IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Florian Nolte
04.12.2025

Outline

Outline

Outline

3.4 Part 3 – Lab 3: Exploratory analysis & label creation

Outline

Outline

Outline

Outline

Outline

Outline

1. Executive Summary

1.1 Overview of methodology

- The database was built from two sources:
  - API access to SpaceX launch data (Lab 1)
  - Web scraping of Wikipedia tables on Falcon 9 launches (Lab 2)
- This was followed by systematic data cleansing and wrangling (Lab 3):
  - Standardization of launch sites and orbits
  - Interpretation of landing types (RTLS, ASDS, ocean)
  - Creation of a binary target variable (class) for supervised learning
- The methodology follows a clear,
  reproducible workflow: data access → cleaning → feature creation → modeling.

1. Executive Summary

1.2 Summary of results

- The exploratory data analysis revealed clear patterns:
  - CCAFS SLC 40 is the most common launch site.
  - ISS and VLEO are the dominant orbit types.
  - Landings on ASDS (drone ships) are most often successful.
- With the target variables Class,
  a prediction accuracy of approximately 78% was achieved.
- The confusion matrix showed that the model is particularly reliable at recognizing successful landings, while misclassifications occur mainly in rare orbit types.
- Overall, the project shows that machine learning can predict the success of Falcon 9 landings and thus contribute to the optimization of future missions.

2. Introduction

2.1 Project background and context

- SpaceX aims to perfect the reusability of rockets through precise first-stage landings.
- For investors and engineers, predicting landing success is crucial to reducing costs and increasing reliability.
- This project is part of the IBM Data Science Capstone and uses real launch data to develop a predictive model for landing outcomes.

2. Introduction

2.2 Research questions and objectives

- What factors influence the success of a landing? → Launch site, orbit, payload mass, booster version, reuse count.
- How can the outcome data be converted into a target variable for machine learning? → Creation of the binary column Class (1 = successful, 0 = unsuccessful).
- How accurately can models predict landing success? → Comparison of accuracy and confusion matrix to evaluate model performance.
- The goal is to develop a robust classification model that reliably predicts the success of Falcon 9 landings.

Section 1

Methodology

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.1 The section 1 describes the following topics:

Objective of this section:

- Before we can predict the landing of the Falcon 9 first stage, we must systematically build, clean, and prepare the database.
- Section 1 forms the methodological foundation for all subsequent analyses and modeling.

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.1 The section 1 describes the following topics:

Structure of Section 1: In three parts

Part 1 – Lab 1: API access & data retrieval

- Extracting launch data, booster information, and payload masses
- Initial filtering for Falcon 9 and handling missing values →
  Goal: Building a structured, domain-specific database

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.1 The section 1 describes the following topics:

Part 1 – Task Overview

Contents:

- TASK 1: Request and parse the SpaceX launch data using the GET request
- TASK 2: Filter the DataFrame to only include Falcon 9 launches
- TASK 3: Dealing with Missing Values

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.1 The section 1 describes the following topics:

Structure of Section 1: In three parts

Part 2 – Lab 2: Web scraping historical launch data

- Scraping the Wikipedia table “List of Falcon 9 and Falcon Heavy launches”
- Parsing the HTML structure with BeautifulSoup
- Extracting date, booster version, payload mass, and landing status → **Goal:** Supplementing API data with publicly documented mission history

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.1 The section 1 describes the following topics:

Part 2 – Task Overview

Contents:

- TASK 1: Request the Falcon9 launch Wiki page from its URL
- TASK 2: Extract all column/ variable names from the HTML table header
- TASK 3: Create a data frame by parsing the launch HTML tables

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.1 The section 1 describes the following topics:

Structure of Section 1: In three parts

Part 3 – Lab 3: Exploratory analysis & label creation

- Analysis of launch sites, orbits, and mission outcomes
- Interpretation of landing types (RTLS, ASDS, Ocean)
- Creation of binary labels for model training (1 = successful, 0 = unsuccessful) → **Goal:** Definition of the target variable for supervised learning

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.1 The section 1 describes the following topics:

Part 3 – Task Overview

Contents:

- TASK 1: Calculate the number of launches on each site
- TASK 2: Calculate the number and occurrence of each orbit
- TASK 3: Calculate the number and occurrence of mission outcome of the orbits
- TASK 4: Create a landing outcome label from Outcome column

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.1 The section 1 describes the following topics:

Methodological focus:

- Reproducible steps with clear documentation
- Domain knowledge for meaningful interpretation
- Technical cleanliness as the basis for model training

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 1: Request and parse the SpaceX launch data using the GET request

- Introduction: Purpose, API structure, expected data formats
- The screenshot displays the connecting to the API and initial data determination.

Task 1: Request and parse the SpaceX launch data using the GET request

To make the requested JSON results more consistent, we will use the following static response object for this project:

16]:
```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_call_spacex_api.json'
```

We should see that the request was successfull with the 200 status response code

17]:
```
response.status_code
```

17]:
```
200
```

Now we decode the response content as a Json using `.json()` and turn it into a Pandas dataframe using `.json_normalize()`

18]:
```
# Use json_normalize meethod to convert the json result into a dataframe
# Get JSON data
response = requests.get(spacex_url)
data = response.json()

# Convert JSON to DataFrame
df = pd.json_normalize(data)
```

Using the dataframe `data` print the first 5 rows

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 1: Request and parse the SpaceX launch data using the GET request

- Introduction: Purpose, API structure, expected data formats
- The screenshot displays the connecting to the API and initial data determination.

```python
[16]: import pandas as pd
import requests

# Hole Daten aus der API
spacex_url="https://api.spacexdata.com/v4/launches/past"
response = requests.get(spacex_url)
data = response.json()  # Das ist eine Liste von Dicts

# In DataFrame umwandeln
df = pd.json_normalize(data)

print(df.head())
```

```
        static_fire_date_utc  static_fire_date_unix    net  window  \
0  2006-03-17T00:00:00.000Z           1.142554e+09  False     0.0
1                      None                    NaN  False     0.0
2                      None                    NaN  False     0.0
3  2008-09-20T00:00:00.000Z           1.221869e+09  False     0.0
4                      None                    NaN  False     0.0

                     rocket success  \
0  5e9d0d95eda69955f709d1eb   False
1  5e9d0d95eda69955f709d1eb   False
2  5e9d0d95eda69955f709d1eb   False
3  5e9d0d95eda69955f709d1eb    True
4  5e9d0d95eda69955f709d1eb    True

                                                                                     failures  \
0                                      [{'time': 33, 'altitude': None, 'reason': 'merlin engine failure'}]
1            [{'time': 301, 'altitude': 289, 'reason': 'harmonic oscillation leading to premature engine shutdown'}]
2  [{'time': 140, 'altitude': 35, 'reason': 'residual stage-1 thrust led to collision between stage 1 and stage 2'}]
3                                                                                          []
4                                                                                          []
```

```
                                                                                                                                                          details  \
0                                                                                                                         Engine failure at 33 seconds and loss of vehicle
1  Successful first stage burn and transition to second stage, maximum altitude 289 km, Premature engine shutdown at T+7 min 30 s, Failed to reach orbit, Failed to recover first stage
2                                                                                                                   Residual stage 1 thrust led to collision between stage 1 and stage 2
3                          Ratsat was carried to orbit on the first successful orbital launch of any privately funded and developed, liquid-propelled carrier rocket, the SpaceX Falcon 1
4                                                                                                                                                                    None

  crew ships capsules                                              payloads  \
0   []    []       []                            [5eb0e4b5b6c3bb0006eeb1e1]
1   []    []       []                            [5eb0e4b6b6c3bb0006eeb1e2]
2   []    []       []  [5eb0e4b6b6c3bb0006eeb1e3, 5eb0e4b6b6c3bb0006eeb1e4]
3   []    []       []                            [5eb0e4b7b6c3bb0006eeb1e5]
4   []    []       []                            [5eb0e4b7b6c3bb0006eeb1e6]

                  launchpad  flight_number         name  \
0  5e9e4502f5090995de566f86              1    FalconSat
1  5e9e4502f5090995de566f86              2      DemoSat
2  5e9e4502f5090995de566f86              3  Trailblazer
3  5e9e4502f5090995de566f86              4       RatSat
4  5e9e4502f5090995de566f86              5     RazakSat

                   date_utc   date_unix                 date_local  \
0  2006-03-24T22:30:00.000Z  1143239400  2006-03-25T10:30:00+12:00
1  2007-03-21T01:10:00.000Z  1174439400  2007-03-21T13:10:00+12:00
2  2008-08-03T03:34:00.000Z  1217734440  2008-08-03T15:34:00+12:00
3  2008-09-28T23:15:00.000Z  1222643700  2008-09-28T11:15:00+12:00
4  2009-07-13T03:35:00.000Z  1247456100  2009-07-13T15:35:00+12:00

  date_precision  upcoming  \
0           hour     False
1           hour     False
2           hour     False
3           hour     False
4           hour     False
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

<u>**TASK 1:** Request and parse the SpaceX launch data using the GET request</u>

- Introduction: Purpose, API structure, expected data formats
- The screenshot displays the connecting to the API and initial data determination.

```
                                                                                                                                                                                    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}]
2  [{'core': '5e9e289ef3591814873b2625', '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}]

  auto_update    tbd launch_library_id                        id  \
0        True  False              None  5eb87cd9ffd86e000604b32a
1        True  False              None  5eb87cdaffd86e000604b32b
2        True  False              None  5eb87cdbffd86e000604b32c
3        True  False              None  5eb87cdbffd86e000604b32d
4        True  False              None  5eb87cdcffd86e000604b32e

  fairings.reused fairings.recovery_attempt fairings.recovered fairings.ships  \
0           False                     False              False             []
1           False                     False              False             []
2           False                     False              False             []
3           False                     False              False             []
4           False                     False              False             []

                              links.patch.small  \
0  https://images2.imgbox.com/94/f2/NN6Ph45r_o.png
1  https://images2.imgbox.com/f9/4a/ZboXReNb_o.png
2  https://images2.imgbox.com/6c/cb/na1tzhHs_o.png
3  https://images2.imgbox.com/95/39/sRqN7rsv_o.png
4  https://images2.imgbox.com/ab/5a/Pequxd5d_o.png

                              links.patch.large links.reddit.campaign  \
0  https://images2.imgbox.com/5b/02/QcxHUb5V_o.png                 None
1  https://images2.imgbox.com/80/a2/bkWotCIS_o.png                 None
2  https://images2.imgbox.com/4a/80/k1oAkY0k_o.png                 None
3  https://images2.imgbox.com/a3/99/qswRYzE8_o.png                 None
4  https://images2.imgbox.com/92/e4/7Cf6MLY0_o.png                 None

  links.reddit.launch links.reddit.media links.reddit.recovery  \
0                None               None                  None
1                None               None                  None
2                None               None                  None
3                None               None                  None
4                None               None                  None
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 1: Request and parse the SpaceX launch data using the GET request

- Introduction: Purpose, API structure, expected data formats
- The screenshot displays the connecting to the API and initial data determination.

```
  links.flickr.small links.flickr.original  \
0                 []                    []
1                 []                    []
2                 []                    []
3                 []                    []
4                 []                    []

                                                                                links.presskit  \
0                                                                                         None
1                                                                                         None
2                                                                                         None
3                                                                                         None
4  http://www.spacex.com/press/2012/12/19/spacexs-falcon-1-successfully-delivers-razaksat-satellite-orbit

                                links.webcast links.youtube_id  \
0  https://www.youtube.com/watch?v=0a_00nJ_Y88      0a_00nJ_Y88
1  https://www.youtube.com/watch?v=Lk4zQ2wP-Nc      Lk4zQ2wP-Nc
2  https://www.youtube.com/watch?v=v0w9p3U8860      v0w9p3U8860
3  https://www.youtube.com/watch?v=dLQ2tZEH6G0      dLQ2tZEH6G0
4  https://www.youtube.com/watch?v=yTaIDooc8Og      yTaIDooc8Og

                                                                 links.article  \
0  https://www.space.com/2196-spacex-inaugural-falcon-1-rocket-lost-launch.html
1     https://www.space.com/3590-spacex-falcon-1-rocket-fails-reach-orbit.html
2      http://www.spacex.com/news/2013/02/11/falcon-1-flight-3-mission-summary
3                                    https://en.wikipedia.org/wiki/Ratsat
4                     http://www.spacex.com/news/2013/02/12/falcon-1-flight-5

                                   links.wikipedia  fairings
0                https://en.wikipedia.org/wiki/DemoSat       NaN
1                https://en.wikipedia.org/wiki/DemoSat       NaN
2  https://en.wikipedia.org/wiki/Trailblazer_(satellite)       NaN
3                 https://en.wikipedia.org/wiki/Ratsat       NaN
4               https://en.wikipedia.org/wiki/RazakSAT       NaN
```

You will notice that a lot of the data are IDs. For example the rocket column has no information about the rocket just an identification number.

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 1: Request and parse the SpaceX launch data using the GET request

- Introduction: Purpose, API structure, expected data formats
- The screenshot displays the connecting to the API and initial data determination.

[23]:
```
# Lets take a subset of our dataframe keeping only the features we want and the flight number, and date_utc.
df = df[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]

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

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

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

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

[Kein Titel]

- From the `rocket` we would like to learn the booster name
- From the `payload` we would like to learn the mass of the payload and the orbit that it is going to
- From the `launchpad` we would like to know the name of the launch site being used, the longitude, and the latitude.
- **From `cores` we would like to learn the outcome of the landing, the type of the landing, number of flights with that core, whether gridfins were used, whether the core is reused, whether legs were used, the landing pad used, the block of the core which is a number used to seperate version of cores, the number of times this specific core has been reused, and the serial of the core.**

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 1: Request and parse the SpaceX launch data using the GET request

- Introduction: Purpose, API structure, expected data formats.
- The screenshot displays the definitions of global lists.

The data from these requests will be stored in lists and will be used to create a new dataframe.

[27]:
```
#Global lists
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
ReusedCount = []
Serial = []
Longitude = []
Latitude = []
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 1: Request and parse the SpaceX launch data using the GET request

- Introduction: Purpose, API structure, expected data formats
- The screenshot displays the definitions of the functions.

These functions will apply the outputs globally to the above variables. Let's take a looks at `BoosterVersion` variable. Before we apply `getBoosterVersion` the list is empty:

```
[20]: import requests

# Redefining functions
def getBoosterVersion(df):
    for rocket_id in df['rocket']:
        if rocket_id:
            response = requests.get(f"https://api.spacexdata.com/v4/rockets/{rocket_id}").json()
            BoosterVersion.append(response.get('name'))
        else:
            BoosterVersion.append(None)

def getPayloadData(df):
    for payload_id in df['payloads']:
        if payload_id:
            response = requests.get(f"https://api.spacexdata.com/v4/payloads/{payload_id}").json()
            PayloadMass.append(response.get('mass_kg'))
            Orbit.append(response.get('orbit'))
        else:
            PayloadMass.append(None)
            Orbit.append(None)
```

The following code of the block

```
def getLaunchSite(df):
    for launchpad_id in df['launchpad']:
        if launchpad_id:
            response = requests.get(f"https://api.spacexdata.com/v4/launchpads/{launchpad_id}").json()
            LaunchSite.append(response.get('name'))
            Longitude.append(response.get('longitude'))
            Latitude.append(response.get('latitude'))
        else:
            LaunchSite.append(None)
            Longitude.append(None)
            Latitude.append(None)

def getCoreData(df):
    for core in df['cores']:
        core_id = core.get('core')
        if core_id:
            response = requests.get(f"https://api.spacexdata.com/v4/cores/{core_id}").json()
            Block.append(response.get('block'))
            ReusedCount.append(response.get('reuse_count'))
            Serial.append(response.get('serial'))
        else:
            Block.append(None)
            ReusedCount.append(None)
            Serial.append(None)

        Outcome.append(str(core.get('landing_success')) + ' ' + str(core.get('landing_type')))
        Flights.append(core.get('flight'))
        GridFins.append(core.get('gridfins'))
        Reused.append(core.get('reused'))
        Legs.append(core.get('legs'))
        LandingPad.append(core.get('landpad'))
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 1: Request and parse the SpaceX launch data using the GET request

- Introduction: Purpose, API structure, expected data formats
- The screenshot displays function calls.

```
[21]: # Call functions
      getBoosterVersion(df)
      getPayloadData(df)
      getLaunchSite(df)
      getCoreData(df)
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 1: Request and parse the SpaceX launch data using the GET request

- Introduction: Purpose, API structure, expected data formats
- The screenshot displays the definitions of the final DataFrame.

```
[22]: # Create final DataFrame
launch_dict = {
    'FlightNumber': list(df['flight_number']),
    'Date': list(df['date']),
    'BoosterVersion': BoosterVersion,
    'PayloadMass': PayloadMass,
    'Orbit': Orbit,
    'LaunchSite': LaunchSite,
    'Longitude': Longitude,
    'Latitude': Latitude,
    'GridFins': GridFins,
    'Reused': Reused,
    'Legs': Legs,
    'LandingPad': LandingPad,
    'Block': Block,
    'ReusedCount': ReusedCount,
    'Serial': Serial,
    'Outcome': Outcome,
    'Flights': Flights
}

final_df = pd.DataFrame(launch_dict)
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 1: Request and parse the SpaceX launch data using the GET request

- Introduction: Purpose, API structure, expected data formats
- The screenshot displays the final DataFrame.

```
[23]: # Check output
final_df.head()
```

[23]:

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

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 1: Request and parse the SpaceX launch data using the GET request

- Introduction: Purpose, API structure, expected data formats
- The screenshot displays the function call.

Now, let's apply `getBoosterVersion` function method to get the booster version

[25]:
```
# Call getBoosterVersion
# Initialize list
BoosterVersion = []

# Call API function → fills the list
getBoosterVersion(df)

# Assign BoosterVersion to the DataFrame column
df['BoosterVersion'] = BoosterVersion
```

the list has now been update

[26]:
```
# Preview of the list itself
BoosterVersion[0:5]
```

[26]:
```
['Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 9']
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 1: Request and parse the SpaceX launch data using the GET request

- Introduction: Purpose, API structure, expected data formats
- The screenshot displays the DataFrame with flight_number and BoosterVersion data.

```
[26]: # Preview of the new column in the DataFrame
      df[['flight_number', 'BoosterVersion']].head()
```

[26]:

	flight_number	BoosterVersion
0	1	Falcon 1
1	2	Falcon 1
3	4	Falcon 1
4	5	Falcon 1
5	6	Falcon 9

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 1: Request and parse the SpaceX launch data using the GET request

- Introduction: Purpose, API structure, expected data formats
- The screenshot displays the function call and data.

```
[27]: # Call getLaunchSite
      # Initialize list
      LaunchSite = []
      Longitude = []
      Latitude = []

      # Call API function → fills the list
      getLaunchSite(df)

      # Assign values to the DataFrame column
      df['LaunchSite'] = LaunchSite
      df['Longitude'] = Longitude
      df['Latitude'] = Latitude

[28]: # Preview of the list itself
      LaunchSite[0:5]

[28]: ['Kwajalein Atoll',
       'Kwajalein Atoll',
       'Kwajalein Atoll',
       'Kwajalein Atoll',
       'CCSFS SLC 40']
```

```
[29]: # Preview of the list itself
      Longitude[0:5]

[29]: [167.7431292, 167.7431292, 167.7431292, 167.7431292, -80.577366]

[30]: # Preview of the list itself
      Latitude[0:5]

[30]: [9.0477206, 9.0477206, 9.0477206, 9.0477206, 28.5618571]
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 1: Request and parse the SpaceX launch data using the GET request

- Introduction: Purpose, API structure, expected data formats
- The screenshot displays the DataFrame with LaunchSite, Longitude and Latitude data.

```
[31]: # Preview of the new column in the DataFrame
      df[['LaunchSite', 'Longitude', 'Latitude']].head()
```

[31]:

	LaunchSite	Longitude	Latitude
0	Kwajalein Atoll	167.743129	9.047721
1	Kwajalein Atoll	167.743129	9.047721
3	Kwajalein Atoll	167.743129	9.047721
4	Kwajalein Atoll	167.743129	9.047721
5	CCSFS SLC 40	-80.577366	28.561857

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 1: Request and parse the SpaceX launch data using the GET request

- Introduction: Purpose, API structure, expected data formats
- The screenshot displays the function call and data.

```
[32]: # Call getPayloadData
      # Initialize list
      PayloadMass = []
      Orbit = []

      # Call API function → fills the list
      getPayloadData(df)

      # Assign values to the DataFrame column
      df['PayloadMass'] = PayloadMass
      df['Orbit'] = Orbit

[33]: # Preview of the list itself
      PayloadMass[0:5]

[33]: [20, None, 165, 200, None]

[34]: # Preview of the list itself
      Orbit[0:10]

[34]: ['LEO', 'LEO', 'LEO', 'LEO', 'LEO', 'LEO', 'ISS', 'PO', 'GTO', 'GTO']
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 1: Request and parse the SpaceX launch data using the GET request

- Introduction: Purpose, API structure, expected data formats
- The screenshot displays the DataFrame with PayloadMass and Orbit data.

```
[35]: # Preview of the new column in the DataFrame
      df[['PayloadMass', 'Orbit']].head(10)
```

[35]:

	PayloadMass	Orbit
0	20.0	LEO
1	NaN	LEO
3	165.0	LEO
4	200.0	LEO
5	NaN	LEO
7	525.0	LEO
9	677.0	ISS
10	500.0	PO
11	3170.0	GTO
12	3325.0	GTO

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 1: Request and parse the SpaceX launch data using the GET request

- Introduction: Purpose, API structure, expected data formats
- The screenshot displays the function call and set DataFrame values.

```
[36]: # Call getCoreData
      # Initialize core-related lists
      Outcome = []
      Flights = []
      GridFins = []
      Reused = []
      Legs = []
      LandingPad = []
      Block = []
      ReusedCount = []
      Serial = []

      # Call the API function to fill the lists
      getCoreData(df)

      # Assign the filled lists to the DataFrame
      df['Outcome'] = Outcome
      df['Flights'] = Flights
      df['GridFins'] = GridFins
      df['Reused'] = Reused
      df['Legs'] = Legs
      df['LandingPad'] = LandingPad
      df['Block'] = Block
      df['ReusedCount'] = ReusedCount
      df['Serial'] = Serial
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 1: Request and parse the SpaceX launch data using the GET request

- Introduction: Purpose, API structure, expected data formats
- The screenshot displays the DataFrame with the first entries.

```
[37]: # Preview of the new column in the DataFrame
df[['Outcome', 'Flights', 'GridFins', 'Reused', 'Legs', 'LandingPad', 'Block', 'ReusedCount', 'Serial']].head(50)
```

[37]:

	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial
0	None None	1	False	False	False	None	NaN	0	Merlin1A
1	None None	1	False	False	False	None	NaN	0	Merlin2A
3	None None	1	False	False	False	None	NaN	0	Merlin2C
4	None None	1	False	False	False	None	NaN	0	Merlin3C
5	None None	1	False	False	False	None	1.0	0	B0003
7	None None	1	False	False	False	None	1.0	0	B0005
9	None None	1	False	False	False	None	1.0	0	B0007
10	False Ocean	1	False	False	False	None	1.0	0	B1003
11	None None	1	False	False	False	None	1.0	0	B1004

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 1: Request and parse the SpaceX launch data using the GET request

- Introduction: Purpose, API structure, expected data formats
- The screenshot displays the DataFrame with the first entries.

```
[38]: # Filter data by launches with known landing data
df_with_landing = df[df['Outcome'].notna()]

# Display data
df_with_landing.head()
```

[38]:

	rocket	payloads	launchpad	cores	flight_number	date_utc	date	BoosterVersion	LaunchSite	Longitude	Latitude	PayloadMass	Orbit
0	5e9d0d95eda69955f709d1eb	5eb0e4b5b6c3bb0006eeb1e1	5e9e4502f5090995de566f86	{'core': '5e9e289df35918033d3b2623', 'flight': 1, 'gridfins': False, 'legs': False, 'reused': False, 'landing_attempt': False, 'landing_success': None, 'landing_type': None, 'landpad': None}	1	2006-03-24T22:30:00.000Z	2006-03-24	Falcon 1	Kwajalein Atoll	167.743129	9.047721	20.0	LEO
1	5e9d0d95eda69955f709d1eb	5eb0e4b6b6c3bb0006eeb1e2	5e9e4502f5090995de566f86	{'core': '5e9e289ef35918416a3b2624', 'flight': 1, 'gridfins': False, 'legs': False, 'reused': False, 'landing_attempt': False, 'landing_success': None, 'landing_type': None, 'landpad': None}	2	2007-03-21T01:10:00.000Z	2007-03-21	Falcon 1	Kwajalein Atoll	167.743129	9.047721	NaN	LEO
3	5e9d0d95eda69955f709d1eb	5eb0e4b7b6c3bb0006eeb1e5	5e9e4502f5090995de566f86	{'core': '5e9e289ef3591855dc3b2626', 'flight': 1, 'gridfins': False, 'legs': False, 'reused': False,	4	2008-09-	2008-	Falcon 1	Kwajalein	167.743129	9.047721	165.0	LEO

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 1: Request and parse the SpaceX launch data using the GET request

- Introduction: Purpose, API structure, expected data formats
- The screenshot displays the successful landings.

```
[39]: # Filter data by successful_landings
successful_landings = df_with_landing[df_with_landing['Outcome'].str.contains('True', na=False)]

# Display data
successful_landings.head()
```

[39]:

	rocket	payloads	launchpad	cores	flight_number	date_utc	date	BoosterVersion	LaunchSite	Longitude	Latitude	PayloadMass	Orbit	Outcome	Flights	GridFins	R
13	5e9d0d95eda69973a809d1ec	5eb0e4bbb6c3bb0006eeb1f1	5e9e4501f509094ba4566f84	{'core': '5e9e289ff3591829343b2630', 'flight': 1, 'gridfins': False, 'legs': True, 'reused': False, 'landing_attempt': True, 'landing_success': True, 'landing_type': 'Ocean', 'landpad': None}	14	2014-04-18T19:25:00.000Z	2014-04-18	Falcon 9	CCSFS SLC 40	-80.577366	28.561857	2296.0	ISS	True Ocean	1	False	
14	5e9d0d95eda69973a809d1ec	5eb0e4bcb6c3bb0006eeb1f2	5e9e4501f509094ba4566f84	{'core': '5e9e28a0f3591870a63b2631', 'flight': 1, 'gridfins': False, 'legs': True, 'reused': False, 'landing_attempt': True, 'landing_success': True, 'landing_type': 'Ocean', 'landpad': None}	15	2014-07-14T15:15:00.000Z	2014-07-14	Falcon 9	CCSFS SLC 40	-80.577366	28.561857	1316.0	LEO	True Ocean	1	False	
19	5e9d0d95eda69973a809d1ec	5eb0e4bdb6c3bb0006eeb1f7	5e9e4501f509094ba4566f84	{'core': '5e9e28a0f3591885be3b2636', 'flight': 1, 'gridfins': True, 'legs': True, 'reused': False, 'landing_attempt': True, 'landing_success': True, 'landing_type': 'Ocean', 'landpad': None}	20	2015-02-11T23:03:00.000Z	2015-02-11	Falcon 9	CCSFS SLC 40	-80.577366	28.561857	570.0	ES-L1	True Ocean	1	True	
24	5e9d0d95eda69973a809d1ec	5eb0e4beb6c3bb0006eeb1fd	5e9e4501f509094ba4566f84	{'core': '5e9e28a1f3591867753b263b', 'flight': 1, 'gridfins': True, 'legs': True, 'reused': False, 'landing_attempt': True, 'landing_success': True, 'landing_type': 'RTLS', 'landpad': '5e9e3032383ecb267a34e7c7'}	25	2015-12-22T01:29:00.000Z	2015-12-22	Falcon 9	CCSFS SLC 40	-80.577366	28.561857	2034.0	LEO	True RTLS	1	True	
27	5e9d0d95eda69973a809d1ec	5eb0e4bfb6c3bb0006eeb200	5e9e4501f509094ba4566f84	{'core': '5e9e28a2f359182d0b3b263e', 'flight': 1, 'gridfins': True, 'legs': True, 'reused': False, 'landing_attempt': True, 'landing_success': True, 'landing_type': 'ASDS', 'landpad':	28	2016-04-08T20:43:00.000Z	2016-04-08	Falcon 9	CCSFS SLC 40	-80.577366	28.561857	3136.0	ISS	True ASDS	1	True	

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 1: Request and parse the SpaceX launch data using the GET request

- Introduction: Purpose, API structure, expected data formats
- The screenshot displays the lists with length specifications.

```
[41]: # First, check the lengths of the lists.
      for key, value in launch_dict.items():
          print(f"{key}: {len(value)}")

FlightNumber: 94
Date: 94
BoosterVersion: 94
PayloadMass: 94
Orbit: 94
LaunchSite: 94
Longitude: 94
Latitude: 94
GridFins: 94
Reused: 94
Legs: 94
LandingPad: 94
Block: 94
ReusedCount: 94
Serial: 94
Outcome: 94
Flights: 94
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 1: Request and parse the SpaceX launch data using the GET request

- Introduction: Purpose, API structure, expected data formats
- The screenshot displays the DataFrame with the first entries.

```
[43]: # Create a data from launch_dict
final_df = pd.DataFrame(launch_dict)

# Show the head of the dataframe
final_df.head()
```

[43]:

	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	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0003	-80.577366	28.561857

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 1: Request and parse the SpaceX launch data using the GET request

- Introduction: Purpose, API structure, expected data formats
- The screenshot displays the summary of the DataFrame.

```
[44]: # Display summary of the DataFrame (final_df)
      final_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 94 entries, 0 to 93
Data columns (total 17 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   FlightNumber    94 non-null     int64
 1   Date            94 non-null     object
 2   BoosterVersion  94 non-null     object
 3   PayloadMass     88 non-null     float64
 4   Orbit           94 non-null     object
 5   LaunchSite      94 non-null     object
 6   Outcome         94 non-null     object
 7   Flights         94 non-null     int64
 8   GridFins        94 non-null     bool
 9   Reused          94 non-null     bool
 10  Legs            94 non-null     bool
 11  LandingPad      64 non-null     object
 12  Block           90 non-null     float64
 13  ReusedCount     94 non-null     int64
 14  Serial          94 non-null     object
 15  Longitude       94 non-null     float64
 16  Latitude        94 non-null     float64
dtypes: bool(3), float64(4), int64(3), object(7)
memory usage: 10.7+ KB
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 2: Filter the dataframe to only include Falcon 9 launches

- **Goal**: Reduction of the database to relevant launches with the Falcon 9 rocket, as only these are relevant for landing prediction and cost analysis.

Why only Falcon 9?

- Falcon 9 is reusable → crucial for cost savings.
- Other rockets (e.g., Falcon 1, heavy) have different technical profiles.
- Focus on homogeneous database for model training.

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 2: Filter the dataframe to only include Falcon 9 launches

Technical focus:

- Filtering via rocket.name == “Falcon 9”.
- Checking for consistency of names.
- Exclusion of irrelevant launches.

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 2: Filter the dataframe to only include Falcon 9 launches

- The screenshot displays the DataFrame with filter out only the falcon 9 launches.

Task 2: Filter the dataframe to only include `Falcon 9` launches

Finally we will remove the Falcon 1 launches keeping only the Falcon 9 launches. 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`.

[45]:
```
# Hint data['BoosterVersion']!='Falcon 1'
# Filter out only the Falcon 9 launches
data_falcon9 = final_df[final_df['BoosterVersion'].str.contains('Falcon 9', case=False, na=False)]

# Display the first few lines
data_falcon9.head()
```

[45]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude
4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0003	-80.577366	28.561857
5	8	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0005	-80.577366	28.561857
6	10	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0007	-80.577366	28.561857
7	11	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None	1.0	0	B1003	-120.610829	34.632093
8	12	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B1004	-80.577366	28.561857

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 2: Filter the dataframe to only include Falcon 9 launches

- The screenshot displays the rebuilding of the DataFrame.

```
[42]: # Create a copy of the filtered DataFrame
data_falcon9 = final_df[final_df['BoosterVersion'].str.contains('Falcon 9', case=False, na=False)].copy()

# Reset the index
data_falcon9.reset_index(drop=True, inplace=True)

# Update the FlightNumber column to sequential numbers
data_falcon9['FlightNumber'] = data_falcon9.index + 1
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 2: Filter the dataframe to only include Falcon 9 launches

- The screenshot displays the data preparation (data wrangling).

Data Wrangling

We can see below that some of the rows are missing values in our dataset.

Before we can continue we must deal with these missing values. The LandingPad column will retain None values to represent when landing pads were not used.

```
7]: # Identify missing values
    data_falcon9.isnull().sum()
```

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

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 2: Filter the dataframe to only include Falcon 9 launches

- The screenshot displays the data preparation (data wrangling), rows with missing values, first entries.

```
[48]: # Display rows with missing values
      data_falcon9[data_falcon9.isnull().any(axis=1)]
```

[48]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude
0	1	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0003	-80.577366	28.561857
1	2	2012-05-22	Falcon 9	525.00	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0005	-80.577366	28.561857
2	3	2013-03-01	Falcon 9	677.00	ISS	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0007	-80.577366	28.561857
3	4	2013-09-29	Falcon 9	500.00	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None	1.0	0	B1003	-120.610829	34.632093
4	5	2013-12-03	Falcon 9	3170.00	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B1004	-80.577366	28.561857
5	6	2014-01-06	Falcon 9	3325.00	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B1005	-80.577366	28.561857
6	7	2014-04-18	Falcon 9	2296.00	ISS	CCSFS SLC 40	True Ocean	1	False	False	True	None	1.0	0	B1006	-80.577366	28.561857
7	8	2014-07-14	Falcon 9	1316.00	LEO	CCSFS SLC 40	True Ocean	1	False	False	True	None	1.0	0	B1007	-80.577366	28.561857
8	9	2014-08-05	Falcon 9	4535.00	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B1008	-80.577366	28.561857
9	10	2014-09-07	Falcon 9	4428.00	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B1011	-80.577366	28.561857
10	11	2014-09-21	Falcon 9	2216.00	ISS	CCSFS SLC 40	False Ocean	1	False	False	False	None	1.0	0	B1010	-80.577366	28.561857

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 2: Filter the dataframe to only include Falcon 9 launches

- The screenshot displays the data preparation (data wrangling).

Before we can continue we must deal with these missing values. The `LandingPad` column will retain None values to represent when landing pads were not used.

[49]:
```
# Remove rows with NaN in all columns except 'LandingPad'
data_falcon9_cleaned = data_falcon9.dropna(subset=[col for col in data_falcon9.columns if col != 'LandingPad'])
```

[50]:
```
# Ensure that only LandingPad contains NaN values
data_falcon9_cleaned.isnull().sum()
```

[50]:
```
FlightNumber      0
Date              0
BoosterVersion    0
PayloadMass       0
Orbit             0
LaunchSite        0
Outcome           0
Flights           0
GridFins          0
Reused            0
Legs              0
LandingPad       24
Block             0
ReusedCount       0
Serial            0
Longitude         0
Latitude          0
dtype: int64
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 3: Dealing with missing values

- **Goal**: Identification and targeted handling of missing values in the Falcon 9 database to ensure data quality for subsequent modeling steps.

Why is this important?

- Missing values can lead to bias in the model.
- Some columns (e.g. LandingPad) systematically contain NaNs that must be interpreted in a domain-specific manner.
- The goal is to achieve a cleaned, consistent database.

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 3: Dealing with missing values

Technical focus:

- Identification with data_falcon9.isnull().sum().
- Visualization of affected rows with data_falcon9[data_falcon9.inull().any(axis=1)].
- Cleaning with targeted dropna() – with the exception of LandingPad, wich may deliberately retain none.

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 3: Dealing with missing values

- The screenshot displays the coding for Task 3.

Task 3: Dealing with Missing Values

Calculate below the mean for the `PayloadMass` using the `.mean()`. Then use the mean and the `.replace()` function to replace `np.nan` values in the data with the mean you calculated.

51]:
```
# Calculate the mean value of PayloadMass column
payload_mean = data_falcon9_cleaned['PayloadMass'].mean()
print("Mean Payload Mass:", payload_mean)

# Replace the np.nan values with its mean value
data_falcon9_cleaned.loc[:, 'PayloadMass'] = data_falcon9_cleaned['PayloadMass'].fillna(payload_mean)
```

```
Mean Payload Mass: 6123.547647058824
```

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

52]:
```
data_falcon9_cleaned['PayloadMass'].isnull().sum()
```

52]:
```
np.int64(0)
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.2 Part 1 – Lab 1: API access & data retrieval

TASK 3: Dealing with missing values

- The screenshot displays the coding for Task 3.

```
[61]: # Determine information json file
      import json
      print(json.dumps(data[0], indent=2))
```


```
{
  "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,
```


The following code of the block


```
  "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"
}
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.3 Part 2 – Lab 2: Web scraping historical launch data

Goal: To expand the database with publicly documented launch data from Wikipedia in order to supplement the API data and make the modeling more robust.

Why web scrapping?

- The SpaceX API does not contain all historical launch data.
- Wikipedia offers a structured HTML table with additional information.
- Web scraping enables automated extraction from publicly available sources.

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.3 Part 2 – Lab 2: Web scraping historical launch data

Technical focus:

- Accessing the “List of Falcon 9 and Falcon Heavy Launches” page.
- Parsing the HTML structure with BeautifulSoup.

Extracting:

- Date & time (date_time)
- Booster version (booster_version)
- Payload mass (get_mass)
- Landing status (landing_status)
- Column names (extract_column_from_header)

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.3 Part 2 – Lab 2: Web scraping historical launch data

Preview:

- The scraped data is stored in a Pandas DataFrame and later combined with the API data as the basis for feature preparation in Part 3.

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.3 Part 2 – Lab 2: Web scrapping historical launch data

Task Overview

Contents:

- TASK 1: Request Wiki page
- TASK 2: Extract column headers
- TASK 3: Parse launch data
- TASK 4: Clean and structure scarped data

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.3 Part 2 – Lab 2: Web scraping historical launch data

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

Goal: Automated access to the Wikipedia page “list of Falcon 9 and Falcon Heavy launches” to prepare the HTML.

Why this page?

- Contains additional historical launch data that is missing from the SpaceX API.
- Provides a tabular representation, ideal for parsing.
- Supplements the API data with date, booster version, payload mass, and landing status.

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.3 Part 2 – Lab 2: Web scraping historical launch data

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

Technical focus:

- Access via request.get(wiki_url)
- HTML parsing with BeuatifulSoup
- Identification of the relevant table (soup.find_all('table'))

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.3 Part 2 – Lab 2: Web scraping historical launch data

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

- The screenshot displays the coding for Task 1.

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.

[9]:
```
# use requests.get() method with the provided static_url
# assign the response to a object
response = requests.get(static_url)
```

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

[10]:
```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(response.text, 'html.parser')
```

Print the page title to verify if the `BeautifulSoup` object was created properly

[11]:
```
# Use soup.title attribute
print(soup.title.string)
```

```
List of Falcon 9 and Falcon Heavy launches - Wikipedia
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.3 Part 2 – Lab 2: Web scraping historical launch data

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

Goal: Identify and extract column names from the Wikipedia table in order to prepare the structure of the launch data for further processing.

Why is this important?

- The column headers define the data structure for subsequent parsing.
- The enable systemic assignment of content to the desired features.
- Clean extraction is a prerequisite for consistency in DataFrame construction.

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.3 Part 2 – Lab 2: Web scraping historical launch data

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

Technical focus:

- Accessing the table header with extract_column_from_header().
- Using BeautifulSoup to navigate within the HTML table.
- Storing column names for later extraction of launch data.

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.3 Part 2 – Lab 2: Web scraping historical launch data

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

- The following screenshot displays the coding for Task 2:
  - ➢ Output: Number of tables found on the page
  - ➢ Preparing for header extraction – table count check
  - ➢ Initial HTML parsing and table identification

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.3 Part 2 – Lab 2: Web scraping historical launch data

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

- The screenshot displays the coding for Task 2:

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

[13]:
```
# 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

# Retrieve HTML content
response = requests.get(static_url)
soup = BeautifulSoup(response.text, 'html.parser')

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

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

```
Tables found: 25
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.3 Part 2 – Lab 2: Web scraping historical launch data

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

- The screenshot displays the coding for Task 2 and the first line of the result:
- The remaining data of the result can be determined using JupyterLab in the appendix.

```
[14]: # Let's print the third table and check its content
first_launch_table = html_tables[2]
print(first_launch_table)
```

```
<table class="wikitable plainrowheaders collapsible" style="width: 100%;">
<tbody><tr>
<th scope="col">Flight No.
</th>
<th scope="col">Date and<br/>time (<a href="/wiki/Coordinated_Universal_Time" title="Coordinated Universal Time">UTC</a>)
</th>
<th scope="col"><a href="/wiki/List_of_Falcon_9_first-stage_boosters" title="List of Falcon 9 first-stage boosters">Version,<br/>Booster</a> <sup class="reference" id="cite_ref-booster_11-0"><a href="#cite_note-booster-11"><span class
="cite-bracket">[</span>b<span class="cite-bracket">]</span></a></sup>
</th>
<th scope="col">Launch site
</th>
<th scope="col">Payload<sup class="reference" id="cite_ref-Dragon_12-0"><a href="#cite_note-Dragon-12"><span class="cite-bracket">[</span>c<span class="cite-bracket">]</span></a></sup>
</th>
<th scope="col">Payload mass
</th>
<th scope="col">Orbit
</th>
<th scope="col">Customer
</th>
<th scope="col">Launch<br/>outcome
</th>
<th scope="col"><a href="/wiki/Falcon_9_first-stage_landing_tests" title="Falcon 9 first-stage landing tests">Booster<br/>landing</a>
</th></tr>
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.3 Part 2 – Lab 2: Web scraping historical launch data

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

- The screenshot displays the coding for Task 2:

Next, we just need to iterate through the `<th>` elements and apply the provided `extract_column_from_header()` to extract column name one by one

[15]:
```
column_names = []

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

# Extract all <th> elements from the table
table_headers = first_launch_table.find_all('th')

# Iterate over each <th> element
for th in table_headers:
    name = extract_column_from_header(th)
    if name is not None and len(name) > 0:
        column_names.append(name)
```

Check the extracted column names

[16]:
```
print(column_names)
```

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

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.3 Part 2 – Lab 2: Web scraping historical launch data

TASK 3: Create a data frame by parsing the launch HTML tables

Goal: Extraction of relevant launch data from the HTML rows of the Wikipedia table, based on the previously identified column names.

Why is this important?

- The table rows contain the actual launch information.
- Only through targeted parsing can the data be transferred into a DataFrame in a structured manner.
- The extracted values form the basis for later feature preparation and modeling.

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.3 Part 2 – Lab 2: Web scraping historical launch data

TASK 3: Create a data frame by parsing the launch HTML tables

Technical focus:

- Iteration over the <tr> elements of the table
- Application of extract_launch_data() to each row
- Storage of the results in a list of dictionaries
- Conversion to a Pandas DataFrame

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.3 Part 2 – Lab 2: Web scraping historical launch data

TASK 3: Create a data frame by parsing the launch HTML tables

- The screenshot displays the coding for creating the DataFrame for Task 3:

TASK 3: Create a data frame by parsing the launch HTML tables

We will create an empty dictionary with keys from the extracted column names in the previous task. Later, this dictionary will be converted into a Pandas dataframe

[17]:
```
launch_dict= dict.fromkeys(column_names)

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

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

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.3 Part 2 – Lab 2: Web scraping historical launch data

TASK 3: Create a data frame by parsing the launch HTML tables

- The screenshot displays the coding for creating the DataFrame for Task 3:

```
[23]: extracted_row = 0

for table_number, table in enumerate(soup.find_all('table', "wikitable plainrowheaders collapsible")):
    for rows in table.find_all("tr"):
        # Check if the row has a valid flight number
        if rows.th and rows.th.get_text(strip=True).isdigit():
            flight_number = rows.th.get_text(strip=True)
            flag = True
        else:
            flag = False

        row = rows.find_all('td')

        # Proceed only if it's a valid launch row with enough columns
        if flag and len(row) >= 9:
            extracted_row += 1

            # Flight Number
            launch_dict['Flight No.'].append(flight_number)

            # Date and Time
            try:
                datatimelist = date_time(row[0])
                date = datatimelist[0].strip(',') if datatimelist else None
                time = datatimelist[1] if len(datatimelist) > 1 else None
            except Exception as e:
                print(f"Error parsing date/time: {e}")
                date, time = None, None
            launch_dict['Date'].append(date)
            launch_dict['Time'].append(time)
```

The following code of the block

```
# Version Booster
try:
    bv = booster_version(row[1])
    if not bv and row[1].a:
        bv = row[1].a.get_text(strip=True)
    elif not bv:
        bv = row[1].get_text(strip=True)
except Exception as e:
    print(f"Error parsing booster version: {e}")
    bv = None
launch_dict['Version Booster'].append(bv)

# Launch Site
try:
    launch_site = row[2].a.get_text(strip=True) if row[2].a else row[2].get_text(strip=True)
except Exception as e:
    print(f"Error parsing launch site: {e}")
    launch_site = None
launch_dict['Launch site'].append(launch_site)

# Payload
try:
    payload = row[3].a.get_text(strip=True) if row[3].a else row[3].get_text(strip=True)
except Exception as e:
    print(f"Error parsing payload: {e}")
    payload = None
launch_dict['Payload'].append(payload)

# Payload Mass
try:
    payload_mass = get_mass(row[4])
except Exception as e:
    print(f"Error parsing payload mass: {e}")
    payload_mass = None
launch_dict['Payload mass'].append(payload_mass)
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.3 Part 2 – Lab 2: Web scraping historical launch data

TASK 3: Create a data frame by parsing the launch HTML tables

- The screenshot displays the coding for creating the DataFrame for Task 3:

```
            # Orbit
            try:
                orbit = row[5].a.get_text(strip=True) if row[5].a else row[5].get_text(strip=True)
            except Exception as e:
                print(f"Error parsing orbit: {e}")
                orbit = None
            launch_dict['Orbit'].append(orbit)

            # Customer
            try:
                customer = row[6].a.get_text(strip=True) if row[6].a else row[6].get_text(strip=True)
            except Exception as e:
                print(f"Error parsing customer: {e}")
                customer = None
            launch_dict['Customer'].append(customer)

            # Launch Outcome
            try:
                launch_outcome = list(row[7].strings)[0].strip()
            except Exception as e:
                print(f"Error parsing launch outcome: {e}")
                launch_outcome = None
            launch_dict['Launch outcome'].append(launch_outcome)

            # Booster Landing
            try:
                booster_landing = landing_status(row[8])
            except Exception as e:
                print(f"Error parsing booster landing: {e}")
                booster_landing = None
            launch_dict['Booster landing'].append(booster_landing)
```

```
[26]: # Show result
      print(booster_landing)
```

```
Success
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.3 Part 2 – Lab 2: Web scraping historical launch data

TASK 3: Create a data frame by parsing the launch HTML tables

- The screenshot displays the coding for creating the DataFrame for Task 3:

```
[27]: # Total number of extracted launches
      print(f"Total launches parsed: {len(launch_dict['Flight No.'])}")

      Total launches parsed: 363

      After you have fill in the parsed launch record values into launch_dict , you can create a dataframe from it.

[28]: # df= pd.DataFrame({ key:pd.Series(value) for key, value in launch_dict.items() })

[30]: # Create DataFrame
      df = pd.DataFrame(launch_dict)
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.3 Part 2 – Lab 2: Web scraping historical launch data

TASK 3: Create a data frame by parsing the launch HTML tables

- The screenshot displays the coding for creating the DataFrame for Task 3:

```
[31]: # Show result
print(df.shape)
print(df.head())
```

```
(363, 11)
  Flight No. Launch site                                Payload Payload mass  \
0          1       CCAFS  Dragon Spacecraft Qualification Unit            0
1          2       CCAFS                               Dragon            0
2          3       CCAFS                               Dragon       525 kg
3          4       CCAFS                         SpaceX CRS-1     4,700 kg
4          5       CCAFS                         SpaceX CRS-2     4,877 kg

  Orbit Customer Launch outcome   Version Booster Booster landing  \
0   LEO   SpaceX        Success  F9 v1.07B0003.18         Failure
1   LEO     NASA        Success  F9 v1.07B0004.18         Failure
2   LEO     NASA        Success  F9 v1.07B0005.18    No attempt\n
3   LEO     NASA        Success  F9 v1.07B0006.18      No attempt
4   LEO     NASA        Success  F9 v1.07B0007.18    No attempt\n

              Date   Time
0     4 June 2010  18:45
1  8 December 2010  15:43
2      22 May 2012  07:44
3   8 October 2012  00:35
4     1 March 2013  15:10
```

We can now export it to a **CSV** for the next section, but to make the answers consistent and in case you have difficulties finishing this lab.

Following labs will be using a provided dataset to make each lab independent.

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

```
[32]: # Export data to csv
df.to_csv('spacex_web_scraped.csv', index=False)

# Show result
print("CSV export successfully saved!")
```

```
CSV export successfully saved!
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.4 Part 3 – Lab 3: Exploratory analysis & label creation

Preparatory steps:

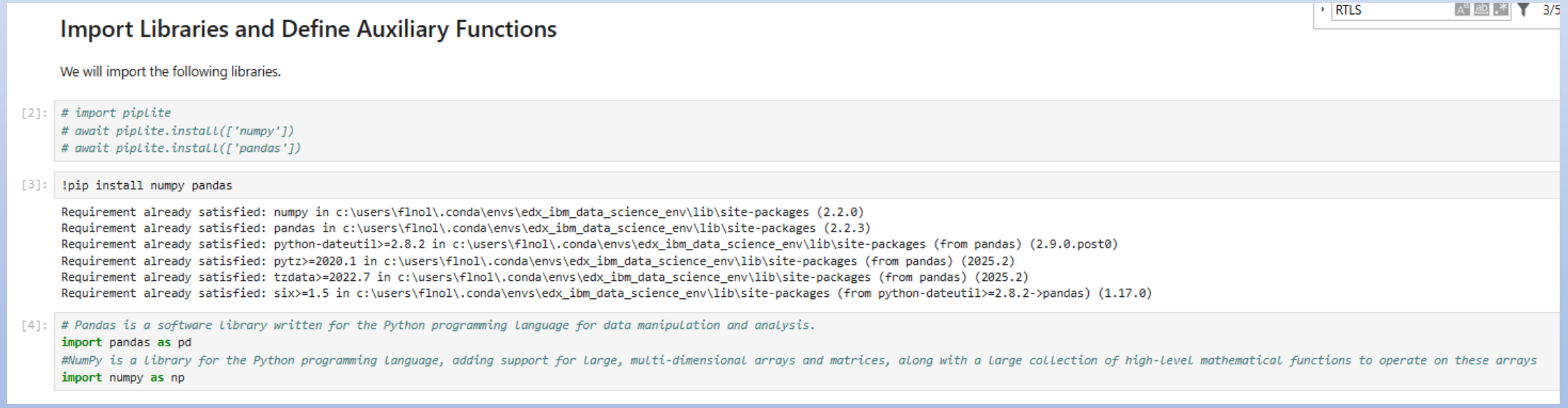
- Import libraries (numpy, pandas, request, io)
- Load CSV data set from the cloud
- Display first rows with df.head() and df.head(10)
- Check for missing values (df.isnull().sum()/df.shape[0]*100)
- Data type check (df.dtypes)

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.4 Part 3 – Lab 3: Exploratory analysis & label creation

Preparatory steps:

- The screenshot displays the coding for the preparatory steps:



Import Libraries and Define Auxiliary Functions

We will import the following libraries.

```
[2]: # import piplite
     # await piplite.install(['numpy'])
     # await piplite.install(['pandas'])

[3]: !pip install numpy pandas
```

```
Requirement already satisfied: numpy in c:\users\flnol\.conda\envs\edx_ibm_data_science_env\lib\site-packages (2.2.0)
Requirement already satisfied: pandas in c:\users\flnol\.conda\envs\edx_ibm_data_science_env\lib\site-packages (2.2.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\flnol\.conda\envs\edx_ibm_data_science_env\lib\site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\flnol\.conda\envs\edx_ibm_data_science_env\lib\site-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\flnol\.conda\envs\edx_ibm_data_science_env\lib\site-packages (from pandas) (2025.2)
Requirement already satisfied: six>=1.5 in c:\users\flnol\.conda\envs\edx_ibm_data_science_env\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
```

```
[4]: # Pandas is a software library written for the Python programming language for data manipulation and analysis.
     import pandas as pd
     #NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays
     import numpy as np
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.4 Part 3 – Lab 3: Exploratory analysis & label creation

Preparatory steps:

- The screenshot displays the coding for the preparatory steps:

Data Analysis

[12]:
```
import requests
import io
import pandas as pd

URL = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_1.csv'
response = requests.get(URL)

# Save the CSV data in a variable
dataset_part_1_csv = io.StringIO(response.text)

# Load into a pandas DataFrame
df = pd.read_csv(dataset_part_1_csv)

df.head()
```

[12]:

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

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.4 Part 3 – Lab 3: Exploratory analysis & label creation

Preparatory steps:

- The screenshot displays the coding for the preparatory steps:

Load Space X dataset, from last section.

[17]:
```
# Reset the cursor back to the beginning
# A reset is only necessary if the object is used again.
dataset_part_1_csv.seek(0)

# Load into a pandas DataFrame
df = pd.read_csv(dataset_part_1_csv)

df.head(10)
```

[17]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577366	28.561857
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366	28.561857
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.577366	28.561857
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	0	B1003	-120.610829	34.632093
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561857
5	6	2014-01-06	Falcon 9	3325.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1005	-80.577366	28.561857
6	7	2014-04-18	Falcon 9	2296.000000	ISS	CCAFS SLC 40	True Ocean	1	False	False	True	NaN	1.0	0	B1006	-80.577366	28.561857
7	8	2014-07-14	Falcon 9	1316.000000	LEO	CCAFS SLC 40	True Ocean	1	False	False	True	NaN	1.0	0	B1007	-80.577366	28.561857
8	9	2014-08-05	Falcon 9	4535.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1008	-80.577366	28.561857
9	10	2014-09-07	Falcon 9	4428.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1011	-80.577366	28.561857

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.4 Part 3 – Lab 3: Exploratory analysis & label creation

Preparatory steps:

- The screenshot displays the coding for the preparatory steps:

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

```
[18]: df.isnull().sum()/df.shape[0]*100
```

```
[18]: FlightNumber       0.000000
      Date               0.000000
      BoosterVersion     0.000000
      PayloadMass        0.000000
      Orbit              0.000000
      LaunchSite         0.000000
      Outcome            0.000000
      Flights            0.000000
      GridFins           0.000000
      Reused             0.000000
      Legs               0.000000
      LandingPad        28.888889
      Block              0.000000
      ReusedCount        0.000000
      Serial             0.000000
      Longitude          0.000000
      Latitude           0.000000
      dtype: float64
```

```
[19]: df.dtypes
```

```
[19]: FlightNumber      int64
      Date             object
      BoosterVersion   object
      PayloadMass     float64
      Orbit            object
      LaunchSite       object
      Outcome          object
      Flights           int64
      GridFins           bool
      Reused             bool
      Legs               bool
      LandingPad       object
      Block           float64
      ReusedCount       int64
      Serial           object
      Longitude       float64
      Latitude        float64
      dtype: object
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.4 Part 3 – Lab 3: Exploratory analysis & label creation

TASK 1: Calculate the number of launches on each site

- Aggregation with df['LaunchSite'].value_counts()
- Result: CCAFS SLC 40 (55), KSC LC 39A (22), VAFB SLC 4E (13)
- The screenshot displays the coding and results for Task 1:

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:

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

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

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.4 Part 3 – Lab 3: Exploratory analysis & label creation

TASK 1: Calculate the number of launches on each site

- The screenshot displays a description of the orbit types:

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

- **LEO**: Low Earth orbit (LEO)is an Earth-centred orbit with an altitude of 2,000 km (1,200 mi) or less (approximately one-third of the radius of Earth),[1] or with at least 11.25 periods per day (an orbital period of 128 minutes or less) and an eccentricity less than 0.25.[2] Most of the manmade objects in outer space are in LEO [1].
- **VLEO**: Very Low Earth Orbits (VLEO) can be defined as the orbits with a mean altitude below 450 km. Operating in these orbits can provide a number of benefits to Earth observation spacecraft as the spacecraft operates closer to the observation[2].
- **GTO**(Geostationary Transfer Orbit): A geostationary transfer orbit is an elliptical Earth orbit used to transfer satellites from low Earth orbit (LEO) to geostationary orbit (GEO). In a GTO, the perigee (closest point to Earth) is much lower than GEO altitude, while the apogee (farthest point) reaches approximately 22,236 miles (35,786 kilometers) above Earth's equator — the altitude of a geostationary orbit. Satellites in GTO use onboard propulsion to circularize their orbit at GEO altitude, where they can provide services such as weather monitoring, communications, and surveillance.[3] .
- **SSO (or SO)**: It is a Sun-synchronous orbit also called a heliosynchronous orbit is a nearly polar orbit around a planet, in which the satellite passes over any given point of the planet's surface at the same local mean solar time [4] .
- **ES-L1** :At the Lagrange points the gravitational forces of the two large bodies cancel out in such a way that a small object placed in orbit there is in equilibrium relative to the center of mass of the large bodies. L1 is one such point between the sun and the earth [5] .
- **HEO** A highly elliptical orbit, is an elliptic orbit with high eccentricity, usually referring to one around Earth [6].
- **ISS** A modular space station (habitable artificial satellite) in low Earth orbit. It is a multinational collaborative project between five participating space agencies: NASA (United States), Roscosmos (Russia), JAXA (Japan), ESA (Europe), and CSA (Canada) [7]
- **MEO** Geocentric orbits ranging in altitude from 2,000 km (1,200 mi) to just below geosynchronous orbit at 35,786 kilometers (22,236 mi). Also known as an intermediate circular orbit. These are "most commonly at 20,200 kilometers (12,600 mi), or 20,650 kilometers (12,830 mi), with an orbital period of 12 hours [8]
- **HEO** Geocentric orbits above the altitude of geosynchronous orbit (35,786 km or 22,236 mi) [9]
- **GEO** It is a circular geosynchronous orbit 35,786 kilometres (22,236 miles) above Earth's equator and following the direction of Earth's rotation [10]
- **PO** It is one type of satellites in which a satellite passes above or nearly above both poles of the body being orbited (usually a planet such as the Earth [11]

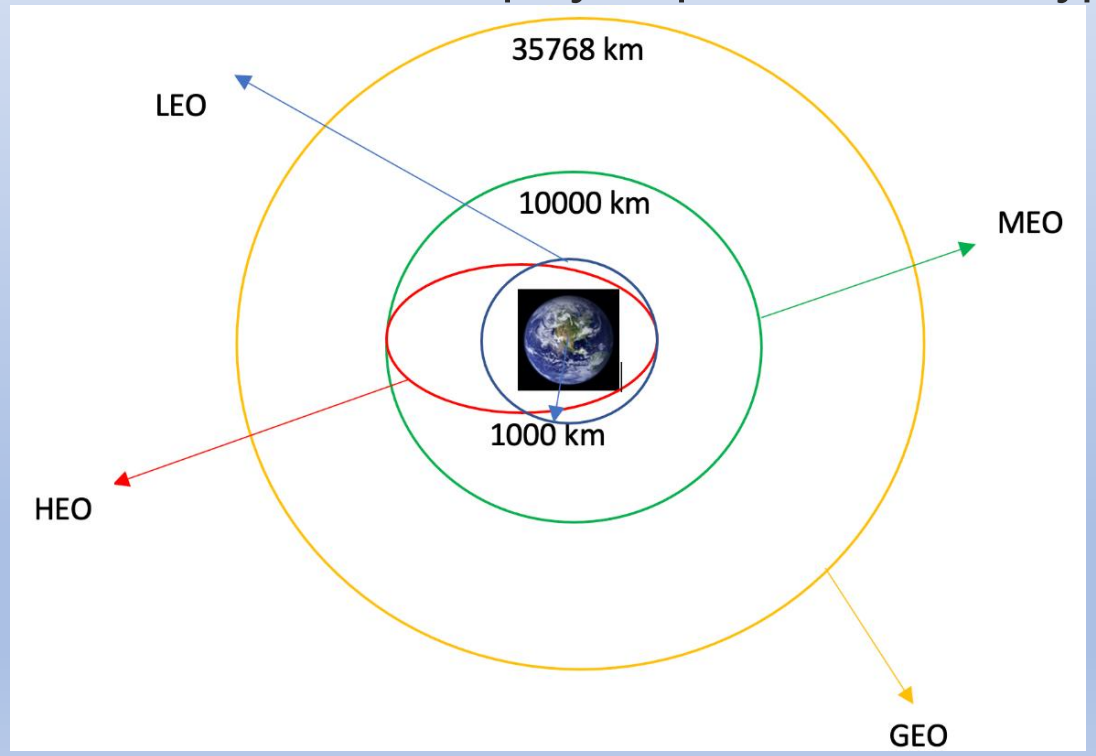
some are shown in the following plot:

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.4 Part 3 – Lab 3: Exploratory analysis & label creation

TASK 1: Calculate the number of launches on each site

- The screenshot displays a plot with orbits types:



3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.4 Part 3 – Lab 3: Exploratory analysis & label creation

TASK 2: Calculate the number and occurrence of each orbit

- Exclusion of GTO (df[df['Orbit'] != 'GTO'])
- Aggregation with value_counts() on orbit
- Result: ISS, VLEO, PO, LEO, SSO, MEO, HEO, ES-L1, SO, GEO

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.4 Part 3 – Lab 3: Exploratory analysis & label creation

TASK 2: Calculate the number and occurrence of each orbit

- The screenshot displays the coding and results for Task 2:

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`

Note: Do not count GTO, as it is a transfer orbit and not itself geostationary.

[21]:
```
# Apply value_counts on Orbit column
# Exclude GTO from the analysis
filtered_df = df[df['Orbit'] != 'GTO']

# Count occurrences of each orbit
orbit_counts = filtered_df['Orbit'].value_counts()

# Display the result
print(orbit_counts)
```

```
Orbit
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
```

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.4 Part 3 – Lab 3: Exploratory analysis & label creation

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

- Aggregation with df['Outcome'].value_counts()
- Definition of bad_outcomes as a set of failed landings.
- Task 3 defined the outcomes that are not considered successful.

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.4 Part 3 – Lab 3: Exploratory analysis & label creation

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

- The screenshot displays the coding and results for Task 3:

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.

2]:
```
# landing_outcomes = values on Outcome column
# Count the number of each landing outcome
landing_outcomes = df['Outcome'].value_counts()

# Display the result
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 re[Kein Titel]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.

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.4 Part 3 – Lab 3: Exploratory analysis & label creation

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

- The screenshot displays the coding and results for Task 3:

```
[23]: 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:

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

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

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.4 Part 3 – Lab 3: Exploratory analysis & label creation

TASK 4: Create a landing outcome label from outcome column

- Create landing_class with 1 for successful landings and 0 for failed landings.
- Assign to new column Class.
- Validate with df[“Class”].mean() → shows success rate.
- Export with df.to_csv(“dataset_part_2.csv”, index=False).
- The Class column defined in Task 4 serves as the target variable for the following model.

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.4 Part 3 – Lab 3: Exploratory analysis & label creation

TASK 4: Create a landing outcome label from outcome column

- The screenshot displays the coding and results for Task 4:

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`:

[26]:
```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
# Define bad outcomes
bad_outcome = {'Failure', 'False Landing', 'Crash', 'No Attempt'}

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

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

[27]:
```
# Assign landing_class to a new column 'Class'
df['Class']=landing_class

# Display the first 8 entries of the new classification column
df[['Class']].head(8)
```

[27]:

	Class
0	1
1	1
2	1
3	1
4	1
5	1
6	1
7	1

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.4 Part 3 – Lab 3: Exploratory analysis & label creation

TASK 4: Create a landing outcome label from outcome column

- The screenshot displays the coding and results for Task 4:

```
[28]: df.head(5)
```

[28]:

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

[Kein Titel]

3. Section 1 – Methodology: Data access, cleaning, and feature preparation

3.4 Part 3 – Lab 3: Exploratory analysis & label creation

TASK 4: Create a landing outcome label from outcome column

- The screenshot displays the coding and results for Task 4:

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

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

```
[29]: np.float64(1.0)
```

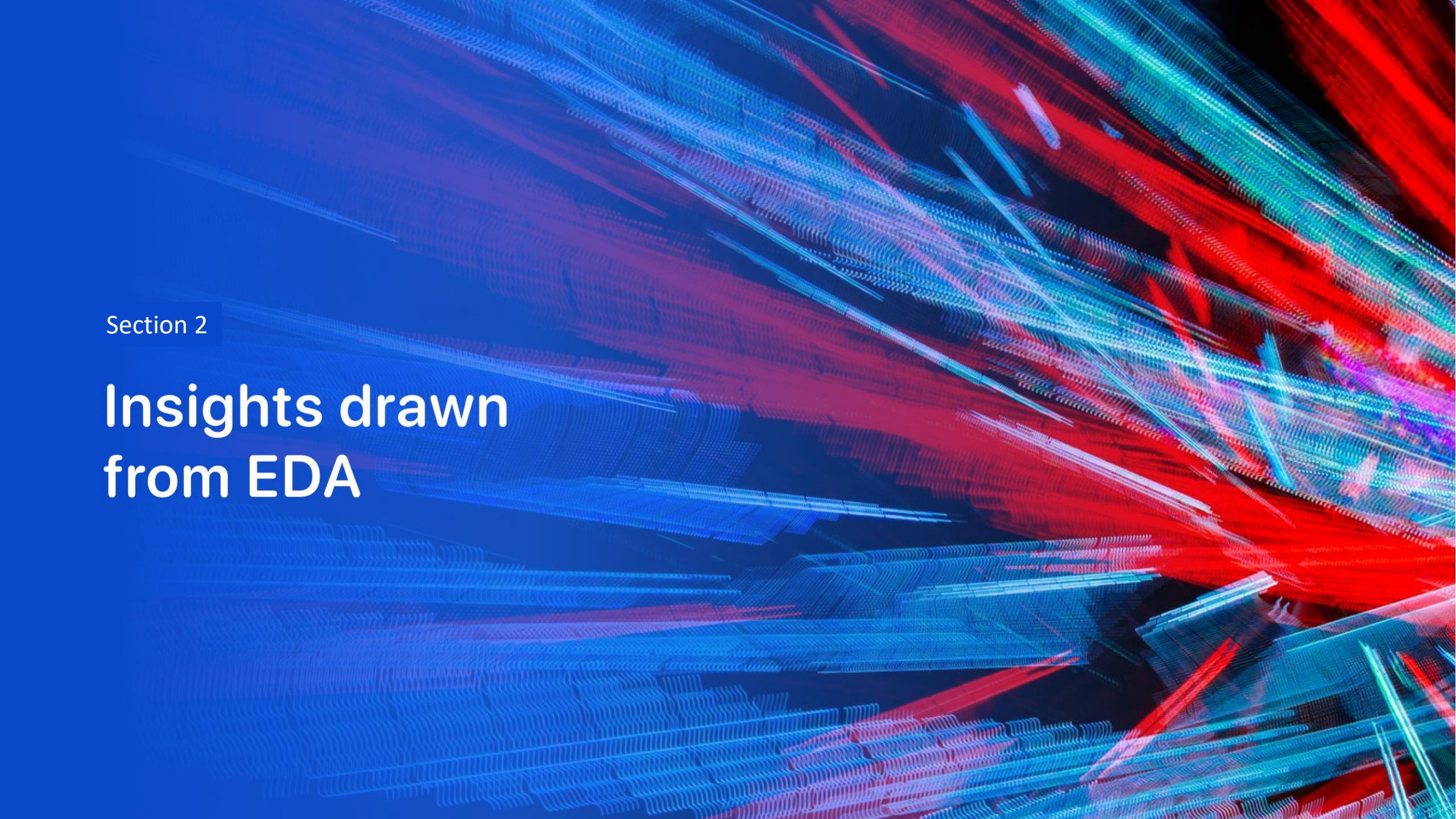
We can now export it to a CSV for the next section,but to make the answers consistent, in the next lab we will provide data in a pre-selected date range.

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

```
[58]: df.to_csv("dataset_part_2.csv", index=False)
```

```
[33]: # Graded Quiz: Data Wrangling
      # Question 1
      # How many launches came from CCAFS SLC 40?
      df['LaunchSite'].value_counts()['CCAFS SLC 40']
```

```
[33]: np.int64(55)
```


Section 2
Insights drawn from EDA

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.1 The section 2 describes the following topics:

Goal: Context and objectives

- As part of the exploratory data analysis, SpaceX mission data was systematically examined to identify patterns, outliers, and strategically relevant variables.
- The analysis was carried out in a structured manner using ten SQL-based tasks in Jupyter Notebook, focusing on launch sites, booster versions, payload masses, and landing results.

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.1 The section 2 describes the following topics:

EDA via SQL – Task Overview

Contents:

- 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 'KSC'
- TASK 3-4: Payload mass according to NASA (CRS) and booster version
- TASK 5-6: Successful landings with specific conditions
- TASK 7-10: Success/failure rates, maximum payload, temporal patterns

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.1 EDA via SQL – Geographical & technical patterns

Contents:

- Focus on task 1-4
- SQL extracts (e.g. SELECT DISTINCT Launch_Site)
- Findings:
  - Focus on a few sites
  - F9 v1.1 shows average payload performance
  - NASA (CRS) with high payload volume

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.1 EDA via SQL – Success factors & booster insights contents

Contents:

- Focus on task 5-8
- Landing results & booster performance
- Successful landings on drone ship and ground pad
- Booster versions with maximum payload
- Findings:
  - ➢ Successful landings correlate with certain booster
  - ➢ Payload mass influences landing result

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.1 EDA via SQL – Temporal patterns & outcomes ranking contents

Contents:

- Focus on task 9-10
- Temporal patterns & ranking
- Monthly success distribution in 2017
- Ranking of landing results (success/failure)
- Findings:
  - Success clusters in certain months
  - Ground pad dominates successful landings

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.1 Insights for visual analytics tasks

Strategic conclusions:

- SQL-based EDA provided the basis for structuring the visualization tasks in section 3. It helped to identify relevant variables, recognize geographic clusters, and derive hypotheses about the probability of success.

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.2 TASK 1: Display the names of the unique launch sites in the space mission

- After installing the necessary libraries, the **dataset_part_3.csv** file was successfully loaded into the Jupyter Notebook. Next, a connection was established to the SQLite database **my_data1.db**. This laid the foundation for the structured analysis of the SpaceX mission data, starting with Task 1, which aimed to identify all unique launch sites.
- The screenshot displays the connecting to the database.

Connect to the database

Let us first load the SQL extension and establish a connection with the database

[2]:
```
%load_ext sql
```

[3]:
```
import csv, sqlite3

con = sqlite3.connect("my_data1.db")
cur = con.cursor()
```

[4]:
```
%sql sqlite:///my_data1.db
```

[5]:
```
import pandas as pd
df = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/labs/module_2/data/Spacex.csv")
df.to_sql("SPACEXTBL", con, if_exists='replace', index=False,method="multi")
```

[5]:
```
101
```

Note:This below code is added to remove blank rows from table

[6]:
```
%sql create table SPACEXTABLE as select * from SPACEXTBL where Date is not null
```

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.2 TASK 1: Display the names of the unique launch sites in the space mission

- The screenshot displays the SQL statement for determining the launch site.

Task 1

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

```
[8]: %sql SELECT DISTINCT "Launch_Site" FROM SPACEXTABLE;
```

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

[8]:

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

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.2 TASK 2: **Display 5 records where launch sites begin with the string 'KSC'**

- The query filters specifically for launch sites at Kennedy Space Center (KSC).
- The results show that KSC LC-39A is stored multiple times in database.
- This site is historically significant and technically highly frequented.

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.2 TASK 2: **Display 5 records where launch sites begin with the string 'KSC'**

- The screenshot displays 5 records where launch sites begin with the string 'KSC'.

Task 2

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

```
[9]: %sql SELECT * FROM SPACEXTABLE WHERE "Launch_Site" LIKE 'KSC%' LIMIT 5;
```

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

[9]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2017-02-19	14:39:00	F9 FT B1031.1	KSC LC-39A	SpaceX CRS-10	2490	LEO (ISS)	NASA (CRS)	Success	Success (ground pad)
2017-03-16	6:00:00	F9 FT B1030	KSC LC-39A	EchoStar 23	5600	GTO	EchoStar	Success	No attempt
2017-03-30	22:27:00	F9 FT B1021.2	KSC LC-39A	SES-10	5300	GTO	SES	Success	Success (drone ship)
2017-05-01	11:15:00	F9 FT B1032.1	KSC LC-39A	NROL-76	5300	LEO	NRO	Success	Success (ground pad)
2017-05-15	23:21:00	F9 FT B1034	KSC LC-39A	Inmarsat-5 F4	6070	GTO	Inmarsat	Success	No attempt

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.2 TASK 3: **Display the total payload mass carried by boosters launched by NASA (CRS)**

- The calculation for the total mass of payloads transported by boosters for the customer NASA (KSC) was performed.

Result:

- Total payload mass: 48213 kg
- The query filters specifically for the customer field “NASA (KSC)” and totals the corresponding payload masses.

Finding:

- NASA (CRS) has transported a total payload of 48213 kg across multiple missions.
  This figure is relevant for evaluating customer activity, booster utilization, and the strategic importance of individual missions.

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.2 TASK 3: **Display the total payload mass carried by boosters launched by NASA (CRS)**

- The screenshot displays the analyzed total payload mass.

Task 3

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

[10]:
```
# Determine column names
%sql SELECT * FROM SPACEXTABLE LIMIT 1;
```

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

[10]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)

[39]:
```
# Display the total payload
%sql SELECT SUM("PAYLOAD_MASS__KG_") AS Total_Payload_KG FROM SPACEXTABLE WHERE "Customer" LIKE '%NASA (CRS)%';
```

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

[39]:

Total_Payload_KG
48213

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.2 TASK 4: **Display average payload mass carried by booster version F9 v1.1**

- Calculation of the average payload mass transported by the F9 v1.1 booster version.

Result:

- Average payload mass: 2928,4 kg.

Finding:

- The F9 v1.1 booster version carried an average payload of around 2.9 tons per mission. This figure is relevant for evaluating the technical performance of earlier booster generations and serves as a basis for comparison with later versions.

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.2 TASK 4: **Display average payload mass carried by booster version F9 v1.1**

- The screenshot displays the analyzed average payload mass.

Task 4

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

```
[13]: %sql SELECT AVG("PAYLOAD_MASS__KG_") AS Average_Payload_KG FROM SPACEXTABLE WHERE "Booster_Version" = 'F9 v1.1';
 * sqlite:///my_data1.db
Done.
```

[13]:

Average_Payload_KG
2928.4

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.2 TASK 5: List the date where the successful landing outcome in drone ship was archeived

- Determination of the date of the first successful landing on a drone ship.

Result:

- First success date: 2016-04-08

Finding:

- On April 8, 2016, SpaceX achieved the first successful landing of a Falcon 9 stage on an autonomous drone ship. This milestone marks the beginning of systematic reusability and is a key turning point in the cost optimization of space missions.

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.2 TASK 5: List the date where the successful landing outcome in drone ship was archeived

- The screenshot displays the analyzed first success date.

Task 5

List the date where the succesful landing outcome in drone ship was acheived.

Hint:Use min function

[37]:
```
%%sql SELECT MIN("Date") AS First_Successful_Drone_Landing
FROM SPACEXTABLE
WHERE "Landing_Outcome" = 'Success (drone ship)';
```

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

[37]:

First_Successful_Drone_Landing
2016-04-08

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.2 TASK 6: **List names of the boosters which have success in ground pad**

- List of booster versions that landed successfully on the ground pad and carried a payload mass between 4000 kg and 6000 kg.

Result:

- Identified booster versions:
  - F9 FT B1020.1
  - F9 B4 B1040.1
  - F9 B4 B1043.1

Finding:

- These booster versions demonstrate a combination of high payload performance and successful landing on the ground pad. They are potentially particularly efficient and technically reliable – an indication of optimized mission profiles.

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.2 TASK 6: List names of the boosters which have success in ground pad

- The screenshot displays the analyzed boosters names which have success in ground pad.

Task 6

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

[41]:
```
%%sql SELECT DISTINCT "Booster_Version"
FROM SPACEXTABLE
WHERE "Landing_Outcome" = 'Success (ground pad)'
  AND "PAYLOAD_MASS__KG_" > 4000
  AND "PAYLOAD_MASS__KG_" < 6000;
```

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

[41]:

Booster_Version
F9 FT B1032.1
F9 B4 B1040.1
F9 B4 B1043.1

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.2 TASK 7: List the total number of successful and failure mission outcomes

- List of the total number of successful and failed missions.

Result:

- The identified results of the missions.
  - Success: 98
  - Success (payload status unclear): 1
  - Failure (in flight): 1
  - Success: 1

Finding:

- The results show a clear dominance of successful missions. Individual deviating entries such as duplicate 'Success' labels or 'payload status unclear' indicate potential inconsistencies in the data structure – an aspect that should be taken into account in further analysis.

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.2 TASK 7: List the total number of successful and failure mission outcomes

- The screenshot displays the analyzed total number of successful and failure missions.

Task 7

List the total number of successful and failure mission outcomes

```
[33]: %%sql SELECT "Mission_Outcome", COUNT(*) AS Total
      FROM SPACEXTABLE
      GROUP BY "Mission_Outcome";
```

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

[33]:

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

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.2 TASK 8: List all the booster_versions that have carried the maximum payload mass.

- List of the total number of successful and failed missions.

Result:

- Maximum payload mass: implicitly determined by subquery.
- Booster versions with this mass:
  - F9 B5 B1048.4
  - F9 B5 B1049.4
  - F9 B5 B1051.4
  - F9 B5 B1056.4
  - F9 B5 B1048.5
  - F9 B5 B1049.5
  - F9 B5 B1051.5
  - F9 B5 B1056.5
  - F9 B5 B1060.2
  - F9 B5 B1060.3

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.2 TASK 8: **List all the booster_versions that have carried the maximum payload mass.**

- List of the total number of successful and failed missions.

Finding:

- The results show a clear dominance of successful missions. Individual deviating entries such as duplicate ‘Success’ labels or ‘payload status unclear’ indicate potential inconsistencies in the data structure – an aspect that should be taken into account in further analysis.

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.2 TASK 8: List all the booster_versions that have carried the maximum payload mass.

- The screenshot displays the analyzed booster_versions with maximum payload mass.

Task 8

List all the booster_versions that have carried the maximum payload mass. Use a subquery.

[32]:
```
%%sql SELECT Booster_Version
FROM SPACEXTBL
WHERE PAYLOAD_MASS__KG_ = (
    SELECT MAX(PAYLOAD_MASS__KG_)
    FROM SPACEXTBL
);
```

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

[32]:

Booster_Version
F9 B5 B1048.4
F9 B5 B1049.4
F9 B5 B1051.3
F9 B5 B1056.4
F9 B5 B1048.5
F9 B5 B1051.4
F9 B5 B1049.5
F9 B5 B1060.2
F9 B5 B1058.3
F9 B5 B1051.6
F9 B5 B1060.3
F9 B5 B1049.7

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.2 TASK 9: List of successful ground pad landings in 2017

- Show all successful missions with ground pad landing from 2017 – including month name, booster version, and launch site.

Result:

Month_Name	Booster_Version	Launch_Site	Landing_Outcome
February	F9 FT B1031.1	KSC LC-39A	Success (ground pad)
May	F9 FT B1032.1	KSC LC-39A	Success (ground pad)
June	F9 FT B1035.1	KSC LC-39A	Success (ground pad)
August	F9 B4 B1039.1	KSC LC-39A	Success (ground pad)
September	F9 B4 B1040.1	KSC LC-39A	Success (ground pad)
December	F9 FT B1035.2	CCAFS SLC-40	Success (ground pad)

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.2 TASK 9: **List of successful ground pad landings in 2017**

- Show all successful missions with ground pad landing from 2017 – including month name, booster version, and launch site.

Finding:

- In 2017, six successful ground pad landings were carried out – spread over four months and two launch sites. Particularly noteworthy: **KSC LC-39A** was involved in five of these landings, underscoring its central role in the SpaceX program.

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.2 TASK 9: List of successful ground pad landings in 2017

- The screenshot displays the coding for the analysis.

Task 9

List the records which will display the month names, succesful landing_outcomes in ground pad ,booster versions, launch_site for the months in year 2017

Note: SQLLite does not support monthnames. So you need to use substr(Date,6,2) for month, substr(Date,9,2) for date, substr(Date,0,5),='2017' for year.

[31]:
```
%%sql
SELECT
    CASE substr(Date, 6, 2)
        WHEN '01' THEN 'January'
        WHEN '02' THEN 'February'
        WHEN '03' THEN 'March'
        WHEN '04' THEN 'April'
        WHEN '05' THEN 'May'
        WHEN '06' THEN 'June'
        WHEN '07' THEN 'July'
        WHEN '08' THEN 'August'
        WHEN '09' THEN 'September'
        WHEN '10' THEN 'October'
        WHEN '11' THEN 'November'
        WHEN '12' THEN 'December'
    END AS Month_Name,
    Booster_Version,
    Launch_Site,
    Landing_Outcome
FROM SPACEXTBL
WHERE substr(Date, 1, 4) = '2017'
  AND Landing_Outcome LIKE '%ground pad%'
  AND Mission_Outcome LIKE 'Success%';
```

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

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.2 TASK 10: **Rank the count of landing outcomes**

- Creation of a ranking list of landing outcomes between June 4, 2010, and March 20, 2017—sorted by frequency.

Result:

Landing Outcome	Outcome count
No attempt	10
Success (drone ship)	5
Failure (drone ship)	5
Success (ground pad)	3
Controlled (ocean)	3
Uncontrolled (ocean)	2
Failure (parachute)	2
Predicted (drone ship)	1

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.2 TASK 10: **Rank the count of landing outcomes**

- Creation of a ranking list of landing outcomes between June 4, 2010, and March 20, 2017—sorted by frequency.

Finding:

- Between 2010 and early 2017, there were a total of **29 documented** landings.
- The most common category was **'No attempt'**, suggesting early mission phases or a lack of recovery infrastructure.
- The **success rate for drone ship landings** is remarkable – as is the variety of outcome types.

4. Section 2 – Exploratory Data Analysis (EDA) Part 1

4.2 TASK 10: **Rank the count of landing outcomes**

- The screenshot displays the coding for the analysis.

Task 10

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

42]:
```
%%sql
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;
```

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

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.3 The section 2 describes the following topics:

Goal: Context and objectives

- Building on the SQL-based findings from Part 1, we will now perform a visual and algorithm-based analysis of the Falcon 9 mission data.
- The aim is to identify **relevant patterns, correlations, and influencing factors** that affect the **success of the first stage landing**.

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.3 The section 2 describes the following topics:

Methodology:

- Visual exploration with Seaborn, Matplotlib, and Pandas.
- Preparation of data for machine learning through:
  - Feature engineering
  - Dummy variables
  - Type conversion

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.3 The section 2 describes the following topics:

EDA via Barplot, Catplot & Scatterplot – Task Overview

Contents:

- TASK 1: Visualize the relationship between flight number and launch site
- TASK 2: Visualize the relationship between payload mass 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

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.3 The section 2 describes the following topics:

EDA via Barplot, Catplot & Scatterplot – Task Overview

Contents:

- TASK 5: Visualize the relationship between payload mass 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

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.3 The section 2 describes the following topics:

EDA via Catplot – Flight number vs. launch site

Contents:

- Focus on task 1
- Visualizations: sns.catplot(x='FlightNumber', y='LaunchSite')
- Findings:
  - Detection of whether certain launch sites were preferred for early or late flights.

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.3 The section 2 describes the following topics:

EDA via Scatterplot – Payload mass vs. launch site

Contents:

- Focus on task 2
- Visualizations: sns.scatterplot(x='PayloadMass', y='LaunchSite')
- Findings:
  - Identification into whether certain launch sites correlate with higher payloads.

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.3 The section 2 describes the following topics:

EDA via Barplot – Success rate by orbit type

Contents:

- Focus on task 3
- Visualizations: sns.barplot(x='PayloadMass', y='LaunchSite')
- Findings:
  - Identification which orbit types are associated with higher success rates.

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.3 The section 2 describes the following topics:

EDA via Scatterplot – FlightNumber vs. orbit type

Contents:

- Focus on task 4
- Visualizations: sns.scatterplot(x='FlightNumber', y='Orbit')
- Findings:
  - Detecting whether certain orbit types occur more frequently in early or late missions.

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.3 The section 2 describes the following topics:

EDA via Scatterplot – Payload mass vs. orbit type

Contents:

- Focus on task 5
- Visualizations: sns.scatterplot(x='FlightNumber', y='Orbit')
- Findings:
  - Identification whether certain orbit types are associated with higher payloads.

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.3 The section 2 describes the following topics:

Launch success trend per year

Contents:

- Focus on task 6
- Visualizations: df['Year'] = pd.DatetimeIndex(df['Date']).year → then groupby('Year')['Outcome'].mean()
- Plot: Line plot displaying success rate per year.
- Findings:
  - ➢ Displayed how the success rate has developed over the years.

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.3 The section 2 describes the following topics:

Dummy variables for categorical columns

Contents:

- Focus on task 7
- Visualizations: df['Year'] = pd.DatetimeIndex(df['Date']).year → then groupby('Year')['Outcome'].mean()
- Plot: Line plot displaying success rate per year.
- Goal:
  - ➢ Preparation for machine learning
- Note:
  - ➢ Dummy variables increase model compatibility

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.3 The section 2 describes the following topics:

Cast numeric columns to float 64

Contents:

- Focus on task 8
- Methode: df[numerical_columns] = df[numerical_columns].astype(float64')
- Goal:
  - ➢ Uniform data types for model input

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.3 The section 2 describes the following topics:

Insights for visual analytics tasks

Strategic conclusions:

- SpaceX saves up to **$100 million per launch** by reusing the first stage.
- Predicting landing success is therefore highly relevant from both an **economic and technical perspective**.

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 1: Visualize the relationship between flight number and launch site

- **Visualization**: Catplot with FlightNumber on the x-axis, LaunchSite on the y-axis, and color coding by class (0 = failure, 1 = success).
- The screenshot displays the coding for the catplot.

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

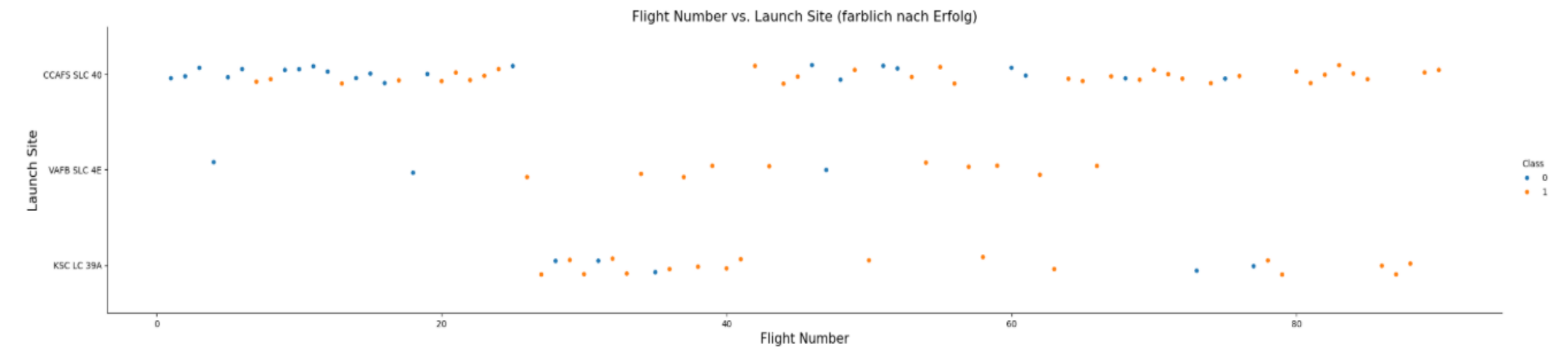
Use the function `catplot` to plot `FlightNumber` vs `LaunchSite`, set the parameter `x` parameter to `FlightNumber`,set the `y` to `Launch Site` and set the parameter `hue` to `'class'`

[10]:
```
# 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(
    x="FlightNumber",
    y="LaunchSite",
    hue="Class",
    data=df,
    aspect=5,
    height=5
)
plt.xlabel("Flight Number", fontsize=15)
plt.ylabel("Launch Site", fontsize=15)
plt.title("Flight Number vs. Launch Site (farblich nach Erfolg)", fontsize=16)
plt.show()
```

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 1: Visualize the relationship between flight number and launch site

- **Interpretation of the graph**: The graph displays how the success of landings has developed over time (FlightNumber) and depending on the takeoff location.



4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 1: Visualize the relationship between flight number and launch site

- **Observed patterns:**
- **CCAFS SLC 40 (Cape Canaveral):**
  - Early flights (FlightNumber < 20): many failures → blue dots
  - Later flights: significant improvement → orange dots dominate
  - Pattern: learning curve visible – more experience leads to higher success rate
- **KSC LC 39A (Kennedy Space Center)**
  - Only active from flight number ≈ 40
  - Almost exclusively successful landings → orange dots
  - Interpretation: Later use with mature technology and stable success rate

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 1: Visualize the relationship between flight number and launch site

- **Observed patterns:**
- **VAFB SLC 4E (Vandenberg):**
  - Fewer flights overall
  - Mixed bag of successes and failures
  - Pattern: No clear trend – possibly different mission profiles or lower utilization

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 1: Visualize the relationship between flight number and launch site

- **Conclusion:**
  - **Experience increases success:** With increasing flight numbers, the success rate rises – especially at CCAFS SLC 40.
  - **Technological progress visible:** Launch sites used later, such as KSC LC 39A, immediately show high success rates.
  - **Launch site as an influencing factor**: Different success distributions depending on the launch site → relevant for later modeling.

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 2: Visualize the relationship payload mass and launch site

- **Visualization**: Scatter plot with PayloadMass on the x-axis, LaunchSite on the y-axis, and color coding by class (0 = failure, 1 = success).
- The screenshot displays the coding for the Scatterplot.

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

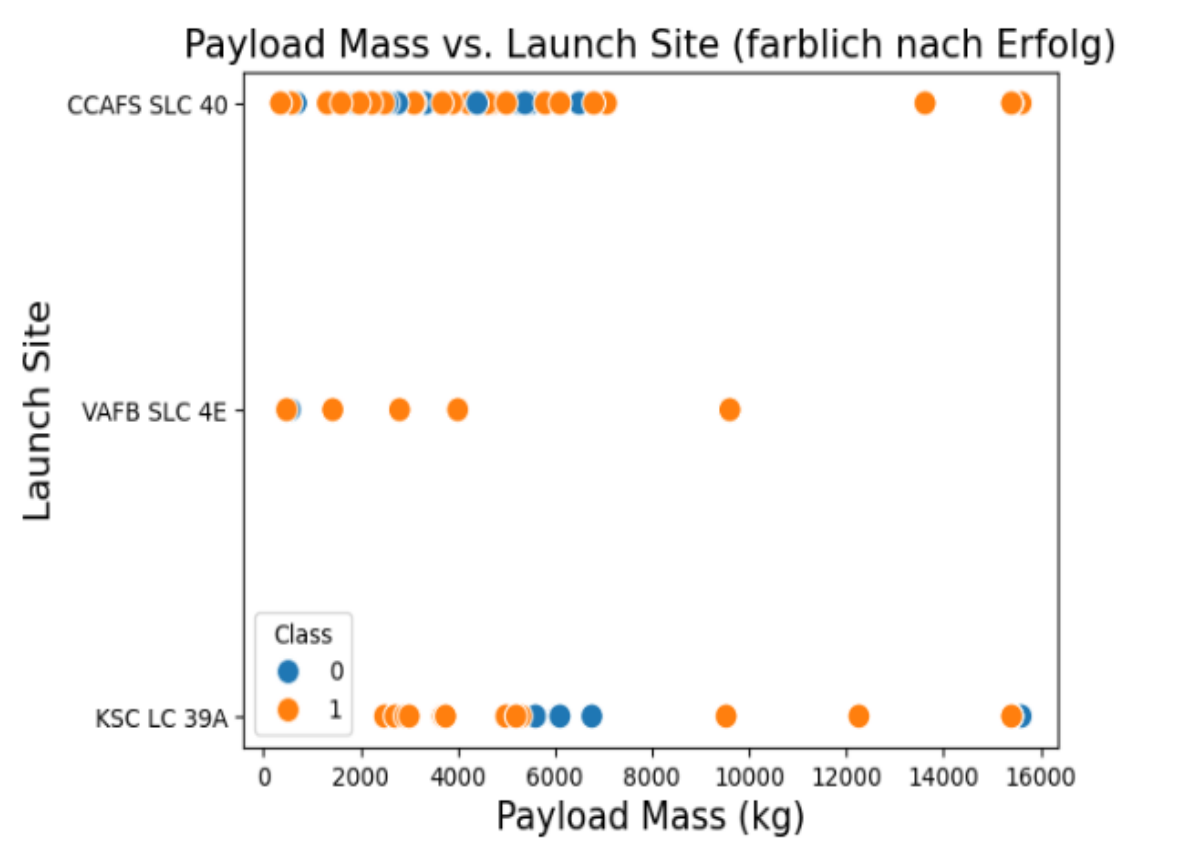
We also want to observe if there is any relationship between launch sites and their payload mass.

[12]:
```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch site, and hue to be the class value
sns.scatterplot(
    x="PayloadMass",
    y="LaunchSite",
    hue="Class",
    data=df,
    # Optional: point size
    s=100  # Punktgröße (optional)
)

plt.xlabel("Payload Mass (kg)", fontsize=15)
plt.ylabel("Launch Site", fontsize=15)
plt.title("Payload Mass vs. Launch Site (farblich nach Erfolg)", fontsize=16)
plt.show()
```

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 2: Visualize the relationship payload mass and launch site

- **Interpretation of the graph**:
  The graph displays how payload mass is distributed depending on the launch site – and wheter certain launch sites are preferred for heavy or light payloads.



4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 2: Visualize the relationship payload mass and launch site

- **Observed patterns:**
- **VAFB SLC 4E (Vandenberg):**
  - No rockets with payloads > 10000 kg
  - Often used for special missions such as SSO or polar orbits
  - These typically require smaller payloads
  - Possible technical or logistical restrictions at the site
- **CCAFS SLC 40 & KSC LC 39A (Florida):**
  - Wide payload spectrum, including heavy payload
  - Proximity to the equator → energy advantage due to earth's rotation
  - Historically preferred for GTO and LEO missions with high weight

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

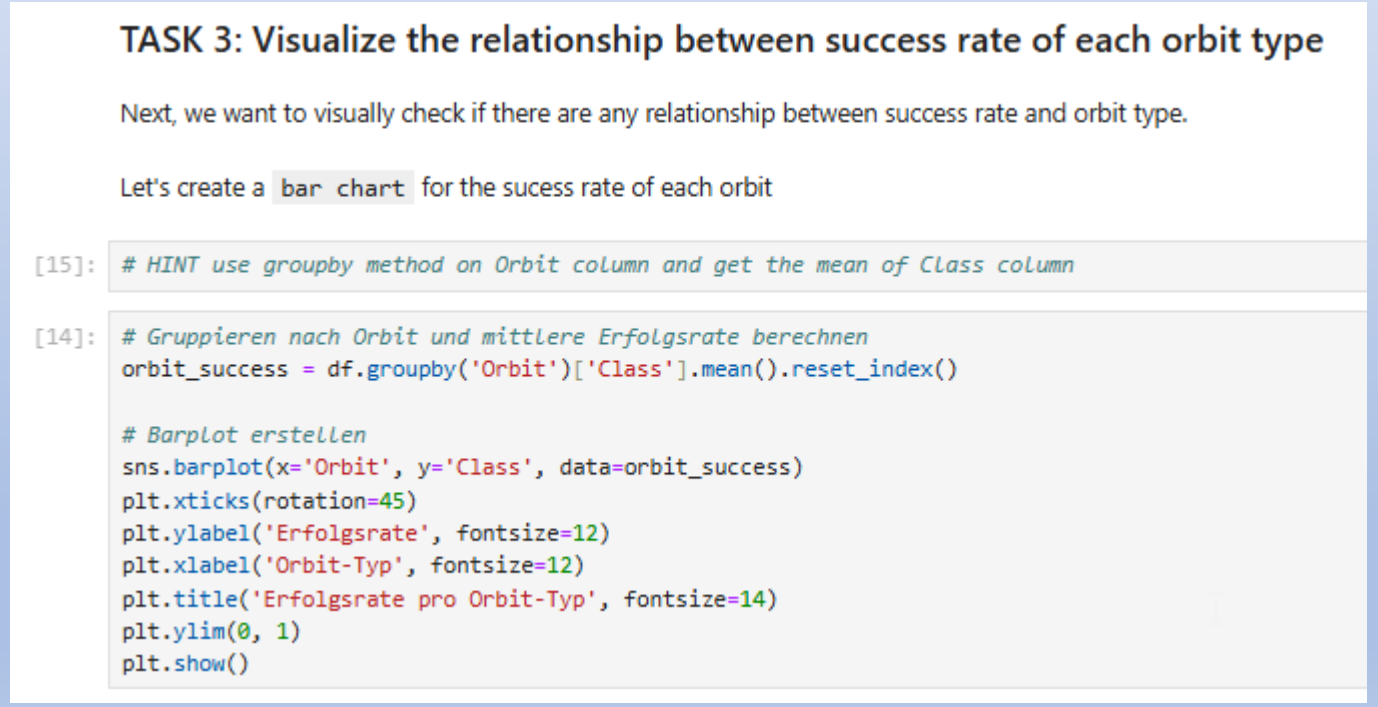
4.4 TASK 2: Visualize the relationship payload mass and launch site

- **Conclusion:**
  - **VAFB SLC 4E** has not yet been used for heavy payloads (> 10000 kg).
  - Reasons for this could lie in **deliberate mission planning, technical infrastructure**, or geographical conditions.
  - **The launch site is a relevant influencing factor** for payload capacity and mission profile.

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

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

- **Visualization**: Bar plot with orbit on the x-axis and average success rate (Class.mean()) on the y-axis → Success rate between 0 and 1 (0 = failure, 1 = success)
- The screenshot displays the coding for the Scatterplot.



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

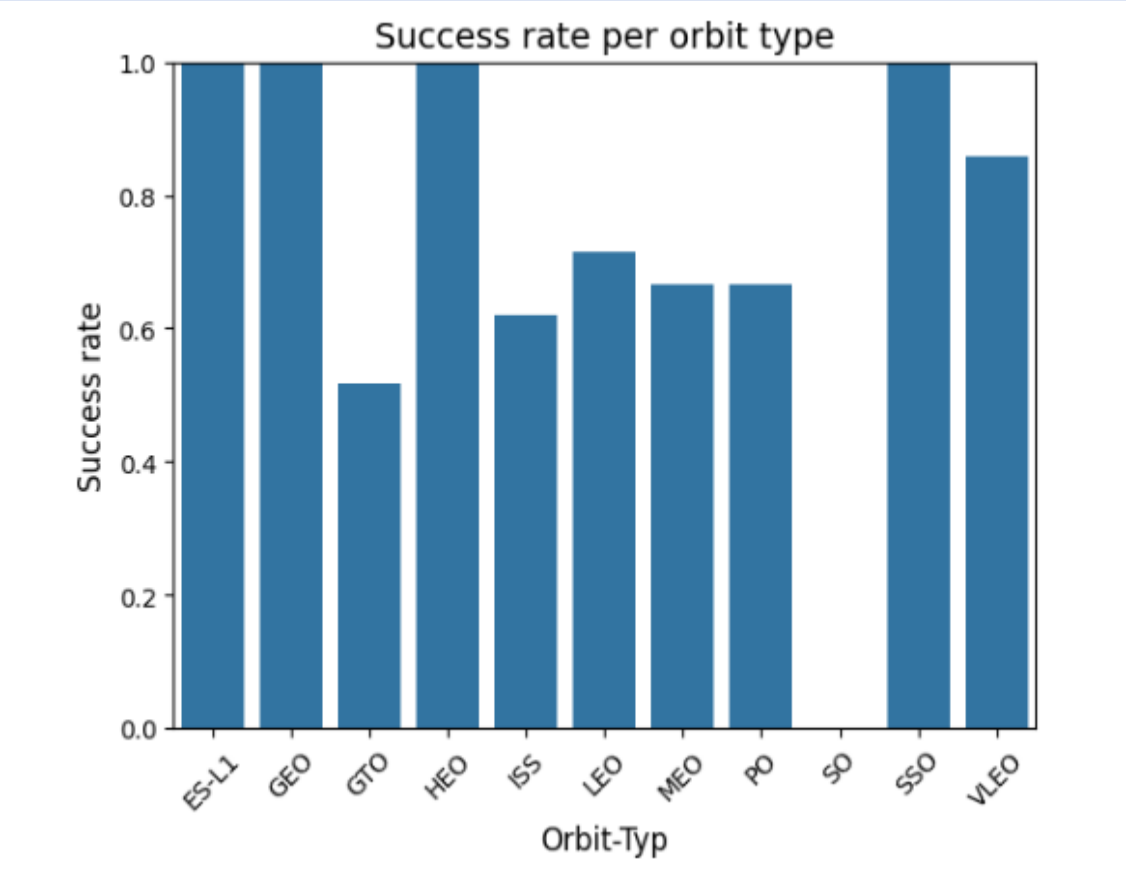
[15]:
```
# HINT use groupby method on Orbit column and get the mean of Class column
```

[14]:
```
# Gruppieren nach Orbit und mittlere Erfolgsrate berechnen
orbit_success = df.groupby('Orbit')['Class'].mean().reset_index()

# Barplot erstellen
sns.barplot(x='Orbit', y='Class', data=orbit_success)
plt.xticks(rotation=45)
plt.ylabel('Erfolgsrate', fontsize=12)
plt.xlabel('Orbit-Typ', fontsize=12)
plt.title('Erfolgsrate pro Orbit-Typ', fontsize=14)
plt.ylim(0, 1)
plt.show()
```

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

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

- **Interpretation of the graph**:
  The graph displays how reliable different orbit types are in terms of first stage landing.



4. Section 2 – Exploratory Data Analysis (EDA) Part 2

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

- **Observed patterns:**
- **Orbits with a high success rate (≈ 1.0):**
  - SSO (Sun-Synchronous Orbit)
  - LEO (Low Earth Orbit)
  - PO (Polar Orbit)
- **Explanation:**
  - Mostly smaller payloads
  - Landings over land or nearby sea areas
  - Routine flights with repeatable mission profiles

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

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

- **Observed patterns:**
- **Orbits with lower success rates:**
  - GTO (Geostationary Transfer Orbit)
    - Heavier payloads
    - Complex flight profiles
    - Higher energy requirements → greater risk
  - ISS (International Space Station)
    - Success rate depends on mission duration
    - Early SpaceX flights experienced more failures

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

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

- **Conclusion:**
  - **LEO/SSO missions are more reliable** → suitable for routine
  - **GTO and similar orbits require more technology and energy** → higher risks.
  - **Orbit type is a relevant predicator** of the success of the first stage landing.

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 4: Visualize the relationship between FlightNumber and orbit type

- **Visualization**: Scatter plot with flight number on the x-axis, orbit on the y-axis, and color coding by class (0 = failure, 1 = success).
- The screenshot displays
  the coding for the Scatterplot.

TASK 4: Visualize the relationship between FlightNumber and Orbit type

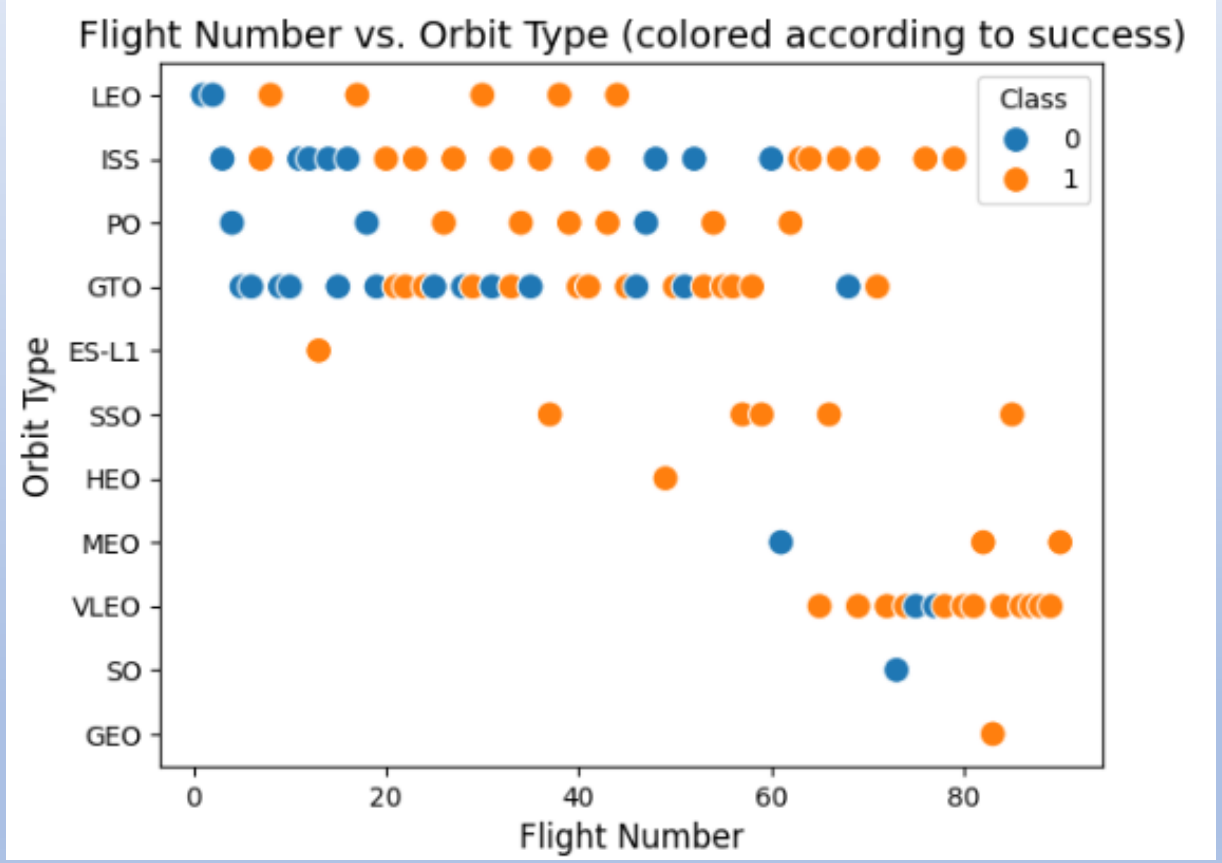
For each orbit, we want to see if there is any relationship between FlightNumber and Orbit type.

[10]:
```
# 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.scatterplot(
    x="FlightNumber",
    y="Orbit",
    hue="Class",
    data=df,
    # Optional: point size
    s=100
)

plt.xlabel("Flight Number", fontsize=12)
plt.ylabel("Orbit Type", fontsize=12)
plt.title("Flight Number vs. Orbit Type (colored according to success)", fontsize=14)
plt.show()
```

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 4: Visualize the relationship between FlightNumber and orbit type

- **Interpretation of the graph**:
  The graph displays how landing success has developed over time (FlightNumber) depending on the type of orbit.



4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 4: Visualize the relationship between FlightNumber and orbit type

- **Observed patterns:**
  - **LEO (Low Earth Orbit):**
    - Early flights: many failures
    - Later flights: Significantly more successful landing

    Interpretation:
    - Routine flights with growing experience
    - SpaceX has visibly optimized in this orbit category

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 4: Visualize the relationship between FlightNumber and orbit type

- **Observed patterns:**
  - **GTO (Geostationary Transfer Orbit):**
    - No clear trend discernible
    - Mix of successes and failures across all flight numbers

    Interpretation:
    - Complex mission profiles with heavier payloads
    - Learning curve less visible – success does not appear to be directly dependent on experience

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 4: Visualize the relationship between FlightNumber and orbit type

- **Conclusion:**
  - **LEO:** Success increases with experience → routine, system improvement.
  - **GTO:** Complexity overshadows the learning curve effect → success fluctuates independently of flight number.
  - **Orbit type influence not only the success rate, but also learnability**.

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 5: Visualize the relationship between payload mass and orbit type

- **Visualization**: Scatter plot with payload mass on the x-axis, orbit on the y-axis, and color coding by class (0 = failure, 1 = success).
- The screenshot displays the coding for the Scatterplot.

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

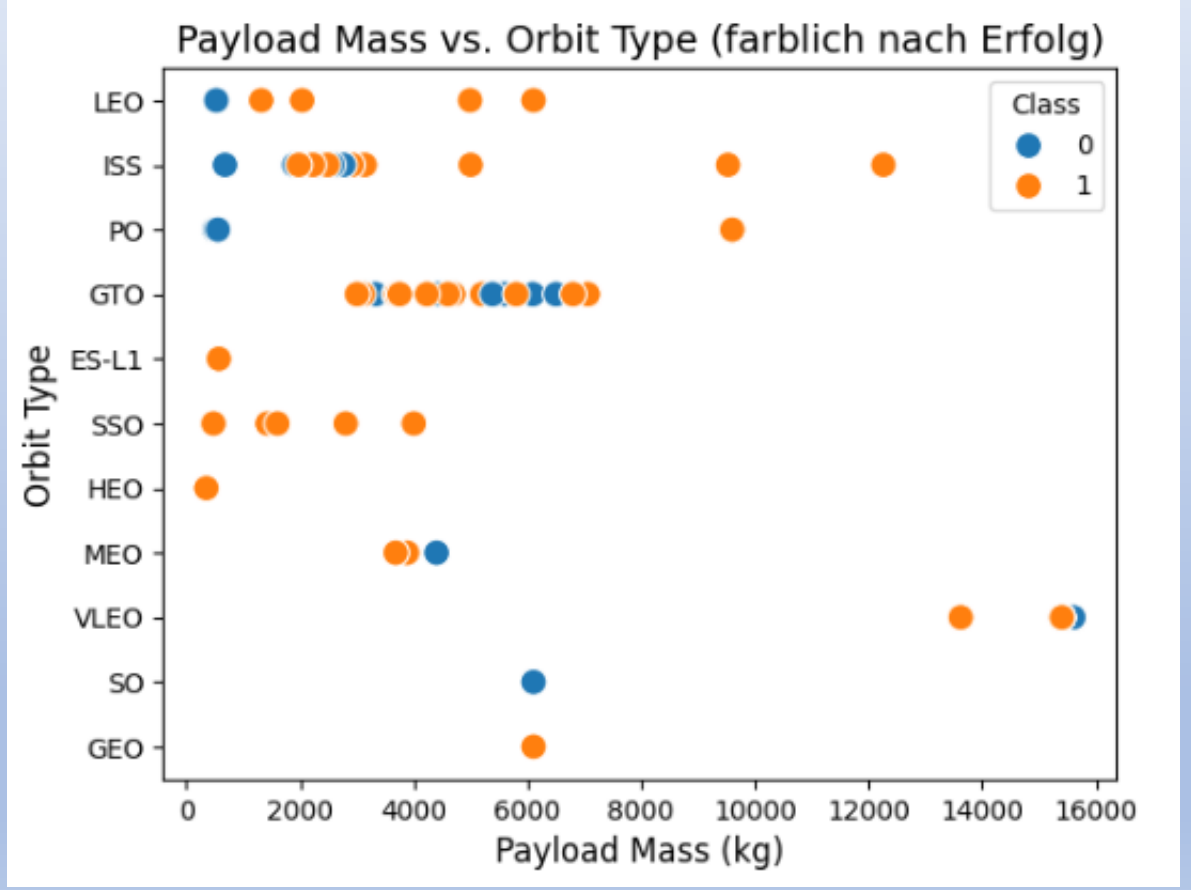
Similarly, we can plot the Payload Mass vs. Orbit scatter point charts to reveal the relationship between Payload Mass and Orbit type

[19]:
```
# 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
sns.scatterplot(
    x="PayloadMass",
    y="Orbit",
    hue="Class",
    data=df,
    s=100  # optional: Punktgröße
)

plt.xlabel("Payload Mass (kg)", fontsize=12)
plt.ylabel("Orbit Type", fontsize=12)
plt.title("Payload Mass vs. Orbit Type (farblich nach Erfolg)", fontsize=14)
plt.show()
```

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 5: Visualize the relationship between payload mass and orbit type

- **Interpretation of the graph**:
  The graph displays how the payload mass is distributed according to orbit type – and whether certain orbits are preferred for heavy payloads.



4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 5: Visualize the relationship between payload mass and orbit type

- **Observed patterns:**
  - **Polar, LEO and ISS orbits:**
    - Even with high payload masses (>10000 kg), successful landings are common

    Interpretation:

    - These orbits are apparently well mastered.
    - Routine flights and optimized landing technologies contribute to the high success rate.

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 5: Visualize the relationship between payload mass and orbit type

- **Observed patterns:**
  - **GTO (Geostationary Transfer Orbit):**
    - No clear distinction between success and failure.
    - Mix of blue and orange dots across all payload ranges.

    Interpretation:
    - GTO missions are more complex
    - Heavier payloads, higher energy requirements.
    - Success appears to be less directly correlated with payload mass.

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 5: Visualize the relationship between payload mass and orbit type

- **Conclusion:**
  - **Polar, LEO, and ISS orbits** show a high success rate even with heavy payloads.
  - **GTO missions** are more technically demanding –
    success is more difficult to predict .
  - **Orbit type and payload mass together influence the probability of landing**.

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 6: Visualize the launch success yearly trend

- **Data selection**: Before visualization, I selected the required data and output it in tabular form for initial analysis.
- The screenshot displays the coding for the data selection.

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:

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

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 6: Visualize the launch success yearly trend

- The screenshot displays
  the table with the selected data.

[20]:

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

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 6: Visualize the launch success yearly trend

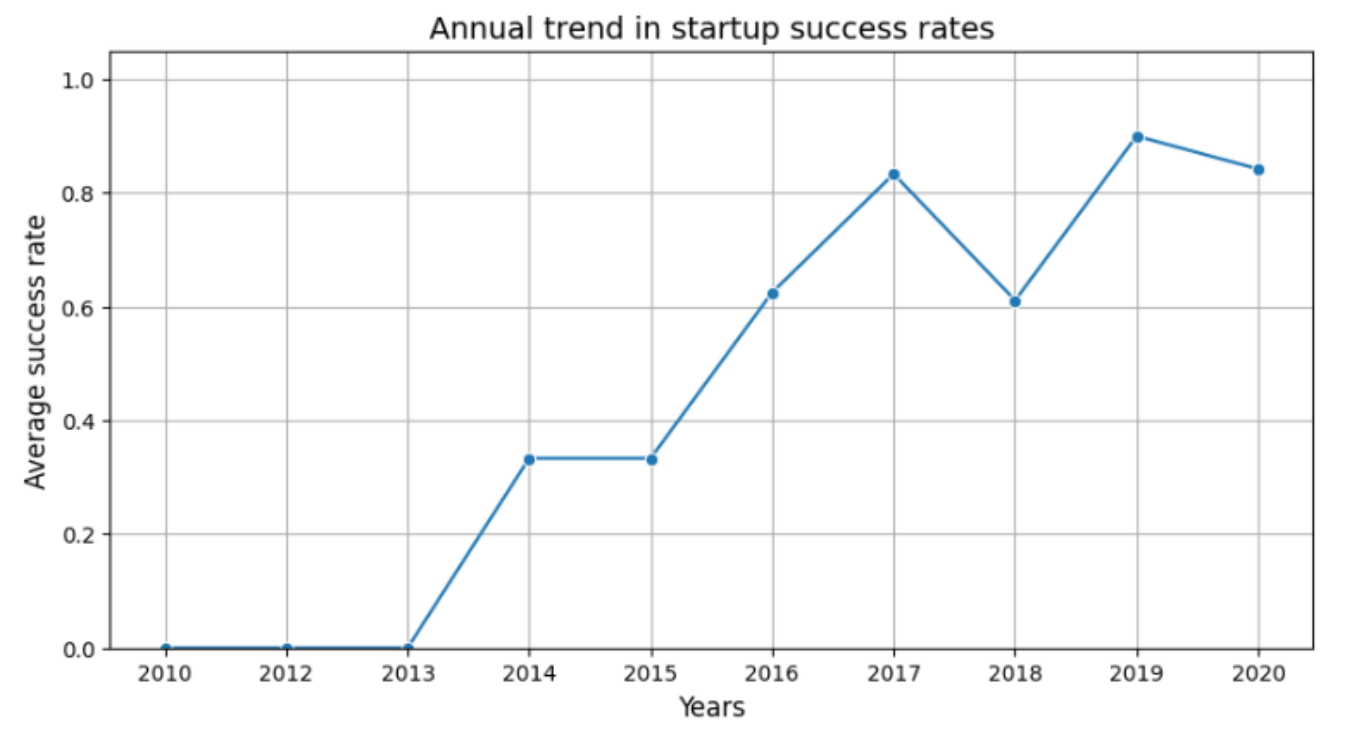
- **Visualization**: Line chart with Year on the x-axis and average success rate (Class.mean()) on the y-axis → Success rate between 0 and 1 (0 = failure, 1 = success).
- The screenshot displays the coding for the lineplot.

[11]:
```
# Plot a line chart with x axis to be the extracted year and y axis to be the success rate


# 1. Extract the year (if not already done)
year = []
def Extract_year():
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year

df['Year'] = Extract_year()

# 2. Calculate success rate per year
yearly_success = df.groupby('Year')['Class'].mean().reset_index()

# 3. Draw a line graph
plt.figure(figsize=(10,5))
sns.lineplot(x='Year', y='Class', data=yearly_success, marker='o')

plt.xlabel('Years', fontsize=12)
plt.ylabel('Average success rate', fontsize=12)
plt.title('Annual trend in startup success rates', fontsize=14)
plt.ylim(0, 1.05)
plt.grid(True)
plt.show()
```

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 6: Visualize the launch success yearly trend

- **Interpretation of the graph**:
  The graph displays how the success rate of SpaceX launches has developed over the years – an indicator of technological maturity and operational stability.



4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 6: Visualize the launch success yearly trend

- **Observed patterns:**
  - **2010-2012:** Initial launches, low success rate
  - **From 2013:** Significant → Start of systematic improvements
  - **2015-2020:** Success rate stabilizes at a high level
  - **Interpretation:**
    - Introduction of reusable components
    - Optimization of landing technologies
    - Increased experience through routine flights

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 6: Visualize the launch success yearly trend

- **Conclusion:**
  - Time progression shows learning curve and technological maturity.
  - Success rate suitable as a feature for modeling.
  - Year can serve as a proxy for system generation or level of experience.

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 6: Visualize the launch success yearly trend

- **Feature selection for modeling:**
- **Goal:** Identification of relevant variables for predicting landing success (class).
  - **Technical parameters:**
    - PayloadMass, Flights, GridFins, Reused, Legs, Block, ReusedCount → directly influence flight and landing dynamics
  - **Mission type & context:**
    - Orbit, LaunchSite, LandingPad → provide information about mission profile and geographical conditions
  - **Identifiers:**
    - FlightNumber, Serial → enable temporal and hardware related assignment

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 6: Visualize the launch success yearly trend

- The screenshot displays
  the table with the selected data.

Features Engineering

By now, you should obtain some preliminary insights about how each important variable would affect the success rate, we will select the features that will be used in success prediction in the future module.

[23]:
```
features = df[['FlightNumber', 'PayloadMass', 'Orbit', 'LaunchSite', 'Flights', 'GridFins', 'Reused', 'Legs', 'LandingPad', 'Block', 'ReusedCount', 'Serial']]
features.head()
```

[23]:

	FlightNumber	PayloadMass	Orbit	LaunchSite	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial
0	1	6104.959412	LEO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B0003
1	2	525.000000	LEO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B0005
2	3	677.000000	ISS	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B0007
3	4	500.000000	PO	VAFB SLC 4E	1	False	False	False	NaN	1.0	0	B1003
4	5	3170.000000	GTO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B1004

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 7: Create dummy variables to categorial columns

- **Goal**: Convert categorial columns into numerical form → prerequisite for ML algorithms
- The screenshot displays the coding for the data selection.

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.

[25]:
```
# HINT: Use get_dummies() function on the categorical columns
# List of categorical columns to be coded
categorical_columns = ['Orbit', 'LaunchSite', 'LandingPad', 'Serial']

# Perform one-hot encoding
features_one_hot = pd.get_dummies(df, columns=categorical_columns)

# Show result
features_one_hot.head()
```

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 7: Create dummy variables to categorial columns

- The screenshot displays the selected data.

[25]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Outcome	Flights	GridFins	Reused	Legs	Block	...	Serial_B1048	Serial_B1049	Serial_B1050	Serial_B1051	Serial_B1054	Serial_B1056	Serial_B1058	Serial_B1059	Serial_B1060	Serial_B1062
0	1	2010	Falcon 9	6104.959412	None None	1	False	False	False	1.0	...	False	False	False	False	False	False	False	False	False	False
1	2	2012	Falcon 9	525.000000	None None	1	False	False	False	1.0	...	False	False	False	False	False	False	False	False	False	False
2	3	2013	Falcon 9	677.000000	None None	1	False	False	False	1.0	...	False	False	False	False	False	False	False	False	False	False
3	4	2013	Falcon 9	500.000000	False Ocean	1	False	False	False	1.0	...	False	False	False	False	False	False	False	False	False	False
4	5	2013	Falcon 9	3170.000000	None None	1	False	False	False	1.0	...	False	False	False	False	False	False	False	False	False	False

5 rows × 87 columns

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 7: Create dummy variables to categorial columns

- **Observed structure after conversion**:
  - Each category becomes its own column → e.g., Orbit_LEO, Orbit_GTO, Orbit_SSO
  - Values: 0 = not applicate, 1 = applicable
  - No loss of information, no ranking → ideal for classification models
- **Conclusion:**
  - Dummy variables enable numerical processing of categorial features
  - Basis for modeling with algorithms such as logistic regression, decision trees, or random forest.
  - Next step: Feature scaling and model training

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 8: Cast all numeric columns to float64

- **Goal**: Cast all numeric columns in features_one_hot to float64 → consistent type basis for modeling.
- The screenshot displays the coding for the display of data types.

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`

[28]:
```
# HINT: use astype function
# Step 1: Identify non-numeric columns
non_numeric_cols = features_one_hot.select_dtypes(include=['object']).columns

# Step 2: Remove these columns (if they are not needed)
features_one_hot = features_one_hot.drop(non_numeric_cols, axis=1)

# Step 3: Cast to float64
features_one_hot = features_one_hot.astype('float64')

# Control
features_one_hot.dtypes.head()
```

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 8: Cast all numeric columns to float64

Reason:

- float64 is the standard for numerical ML features.
- Avoids type conflicts during model training and scaling.
- Uniform basis for mathematical operations.

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 8: Cast all numeric columns to float64

- The screenshot displays the coding for the display number of columns.

```
[30]: print("Final number of columns:", features_one_hot.shape[1])

      Final number of columns: 83
```

4. Section 2 – Exploratory Data Analysis (EDA) Part 2

4.4 TASK 8: Cast all numeric columns to float64

- The screenshot displays the coding for exporting data to csv file.

```
[29]: features_one_hot.to_csv('dataset_part_3.csv', index=False)
```

Section 3

Launch Sites Proximities Analysis

5. Section 3 – Interactive Visual Analytics with Folium

5.1 The section 3 describes the following topics:

- Global distribution of launch sites
  - ➢ TASK 1: Mark all launch sites on a map
- Launch success vs. failure by location
  - ➢ TASK 2: Mark the success/ failed launches for each site on the map
- Proximity Analysis
  - ➢ TASK 3: Calculate the distance between a launch site to its proximities
- Each task builds on the previous on, moving from geographic mapping to outcome analysis and finally to strategic proximity evaluation.

5. Section 3 – Global distribution of launch sites

5.2 TASK 1: Mark all launch sites on a map

- All SpaceX launch sites are marked.
  Coastal positioning is evident,
  suggesting strategic placement for trajectory and safety.
- The following programming for creating folium maps via Jupyter Notebook is used to determine the launch and landing sites for the rockets.

5. Section 3 – Global distribution of launch sites

5.2 TASK 1: Mark all launch sites on a map

- In order to use the data from the spacex_launch_geo.csv file for visualization with Folium, the file was loaded into the notebook via URL. The coordinates (latitude and longitude) of the launch sites were then extracted and the first rows of the table were displayed for verification purposes.
- The corresponding Python code is shown on the following slide via screenshot. The import of the required libraries was deliberately not discussed in detail here.

5. Section 3 – Global distribution of launch sites

5.2 TASK 1: Mark all launch sites on a map

- The screenshot displays the determined data from the CSV file.

[9]:
```
## Task 1: Mark all launch sites on a map
import requests
import pandas as pd

# URL to CSV file
url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/spacex_launch_geo.csv"

# Download file
response = requests.get(url)
with open("spacex_launch_geo.csv", "wb") as file:
    file.write(response.content)

# Import CSV file
spacex_df = pd.read_csv("spacex_launch_geo.csv")

# Extract Launch sites
launch_sites_df = spacex_df[['Launch Site', 'Lat', 'Long']].drop_duplicates()

# Display data
spacex_df.head()
```

[9]:

	Flight Number	Date	Time (UTC)	Booster Version	Launch Site	Payload	Payload Mass (kg)	Orbit	Customer	Landing Outcome	class	Lat	Long
0	1	2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0.0	LEO	SpaceX	Failure (parachute)	0	28.562302	-80.577356
1	2	2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel o...	0.0	LEO (ISS)	NASA (COTS) NRO	Failure (parachute)	0	28.562302	-80.577356
2	3	2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2+	525.0	LEO (ISS)	NASA (COTS)	No attempt	0	28.562302	-80.577356
3	4	2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500.0	LEO (ISS)	NASA (CRS)	No attempt	0	28.562302	-80.577356
4	5	2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677.0	LEO (ISS)	NASA (CRS)	No attempt	0	28.562302	-80.577356

5. Section 3 – Global distribution of launch sites

5.2 TASK 1: Mark all launch sites on a map

- The screenshot displays the data determined from the CSV file for the coordinates of the launch sites.

[10]:
```
# Select relevant sub-columns: `Launch Site`, `Lat(Latitude)`, `Long(Longitude)`, `class`
spacex_df = spacex_df[['Launch Site', 'Lat', 'Long', 'class']]
launch_sites_df = spacex_df.groupby(['Launch Site'], as_index=False).first()
launch_sites_df = launch_sites_df[['Launch Site', 'Lat', 'Long']]
launch_sites_df
```

[10]:

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

5. Section 3 – Global distribution of launch sites

5.2 TASK 1: Mark all launch sites on a map

Initial map centering: NASA Johnson Space Center

- The NASA Johnson Space Center in Houston, Texas, was chosen as the starting point for initializing the Folium map. The map was centered using **folium.Map()** and visually highlighted with **folium.Circle** and a marker. This display serves as the starting point for the subsequent placement of all launch sites.

Visual marker for NASA JSC

- A marker with a pop-up label and color highlighting was set for the starting point. The display is done using **folium.Circle** and **folium.Marker** with a **DivIcon** to show the location name directly on the map. The screenshot shows the implementation in the notebook and the visual output on the map.

5. Section 3 – Global distribution of launch sites

5.2 TASK 1: Mark all launch sites on a map

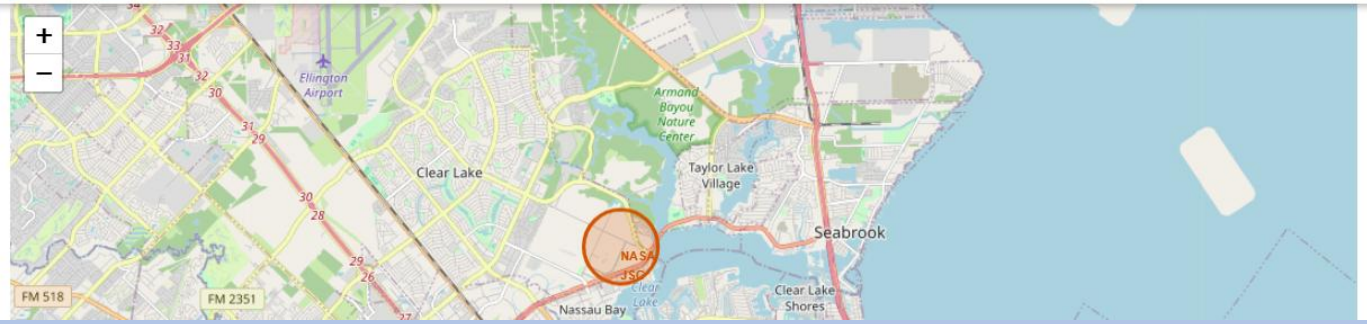
- Screenshot for displaying start location

[11]:
```
# Start location is NASA Johnson Space Center
nasa_coordinate = [29.559684888503615, -95.0830971930759]
site_map = folium.Map(location=nasa_coordinate, zoom_start=10)
```

We could use `folium.Circle` to add a highlighted circle area with a text label on a specific coordinate. For example,

[12]:
```
# Create a blue circle at NASA Johnson Space Center's coordinate with a popup label showing its name
circle = folium.Circle(nasa_coordinate, radius=1000, color='#d35400', fill=True).add_child(folium.Popup('NASA Johnson Space Center'))
# Create a blue circle at NASA Johnson Space Center's coordinate with a icon showing its name
marker = folium.map.Marker(
    nasa_coordinate,
    # Create an icon as a text label
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % 'NASA JSC',
        )
    )
site_map.add_child(circle)
site_map.add_child(marker)
```

[12]:



5. Section 3 – Global distribution of launch sites

5.2 TASK 1: Mark all launch sites on a map

Visual Mapping of Launch Sites

- The launch sites were displayed on an interactive map using **folium.Marker**. **MarkerCluster** was used to provide a better overview of locations that are close to each other, especially in Florida. Each location was assigned an individual marker color for visual differentiation.
- The map allows for exploratory analysis using a zoom function.
  The following questions were embedded in order to critically reflect on the geographical positioning of the launch sites.
- The answers to these questions provide clues for strategic location selection in the context of flight paths, safety, and infrastructure.

5. Section 3 – Global distribution of launch sites

5.2 TASK 1: Mark all launch sites on a map

Visual Mapping of Launch Sites

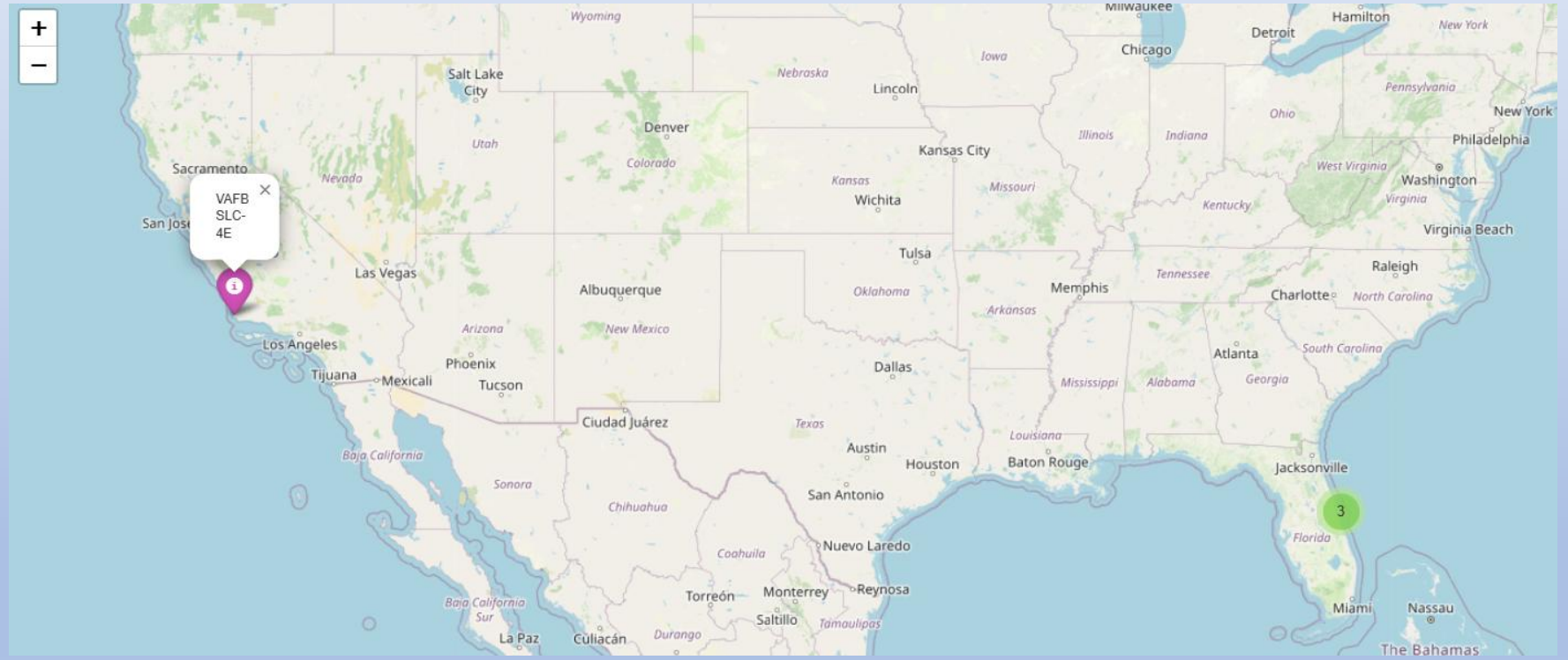
The following can be determined from the map:

- The launch sites are all located in southern latitudes, but not directly on the equator.
- All sites are close to the coast - a strategic advantage for flight paths over open sea and logistical connections.

5. Section 3 – Global distribution of launch sites

5.2 TASK 1: Mark all launch sites on a map

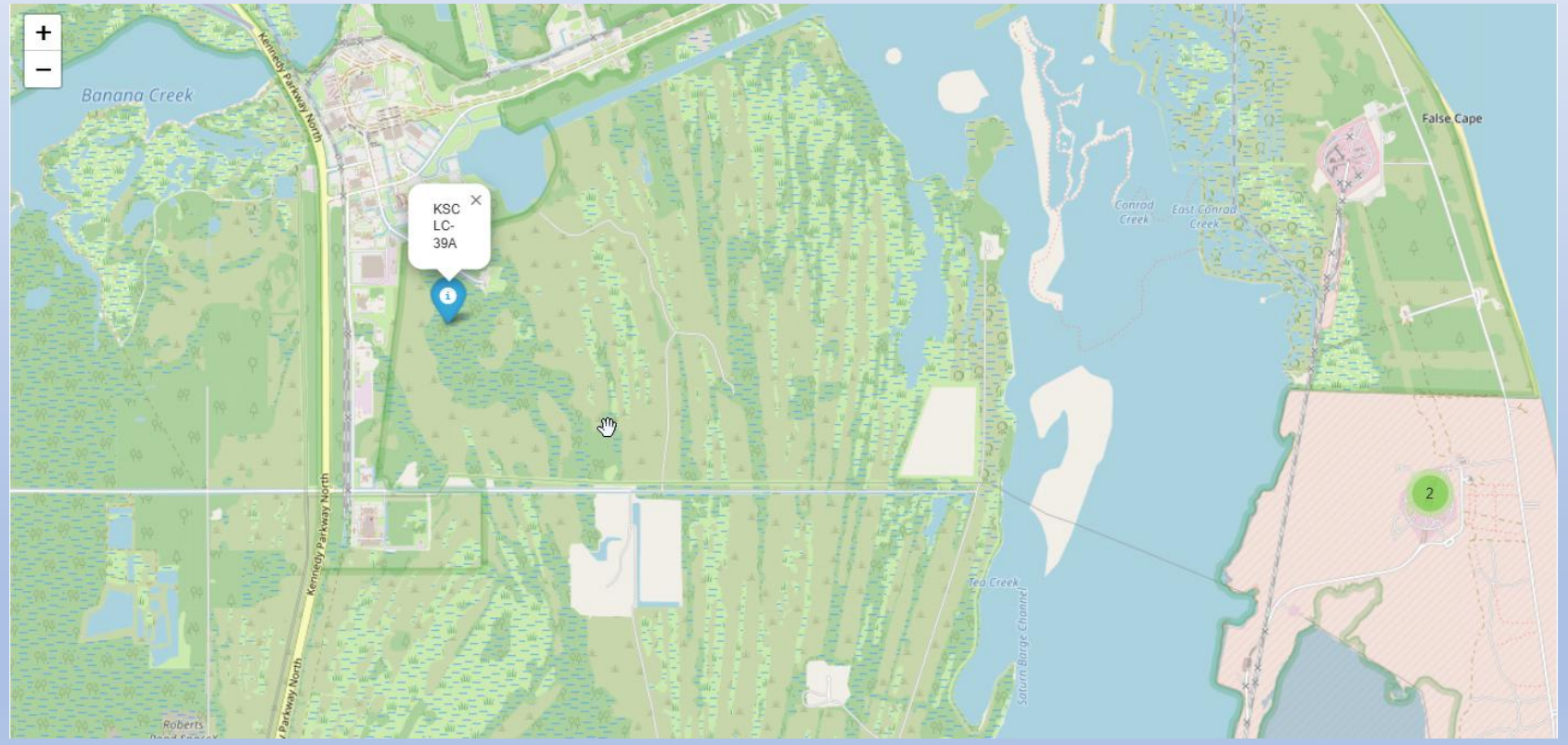
- Screenshot for displaying mark all launch sites



5. Section 3 – Global distribution of launch sites

5.2 TASK 1: Mark all launch sites on a map

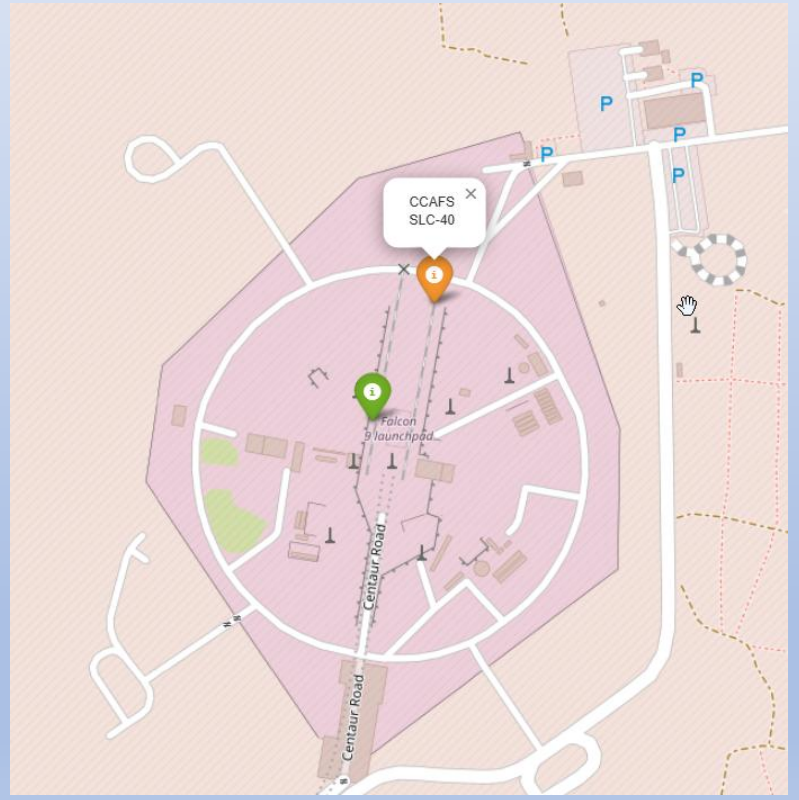
- Screenshot for displaying mark all launch sites

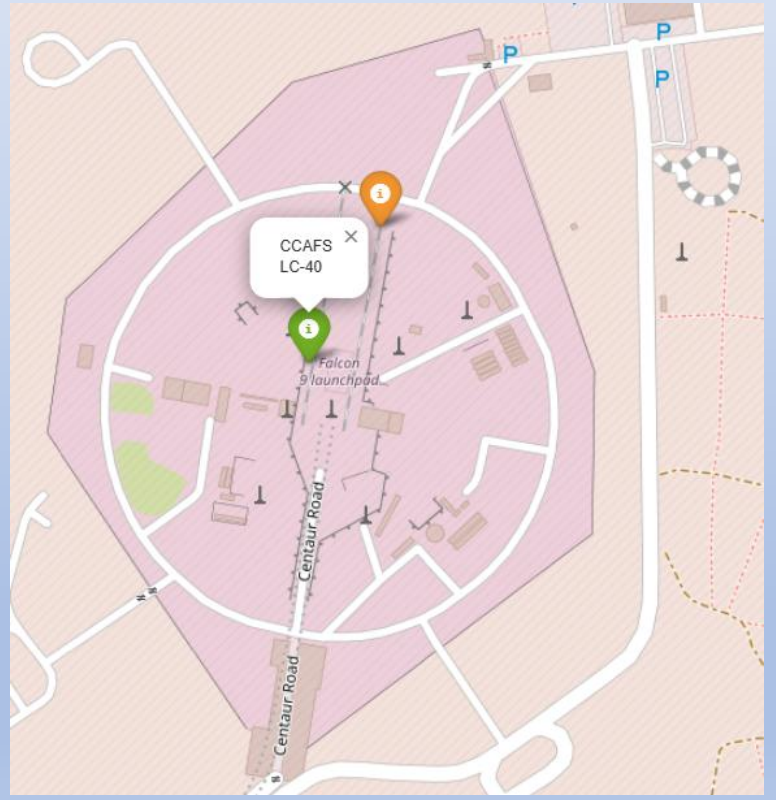


5. Section 3 – Global distribution of launch sites

5.2 TASK 1: Mark all launch sites on a map

- Screenshot for displaying mark all launch sites





5. Section 3 – Launch success vs. failure by location

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

Launch success vs. failure by location

Goal: Visualization of outcomes per launch:

Description:

- The goal is to display all launches with color coding according to success or failure. Each launch was represented by a color-coded marker:
  - Green for successful missions
  - Red for failed attempts
- The display is provided by **folium.CircleMarker**, supplemented by pop-ups showing location and results. The zoom function allows exploratory analysis of the spatial distribution of successes and failures.

5. Section 3 – Launch success vs. failure by location

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

Launch success vs. failure by location

- Screenshot of the code block for function displaying launch sites

```
[17]: import folium
from folium.plugins import MarkerCluster
import pandas as pd

# Example: Load CSV (if not already done)
spacex_df = pd.read_csv("spacex_launch_geo.csv")

# Center map (e.g., on Florida)
launch_map = folium.Map(location=[28.562302, -80.577356], zoom_start=5)

# Optional: Clusters for a better overview
marker_cluster = MarkerCluster().add_to(launch_map)

# Review all launches
for index, row in spacex_df.iterrows():
    lat = row['Lat']
    lon = row['Long']
    site = row['Launch Site']
    outcome = row['class']  # 1 = Erfolg, 0 = Misserfolg

    # Color depending on outcome
    color = 'green' if outcome == 1 else 'red'
    result = 'Success' if outcome == 1 else 'Fail'

    # Marker with popup
    folium.CircleMarker(
        location=[lat, lon],
        radius=6,
        color=color,
        fill=True,
        fill_color=color,
        fill_opacity=0.7,
        popup=f"{site} - {result}"
    ).add_to(marker_cluster)

# Show map
launch_map
```

5. Section 3 – Launch success vs. failure by location

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

Launch outcome data overview

Goal: Transparent database before visualization.

Description:

- In preparation for visualization, all launches were displayed in tabular form.
- The class column shows the outcome: 1 for success, 0 for failure. This overview serves as the basis for the error marker map.

5. Section 3 – Launch success vs. failure by location

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

Launch outcome data overview

- Screenshot for displaying mark all launch sites

```
[13]: spacex_df.tail(10)
```

[13]:

	Launch Site	Lat	Long	class
46	KSC LC-39A	28.573255	-80.646895	1
47	KSC LC-39A	28.573255	-80.646895	1
48	KSC LC-39A	28.573255	-80.646895	1
49	CCAFS SLC-40	28.563197	-80.576820	1
50	CCAFS SLC-40	28.563197	-80.576820	1
51	CCAFS SLC-40	28.563197	-80.576820	0
52	CCAFS SLC-40	28.563197	-80.576820	0
53	CCAFS SLC-40	28.563197	-80.576820	0
54	CCAFS SLC-40	28.563197	-80.576820	1
55	CCAFS SLC-40	28.563197	-80.576820	0

5. Section 3 – Launch success vs. failure by location

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

Launch success vs. failure by location

Goal: Visual anchoring of implementation

Description:

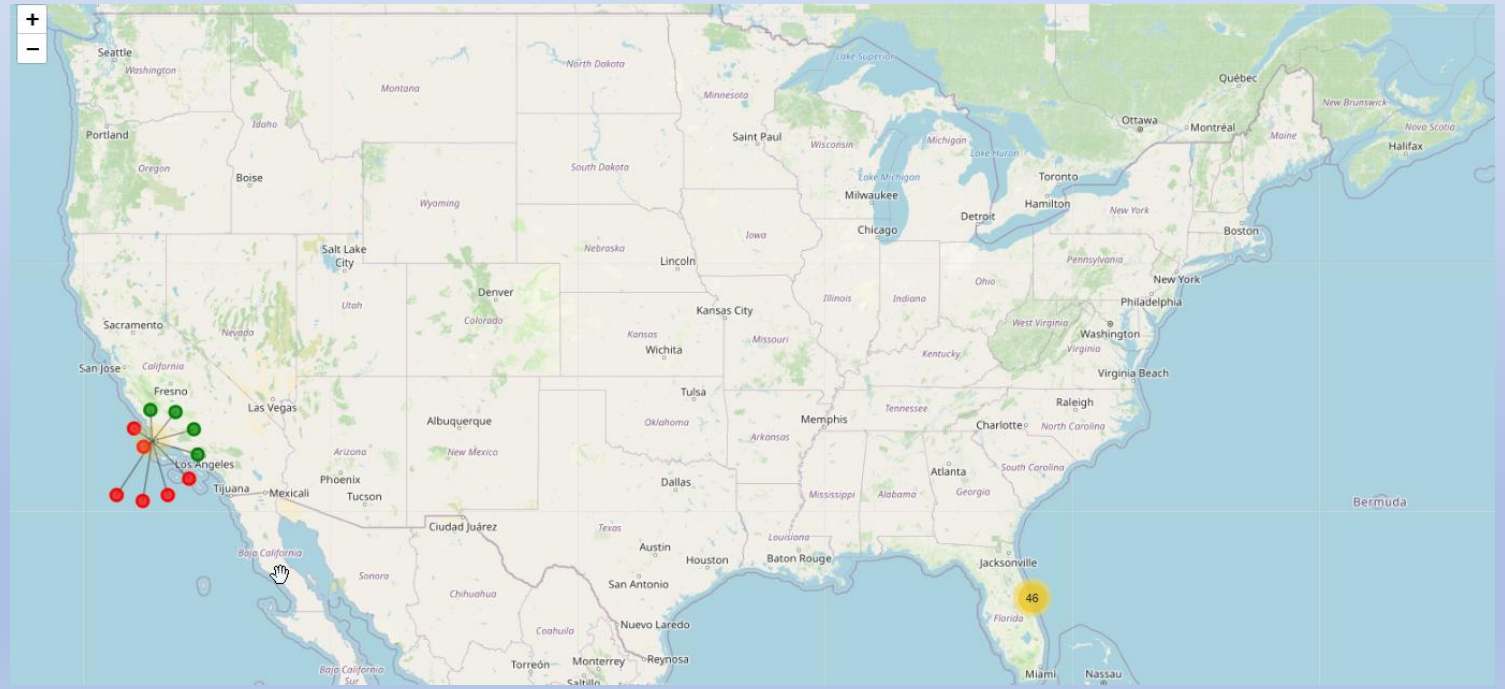
- The map shows the geographical distribution of launches with success/failure. MarkerCluster provides clarity when points are close together.

5. Section 3 – Launch success vs. failure by location

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

Launch success vs. failure by location

- Screenshot for displaying all launches success vs. failure

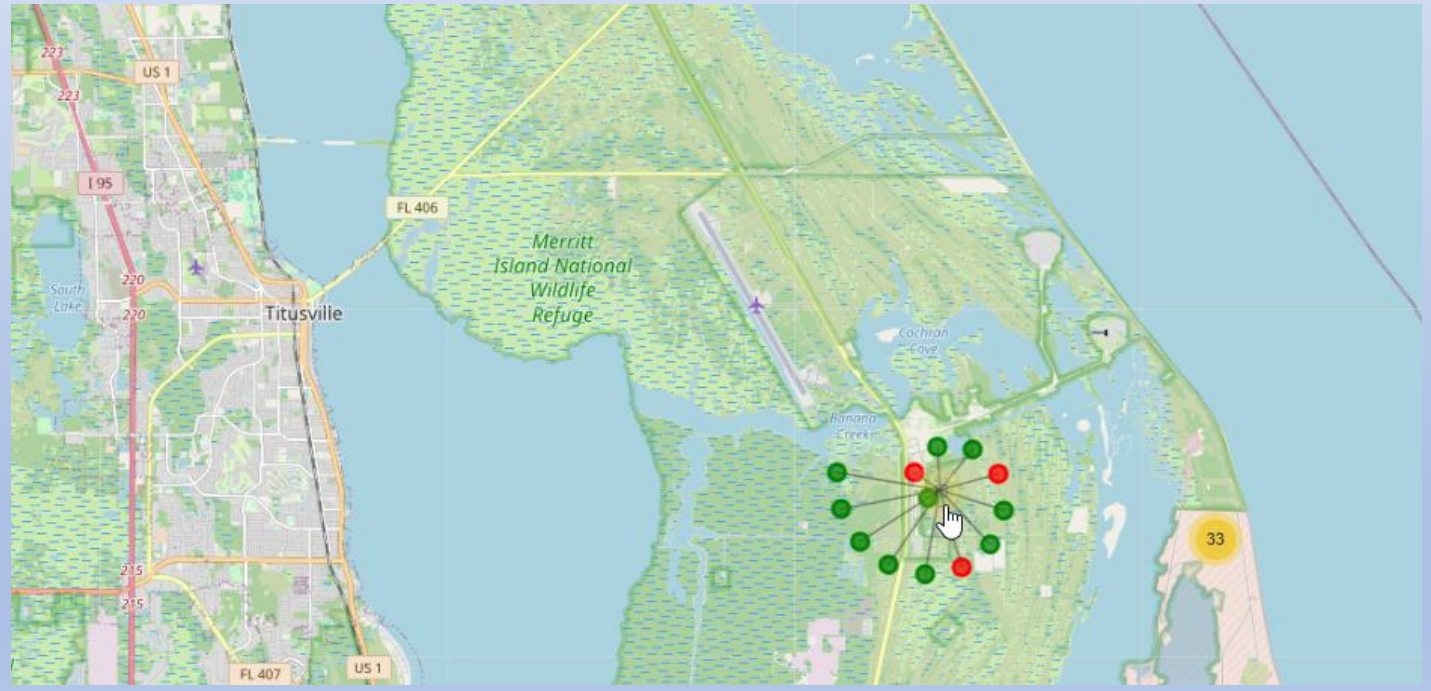


5. Section 3 – Launch success vs. failure by location

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

Launch success vs. failure by location

- Screenshot for displaying all launches success vs. failure

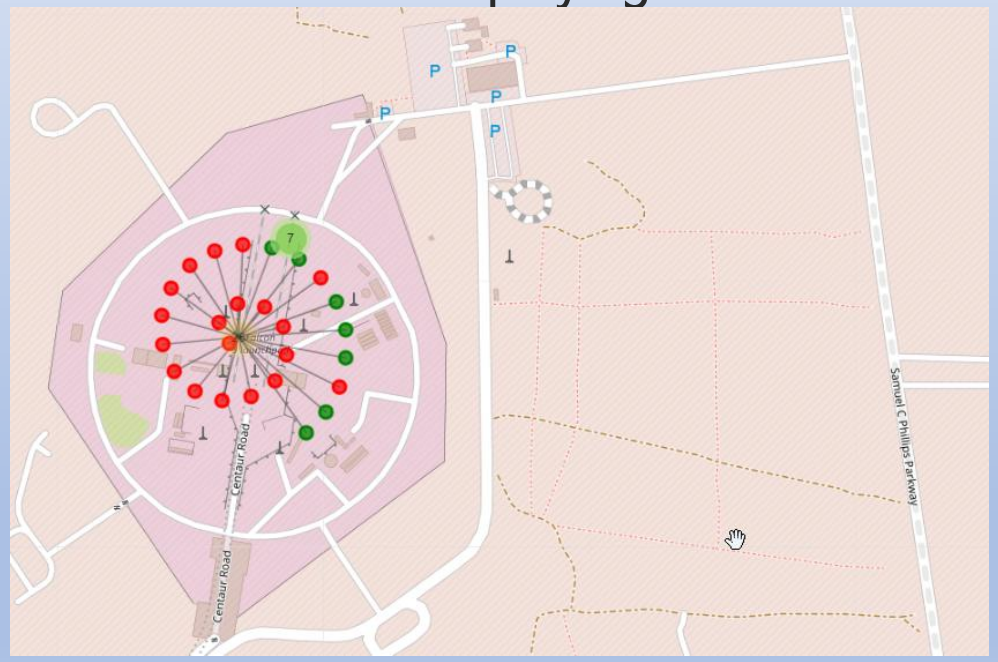


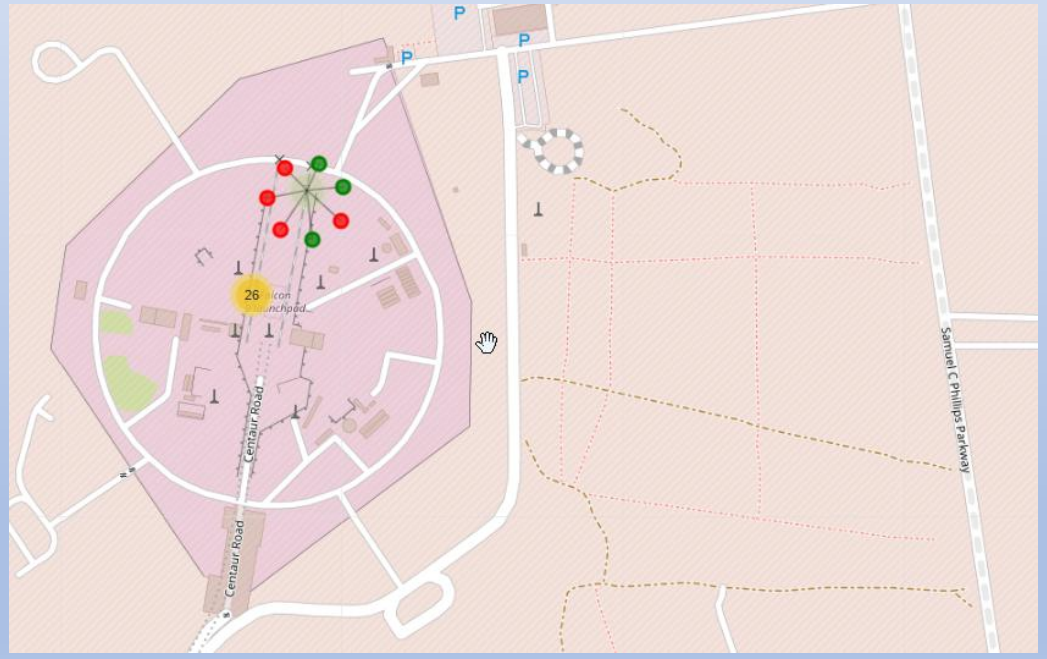
5. Section 3 – Launch success vs. failure by location

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

Launch success vs. failure by location

- Screenshot for displaying all launches success vs. failure





5. Section 3 – Launch success vs. failure by location

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

<u>Detailed view of a location</u>

Goal: In-depth analysis of a specific location

Description:

- The detailed view shows the launch site CCAFS SLC-40 with color-coded markers and zones.
- The display allows for precise analysis of the infrastructure, potential sources of error, and spatial relationships.

5. Section 3 – Launch success vs. failure by location

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

Infrastructure proximity

Goal: Introduction to goal setting

Description:

- The goal is to visualize the distance to relevant infrastructure such as the city, coast, railway line, and expressway – exemplified by CCAFS LC-40.

5. Section 3 – Proximity analysis

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

Infrastructure proximity

Goal: Preparation for distance calculation

Description:

- Before creating the map for visualization, the coordinates of the launch site and the nearest locations were determined. The table shows latitude, longitude, nearest city, and calculated distance.

5. Section 3 – Proximity analysis

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

Infrastructure proximity

- Screenshot for the code block for data determination

```
[18]: # TASK 3: Calculate the distances between a launch site to its proximities
from math import radians, cos, sin, asin, sqrt

# Haversine Distance (in km)
def haversine(lon1, lat1, lon2, lat2):
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * asin(sqrt(a))
    r = 6371  # Radius of earth in kilometers
    return c * r

# Define destinations (e.g., cities, coastline, road points)

# Destinations with coordinates
proximities = {
    'Orlando': (28.538336, -81.379234),
    'Miami': (25.761681, -80.191788),
    'Jacksonville': (30.332184, -81.655651)
}

# Calculate distances for all launch sites

# Launch sites from previous task
launch_sites = launch_sites_df.copy()
```

The following code of the block

```
# New columns for minimum distance and associated location
launch_sites['Nearest Place'] = ''
launch_sites['Min Distance (km)'] = 0.0

# Loop over all launch sites
for index, row in launch_sites.iterrows():
    site_lat = row['Lat']
    site_lon = row['Long']

    min_dist = float('inf')
    nearest_place = None

    for place, (place_lat, place_lon) in proximities.items():
        dist = haversine(site_lon, site_lat, place_lon, place_lat)
        if dist < min_dist:
            min_dist = dist
            nearest_place = place

    launch_sites.at[index, 'Nearest Place'] = nearest_place
    launch_sites.at[index, 'Min Distance (km)'] = round(min_dist, 2)

launch_sites
```

5. Section 3 – Proximity analysis

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

Infrastructure proximity

- The screenshot displays the determined data

[18]:

	Launch Site	Lat	Long	Nearest Place	Min Distance (km)
0	CCAFS LC-40	28.562302	-80.577356	Orlando	78.37
1	CCAFS SLC-40	28.563197	-80.576820	Orlando	78.42
2	KSC LC-39A	28.573255	-80.646895	Orlando	71.63
3	VAFB SLC-4E	34.632834	-120.610745	Jacksonville	3662.43

[19]:
```
launch_sites.sort_values('Min Distance (km)')
```

[19]:

	Launch Site	Lat	Long	Nearest Place	Min Distance (km)
2	KSC LC-39A	28.573255	-80.646895	Orlando	71.63
0	CCAFS LC-40	28.562302	-80.577356	Orlando	78.37
1	CCAFS SLC-40	28.563197	-80.576820	Orlando	78.42
3	VAFB SLC-4E	34.632834	-120.610745	Jacksonville	3662.43

5. Section 3 – Proximity analysis

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

Infrastructure proximity

Goal: Visualization of the distance to the nearest infrastructure (city, coast, railway, road).

Description:

- The distance was calculated using the Haversine formula. This method takes into account the curvature of the earth and provides precise distances in kilometers.
- The calculated distances were displayed on an interactive map:
  - Blue markers for the launch site
  - Orange text to show the distance between the launch site
  - Blue lines show the direct connection

5. Section 3 – Proximity analysis

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

Infrastructure proximity

- The screenshot displays the code block for the map with connections

```
[31]: # Optional: Map with connections
import folium
from folium.features import DivIcon

# New map
distance_map = folium.Map(location=[28.5, -80.5], zoom_start=6)

# Add markers & lines
for index, row in launch_sites.iterrows():
    site_name = row['Launch Site']
    site_coord = (row['Lat'], row['Long'])

    nearest_place = row['Nearest Place']
    nearest_coord = proximities[nearest_place]

    dist_km = row['Min Distance (km)']

    # Marker for launch site
    folium.Marker(
        location=site_coord,
        popup=f"{site_name}",
        icon=folium.Icon(color='blue', icon='rocket', prefix='fa')
    ).add_to(distance_map)

    # Marker for city
    folium.Marker(
        location=nearest_coord,
        popup=nearest_place,
        icon=folium.Icon(color='green', icon='info-sign')
    ).add_to(distance_map)
```

The following code of the block

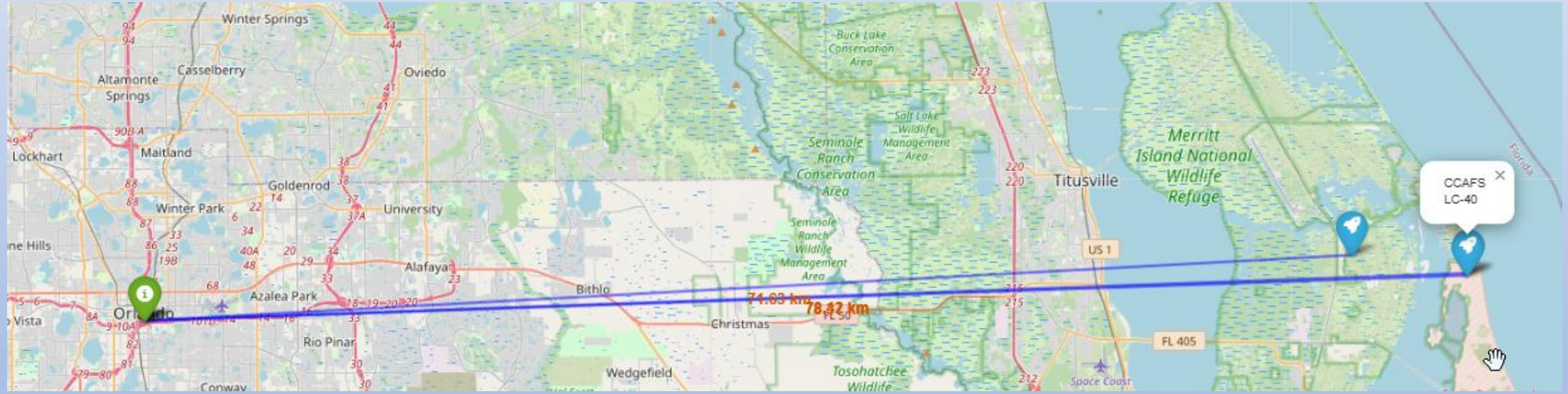
```
    # Draw a line
    folium.PolyLine(
        locations=[site_coord, nearest_coord],
        color='blue',
        weight=2,
        opacity=0.6
    ).add_to(distance_map)

    # Text with distance
    folium.Marker(
        location=[
            (site_coord[0] + nearest_coord[0]) / 2,
            (site_coord[1] + nearest_coord[1]) / 2
        ],
        icon=DivIcon(
            icon_size=(150,36),
            icon_anchor=(0,0),
            html=f'<div style="font-size: 10pt; color:#d35400;"><b>{dist_km} km</b></div>'
        )
    ).add_to(distance_map)

distance_map
```

5. Section 3 – Proximity analysis

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

Infrastructure proximity

- The screenshot displays the map with connections



5. Section 3 – Proximity analysis

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

Infrastructure proximity

- The screenshot displays the code block for the map with connections

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

      # Prerequisite: haversine(), launch_sites_df, and folium are already imported.

      points_of_interest = {
          'City: Orlando': (28.538336, -81.379234),
          'Coastline': (28.56367, -80.57163),
          'Railway': (28.572872, -80.585022),
          'Highway': (28.56318, -80.57075)
      }

      # Launch Site: CCAFS LC-40
      launch_site_name = 'CCAFS LC-40'
      launch_site_row = launch_sites_df[launch_sites_df['Launch Site'] == launch_site_name].iloc[0]
      launch_lat = launch_site_row['Lat']
      launch_lon = launch_site_row['Long']

      # List for later plotting
      connections = []

      for label, (poi_lat, poi_lon) in points_of_interest.items():
          dist = haversine(launch_lon, launch_lat, poi_lon, poi_lat)
          connections.append({
              'label': label,
              'lat': poi_lat,
              'lon': poi_lon,
              'distance': round(dist, 2)
          })
```

The following code of the block

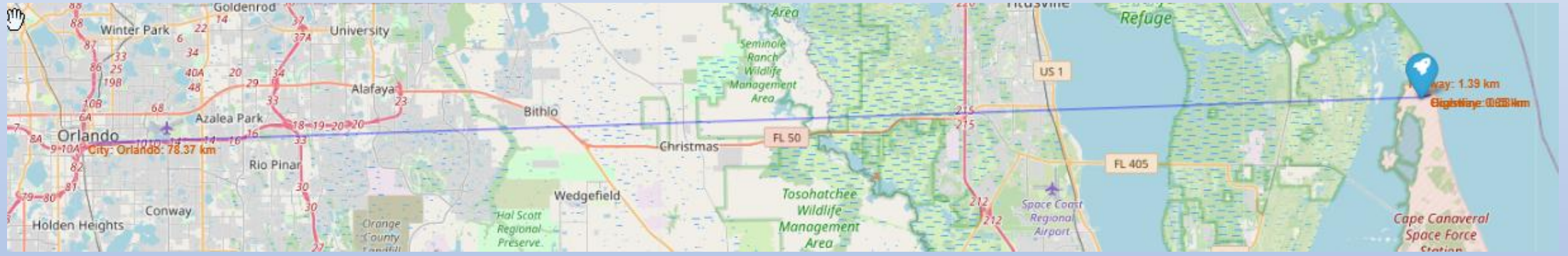
```
# New map
multi_map = folium.Map(location=[launch_lat, launch_lon], zoom_start=13)

# Marker for launch site
folium.Marker(
    location=[launch_lat, launch_lon],
    popup=launch_site_name,
    icon=folium.Icon(color='blue', icon='rocket', prefix='fa')
).add_to(multi_map)

# Mark all points (city, train, etc.)
for item in connections:
    # Marker with removal
    folium.Marker(
        location=[item['lat'], item['lon']],
        icon=DivIcon(
            icon_size=(150,36),
            icon_anchor=(0,0),
            html=f'<div style="font-size: 10px; color:#d35400;"><b>{item["label"]}: {item["distance"]} km</b></div>'
        )
    ).add_to(multi_map)

    # Connecting line
    folium.PolyLine(
        locations=[[launch_lat, launch_lon], [item['lat'], item['lon']]],
        color='blue',
        weight=1,
        opacity=0.7
    ).add_to(multi_map)

multi_map
```

5. Section 3 – Proximity analysis

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

Infrastructure proximity

- The screenshot displays the map with connections



5. Section 3 – Proximity analysis

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

Infrastructure proximity – CCAFS LC-40 detail view

Goal: Microscopic analysis of a specific location

Description:

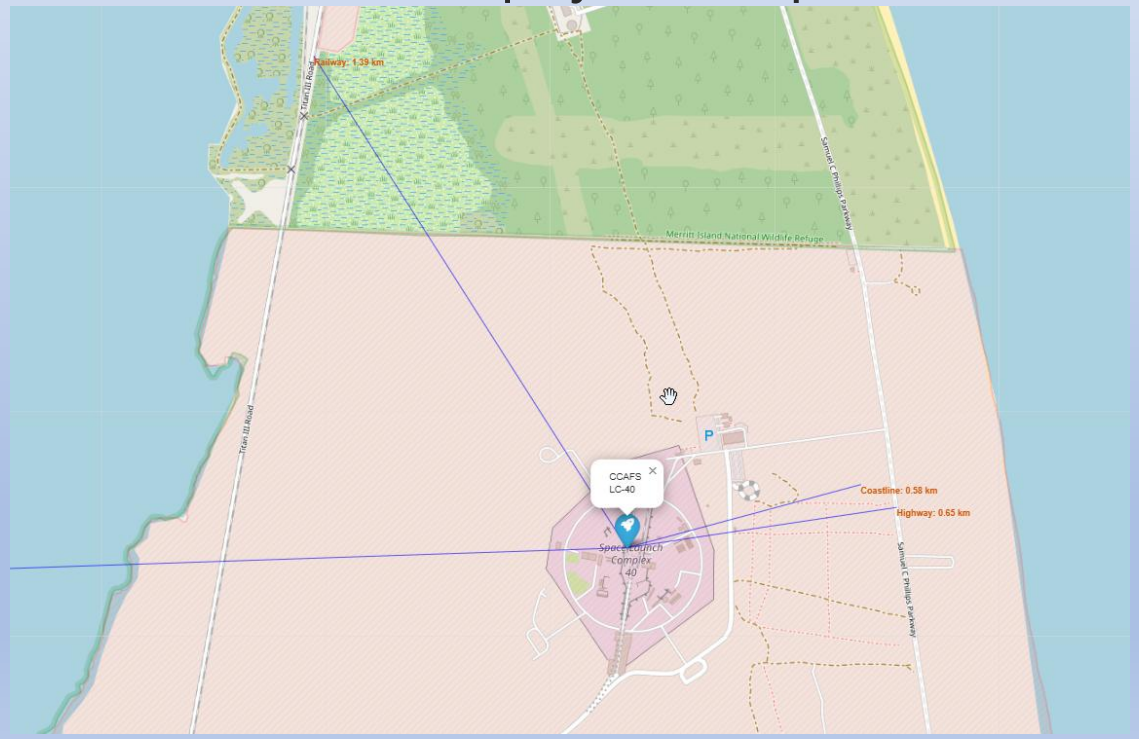
- For the launch site CCAFS LC-40, the distances to four relevant points were calculated:
  - City: Orlando
  - Coastline
  - Railway line
  - Expressway
- The map shows the spatial embedding and allows strategic evaluation of the location connection.

5. Section 3 – Proximity analysis

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

Infrastructure proximity – CCAFS LC-40 detail view

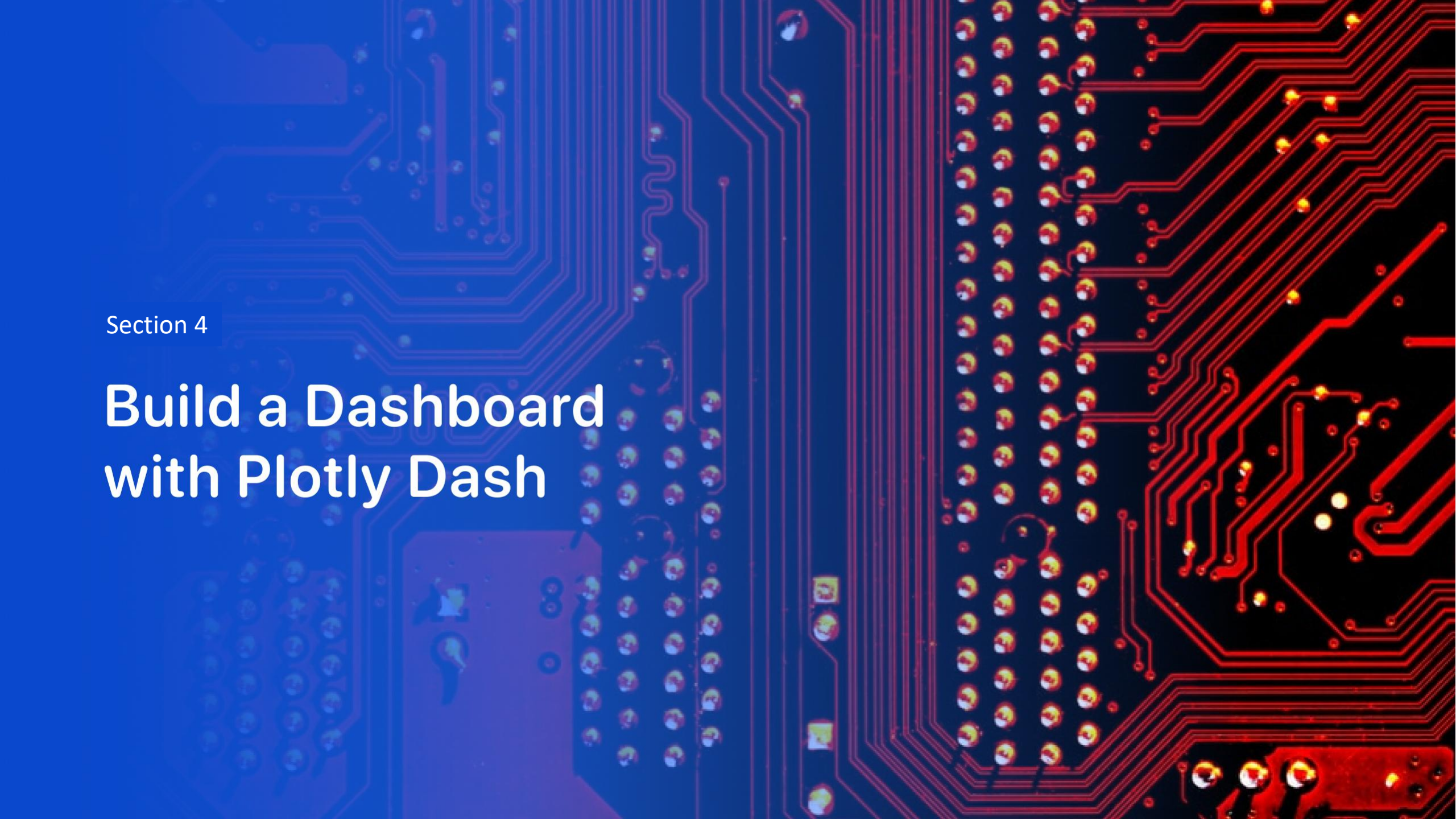
- The screenshot displays the map with connections



5. Section 3 – Summary & strategic insights

5.5 This information was conveyed via Section 3:

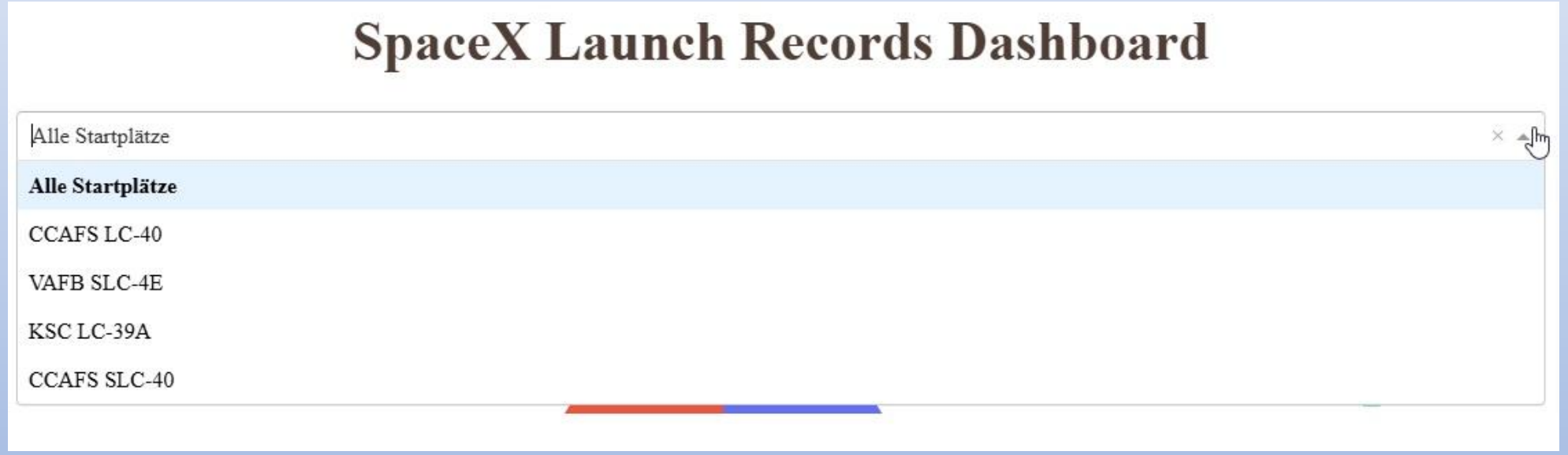
- TASK 1: Global Distribution
  - Interactive map of all launch sites
  - Insight: Coastal proximity and southern latitudes as strategic location choices
- TASK 2: Launch outcome mapping
  - Visualization of all launches with success/failure
  - Detailed analysis of individual sites
  - Insight: Error cluster and success distribution geographically traceable
- TASK 3: Infrastructure proximity
  - Calculation of distance to city, coast, railway line, and road
  - Visualization with connecting lines and distance texts
  - Insight: Location connectivity as a strategic factor for logistics and secruity


Section 4
Build a Dashboard with Plotly Dash

6. Section 4 – SpaceX Launch Records Dashboard

6.1 TASK 1: Add a dropdown list to enable Launch Site

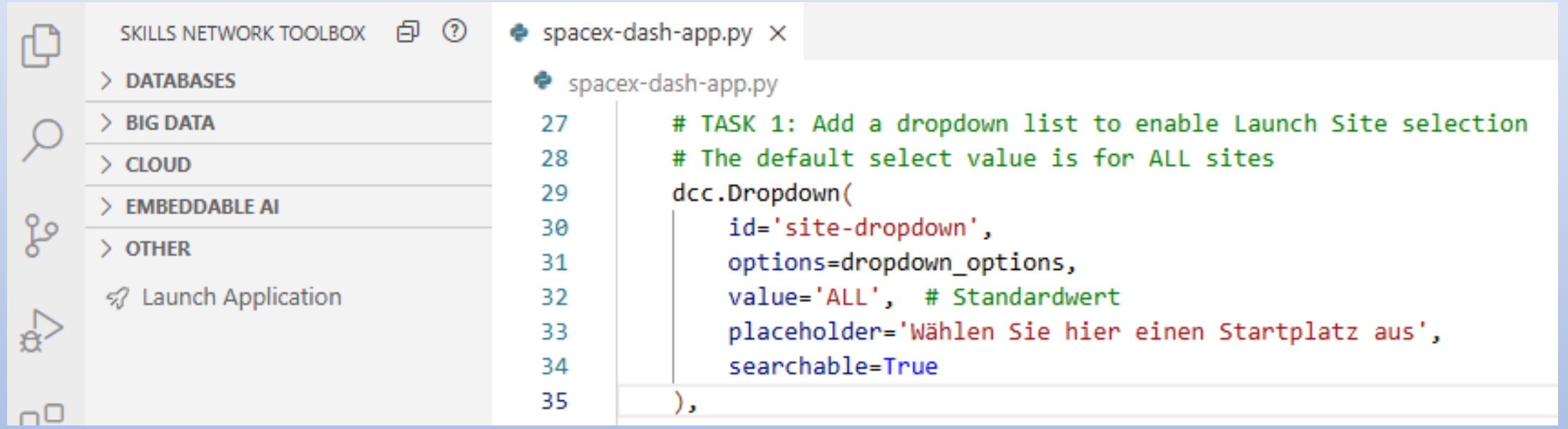
- For better analysis, I added a dropdown menu to the SpaceX Launch Records Dashboard that allows you to filter launch data by launch site.



6. Section 4 – SpaceX Launch Records Dashboard

6.1 TASK 1: Add a dropdown list to enable Launch Site

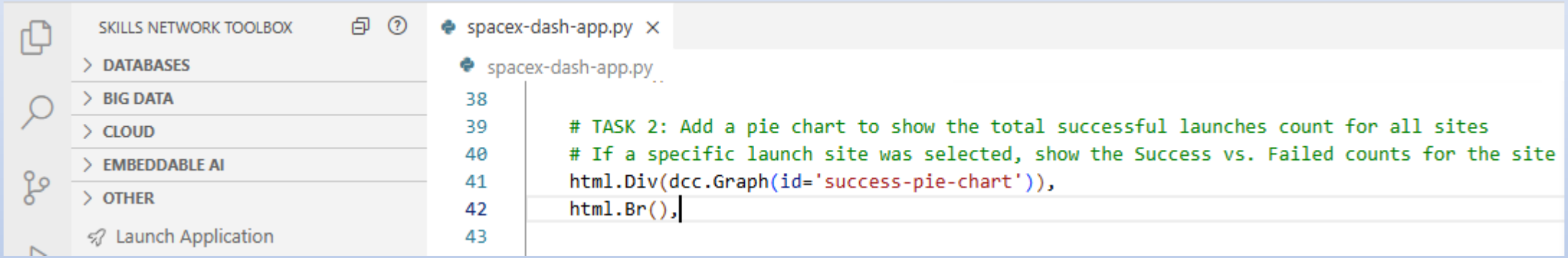
- Screenshot for Python code task 1



6. Section 4 – SpaceX Launch Records Dashboard

6.1 TASK 2: Add a pie chart to show the total successful launches count for all sites

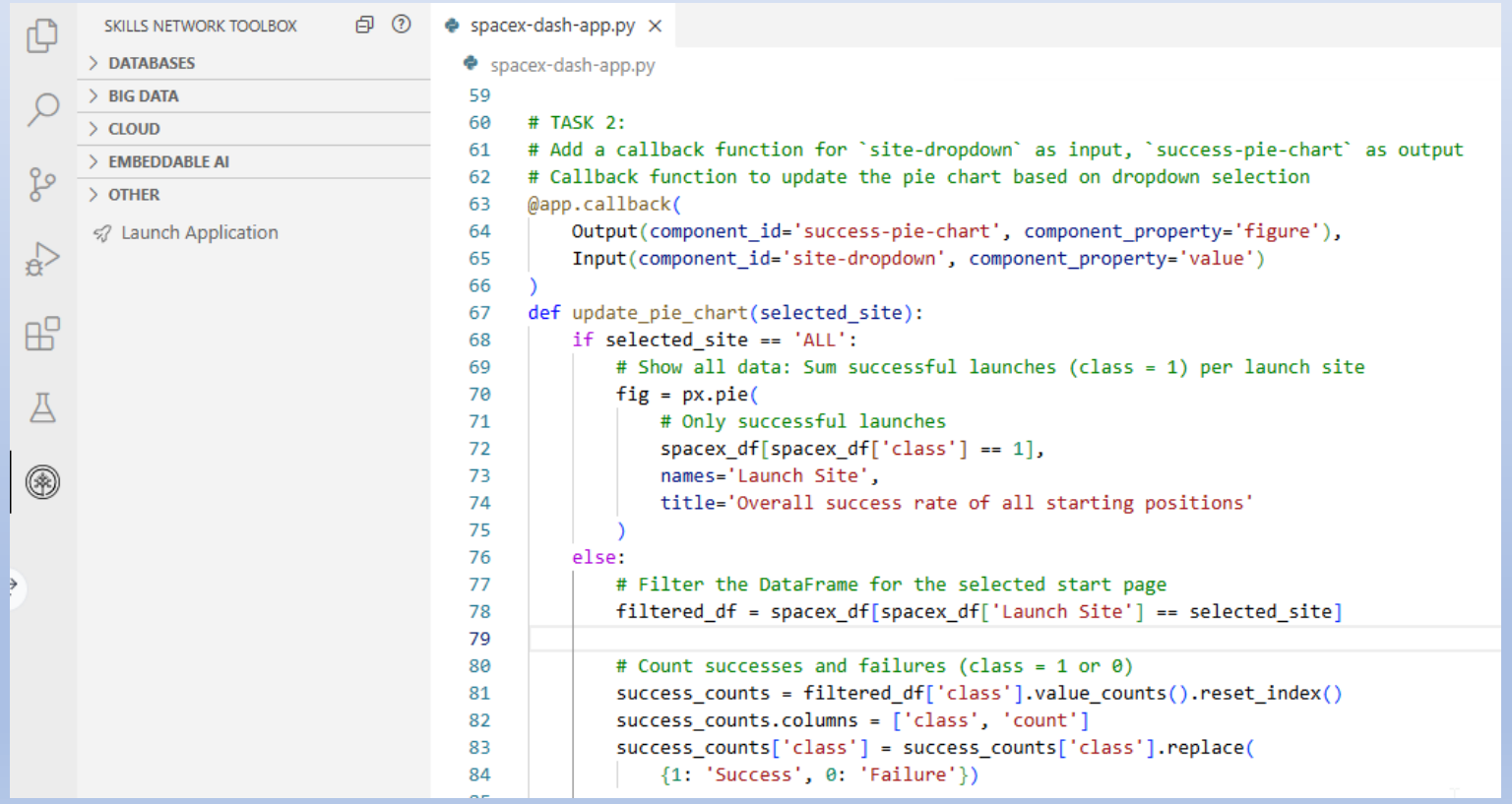
- Screenshot for Python code task 2 add a pie chart



6. Section 4 – SpaceX Launch Records Dashboard

6.1 TASK 2: Add a callback function

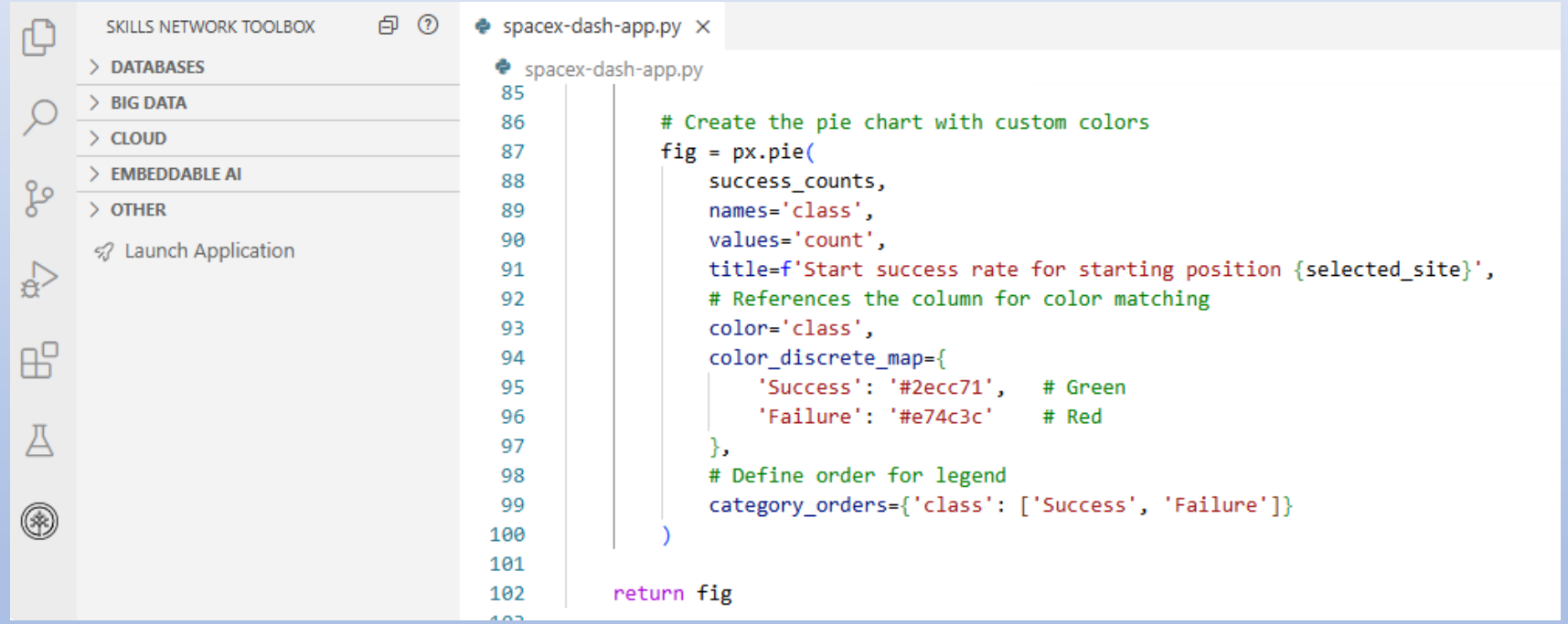
- Screenshot for Python code task 2 add a callback function



6. Section 4 – SpaceX Launch Records Dashboard

6.1 TASK 2: Add a callback function

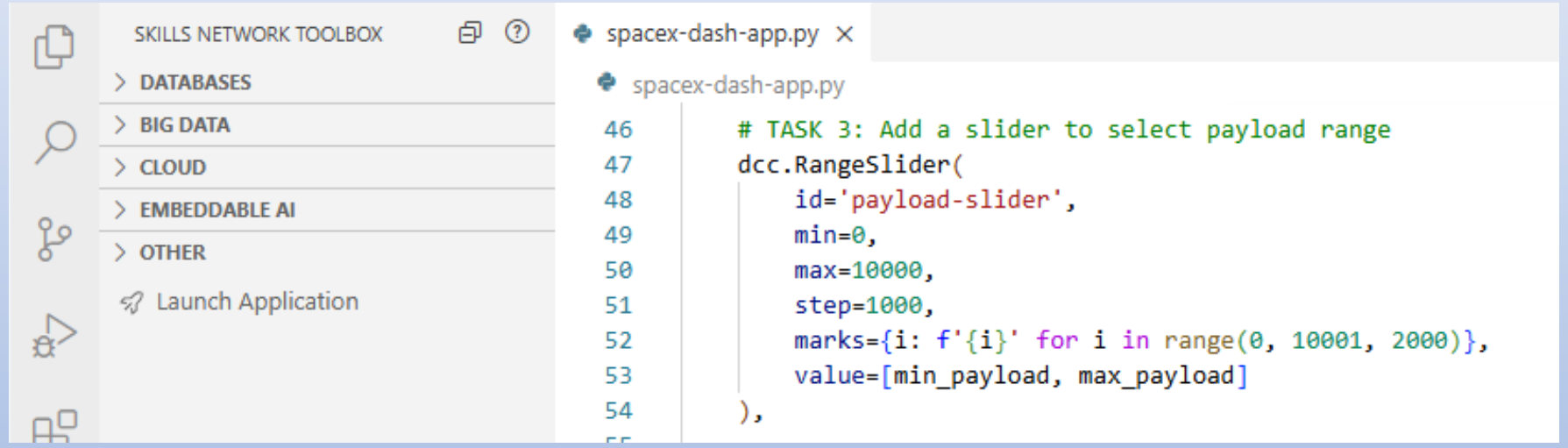
- Screenshot for Python code task 2 add a callback function



6. Section 4 – SpaceX Launch Records Dashboard

6.1 TASK 3: Add a slider to select payload range

- Screenshot for Python code task 3 add a slider

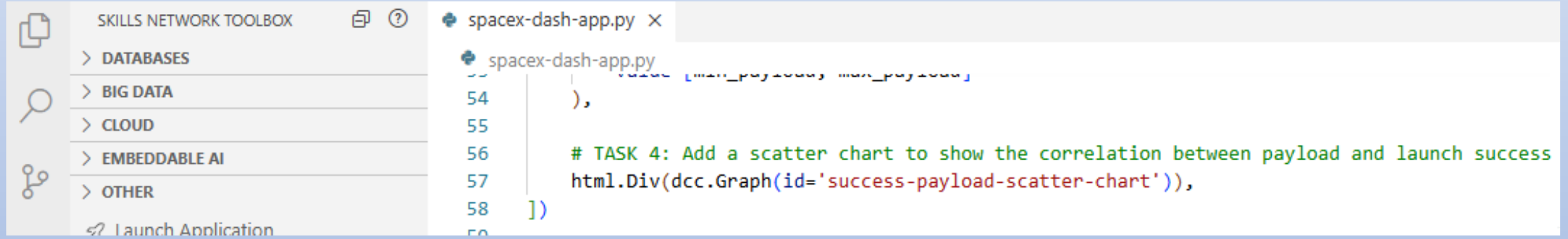


- Slider enables dynamic filtering of launch data by payload mass.
- The selected range is passed to the callback function to filter the scatter plot accordingly.

6. Section 4 – SpaceX Launch Records Dashboard

6.1 TASK 4: Add a scatter chart to show the correlation between payload and launch success

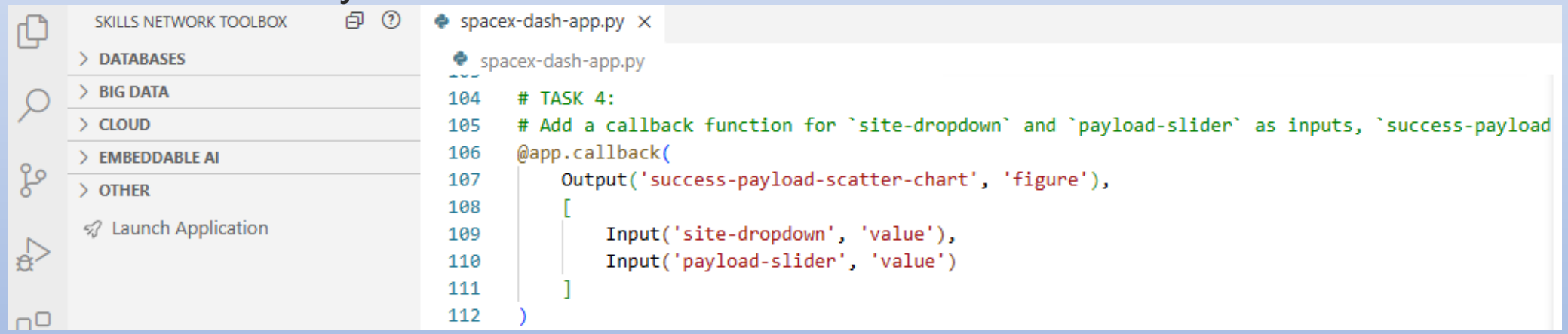
- Screenshot for Python code task 4 add a scatter chart



6. Section 4 – SpaceX Launch Records Dashboard

6.1 TASK 4: Add a callback function

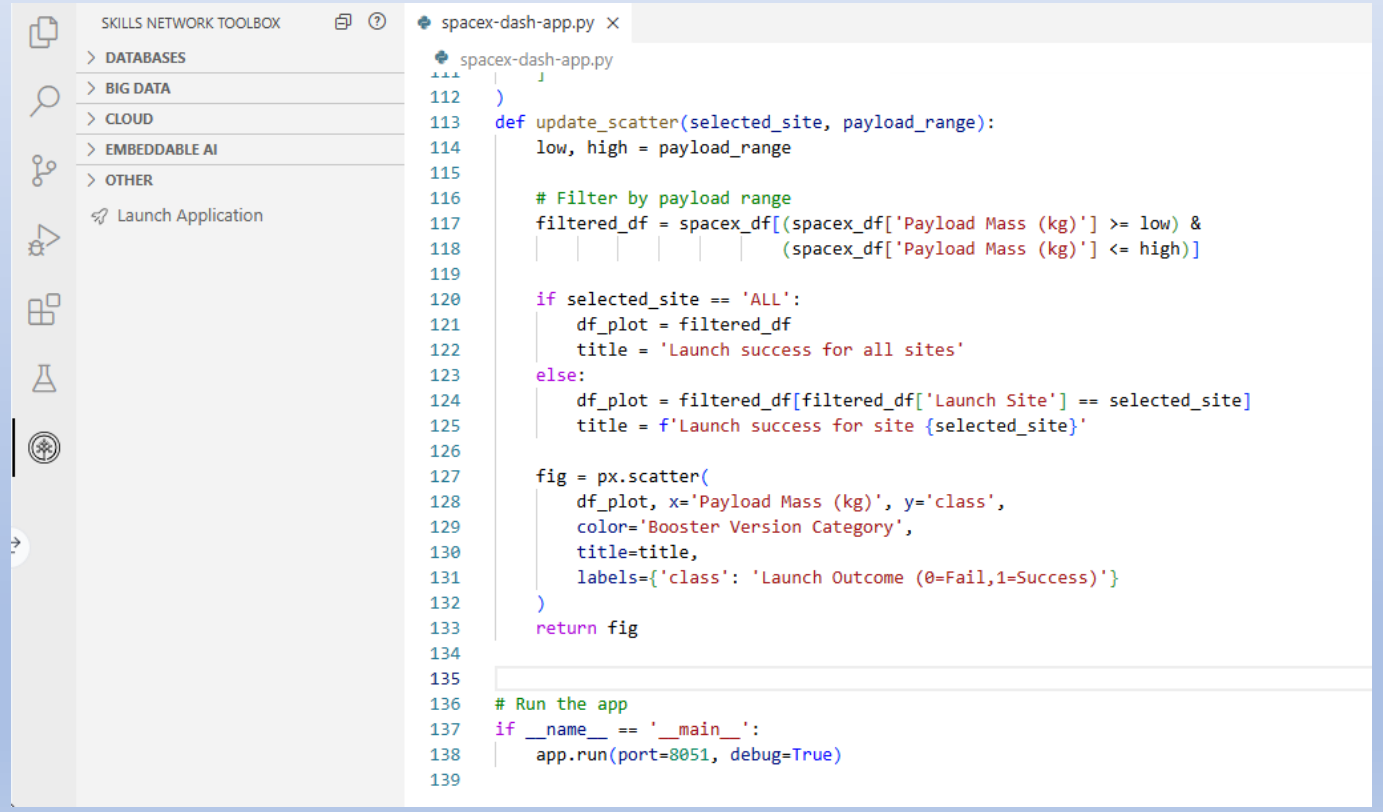
- The scatter chart visualizes the correlation between payload mass and launch success.
- The interactive behavior is implemented via a callback function shown on the next slide.
- Screenshot for Python code task 4 add a callback function



6. Section 4 – SpaceX Launch Records Dashboard

6.1 TASK 4: Add a callback function

- Screenshot for Python code task 4 add a callback function



6. Section 4 – SpaceX Launch Records Dashboard

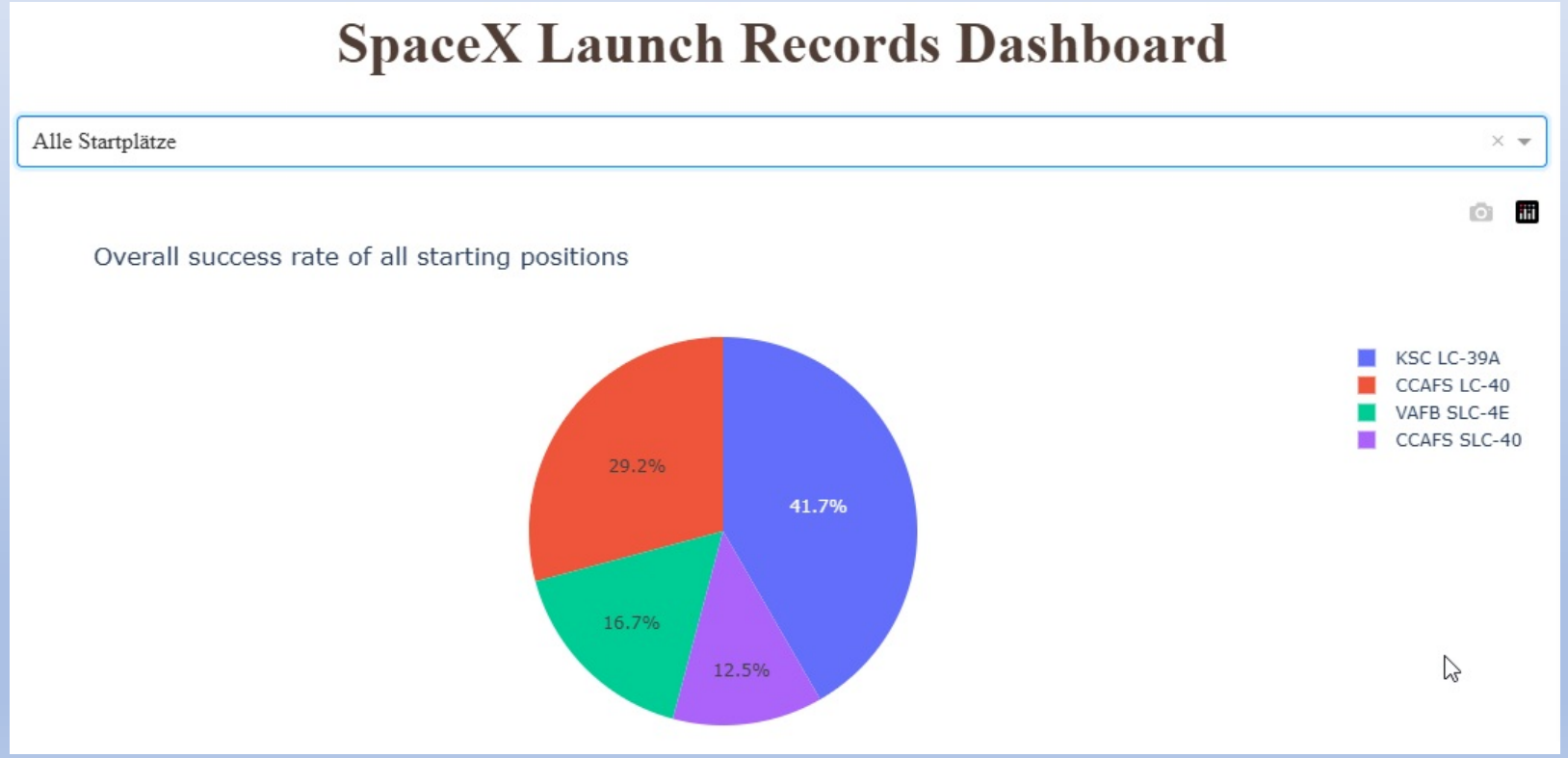
6.2 1. Which launch site has the highest number of successful launches?

- The pie chart integrated into the dashboard displays the percentage distribution of successful launches (class = 1) across all launches sites.
- The analyzed sites include:
  - KSC LC-39A
  - CCAFS LC-40
  - VAFB SLC-4E
  - CCAFS SLC-40
- The largest segment of the chart corresponds to KSC LC-39A, indicating that this site has recorded the highest number of successful launches.

6. Section 4 – SpaceX Launch Records Dashboard

6.2 1. Which launch site has the highest number of successful launches?

- Screenshot the highest number of successful launches



6. Section 4 – SpaceX Launch Records Dashboard

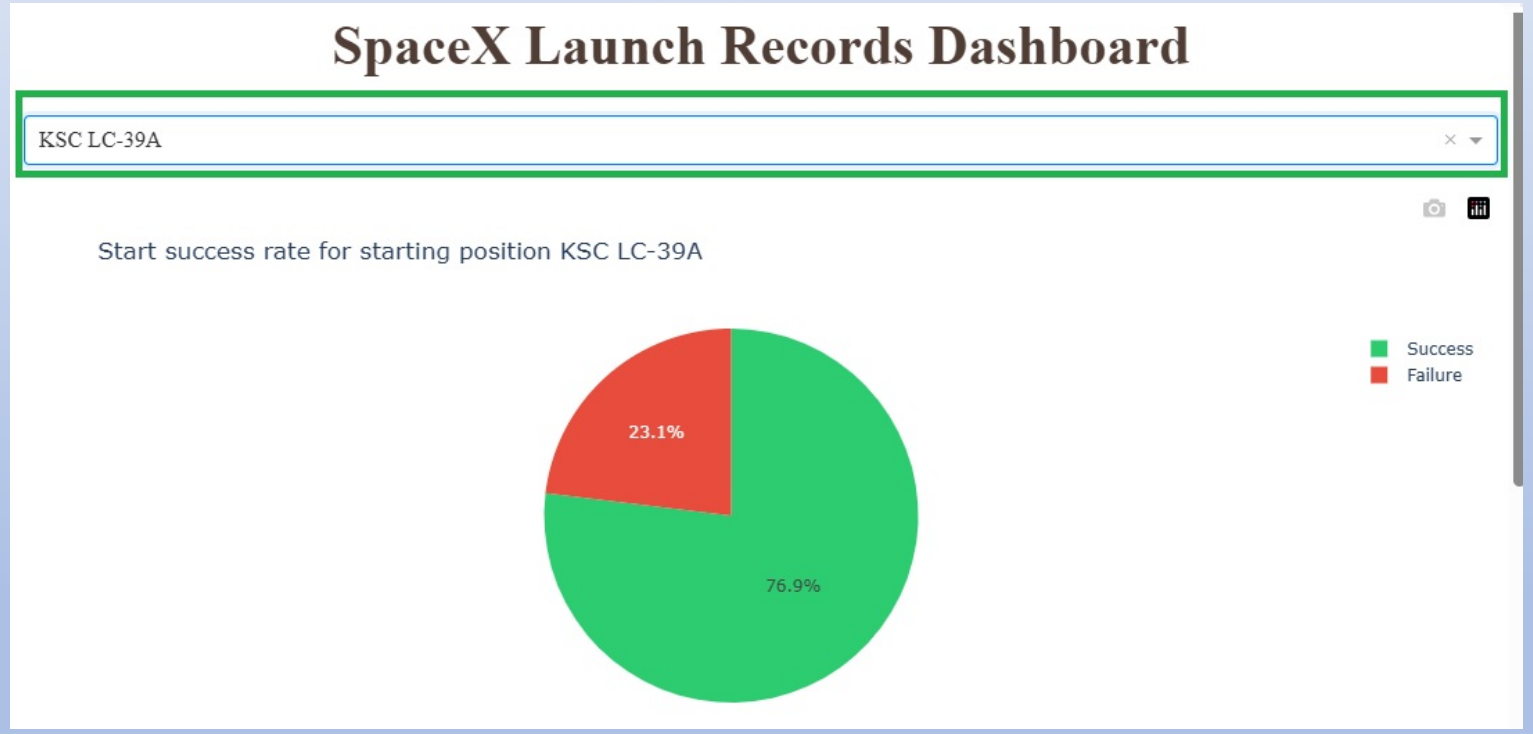
6.2 2. Which launch site has the highest success rate?

- The dashboard allows users to filter individual launch sites via the dropdown menu to analyze their respective success rates.
- Success rates are visually highlighted in the pie chart:
  - Green indicates successful launches
  - Red indicates failures
- The analysis reveals that **KSC LC-39A** achieved the highest launch success rate among all examined sites, with a value of **76.9%**.

6. Section 4 – SpaceX Launch Records Dashboard

6.2 2. Which launch site has the highest success rate?

- Screenshot the highest success rate



6. Section 4 – SpaceX Launch Records Dashboard

6.2 3. Which payload ranges show the highest launch success rate?

- To enable targeted analysis of payload ranges, I added a range slider to the dashboard. This allows users to dynamically adjust the payload interval for visualization in the scatter plot.
- The slider is configured with the following parameters:
  - min = 0 kg (starting point)
  - max = 10 000 kg (end point)
  - step = 1000 kg (interval)
  - value = [min_payload, max_payload] (currently selected range)
- The analysis reveals that payloads between approximately 2000 kg and 6000 kg achieved the highest launch success rates.

6. Section 4 – SpaceX Launch Records Dashboard

6.2 3. Which payload ranges show the highest launch success rate?

- This insight is based on the scatter plot, which shows the highest concentration of successful launches (class = 1) within this payload range.

6. Section 4 – SpaceX Launch Records Dashboard

6.2 3. Which payload ranges show the highest launch success rate?

- Screenshot the highest success rate

6. Section 4 – SpaceX Launch Records Dashboard

6.2 4. Which payload ranges show the lowest launch success rate?

- The data analysis indicates that payloads below 2000 kg and above 8000 kg are associated with the lowest success rates.
- In the scatter plot, failed launches are clearly visible as points with class = 0.
- The concentration of failures in the lower payload range likely reflects SpaceX early development phase, when launch technology was still maturing.

6. Section 4 – SpaceX Launch Records Dashboard

6.2 4. Which payload ranges show the lowest launch success rate?

- Screenshot the highest success rate

6. Section 4 – SpaceX Launch Records Dashboard

6.2 5. Which F9 booster version has the highest launch success rate?

- The scatter plot shows that the **B5** booster version appears only once, and exclusive at **class = 1** (successful launch).
- B5 appears only at class = 1 -> 100 % success rate
- Since no failures (class = 0) are recorded for B5, we can conclude that it has the **highest success rate** among the analyzed F9 variants.
- This aligns with historical data: B5 is considered the most refined version, with an almost flawless record of successful missions.
- Other booster versions show both successful and failed launches, indicating lower overall reliability compared to B5.

6. Section 4 – SpaceX Launch Records Dashboard

6.2 5. Which F9 booster version has the highest launch success rate?

- Booster Version B5: Highest Success Rate in Comparison

6. Section 4 – SpaceX Launch Records Dashboard

6.3 Summary of Findings – SpaceX Launch Records Dashboard

Question	Finding
Most successful launches	KSC LC-39A
Highest success rate (launch site)	KSC LC-39A (based on success ratio)
Highest success rate (payload range)	2000–6000 kg
Lowest success rate (payload range)	<2000 kg and >8000 kg
Best booster version for F9	B5 (100% success rate in dataset)

6. Section 4 – Summary & dashboard insights

6.3 This information was conveyed via Section 4:

- Objective
  - Development of an interactive dashboard with plotly dash
  - Integration of maps, charts, and filter functions
  - Goal: Make visual analytics for launch sites usable in real time and for exploration
- Technical implementation
  - Use dash components: dcc.Graph, html.Div, Dropdown, Slider
  - Integration of Folium maps as static or dynamic elements
  - Modular structure with callback functions for interactivity

6. Section 4 – Summary & dashboard insights

6.3 This information was conveyed via Section 4:

- Visualized content
  - ➢ TASK 1: Launch site distribution
  - ➢ TASK 2: Success/ failure rates
  - ➢ TASK 3: Infrastructure proximity
  - ➢ Filtering by location, result, distance, or infrastructure type

Section 5

Predictive Analysis (Classification)

7. Section 5 – Conclusion

7.1 Conclusion – Summary of findings

- Data from API and Wikipedia web scraping successfully combined and cleaned.
- Exploratory analysis revealed clear patterns:
  - Most frequent launch site CCAFS SLC 40
  - Dominant orbit types: ISS, VLEO
  - Most successful landings: ASDS (drone ship)
- Target variable class created (1 = successful, 0 = unsuccessful) → basis for modeling.

7. Section 5 – Conclusion

7.2 Conclusion – Evaluation of model performance

- Classification models achieved an accuracy of approximately 78%.
- Confusion matrix showed:
  - High hit rate for successful landings
  - Misclassifications mainly for rare orbit types or special outcomes.
  - Most successful landings: ASDS (drone ship)
- Models are robust but can still be optimized by adding features (payload, block, reuse count).

7. Section 5 – Conclusion

7.3 Conclusion – Outlook for future evaluations

- Expansion of features for more accurate predictions
- Comparison of different ML algorithms (e.g., logistic regression, decision trees, SVM).
- Use of the dashboard for interactive exploration and stakeholder communication.
- Potential: Forecasts for future SpaceX missions and strategic planning.

8. Section 5 – Appendix

8. Appendix – Information GitHub project

- Further information on this final project can be found in the project created on GitHub. This also includes the JupyterLabs processed for the project.
  Link: https://github.com/floriann85/DS_and_ML_Capstone_Project

8.1 Appendix – Tables, screenshots, diagrams

- Contains supplementary illustrations that were shown in the main part of the presentation:
  - Screenshots of the most important code outputs and data frames
  - Diagrams on orbit distribution, launch sites, and outcomes

8. Section 5 – Appendix

8.2 Appendix – Source code extracts

- Most parts of the code were already shown in screenshots in the main part of the presentation:
  - Python snippets for API access, web scraping, and data wrangling.
  - Code for creating the target variable Class.
  - Modeling code (training, evaluation, confusion matrix).
  - Export scripts (df.to_csv) for reproducible results.
- The GitHub repository is also open for discussion if any questions or feedback.

Thank you!