IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

<Francisco Javier Guerra Luna>
<14 Julio 2024>

Outline

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

Executive Summary

- Summary of methodologies
- Summary of all results

Introduction

- Project background and context
  - SpaceX advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is because SpaceX can reuse the first stage. Therefore if we can determine if the first stage will land, we can determine the cost of a launch. This information can be used if an alternate company wants to bid against SpaceX for a rocket launch.
  - Problems you want to find answers
    - In this capstone, we will predict if the Falcon 9 first stage will land successfully

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - Describe how data was collected :
    - By making a get request to the SpaceX API. You will also do some basic data wrangling and formating.
- Perform data wrangling
  - Describe how data was processed
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - How to build, tune, evaluate classification models

Data Collection

- Describe how data sets were collected.
- You need to present your data collection process use key phrases and flowcharts

Data Collection – SpaceX API

- Present your data collection with SpaceX REST calls using key phrases and flowcharts

- Add the GitHub URL of the completed SpaceX API calls notebook (must include completed code cell and outcome cell), as an external reference and peer-review purpose

Data was collected using the SpaceX API, which provides detailed information on rocket launches, including payload, launch site, and landing outcomes.

Title: Data Collection Flowchart

Content: Create a simple flowchart showing:

Start: Identify data requirements

Step 1: Query SpaceX API for launch data

Step 2: Retrieve datasets from additional sources

Step 3: Aggregate and preprocess data

End: Consolidate into a unified dataset

Place your flowchart of SpaceX API calls here

Link github: https://github.com/magno000/SpaceX/blob/main/jupyter-labs-spacex-data-collection-api_Sol.ipynb

Data Collection – SpaceX API

- Present your data collection with SpaceX REST calls using key phrases and flowcharts

- Add the GitHub URL of the completed SpaceX API calls notebook (must include completed code cell and outcome cell), as an external reference and peer-review purpose

From the `rocket` column we would like to learn the booster name.

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

From the `launchpad` we would like to know the name of the launch site being used, the logitude, and the latitude.

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

From the `payload` we would like to learn the mass of the payload and the orbit that it is going to.

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

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, wheter the core is reused, wheter 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.

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

Data Collection – SpaceX API

- Present your data collection with SpaceX REST calls using key phrases and flowcharts

- Add the GitHub URL of the completed SpaceX API calls notebook (must include completed code cell and outcome cell), as an external reference and peer-review purpose

Now let's start requesting rocket launch data from SpaceX API with the following URL:

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

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

Check the content of the response

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

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:

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

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

```
response.status_code
```

```
200
```

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

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

Using the dataframe `data` print the first 5 rows

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

	static_fire_date_utc	static_fire_date_unix	net	window	rocket	success	failures	details	crew	ships	capsu
0	2006-03-17T00:00:00.000Z	1.142554e+09	False	0.0	5e9d0d95eda69955f709d1eb	False	[{'time': 33, 'altitude': None, 'reason': 'merlin engine failure'}]	Engine failure at 33 seconds and loss of vehicle	[]	[]	
1	None	NaN	False	0.0	5e9d0d95eda69955f709d1eb	False	[{'time': 301, 'altitude': 289, 'reason': 'harmonic oscillation leading to premature engine shutdown'}]	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	[]	[]	

Data Collection – SpaceX API

- Present your data collection with SpaceX REST calls using key phrases and flowcharts

- Add the GitHub URL of the completed SpaceX API calls notebook (must include completed code cell and outcome cell), as an external reference and peer-review purpose

We will now use the API again to get information about the launches using the IDs given for each launch. Specifically we will be using columns `rocket`, `payloads`, `launchpad`, and `cores`.

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

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

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

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

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

```
data.head()
```

	rocket	payloads	launchpad	cores	flight_number	da
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	2006-03-24T22:30:0
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	2007-03-21T01:10:0
3	5e9d0d95eda69955f709d1eb	5eb0e4b7b6c3bb0006eeb1e5	5e9e4502f5090995de566f86	{'core': '5e9e289ef3591855dc3b2626', 'flight': 1, 'gridfins': False, 'legs': False, 'reused': False, 'landing_attempt': False, 'landing_success': None, 'landing_type': None, 'landpad': None}	4	2008-09-28T23:15:0

Data Collection – SpaceX API

- Present your data collection with SpaceX REST calls using key phrases and flowcharts

- Add the GitHub URL of the completed SpaceX API calls notebook (must include completed code cell and outcome cell), as an external reference and peer-review purpose

Finally lets construct our dataset using the data we have obtained. We we combine the columns into a dictionary.

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

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

[23]:
```
# Create a data from launch_dict
launch_df = pd.DataFrame(launch_dict)
```

Show the summary of the dataframe

[24]:
```
# Show the head of the dataframe
launch_df.head()
```

[24]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block
0	1	2006-03-24	Falcon 1	20.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN
1	2	2007-03-21	Falcon 1	NaN	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN
2	4	2008-09-28	Falcon 1	165.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN
3	5	2009-07-13	Falcon 1	200.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN
4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0

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`.

```
# Hint data['BoosterVersion']!='Falcon 1'
# Filter the DataFrame to include only Falcon 9 launches
#data_falcon9 = launch_df[launch_df['BoosterVersion'].str.contains("Falcon 9", na=False)]
data_falcon9 = launch_df[launch_df['BoosterVersion'] != 'Falcon 1']
```

```
data_falcon9.head()
```

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block
4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0
5	8	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0
6	10	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False	False	False	None	1.0
7	11	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None	1.0
8	12	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1.0

Data Collection - Scraping

- Present your web scraping process using key phrases and flowcharts

- Add the GitHub URL of the completed web scraping notebook, as an external reference and peer-review purpose

Flowchart for Web Scraping Process:

Start

Action: Identify target websites

Send HTTP Request

Action: Use Requests library to fetch web page

Tool: Requests

Receive HTML Response

Action: Obtain HTML content of the page

Parse HTML Content

Action: Use BeautifulSoup to parse HTML

Tool: BeautifulSoup

Extract Data

Action: Locate and extract data fields

Store Data

Action: Save data into DataFrame

Tool: Pandas DataFrame

End

Action: Consolidate data for analysis

Link github: https://github.com/magno000/SpaceX/blob/main/jupyter-labs-webscraping_Sol.ipynb

Data Collection - Scraping

- Present your web scraping process using key phrases and flowcharts

- Add the GitHub URL of the completed web scraping notebook, as an external reference and peer-review purpose

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

Next, request the HTML page from the above URL and get a `response` object

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.

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

```
# Check if the request was successful
if response.status_code == 200:
    print("Request successful!")
else:
    print("Request failed with status code:", response.status_code)
```

```
Request successful!
```

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

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

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

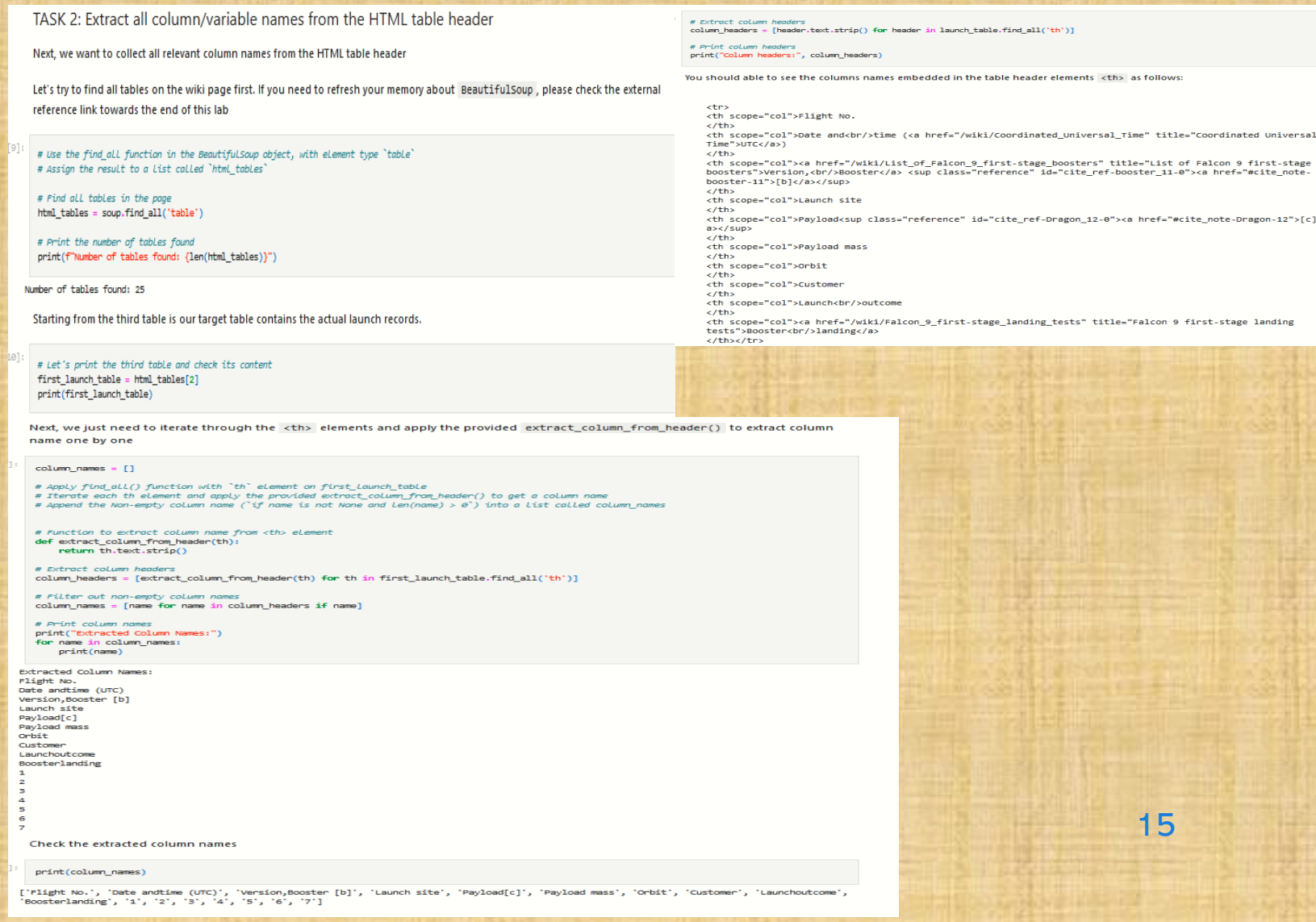
```
# Use soup.title attribute

# Check if the request was successful
if response.status_code == 200:
    print("Request successful!")

    # Parse the HTML content
    soup = BeautifulSoup(response.content, 'html.parser')

    # Print the title of the page to verify
    print("Page title:", soup.title.string)
else:
    print("Request failed with status code:", response.status_code)
```

```
Request successful!
Page title: List of Falcon 9 and Falcon Heavy launches - Wikipedia
```

Data Collection - Scraping

- Present your web scraping process using key phrases and flowcharts

- Add the GitHub URL of the completed web scraping notebook, as an external reference and peer-review purpose



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

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

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

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

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

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

Number of tables found: 25

Starting from the third table is our target table contains the actual launch records.

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

```
# Extract column headers
column_headers = [header.text.strip() for header in launch_table.find_all('th')]

# Print column headers
print("Column headers:", column_headers)
```

You should able to see the columns names embedded in the table header elements `<th>` as follows:

```
<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">[b]</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">[c]
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>
```

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

```
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


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

# Extract column headers
column_headers = [extract_column_from_header(th) for th in first_launch_table.find_all('th')]

# Filter out non-empty column names
column_names = [name for name in column_headers if name]

# Print column names
print("Extracted Column Names:")
for name in column_names:
    print(name)
```

```
Extracted Column Names:
Flight No.
Date andtime (UTC)
Version,Booster [b]
Launch site
Payload[c]
Payload mass
Orbit
Customer
Launchoutcome
Boosterlanding
1
2
3
4
5
6
7
```

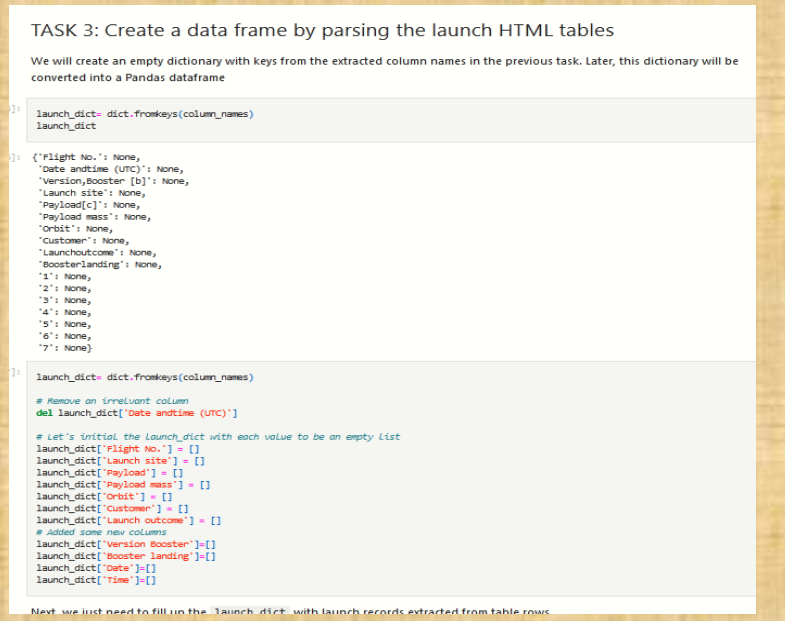
Check the extracted column names

```
print(column_names)
```

```
['Flight No.', 'Date andtime (UTC)', 'Version,Booster [b]', 'Launch site', 'Payload[c]', 'Payload mass', 'Orbit', 'Customer', 'Launchoutcome',
'Boosterlanding', '1', '2', '3', '4', '5', '6', '7']
```

Data Collection - Scraping

- Present your web scraping process using key phrases and flowcharts
- Add the GitHub URL of the completed web scraping notebook, as an external reference and peer-review purpose



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

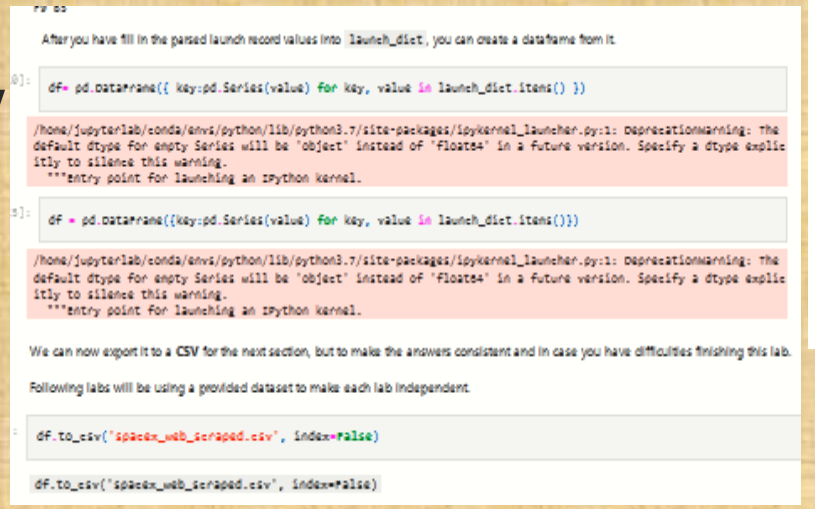
We will create an empty dictionary with keys from the extracted column names in the previous task. Later, this dictionary will be converted into a Pandas dataframe

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

```
{'Flight No.': None,
 'Date andtime (UTC)': None,
 'Version,Booster [b]': None,
 'Launch site': None,
 'Payload[c]': None,
 'Payload mass': None,
 'Orbit': None,
 'Customer': None,
 'Launchoutcome': None,
 'Boosterlanding': None,
 '1': None,
 '2': None,
 '3': None,
 '4': None,
 '5': None,
 '6': None,
 '7': None}
```

```
launch_dict= dict.fromkeys(column_names)

# Remove an irrelvant column
del launch_dict['Date andtime (UTC)']

# 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']=[]
```

Next, we just need to fill up the `launch_dict` with launch records extracted from table rows.



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

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

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/ipykernel_launcher.py:1: DeprecationWarning: The default dtype for empty Series will be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to silence this warning.
  """Entry point for launching an IPython kernel.
```

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

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/ipykernel_launcher.py:1: DeprecationWarning: The default dtype for empty Series will be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to silence this warning.
  """Entry point for launching an IPython kernel.
```

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)
```

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

Next, we just need to fill up the `launch_dict` with launch records extracted from table rows.

Usually, HTML tables in Wild pages are likely to contain unexpected annotations and other types of noises, such as reference links `B0004.1[8]`, missing values `N/A [e]`, inconsistent formatting, etc.

To simplify the parsing process, we have provided an incomplete code snippet below to help you to fill up the `launch_dict`. Please complete the following code snippet with TODOs or you can choose to write your own logic to parse all launch tables:

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

            # Date value
            # TODO: Append the date into launch_dict with key `Date`
            date = datatimelist[0].strip(",")
            #print(date)

            # Time value
            # TODO: Append the time into launch_dict with key `Time`
            time = datatimelist[1]
            #print(time)

            # Booster version
            # TODO: Append the bv into launch_dict with key `Version Booster`
            bv=booster_version(row[1])
            if not(bv):
                bv=row[1].a.string
            print(bv)

            # Launch Site
            # TODO: Append the bv into launch_dict with key `Launch Site`
            launch_site = row[2].a.string
            #print(launch_site)

            # Payload
            # TODO: Append the payload into launch_dict with key `Payload`
            payload = row[3].a.string
            #print(payload)

            # Payload Mass
            # TODO: Append the payload_mass into launch_dict with key `Payload Mass`
            payload_mass = get_mass(row[4])
            #print(payload)

            # Orbit
            # TODO: Append the orbit into launch_dict with key `Orbit`
            orbit = row[5].a.string
            #print(orbit)

            # Customer
            # TODO: Append the customer into launch_dict with key `Customer`
            #customer = row[6].a.string
            #print(customer)

             # Customer
            customer = row[6].a.get_text().strip() if row[6].a else None


            # Launch Outcome
            # TODO: Append the launch_outcome into launch_dict with key `Launch Outcome`
            launch_outcome = list(row[7].strings)[0]
            #print(launch_outcome)

            # Booster Landing
            # TODO: Append the launch_outcome into launch_dict with key `Booster Landing`
            booster_landing = landing_status(row[8])
            #print(booster_landing)
```

```
F9 v1.0B0003.1
F9 v1.0B0004.1
F9 v1.0B0005.1
F9 v1.0B0006.1
F9 v1.0B0007.1
F9 v1.1B1003
```

Data Wrangling

- Describe how data were processed
  - We will perform some Exploratory Data Analysis (EDA) to find some patterns in the data and determine what would be the label for training supervised models.
  - In the data set, there are several different cases where the booster did not land successfully. Sometimes a landing was attempted but failed due to an accident; for example, True Ocean means the mission outcome was successfully landed to a specific region of the ocean while False Ocean means the mission outcome was unsuccessfully landed to a specific region of the ocean. True RTLS means the mission outcome was successfully landed to a ground pad False RTLS means the mission outcome was unsuccessfully landed to a ground pad.True ASDS means the mission outcome was successfully landed on a drone ship False ASDS means the mission outcome was unsuccessfully landed on a drone ship.
  - In this lab we will mainly convert those outcomes into Training Labels with 1 means the booster successfully landed 0 means it was unsuccessful.
- You need to present your data wrangling process using key phrases and flowcharts
- Add the GitHub URL of your completed data wrangling related notebooks, as an external reference and peer-review purpose

Data Wrangling

- Describe how data were processed
- You need to present your data wrangling process using key phrases and flowcharts
- Add the GitHub URL of your completed data wrangling related notebooks, as an external reference and peer-review purpose

Data Analysis

Load Space X dataset, from last section.

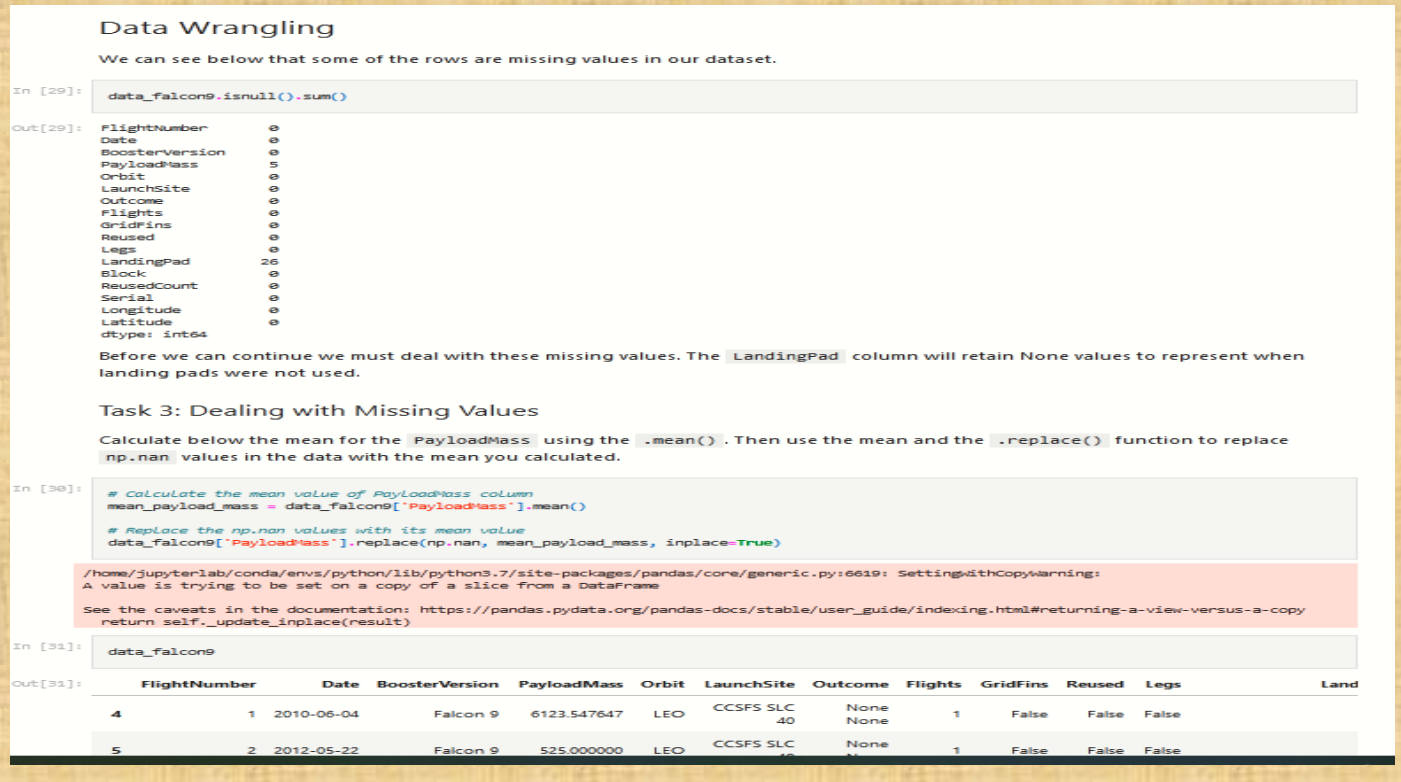
```
df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_1.csv")
df.head(10)
```

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0
5	6	2014-01-06	Falcon 9	3325.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0
6	7	2014-04-18	Falcon 9	2296.000000	ISS	CCAFS SLC 40	True Ocean	1	False	False	True	NaN	1.0
7	8	2014-07-14	Falcon 9	1316.000000	LEO	CCAFS SLC 40	True Ocean	1	False	False	True	NaN	1.0
8	9	2014-08-05	Falcon 9	4535.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0
9	10	2014-09-07	Falcon 9	4428.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0

Link github:
https://github.com/magno000/SpaceX/blob/main/labs-jupyter-spacex-Data%20wrangling_Sol.ipynb

Data Wrangling

- Describe how data were processed
- You need to present your data wrangling process using key phrases and flowcharts
- Add the GitHub URL of your completed data wrangling related notebooks, as an external reference and peer-review purpose



Data Wrangling

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

```
In [29]: data_falcon9.isnull().sum()
```

```
Out[29]: 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
```

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.

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.

```
In [30]: # Calculate the mean value of PayloadMass column
mean_payload_mass = data_falcon9['PayloadMass'].mean()

# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'].replace(np.nan, mean_payload_mass, inplace=True)
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/pandas/core/generic.py:6619: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return self._update_inplace(result)
```

```
In [31]: data_falcon9
```

Out[31]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	Land
4	1	2010-06-04	Falcon 9	6123.547647	LEO	CCSFS SLC 40	None None	1	False	False	False	
5	2	2012-05-22	Falcon 9	525.000000	LEO	CCSFS SLC	None	1	False	False	False	

Link github:
https://github.com/magno000/SpaceX/blob/main/labs-jupyter-spacex-Data%20wrangling_Sol.ipynb

Data Wrangling

- Describe how data were processed
- You need to present your data wrangling process using key phrases and flowcharts
- Add the GitHub URL of your completed data wrangling related notebooks, as an external reference and peer-review purpose

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

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

```
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
```

Identify which columns are numerical and categorical:

```
df.dtypes
```

```
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
```

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:

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

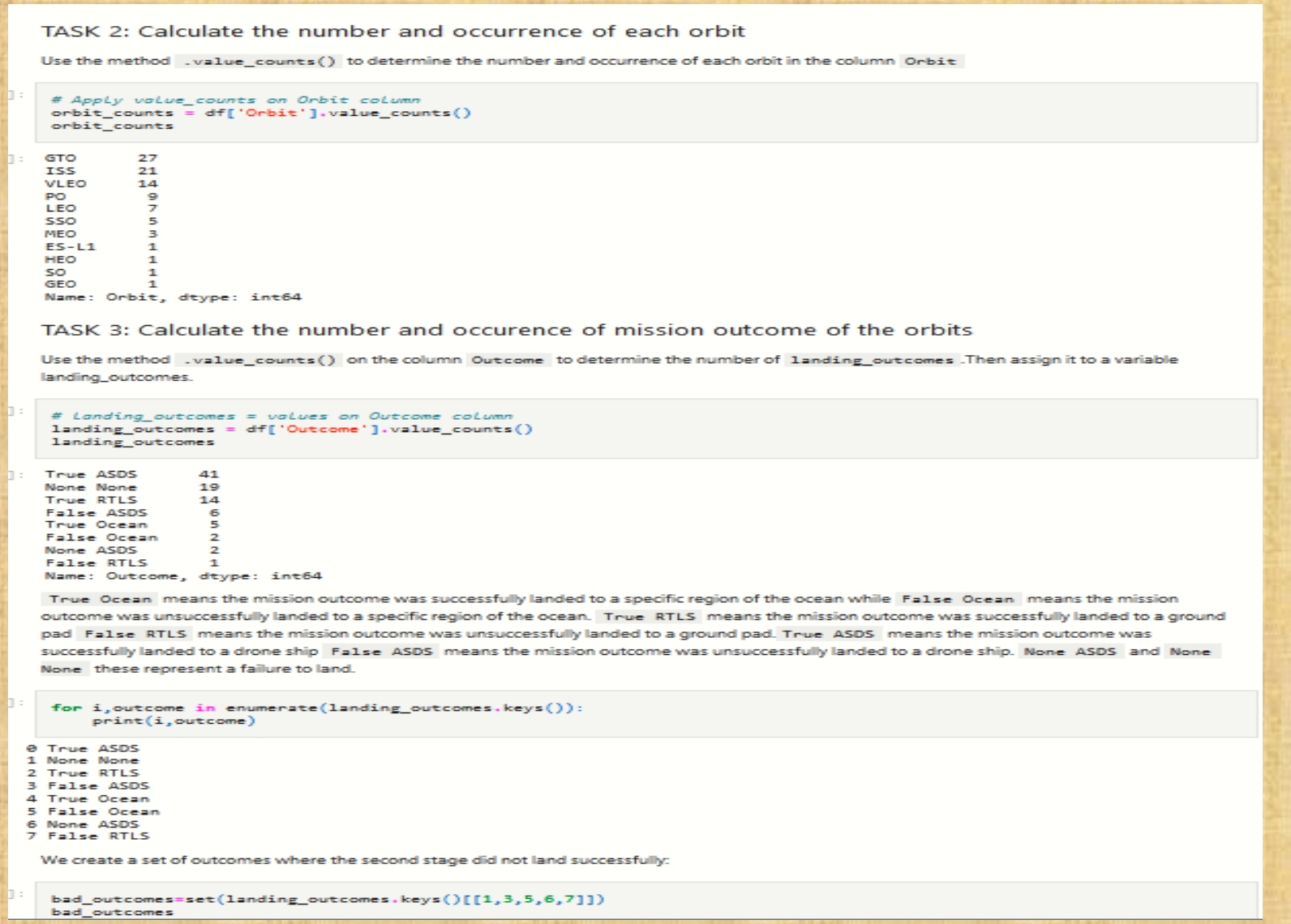
```
CCAFS SLC 40    55
KSC LC 39A      22
VAFB SLC 4E     13
Name: LaunchSite, dtype: int64
```

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

Link github:
https://github.com/magno000/SpaceX/blob/main/labs-jupyter-spacex-Data%20wrangling_Sol.ipynb

Data Wrangling

- Describe how data were processed
- You need to present your data wrangling process using key phrases and flowcharts
- Add the GitHub URL of your completed data wrangling related notebooks, as an external reference and peer-review purpose



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`

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

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

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.

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

```
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: Outcome, dtype: int64
```

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

```
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
```

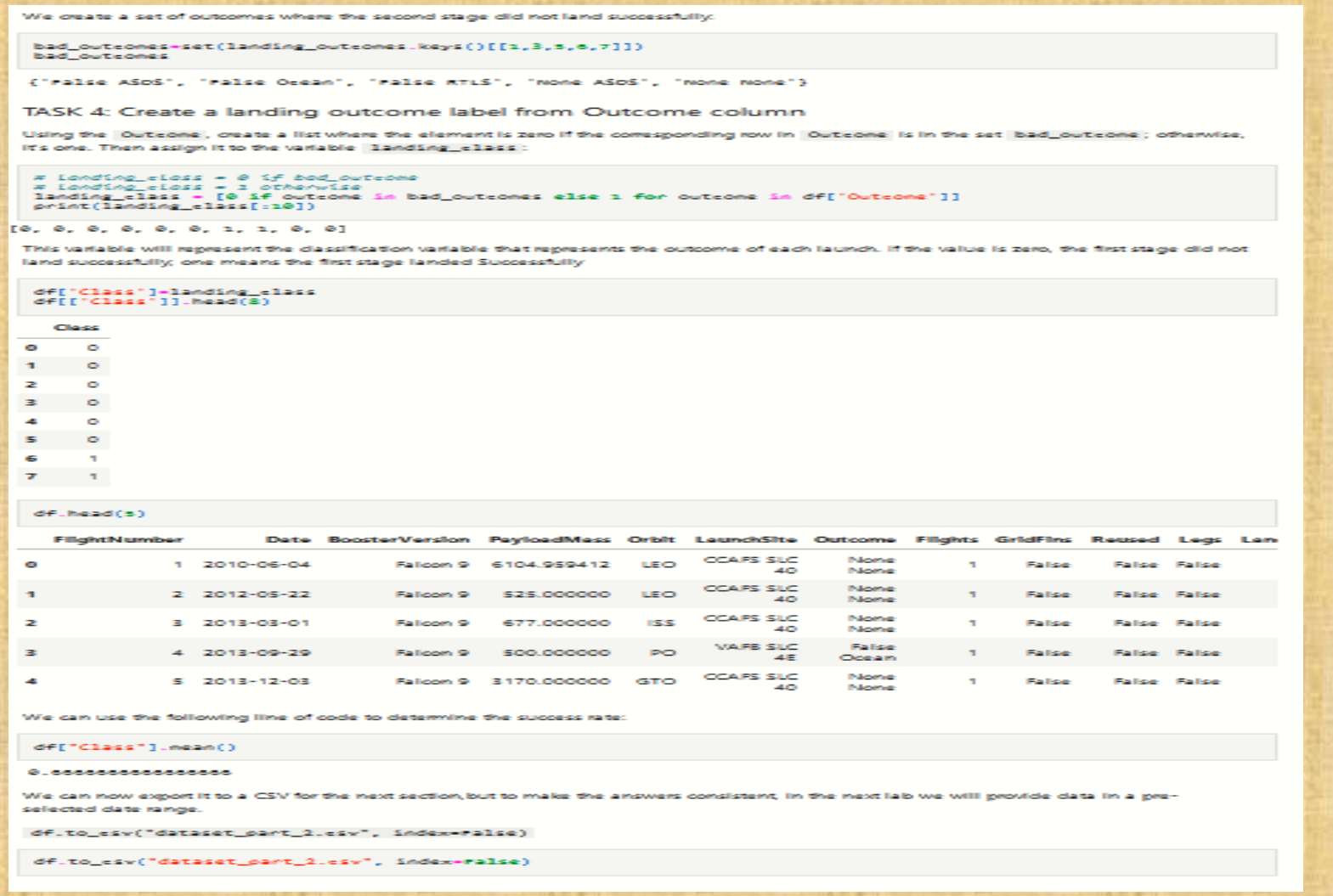
We create a set of outcomes where the second stage did not land successfully:

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

Link github:
https://github.com/magno000/SpaceX/blob/main/labs-jupyter-spacex-Data%20wrangling_Sol.ipynb

Data Wrangling

- Describe how data were processed
- You need to present your data wrangling process using key phrases and flowcharts
- Add the GitHub URL of your completed data wrangling related notebooks, as an external reference and peer-review purpose



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

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

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

TASK 4: Create a landing outcome label from Outcome column

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

```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
landing_class = [0 if outcome in bad_outcomes else 1 for outcome in df['Outcome']]
print(landing_class[:10])
```

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

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

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

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

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

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	Lan
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	

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

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

```
0.6666666666666666
```

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)
```

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

Link github:
https://github.com/magno000/SpaceX/blob/main/labs-jupyter-spacex-Data%20wrangling_Sol.ipynb

EDA with Data Visualization

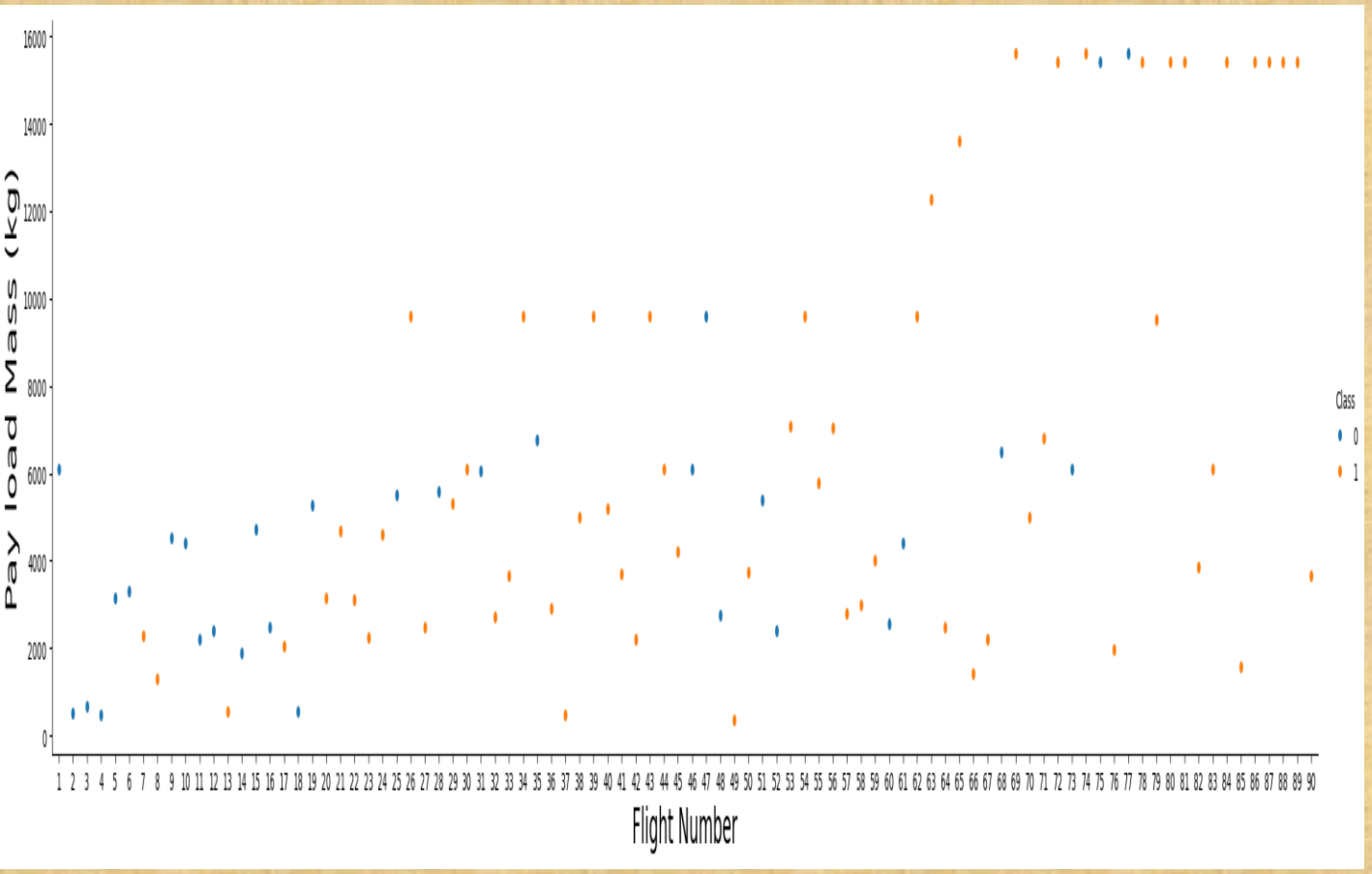
- Summarize what charts were plotted and why you used those charts

  - In this assignment, we will predict if the Falcon 9 first stage will land successfully. SpaceX advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is due to the fact that SpaceX can reuse the first stage.
  - we will perform Exploratory Data Analysis and Feature Engineering.
  - Falcon 9 first stage will land successfully

- Add the GitHub URL of your completed EDA with data visualization notebook, as an external reference and peer-review purpose:

  - https://github.com/magno000/SpaceX/blob/main/edadataviz_Sol.ipynb

EDA with Data Visualization




```
from js import fetch
import io

URL = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_2.csv"
resp = await fetch(URL)
dataset_part_2_csv = io.BytesIO((await resp.arrayBuffer()).to_py())
df=pd.read_csv(dataset_part_2_csv)
df.head(5)
```

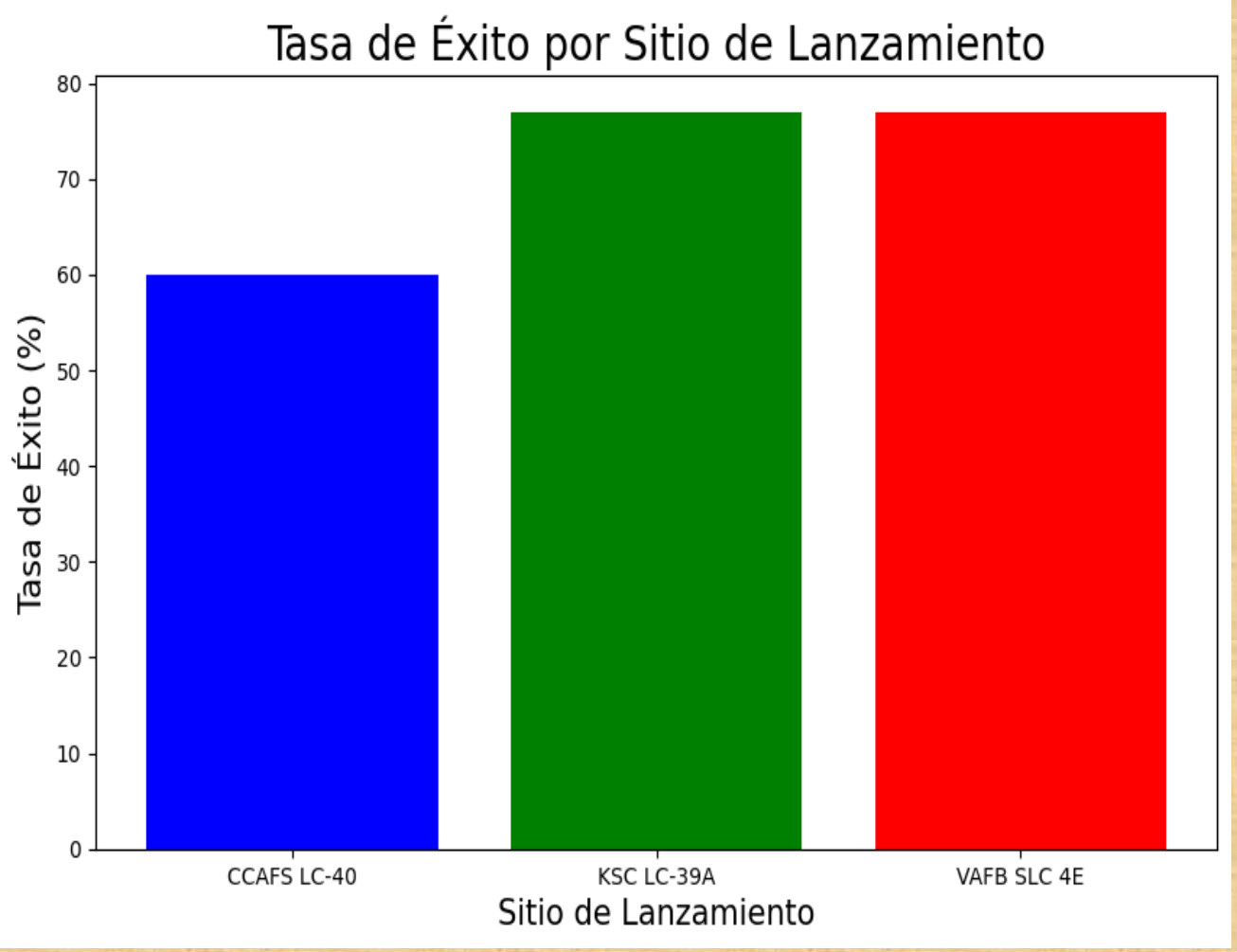
	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Blo
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	

First, let's try to see how the `FlightNumber` (indicating the continuous launch attempts.) and `Payload` variables would affect the launch outcome.

We can plot out the `FlightNumber` vs. `PayloadMass` and overlay the outcome of the launch. We see that as the flight number increases, the first stage is more likely to land successfully. The payload mass is also important; it seems the more massive the payload, the less likely the first stage will return.

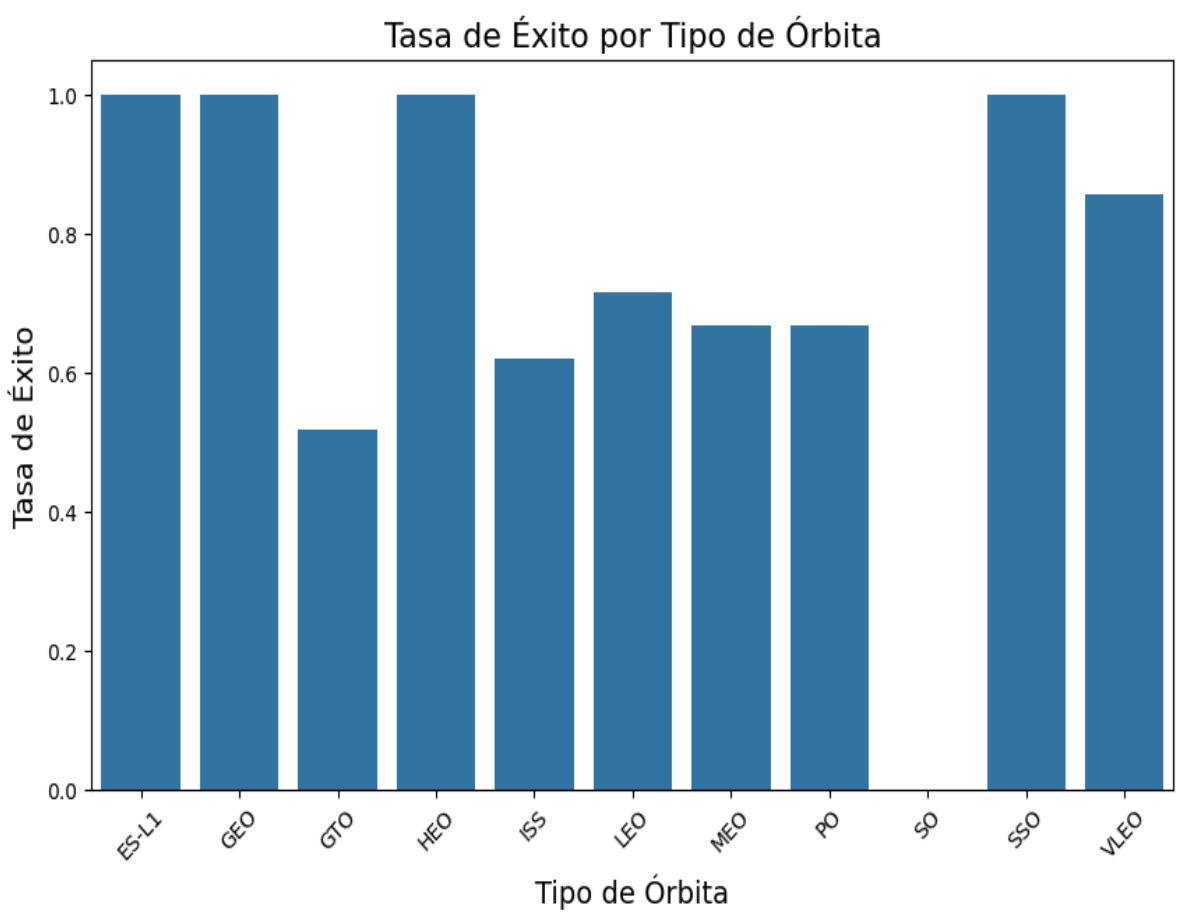
```
sns.catplot(y="PayloadMass", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Pay load Mass (kg)",fontsize=20)
plt.show()
```

EDA with Data Visualization




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

# Datos de las tasas de éxito
launch_sites = ['CCAFS LC-40', 'KSC LC-39A', 'VAFB SLC 4E']
success_rates = [60, 77, 77]

# Crear la gráfica de barras
plt.figure(figsize=(10, 6))
plt.bar(launch_sites, success_rates, color=['blue', 'green', 'red'])

# Añadir títulos y etiquetas
plt.title('Tasa de Éxito por Sitio de Lanzamiento', fontsize=20)
plt.xlabel('Sitio de Lanzamiento', fontsize=15)
plt.ylabel('Tasa de Éxito (%)', fontsize=15)

# Mostrar la gráfica
plt.show()
```

EDA with Data Visualization




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

# Calcular la tasa de éxito para cada tipo de órbita
success_rate_by_orbit = df.groupby('Orbit')['Class'].mean().reset_index()
success_rate_by_orbit.columns = ['Orbit', 'Success Rate']

# Crear el gráfico de barras
plt.figure(figsize=(10, 6))
sns.barplot(x='Orbit', y='Success Rate', data=success_rate_by_orbit)
plt.xlabel('Tipo de Órbita', fontsize=14)
plt.ylabel('Tasa de Éxito', fontsize=14)
plt.title('Tasa de Éxito por Tipo de Órbita', fontsize=16)
plt.xticks(rotation=45)
plt.show()
```

EDA with SQL

- Using bullet point format, summarize the SQL queries you performed
  - QUERYS:
    - Display the names of the unique launch sites in the space mission
    - Display 5 records where launch sites begin with the string 'CCA'
    - Display the total payload mass carried by boosters launched by NASA (CRS)
    - Display average payload mass carried by booster version F9 v1.1
    - List the date when the first succesful landing outcome in ground pad was acheived.
    - List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000.
    - List the total number of successful and failure mission outcomes
    - List the names of the booster_versions which have carried the maximum payload mass. Use a subquery.
    - List the records which will display the month names, failure landing_outcomes in drone ship ,booster versions, launch_site for the months in year 2015.
    - 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.
- Add the GitHub URL of your completed EDA with SQL notebook, as an external reference and peer-review purpose:
  - https://github.com/magno000/SpaceX/blob/main/jupyter-labs-eda-sql-coursera_sqllite_Sol.ipynb

EDA with SQL

Task 1

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

```
[8]: query = 'SELECT DISTINCT "Launch_Site" FROM SPACEXTABLE'

[9]: cur.execute(query)
     launch_sites = cur.fetchall()

10]: for site in launch_sites:
         print(site[0])
```

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

Task 2

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

```
11]: query = '''
     SELECT * FROM SPACEXTABLE
     WHERE "Launch_Site" LIKE 'CCA%'
     LIMIT 5
     '''

12]: cur.execute(query)
     records = cur.fetchall()

13]: for record in records:
         print(record[0])
```

```
2010-06-04
2010-12-08
2012-05-22
2012-10-08
```

Task 3

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

```
1]: # Consulta SQL para obtener la masa total de carga útil transportada por los cohetes lanzados por NASA (CRS)
    query = '''
    SELECT SUM("PAYLOAD_MASS__KG_") as total_payload_mass
    FROM SPACEXTABLE
    WHERE "Customer" = 'NASA (CRS)'
    '''

2]: cur.execute(query)
    total_payload_mass = cur.fetchone()[0]
    print(total_payload_mass)
```

```
45596
```

```
3]: print("Total payload mass carried by boosters launched by NASA (CRS):", total_payload_mass, "kg")
```

```
Total payload mass carried by boosters launched by NASA (CRS): 45596 kg
```

Task 4

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

```
7]: query = '''
    SELECT AVG("PAYLOAD_MASS__KG_") as average_payload_mass
    FROM SPACEXTABLE
    WHERE "Booster_Version" = 'F9 v1.1'
    '''

    cur.execute(query)
    average_payload_mass = cur.fetchone()[0]

    print("Average payload mass carried by booster version F9 v1.1:", average_payload_mass, "kg")
```

```
Average payload mass carried by booster version F9 v1.1: 2928.4 kg
```

EDA with SQL

Task 5

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

Hint:Use min function

```
query = '''
SELECT MIN(Date) as first_successful_landing_date
FROM SPACEXTABLE
WHERE "Landing_Outcome" = 'Success (ground pad)'
'''

cur.execute(query)
first_successful_landing_date = cur.fetchone()[0]

print("Date of the first successful landing on a ground pad:", first_successful_landing_date)
```

```
Date of the first successful landing on a ground pad: 2015-12-22
```

Task 6

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

```
# Consulta SQL para obtener los nombres de los boosters con éxito en nave dron y masa de carga útil entre 4000 y 6000 kg
query = '''
SELECT "Booster_Version"
FROM SPACEXTABLE
WHERE "Landing_Outcome" = 'Success (drone ship)'
AND "PAYLOAD_MASS__KG_" > 4000
AND "PAYLOAD_MASS__KG_" < 6000
'''

# Ejecutar la consulta y obtener los resultados
cur.execute(query)
booster_names = cur.fetchall()

# Mostrar los resultados
for booster in booster_names:
    print("Booster Version:", booster[0])
```

```
Booster Version: F9 FT B1022
Booster Version: F9 FT B1026
Booster Version: F9 FT  B1021.2
Booster Version: F9 FT  B1031.2
```

Task 7

List the total number of successful and failure mission outcomes

```
# Consulta SQL para contar el número total de resultados de misiones exitosas y fallidas
query = '''
SELECT "Landing_Outcome", COUNT(*) as total
FROM SPACEXTABLE
GROUP BY "Landing_Outcome"
'''

# Ejecutar la consulta y obtener los resultados
cur.execute(query)
outcome_counts = cur.fetchall()

# Mostrar los resultados
for outcome, count in outcome_counts:
    print("Landing Outcome:", outcome, "| Total:", count)
```

```
Landing Outcome: Controlled (ocean) | Total: 5
Landing Outcome: Failure | Total: 3
Landing Outcome: Failure (drone ship) | Total: 5
Landing Outcome: Failure (parachute) | Total: 2
Landing Outcome: No attempt | Total: 21
Landing Outcome: No attempt  | Total: 1
Landing Outcome: Precluded (drone ship) | Total: 1
Landing Outcome: Success | Total: 38
Landing Outcome: Success (drone ship) | Total: 14
Landing Outcome: Success (ground pad) | Total: 9
Landing Outcome: Uncontrolled (ocean) | Total: 2
```

Task 8

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

```
# Consulta SQL para listar las versiones de los propulsores que han transportado la masa de carga útil máxima
query = '''
SELECT "Booster_Version"
FROM SPACEXTABLE
WHERE "PAYLOAD_MASS__KG_" = (
    SELECT MAX("PAYLOAD_MASS__KG_")
    FROM SPACEXTABLE
)
'''

# Ejecutar la consulta y obtener los resultados
cur.execute(query)
booster_versions = cur.fetchall()

# Mostrar los resultados
print("Booster versions with the maximum payload mass:")
for booster in booster_versions:
    print(booster[0])
```

```
Booster versions with the maximum payload mass:
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
```

EDA with SQL

Task 9

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

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

```
# Consulta SQL para listar los registros que cumplen con los criterios especificados
query = '''
SELECT
    CASE
        WHEN substr(Date, 6, 2) = '01' THEN 'January'
        WHEN substr(Date, 6, 2) = '02' THEN 'February'
        WHEN substr(Date, 6, 2) = '03' THEN 'March'
        WHEN substr(Date, 6, 2) = '04' THEN 'April'
        WHEN substr(Date, 6, 2) = '05' THEN 'May'
        WHEN substr(Date, 6, 2) = '06' THEN 'June'
        WHEN substr(Date, 6, 2) = '07' THEN 'July'
        WHEN substr(Date, 6, 2) = '08' THEN 'August'
        WHEN substr(Date, 6, 2) = '09' THEN 'September'
        WHEN substr(Date, 6, 2) = '10' THEN 'October'
        WHEN substr(Date, 6, 2) = '11' THEN 'November'
        WHEN substr(Date, 6, 2) = '12' THEN 'December'
    END AS Month,
    Landing_Outcome,
    Booster_Version,
    Launch_Site
FROM SPACEXTABLE
WHERE substr(Date, 0, 5) = '2015'
    AND Landing_Outcome = 'Failure (drone ship)'
'''

# Ejecutar la consulta y obtener los resultados
cur.execute(query)
records = cur.fetchall()

# Mostrar los resultados
print("Records for failure landing outcomes on drone ship in year 2015:")
for record in records:
    print(record)
```

```
Records for failure landing outcomes on drone ship in year 2015:
('January', 'Failure (drone ship)', 'F9 v1.1 B1012', 'CCAFS LC-40')
('April', 'Failure (drone ship)', 'F9 v1.1 B1015', 'CCAFS LC-40')
```

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.

```
# Consulta SQL para clasificar el recuento de resultados de aterrizaje entre las fechas especificadas
query = '''
SELECT
    Landing_Outcome,
    COUNT(*) as Count_Landing_Outcome
FROM SPACEXTABLE
WHERE Date BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY Landing_Outcome
ORDER BY Count_Landing_Outcome DESC
'''

# Ejecutar la consulta y obtener los resultados
cur.execute(query)
records = cur.fetchall()

# Mostrar los resultados
print("Ranking of landing outcomes between 2010-06-04 and 2017-03-20:")
for record in records:
    print(record)

# Cerrar la conexión
conn.close()
```

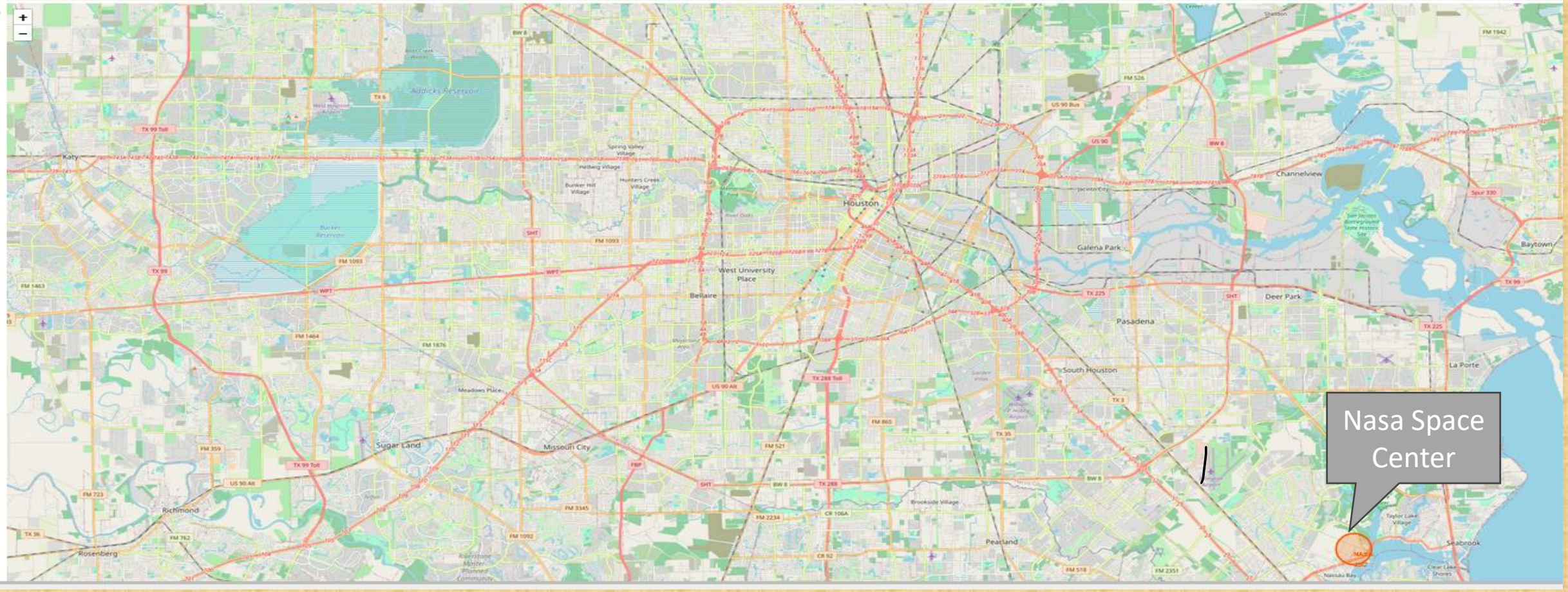
```
Ranking of landing outcomes between 2010-06-04 and 2017-03-20:
('No attempt', 10)
('Success (drone ship)', 5)
('Failure (drone ship)', 5)
('Success (ground pad)', 3)
('Controlled (ocean)', 3)
('Uncontrolled (ocean)', 2)
('Failure (parachute)', 2)
('Precluded (drone ship)', 1)
```

Build an Interactive Map with Folium

- Summarize what map objects such as markers, circles, lines, etc. you created and added to a folium map
  - Finding an optimal location for building a launch site certainly involves many factors and hopefully we could discover some of the factors by analyzing the existing launch site locations.
- Explain why you added those objects
  - First, let's try to add each site's location on a map using site's latitude and longitude coordinates.
  - Let's try to enhance the map by adding the launch outcomes for each site, and see which sites have high success rates.
  - Calculate the distances between a launch site to its proximities.
- Add the GitHub URL of your completed interactive map with Folium map, as an external reference and peer-review purpose:
  - https://github.com/magno000/SpaceX/blob/main/lab_jupyter_launch_site_location_Sol.ipynb

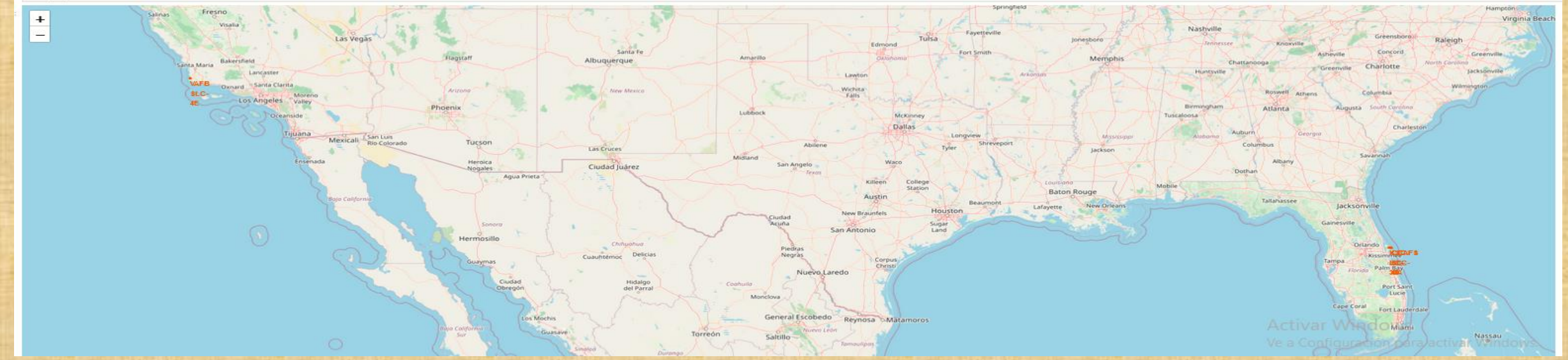
Build an Interactive Map with Folium

```
# 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)
```



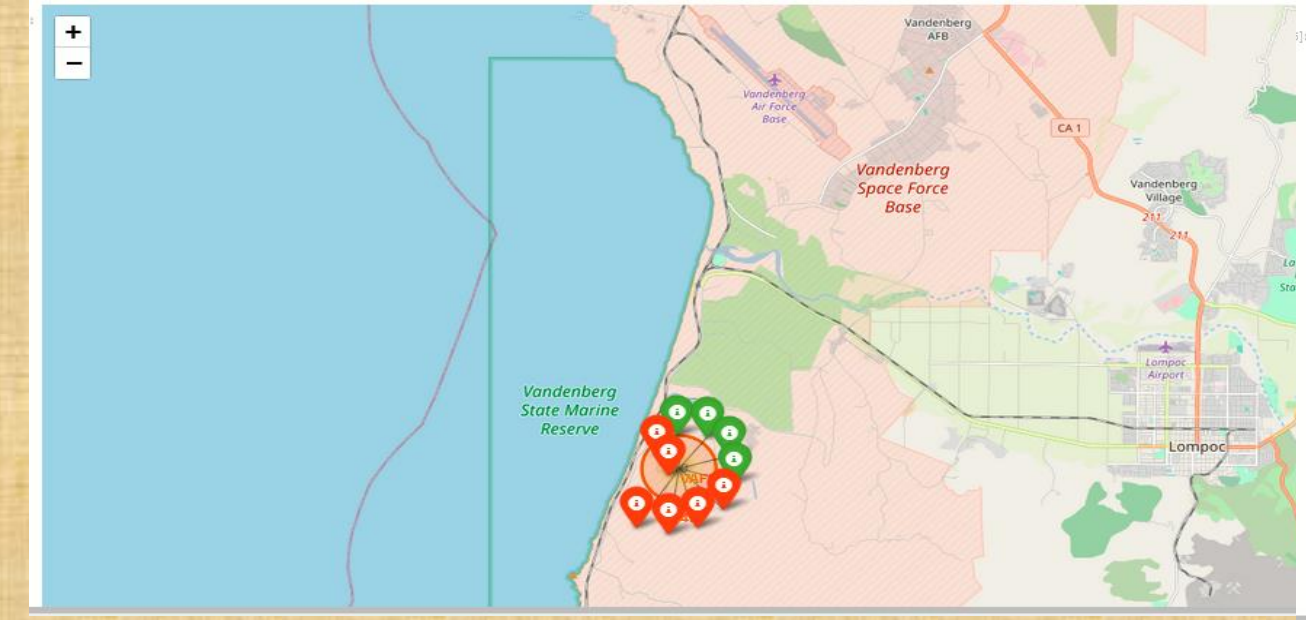
Build an Interactive Map with Folium

```
# Initial the map
site_map = folium.Map(location=nasa_coordinate, zoom_start=5)
# For each launch site, add a Circle object based on its coordinate (Lat, Long) values. In addition, add Launch site name as a popup label
# Añadir un círculo y marcador para cada sitio de Lanzamiento
for _, row in launch_sites_df.iterrows():
    # Extraer coordenadas y nombre del sitio
    site_name = row['Launch Site']
    coordinate = [row['Lat'], row['Long']]

    # Crear un círculo en el sitio de lanzamiento
    circle = folium.Circle(coordinate, radius=1000, color='#d35400', fill=True).add_child(folium.Popup(site_name))
    # Crear un marcador en el sitio de lanzamiento
    marker = folium.map.Marker(
        coordinate,
        icon=DivIcon(
            icon_size=(20, 20),
            icon_anchor=(0, 0),
            html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % site_name,
        )
    )
    # Añadir círculo y marcador al mapa
    site_map.add_child(circle)
    site_map.add_child(marker)

# Mostrar el mapa
site_map
```

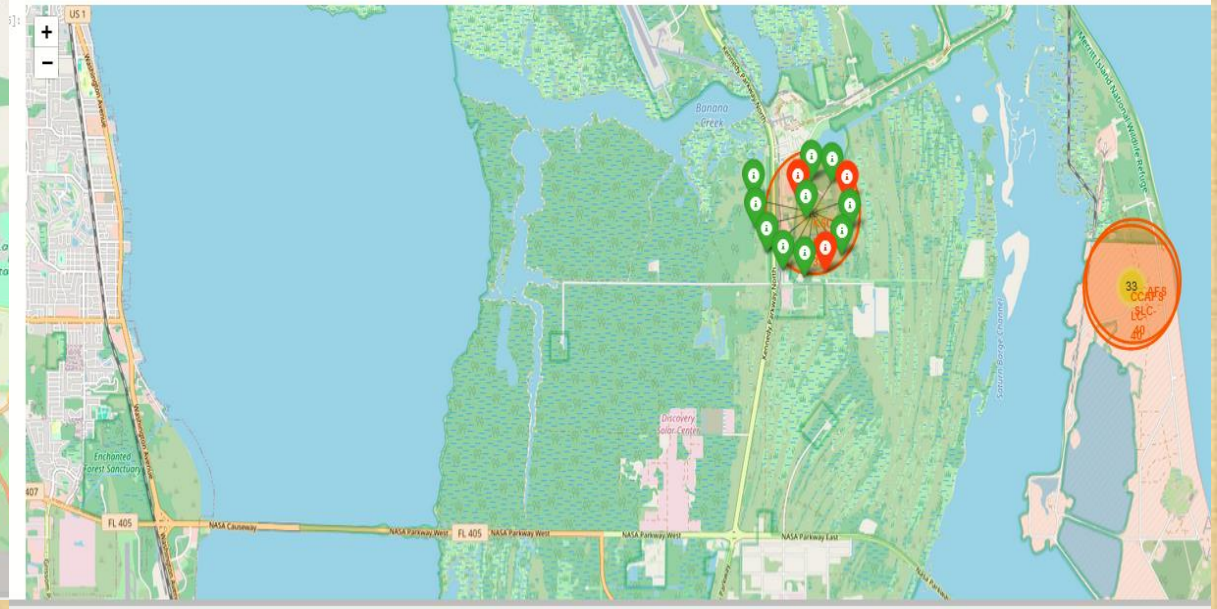


Build an Interactive Map with Folium

```
# Crear un objeto MarkerCluster
marker_cluster = MarkerCluster().add_to(site_map)

# Añadir marcadores para cada registro de lanzamiento
for _, row in spacex_df.iterrows():
    # Extraer coordenadas y color del marcador
    coordinate = [row['Lat'], row['Long']]
    marker_color = row['marker_color']

    # Crear y añadir el marcador al cluster
    marker = folium.Marker(
        location=coordinate,
        icon=folium.Icon(color=marker_color),
        popup=f"{row['Launch Site']}: {'Success' if row['class'] == 1 else 'Failed'}"
    )
    marker.add_to(marker_cluster)

# Mostrar el mapa
site_map
```

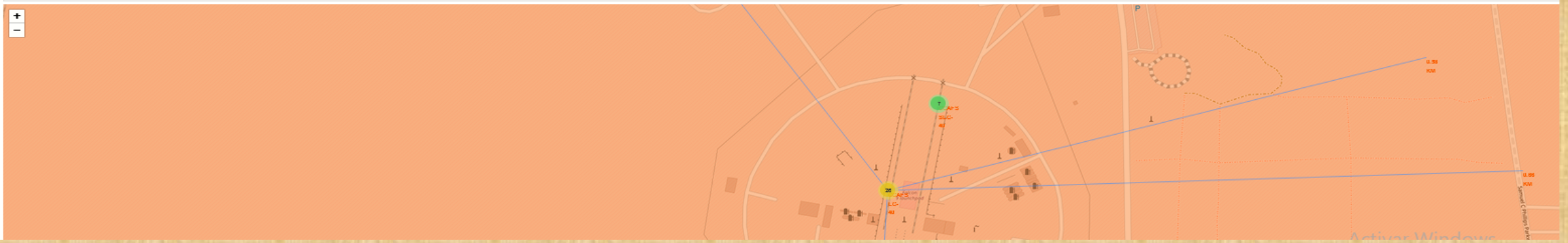



```
# Crear un objeto MarkerCluster
marker_cluster = MarkerCluster().add_to(site_map)

# Añadir marcadores para cada registro de lanzamiento
for _, row in spacex_df.iterrows():
    # Extraer coordenadas y color del marcador
    coordinate = [row['Lat'], row['Long']]
    marker_color = row['marker_color']

    # Crear y añadir el marcador al cluster
    marker = folium.Marker(
        location=coordinate,
        icon=folium.Icon(color=marker_color),
        popup=f"{row['Launch Site']}: {'Success' if row['class'] == 1 else 'Failed'}"
    )
    marker.add_to(marker_cluster)

# Mostrar el mapa
site_map
```

Build an Interactive Map with Folium

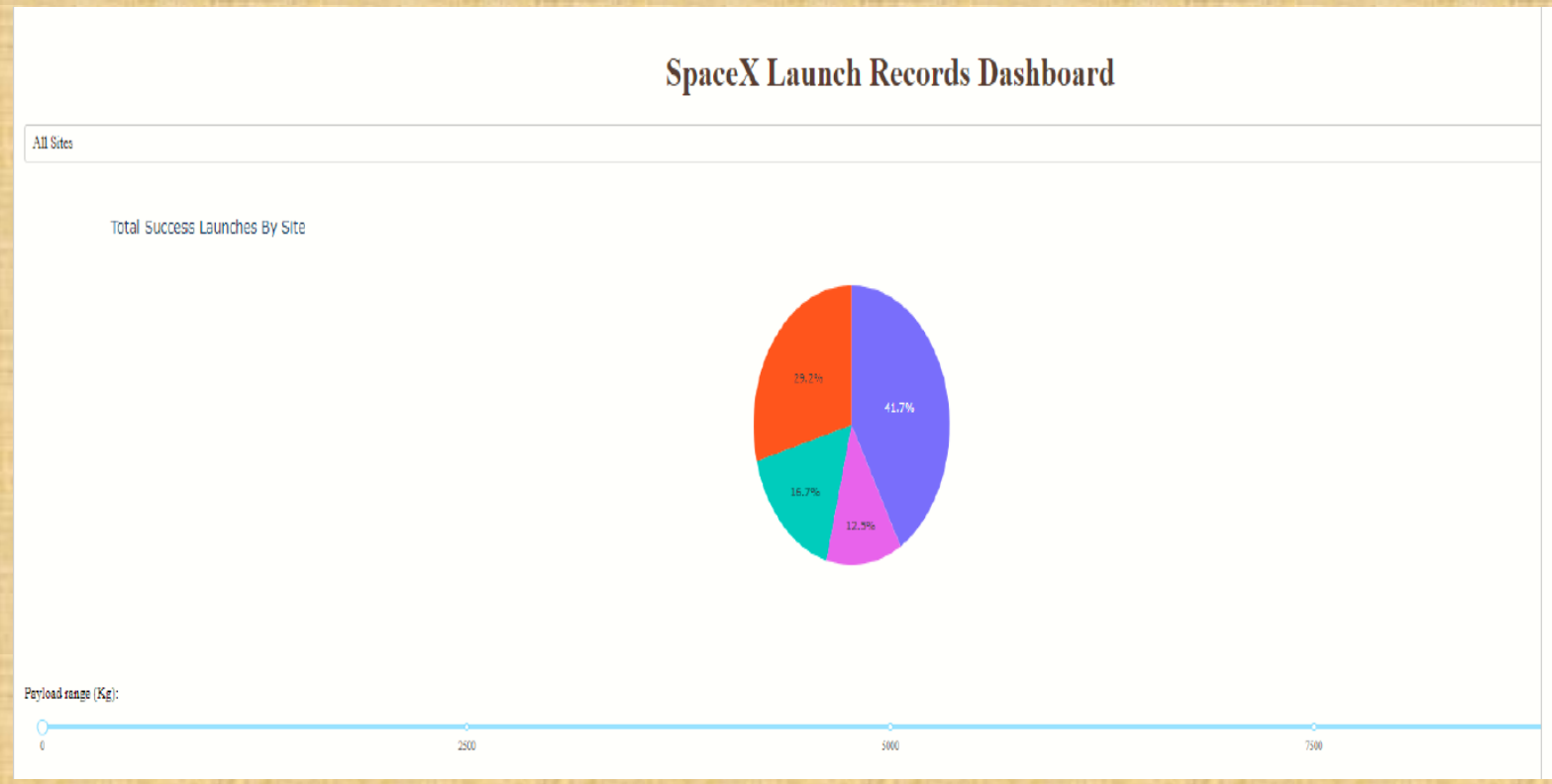
```
# Añadir marcadores en los puntos de interés
city_coordinate = [city_lat, city_lon]
railway_coordinate = [railway_lat, railway_lon]
highway_coordinate = [highway_lat, highway_lon]

city_marker = folium.Marker(
    city_coordinate,
    icon=DivIcon(
        icon_size=(20, 20),
        icon_anchor=(0, 0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:10.1f} Km".format(distance_city),
    )
)
site_map.add_child(city_marker)

railway_marker = folium.Marker(
    railway_coordinate,
    icon=DivIcon(
        icon_size=(20, 20),
        icon_anchor=(0, 0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:10.1f} Km".format(distance_railway),
    )
)
site_map.add_child(railway_marker)

highway_marker = folium.Marker(
    highway_coordinate,
    icon=DivIcon(
        icon_size=(20, 20),
        icon_anchor=(0, 0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:10.1f} Km".format(distance_highway),
    )
)
site_map.add_child(highway_marker)

# Crear líneas Polyline entre el sitio de lanzamiento y los puntos de interés
city_line = folium.PolyLine(locations=[[launch_site_lat, launch_site_lon], city_coordinate], weight=1)
railway_line = folium.PolyLine(locations=[[launch_site_lat, launch_site_lon], railway_coordinate], weight=1)
highway_line = folium.PolyLine(locations=[[launch_site_lat, launch_site_lon], highway_coordinate], weight=1)

site_map.add_child(city_line)
site_map.add_child(railway_line)
site_map.add_child(highway_line)

# Mostrar el mapa
site_map
```

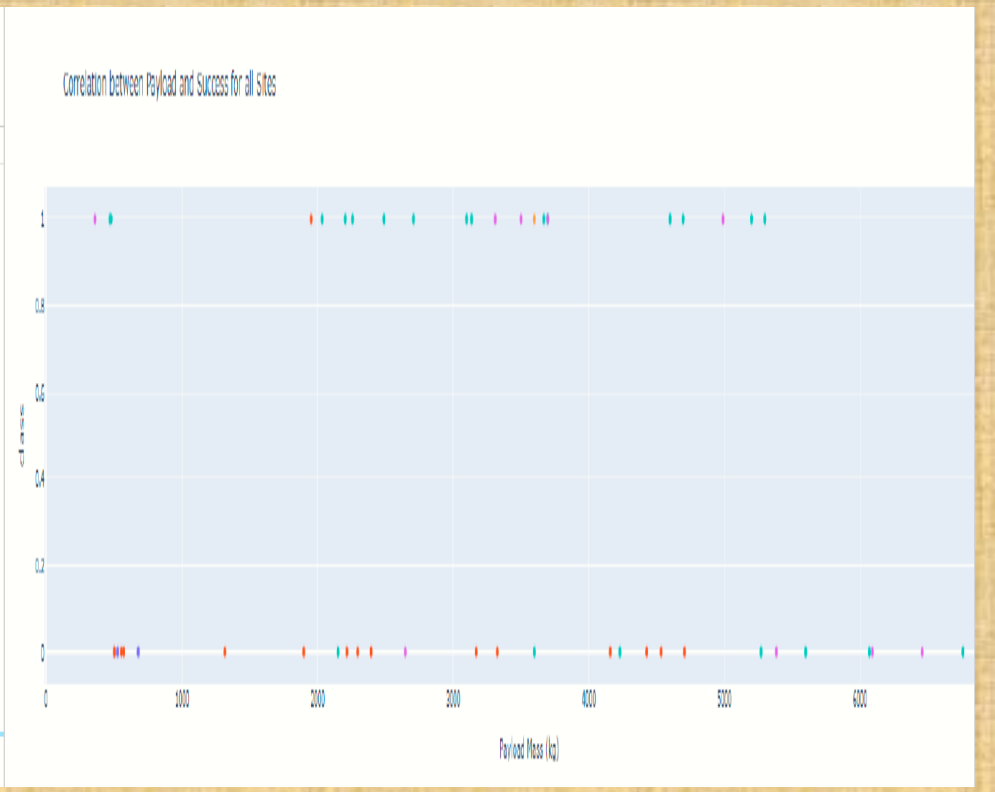


Build a Dashboard with Plotly Dash

- Summarize what plots/graphs and interactions you have added to a dashboard
  - Add a Launch Site Drop-down Input Component .
  - Add a callback function to render success-pie-chart based on selected site dropdown.
  - Add a Range Slider to Select Payload.
  - Add a callback function to render the success-payload-scatter-chart scatter plot
- Explain why you added those plots and interactions
- Add the GitHub URL of your completed Plotly Dash lab, as an external reference and peer-review purpose:
  - https://github.com/magno000/SpaceX/blob/main/spacex_dash_app.py

Build a Dashboard with Plotly Dash





Predictive Analysis (Classification)

- Summarize how you built, evaluated, improved, and found the best performing classification model
  - Based on the scores of the Test Set, we can not confirm which method performs best.
  - Same Test Set scores may be due to the small test sample size (18 samples). Therefore, we tested all methods based on the whole Dataset.
  - The scores of the whole Dataset confirm that the best model is the Decision Tree Model. This model has not only higher scores, but also the highest accuracy.
- You need present your model development process using key phrases and flowchart:
  - Load the dataframe
  - Create a NumPy array from the column Class in data, by applying the method to_numpy() then assign it to the variable Y,make sure the output is a Pandas series (only one bracket df['name of column']).
  - Standardize the data in X then reassign it to the variable X using the transform provided below.
  - Use the function train_test_split to split the data X and Y into training and test data. Set the parameter test_size to 0.2 and random_state to 2. The training data and test data should be assigned to the following labels.
  - Create a logistic regression object then create a GridSearchCV object logreg_cv with cv = 10. Fit the object to find the best parameters from the dictionary parameters. And calculate the accuracy on the test data using the method score y lets look at the confusion matrix.
  - Do the same for the models: Support Vector Machines (SVM), Decision Tree Classifier (DT), K-nearest neighbors (KNN)
- Add the GitHub URL of your completed predictive analysis lab, as an external reference and peer-review purpose:
  - https://github.com/magno000/SpaceX/blob/main/SpaceX_Machine%20Learning%20Prediction_Part_5_Sol.ipynb

Results

- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results

Load the data

```
from js import fetch
import io

URL1 = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/da
resp1 = await fetch(URL1)
text1 = io.BytesIO((await resp1.arrayBuffer()).to_py())
data = pd.read_csv(text1)
```

```
data.head(100)
```

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	
...	...	...	...	...	...	...	...	...	...	...	...	
85	86	2020-09-03	Falcon 9	15400.000000	VLEO	KSC LC 39A	True ASDS	2	True	True	True	5e9e303238
86	87	2020-10-06	Falcon 9	15400.000000	VLEO	KSC LC 39A	True ASDS	3	True	True	True	5e9e303238
87	88	2020-10-18	Falcon 9	15400.000000	VLEO	KSC LC 39A	True ASDS	6	True	True	True	5e9e303238
88	89	2020-10-24	Falcon 9	15400.000000	VLEO	CCAFS SLC 40	True ASDS	3	True	True	True	5e9e303338
89	90	2020-11-05	Falcon 9	3681.000000	MEO	CCAFS SLC 40	True ASDS	1	True	False	True	5e9e303238

90 rows × 18 columns

```
URL2 = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/da
resp2 = await fetch(URL2)
text2 = io.BytesIO((await resp2.arrayBuffer()).to_py())
X = pd.read_csv(text2)
```

```
X.head(100)
```

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	...	Serial_B105
0	1.0	6104.959412	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.
1	2.0	525.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.
2	3.0	677.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.
3	4.0	500.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.
4	5.0	3170.000000	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.
...	...	...	...	...	...	...	...	...	...	...	...	...
85	86.0	15400.000000	2.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0	...	0.
86	87.0	15400.000000	3.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0	...	1.
87	88.0	15400.000000	6.0	5.0	5.0	0.0	0.0	0.0	0.0	0.0	...	0.
88	89.0	15400.000000	3.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0	...	0.
89	90.0	3681.000000	1.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.

90 rows × 83 columns

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 90 entries, 0 to 89
Data columns (total 83 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   FlightNumber                            90 non-null     float64
 1   PayloadMass                             90 non-null     float64
 2   Flights                                 90 non-null     float64
 3   Block                                   90 non-null     float64
 4   ReusedCount                             90 non-null     float64
 5   Orbit_ES-L1                             90 non-null     float64
 6   Orbit_GEO                               90 non-null     float64
 7   Orbit_GTO                               90 non-null     float64
 8   Orbit_HEO                               90 non-null     float64
 9   Orbit_ISS                               90 non-null     float64
 10  Orbit_LEO                               90 non-null     float64
 11  Orbit_MEO                               90 non-null     float64
 12  Orbit_PO                                90 non-null     float64
 13  Orbit_SO                                90 non-null     float64
 14  Orbit_SSO                               90 non-null     float64
 15  Orbit_VLEO                              90 non-null     float64
 16  LaunchSite_CCAFS SLC 40                 90 non-null     float64
 17  LaunchSite_KSC LC 39A                   90 non-null     float64
 18  LaunchSite_VAFB SLC 4E                  90 non-null     float64
 19  LandingPad_5e9e3032383ecb267a34e7c7      90 non-null     float64
 20  LandingPad_5e9e3032383ecb554034e7c9      90 non-null     float64
 21  LandingPad_5e9e3032383ecb6bb234e7ca      90 non-null     float64
 22  LandingPad_5e9e3032383ecb761634e7cb      90 non-null     float64
 23  LandingPad_5e9e3033383ecbb9e534e7cc      90 non-null     float64
 24  Serial_B0003                            90 non-null     float64
 25  Serial_B0005                            90 non-null     float64
 26  Serial_B0007                            90 non-null     float64
 27  Serial_B1003                            90 non-null     float64
 28  Serial_B1004                            90 non-null     float64
 29  Serial_B1005                            90 non-null     float64
 30  Serial_B1006                            90 non-null     float64
 31  Serial_B1007                            90 non-null     float64
 32  Serial_B1008                            90 non-null     float64
 33  Serial_B1010                            90 non-null     float64
 34  Serial_B1011                            90 non-null     float64
 35  Serial_B1012                            90 non-null     float64
 36  Serial_B1013                            90 non-null     float64
 37  Serial_B1015                            90 non-null     float64
 38  Serial_B1016                            90 non-null     float64
 39  Serial_B1017                            90 non-null     float64
 40  Serial_B1018                            90 non-null     float64
 41  Serial_B1019                            90 non-null     float64
 42  Serial_B1020                            90 non-null     float64
 43  Serial_B1021                            90 non-null     float64
 44  Serial_B1022                            90 non-null     float64
 45  Serial_B1023                            90 non-null     float64
 46  Serial_B1025                            90 non-null     float64
```

```
# Crear una matriz NumPy a partir de la columna 'Class' en el DataFrame 'data'
Y = data['Class'].to_numpy()

# Verificar que Y es una serie de Pandas
print(type(Y))
print(Y[:5])
```

```
<class 'numpy.ndarray'>
[0 0 0 0 0]
```

Results

- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results

TASK 2

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

```
# students get this  / Esta forma es menos intuitiva
# transform = preprocessing.StandardScaler()

# Importar la biblioteca necesaria
from sklearn.preprocessing import StandardScaler

# Asumimos que 'X' es tu DataFrame original antes de la transformación
# Guardar los nombres de las columnas
column_names = X.columns

# Crear el transformador
transform = StandardScaler()

# Ajustar y transformar los datos en X
X = transform.fit_transform(X)

# Convertir de nuevo a DataFrame para mantener la consistencia
X = pd.DataFrame(X, columns=column_names)

# Verificar los datos transformados
X.head()
```

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	...	Serial
0	-1.712912	-5.295263e-17	-0.653913	-1.575895	-0.97344	-0.106	-0.106	-0.654654	-0.106	-0.551677	...	-0
1	-1.674419	-1.195232e+00	-0.653913	-1.575895	-0.97344	-0.106	-0.106	-0.654654	-0.106	-0.551677	...	-0
2	-1.635927	-1.162673e+00	-0.653913	-1.575895	-0.97344	-0.106	-0.106	-0.654654	-0.106	1.812654	...	-0
3	-1.597434	-1.200587e+00	-0.653913	-1.575895	-0.97344	-0.106	-0.106	-0.654654	-0.106	-0.551677	...	-0
4	-1.558942	-6.286706e-01	-0.653913	-1.575895	-0.97344	-0.106	-0.106	1.527525	-0.106	-0.551677	...	-0

5 rows × 83 columns

`X_train, X_test, Y_train, Y_test`

```
# Importar la función train_test_split de sklearn
from sklearn.model_selection import train_test_split

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)

# Verificar las dimensiones de los conjuntos resultantes
print("Dimensiones de X_train:", X_train.shape)
print("Dimensiones de X_test:", X_test.shape)
print("Dimensiones de Y_train:", Y_train.shape)
print("Dimensiones de Y_test:", Y_test.shape)
```

```
Dimensiones de X_train: (72, 83)
Dimensiones de X_test: (18, 83)
Dimensiones de Y_train: (72,)
Dimensiones de Y_test: (18,)
```

we can see we only have 18 test samples.

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

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

tree = DecisionTreeClassifier()
```

```
tree_cv = GridSearchCV(tree, parameters, cv=10)
tree_cv.fit(X_train, Y_train)
```

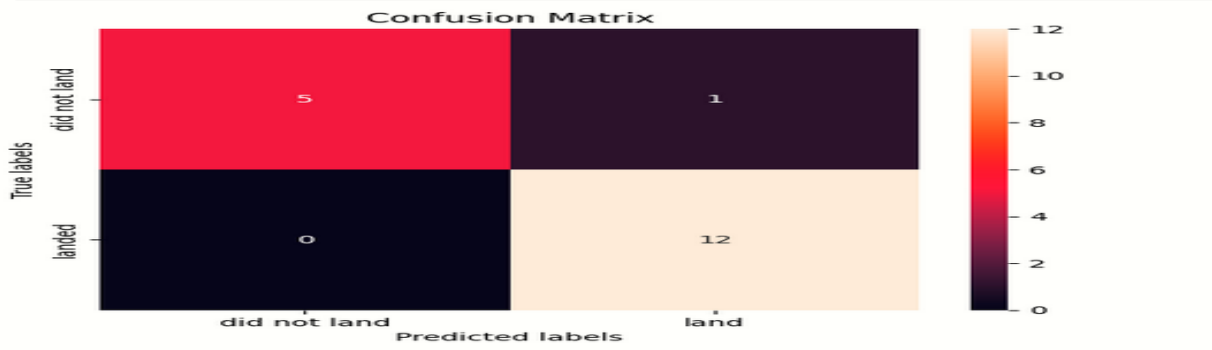
Calculate the accuracy of tree_cv on the test data using the method `score`:

```
# Calcular la precisión en los datos de prueba
accuracy = tree_cv.score(X_test, Y_test)
print("Accuracy of Decision Tree classifier on test set:", accuracy)
```

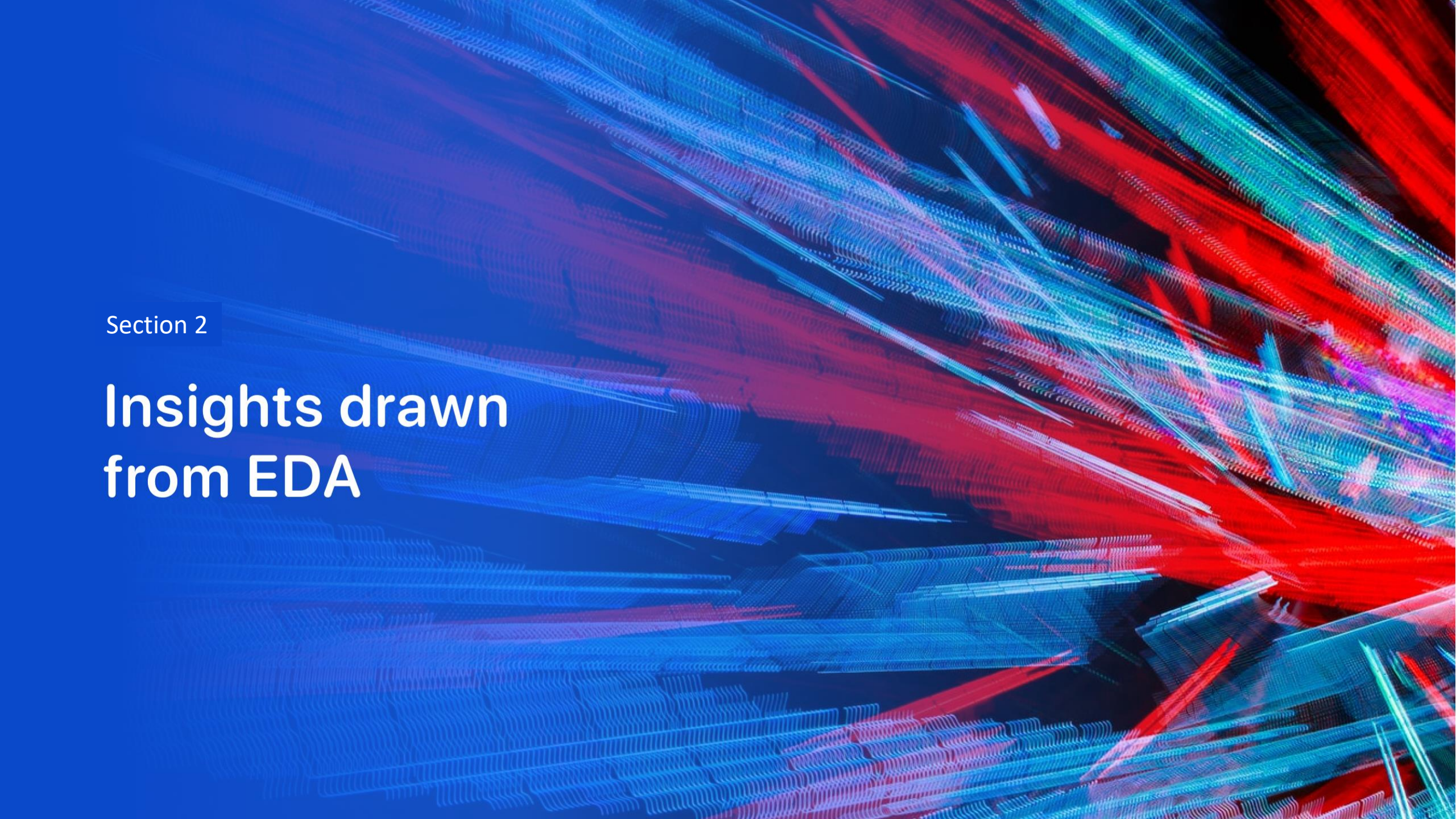
```
Accuracy of Decision Tree classifier on test set: 0.9444444444444444
```

We can plot the confusion matrix

```
yhat = tree_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```

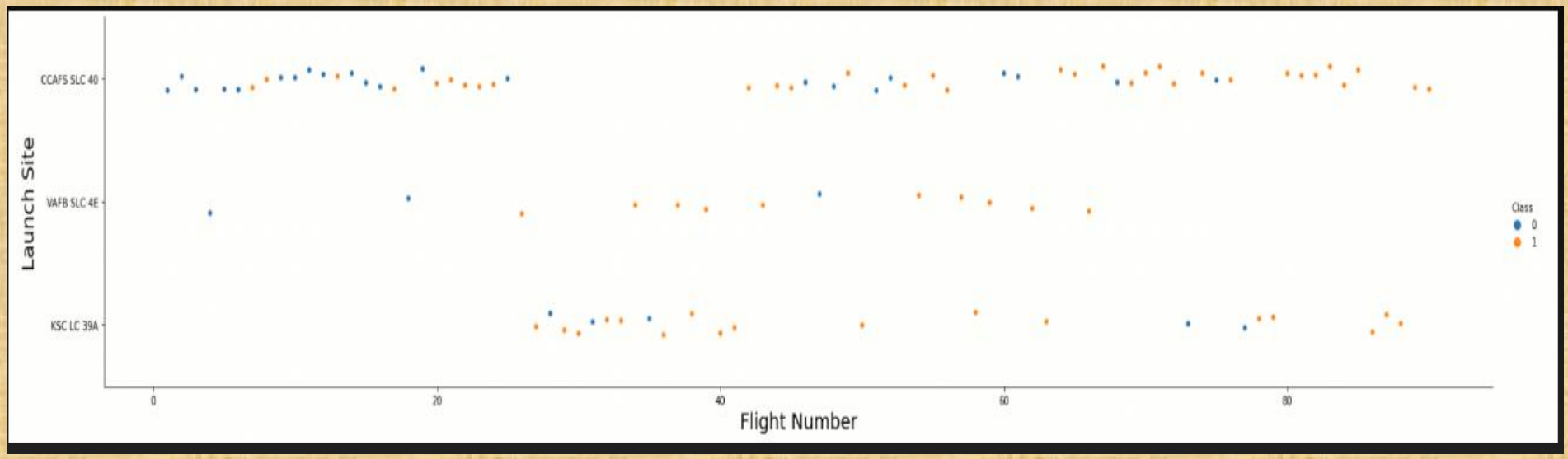


Decision Tree Model is the best algorithm for this dataset.


Section 2
Insights drawn from EDA

Flight Number vs. Launch Site

- Show a scatter plot of Flight Number vs. Launch Site
- Show the screenshot of the scatter plot with explanations

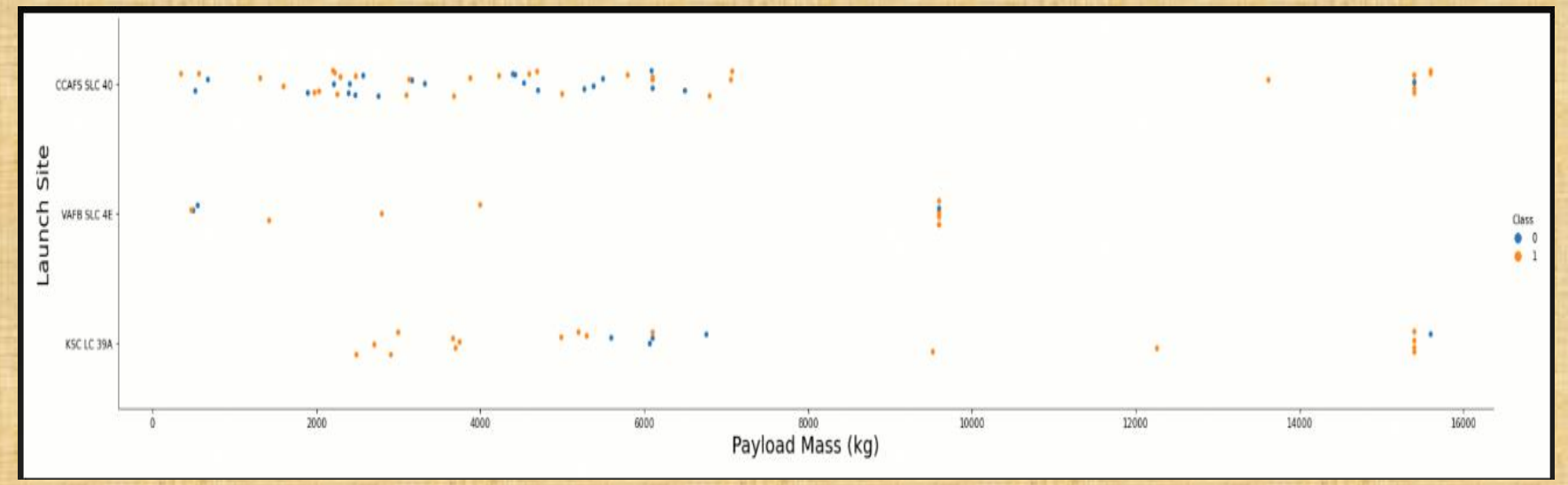


Explanation:

- The earliest flights all failed while the latest flights all succeeded.
- The CCAFS SLC 40 launch site has about a half of all launches.
- VAFB SLC 4E and KSC LC 39A have higher success rates.
- It can be assumed that each new launch has a higher rate of success.

Payload vs. Launch Site

- Show a scatter plot of Payload vs. Launch Site
- Show the screenshot of the scatter plot with explanations



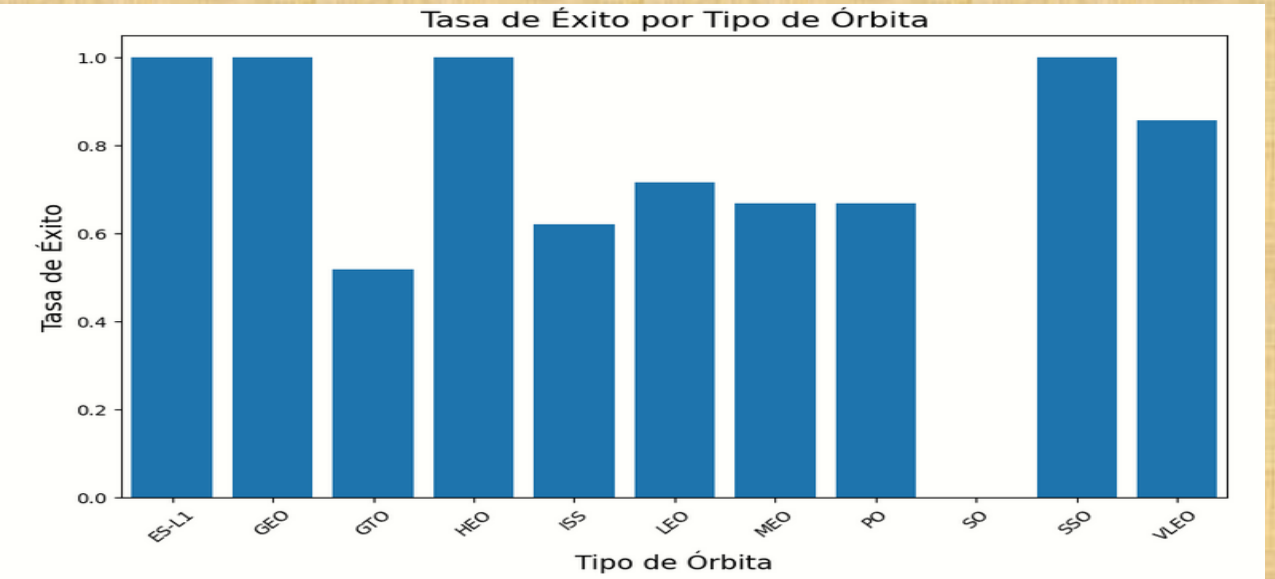
Explanation:

- For every launch site the higher the payload mass, the higher the success rate.
- Most of the launches with payload mass over 7000 kg were successful.
- KSC LC 39A has a 100% success rate for payload mass under 5500 kg too.

Success Rate vs. Orbit Type

- Show a bar chart for the success rate of each orbit type

- Show the screenshot of the scatter plot with explanations



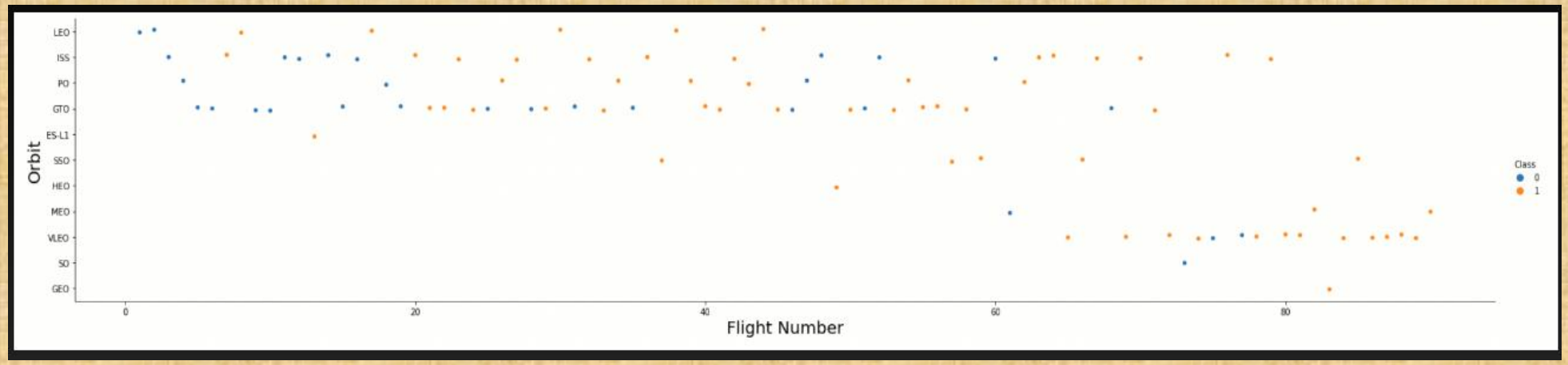
Explanation:

• Orbits with 100% success rate:
- ES-L1, GEO, HEO, SSO
• Orbits with 0% success rate:
- SO
• Orbits with success rate between 50% and 85%:
- GTO, ISS, LEO, MEO, PO

Flight Number vs. Orbit Type

- Show a scatter point of Flight number vs. Orbit type

- Show the screenshot of the scatter plot with explanations

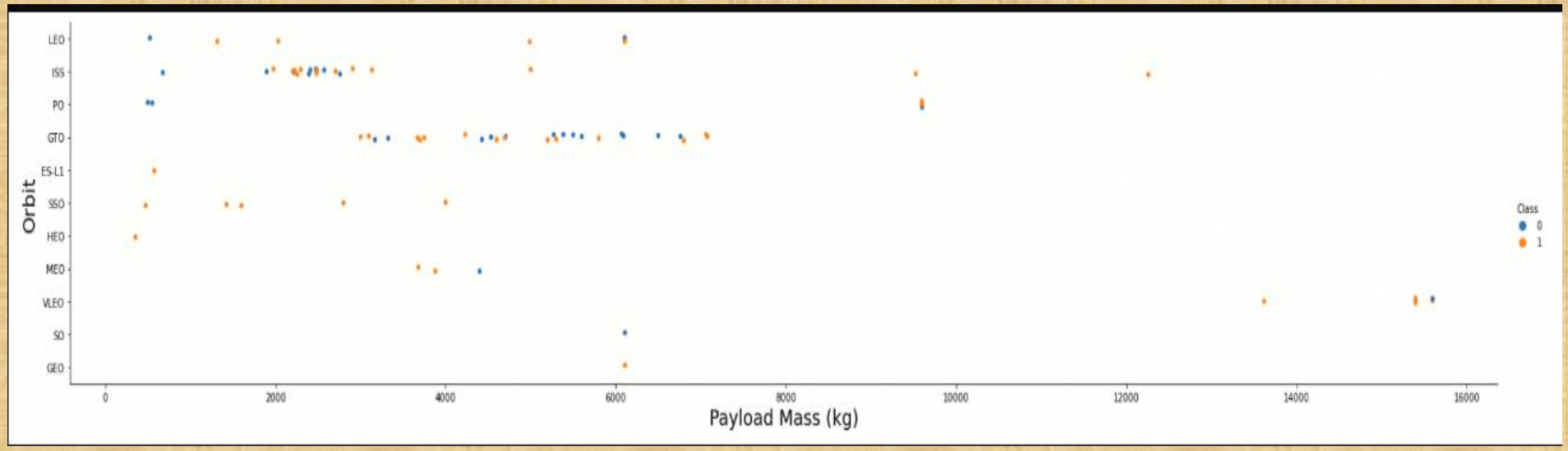


Explanation:

• In the LEO orbit the Success appears related to the number of flights; on the other hand, there seems to be no relationship between flight number when in GTO orbit.

Payload vs. Orbit Type

- Show a scatter point of payload vs. orbit type
- Show the screenshot of the scatter plot with explanations



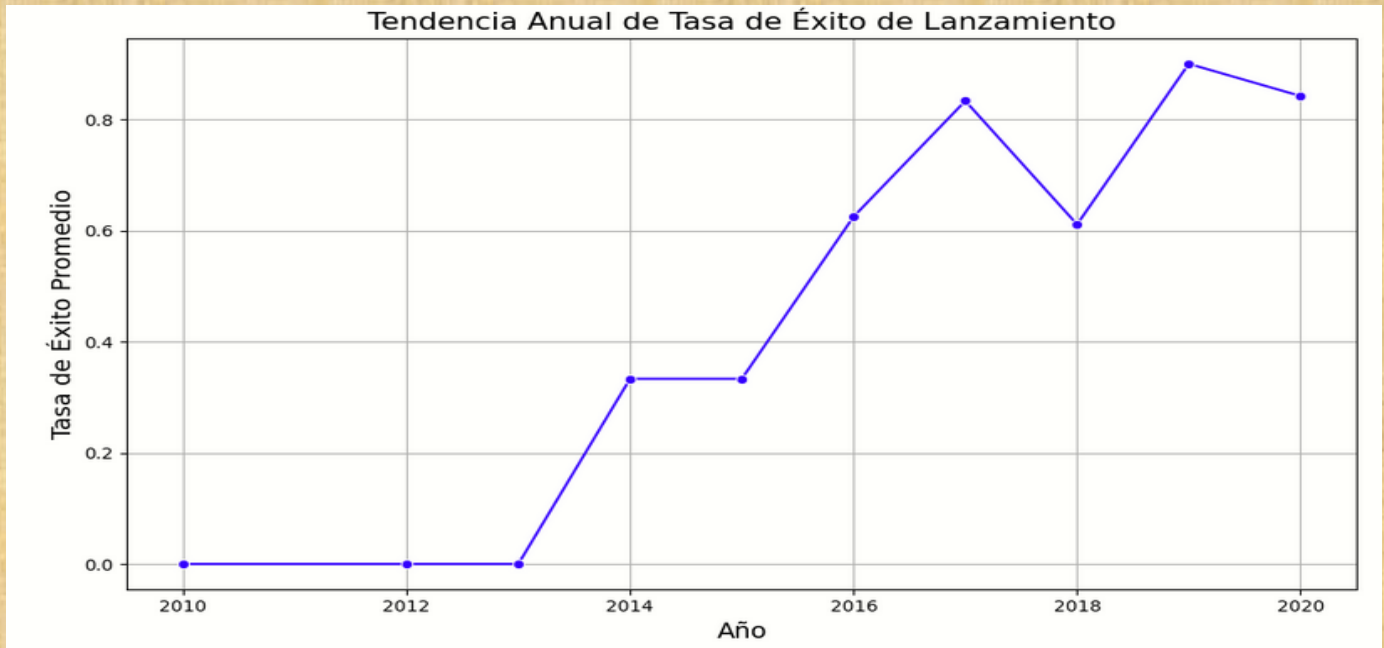
Explanation:

- Heavy payloads have a negative influence on GTO orbits and positive on GTO and Polar LEO (ISS) orbits.

Launch Success Yearly Trend

- Show a line chart of yearly average success rate

- Show the screenshot of the scatter plot with explanations



Explanation:

- The success rate since 2013 kept increasing till 2020.

All Launch Site Names

- Find the names of the unique launch sites
- Present your query result with a short explanation here

Task 1

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

```
query = 'SELECT DISTINCT "Launch_Site" FROM SPACEXTABLE'
```

```
cur.execute(query)
launch_sites = cur.fetchall()
```

```
for site in launch_sites:
    print(site[0])
```

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

Explanation:

• Displaying the names of the unique launch sites in the space mission.

Launch Site Names Begin with 'CCA'

- Find 5 records where launch sites begin with `CCA`
- Present your query result with a short explanation here

Task 2

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

```
query = '''
SELECT * FROM SPACEXTABLE
WHERE "Launch_Site" LIKE 'CCA%'
LIMIT 5
'''
```

```
cur.execute(query)
records = cur.fetchall()
```

```
for record in records:
    print(record[0])
```

```
2010-06-04
2010-12-08
2012-05-22
2012-10-08
2013-03-01
```

Explanation:

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

Total Payload Mass

- Calculate the total payload carried by boosters from NASA
- Present your query result with a short explanation here

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

```
# Consulta SQL para obtener la masa total de carga útil transportada por los cohetes lanzados por NASA (CRS)
query = '''
SELECT SUM("PAYLOAD_MASS__KG_") as total_payload_mass
FROM SPACEXTABLE
WHERE "Customer" = 'NASA (CRS)'
'''
```

```
cur.execute(query)
total_payload_mass = cur.fetchone()[0]
print(total_payload_mass)
```

```
45596
```

```
print("Total payload mass carried by boosters launched by NASA (CRS):", total_payload_mass, "kg")
```

```
Total payload mass carried by boosters launched by NASA (CRS): 45596 kg
```

Explanation:

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

Average Payload Mass by F9 v1.1

- Calculate the average payload mass carried by booster version F9 v1.1
- Present your query result with a short explanation here

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

```
query = '''
SELECT AVG("PAYLOAD_MASS__KG_") as average_payload_mass
FROM SPACEXTABLE
WHERE "Booster_Version" = 'F9 v1.1'
'''

cur.execute(query)
average_payload_mass = cur.fetchone()[0]

print("Average payload mass carried by booster version F9 v1.1:", average_payload_mass, "kg")
```

```
Average payload mass carried by booster version F9 v1.1: 2928.4 kg
```

Explanation:

- Displaying average payload mass carried by booster version F9 v1.1.

First Successful Ground Landing Date

- Find the dates of the first successful landing outcome on ground pad
- Present your query result with a short explanation here

```
query = '''
SELECT MIN(Date) as first_successful_landing_date
FROM SPACEXTABLE
WHERE "Landing_Outcome" = 'Success (ground pad)'
'''

cur.execute(query)
first_successful_landing_date = cur.fetchone()[0]

print("Date of the first successful landing on a ground pad:", first_successful_landing_date)
```

```
Date of the first successful landing on a ground pad: 2015-12-22
```

Explanation:

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

Successful Drone Ship Landing with Payload between 4000 and 6000

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

- Present your query result with a short explanation here

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

```
# Consulta SQL para obtener los nombres de los boosters con éxito en nave dron y masa de carga útil entre 4000 y
query = '''
SELECT "Booster_Version"
FROM SPACEXTABLE
WHERE "Landing_Outcome" = 'Success (drone ship)'
AND "PAYLOAD_MASS__KG_" > 4000
AND "PAYLOAD_MASS__KG_" < 6000
'''

# Ejecutar la consulta y obtener los resultados
cur.execute(query)
booster_names = cur.fetchall()

# Mostrar los resultados
for booster in booster_names:
    print("Booster Version:", booster[0])
```

```
Booster Version: F9 FT B1022
Booster Version: F9 FT B1026
Booster Version: F9 FT  B1021.2
Booster Version: F9 FT  B1031.2
```

Explanation:

- **Listing the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000.**

Total Number of Successful and Failure Mission Outcomes

- Calculate the total number of successful and failure mission outcomes
- Present your query result with a short explanation here

List the total number of successful and failure mission outcomes

```
# Consulta SQL para contar el número total de resultados de misiones exitosas y fallidas
query = '''
SELECT "Landing_Outcome", COUNT(*) as total
FROM SPACEXTABLE
GROUP BY "Landing_Outcome"
'''

# Ejecutar la consulta y obtener los resultados
cur.execute(query)
outcome_counts = cur.fetchall()

# Mostrar los resultados
for outcome, count in outcome_counts:
    print("Landing Outcome:", outcome, "| Total:", count)
```

```
Landing Outcome: Controlled (ocean) | Total: 5
Landing Outcome: Failure | Total: 3
Landing Outcome: Failure (drone ship) | Total: 5
Landing Outcome: Failure (parachute) | Total: 2
Landing Outcome: No attempt | Total: 21
Landing Outcome: No attempt  | Total: 1
Landing Outcome: Precluded (drone ship) | Total: 1
Landing Outcome: Success | Total: 38
Landing Outcome: Success (drone ship) | Total: 14
Landing Outcome: Success (ground pad) | Total: 9
Landing Outcome: Uncontrolled (ocean) | Total: 2
```

Explanation:

- Listing the total number of successful and failure mission outcomes.

Boosters Carried Maximum Payload

- List the names of the booster which have carried the maximum payload mass
- Present your query result with a short explanation here

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

```
# Consulta SQL para listar las versiones de los propulsores que han transportado la masa de carga útil máxima
query = '''
SELECT "Booster_Version"
FROM SPACEXTABLE
WHERE "PAYLOAD_MASS__KG_" = (
    SELECT MAX("PAYLOAD_MASS__KG_")
    FROM SPACEXTABLE
)
'''

# Ejecutar la consulta y obtener los resultados
cur.execute(query)
booster_versions = cur.fetchall()

# Mostrar los resultados
print("Booster versions with the maximum payload mass:")
for booster in booster_versions:
    print(booster[0])
```

```
Booster versions with the maximum payload mass:
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
```

Explanation:

• Listing the names of the booster versions which have carried the maximum payload mass.

2015 Launch Records

- List the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015
- Present your query result with a short explanation here

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

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

```
# Consulta SQL para listar los registros que cumplen con los criterios especificados
query = '''
SELECT
    CASE
        WHEN substr(Date, 6, 2) = '01' THEN 'January'
        WHEN substr(Date, 6, 2) = '02' THEN 'February'
        WHEN substr(Date, 6, 2) = '03' THEN 'March'
        WHEN substr(Date, 6, 2) = '04' THEN 'April'
        WHEN substr(Date, 6, 2) = '05' THEN 'May'
        WHEN substr(Date, 6, 2) = '06' THEN 'June'
        WHEN substr(Date, 6, 2) = '07' THEN 'July'
        WHEN substr(Date, 6, 2) = '08' THEN 'August'
        WHEN substr(Date, 6, 2) = '09' THEN 'September'
        WHEN substr(Date, 6, 2) = '10' THEN 'October'
        WHEN substr(Date, 6, 2) = '11' THEN 'November'
        WHEN substr(Date, 6, 2) = '12' THEN 'December'
    END AS Month,
    Landing_Outcome,
    Booster_Version,
    Launch_Site
FROM SPACEXTABLE
WHERE substr(Date, 0, 5) = '2015'
    AND Landing_Outcome = 'Failure (drone ship)'
'''

# Ejecutar la consulta y obtener los resultados
cur.execute(query)
records = cur.fetchall()

# Mostrar los resultados
print("Records for failure landing outcomes on drone ship in year 2015:")
for record in records:
    print(record)
```

```
Records for failure landing outcomes on drone ship in year 2015:
('January', 'Failure (drone ship)', 'F9 v1.1 B1012', 'CCAFS LC-40')
('April', 'Failure (drone ship)', 'F9 v1.1 B1015', 'CCAFS LC-40')
```

Explanation:

- Listing the failed landing outcomes in drone ship, their booster versions and launch site names for the months in year 2015.

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

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

- Present your query result with a short explanation here

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.

```
# Consulta SQL para clasificar el recuento de resultados de aterrizaje entre las fechas especificadas
query = '''
SELECT
    Landing_Outcome,
    COUNT(*) as Count_Landing_Outcome
FROM SPACEXTABLE
WHERE Date BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY Landing_Outcome
ORDER BY Count_Landing_Outcome DESC
'''

# Ejecutar la consulta y obtener los resultados
cur.execute(query)
records = cur.fetchall()

# Mostrar los resultados
print("Ranking of landing outcomes between 2010-06-04 and 2017-03-20:")
for record in records:
    print(record)

# Cerrar la conexión
con.close()
```

```
Ranking of landing outcomes between 2010-06-04 and 2017-03-20:
('No attempt', 10)
('Success (drone ship)', 5)
('Failure (drone ship)', 5)
('Success (ground pad)', 3)
('Controlled (ocean)', 3)
('Uncontrolled (ocean)', 2)
('Failure (parachute)', 2)
('Precluded (drone ship)', 1)
```

Reference Links

Explanation:

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

Section 3

Launch Sites Proximities Analysis

All launch sites' location markers on a global map

Explanation:

- Most of Launch sites are in proximity to the Equator line. The land is moving faster at the equator than any other place on the surface of the Earth. This is because of inertia.

This speed will help the spacecraft keep up a good enough speed to stay in orbit.

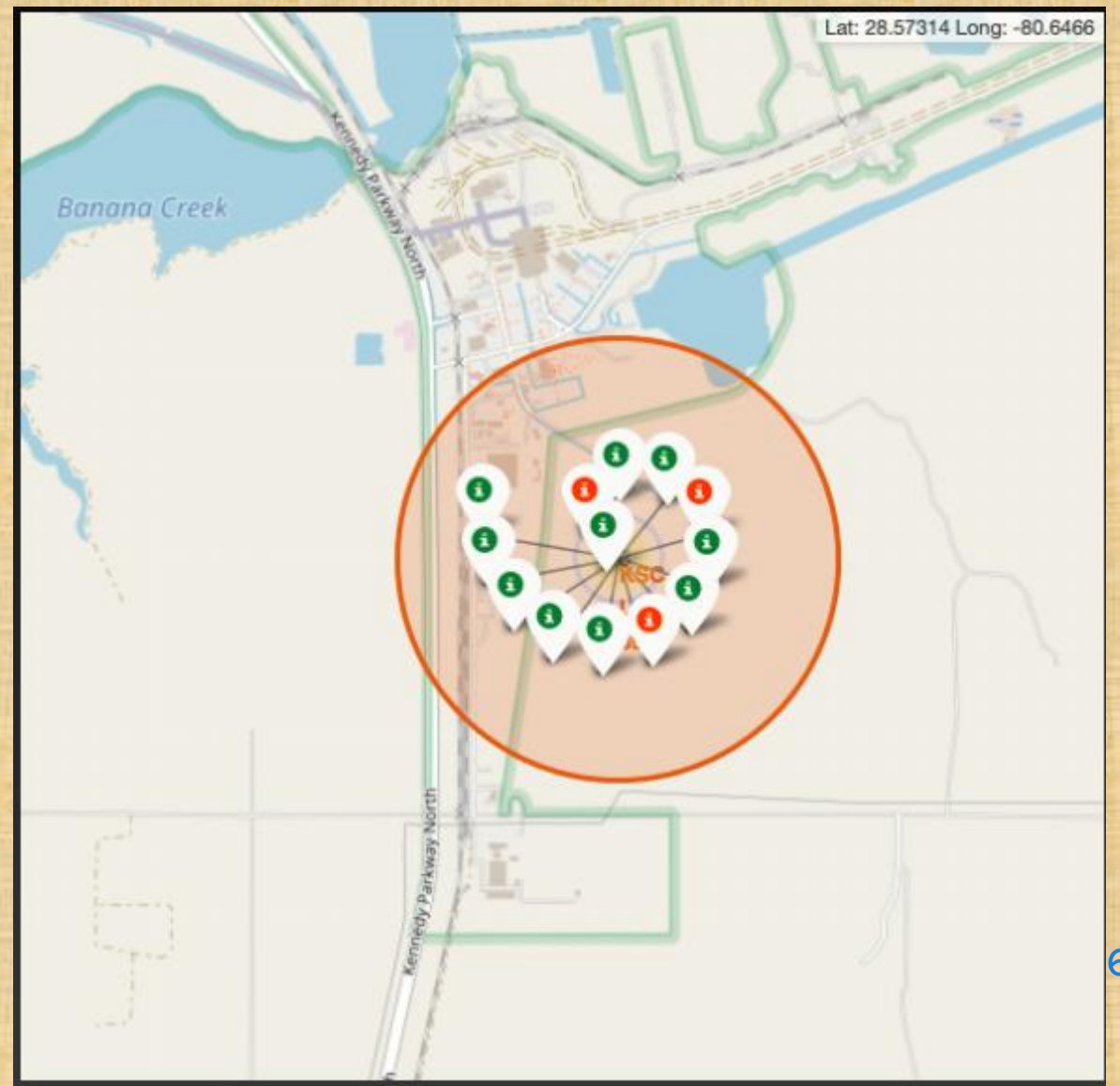
- All launch sites are in very close proximity to the coast, it minimises the risk of having any debris dropping or exploding near people.

Colour-labeled launch records on the map

Explanation:

- From the colour-labeled markers we should be able to easily identify which launch sites have relatively high success rates.
- **Green Marker** = Successful Launch
- **Red Marker** = Failed Launch

- Launch Site KSC LC-39A has a very high Success Rate.



Distance from the launch site KSC LC-39A to its proximities

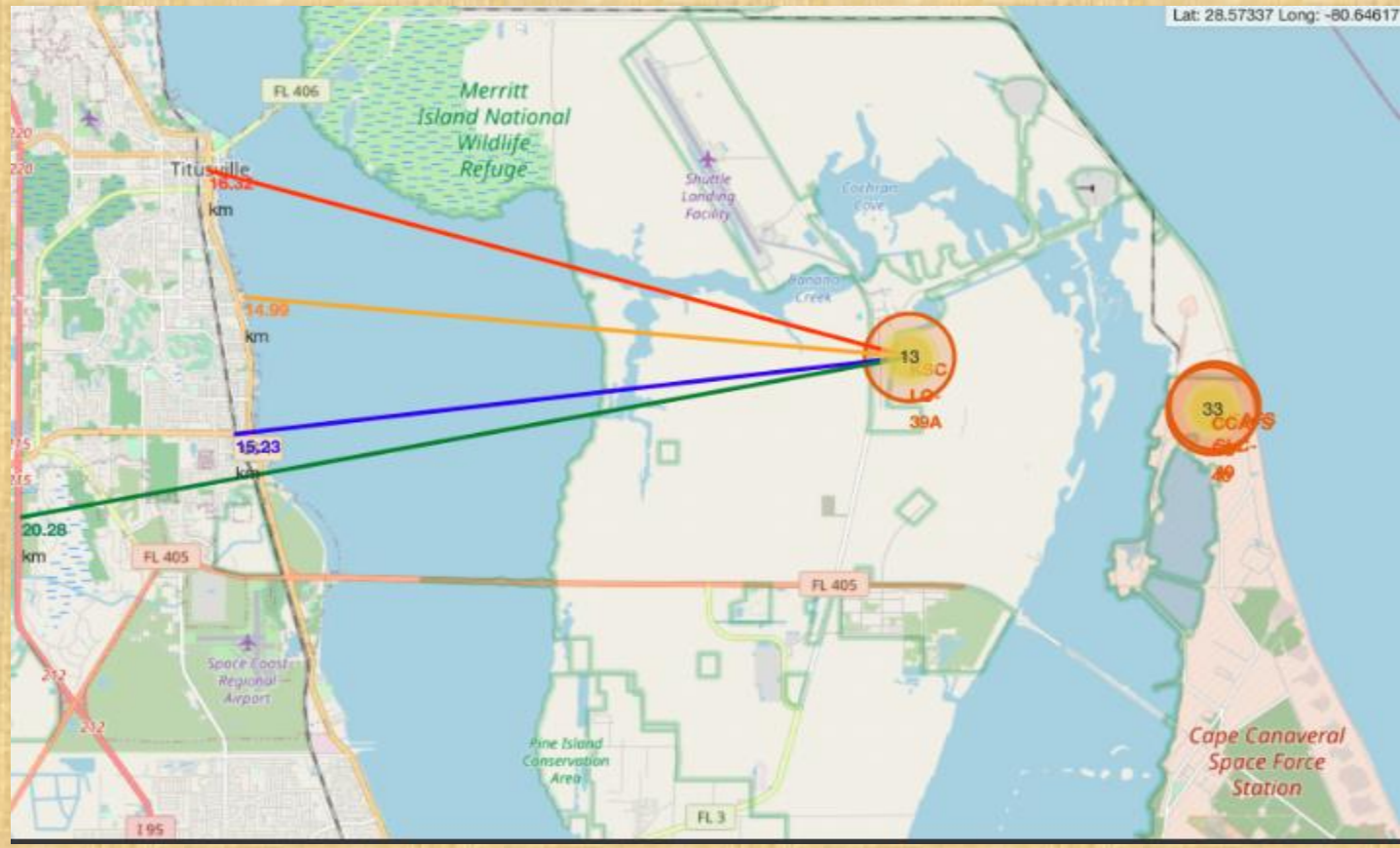
Explanation:

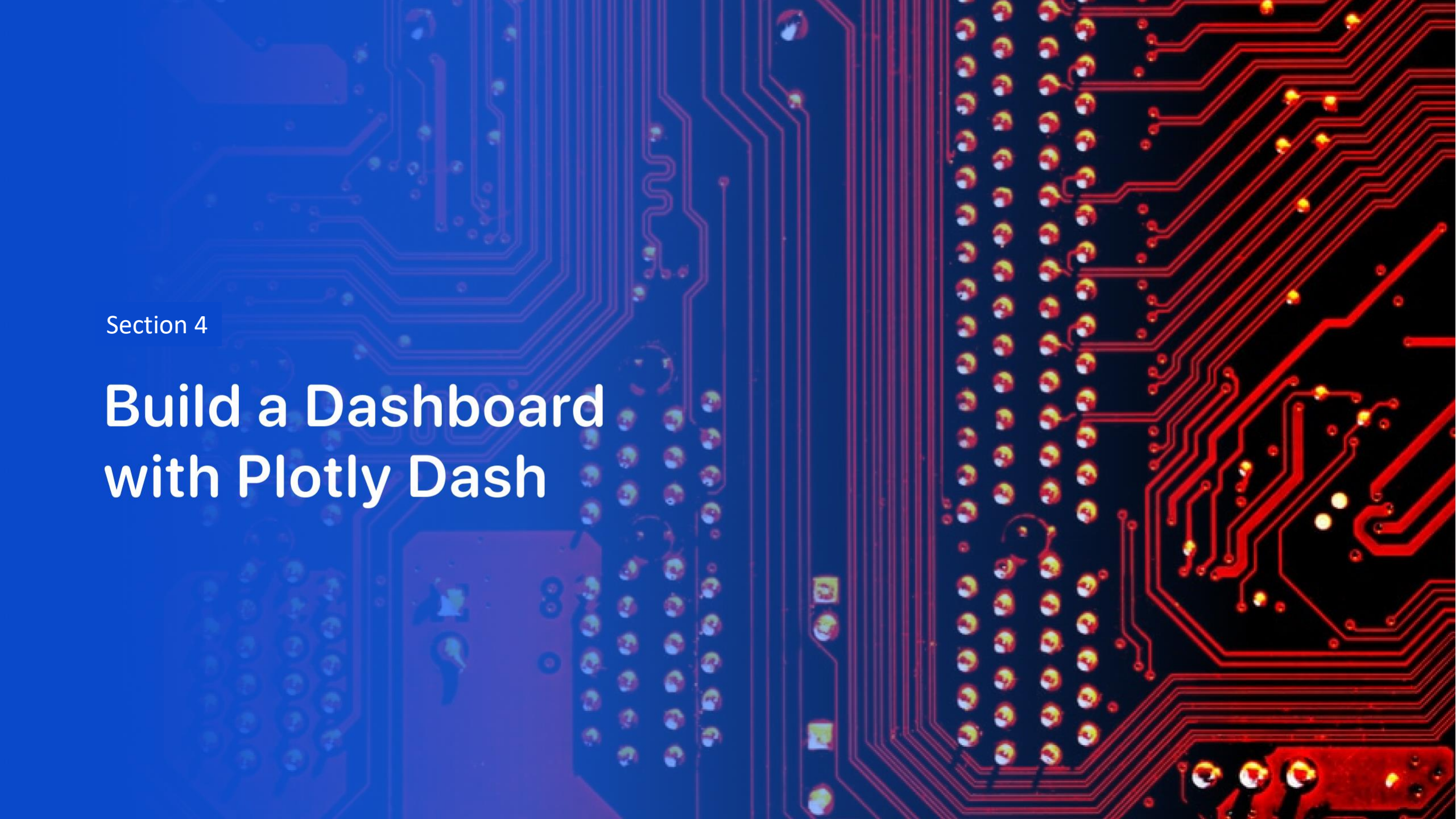
• From the visual analysis of the launch site KSC LC-39A we can clearly see that it is:

- relative close to railway (15.23 km)
- relative close to highway (20.28 km)
- relative close to coastline (14.99 km)

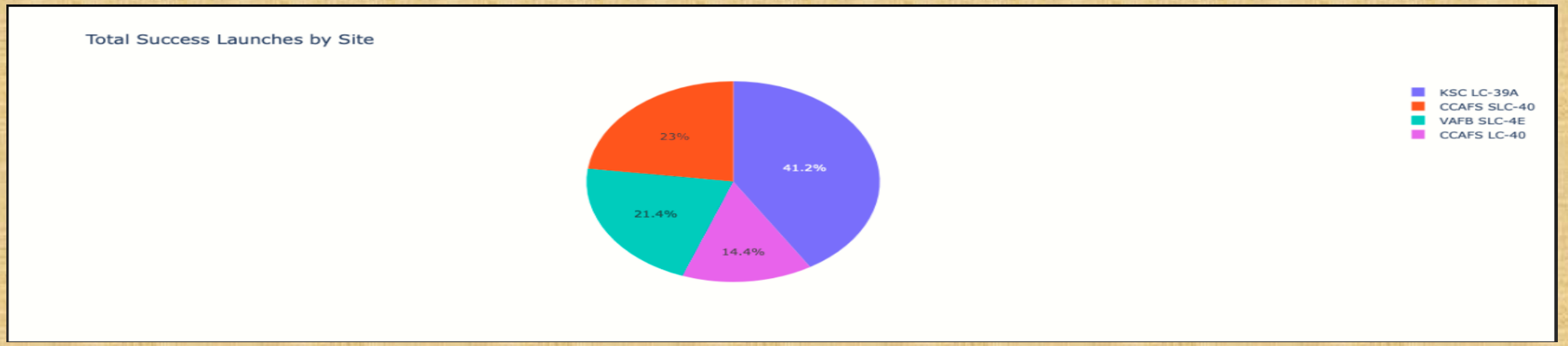
• Also the launch site KSC LC-39A is relative close to its closest city Titusville (16.32 km).

• Failed rocket with its high speed can cover distances like 15-20 km in few seconds. It could be potentially dangerous to populated areas.




Section 4
Build a Dashboard with Plotly Dash

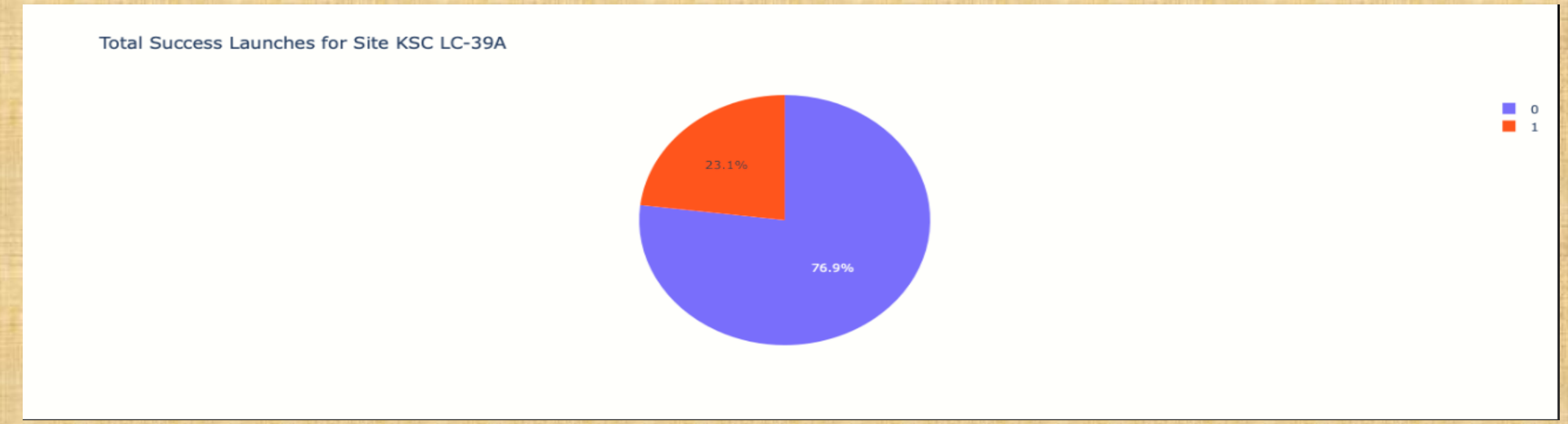
Launch success count for all sites



Explanation:

- The chart clearly shows that from all the sites, KSC LC-39A has the most successful launches.

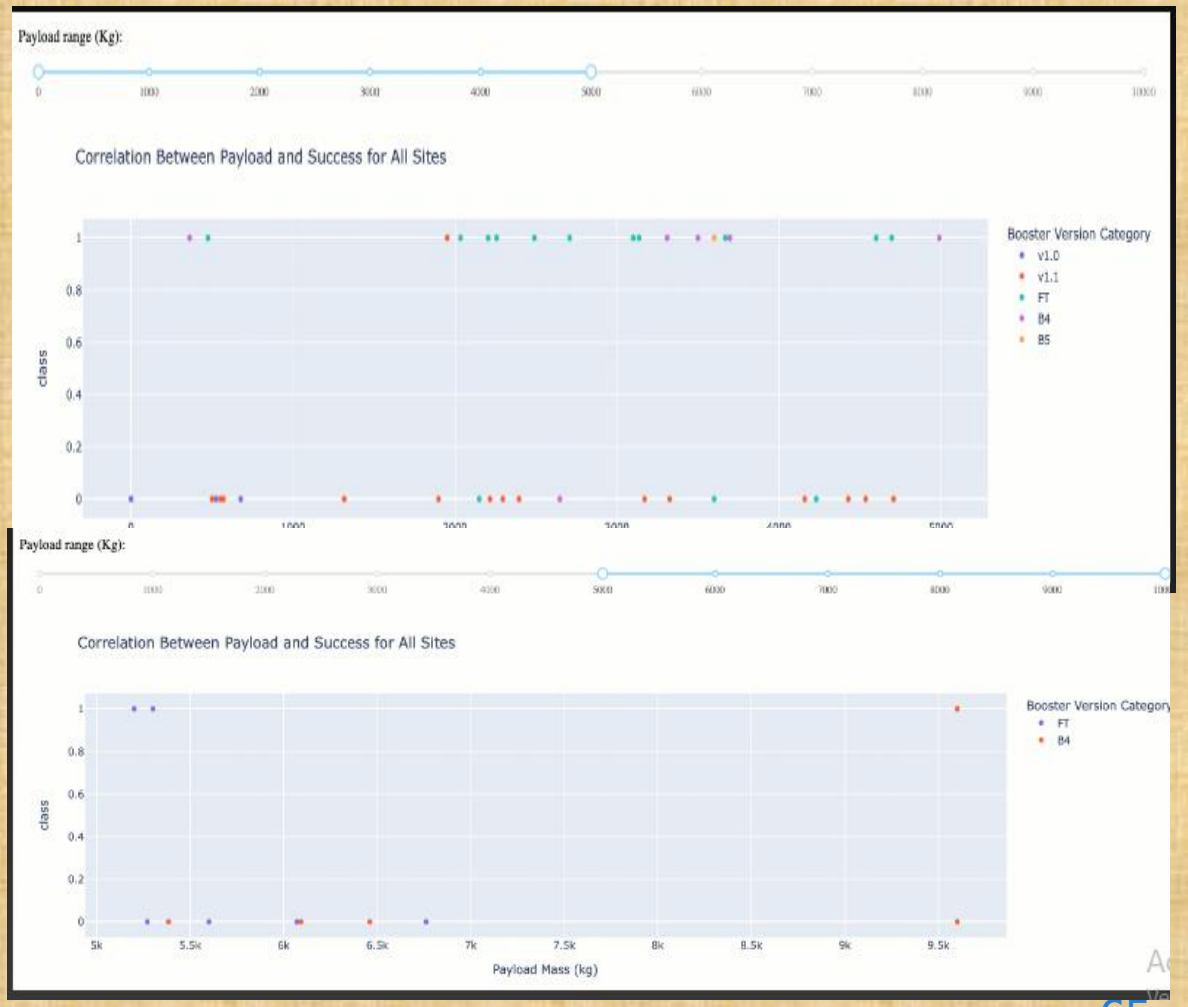
Launch site with highest launch success ratio



Explanation:

- KSC LC-39A has the highest launch success rate (76.9%) with 10 successful and only 3 failed landings.

Payload Mass vs. Launch Outcome for all sites



Explanation:

- The charts show that payloads between 2000 and 5500 kg have the highest success rate.

Section 5

Predictive Analysis (Classification)

Classification Accuracy

- Visualize the built model accuracy for all built classification models, in a bar chart

- Find which model has the highest classification accuracy

```
import pandas as pd
from sklearn.metrics import classification_report

# Obtener los classification_report para cada modelo
logreg_report = classification_report(Y_test, logreg_cv.predict(X_test), output_dict=True)
svm_report = classification_report(Y_test, svm_cv.predict(X_test), output_dict=True)
tree_report = classification_report(Y_test, tree_cv.predict(X_test), output_dict=True)
knn_report = classification_report(Y_test, knn_cv.predict(X_test), output_dict=True)

# Crear un DataFrame para organizar los datos
df = pd.DataFrame({
    'Modelo': ['Logistic Regression', 'Support Vector Machine', 'Decision Tree', 'K Nearest Neighbors'],
    'Accuracy': [logreg_report['accuracy'], svm_report['accuracy'], tree_report['accuracy'], knn_report['accuracy']],
    'Precision': [logreg_report['1']['precision'], svm_report['1']['precision'], tree_report['1']['precision'], knn_report['1']['precision']],
    'Recall': [logreg_report['1']['recall'], svm_report['1']['recall'], tree_report['1']['recall'], knn_report['1']['recall']],
    'F1-score': [logreg_report['1']['f1-score'], svm_report['1']['f1-score'], tree_report['1']['f1-score'], knn_report['1']['f1-score']]
})

# Mostrar el DataFrame
print(df)
```

```
                    Modelo  Accuracy  Precision  Recall  F1-score
0      Logistic Regression  0.833333   0.800000     1.0  0.888889
1  Support Vector Machine   0.833333   0.800000     1.0  0.888889
2            Decision Tree  0.944444   0.923077     1.0  0.960000
3      K Nearest Neighbors  0.833333   0.800000     1.0  0.888889
```

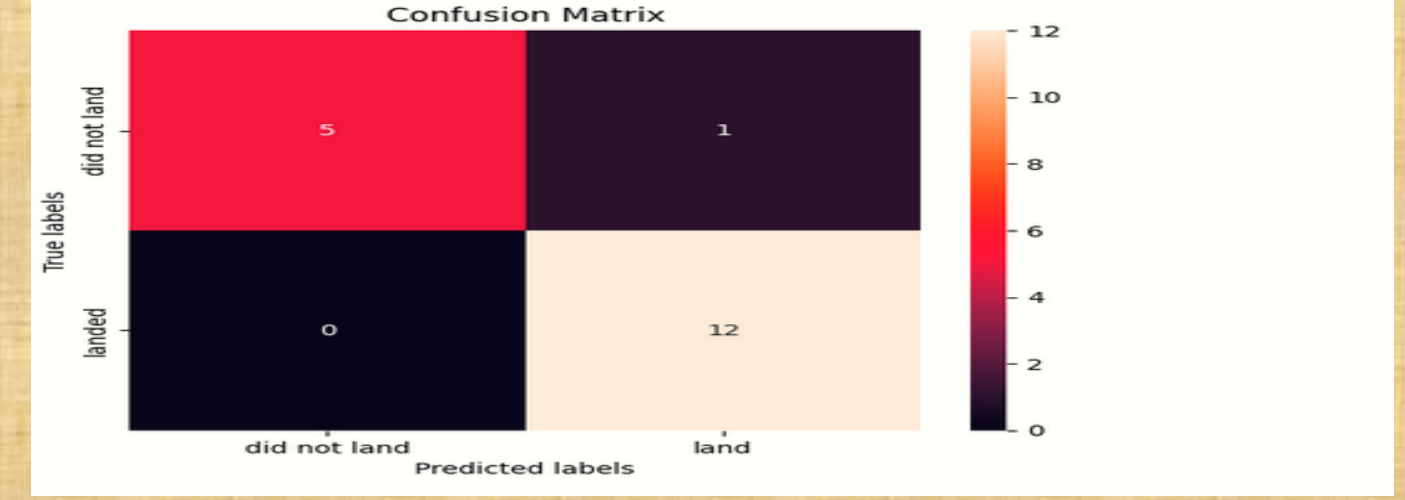
Explanation:

• The performance metrics of the Test Dataset confirm that the best model is the Decision Tree Model. This model has not only higher scores, but also the highest accuracy, precision and F-1 Score.

Confusion Matrix

- Show the confusion matrix of the best performing model with an explanation

```
yhat = tree_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



Explanation:

- Examining the confusion matrix, we see that decision tree model can distinguish between the different classes. We see that the major problem is false positives.

Conclusions

- Decision Tree Model is the best algorithm for this dataset.
- Launches with a low payload mass show better results than launches with a larger payload mass.
- Most of launch sites are in proximity to the Equator line and all the sites are in very close proximity to the coast.
- The success rate of launches increases over the years.
- KSC LC-39A has the highest success rate of the launches
- from all the sites.
- Orbits ES-L1, GEO, HEO and SSO have 100% success rate.

Appendix

Special Thanks to:

- Instructors
- Coursera
- IBM

Thank you!