 cognitiveclass.ai logo

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 [2]:
```
# Requests allows us to make HTTP requests which we will use to get data from an
import requests
# Pandas is a software library written for the Python programming language for da
import pandas as pd
# NumPy is a library for the Python programming language, adding support for larg
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 [3]:
```
# Takes the dataset and uses the rocket column to call the API and append the dat
def getBoosterVersion(data):
    for x in data['rocket']:
        response = requests.get("https://api.spacexdata.com/v4/rockets/"+str(x)).
        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 [4]:
```
# Takes the dataset and uses the launchpad column to call the API and append the
def getLaunchSite(data):
    for x in data['launchpad']:
        response = requests.get("https://api.spacexdata.com/v4/launchpads/"+str(x
        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 [5]:
```
# Takes the dataset and uses the payloads column to call the API and append the d
def getPayloadData(data):
    for load in data['payloads']:
        response = requests.get("https://api.spacexdata.com/v4/payloads/"+load).j
        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 [6]:
```
# Takes the dataset and uses the cores column to call the API and append the data
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cores/"+co
                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_typ
            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 [7]:
```
spacex_url="https://api.spacexdata.com/v4/launches/past"
```

In [9]:
```
response = requests.get(spacex_url)
```

Check the content of the response

In [10]:
```
print(response.content)
```

```
b'[{"fairings":{"reused":false,"recovery_attempt":false,"recovered":false,
hips":[]},"links":{"patch":{"small":"https://images2.imgbox.com/3c/0e/T8iJcS
N3_o.png","large":"https://images2.imgbox.com/40/e3/GypSkayF_o.png"},"reddi
t":{"campaign":null,"launch":null,"media":null,"recovery":null},"flickr":{"s
mall":[],"original":[]},"presskit":null,"webcast":"https://www.youtube.com/w
atch?v=0a_00nJ_Y88","youtube_id":"0a_00nJ_Y88","article":"https://www.space.
com/2196-spacex-inaugural-falcon-1-rocket-lost-launch.html","wikipedia":"htt
ps://en.wikipedia.org/wiki/DemoSat"},"static_fire_date_utc":"2006-03-17T00:0
0:00.000Z","static_fire_date_unix":1142553600,"net":false,"window":0,"rocke
t":"5e9d0d95eda69955f709d1eb","success":false,"failures":[{"time":33,"altitu
de":null,"reason":"merlin engine failure"}],"details":"Engine failure at 33
seconds and loss of vehicle","crew":[],"ships":[],"capsules":[],"payloads":
["5eb0e4b5b6c3bb0006eeb1e1"],"launchpad":"5e9e4502f5090995de566f86","flight_
number":1,"name":"FalconSat","date_utc":"2006-03-24T22:30:00.000Z","date_uni
x":1143239400,"date_local":"2006-03-25T10:30:00+12:00","date_precision":"hou
r","upcoming":false,"cores":[{"core":"5e9e289df35918033d3b2623","flight":
1,"gridfins":false,"legs":false,"reused":false,"landing_attempt":false,"land
ing_success":null,"landing_type":null,"landpad":null}],"auto_update":true,"t
bd":false,"launch_library_id":null,"id":"5eb87cd9ffd86e000604b32a"},{"fairin
```

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 [ ]:
```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.clo
```

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

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

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

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

Using the dataframe `data` print the first 5 rows

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

Out[12]:

	static_fire_date_utc	static_fire_date_unix	net	window	rocket	success	[{
0	2006-03-17T00:00:00.000Z	1.142554e+09	False	0.0	5e9d0d95eda69955f709d1eb	False	
1	None	NaN	False	0.0	5e9d0d95eda69955f709d1eb	False	
2	None	NaN	False	0.0	5e9d0d95eda69955f709d1eb	False	
3	2008-09-20T00:00:00.000Z	1.221869e+09	False	0.0	5e9d0d95eda69955f709d1eb	True	

	static_fire_date_utc	static_fire_date_unix	net	window	rocket	success
4	None	NaN	False	0.0	5e9d0d95eda69955f709d1eb	True

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 [14]:
```
# Lets take a subset of our dataframe keeping only the features we want and the f
data =df[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc'

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

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

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

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

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

to seperate version of cores, the number of times this specific core has been reused, and the serial of the core.

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

In [15]:
```
#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 [16]:
```
BoosterVersion
```

Out[16]:
```
[]
```

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

In [17]:
```
# Call getBoosterVersion
getBoosterVersion(data)
```

the list has now been update

In [18]:
```
BoosterVersion[0:5]
```

Out[18]:
```
['Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 9']
```

we can apply the rest of the functions here:

In [19]:
```
# Call getLaunchSite
getLaunchSite(data)
```

In [20]:
```
# Call getPayloadData
getPayloadData(data)
```

In [21]:
```
# Call getCoreData
getCoreData(data)
```

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

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

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

In [23]:
```
# Create a data from launch_dict
df2 = pd.DataFrame(launch_dict)
```

Show the summary of the dataframe

In [24]:
```
# Show the head of the dataframe
df2.head()
```

Out[24]:

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

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 [25]:
```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = df2[df2['BoosterVersion']!='Falcon 1']
```

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

In [26]:
```
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9
```

```
/opt/conda/envs/Python-3.9/lib/python3.9/site-packages/pandas/core/indexing.p
1773: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  self._setitem_single_column(ilocs[0], value, pi)
```

Out[26]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	Grid
4	1	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	F
5	2	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	F
6	3	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	F
7	4	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	F
8	5	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	F
...	...	...	...	...	...	...	...	...	
89	86	2020-09-03	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	2	
90	87	2020-10-06	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	3	
91	88	2020-10-18	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	6	
92	89	2020-10-24	Falcon 9	15600.0	VLEO	CCSFS SLC 40	True ASDS	3	
93	90	2020-11-05	Falcon 9	3681.0	MEO	CCSFS SLC 40	True ASDS	1	

90 rows × 17 columns

Data Wrangling

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

In [27]:
```
data_falcon9.isnull().sum()
```

Out[27]:
```
FlightNumber       0
Date               0
BoosterVersion     0
PayloadMass        5
Orbit              0
LaunchSite         0
Outcome            0
Flights            0
GridFins           0
Reused             0
Legs               0
LandingPad        26
Block              0
ReusedCount        0
Serial             0
Longitude          0
Latitude           0
dtype: int64
```

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 [28]:
```
# Calculate the mean value of PayloadMass column
payload_mean = data_falcon9['PayloadMass'].mean()


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

```
/tmp/wsuser/ipykernel_155/1636456811.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  data_falcon9['PayloadMass'] = data_falcon9['PayloadMass'].replace(np.nan, pay
load_mean)
```

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 (https://www.linkedin.com/in/joseph-s-50398b136/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01) has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Change Log

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

cognitiveclass.ai logo

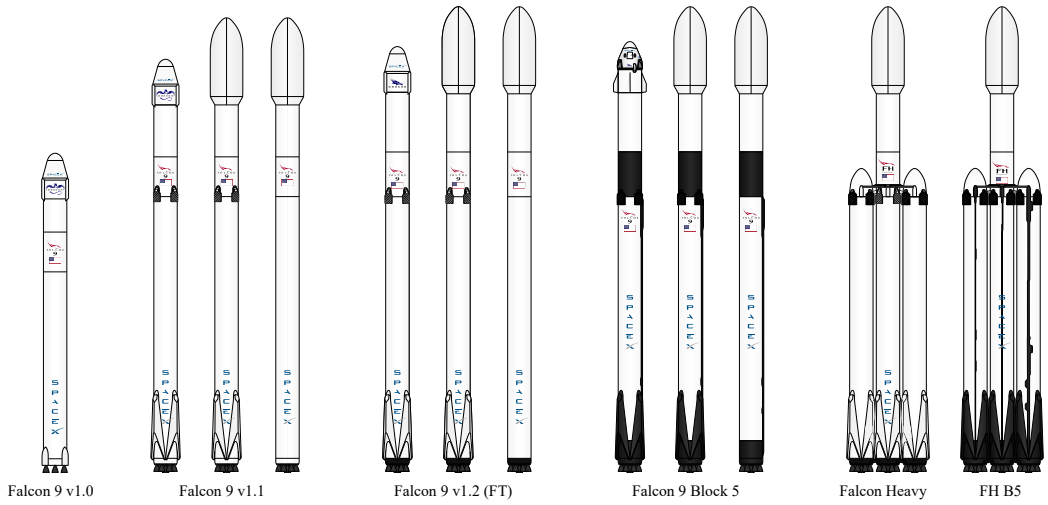
Space X Falcon 9 First Stage Landing Prediction

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 (https://en.wikipedia.org/wiki/List_of_Falcon_9_and_Falcon_Heavy_launches?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01)



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 [2]:
```
!pip3 install beautifulsoup4
!pip3 install requests
```

```
Requirement already satisfied: beautifulsoup4 in /opt/conda/envs/Python-3.9/l
b/python3.9/site-packages (4.10.0)
Requirement already satisfied: soupsieve>1.2 in /opt/conda/envs/Python-3.9/lib/
python3.9/site-packages (from beautifulsoup4) (2.3.1)
Requirement already satisfied: requests in /opt/conda/envs/Python-3.9/lib/pytho
n3.9/site-packages (2.26.0)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/envs/Python-3.9/lib/p
ython3.9/site-packages (from requests) (3.3)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /opt/conda/envs/Python-
3.9/lib/python3.9/site-packages (from requests) (1.26.7)
Requirement already satisfied: charset-normalizer~=2.0.0 in /opt/conda/envs/Pyt
hon-3.9/lib/python3.9/site-packages (from requests) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/envs/Python-3.
9/lib/python3.9/site-packages (from requests) (2021.10.8)
```

In [3]:
```
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 [4]:
```
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.
    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 [5]:
```
static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Fal
```

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 [6]:
```
# use requests.get() method with the provided static_url
response = requests.get(static_url)
# assign the response to a object
```

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

In [7]:
```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text conte
soup = BeautifulSoup(response.text, 'html')
```

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

In [8]:
```
# Use soup.title attribute
soup.title
```

Out[8]:
```
<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 [9]:
```
# Use the find_all function in the BeautifulSoup object, with element type `table
# Assign the result to a list called `html_tables`
html_tables = soup.find_all('table')
```

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

In [10]:
```
# Let's print the third table and check its content
first_launch_table = html_tables[2]
print(first_launch_table)
```

```
<table class="wikitable plainrowheaders collapsible" style="width: 100%;">
<tbody><tr>
<th scope="col">Flight No.
</th>
<th scope="col">Date and<br/>time (<a href="/wiki/Coordinated_Universal_Tim
e" 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 h
ref="#cite_note-Dragon-12">[c]</a></sup>
</th>
<th scope="col">Payload mass
</th>
```

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" ti
tle="List of Falcon 9 first-stage boosters">Version,<br/>Booster</a> <su
p class="reference" id="cite_ref-booster_11-0"><a href="#cite_note-boost
er-11">[b]</a></sup>
</th>
<th scope="col">Launch site
</th>
<th scope="col">Payload<sup class="reference" id="cite_ref-Dragon_12-0">
<a href="#cite_note-Dragon-12">[c]</a></sup>
</th>
<th scope="col">Payload mass
</th>
<th scope="col">Orbit
</th>
<th scope="col">Customer
</th>
```

```
<th scope="col">Launch<br/>outcome
</th>
<th scope="col"><a href="/wiki/Falcon_9_first-stage_landing_tests" title
="Falcon 9 first-stage landing tests">Booster<br/>landing</a>
</th></tr>
```

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

In [11]:
```
column_names = []

# Apply find_all() function with `th` element on first_launch_table
# Iterate each th element and apply the provided extract_column_from_header() to
# Append the Non-empty column name (`if name is not None and len(name) > 0`) into
tc = first_launch_table.find_all('th')
for th in tc:
    name = extract_column_from_header(th)
    if name is not None and len(name) > 0:
            column_names.append(name)
```

Check the extracted column names

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

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

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

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

In [13]:
```
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 [14]:
```
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowhead
   # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number corresponding to launch
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
        else:
            flag=False
        #get table element
        row=rows.find_all('td')
        #if it is number save cells in a dictonary
        if flag:
            extracted_row += 1
            # Flight Number value
            # TODO: Append the flight_number into launch_dict with key `Flight No
            #print(flight_number)
            datatimelist=date_time(row[0])

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

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

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

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

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

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

            # Orbit
            # TODO: Append the orbit into launch_dict with key `Orbit`
```

```
            orbit = row[5].a.string
            #print(orbit)

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

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

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

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

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

In [15]:
```
df=pd.DataFrame(launch_dict)
```

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 (https://www.linkedin.com/in/yan-luo-96288783/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=100065SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01)

Nayef Abou Tayoun (https://www.linkedin.com/in/nayefaboutayoun/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=100065SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01)

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

 cognitiveclass.ai logo

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:

Type *Markdown* and LaTeX: α^2

Objectives

Perform exploratory Data Analysis and determine Training Labels

- Exploratory Data Analysis
- Determine Training Labels

Import Libraries and Define Auxiliary Functions

We will import the following libraries.

In [1]:
```
# Pandas is a software library written for the Python programming language for da
import pandas as pd
#NumPy is a library for the Python programming language, adding support for large
import numpy as np
```

Data Analysis

Load Space X dataset, from last section.

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

Out[2]:

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

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

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

Out[3]:
```
FlightNumber       0.000
Date               0.000
BoosterVersion     0.000
PayloadMass        0.000
Orbit              0.000
LaunchSite         0.000
Outcome            0.000
Flights            0.000
GridFins           0.000
Reused             0.000
Legs               0.000
LandingPad        40.625
Block              0.000
ReusedCount        0.000
Serial             0.000
Longitude          0.000
Latitude           0.000
dtype: float64
```

Identify which columns are numerical and categorical:

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

Out[4]:
```
FlightNumber        int64
Date               object
BoosterVersion     object
PayloadMass       float64
Orbit              object
LaunchSite         object
Outcome            object
Flights             int64
GridFins             bool
Reused               bool
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 (https://en.wikipedia.org/wiki/List_of_Cape_Canaveral_and_Merritt_Island_launch_sites?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=1000655SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01)

Launch Complex 40 **VAFB SLC 4E** , Vandenberg Air Force Base Space Launch Complex 4E **(SLC-4E)**, Kennedy Space Center Launch Complex 39A **KSC LC 39A** .The location of each Launch Is placed in the column `LaunchSite`

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

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

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

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

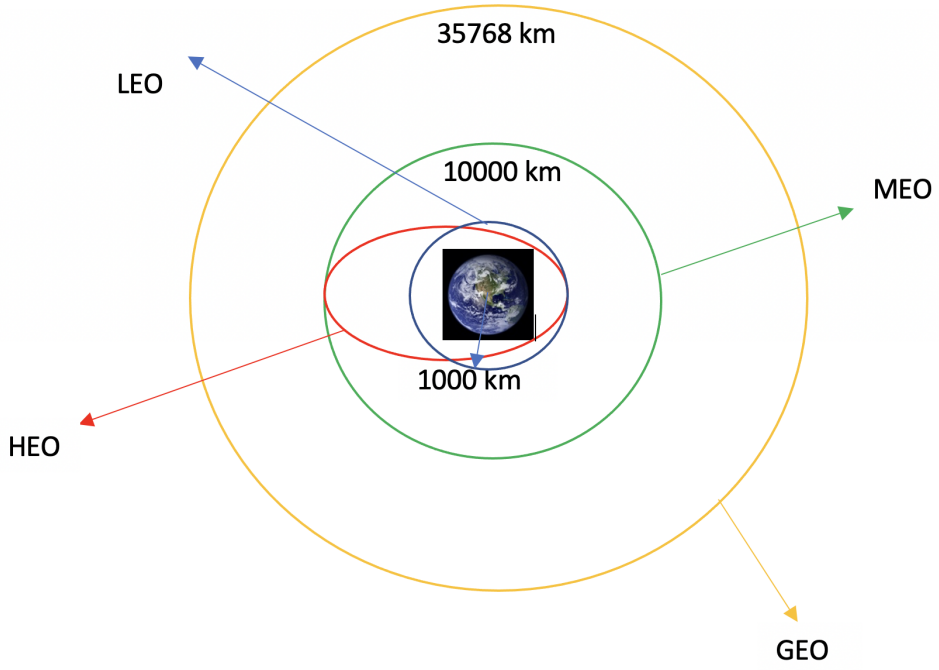
Each launch aims to an dedicated orbit, and here are some common orbit types:

- **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] (https://en.wikipedia.org/wiki/Low_Earth_orbit?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=100SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01).
- **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] (https://www.researchgate.net/publication/271499606_Very_Low_Earth_Orbit_mission_concepts_utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=100SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01).
- **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] (https://www.space.com/29222-geosynchronous-orbit.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=100SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01).
- **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] (https://en.wikipedia.org/wiki/Sun-synchronous_orbit?

utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=100SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01).

- **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] (https://en.wikipedia.org/wiki/Lagrange_point?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=100SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01#L1_point) .
- **HEO** A highly elliptical orbit, is an elliptic orbit with high eccentricity, usually referring to one around Earth [6] (https://en.wikipedia.org/wiki/Highly_elliptical_orbit?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=100SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01).
- **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] (https://en.wikipedia.org/wiki/International_Space_Station?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=100SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01)
- **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] (https://en.wikipedia.org/wiki/List_of_orbits?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=100SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01)
- **HEO** Geocentric orbits above the altitude of geosynchronous orbit (35,786 km or 22,236 mi) [9] (https://en.wikipedia.org/wiki/List_of_orbits?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=100SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01)
- **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] (https://en.wikipedia.org/wiki/Geostationary_orbit?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=100SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01)
- **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] (https://en.wikipedia.org/wiki/Polar_orbit?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=100SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01)

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

Out[6]:
```
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 [25]:
```
# landing_outcomes = values on Outcome column
df['Outcome'].value_counts()
landing_outcome = df['Outcome'].value_counts()
```

`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 [26]:
```
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 [27]:
```
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
bad_outcomes
```

Out[27]:
```
{'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 [28]:
```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
landing_class = df['Outcome'].map(lambda x: 0 if x in bad_outcomes else 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 [29]:
```
df['Class']=landing_class
df[['Class']].head(8)
```

Out[29]:

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

In [30]:
```
df.head(5)
```

Out[30]:

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

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

In [31]:
```
df["Class"].mean()
```

Out[31]:
```
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

Joseph Santarcangelo (https://www.linkedin.com/in/joseph-s-50398b136/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=100065SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01) has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Nayef Abou Tayoun (https://www.linkedin.com/in/nayefaboutayoun/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=100065SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01) is a Data Scientist at IBM and pursuing a Master of Management in Artificial intelligence degree at Queen's University.

Change Log

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

 cognitiveclass.ai logo

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

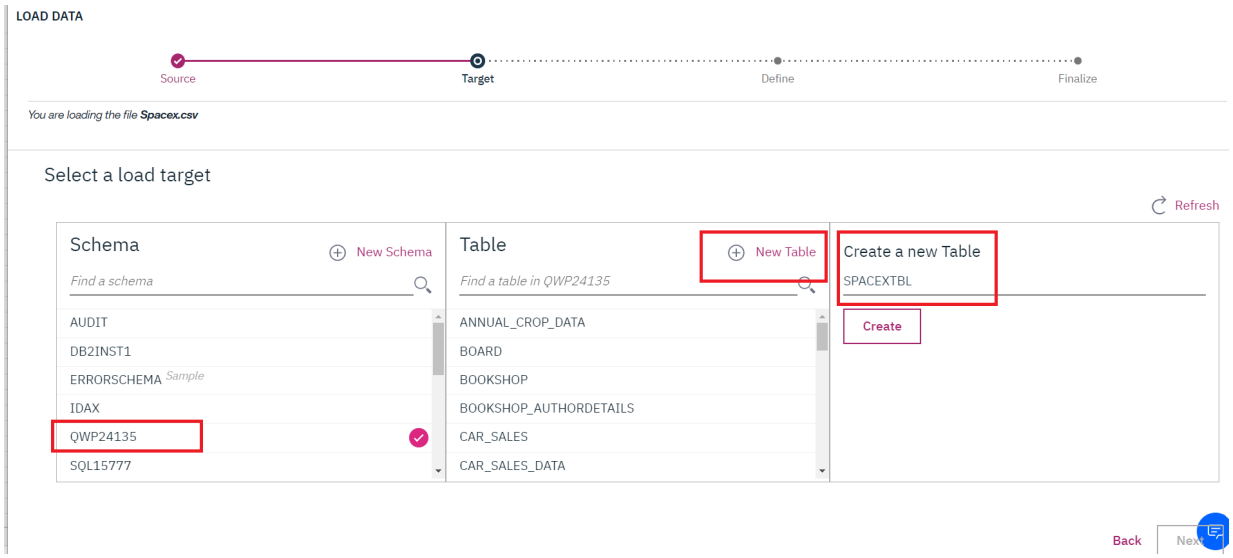
This assignment requires you to load the spacex dataset.

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 (https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/labs/module_2/data/Spacex.csv?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01)

Store the dataset in database table

it is highly recommended to manually load the table using the database console LOAD tool in DB2.



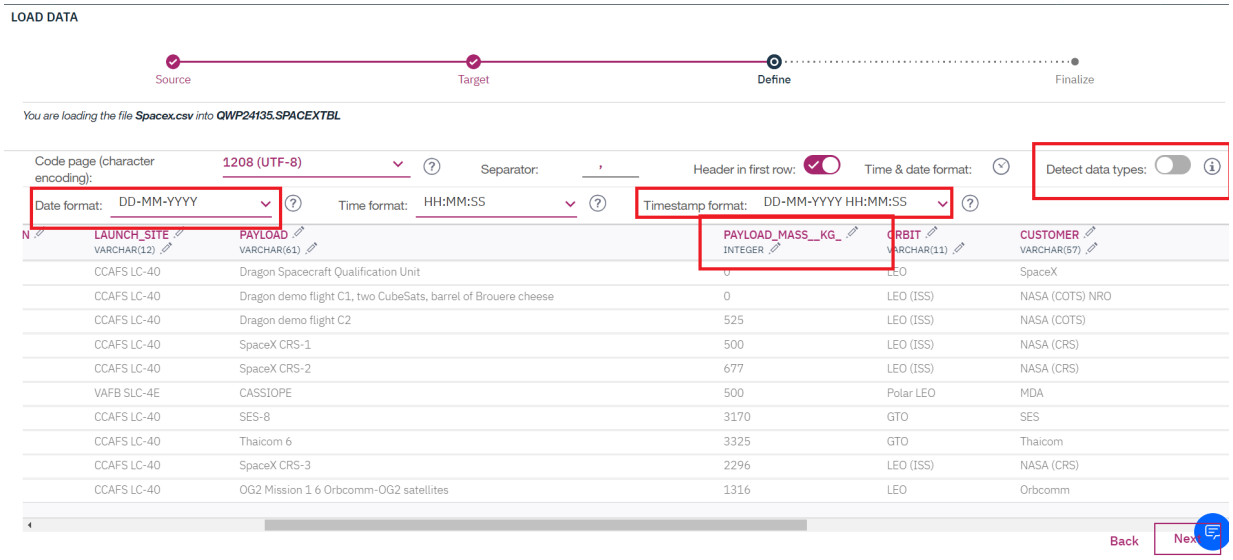
Now open the Db2 console, open the LOAD tool, Select / Drag the .CSV file for the dataset, Next create a New Table, and then follow the steps on-screen instructions to load the data. Name the new table as follows:

SPACEXDATASET

Follow these steps while using old DB2 UI which is having Open Console Screen

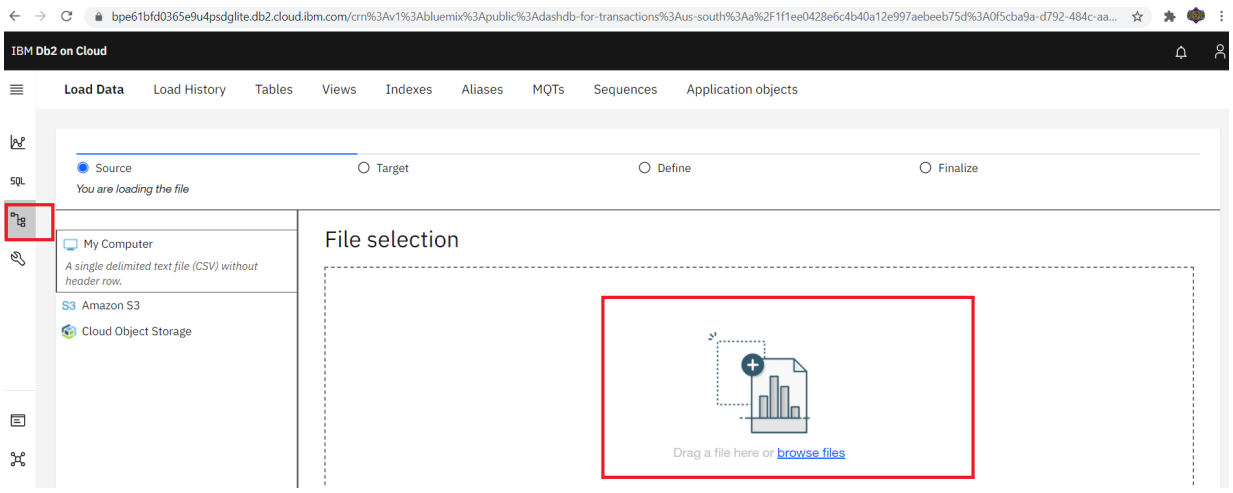
Note:While loading Spacex dataset, ensure that detect datatypes is disabled. Later click on the pencil icon(edit option).

1. Change the Date Format by manually typing DD-MM-YYYY and timestamp format as DD-MM-YYYY HH:MM:SS
2. Change the PAYLOAD_MASS__KG_ datatype to INTEGER.

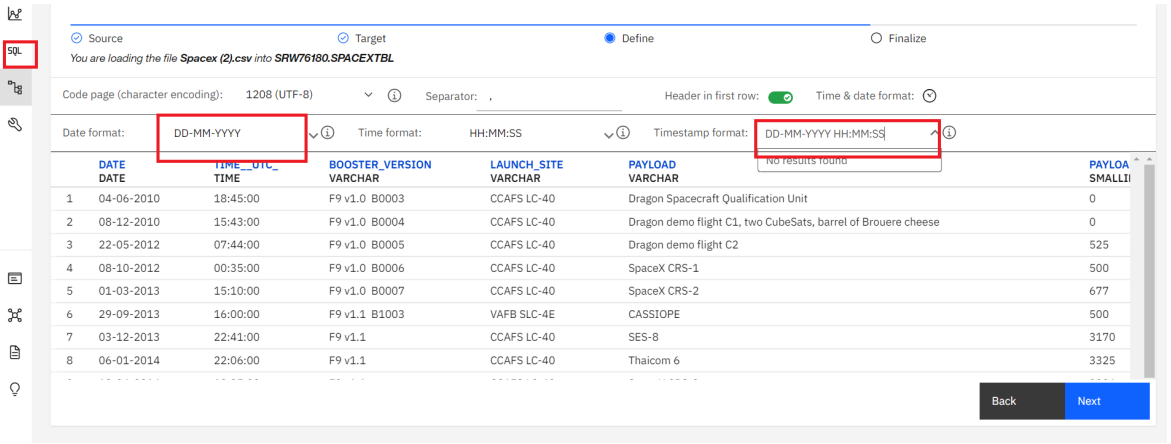


Changes to be considered when having DB2 instance with the new UI having Go to UI screen

- Refer to this insruction in this link (https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DB0201EN-SkillsNetwork/labs/Labs_Coursera_V5/labs/Lab%20-%20Sign%20up%20for%20IBM%20Cloud%20-%20Create%20Db2%20service%20instance%20-%20Get%20started%20with%20the%20Db2%20console/instructional-labs.md.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01) for viewing the new Go to UI screen.
- Later click on **Data link(below SQL)** in the Go to UI screen and click on **Load Data** tab.
- Later browse for the downloaded spacex file.



- Once done select the schema andload the file.



In [1]:
```
!pip install sqlalchemy==1.3.9
!pip install ibm_db_sa
!pip install ipython-sql
```

```
/opt/conda/envs/Python-3.7-OpenCE/lib/python3.7/site-packages/secretstorage/dh
rypto.py:16: CryptographyDeprecationWarning: int_from_bytes is deprecated, use
int.from_bytes instead
  from cryptography.utils import int_from_bytes
/opt/conda/envs/Python-3.7-OpenCE/lib/python3.7/site-packages/secretstorage/uti
l.py:25: CryptographyDeprecationWarning: int_from_bytes is deprecated, use int.
from_bytes instead
  from cryptography.utils import int_from_bytes
Collecting sqlalchemy==1.3.9
  Downloading SQLAlchemy-1.3.9.tar.gz (6.0 MB)
     |████████████████████████████████| 6.0 MB 27.9 MB/s eta 0:00:01
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 size=1207786 sha256=8d97d3008e2cd7e3b21929d62fb433e682b578c62fa2dfe23bd2
56ea8e84f657
  Stored in directory: /tmp/wsuser/.cache/pip/wheels/03/71/13/010faf12246f72dc7
6b4150e6e599d13a85b4435e06fb9e51f
Successfully built sqlalchemy
Installing collected packages: sqlalchemy
  Attempting uninstall: sqlalchemy
    Found existing installation: SQLAlchemy 1.4.22
    Uninstalling SQLAlchemy-1.4.22:
      Successfully uninstalled SQLAlchemy-1.4.22
Successfully installed sqlalchemy-1.3.9
/opt/conda/envs/Python-3.7-OpenCE/lib/python3.7/site-packages/secretstorage/dhc
rypto.py:16: CryptographyDeprecationWarning: int_from_bytes is deprecated, use
int.from_bytes instead
  from cryptography.utils import int_from_bytes
/opt/conda/envs/Python-3.7-OpenCE/lib/python3.7/site-packages/secretstorage/uti
l.py:25: CryptographyDeprecationWarning: int_from_bytes is deprecated, use int.
from_bytes instead
  from cryptography.utils import int_from_bytes
Requirement already satisfied: ibm_db_sa in /opt/conda/envs/Python-3.7-OpenCE/l
ib/python3.7/site-packages (0.3.7)
Requirement already satisfied: sqlalchemy>=0.7.3 in /opt/conda/envs/Python-3.7-
OpenCE/lib/python3.7/site-packages (from ibm_db_sa) (1.3.9)
Requirement already satisfied: ibm-db>=2.0.0 in /opt/conda/envs/Python-3.7-Open
CE/lib/python3.7/site-packages (from ibm_db_sa) (3.0.4)
/opt/conda/envs/Python-3.7-OpenCE/lib/python3.7/site-packages/secretstorage/dhc
rypto.py:16: CryptographyDeprecationWarning: int_from_bytes is deprecated, use
int.from_bytes instead
  from cryptography.utils import int_from_bytes
/opt/conda/envs/Python-3.7-OpenCE/lib/python3.7/site-packages/secretstorage/uti
l.py:25: CryptographyDeprecationWarning: int_from_bytes is deprecated, use int.
from_bytes instead
  from cryptography.utils import int_from_bytes
Collecting ipython-sql
  Downloading ipython_sql-0.4.0-py3-none-any.whl (19 kB)
Requirement already satisfied: sqlalchemy>=0.6.7 in /opt/conda/envs/Python-3.7-
OpenCE/lib/python3.7/site-packages (from ipython-sql) (1.3.9)
Collecting sqlparse
```

```
  Downloading sqlparse-0.4.2-py3-none-any.whl (42 kB)
     |████████████████████████████████| 42 kB 1.1 MB/s  eta 0:00:01
Requirement already satisfied: ipython>=1.0 in /opt/conda/envs/Python-3.7-OpenC
E/lib/python3.7/site-packages (from ipython-sql) (7.15.0)
Requirement already satisfied: six in /opt/conda/envs/Python-3.7-OpenCE/lib/pyt
hon3.7/site-packages (from ipython-sql) (1.15.0)
Collecting prettytable<1
  Downloading prettytable-0.7.2.tar.bz2 (21 kB)
Requirement already satisfied: ipython-genutils>=0.1.0 in /opt/conda/envs/Pytho
n-3.7-OpenCE/lib/python3.7/site-packages (from ipython-sql) (0.2.0)
Requirement already satisfied: decorator in /opt/conda/envs/Python-3.7-OpenCE/l
ib/python3.7/site-packages (from ipython>=1.0->ipython-sql) (4.4.2)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in
/opt/conda/envs/Python-3.7-OpenCE/lib/python3.7/site-packages (from ipython>=1.
0->ipython-sql) (3.0.5)
Requirement already satisfied: jedi>=0.10 in /opt/conda/envs/Python-3.7-OpenCE/
lib/python3.7/site-packages (from ipython>=1.0->ipython-sql) (0.17.1)
Requirement already satisfied: traitlets>=4.2 in /opt/conda/envs/Python-3.7-Ope
nCE/lib/python3.7/site-packages (from ipython>=1.0->ipython-sql) (4.3.3)
Requirement already satisfied: pexpect; sys_platform != "win32" in /opt/conda/e
nvs/Python-3.7-OpenCE/lib/python3.7/site-packages (from ipython>=1.0->ipython-s
ql) (4.8.0)
Requirement already satisfied: pygments in /opt/conda/envs/Python-3.7-OpenCE/li
b/python3.7/site-packages (from ipython>=1.0->ipython-sql) (2.6.1)
Requirement already satisfied: backcall in /opt/conda/envs/Python-3.7-OpenCE/li
b/python3.7/site-packages (from ipython>=1.0->ipython-sql) (0.2.0)
Requirement already satisfied: pickleshare in /opt/conda/envs/Python-3.7-OpenC
E/lib/python3.7/site-packages (from ipython>=1.0->ipython-sql) (0.7.5)
Requirement already satisfied: setuptools>=18.5 in /opt/conda/envs/Python-3.7-O
penCE/lib/python3.7/site-packages (from ipython>=1.0->ipython-sql) (47.3.1.post
20200622)
Requirement already satisfied: wcwidth in /opt/conda/envs/Python-3.7-OpenCE/li
b/python3.7/site-packages (from prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->i
python>=1.0->ipython-sql) (0.2.4)
Requirement already satisfied: parso<0.8.0,>=0.7.0 in /opt/conda/envs/Python-3.
7-OpenCE/lib/python3.7/site-packages (from jedi>=0.10->ipython>=1.0->ipython-sq
l) (0.7.0)
Requirement already satisfied: ptyprocess>=0.5 in /opt/conda/envs/Python-3.7-Op
enCE/lib/python3.7/site-packages (from pexpect; sys_platform != "win32"->ipytho
n>=1.0->ipython-sql) (0.6.0)
Building wheels for collected packages: prettytable
  Building wheel for prettytable (setup.py) ... done
  Created wheel for prettytable: filename=prettytable-0.7.2-py3-none-any.whl si
ze=13700 sha256=860db9c6d2075af198a96a737a819951f20e2ba2d33065765ae88d986ab4207
5
  Stored in directory: /tmp/wsuser/.cache/pip/wheels/8c/76/0b/eb9eb3da7e2335e35
77e3f96a0ae9f74f206e26457bd1a2bc8
Successfully built prettytable
Installing collected packages: sqlparse, prettytable, ipython-sql
Successfully installed ipython-sql-0.4.0 prettytable-0.7.2 sqlparse-0.4.2
```

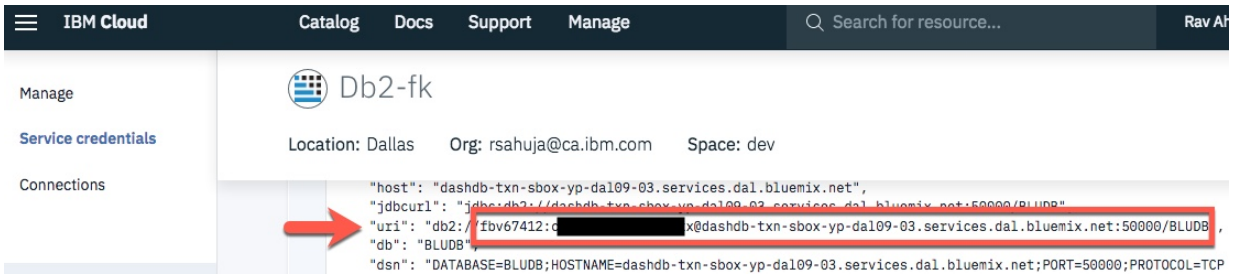
Connect to the database

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

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

DB2 magic in case of old UI service credentials.

In the next cell enter your db2 connection string. Recall you created Service Credentials for your Db2 instance before. From the **uri** field of your Db2 service credentials copy everything after db2:// (except the double quote at the end) and paste it in the cell below after ibm_db_sa://



in the following format

%sql ibm_db_sa://my-username:my-password@my-hostname:my-port/my-db-name

DB2 magic in case of new UI service credentials.

- Use the following format.
- Add security=SSL at the end

%sql ibm_db_sa://my-username:my-password@my-hostname:my-port/my-db-name?security=SSL

In [3]:
```
#jdbc:db2://fbd88901-ebdb-4a4f-a32e-9822b9fb237b.c1ogj3sd0tgtu0lqde00.databases.c
```

In [ ]:

```
```

In [7]:

```
# %sql ibm_db_sa://npj88637:U2doAY3Ir4HXvGQh@fbd88901-ebdb-4a4f-a32e-9822b9fb237b
```

Tasks

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

Task 1

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

In [8]:

```
sql SELECT DISTINCT LAUNCH_SITE FROM SPACEXTBL ORDER BY 1;
```

```
 * ibm_db_sa://fvp19040:***@54a2f15b-5c0f-46df-8954-7e38e612c2bd.c1ogj3sd0tgt
lqde00.databases.appdomain.cloud:32733/bludb
Done.
```

Out[8]:

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

Task 2

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

In [9]:
```
sql SELECT * FROM SPACEXTBL WHERE LAUNCH_SITE LIKE 'CCA%' LIMIT 5;
```

```
 * ibm_db_sa://fvp19040:***@54a2f15b-5c0f-46df-8954-7e38e612c2bd.c1ogj3sd0tgt
lqde00.databases.appdomain.cloud:32733/bludb
Done.
```

Out[9]:

DATE	time__utc_	booster_version	launch_site	payload	payload_mass__kg_	orbit	customer
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO
2012-05-22	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)
2012-10-08	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)

Task 3

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

In [10]:
```
sql SELECT SUM(PAYLOAD_MASS__KG_) AS TOTAL_PAYLOAD FROM SPACEXTBL WHERE PAYLOAD
```

```
 * ibm_db_sa://fvp19040:***@54a2f15b-5c0f-46df-8954-7e38e612c2bd.c1ogj3sd0tgt
lqde00.databases.appdomain.cloud:32733/bludb
Done.
```

Out[10]:

total_payload
111268

Task 4

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

In [11]:
```
sql SELECT AVG(PAYLOAD_MASS__KG_) AS AVG_PAYLOAD FROM SPACEXTBL WHERE BOOSTER_VER
```

```
 * ibm_db_sa://fvp19040:***@54a2f15b-5c0f-46df-8954-7e38e612c2bd.c1ogj3sd0tgt
lqde00.databases.appdomain.cloud:32733/bludb
Done.
```

Out[11]:

avg_payload
2928

Task 5

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

Hint:Use min function

In [13]:
```
sql SELECT MIN(DATE) AS FIRST_SUCCESS_GP FROM SPACEXTBL WHERE LANDING__OUTCOME =
```

```
 * ibm_db_sa://fvp19040:***@54a2f15b-5c0f-46df-8954-7e38e612c2bd.c1ogj3sd0tgt
lqde00.databases.appdomain.cloud:32733/bludb
Done.
```

Out[13]:

first_success_gp
2015-12-22

Task 6

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

In [14]:
```
sql SELECT DISTINCT BOOSTER_VERSION FROM SPACEXTBL WHERE PAYLOAD_MASS__KG_ BETWEE
```

```
 * ibm_db_sa://fvp19040:***@54a2f15b-5c0f-46df-8954-7e38e612c2bd.c1ogj3sd0tgt
lqde00.databases.appdomain.cloud:32733/bludb
Done.
```

Out[14]:

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

Task 7

List the total number of successful and failure mission outcomes

In [15]:
```
sql SELECT MISSION_OUTCOME, COUNT(*) AS QTY FROM SPACEXTBL GROUP BY MISSION_OUTCC
```

```
 * ibm_db_sa://fvp19040:***@54a2f15b-5c0f-46df-8954-7e38e612c2bd.c1ogj3sd0tgt
lqde00.databases.appdomain.cloud:32733/bludb
Done.
```

Out[15]:

mission_outcome	qty
Failure (in flight)	1
Success	99
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 [23]:
```
sql SELECT DISTINCT BOOSTER_VERSION FROM SPACEXTBL WHERE PAYLOAD_MASS__KG_ = (SEL
```

```
 * ibm_db_sa://fvp19040:***@54a2f15b-5c0f-46df-8954-7e38e612c2bd.c1ogj3sd0tgt
lqde00.databases.appdomain.cloud:32733/bludb
Done.
```

Out[23]:

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

Task 9

List the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015

In [24]:
```
sql SELECT BOOSTER_VERSION, LAUNCH_SITE FROM SPACEXTBL WHERE LANDING__OUTCOME =
```

```
 * ibm_db_sa://fvp19040:***@54a2f15b-5c0f-46df-8954-7e38e612c2bd.c1ogj3sd0tgt
lqde00.databases.appdomain.cloud:32733/bludb
Done.
```

Out[24]:

booster_version	launch_site
F9 v1.1 B1012	CCAFS LC-40
F9 v1.1 B1015	CCAFS LC-40

Task 10

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

In [25]:
```
sql SELECT LANDING__OUTCOME, COUNT(*) AS QTY FROM SPACEXTBL WHERE DATE BETWEEN '2
```

```
 * ibm_db_sa://fvp19040:***@54a2f15b-5c0f-46df-8954-7e38e612c2bd.c1ogj3sd0tgt
lqde00.databases.appdomain.cloud:32733/bludb
Done.
```

Out[25]:

landing__outcome	qty
No attempt	10
Failure (drone ship)	5
Success (drone ship)	5
Controlled (ocean)	3
Success (ground pad)	3
Failure (parachute)	2
Uncontrolled (ocean)	2
Precluded (drone ship)	1

Reference Links

- Hands-on Lab : String Patterns, Sorting and Grouping (https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DB0201EN-SkillsNetwork/labs/Labs_Coursera_V5/labs/Lab%20-%20String%20Patterns%20-%20Sorting%20-%20Grouping/instructional-labs.md.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=100SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01&origin=www.coursera.org)
- Hands-on Lab: Built-in functions (https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DB0201EN-SkillsNetwork/labs/Labs_Coursera_V5/labs/Lab%20-%20Built-in%20functions%20/Hands-

on_Lab__Built-in_Functions.md.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=100SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01&origin=www.coursera.org)

- Hands-on Lab : Sub-queries and Nested SELECT Statements (https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DB0201EN-SkillsNetwork/labs/Labs_Coursera_V5/labs/Lab%20-%20Sub-queries%20and%20Nested%20SELECTs%20/instructional-labs.md.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=100SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01&origin=www.coursera.org)
- Hands-on Tutorial: Accessing Databases with SQL magic (https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DB0201EN-SkillsNetwork/labs/Module%205/DB0201EN-Week3-1-3-SQLmagic.ipynb?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=100SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01)
- Hands-on Lab: Analyzing a real World Data Set (https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DB0201EN-SkillsNetwork/labs/Module%205/DB0201EN-Week3-1-4-Analyzing.ipynb?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=100SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01)

Author(s)

Lakshmi Holla

Other Contributors

Rav Ahuja

Change log

Date	Version	Changed by	Change Description
2021-08-24	0.3	Lakshmi Holla	Added library update
2021-07-09	0.2	Lakshmi Holla	Changes made in magic sql
2021-05-20	0.1	Lakshmi Holla	Created Initial Version

cognitiveclass.ai logo

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]:
```
# andas is a software library written for the Python programming language for dat
import pandas as pd
#NumPy is a library for the Python programming language, adding support for large
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a MatLab like p
import matplotlib.pyplot as plt
#Seaborn is a Python data visualization library based on matplotlib. It provides
import seaborn as sns
```

Exploratory Data Analysis

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

In [2]:
```
df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.clou

# If you were unable to complete the previous lab correctly you can uncomment and

# df = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.

df.head(5)
```

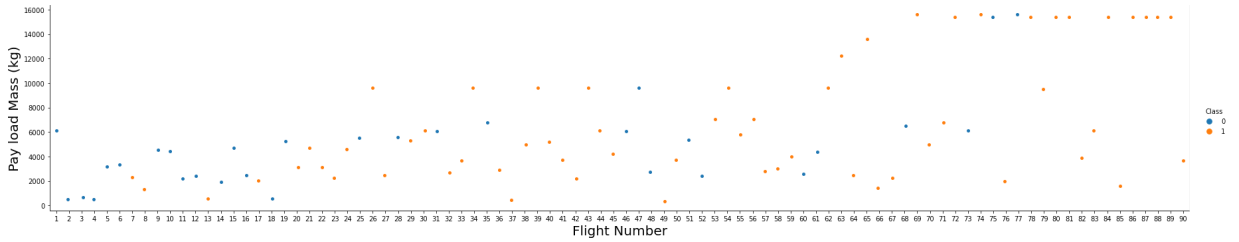
Out[2]:

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

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 [3]:
```
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()
```



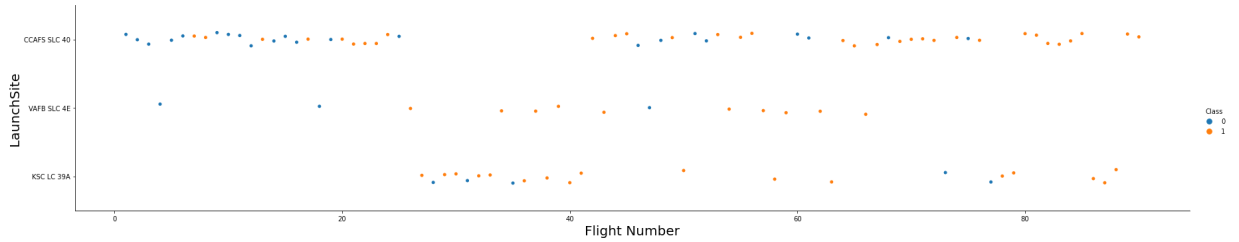
We see that different launch sites have different success rates. `CCAFS LC-40`, has a success rate of 60 %, while `KSC LC-39A` and `VAFB SLC 4E` has a success rate of 77%.

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

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

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

In [4]:
```
# Plot a scatter point chart with x axis to be Flight Number and y axis to be the
sns.catplot(y="LaunchSite", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("LaunchSite",fontsize=20)
plt.show()
```

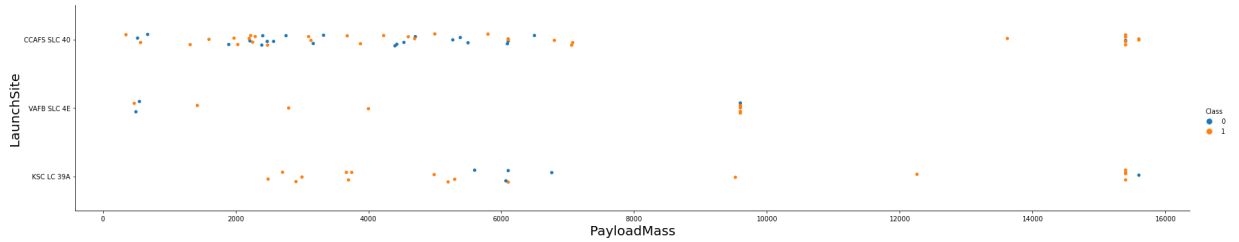


Now try to explain the patterns you found in the Flight Number vs. Launch Site scatter point plots.

TASK 2: Visualize the relationship between Payload and Launch Site

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

In [5]:
```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to b
sns.catplot(y="LaunchSite", x="PayloadMass", hue="Class", data=df, aspect = 5)
plt.xlabel("PayloadMass",fontsize=20)
plt.ylabel("LaunchSite",fontsize=20)
plt.show()
```



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

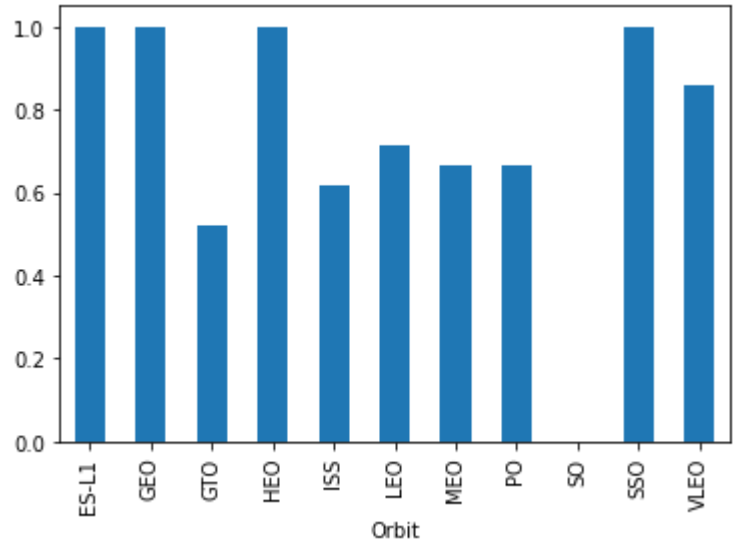
TASK 3: Visualize the relationship between success rate of each orbit type

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 [6]:
```
# HINT use groupby method on Orbit column and get the mean of Class column
df.groupby('Orbit')['Class'].mean().plot.bar()
```

Out[6]:
```
<AxesSubplot:xlabel='Orbit'>
```

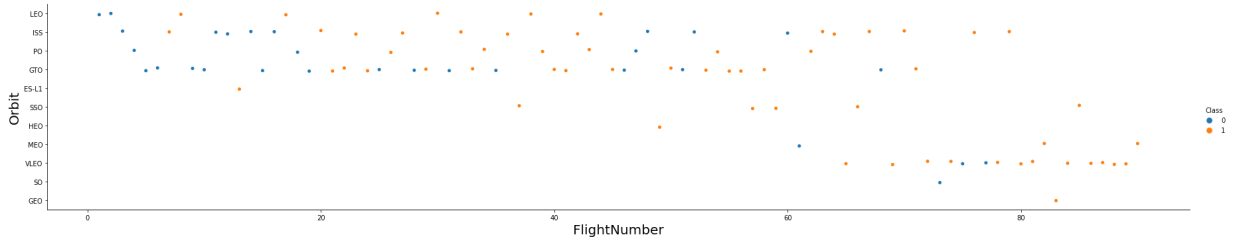


Analyze the ploted bar chart try to find which orbits have high sucess rate.

TASK 4: Visualize the relationship between FlightNumber and Orbit type

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

In [7]:
```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the
sns.catplot(y="Orbit", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("FlightNumber",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```

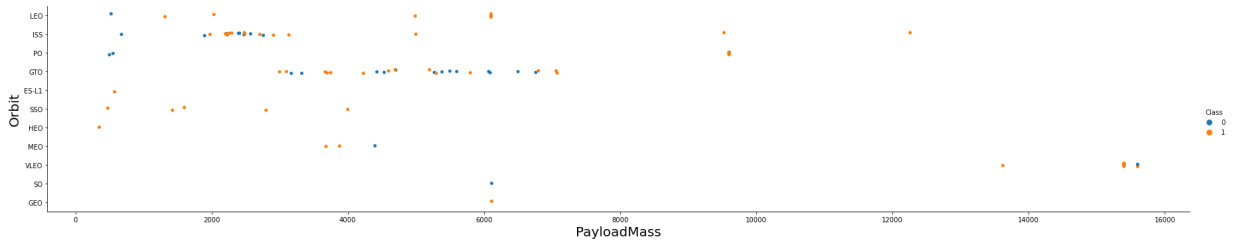


You should see that in the LEO orbit the Success appears related to the number of flights; on the other hand, there seems to be no relationship between flight number when in GTO orbit.

TASK 5: Visualize the relationship between Payload and Orbit type

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

In [8]:
```
# Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit
sns.catplot(y="Orbit", x="PayloadMass", hue="Class", data=df, aspect = 5)
plt.xlabel("PayloadMass",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```



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

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

TASK 6: Visualize the launch success yearly trend

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

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

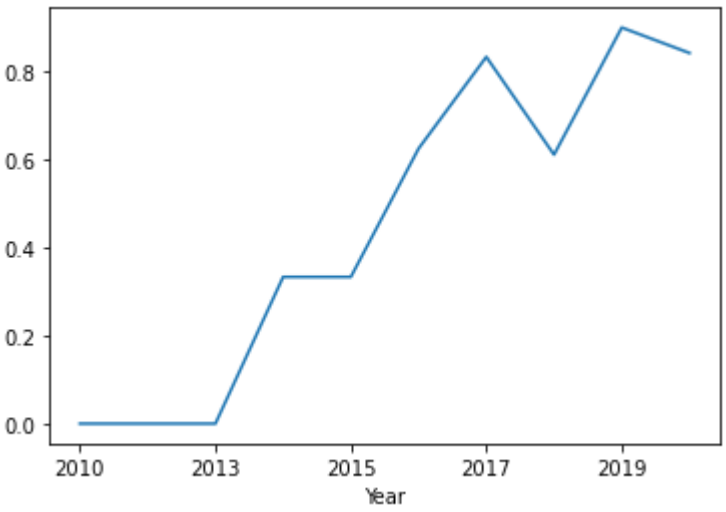
In [9]:
```
# A function to Extract years from the date
year=[]
def Extract_year(date):
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year

Extract_year(df['Date'])
```

Out[9]:
```
['2010',
 '2012',
 '2013',
 '2013',
 '2013',
 '2014',
 '2014',
 '2014',
 '2014',
 '2014',
 '2014',
 '2014',
 '2015',
 '2015',
 '2015',
 '2015',
 '2015',
 '2015',
 '2015',
 '2016',
 '2016',
 '2016',
 '2016',
 '2016',
 '2016',
 '2016',
 '2016',
 '2016',
 '2017',
 '2017',
 '2017',
 '2017',
 '2017',
 '2017',
 '2017',
 '2017',
 '2017',
 '2017',
 '2017',
 '2017',
 '2017',
 '2017',
 '2017',
 '2017',
 '2017',
 '2017',
 '2017',
 '2017',
 '2017',
 '2018',
 '2018',
 '2018',
 '2018',
```

```
'2018',
'2018',
'2018',
'2018',
'2018',
'2018',
'2018',
'2018',
'2018',
'2018',
'2018',
'2018',
'2018',
'2018',
'2019',
'2019',
'2019',
'2019',
'2019',
'2019',
'2019',
'2019',
'2019',
'2019',
'2020',
'2020',
'2020',
'2020',
'2020',
'2020',
'2020',
'2020',
'2020',
'2020',
'2020',
'2020',
'2020',
'2020',
'2020',
'2020',
'2020',
'2020',
'2020',
'2020']
```

In [10]:
```
# Plot a line chart with x axis to be the extracted year and y axis to be the suc
temp_df = df.copy()
temp_df['Year'] = year
temp_df.groupby('Year')['Class'].mean().plot()
```

Out[10]: <AxesSubplot:xlabel='Year'>



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

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 [11]:
```
features = df[['FlightNumber', 'PayloadMass', 'Orbit', 'LaunchSite', 'Flights', '
features.head()
```

Out[11]:

	FlightNumber	PayloadMass	Orbit	LaunchSite	Flights	GridFins	Reused	Legs	LandingPad	E
0	1	6104.959412	LEO	CCAFS SLC 40	1	False	False	False	NaN	
1	2	525.000000	LEO	CCAFS SLC 40	1	False	False	False	NaN	
2	3	677.000000	ISS	CCAFS SLC 40	1	False	False	False	NaN	
3	4	500.000000	PO	VAFB SLC 4E	1	False	False	False	NaN	
4	5	3170.000000	GTO	CCAFS SLC 40	1	False	False	False	NaN	

TASK 7: Create dummy variables to categorical columns

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

In [12]:
```
# HINT: Use get_dummies() function on the categorical columns
features_one_hot = pd.get_dummies(df, columns=['Orbit', 'LaunchSite','LandingPad'
features_one_hot.head()
```

Out[12]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Outcome	Flights	GridFins	Reused	Legs
0	1	2010-06-04	Falcon 9	6104.959412	None None	1	False	False	False
1	2	2012-05-22	Falcon 9	525.000000	None None	1	False	False	False
2	3	2013-03-01	Falcon 9	677.000000	None None	1	False	False	False
3	4	2013-09-29	Falcon 9	500.000000	False Ocean	1	False	False	False
4	5	2013-12-03	Falcon 9	3170.000000	None None	1	False	False	False

5 rows × 86 columns

TASK 8: Cast all numeric columns to `float64`

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

In [13]:
```
# HINT: use astype function
features_one_hot = features_one_hot.astype('float64', errors='ignore')
```

In [14]:
```
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

Joseph Santarcangelo (https://www.linkedin.com/in/joseph-s-50398b136/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01) has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Nayef Abou Tayoun (https://www.linkedin.com/in/nayefaboutayoun/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01) is a Data Scientist at IBM and pursuing a Master of Management in Artificial intelligence degree at Queen's University.

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-10-12	1.1	Lakshmi Holla	Modified markdown
2020-09-20	1.0	Joseph	Modified Multiple Areas
2020-11-10	1.1	Nayef	updating the input data

cognitiveclass.ai logo

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 [2]:
```
!pip install folium==0.7.0
!pip3 install wget
```

```
Collecting folium==0.7.0
  Downloading folium-0.7.0-py3-none-any.whl (85 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 85.5/85.5 KB 6.9 MB/s eta 0:00:00
Requirement already satisfied: jinja2 in /home/jupyterlab/conda/envs/python/li
b/python3.7/site-packages (from folium==0.7.0) (3.1.1)
Requirement already satisfied: branca>=0.3.0 in /home/jupyterlab/conda/envs/pyt
hon/lib/python3.7/site-packages (from folium==0.7.0) (0.5.0)
Requirement already satisfied: numpy in /home/jupyterlab/conda/envs/python/lib/
python3.7/site-packages (from folium==0.7.0) (1.21.6)
Requirement already satisfied: requests in /home/jupyterlab/conda/envs/python/l
ib/python3.7/site-packages (from folium==0.7.0) (2.27.1)
Requirement already satisfied: six in /home/jupyterlab/conda/envs/python/lib/py
thon3.7/site-packages (from folium==0.7.0) (1.16.0)
Requirement already satisfied: MarkupSafe>=2.0 in /home/jupyterlab/conda/envs/p
ython/lib/python3.7/site-packages (from jinja2->folium==0.7.0) (2.1.1)
Requirement already satisfied: certifi>=2017.4.17 in /home/jupyterlab/conda/env
s/python/lib/python3.7/site-packages (from requests->folium==0.7.0) (2021.10.8)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /home/jupyterlab/conda/
envs/python/lib/python3.7/site-packages (from requests->folium==0.7.0) (1.26.9)
Requirement already satisfied: idna<4,>=2.5 in /home/jupyterlab/conda/envs/pyth
on/lib/python3.7/site-packages (from requests->folium==0.7.0) (3.3)
Requirement already satisfied: charset-normalizer~=2.0.0 in /home/jupyterlab/co
nda/envs/python/lib/python3.7/site-packages (from requests->folium==0.7.0) (2.
0.12)
Installing collected packages: folium
  Attempting uninstall: folium
    Found existing installation: folium 0.5.0
    Uninstalling folium-0.5.0:
      Successfully uninstalled folium-0.5.0
Successfully installed folium-0.7.0
Requirement already satisfied: wget in /home/jupyterlab/conda/envs/python/lib/p
ython3.7/site-packages (3.2)
```

In [3]:
```
import folium
import wget
import pandas as pd
```

In [4]:
```
# Import folium MarkerCluster plugin
from folium.plugins import MarkerCluster
# Import folium MousePosition plugin
from folium.plugins import MousePosition
# Import folium DivIcon plugin
from folium.features import DivIcon
```

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

Generating Maps with Python (https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/labs/module_3/DV0101EN-3-5-1-

Generating-Maps-in-Python-py-v2.0.ipynb)

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 [5]:
```
# Download and read the `spacex_launch_geo.csv`
spacex_csv_file = wget.download('https://cf-courses-data.s3.us.cloud-object-stora
spacex_df=pd.read_csv(spacex_csv_file)
```

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

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

Out[6]:

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

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

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

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

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

In [8]:
```
# Create a blue circle at NASA Johnson Space Center's coordinate with a popup lab
circle = folium.Circle(nasa_coordinate, radius=1000, color='#d35400', fill=True).
# Create a blue circle at NASA Johnson Space Center's coordinate with a icon show
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
        )
    )
site_map.add_child(circle)
site_map.add_child(marker)
```

Out[8]: Make this Notebook Trusted to load map: File -> Trust Notebook

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(...))
```

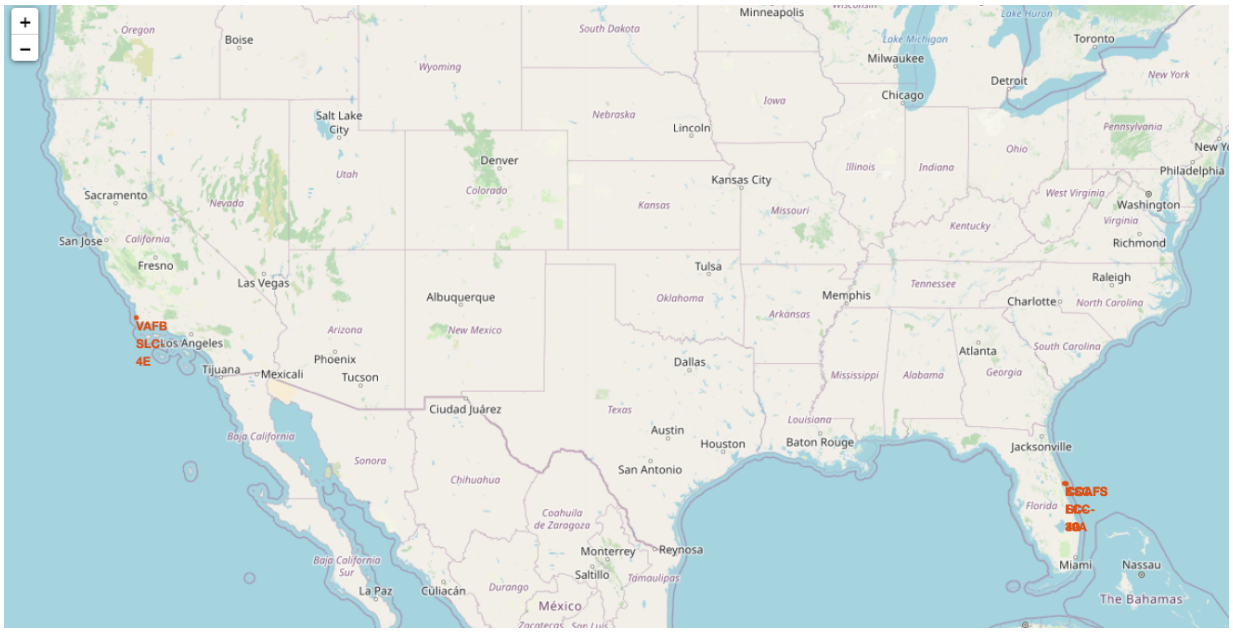
An example of folium.Marker:

```
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 [9]:
```
# Initial the map
site_map = folium.Map(location=nasa_coordinate, zoom_start=5)
# For each launch site, add a Circle object based on its coordinate (Lat, Long) v
for ix, row in launch_sites_df.iterrows():
    ls_name = row['Launch Site']
    ls_lat  = row['Lat']
    ls_long = row['Long']

    #print('{}\t{}\t{}'.format(ls_name, ls_lat, ls_long))

    coordinate = [ls_lat, ls_long]
    circle = folium.Circle(coordinate, radius=1000, color='#000000', fill=True).a
    marker = folium.map.Marker(coordinate, icon=DivIcon(icon_size=(20,20),icon_ar

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

site_map
```

Out[9]: 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.

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

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

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

Out[10]:

	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

In [11]:
```
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 [12]:
```
# 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
```

In [13]:
```
# Function to assign color to launch outcome
def assign_marker_color(launch_outcome):
    if launch_outcome == 1:
        return 'green'
    else:
        return 'red'

spacex_df['marker_color'] = spacex_df['class'].apply(assign_marker_color)
spacex_df.tail(10)
```

Out[13]:

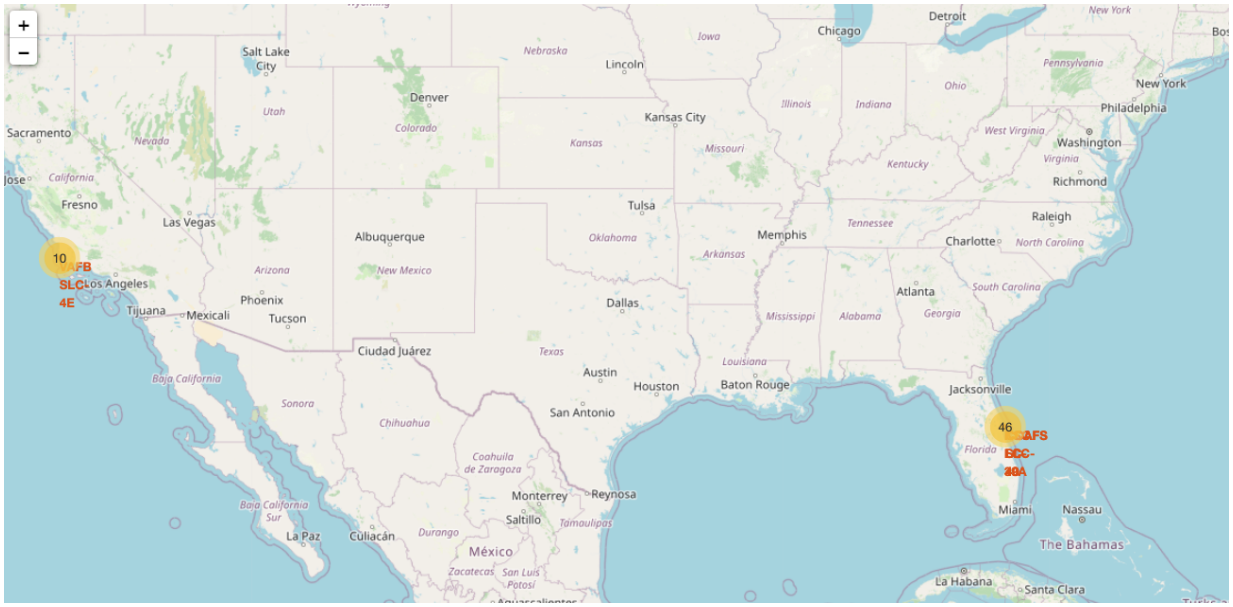
	Launch Site	Lat	Long	class	marker_color
46	KSC LC-39A	28.573255	-80.646895	1	green
47	KSC LC-39A	28.573255	-80.646895	1	green
48	KSC LC-39A	28.573255	-80.646895	1	green
49	CCAFS SLC-40	28.563197	-80.576820	1	green
50	CCAFS SLC-40	28.563197	-80.576820	1	green
51	CCAFS SLC-40	28.563197	-80.576820	0	red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

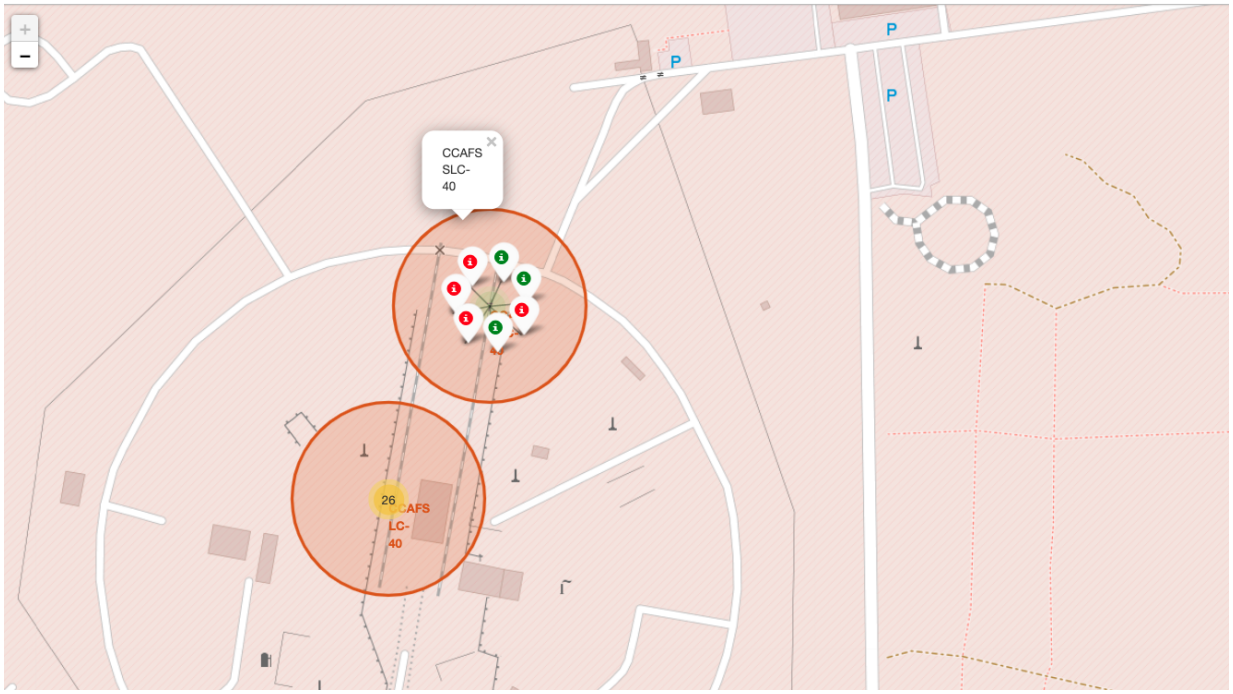
TODO: For each launch result in `spacex_df` data frame, add a `folium.Marker` to `marker_cluster`

In [16]:
```
# 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 success
# e.g., icon=folium.Icon(color='white', icon_color=row['marker_color']
for index, record in spacex_df.iterrows():
    ls_name = record['Launch Site']
    ls_lat  = record['Lat']
    ls_long = record['Long']
    coordinate = [ls_lat, ls_long]

    #rint('{}\t{}\t{}'.format(ls_name, ls_lat, ls_long))

    # TODO: Create and add a Marker cluster to the site map
    # marker = folium.Marker(...)
    marker = folium.map.Marker(coordinate, icon=folium.Icon(color='white', icon_c
    marker_cluster.add_child(marker)

site_map
```

Out[16]: 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.

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 [17]:
```
# Add Mouse Position to get the coordinate (Lat, Long) for a mouse over on the mc
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[17]: 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.

You can calculate the distance between two points on the map based on their `Lat` and `Long` values using the following method:

In [18]:
```
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 [19]:
```
# find coordinate of the closet coastline
# e.g.,: Lat: 28.56367  Lon: -80.57163
# distance_coastline = calculate_distance(launch_site_lat, launch_site_lon, coast
distance = calculate_distance(28.57468,-80.65229,28.573255 ,-80.646895)
distance
```

Out[19]:
```
0.5503149993453544
```

TODO: After obtained its coordinate, create a `folium.Marker` to show the distance

In [20]:
```
# Create and add a folium.Marker on your selected closest coastline point on the
# Display the distance between coastline point and launch site using the icon pro
# for example
coordinate = [28.57468,-80.65229]
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
    )
)
site_map.add_child(distance_marker)
site_map
```

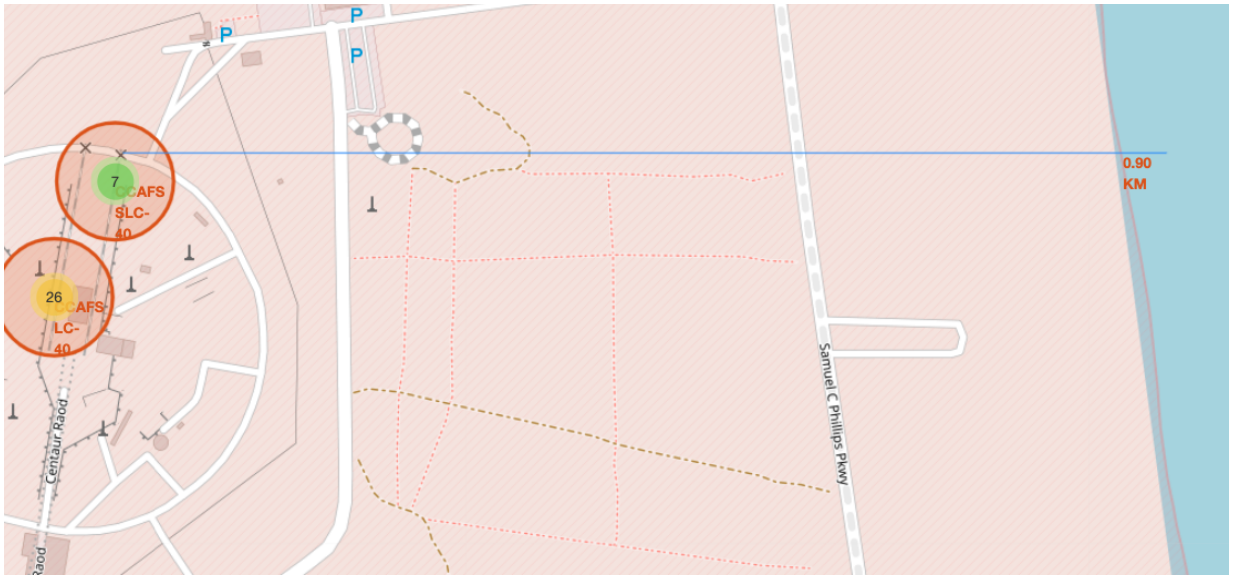
Out[20]: Make this Notebook Trusted to load map: File -> Trust Notebook

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

In [21]:
```
# Create a `folium.PolyLine` object using the coastline coordinates and launch si
# lines=folium.PolyLine(locations=coordinates, weight=1)
coordinates=[[28.57468,-80.65229],[28.573255 ,-80.646895]]
lines=folium.PolyLine(locations=coordinates, weight=1)
site_map.add_child(lines)
```

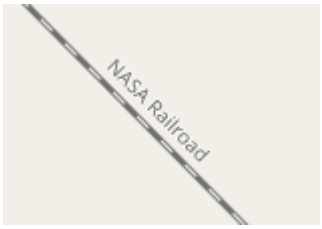
Out[21]: Make this Notebook Trusted to load map: File -> Trust Notebook

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

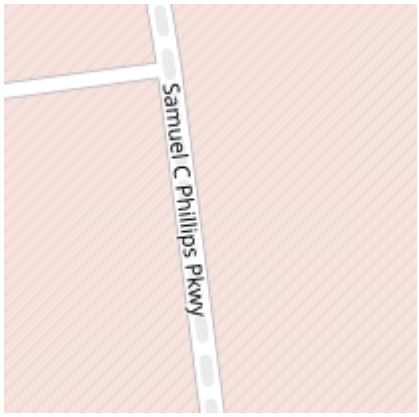


TODO: Similarly, you can draw a line betwee a launch site to its closest city, railway, highway, etc. You need to use `MousePosition` to find the their coordinates on the map first

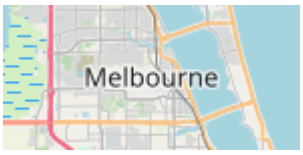
A railway map symbol may look like this:



A highway map symbol may look like this:



A city map symbol may look like this:



In [22]:
```
# Create a marker with distance to a closest city, railway, highway, etc.
# Draw a line between the marker to the launch site
coordinates=[[28.52361, -80.64857],[28.573255 ,-80.646895]]
lines=folium.PolyLine(locations=coordinates, weight=1)
site_map.add_child(lines)

distance_marker = folium.Marker(
    coordinates[0],
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:16
    )
)
site_map.add_child(distance_marker)
```

Out[22]: Make this Notebook Trusted to load map: File -> Trust Notebook

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 Ploty Dash on detailed launch records.

Authors

Yan Luo (https://www.linkedin.com/in/yan-luo-96288783/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01)

Other Contributors

Joseph Santarcangelo

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-05-26	1.0	Yan	Created the initial version

 cognitiveclass.ai logo

Space X Falcon 9 First Stage Landing Prediction

Assignment: Machine Learning Prediction

Estimated time needed: **60** minutes

Space X advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is because Space X can reuse the first stage. Therefore if we can determine if the first stage will land, we can determine the cost of a launch. This information can be used if an alternate company wants to bid against space X for a rocket launch. In this lab, you will create a machine learning pipeline to predict if the first stage will land given the data from the preceding labs.

Several examples of an unsuccessful landing are shown here:

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

Objectives

Perform exploratory Data Analysis and determine Training Labels

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

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

- Find the method performs best using test data

Type *Markdown* and LaTeX: α^2

Import Libraries and Define Auxiliary Functions

We will import the following libraries for the lab

In [1]:
```
# Pandas is a software library written for the Python programming language for da
import pandas as pd
# NumPy is a library for the Python programming language, adding support for larg
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a MatLab like p
import matplotlib.pyplot as plt
#Seaborn is a Python data visualization library based on matplotlib. It provides
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 [2]:
```
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(['
```

Load the dataframe

Load the data

In [3]:
```
data = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.

# If you were unable to complete the previous lab correctly you can uncomment and

# data = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomai

data.head()
```

Out[3]:

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

In [4]:
```
X = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.clo

# If you were unable to complete the previous lab correctly you can uncomment and

# X = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.c

X.head(100)
```

Out[4]:

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

90 rows × 83 columns

TASK 1

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

In [5]:
```
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 [6]:
```
# students get this
transform = preprocessing.StandardScaler()
```

In [7]:
```
X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))
X[0:3]
```

Out[7]:
```
array([[-1.71291154e+00, -1.94814463e-16, -6.53912840e-01,
        -1.57589457e+00, -9.73440458e-01, -1.05999788e-01,
        -1.05999788e-01, -6.54653671e-01, -1.05999788e-01,
        -5.51677284e-01,  3.44342023e+00, -1.85695338e-01,
        -3.33333333e-01, -1.05999788e-01, -2.42535625e-01,
        -4.29197538e-01,  7.97724035e-01, -5.68796459e-01,
        -4.10890702e-01, -4.10890702e-01, -1.50755672e-01,
        -7.97724035e-01, -1.50755672e-01, -3.92232270e-01,
         9.43398113e+00, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
        -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
        -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.50755672e-01,
        -1.50755672e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -2.15665546e-01,
        -1.85695338e-01, -2.15665546e-01, -2.67261242e-01,
        -1.05999788e-01, -2.42535625e-01, -1.05999788e-01,
        -2.15665546e-01, -1.85695338e-01, -2.15665546e-01,
        -1.85695338e-01, -1.05999788e-01,  1.87082869e+00,
        -1.87082869e+00,  8.35531692e-01, -8.35531692e-01,
         1.93309133e+00, -1.93309133e+00],
       [-1.67441914e+00, -1.19523159e+00, -6.53912840e-01,
        -1.57589457e+00, -9.73440458e-01, -1.05999788e-01,
        -1.05999788e-01, -6.54653671e-01, -1.05999788e-01,
        -5.51677284e-01,  3.44342023e+00, -1.85695338e-01,
        -3.33333333e-01, -1.05999788e-01, -2.42535625e-01,
        -4.29197538e-01,  7.97724035e-01, -5.68796459e-01,
        -4.10890702e-01, -4.10890702e-01, -1.50755672e-01,
        -7.97724035e-01, -1.50755672e-01, -3.92232270e-01,
        -1.05999788e-01,  9.43398113e+00, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
        -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
        -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.50755672e-01,
        -1.50755672e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -2.15665546e-01,
        -1.85695338e-01, -2.15665546e-01, -2.67261242e-01,
        -1.05999788e-01, -2.42535625e-01, -1.05999788e-01,
        -2.15665546e-01, -1.85695338e-01, -2.15665546e-01,
```

```
 -1.85695338e-01, -1.05999788e-01,  1.87082869e+00,
 -1.87082869e+00,  8.35531692e-01, -8.35531692e-01,
  1.93309133e+00, -1.93309133e+00],
[-1.63592675e+00, -1.16267307e+00, -6.53912840e-01,
 -1.57589457e+00, -9.73440458e-01, -1.05999788e-01,
 -1.05999788e-01, -6.54653671e-01, -1.05999788e-01,
  1.81265393e+00, -2.90408935e-01, -1.85695338e-01,
 -3.33333333e-01, -1.05999788e-01, -2.42535625e-01,
 -4.29197538e-01,  7.97724035e-01, -5.68796459e-01,
 -4.10890702e-01, -4.10890702e-01, -1.50755672e-01,
 -7.97724035e-01, -1.50755672e-01, -3.92232270e-01,
 -1.05999788e-01, -1.05999788e-01,  9.43398113e+00,
 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
 -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
 -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
 -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
 -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
 -1.05999788e-01, -1.50755672e-01, -1.50755672e-01,
 -1.50755672e-01, -1.05999788e-01, -1.05999788e-01,
 -1.05999788e-01, -1.50755672e-01, -2.15665546e-01,
 -1.85695338e-01, -2.15665546e-01, -2.67261242e-01,
 -1.05999788e-01, -2.42535625e-01, -1.05999788e-01,
 -2.15665546e-01, -1.85695338e-01, -2.15665546e-01,
 -1.85695338e-01, -1.05999788e-01,  1.87082869e+00,
 -1.87082869e+00,  8.35531692e-01, -8.35531692e-01,
  1.93309133e+00, -1.93309133e+00]])
```

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 [8]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_s
```

we can see we only have 18 test samples.

In [9]:
```
Y_test.shape
```

Out[9]:
```
(18,)
```

TASK 4

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

In [10]:
```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

In [11]:
```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2
lr=LogisticRegression()

logreg_cv = GridSearchCV(estimator=lr, cv=10, param_grid=parameters)
logreg_cv.fit(X_train, Y_train)
```

Out[11]:
```
GridSearchCV(cv=10, estimator=LogisticRegression(),
             param_grid={'C': [0.01, 0.1, 1], 'penalty': ['l2'],
                         'solver': ['lbfgs']})
```

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

In [12]:
```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

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

TASK 5

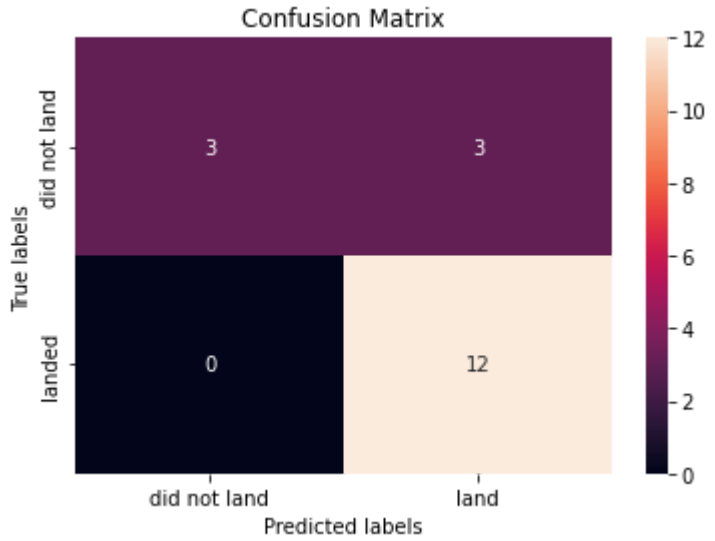
Calculate the accuracy on the test data using the method `score`:

In [13]:
```
print("accuracy :",logreg_cv.score(X_test, Y_test))
```

```
accuracy : 0.8333333333333334
```

Lets look at the confusion matrix:

In [14]:
```
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 [15]:
```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

In [16]:
```
svm_cv = GridSearchCV(estimator=svm, cv=10, param_grid=parameters)
svm_cv.fit(X_train, Y_train)
```

Out[16]:
```
GridSearchCV(cv=10, estimator=SVC(),
             param_grid={'C': array([1.00000000e-03, 3.16227766e-02, 1.00000000
e+00, 3.16227766e+01,
       1.00000000e+03]),
                         'gamma': array([1.00000000e-03, 3.16227766e-02, 1.0000
0000e+00, 3.16227766e+01,
       1.00000000e+03]),
                         'kernel': ('linear', 'rbf', 'poly', 'rbf', 'sigmoi
d')})
```

In [17]:
```
print("tuned hpyerparameters :(best parameters) ",svm_cv.best_params_)
print("accuracy :",svm_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.0316227766016
79, 'kernel': 'sigmoid'}
accuracy : 0.8482142857142856
```

TASK 7

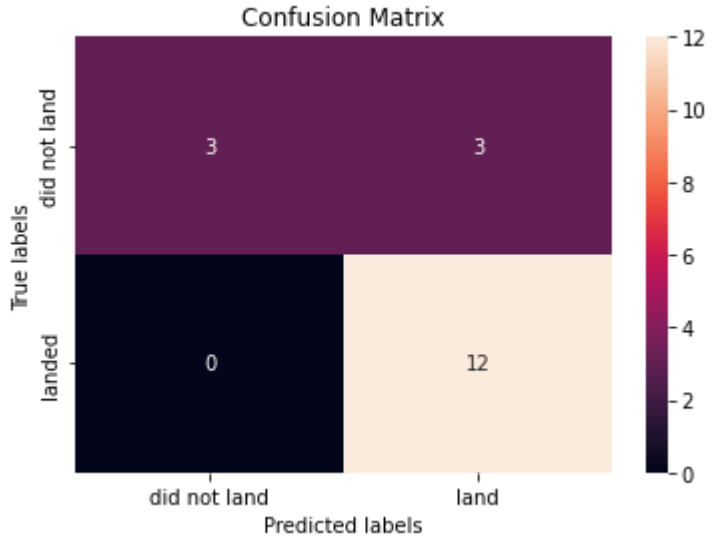
Calculate the accuracy on the test data using the method `score` :

In [18]:
```
print("accuracy :", svm_cv.score(X_test, Y_test))
```

```
accuracy : 0.8333333333333334
```

We can plot the confusion matrix

In [19]:
```
yhat=svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



TASK 8

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

In [20]:
```
parameters = {'criterion': ['gini', 'entropy'],
     'splitter': ['best', 'random'],
     'max_depth': [2*n for n in range(1,10)],
     'max_features': ['auto', 'sqrt'],
     'min_samples_leaf': [1, 2, 4],
     'min_samples_split': [2, 5, 10]}

tree = DecisionTreeClassifier()
```

In [21]:
```
tree_cv = GridSearchCV(estimator=tree, cv=10, param_grid=parameters)
tree_cv.fit(X_train, Y_train)
```

Out[21]:
```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18],
                         'max_features': ['auto', 'sqrt'],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'splitter': ['best', 'random']})
```

In [22]:
```
print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', 'max_depth
6, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 5, 'spli
tter': 'best'}
accuracy : 0.8892857142857145
```

TASK 9

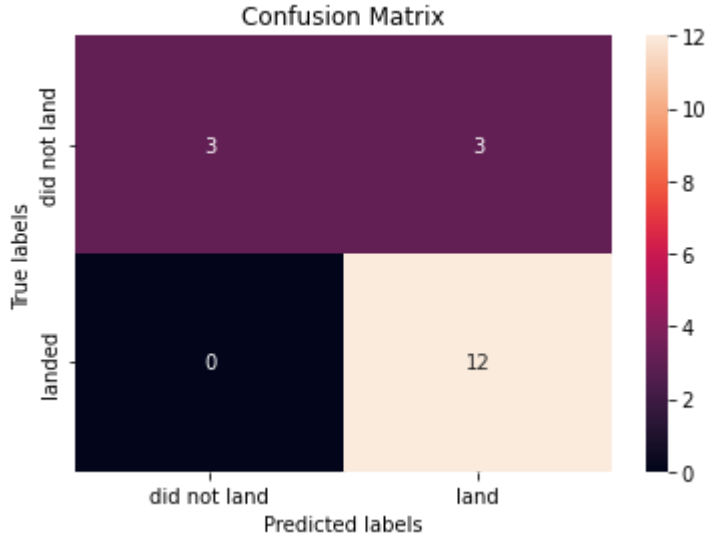
Calculate the accuracy of tree_cv on the test data using the method `score` :

In [23]:
```
print("accuracy :", tree_cv.score(X_test, Y_test))
```

```
accuracy : 0.6111111111111112
```

We can plot the confusion matrix

In [24]:
```
yhat = svm_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]:
```
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()
```

In [26]:
```
knn_cv = GridSearchCV(estimator=KNN, cv=10, param_grid=parameters)
knn_cv.fit(X_train, Y_train)
```

Out[26]:
```
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brut
e'],
                         'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'p': [1, 2]})
```

In [27]:
```
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

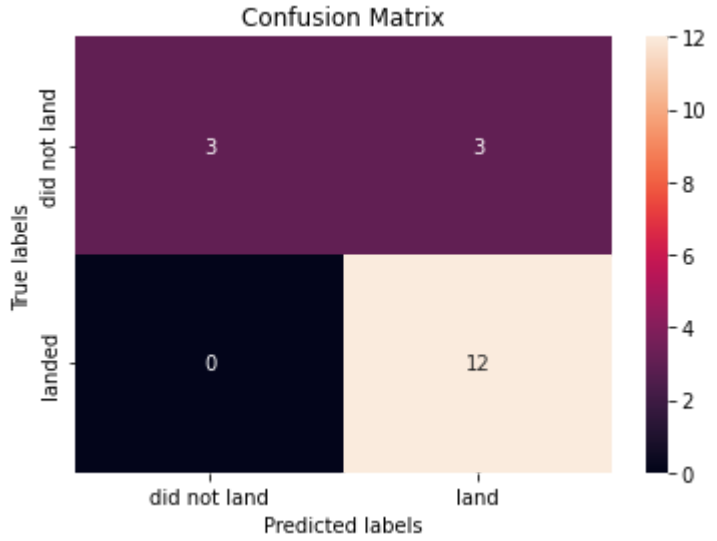
Calculate the accuracy of tree_cv on the test data using the method `score` :

In [28]:
```
print("accuracy :", knn_cv.score(X_test, Y_test))
```

```
accuracy : 0.8333333333333334
```

We can plot the confusion matrix

In [29]:
```
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



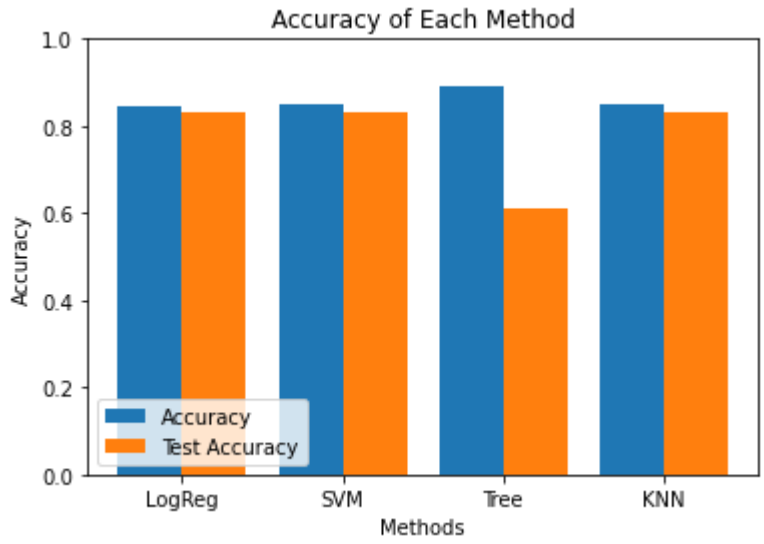
TASK 12

Find the method performs best:

In [30]:
```
print("Model\t\tAccuracy\tTestAccuracy")#,logreg_cv.best_score_)
print("LogReg\t\t{}\t\t{}".format((logreg_cv.best_score_).round(5), logreg_cv.sc
print("SVM\t\t{}\t\t{}".format((svm_cv.best_score_).round(5), svm_cv.score(X_test
print("Tree\t\t{}\t\t{}".format((tree_cv.best_score_).round(5), tree_cv.score(X_t
print("KNN\t\t{}\t\t{}".format((knn_cv.best_score_).round(5), knn_cv.score(X_test

comparison = {}

comparison['LogReg'] = {'Accuracy': logreg_cv.best_score_.round(5), 'TestAccuracy
comparison['SVM'] = {'Accuracy': svm_cv.best_score_.round(5), 'TestAccuracy': svm
comparison['Tree'] = {'Accuracy': tree_cv.best_score_.round(5), 'TestAccuracy': t
comparison['KNN'] = {'Accuracy': knn_cv.best_score_.round(5), 'TestAccuracy': knn
```

```
Model           Accuracy        TestAccuracy
LogReg          0.84643         0.83333
SVM             0.84821         0.83333
Tree            0.88929         0.61111
KNN             0.84821         0.83333
```

In [31]:
```
x = []
y1 = []
y2 = []
for meth in comparison.keys():
    x.append(meth)
    y1.append(comparison[meth]['Accuracy'])
    y2.append(comparison[meth]['TestAccuracy'])


x_axis = np.arange(len(x))

plt.bar(x_axis - 0.2, y1, 0.4, label = 'Accuracy')
plt.bar(x_axis + 0.2, y2, 0.4, label = 'Test Accuracy')

plt.ylim([0,1])
plt.xticks(x_axis, x)

plt.xlabel("Methods")
plt.ylabel("Accuracy")
plt.title("Accuracy of Each Method")
plt.legend(loc='lower left')
plt.show()
```



Authors

Joseph Santarcangelo (https://www.linkedin.com/in/joseph-s-50398b136/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=100065SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01) has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-08-31	1.1	Lakshmi Holla	Modified markdown
2020-09-20	1.0	Joseph	Modified Multiple Areas