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 [1]:

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

In [2]:

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

In [ ]:

```
## Exploratory Data Analysis
```

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

In [58]:

```
from js import fetch
import io

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

Out[58]:

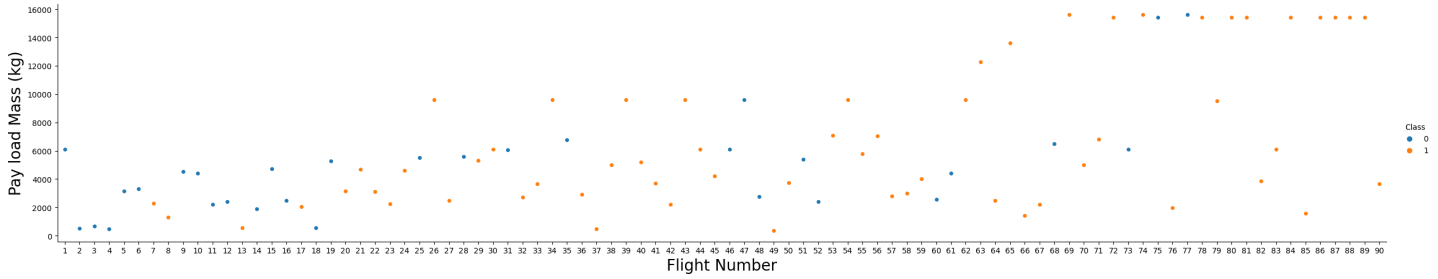
	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	Landing
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	
2	3	2013-03-01	Falcon 9	677.000000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

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 [4]:

```
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%.

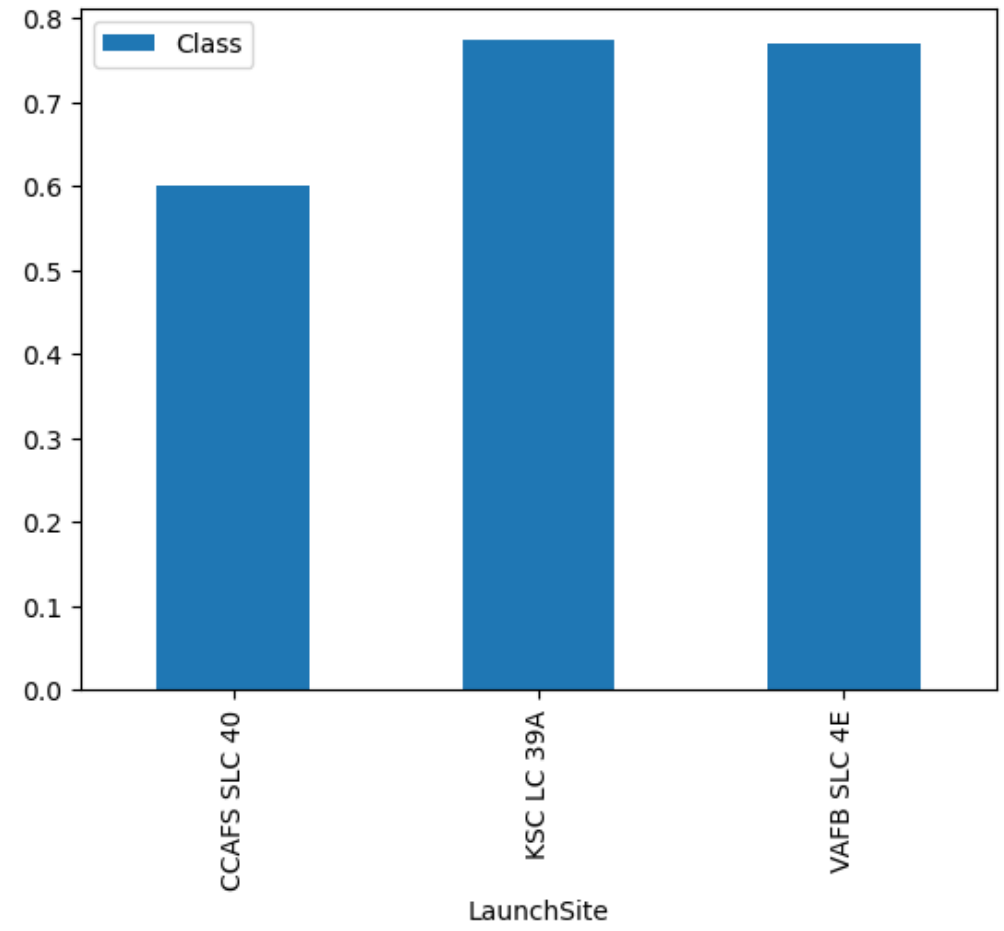
Next, let's drill down to each site visualize its detailed launch records.

In [17]:

```
### TASK 1: Visualize the relationship between Flight Number and Launch Site
df_=df.groupby("LaunchSite")['Class'].mean().reset_index()
df_.plot(kind="bar",x="LaunchSite", y="Class")
```

Out[17]:

```
<AxesSubplot:xlabel='LaunchSite'>
```



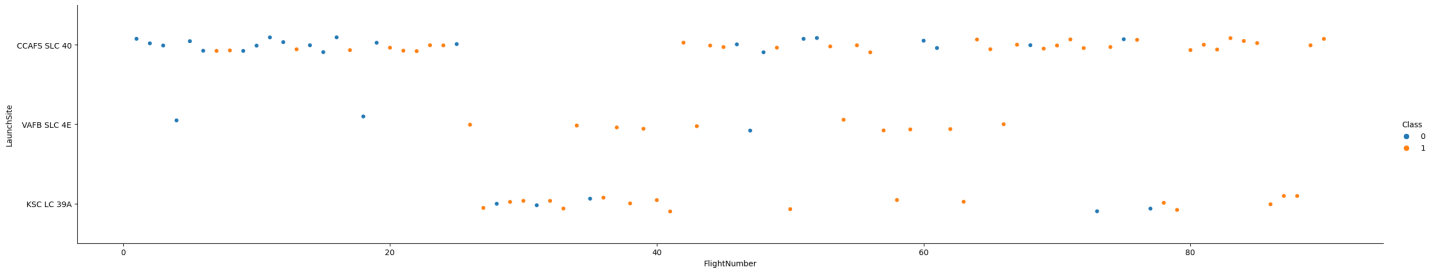
Use the function `catplot` to plot `FlightNumber` vs `LaunchSite`, set the parameter `x` parameter to `FlightNumber`,set the `y` to `Launch Site` and set the parameter `hue` to `'class'`

In [21]:

```
# Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to be the class value
sns.catplot(x="FlightNumber", y="LaunchSite", hue="Class", data=df , aspect=5)
```

Out[21]:

```
<seaborn.axisgrid.FacetGrid at 0x4926588>
```



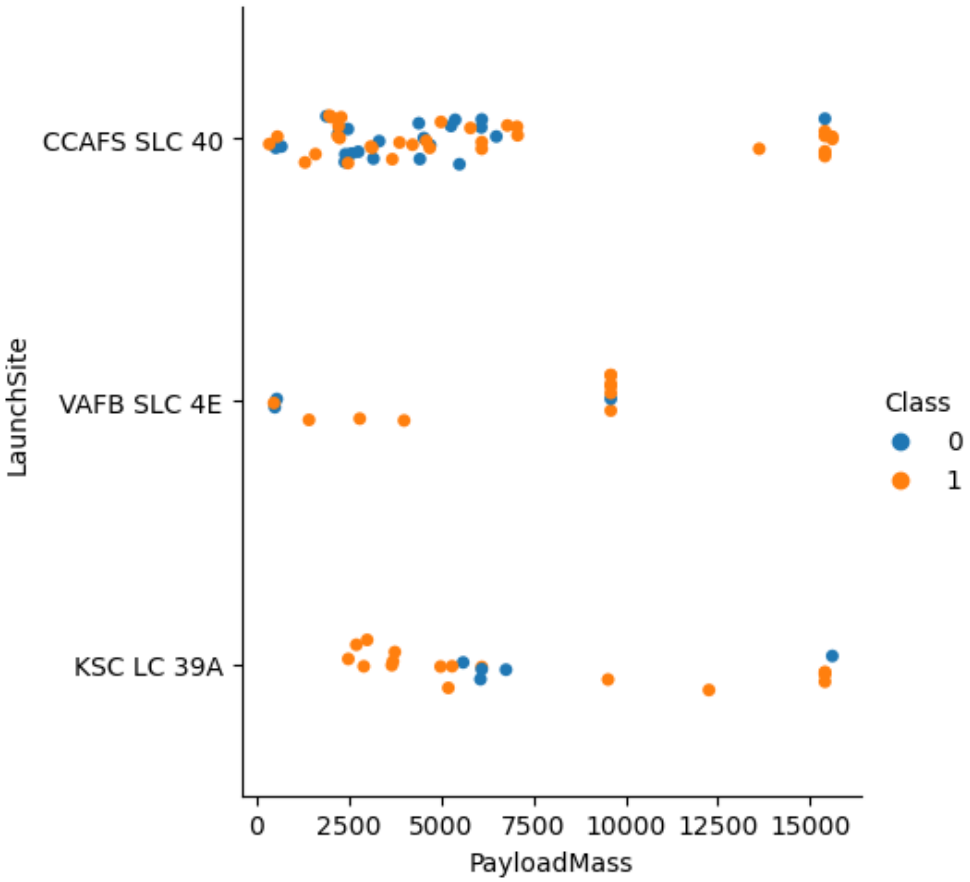
Now try to explain the patterns you found in the Flight Number vs. Launch Site scatter point plots.

In [25]:

```
### TASK 2: Visualize the relationship between Payload and Launch Site
sns.catplot(x="PayloadMass",y="LaunchSite",data=df, hue="Class")
```

Out[25]:

```
<seaborn.axisgrid.FacetGrid at 0x738f830>
```



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

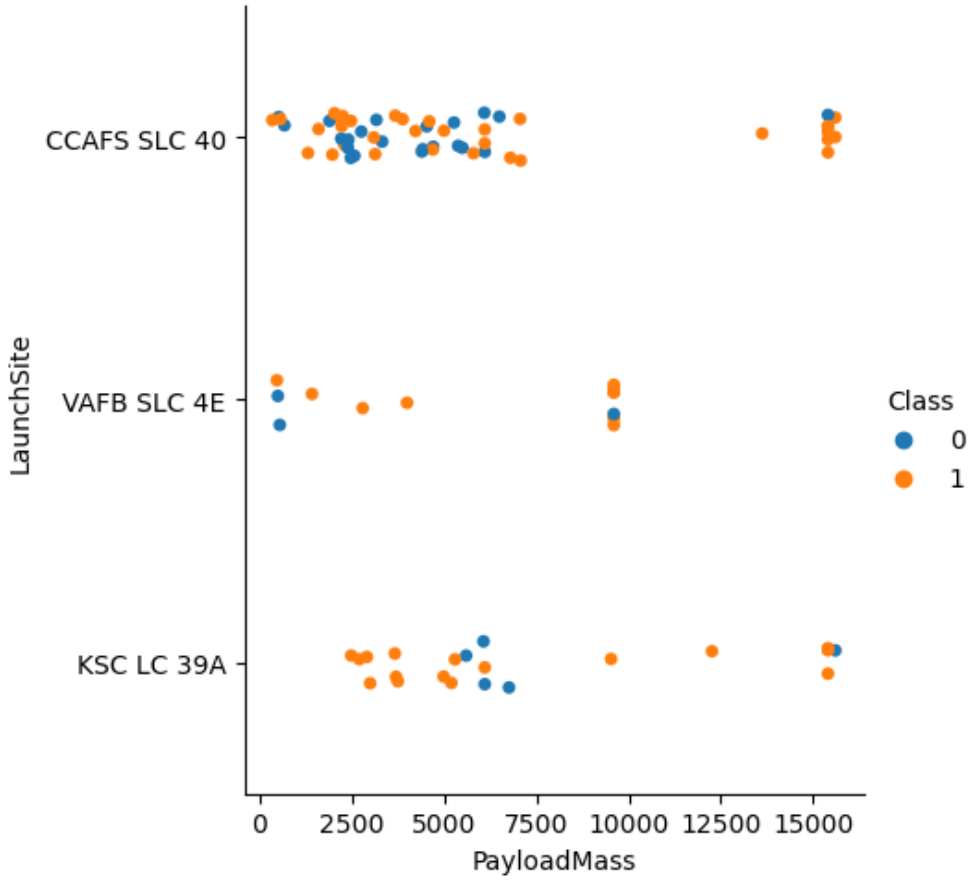
In [26]:

```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the la
```

```
unch site, and hue to be the class value
sns.catplot(x="PayloadMass",y="LaunchSite",data=df, hue="Class")
```

Out[26]:

```
<seaborn.axisgrid.FacetGrid at 0x4d3e998>
```



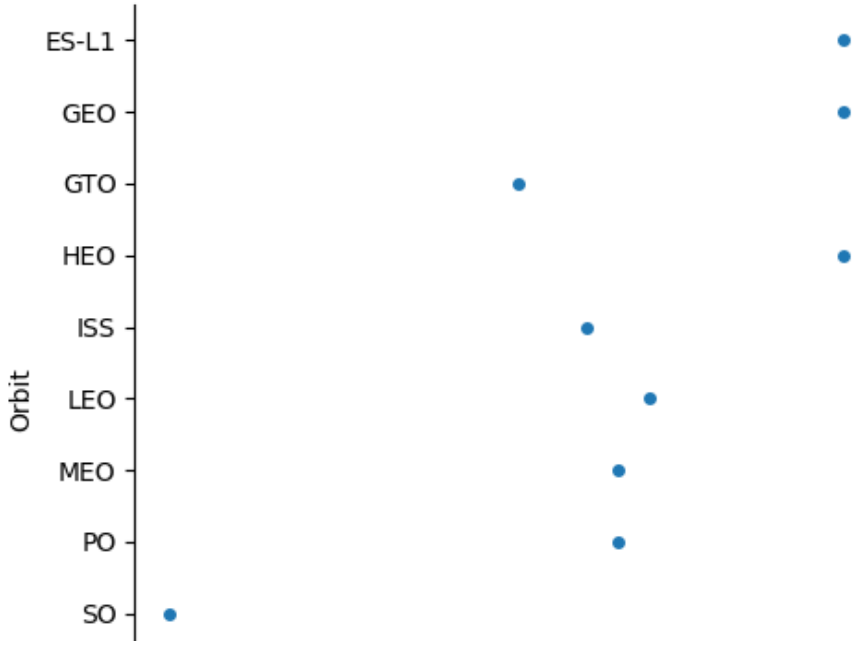
Now if you observe Payload Vs. Launch Site scatter point chart you will find for the VAFB-SLC launchsite there are no rockets launched for heavypayload mass(greater than 10000).

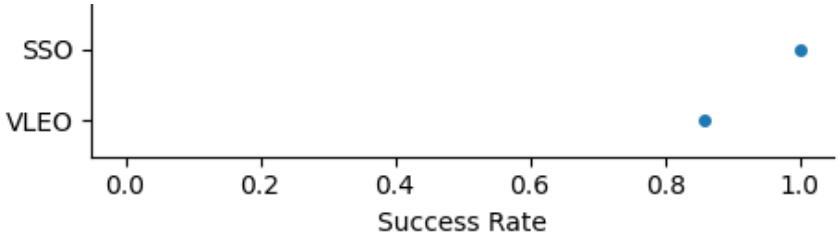
In [38]:

```
### TASK  3: Visualize the relationship between success rate of each orbit type
df0=df.groupby("Orbit")["Class"].mean().reset_index()
sns.catplot(x="Class",y="Orbit",data=df0)
plt.xlabel("Success Rate")
```

Out[38]:

```
Text(0.5, 9.444444444444438, 'Success Rate')
```





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
```

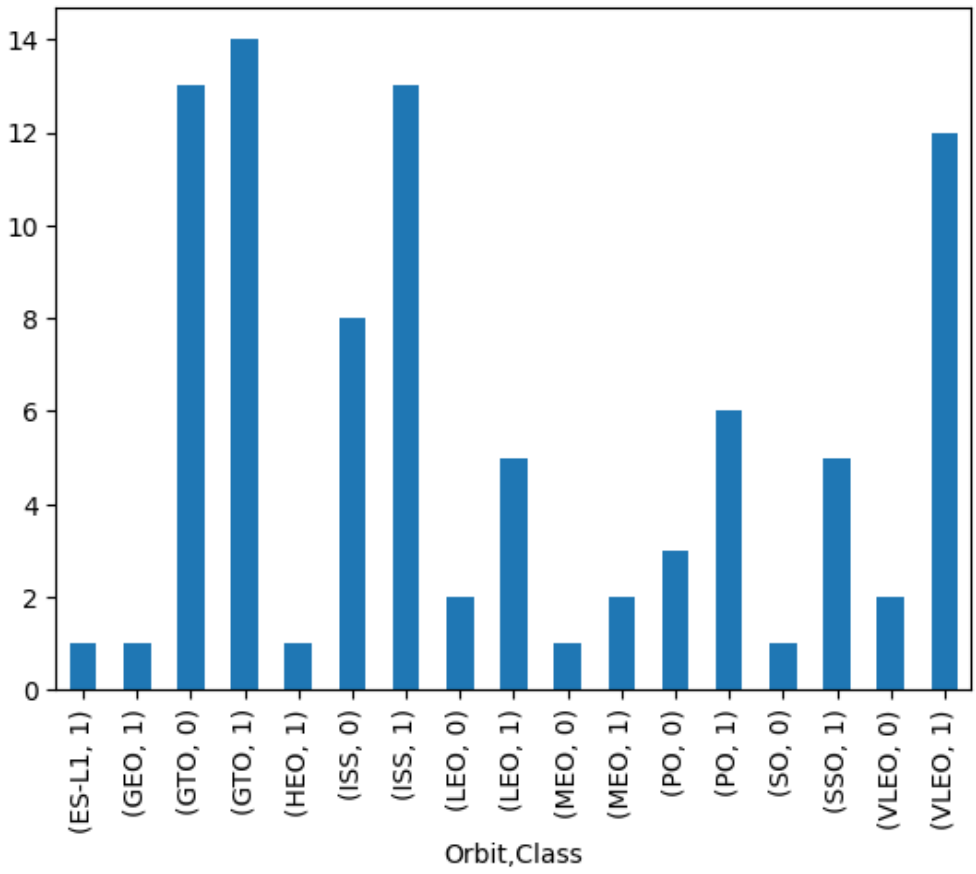
Analyze the ploted bar chart try to find which orbits have high sucess rate.

In [55]:

```
### TASK  4: Visualize the relationship between FlightNumber and Orbit type
df1=df.groupby(["Orbit","Class"])['FlightNumber'].count()
#df1.columns=['Orbit',"Flight Counts"]

df1.plot(kind="bar", x="Orbit", y="FlightNumber" )
```

Out[55]:

```
<AxesSubplot:xlabel='Orbit,Class'>
```



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

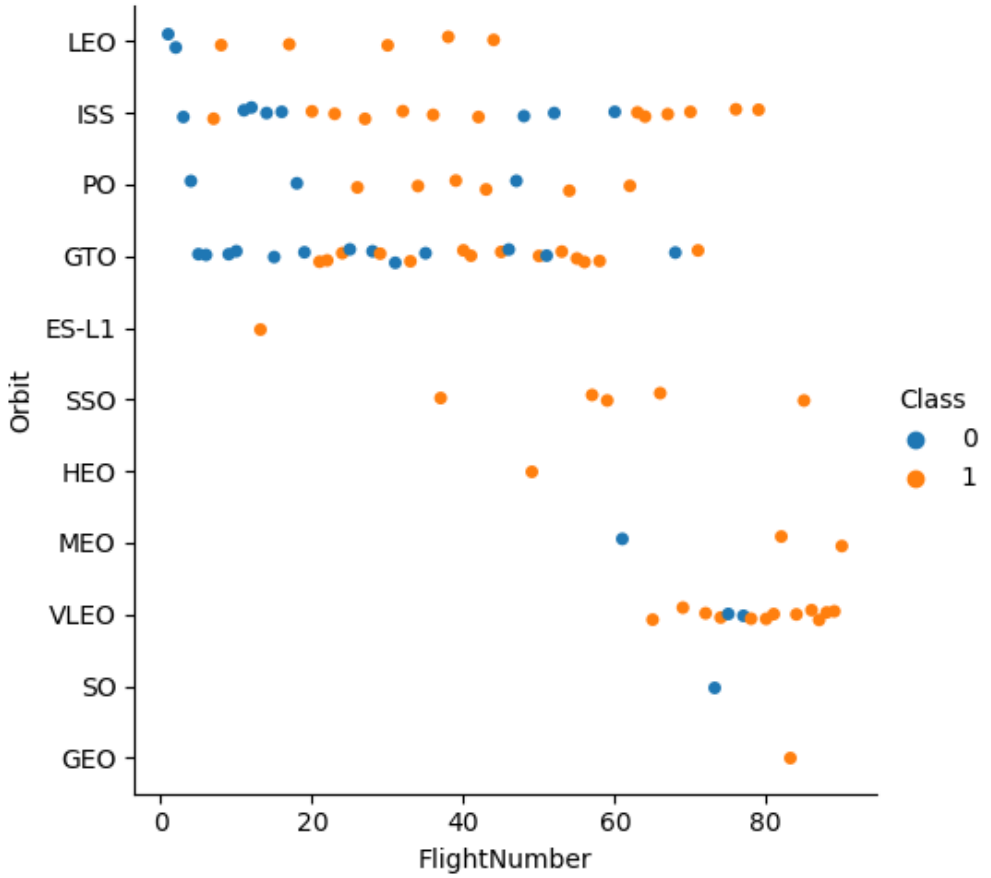
In [56]:

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

Out[56]:

Out[50]:

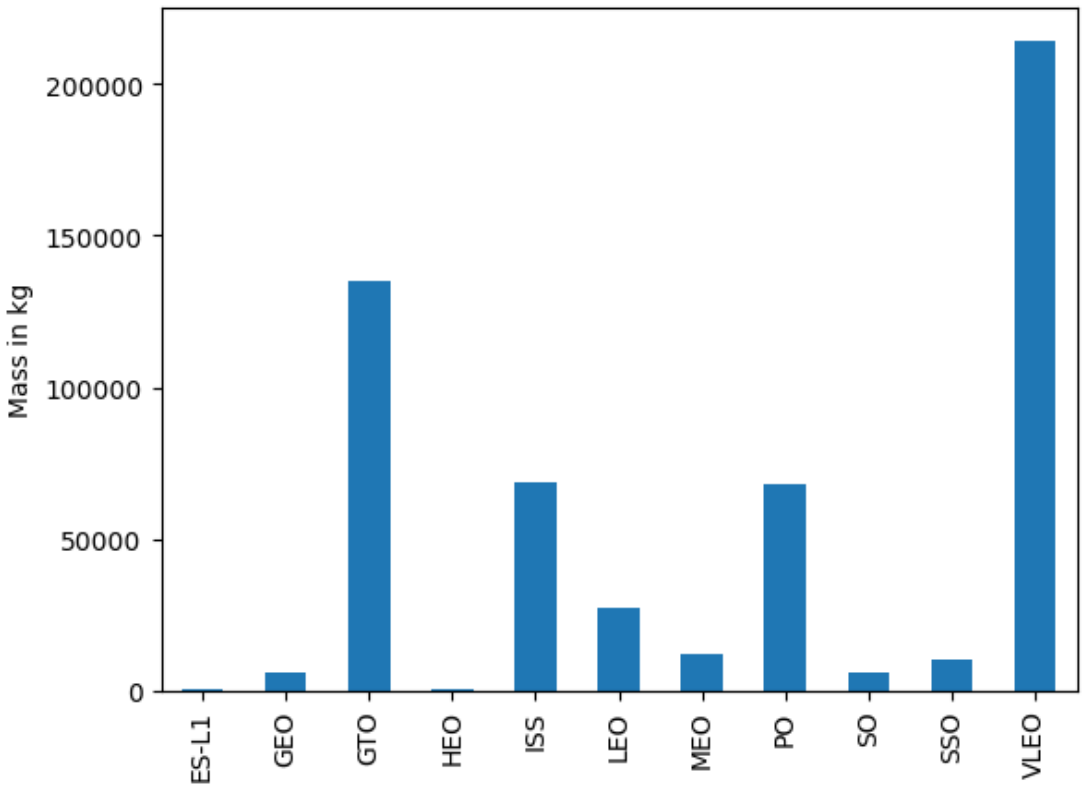
<seaborn.axisgrid.FacetGrid at 0xa02a5e8>



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 [64]:

```
### TASK  5: Visualize the relationship between Payload and Orbit type
df1=df.groupby("Orbit")["PayloadMass"].sum()
df1.plot.bar()
plt.xlabel("Orbits")
plt.ylabel("Mass in kg")
plt.show()
```



Orbits

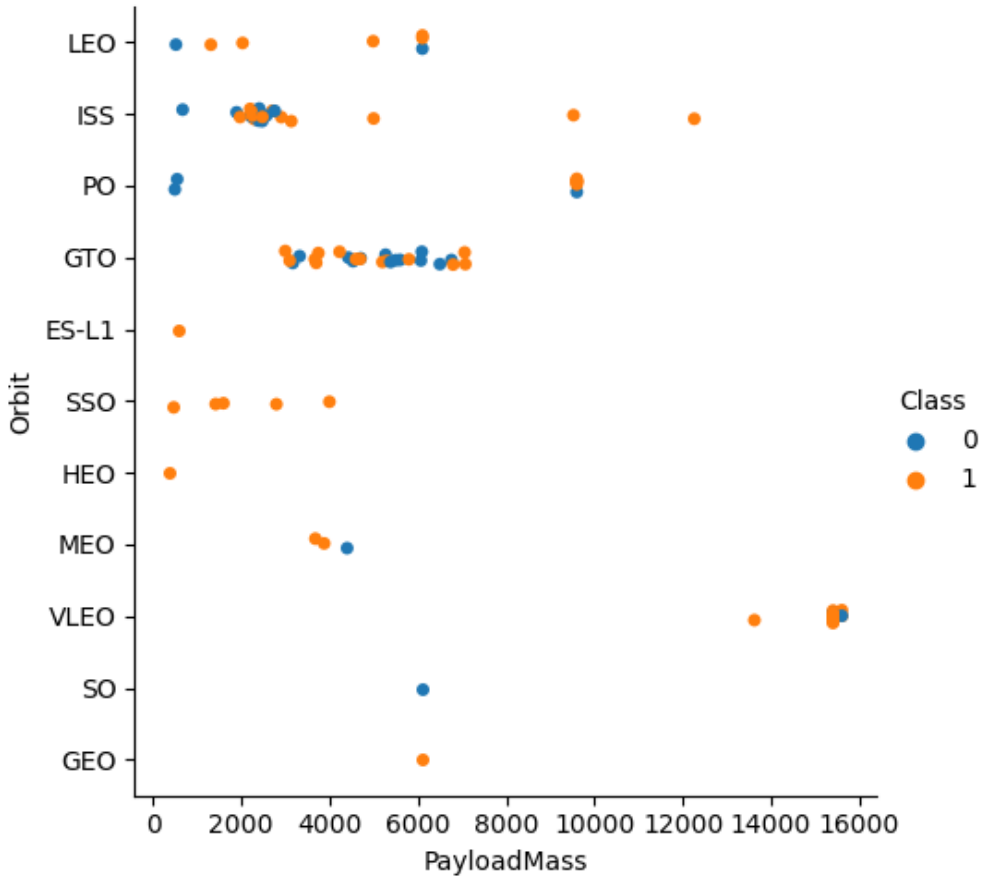
Similarly, we can plot the Payload vs. Orbit scatter point charts to reveal the relationship between Payload and Orbit type

In [66]:

```
# Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, and hu
e to be the class value
sns.catplot(x="PayloadMass", y="Orbit", data=df, hue="Class")
```

Out[66]:

```
<seaborn.axisgrid.FacetGrid at 0x8b4ec40>
```



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:

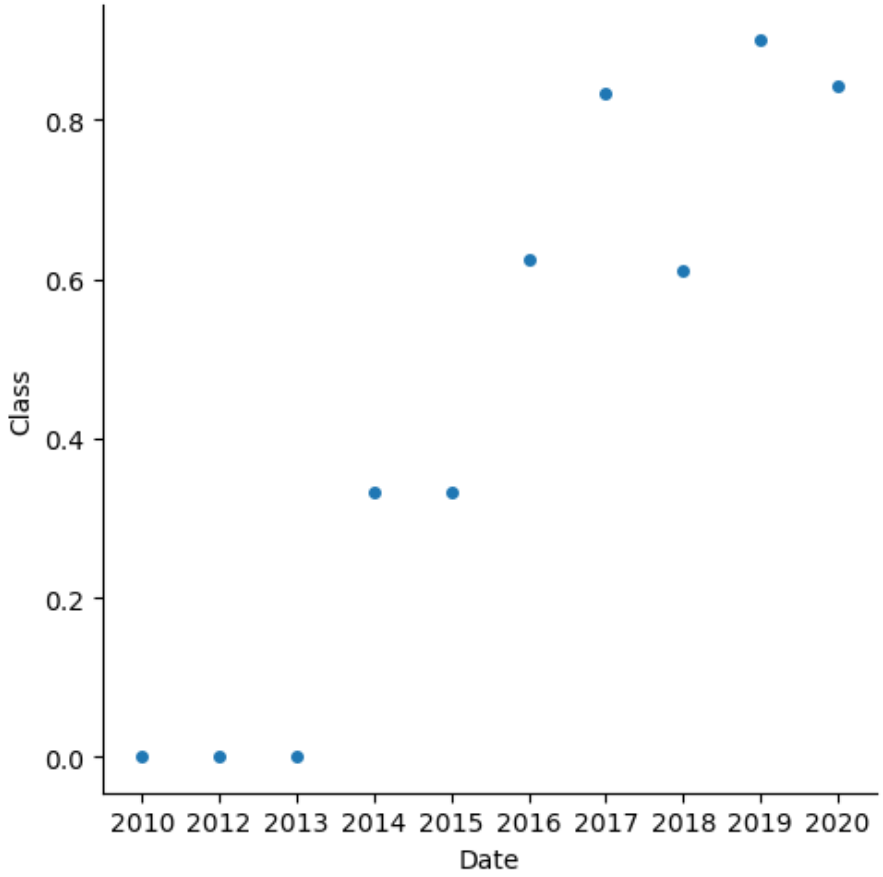
In [69]:

```
# 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()"""
df1=df.groupby('Date')["Class"].mean().reset_index()
sns.catplot(x="Date", y="Class", data=df1)
```

Out[69]:

```
<seaborn.axisgrid.FacetGrid at 0xba82b80>
```

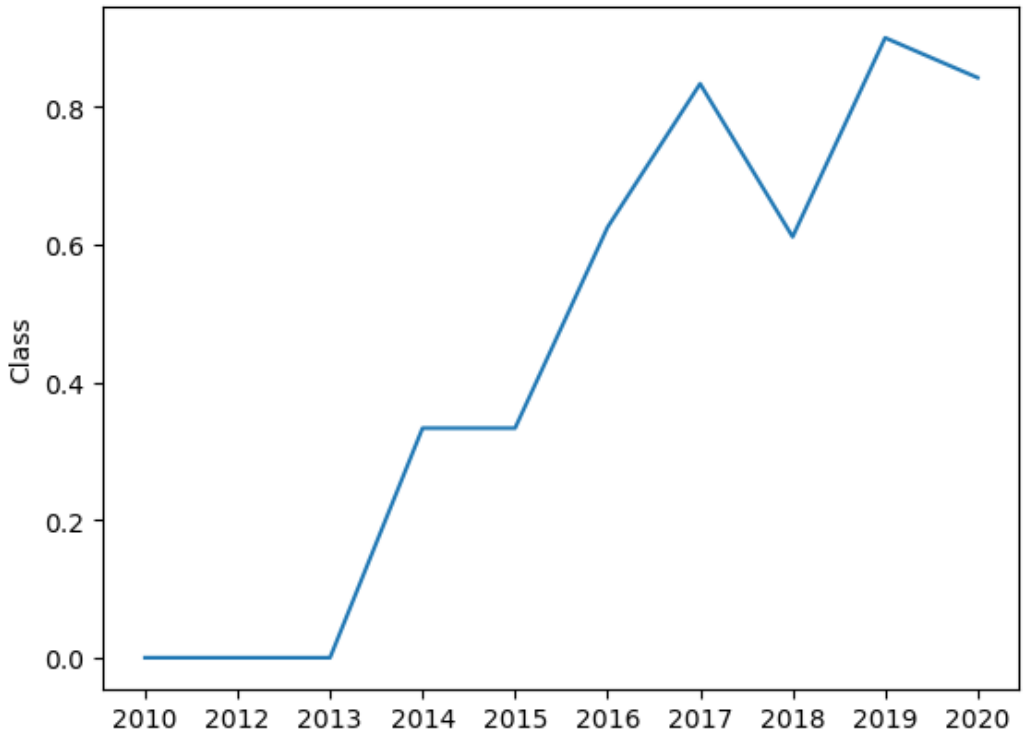


In [70]:

```
# Plot a line chart with x axis to be the extracted year and y axis to be the success rat
e
sns.lineplot(x="Date", y="Class", data=df1)
```

Out[70]:

```
<AxesSubplot:xlabel='Date', ylabel='Class'>
```



Date

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 [71]:

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

Out[71]:

	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 [80]:

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

Out[80]:

	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	Orbit_LEO	Orbit_MEO	Orbit_PO	Orbit_SO	Orbit_SSO	...	Serial
0	0	0	0	0	0	1	0	0	0	0	...	
1	0	0	0	0	0	1	0	0	0	0	...	
2	0	0	0	0	1	0	0	0	0	0	...	
3	0	0	0	0	0	0	0	1	0	0	...	
4	0	0	1	0	0	0	0	0	0	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
85	0	0	0	0	0	0	0	0	0	0	...	
86	0	0	0	0	0	0	0	0	0	0	...	
87	0	0	0	0	0	0	0	0	0	0	...	
88	0	0	0	0	0	0	0	0	0	0	...	
89	0	0	0	0	0	0	1	0	0	0	...	

90 rows × 72 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 [85]:

```
### TASK  8: Cast all numeric columns to `float64
features_one_hot.dtypes
features_one_hot.iloc[:,:].astype(float)
```

Out[85]:

	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	Orbit_LEO	Orbit_MEO	Orbit_PO	Orbit_SO	Orbit_SSO	...	Serial
0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	
1	0.0	0.0	0.0	0.0	0.0	1.0	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	...	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	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	...	
...	...	...	...	...	...	...	...	...	...	...	...	
85	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	...	
87	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	...	
89	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	

90 rows × 72 columns

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

In [86]:

```
# HINT: use astype function
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]

Change Log

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

Assignment: SQL Notebook for Peer Assignment

Estimated time needed: 60 minutes.

Introduction

Using this Python notebook you will:

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

Overview of the DataSet

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

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

Therefore if we can determine if the first stage will land, we can determine the cost of a launch.

This information can be used if an alternate company wants to bid against SpaceX for a rocket launch.

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

Download the datasets

This assignment requires you to load the spacex dataset.

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

[Spacex DataSet]

In [1]:

```
!pip install sqlalchemy==1.3.9
```

```
Collecting sqlalchemy==1.3.9
  Downloading SQLAlchemy-1.3.9.tar.gz (6.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.0/6.0 MB 77.8 MB/s eta 0:00:00:00:010:01
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: sqlalchemy
  Building wheel for sqlalchemy (setup.py) ... done
  Created wheel for sqlalchemy: filename=SQLAlchemy-1.3.9-cp37-cp37m-linux_x86_64.whl siz
e=1159121 sha256=f9ca22efb58f98025f1a169cd85435eb8ea830ac2b407aa814278144b52844b4
  Stored in directory: /home/jupyterlab/.cache/pip/wheels/03/71/13/010faf12246f72dc76b415
0e6e599d13a85b4435e06fb9e51f
Successfully built sqlalchemy
Installing collected packages: sqlalchemy
  Attempting uninstall: sqlalchemy
    Found existing installation: SQLAlchemy 1.3.24
    Uninstalling SQLAlchemy-1.3.24:
      Successfully uninstalled SQLAlchemy-1.3.24
Successfully installed sqlalchemy-1.3.9
```

Connect to the database

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

In [2]:

```
%load_ext sql
```

In [3]:

```
import csv, sqlite3

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

In [4]:

```
!pip install -q pandas==1.1.5
```

In [5]:

```
%sql sqlite:///my_data1.db
```

Out[5]:

```
'Connected: @my_data1.db'
```

In [6]:

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

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/pandas/core/generic.py:2882: UserWarning: The spaces in these column names will not be changed. In pandas versions < 0.14, spaces were converted to underscores.
  both result in 0.1234 being formatted as 0.12.
```

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

In [7]:

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

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

Out[7]:

```
[]
```

Tasks

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

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

Task 1

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

In [9]:

```
%sql select distinct Launch_Site from SpaceXTBL
```

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

Out[9]:

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

Task 2

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

In [11]:

```
%sql select * from SpaceXTBL where Launch_Site LIKE 'CCA%' Limit 5
```

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

Out[11]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Land
2010-04-06	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	
2010-08-12	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	
2012-05-22	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	
2012-08-10	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	
2013-01-03	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	

Task 3

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

In [17]:

```
%sql select SUM(PAYLOAD_MASS__KG_) AS TOTAL_MASS_Kg FROM SpaceXTBL Where Customer='NASA (CRS)'
```

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

Out[17]:

TOTAL_MASS_Kg
45596

Task 4

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

In [19]:

```
%sql select AVG(PAYLOAD_MASS__KG_) AS TOTAL_MASS_Kg FROM SpaceXTBL where Booster_version
like 'F9 v1.1%'
```

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

Out[19]:

TOTAL_MASS_Kg
2534.6666666666665

Task 5

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

Hint:Use min function

In [25]:

```
%sql select * from SpaceXTBL where Landing_Outcome='Success (ground pad)' order by Date,
"Time(UTC)" limit 5
```

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

Out[25]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landin
2015-12-22	01:29:00	F9 FT B1019	CCAFS LC-40	OG2 Mission 2 11 Orbcomm-OG2 satellites	2034	LEO	Orbcomm	Success	Succ
2016-07-18	04:45:00	F9 FT B1025.1	CCAFS LC-40	SpaceX CRS-9	2257	LEO (ISS)	NASA (CRS)	Success	Succ
2017-01-05	11:15:00	F9 FT B1032.1	KSC LC-39A	NROL-76	5300	LEO	NRO	Success	Succ
2017-02-19	14:39:00	F9 FT B1031.1	KSC LC-39A	SpaceX CRS-10	2490	LEO (ISS)	NASA (CRS)	Success	Succ
2017-03-06	21:07:00	F9 FT B1035.1	KSC LC-39A	SpaceX CRS-11	2708	LEO (ISS)	NASA (CRS)	Success	Succ

Task 6

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

In [26]:

```
%sql select * from SpaceXTBL where Landing_Outcome='Success (drone ship)' and PAYLOAD_MAS
S__KG_ > 4000 and PAYLOAD_MASS__KG_ < 6000
```

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

Out[26]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_
2016-06-05	05:21:00	F9 FT B1022	CCAFS LC-40	JCSAT-14	4696	GTO	SKY Perfect JSAT Group	Success	Succ
2016-08-14	05:26:00	F9 FT B1026	CCAFS LC-40	JCSAT-16	4600	GTO	SKY Perfect JSAT Group	Success	Succ
2017-03-30	22:27:00	F9 FT B1021.2	KSC LC-39A	SES-10	5300	GTO	SES	Success	Succ
2017-11-10	22:53:00	F9 FT B1031.2	KSC LC-39A	SES-11 / EchoStar 105	5200	GTO	SES EchoStar	Success	Succ

Task 7

List the total number of successful and failure mission outcomes

In [28]:

```
%sql select Mission_Outcome,count(Mission_Outcome) from SpaceXTBL group by Mission_Outcom
e
```

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

Out[28]:

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

Task 8

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

In [33]:

```
%sql select Booster_Version,max(PAYLOAD_MASS__KG_) as Maximum from SpaceXTBL group by Bo
oster_Version order by Maximum desc limit 2
```

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

Out[33]:

Booster_Version	Maximum
F9 B5 B1060.3	15600
F9 B5 B1060.2	15600

Task 9

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

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

In [41]:

```
%sql select substr(Date,6,2) as month, Landing_Outcome from SpaceXTBL where substr(Date,
1,4)='2015' and Landing_Outcome='Failure (drone ship)'
```

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

Out[41]:

month	Landing_Outcome
10	Failure (drone ship)
04	Failure (drone ship)

Task 10

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

In [46]:

```
%sql select count(*) as Number, Landing_Outcome, Date from SpaceXTBL group by Landing_Ou
tcome having Date>="2010-06-04" and Date<="2017-03-2" order by Number  desc
```

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

Out[46]:

Number	Landing_Outcome	Date
21	No attempt	2012-05-22
14	Success (drone ship)	2016-08-04
9	Success (ground pad)	2015-12-22
5	Failure (drone ship)	2015-10-01
5	Controlled (ocean)	2014-04-18
2	Uncontrolled (ocean)	2013-09-29
1	Precluded (drone ship)	2015-06-28

Reference Links

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

Author(s)

Lakshmi Holla

Other Contributors

Rav Ahuja

Change log

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

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 manip
ulation and analysis.
import pandas as pd
# NumPy is a library for the Python programming language, adding support for large, multi
-dimensional arrays and matrices, along with a large collection of high-level mathematica
l functions to operate on these arrays
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 [2]:

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

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

In [3]:

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

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

In [4]:

```
# Takes the dataset and uses the payloads column to call the API and append the data to t
he 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 [5]:

```
# Takes the dataset and uses the cores column to call the API and append the data to the
lists
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cores/"+core['cor
e']).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 [6]:

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

In [7]:

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

Check the content of the response

In [10]:

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

```
<class 'bytes'>
```

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 [49]:

```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-D
```

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

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

In [50]:

```
response=requests.get(static_json_url)
response.status_code
```

Out[50]:

```
200
```

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

In [54]:

```
# Use json_normalize meethod to convert the json result into a dataframe
data0=pd.json_normalize(response.json())
data=pd.DataFrame(data0)
data.head()
```

Out[54]:

	static_fire_date_utc	static_fire_date_unix	tbd	net	window	rocket	success	details	crew	sl
0	2006-03-17T00:00:00.000Z	1.142554e+09	False	False	0.0	5e9d0d95eda69955f709d1eb	False	Engine failure at 33 seconds and loss of vehicle	[]	
1	None	NaN	False	False	0.0	5e9d0d95eda69955f709d1eb	False	Successful first stage burn and transition to second stage, maximum altitude 289 km, Premature engine shutdown at T+7 min 30 s, Failed to reach orbit, Failed to recover first stage	[]	
2	None	NaN	False	False	0.0	5e9d0d95eda69955f709d1eb	False	Residual stage 1 thrust led to collision between stage 1 and stage 2	[]	
								Ratsat was carried to orbit on the first successful		

	static_fire_date_utc	static_fire_date_unix	tbd	net	window	rocket	success	details	crew	sl
3	2008-09-20T00:00:00.000Z	1.221869e+09	False	False	0.0	5e9d0d95eda69955f709d1eb	True	launch of any privately funded and developed, liquid-propelled carrier rocket, the SpaceX Falcon 1	[]	
4	None	NaN	False	False	0.0	5e9d0d95eda69955f709d1eb	True	None	[]	

Using the dataframe `data` print the first 5 rows

In [55]:

```
# Get the head of the dataframe
data.head(1)
```

Out[55]:

	static_fire_date_utc	static_fire_date_unix	tbd	net	window	rocket	success	details	crew	ships
0	2006-03-17T00:00:00.000Z	1.142554e+09	False	False	0.0	5e9d0d95eda69955f709d1eb	False	Engine failure at 33 seconds and loss of vehicle	[]	[

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 [56]:

```
# Lets take a subset of our dataframe keeping only the features we want and the flight nu
mber, and date_utc.
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]
print(data.head(1))
# We will remove rows with multiple cores because those are falcon rockets with 2 extra r
ocket boosters and rows that have multiple payloads in a single rocket.
data = data[data['cores'].map(len)==1]
data = data[data['payloads'].map(len)==1]
print(data.head(1))

# Since payloads and cores are lists of size 1 we will also extract the single value in t
he list and replace the feature.
data['cores'] = data['cores'].map(lambda x : x[0])
data['payloads'] = data['payloads'].map(lambda x : x[0])
print(data.head(1))
# We also want to convert the date_utc to a datetime datatype and then extracting the dat
e leaving the time
data['date'] = pd.to_datetime(data['date_utc']).dt.date
```

```
print(data.head(1))
# Using the date we will restrict the dates of the launches
data = data[data['date'] <= datetime.date(2020, 11, 13)]
print(data.head(1))
```

```
                     rocket                     payloads  \
0  5e9d0d95eda69955f709d1eb  [5eb0e4b5b6c3bb0006eeb1e1]

                  launchpad  \
0  5e9e4502f5090995de566f86


cores  \
0  [{'core': '5e9e289df35918033d3b2623', 'flight': 1, 'gridfins': False, 'legs': False, '
reused': False, 'landing_attempt': False, 'landing_success': None, 'landing_type': None,
'landpad': None}]

   flight_number                  date_utc
0              1  2006-03-24T22:30:00.000Z
                     rocket                     payloads  \
0  5e9d0d95eda69955f709d1eb  [5eb0e4b5b6c3bb0006eeb1e1]

                  launchpad  \
0  5e9e4502f5090995de566f86


cores  \
0  [{'core': '5e9e289df35918033d3b2623', 'flight': 1, 'gridfins': False, 'legs': False, '
reused': False, 'landing_attempt': False, 'landing_success': None, 'landing_type': None,
'landpad': None}]

   flight_number                  date_utc
0              1  2006-03-24T22:30:00.000Z
                     rocket                   payloads  \
0  5e9d0d95eda69955f709d1eb  5eb0e4b5b6c3bb0006eeb1e1

                  launchpad  \
0  5e9e4502f5090995de566f86


cores  \
0  {'core': '5e9e289df35918033d3b2623', 'flight': 1, 'gridfins': False, 'legs': False, 'r
eused': False, 'landing_attempt': False, 'landing_success': None, 'landing_type': None, '
landpad': None}

   flight_number                  date_utc
0              1  2006-03-24T22:30:00.000Z
                     rocket                   payloads  \
0  5e9d0d95eda69955f709d1eb  5eb0e4b5b6c3bb0006eeb1e1

                  launchpad  \
0  5e9e4502f5090995de566f86


cores  \
0  {'core': '5e9e289df35918033d3b2623', 'flight': 1, 'gridfins': False, 'legs': False, 'r
eused': False, 'landing_attempt': False, 'landing_success': None, 'landing_type': None, '
landpad': None}

   flight_number                  date_utc        date
0              1  2006-03-24T22:30:00.000Z  2006-03-24
                     rocket                   payloads  \
0  5e9d0d95eda69955f709d1eb  5eb0e4b5b6c3bb0006eeb1e1

                  launchpad  \
0  5e9e4502f5090995de566f86


cores  \
0  {'core': '5e9e289df35918033d3b2623', 'flight': 1, 'gridfins': False, 'legs': False, 'r
eused': False, 'landing_attempt': False, 'landing_success': None, 'landing_type': None, '
landpad': None}
```

```
   flight_number                  date_utc        date
0              1  2006-03-24T22:30:00.000Z  2006-03-24
```

- **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 [57]:

```
#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 [58]:

```
BoosterVersion
```

Out[58]:

```
[]
```

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

In [59]:

```
# Call getBoosterVersion
getBoosterVersion(data)
```

the list has now been update

In [60]:

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

Out[60]:

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

we can apply the rest of the functions here:

In [61]:

```
# Call getLaunchSite
```

```
# Call getLaunchSite
getLaunchSite(data)
```

In [62]:

```
# Call getPayloadData
getPayloadData(data)
```

In [63]:

```
# Call getCoreData
getCoreData(data)
```

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

In [64]:

```
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 [66]:

```
# Create a data from launch_dict
df_new=pd.DataFrame(launch_dict)
df_new.head(3)
```

Out[66]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	Landing
0	1	2006-03-24	Falcon 1	20.0	LEO	Kwajalein Atoll	None None	1	False	False	False	N
1	2	2007-03-21	Falcon 1	NaN	LEO	Kwajalein Atoll	None None	1	False	False	False	N
2	4	2008-09-28	Falcon 1	165.0	LEO	Kwajalein Atoll	None None	1	False	False	False	N

Show the summary of the dataframe

In [67]:

```
df_new.head(5)
```

Out[67]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	Landing
0	1	2006-03-24	Falcon 1	20.0	LEO	Kwajalein Atoll	None None	1	False	False	False	N

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

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 [72]:

```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9=df_new[df_new['BoosterVersion']=='Falcon 9']
data_falcon9
```

Out[72]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	
4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	
5	8	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False	False	False	
6	10	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False	False	False	
7	11	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False	False	False	
8	12	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	False	False	False	
...	...	...	...	...	...	...	...	...	...	...	...	
89	102	2020-09-03	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	2	True	True	True	5e9e3
90	103	2020-10-06	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	3	True	True	True	5e9e3
91	104	2020-10-18	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	6	True	True	True	5e9e3
92	105	2020-10-24	Falcon 9	15600.0	VLEO	CCSFS SLC 40	True ASDS	3	True	True	True	5e9e3
93	106	2020-11-05	Falcon 9	3681.0	MEO	CCSFS SLC 40	True ASDS	1	True	False	True	5e9e3

90 rows × 17 columns

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

In [85]:

```
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9.head(50)
```

Out[85]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	
4	1	2010-06-04	Falcon 9	6123.547647	LEO	CCSFS SLC 40	None None	1	False	False	False	
5	2	2012-05-22	Falcon 9	525.000000	LEO	CCSFS SLC 40	None None	1	False	False	False	
6	3	2013-03-01	Falcon 9	677.000000	ISS	CCSFS SLC 40	None None	1	False	False	False	
7	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	
8	5	2013-12-03	Falcon 9	3170.000000	GTO	CCSFS SLC 40	None None	1	False	False	False	
9	6	2014-01-06	Falcon 9	3325.000000	GTO	CCSFS SLC 40	None None	1	False	False	False	
10	7	2014-04-18	Falcon 9	2296.000000	ISS	CCSFS SLC 40	True Ocean	1	False	False	True	
11	8	2014-07-14	Falcon 9	1316.000000	LEO	CCSFS SLC 40	True Ocean	1	False	False	True	
12	9	2014-08-05	Falcon 9	4535.000000	GTO	CCSFS SLC 40	None None	1	False	False	False	
13	10	2014-09-07	Falcon 9	4428.000000	GTO	CCSFS SLC 40	None None	1	False	False	False	
14	11	2014-09-21	Falcon 9	2216.000000	ISS	CCSFS SLC 40	False Ocean	1	False	False	False	
15	12	2015-01-10	Falcon 9	2395.000000	ISS	CCSFS SLC 40	False ASDS	1	True	False	True	5e9e30
16	13	2015-02-11	Falcon 9	570.000000	ES-L1	CCSFS SLC 40	True Ocean	1	True	False	True	
17	14	2015-04-14	Falcon 9	1898.000000	ISS	CCSFS SLC 40	False ASDS	1	True	False	True	5e9e30
18	15	2015-04-27	Falcon 9	4707.000000	GTO	CCSFS SLC 40	None None	1	False	False	False	
19	16	2015-06-28	Falcon 9	2477.000000	ISS	CCSFS SLC 40	None ASDS	1	True	False	True	5e9e30
20	17	2015-12-22	Falcon 9	2034.000000	LEO	CCSFS SLC 40	True RTLS	1	True	False	True	5e9e30
21	18	2016-01-17	Falcon 9	553.000000	PO	VAFB SLC 4E	False ASDS	1	True	False	True	5e9e30
22	19	2016-03-04	Falcon 9	5271.000000	GTO	CCSFS SLC 40	False ASDS	1	True	False	True	5e9e30
23	20	2016-04-08	Falcon 9	3136.000000	ISS	CCSFS SLC 40	True ASDS	1	True	False	True	5e9e30
24	21	2016-05-06	Falcon 9	4696.000000	GTO	CCSFS SLC 40	True ASDS	1	True	False	True	5e9e30
25	22	2016-05-27	Falcon 9	3100.000000	GTO	CCSFS SLC 40	True ASDS	1	True	False	True	5e9e30
26	23	2016-07-18	Falcon 9	2257.000000	ISS	CCSFS SLC 40	True RTLS	1	True	False	True	5e9e30
27	24	2016-08-14	Falcon 9	4600.000000	GTO	CCSFS SLC 40	True ASDS	1	True	False	True	5e9e30
28	25	2016-09-01	Falcon 9	5500.000000	GTO	CCSFS SLC 40	None ASDS	1	True	False	True	5e9e30
29	26	2017-01-14	Falcon 9	9600.000000	PO	VAFB SLC 4E	True ASDS	1	True	False	True	5e9e30
30	27	2017-02-19	Falcon 9	2490.000000	ISS	KSC LC 39A	True RTLS	1	True	False	True	5e9e30
31	28	2017-03-16	Falcon 9	5600.000000	GTO	KSC LC 39A	None None	1	False	False	False	

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	
32	29	2017-03-30	Falcon 9	5300.000000	GTO	KSC LC 39A	True ASDS	2	True	True	True	5e9e30
33	30	2017-05-01	Falcon 9	6123.547647	LEO	KSC LC 39A	True RTLS	1	True	False	True	5e9e30
34	31	2017-05-15	Falcon 9	6070.000000	GTO	KSC LC 39A	None None	1	False	False	False	
35	32	2017-06-03	Falcon 9	2708.000000	ISS	KSC LC 39A	True RTLS	1	True	False	True	5e9e30
36	33	2017-06-23	Falcon 9	3669.000000	GTO	KSC LC 39A	True ASDS	2	True	True	True	5e9e30
37	34	2017-06-25	Falcon 9	9600.000000	PO	VAFB SLC 4E	True ASDS	1	True	False	True	5e9e30
38	35	2017-07-05	Falcon 9	6761.000000	GTO	KSC LC 39A	None None	1	False	False	False	
39	36	2017-08-14	Falcon 9	2910.000000	ISS	KSC LC 39A	True RTLS	1	True	False	True	5e9e30
40	37	2017-08-24	Falcon 9	475.000000	SSO	VAFB SLC 4E	True ASDS	1	True	False	True	5e9e30
41	38	2017-09-07	Falcon 9	4990.000000	LEO	KSC LC 39A	True RTLS	1	True	False	True	5e9e30
42	39	2017-10-09	Falcon 9	9600.000000	PO	VAFB SLC 4E	True ASDS	1	True	False	True	5e9e30
43	40	2017-10-11	Falcon 9	5200.000000	GTO	KSC LC 39A	True ASDS	2	True	True	True	5e9e30
44	41	2017-10-30	Falcon 9	3700.000000	GTO	KSC LC 39A	True ASDS	1	True	False	True	5e9e30
45	42	2017-12-15	Falcon 9	2205.000000	ISS	CCSFS SLC 40	True RTLS	2	True	True	True	5e9e30
46	43	2017-12-23	Falcon 9	9600.000000	PO	VAFB SLC 4E	True Ocean	2	True	True	False	
47	44	2018-01-08	Falcon 9	6123.547647	LEO	CCSFS SLC 40	True RTLS	1	True	False	True	5e9e30
48	45	2018-01-31	Falcon 9	4230.000000	GTO	CCSFS SLC 40	True Ocean	2	True	True	True	
49	46	2018-03-06	Falcon 9	6092.000000	GTO	CCSFS SLC 40	None None	1	True	False	True	
50	47	2018-03-30	Falcon 9	9600.000000	PO	VAFB SLC 4E	None None	2	True	True	True	
51	48	2018-04-02	Falcon 9	2760.000000	ISS	CCSFS SLC 40	None None	2	True	True	True	
52	49	2018-04-18	Falcon 9	350.000000	HEO	CCSFS SLC 40	True ASDS	1	True	False	True	5e9e30
53	50	2018-05-11	Falcon 9	3750.000000	GTO	KSC LC 39A	True ASDS	1	True	False	True	5e9e30

Data Wrangling

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

In [77]:

```
print(data_falcon9.shape)
data_falcon9.isnull().sum()
```

(90, 17)

Out[77]:

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

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

Task 3: Dealing with Missing Values

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

In [86]:

```
# Calculate the mean value of PayloadMass column
data_falcon9['PayloadMass'].replace(np.nan,data_falcon9['PayloadMass'].mean(),inplace=True)
data_falcon9['PayloadMass'].replace(np.nan,data_falcon9['PayloadMass'].mean(),inplace=True)
data_falcon9.isnull().sum()
# Replace the np.nan values with its mean value
data_falcon9.to_csv('dataset_part_1.csv',index=False)
```

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

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

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

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

Authors

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

Change Log

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

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

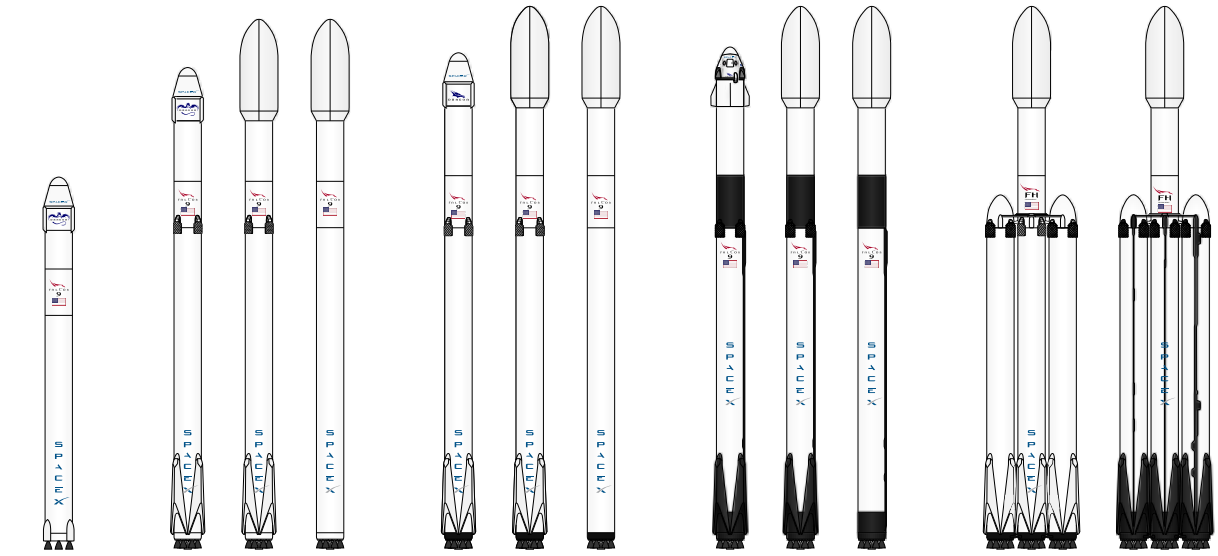
Space X Falcon 9 First Stage Landing Prediction

Web scraping Falcon 9 and Falcon Heavy Launches Records from Wikipedia

Estimated time needed: 40 minutes

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

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



Falcon 9 first stage will land successfully

Several examples of an unsuccessful landing are shown here:

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

2020 [edit]

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

[hide] Flight No.	Date and time (UTC)	Version, Booster[b]	Launch site	Payload[c]	Payload mass	Orbit	Customer	Launch outcome	Booster landing
78	7 January 2020, 02:19:21[492]	F9 B5 △ B1049.4	CCAFS, SLC-40	Starlink 2 v1.0 (60 satellites)	15,600 kg (34,400 lb)[5]	LEO	SpaceX	Success	Success (drone ship)
	Third large batch and second operational flight of Starlink constellation. One of the 60 satellites included a test coating to make the satellite less reflective, and thus less likely to interfere with ground-based astronomical observations.[493]								
79	19 January 2020, 15:30[494]	F9 B5 △ B1046.4	KSC, LC-39A	Crew Dragon in-flight abort test[495] (Dragon C205.1)	12,050 kg (26,570 lb)	Sub-orbital[496]	NASA (CTS)[497]	Success	No attempt
	An atmospheric test of the Dragon 2 abort system after Max Q. The capsule fired its SuperDraco engines, reached an apogee of 40 km (25 mi), deployed parachutes after reentry, and splashed down in the ocean 31 km (19 mi) downrange from the launch site. The test was previously slated to be accomplished with the Crew Dragon Demo-1 capsule;[498] but that test article exploded during a ground test of SuperDraco engines on 20 April 2019.[419] The abort test used the capsule originally intended for the first crewed flight.[499] As expected, the booster was destroyed by aerodynamic forces after the capsule aborted.[500] First flight of a Falcon 9 with only one functional stage — the second stage had a mass simulator in place of its engine.								
80	29 January 2020, 14:07[501]	F9 B5 △ B1051.3	CCAFS, SLC-40	Starlink 3 v1.0 (60 satellites)	15,600 kg (34,400 lb)[5]	LEO	SpaceX	Success	Success (drone ship)
	Third operational and fourth large batch of Starlink satellites, deployed in a circular 290 km (180 mi) orbit. One of the fairing halves was caught, while the other was fished out of the ocean.[502]								
81	17 February 2020, 15:05[503]	F9 B5 △ B1056.4	CCAFS, SLC-40	Starlink 4 v1.0 (60 satellites)	15,600 kg (34,400 lb)[5]	LEO	SpaceX	Success	Failure (drone ship)
	Fourth operational and fifth large batch of Starlink satellites. Used a new flight profile which deployed into a 212 km × 386 km (132 mi × 240 mi) elliptical orbit instead of launching into a circular orbit and firing the second stage engine twice. The first stage booster failed to land on the drone ship[504] due to incorrect wind data.[505] This was the first time a flight proven booster failed to land.								
82	7 March 2020, 04:50[506]	F9 B5 △ B1059.2	CCAFS, SLC-40	SpaceX CRS-20 (Dragon C112.3 △)	1,977 kg (4,359 lb)[507]	LEO (ISS)	NASA (CRS)	Success	Success (ground pad)
	Last launch of phase 1 of the CRS contract. Carries *Bartolomeo*, an ESA platform for hosting external payloads onto ISS.[508] Originally scheduled to launch on 2 March 2020, the launch date was pushed back due to a second stage engine failure. SpaceX decided to swap out the second stage instead of replacing the faulty part.[509] It was SpaceX's 50th successful landing of a first stage booster, the third flight of the Dragon C112 and the last launch of the cargo Dragon spacecraft.								
83	18 March 2020, 12:16[510]	F9 B5 △ B1048.5	KSC, LC-39A	Starlink 5 v1.0 (60 satellites)	15,600 kg (34,400 lb)[5]	LEO	SpaceX	Success	Failure (drone ship)
	Fifth operational launch of Starlink satellites. It was the first time a first stage booster flew for a fifth time and the second time the fairings were reused (Starlink flight in May 2019).[511] Towards the end of the first stage burn, the booster suffered premature shut down of an engine, the first of a Merlin 1D variant and first since the CRS-1 mission in October 2012. However, the payload still reached the targeted orbit.[512] This was the second Starlink launch booster landing failure in a row, later revealed to be caused by residual cleaning fluid trapped inside a sensor.[513]								
84	22 April 2020, 19:30[514]	F9 B5 △ B1051.4	KSC, LC-39A	Starlink 6 v1.0 (60 satellites)	15,600 kg (34,400 lb)[5]	LEO	SpaceX	Success	Success (drone ship)

Objectives

Web scrap Falcon 9 launch records with `BeautifulSoup`:

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

First let's import required packages for this lab

In [1]:

```
!pip3 install beautifulsoup4
!pip3 install requests
```

```
Requirement already satisfied: beautifulsoup4 in /home/jupyterlab/conda/envs/python/lib/p
ython3.7/site-packages (4.11.1)
Requirement already satisfied: soupsieve>1.2 in /home/jupyterlab/conda/envs/python/lib/py
thon3.7/site-packages (from beautifulsoup4) (2.3.2.post1)
Requirement already satisfied: requests in /home/jupyterlab/conda/envs/python/lib/python3
.7/site-packages (2.29.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /home/jupyterlab/conda/envs/py
thon/lib/python3.7/site-packages (from requests) (3.1.0)
```

```
Requirement already satisfied: idna<4,>=2.5 in /home/jupyterlab/conda/envs/python/lib/pyt
hon3.7/site-packages (from requests) (3.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /home/jupyterlab/conda/envs/pytho
n/lib/python3.7/site-packages (from requests) (1.26.15)
Requirement already satisfied: certifi>=2017.4.17 in /home/jupyterlab/conda/envs/python/l
ib/python3.7/site-packages (from requests) (2023.5.7)
```

In [2]:

```
import sys

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

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

In [3]:

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

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

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


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


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

    colunm_name = ' '.join(row.contents)

    # Filter the digit and empty names
```

```
    if not(colunm_name.strip().isdigit()):
        colunm_name = colunm_name.strip()
        return colunm_name
```

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

In [4]:

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

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

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

First, let's perform an HTTP GET method to request the Falcon9 Launch HTML page, as an HTTP response.

In [12]:

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

Out[12]:

```
200
```

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

In [13]:

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

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

In [14]:

```
# Use soup.title attribute
print(soup.title)
```

```
<title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

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

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

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

In [22]:

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

Out[22]:

```
25
```

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

In [24]:

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

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

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

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

In [31]:

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

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

Check the extracted column names

In [32]:

```
print(column_names)
```

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

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

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

In [65]:

```
launch_dict= dict.fromkeys(column_names)

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

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

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

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

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

In [66]:

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

            # Date value
            # TODO: Append the date into launch_dict with key `Date`
```

```
            date = datatimelist[0].strip(',')
            #print(date)
            launch_dict['Date'].append(date)

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

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

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

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

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

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

            # Customer
            # TODO: Append the customer into launch_dict with key `Customer`
            if row[6].a is None:
                customer="No"
                launch_dict['Customer'].append(customer)
            else:
                customer = row[6].a.string
                launch_dict['Customer'].append(customer)

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

            # Booster landing
            # TODO: Append the launch_outcome into launch_dict with key `Booster landing
`


            booster_landing = landing_status(row[8]).strip()
            launch_dict['Booster landing'].append(booster_landing)
```

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

In [68]:

```
df= pd.DataFrame({ key:pd.Series(value) for key, value in launch_dict.items() })
df.tail(50)
df.to_csv('spacex_web_scraped.csv', index=False)
```

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

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

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

Authors

[Yan Luo](#)

[Nayef Abou Tayoun](#)

Change Log

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

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 [3]:

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

In [8]:

```
import folium
import pandas as pd
```

In [9]:

```
# 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

In [10]:

```
## 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 [11]:

```
# Download and read the `spacex_launch_geo.csv`
from js import fetch
import io

URL = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/spacex_launch_geo.csv'
resp = await fetch(URL)
spacex_csv_file = io.BytesIO((await resp.arrayBuffer()).to_py())
spacex_df=pd.read_csv(spacex_csv_file)
spacex_df.head()
```

Out[11]:

	Flight Number	Date	Time (UTC)	Booster Version	Launch Site	Payload	Payload Mass (kg)	Orbit	Customer	Landing Outcome	class	Lat	Lor
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	80.5773
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	80.5773
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	80.5773
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	80.5773
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	80.5773

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

In [14]:

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

Out[14]:

	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

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 [15]:

```
# 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 [16]:

```
# Create a blue circle at NASA Johnson Space Center's coordinate with a popup label showi
ng its name
circle = folium.Circle(nasa_coordinate, radius=1000, color='#d35400', fill=True).add_chi
ld(folium.Popup('NASA Johnson Space Center'))
# Create a blue circle at NASA Johnson Space Center's coordinate with a icon showing its
name
marker = folium.map.Marker(
    nasa_coordinate,
    # Create an icon as a text label
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % 'NASA JSC',
        )
    )
site_map.add_child(circle)
site_map.add_child(marker)

print("")
```

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 [17]:

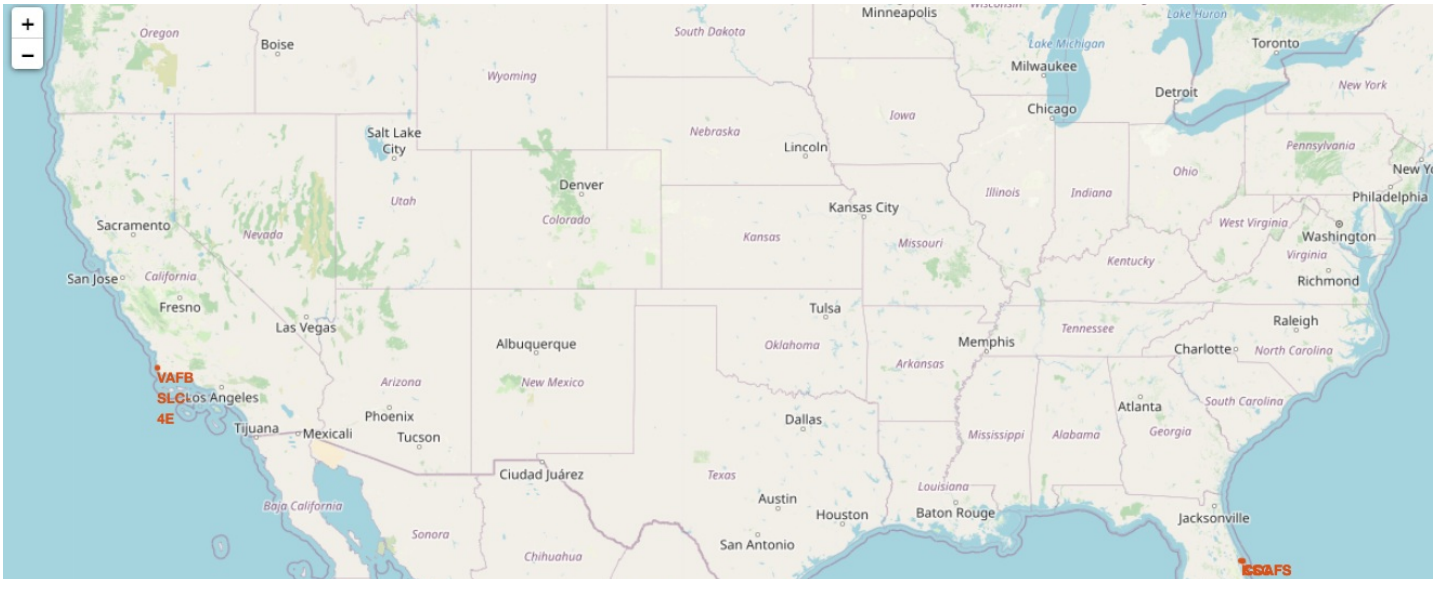
```
# Initial the map
site_map = folium.Map(location=nasa_coordinate, zoom_start=5)

def init0(site_map):
    radius=1000
    color="#d35400"
    for index, sm_row in spacex_df.iterrows():
```

```
        circle = folium.Circle([sm_row['Lat'],sm_row['Long']], radius=radius, color=colo
r, fill=False).add_child(folium.Popup(str(sm_row["Launch Site"])))
        # Create a blue circle at NASA Johnson Space Center's coordinate with a icon sho
wing its name
        marker = folium.map.Marker(
        [sm_row['Lat'],sm_row['Long']],
        # 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>' % sm_row['L
aunch Site'],
           )
        )
        site_map.add_child(circle)
        site_map.add_child(marker)

init0(site_map)
site_map
```

Out[17]:

Make this Notebook Trusted to load map: File -> Trust Notebook

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 [24]:

```
# Task 2: Mark the success/failed launches for each site on the map
spacex_df[["Lat","Long","Launch Site"]].groupby("Launch Site").first()
```

Out[24]:

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

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 [25]:

```
spacex_df.tail(10)
```

Out[25]:

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

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

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

Let's first create a `MarkerCluster` object

Let's first create a `MarkerCluster` object

In [26]:

```
marker_cluster = MarkerCluster()
```

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

In [25]:

```
# Apply a function to check the value of `class` column
# If class=1, marker_color value will be green
# If class=0, marker_color value will be red
```

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

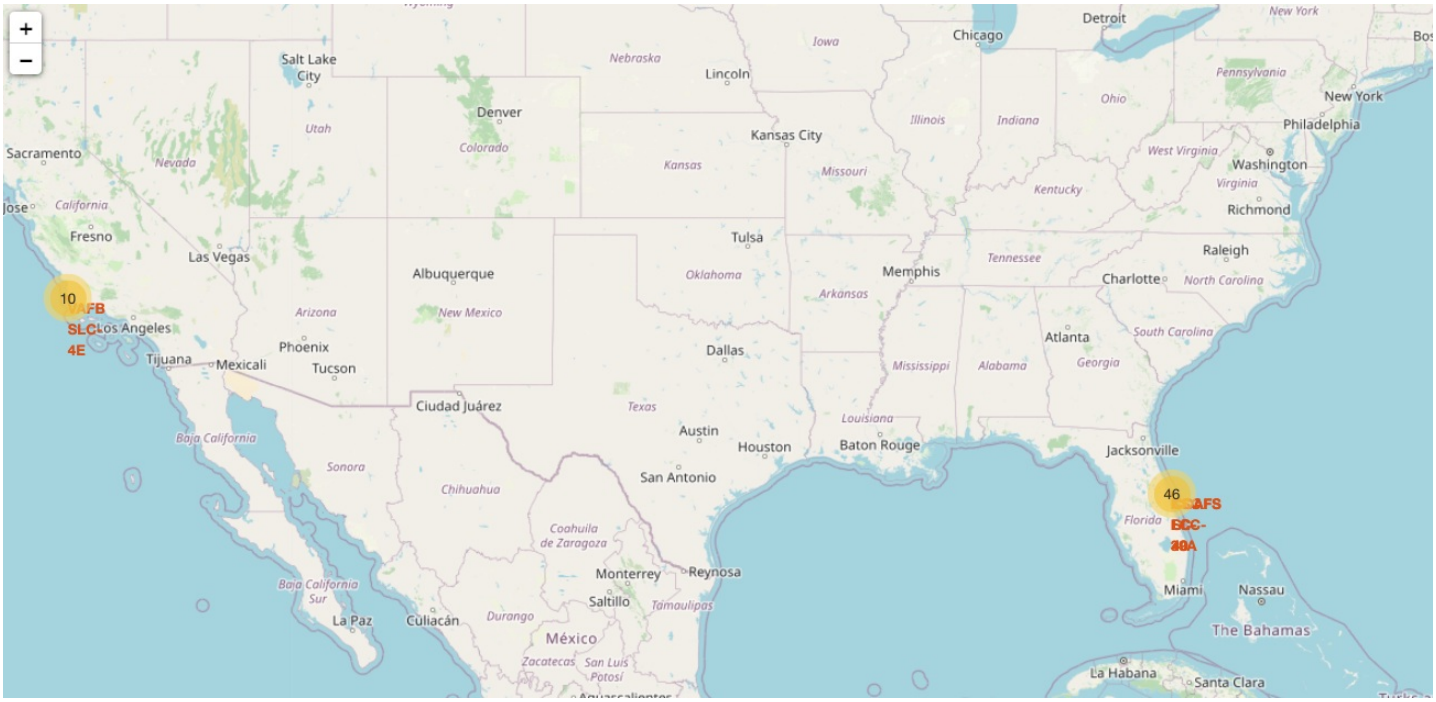
In [27]:

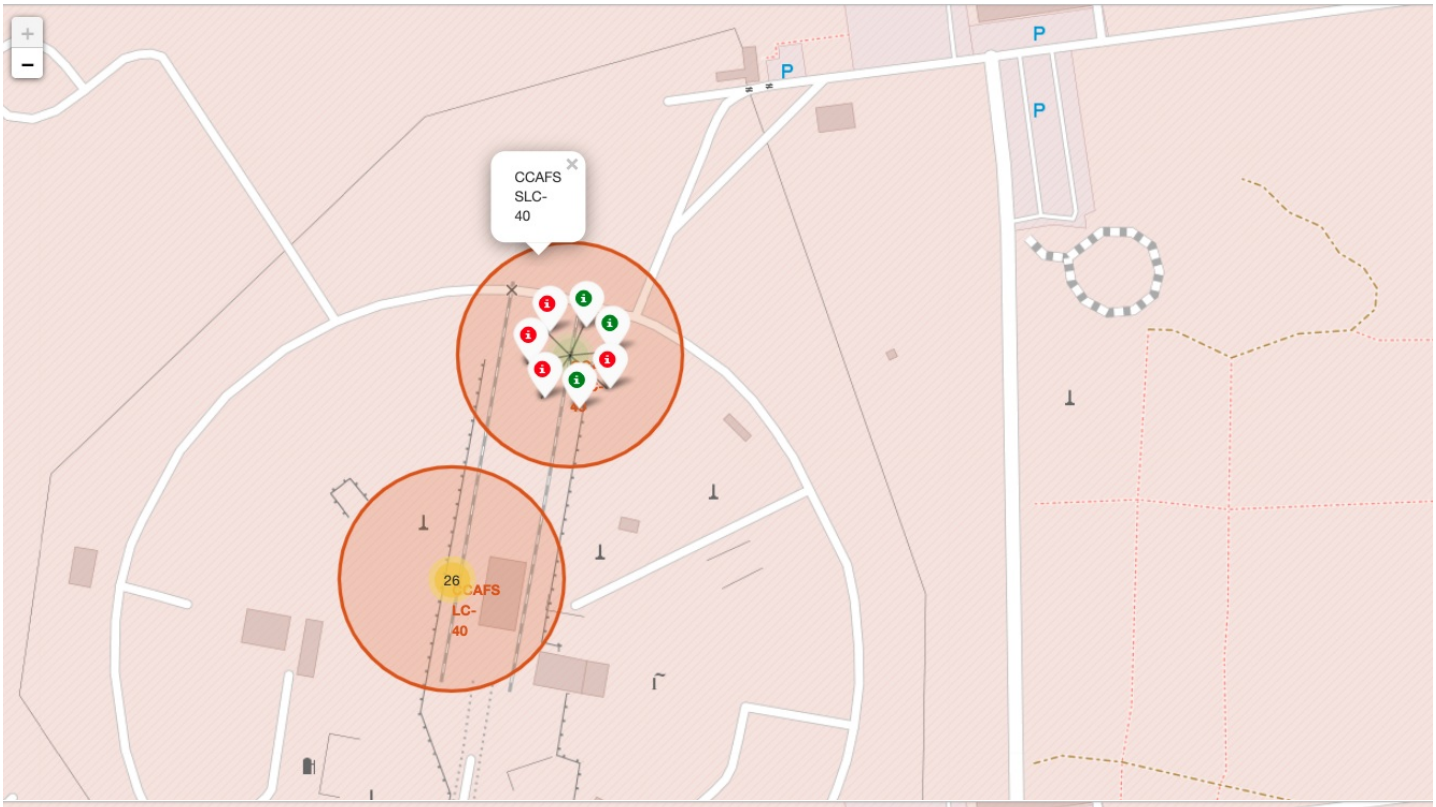
```
# Add marker_cluster to current site_map
site_map.add_child(marker_cluster)

# 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 fa
iled,
# e.g., icon=folium.Icon(color='white', icon_color=row['marker_color']
html=str()
for index, record in spacex_df.iterrows():
    # TODO: Create and add a Marker cluster to the site map
    if record["class"]==1:
        html='<div style="font-size: 12; color:#66cdaa;"><b>%s</b></div>'
    else:
        html='<div style="font-size: 12; color:"#df0000";"><b>%s</b></div>'
    marker = folium.map.Marker(
        [record['Lat'],record['Long']],
        # Create an icon as a text label
        icon=DivIcon(
           icon_size=(20,20),
           icon_anchor=(0,0),
           html= html % record['class'],
           )
        )
    marker_cluster.add_child(marker)

site_map
```

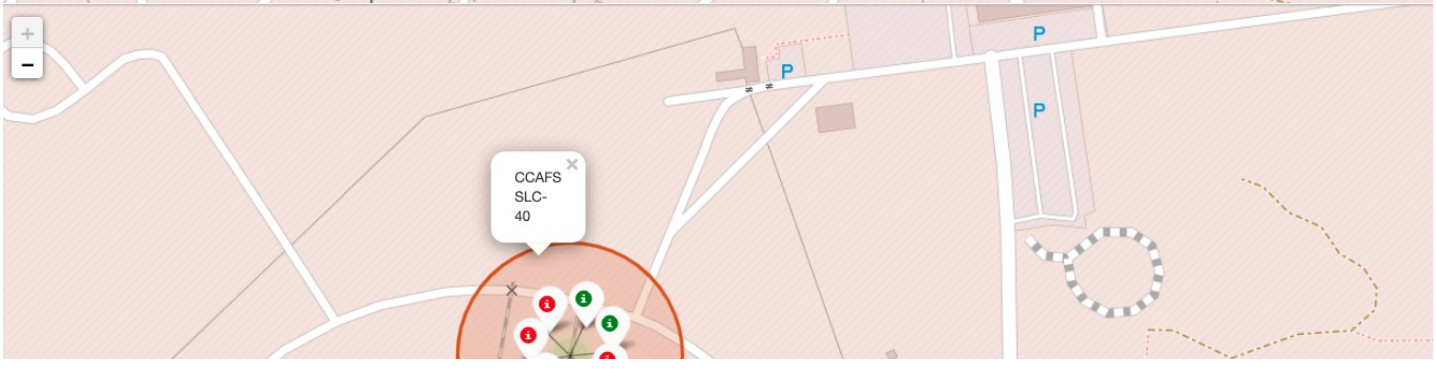
Out[27]:

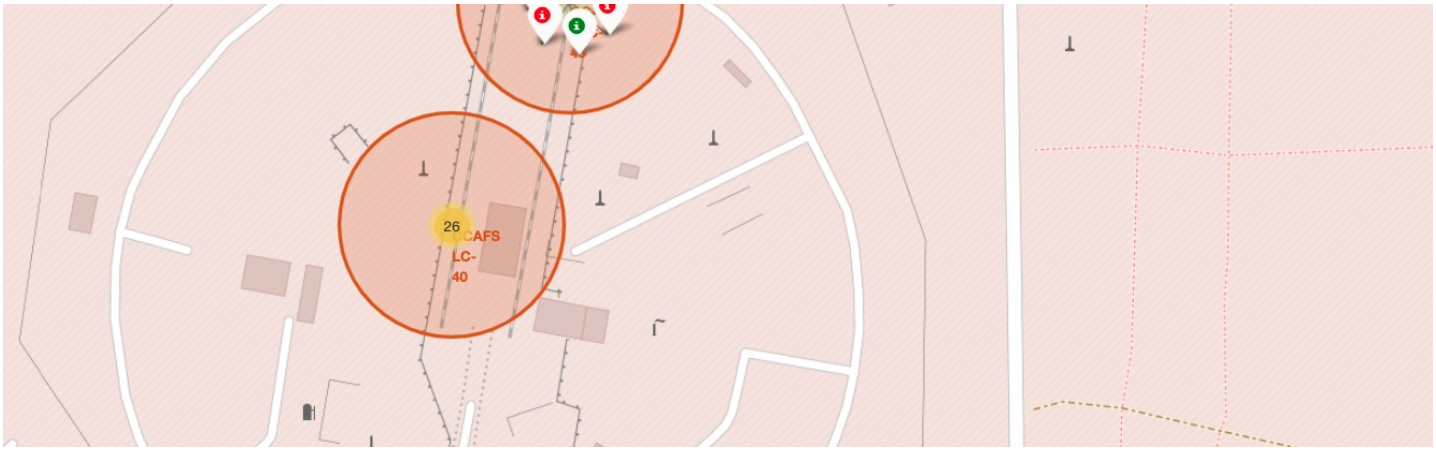
Make this Notebook Trusted to load map: File -> Trust Notebook

Your updated map may look like the following screenshots:









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

In [ ]:

```
# 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 [27]:

```
# 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[27]:

Make this Notebook Trusted to load map: File -> Trust Notebook

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 [30]:

```
from math import sin, cos, sqrt, atan2, radians

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

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

    dlon = lon2 - lon1
    dlat = lat2 - lat1

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

    distance = R * c
    return distance
```

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

In [31]:

```
# find coordinate of the closet coastline
# e.g.,: Lat: 28.56367  Lon: -80.57163
distance_coastline = calculate_distance(28.562302, -80.577356, 28.56367, -80.57163)
spacex_df[["Lat","Long","Launch Site"]].groupby("Launch Site").first()
```

Out[31]:

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

In [35]:

```
# Create and add a folium.Marker on your selected closest coastline point on the map
# Display the distance between coastline point and launch site using the icon property
# for example
coordinate=(28.56367,-80.57163)
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_coastline),
        )
    )
```

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

In [38]:

```
# Create a `folium.PolyLine` object using the coastline coordinates and launch site coord
inate
lines=folium.PolyLine(locations=((28.562302, -80.577356), (28.56367, -80.57163)), weight=
1)
site_map.add_child(lines)
```

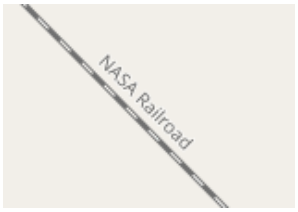
Out[38]:

Make this Notebook Trusted to load map: File -> Trust Notebook

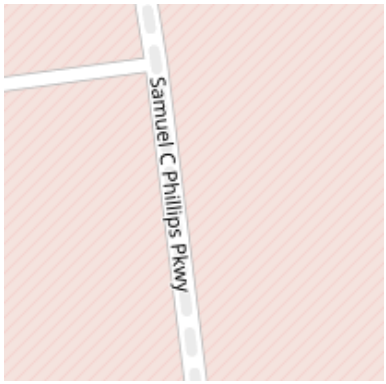
Your updated map with distance line should look like the following screenshot:

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 [ ]:

```

```

In [ ]:

```

```

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 Plotly Dash on detailed launch records.

Authors

[Pratiksha Verma]

Change Log

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

Space X Falcon 9 First Stage Landing Prediction

Lab 2: Data wrangling

Estimated time needed: 60 minutes

In this lab, we will perform some Exploratory Data Analysis (EDA) to find some patterns in the data and determine what would be the label for training supervised models.

In the data set, there are several different cases where the booster did not land successfully. Sometimes a landing was attempted but failed due to an accident; for example, `True Ocean` means the mission outcome was successfully landed to a specific region of the ocean while `False Ocean` means the mission outcome was unsuccessfully landed to a specific region of the ocean. `True RTLS` means the mission outcome was successfully landed to a ground pad `False RTLS` means the mission outcome was unsuccessfully landed to a ground pad. `True ASDS` means the mission outcome was successfully landed on a drone ship `False ASDS` means the mission outcome was unsuccessfully landed on a drone ship.

In this lab we will mainly convert those outcomes into Training Labels with `1` means the booster successfully landed `0` means it was unsuccessful.

Falcon 9 first stage will land successfully

Several examples of an unsuccessful landing are shown here:

Objectives

Perform exploratory Data Analysis and determine Training Labels

- **Exploratory Data Analysis**
- **Determine Training Labels**

Import Libraries and Define Auxiliary Functions

We will import the following libraries.

In [1]:

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

In [2]:

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

Data Analysis

In [3]:

```
from js import fetch
import io

URL = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-Sk
illsNetwork/datasets/dataset_part_1.csv'
resp = await fetch(URL)
dataset_part_1_csv = io.BytesIO((await resp.arrayBuffer()).to_py())
```

Load Space X dataset, from last section.

In [4]:

```
df=pd.read_csv(dataset_part_1_csv)
df.head(10)
```

Out[4]:

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

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

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

In [5]:

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

Out[5]:

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

In [6]:

```
df.dtypes
```

Out[6]:

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

TASK 1: Calculate the number of launches on each site

The data contains several Space X launch facilities: Cape Canaveral Space Launch Complex 40 VAFB SLC 4E , Vandenberg Air Force Base Space Launch Complex 4E (SLC-4E), Kennedy Space Center Launch Complex 39A

KSC LC 39A .The location of each Launch Is placed in the column `LaunchSite`

Next, let's see the number of launches for each site.

Use the method `value_counts()` **on the column** `LaunchSite` **to determine the number of launches on each site:**

In [8]:

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

Out[8]:

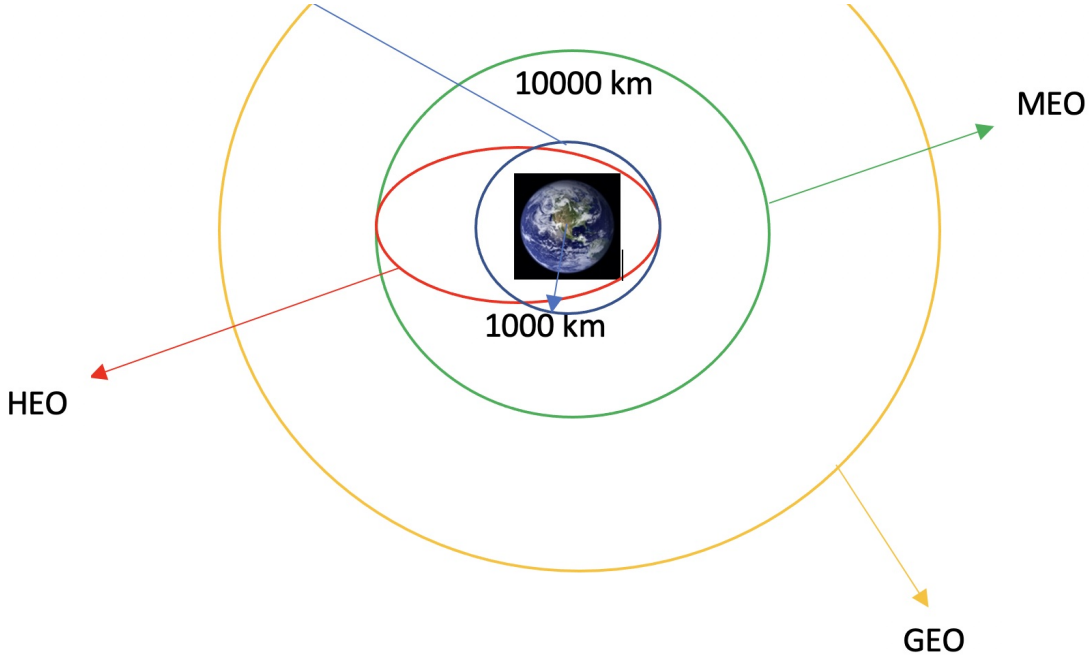
```
CCAFS SLC 40     55
KSC LC 39A       22
VAFB SLC 4E      13
Name: LaunchSite, dtype: int64
```

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

- **LEO: Low Earth orbit (LEO)is an Earth-centred orbit with an altitude of 2,000 km (1,200 mi) or less (approximately one-third of the radius of Earth),[1] or with at least 11.25 periods per day (an orbital period of 128 minutes or less) and an eccentricity less than 0.25.[2] Most of the manmade objects in outer space are in LEO \[1].**
- **VLEO: Very Low Earth Orbits (VLEO) can be defined as the orbits with a mean altitude below 450 km. Operating in these orbits can provide a number of benefits to Earth observation spacecraft as the spacecraft operates closer to the observation\[2].**
- **GTO A geosynchronous orbit is a high Earth orbit that allows satellites to match Earth's rotation. Located at 22,236 miles (35,786 kilometers) above Earth's equator, this position is a valuable spot for monitoring weather, communications and surveillance. Because the satellite orbits at the same speed that the Earth is turning, the satellite seems to stay in place over a single longitude, though it may drift north to south," NASA wrote on its Earth Observatory website \[3] .**
- **SSO (or SO): It is a Sun-synchronous orbit also called a heliosynchronous orbit is a nearly polar orbit around a planet, in which the satellite passes over any given point of the planet's surface at the same local mean solar time [4] .**

- **ES-L1 :At the Lagrange points the gravitational forces of the two large bodies cancel out in such a way that a small object placed in orbit there is in equilibrium relative to the center of mass of the large bodies. L1 is one such point between the sun and the earth \[5] .**
- **HEO A highly elliptical orbit, is an elliptic orbit with high eccentricity, usually referring to one around Earth \[6].**
- **ISS A modular space station (habitable artificial satellite) in low Earth orbit. It is a multinational collaborative project between five participating space agencies: NASA (United States), Roscosmos (Russia), JAXA (Japan), ESA (Europe), and CSA (Canada) \[7]**
- **MEO Geocentric orbits ranging in altitude from 2,000 km (1,200 mi) to just below geosynchronous orbit at 35,786 kilometers (22,236 mi). Also known as an intermediate circular orbit. These are "most commonly at 20,200 kilometers (12,600 mi), or 20,650 kilometers (12,830 mi), with an orbital period of 12 hours \[8]**
- **HEO Geocentric orbits above the altitude of geosynchronous orbit (35,786 km or 22,236 mi) \[9]**
- **GEO It is a circular geosynchronous orbit 35,786 kilometres (22,236 miles) above Earth's equator and following the direction of Earth's rotation \[10]**
- **PO It is one type of satellites in which a satellite passes above or nearly above both poles of the body being orbited (usually a planet such as the Earth \[11]**

some are shown in the following plot:



TASK 2: Calculate the number and occurrence of each orbit

Use the method `.value_counts()` to determine the number and occurrence of each orbit in the column `Orbit`

In [9]:

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

Out[9]:

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

TASK 3: Calculate the number and occurence of mission outcome per orbit type

Use the method `.value_counts()` on the column `Outcome` to determine the number of `landing_outcomes`.Then assign it to a variable landing_outcomes.

In [17]:

```
landing_outcomes = df['Outcome'].value_counts()
 landing_outcomes.keys()
```

Out[17]:

```
Index(['True ASDS', 'None None', 'True RTLS', 'False ASDS', 'True Ocean',
       'False Ocean', 'None ASDS', 'False RTLS'],
      dtype='object')
```

`True Ocean` means the mission outcome was successfully landed to a specific region of the ocean while `False Ocean` means the mission outcome was unsuccessfully landed to a specific region of the ocean. `True`

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

In [12]:

```
for i,outcome in enumerate(landing_outcomes.keys()):
    print(i,outcome)
```

```
0 True ASDS
1 None None
2 True RTLS
3 False ASDS
4 True Ocean
5 False Ocean
6 None ASDS
7 False RTLS
```

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

In [18]:

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

Out[18]:

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

TASK 4: Create a landing outcome label from Outcome column

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

In [46]:

```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
df['Outcome'].head()
list_1=list(df['Outcome'])
single_vals=set(df['Outcome'])
print(single_vals)
good_outcome=['True ASDS','True Ocean','True RTLS']
converted_list=[]
for i in list_1:
    if i in good_outcome:
        converted_list.append(1)
    else:
        converted_list.append(0)
landing_class=converted_list
landing_class
```

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

Out[46]:

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

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

In [47]:

```
df['Class']=landing_class
df[['Class']].head(8)
df[['Class','Outcome']].head(30)
```

Out[47]:

	Class	Outcome
0	0	None None
1	0	None None
2	0	None None
3	0	False Ocean
4	0	None None
5	0	None None
6	1	True Ocean
7	1	True Ocean
8	0	None None
9	0	None None
10	0	False Ocean
11	0	False ASDS
12	1	True Ocean
13	0	False ASDS
14	0	None None
15	0	None ASDS
16	1	True RTLS
17	0	False ASDS
18	0	False ASDS
19	1	True ASDS
20	1	True ASDS
21	1	True ASDS
22	1	True RTLS
23	1	True ASDS
24	0	None ASDS
25	1	True ASDS
26	1	True RTLS
27	0	None None
28	1	True ASDS
29	1	True RTLS

In [48]:

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

Out[48]:

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

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

In [50]:

```
print(df["Class"].mean())
df.to_csv("dataset_part_2.csv", index=False)
```

```
0.6666666666666666
```

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

```
df.to_csv("dataset_part\_2.csv", index=False)
```

Authors

[Pratiksha Verma]

Change Log

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

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**

- 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

In [1]:

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

We will import the following libraries for the lab

In [2]:

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

This function is to plot the confusion matrix.

In [3]:

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

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

Load the dataframe

Load the data

In [4]:

```
from js import fetch
import io

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

In [5]:

```
data.head()
```

Out[5]:

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

In [6]:

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

In [7]:

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

Out[7]:

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

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	...	Se

90 rows × 83 columns

TASK 1

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

In [8]:

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

TASK 2

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

In [9]:

```
# students get this
X = preprocessing.StandardScaler().fit_transform(X)
```

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

TASK 3

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

`X_train, X_test, Y_train, Y_test`

In [10]:

```
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 [35]:

```
X_test.shape
```

Out[35]:

```
(18, 83)
```

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`.

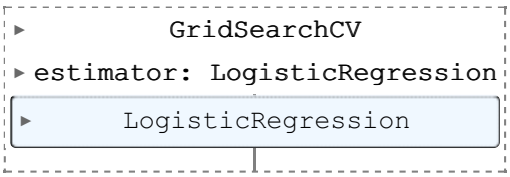
In [12]:

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

In [13]:

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

Out[13]:



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 [14]:

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

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

TASK 5

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

In [15]:

```
logreg_cv.score(X_test,Y_test)
```

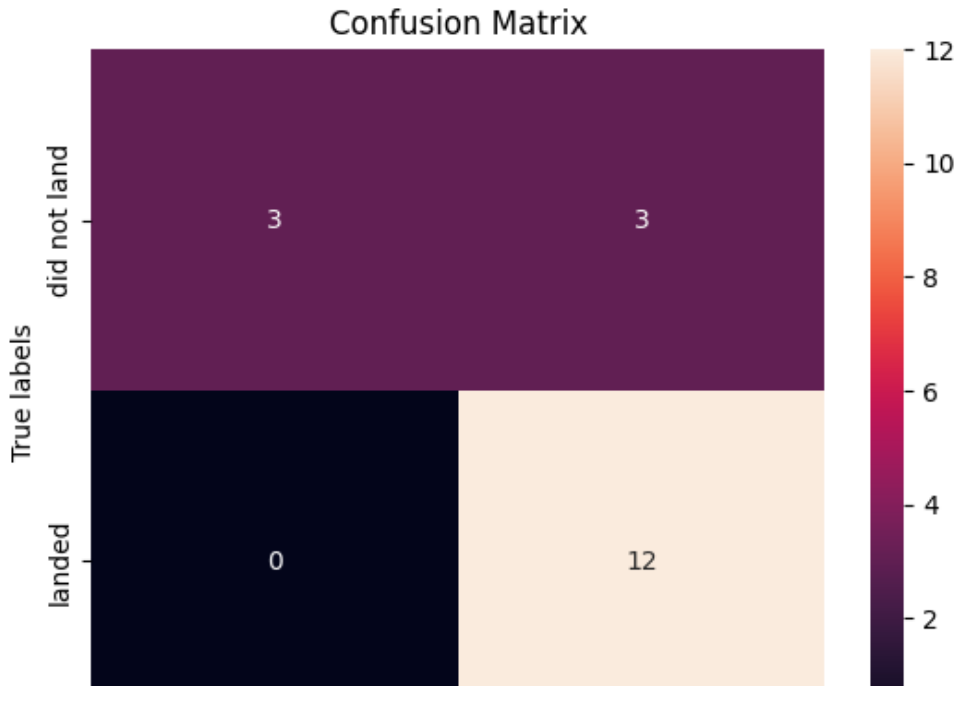
Out[15]:

```
0.8333333333333334
```

Lets look at the confusion matrix:

In [16]:

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



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

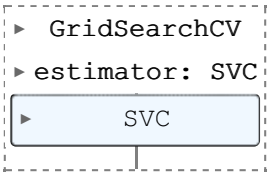
TASK 6

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 [17]:

```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
grid_obj=GridSearchCV(estimator=svm, cv=10, param_grid=parameters)
grid_obj.fit(X_train,Y_train)
```

Out[17]:



In [ ]:

```

```

In [18]:

```
print("tuned hpyerparameters :(best parameters) ",grid_obj.best_params_)
print("accuracy :",grid_obj.best_score_)
```

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

TASK 7

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

In [19]:

```
grid_obj.score(X_test,Y_test)
```

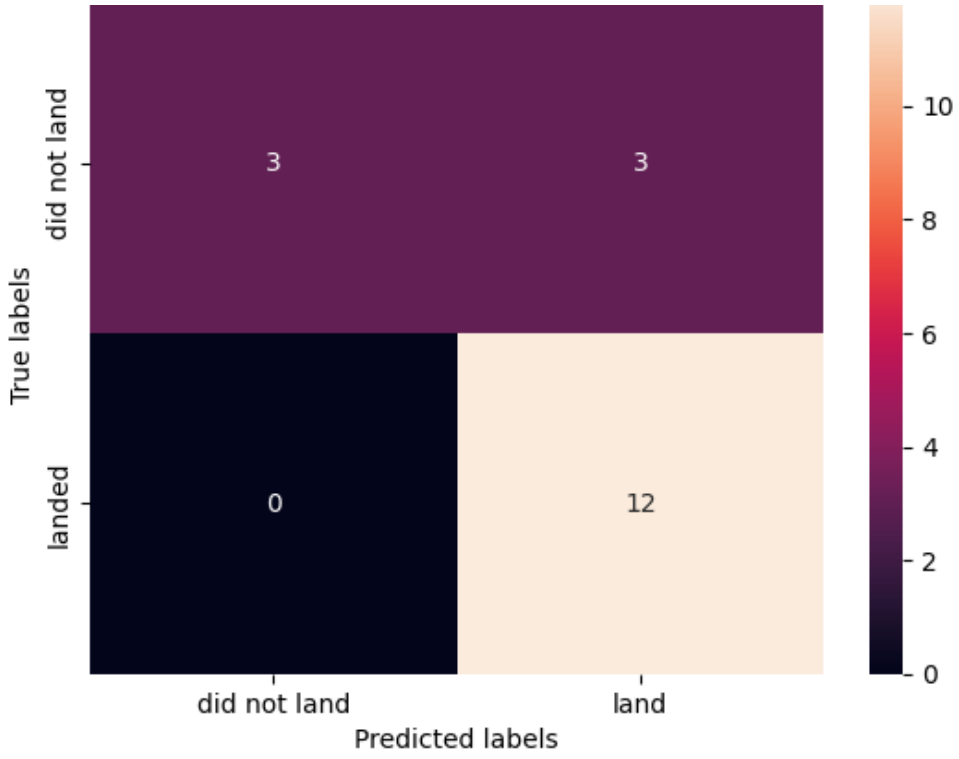
Out[19]:

```
0.8333333333333334
```

We can plot the confusion matrix

In [20]:

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



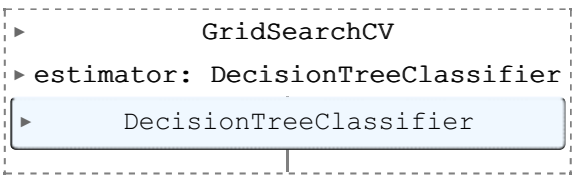
TASK 8

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

In [21]:

```
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn

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

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

Out[21]:



In [ ]:

```

```

In [22]:

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

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 12, 'max_fea
tures': 'sqrt', 'min_samples_leaf': 4, 'min_samples_split': 10, 'splitter': 'random'}
accuracy : 0.875
```

accuracy : 0.875

TASK 9

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

In [36]:

```
tree_cv.score(X_test,Y_test)
```

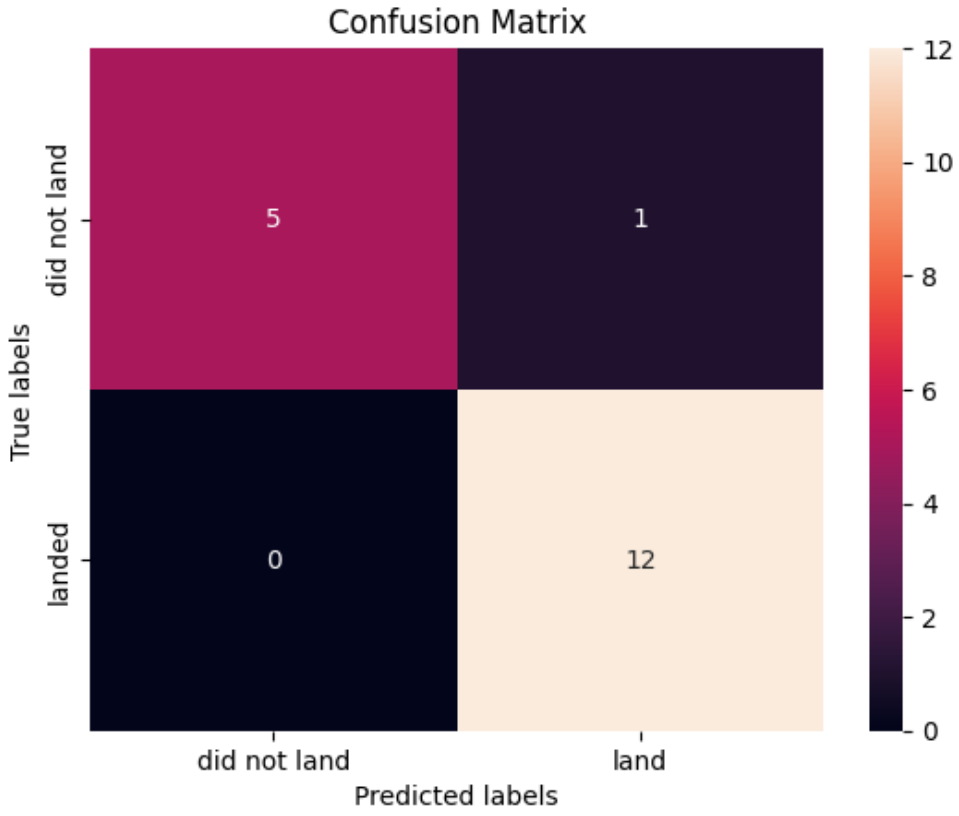
Out[36]:

```
0.9444444444444444
```

We can plot the confusion matrix

In [24]:

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



TASK 10

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

In [25]:

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

KNN = KNeighborsClassifier()
knn_cv=GridSearchCV(estimator=KNN, cv=10, param_grid=parameters)
knn_cv.fit(X_train,Y_train)
```

Out[25]:

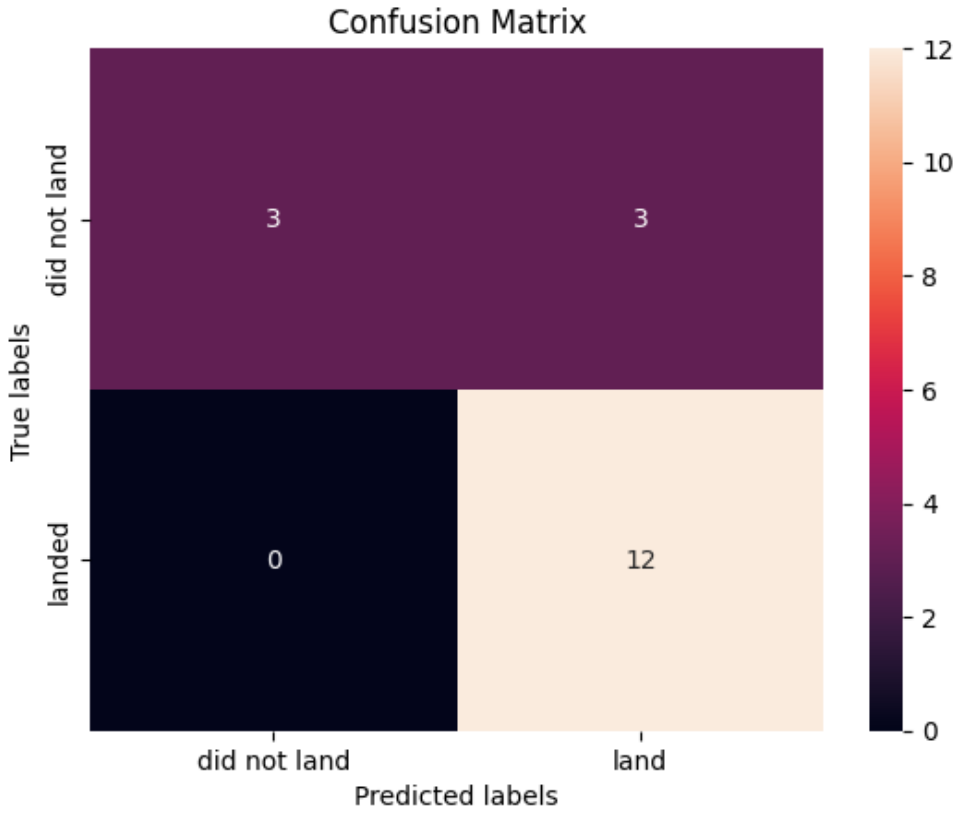
```
┌----------------------------------------┐
```

GridSearchCV
estimator: KNeighborsClassifier
KNeighborsClassifier

In [30]:

```
print(knn_cv.score(X_test,Y_test))
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```

0.8333333333333334



In [26]:

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

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

TASK 11

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

In [28]:

```
knn_cv.score(X_test,Y_test)
```

Out[28]:

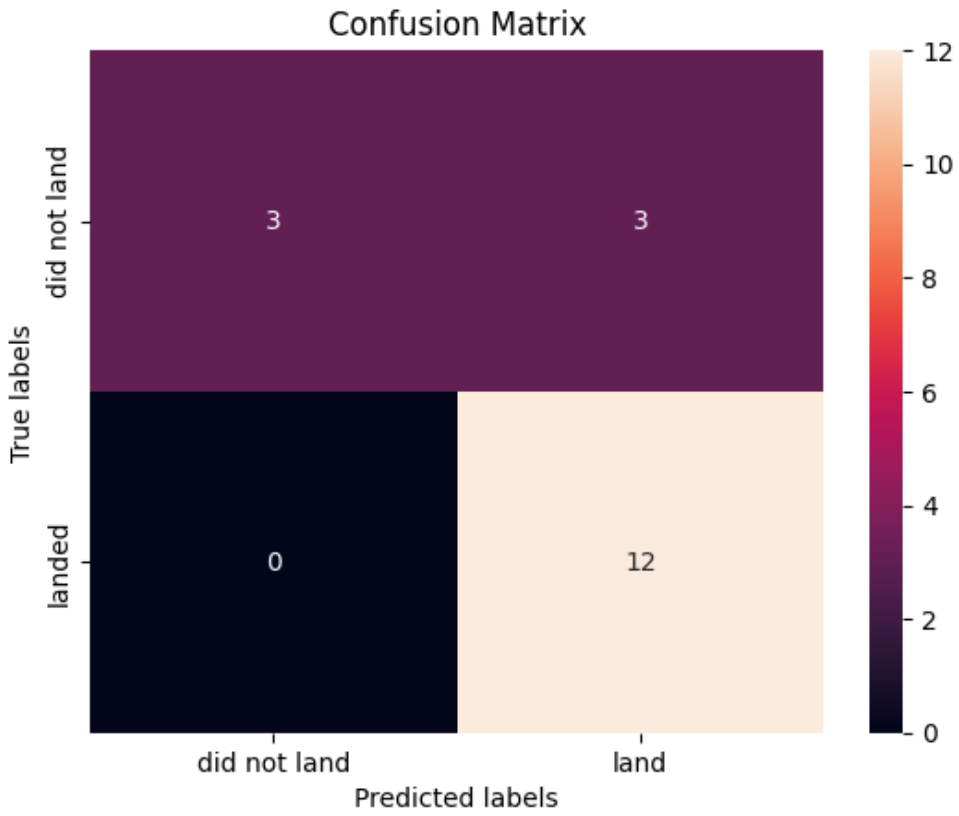
0.8333333333333334

We can plot the confusion matrix

In [29]:

```
yhat = knn_cv.predict(X_test)
```

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



TASK 12

Find the method performs best:

In [38]:

```
dt=DecisionTreeClassifier(criterion='gini', max_depth= 12, max_features='sqrt', min_samp
les_leaf=4, min_samples_split=10, splitter='random')
dt.fit(X_train,Y_train)
dt.score(X_test,Y_test)
```

Out[38]:

```
0.8333333333333334
```

Authors

[Pratiksha Verma]

Change Log

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