad_outcome` ; otherwise, it's one. Then assign it to the variable `landing_class` :

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

# Define the set of bad outcomes
bad_outcome = {'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}  # Adjust this set based on what constitutes a bad outcome in your data

# Create the landing_class list
landing_class = [0 if outcome in bad_outcome else 1 for outcome in df['Outcome']]

# Assign the landing_class list to the DataFrame
df['landing_class'] = landing_class

# Display the first few rows of the updated DataFrame
print(df.head())
```

```
   FlightNumber        Date BoosterVersion  PayloadMass Orbit    LaunchSite  \
0             1  2010-06-04       Falcon 9  6104.959412   LEO  CCAFS SLC 40
1             2  2012-05-22       Falcon 9   525.000000   LEO  CCAFS SLC 40
2             3  2013-03-01       Falcon 9   677.000000   ISS  CCAFS SLC 40
3             4  2013-09-29       Falcon 9   500.000000    PO   VAFB SLC 4E
4             5  2013-12-03       Falcon 9  3170.000000   GTO  CCAFS SLC 40

       Outcome  Flights  GridFins  Reused   Legs LandingPad  Block  \
0    None None        1     False   False  False        NaN    1.0
1    None None        1     False   False  False        NaN    1.0
2    None None        1     False   False  False        NaN    1.0
3  False Ocean        1     False   False  False        NaN    1.0
4    None None        1     False   False  False        NaN    1.0

   ReusedCount Serial   Longitude   Latitude  landing_class
0            0  B0003  -80.577366  28.561857              0
1            0  B0005  -80.577366  28.561857              0
2            0  B0007  -80.577366  28.561857              0
3            0  B1003 -120.610829  34.632093              0
4            0  B1004  -80.577366  28.561857              0
```

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 Succes

TASK 3: Calculate the number and occurence of mission outcome of the orbits

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

```
[10]: # landing_outcomes = values on Outcome column

      # Apply value_counts() on the 'landing_outcome' column to determine the number of landing outcomes
      landing_outcomes = df['Outcome'].value_counts()

      # Display the number of landing outcomes
      print(landing_outcomes)
```

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

`True Ocean` means the mission outcome was successfully landed to a specific region of the ocean while `False Ocean` means the mission outcome was unsuccessfully landed to a specific region of the ocean. `True RTLS` means the mission outcome was successfully landed to a ground pad `False RTLS` means the mission outcome was unsuccessfully landed to a ground pad. `True ASDS` means the mission outcome was successfully landed to a drone ship `False ASDS` means the mission outcome was unsuccessfully landed to a drone ship. `None ASDS` and `None None` these represent a failure to land.

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

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

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

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

```
[12]: {'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

Appendix - EDA Visual Analytics code

TASK 1: Visualize the relationship between Flight Number and Launch Site

Use the function `catplot` to plot `FlightNumber` vs `LaunchSite`, set the parameter `x` parameter to `FlightNumber`,set the `y` to `Launch Site` and set the parameter `hue` to `'class'`

```
# Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to be the class value
import seaborn as sns
import matplotlib.pyplot as plt

# Plot a scatter point chart with x axis as Flight Number, y axis as Launch Site, and hue as class value
sns.catplot(x="FlightNumber", y="LaunchSite", hue="Class", data=df, aspect=3, kind="strip")

# Set the axis labels and title
plt.xlabel("Flight Number", fontsize=15)
plt.ylabel("Launch Site", fontsize=15)
plt.title("Flight Number vs. Launch Site", fontsize=20)

# Display the plot
plt.show()
```

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.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Plot a scatter point chart with x axis as Payload Mass, y axis as Launch Site, and hue as class value
sns.catplot(x="PayloadMass", y="LaunchSite", hue="Class", data=df, aspect=3, kind="strip")

# Set the axis labels and title
plt.xlabel("Payload Mass (kg)", fontsize=15)
plt.ylabel("Launch Site", fontsize=15)
plt.title("Payload Mass vs. Launch Site", fontsize=20)

# Display the plot
plt.show()
```

TASK 3: Visualize the relationship between success rate of each orbit type

Next, we want to visually check if there are any relationship between success rate and orbit type.

Let's create a `bar chart` for the sucess rate of each orbit

```
# HINT use groupby method on Orbit column and get the mean of Class column

# Calculate the success rate for each orbit type
orbit_success_rate = df.groupby('Orbit')['Class'].mean().reset_index()

# Rename the columns for better understanding
orbit_success_rate.columns = ['Orbit', 'Success_Rate']

# Create a bar chart for the success rate of each orbit type
plt.figure(figsize=(12, 6))
plt.bar(orbit_success_rate['Orbit'], orbit_success_rate['Success_Rate'], color='skyblue')

# Set the axis labels and title
plt.xlabel("Orbit Type", fontsize=15)
plt.ylabel("Success Rate", fontsize=15)
plt.title("Success Rate by Orbit Type", fontsize=20)
plt.xticks(rotation=45)

# Display the plot
plt.show()
```

TASK 4: Visualize the relationship between FlightNumber and Orbit type

For each orbit, we want to see if there is any relationship between FlightNumber and Orbit type.

```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class value
import seaborn as sns
import matplotlib.pyplot as plt

# Plot a scatter point chart with x axis as Flight Number, y axis as Orbit, and hue as class value
sns.catplot(x="FlightNumber", y="Orbit", hue="Class", data=df, aspect=3, kind="strip")

# Set the axis labels and title
plt.xlabel("Flight Number", fontsize=15)
plt.ylabel("Orbit Type", fontsize=15)
plt.title("Flight Number vs. Orbit Type", fontsize=20)

# Display the plot
plt.show()
```

TASK 5: Visualize the relationship between Payload Mass and Orbit type

Similarly, we can plot the Payload Mass vs. Orbit scatter point charts to reveal the relationship between Payload Mass and Orbit type

```
# Plot a scatter point chart with x axis to be Payload Mass and y axis to be the Orbit, and hue to be the class value
import seaborn as sns
import matplotlib.pyplot as plt

# Plot a scatter point chart with x axis as Payload Mass, y axis as Orbit, and hue as class value
sns.catplot(x="PayloadMass", y="Orbit", hue="Class", data=df, aspect=3, kind="strip")

# Set the axis labels and title
plt.xlabel("Payload Mass (kg)", fontsize=15)
plt.ylabel("Orbit Type", fontsize=15)
plt.title("Payload Mass vs. Orbit Type", fontsize=20)

# Display the plot
plt.show()
```

```
# Calculate the average success rate for each year
yearly_success_rate = df.groupby('Year')['Class'].mean().reset_index()

# Rename the columns for better understanding
yearly_success_rate.columns = ['Year', 'Success_Rate']

import matplotlib.pyplot as plt

# Plot a line chart with x axis as Year and y axis as Success Rate
plt.figure(figsize=(12, 6))
plt.plot(yearly_success_rate['Year'], yearly_success_rate['Success_Rate'], marker='o', linestyle='-', color='b')

# Set the axis labels and title
plt.xlabel("Year", fontsize=15)
plt.ylabel("Average Success Rate", fontsize=15)
plt.title("Yearly Launch Success Rate Trend", fontsize=20)
plt.grid(True)

# Display the plot
plt.show()
```

Appendix - EDA SQL

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

```
# Write the SQL query to get unique launch sites
query = 'SELECT DISTINCT "Launch_Site" FROM SPACEXTBL'

# Execute the query and load the results into a DataFrame
unique_launch_sites = pd.read_sql_query(query, conn)

# Display the results
print(unique_launch_sites)
```

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

```
        Date Time (UTC) Booster_Version  Launch_Site  \
0  2010-06-04   18:45:00  F9 v1.0  B0003  CCAFS LC-40
1  2010-12-08   15:43:00  F9 v1.0  B0004  CCAFS LC-40
2  2012-05-22    7:44:00  F9 v1.0  B0005  CCAFS LC-40
3  2012-10-08    0:35:00  F9 v1.0  B0006  CCAFS LC-40
4  2013-03-01   15:10:00  F9 v1.0  B0007  CCAFS LC-40

                                        Payload  PAYLOAD_MASS__KG_  \
0              Dragon Spacecraft Qualification Unit                 0
1  Dragon demo flight C1, two CubeSats, barrel of...                0
2                            Dragon demo flight C2                525
3                                      SpaceX CRS-1                500
4                                      SpaceX CRS-2                677

       Orbit         Customer Mission_Outcome      Landing_Outcome
0        LEO           SpaceX         Success  Failure (parachute)
1  LEO (ISS)  NASA (COTS) NRO         Success  Failure (parachute)
2  LEO (ISS)      NASA (COTS)         Success           No attempt
3  LEO (ISS)       NASA (CRS)         Success           No attempt
4  LEO (ISS)       NASA (CRS)         Success           No attempt
Index(['Date', 'Time (UTC)', 'Booster_Version', 'Launch_Site', 'Payload',
       'PAYLOAD_MASS__KG_', 'Orbit', 'Customer', 'Mission_Outcome',
       'Landing_Outcome'],
      dtype='object')
```

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

```
# SQL query to get the average payload mass carried by booster version F9 v1.1
query = '''
SELECT AVG("PAYLOAD_MASS__KG_") AS Average_Payload_Mass
FROM SPACEXTBL
WHERE "Booster_Version" = 'F9 v1.1';
'''

# Execute the query and load the results into a DataFrame
average_payload_mass = pd.read_sql_query(query, conn)

# Display the results
print(average_payload_mass)
```

```
   Average_Payload_Mass
0                2928.4
```

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

```
# SQL query to list booster versions with the maximum payload mass using a subquery
query = '''
SELECT "Booster_Version"
FROM SPACEXTBL
WHERE "PAYLOAD_MASS__KG_" = (
    SELECT MAX("PAYLOAD_MASS__KG_")
    FROM SPACEXTBL
);
'''

# Execute the query and load the results into a DataFrame
boosters_max_payload = pd.read_sql_query(query, conn)

# Display the results
print(boosters_max_payload)
```

```
   Booster_Version
0    F9 B5 B1048.4
1    F9 B5 B1049.4
2    F9 B5 B1051.3
3    F9 B5 B1056.4
4    F9 B5 B1048.5
5    F9 B5 B1051.4
6    F9 B5 B1049.5
7   F9 B5 B1060.2
8   F9 B5 B1058.3
9    F9 B5 B1051.6
10   F9 B5 B1060.3
11  F9 B5 B1049.7
```

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.

```
# SQL query to rank the count of landing outcomes between the specified dates in descending order
query = '''
SELECT "Landing_Outcome", COUNT(*) AS Outcome_Count
FROM SPACEXTBL
WHERE "Date" BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY "Landing_Outcome"
ORDER BY Outcome_Count DESC;
'''

# Execute the query and load the results into a DataFrame
landing_outcomes_ranked = pd.read_sql_query(query, conn)

# Display the results
print(landing_outcomes_ranked)
```

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

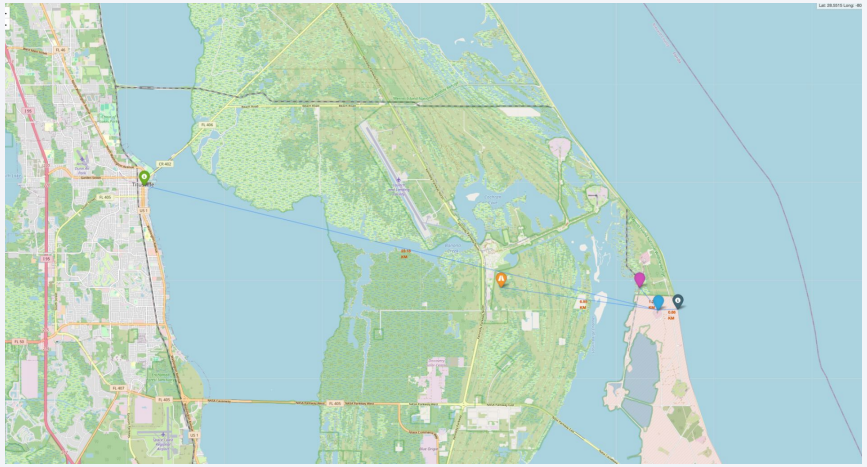
Appendix - Visualizations

```python
# Create a marker with distance to a closest city, railway, highway, etc.
# Draw a line between the marker to the launch site

import folium
from folium.plugins import MarkerCluster, MousePosition
from folium.features import DivIcon
import numpy as np

def haversine(lat1, lon1, lat2, lon2):
    # Convert latitude and longitude from degrees to radians
    lat1, lon1, lat2, lon2 = map(np.radians, [lat1, lon1, lat2, lon2])

    # Haversine formula
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = np.sin(dlat/2.0)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2.0)**2
    c = 2 * np.arcsin(np.sqrt(a))
    r = 6371  # Radius of Earth in kilometers. Use 3956 for miles.
    return c * r

# Coordinates of the launch site (CCAFS SLC-40)
launch_site_lat = 28.563197
launch_site_lon = -80.576820

# Coordinates of the closest coastline point
coastline_lat = 28.56353
coastline_lon = -80.56798

# Coordinates of Titusville, FL
titusville_lat = 28.6122
titusville_lon = -80.8076

# Coordinates of the FEC Railway near Cape Canaveral, FL
railway_lat = 28.5721
railway_lon = -80.5853

# Coordinates of the nearest point on Max Brewer Causeway (Route 405)
highway_lat = 28.5721
highway_lon = -80.6473

# Calculate distances
distance_coastline = haversine(launch_site_lat, launch_site_lon, coastline_lat, coastline_lon)
distance_titusville = haversine(launch_site_lat, launch_site_lon, titusville_lat, titusville_lon)
distance_railway = haversine(launch_site_lat, launch_site_lon, railway_lat, railway_lon)
distance_highway = haversine(launch_site_lat, launch_site_lon, highway_lat, highway_lon)

print(f"Distance between the launch site and the closest coastline point: {distance_coastline:.2f} km")
print(f"Distance between the launch site and Titusville, FL: {distance_titusville:.2f} km")
print(f"Distance between the launch site and the FEC Railway: {distance_railway:.2f} km")
print(f"Distance between the launch site and the nearest highway: {distance_highway:.2f} km")

# Start location is NASA Johnson Space Center
nasa_coordinate = [29.559684888503615, -95.0830971930759]
site_map = folium.Map(location=nasa_coordinate, zoom_start=5)

# Initialize MarkerCluster
marker_cluster = MarkerCluster().add_to(site_map)

# Add a marker for the launch site (CCAFS SLC-40)
launch_site_marker = folium.Marker(
    location=[launch_site_lat, launch_site_lon],
    popup='CCAFS SLC-40 Launch Site',
    icon=folium.Icon(color='blue', icon='rocket')
)
site_map.add_child(launch_site_marker)
```

```python
# Add a marker for the closest coastline point
coastline_marker = folium.Marker(
    location=[coastline_lat, coastline_lon],
    popup='Closest Coastline',
    icon=folium.Icon(color='cadetblue', icon='info-sign')
)
site_map.add_child(coastline_marker)

# Add a marker for Titusville, FL
titusville_marker = folium.Marker(
    location=[titusville_lat, titusville_lon],
    popup='Titusville, FL',
    icon=folium.Icon(color='green', icon='info-sign')
)
site_map.add_child(titusville_marker)

# Add a marker for the FEC Railway
railway_marker = folium.Marker(
    location=[railway_lat, railway_lon],
    popup='FEC Railway',
    icon=folium.Icon(color='purple', icon='train')
)
site_map.add_child(railway_marker)

# Add a marker for the nearest highway (Max Brewer Causeway)
highway_marker = folium.Marker(
    location=[highway_lat, highway_lon],
    popup='Nearest Highway (Max Brewer Causeway)',
    icon=folium.Icon(color='orange', icon='road')
)
site_map.add_child(highway_marker)

# Add markers displaying the distances
distance_markers = [
    ((launch_site_lat + coastline_lat) / 2, (launch_site_lon + coastline_lon) / 2, distance_coastline, 'coastline', '#d35400'),
    ((launch_site_lat + titusville_lat) / 2, (launch_site_lon + titusville_lon) / 2, distance_titusville, 'Titusville', '#d35400'),
    ((launch_site_lat + railway_lat) / 2, (launch_site_lon + railway_lon) / 2, distance_railway, 'railway', '#d35400'),
    ((launch_site_lat + highway_lat) / 2, (launch_site_lon + highway_lon) / 2, distance_highway, 'highway', '#d35400')
]

for lat, lon, distance, label, color in distance_markers:
    distance_marker = folium.Marker(
        location=[lat, lon],
        icon=DivIcon(
            icon_size=(20, 20),
            icon_anchor=(0, 0),
            html=f'<div style="font-size: 12; color:{color};"><b>{distance:.2f} KM</b></div>'
        )
    )
    site_map.add_child(distance_marker)

# Create PolyLines between the launch site and each location
coordinates_list = [
    ([launch_site_lat, launch_site_lon], [coastline_lat, coastline_lon]),
    ([launch_site_lat, launch_site_lon], [titusville_lat, titusville_lon]),
    ([launch_site_lat, launch_site_lon], [railway_lat, railway_lon]),
    ([launch_site_lat, launch_site_lon], [highway_lat, highway_lon])
]

for coordinates in coordinates_list:
    lines = folium.PolyLine(locations=coordinates, weight=1)
    site_map.add_child(lines)
```

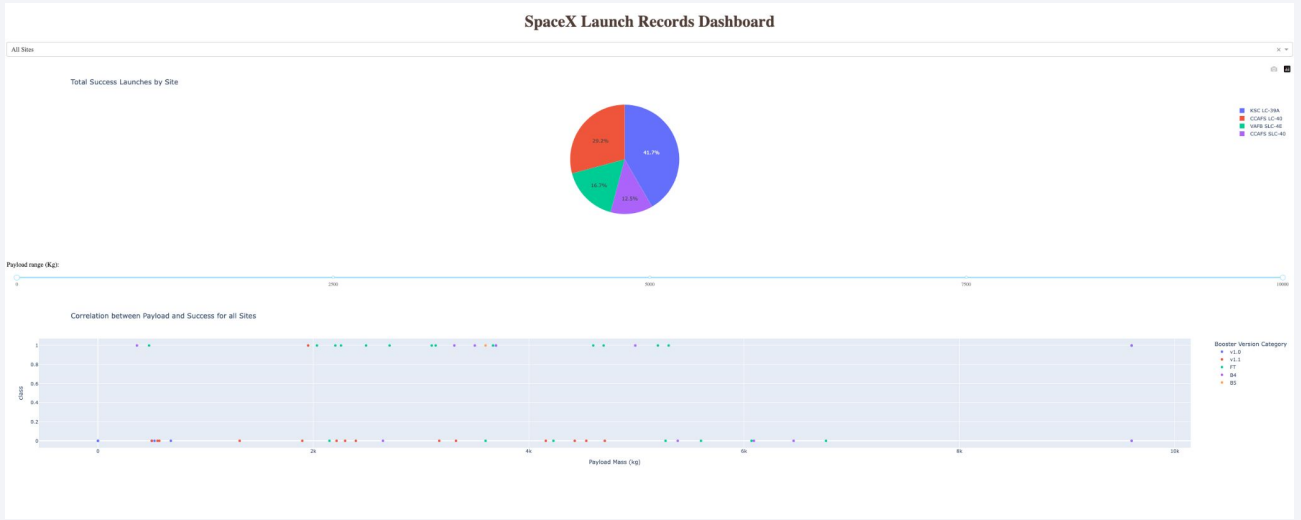
```python
# Add Mouse Position to get the coordinate (Lat, Long) for a mouse over on the map
formatter = "function(num) {return L.Util.formatNum(num, 5);};"
mouse_position = MousePosition(
    position='topright',
    separator=' Long: ',
    empty_string='NaN',
    lng_first=False,
    num_digits=5,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

site_map.add_child(mouse_position)

# Display the map
site_map
```

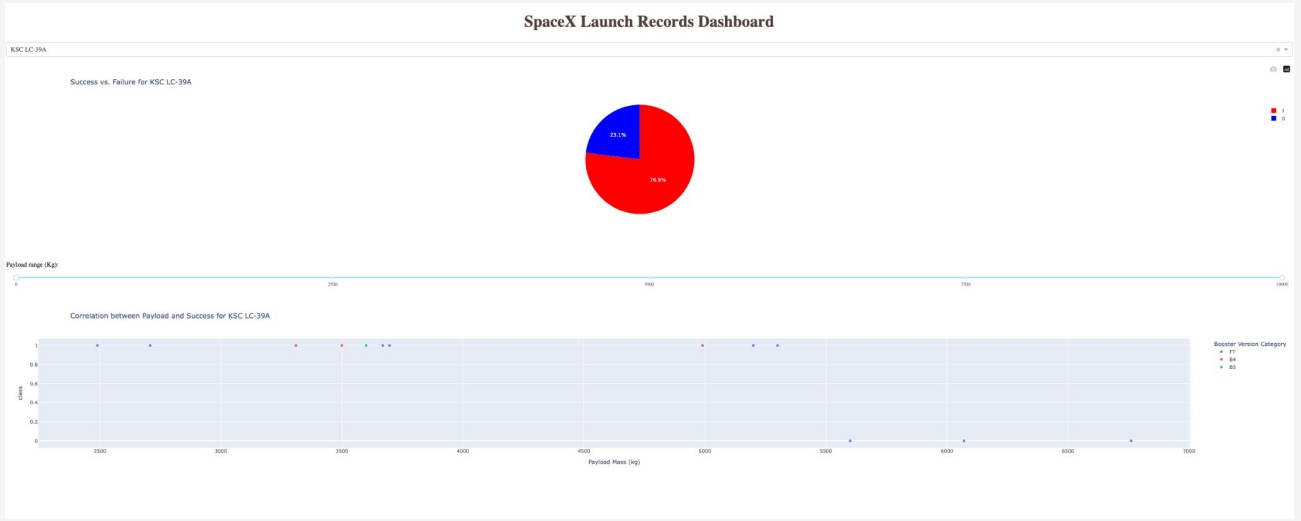
```
Distance between the launch site and the closest coastline point: 0.86 km
Distance between the launch site and Titusville, FL: 23.18 km
Distance between the launch site and the FEC Railway: 1.29 km
Distance between the launch site and the nearest highway: 6.95 km
```



Appendix - Dashboard

```python
import dash
from dash import dcc, html, Input, Output
import pandas as pd
import plotly.express as px

# Read the SpaceX launch data into pandas dataframe
spacex_df = pd.read_csv("spacex_launch_dash.csv")
max_payload = spacex_df['Payload Mass (kg)'].max()
min_payload = spacex_df['Payload Mass (kg)'].min()

# Create a dash application
app = dash.Dash(__name__)

# Create an app layout
app.layout = html.Div(children=[
    html.H1('SpaceX Launch Records Dashboard',
            style={'textAlign': 'center', 'color': '#503D36',
                   'font-size': 40}),
    dcc.Dropdown(id='site-dropdown',
                 options=[
                     {'label': 'All Sites', 'value': 'ALL'},
                     {'label': 'CCAFS LC-40', 'value': 'CCAFS LC-40'},
                     {'label': 'CCAFS SLC-40', 'value': 'CCAFS SLC-40'},
                     {'label': 'KSC LC-39A', 'value': 'KSC LC-39A'},
                     {'label': 'VAFB SLC-4E', 'value': 'VAFB SLC-4E'}
                 ],
                 value='ALL',
                 placeholder='Select a Launch Site here',
                 searchable=True),
    html.Br(),
    html.Div(dcc.Graph(id='success-pie-chart')),
    html.Br(),
    html.P("Payload range (Kg):"),
    dcc.RangeSlider(id='payload-slider',
                    min=0, max=10000, step=1000,
                    marks={0: '0', 2500: '2500', 5000: '5000', 7500: '7500', 10000: '10000'},
                    value=[min_payload, max_payload]),
    html.Br(),
    html.Div(dcc.Graph(id='success-payload-scatter-chart')),
])

@app.callback(
    Output(component_id='success-pie-chart', component_property='figure'),
    Input(component_id='site-dropdown', component_property='value')
)
def get_pie_chart(entered_site):
    color_map = {0: 'blue', 1: 'red'}  # Specify the color map for class
    if entered_site == 'ALL':
        success_counts = spacex_df[spacex_df['class'] == 1].groupby(['Launch Site']).size().reset_index(name='count')
        fig = px.pie(success_counts,
                     values='count',
                     names='Launch Site',
                     title='Total Success Launches by Site')
    else:
        filtered_df = spacex_df[spacex_df['Launch Site'] == entered_site]
        success_counts = filtered_df.groupby('class').size().reset_index(name='count')
        fig = px.pie(success_counts,
                     values='count',
                     names='class',
                     title=f'Success vs. Failure for {entered_site}',
                     color='class',
                     color_discrete_map=color_map)
    return fig

@app.callback(
    Output(component_id='success-payload-scatter-chart', component_property='figure'),
    [Input(component_id='site-dropdown', component_property='value'),
     Input(component_id='payload-slider', component_property='value')]
)
def update_scatter_chart(entered_site, payload_range):
    low, high = payload_range
    filtered_df = spacex_df[(spacex_df['Payload Mass (kg)'] >= low) & (spacex_df['Payload Mass (kg)'] <= high)]
    if entered_site == 'ALL':
        fig = px.scatter(filtered_df,
                         x='Payload Mass (kg)', y='class',
                         color='Booster Version Category',
                         title='Correlation between Payload and Success for all Sites')
    else:
        filtered_df = filtered_df[filtered_df['Launch Site'] == entered_site]
        fig = px.scatter(filtered_df,
                         x='Payload Mass (kg)', y='class',
                         color='Booster Version Category',
                         title=f'Correlation between Payload and Success for {entered_site}')
    return fig

if __name__ == '__main__':
    app.run_server(debug=True)
```





Appendix - Machine Learning Prediction

```
[ ] # Modified Grid Search
    import numpy as np
    from sklearn.model_selection import RandomizedSearchCV
    from sklearn.svm import SVC

    # Define a simplified parameter grid
    parameters = {'kernel': ('linear', 'rbf'),
                  'C': [0.1, 1, 10],
                  'gamma': [0.001, 0.01]}

    # Create the SVM object
    svm = SVC()

    # Create the RandomizedSearchCV object with cv = 10 and parallel computation
    svm_cv = RandomizedSearchCV(svm, parameters, n_iter=10, cv=10, n_jobs=-1)

    # Fit the RandomizedSearchCV object to find the best parameters
    svm_cv.fit(X_train, Y_train)

    # Print the best parameters and the accuracy
    print("Tuned hyperparameters: (best parameters) ", svm_cv.best_params_)
    print("Accuracy: ", svm_cv.best_score_)

    # Calculate the accuracy on the test data using the best estimator
    test_accuracy = svm_cv.best_estimator_.score(X_test, Y_test)
    print("Test Accuracy: ", test_accuracy)
```

```
[ ] # Initial Data Tree
    import numpy as np
    from sklearn.model_selection import GridSearchCV
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import StandardScaler
    import pandas as pd

    # Load your data from local files
    data = pd.read_csv("/content/dataset_part_2.csv")
    X = pd.read_csv("/content/dataset_part_3.csv")

    # Convert 'Class' column to a NumPy array
    Y = data['Class'].to_numpy()

    # Standardize the data
    scaler = StandardScaler()
    X = scaler.fit_transform(X)

    # Split the data into training and test sets
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)

    # Define the parameter grid
    parameters = {'criterion': ['gini', 'entropy'],
                  'splitter': ['best', 'random'],
                  'max_depth': [2*n for n in range(1,10)],
                  'max_features': ['auto', 'sqrt'],
                  'min_samples_leaf': [1, 2, 4],
                  'min_samples_split': [2, 5, 10]}

    # Create the Decision Tree Classifier object
    tree = DecisionTreeClassifier()
```

```
[ ] from sklearn.model_selection import GridSearchCV
    from sklearn.linear_model import LogisticRegression

    # Define parameter grid with different solvers and increased max_iter
    param_grid = {'C': [0.1, 1, 10, 100], 'solver': ['liblinear']}
    logreg = LogisticRegression(max_iter=2000)  # Increase max_iter to 2000

    # Initialize GridSearchCV
    logreg_cv = GridSearchCV(logreg, param_grid, cv=5)

    # Assuming X_train and Y_train are already defined and preprocessed
    logreg_cv.fit(X_train, Y_train)

    # Print the best parameters and the accuracy
    print("Tuned hyperparameters: (best parameters) ", logreg_cv.best_params_)
    print("Accuracy: ", logreg_cv.best_score_)
```

```
[ ] # Initial KNN
    import numpy as np
    from sklearn.model_selection import GridSearchCV, train_test_split
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.preprocessing import StandardScaler
    import pandas as pd

    # Load your data from local files
    data = pd.read_csv("/content/dataset_part_2.csv")
    X = pd.read_csv("/content/dataset_part_3.csv")

    # Convert 'Class' column to a NumPy array
    Y = data['Class'].to_numpy()

    # Standardize the data
    scaler = StandardScaler()
    X = scaler.fit_transform(X)

    # Split the data into training and test sets
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)

    # Define the parameter grid
    parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                  'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                  'p': [1, 2]}

    # Create the KNN Classifier object
    KNN = KNeighborsClassifier()
```

Thank you!