Interactive Visual Analytics
and Dashboard

IBM Developer
SKILLS NETWORK

# Winning Space Race with Data Science

Troy Altus
April 29, 2023

**Github Repository:**
**https://github.com/altustd/Coursera-IBM-Applied-Data-Science-Capstone**

# Outline

# Executive Summary

- Methodology and results presented herein for the IBM Data Science Professional Certificate capstone project
- Capstone project is the tenth and final course in the IBM Data Science Professional Certificate curriculum
- Project topic deals with the reusability of Space-X boosters

# Introduction

- The Space-X business model benefits from cost savings associated with the reuse of boosters after they are launched, recovered and serviced
    - First stage boosters have the capability to re-entering earth’s atmosphere and landing by firing engines in retrograde fashion
    - Boosters are landed on terrestrial landing pads or floating barges
    - Multiple boosters can be landed from a single flight, as demonstrated by Falcon 9 heavy
- The work carried out in this project demonstrates various Data Science techniques based on actual flight data

# Methodology

# Methodology Overview

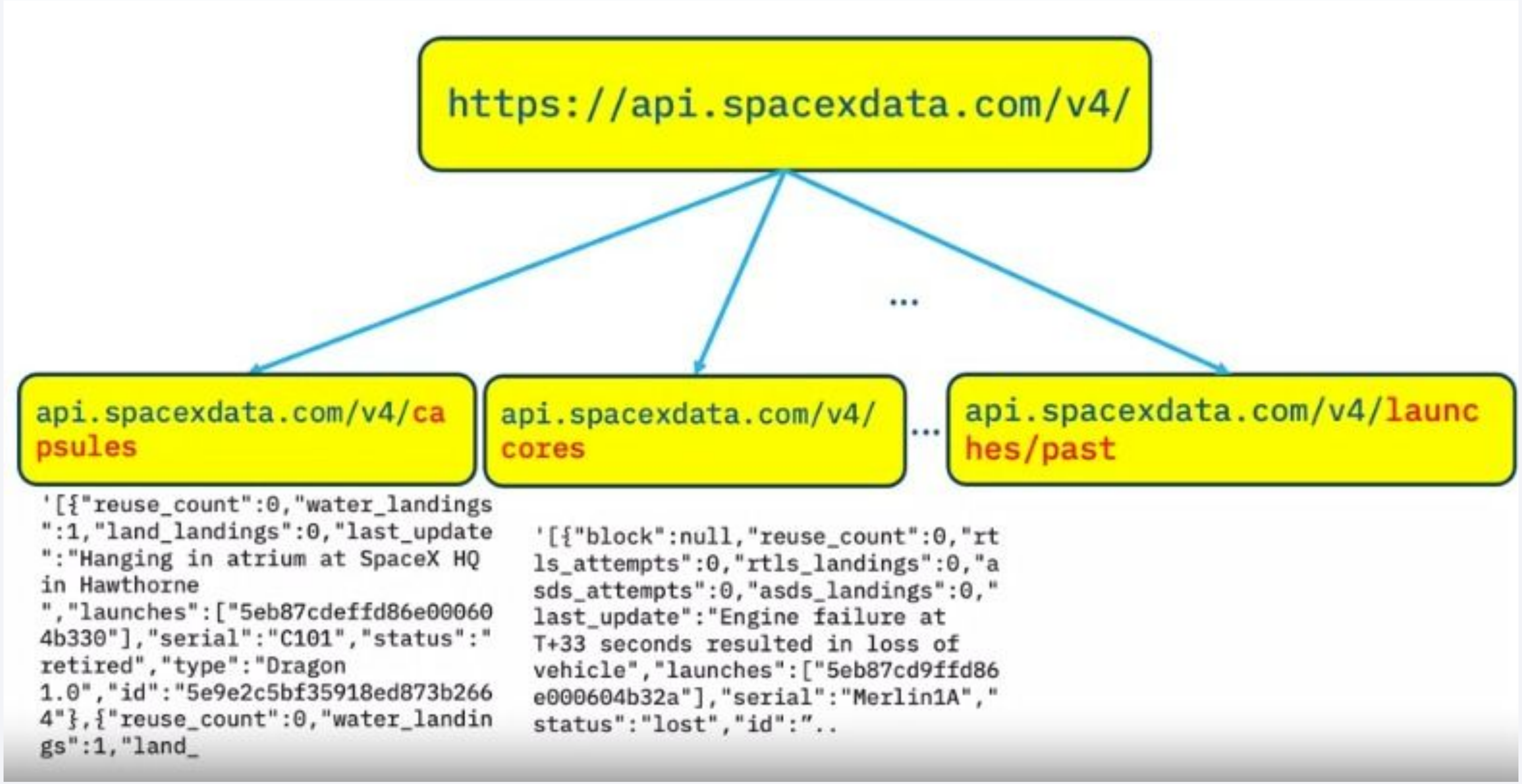
- Data collection
- Data wrangling
- Exploratory data analysis (EDA) using visualization and SQL
- Interactive visual analytics using Folium and Plotly Dash
- Predictive analysis using classification models

# Methodology: Data Collection

- Goal is to transform raw data into a clean dataset by wrangling data using an API, filtering and sampling the data, and dealing with nulls
- SpaceX REST API gives data about launches
  - Type of rocket used
  - Payload delivered
  - Launch specifications
  - Landing specification
  - Landing outcome
- Additional Falcon 9 launch data obtained via web scraping related Wiki pages
  - Python BeautifulSoup package used to web scrape HTML tables containing Falcon 9 launch records
  - Data is parsed and converted to a Pandas data frame for further visualization and analysis
- Original data set (Falcon 1 + Falcon 9) is filtered to only include Falcon 9 launches
- Null values are addressed using realistic assumptions
  - PayloadMass null values replaced by mean value
  - LandingPad null values are dealt with via one hot encoding

# Methodology: Data Collection – SpaceX API

## Top-level API Flowchart



# Methodology: Data Collection – SpaceX API

## API endpoint call

- DefinesSpecific url pointing to historical launch data
- Perform “get” request from the requests library
- Response is in the form of a list of .json objects

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

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

Check the content of the response

```
[29]: # Uncomment for result.

print(response.content)
```


b'[{"fairings":{"reused":false,"recovery_attempt":false,"recovered":false,"ships":[]},"links":{"patch":{"small":"https://images2.imgbox.com/94/f2/NN6Ph45r_o.png","large":"https://images2.imgbox.com/5b/02/QcxHUb5V_o.png"},"reddit":{"campaign":null,"launch":null,"media":null,"recovery":null},"flickr":{"small":[],"original":[]},"presskit":null,"webcast":"https://www.youtube.com/watch?v=0a_00nJ_Y88","youtube_id":"0a_00nJ_Y88","article":"https://www.space.com/2196-spacex-inaugural-falcon-1-rocket-lost-launch.html","wikipedia":"https://en.wikipedia.org/wiki/DemoSat"},"static_fire_date_utc":"2006-03-17T00:00:00.000Z","static_fire_date_unix":1142553600,"net":false,"window":0,"rocket":"5e9d0d95eda69955f709d1eb","success":false,"failures":[{"time":33,"altitude":null,"reason":"merlin engine failure"}],"details":"Engine failure at 33 seconds and loss of vehicle","crew":[],"ships":[],"capsules":[],"payloads":["5eb0e4b5b6c3bb0006eeb1e1"],"launchpad":"5e9e4502f5090995de566f86","flight_number":1,"name":"FalconSat","date_utc":"2006-03-24T22:30:00.000Z","date_unix":1143239400,"date_local":"2006-03-25T10:30:00+12:00","date_precision":"hour","upcoming":false,"cores":[{"core":"5e9e289df35918033d3b2623","flight":1,"gridfins":false,"legs":false,"reused":false,"landing_attempt":false,"landing_success":null,"landing_type":null,"landpad":null}],"auto_update":true,"tbd":false,"launch_library_id":null,"id":"5eb87cd9ffd86e000604b32a"},{"fairings":{"reused":false,"recovery_attempt":false,"recovered":false,"ships":[]},"links":{"patch":{"small":"https://images2.imgbox.com/f9/4a/ZboXReNb_o.png","large":"https://images2.imgbox.com/80/a2/bkWotCIS_o.png"},"reddit":{"campaign":null,"launch":null,"media":null,"recovery":null},"flickr":{"small":[],"original":[]},"presskit":null,"webcast":"https://www.youtube.com/watch?v=Lk4zQ2wP-Nc","youtube_id":"Lk4zQ2wP-Nc","article":"https://www.space.com/3590-spacex-falcon-1-rocket-fails-reach-orbit.html","wikipedia":"https://en.wikipedia.org/wiki/DemoSat"},"static_fire_date_utc":null,"static_fire_date_unix":null,"net":false,"window":0,"rocket":"5e9d0d95eda69955f709d1eb","success":fal


Jupyter Notebook can be found here:

# Methodology: Data Collection – SpaceX API

## API Data Wrangling

- Individual .json objects from API call are converted to a DataFrame using “json_normalize” function
- This allows us to convert to a flat table

[11]:
```python
# Use json_normalize meethod to convert the json result into a dataframe
respjson = response.json()
data = pd.json_normalize(respjson)
```

Using the dataframe `data` print the first 5 rows

[12]:
```python
# Get the head of the dataframe
data.head(5)
```

[12]:

| | static_fire_date_utc | static_fire_date_unix | net | window | rocket | success | failures | details | crew | ships | capsules | pay |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2006-03-17T00:00:00.000Z | 1.142554e+09 | False | 0.0 | 5e9d0d95eda69955f709d1eb | False | [{'time': 33, 'altitude': None, 'reason': 'merlin engine failure'}] | Engine failure at 33 seconds and loss of vehicle | [] | [] | [] | [5eb0e4b5b6c3bb0006ee |
| **1** | None | NaN | False | 0.0 | 5e9d0d95eda69955f709d1eb | False | [{'time': 301, 'altitude': 289, 'reason': 'harmonic oscillation leading to premature engine | Successful first stage burn and transition to second stage, maximum altitude 289 km, Premature engine shutdown at T+7 min 30 s, Failed | [] | [] | [] | [5eb0e4b6b6c3bb0006ee |

# Methodology: Data Collection - Web Scraping

## Web Scraping Related Wiki Pages

- Python BeautifulSoup library
- Tabular data is parsed and converted to a Pandas DataFrame


| | FlightNumber | Date | BoosterVersion | PayloadMass | Orbit | LaunchSite | Outcome | Flights | GridFins | Reused | Legs | LandingPad | Block | ReusedCount | Serial | Longitude | Latitude |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2006-03-24 | Falcon 1 | 20.0 | LEO | Kwajalein Atoll | None None | 1 | False | False | False | None | NaN | 0 | Merlin1A | 167.743129 | 9.047721 |
| 1 | 2 | 2007-03-21 | Falcon 1 | NaN | LEO | Kwajalein Atoll | None None | 1 | False | False | False | None | NaN | 0 | Merlin2A | 167.743129 | 9.047721 |
| 2 | 4 | 2008-09-28 | Falcon 1 | 165.0 | LEO | Kwajalein Atoll | None None | 1 | False | False | False | None | NaN | 0 | Merlin2C | 167.743129 | 9.047721 |
| 3 | 5 | 2009-07-13 | Falcon 1 | 200.0 | LEO | Kwajalein Atoll | None None | 1 | False | False | False | None | NaN | 0 | Merlin3C | 167.743129 | 9.047721 |
| 4 | 6 | 2010-06-04 | Falcon 9 | NaN | LEO | CCAFS SLC 40 | None None | 1 | False | False | False | None | 1.0 | 0 | B0003 | -80.577366 | 28.561857 |

# Methodology: Data Wrangling

API is used a second time to target a different endpoint

- Call to the first API endpoint provides ID number, not actual data
- Pre-defined functions provided to create the data set
- Data is stored in lists




**Jupyter Notebook can be found here:**

# Methodology: Sampling/Filtering

Data includes Falcon 1 booster, but we are only concerned with Falcon 9 for this study

- Filter the data dataframe using the BoosterVersion column to only keep the Falcon 9 launches.
- Save the filtered data to a new dataframe called data_falcon9.

| | FlightNumber | Date | BoosterVersion | PayloadMass | Orbit | LaunchSite | Outcome | Flights | GridFins | Reused | Legs | LandingPad | Block | ReusedCount | Serial | Longitude | Latitude |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2006-03-24 | Falcon 1 | 20.0 | LEO | Kwajalein Atoll | None None | 1 | False | False | False | None | NaN | 0 | Merlin1A | 167.743129 | 9.047721 |
| 1 | 2 | 2007-03-21 | Falcon 1 | NaN | LEO | Kwajalein Atoll | None None | 1 | False | False | False | None | NaN | 0 | Merlin2A | 167.743129 | 9.047721 |
| 2 | 4 | 2008-09-28 | Falcon 1 | 165.0 | LEO | Kwajalein Atoll | None None | 1 | False | False | False | None | NaN | 0 | Merlin2C | 167.743129 | 9.047721 |
| 3 | 5 | 2009-07-13 | Falcon 1 | 200.0 | LEO | Kwajalein Atoll | None None | 1 | False | False | False | None | NaN | 0 | Merlin3C | 167.743129 | 9.047721 |
| 4 | 6 | 2010-06-04 | Falcon 9 | NaN | LEO | CCAFS SLC 40 | None None | 1 | False | False | False | None | 1.0 | 0 | B0003 | -80.577366 | 28.561857 |

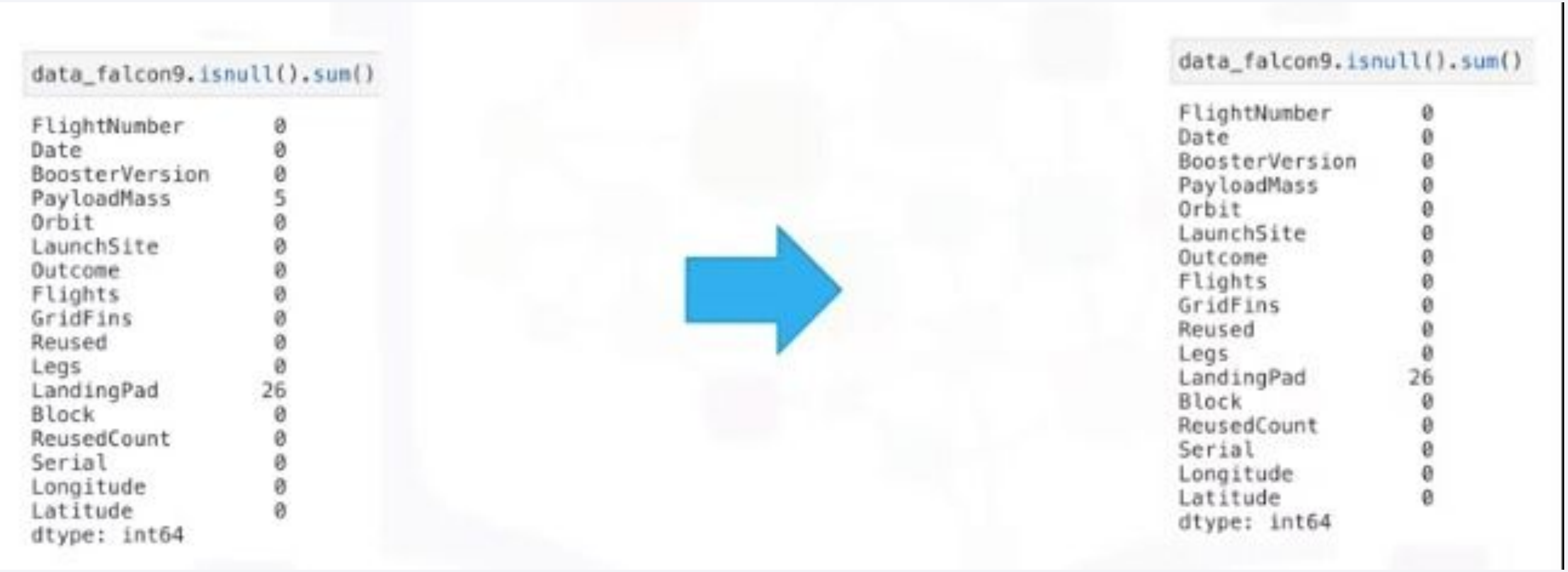
```
[25]: data_falcon9.head()
```

[25]:

| | FlightNumber | Date | BoosterVersion | PayloadMass | Orbit | LaunchSite | Outcome | Flights | GridFins | Reused | Legs | LandingPad | Block | ReusedCount | Serial | Long |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 6 | 2010-06-04 | Falcon 9 | NaN | LEO | CCAFS SLC 40 | None None | 1 | False | False | False | None | 1.0 | 0 | B0003 | -80.5 |
| 5 | 8 | 2012-05-22 | Falcon 9 | 525.0 | LEO | CCAFS SLC 40 | None None | 1 | False | False | False | None | 1.0 | 0 | B0005 | -80.5 |
| 6 | 10 | 2013-03-01 | Falcon 9 | 677.0 | ISS | CCAFS SLC 40 | None None | 1 | False | False | False | None | 1.0 | 0 | B0007 | -80.5 |
| 7 | 11 | 2013-09-29 | Falcon 9 | 500.0 | PO | VAFB SLC 4E | False Ocean | 1 | False | False | False | None | 1.0 | 0 | B1003 | -120.6 |
| 8 | 12 | 2013-12-03 | Falcon 9 | 3170.0 | GTO | CCAFS SLC 40 | None None | 1 | False | False | False | None | 1.0 | 0 | B1004 | -80.5 |

# Methodology: Sampling/Filtering
## Addressing Null Values

- Replace null values for payload mass with mean values
- Leave the column “Landing Pad” with null values as it is represented when a landing pad is not used.  This will be dealt with when using One Hot encoding later on.



# Methodology: Exploratory Data Analysis

Exploratory data analysis (EDA) is used by data scientists to analyze and investigate data sets and summarize their main characteristics, often employing data visualization methods. It helps determine how best to manipulate data sources to get the answers you need, making it easier for data scientists to discover patterns, spot anomalies, test a hypothesis, or check assumptions.

EDA is primarily used to see what data can reveal beyond the formal modeling or hypothesis testing task and provides a provides a better understanding of data set variables and the relationships between them. It can also help determine if the statistical techniques you are considering for data analysis are appropriate. Originally developed by American mathematician John Tukey in the 1970s, EDA techniques continue to be a widely used method in the data discovery process today.

REF: https://www.ibm.com/topics/exploratory-data-analysis

# Methodology: Exploratory Data Analysis

For this Capstone project, Exploratory Data Analysis was accomplished through completion of two labs:

1. Exploratory Data Analysis  using SQL
    a. IBM Skills Network Labs version used
    b. Download datasets and store in table
    c. Connect to the database
    d. Write and execute SQL queries to solve the assignment tasks.

2. Exploratory Data Analysis with visualization and Feature Engineering  using Pandas and Matplotlib
    a. Visualize the relationship between different parameters
    b. Visualize the launch success yearly trend
    c. Create dummy variables to categorical columns

# Methodology: EDA with SQL

A dataset of historical launches was used to determine first stage landing success.
Data included are as follows:

| | |
|---|---|
| Date | Payload mass |
| Time (UTC) | Orbit |
| Booster version | Customer |
| Launch site | Mission outcome |
| Payload | Landing outcome |

The general Exploratory Data Analysis steps were:

- Understand the Spacex DataSet
- Load the dataset into the corresponding table in a Db2 database
- Execute SQL queries to answer assignment questions

# Methodology: EDA with SQL

## SQL Queries

Per lab instructions, the following SQL queries were carried out:

Task 1. Display the names of the unique launch sites in the space mission
Task 2. Display 5 records where launch sites begin with the string 'CCA'
Task 3. Display the total payload mass carried by boosters launched by NASA (CRS)
Task 4. Display average payload mass carried by booster version F9 v1.1
Task 5. List the date when the first succesful landing outcome in ground pad was acheived
Task 6. List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000
Task 7. List the total number of successful and failure mission outcomes
Task 8. List the names of the booster_versions which have carried the maximum payload mass. Use a subquery
Task 9. List the records which will display the month names, failure landing_outcomes in drone ship ,booster versions, launch_site for the months in year 2015



**Jupyter Notebook can be found here:**

# Methodology:  EDA with Visualization and Feature Engineering

Historical SpaceX data was used to identify which launch parameters were associated with successful first stage landings.  Data for these parameters were isolated and prepared for use in a predictive model (future step in Capstone project)

Pandas and Seaborn were used to graphically explore the SpaceX launch data.

- Seaborn is a data visualization tool built on Matplotlib.
- One-hot encoding was used to convert categorical variables into dummy numerical variables for easy use in a machine learning algorithm.

Observations were as follows:The more massive the payload, the less likely the first stage will return

- Success rate increased as more flights were attempted
- No launches of payload mass exceeding 10,000 kg were attempted at Vandenburg Air Force Base (VAFB-SLC).
- The LEO success rate appears related to the number of flights; on the other hand, there seems to be no relationship between flight number when in GTO orbit.
- With heavy payloads, positive landing rates are higher for polar, LEO and ISS.  This was not distinguishable for GTO launches.

# Methodology:  EDA with Visualization and Feature Engineering

Per lab instructions, the Pandas and Matplotlib were used to carry out the following tasks:

Task 1: Visualize the relationship between Flight Number and Launch Site
Task 2: Visualize the relationship between Payload and Launch Site
Task 3: Visualize the relationship between success rate of each orbit type
Task 4: Visualize the relationship between FlightNumber and Orbit type
Task 5: Visualize the relationship between Payload and Orbit type
Task 6: Visualize the launch success yearly trend
Task 7: Create dummy variables to categorical columns
Task 8: Cast all numeric columns to float64

**Jupyter Notebook can be found here:**

# Methodology: Interactive Maps with Folium

Folium was used for interactive plotting:

- Plot of east coast and west coast US launch sites and Johnson Space Center in Houston
- Launch sites with launch results (successes and failures)
- Launch site distance to points of interest: major roads, railways, coastline and closest city

From the plots, the following observations can be made:

1. Launch sites are close in proximity to essential infrastructure such as railways and major surface roads
2. Launch sites are reasonably far away from populated areas
3. Launch sites are near the coastline so flight path is over ocean

**Jupyter Notebook can be found here:**

# Methodology: Dashboarding with Plotly Dash

## Plotly – An Overview

- Interactive, open-source plotting library
- Supports over 40 unique chart types
- Includes chart types like statistical, financial, maps, scientific, and 3-dimensional
- Visualizations can be displayed in Jupyter notebook, saved to HTML files, or can be used in developing Python-built web applications

# Methodology: Dashboarding with Plotly Dash

## Plotly Sub-modules

- Plotly Graph Objects: Low-level interface to figures, traces, and layout

  `plotly.graph_objects.Figure`

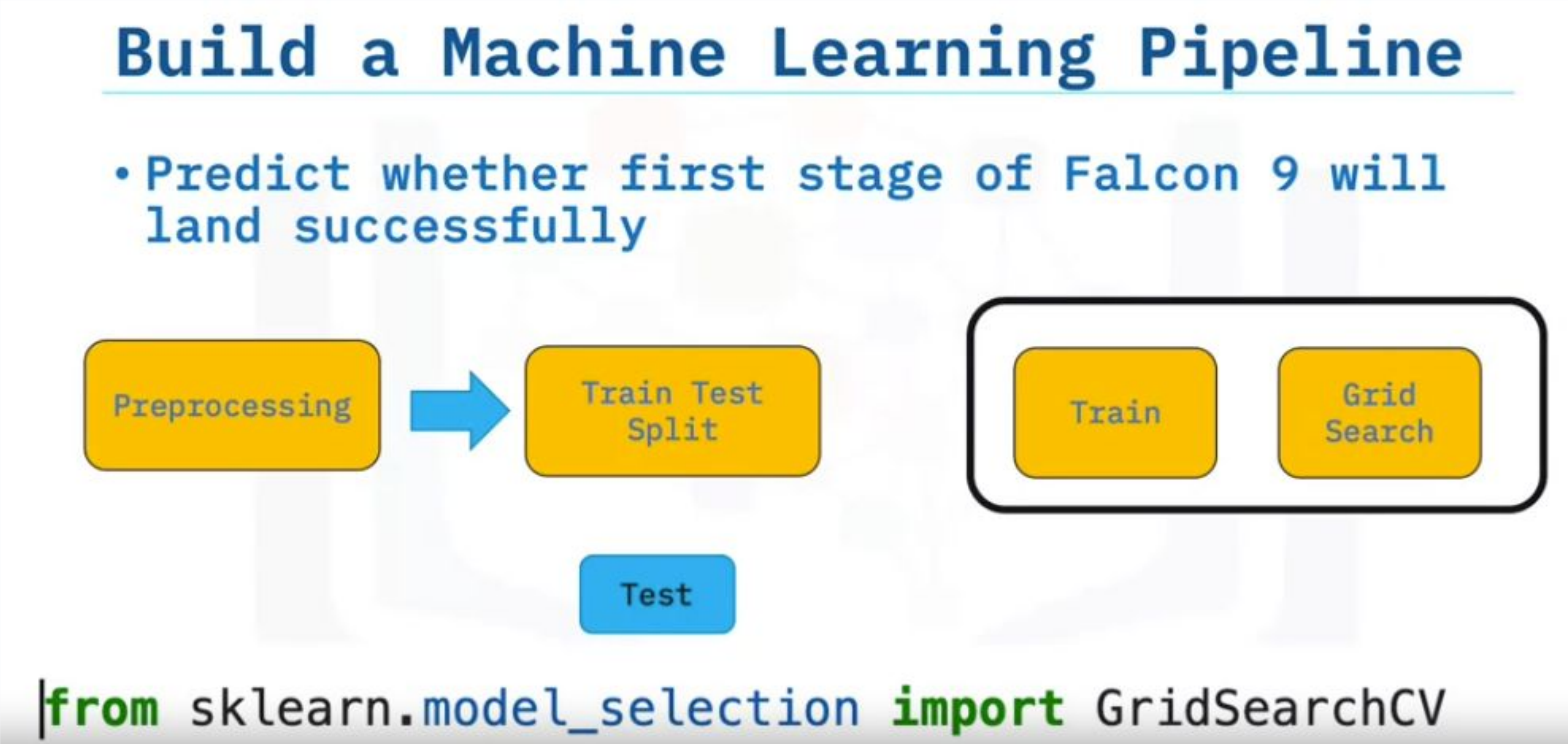
- Plotly Express: High-level wrapper

# Methodology: Dashboarding with Plotly Dash

## Dash – An Overview

- Open-source User Interface Python library from Plotly
- Easy to build GUI
- Declarative and Reactive
- Rendered in web browser and can be deployed to servers
- Inherently cross-platform and mobile ready

# Methodology: Predictive Analysis



# Results

# Results: Exploratory Data Analysis
## SQL Queries

### Task 1

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

```
[9]: %sql SELECT DISTINCT(launch_site) FROM spacextbl;
```

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

[9]:

| launch_site |
|---|
| CCAFS LC-40 |
| VAFB SLC-4E |
| KSC LC-39A |
| CCAFS SLC-40 |

### Task 2

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

```
[23]: %sql SELECT * FROM spacextbl WHERE launch_site LIKE 'CCA%' LIMIT 5;
```

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

[23]:

| id | date | time_utc | booster_version | launch_site | payload | payload_mass_kg | orbit | customer | mission_outcome | landing_outcome |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 04-06-2010 | 18:45:00 | F9 v1.0 B0003 | CCAFS LC-40 | Dragon Spacecraft Qualification Unit | 0 | LEO | SpaceX | Success | Failure (parachute) |
| 1 | 08-12-2010 | 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) |
| 2 | 22-05-2012 | 07:44:00 | F9 v1.0 B0005 | CCAFS LC-40 | Dragon demo flight C2 | 525 | LEO (ISS) | NASA (COTS) | Success | No attempt |
| 3 | 08-10-2012 | 00:35:00 | F9 v1.0 B0006 | CCAFS LC-40 | SpaceX CRS-1 | 500 | LEO (ISS) | NASA (CRS) | Success | No attempt |
| 4 | 01-03-2013 | 15:10:00 | F9 v1.0 B0007 | CCAFS LC-40 | SpaceX CRS-2 | 677 | LEO (ISS) | NASA (CRS) | Success | No attempt |

# Results: Exploratory Data Analysis
## SQL Queries

### Task 3

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

[24]:
```
%sql SELECT SUM(payload_mass_kg) AS total_pl_mass_all_NASACRS FROM spacextbl WHERE customer = 'NASA (CRS)';
```

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

[24]:

| total_pl_mass_all_NASACRS |
|---|
| 45596 |

### Task 4

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

[26]:
```
%sql SELECT AVG(payload_mass_kg) FROM spacextbl WHERE booster_version LIKE 'F9 v1.1%'
```

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

[26]:

| AVG(payload_mass_kg) |
|---|
| 2534.6666666666665 |

# Results: Exploratory Data Analysis
## SQL Queries

### Task 5

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

*Hint:Use min function*

13]:
```
%sql SELECT MIN(id), date, landing_outcome FROM spacextbl WHERE landing_outcome = "Success (ground pad)"
```

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

13]:

| MIN(id) | date | landing_outcome |
|---|---|---|
| 19 | 22-12-2015 | Success (ground pad) |

### Task 6

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

[14]:
```
%%sql SELECT booster_version FROM spacextbl
WHERE landing_outcome = 'Success (drone ship)' AND payload_mass_kg > 4000 AND payload_mass_kg < 6000;
```

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

[14]:

| booster_version |
|---|
| F9 FT B1022 |
| F9 FT B1026 |
| F9 FT B1021.2 |
| F9 FT B1031.2 |

# Results: Exploratory Data Analysis
## SQL Queries

### Task 7

List the total number of successful and failure mission outcomes

[15]:
```
%%sql
SELECT DISTINCT(mission_outcome), COUNT(mission_outcome) AS counts
FROM spacextbl
GROUP BY mission_outcome;
```

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

[15]:

| mission_outcome | counts |
|---|---|
| Failure (in flight) | 1 |
| Success | 98 |
| Success | 1 |
| Success (payload status unclear) | 1 |

Clean up results using CASE

[16]:
```
%%sql
SELECT
    SUM(CASE WHEN mission_outcome LIKE 'Success%' THEN 1 else 0 END) AS overall_mission_success,
    SUM(CASE WHEN mission_outcome LIKE 'Failure%' THEN 1 else 0 END) AS overall__mission_failure,
    COUNT(*) AS total
FROM spacextbl;
```

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

[16]:

| overall_mission_success | overall__mission_failure | total |
|---|---|---|
| 100 | 1 | 101 |

# Results: Exploratory Data Analysis
## SQL Queries

### Task 8

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

[17]:
```
%%sql SELECT DISTINCT(booster_version), payload_mass_kg
FROM spacextbl
WHERE payload_mass_kg = (SELECT MAX(payload_mass_kg)
FROM spacextbl)
```

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

[17]:

| booster_version | payload_mass_kg |
|---|---|
| F9 B5 B1048.4 | 15600 |
| F9 B5 B1049.4 | 15600 |
| F9 B5 B1051.3 | 15600 |
| F9 B5 B1056.4 | 15600 |
| F9 B5 B1048.5 | 15600 |
| F9 B5 B1051.4 | 15600 |
| F9 B5 B1049.5 | 15600 |
| F9 B5 B1060.2 | 15600 |
| F9 B5 B1058.3 | 15600 |
| F9 B5 B1051.6 | 15600 |
| F9 B5 B1060.3 | 15600 |
| F9 B5 B1049.7 | 15600 |

# Results: Exploratory Data Analysis
## SQL Queries

### Task 9

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

**Note: SQLLite does not support monthnames. So you need to use substr(Date, 4, 2) as month to get the months and substr(Date,7,4)='2015' for year.**

[18]:
```
%%sql SELECT COUNT(mission_outcome) AS failed_mission
    FROM spacextbl
    WHERE mission_outcome LIKE 'Failure%' ;
```

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

[18]:

| failed_mission |
|---|
| 1 |

[19]:
```
%%sql SELECT id, date, landing_outcome,booster_version, launch_site
FROM spacextbl
WHERE landing_outcome = "Failure (drone ship)" AND date LIKE "%2015"
ORDER BY id
```

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

[19]:

| id | date | landing_outcome | booster_version | launch_site |
|---|---|---|---|---|
| 13 | 10-01-2015 | Failure (drone ship) | F9 v1.1 B1012 | CCAFS LC-40 |
| 16 | 14-04-2015 | Failure (drone ship) | F9 v1.1 B1015 | CCAFS LC-40 |

# Results: Exploratory Data Analysis

## SQL Queries

### Task 10

Rank the count of successful landing_outcomes between the date 04-06-2010 and 20-03-2017 in descending order.

[20]:
```
%%sql SELECT id, date, landing_outcome
    FROM spacextbl
    WHERE landing_outcome LIKE "Success%"
    ORDER BY id DESC
```

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

[20]:

| id | date | landing_outcome |
|---|---|---|
| 100 | 06-12-2020 | Success |
| 99 | 25-11-2020 | Success |
| 98 | 21-11-2020 | Success |
| 97 | 16-11-2020 | Success |
| 96 | 05-11-2020 | Success |
| 95 | 24-10-2020 | Success |
| 94 | 18-10-2020 | Success |
| 93 | 06-10-2020 | Success |
| 92 | 03-09-2020 | Success |
| 91 | 30-08-2020 | Success |
| 90 | 18-08-2020 | Success |
| 89 | 07-08-2020 | Success |
| 88 | 20-07-2020 | Success |
| 87 | 30-06-2020 | Success |
| 86 | 13-06-2020 | Success |
| 85 | 04-06-2020 | Success |
| 84 | 30-05-2020 | Success |
| 83 | 22-04-2020 | Success |
| 81 | 07-03-2020 | Success |
| 79 | 29-01-2020 | Success |
| 77 | 07-01-2020 | Success |
| 76 | 17-12-2019 | Success |
| 75 | 05-12-2019 | Success |
| 74 | 11-11-2019 | Success |
| 72 | 25-07-2019 | Success |
| 71 | 12-06-2019 | Success |
| 70 | 24-05-2019 | Success |
| 69 | 04-05-2019 | Success |
| 68 | 02-03-2019 | Success |
| 67 | 22-02-2019 | Success |
| 66 | 11-01-2019 | Success |
| 63 | 03-12-2018 | Success |
| 62 | 15-11-2018 | Success |
| 61 | 08-10-2018 | Success |
| 60 | 10-09-2018 | Success |
| 59 | 07-08-2018 | Success |
| 58 | 25-07-2018 | Success |
| 57 | 22-07-2018 | Success |
| 53 | 11-05-2018 | Success (drone ship) |
| 52 | 18-04-2018 | Success (drone ship) |
| 46 | 08-01-2018 | Success (ground pad) |
| 44 | 15-12-2017 | Success (ground pad) |
| 43 | 30-10-2017 | Success (drone ship) |
| 42 | 11-10-2017 | Success (drone ship) |
| 41 | 09-10-2017 | Success (drone ship) |
| 40 | 07-09-2017 | Success (ground pad) |
| 39 | 24-08-2017 | Success (drone ship) |
| 38 | 14-08-2017 | Success (ground pad) |
| 36 | 25-06-2017 | Success (drone ship) |
| 35 | 23-06-2017 | Success (drone ship) |
| 34 | 03-06-2017 | Success (ground pad) |
| 32 | 01-05-2017 | Success (ground pad) |
| 31 | 30-03-2017 | Success (drone ship) |
| 29 | 19-02-2017 | Success (ground pad) |
| 28 | 14-01-2017 | Success (drone ship) |
| 27 | 14-08-2016 | Success (drone ship) |
| 26 | 18-07-2016 | Success (ground pad) |
| 24 | 27-05-2016 | Success (drone ship) |
| 23 | 06-05-2016 | Success (drone ship) |
| 22 | 08-04-2016 | Success (drone ship) |
| 19 | 22-12-2015 | Success (ground pad) |

# Results: Exploratory Data Analysis
## SQL Queries

```
[21]: %%sql SELECT date, payload, landing_outcome
        FROM spacextbl
        WHERE landing_outcome LIKE 'Success%' AND date > '04-06-2010' AND date < '20-03-2017';

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

[21]:

| date | payload | landing_outcome |
|---|---|---|
| 08-04-2016 | SpaceX CRS-8 | Success (drone ship) |
| 06-05-2016 | JCSAT-14 | Success (drone ship) |
| 18-07-2016 | SpaceX CRS-9 | Success (ground pad) |
| 14-08-2016 | JCSAT-16 | Success (drone ship) |
| 14-01-2017 | Iridium NEXT 1 | Success (drone ship) |
| 19-02-2017 | SpaceX CRS-10 | Success (ground pad) |
| 14-08-2017 | SpaceX CRS-12 | Success (ground pad) |
| 07-09-2017 | Boeing X-37B OTV-5 | Success (ground pad) |
| 09-10-2017 | Iridium NEXT 3 | Success (drone ship) |
| 11-10-2017 | SES-11 / EchoStar 105 | Success (drone ship) |
| 15-12-2017 | SpaceX CRS-13 | Success (ground pad) |
| 08-01-2018 | Zuma | Success (ground pad) |
| 18-04-2018 | Transiting Exoplanet Survey Satellite (TESS) | Success (drone ship) |
| 11-05-2018 | Bangabandhu-1 | Success (drone ship) |
| 07-08-2018 | Merah Putih | Success |
| 10-09-2018 | Telstar 18V / Apstar-5C | Success |
| 08-10-2018 | SAOCOM 1A | Success |
| 15-11-2018 | Es hail 2 | Success |
| 11-01-2019 | Iridium NEXT-8 | Success |
| 12-06-2019 | RADARSAT Constellation, SpaceX CRS-18 | Success |
| 11-11-2019 | Starlink 1 v1.0, SpaceX CRS-19 | Success |
| 05-12-2019 | SpaceX CRS-19, JCSat-18 / Kacific 1 | Success |
| 17-12-2019 | JCSat-18 / Kacific 1, Starlink 2 v1.0 | Success |
| 07-01-2020 | Starlink 2 v1.0, Crew Dragon in-flight abort test | Success |
| 07-03-2020 | SpaceX CRS-20, Starlink 5 v1.0 | Success |
| 04-06-2020 | Starlink 7 v1.0, Starlink 8 v1.0 | Success |
| 13-06-2020 | Starlink 8 v1.0, SkySats-16, -17, -18, GPS III-03 | Success |
| 07-08-2020 | Starlink 9 v1.0, SXRS-1, Starlink 10 v1.0 | Success |
| 18-08-2020 | Starlink 10 v1.0, SkySat-19, -20, -21, SAOCOM 1B | Success |
| 06-10-2020 | Starlink 12 v1.0, Starlink 13 v1.0 | Success |
| 18-10-2020 | Starlink 13 v1.0, Starlink 14 v1.0 | Success |
| 05-11-2020 | GPS III-04 , Crew-1 | Success |
| 16-11-2020 | Crew-1, Sentinel-6 Michael Freilich | Success |
| 06-12-2020 | SpaceX CRS-21 | Success |

# Results: Exploratory Data Analysis
## SQL Queries

```
22]: %%sql
SELECT landing_outcome, COUNT(landing_outcome) AS Total_Landing_Outcome
FROM spacextbl
WHERE landing_outcome = 'Failure (drone ship)' OR landing_outcome = 'Success (ground pad)'
AND date BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY landing_outcome
ORDER BY Total_Landing_Outcome DESC;

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

22]:

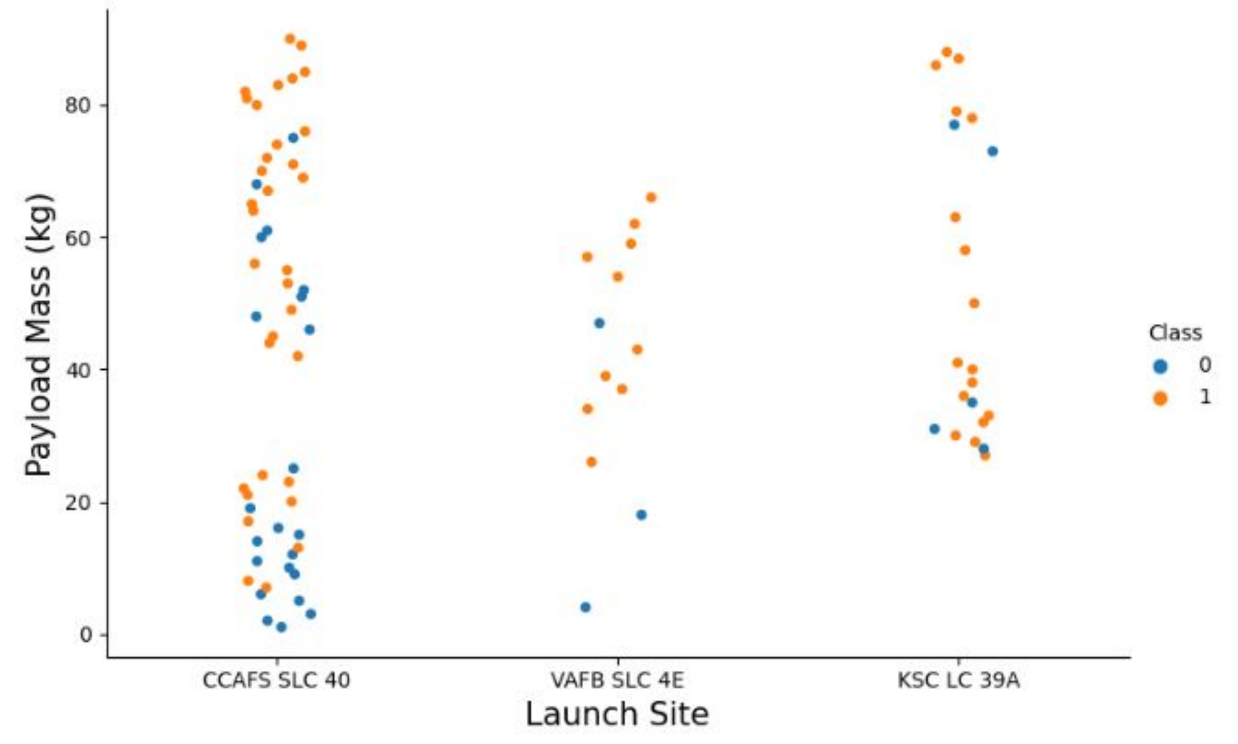
| landing_outcome | Total_Landing_Outcome |
|---|---|
| Failure (drone ship) | 5 |

# Results: Exploratory Data Analysis
## Visualization and Feature Engineering

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'`

[20]:
```
# 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
sns.catplot(y="FlightNumber",x="LaunchSite",hue='Class',data=df, aspect = 1.5)
plt.xlabel("Launch Site",fontsize=15)
plt.ylabel("Payload Mass (kg)",fontsize=15)
plt.show()
```

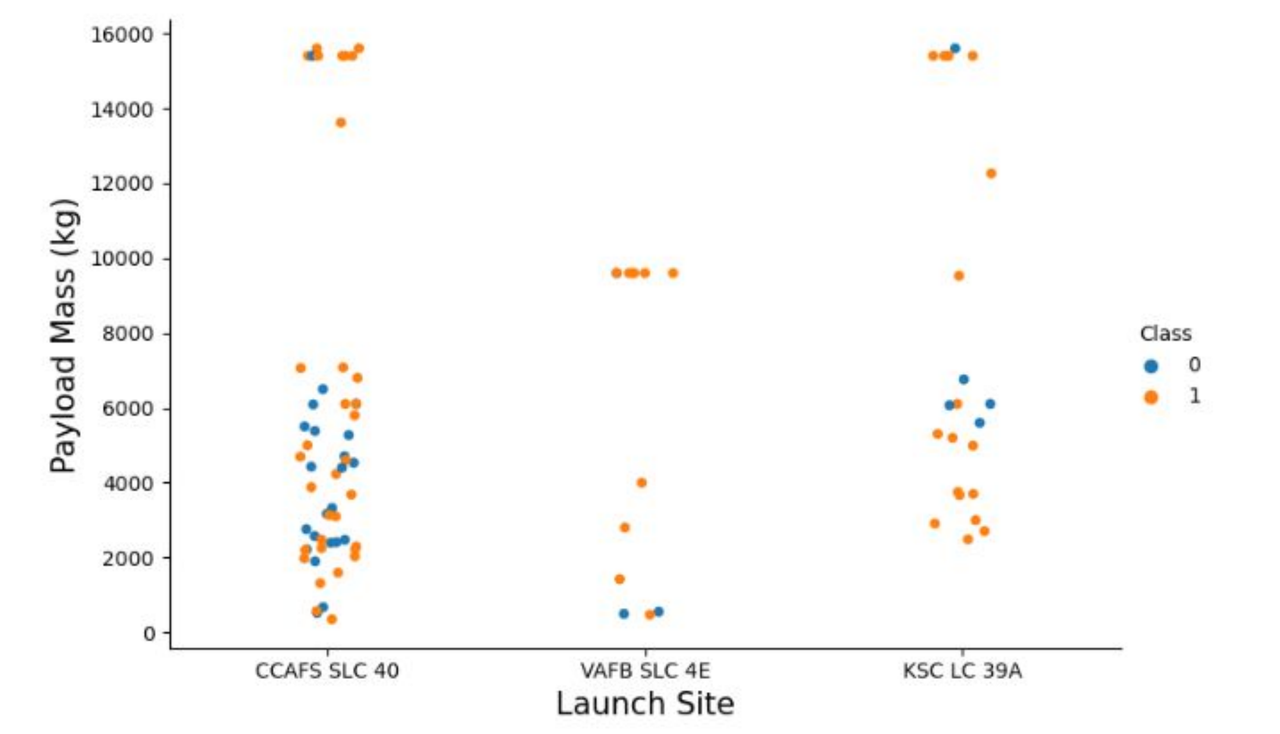


# Results: Exploratory Data Analysis
## Visualization and Feature Engineering

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

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

[7]:
```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch site, and hue to be the class value
sns.catplot(y="PayloadMass", x="LaunchSite", hue="Class", data=df, aspect = 2)
plt.xlabel("Launch Site",fontsize=15)
plt.ylabel("Payload Mass (kg)",fontsize=15)
plt.show()
```



# Results: Exploratory Data Analysis
## Visualization and Feature Engineering

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

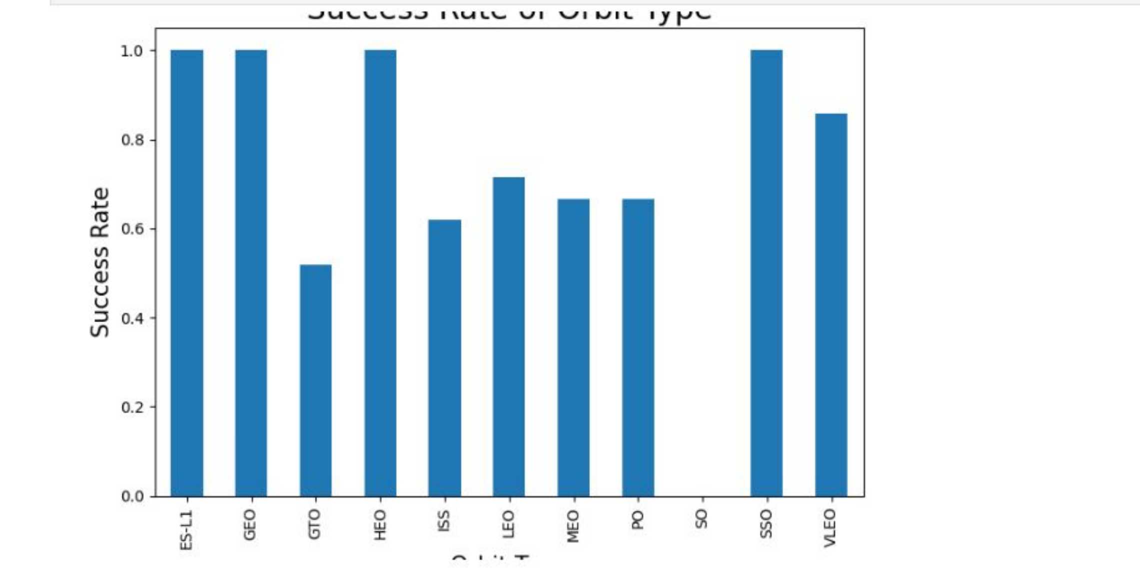
Next, we want to visually check if there are any relationship between success rate and orbit type.

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

[32]:
```
# HINT use groupby method on Orbit column and get the mean of Class column

df.groupby('Orbit').mean()['Class'].plot(kind='bar')
plt.title('Success Rate of Orbit Type', fontsize=20)
plt.xlabel('Orbit Type', fontsize=15)
plt.ylabel('Success Rate', fontsize=15)

plt.show()
```



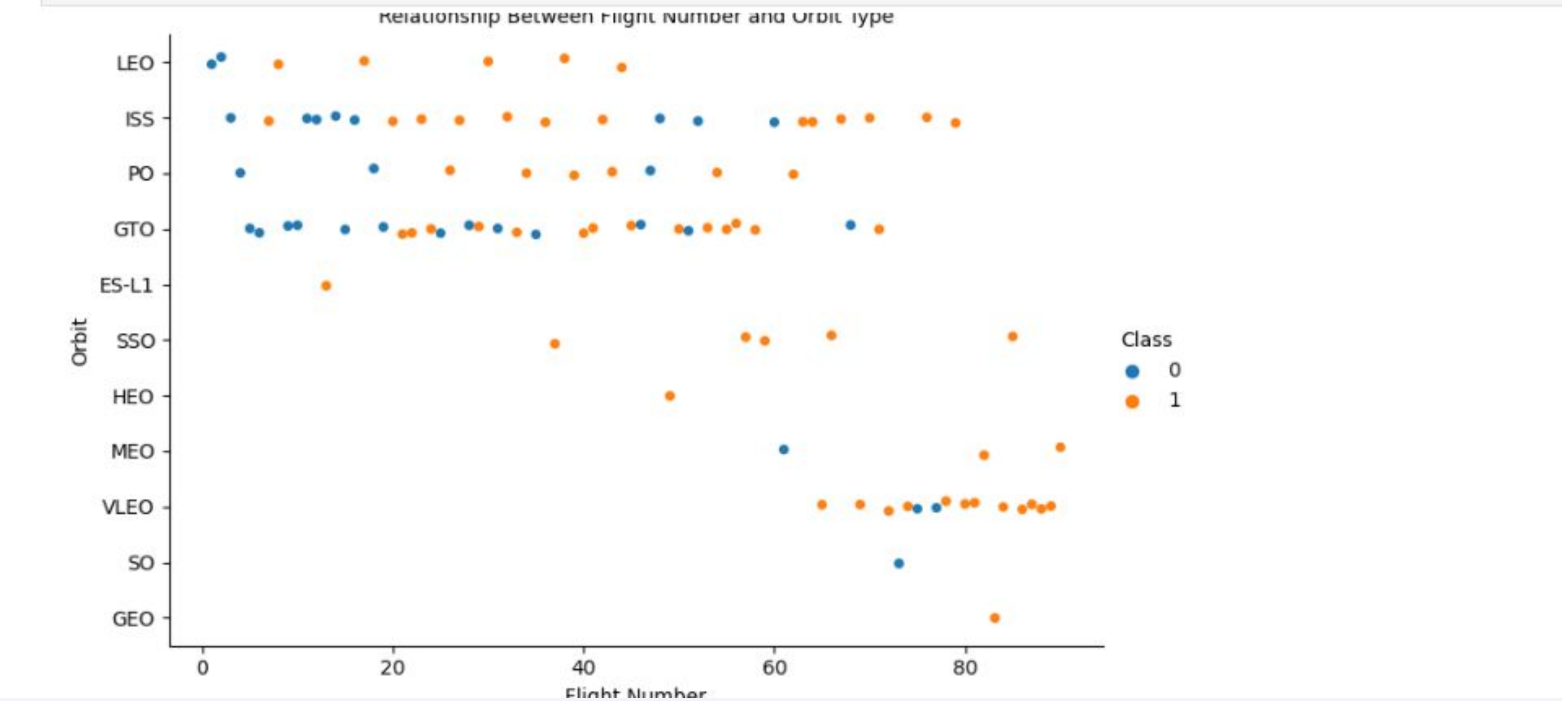
# Results: Exploratory Data Analysis
## Visualization and Feature Engineering

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.

[38]:
```
# 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

sns.catplot(x='FlightNumber', y='Orbit', data=df, hue='Class', aspect = 1.5)
plt.title('Relationship Between Flight Number and Orbit Type', fontsize=10)
plt.xlabel('Flight Number', fontsize=10)
plt.show()
```



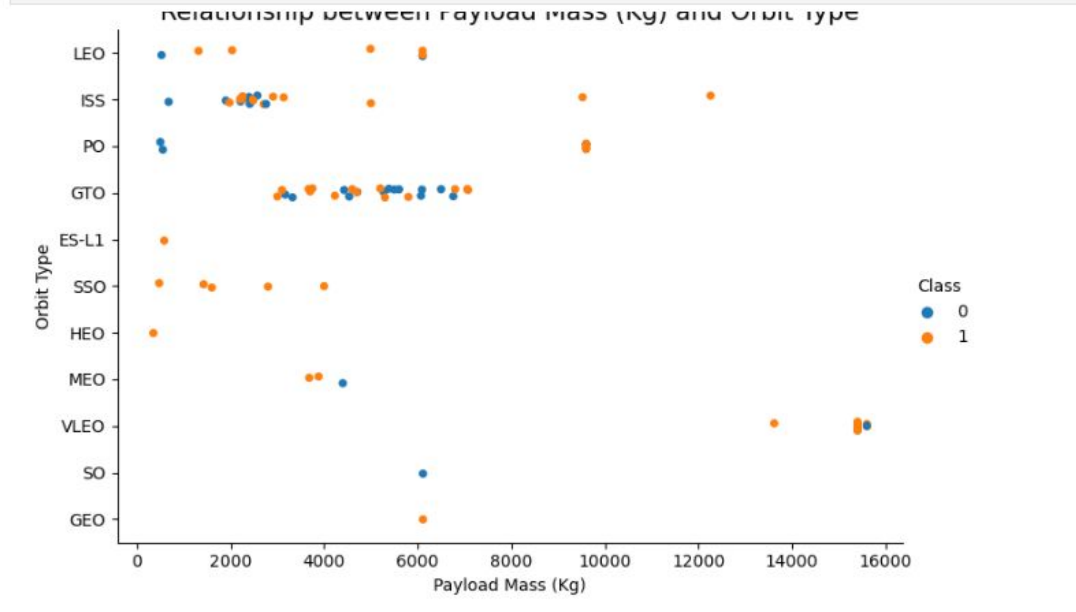
# Results: Exploratory Data Analysis
## Visualization and Feature Engineering

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

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

```
# Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, and hue to be the class value

sns.catplot(x='PayloadMass', y='Orbit', data=df, hue='Class', aspect = 5)
plt.title('Relationship between Payload Mass (Kg) and Orbit Type', fontsize=30)
plt.xlabel('Payload Mass (Kg)', fontsize=20)
plt.ylabel('Orbit Type', fontsize=20)
plt.show()
```



# Results: Exploratory Data Analysis
## Visualization and Feature Engineering

### TASK 6: Visualize the launch success yearly trend

You can plot a line chart with x axis to be `Year` and y axis to be average success rate, to get the average launch success trend.

The function will help you get the year from the date:

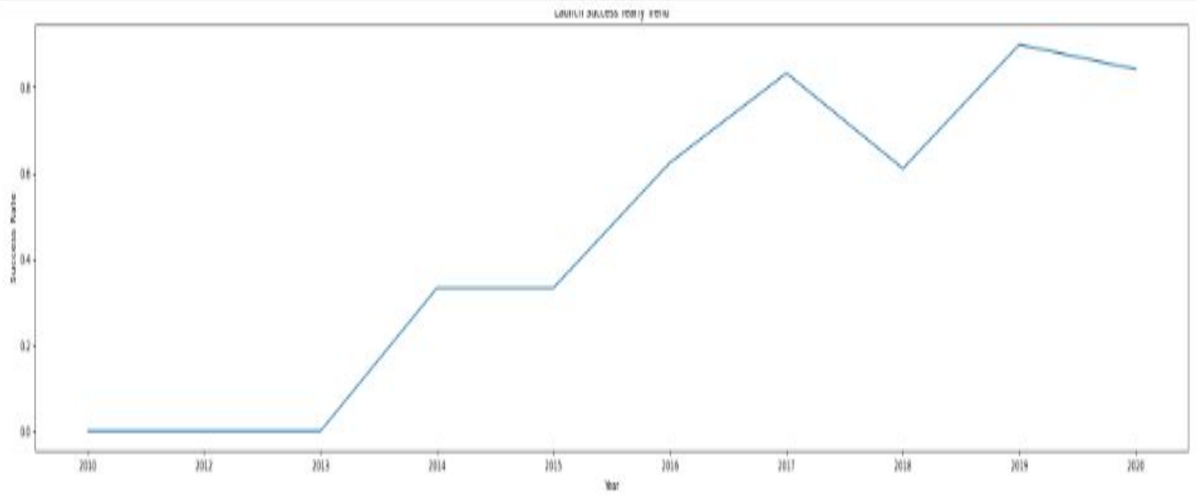
[11]:
```
# A function to Extract years from the date
year=[]
def Extract_year(date):
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year

Extract_year(1)
df['Year']=year

avg_year=df.groupby(by='Year').mean()
avg_year.reset_index(inplace=True)
```

[12]:
```
# Plot a line chart with x axis to be the extracted year and y axis to be the success

plt.plot(avg_year['Year'], avg_year['Class'])
plt.title('Launch Success Yearly Trend')
plt.xlabel('Year')
plt.ylabel('Success Rate')
plt.show()
```




you can observe that the sucess rate since 2013 kept increasing till 2020

# Results: Exploratory Data Analysis
## Visualization and Feature Engineering

### TASK 7: Create dummy variables to categorical columns

Use the function `get_dummies` and `features` dataframe to apply OneHotEncoder to the column `Orbits` , `LaunchSite` , `LandingPad` , and `Serial` . Assign the value to the variable `features_one_hot` , display the results using the method head. Your result dataframe must include all features including the encoded ones.

[44]:
```
# HINT: Use get_dummies() function on the categorical columns

features_one_hot = pd.get_dummies(features, columns = ['Orbit', 'LaunchSite', 'LandingPad', 'Serial'])
features_one_hot.head(10)
```

[44]:

| | FlightNumber | PayloadMass | Flights | GridFins | Reused | Legs | Block | ReusedCount | Orbit_ES-L1 | Orbit_GEO | ... | Serial_B1048 | Serial_B1049 | Serial_B1050 | Serial_B1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 6104.959412 | 1 | False | False | False | 1.0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | |
| **1** | 2 | 525.000000 | 1 | False | False | False | 1.0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | |
| **2** | 3 | 677.000000 | 1 | False | False | False | 1.0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | |
| **3** | 4 | 500.000000 | 1 | False | False | False | 1.0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | |
| **4** | 5 | 3170.000000 | 1 | False | False | False | 1.0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | |
| **5** | 6 | 3325.000000 | 1 | False | False | False | 1.0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | |
| **6** | 7 | 2296.000000 | 1 | False | False | True | 1.0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | |
| **7** | 8 | 1316.000000 | 1 | False | False | True | 1.0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | |
| **8** | 9 | 4535.000000 | 1 | False | False | False | 1.0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | |
| **9** | 10 | 4428.000000 | 1 | False | False | False | 1.0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | |

10 rows × 80 columns

Mode: Command Ln 1, Col 1 jupyter-labs-eda-dataviz.ipynb.jupyterl

# Results: Exploratory Data Analysis

## Visualization and Feature Engineering

### TASK 8: Cast all numeric columns to `float64`

Now that our `features_one_hot` dataframe only contains numbers cast the entire dataframe to variable type `float64`

```
# HINT: use astype function

features_one_hot.astype('float64')
```

| | FlightNumber | PayloadMass | Flights | GridFins | Reused | Legs | Block | ReusedCount | Orbit_ES-L1 | Orbit_GEO | ... | Serial_B1048 | Serial_B1049 | Serial_B1050 | Serial_B1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1.0 | 6104.959412 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | |
| **1** | 2.0 | 525.000000 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | |
| **2** | 3.0 | 677.000000 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | |
| **3** | 4.0 | 500.000000 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | |
| **4** | 5.0 | 3170.000000 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **85** | 86.0 | 15400.000000 | 2.0 | 1.0 | 1.0 | 1.0 | 5.0 | 2.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | |
| **86** | 87.0 | 15400.000000 | 3.0 | 1.0 | 1.0 | 1.0 | 5.0 | 2.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | |
| **87** | 88.0 | 15400.000000 | 6.0 | 1.0 | 1.0 | 1.0 | 5.0 | 5.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | |
| **88** | 89.0 | 15400.000000 | 3.0 | 1.0 | 1.0 | 1.0 | 5.0 | 2.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | |
| **89** | 90.0 | 3681.000000 | 1.0 | 1.0 | 0.0 | 1.0 | 5.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | |

90 rows × 80 columns

# Results: Interactive Maps with Folium

## Task 1: Mark all launch sites on a map

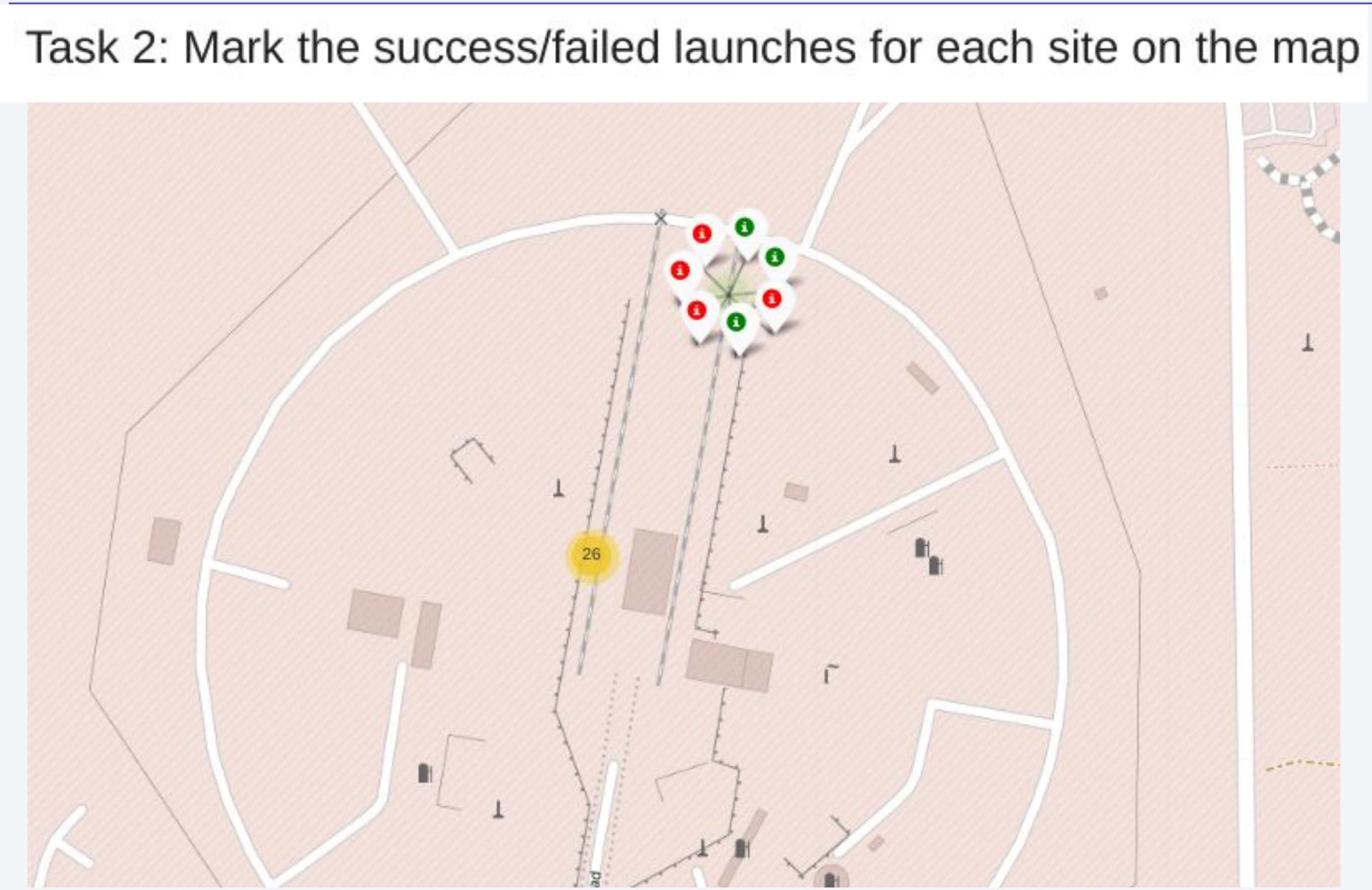

Interactive map with launch sites and NASA JSC

# Results: Interactive Maps with Folium

Task 2: Mark the success/failed launches for each site on the map


Interactive map with KSC LC-40 Launches

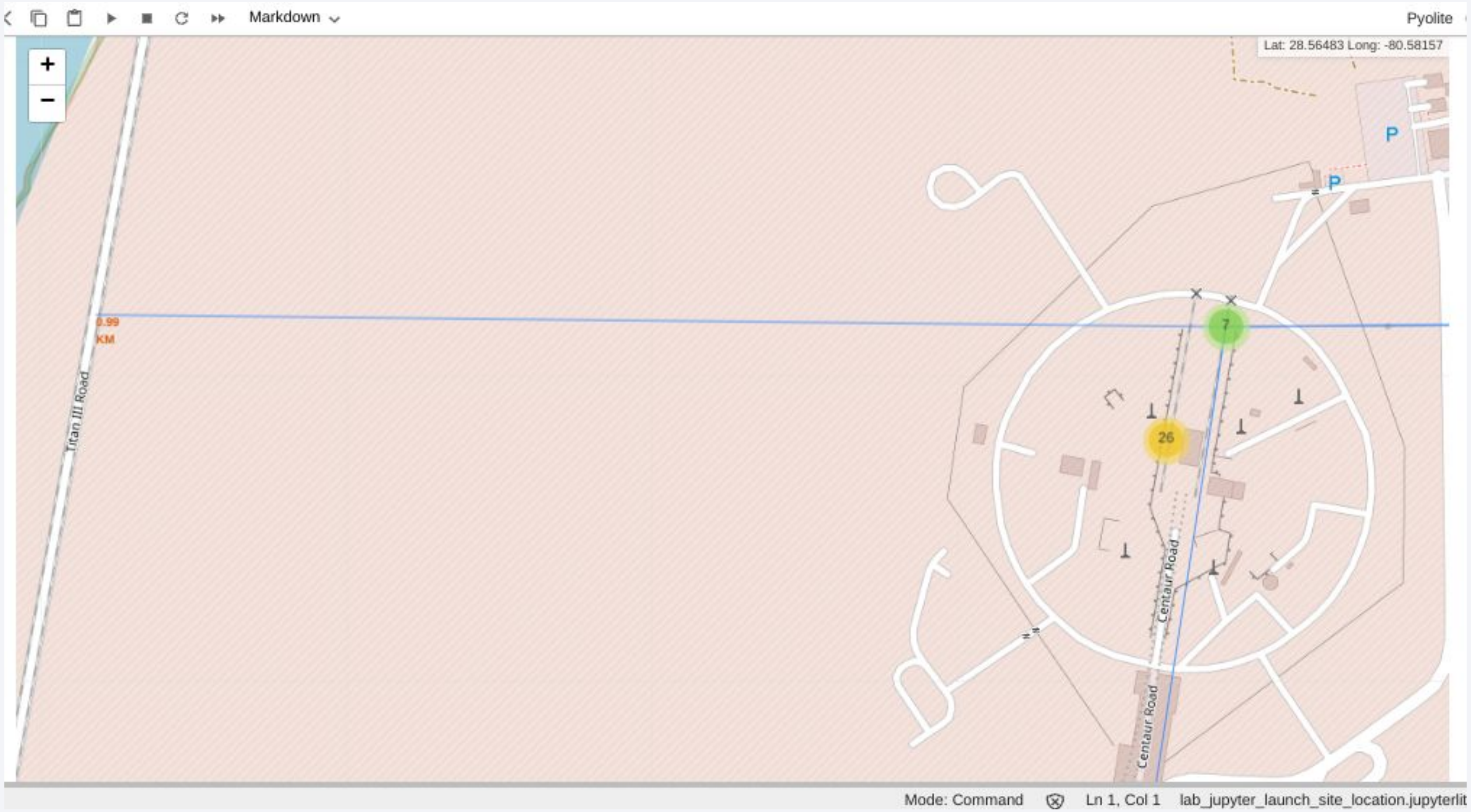
# Results: Interactive Maps with Folium




Interactive map with KSC SLC-40 Launches

# Results: Interactive Maps with Folium

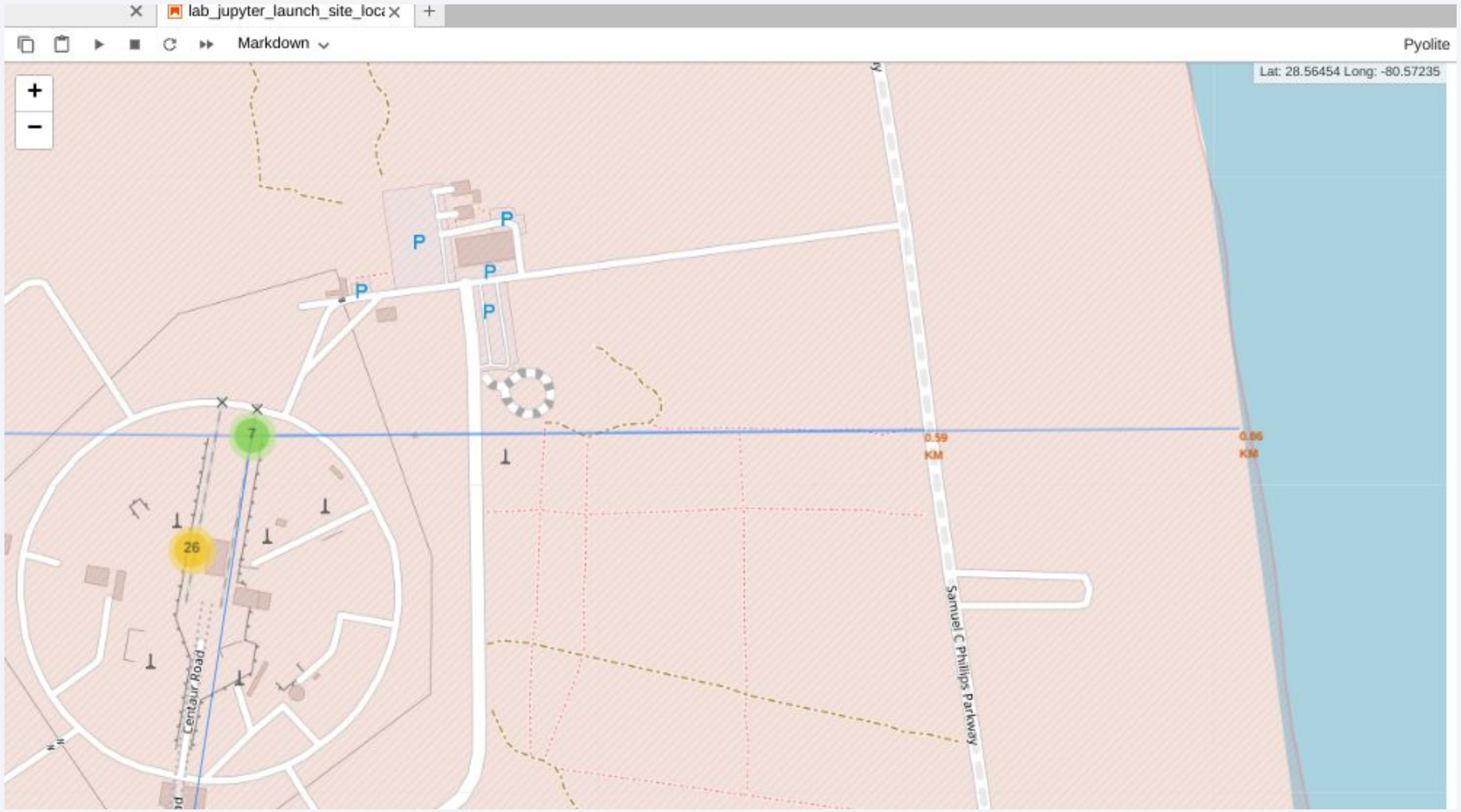
TASK 3: Calculate the distances between a launch site to its proximities




Interactive map with SLC-40 distance to closest railway (along Titan III Road)

# Results: Interactive Maps with Folium

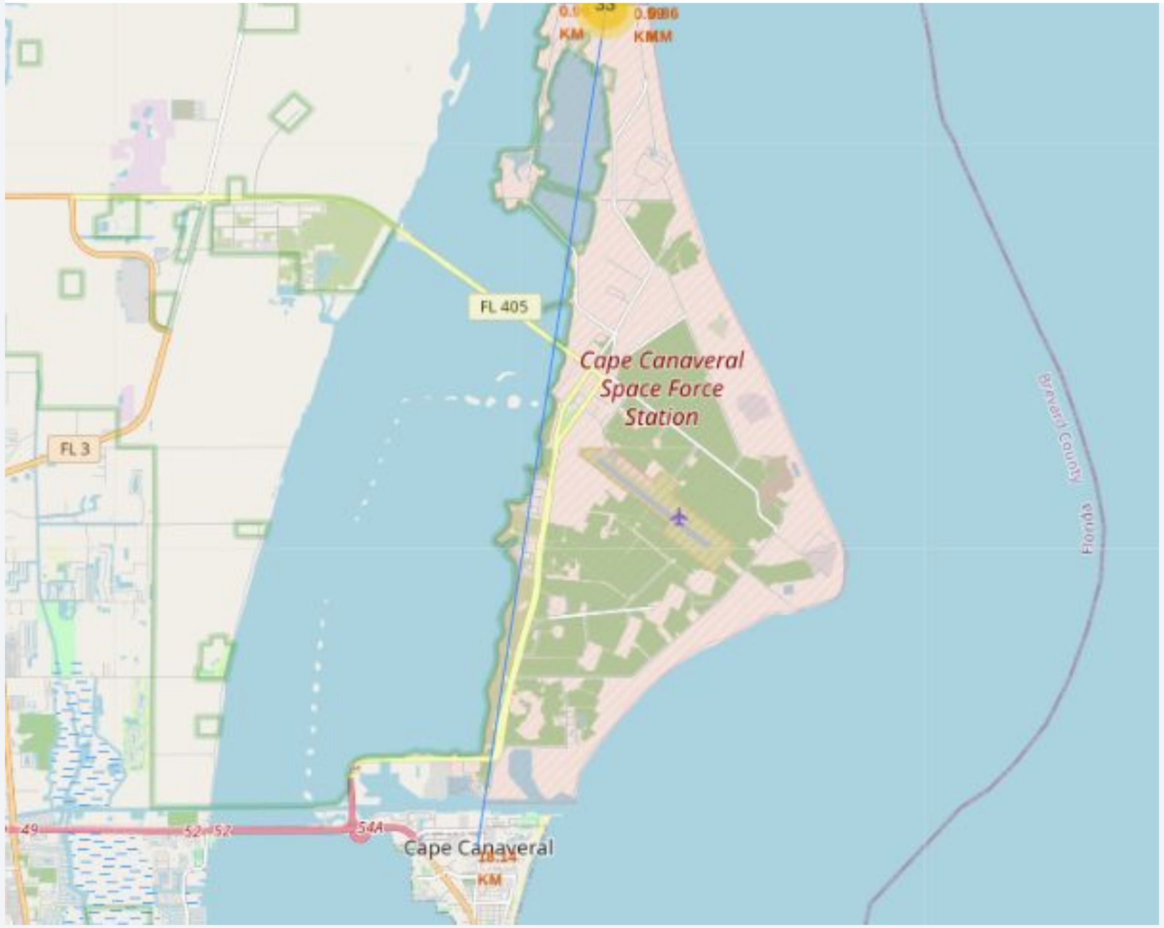
TASK 3: Calculate the distances between a launch site to its proximities




Interactive map with SLC-40 distance to closest road (Samuel C Phillips Parkway) and coastline

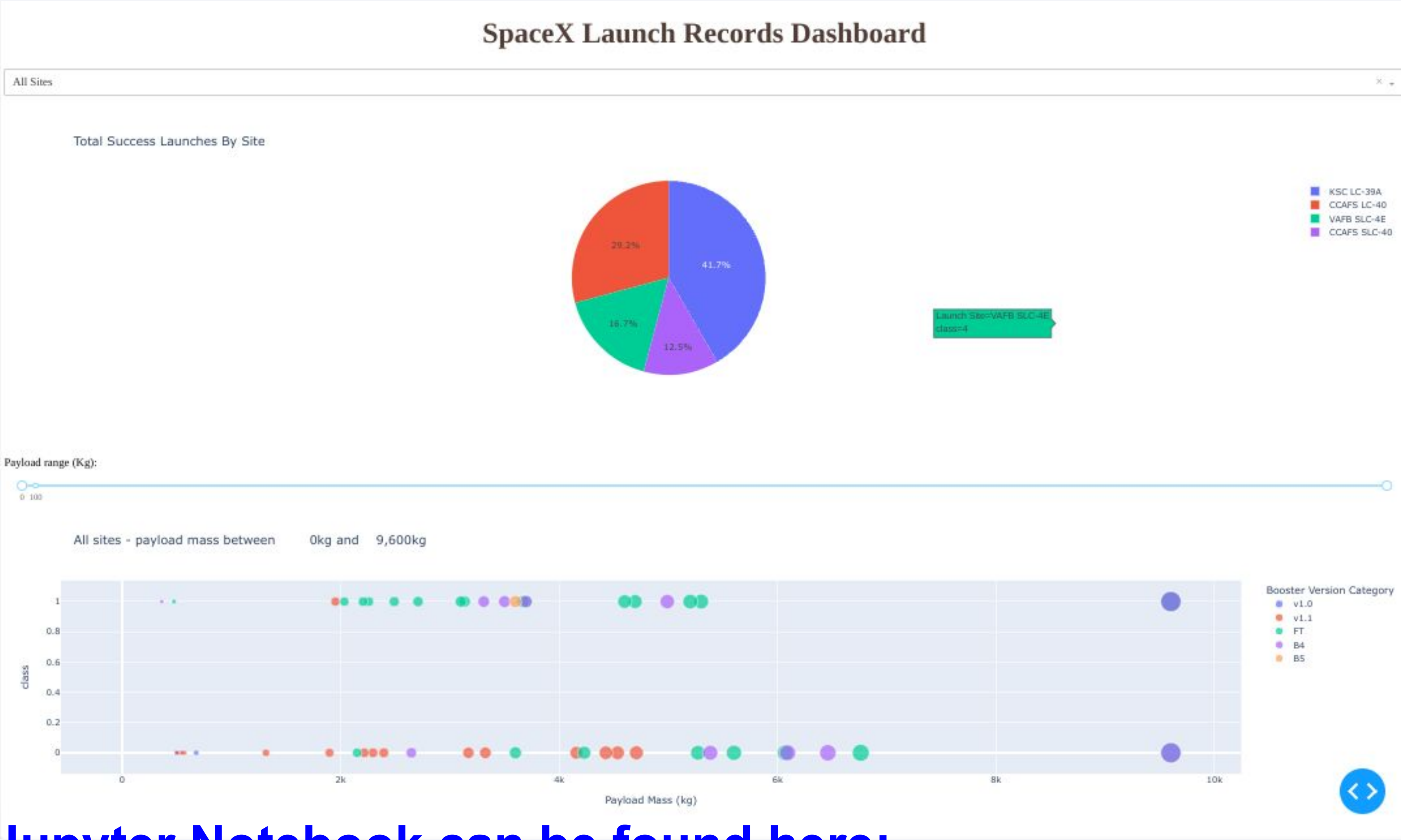
# Results: Interactive Maps with Folium

TASK 3: Calculate the distances between a launch site to its proximities




Interactive map with SLC-40 distance to closest city (Cape Canaveral, FL)

# Results: Dashboarding with Plotly Dash




**Jupyter Notebook can be found here:**

# Results: Predictive Analysis

## TASK 1

Create a NumPy array from the column `Class` in `data`, by applying the method `to_numpy()` then assign it to the variable `Y`,make sure the output is a Pandas series (only one bracket df['name of column']).

```
[8]: Y = data['Class'].to_numpy()
print(Y)
```

```
[0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 1 0 0 1 1 1 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1
 1 1 1 1 1 1 1 1 0 0 0 1 1 0 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 0 1
 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

## TASK 2

Standardize the data in `X` then reassign it to the variable `X` using the transform provided below.

```
[9]: # students get this
transform = preprocessing.StandardScaler()
```

```
[10]: X = transform.fit_transform(X)
print(X)
```

```
[[-1.71291154e+00 -1.94814463e-16 -6.53912840e-01 ... -8.35531692e-01
   1.93309133e+00 -1.93309133e+00]
 [-1.67441914e+00 -1.19523159e+00 -6.53912840e-01 ... -8.35531692e-01
   1.93309133e+00 -1.93309133e+00]
 [-1.63592675e+00 -1.16267307e+00 -6.53912840e-01 ... -8.35531692e-01
   1.93309133e+00 -1.93309133e+00]
 ...
 [ 1.63592675e+00  1.99100483e+00  3.49060516e+00 ...  1.19684269e+00
  -5.17306132e-01  5.17306132e-01]
 [ 1.67441914e+00  1.99100483e+00  1.00389436e+00 ...  1.19684269e+00
  -5.17306132e-01  5.17306132e-01]
 [ 1.71291154e+00 -5.19213966e-01 -6.53912840e-01 ... -8.35531692e-01
  -5.17306132e-01  5.17306132e-01]]
```

We split the data into training and testing data using the function `train_test_split`. The training data is divided into validation data, a second set used for training data; then the models are trained and hyperparameters are selected using the function `GridSearchCV`.

# Results: Predictive Analysis

## TASK 3

Use the function train_test_split to split the data X and Y into training and test data. Set the parameter test_size to 0.2 and random_state to 2. The training data and test data should be assigned to the following labels.

`X_train, X_test, Y_train, Y_test`

[11]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state = 2)
```

we can see we only have 18 test samples.

[12]:
```
Y_test.shape
```

[12]:
```
(18,)
```

# Results: Predictive Analysis

## TASK 4

Create a logistic regression object then create a GridSearchCV object `logreg_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

```
[13]: parameters ={'C':[0.01,0.1,1],
                   'penalty':['l2'],
                   'solver':['lbfgs']}
```

```
[14]: lr_parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
      lr=LogisticRegression()
      logreg_cv = GridSearchCV(lr, lr_parameters, cv=10)
      logreg_cv.fit(X_train, Y_train)
```

```
[14]: GridSearchCV(cv=10, estimator=LogisticRegression(),
                   param_grid={'C': [0.01, 0.1, 1], 'penalty': ['l2'],
                               'solver': ['lbfgs']})
```

We output the `GridSearchCV` object for logistic regression. We display the best parameters using the data attribute `best_params_` and the accuracy on the validation data using the data attribute `best_score_`.

```
[15]: print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
      print("accuracy :",logreg_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8464285714285713
```

# Results: Predictive Analysis

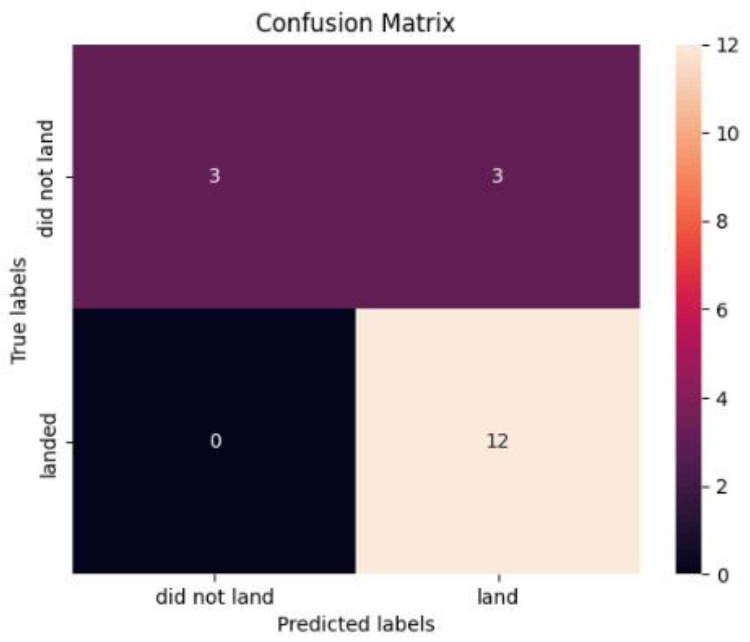
## TASK 5

Calculate the accuracy on the test data using the method `score` :

```
[16]: print('Logistic Regression Test Accuracy Score :',logreg_cv.score(X_test, Y_test))
```

```
Logistic Regression Test Accuracy Score : 0.8333333333333334
```

```
[17]: yhat=logreg_cv.predict(X_test)
      plot_confusion_matrix(Y_test,yhat)
```




Examining the confusion matrix, we see that logistic regression can distinguish between the different classes. We see that the major problem is false positives.

# Results: Predictive Analysis

## TASK 6

Create a support vector machine object then create a `GridSearchCV` object `svm_cv` with cv - 10. Fit the object to find the best parameters from the dictionary `parameters`.

```
[18]: svm_parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
                    'C': np.logspace(-3, 3, 5),
                    'gamma':np.logspace(-3, 3, 5)}
      svm = SVC()
```

```
[19]: svm_cv = GridSearchCV(svm, svm_parameters, cv=10)
      svm_cv.fit(X_train, Y_train)
```

```
[19]: GridSearchCV(cv=10, estimator=SVC(),
                   param_grid={'C': array([1.00000000e-03, 3.16227766e-02, 1.00000000e+00, 3.16227766e+01,
             1.00000000e+03]),
                               'gamma': array([1.00000000e-03, 3.16227766e-02, 1.00000000e+00, 3.16227766e+01,
             1.00000000e+03]),
                               'kernel': ('linear', 'rbf', 'poly', 'rbf', 'sigmoid')})
```

```
[20]: print("tuned hpyerparameters :(best parameters) ",svm_cv.best_params_)
      print("accuracy :",svm_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
accuracy : 0.8482142857142856
```

# Results: Predictive Analysis

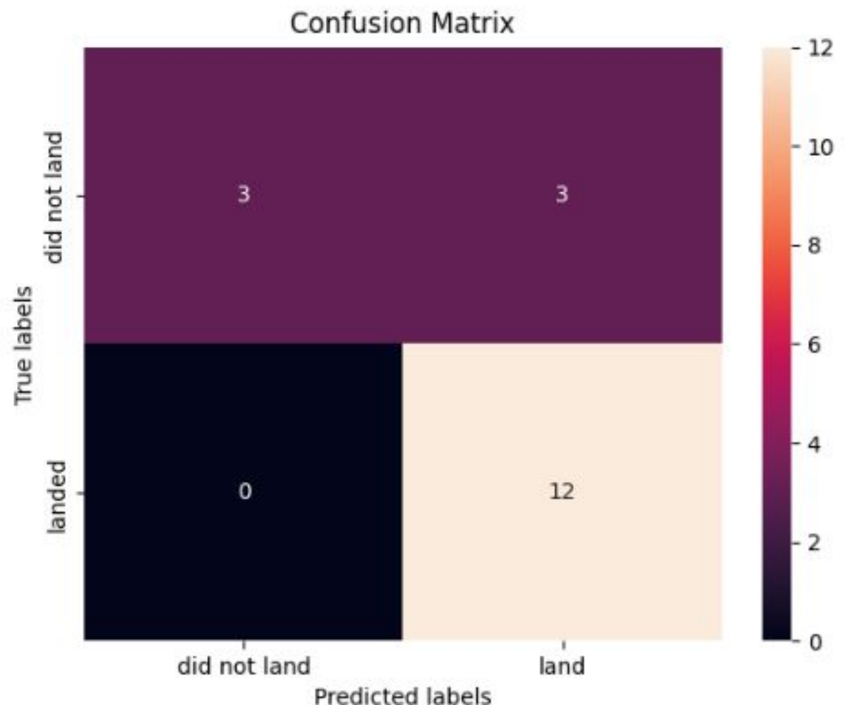
## TASK 7

Calculate the accuracy on the test data using the method `score` :

```
[21]: print('SVM Test Accuracy Score :', svm_cv.score(X_test, Y_test))
```

```
SVM Test Accuracy Score : 0.8333333333333334
```

```
[22]: yhat=svm_cv.predict(X_test)
      plot_confusion_matrix(Y_test,yhat)
```



# Results: Predictive Analysis

## TASK 8

Create a decision tree classifier object then create a `GridSearchCV` object `tree_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

```
[35]: tree_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]}

      tree = DecisionTreeClassifier()
```

```
[36]: tree_cv = GridSearchCV(tree, tree_parameters, cv=10)
      tree_cv.fit(X_train, Y_train)
```

```
[36]: GridSearchCV(cv=10, estimator=DecisionTreeClassifier(),
                   param_grid={'criterion': ['gini', 'entropy'],
                               'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18],
                               'max_features': ['auto', 'sqrt'],
                               'min_samples_leaf': [1, 2, 4],
                               'min_samples_split': [2, 5, 10],
                               'splitter': ['best', 'random']})
```

```
[37]: print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
      print("accuracy :",tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', 'max_depth': 2, 'max_features': 'auto', 'min_samples_leaf': 2, 'm
in_samples_split': 2, 'splitter': 'random'}
accuracy : 0.875
```

# Results: Predictive Analysis

## TASK 9

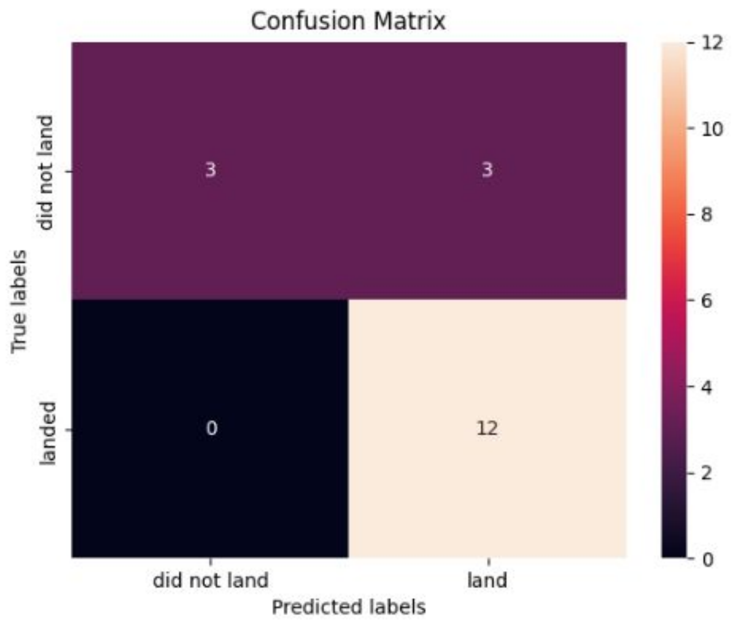
Calculate the accuracy of tree_cv on the test data using the method `score` :

```
[38]: print('Decision Tree Test Accuracy Score :', tree_cv.score(X_test, Y_test))
```

```
Decision Tree Test Accuracy Score : 0.8333333333333334
```

We can plot the confusion matrix

```
[39]: yhat = svm_cv.predict(X_test)
      plot_confusion_matrix(Y_test,yhat)
```



# Results: Predictive Analysis

## TASK 10

Create a k nearest neighbors object then create a `GridSearchCV` object `knn_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

```
[40]: KNN_parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                   'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                   'p': [1,2]}

      KNN = KNeighborsClassifier()
```

```
[41]: knn_cv = GridSearchCV(KNN, KNN_parameters, cv=10)
      knn_cv.fit(X_train, Y_train)
```

```
[41]: GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
                   param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                               'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                               'p': [1, 2]})
```

```
[42]: print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
      print("accuracy :",knn_cv.best_score_)

tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}
accuracy : 0.8482142857142858
```

# Results: Predictive Analysis

## TASK 11

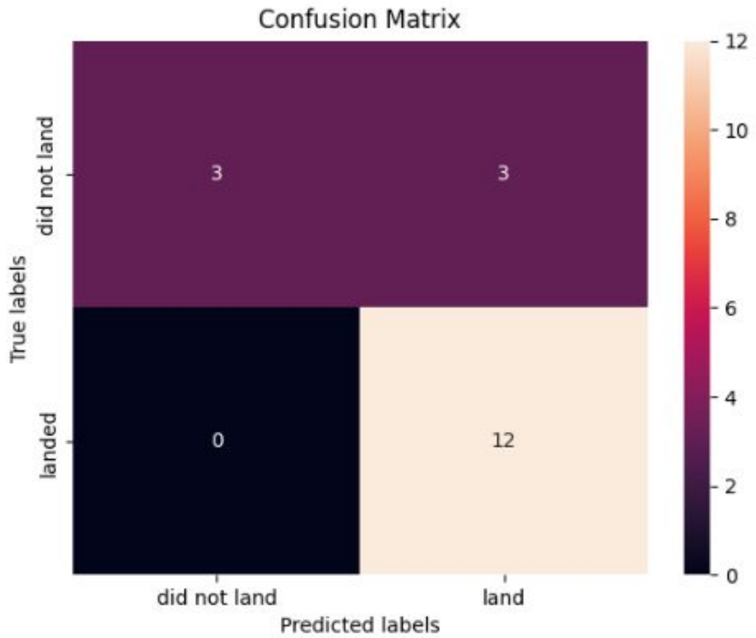
Calculate the accuracy of knn_cv on the test data using the method `score`:

```
print('KNN Test Accuracy Score :', knn_cv.score(X_test, Y_test))
```

```
KNN Test Accuracy Score : 0.8333333333333334
```

We can plot the confusion matrix

```
[44]: yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



# Results: Predictive Analysis

## TASK 12

Find the method performs best:

```
[45]: #all of the tests revealed all had the same accuracy score so all perform the same. Due to the small data set for each test.
print('Logistic Regression Test Accuracy Score :',logreg_cv.score(X_test, Y_test))
print('SVM Test Accuracy Score :', svm_cv.score(X_test, Y_test))
print('Decision Tree Test Accuracy Score :', tree_cv.score(X_test, Y_test))
print('KNN Test Accuracy Score :', knn_cv.score(X_test, Y_test))
```

```
Logistic Regression Test Accuracy Score : 0.8333333333333334
SVM Test Accuracy Score : 0.8333333333333334
Decision Tree Test Accuracy Score : 0.8333333333333334
KNN Test Accuracy Score : 0.8333333333333334
```

# Conclusions

- Data was obtained and converted into useful forms
- Exploratory data analysis (EDA) carried out using SQL queries and visualizations in Seabor/Matplotlib
- Interactive visual analytics via Folium
- Dashboarding using Plotly Dash
- Predictive analysis was demonstrated for various classification models

# Appendix

# Thank you!