APPLIED DATA SCIENCE CAPSTONE BY IBM (COURSERA)

LOUKAS ANASTASIADIS

EXECUTIVE SUMMARY

- Objective : The objective is to find out how and why a falcon 9 launch is successful , thus predicting the cost of a falcon 9 rocket launch.
- Approach: I used the get request method and also webscraping to collect data . Then i cleaned them and used data wrangling. I seperated falcon 1 from falcon 9 launches and also selected the attributes that mattered the most. After that i used data viz techniques with folium and plotly to vizualize my findings. In addition i used a lot of machine learning models via gridsearch to find the best parameteres for good predictions.
- Findings : There are a lot of factors that contribute into a succeful rocket launch . In the following slides a lot of aspects are examined concerning what makes a rocket launch successful or not.

TABLE OF CONTENTS

INTRODUCTION

- This capstone project is all about falcon 9 launches from SpaceX. Rocket companies are trying to make space flight comercial . Falcon 9 rockets are able to reuse the first stage of the launch , thus making rocket launches less expensive . In this capstone i am going to take data from urls and also webscraping . I am going to clean and preprocess it (no null or 0 values) and gain valuable insights using sql and data visualization. Finally i am going to use machine learning techniques to understand which is the best model for predicting if a rocket lauch will be successful or not . The point is to find out when and why a rocket launch is successful or not and what factors come into play.

CHAPTER 1
WEEK 1

DATA COLLECTION API

USING THE LABS ENVIRONMENT

SpaceX Falcon 9 first stage Landing Prediction

Lab 1: Collecting the data

Estimated time needed: **45** minutes

In this capstone, 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 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. In this lab, you will collect and make sure the data is in the correct format from an API. The following is an example of a successful and launch.

Objectives

In this lab, you will make a get request to the SpaceX API. You will also do some basic data wrangling and formating.

- Request to the SpaceX API
- Clean the requested data

Import Libraries and Define Auxiliary Functions

We will import the following libraries into the lab

[1]:
```
# Requests allows us to make HTTP requests which we will use to get data from an API
import requests
# Pandas is a software library written for the Python programming language for data manipulation and analysis.
import pandas as pd
# NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level m
import numpy as np
# Datetime is a library that allows us to represent dates
import datetime

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

Below we will define a series of helper functions that will help us use the API to extract information using identification numbers in the launch data.

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

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

[2]:
```
# 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.

[3]:
```
# 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.

[4]:
```
# 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.

core has been reused, and the serial of the core.

[5]:
```
# 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'])
```

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

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

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

Check the content of the response

[8]:
```
print(response.content)
```

```
b'[{"fairings":{"reused":false,"recovery_attempt":false,"recovered":false,"ships":[]},"links":{"patch":{"small":"https://images2.imgbox.com/94/f2/NN6Ph45r_o.png","lar
e":"https://images2.imgbox.com/5b/02/QcxHUb5V_o.png"},"reddit":{"campaign":null,"launch":null,"media":null,"recovery":null},"flickr":{"small":[],"original":[]},"press
t":null,"webcast":"https://www.youtube.com/watch?v=0a_00nJ_Y88","youtube_id":"0a_00nJ_Y88","article":"https://www.space.com/2196-spacex-inaugural-falcon-1-rocket-lost
aunch.html","wikipedia":"https://en.wikipedia.org/wiki/DemoSat"},"static_fire_date_utc":"2006-03-17T00:00:00.000Z","static_fire_date_unix":1142553600,"net":false,"wir
w":0,"rocket":"5e9d0d95eda69955f709d1eb","success":false,"failures":[{"time":33,"altitude":null,"reason":"merlin engine failure"}],"details":"Engine failure at 33 sec
ds and loss of vehicle","crew":[],"ships":[],"capsules":[],"payloads":["5eb0e4b5b6c3bb0006eeb1e1"],"launchpad":"5e9e4502f5090995de566f86","flight_number":1,"name":"Fa
```

s://youtu.be/5eWW8zKAPL4","youtube_id":"5eWW8zKAPL4","article":null,"wikipedia":"https://en.wikipedia.org/wiki/SpaceX_Crew-5"},"static_fire_date_utc":null,"static_fire_date_unix":null,"net":false,"window":null,"rocket":"5e9d0d95eda69973a809d1ec","success":true,"failures":[],"details":null,"crew":["62dd7196202306255024d13c","62dd71c9202306255024d13d","62dd7210202306255024d13e","62dd7253202306255024d13f"],"ships":[],"capsules":["617c05591bad2c661a6e2909"],"payloads":["62dd73ed202306255024d145"],"launchpad":"5e9e4502f509094188566f88","flight_number":187,"name":"Crew-5","date_utc":"2022-10-05T16:00:00.000Z","date_unix":1664985600,"date_local":"2022-10-05T12:00:00-04:00","date_precision":"hour","upcoming":false,"cores":[{"core":"633d9da635a71d1d9c66797b","flight":1,"gridfins":true,"legs":true,"reused":false,"landing_attempt":true,"landing_success":true,"landing_type":"ASDS","landpad":"5e9e3033383ecbb9e534e7cc"}],"auto_update":true,"tbd":false,"launch_library_id":"f33d5ece-e825-4cd8-809f-1d4c72a2e0d3","id":"62dd70d5202306255024d139"}]'

You should see the response contains massive information about SpaceX launches. Next, let's try to discover some more relevant information for this project.

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:

[9]:
```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_call_spacex_api.json'
```

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

[10]:
```
response.status_code
```

[10]:
```
200
```

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

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

Using the dataframe `data` print the first 5 rows

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

WE SEE SOME OF THE DATA HERE

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

[12]:

	static_fire_date_utc	static_fire_date_unix	net	window	rocket	success	failures	details	crew	ships	capsules	payloads	launchpad	flight_numbe
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	[]	[]	[]	[5eb0e4b5b6c3bb0006eeb1e1]	5e9e4502f5090995de566f86	1
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	[]	[]	[]	[5eb0e4b6b6c3bb0006eeb1e2]	5e9e4502f5090995de566f86	2
2	None	NaN	False	0.0	5e9d0d95eda69955f709d1eb	False	[{'time': 140, 'altitude': 35, 'reason': 'residual stage-1 thrust led to collision between stage 1 and stage 2'}]	Residual stage 1 thrust led to collision between stage 1 and stage 2	[]	[]	[]	[5eb0e4b6b6c3bb0006eeb1e3, 5eb0e4b6b6c3bb0006eeb1e4]	5e9e4502f5090995de566f86	3

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

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`.

[13]:
```
# 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 in a single rocket.
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()
```

[13]:

	rocket	payloads	launchpad	cores	flight_number	date_utc	date
0	5e9d0d95eda69955f709d1eb	5eb0e4b5b6c3bb0006eeb1e1	5e9e4502f5090995de566f86	{'core': '5e9e289df35918033d3b2623', 'flight': 1, 'gridfins': False, 'legs': False, 'reused': False, 'landing_attempt': False, 'landing_success': None, 'landing_type': None, 'landpad': None}	1	2006-03-24T22:30:00.000Z	2006-03-24
1	5e9d0d95eda69955f709d1eb	5eb0e4b6b6c3bb0006eeb1e2	5e9e4502f5090995de566f86	{'core': '5e9e289ef35918416a3b2624', 'flight': 1, 'gridfins': False, 'legs': False, 'reused': False, 'landing_attempt': False, 'landing_success': None, 'landing_type': None, 'landpad': None}	2	2007-03-21T01:10:00.000Z	2007-03-21
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	2008-09-28T23:15:00.000Z	2008-09-28
4	5e9d0d95eda69955f709d1eb	5eb0e4b7b6c3bb0006eeb1e6	5e9e4502f5090995de566f86	{'core': '5e9e289ef359184f103b2627', 'flight': 1, 'gridfins': False, 'legs': False, 'reused': False, 'landing_attempt': False, 'landing_success': None, 'landing_type': None, 'landpad': None}	5	2009-07-13T03:35:00.000Z	2009-07-13
5	5e9d0d95eda69973a809d1ec	5eb0e4b7b6c3bb0006eeb1e7	5e9e4501f509094ba4566f84	{'core': '5e9e289ef359185f2b3b2628', 'flight': 1, 'gridfins': False, 'legs': False, 'reused': False, 'landing_attempt': False, 'landing_success': None, 'landing_type': None, 'landpad': None}	6	2010-06-04T18:45:00.000Z	2010-06-04

5	5e9d0d95eda69973a809d1ec	5eb0e4b7b6c3bb0006eeb1e7	5e9e4501f509094ba4566f84	False, 'landing_attempt': False, 'landing_success': None, 'landing_type': None, 'landpad': None}	6	2010-06-04T18:45:00.000Z	2010-06-04

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

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

[14]:
```
#Global variables
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
ReusedCount = []
Serial = []
Longitude = []
Latitude = []
```

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

[15]:
```
BoosterVersion
```

[15]:
```
[]
```

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

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

[16]:
```
# Call getBoosterVersion
getBoosterVersion(data)
```

the list has now been update

[17]:
```
BoosterVersion[0:5]
```

[17]:
```
['Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 9']
```

we can apply the rest of the functions here:

[18]:
```
# Call getLaunchSite
getLaunchSite(data)
```

[19]:
```
# Call getPayloadData
getPayloadData(data)
```

[20]:
```
# Call getCoreData
getCoreData(data)
```

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

[21]:
```
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}
```

```
'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.

[22]:
```
# Create a data from launch_dict
data = pd.DataFrame(launch_dict)
```

Show the summary of the dataframe

[23]:
```
# Show the head of the dataframe
data.head()
```

[23]:

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

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`.

[24]:
```
data_falcon9 = data[data['BoosterVersion']!='Falcon 1']
data.head()
```

[24]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude
0	1	2006-03-24	Falcon 1	20.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0	Merlin1A	167.743129	9.047721

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`.

[24]:
```
data_falcon9 = data[data['BoosterVersion']!='Falcon 1']
data.head()
```

[24]:

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

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

[25]:
```
FlightNumber      90
Date              90
BoosterVersion    90
PayloadMass       85
Orbit             90
LaunchSite        90
Outcome           90
Flights           90
GridFins          90
Reused            90
Legs              90
LandingPad        64
Block             90
ReusedCount       90
Serial            90
Longitude         90
Latitude          90
dtype: int64
```

Now that we have removed some values we should reset the FlgihtNumber column

Now that we have removed some values we should reset the FlgihtNumber column

[25]:
```
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9
```

[25]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude
4	1	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0003	-80.577366	28.561857
5	2	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0005	-80.577366	28.561857
6	3	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0007	-80.577366	28.561857
7	4	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None	1.0	0	B1003	-120.610829	34.632093
8	5	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B1004	-80.577366	28.561857
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
89	86	2020-09-03	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	2	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	12	B1060	-80.603956	28.608058
90	87	2020-10-06	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	3	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	13	B1058	-80.603956	28.608058
91	88	2020-10-18	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	6	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	12	B1051	-80.603956	28.608058
92	89	2020-10-24	Falcon 9	15600.0	VLEO	CCSFS SLC 40	True ASDS	3	True	True	True	5e9e3033383ecbb9e534e7cc	5.0	12	B1060	-80.577366	28.561857
93	90	2020-11-05	Falcon 9	3681.0	MEO	CCSFS SLC 40	True ASDS	1	True	False	True	5e9e3032383ecb6bb234e7ca	5.0	8	B1062	-80.577366	28.561857

90 rows × 17 columns

Data Wrangling

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

[26]:
```
data_falcon9.isnull().sum()
```

[26]:
```
FlightNumber      0
Date              0
BoosterVersion    0
```

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

[26]:
```
data_falcon9.isnull().sum()
```

[26]:
```
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.

[27]:
```
# Calculate the mean value of PayloadMass column
payload_mean = data['PayloadMass'].mean()


# Replace the np.nan values with its mean value
data['PayloadMass'].replace(to_replace=np.nan, value=payload_mean, inplace=True)

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

[27]:
```
FlightNumber      0
Date              0
BoosterVersion    0
```

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.

[27]:
```
# Calculate the mean value of PayloadMass column
payload_mean = data['PayloadMass'].mean()


# Replace the np.nan values with its mean value
data['PayloadMass'].replace(to_replace=np.nan, value=payload_mean, inplace=True)

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

[27]:
```
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
```

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

Now we should have no missing values in our dataset except for in `LandingPad`.

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.

[28]:
```
data_falcon9.to_csv('dataset_part_1.csv', index=False)
```

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

Now we should have no missing values in our dataset except for in `LandingPad`.

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.

[28]:
```
data_falcon9.to_csv('dataset_part_1.csv', index=False)
```

Authors

Joseph Santarcangelo has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-09-20	1.1	Joseph	get result each time you run
2020-09-20	1.1	Azim	Created Part 1 Lab using SpaceX API
2020-09-20	1.0	Joseph	Modified Multiple Areas

CHAPTER 2
WEEK 1

WEBSCRAPING

USING ANACONDA JUPYTER NOTEBOOK

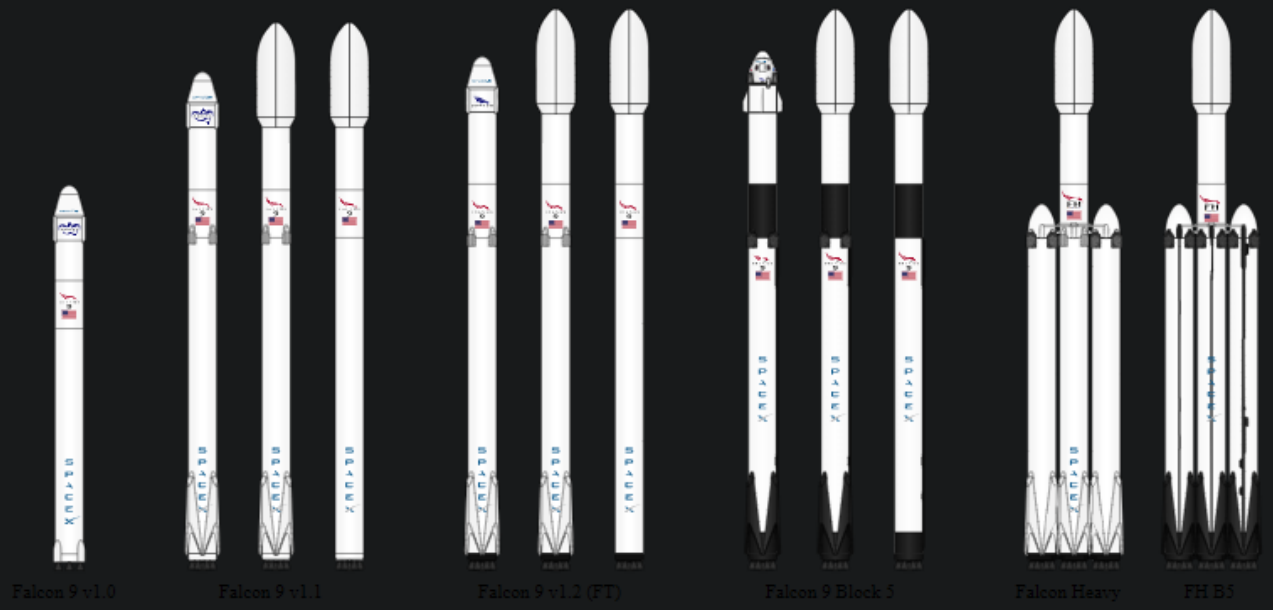
Space X Falcon 9 First Stage Landing Prediction

Web scraping Falcon 9 and Falcon Heavy Launches Records from Wikipedia

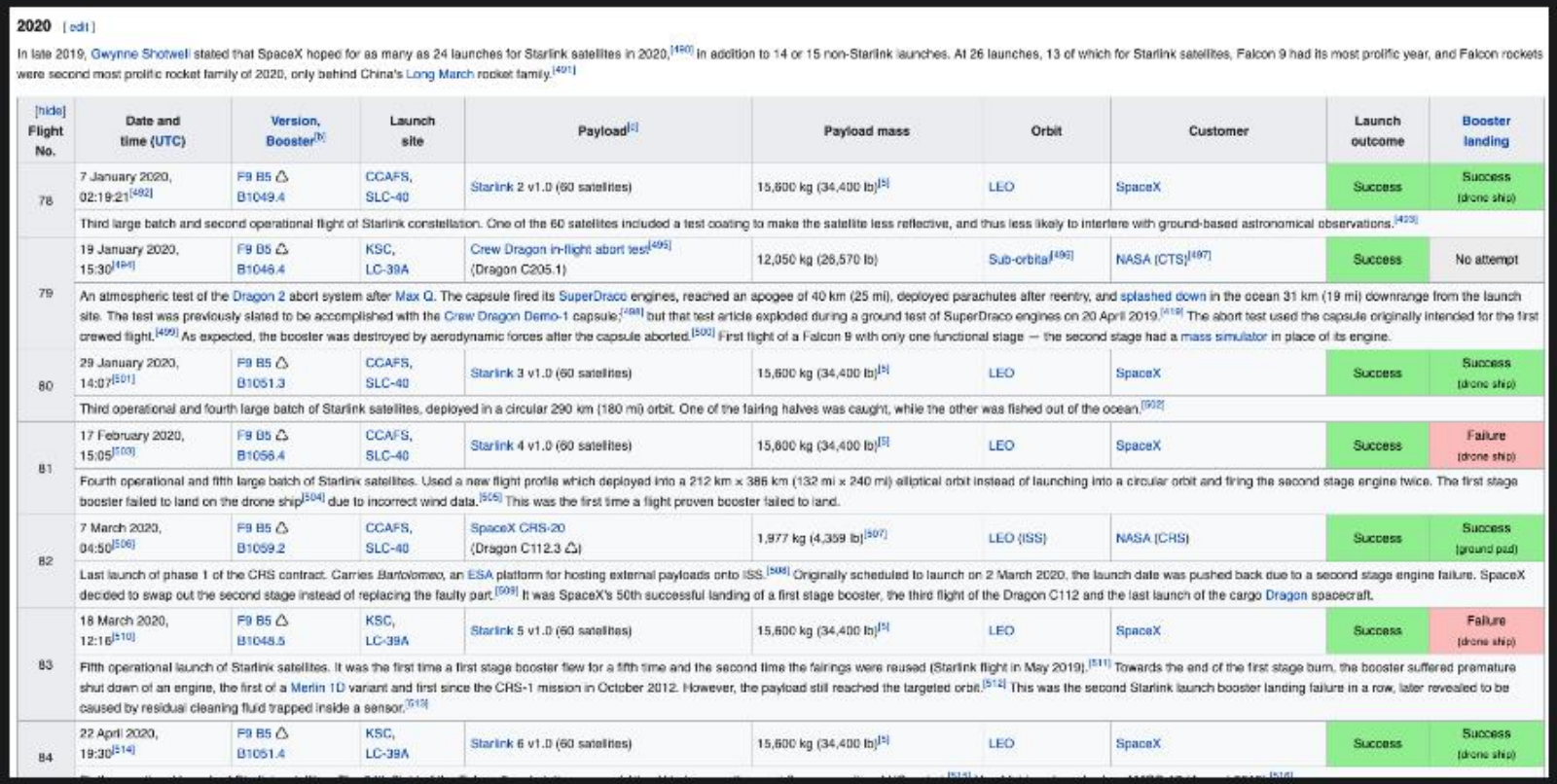
Estimated time needed: **40** minutes

In this lab, you will be performing web scraping to collect Falcon 9 historical launch records from a Wikipedia page titled `List of Falcon 9 and Falcon Heavy launches`

https://en.wikipedia.org/wiki/List_of_Falcon_9_and_Falcon_Heavy_launches



More specifically, the launch records are stored in a HTML table shown below:



2020 [edit]

In late 2019, Gwynne Shotwell stated that SpaceX hoped for as many as 24 launches for Starlink satellites in 2020,[490] in addition to 14 or 15 non-Starlink launches. At 26 launches, 13 of which for Starlink satellites, Falcon 9 had its most prolific year, and Falcon rockets were second most prolific rocket family of 2020, only behind China's Long March rocket family.[491]

[hide] Flight No.	Date and time (UTC)	Version, Booster[b]	Launch site	Payload[c]	Payload mass	Orbit	Customer	Launch outcome	Booster landing
78	7 January 2020, 02:19:21[492]	F9 B5 △ B1049.4	CCAFS, SLC-40	Starlink 2 v1.0 (60 satellites)	15,600 kg (34,400 lb)[5]	LEO	SpaceX	Success	Success (drone ship)
	Third large batch and second operational flight of Starlink constellation. One of the 60 satellites included a test coating to make the satellite less reflective, and thus less likely to interfere with ground-based astronomical observations.[423]								
79	19 January 2020, 15:30[494]	F9 B5 △ B1046.4	KSC, LC-39A	Crew Dragon in-flight abort test[495] (Dragon C205.1)	12,050 kg (26,570 lb)	Sub-orbital[496]	NASA (CTS)[497]	Success	No attempt
	An atmospheric test of the Dragon 2 abort system after Max Q. The capsule fired its SuperDraco engines, reached an apogee of 40 km (25 mi), deployed parachutes after reentry, and splashed down in the ocean 31 km (19 mi) downrange from the launch site. The test was previously slated to be accomplished with the Crew Dragon Demo-1 capsule,[498] but that test article exploded during a ground test of SuperDraco engines on 20 April 2019.[419] The abort test used the capsule originally intended for the first crewed flight.[499] As expected, the booster was destroyed by aerodynamic forces after the capsule aborted.[500] First flight of a Falcon 9 with only one functional stage — the second stage had a mass simulator in place of its engine.								
80	29 January 2020, 14:07[501]	F9 B5 △ B1051.3	CCAFS, SLC-40	Starlink 3 v1.0 (60 satellites)	15,600 kg (34,400 lb)[5]	LEO	SpaceX	Success	Success (drone ship)
	Third operational and fourth large batch of Starlink satellites, deployed in a circular 290 km (180 mi) orbit. One of the fairing halves was caught, while the other was fished out of the ocean.[502]								
81	17 February 2020, 15:05[503]	F9 B5 △ B1056.4	CCAFS, SLC-40	Starlink 4 v1.0 (60 satellites)	15,600 kg (34,400 lb)[5]	LEO	SpaceX	Success	Failure (drone ship)
	Fourth operational and fifth large batch of Starlink satellites. Used a new flight profile which deployed into a 212 km × 386 km (132 mi × 240 mi) elliptical orbit instead of launching into a circular orbit and firing the second stage engine twice. The first stage booster failed to land on the drone ship[504] due to incorrect wind data.[505] This was the first time a flight proven booster failed to land.								
82	7 March 2020, 04:50[506]	F9 B5 △ B1059.2	CCAFS, SLC-40	SpaceX CRS-20 (Dragon C112.3 △)	1,977 kg (4,359 lb)[507]	LEO (ISS)	NASA (CRS)	Success	Success (ground pad)
	Last launch of phase 1 of the CRS contract. Carries *Bartolomeo*, an ESA platform for hosting external payloads onto ISS.[508] Originally scheduled to launch on 2 March 2020, the launch date was pushed back due to a second stage engine failure. SpaceX decided to swap out the second stage instead of replacing the faulty part.[509] It was SpaceX's 50th successful landing of a first stage booster, the third flight of the Dragon C112 and the last launch of the cargo Dragon spacecraft.								
83	18 March 2020, 12:16[510]	F9 B5 △ B1048.5	KSC, LC-39A	Starlink 5 v1.0 (60 satellites)	15,600 kg (34,400 lb)[5]	LEO	SpaceX	Success	Failure (drone ship)
	Fifth operational launch of Starlink satellites. It was the first time a first stage booster flew for a fifth time and the second time the fairings were reused (Starlink flight in May 2019).[511] Towards the end of the first stage burn, the booster suffered premature shut down of an engine, the first of a Merlin 1D variant and first since the CRS-1 mission in October 2012. However, the payload still reached the targeted orbit.[512] This was the second Starlink launch booster landing failure in a row, later revealed to be caused by residual cleaning fluid trapped inside a sensor.[513]								
84	22 April 2020, 19:30[514]	F9 B5 △ B1051.4	KSC, LC-39A	Starlink 6 v1.0 (60 satellites)	15,600 kg (34,400 lb)[5]	LEO	SpaceX	Success	Success (drone ship)

Objectives

Web scrap Falcon 9 launch records with `BeautifulSoup`:

- Extract a Falcon 9 launch records HTML table from Wikipedia
- Parse the table and convert it into a Pandas data frame

First let's import required packages for this lab

In [1]:
```
!pip3 install beautifulsoup4
!pip3 install requests
```

```
Requirement already satisfied: beautifulsoup4 in c:\users\louky\anaconda3\lib\site-packages (4.12.2)
```

In [2]:
```
import sys

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

and we will provide some helper functions for you to process web scraped HTML table

In [3]:
```
def date_time(table_cells):
    """
    This function returns the data and time from the HTML  table cell
    Input: the  element of a table data cell extracts extra row
    """
    return [data_time.strip() for data_time in list(table_cells.strings)][0:2]

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

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


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


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

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

    colunm_name = ' '.join(row.contents)

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

To keep the lab tasks consistent, you will be asked to scrape the data from a snapshot of the `List of Falcon 9 and Falcon Heavy launches` Wikipage updated on `9th June 2021`

In [4]:
```
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.

In [5]:
```
# use requests.get() method with the provided static_url
response = requests.get(static_url)
# assign the response to a object
```

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

In [6]:
```
# 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

In [7]:
```
# Use soup.title attribute
soup.title
```

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

In [7]:
```
# Use soup.title attribute
soup.title
```

Out[7]:
```
<title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

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

In [8]:
```
# Use the find_all function in the BeautifulSoup object, with element type `table`
# Assign the result to a list called `html_tables`
html_tables = soup.find_all('table')
```

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

In [9]:
```
# Let's print the third table and check its content
first_launch_table = html_tables[2]
print(first_launch_table)
```

```
<table class="wikitable plainrowheaders collapsible" style="width: 100%;">
<tbody><tr>
<th scope="col">Flight No.
</th>
<th scope="col">Date and<br/>time (<a href="/wiki/Coordinated_Universal_Time" title="Coordinated Universal Time">UTC</a>)
</th>
<th scope="col"><a href="/wiki/List_of_Falcon_9_first-stage_boosters" title="List of Falcon 9 first-stage boosters">Version,<
br/>Booster</a> <sup class="reference" id="cite_ref-booster_11-0"><a href="#cite_note-booster-11">[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>
```

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

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">Vers
ion,<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<b
r/>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

In [10]:
```
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
for th in first_launch_table.find_all('th'):
    name = extract_column_from_header(th)
    if name is not None and len(name) > 0:
        column_names.append(name)
```

Check the extracted column names

In [11]:
```
print(column_names)
```

In [11]:
```
print(column_names)
```

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

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

In [12]:
```
launch_dict= dict.fromkeys(column_names)

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

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

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

Usually, HTML tables in Wiki 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:

In [13]:
```
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()
```

In [13]:

```
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)
            launch_dict['Flight No.'].append(flight_number)
            datatimelist=date_time(row[0])

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

            # Time value
            # TODO: Append the time into launch_dict with key `Time`
            time = datatimelist[1]
            launch_dict['Time'].append(time)
            #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
            launch_dict['Version Booster'].append(bv)
            print(bv)

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

            # Payload
            # TODO: Append the payload into launch_dict with key `Payload`
```

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

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

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

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

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

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

            # Booster landing
            # TODO: Append the launch_outcome into launch_dict with key `Booster landing`
            booster_landing = landing_status(row[8])
            launch_dict['Booster landing'].append(booster_landing)
            #print(booster_landing)
df = pd.DataFrame(launch_dict)
```

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

```
F9 v1.1
F9 v1.1
F9 v1.1
F9 v1.1
F9 v1.1
F9 v1.1
F9 v1.1
F9 v1.1
F9 v1.1
F9 v1.1
```

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

In [14]:
```
df= pd.DataFrame({ key:pd.Series(value) for key, value in launch_dict.items() })
```

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.

In [15]:
```
df.to_csv('spacex_web_scraped.csv', index=False)
```

Authors

Yan Luo

Nayef Abou Tayoun

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-06-09	1.0	Yan Luo	Tasks updates
2020-11-10	1.0	Nayef	Created the initial version

CHAPTER 3
WEEK 1

DATA WRANGLING

USING ANACONDA JUPYTER NOTEBOOK

Space X Falcon 9 First Stage Landing Prediction

Lab 2: Data wrangling

Estimated time needed: 60 minutes

In this lab, 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.

Falcon 9 first stage will land successfully

Objectives

Perform exploratory Data Analysis and determine Training Labels

- Exploratory Data Analysis
- Determine Training Labels

Import Libraries and Define Auxiliary Functions

We will import the following libraries.

In [1]:
```
# Pandas is a software library written for the Python programming language for data manipulation and analysis.
import pandas as pd
#NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along w
import numpy as np
```

Data Analysis

Load Space X dataset, from last section.

In [2]:
```
df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_pa
df.head(10)
```

Out[2]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	0	B1003
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004
5	6	2014-01-06	Falcon 9	3325.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1005

5	6	2014-01-06	Falcon 9	3325.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1005
6	7	2014-04-18	Falcon 9	2296.000000	ISS	CCAFS SLC 40	True Ocean	1	False	False	True	NaN	1.0	0	B1006
7	8	2014-07-14	Falcon 9	1316.000000	LEO	CCAFS SLC 40	True Ocean	1	False	False	True	NaN	1.0	0	B1007
8	9	2014-08-05	Falcon 9	4535.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1008
9	10	2014-09-07	Falcon 9	4428.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1011

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

In [3]:
```
df.isnull().sum()/len(df)*100
```

Out[3]:
```
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:

In [4]:
```
df.dtypes
```

Out[4]:
```
FlightNumber      int64
Date             object
BoosterVersion   object
PayloadMass     float64
Orbit            object
LaunchSite       object
Outcome          object
Flights           int64
GridFins           bool
Reused             bool
```

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:

In [5]:
```
# Apply value_counts() on column LaunchSite
df['LaunchSite'].value_counts()
```

Out[5]:
```
LaunchSite
CCAFS SLC 40    55
KSC LC 39A      22
VAFB SLC 4E     13
Name: count, dtype: int64
```

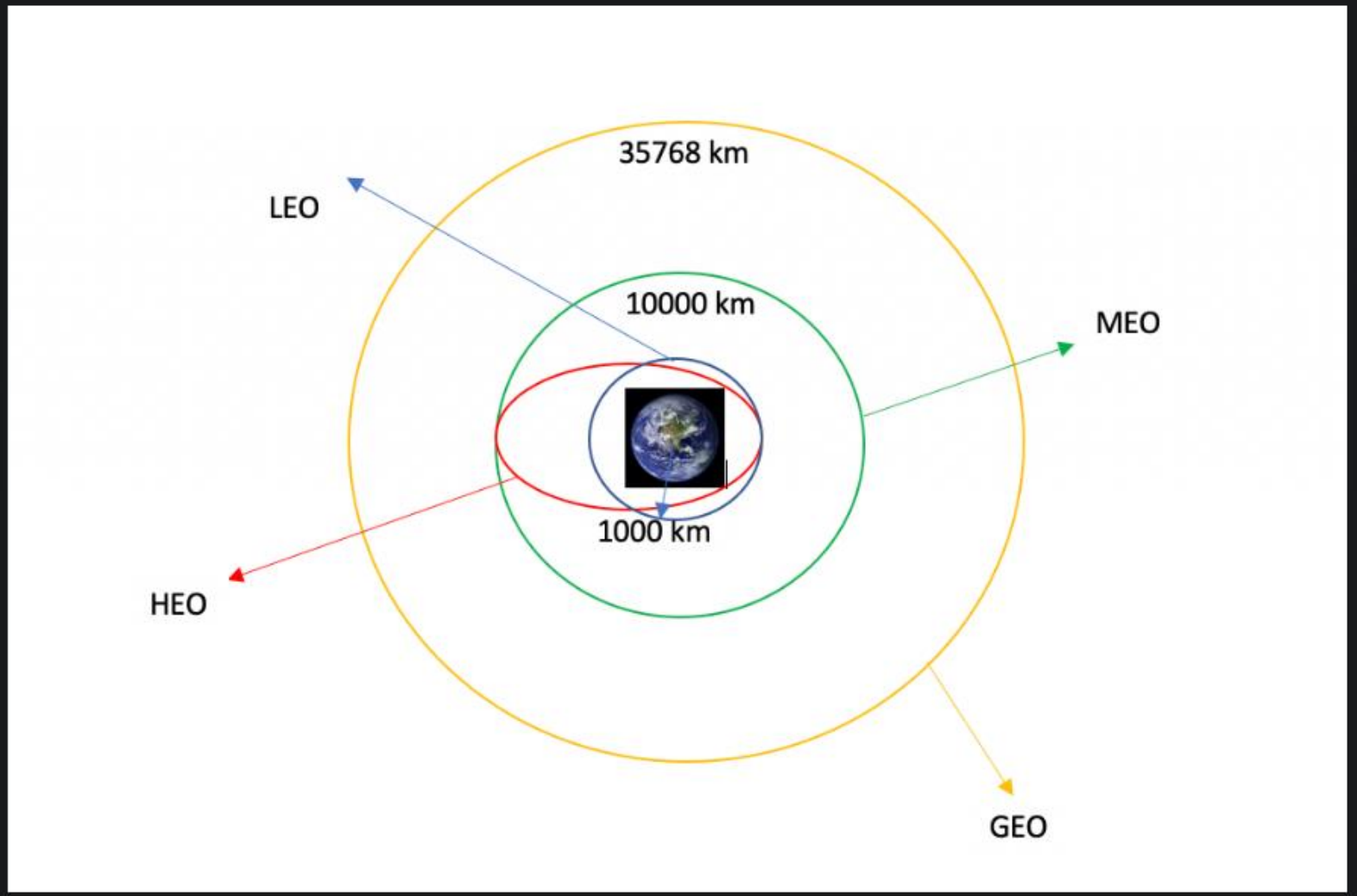
Each launch aims to an dedicated orbit, and here are some common orbit types:

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

some are shown in the following plot:

- PO It is one type of satellites in which a satellite passes above or nearly above both poles of the body being orbited (usually a planet such as the Earth [11]

some are shown in the following plot:



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`

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`

In [6]:
```
# Apply value_counts on Orbit column
df['Orbit'].value_counts()
```

Out[6]:
```
Orbit
GTO      27
ISS      21
VLEO     14
PO        9
LEO       7
SSO       5
MEO       3
ES-L1     1
HEO       1
SO        1
GEO       1
Name: count, 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.

In [7]:
```
# landing_outcomes = values on Outcome column
landing_outcomes = df['Outcome'].value_counts()
landing_outcomes
```

Out[7]:
```
Outcome
True ASDS      41
None None      19
True RTLS      14
False ASDS      6
True Ocean      5
False Ocean     2
None ASDS       2
False RTLS      1
Name: count, dtype: int64
```

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

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.

In [8]:
```
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:

In [9]:
```
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
bad_outcomes
```

Out[9]:
```
{'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`:

In [10]:
```
# 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']]
landing_class
```

Out[10]:
```
[0,
 0,
 0,
 0,
 0,
 0,
 1,
 1,
 0,
 0,
 0,
 0,
 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

In [11]:
```
df['Class']=landing_class
df[['Class']].head(8)
```

Out[11]:

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

In [12]:
```
df.head(5)
```

Out[12]:

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

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

In [13]:
```
df["Class"].mean()
```

Out[13]:
```
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.

In [14]:
```
df.to_csv("dataset_part_2.csv", index=False)
```

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.

In [14]:
```
df.to_csv("dataset_part_2.csv", index=False)
```

Authors

Joseph Santarcangelo has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Nayef Abou Tayoun is a Data Scientist at IBM and pursuing a Master of Management in Artificial intelligence degree at Queen's University.

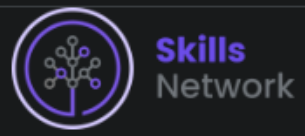
Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-08-31	1.1	Lakshmi Holla	Changed Markdown
2020-09-20	1.0	Joseph	Modified Multiple Areas
2020-11-04	1.1.	Nayef	updating the input data
2021-05-026	1.1.	Joseph	updating the input data

CHAPTER 4
WEEK 2

EDA WITH SQL

USING THE LABS ENVIRONMENT



Assignment: SQL Notebook for Peer Assignment

Estimated time needed: **60** minutes.

Introduction

Using this Python notebook you will:

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

Overview of the DataSet

SpaceX has gained worldwide attention for a series of historic milestones.

It is the only private company ever to return a spacecraft from low-earth orbit, which it first accomplished in December 2010. SpaceX advertises Falcon 9 rocket launches on its website with a of 62 million dollars wheras other providers cost upward of 165 million dollars each, much of the savings is because Space X 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.

This dataset includes a record for each payload carried during a SpaceX mission into outer space.

Download the datasets

This assignment requires you to load the spacex dataset.

In many cases the dataset to be analyzed is available as a .CSV (comma separated values) file, perhaps on the internet. Click on the link below to download and save the dataset (.CSV file):

This assignment requires you to load the spacex dataset.

In many cases the dataset to be analyzed is available as a .CSV (comma separated values) file, perhaps on the internet. Click on the link below to download and save the dataset (.CSV file):

Spacex DataSet

[1]:
```
!pip install sqlalchemy==1.3.9
```

```
Requirement already satisfied: sqlalchemy==1.3.9 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (1.3.9)
```

Connect to the database

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

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

[3]:
```
import csv, sqlite3

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

[4]:
```
!pip install -q pandas==1.1.5
```

[5]:
```
%sql sqlite:///my_data1.db
```

[5]:
```
'Connected: @my_data1.db'
```

[6]:
```
import pandas as pd
df = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/labs/module_2/data/Spacex.csv")
df.to_sql("SPACEXTBL", con, if_exists='replace', index=False,method="multi")
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/pandas/core/generic.py:2615: UserWarning: The spaces in these column names will not be changed. In pandas versions < 0.14, spaces were converted to underscores.
  method=method,
```

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

[7]:
```
%sql create table SPACEXTABLE as select * from SPACEXTBL where Date is not null
```

```
 * sqlite:///my_data1.db
(sqlite3.OperationalError) table SPACEXTABLE already exists
```

[7]:
```
%sql create table SPACEXTABLE as select * from SPACEXTBL where Date is not null
```

```
 * sqlite:///my_data1.db
(sqlite3.OperationalError) table SPACEXTABLE already exists
[SQL: create table SPACEXTABLE as select * from SPACEXTBL where Date is not null]
(Background on this error at: http://sqlalche.me/e/e3q8)
```

Tasks

Now write and execute SQL queries to solve the assignment tasks.

Note: If the column names are in mixed case enclose it in double quotes For Example "Landing_Outcome"

Task 1

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

[11]:
```
%sql select DISTINCT Launch_Site from SPACEXTABLE
```

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

[11]:

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

Task 2

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

[13]:
```
%sql select * from SPACEXTABLE where Launch_Site  LIKE 'CCA%' LIMIT 5
```

Task 2

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

[13]:
```
%sql select * from SPACEXTABLE where Launch_Site  LIKE 'CCA%' LIMIT 5
```

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

[13]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Task 3

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

[25]:
```
%sql select SUM(PAYLOAD_MASS__KG_) from SPACEXTABLE where Customer='NASA (CRS)'
```

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

[25]:

SUM(PAYLOAD_MASS__KG_)
45596

Task 4

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

[26]:
```
%sql select AVG(PAYLOAD_MASS__KG_) from SPACEXTABLE where  Booster_Version like 'F9 v1.1%'
```

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

[26]:

AVG(PAYLOAD_MASS__KG_)
2534.6666666666665

Task 5

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

Hint:Use min function

[31]:
```
%sql select min(Date) from  SPACEXTABLE where Landing_Outcome='Success'
```

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

[31]:

min(Date)
2018-07-22

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

[35]:
```
%sql select distinct(Booster_Version) from SPACEXTABLE where PAYLOAD_MASS__KG_ between 4000 and 6000 and Landing_Outcome='Success (drone ship)'
```

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

[35]:

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

Task 7

List the total number of successful and failure mission outcomes

[50]:
```
%sql select count(Mission_Outcome),Mission_Outcome from SPACEXTABLE group by Mission_Outcome
```

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

[50]:

count(Mission_Outcome)	Mission_Outcome
1	Failure (in flight)

Task 7

List the total number of successful and failure mission outcomes

[50]:
```
%sql select count(Mission_Outcome),Mission_Outcome from SPACEXTABLE group by Mission_Outcome
```

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

[50]:

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

Task 8

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

[55]:
```
%sql select  DISTINCT(Booster_Version),PAYLOAD_MASS__KG_ from SPACEXTABLE where PAYLOAD_MASS__KG_ = (select MAX(PAYLOAD_MASS__KG_) from SPACEXTABLE)
```

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

[55]:

Booster_Version	PAYLOAD_MASS__KG_
F9 B5 B1048.4	15600
F9 B5 B1049.4	15600
F9 B5 B1051.3	15600
F9 B5 B1056.4	15600
F9 B5 B1048.5	15600
F9 B5 B1051.4	15600
F9 B5 B1049.5	15600
F9 B5 B1060.2	15600
F9 B5 B1058.3	15600

F9 B5 B1060.3	15600
F9 B5 B1049.7	15600

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.

[69]:
```
%sql select * from SPACEXTABLE  where Landing_Outcome like 'Success (%' limit 8
```

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

[69]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2015-12-22	1:29:00	F9 FT B1019	CCAFS LC-40	OG2 Mission 2 11 Orbcomm-OG2 satellites	2034	LEO	Orbcomm	Success	Success (ground pad)
2016-04-08	20:43:00	F9 FT B1021.1	CCAFS LC-40	SpaceX CRS-8	3136	LEO (ISS)	NASA (CRS)	Success	Success (drone ship)
2016-05-06	5:21:00	F9 FT B1022	CCAFS LC-40	JCSAT-14	4696	GTO	SKY Perfect JSAT Group	Success	Success (drone ship)
2016-05-27	21:39:00	F9 FT B1023.1	CCAFS LC-40	Thaicom 8	3100	GTO	Thaicom	Success	Success (drone ship)
2016-07-18	4:45:00	F9 FT B1025.1	CCAFS LC-40	SpaceX CRS-9	2257	LEO (ISS)	NASA (CRS)	Success	Success (ground pad)
2016-08-14	5:26:00	F9 FT B1026	CCAFS LC-40	JCSAT-16	4600	GTO	SKY Perfect JSAT Group	Success	Success (drone ship)
2017-01-14	17:54:00	F9 FT B1029.1	VAFB SLC-4E	Iridium NEXT 1	9600	Polar LEO	Iridium Communications	Success	Success (drone ship)
2017-02-19	14:39:00	F9 FT B1031.1	KSC LC-39A	SpaceX CRS-10	2490	LEO (ISS)	NASA (CRS)	Success	Success (ground pad)

[64]:
```
%sql select substr(Date, 6, 2) as Month ,substr(Date, 0, 5) as Year,Launch_Site,Booster_Version,Landing_Outcome from SPACEXTABLE WHERE substr(Date, 0, 5) = '2015' AND Landing_Outcome like 'Failure%'
```

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

[64]:

Month	Year	Launch_Site	Booster_Version	Landing_Outcome
01	2015	CCAFS LC-40	F9 v1.1 B1012	Failure (drone ship)
04	2015	CCAFS LC-40	F9 v1.1 B1015	Failure (drone ship)

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.

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.

[92]:
```
%sql select Landing_Outcome,count(*) as count_of_outcomes from SPACEXTABLE where ((Landing_Outcome='Success (ground pad)' or Landing_Outcome='Failure (drone ship)')  AND (Date between '2010-06-04' and '2017-03-20')) group by Landing_Outcome order by count_of_outcomes desc
```

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

[92]:

Landing_Outcome	count_of_outcomes
Failure (drone ship)	5
Success (ground pad)	3

Reference Links

- Hands-on Lab : String Patterns, Sorting and Grouping
- Hands-on Lab: Built-in functions
- Hands-on Lab : Sub-queries and Nested SELECT Statements
- Hands-on Tutorial: Accessing Databases with SQL magic
- Hands-on Lab: Analyzing a real World Data Set

Author(s)

Lakshmi Holla

Other Contributors

Rav Ahuja

Change log

Date	Version	Changed by	Change Description
2021-07-09	0.2	Lakshmi Holla	Changes made in magic sql
2021-05-20	0.1	Lakshmi Holla	Created Initial Version

CHAPTER 5
WEEK 2

EDA DATAVIZ

(dark reader extension stopped working here because of the unique cells in jupyterlite)

USING THE LABS ENVIRONMENT

SpaceX Falcon 9 First Stage Landing Prediction

Assignment: Exploring and Preparing Data

Estimated time needed: **70** minutes

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.

In this lab, you will perform Exploratory Data Analysis and Feature Engineering.

Falcon 9 first stage will land successfully

Several examples of an unsuccessful landing are shown here:

Objectives

Perform exploratory Data Analysis and Feature Engineering using `Pandas` and `Matplotlib`

- Exploratory Data Analysis
- Preparing Data Feature Engineering

Import Libraries and Define Auxiliary Functions ¶

We will import the following libraries the lab

[1]:
```
import piplite
await piplite.install(['numpy'])
await piplite.install(['pandas'])
await piplite.install(['seaborn'])
```

[2]:
```
# pandas is a software library written for the Python programming language for data manipulation and analysis.
import pandas as pd
#NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to opera
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a MatLab like plotting framework. We will use this in our plotter function to plot data.
import matplotlib.pyplot as plt
#Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics
import seaborn as sns
```

[3]:
```
## Exploratory Data Analysis
```

First, let's read the SpaceX dataset into a Pandas dataframe and print its summary

[4]:
```
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)
```

```
df=pd.read_csv(dataset_part_2_csv)
df.head(5)
```

[4]:

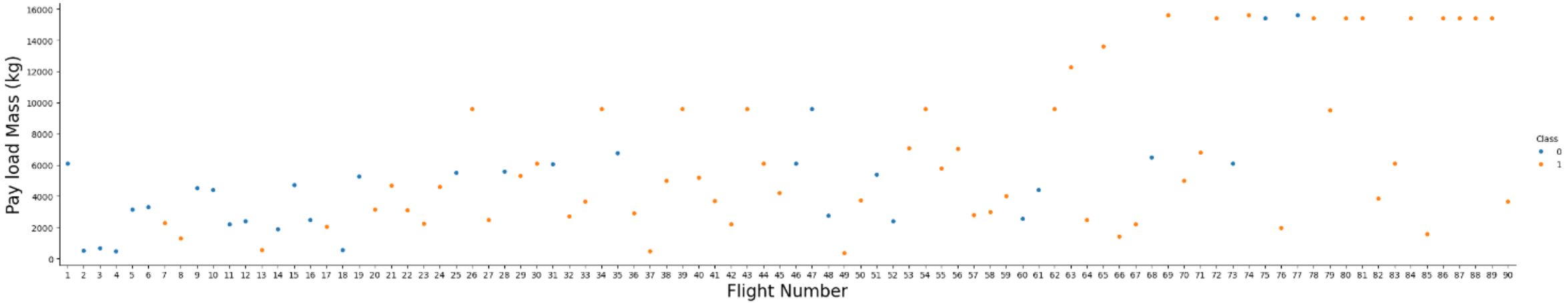
	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577366	28.561857	0
1	2	2012-05-22	Falcon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

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.

[5]:

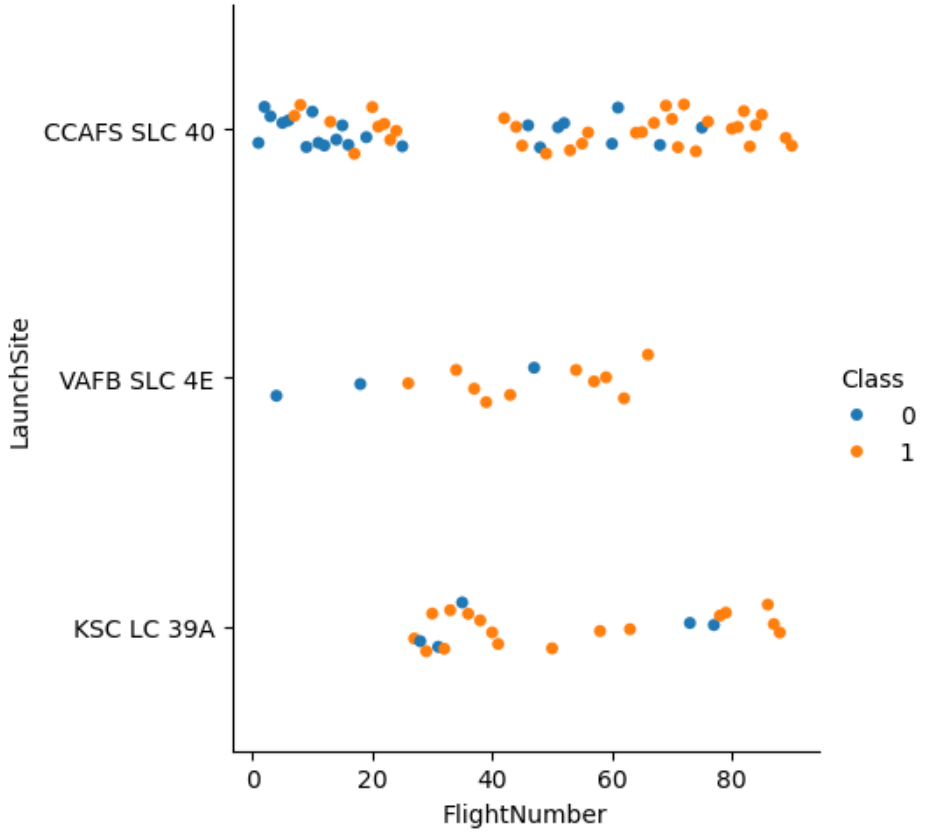
```
sns.catplot(y="PayloadMass", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Pay load Mass (kg)",fontsize=20)
plt.show()
```



We see that different launch sites have different success rates. `CCAFS LC-40`, has a success rate of 60 %, while `KSC LC-39A` and `VAFB SLC 4E` has a success rate of 77%.

Next, let's drill down to each site visualize its detailed launch records.

[8]:
```
### TASK 1: Visualize the relationship between Flight Number and Launch Site
sns.catplot(x='FlightNumber', y='LaunchSite', hue='Class', data=df)
plt.show()
```



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

[ ]:
```
# Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to be the class value
```

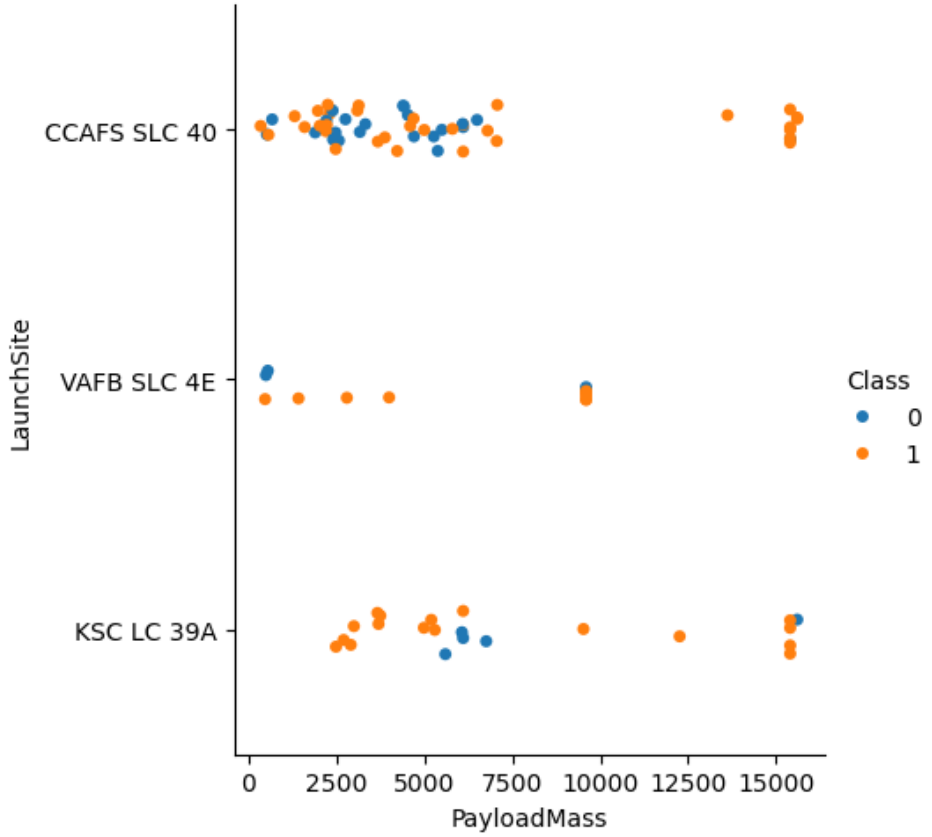
Now try to explain the patterns you found in the Flight Number vs. Launch Site scatter point plots.

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

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

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

11]:
```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch site, and hue to be the class value
sns.catplot(x='PayloadMass', y='LaunchSite', hue='Class', data=df)
plt.show()
```



Now if you observe Payload Vs. Launch Site scatter point chart you will find for the VAFB-SLC launchsite there are no rockets launched for heavypayload mass(greater than 10000).

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

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

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

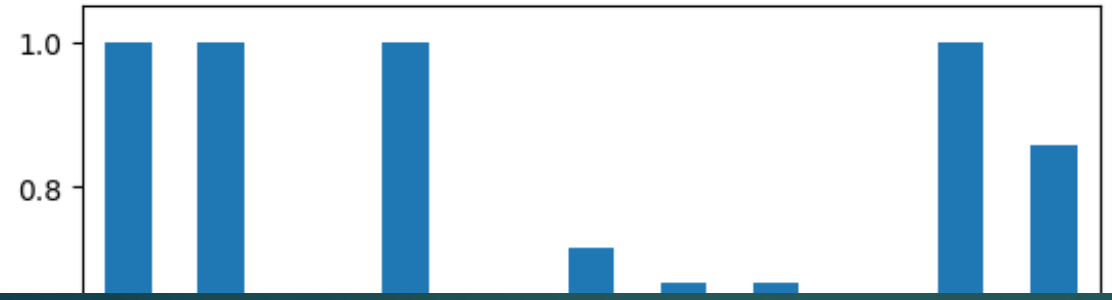
Let's create a `bar chart` for the sucess rate of each orbit

[15]:
```
# HINT use groupby method on Orbit column and get the mean of Class column
orbit_success_rate = df.groupby('Orbit')['Class'].mean()
orbit_success_rate
```

[15]:
```
Orbit
ES-L1    1.000000
GEO      1.000000
GTO      0.518519
HEO      1.000000
ISS      0.619048
LEO      0.714286
MEO      0.666667
PO       0.666667
SO       0.000000
SSO      1.000000
VLEO     0.857143
Name: Class, dtype: float64
```

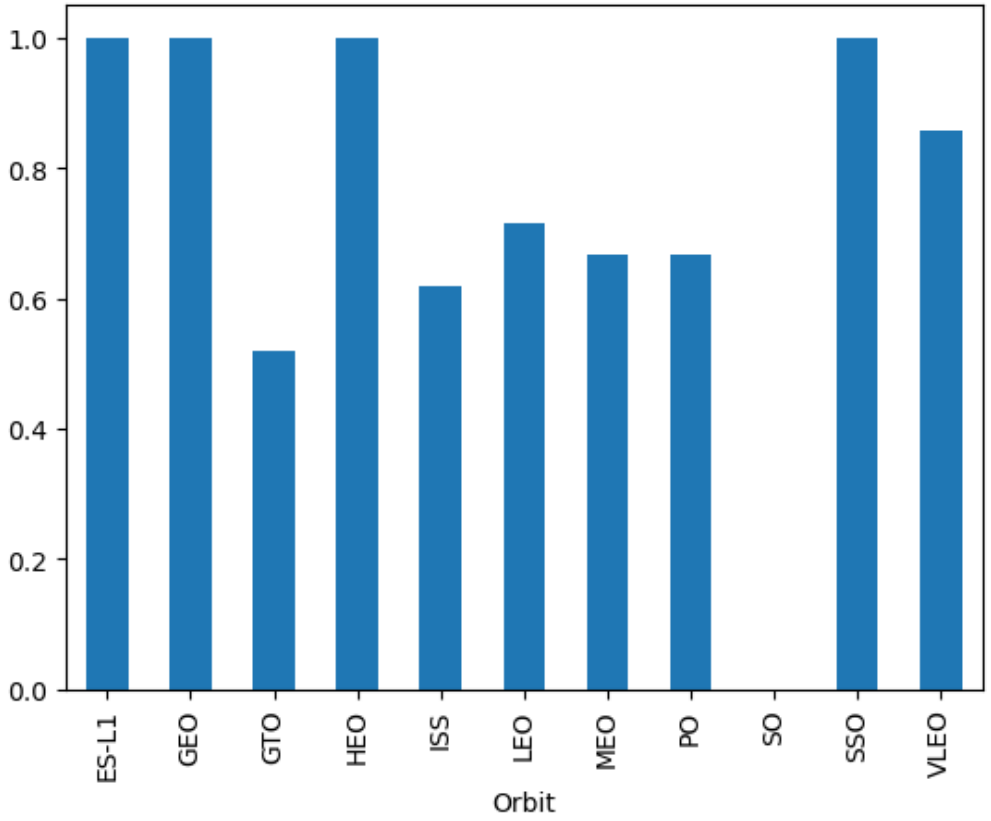
[19]:
```
orbit_success_rate.plot(kind='bar')
```

[19]:
```
<AxesSubplot:xlabel='Orbit'>
```



[19]:
```
orbit_success_rate.plot(kind='bar')
```

[19]:
```
<AxesSubplot:xlabel='Orbit'>
```



Analyze the ploted bar chart try to find which orbits have high sucess rate.

[ ]:
```
### TASK  4: Visualize the relationship between FlightNumber and Orbit type
```

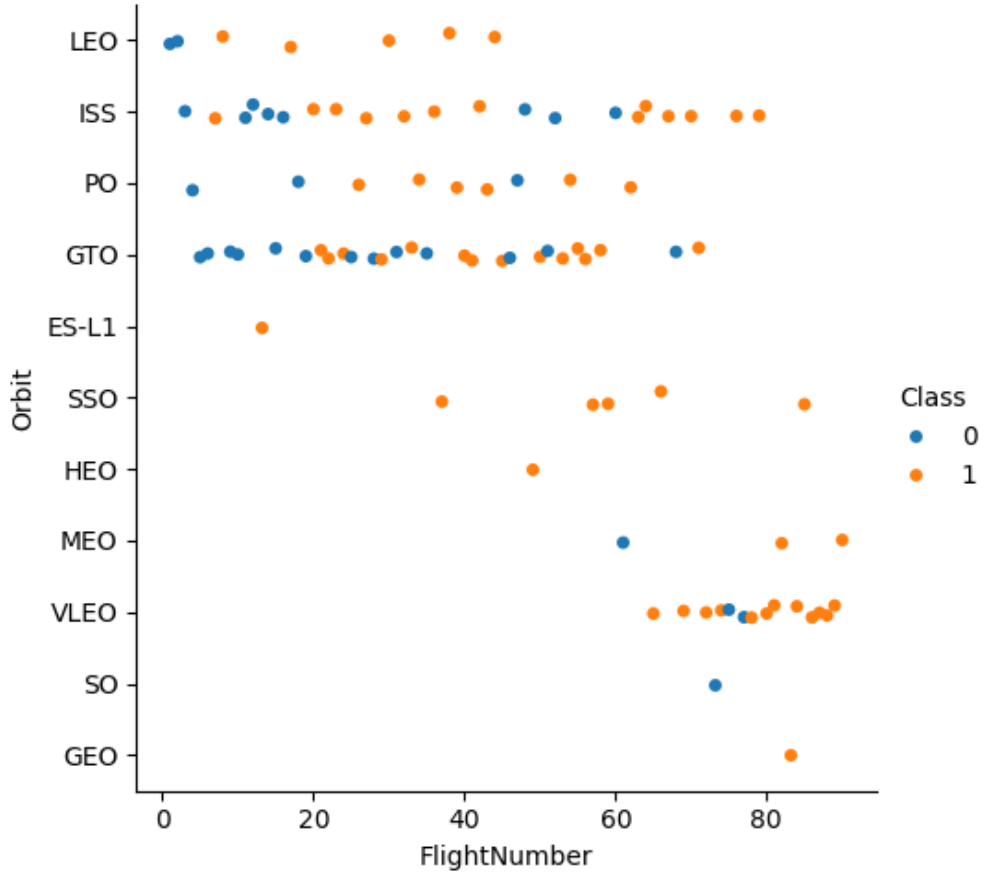
For each orbit, we want to see if there is any relationship between FlightNumber and Orbit type.

[20]:
```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class value
sns.catplot(x='FlightNumber', y='Orbit', hue='Class', data=df)
```

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

[20]:
```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class value
sns.catplot(x='FlightNumber', y='Orbit', hue='Class', data=df)
```

[20]:
```
<seaborn.axisgrid.FacetGrid at 0x6c595d8>
```



You should see that 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.

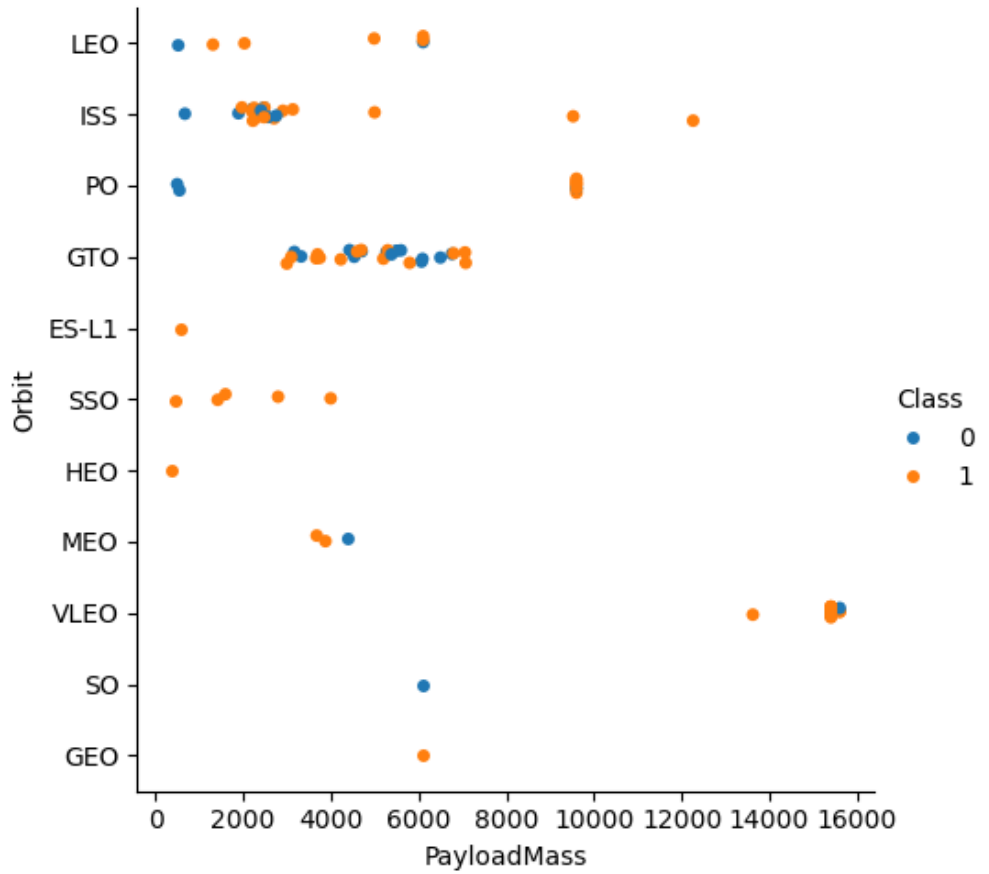
[22]:
```
### TASK  5: Visualize the relationship between Payload and Orbit type
sns.catplot(x='PayloadMass', y='Orbit', hue='Class', data=df)
```

[22]:
```
<seaborn.axisgrid.FacetGrid at 0x6fa98a0>
```

You should see that 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.

[22]:
```
### TASK  5: Visualize the relationship between Payload and Orbit type
sns.catplot(x='PayloadMass', y='Orbit', hue='Class', data=df)
```

[22]:
```
<seaborn.axisgrid.FacetGrid at 0x6fa98a0>
```



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

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

With heavy payloads the successful landing or positive landing rate are more for Polar,LEO and ISS.

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

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

With heavy payloads the successful landing or positive landing rate are more for Polar,LEO and ISS.

However for GTO we cannot distinguish this well as both positive landing rate and negative landing(unsuccessful mission) are both there here.

[ ]:
```
### TASK  6: Visualize the launch success yearly trend
```

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

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

[23]:
```
# A function to Extract years from the date
year=[]
def Extract_year():
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year
Extract_year()
df['Date'] = year
df.head()
```

[23]:

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

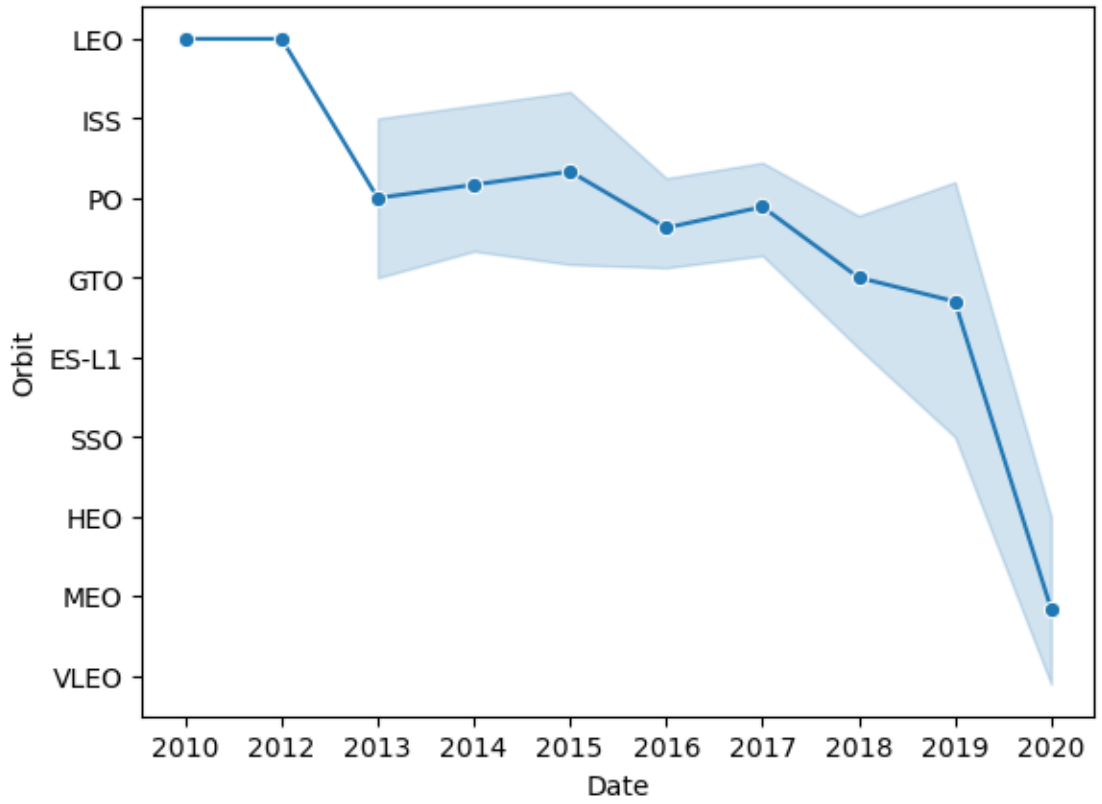
[ ]:
```
# Plot a line chart with x axis to be the extracted year and y axis to be the success rate
```

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

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

[31]:
```
## Features Engineering
sns.lineplot(data=df, x='Date', y='Orbit', marker='o')
```

[31]:
```
<AxesSubplot:xlabel='Date', ylabel='Orbit'>
```



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

[32]:
```
features = df[['FlightNumber', 'PayloadMass', 'Orbit', 'LaunchSite', 'Flights', 'GridFins', 'Reused', 'Legs', 'LandingPad', 'Block', 'ReusedCount', 'Serial']]
features.head()
```

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

[32]:
```
features = df[['FlightNumber', 'PayloadMass', 'Orbit', 'LaunchSite', 'Flights', 'GridFins', 'Reused', 'Legs', 'LandingPad', 'Block', 'ReusedCount', 'Serial']]
features.head()
```

[32]:

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

[33]:
```
### TASK  7: Create dummy variables to categorical columns
features_one_hot = pd.get_dummies(features, columns=['Orbit', 'LaunchSite', 'LandingPad', 'Serial'])
features_one_hot.head()
```

[33]:

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	...	Serial_B1048	Serial_B1049	Serial_B1050	Serial_B1051	Serial_B1054	Serial_B1056	Seri
0	1	6104.959412	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	
1	2	525.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	
2	3	677.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	
3	4	500.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	
4	5	3170.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	

5 rows × 80 columns

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

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

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

35]:
```
### TASK  8: Cast all numeric columns to `float64`
numeric_columns = features_one_hot.select_dtypes(include='number').columns.tolist()
features_one_hot[numeric_columns] = features_one_hot[numeric_columns].astype('float64')
features_one_hot
```

35]:

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	...	Serial_B1048	Serial_B1049	Serial_B1050	Serial_B1051	Serial_B1054	Serial_B1056	Se
0	1.0	6104.959412	1.0	False	False	False	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	
1	2.0	525.000000	1.0	False	False	False	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	
2	3.0	677.000000	1.0	False	False	False	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	
3	4.0	500.000000	1.0	False	False	False	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	
4	5.0	3170.000000	1.0	False	False	False	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
85	86.0	15400.000000	2.0	True	True	True	5.0	2.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	
86	87.0	15400.000000	3.0	True	True	True	5.0	2.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	
87	88.0	15400.000000	6.0	True	True	True	5.0	5.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0.0	0.0	
88	89.0	15400.000000	3.0	True	True	True	5.0	2.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	
89	90.0	3681.000000	1.0	True	False	True	5.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	

90 rows × 80 columns

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

[ ]:
```
# HINT: use astype function
```

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.

38]:
```
features_one_hot.to_csv('dataset_part\gg_3.csv', index=False)
```

[ ]:
```
# HINT: use astype function
```

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.

[38]:
```
features_one_hot.to_csv('dataset_part\gg_3.csv', index=False)
```

Authors

+ 1 cell hidden

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2022-11-09	1.0	Pratiksha Verma	Converted initial version to Jupyterlite

¶

CHAPTER 6
WEEK 3

ANALYSIS WITH FOLIUM

(dark reader extension stopped working here because of the unique cells in jupyterlite)

USING THE LABS ENVIRONMENT

Launch Sites Locations Analysis with Folium

Estimated time needed: **40** minutes

The launch success rate may depend on many factors such as payload mass, orbit type, and so on. It may also depend on the location and proximities of a launch site, i.e., the initial position of rocket trajectories. 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.

In the previous exploratory data analysis labs, you have visualized the SpaceX launch dataset using `matplotlib` and `seaborn` and discovered some preliminary correlations between the launch site and success rates. In this lab, you will be performing more interactive visual analytics using `Folium`.

Objectives

This lab contains the following tasks:

- **TASK 1:** Mark all launch sites on a map
- **TASK 2:** Mark the success/failed launches for each site on the map
- **TASK 3:** Calculate the distances between a launch site to its proximities

After completed the above tasks, you should be able to find some geographical patterns about launch sites.

Let's first import required Python packages for this lab:

1]:
```
import piplite
await piplite.install(['folium'])
await piplite.install(['pandas'])
```

2]:
```
import folium
import pandas as pd
```

3]:
```
# Import folium MarkerCluster plugin
from folium.plugins import MarkerCluster
```

[3]:
```
# Import folium MarkerCluster plugin
from folium.plugins import MarkerCluster
# Import folium MousePosition plugin
from folium.plugins import MousePosition
# Import folium DivIcon plugin
from folium.features import DivIcon
```

If you need to refresh your memory about folium, you may download and refer to this previous folium lab:

Generating Maps with Python

[4]:
```
## Task 1: Mark all launch sites on a map
```

First, let's try to add each site's location on a map using site's latitude and longitude coordinates

The following dataset with the name `spacex_launch_geo.csv` is an augmented dataset with latitude and longitude added for each site.

```
# Download and read the `spacex_launch_geo.csv` ...
```

Now, you can take a look at what are the coordinates for each site.

[6]:
```
# Select relevant sub-columns: `Launch Site`, `Lat(Latitude)`, `Long(Longitude)`, `class`
spacex_df = spacex_df[['Launch Site', 'Lat', 'Long', 'class']]
launch_sites_df = spacex_df.groupby(['Launch Site'], as_index=False).first()
launch_sites_df = launch_sites_df[['Launch Site', 'Lat', 'Long']]
launch_sites_df
```

[6]:

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

Above coordinates are just plain numbers that can not give you any intuitive insights about where are those launch sites. If you are very good at geography, you can interpret those numbers directly in your mind. If not, that's fine too. Let's visualize those locations by pinning them on a map.

Above coordinates are just plain numbers that can not give you any intuitive insights about where are those launch sites. If you are very good at geography, you can interpret those numbers directly in your mind. If not, that's fine too. Let's visualize those locations by pinning them on a map.

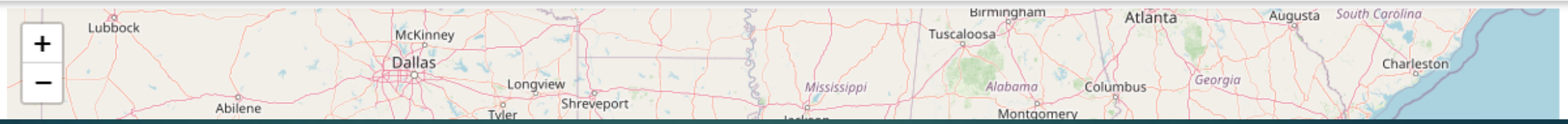
We first need to create a folium `Map` object, with an initial center location to be NASA Johnson Space Center at Houston, Texas.

[7]:
```
# Start location is NASA Johnson Space Center
nasa_coordinate = [29.559684888503615, -95.0830971930759]
site_map = folium.Map(location=nasa_coordinate, zoom_start=10)
```

We could use `folium.Circle` to add a highlighted circle area with a text label on a specific coordinate. For example,

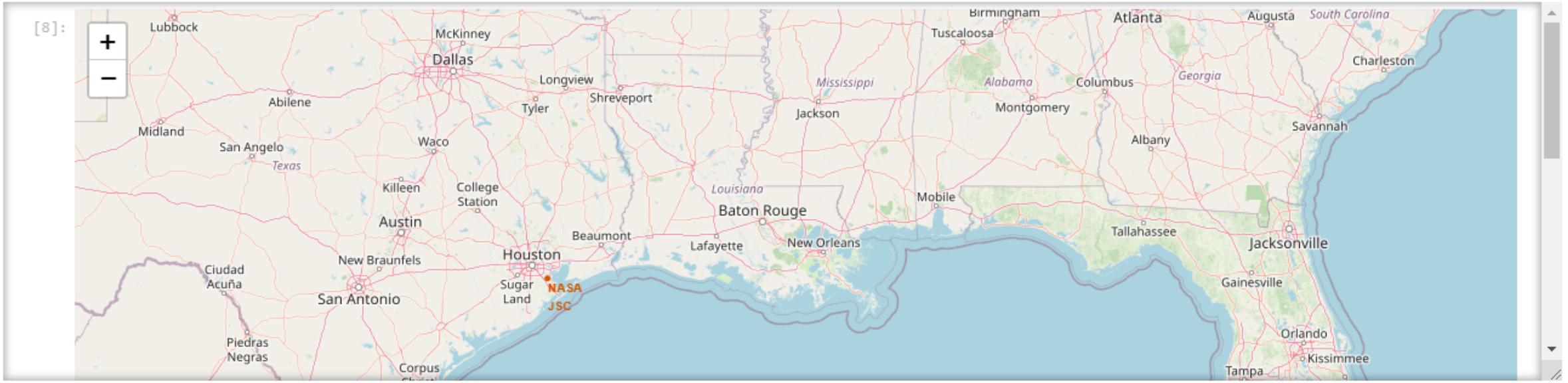
and you should find a small yellow circle near the city of Houston and you can zoom-in to see a larger circle.

[8]:
```
# 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)
```

[8]:


```
site_map.add_child(marker)
```

[8]:



Now, let's add a circle for each launch site in data frame `launch_sites`

TODO: Create and add `folium.Circle` and `folium.Marker` for each launch site on the site map

An example of folium.Circle:

`folium.Circle(coordinate, radius=1000, color='#000000', fill=True).add_child(folium.Popup(...))`

An example of folium.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>' % 'label', ))`

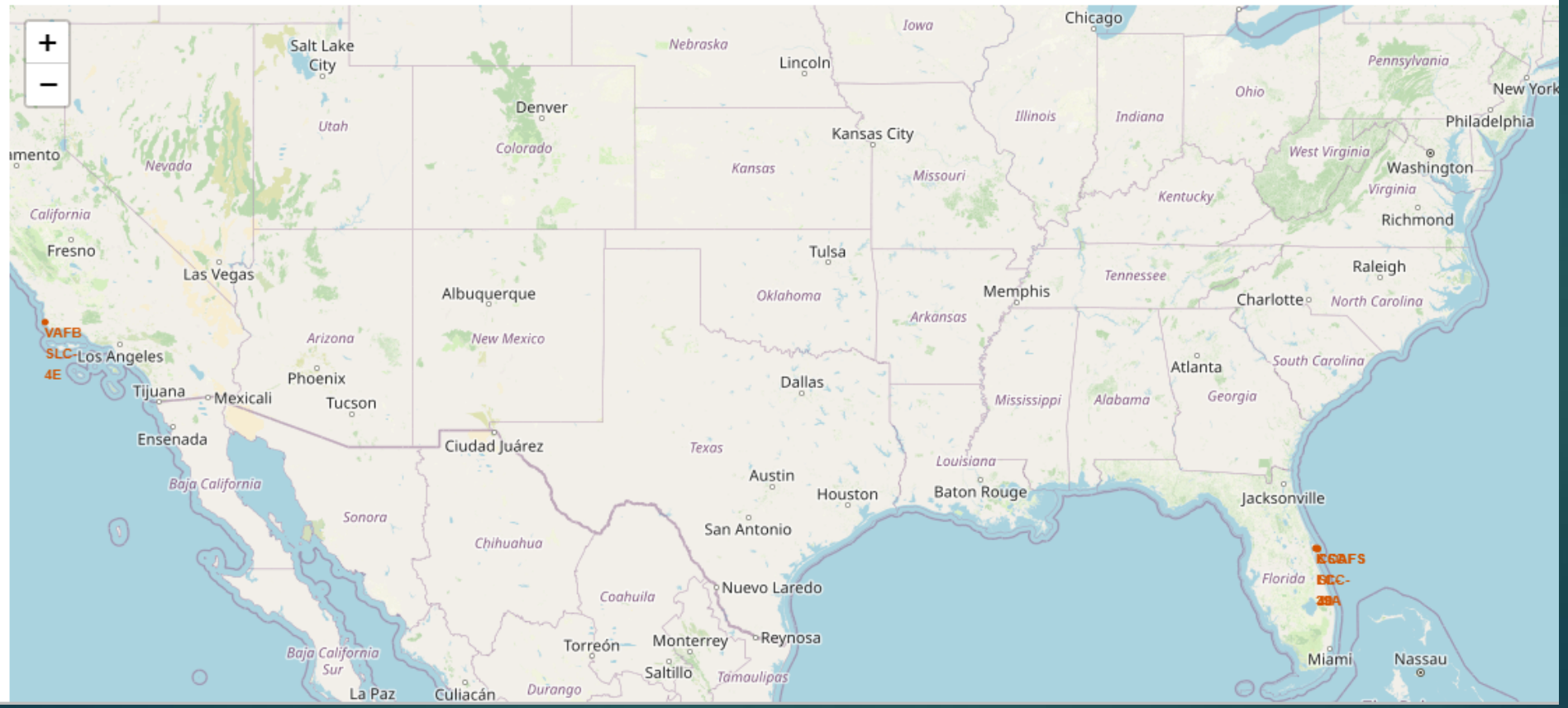
[9]:

```
# 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
# instantiate a feature group for the incidents in the dataframe


# loop through the 100 crimes and add each to the incidents feature group
```

```
# loop through the 100 crimes and add each to the incidents feature group
for index, row in launch_sites_df.iterrows():
    myCordinate = [row['Lat'],row['Long']]
    marker = folium.map.Marker(
    myCordinate,
    # 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>' % row['Launch Site'],
        )
    )
    circle = folium.Circle(myCordinate, radius=10, color='#d35400', fill=True).add_child(folium.Popup(row['Launch Site']))
    site_map.add_child(marker)
    site_map.add_child(circle)
site_map
```

[9]:



Now, you can explore the map by zoom-in/out the marked areas , and try to answer the following questions:

- Are all launch sites in proximity to the Equator line?
- Are all launch sites in very close proximity to the coast?

Also please try to explain your findings.

[10]:
```
# Task 2: Mark the success/failed launches for each site on the map
```

Next, let's try to enhance the map by adding the launch outcomes for each site, and see which sites have high success rates. Recall that data frame spacex_df has detailed launch records, and the `class` column indicates if this launch was successful or not

[11]:
```
spacex_df.tail(10)
```

[11]:

	Launch Site	Lat	Long	class
46	KSC LC-39A	28.573255	-80.646895	1
47	KSC LC-39A	28.573255	-80.646895	1
48	KSC LC-39A	28.573255	-80.646895	1
49	CCAFS SLC-40	28.563197	-80.576820	1
50	CCAFS SLC-40	28.563197	-80.576820	1
51	CCAFS SLC-40	28.563197	-80.576820	0
52	CCAFS SLC-40	28.563197	-80.576820	0
53	CCAFS SLC-40	28.563197	-80.576820	0
54	CCAFS SLC-40	28.563197	-80.576820	1
55	CCAFS SLC-40	28.563197	-80.576820	0

Next, let's create markers for all launch records. If a launch was successful `(class=1)`, then we use a green marker and if a launch was failed, we use a red marker `(class=0)`

Note that a launch only happens in one of the four launch sites, which means many launch records will have the exact same coordinate. Marker clusters can be a good way to simplify a map containing many markers having the same coordinate.

Let's first create a `MarkerCluster` object

Note that a launch only happens in one of the four launch sites, which means many launch records will have the exact same coordinate. Marker clusters can be a good way to simplify a map containing many markers having the same coordinate.

Let's first create a `MarkerCluster` object

[12]:
```
marker_cluster = MarkerCluster()
```

TODO: Create a new column in `launch_sites` dataframe called `marker_color` to store the marker colors based on the `class` value

[13]:
```
def check_color(row):
    if row['class']==0:
        return 'red'
    elif row['class']==1:
        return 'green'
# Apply a function to check the value of `class` column
# If class=1, marker_color value will be green
# If class=0, marker_color value will be red
```

TODO: For each launch result in `spacex_df` data frame, add a `folium.Marker` to `marker_cluster`

[14]:
```
# Add marker_cluster to current site_map
site_map.add_child(marker_cluster)

# for each row in spacex_df data frame
# create a Marker object with its coordinate
# and customize the Marker's icon property to indicate if this launch was successed or failed,
# e.g., icon=folium.Icon(color='white', icon_color=row['marker_color']
for index, record in spacex_df.iterrows():
    myCordinate = [record['Lat'],record['Long']]
    marker = folium.map.Marker(
    myCordinate,
    # Create an icon as a text label
    icon=folium.Icon(color='white', icon_color=check_color(record)))
    #circle = folium.Circle(myCordinate, radius=1000, color='#d35400', fill=True).add_child(folium.Popup(row['Launch Site']))
    site_map.add_child(marker)
    #site_map.add_child(circle)
    marker_cluster.add_child(marker)
```

```
    # Create an icon as a text label
    icon=folium.Icon(color='white', icon_color=check_color(record)))
    #circle = folium.Circle(myCordinate, radius=1000, color='#d35400', fill=True).add_child(folium.Popup(row['Launch Site']))
    site_map.add_child(marker)
    #site_map.add_child(circle)
    marker_cluster.add_child(marker)



    # TODO: Create and add a Marker cluster to the site map
    # marker = folium.Marker(...)
    # marker_cluster.add_child(marker)

site_map
```

[14]:

Your updated map may look like the following screenshots:

From the color-labeled markers in marker clusters, you should be able to easily identify which launch sites have relatively high success rates.

[15]:
```
# TASK 3: Calculate the distances between a launch site to its proximities
```

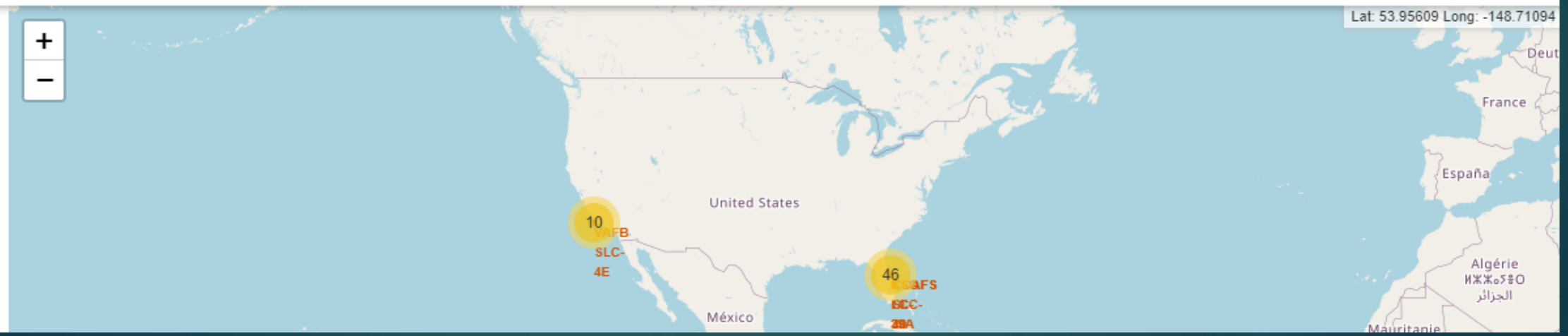
Next, we need to explore and analyze the proximities of launch sites.

Let's first add a `MousePosition` on the map to get coordinate for a mouse over a point on the map. As such, while you are exploring the map, you can easily find the coordinates of any point of interests (such as railway)

[16]:
```
# Add Mouse Position to get the coordinate (Lat, Long) for a mouse over on the map
formatter = "function(num) {return L.Util.formatNum(num, 5);};"
mouse_position = MousePosition(
    position='topright',
    separator=' Long: ',
    empty_string='NaN',
    lng_first=False,
    num_digits=20,
    prefix='Lat:',
    lat_formatter=formatter,
    lng_formatter=formatter,
)

site_map.add_child(mouse_position)
site_map
```

[16]:



Now zoom in to a launch site and explore its proximity to see if you can easily find any railway, highway, coastline, etc. Move your mouse to these points and mark down their coordinates (shown on the top-left) in order to the distance to the launch site.

Now zoom in to a launch site and explore its proximity to see if you can easily find any railway, highway, coastline, etc. Move your mouse to these points and mark down their coordinates (shown on the top-left) in order to the distance to the launch site.

[17]:
```
from math import sin, cos, sqrt, atan2, radians

def calculate_distance(lat1, lon1, lat2, lon2):
    # approximate radius of earth in km
    R = 6373.0

    lat1 = radians(lat1)
    lon1 = radians(lon1)
    lat2 = radians(lat2)
    lon2 = radians(lon2)

    dlon = lon2 - lon1
    dlat = lat2 - lat1

    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))

    distance = R * c
    return distance
```

TODO: Mark down a point on the closest coastline using MousePosition and calculate the distance between the coastline point and the launch site.

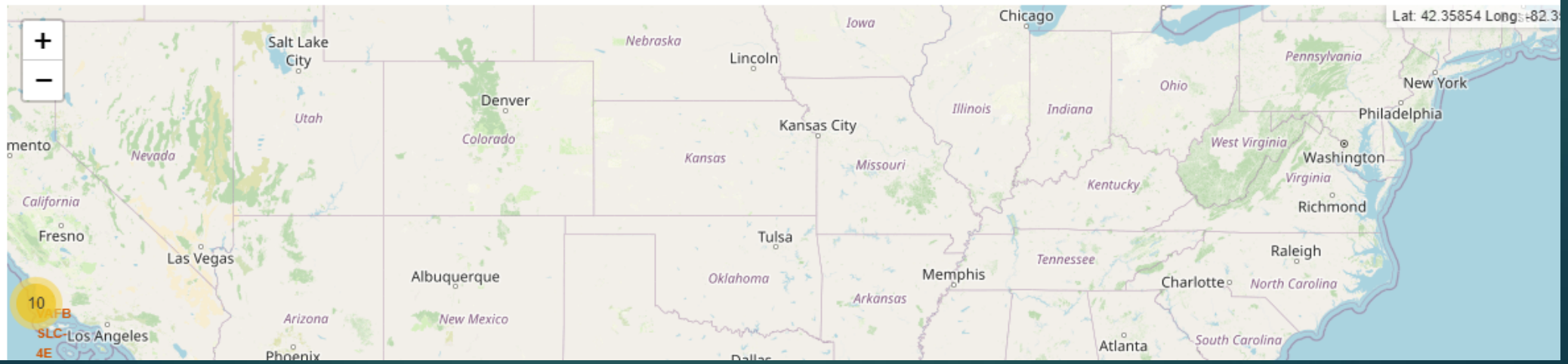
[18]:
```
# find coordinate of the closet coastline
# e.g.,: Lat: 28.56367  Lon: -80.57163
# launch site : 28.56199 ,-80.57716
# coastline : 28.56306 ,-80.56785
launch_site_lat = 28.56199
launch_site_lon = -80.57716
coastline_lat = 28.56306
coastline_lon = -80.56785
distance_coastline = calculate_distance(launch_site_lat, launch_site_lon, coastline_lat, coastline_lon)
distance_coastline
```

[18]:
```
0.9172729928147323
```

[18]:
```
# find coordinate of the closet coastline
# e.g.,: Lat: 28.56367  Lon: -80.57163
# launch site : 28.56199 ,-80.57716
# coastline : 28.56306 ,-80.56785
launch_site_lat = 28.56199
launch_site_lon = -80.57716
coastline_lat = 28.56306
coastline_lon = -80.56785
distance_coastline = calculate_distance(launch_site_lat, launch_site_lon, coastline_lat, coastline_lon)
distance_coastline
```

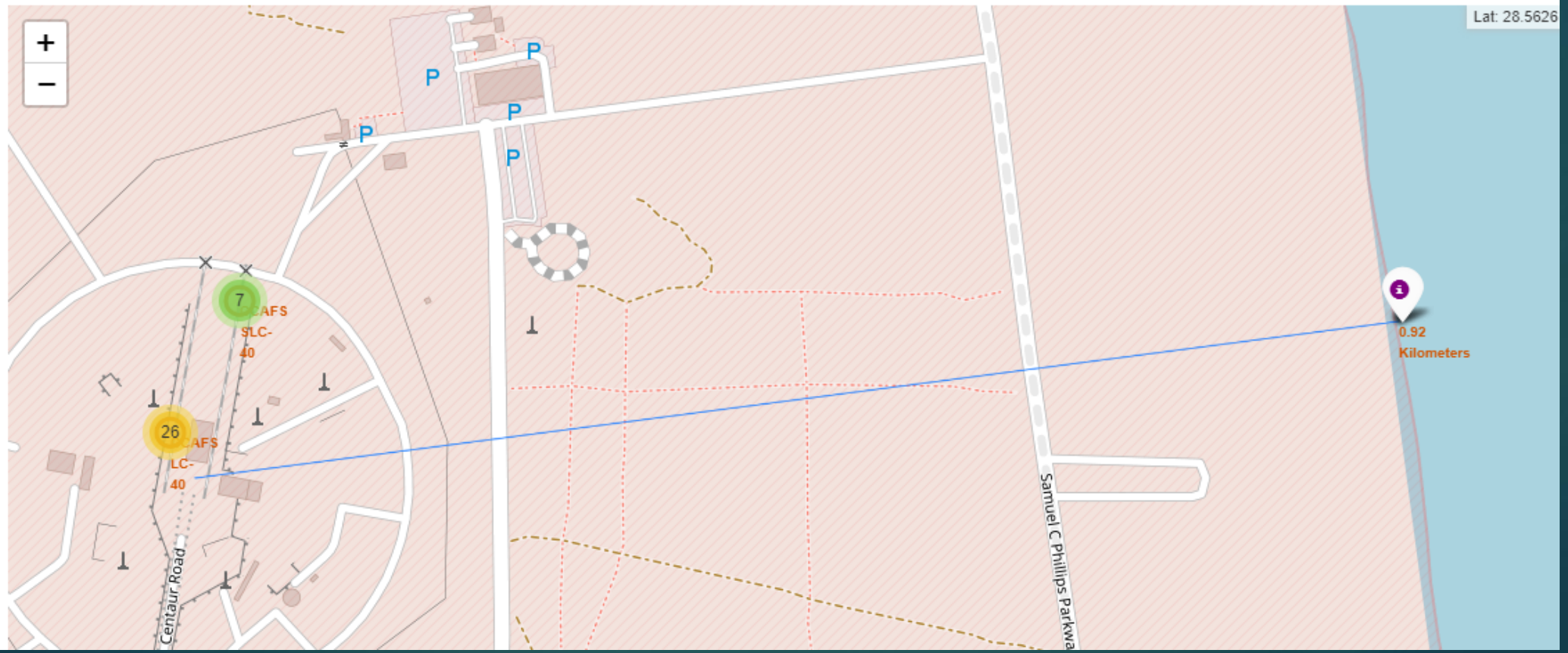
[18]:
```
0.9172729928147323
```

[22]:
```
# Create and add a folium.Marker on your selected closest coastline point on the map
# Display the distance between coastline point and launch site using the icon property
# for example
coordinate_cl = [28.56306,-80.56785]
marker = folium.map.Marker(
    coordinate_cl,
    # 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>' %"{:10.2f} Kilometers".format(distance_coastline),
        )
)
site_map.add_child(marker)
site_map
```

[22]:


[22]:
```
# Create and add a folium.Marker on your selected closest coastline point on the map
# Display the distance between coastline point and launch site using the icon property
# for example
coordinate_cl = [28.56306,-80.56785]
marker = folium.map.Marker(
    coordinate_cl,
    # 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>' %"{:10.2f} Kilometers".format(distance_coastline),
        )
)
site_map.add_child(marker)
site_map
```

[22]:



TODO: Similarly, you can draw a line betwee a launch site to its closest city, railway, highway, etc. You need to use `MousePosition` to find the their coordinates on the map first

A railway map symbol may look like this:

A highway map symbol may look like this:

A city map symbol may look like this:

[21]:
```
# Create a marker with distance to a closest city, railway, highway, etc.
# Draw a line between the marker to the launch site
```

[ ]:

- Are launch sites in close proximity to railways?
- Are launch sites in close proximity to highways?
- Are launch sites in close proximity to coastline?
- Do launch sites keep certain distance away from cities?

Also please try to explain your findings.

Next Steps:

Now you have discovered many interesting insights related to the launch sites' location using folium, in a very interactive way. Next, you will need to build a dashboard using Ploty Dash on detailed launch records.

Authors

Pratiksha Verma

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2022-11-09	1.0	Pratiksha Verma	Converted initial version to Jupyterlite

CHAPTER 7
WEEK 3

BUILD A DASHBOARD APPLICATION WITH PLOTLY DASH

USING THE LABS ENVIRONMENT

Table of Contents

you may refer to the lab you have learned before:

Plotly Dash Lab

TASK 1: Add a Launch Site Drop-down Input Component

We have four different launch sites and we would like to first see which one has the largest success count. Then,
we would like to select one specific site and check its detailed success rate (class=0 vs. class=1).

As such, we will need a dropdown menu to let us select different launch sites.

- Find and complete a commented `dcc.Dropdown(id='site-dropdown',...)` input with following attributes:
  - `id` attribute with value `site-dropdown`
  - `options` attribute is a list of dict-like option objects (with `label` and `value` attributes). You can set the `label` and `value` all to be the launch site names in the `spacex_df`
    and you need to include the default `All` option. e.g.,

```
options=[{'label': 'All Sites', 'value'
```

  - `value` attribute with default dropdown value to be `ALL` meaning all sites are selected
  - `placeholder` attribute to show a text description about this input area,
    such as `Select a Launch Site here`
  - `searchable` attribute to be True so we can enter keywords to search launch sites

File Edit Selection View Go Run Terminal Help

EXPLORER

OPEN EDITORS

PROJECT

.theia

spacex_dash_...

spacex_launch...

Welcome

spacex_dash_app.py

spacex_dash_app.py

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

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

# Create an app layout
app.layout = html.Div(children=[html.H1("Louka's SpaceX Launch Records Dashboard",
                                        style={'textAlign': 'center', 'color': '#503D36',
                                               'font-size': 40}),

                                # TASK 1: Add a dropdown list to enable Launch Site selection
                                # The default select value is for ALL sites
                                dcc.Dropdown(id='site-dropdown',
                                    options=[
                                        {'label': 'All Sites', 'value': 'ALL'},
                                        {'label': 'CCAFS LC-40', 'value': 'CCAFS LC-40'},
                                        {'label': 'VAFB SLC-4E', 'value': 'VAFB SLC-4E'},
                                        {'label': 'KSC LC-39A', 'value': 'KSC LC-39A'},
                                        {'label': 'CCAFS SLC-40', 'value': 'CCAFS SLC-40'},
                                    ],
                                    value='ALL',
                                    placeholder="Select a Launch Site here",
                                    searchable=True
                                    ),



                                html.Br(),

                                # TASK 2: Add a pie chart to show the total successful launches count for all s
                                # If a specific launch site was selected, show the Success vs. Failed counts fo
```

theia@theiadocker-loukasanasta: /home/project

theia@theiadocker-loukasanasta: /home/project

TASK 2: Add a callback function to render `success-pie-chart` based on selected site dropdown

The general idea of this callback function is to get the selected launch site from `site-dropdown` and render a pie chart visualizing launch success counts.

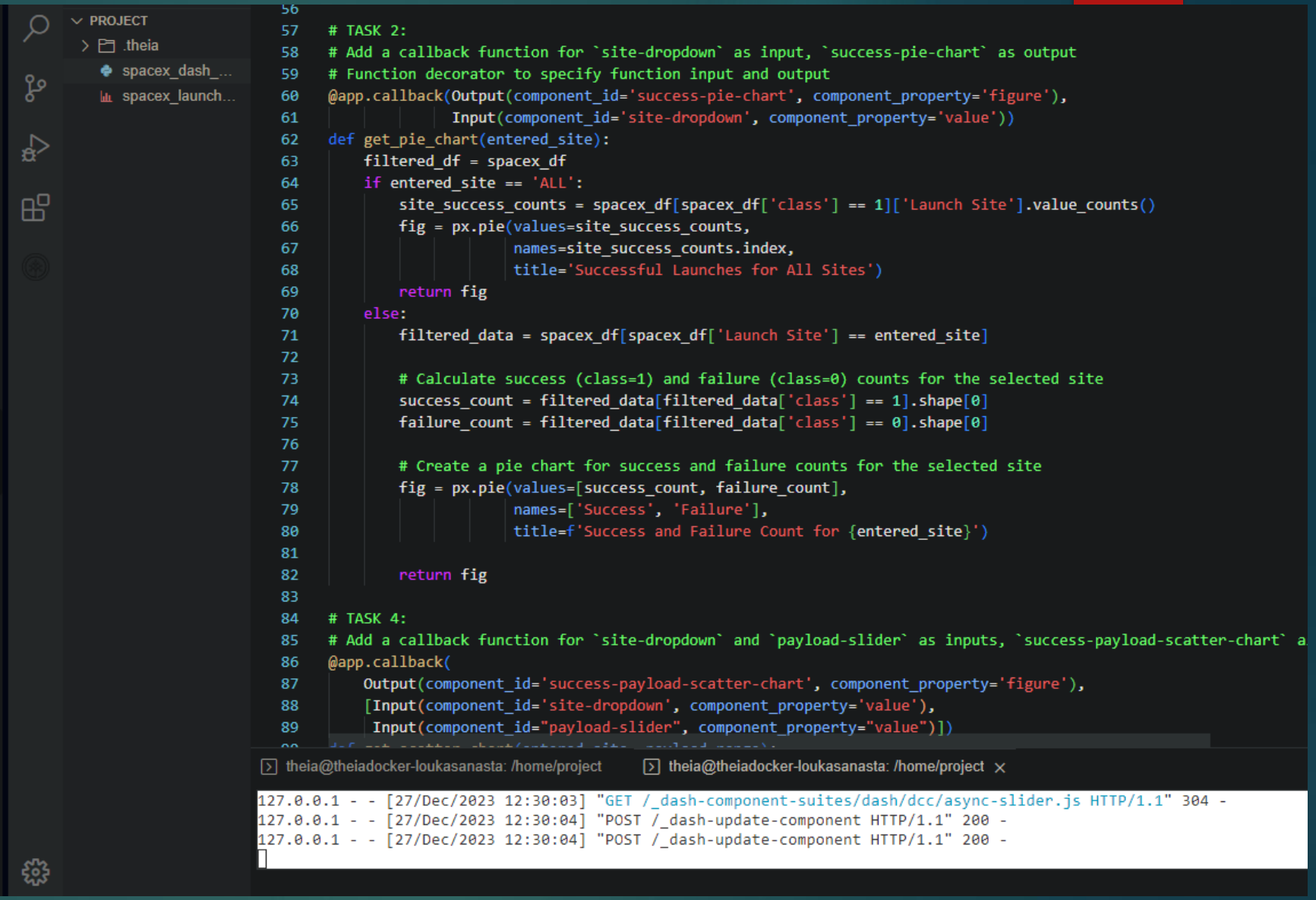
Dash callback function is a type of Python function which will be automatically called by Dash whenever receiving an input component updates, such as a click or dropdown selecting event.

If you need to refresh your memory about Plotly Dash callback functions, you may refer to the lab you have learned before:

Plotly Dash Lab

Let's add a callback function in `spacex_dash_app.py` including the following application logic:

- Input is set to be the `site-dropdown` dropdown, i.e., `Input(component_id='site-dropdown', component_property='value')`
- Output to be the graph with id `success-pie-chart`, i.e., `Output(component_id='success-pie-chart', component_property='figure')`
- A `If-Else` statement to check if ALL sites were selected or just a specific launch site was selected
  - If ALL sites are selected, we will use all rows in the



TASK 3: Add a Range Slider to Select Payload

Next, we want to find if variable payload is correlated to mission outcome. From a dashboard point of view, we want to be able to easily select different payload range and see if we can identify some visual patterns.

Find and complete a commented `dcc.RangeSlider(id='payload-slider',...)` input with the following attribute:

- `id` to be `payload-slider`
- `min` indicating the slider starting point, we set its value to be 0 (Kg)
- `max` indicating the slider ending point to, we set its value to be 10000 (Kg)
- `step` indicating the slider interval on the slider, we set its value to be 1000 (Kg)
- `value` indicating the current selected range, we could set it to be `min_payload` and `max_payload`

Here is an example of `RangeSlider`:

```
dcc.RangeSlider(id='id',
                min=0, max=10000, step=1000,
                marks={0: '0',
                       100: '100'},
                value=[min_value, max_value],
```

You completed payload range slider should be similar the following screenshot:

EXPLORER
OPEN EDITORS
PROJECT
.theia
spacex_dash_...
spacex_launch...

Welcome | spacex_dash_app.py

spacex_dash_app.py

```
                    placeholder="Select a Launch Site here",
                    searchable=True
                    ),



                html.Br(),

                # TASK 2: Add a pie chart to show the total successful launches count for all sites
                # If a specific launch site was selected, show the Success vs. Failed counts for the si
                html.Div(dcc.Graph(id='success-pie-chart')),
                html.Br(),

                html.P("Payload range (Kg):"),
                # TASK 3: Add a slider to select payload range
                dcc.RangeSlider(id='payload-slider',
                    min=0, max=10000, step=1000,
                    marks={0: '0',
                        100: '100'},
                    value=[min_payload, max_payload]),
                # TASK 4: Add a scatter chart to show the correlation between payload and launch succes
                html.Div(dcc.Graph(id='success-payload-scatter-chart'))
                ])


ack function for `site-dropdown` as input, `success-pie-chart` as output
corator to specify function input and output
(Output(component_id='success-pie-chart', component_property='figure'),
 Input(component_id='site-dropdown', component_property='value'))
hart(entered_site):
df = spacex_df
d_site == 'ALL':
success_counts = spacex_df[spacex_df['class'] == 1]['Launch Site'].value_counts()
 px.pie(values=site success counts,
```

theia@theiadocker-loukasanasta: /home/project | theia@theiadocker-loukasanasta: /home/project

```
127.0.0.1 - - [27/Dec/2023 12:30:03] "GET /_dash-component-suites/dash/dcc/async-slider.js HTTP/1.1" 304 -
```

`Plotly Dash Reference` towards the end of this lab.

TASK 4: Add a callback function to render the `success-payload-scatter-chart` scatter plot

Next, we want to plot a scatter plot with the x axis to be the payload and the y axis to be the launch outcome (i.e., `class` column).
As such, we can visually observe how payload may be correlated with mission outcomes for selected site(s).

In addition, we want to color-label the Booster version on each scatter point so that we may
observe mission outcomes with different boosters.

Now, let' s add a call function including the following application logic:

- Input to be `[Input(component_id='site-dropdown', component_property='value'), Input(component_id="payload-slider", component_property="value")]`
  Note that we have two input components, one to receive selected launch site and another to receive selected payload range
- Output to be `Output(component_id='success-payload-scatter-chart', component_property='figure')`
- A `If-Else` statement to check if ALL sites were selected or just a specific launch site was selected
  - If ALL sites are selected, render a scatter plot to display all values for variable `Payload Mass (kg)` and variable `class`.

EXPLO... Welcome spacex_dash_app.py

OPEN EDIT...
PROJECT
.theia
spa...
spa...

spacex_dash_app.py

```
        # Create a pie chart for success and failure counts for the selected site
        fig = px.pie(values=[success_count, failure_count],
                     names=['Success', 'Failure'],
                     title=f'Success and Failure Count for {entered_site}')

        return fig

# TASK 4:
# Add a callback function for `site-dropdown` and `payload-slider` as inputs, `success-payload-scatter-chart` as output
@app.callback(
    Output(component_id='success-payload-scatter-chart', component_property='figure'),
    [Input(component_id='site-dropdown', component_property='value'),
     Input(component_id="payload-slider", component_property="value")])
def get_scatter_chart(entered_site, payload_range):
    if entered_site == 'ALL':
        scatter_data = spacex_df[spacex_df['Payload Mass (kg)'].between(payload_range[0], payload_range[1])]
        fig = px.scatter(scatter_data, x='Payload Mass (kg)', y='class', color='Booster Version Category',
                         title='Payload Success Scatter Chart for All Sites')
        return fig
    else:
        filtered_data = spacex_df[spacex_df['Launch Site'] == entered_site]
        filtered_data = filtered_data[filtered_data['Payload Mass (kg)'].between(payload_range[0], payload_range[1])]

        fig = px.scatter(filtered_data, x='Payload Mass (kg)', y='class', color='Booster Version Category',
                         title=f'Payload Success Scatter Chart for {entered_site}')
        return fig

# Run the app
if __name__ == '__main__':
    app.run_server()
```

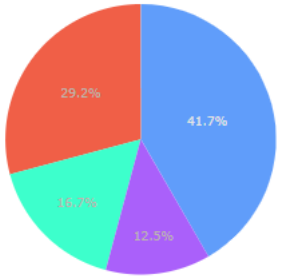
theia@theiadocker-loukasanasta: /home/project theia@theiadocker-loukasanasta: /home/project

```
127.0.0.1 - - [27/Dec/2023 12:30:03] "GET /_dash-component-suites/dash/dcc/async-slider.js HTTP/1.1" 304 -
127.0.0.1 - - [27/Dec/2023 12:30:04] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [27/Dec/2023 12:30:04] "POST /_dash-update-component HTTP/1.1" 200 -
```

Louka's SpaceX Launch Records Dashboard

All Sites

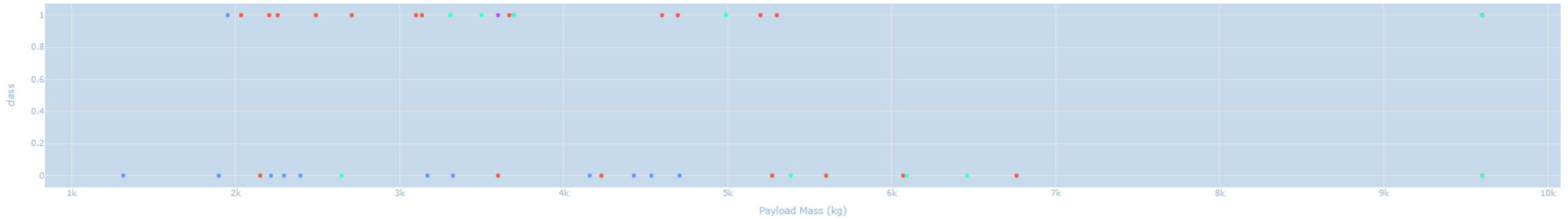
Successful Launches for All Sites

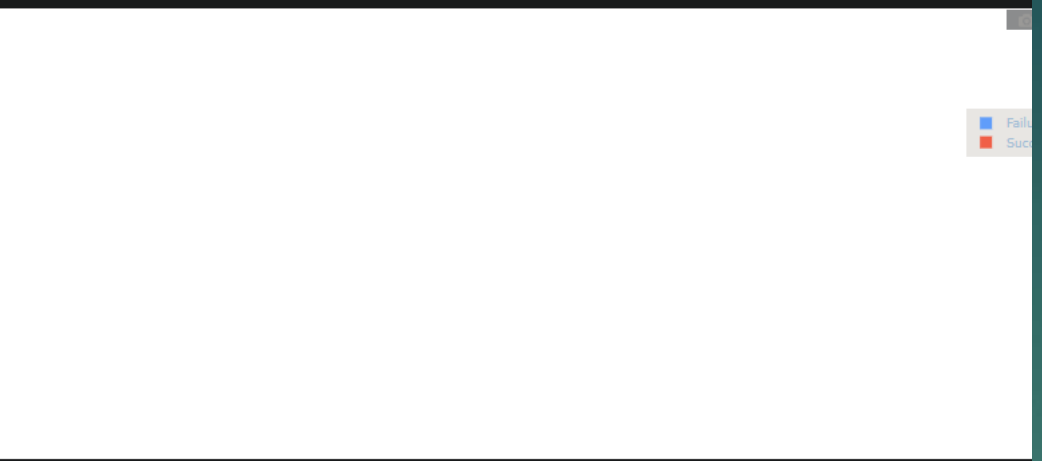


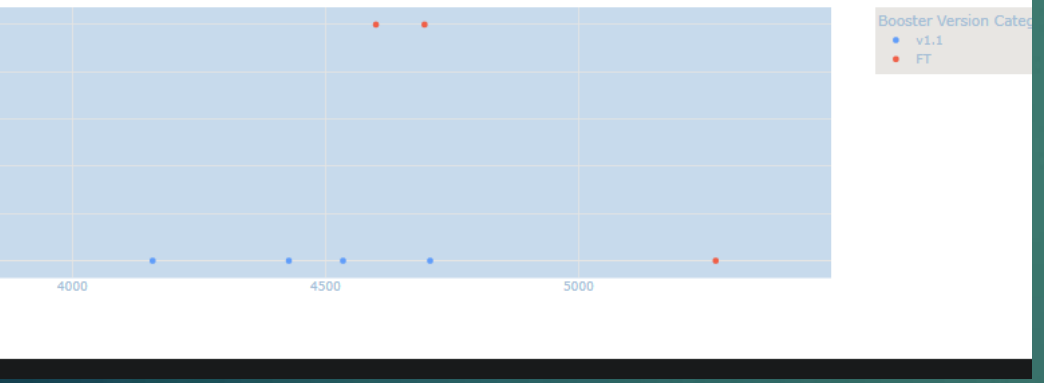
Payload range (Kg):

0 100

Payload Success Scatter Chart for All Sites

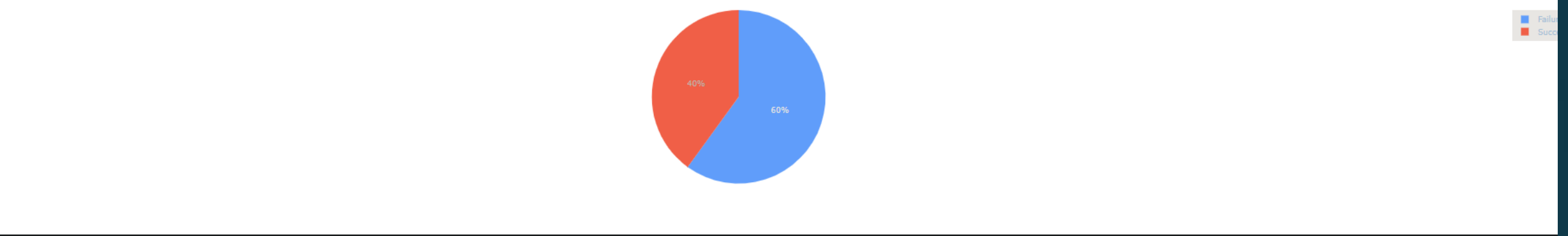



Fail
Succ


Booster Version Categ
v1.1
FT
4000
4500
5000

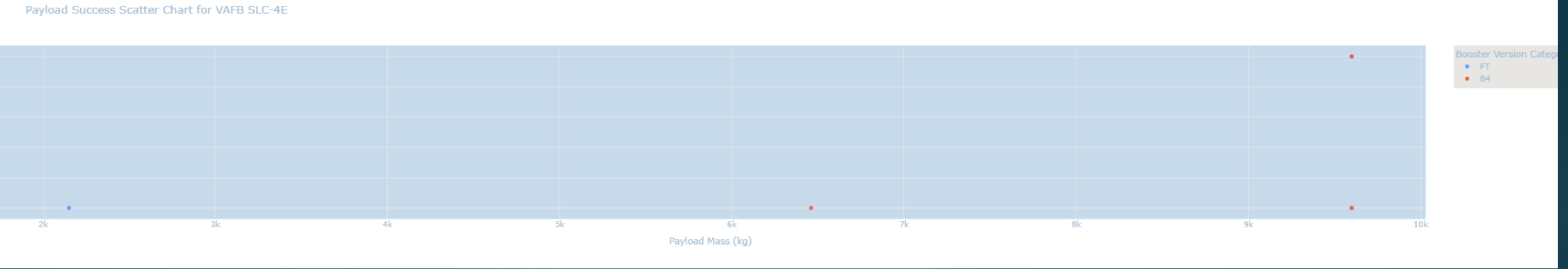
Louka's SpaceX Launch Records Dashboard

VAFB SLC-4E



Payload range (Kg):

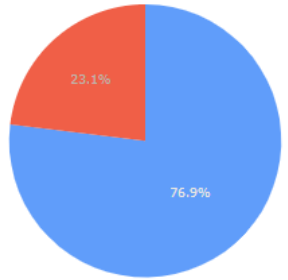
0 100



Louka's SpaceX Launch Records Dashboard

KSC LC-39A

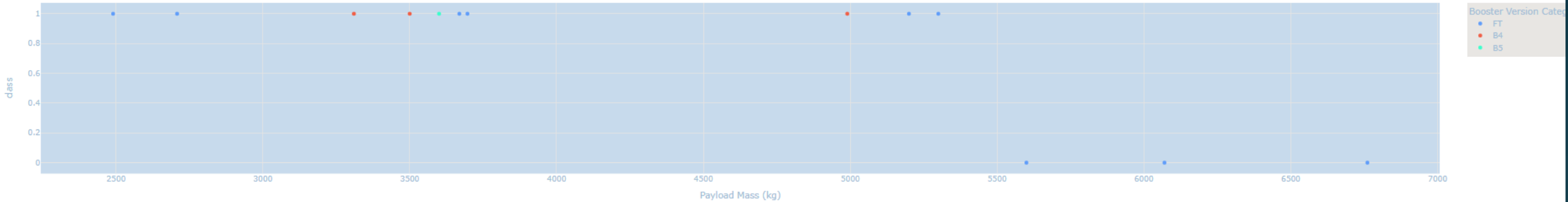
Success and Failure Count for KSC LC-39A



Payload range (Kg):

0 100

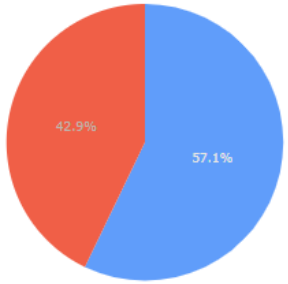
Payload Success Scatter Chart for KSC LC-39A



Louka's SpaceX Launch Records Dashboard

CCAFS SLC-40

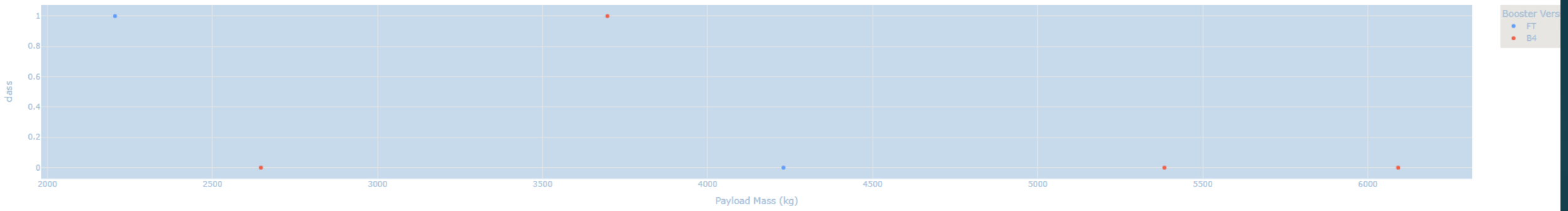
Success and Failure Count for CCAFS SLC-40



ayload range (Kg):

0 100

Payload Success Scatter Chart for CCAFS SLC-40



CHAPTER 8
WEEK 4

MACHINE LEARNING PREDICTION

USING THE LABS ENVIRONMENT

Objectives ¶

Perform exploratory Data Analysis and determine Training Labels

- create a column for the class
- Standardize the data
- Split into training data and test data

-Find best Hyperparameter for SVM, Classification Trees and Logistic Regression

- Find the method performs best using test data

Import Libraries and Define Auxiliary Functions

[1]:
```
import piplite
await piplite.install(['numpy'])
await piplite.install(['pandas'])
await piplite.install(['seaborn'])
```

We will import the following libraries for the lab

[2]:
```
# Pandas is a software library written for the Python programming language for data manipulation and analysis.
import pandas as pd
# NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a MatLab like plotting framework. We will use this in our plotter function to plot data.
import matplotlib.pyplot as plt
#Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics
import seaborn as sns
# Preprocessing allows us to standarsize our data
from sklearn import preprocessing
# Allows us to split our data into training and testing data
from sklearn.model_selection import train_test_split
```

```
from sklearn.model_selection import GridSearchCV
# Logistic Regression classification algorithm
from sklearn.linear_model import LogisticRegression
# Support Vector Machine classification algorithm
from sklearn.svm import SVC
# Decision Tree classification algorithm
from sklearn.tree import DecisionTreeClassifier
# K Nearest Neighbors classification algorithm
from sklearn.neighbors import KNeighborsClassifier
```

This function is to plot the confusion matrix.

[3]:
```
def plot_confusion_matrix(y,y_predict):
    "this function plots the confusion matrix"
    from sklearn.metrics import confusion_matrix

    cm = confusion_matrix(y, y_predict)
    ax= plt.subplot()
    sns.heatmap(cm, annot=True, ax = ax); #annot=True to annotate cells
    ax.set_xlabel('Predicted labels')
    ax.set_ylabel('True labels')
    ax.set_title('Confusion Matrix');
    ax.xaxis.set_ticklabels(['did not land', 'land']); ax.yaxis.set_ticklabels(['did not land', 'landed'])
    plt.show()
```

Load the dataframe

Load the data

[4]:
```
from js import fetch
import io

URL1 = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_2.csv"
resp1 = await fetch(URL1)
text1 = io.BytesIO((await resp1.arrayBuffer()).to_py())
data = pd.read_csv(text1)
```

[5]:
```
data.head()
```

[5]:

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

[5]:
```
data.head()
```

[5]:

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

[6]:
```
URL2 = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_3.csv'
resp2 = await fetch(URL2)
text2 = io.BytesIO((await resp2.arrayBuffer()).to_py())
X = pd.read_csv(text2)
```

[7]:
```
X.head(100)
```

[7]:

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	...	Serial_B1058	Serial_B1059	Serial_B1060	Serial_B1062	GridFins_False	GridFins_True	Reused_False	R
0	1.0	6104.959412	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	
1	2.0	525.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	
2	3.0	677.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	
3	4.0	500.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	
4	5.0	3170.000000	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
85	86.0	15400.000000	2.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	
86	87.0	15400.000000	3.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	1.0	0.0	
87	88.0	15400.000000	6.0	5.0	5.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	
88	89.0	15400.000000	3.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	
89	90.0	3681.000000	1.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0.0	1.0	1.0	

90 rows × 83 columns

TASK 1

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

[9]:
```
Y = pd.Series(data['Class'].to_numpy())
Y
```

[9]:
```
0     0
1     0
2     0
3     0
4     0
     ..
85    1
86    1
87    1
88    1
89    1
Length: 90, dtype: int64
```

TASK 2

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

[10]:
```
# students get this
transform = preprocessing.StandardScaler()
X = transform.fit_transform(X)
```

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

TASK 3

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

`X_train, X_test, Y_train, Y_test`

TASK 3

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

`X_train, X_test, Y_train, Y_test`

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

we can see we only have 18 test samples.

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

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

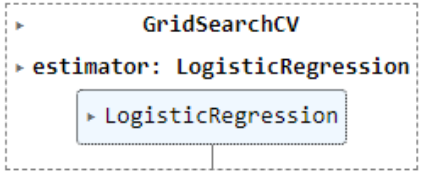
TASK 4

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

[16]:
```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

[18]:
```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()
logreg_cv = GridSearchCV(lr, parameters, cv=10)
logreg_cv.fit(X_train, Y_train)
```

[18]:



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

[19]:
```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

▸ estimator: LogisticRegression

▸ LogisticRegression

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

[19]:
```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

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

TASK 5

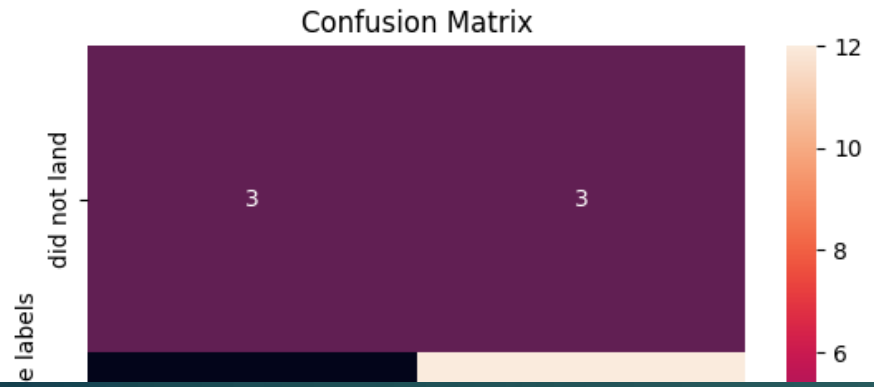
Calculate the accuracy on the test data using the method `score`:

[42]:
```
accuracy_lr = logreg_cv.score(X_test, Y_test)
accuracy_lr
```

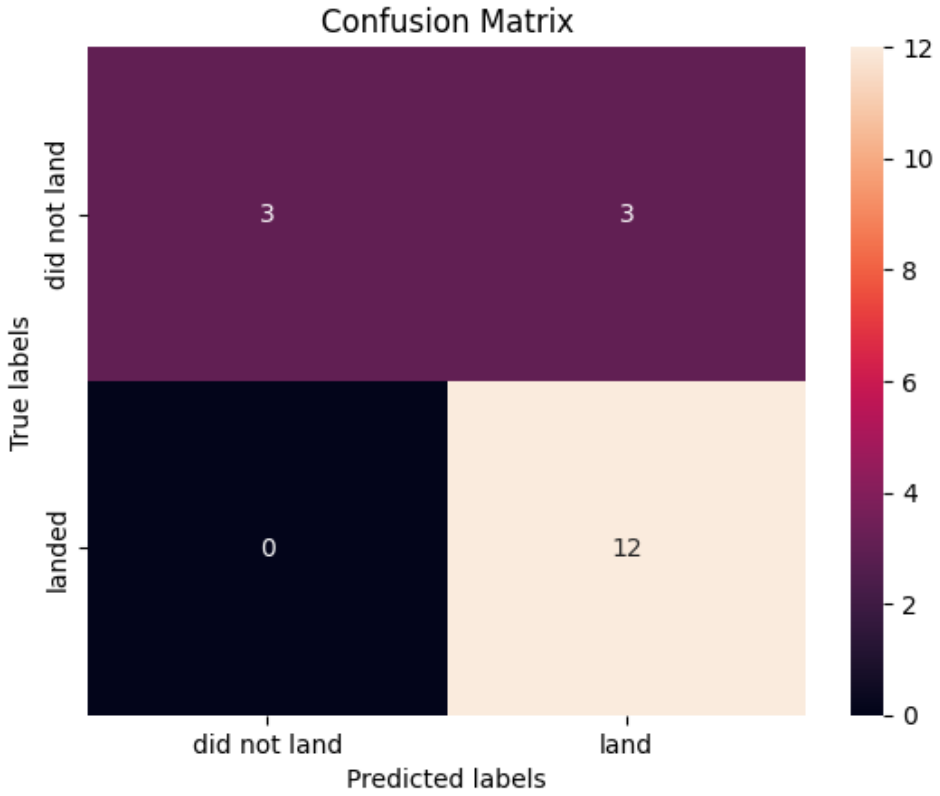
[42]:
```
0.8333333333333334
```

Lets look at the confusion matrix:

[21]:
```
yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



```
plot_confusion_matrix(Y_test,yhat)
```



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

TASK 6

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

[22]:
```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

[23]:
```
svm_cv = GridSearchCV(svm, parameters, cv=10)
svm_cv.fit(X_train, Y_train)
```

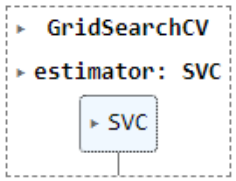
TASK 6

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

[22]:
```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

[23]:
```
svm_cv = GridSearchCV(svm, parameters, cv=10)
svm_cv.fit(X_train, Y_train)
```

[23]:



[24]:
```
print("tuned hpyerparameters :(best parameters) ",svm_cv.best_params_)
print("accuracy :",svm_cv.best_score_)
```

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

TASK 7

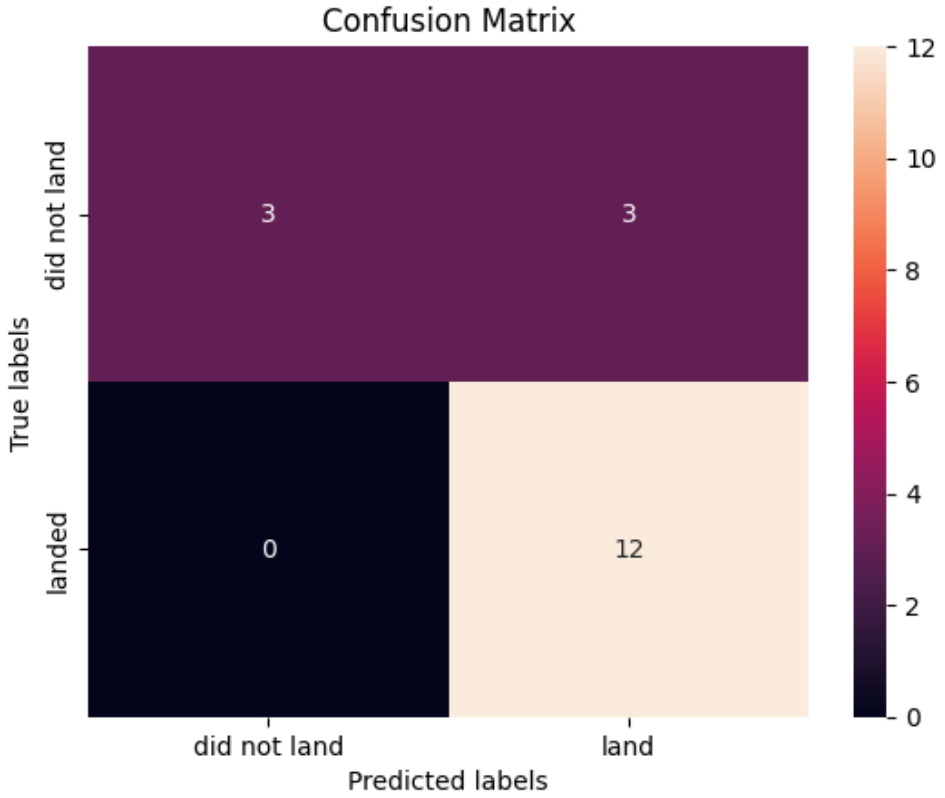
Calculate the accuracy on the test data using the method `score`:

[26]:
```
accuracy_svm = svm_cv.score(X_test, Y_test)
accuracy_svm
```

[26]:
```
0.8333333333333334
```

We can plot the confusion matrix

[27]:
```
yhat=svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



TASK 8

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

[31]:
```
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()
```

[32]:
```
tree_cv = GridSearchCV(tree, parameters, cv=10)
tree_cv.fit(X_train, Y_train)
```

[32]:
```
GridSearchCV
estimator: DecisionTreeClassifier
    DecisionTreeClassifier
```

[33]:
```
print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 6, 'max_features': 'sqrt', 'min_samples_leaf': 4, 'min_samples_split': 10, 'splitter': 'best'}
accuracy : 0.8625
```

TASK 9

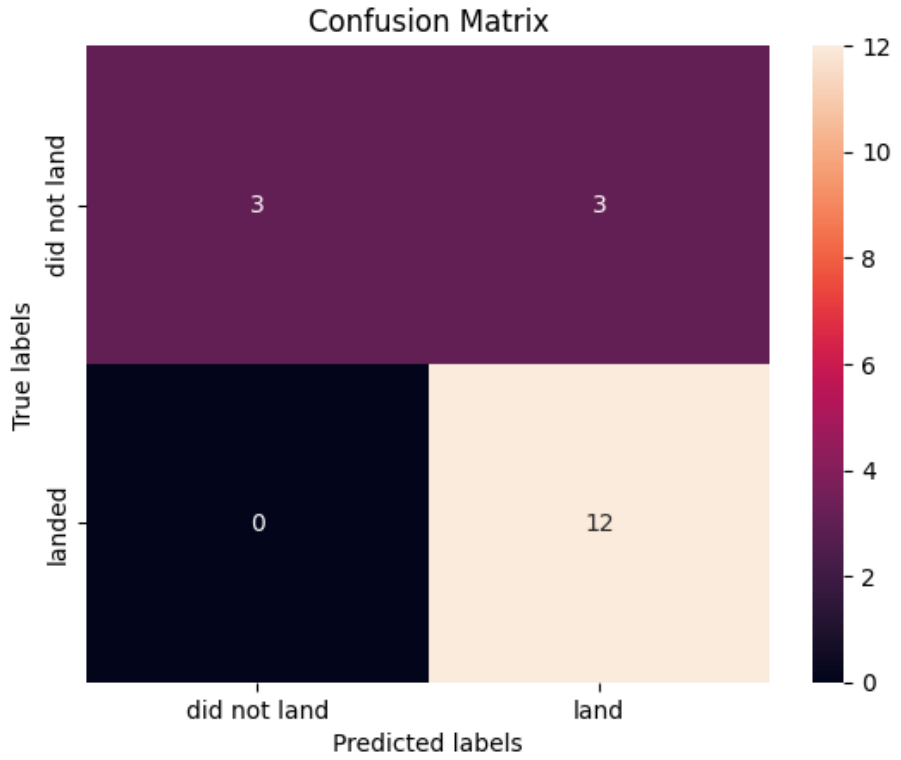
Calculate the accuracy of tree_cv on the test data using the method `score` :

[34]:
```
accuracy_tree = tree_cv.score(X_test, Y_test)
accuracy_tree
```

[34]:
```
0.8333333333333334
```

We can plot the confusion matrix

[35]:
```
yhat = tree_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



TASK 10

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

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

KNN = KNeighborsClassifier()
```

7]:
```
knn_cv = GridSearchCV(KNN, parameters, cv=10)
knn_cv.fit(X_train, Y_train)
```

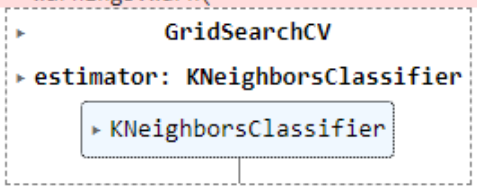
```
/lib/python3.11/site-packages/threadpoolctl.py:1019: RuntimeWarning: libc not found. The ctypes module in Python 3.11 is maybe too old for this OS.
  warnings.warn(
```

7]: ▸ GridSearchCV

```
knn_cv.fit(X_train, Y_train)
```

```
/lib/python3.11/site-packages/threadpoolctl.py:1019: RuntimeWarning: libc not found. The ctypes module in Python 3.11 is maybe too old for this OS.
  warnings.warn(
```

[37]:



[38]:
```
print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
print("accuracy :",knn_cv.best_score_)
```

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

TASK 11

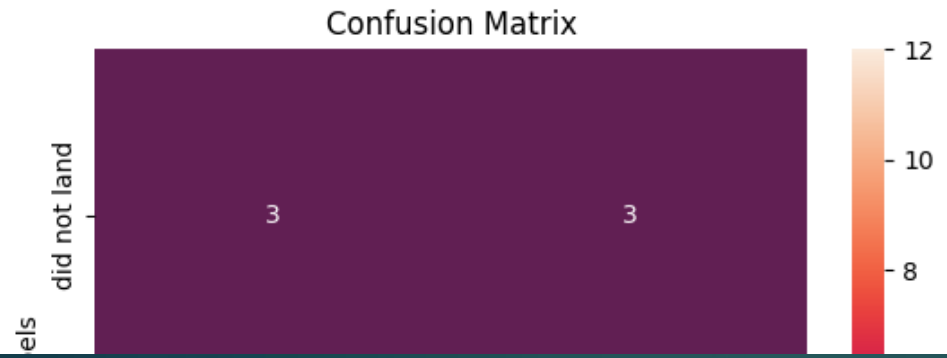
Calculate the accuracy of knn_cv on the test data using the method `score`:

[39]:
```
accuracy_knn = knn_cv.score(X_test, Y_test)
accuracy_knn
```

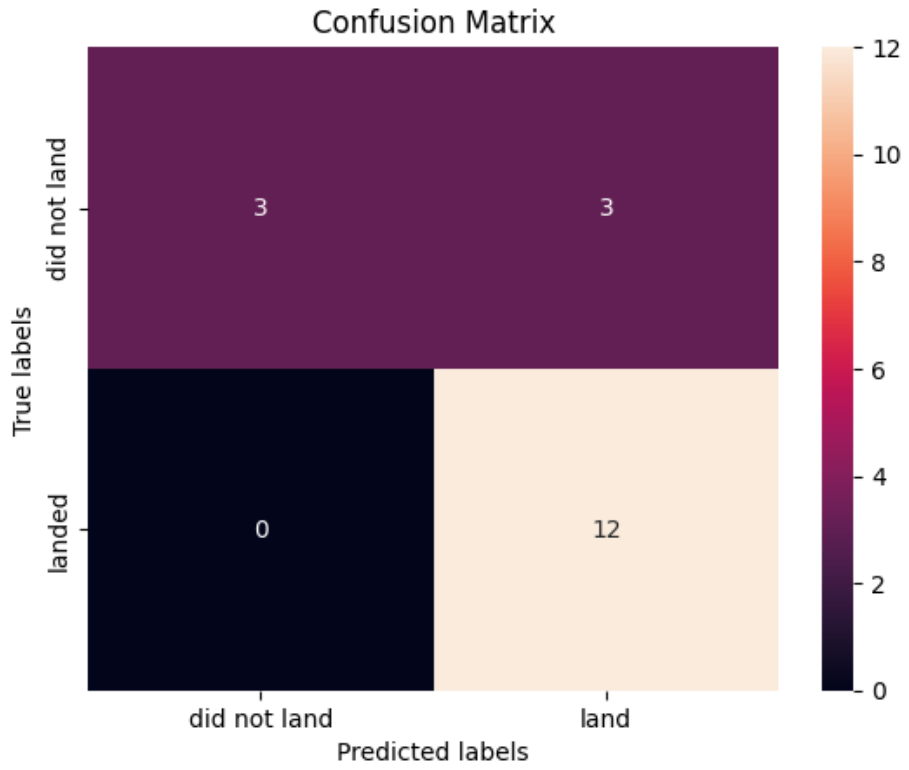
[39]:
```
0.8333333333333334
```

We can plot the confusion matrix

[40]:
```
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



[40]:
```
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



TASK 12

Find the method performs best:

[43]:
```
accuracies = {
    'Logistic Regression': accuracy_lr,
    'Support Vector Machine': accuracy_svm,
    'Decision Tree': accuracy_tree,
    'K Nearest Neighbors': accuracy_knn
}

# Find the method with the highest accuracy
```

Find the method performs best:

[43]:
```
accuracies = {
    'Logistic Regression': accuracy_lr,
    'Support Vector Machine': accuracy_svm,
    'Decision Tree': accuracy_tree,
    'K Nearest Neighbors': accuracy_knn
}

# Find the method with the highest accuracy
best_method = max(accuracies, key=accuracies.get)
best_accuracy = accuracies[best_method]

print("Method with the best accuracy:", best_method)
print("Accuracy:", best_accuracy)
```

```
Method with the best accuracy: Logistic Regression
Accuracy: 0.8333333333333334
```

Authors

Pratiksha Verma

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2022-11-09	1.0	Pratiksha Verma	Converted initial version to Jupyterlite

CONCLUSION

- In conclusion there are a lot of factors that contribute into a successful launch. After cleaning my dataframe and selecting the attributes that matter the most in my investigation i found out the following . The average success rate is 66% . I found out that different launch sites have different success rates. The payload mass is also a factor we cannot overlook. We also tried to examine if there is a relationship between the orbit and the flightnumber. LEO orbit seems to have a relationship while GTO doesn't . Unfortunately the models i used finding the best parameteres via Gridsearch in the machine learning chapter had all the same performance.