(https://skills.network/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork865-2022-01-01)

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.

Several examples of an unsuccessful landing are shown here:

Most unsuccessful landings are planned. Space X performs a controlled landing in the oceans.

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

In [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 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.

In [311]:

```
# 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.

In [312]:

```
# 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.

In [313]:

```
# 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.

In [314]:

```
# 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:

In [315]:

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

In [316]:

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

Check the content of the response

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:

In [318]:
```
1 static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/dat
```

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

In [319]:
```
1 response.status_code
```

Out[319]: 200

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

In [320]:
```
1 # Use json_normalize meethod to convert the json result into a dataframe
2 data = response.json()
3 data = pd.json_normalize(data)
```

Using the dataframe `data` print the first 5 rows

In [321]:

```
# Get the head of the dataframe
data.head(5)
```

Out[321]:

	static_fire_date_utc	static_fire_date_unix	net	window	rocket	success	failures	details	crew	ships	capsules	
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	[]	[]	[]	[5e
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	[]	[]	[]	[5e
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	[]	[]	[]	[5e 5e

	static_fire_date_utc	static_fire_date_unix	net	window	rocket	success	failures	details	crew	ships	capsules	
3	2008-09-20T00:00:00.000Z	1.221869e+09	False	0.0	5e9d0d95eda69955f709d1eb	True	[]	Ratsat was carried to orbit on the first successful orbital launch of any privately funded and developed, liquid-propelled carrier rocket, the SpaceX Falcon 1	[]	[]	[]	[5e
4	None	NaN	False	0.0	5e9d0d95eda69955f709d1eb	True	[]	None	[]	[]	[]	[5e

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`.

In [322]:

```
# 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 t
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 f
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)]
```

- 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.

In [323]:

```
#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:

In [324]:

```
BoosterVersion
```

Out[324]: []

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

In [325]:

```
# Call getBoosterVersion
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'])
```

the list has now been updated

In [326]:

```
getBoosterVersion(data)
print (BoosterVersion)
```

```
['Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon
9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falc
on 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'F
alcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9',
'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon
9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falc
on 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'F
alcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9',
'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon
9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9', 'Falcon 9']
```

we can apply the rest of the functions here:

In [327]:

```
# Call getLaunchSite

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'])
getLaunchSite(data)
print(LaunchSite)
```

```
['Kwajalein Atoll', 'Kwajalein Atoll', 'Kwajalein Atoll', 'Kwajalein Atoll', 'CCSFS SLC 40', 'CCSFS SLC 40', 'CCSFS S
LC 40', 'VAFB SLC 4E', 'CCSFS SLC 40', 'CCSFS SLC 40', 'CCSFS SLC 40', 'CCSFS SLC 40', 'CCSFS SLC 40', 'CCSFS SLC 4
0', 'CCSFS SLC 40', 'CCSFS SLC 40', 'CCSFS SLC 40', 'CCSFS SLC 40', 'CCSFS SLC 40', 'CCSFS SLC 40', 'CCSFS SLC 40',
'VAFB SLC 4E', 'CCSFS SLC 40', 'CCSFS SLC 40', 'CCSFS SLC 40', 'CCSFS SLC 40', 'CCSFS SLC 40', 'CCSFS SLC 40', 'CCSFS
SLC 40', 'VAFB SLC 4E', 'KSC LC 39A', 'KSC LC 39A', 'KSC LC 39A', 'KSC LC 39A', 'KSC LC 39A', 'KSC LC 39A', 'KSC LC 3
9A', 'VAFB SLC 4E', 'KSC LC 39A', 'KSC LC 39A', 'VAFB SLC 4E', 'KSC LC 39A', 'VAFB SLC 4E', 'KSC LC 39A', 'KSC LC 39
A', 'CCSFS SLC 40', 'VAFB SLC 4E', 'CCSFS SLC 40', 'CCSFS SLC 40', 'CCSFS SLC 40', 'VAFB SLC 4E', 'CCSFS SLC 40', 'CC
SFS SLC 40', 'KSC LC 39A', 'CCSFS SLC 40', 'CCSFS SLC 40', 'CCSFS SLC 40', 'VAFB SLC 4E', 'CCSFS SLC 40', 'CCSFS SLC
40', 'VAFB SLC 4E', 'KSC LC 39A', 'VAFB SLC 4E', 'CCSFS SLC 40', 'CCSFS SLC 40', 'VAFB SLC 4E', 'KSC LC 39A', 'CCSFS
SLC 40', 'CCSFS SLC 40', 'VAFB SLC 4E', 'CCSFS SLC 40', 'CCSFS SLC 40', 'CCSFS SLC 40', 'CCSFS SLC 40', 'CCSFS SLC 4
0', 'CCSFS SLC 40', 'KSC LC 39A', 'CCSFS SLC 40', 'CCSFS SLC 40', 'CCSFS SLC 40', 'KSC LC 39A', 'KSC LC 39A', 'KSC LC
39A', 'CCSFS SLC 40', 'CCSFS SLC 40', 'CCSFS SLC 40', 'CCSFS SLC 40', 'CCSFS SLC 40', 'CCSFS SLC 40', 'KSC LC 39A',
'KSC LC 39A', 'KSC LC 39A', 'CCSFS SLC 40', 'CCSFS SLC 40']
```

In [328]:
```
# Call getPayloadData
def getPayloadData(data):
    for load in data['payloads']:
        response = requests.get("https://api.spacexdata.com/v4/payloads/"+load).json()
        PayloadMass.append(response['mass_kg'])
        Orbit.append(response['orbit'])
```

In [329]:
```
getPayloadData(data)
```

In [330]:
```
print(PayloadMass)
```

```
[20, None, 165, 200, None, 525, 677, 500, 3170, 3325, 2296, 1316, 4535, 4428, 2216, 2395, 570, 1898, 4707, 2477, 203
4, 553, 5271, 3136, 4696, 3100, 2257, 4600, 5500, 9600, 2490, 5600, 5300, None, 6070, 2708, 3669, 9600, 6761, 2910, 4
75, 4990, 9600, 5200, 3700, 2205, 9600, None, 4230, 6092, 9600, 2760, 350, 3750, 5383.85, 2410, 7076, 9600, 5800, 706
0, 2800, 3000, 4000, 2573, 4400, 9600, 12259, 2482, 13200, 1425, 2227.7, 6500, 15600, 5000, 6800, 15600, None, 15600,
15600, 1977, 15600, 15600, 9525, 15600, 15600, 3880, None, 15600, 1600, 15600, 15600, 15600, 15600, 3681]
```

In [331]:
```
# Call getCoreData
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'])
```

In [332]:
```
getCoreData(data)
```

In [333]:
```
print(Outcome)
```

```
['None None', 'None None', 'None None', 'None None', 'None None', 'None None', 'None None', 'False Ocean', 'None Non
e', 'None None', 'True Ocean', 'True Ocean', 'None None', 'None None', 'False Ocean', 'False ASDS', 'True Ocean', 'Fa
lse ASDS', 'None None', 'None ASDS', 'True RTLS', 'False ASDS', 'False ASDS', 'True ASDS', 'True ASDS', 'True ASDS',
'True RTLS', 'True ASDS', 'None ASDS', 'True ASDS', 'True RTLS', 'None None', 'True ASDS', 'True RTLS', 'None None',
'True RTLS', 'True ASDS', 'True ASDS', 'None None', 'True RTLS', 'True ASDS', 'True RTLS', 'True ASDS', 'True ASDS',
'True ASDS', 'True RTLS', 'True Ocean', 'True RTLS', 'True Ocean', 'None None', 'None None', 'None None', 'True ASD
S', 'True ASDS', 'None None', 'None None', 'True ASDS', 'True ASDS', 'True ASDS', 'True ASDS', 'True RTLS', 'True ASD
S', 'True ASDS', 'False RTLS', 'None None', 'True ASDS', 'True ASDS', 'True ASDS', 'True ASDS', 'True RTLS', 'True RT
LS', 'None None', 'True ASDS', 'True ASDS', 'True ASDS', 'True ASDS', 'None None', 'True ASDS', 'False ASDS', 'True R
TLS', 'False ASDS', 'True ASDS', 'True ASDS', 'True ASDS', 'True ASDS', 'True ASDS', 'True ASDS', 'True ASDS', 'True
RTLS', 'True ASDS', 'True ASDS', 'True ASDS', 'True ASDS', 'True ASDS']
```

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

In [334]:
```
launch_dict = {'FlightNumber': list(data['flight_number']),
'Date': list(data['date']),
'BoosterVersion':BoosterVersion,
'PayloadMass':PayloadMass,
'Orbit':Orbit,
'LaunchSite':LaunchSite,
'Outcome':Outcome,
'Flights':Flights,
'GridFins':GridFins,
'Reused':Reused,
'Legs':Legs,
'LandingPad':LandingPad,
'Block':Block,
'ReusedCount':ReusedCount,
'Serial':Serial,
'Longitude': Longitude,
'Latitude': Latitude}

```

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

In [335]:

```
# Create a data from launch_dict

data2 = pd.DataFrame(launch_dict)
```

Show the summary of the dataframe

In [336]:

```
# Show the head of the dataframe
data2.head()
```

Out[336]:

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

In [337]:

```
1 data2.info()
2 data2
```

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

Out[337]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad
0	1	2006-03-24	Falcon 1	20.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None
1	2	2007-03-21	Falcon 1	NaN	LEO	Kwajalein Atoll	None None	1	False	False	False	None
2	4	2008-09-28	Falcon 1	165.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None
3	5	2009-07-13	Falcon 1	200.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None
4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None
...	...	...	...	...	...	...	...	...	...	...	...	...
89	102	2020-09-03	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	2	True	True	True	5e9e3032383ecb6bb234e7ca
90	103	2020-10-06	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	3	True	True	True	5e9e3032383ecb6bb234e7ca
91	104	2020-10-18	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	6	True	True	True	5e9e3032383ecb6bb234e7ca
92	105	2020-10-24	Falcon 9	15600.0	VLEO	CCSFS SLC 40	True ASDS	3	True	True	True	5e9e3033383ecbb9e534e7cc
93	106	2020-11-05	Falcon 9	3681.0	MEO	CCSFS SLC 40	True ASDS	1	True	False	True	5e9e3032383ecb6bb234e7ca

94 rows × 17 columns

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` .

In [338]:
```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = data2[data2['BoosterVersion'] == 'Falcon9']
data_falcon9
```

Out[338]:

FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCou

In [339]:
```
data_falcon9.to_csv('data_falcon_9_week01.csv' , index=False)
```

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

In [340]:
```
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9
```

Out[340]:

FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCou

Data Wrangling

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

In [341]:
```
1 data2.isna().sum()
```

Out[341]:
```
FlightNumber       0
Date               0
BoosterVersion     0
PayloadMass        6
Orbit              0
LaunchSite         0
Outcome            0
Flights            0
GridFins           0
Reused             0
Legs               0
LandingPad        30
Block              4
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.

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

In [342]:
```
# Calculate the mean value of PayloadMass column

PayloadMass_avg = data2[['PayloadMass']].mean()
print(PayloadMass_avg)
print(data2['PayloadMass'])
```

```
PayloadMass    5919.165341
dtype: float64
0        20.0
1         NaN
2       165.0
3       200.0
4         NaN
       ...
89    15600.0
90    15600.0
91    15600.0
92    15600.0
93     3681.0
Name: PayloadMass, Length: 94, dtype: float64
```

In [343]:
```
# Replace the np.nan values with its mean value
data2['PayloadMass'] = data2['PayloadMass'].replace(np.nan, data2['PayloadMass'].mean())
```

In [344]:
```
print(data2['PayloadMass'])
```

```
0        20.000000
1      5919.165341
2       165.000000
3       200.000000
4      5919.165341
          ...
89    15600.000000
90    15600.000000
91    15600.000000
92    15600.000000
93     3681.000000
Name: PayloadMass, Length: 94, dtype: float64
```

In [345]:
```
1 data2.isna().sum()
```

Out[345]:
```
FlightNumber       0
Date               0
BoosterVersion     0
PayloadMass        0
Orbit              0
LaunchSite         0
Outcome            0
Flights            0
GridFins           0
Reused             0
Legs               0
LandingPad        30
Block              4
ReusedCount        0
Serial             0
Longitude          0
Latitude           0
dtype: int64
```

In [346]:

```
1 data2.info()
2 data2
```

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

Out[346]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad
0	1	2006-03-24	Falcon 1	20.000000	LEO	Kwajalein Atoll	None None	1	False	False	False	None
1	2	2007-03-21	Falcon 1	5919.165341	LEO	Kwajalein Atoll	None None	1	False	False	False	None
2	4	2008-09-28	Falcon 1	165.000000	LEO	Kwajalein Atoll	None None	1	False	False	False	None
3	5	2009-07-13	Falcon 1	200.000000	LEO	Kwajalein Atoll	None None	1	False	False	False	None
4	6	2010-06-04	Falcon 9	5919.165341	LEO	CCSFS SLC 40	None None	1	False	False	False	None
...	...	...	...	...	...	...	...	...	...	...	...	...
89	102	2020-09-03	Falcon 9	15600.000000	VLEO	KSC LC 39A	True ASDS	2	True	True	True	5e9e3032383ecb6bb234e7ca
90	103	2020-10-06	Falcon 9	15600.000000	VLEO	KSC LC 39A	True ASDS	3	True	True	True	5e9e3032383ecb6bb234e7ca
91	104	2020-10-18	Falcon 9	15600.000000	VLEO	KSC LC 39A	True ASDS	6	True	True	True	5e9e3032383ecb6bb234e7ca
92	105	2020-10-24	Falcon 9	15600.000000	VLEO	CCSFS SLC 40	True ASDS	3	True	True	True	5e9e3033383ecbb9e534e7cc
93	106	2020-11-05	Falcon 9	3681.000000	MEO	CCSFS SLC 40	True ASDS	1	True	False	True	5e9e3032383ecb6bb234e7ca

94 rows × 17 columns

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.

data_falcon9.to_csv('dataset_part_1.csv', index=False) `data_falcon9.to_csv('dataset_part_1.csv', index=False)`

In [347]:
```
1 data_falcon9.to_csv('dataset_part_1.csv', index=False)
```

Authors

Joseph Santarcangelo (https://www.linkedin.com/in/joseph-s-50398b136/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork865-2022-01-01) 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

(https://skills.network/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2022-01-01)

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:

Most unsuccessful landings are planned. Space X performs a controlled landing in the oceans.

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

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

In [ ]:
```
# 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 ma
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a MatLab like plotting framework. We will use th
import matplotlib.pyplot as plt
#Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawin
import seaborn as sns
```

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

```

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

In [ ]:

```
from js import fetch
import io

URL = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/data
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)
```

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.

In [ ]:

```
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.

In [ ]:

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

```

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

In [ ]:

```
# 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 c
```

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

In [ ]:
```
### 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.

In [ ]:
```
# 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
```

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).

In [ ]:
```
### 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

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

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

In [ ]:
```
### 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.

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

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.

In [ ]:
```
### TASK  5: Visualize the relationship between Payload and Orbit type

```

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

In [ ]:
```
# 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.

In [ ]:
```
### 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:

In [ ]:
```
# 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()

```

In [ ]:
```
# 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

In [ ]:
```
## Features Engineering

```

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

In [ ]:
```
features = df[['FlightNumber', 'PayloadMass', 'Orbit', 'LaunchSite', 'Flights', 'GridFins', 'Reused', 'Legs', 'Lan
features.head()
```

In [ ]:
```
### TASK  7: Create dummy variables to categorical columns

```

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

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

In [ ]:
```
### TASK  8: Cast all numeric columns to `float64`

```

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

In [ ]:
```
# 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.

```
features_one_hot.to_csv('dataset_part_3.csv', index=False)
```

Authors

[Pratiksha Verma (https://www.linkedin.com/in/pratiksha-verma-6487561b1/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork865-2022-01-01)](https://www.linkedin.com/in/pratiksha-verma-6487561b1/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork865-2022-01-01)

Change Log

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

(https://skills.network/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2022-01-01)

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:

Most unsuccessful landings are planned. Space X performs a controlled landing in the oceans.

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

In [9]:
```
# 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 ma
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a MatLab like plotting framework. We will use th
import matplotlib.pyplot as plt
#Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawin
import seaborn as sns
```

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

```

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

In [10]:

```
URL = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/data
df=pd.read_csv(URL)
df.head(5)
```

Out[10]:

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

In [11]:

```
df.describe()
```

Out[11]:

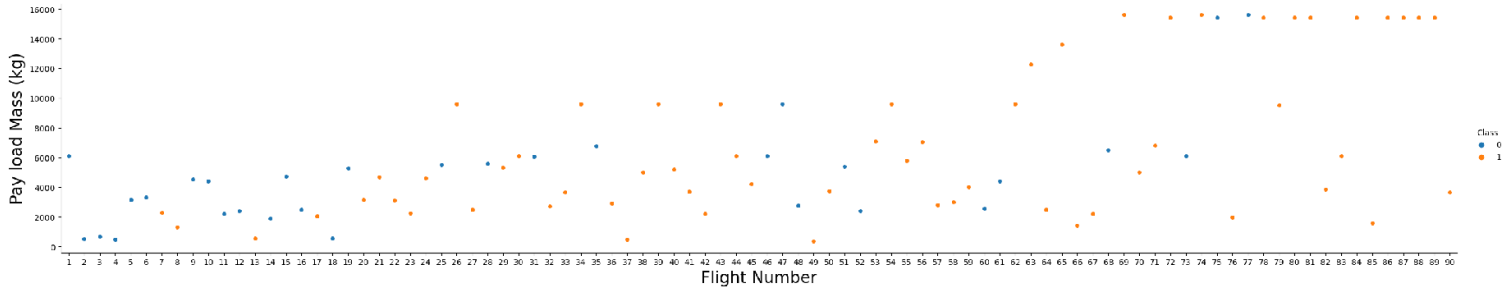
	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Longitude	Latitude	Class
count	90.000000	90.000000	90.000000	90.000000	90.000000	90.000000	90.000000	90.000000
mean	45.500000	6104.959412	1.788889	3.500000	1.655556	-86.366477	29.449963	0.666667
std	26.124701	4694.671720	1.213172	1.595288	1.710254	14.149518	2.141306	0.474045
min	1.000000	350.000000	1.000000	1.000000	0.000000	-120.610829	28.561857	0.000000
25%	23.250000	2510.750000	1.000000	2.000000	0.000000	-80.603956	28.561857	0.000000
50%	45.500000	4701.500000	1.000000	4.000000	1.000000	-80.577366	28.561857	1.000000
75%	67.750000	8912.750000	2.000000	5.000000	3.000000	-80.577366	28.608058	1.000000
max	90.000000	15600.000000	6.000000	5.000000	5.000000	-80.577366	34.632093	1.000000

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

In [12]:
```
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()
```

```
C:\Users\parichea\AppData\Local\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout h
as changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



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.

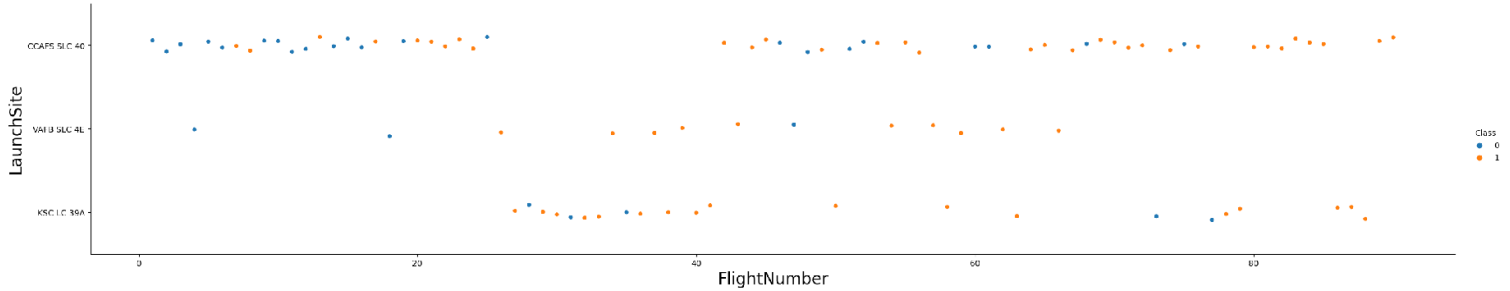
In [16]:
```
### TASK 1: Visualize the relationship between Flight Number and Launch Site

```

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

In [15]:

```
# 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 c
sns.catplot(y="LaunchSite", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("FlightNumber",fontsize=20)
plt.ylabel("LaunchSite",fontsize=20)
plt.show()
```

```
C:\Users\parichea\AppData\Local\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout h
as changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



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

In [18]:

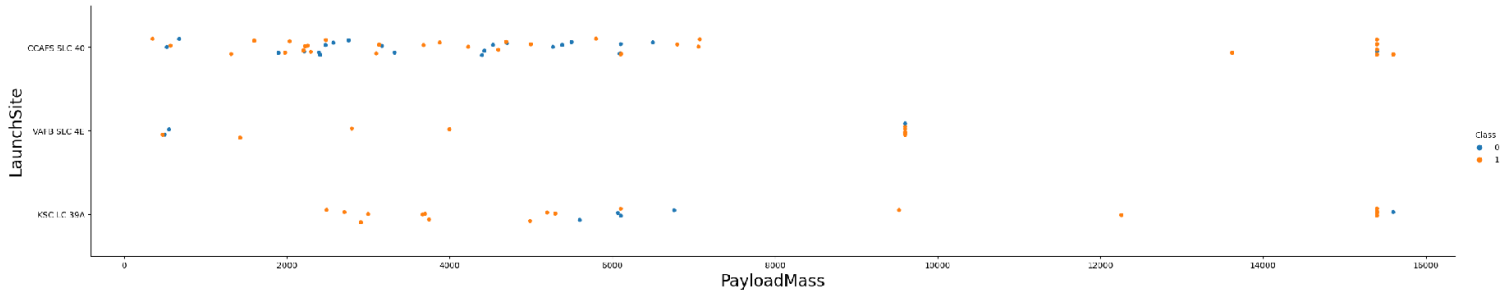
```
### TASK 2: Visualize the relationship between Payload and Launch Site

```

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

In [19]:

```
# 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
sns.catplot(y="LaunchSite", x="PayloadMass", hue="Class", data=df, aspect = 5)
plt.xlabel("PayloadMass",fontsize=20)
plt.ylabel("LaunchSite",fontsize=20)
plt.show()
```

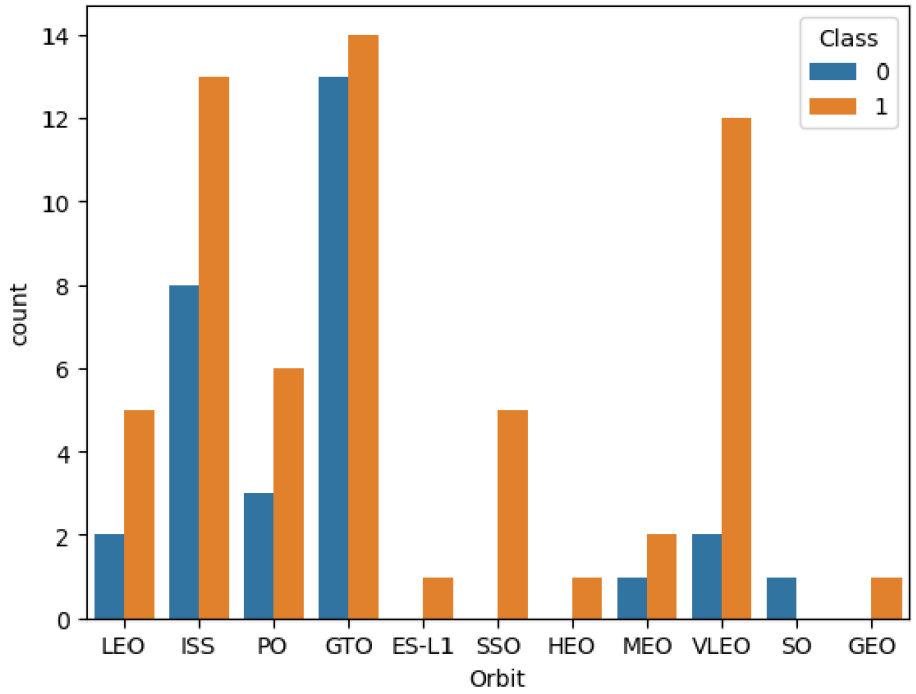
```
C:\Users\parichea\AppData\Local\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout h
as changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



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).

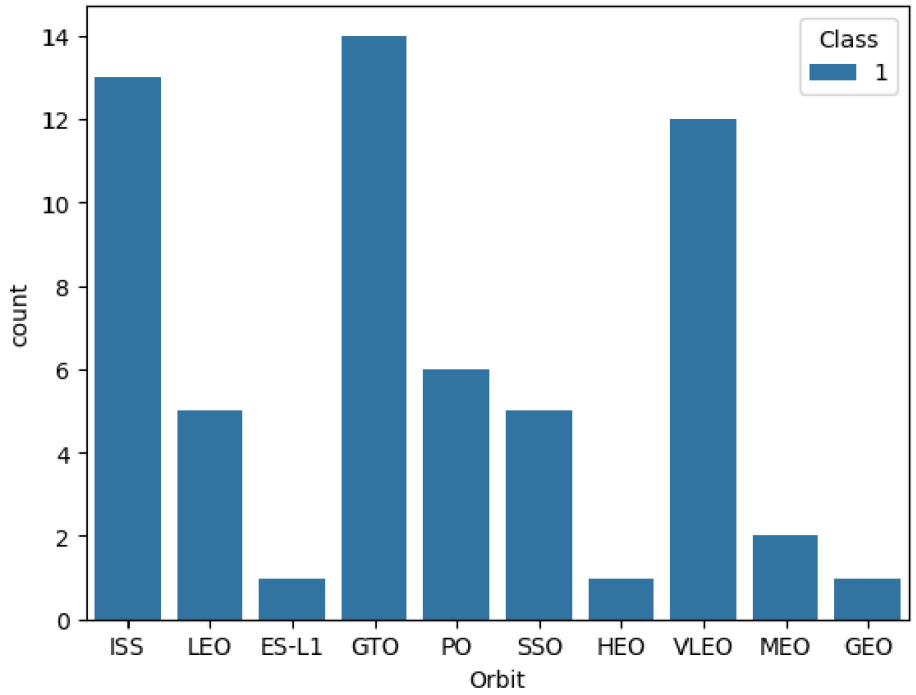
In [21]:

```
### TASK  3: Visualize the relationship between success rate of each orbit type
sns.countplot(data=df, x='Orbit', hue='Class')
plt.show()
```

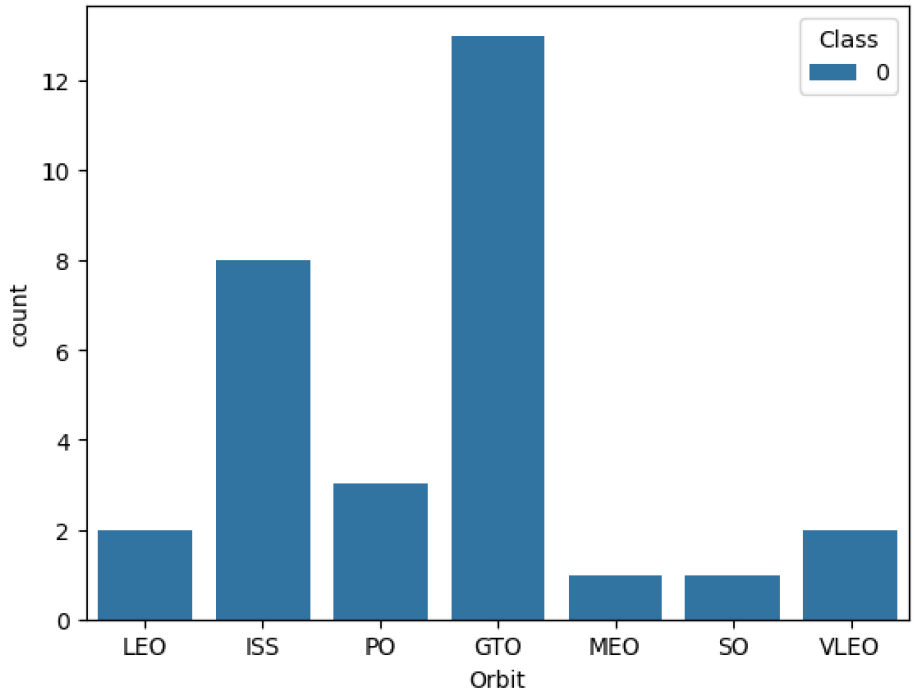


In [26]:

```
sns.countplot(data=df_success, x='Orbit', hue='Class')
plt.show()
```



In [27]:
```
sns.countplot(data=df_fail, x='Orbit', hue='Class')
plt.show()
```



In [22]:
```
df_success=df[df['Class']==1]
df_fail= df[df['Class']==0]
```

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

In [23]:
```
y=set(df_success['Orbit'])
y
```

Out[23]: {'ES-L1', 'GEO', 'GTO', 'HEO', 'ISS', 'LEO', 'MEO', 'PO', 'SSO', 'VLEO'}

In [24]:
```
x=set(df_fail['Orbit'])
x
```

Out[24]:
```
{'GTO', 'ISS', 'LEO', 'MEO', 'PO', 'SO', 'VLEO'}
```

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

In [25]:
```
per=(df_success['Orbit'].value_counts())
per
```

Out[25]:
```
Orbit
GTO      14
ISS      13
VLEO     12
PO        6
LEO       5
SSO       5
MEO       2
ES-L1     1
HEO       1
GEO       1
Name: count, dtype: int64
```

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

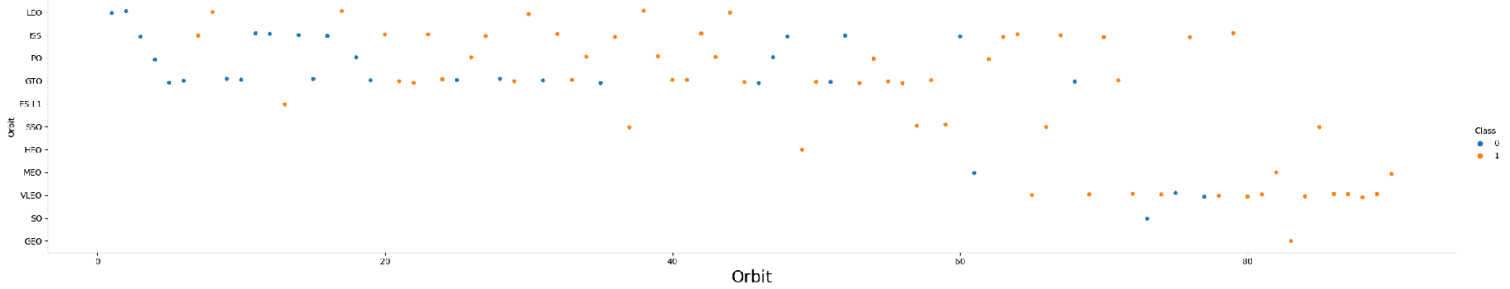
In [ ]:
```
### TASK  4: Visualize the relationship between FlightNumber and Orbit type

```

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

In [28]:

```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class va
sns.catplot(y='Orbit', x='FlightNumber', hue='Class',data=df,aspect = 5)
plt.xlabel('FlightNumber', fontsize=20)
plt.xlabel('Orbit', fontsize=20)
plt.show()

```

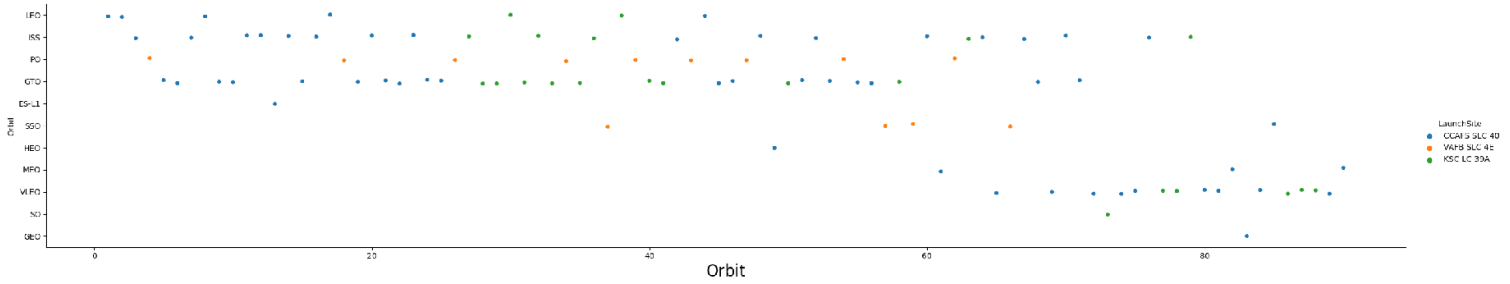
```
C:\Users\parichea\AppData\Local\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout h
as changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



In [29]:

```
sns.catplot(y='Orbit', x='FlightNumber', hue='LaunchSite',data=df,aspect = 5)
plt.xlabel('FlightNumber', fontsize=20)
plt.xlabel('Orbit', fontsize=20)
plt.show()
```

```
C:\Users\parichea\AppData\Local\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout h
as changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



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.

In [ ]:

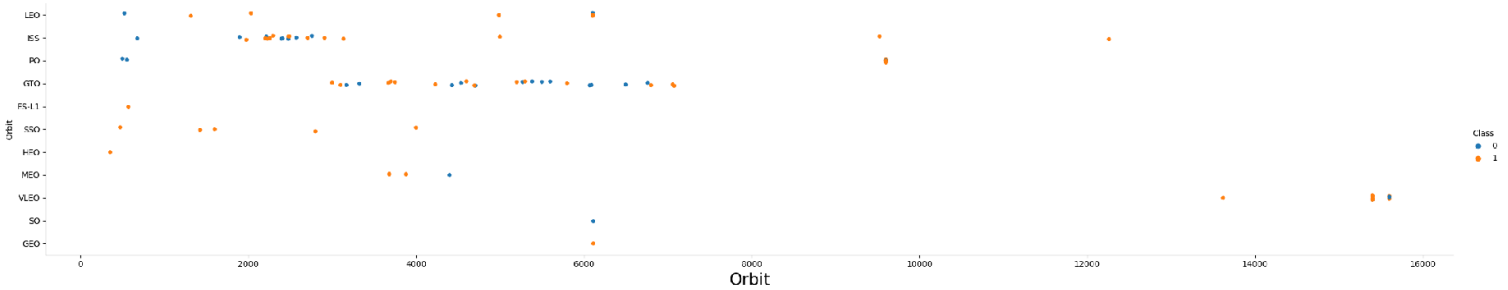
```
### TASK  5: Visualize the relationship between Payload and Orbit type

```

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

In [30]:

```
# Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, and hue to be the class value
sns.catplot(y='Orbit', x='PayloadMass', hue='Class',data=df,aspect = 5)
plt.xlabel('PayloadMass', fontsize=20)
plt.xlabel('Orbit', fontsize=20)
plt.show()
```

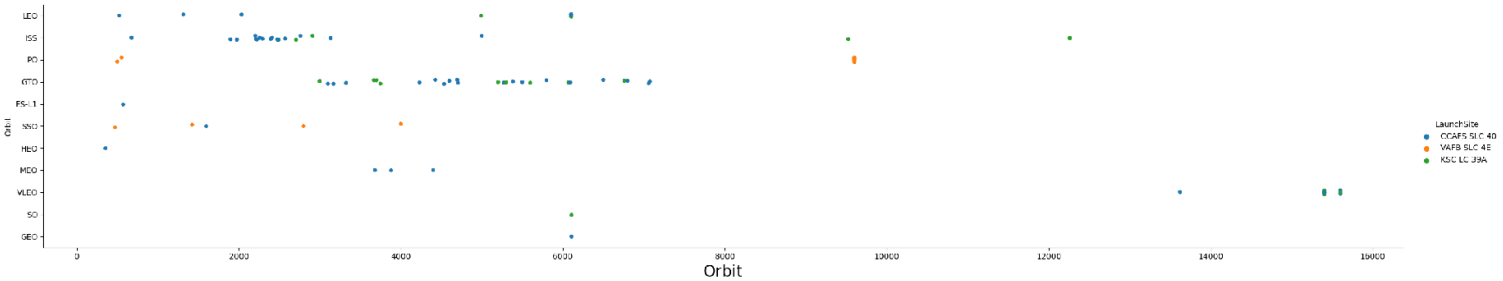
```
C:\Users\parichea\AppData\Local\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout h
as changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



In [31]:

```
sns.catplot(y='Orbit', x='PayloadMass', hue='LaunchSite',data=df,aspect = 5)
plt.xlabel('PayloadMass', fontsize=20)
plt.xlabel('Orbit', fontsize=20)
plt.show()

```

```
C:\Users\parichea\AppData\Local\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout h
as changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



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.

In [ ]:

```
### 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:

In [32]:

```
# 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()

```

Out[32]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCo
0	1	2010	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	
1	2	2012	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	
2	3	2013	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	
3	4	2013	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	
4	5	2013	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	

In [33]:

```
df_success=df[df['Class']==1]
```

In [37]:

```
df_line=df_success[['Date', 'Class']]
df_line
```

Out[37]:

	Date	Class
6	2014	1
7	2014	1
12	2015	1
16	2015	1
19	2016	1
20	2016	1
21	2016	1
22	2016	1
23	2016	1
25	2017	1
26	2017	1
28	2017	1
29	2017	1
31	2017	1
32	2017	1
33	2017	1
35	2017	1
36	2017	1
37	2017	1
38	2017	1
39	2017	1
40	2017	1
41	2017	1
42	2017	1
43	2018	1
44	2018	1

	Date	Class
48	2018	1
49	2018	1
52	2018	1
53	2018	1
54	2018	1
55	2018	1
56	2018	1
57	2018	1
58	2018	1
61	2019	1
62	2019	1
63	2019	1
64	2019	1
65	2019	1
66	2019	1
68	2019	1
69	2019	1
70	2019	1
71	2020	1
73	2020	1
75	2020	1
77	2020	1
78	2020	1
79	2020	1
80	2020	1
81	2020	1
82	2020	1

	Date	Class
83	2020	1
84	2020	1
85	2020	1
86	2020	1
87	2020	1
88	2020	1
89	2020	1

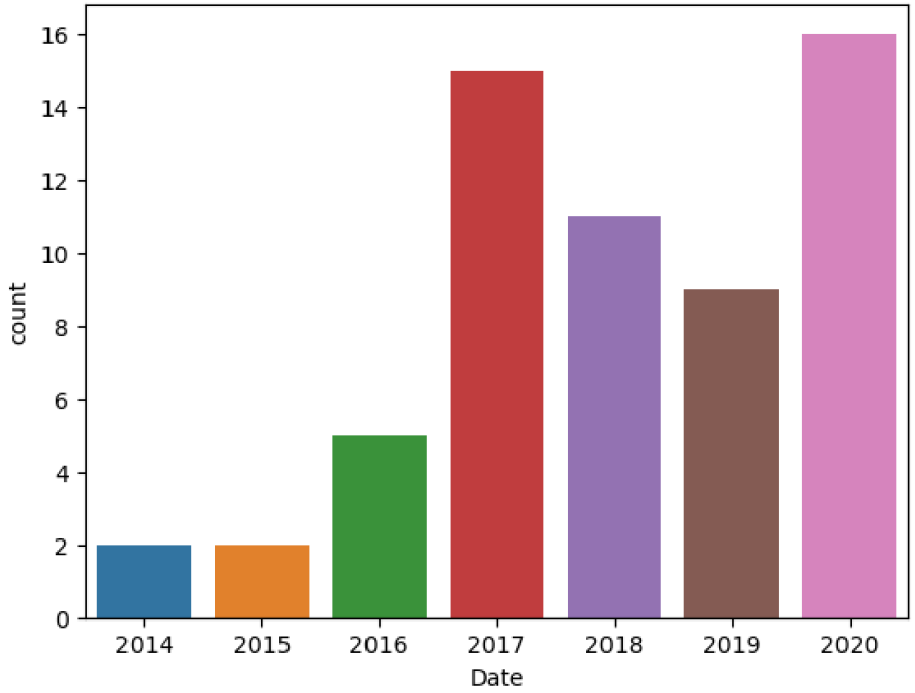
In [35]:
```
1 df_success['Class'].count()
```

Out[35]: 60

In [ ]:
```
1
```

In [38]:

```
# Plot a line chart with x axis to be the extracted year and y axis to be the success rate
sns.countplot(x='Date', data=df_success)
plt.show()
```



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

In [ ]:

```
## Features Engineering

```

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

In [39]:
```
features = df[['FlightNumber', 'PayloadMass', 'Orbit', 'LaunchSite', 'Flights', 'GridFins', 'Reused', 'Legs', 'Lan
features.head()
```

Out[39]:

	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

In [ ]:
```
### TASK  7: Create dummy variables to categorical columns

```

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

In [46]:
```
# HINT: Use get_dummies() function on the categorical columns
features_one_hot= pd.get_dummies(df[['Orbit', 'LaunchSite','LandingPad', 'Serial']])
features_one_hot.head()
```

Out[46]:

	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	Orbit_LEO	Orbit_MEO	Orbit_PO	Orbit_SO	Orbit_SSO	...	Serial_B1048	Serial_B1049
0	False	False	False	False	False	True	False	False	False	False	...	False	False
1	False	False	False	False	False	True	False	False	False	False	...	False	False
2	False	False	False	False	True	False	False	False	False	False	...	False	False
3	False	False	False	False	False	False	False	True	False	False	...	False	False
4	False	False	True	False	False	False	False	False	False	False	...	False	False

5 rows × 72 columns

In [47]:
```
df_Dummy= features_one_hot.astype(float)
```

In [60]:
```
df_Dummy
```

Out[60]:

	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	Orbit_LEO	Orbit_MEO	Orbit_PO	Orbit_SO	Orbit_SSO	...	Serial_B1048	Serial_B104
0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.(
1	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.(
2	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.(
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.(
4	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.(
...	...	...	...	...	...	...	...	...	...	...	...	...	..
85	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.(
86	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.(
87	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.(
88	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.(
89	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.(

90 rows × 72 columns

In [ ]:
```
df=df.drop(['Orbit', 'LaunchSite','LandingPad', 'Serial'], axis=1)
```

In [53]:
```
1 df
```

Out[53]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Outcome	Flights	GridFins	Reused	Legs	Block	ReusedCount	Longitude	Latitude	C
0	1	2010	Falcon 9	6104.959412	None None	1	False	False	False	1.0	0	-80.577366	28.561857	
1	2	2012	Falcon 9	525.000000	None None	1	False	False	False	1.0	0	-80.577366	28.561857	
2	3	2013	Falcon 9	677.000000	None None	1	False	False	False	1.0	0	-80.577366	28.561857	
3	4	2013	Falcon 9	500.000000	False Ocean	1	False	False	False	1.0	0	-120.610829	34.632093	
4	5	2013	Falcon 9	3170.000000	None None	1	False	False	False	1.0	0	-80.577366	28.561857	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
85	86	2020	Falcon 9	15400.000000	True ASDS	2	True	True	True	5.0	2	-80.603956	28.608058	
86	87	2020	Falcon 9	15400.000000	True ASDS	3	True	True	True	5.0	2	-80.603956	28.608058	
87	88	2020	Falcon 9	15400.000000	True ASDS	6	True	True	True	5.0	5	-80.603956	28.608058	
88	89	2020	Falcon 9	15400.000000	True ASDS	3	True	True	True	5.0	2	-80.577366	28.561857	
89	90	2020	Falcon 9	3681.000000	True ASDS	1	True	False	True	5.0	0	-80.577366	28.561857	

90 rows × 14 columns

In [59]:
```
1 df=pd.concat([df, df_Dummy], axis=1)
```

In [56]:
```
df
```

Out[56]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Outcome	Flights	GridFins	Reused	Legs	Block	...	Serial_B1048	Serial_B1049	Serial_B
0	1	2010	Falcon 9	6104.959412	None None	1	False	False	False	1.0	...	0.0	0.0	
1	2	2012	Falcon 9	525.000000	None None	1	False	False	False	1.0	...	0.0	0.0	
2	3	2013	Falcon 9	677.000000	None None	1	False	False	False	1.0	...	0.0	0.0	
3	4	2013	Falcon 9	500.000000	False Ocean	1	False	False	False	1.0	...	0.0	0.0	
4	5	2013	Falcon 9	3170.000000	None None	1	False	False	False	1.0	...	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
85	86	2020	Falcon 9	15400.000000	True ASDS	2	True	True	True	5.0	...	0.0	0.0	
86	87	2020	Falcon 9	15400.000000	True ASDS	3	True	True	True	5.0	...	0.0	0.0	
87	88	2020	Falcon 9	15400.000000	True ASDS	6	True	True	True	5.0	...	0.0	0.0	
88	89	2020	Falcon 9	15400.000000	True ASDS	3	True	True	True	5.0	...	0.0	0.0	
89	90	2020	Falcon 9	3681.000000	True ASDS	1	True	False	True	5.0	...	0.0	0.0	

90 rows × 86 columns

In [58]:
```
df.to_csv('week02_02.csv', index=False)
```

In [ ]:
```
### TASK  8: Cast all numeric columns to `float64`

```

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

In [65]:
```
# HINT: use astype function
df_Dummy= features_one_hot.astype(float)
```

In [66]:
```
features_one_hot.to_csv('dataset_part_3.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.

```
features_one_hot.to_csv('dataset_part_3.csv', index=False)
```

Authors

Pratiksha Verma (https://www.linkedin.com/in/pratiksha-verma-6487561b1/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork865-2022-01-01)

Change Log

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

(https://skills.network/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2022-01-01)

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:

In [202]:
```
!pip install folium
```

```
Requirement already satisfied: folium in c:\users\parichea\appdata\local\anaconda3\lib\site-packages (0.15.1)
Requirement already satisfied: branca>=0.6.0 in c:\users\parichea\appdata\local\anaconda3\lib\site-packages (from fol
ium) (0.7.1)
Requirement already satisfied: jinja2>=2.9 in c:\users\parichea\appdata\local\anaconda3\lib\site-packages (from foliu
m) (3.1.2)
Requirement already satisfied: numpy in c:\users\parichea\appdata\local\anaconda3\lib\site-packages (from folium) (1.
24.3)
Requirement already satisfied: requests in c:\users\parichea\appdata\local\anaconda3\lib\site-packages (from folium)
(2.31.0)
Requirement already satisfied: xyzservices in c:\users\parichea\appdata\local\anaconda3\lib\site-packages (from foliu
m) (2022.9.0)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\parichea\appdata\local\anaconda3\lib\site-packages (from j
inja2>=2.9->folium) (2.1.1)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\parichea\appdata\local\anaconda3\lib\site-package
s (from requests->folium) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\parichea\appdata\local\anaconda3\lib\site-packages (from requ
ests->folium) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\parichea\appdata\local\anaconda3\lib\site-packages (fro
m requests->folium) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\parichea\appdata\local\anaconda3\lib\site-packages (fro
m requests->folium) (2023.7.22)
```

In [203]:
```
import folium
import pandas as pd
```

In [204]:
```
# 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 (https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/labs/module_3/DV0101EN-3-5-1-Generating-Maps-in-Python-py-v2.0.ipynb)

In [205]:
```
## 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.

In [206]:
```
# Download and read the `spacex_launch_geo.csv`
URL = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/spac
```

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

In [207]:
```
df = pd.read_csv(URL)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 56 entries, 0 to 55
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Flight Number      56 non-null     int64  
 1   Date               56 non-null     object 
 2   Time (UTC)         56 non-null     object 
 3   Booster Version    56 non-null     object 
 4   Launch Site        56 non-null     object 
 5   Payload            56 non-null     object 
 6   Payload Mass (kg)  56 non-null     float64
 7   Orbit              56 non-null     object 
 8   Customer           56 non-null     object 
 9   Landing Outcome    56 non-null     object 
 10  class              56 non-null     int64  
 11  Lat                56 non-null     float64
 12  Long               56 non-null     float64
dtypes: float64(3), int64(2), object(8)
memory usage: 5.8+ KB
```

In [208]:
```
df.head()
```

Out[208]:

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

In [209]:
```
df.describe()
```

Out[209]:

	Flight Number	Payload Mass (kg)	class	Lat	Long
count	56.000000	56.000000	56.000000	56.000000	56.000000
mean	28.500000	3696.648214	0.428571	29.648980	-87.742252
std	16.309506	2568.509679	0.499350	2.344768	15.463732
min	1.000000	0.000000	0.000000	28.562302	-120.610745
25%	14.750000	2121.000000	0.000000	28.562302	-80.646895
50%	28.500000	3412.500000	0.000000	28.563197	-80.577356
75%	42.250000	5042.500000	1.000000	28.573255	-80.577356
max	56.000000	9600.000000	1.000000	34.632834	-80.576820

In [210]:
```
# Select relevant sub-columns: `Launch Site`, `Lat(Latitude)`, `Long(Longitude)`, `class`
spacex_df = df[['Launch Site', 'Lat', 'Long', 'class']]
```

In [211]:
```
Launch_Site_df= spacex_df.groupby(['Launch Site'], as_index=False).first()
```

In [212]:
```
Launch_Site_df
```

Out[212]:

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

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.

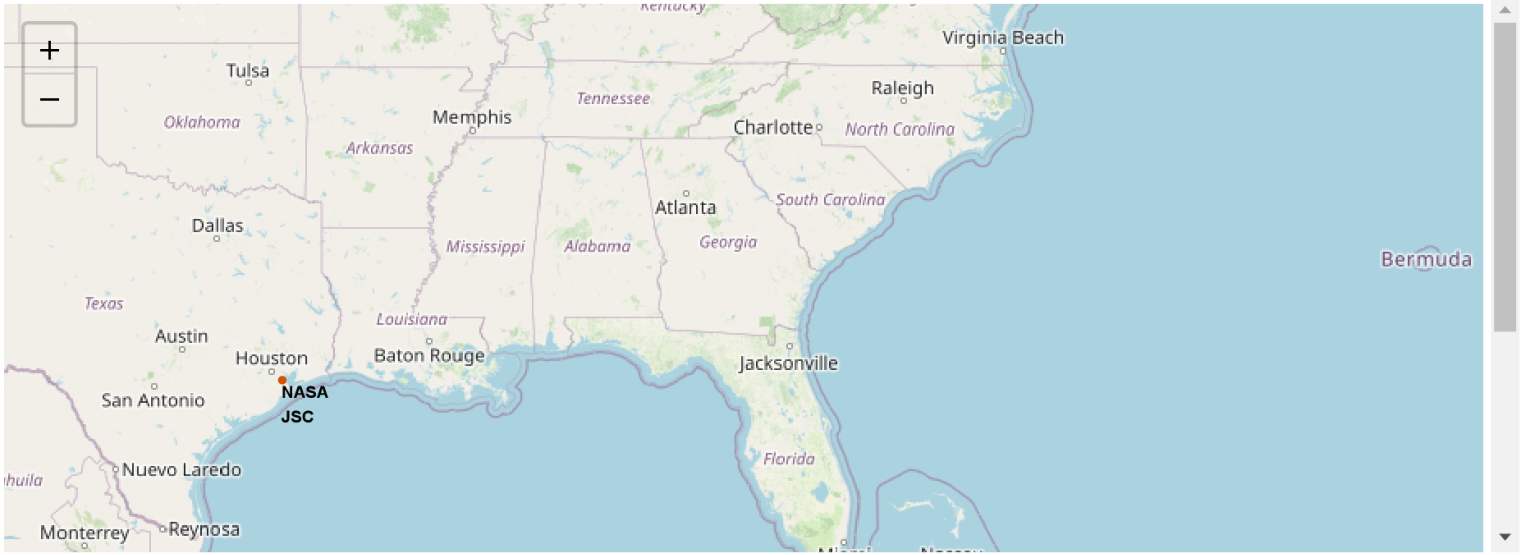
In [213]:
```
# Start location is NASA Johnson Space Center
nasa_coordinate = [29.559684888503615, -95.0830971930759]
site_map = folium.Map(location=nasa_coordinate, zoom_start=10)
```

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

In [214]:

```
# 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 John
# 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)
```

Out[214]:



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

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', ))
```

In [215]:

```
# 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 si

```

In [216]:

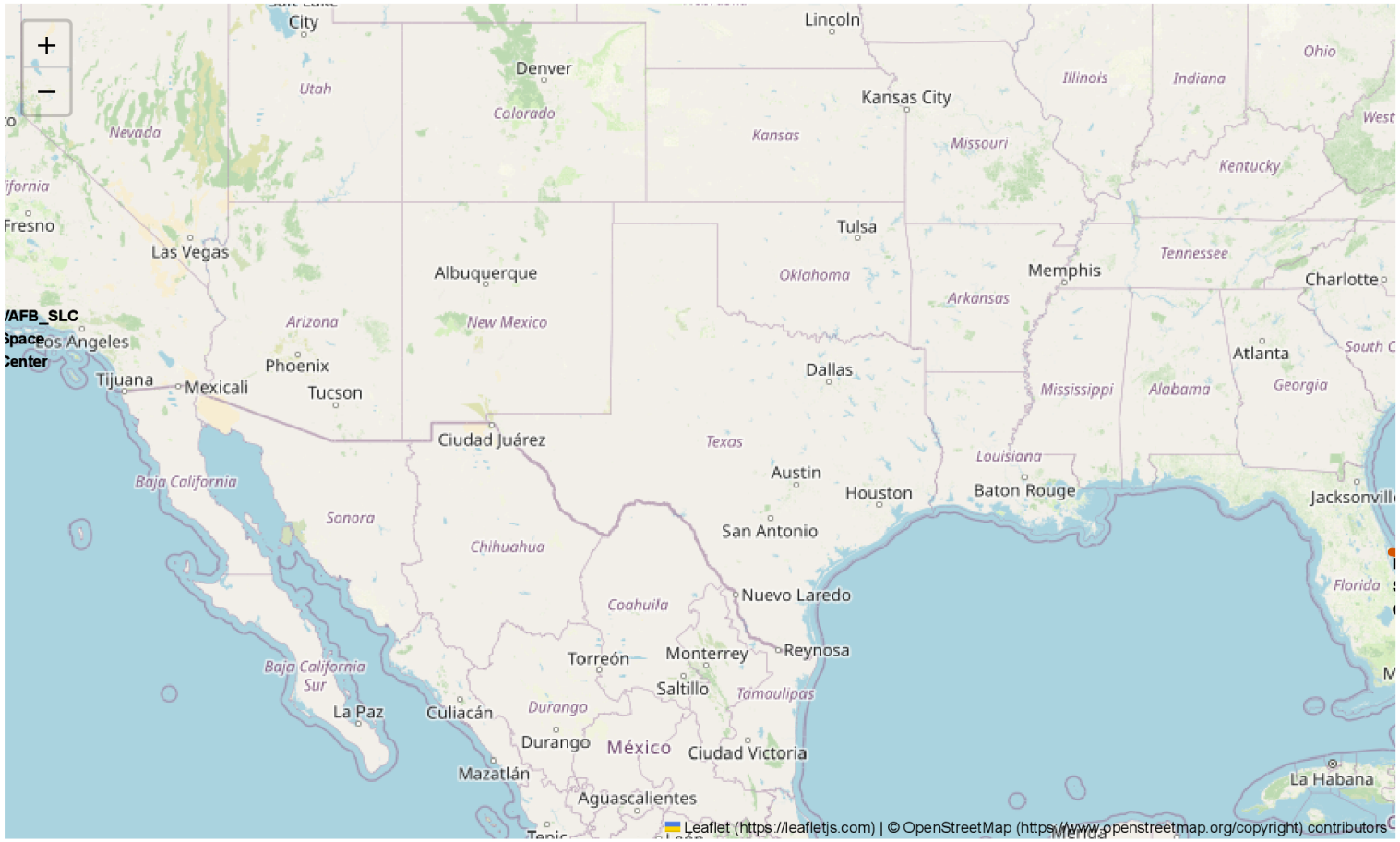
```
CCAFS_SLC_coordinate =  [28.563197, -80.576820]
site_map = folium.Map(location=CCAFS_SLC_coordinate, zoom_start=10)
#-----------------------------------------------------
CCAFS_LC_coordinate = [28.562302, -80.577356]
site_map = folium.Map(location=CCAFS_LC_coordinate, zoom_start=10)
#-----------------------------------------------------
KSC_LC_coordinate =  [28.573255, -80.646895]
site_map = folium.Map(location=KSC_LC_coordinate, zoom_start=10)
#-----------------------------------------------------
VAFB_SLC_coordinate =  [34.632834, -120.610745]
site_map = folium.Map(location=VAFB_SLC_coordinate, zoom_start=10)
```

In [218]:

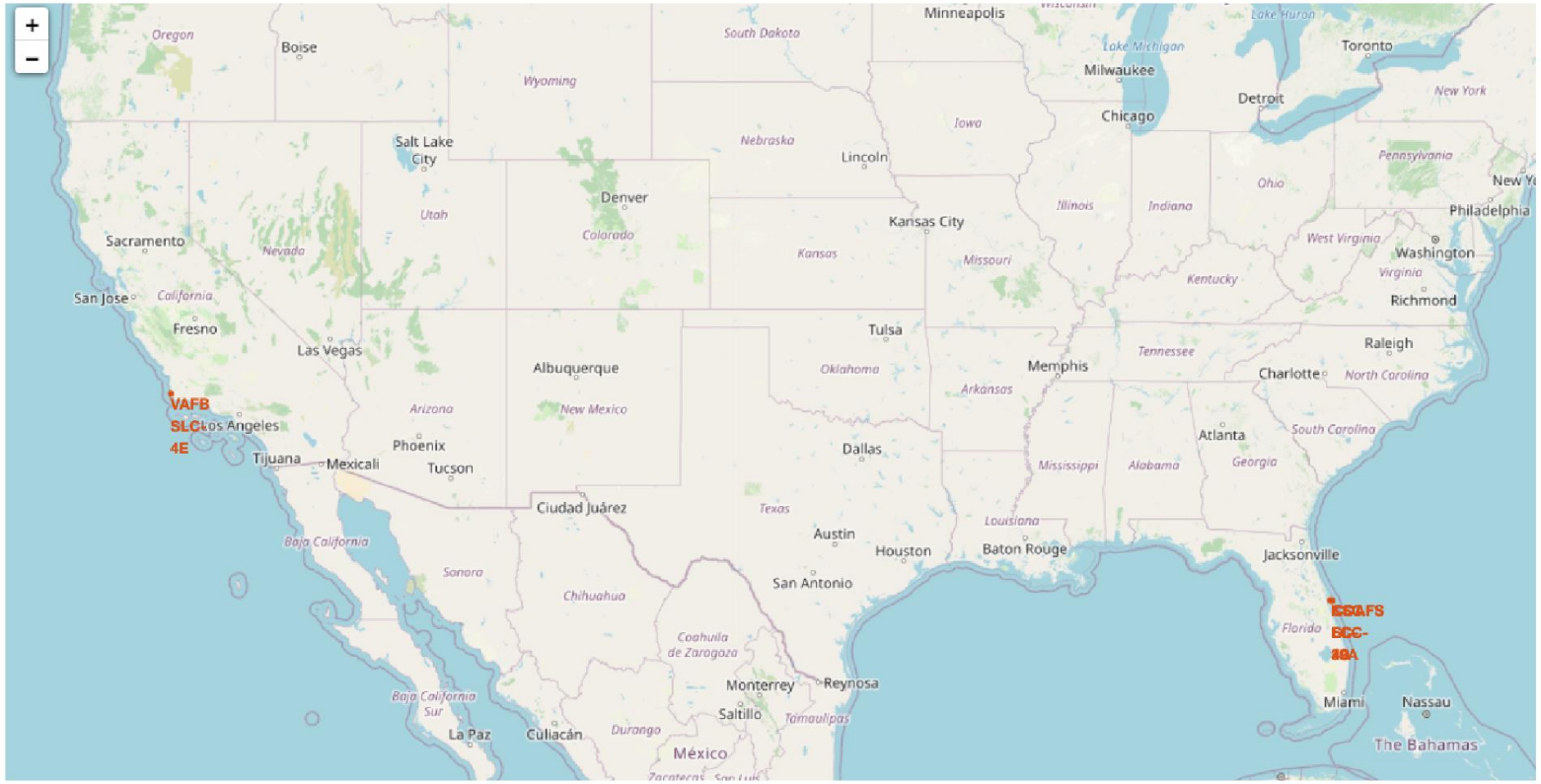
```
CCAFS_SLC_coordinate =  [28.563197, -80.576820]
folium.Circle(CCAFS_SLC_coordinate, radius=1000, color='#d35400', fill=True).add_child(folium.Popup('CCAFS_SLC Spa
circle = folium.Circle(CCAFS_SLC_coordinate, radius=1000, color='#d35400', fill=True).add_child(folium.Popup('CCAF
marker = folium.map.Marker(
    CCAFS_SLC_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>' % 'CCAFS_SLC Space Center',
        )
    )
site_map.add_child(circle)
site_map.add_child(marker)

CCAFS_LC_coordinate = [28.562302, -80.577356]
folium.Circle(CCAFS_LC_coordinate, radius=1000, color='#d35400', fill=True).add_child(folium.Popup('CCAFS_LC Space
circle = folium.Circle(CCAFS_LC_coordinate, radius=1000, color='#d35400', fill=True).add_child(folium.Popup('CCAFS
marker = folium.map.Marker(
    CCAFS_LC_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>' % 'CCAFS_LC Space Center',
        )
    )
site_map.add_child(circle)
site_map.add_child(marker)


KSC_LC_coordinate =  [28.573255, -80.646895]
folium.Circle(KSC_LC_coordinate, radius=1000, color='#d35400', fill=True).add_child(folium.Popup('KSC_LC Space Cen
circle = folium.Circle(KSC_LC_coordinate, radius=1000, color='#d35400', fill=True).add_child(folium.Popup('KSC_LC
marker = folium.map.Marker(
    KSC_LC_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>' % 'KSC_LC Space Center',
        )
    )
```

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

VAFB_SLC_coordinate =  [34.632834, -120.610745]
folium.Circle(VAFB_SLC_coordinate, radius=1000, color='#d35400', fill=True).add_child(folium.Popup('VAFB_SLC Space
circle = folium.Circle(VAFB_SLC_coordinate, radius=1000, color='#d35400', fill=True).add_child(folium.Popup('VAFB_
marker = folium.map.Marker(
    VAFB_SLC_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>' % 'VAFB_SLC Space Center',
        )
    )
site_map.add_child(circle)
site_map.add_child(marker)

```

Out[218]:



The generated map with marked launch sites should look similar to the following:



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.

In [219]:

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

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

In [220]:
```
1 spacex_df.head(10)
```

Out[220]:

	Launch Site	Lat	Long	class
0	CCAFS LC-40	28.562302	-80.577356	0
1	CCAFS LC-40	28.562302	-80.577356	0
2	CCAFS LC-40	28.562302	-80.577356	0
3	CCAFS LC-40	28.562302	-80.577356	0
4	CCAFS LC-40	28.562302	-80.577356	0
5	CCAFS LC-40	28.562302	-80.577356	0
6	CCAFS LC-40	28.562302	-80.577356	0
7	CCAFS LC-40	28.562302	-80.577356	0
8	CCAFS LC-40	28.562302	-80.577356	0
9	CCAFS LC-40	28.562302	-80.577356	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

In [221]:
```
1 marker_cluster = MarkerCluster()
2
```

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

In [222]:

```
# 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
def assign_marker_color(launch_outcome):
    if launch_outcome == 1:
        return 'green'
    else:
        return 'red'
spacex_df['marker_color'] = spacex_df['class'].apply(assign_marker_color)
spacex_df.tail(10)
```

```
C:\Users\parichea\AppData\Local\Temp\ipykernel_24884\2727443614.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  spacex_df['marker_color'] = spacex_df['class'].apply(assign_marker_color)
```

Out[222]:

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

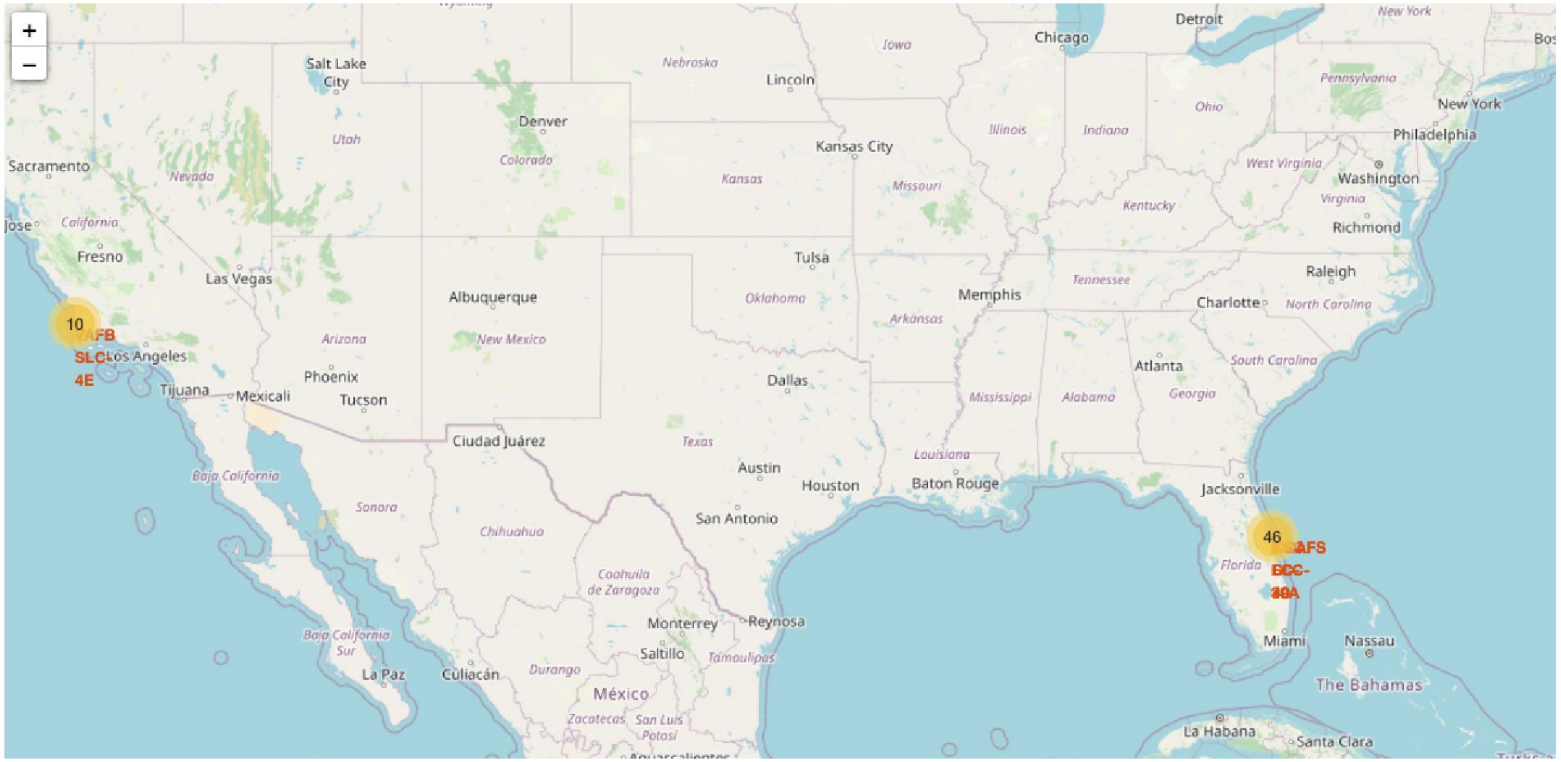
TODO: For each launch result in `spacex_df` data frame, add a `folium.Marker` to `marker_cluster`

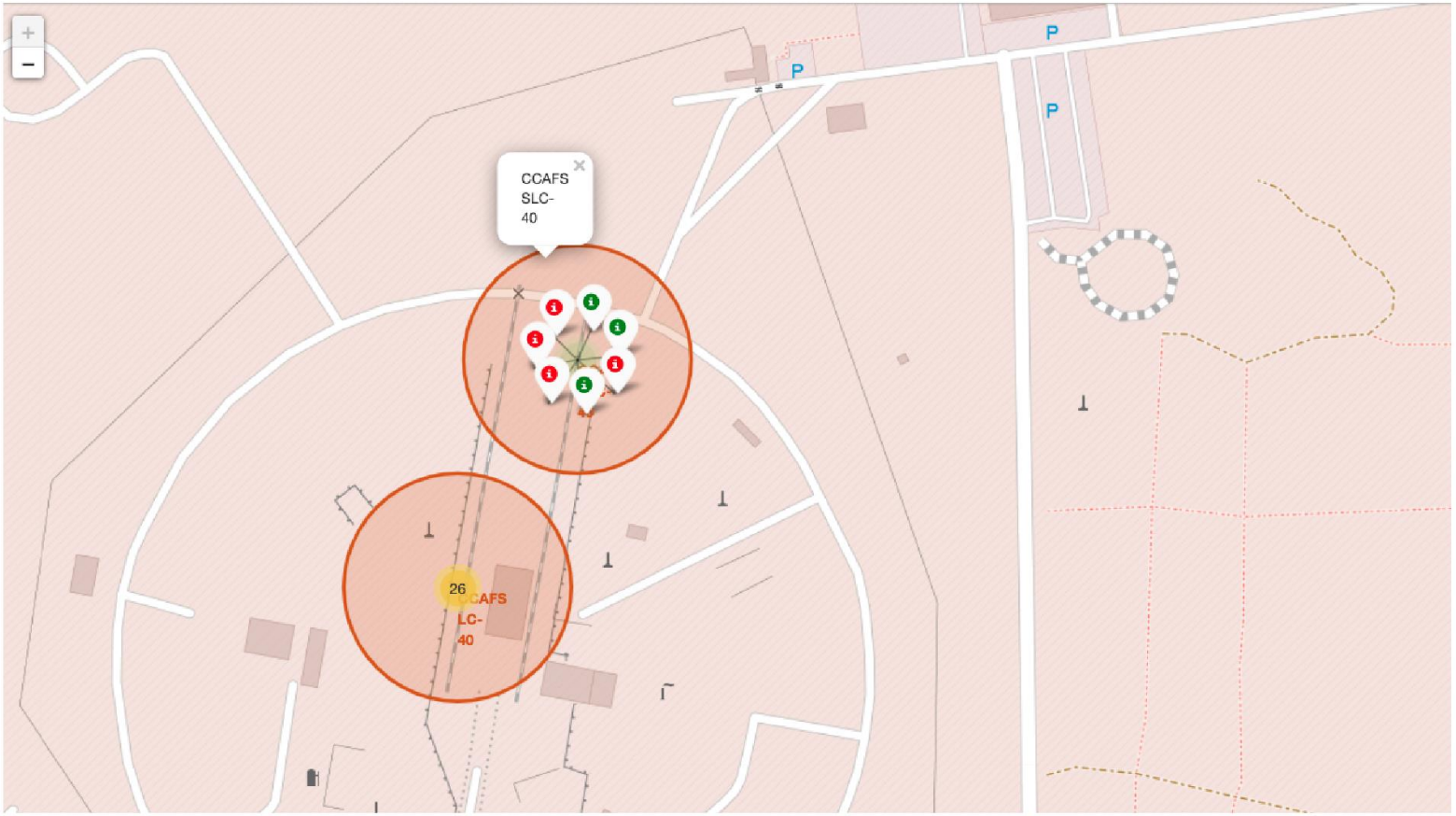
In [223]:

```
# 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():
    # TODO: Create and add a Marker cluster to the site map
    coordinate = [record['Lat'], record['Long']]
    folium.map.Marker(coordinate, icon=folium.Icon(color='white', icon_color=record['marker_color'])).add_to(marke
site_map
```

Out[223]:

Your updated map may look like the following screenshots:


+
−
Salt Lake City
Denver
Sacramento
Nevada
Utah
Colorado
Nebraska
Lincoln
Iowa
Chicago
Detroit
New York
Kansas City
Kansas
Missouri
Illinois
Indiana
Ohio
Pennsylvania
Philadelphia
West Virginia
Washington
Virginia
Kentucky
Richmond
California
Fresno
Las Vegas
Albuquerque
Tulsa
Oklahoma
Memphis
Tennessee
Raleigh
Charlotte
North Carolina
Arkansas
10
FB
SLC
4E
Los Angeles
Arizona
New Mexico
Phoenix
Tijuana
Mexicali
Tucson
Dallas
Atlanta
South Carolina
Mississippi
Alabama
Georgia
Ciudad Juárez
Texas
Austin
Houston
Baton Rouge
Louisiana
Jacksonville
Baja California
Sonora
Chihuahua
San Antonio
46
AFS
BCC-
Florida
Coahuila de Zaragoza
Monterrey
Reynosa
Baja California Sur
Saltillo
Tamaulipas
Miami
Nassau
La Paz
Culiacán
Durango
México
The Bahamas
Zacatecas
San Luis Potosí
La Habana
Santa Clara



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

In [224]:

```
# 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 points of interests (such as railway)

In [226]:

```
# 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
```

Out[226]:

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.

In [227]:

```
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
```

In [228]:

```
# find coordinate of the closet coastline
launch_site_lat = 28.563197
launch_site_lon = -80.576820
coastline_lat = 28.56334
coastline_lon = -80.56799
distance_coastline = calculate_distance(launch_site_lat, launch_site_lon, coastline_lat, coastline_lon)
print(distance_coastline, 'km')
```

```
0.8627671182499878 km
```

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

In [229]:

```
# 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

distance_marker =folium.Marker(
   coordinate,
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % '(:10.2f) KM'.format(distance_coastlin
        )
    )
```

In [230]:

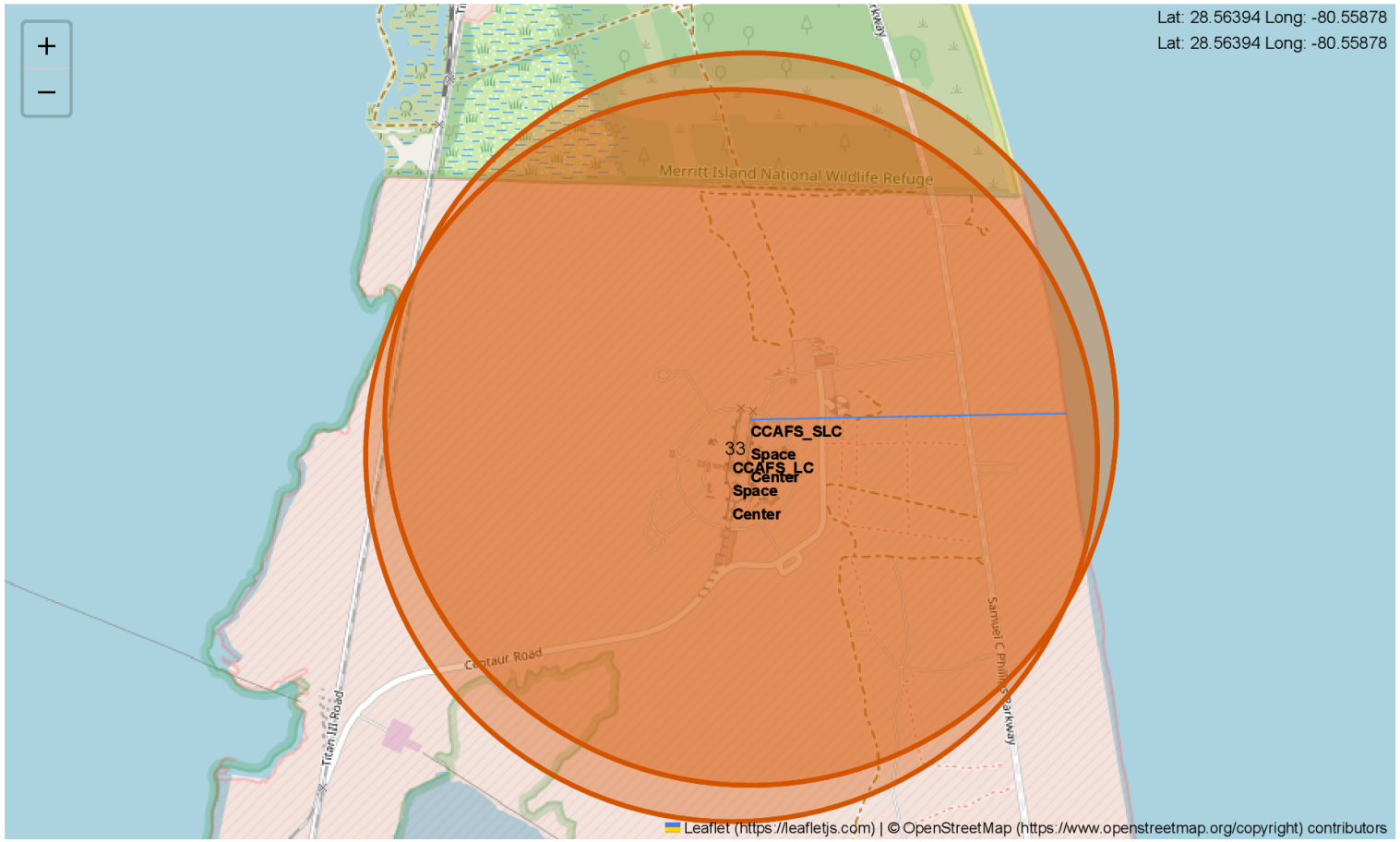
```
print(distance_marker)
```

```
<folium.map.Marker object at 0x000001E069E9E590>
```

In [231]:

```
# Create a `folium.PolyLine` object using the coastline coordinates and launch site coordinate

coordinates = [[launch_site_lat, launch_site_lon], [coastline_lat, coastline_lon]]
lines=folium.PolyLine(locations=coordinates, weight=1)
lines=folium.PolyLine(locations=coordinates, weight=1)
site_map.add_child(lines)
```

Out[231]:

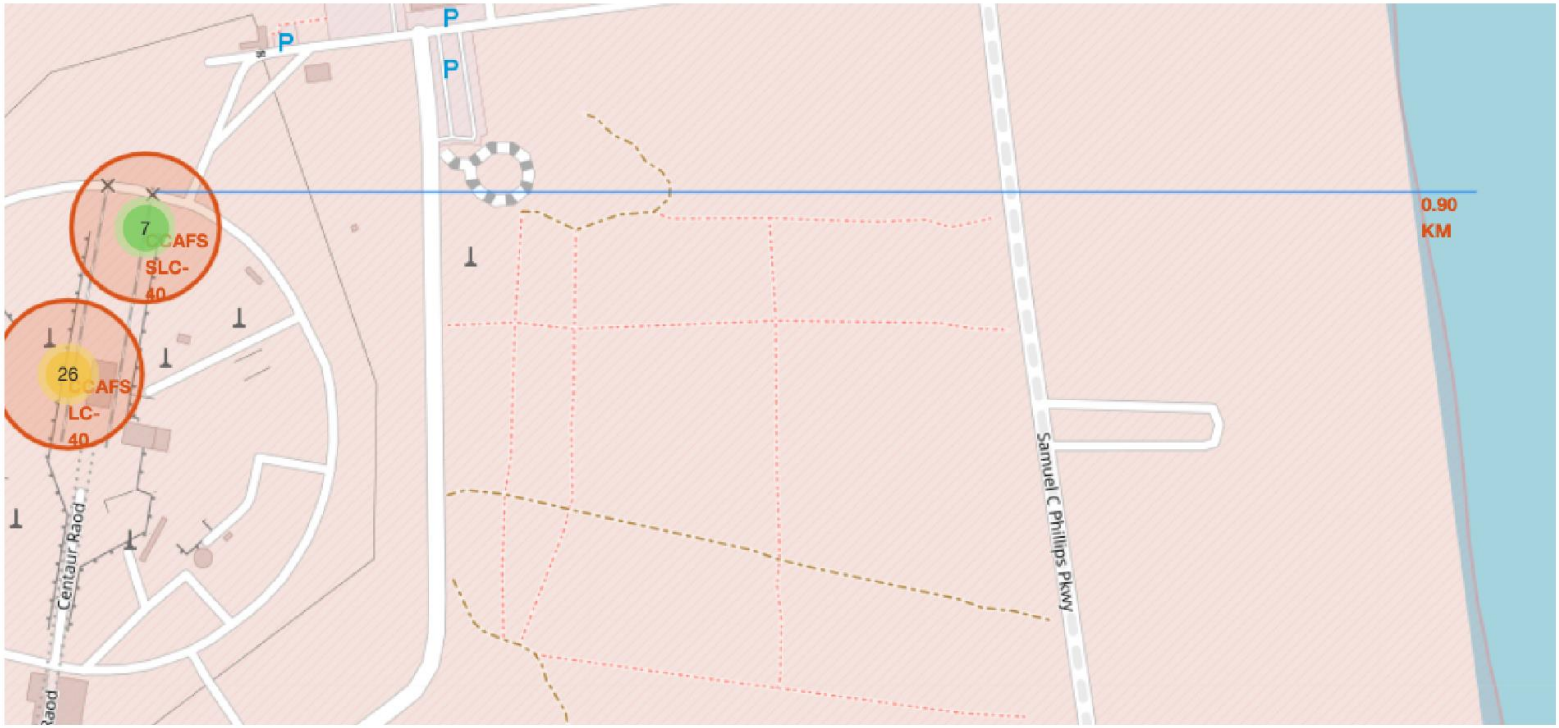


In [ ]:

```
#CCAFS_LC_coordinate = [28.562302, -80.577356]
#site_map = folium.Map(location=CCAFS_LC_coordinate, zoom_start=10)
#launch_site_lat = 28.562302
#launch_site_lon = -80.577356
#coastline_lat = 28.56334
#coastline_lon = -80.56799
#distance_coastline = calculate_distance(launch_site_lat, launch_site_lon, coastline_lat, coastline_lon)
#print(distance_coastline, 'km')
#-----------------------------------------------------
#KSC_LC_coordinate =  [28.573255, -80.646895]
#site_map = folium.Map(location=KSC_LC_coordinate, zoom_start=10)
#launch_site_lat = 28.573255
#launch_site_lon = -80.646895
#coastline_lat = 28.56334
#coastline_lon = -80.56799
#distance_coastline = calculate_distance(launch_site_lat, launch_site_lon, coastline_lat, coastline_lon)
#print(distance_coastline, 'km')
#-----------------------------------------------------
#VAFB_SLC_coordinate =  [34.632834, -120.610745]
#site_map = folium.Map(location=VAFB_SLC_coordinate, zoom_start=10)
#launch_site_lat = 34.632834
#launch_site_lon = -120.610745
#coastline_lat = 28.56334
#coastline_lon = -80.56799
#distance_coastline = calculate_distance(launch_site_lat, launch_site_lon, coastline_lat, coastline_lon)
#print(distance_coastline, 'km')
```

TODO: Draw a `PolyLine` between a launch site to the selected coastline point

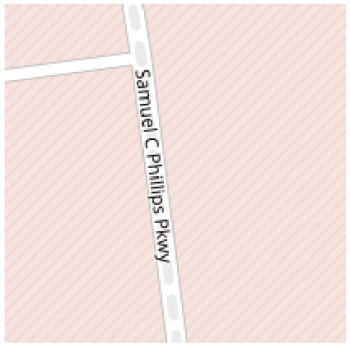
Your updated map with distance line should look like the following screenshot:



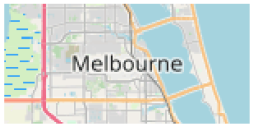
TODO: Similarly, you can draw a line betwee a launch site to its closest city, railway, highway, etc. You need to use `MousePosition` to find the their coordinates on the map first

A railway map symbol may look like this:

A highway map symbol may look like this:



A city map symbol may look like this:



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

```

In [232]:
```
closest_highway = 28.56335, -80.57085
closest_railroad = 28.57206, -80.58525
closest_city = 28.10473, -80.64531
```

In [233]:

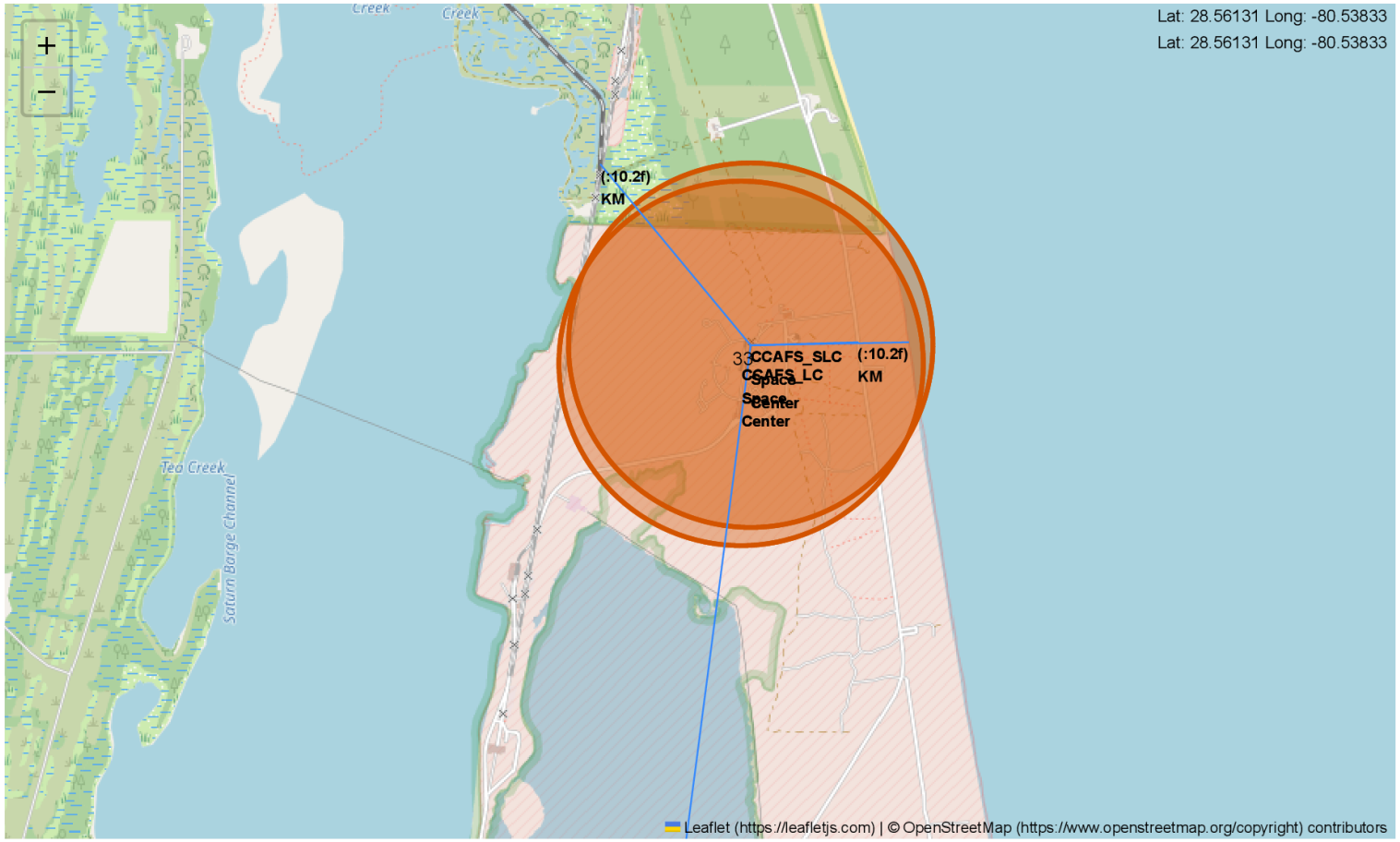
```
distance_highway = calculate_distance(launch_site_lat, launch_site_lon, closest_highway[0], closest_highway[1])
print('distance_highway', distance_highway, 'km')
distance_railroad = calculate_distance(launch_site_lat, launch_site_lon, closest_railroad[0], closest_railroad[1])
print('distance_railroad', distance_railroad, 'km')
distance_city = calculate_distance(launch_site_lat, launch_site_lon, closest_city[0], closest_city[1])
print('distance_city', distance_city, 'km')
```

```
distance_highway 0.5834695366934144 km
distance_railroad 1.2845344718142522 km
distance_city 51.43416999517233 km
```

In [234]:

```
#closesthighway marker
distance_marker =folium.Marker(
    closest_highway,
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % '(:10.2f) KM'.format(distance_highway)
        )
    )
site_map.add_child(distance_marker)
coordinates = [[launch_site_lat, launch_site_lon], closest_highway]
lines=folium.PolyLine(locations=coordinates, weight=1)
site_map.add_child(lines)
#--------------------------------------
distance_marker =folium.Marker(
    closest_railroad,
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % '(:10.2f) KM'.format(distance_railroad
        )
    )
site_map.add_child(distance_marker)
coordinates = [[launch_site_lat, launch_site_lon], closest_railroad]
lines=folium.PolyLine(locations=coordinates, weight=1)
site_map.add_child(lines)
#-------------------------------
distance_marker =folium.Marker(
    closest_city,
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % '(:10.2f) KM'.format(distance_city),
        )
    )
site_map.add_child(distance_marker)
coordinates = [[launch_site_lat, launch_site_lon], closest_city]
lines=folium.PolyLine(locations=coordinates, weight=1)
site_map.add_child(lines)
```

Out[234]:



After you plot distance lines to the proximities, you can answer the following questions easily:

- 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 (https://www.linkedin.com/in/pratiksha-verma-6487561b1/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork865-2022-01-01)](https://www.linkedin.com/in/pratiksha-verma-6487561b1/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork865-2022-01-01)

Change Log

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

(https://skills.network/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2022-01-01)

Space X Falcon 9 First Stage Landing Prediction

Assignment: Machine Learning Prediction

Estimated time needed: **60** minutes

Space X 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 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 space X for a rocket launch. In this lab, you will create a machine learning pipeline to predict if the first stage will land given the data from the preceding labs.

Several examples of an unsuccessful landing are shown here:

Most unsuccessful landings are planed. Space X; performs a controlled landing in the oceans.

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

We will import the following libraries for the lab

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 m
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a MatLab like plotting framework. We will use th
import matplotlib.pyplot as plt
#Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawin
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
# Allows us to test parameters of classification algorithms and find the best one
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
```

In [ ]:

```
## Load the dataframe
```

In [7]:

```
URL1 = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dat
```

In [9]:

```
data=pd.read_csv(URL1)
data.head(2)
```

Out[9]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedC
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	

In [10]:
```
URL2 = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dat
```

In [15]:
```
df=pd.read_csv(URL2)
df.head(2)
```

Out[15]:

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	...	Serial_B1058	Serial_B10
0	1.0	6104.959412	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0
1	2.0	525.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0

2 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']).

In [17]:
```
y=data['Class'].to_numpy()
y
```

Out[17]:
```
array([0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1,
       1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1], dtype=int64)
```

TASK 2

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

In [18]:
```
# students get this
transform = preprocessing.StandardScaler()
```

In [20]:
```
x = transform.fit_transform(df)
x
```

Out[20]:
```
array([[-1.71291154e+00, -1.94814463e-16, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       [-1.67441914e+00, -1.19523159e+00, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       [-1.63592675e+00, -1.16267307e+00, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       ...,
       [ 1.63592675e+00,  1.99100483e+00,  3.49060516e+00, ...,
         1.19684269e+00, -5.17306132e-01,  5.17306132e-01],
       [ 1.67441914e+00,  1.99100483e+00,  1.00389436e+00, ...,
         1.19684269e+00, -5.17306132e-01,  5.17306132e-01],
       [ 1.71291154e+00, -5.19213966e-01, -6.53912840e-01, ...,
        -8.35531692e-01, -5.17306132e-01,  5.17306132e-01]])
```

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

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
```

In [22]:
```
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.

In [23]:
```
1 Y_test.shape
```

Out[23]: (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` .

In [27]:
```
1 Lr=LogisticRegression()
```

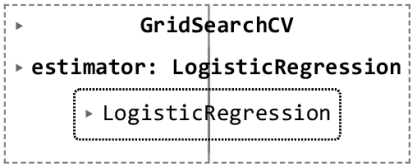
In [32]:
```
1 parameters ={'C':[0.01,0.1,1],
2              'penalty':['l2'],
3              'solver':['lbfgs']}
```

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

In [33]:
```
1 logreg_cv= GridSearchCV(Lr, parameters, cv=10)
```

In [34]:
```
1 logreg_cv.fit(X_train, Y_train)
```

Out[34]:



TASK 5

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

In [107]:
```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

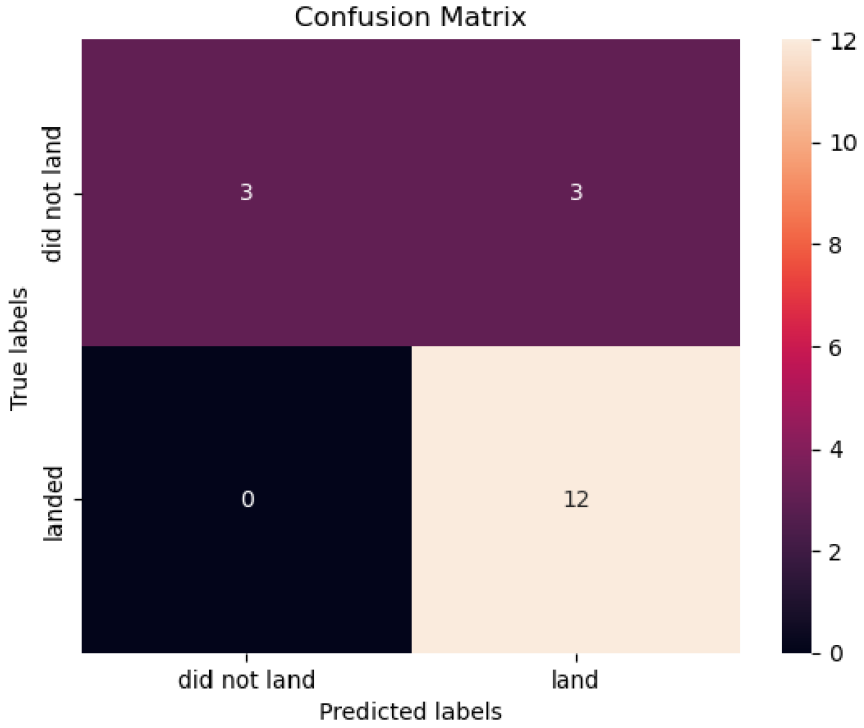
```
tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8464285714285713
```

In [108]:
```
logreg_cv.score(X_test, Y_test)
```

Out[108]: 0.8333333333333334

Lets look at the confusion matrix:

In [109]:
```
yhatlog=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhatlog)
```



In [110]:
```
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()
```

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` .

In [111]:
```
Xs_train, Xs_test, Ys_train, Ys_test= train_test_split(x, y, test_size=0.2, random_state=2)
```

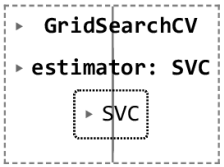
In [112]:
```
svm = SVC()
```

In [113]:
```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}

```

In [114]:
```
svm_cv= GridSearchCV(svm, parameters, cv=10)
```

In [115]:
```
1 svm_cv.fit(Xs_train, Ys_train)
```

Out[115]:



In [121]:
```
1 print("tuned hpyerparameters :(best parameters) ",svm_cv.best_params_)
2 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` :

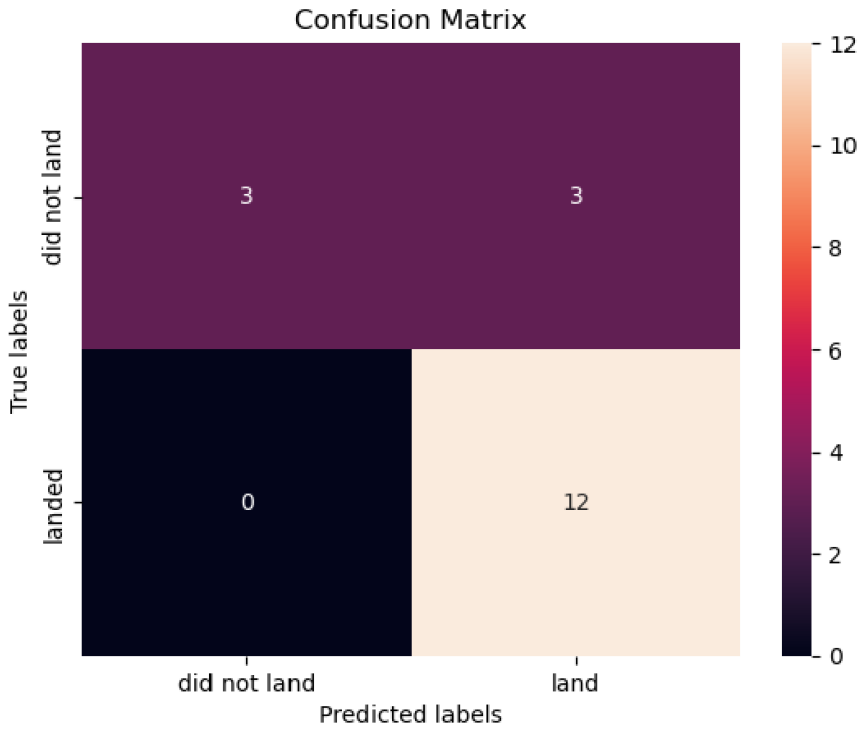
In [122]:
```
1 svm_cv.score(Xs_test, Ys_test)
```

Out[122]:
```
0.8333333333333334
```

We can plot the confusion matrix

In [123]:
```
1 yhatsvm=svm_cv.predict(Xs_test)
```

In [124]:
```
1 plot_confusion_matrix(Ys_test,yhatsvm)
```



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` .

In [126]:
```
1 Xt_train, Xt_test, Yt_train, Yt_test= train_test_split(x, y, test_size=0.2, random_state=12)
```

In [127]:
```
tree = DecisionTreeClassifier()
```

In [128]:
```
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()
```

In [130]:
```
tree_cv= GridSearchCV(tree, parameters, cv=10)
```

In [131]:
```
tree_cv.fit(Xt_train, Yt_train)
```

```
C:\Users\parichea\AppData\Local\anaconda3\Lib\site-packages\sklearn\model_selection\_validation.py:425: FitFailedWa
rning:
3240 fits failed out of a total of 6480.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
3240 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\parichea\AppData\Local\anaconda3\Lib\site-packages\sklearn\model_selection\_validation.py", line 7
32, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\parichea\AppData\Local\anaconda3\Lib\site-packages\sklearn\base.py", line 1144, in wrapper
    estimator._validate_params()
  File "C:\Users\parichea\AppData\Local\anaconda3\Lib\site-packages\sklearn\base.py", line 637, in _validate_params
    validate_parameter_constraints(
  File "C:\Users\parichea\AppData\Local\anaconda3\Lib\site-packages\sklearn\utils\_param_validation.py", line 95, i
n validate_parameter_constraints
```

In [132]:
```
print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', 'max_depth': 6, 'max_features': 'sqrt', 'min_sampl
es_leaf': 1, 'min_samples_split': 10, 'splitter': 'random'}
accuracy : 0.9017857142857142
```

TASK 9

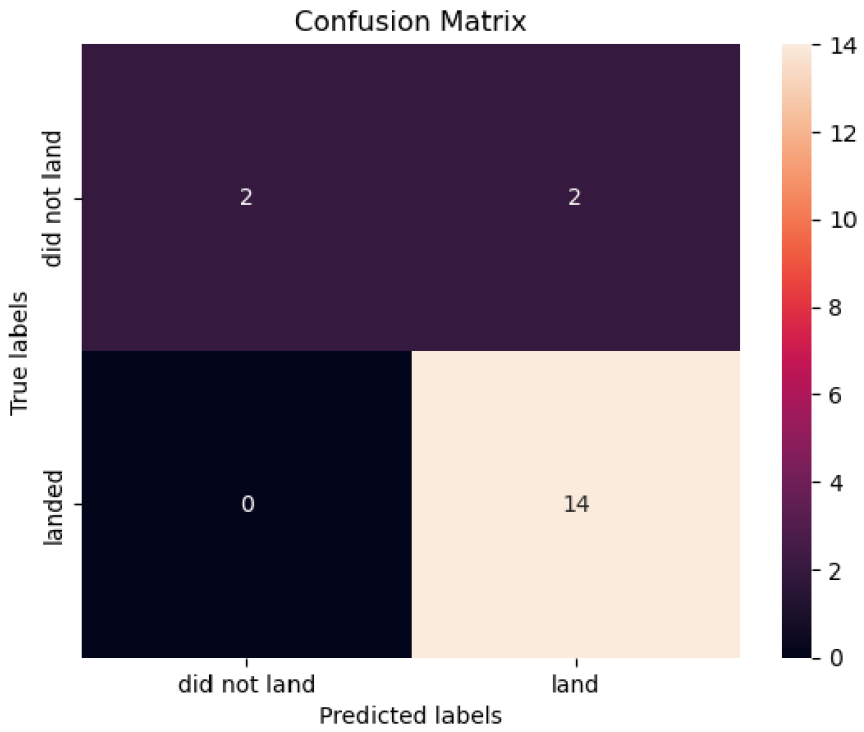
Calculate the accuracy of tree_cv on the test data using the method `score` :

In [133]:
```
1 tree_cv.score(Xt_train, Yt_train)
```

Out[133]: 0.875

We can plot the confusion matrix

In [136]:
```
yhattree = tree_cv.predict(Xt_test)
plot_confusion_matrix(Yt_test,yhattree)
```



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` .

In [137]:
```
Xk_train, Xk_test, Yk_train, Yk_test= train_test_split(x, y, test_size=0.2, random_state=12)
```

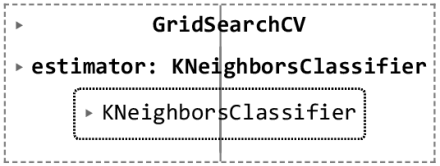
In [138]:
```
KNN = KNeighborsClassifier()
```

In [141]:
```
parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'p': [1,2]}


```

In [142]:
```
knn_cv=GridSearchCV(KNN, parameters, cv=10)
```

In [143]:
```
knn_cv.fit(Xk_train, Yk_train)
```

Out[143]:



In [144]:
```
print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
print("accuracy :",knn_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 3, 'p': 1}
accuracy : 0.8446428571428571
```

TASK 11

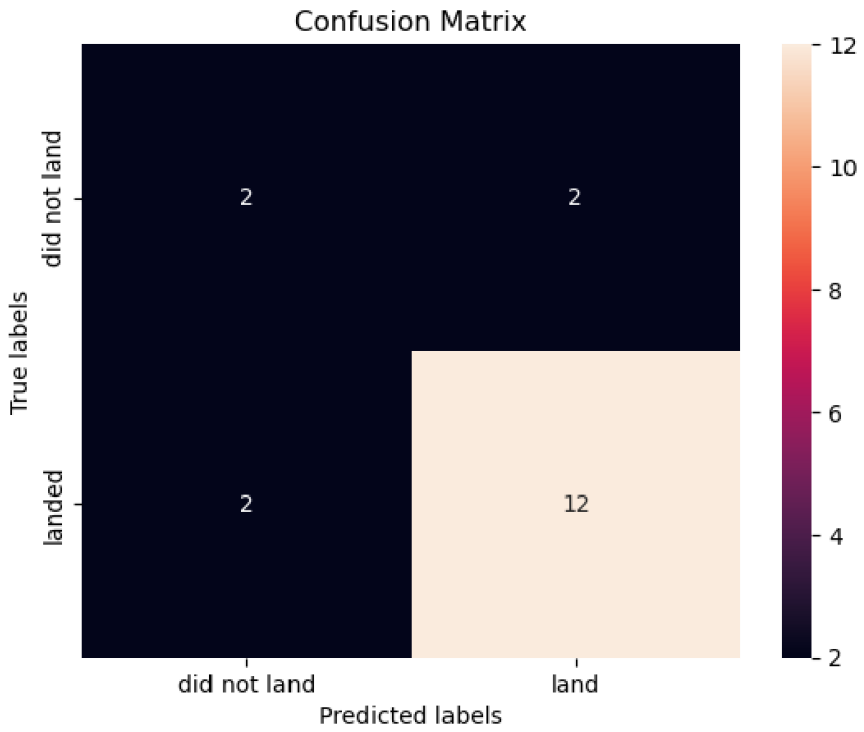
Calculate the accuracy of knn_cv on the test data using the method `score` :

In [145]:
```
knn_cv.score(Xk_test, Yk_test)
```

Out[145]:
```
0.7777777777777778
```

We can plot the confusion matrix

In [146]:
```
1 yhatknn = knn_cv.predict(Xk_test)
2 plot_confusion_matrix(Yk_test,yhatknn)
```



TASK 12

Find the method performs best:

In [147]:
```
1 accu= []
2 Method=[]
```

In [149]:

```
accu.append(logreg_cv.score(X_test, Y_test))
Method.append('logistic regression')
#-----------------------------------
accu.append(svm_cv.score(Xs_test, Ys_test))
Method.append('SVM')
#-----------------------------------
accu.append(tree_cv.score(Xt_test, Yt_test))
Method.append('Decision Tree')
#-----------------------------------
accu.append(knn_cv.score(Xk_test, Yk_test))
Method.append('KNN')
```

In [150]:

```
print(accu)
print(Method)
```

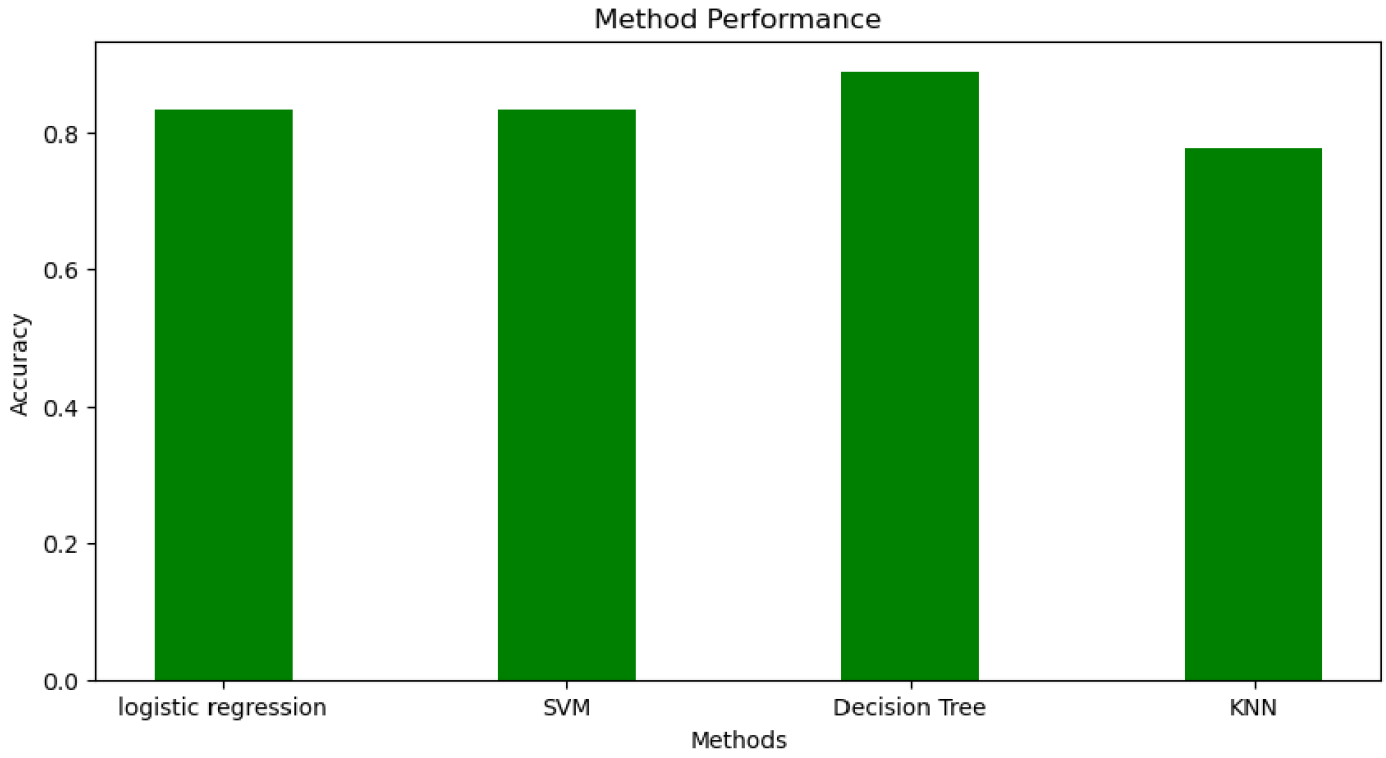
```
[0.8333333333333334, 0.8333333333333334, 0.8333333333333334, 0.8888888888888888, 0.7777777777777778]
['logistic regression', 'logistic regression', 'SVM', 'Decision Tree', 'KNN']
```

In [151]:

```
import matplotlib.pyplot as plt
```

In [155]:

```
fig=plt.figure(figsize=(10,5))
plt.bar(Method, accu, width=0.4, color='green')
plt.xlabel('Methods')
plt.ylabel('Accuracy')
plt.title('Method Performance')
plt.show()
```



Authors

Pratiksha Verma (https://www.linkedin.com/in/pratiksha-verma-6487561b1/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-

Change Log

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