Applied Data Science Capstone

IBM: Data Science Professional Certificate

Course 10: Applied Data Science Capstone

Week 1: Introduction

Data Collection Overview

- This project will be utilizing data from the SpaceX launch that is gathered via the SpaceX REST API.

Data Collection via API:

1. get the URL of the data we want
2. perform a get request using the request library to obtain the launch data
3. view the result using the .json() method
4. convert the .json to a dataframe using the .json_normalize() function

Data Collection via Web Scraping:

1. Webscrape with BeautifulSoup
2. Parse the data into readable tables

Lab: Collecting the Data (API)

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.

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

In [1]:

```
# Requests allows us to make HTTP requests which we will use to get data from an API
import requests
# Pandas is a software library written for the Python programming language for data man
import pandas as pd
# NumPy is a library for the Python programming language, adding support for large, mul
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)
```

```
c:\users\orgil\appdata\local\programs\python\python39\lib\site-packages\numpy\_distribut
or_init.py:30: UserWarning: loaded more than 1 DLL from .libs:
c:\users\orgil\appdata\local\programs\python\python39\lib\site-packages\numpy\.libs\libo
penblas.GK7GX5KEQ4F6UYO3P26ULGBQYHGQO7J4.gfortran-win_amd64.dll
c:\users\orgil\appdata\local\programs\python\python39\lib\site-packages\numpy\.libs\libo
penblas.XWYDX2IKJW2NMTWSFYNGFUWKQU3LYTCZ.gfortran-win_amd64.dll
  warnings.warn("loaded more than 1 DLL from .libs:"
```

In [2]:

```
pd.options.mode.chained_assignment = None  # default='warn'
```

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 data to t
def getBoosterVersion(data):
    for x in data['rocket']:
```

```
        response = requests.get("https://api.spacexdata.com/v4/rockets/"+str(x)).json()
        BoosterVersion.append(response['name'])
```

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

In [4]:

```
# Takes the dataset and uses the launchpad column to call the API and append the data t
def getLaunchSite(data):
    for x in data['launchpad']:
        response = requests.get("https://api.spacexdata.com/v4/launchpads/"+str(x)).jso
        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 data to
def getPayloadData(data):
    for load in data['payloads']:
        response = requests.get("https://api.spacexdata.com/v4/payloads/"+load).json()
        PayloadMass.append(response['mass_kg'])
        Orbit.append(response['orbit'])
```

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 to th
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cores/"+core['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_type']))
            Flights.append(core['flight'])
            GridFins.append(core['gridfins'])
            Reused.append(core['reused'])
            Legs.append(core['legs'])
            LandingPad.append(core['landpad'])
```

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

In [7]:

```
# get dataset URL
spacex_url="https://api.spacexdata.com/v4/launches/past"
```

In [8]:

```
# complete get request
```

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

In [9]:
```
# view the content of the response
# print(response.content)
```

Task 1: Request and parse the SpaceX launch data using the get request

To make the requested JSON results more consistent, we will use the following static response object for this project:

In [10]:
```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM
```

In [11]:
```
# view if the request was successful (status code 200)
response.status_code
```

Out[11]: 200

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

In [12]:
```
# use .json_normalize method to convert the json into a dataframe
data = pd.json_normalize(response.json())
```

In [13]:
```
# get the head of this dataframe
data.head()
```

Out[13]:

	static_fire_date_utc	static_fire_date_unix	tbd	net	window	rocket	success
0	2006-03-17T00:00:00.000Z	1.142554e+09	False	False	0.0	5e9d0d95eda69955f709d1eb	False

a

	static_fire_date_utc	static_fire_date_unix	tbd	net	window	rocket	success
1	None	NaN	False	False	0.0	5e9d0d95eda69955f709d1eb	False
2	None	NaN	False	False	0.0	5e9d0d95eda69955f709d1eb	False
3	2008-09-20T00:00:00.000Z	1.221869e+09	False	False	0.0	5e9d0d95eda69955f709d1eb	True
4	None	NaN	False	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 flight
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]

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

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

# We also want to convert the date_utc to a datetime datatype and then extracting the d
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]:
```
# empty boosterversion list
BoosterVersion
```

Out[16]: `[]`

In [17]:
```
# Call getBoosterVersion to get the booster version
getBoosterVersion(data)
```

In [18]:
```
# the list has now been updated
BoosterVersion[0:5]
```

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

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 dataframe from launch_dict
data = pd.DataFrame(data = launch_dict)
```

In [24]:
```
# show head of dataframe
data.head()
```

Out[24]:

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

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 = data[data['BoosterVersion']!= 'Falcon 1']
```

In [26]:
```
type(data_falcon9)
```

Out[26]: `pandas.core.frame.DataFrame`

In [27]:
```
data_falcon9.head()
```

Out[27]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins
4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False
5	8	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False
6	10	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins
7	11	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False
8	12	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	False

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

In [28]:

```python
# reset FlightNumber column
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9
```

Out[28]:

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

90 rows × 17 columns

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

In [29]:

```python
# view how many rows have empty values
data_falcon9.isnull().sum()
```

Out[29]:

```
FlightNumber      0
Date              0
```

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

In [30]:

```
data_falcon9.count()
```

Out[30]:

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

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 [31]:

```
data_falcon9.head()
```

Out[31]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins
4	1	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False
5	2	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False
6	3	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False
7	4	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins
8	5	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	False

In [32]:

```
# Calculate the mean value of PayloadMass column
payloadmass_mean = data_falcon9["PayloadMass"].mean()

# Replace the np.nan values with its mean value
data_falcon9["PayloadMass"].fillna(value=payloadmass_mean, inplace=True)
```

In [33]:

```
data_falcon9.head()
```

Out[33]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins
4	1	2010-06-04	Falcon 9	6123.547647	LEO	CCSFS SLC 40	None None	1	False
5	2	2012-05-22	Falcon 9	525.000000	LEO	CCSFS SLC 40	None None	1	False
6	3	2013-03-01	Falcon 9	677.000000	ISS	CCSFS SLC 40	None None	1	False
7	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False
8	5	2013-12-03	Falcon 9	3170.000000	GTO	CCSFS SLC 40	None None	1	False

In [34]:

```
# view how many rows have empty values
data_falcon9.isnull().sum()
```

Out[34]:

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

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

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

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

```
data_falcon9.to_csv('dataset_part\_1.csv', index=False)
```

Lab: Collecting the Data (Webscraping)

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`

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

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

In [35]:

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

In [36]:

```
# import libraries
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 [37]:

```
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.string
    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 [38]:

```
# get url
static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_He
```

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 [39]:

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

Out[39]: <Response [200]>

In [40]:

```
# view status code
request_wiki.status_code
```

Out[40]: 200

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

In [41]:
```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(request_wiki.content, 'html.parser')
```

In [42]:
```
soup.title
```

Out[42]:
```
<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.

In [43]:
```
# 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 [44]:
```
# Let's print the third table and check its content
first_launch_table = html_tables[2]
print(first_launch_table.prettify())
```

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

```
    </th>
    <th scope="col">
     Payload
     <sup class="reference" id="cite_ref-Dragon_12-0">
      <a href="#cite_note-Dragon-12">
       [c]
      </a>
     </sup>
    </th>
    <th scope="col">
     Payload mass
    </th>
    <th scope="col">
     Orbit
    </th>
    <th scope="col">
     Customer
    </th>
    <th scope="col">
     Launch
     <br/>
     outcome
    </th>
    <th scope="col">
     <a href="/wiki/Falcon_9_first-stage_landing_tests" title="Falcon 9 first-stage landi
ng tests">
      Booster
      <br/>
      landing
     </a>
    </th>
   </tr>
   <tr>
    <th rowspan="2" scope="row" style="text-align:center;">
     1
    </th>
    <td>
     4 June 2010,
     <br/>
     18:45
    </td>
    <td>
     <a href="/wiki/Falcon_9_v1.0" title="Falcon 9 v1.0">
      F9 v1.0
     </a>
     <sup class="reference" id="cite_ref-MuskMay2012_13-0">
      <a href="#cite_note-MuskMay2012-13">
       [7]
      </a>
     </sup>
     <br/>
     B0003.1
     <sup class="reference" id="cite_ref-block_numbers_14-0">
      <a href="#cite_note-block_numbers-14">
       [8]
      </a>
     </sup>
    </td>
    <td>
     <a href="/wiki/Cape_Canaveral_Space_Force_Station" title="Cape Canaveral Space Force
Station">
      CCAFS
     </a>
     ,
     <br/>
```

```
    <a href="/wiki/Cape_Canaveral_Space_Launch_Complex_40" title="Cape Canaveral Space L
aunch Complex 40">
     SLC-40
    </a>
   </td>
   <td>
    <a href="/wiki/Dragon_Spacecraft_Qualification_Unit" title="Dragon Spacecraft Qualif
ication Unit">
     Dragon Spacecraft Qualification Unit
    </a>
   </td>
   <td>
   </td>
   <td>
    <a href="/wiki/Low_Earth_orbit" title="Low Earth orbit">
     LEO
    </a>
   </td>
   <td>
    <a href="/wiki/SpaceX" title="SpaceX">
     SpaceX
    </a>
   </td>
   <td class="table-success" style="background: LightGreen; color: black; vertical-alig
n: middle; text-align: center;">
    Success
   </td>
   <td class="table-failure" style="background: #ffbbbb; color: black; vertical-align: m
iddle; text-align: center;">
    Failure
    <sup class="reference" id="cite_ref-ns20110930_15-0">
     <a href="#cite_note-ns20110930-15">
      [9]
     </a>
    </sup>
    <sup class="reference" id="cite_ref-16">
     <a href="#cite_note-16">
      [10]
     </a>
    </sup>
    <br/>
    <small>
     (parachute)
    </small>
   </td>
  </tr>
  <tr>
   <td colspan="9">
    First flight of Falcon 9 v1.0.
    <sup class="reference" id="cite_ref-sfn20100604_17-0">
     <a href="#cite_note-sfn20100604-17">
      [11]
     </a>
    </sup>
    Used a boilerplate version of Dragon capsule which was not designed to separate from
the second stage.
    <small>
     (
     <a href="#First_flight_of_Falcon_9">
      more details below
     </a>
     )
    </small>
    Attempted to recover the first stage by parachuting it into the ocean, but it burned
up on reentry, before the parachutes even deployed.
```

```
       <sup class="reference" id="cite_ref-parachute_18-0">
        <a href="#cite_note-parachute-18">
         [12]
        </a>
       </sup>
      </td>
     </tr>
     <tr>
      <th rowspan="2" scope="row" style="text-align:center;">
       2
      </th>
      <td>
       8 December 2010,
       <br/>
       15:43
       <sup class="reference" id="cite_ref-spaceflightnow_Clark_Launch_Report_19-0">
        <a href="#cite_note-spaceflightnow_Clark_Launch_Report-19">
         [13]
        </a>
       </sup>
      </td>
      <td>
       <a href="/wiki/Falcon_9_v1.0" title="Falcon 9 v1.0">
        F9 v1.0
       </a>
       <sup class="reference" id="cite_ref-MuskMay2012_13-1">
        <a href="#cite_note-MuskMay2012-13">
         [7]
        </a>
       </sup>
       <br/>
       B0004.1
       <sup class="reference" id="cite_ref-block_numbers_14-1">
        <a href="#cite_note-block_numbers-14">
         [8]
        </a>
       </sup>
      </td>
      <td>
       <a href="/wiki/Cape_Canaveral_Space_Force_Station" title="Cape Canaveral Space Force
Station">
        CCAFS
       </a>
       ,
       <br/>
       <a href="/wiki/Cape_Canaveral_Space_Launch_Complex_40" title="Cape Canaveral Space L
aunch Complex 40">
        SLC-40
       </a>
      </td>
      <td>
       <a href="/wiki/SpaceX_Dragon" title="SpaceX Dragon">
        Dragon
       </a>
       <a class="mw-redirect" href="/wiki/COTS_Demo_Flight_1" title="COTS Demo Flight 1">
        demo flight C1
       </a>
       <br/>
       (Dragon C101)
      </td>
      <td>
      </td>
      <td>
       <a href="/wiki/Low_Earth_orbit" title="Low Earth orbit">
        LEO
```

```
      </a>
      (
      <a href="/wiki/International_Space_Station" title="International Space Station">
       ISS
      </a>
      )
     </td>
     <td>
      <div class="plainlist">
       <ul>
        <li>
         <a href="/wiki/NASA" title="NASA">
          NASA
         </a>
         (
         <a href="/wiki/Commercial_Orbital_Transportation_Services" title="Commercial Orbi
tal Transportation Services">
          COTS
         </a>
         )
        </li>
        <li>
         <a href="/wiki/National_Reconnaissance_Office" title="National Reconnaissance Off
ice">
          NRO
         </a>
        </li>
       </ul>
      </div>
     </td>
     <td class="table-success" style="background: LightGreen; color: black; vertical-alig
n: middle; text-align: center;">
      Success
      <sup class="reference" id="cite_ref-ns20110930_15-1">
       <a href="#cite_note-ns20110930-15">
        [9]
       </a>
      </sup>
     </td>
     <td class="table-failure" style="background: #ffbbbb; color: black; vertical-align: m
iddle; text-align: center;">
      Failure
      <sup class="reference" id="cite_ref-ns20110930_15-2">
       <a href="#cite_note-ns20110930-15">
        [9]
       </a>
      </sup>
      <sup class="reference" id="cite_ref-20">
       <a href="#cite_note-20">
        [14]
       </a>
      </sup>
      <br/>
      <small>
       (parachute)
      </small>
     </td>
    </tr>
    <tr>
     <td colspan="9">
      Maiden flight of
      <a class="mw-redirect" href="/wiki/Dragon_capsule" title="Dragon capsule">
       Dragon capsule
      </a>
      , consisting of over 3 hours of testing thruster maneuvering and reentry.
```

```
        <sup class="reference" id="cite_ref-spaceflightnow_Clark_unleashing_Dragon_21-0">
         <a href="#cite_note-spaceflightnow_Clark_unleashing_Dragon-21">
          [15]
         </a>
        </sup>
        Attempted to recover the first stage by parachuting it into the ocean, but it disint
egrated upon reentry, before the parachutes were deployed.
        <sup class="reference" id="cite_ref-parachute_18-1">
         <a href="#cite_note-parachute-18">
          [12]
         </a>
        </sup>
        <small>
         (
         <a href="#COTS_demo_missions">
          more details below
         </a>
         )
        </small>
        It also included two
        <a href="/wiki/CubeSat" title="CubeSat">
         CubeSats
        </a>
        ,
        <sup class="reference" id="cite_ref-NRO_Taps_Boeing_for_Next_Batch_of_CubeSats_22-
0">
         <a href="#cite_note-NRO_Taps_Boeing_for_Next_Batch_of_CubeSats-22">
          [16]
         </a>
        </sup>
        and a wheel of
        <a href="/wiki/Brou%C3%A8re" title="Brouère">
         Brouère
        </a>
        cheese.
       </td>
      </tr>
      <tr>
       <th rowspan="2" scope="row" style="text-align:center;">
        3
       </th>
       <td>
        22 May 2012,
        <br/>
        07:44
        <sup class="reference" id="cite_ref-BBC_new_era_23-0">
         <a href="#cite_note-BBC_new_era-23">
          [17]
         </a>
        </sup>
       </td>
       <td>
        <a href="/wiki/Falcon_9_v1.0" title="Falcon 9 v1.0">
         F9 v1.0
        </a>
        <sup class="reference" id="cite_ref-MuskMay2012_13-2">
         <a href="#cite_note-MuskMay2012-13">
          [7]
         </a>
        </sup>
        <br/>
        B0005.1
        <sup class="reference" id="cite_ref-block_numbers_14-2">
         <a href="#cite_note-block_numbers-14">
          [8]
```

```
     </a>
    </sup>
   </td>
   <td>
    <a href="/wiki/Cape_Canaveral_Space_Force_Station" title="Cape Canaveral Space Force
Station">
     CCAFS
    </a>
    ,
    <br/>
    <a href="/wiki/Cape_Canaveral_Space_Launch_Complex_40" title="Cape Canaveral Space L
aunch Complex 40">
     SLC-40
    </a>
   </td>
   <td>
    <a href="/wiki/SpaceX_Dragon" title="SpaceX Dragon">
     Dragon
    </a>
    <a class="mw-redirect" href="/wiki/Dragon_C2%2B" title="Dragon C2+">
     demo flight C2+
    </a>
    <sup class="reference" id="cite_ref-C2_24-0">
     <a href="#cite_note-C2-24">
      [18]
     </a>
    </sup>
    <br/>
    (Dragon C102)
   </td>
   <td>
    525 kg (1,157 lb)
    <sup class="reference" id="cite_ref-25">
     <a href="#cite_note-25">
      [19]
     </a>
    </sup>
   </td>
   <td>
    <a href="/wiki/Low_Earth_orbit" title="Low Earth orbit">
     LEO
    </a>
    (
    <a href="/wiki/International_Space_Station" title="International Space Station">
     ISS
    </a>
    )
   </td>
   <td>
    <a href="/wiki/NASA" title="NASA">
     NASA
    </a>
    (
    <a href="/wiki/Commercial_Orbital_Transportation_Services" title="Commercial Orbital
Transportation Services">
     COTS
    </a>
    )
   </td>
   <td class="table-success" style="background: LightGreen; color: black; vertical-alig
n: middle; text-align: center;">
    Success
    <sup class="reference" id="cite_ref-26">
     <a href="#cite_note-26">
      [20]
```

```
     </a>
    </sup>
   </td>
   <td class="table-noAttempt" style="background: #ececec; color: black; vertical-align:
middle; white-space: nowrap; text-align: center;">
    No attempt
   </td>
  </tr>
  <tr>
   <td colspan="9">
    Dragon spacecraft demonstrated a series of tests before it was allowed to approach t
he
    <a href="/wiki/International_Space_Station" title="International Space Station">
     International Space Station
    </a>
    . Two days later, it became the first commercial spacecraft to board the ISS.
    <sup class="reference" id="cite_ref-BBC_new_era_23-1">
     <a href="#cite_note-BBC_new_era-23">
      [17]
     </a>
    </sup>
    <small>
     (
     <a href="#COTS_demo_missions">
      more details below
     </a>
     )
    </small>
   </td>
  </tr>
  <tr>
   <th rowspan="3" scope="row" style="text-align:center;">
    4
   </th>
   <td rowspan="2">
    8 October 2012,
    <br/>
    00:35
    <sup class="reference" id="cite_ref-SFN_LLog_27-0">
     <a href="#cite_note-SFN_LLog-27">
      [21]
     </a>
    </sup>
   </td>
   <td rowspan="2">
    <a href="/wiki/Falcon_9_v1.0" title="Falcon 9 v1.0">
     F9 v1.0
    </a>
    <sup class="reference" id="cite_ref-MuskMay2012_13-3">
     <a href="#cite_note-MuskMay2012-13">
      [7]
     </a>
    </sup>
    <br/>
    B0006.1
    <sup class="reference" id="cite_ref-block_numbers_14-3">
     <a href="#cite_note-block_numbers-14">
      [8]
     </a>
    </sup>
   </td>
   <td rowspan="2">
    <a href="/wiki/Cape_Canaveral_Space_Force_Station" title="Cape Canaveral Space Force
Station">
     CCAFS
```

```
      </a>
      ,
      <br/>
      <a href="/wiki/Cape_Canaveral_Space_Launch_Complex_40" title="Cape Canaveral Space L
aunch Complex 40">
       SLC-40
      </a>
     </td>
     <td>
      <a href="/wiki/SpaceX_CRS-1" title="SpaceX CRS-1">
       SpaceX CRS-1
      </a>
      <sup class="reference" id="cite_ref-sxManifest20120925_28-0">
       <a href="#cite_note-sxManifest20120925-28">
        [22]
       </a>
      </sup>
      <br/>
      (Dragon C103)
     </td>
     <td>
      4,700 kg (10,400 lb)
     </td>
     <td>
      <a href="/wiki/Low_Earth_orbit" title="Low Earth orbit">
       LEO
      </a>
      (
      <a href="/wiki/International_Space_Station" title="International Space Station">
       ISS
      </a>
      )
     </td>
     <td>
      <a href="/wiki/NASA" title="NASA">
       NASA
      </a>
      (
      <a href="/wiki/Commercial_Resupply_Services" title="Commercial Resupply Services">
       CRS
      </a>
      )
     </td>
     <td class="table-success" style="background: LightGreen; color: black; vertical-alig
n: middle; text-align: center;">
      Success
     </td>
     <td rowspan="2" style="background:#ececec; text-align:center;">
      <span class="nowrap">
       No attempt
      </span>
     </td>
    </tr>
    <tr>
     <td>
      <a href="/wiki/Orbcomm_(satellite)" title="Orbcomm (satellite)">
       Orbcomm-OG2
      </a>
      <sup class="reference" id="cite_ref-Orbcomm_29-0">
       <a href="#cite_note-Orbcomm-29">
        [23]
       </a>
      </sup>
     </td>
     <td>
```

```
        172 kg (379 lb)
        <sup class="reference" id="cite_ref-gunter-og2_30-0">
         <a href="#cite_note-gunter-og2-30">
          [24]
         </a>
        </sup>
       </td>
       <td>
        <a href="/wiki/Low_Earth_orbit" title="Low Earth orbit">
         LEO
        </a>
       </td>
       <td>
        <a href="/wiki/Orbcomm" title="Orbcomm">
         Orbcomm
        </a>
       </td>
       <td class="table-partial" style="background: wheat; color: black; vertical-align: mid
dle; text-align: center;">
        Partial failure
        <sup class="reference" id="cite_ref-nyt-20121030_31-0">
         <a href="#cite_note-nyt-20121030-31">
          [25]
         </a>
        </sup>
       </td>
      </tr>
      <tr>
       <td colspan="9">
        CRS-1 was successful, but the
        <a href="/wiki/Secondary_payload" title="Secondary payload">
         secondary payload
        </a>
        was inserted into an abnormally low orbit and subsequently lost. This was due to one
of the nine
        <a href="/wiki/SpaceX_Merlin" title="SpaceX Merlin">
         Merlin engines
        </a>
        shutting down during the launch, and NASA declining a second reignition, as per
        <a href="/wiki/International_Space_Station" title="International Space Station">
         ISS
        </a>
        visiting vehicle safety rules, the primary payload owner is contractually allowed to
decline a second reignition. NASA stated that this was because SpaceX could not guarante
e a high enough likelihood of the second stage completing the second burn successfully w
hich was required to avoid any risk of secondary payload's collision with the ISS.
        <sup class="reference" id="cite_ref-OrbcommTotalLoss_32-0">
         <a href="#cite_note-OrbcommTotalLoss-32">
          [26]
         </a>
        </sup>
        <sup class="reference" id="cite_ref-sn20121011_33-0">
         <a href="#cite_note-sn20121011-33">
          [27]
         </a>
        </sup>
        <sup class="reference" id="cite_ref-34">
         <a href="#cite_note-34">
          [28]
         </a>
        </sup>
       </td>
      </tr>
      <tr>
       <th rowspan="2" scope="row" style="text-align:center;">
```

```
 5
</th>
<td>
 1 March 2013,
 <br/>
 15:10
</td>
<td>
 <a href="/wiki/Falcon_9_v1.0" title="Falcon 9 v1.0">
  F9 v1.0
 </a>
 <sup class="reference" id="cite_ref-MuskMay2012_13-4">
  <a href="#cite_note-MuskMay2012-13">
   [7]
  </a>
 </sup>
 <br/>
 B0007.1
 <sup class="reference" id="cite_ref-block_numbers_14-4">
  <a href="#cite_note-block_numbers-14">
   [8]
  </a>
 </sup>
</td>
<td>
 <a href="/wiki/Cape_Canaveral_Space_Force_Station" title="Cape Canaveral Space Force
Station">
  CCAFS
 </a>
 ,
 <br/>
 <a href="/wiki/Cape_Canaveral_Space_Launch_Complex_40" title="Cape Canaveral Space L
aunch Complex 40">
  SLC-40
 </a>
</td>
<td>
 <a href="/wiki/SpaceX_CRS-2" title="SpaceX CRS-2">
  SpaceX CRS-2
 </a>
 <sup class="reference" id="cite_ref-sxManifest20120925_28-1">
  <a href="#cite_note-sxManifest20120925-28">
   [22]
  </a>
 </sup>
 <br/>
 (Dragon C104)
</td>
<td>
 4,877 kg (10,752 lb)
</td>
<td>
 <a href="/wiki/Low_Earth_orbit" title="Low Earth orbit">
  LEO
 </a>
 (
 <a class="mw-redirect" href="/wiki/ISS" title="ISS">
  ISS
 </a>
 )
</td>
<td>
 <a href="/wiki/NASA" title="NASA">
  NASA
 </a>
```

```
    (
    <a href="/wiki/Commercial_Resupply_Services" title="Commercial Resupply Services">
     CRS
    </a>
    )
   </td>
   <td class="table-success" style="background: LightGreen; color: black; vertical-alig
n: middle; text-align: center;">
    Success
   </td>
   <td class="table-noAttempt" style="background: #ececec; color: black; vertical-align:
middle; white-space: nowrap; text-align: center;">
    No attempt
   </td>
  </tr>
  <tr>
   <td colspan="9">
    Last launch of the original Falcon 9 v1.0
    <a href="/wiki/Launch_vehicle" title="Launch vehicle">
     launch vehicle
    </a>
    , first use of the unpressurized trunk section of Dragon.
    <sup class="reference" id="cite_ref-sxf9_20110321_35-0">
     <a href="#cite_note-sxf9_20110321-35">
      [29]
     </a>
    </sup>
   </td>
  </tr>
  <tr>
   <th rowspan="2" scope="row" style="text-align:center;">
    6
   </th>
   <td>
    29 September 2013,
    <br/>
    16:00
    <sup class="reference" id="cite_ref-pa20130930_36-0">
     <a href="#cite_note-pa20130930-36">
      [30]
     </a>
    </sup>
   </td>
   <td>
    <a href="/wiki/Falcon_9_v1.1" title="Falcon 9 v1.1">
     F9 v1.1
    </a>
    <sup class="reference" id="cite_ref-MuskMay2012_13-5">
     <a href="#cite_note-MuskMay2012-13">
      [7]
     </a>
    </sup>
    <br/>
    B1003
    <sup class="reference" id="cite_ref-block_numbers_14-5">
     <a href="#cite_note-block_numbers-14">
      [8]
     </a>
    </sup>
   </td>
   <td>
    <a class="mw-redirect" href="/wiki/Vandenberg_Air_Force_Base" title="Vandenberg Air
Force Base">
     VAFB
    </a>
```

```
      ,
      <br/>
      <a href="/wiki/Vandenberg_Space_Launch_Complex_4" title="Vandenberg Space Launch Com
plex 4">
       SLC-4E
      </a>
     </td>
     <td>
      <a href="/wiki/CASSIOPE" title="CASSIOPE">
       CASSIOPE
      </a>
      <sup class="reference" id="cite_ref-sxManifest20120925_28-2">
       <a href="#cite_note-sxManifest20120925-28">
        [22]
       </a>
      </sup>
      <sup class="reference" id="cite_ref-CASSIOPE_MDA_37-0">
       <a href="#cite_note-CASSIOPE_MDA-37">
        [31]
       </a>
      </sup>
     </td>
     <td>
      500 kg (1,100 lb)
     </td>
     <td>
      <a href="/wiki/Polar_orbit" title="Polar orbit">
       Polar orbit
      </a>
      <a href="/wiki/Low_Earth_orbit" title="Low Earth orbit">
       LEO
      </a>
     </td>
     <td>
      <a href="/wiki/Maxar_Technologies" title="Maxar Technologies">
       MDA
      </a>
     </td>
     <td class="table-success" style="background: LightGreen; color: black; vertical-alig
n: middle; text-align: center;">
      Success
      <sup class="reference" id="cite_ref-pa20130930_36-1">
       <a href="#cite_note-pa20130930-36">
        [30]
       </a>
      </sup>
     </td>
     <td class="table-no2" style="background: #ffdddd; color: black; vertical-align: middl
e; text-align: center;">
      Uncontrolled
      <br/>
      <small>
       (ocean)
      </small>
      <sup class="reference" id="cite_ref-ocean_landing_38-0">
       <a href="#cite_note-ocean_landing-38">
        [d]
       </a>
      </sup>
     </td>
    </tr>
    <tr>
     <td colspan="9">
      First commercial mission with a private customer, first launch from Vandenberg, and
demonstration flight of Falcon 9 v1.1 with an improved 13-tonne to LEO capacity.
```

```
    <sup class="reference" id="cite_ref-sxf9_20110321_35-1">
     <a href="#cite_note-sxf9_20110321-35">
      [29]
     </a>
    </sup>
    After separation from the second stage carrying Canadian commercial and scientific s
atellites, the first stage booster performed a controlled reentry,
    <sup class="reference" id="cite_ref-39">
     <a href="#cite_note-39">
      [32]
     </a>
    </sup>
    and an
    <a href="/wiki/Falcon_9_first-stage_landing_tests" title="Falcon 9 first-stage landi
ng tests">
     ocean touchdown test
    </a>
    for the first time. This provided good test data, even though the booster started ro
lling as it neared the ocean, leading to the shutdown of the central engine as the roll
depleted it of fuel, resulting in a hard impact with the ocean.
    <sup class="reference" id="cite_ref-pa20130930_36-2">
     <a href="#cite_note-pa20130930-36">
      [30]
     </a>
    </sup>
    This was the first known attempt of a rocket engine being lit to perform a supersoni
c retro propulsion, and allowed SpaceX to enter a public-private partnership with
    <a href="/wiki/NASA" title="NASA">
     NASA
    </a>
    and its Mars entry, descent, and landing technologies research projects.
    <sup class="reference" id="cite_ref-40">
     <a href="#cite_note-40">
      [33]
     </a>
    </sup>
    <small>
     (
     <a href="#Maiden_flight_of_v1.1">
      more details below
     </a>
     )
    </small>
   </td>
  </tr>
  <tr>
   <th rowspan="2" scope="row" style="text-align:center;">
    7
   </th>
   <td>
    3 December 2013,
    <br/>
    22:41
    <sup class="reference" id="cite_ref-sfn_wwls20130624_41-0">
     <a href="#cite_note-sfn_wwls20130624-41">
      [34]
     </a>
    </sup>
   </td>
   <td>
    <a href="/wiki/Falcon_9_v1.1" title="Falcon 9 v1.1">
     F9 v1.1
    </a>
    <br/>
    B1004
```

```
   </td>
   <td>
    <a href="/wiki/Cape_Canaveral_Space_Force_Station" title="Cape Canaveral Space Force
Station">
     CCAFS
    </a>
    ,
    <br/>
    <a href="/wiki/Cape_Canaveral_Space_Launch_Complex_40" title="Cape Canaveral Space L
aunch Complex 40">
     SLC-40
    </a>
   </td>
   <td>
    <a href="/wiki/SES-8" title="SES-8">
     SES-8
    </a>
    <sup class="reference" id="cite_ref-sxManifest20120925_28-3">
     <a href="#cite_note-sxManifest20120925-28">
      [22]
     </a>
    </sup>
    <sup class="reference" id="cite_ref-spx-pr_42-0">
     <a href="#cite_note-spx-pr-42">
      [35]
     </a>
    </sup>
    <sup class="reference" id="cite_ref-aw20110323_43-0">
     <a href="#cite_note-aw20110323-43">
      [36]
     </a>
    </sup>
   </td>
   <td>
    3,170 kg (6,990 lb)
   </td>
   <td>
    <a href="/wiki/Geostationary_transfer_orbit" title="Geostationary transfer orbit">
     GTO
    </a>
   </td>
   <td>
    <a href="/wiki/SES_S.A." title="SES S.A.">
     SES
    </a>
   </td>
   <td class="table-success" style="background: LightGreen; color: black; vertical-alig
n: middle; text-align: center;">
    Success
    <sup class="reference" id="cite_ref-SNMissionStatus7_44-0">
     <a href="#cite_note-SNMissionStatus7-44">
      [37]
     </a>
    </sup>
   </td>
   <td class="table-noAttempt" style="background: #ececec; color: black; vertical-align:
middle; white-space: nowrap; text-align: center;">
    No attempt
    <br/>
    <sup class="reference" id="cite_ref-sf10120131203_45-0">
     <a href="#cite_note-sf10120131203-45">
      [38]
     </a>
    </sup>
   </td>
```

```
   </tr>
   <tr>
    <td colspan="9">
     First
     <a href="/wiki/Geostationary_transfer_orbit" title="Geostationary transfer orbit">
      Geostationary transfer orbit
     </a>
     (GTO) launch for Falcon 9,
     <sup class="reference" id="cite_ref-spx-pr_42-1">
      <a href="#cite_note-spx-pr-42">
       [35]
      </a>
     </sup>
     and first successful reignition of the second stage.
     <sup class="reference" id="cite_ref-46">
      <a href="#cite_note-46">
       [39]
      </a>
     </sup>
     SES-8 was inserted into a
     <a href="/wiki/Geostationary_transfer_orbit" title="Geostationary transfer orbit">
      Super-Synchronous Transfer Orbit
     </a>
     of 79,341 km (49,300 mi) in apogee with an
     <a href="/wiki/Orbital_inclination" title="Orbital inclination">
      inclination
     </a>
     of 20.55° to the
     <a href="/wiki/Equator" title="Equator">
      equator
     </a>
     .
    </td>
   </tr>
  </tbody>
 </table>
```

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

```
<tr>
<th scope="col">Flight No.
</th>
<th scope="col">Date and<br/>time (<a
href="/wiki/Coordinated_Universal_Time" title="Coordinated Universal
Time">UTC</a>)
</th>
<th scope="col"><a href="/wiki/List_of_Falcon_9_first-stage_boosters"
title="List of Falcon 9 first-stage boosters">Version,<br/>Booster</a>
<sup class="reference" id="cite_ref-booster_11-0"><a href="#cite_note-
booster-11">[b]</a></sup>
</th>
<th scope="col">Launch site
</th>
<th scope="col">Payload<sup class="reference" id="cite_ref-Dragon_12-0">
<a href="#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 [45]:

```
column_names = []

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

In [46]:

```
# check on the extracted column names
print(column_names)
```

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

Task 3: Create a dataframe 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 [47]:

```
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 [48]:

```
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders co
   # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number corresponding to launch a num
        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])
            launch_dict['Flight No.'].append('flight_number')

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

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

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

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

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

```
            payload = row[3].a.string
            launch_dict['Payload'].append(payload)
            #print(payload)

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

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

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

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

            # Booster landing
            # TODO: Append the launch_outcome into launch_dict with key `Booster landin
            booster_landing = landing_status(row[8])
            launch_dict['Booster landing'].append(launch_outcome)
            #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
F9 v1.1
F9 FT
F9 FT
F9 FT
F9 FT
F9 FT
F9 FT
```

```
F9 FT
F9 FT
F9 FT
F9 FT
F9 FT♲
F9 FT
F9 FT
F9 FT
F9 FTB1029.2
F9 FT
F9 FT
F9 B4
F9 FT
F9 B4
F9 B4
F9 FTB1031.2
F9 B4
F9 FTB1035.2
F9 FTB1036.2
F9 B4
F9 FTB1032.2
F9 FTB1038.2
F9 B4
F9 B4B1041.2
F9 B4B1039.2
F9 B4
F9 B5B1046.1
F9 B4B1043.2
F9 B4B1040.2
F9 B4B1045.2
F9 B5
F9 B5B1048
F9 B5B1046.2
F9 B5
F9 B5B1048.2
F9 B5B1047.2
F9 B5B1046.3
F9 B5
F9 B5
F9 B5B1049.2
F9 B5B1048.3
F9 B5[268]
F9 B5
F9 B5B1049.3
F9 B5B1051.2
F9 B5B1056.2
F9 B5B1047.3
F9 B5
F9 B5
F9 B5B1056.3
F9 B5
F9 B5
F9 B5
F9 B5
F9 B5
F9 B5
F9 B5
F9 B5
F9 B5
F9 B5
F9 B5
F9 B5B1058.2
F9 B5
F9 B5B1049.6
F9 B5
```

```
F9 B5B1060.2
F9 B5B1058.3
F9 B5B1051.6
F9 B5
F9 B5
F9 B5
F9 B5
F9 B5 ♺
F9 B5 ♺
F9 B5 ♺
F9 B5 ♺
F9 B5
F9 B5B1051.8
F9 B5B1058.5
F9 B5 ♺
F9 B5 ♺
F9 B5 ♺
F9 B5 ♺
F9 B5 ♺
F9 B5B1060.6
F9 B5 ♺
F9 B5B1061.2
F9 B5B1060.7
F9 B5B1049.9
F9 B5B1051.10
F9 B5B1058.8
F9 B5B1063.2
F9 B5B1067.1
F9 B5
```

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

In [49]:
```
df= pd.DataFrame({ key:pd.Series(value) for key, value in launch_dict.items() })
```

In [50]:
```
df
```

Out[50]:

	Flight No.	Launch site	Payload	Payload mass	Orbit	Customer	Launch outcome	Version Booster	Booster landing
0	flight_number	CCAFS	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success\n	F9 v1.0B0003.1	Success\n
1	flight_number	CCAFS	Dragon	0	LEO	NASA (COTS)\nNRO	Success	F9 v1.0B0004.1	Success
2	flight_number	CCAFS	Dragon	525 kg	LEO	NASA (COTS)	Success	F9 v1.0B0005.1	Success
3	flight_number	CCAFS	SpaceX CRS-1	4,700 kg	LEO	NASA (CRS)	Success\n	F9 v1.0B0006.1	Success\n
4	flight_number	CCAFS	SpaceX CRS-2	4,877 kg	LEO	NASA (CRS)	Success\n	F9 v1.0B0007.1	Success\n
...	...	...	...	...	...	...	...	...	...

	Flight No.	Launch site	Payload	Payload mass	Orbit	Customer	Launch outcome	Version Booster	Booster landing
116	flight_number	CCSFS	Starlink	15,600 kg	LEO	SpaceX	Success\n	F9 B5B1051.10	Success\n
117	flight_number	KSC	Starlink	~14,000 kg	LEO	SpaceX Capella Space and Tyvak	Success\n	F9 B5B1058.8	Success\n
118	flight_number	CCSFS	Starlink	15,600 kg	LEO	SpaceX	Success\n	F9 B5B1063.2	Success\n
119	flight_number	KSC	SpaceX CRS-22	3,328 kg	LEO	NASA (CRS)	Success\n	F9 B5B1067.1	Success\n
120	flight_number	CCSFS	SXM-8	7,000 kg	GTO	Sirius XM	Success\n	F9 B5	Success\n

121 rows × 11 columns

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)
```

Lab: Data Wrangling

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.

Objectives:

Perform exploratory Data Analysis and determine Training Labels

- Exploratory Data Analysis
- Determine Training Labels

In [51]:
```
# Pandas is a software library written for the Python programming language for data man
import pandas as pd
#NumPy is a library for the Python programming language, adding support for large, mult
import numpy as np
```

In [52]:
```
# import spaceX dataset from last section
df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-
df.head(10)
```

Out[52]:

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

In [53]:
```
# identify and calculate percentage of the missing values for each attribute
df.isnull().sum()/df.count()*100
```

Out[53]:
```
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
```

In [54]:

```
# identify which columns are numerical or categorical
df.dtypes
```

Out[54]:

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

Task 1: Calculate the number of launches on each site

The data contains several Space X launch facilities: Cape Canaveral Space Launch Complex 40 **VAFB SLC 4E** , Vandenberg Air Force Base Space Launch Complex 4E **(SLC-4E)**, Kennedy Space Center Launch Complex 39A **KSC LC 39A** .The location of each Launch Is placed in the column `LaunchSite`

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

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

In [55]:

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

Out[55]:

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

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

- **LEO**: Low Earth orbit (LEO)is an Earth-centred orbit with an altitude of 2,000 km (1,200 mi) or less (approximately one-third of the radius of Earth),[1] or with at least 11.25 periods per day (an orbital period of 128 minutes or less) and an eccentricity less than 0.25.[2] Most of the manmade objects in outer space are in LEO \[1].
- **VLEO**: Very Low Earth Orbits (VLEO) can be defined as the orbits with a mean altitude below 450 km. Operating in these orbits can provide a number of benefits to Earth observation spacecraft as the spacecraft operates closer to the observation\[2].

- **GTO** A geosynchronous orbit is a high Earth orbit that allows satellites to match Earth's rotation. Located at 22,236 miles (35,786 kilometers) above Earth's equator, this position is a valuable spot for monitoring weather, communications and surveillance. Because the satellite orbits at the same speed that the Earth is turning, the satellite seems to stay in place over a single longitude, though it may drift north to south," NASA wrote on its Earth Observatory website \[3] .

- **SSO (or SO)**: It is a Sun-synchronous orbit also called a heliosynchronous orbit is a nearly polar orbit around a planet, in which the satellite passes over any given point of the planet's surface at the same local mean solar time [4] .

- **ES-L1** :At the Lagrange points the gravitational forces of the two large bodies cancel out in such a way that a small object placed in orbit there is in equilibrium relative to the center of mass of the large bodies. L1 is one such point between the sun and the earth \[5] .

- **HEO** A highly elliptical orbit, is an elliptic orbit with high eccentricity, usually referring to one around Earth \[6].

- **ISS** A modular space station (habitable artificial satellite) in low Earth orbit. It is a multinational collaborative project between five participating space agencies: NASA (United States), Roscosmos (Russia), JAXA (Japan), ESA (Europe), and CSA (Canada) \[7]

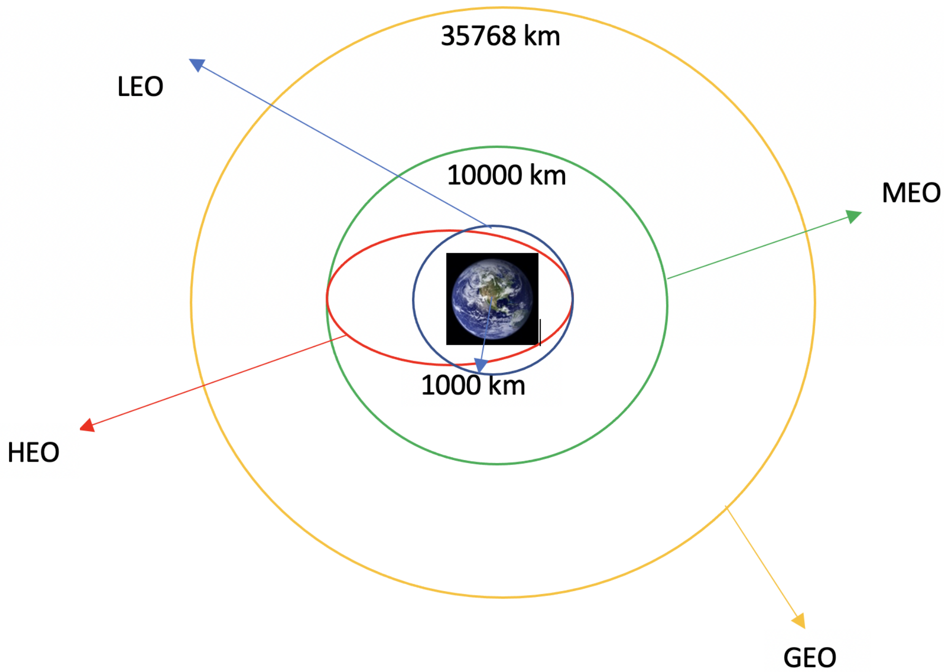
- **MEO** Geocentric orbits ranging in altitude from 2,000 km (1,200 mi) to just below geosynchronous orbit at 35,786 kilometers (22,236 mi). Also known as an intermediate circular orbit. These are "most commonly at 20,200 kilometers (12,600 mi), or 20,650 kilometers (12,830 mi), with an orbital period of 12 hours \[8]

- **HEO** Geocentric orbits above the altitude of geosynchronous orbit (35,786 km or 22,236 mi) \[9]

- **GEO** It is a circular geosynchronous orbit 35,786 kilometres (22,236 miles) above Earth's equator and following the direction of Earth's rotation \[10]

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

some are shown in the following plot:



Task 2: Calculate the number and occurence of each orbit

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

In [56]:
```
# Apply value_counts on Orbit column
df['Orbit'].value_counts()
```

Out[56]:
```
GTO       27
ISS       21
VLEO      14
PO         9
LEO        7
SSO        5
MEO        3
SO         1
HEO        1
GEO        1
ES-L1      1
Name: Orbit, dtype: int64
```

Task 3: Calculate the number and occurrence of mission outcome per orbit type

Use the method `value_counts()` to determine the number and occurrence of each orbit in the column `Outcome` , then assign it to the variable `landing_outcomes` :

In [57]:
```
# landing_outcomes = values on Outcome column
landing_outcomes = df['Outcome'].value_counts()
landing_outcomes
```

Out[57]:
```
True ASDS      41
```

```
None None      19
True RTLS      14
False ASDS      6
True Ocean      5
None ASDS       2
False Ocean     2
False RTLS      1
Name: Outcome, dtype: int64
```

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

In [58]:

```
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 None ASDS
6 False Ocean
7 False RTLS
```

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

In [59]:

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

Out[59]:

```
{'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 [60]:

```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise

landing_class = []
for outcome in enumerate(df['Outcome']):
    landing_class = [0 if outcome in bad_outcomes else 1 for outcome in df['Outcome']]
```

In [61]:

```
print(df['Outcome'], landing_class)
```

```
0       None None
1       None None
2       None None
3      False Ocean
4       None None
```

```
        ...
85      True ASDS
86      True ASDS
87      True ASDS
88      True ASDS
89      True ASDS
Name: Outcome, Length: 90, dtype: object [0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0,
0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0,
1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

In [62]:

```
df['Outcome'].value_counts()
```

Out[62]:

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

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 [63]:

```
# create dataframe of the landing_class
df['Class']=landing_class
df[['Class']].head(8)
```

Out[63]:

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

In [64]:

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

Out[64]:

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

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False

In [65]:

```
# get the success rate
df["Class"].mean()
```

Out[65]: 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)
```

Week 2: Exploratory Data Analysis

Lab: Exporatory Data Analysis with SQL

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

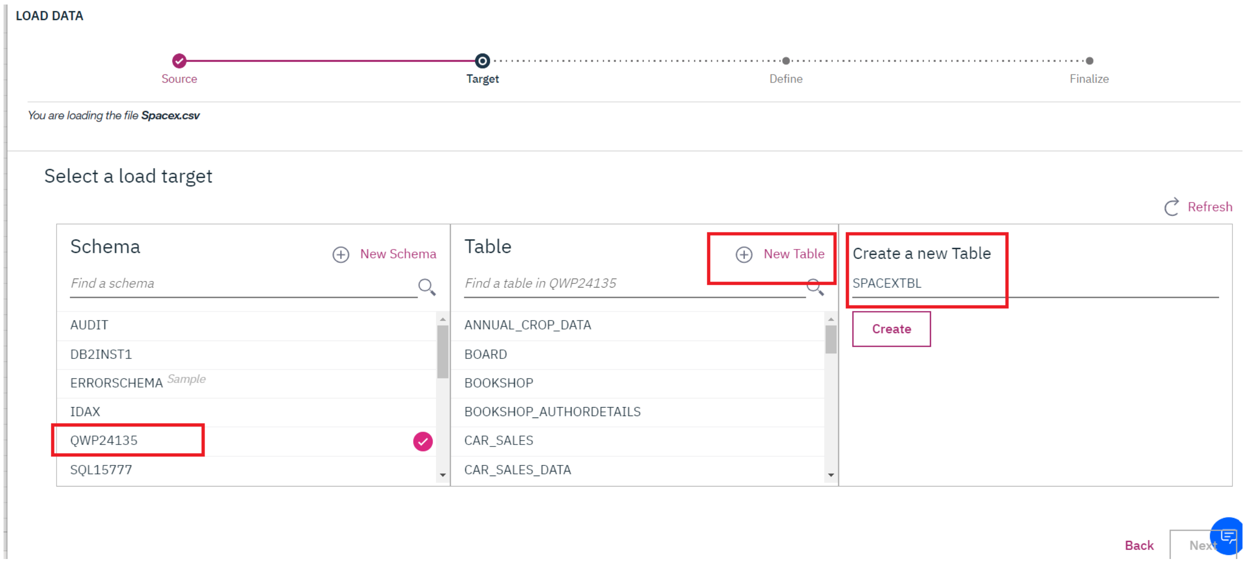
This assignment requires you to load the spacex dataset.

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

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:

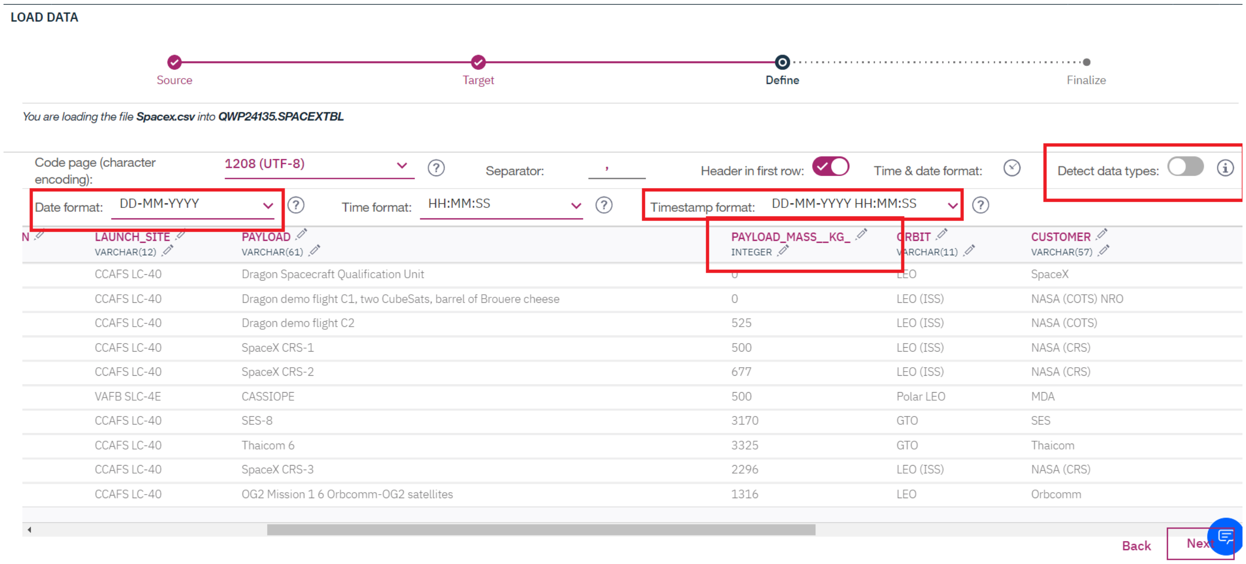
SPACEXDATASET

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).

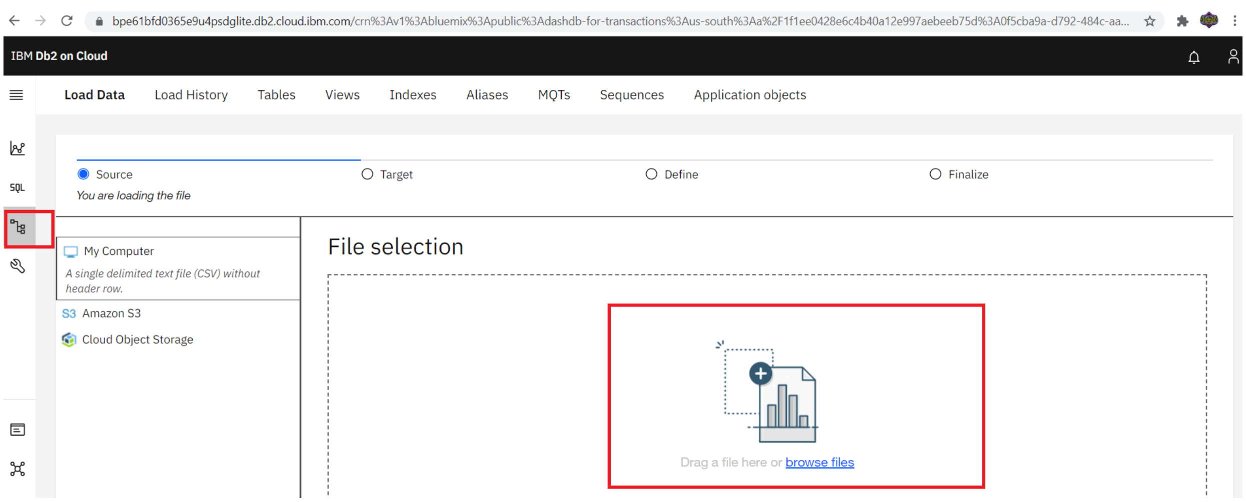
1. Change the Date Format by manually typing DD-MM-YYYY and timestamp format as DD-MM-YYYY HH\:MM:SS

2. Change the PAYLOAD*MASS_KG_* datatype to INTEGER.

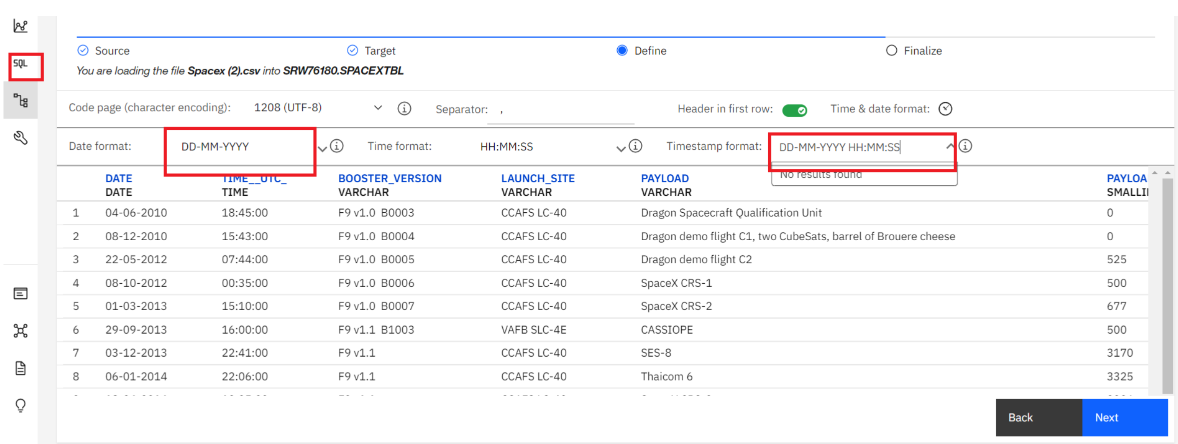


Changes to be considered when having DB2 instance with the new UI having Go to UI screen

- Refer to this insruction in this link 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.



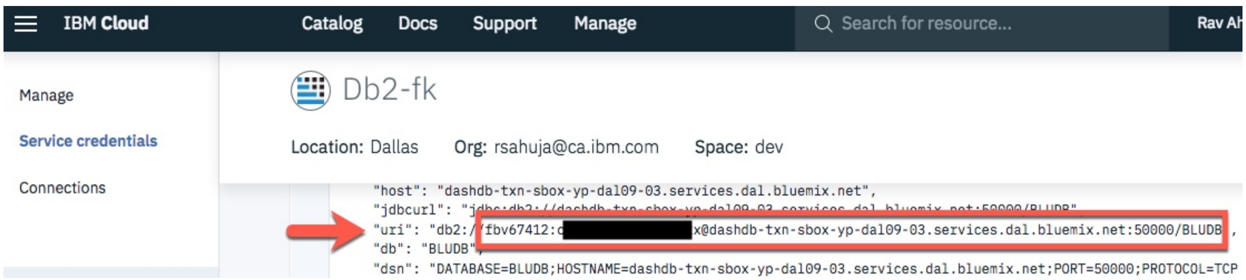
In [66]:
```
#!pip install sqlalchemy==1.3.9
#!pip install ibm_db_sa
```

Load the SQL Extension and establish a connection with the database

In [67]:
```
%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 [68]:

```
%sql ibm_db_sa://fgl32023:25q97r4n99-mt9x2@dashdb-txn-sbox-yp-dal09-04.services.dal.blu
```

```
DB2/LINUXX8664
```

In [69]:

```
%%sql

select * from SPACEXTBL limit 3;
```

```
 * ibm_db_sa://fgl32023:***@dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net:50000/B
LUDB
Done.
```

Out[69]:

Date	Time__UTC_	Booster_Version	Launch_Site	Payload	payload_mass__kg_	Orbit	Customer	Mis
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)	

Task 1: Display the names of the unique launch sites in the space mission

In [70]:

```
%%sql

SELECT DISTINCT("Launch_Site")
FROM SPACEXTBL;
```

```
 * ibm_db_sa://fgl32023:***@dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net:50000/B
LUDB
Done.
```

Out[70]:

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

Task 2: Display 5 records where launch sites begin with string 'CCA'

In [71]:
```
%%sql

SELECT *
FROM SPACEXTBL
WHERE "Launch_Site" LIKE 'CCA%'
LIMIT 5;
```

```
 * ibm_db_sa://fgl32023:***@dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net:50000/B
LUDB
Done.
```

Out[71]:

Date	Time__UTC_	Booster_Version	Launch_Site	Payload	payload_mass__kg_	Orbit	Customer	Mis
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 [72]:
```
%%sql

SELECT SUM(payload_mass__kg_) AS TOTAL_NASA_mass
FROM SPACEXTBL
WHERE "Customer" LIKE '%CRS%';
```

```
 * ibm_db_sa://fgl32023:***@dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net:50000/B
LUDB
Done.
```

Out[72]:

total_nasa_mass
48213

Task 4: Display average payload mass carried by booster version F9 v1.1

In [73]:
```
%%sql

SELECT AVG(payload_mass__kg_) AS AVG_MASS
FROM SPACEXTBL
WHERE "Booster_Version" LIKE 'F9 v1.1%';
```

```
 * ibm_db_sa://fgl32023:***@dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net:50000/B
LUDB
Done.
```

Out[73]:

avg_mass
2534.666666

Task 5: List the date when the first successful landing outcome in ground pad was achieved

In [74]:

```
%%sql

SELECT MIN("Date")
FROM SPACEXTBL
WHERE "Landing__Outcome" LIKE '%ground pad%';
```

```
 * ibm_db_sa://fgl32023:***@dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net:50000/B
LUDB
Done.
```

Out[74]:

1
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 [75]:

```
%%sql

SELECT "Booster_Version", "Landing__Outcome", payload_mass__kg_
FROM SPACEXTBL
WHERE (("Landing__Outcome" ='Success (drone ship)') AND (payload_mass__kg_ > 4000 AND p
```

```
 * ibm_db_sa://fgl32023:***@dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net:50000/B
LUDB
Done.
```

Out[75]:

Booster_Version	Landing__Outcome	payload_mass__kg_
F9 FT B1022	Success (drone ship)	4696
F9 FT B1026	Success (drone ship)	4600
F9 FT B1021.2	Success (drone ship)	5300
F9 FT B1031.2	Success (drone ship)	5200

Task 7: List the total number of successful and failure mission outcomes

In [76]:

```
%%sql

SELECT COUNT(*),
SUM("Landing__Outcome" LIKE '%Success%') AS SUCCESSES,
SUM("Landing__Outcome" LIKE '%Failure%') AS FAILURES
FROM SPACEXTBL;
```

```
 * ibm_db_sa://fgl32023:***@dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net:50000/B
LUDB
Done.
```

Out[76]:

1	successes	failures
101	61	10

Task 8: List the names of the booster_versions which have carried the maximum payload mass. Use a subquery.

In [77]:

```
%%sql

SELECT "Booster_Version", payload_mass__kg_
FROM SPACEXTBL
WHERE payload_mass__kg_ = (SELECT MAX(payload_mass__kg_) FROM SPACEXTBL);
```

```
 * ibm_db_sa://fgl32023:***@dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net:50000/B
LUDB
Done.
```

Out[77]:

Booster_Version	payload_mass__kg_
F9 B5 B1048.4	15600
F9 B5 B1049.4	15600
F9 B5 B1051.3	15600
F9 B5 B1056.4	15600
F9 B5 B1048.5	15600
F9 B5 B1051.4	15600
F9 B5 B1049.5	15600
F9 B5 B1060.2	15600
F9 B5 B1058.3	15600
F9 B5 B1051.6	15600
F9 B5 B1060.3	15600
F9 B5 B1049.7	15600

Task 9: List the records which will display the month names, failure landing in drone ship outcomes, booster versions, and launch_sites for the months in 2015

In [78]:

```
%%sql

SELECT MONTHNAME("Date") AS Month, "Booster_Version", "Launch_Site", "Landing__Outcome"
FROM SPACEXTBL
WHERE (("Landing__Outcome" LIKE '%Failure%') AND YEAR("Date")=2015);
```

```
 * ibm_db_sa://fgl32023:***@dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net:50000/B
LUDB
Done.
```

Out[78]:

MONTH	Booster_Version	Launch_Site	Landing__Outcome
January	F9 v1.1 B1012	CCAFS LC-40	Failure (drone ship)
April	F9 v1.1 B1015	CCAFS LC-40	Failure (drone ship)

Task 10: Rank the count of successful landing_outcomes between date 2010-06-04 and 2017-03-20 in descending order

In [79]:
```
%%sql

SELECT "Landing__Outcome", COUNT(*) AS TOTAL
FROM SPACEXTBL
WHERE (("Landing__Outcome" LIKE '%Success%') AND ("Date" > '2010-06-04') AND ("Date" <
GROUP BY "Landing__Outcome"
ORDER BY "Landing__Outcome" DESC;
```

```
 * ibm_db_sa://fgl32023:***@dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net:50000/B
LUDB
Done.
```

Out[79]:

Landing__Outcome	total
Success (ground pad)	3
Success (drone ship)	5

Lab: Exploratory Data Analysis with Visualization

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.

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

- Exploratory Data Analysis
- Preparing Data Feature Engineering

In [80]:
```
# pandas is a software library written for the Python programming language for data man
import pandas as pd
#NumPy is a library for the Python programming language, adding support for large, mult
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a MatLab like plottin
import matplotlib.pyplot as plt
#Seaborn is a Python data visualization library based on matplotlib. It provides a high
import seaborn as sns

%matplotlib inline
```

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

In [81]:
```
df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-

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

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

df.head(5)
```

Out[81]:

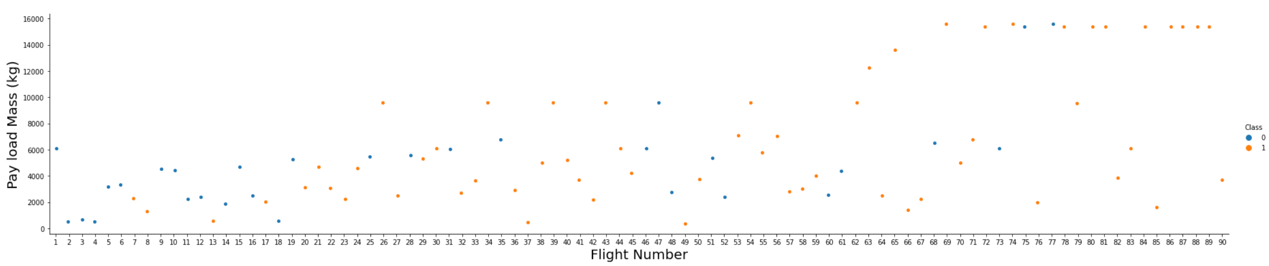
	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False
3	4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

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

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

In [82]:

```
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%.

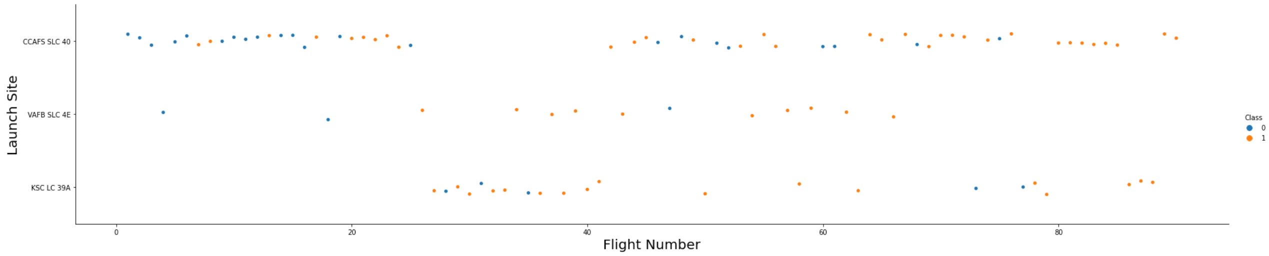
Next, let's drill down to each site visualize its detailed launch records.

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 [83]:

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

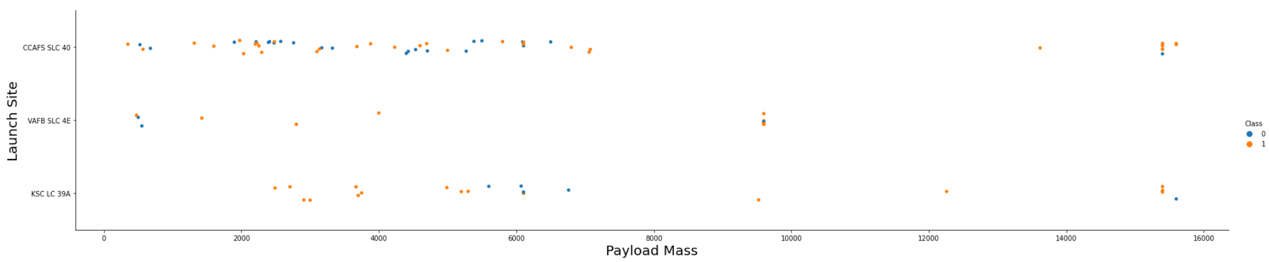


Now try to explain the patterns you found in the Flight Number vs. Launch Site scatter point plots.

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 [84]:
```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the
sns.catplot(y="LaunchSite", x="PayloadMass", hue="Class", data=df, aspect = 5)
plt.xlabel("Payload Mass",fontsize=20)
plt.ylabel("Launch Site",fontsize=20)
plt.show()
```



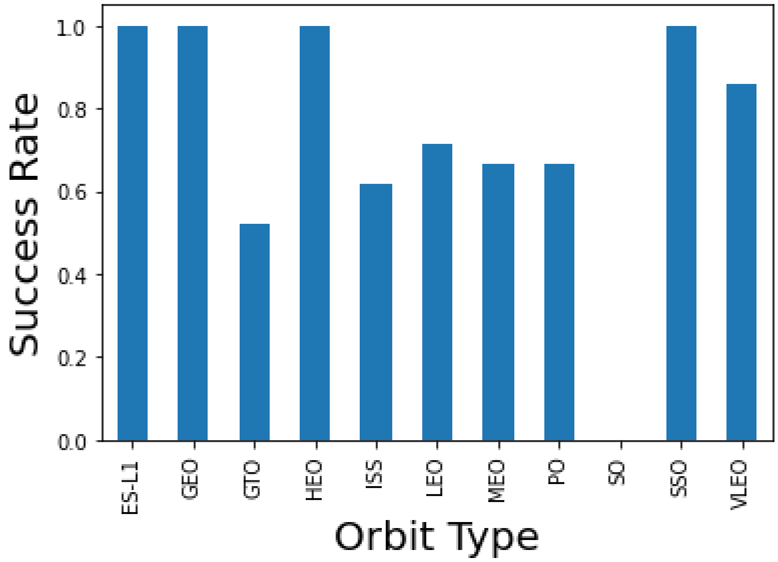
Task 3: Visualize the relationship between success rate of each orbit type

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

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

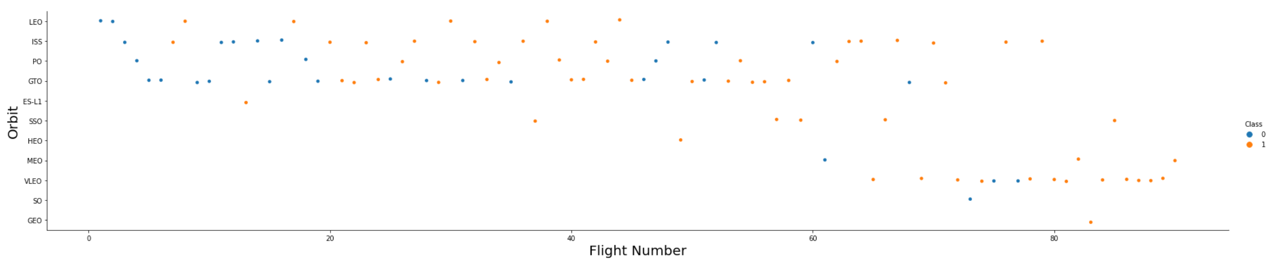
In [85]:
```
# HINT use groupby method on Orbit column and get the mean of Class column
df.groupby(['Orbit']).mean()['Class'].plot(kind='bar')
plt.xlabel("Orbit Type",fontsize=20)
plt.ylabel("Success Rate",fontsize=20)
plt.show()
```



Task 4: Visualize the relationship between the flight number and orbit type

In [86]:

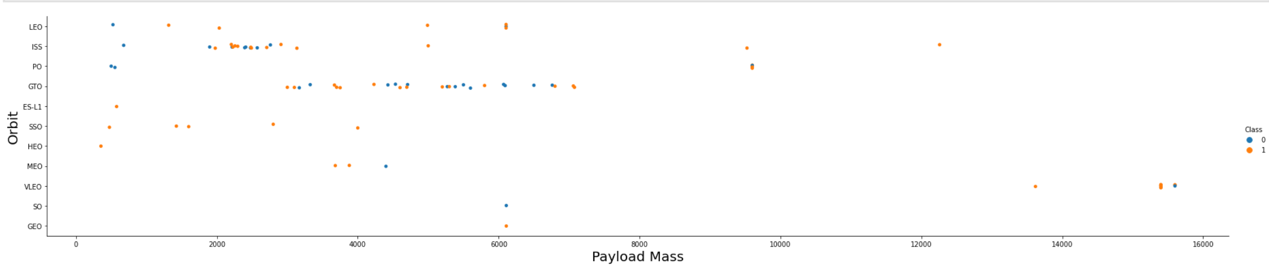
```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit,
sns.catplot(y="Orbit", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```



Task 5: Visualize the relationship between payload and orbit type

In [175…

```
# Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, and
sns.catplot(y="Orbit", x="PayloadMass", hue="Class", data=df, aspect = 5)
plt.xlabel("Payload Mass",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
# You should observe that Heavy payloads have a negative influence on GTO orbits and po
```



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:

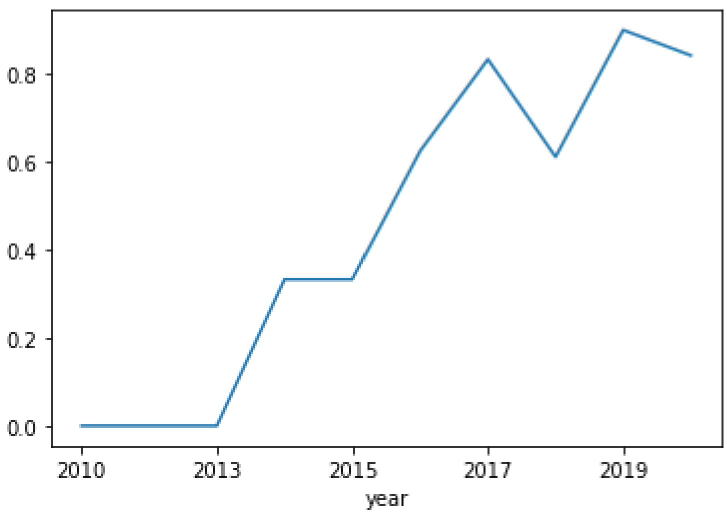
In [88]:
```
# A function to Extract years from the date
year=[]
def Extract_year(date):
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year
```

In [89]:
```
# create dataframe to hold the years
df1 = pd.DataFrame(Extract_year(df['Date']),columns =['year'])

# add class column to dataframe
df1['Class']=df['Class']

# use plot() function to create line plot
df1.groupby('year')['Class'].mean().plot()
```

Out[89]: <AxesSubplot:xlabel='year'>



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.

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 [90]:
```
features = df[['FlightNumber', 'PayloadMass', 'Orbit', 'LaunchSite', 'Flights', 'GridFi
features.head(3)
```

Out[90]:

	FlightNumber	PayloadMass	Orbit	LaunchSite	Flights	GridFins	Reused	Legs	LandingPad	Block

	FlightNumber	PayloadMass	Orbit	LaunchSite	Flights	GridFins	Reused	Legs	LandingPad	Block
0	1	6104.959412	LEO	CCAFS SLC 40	1	False	False	False	NaN	1.0
1	2	525.000000	LEO	CCAFS SLC 40	1	False	False	False	NaN	1.0
2	3	677.000000	ISS	CCAFS SLC 40	1	False	False	False	NaN	1.0

In [91]:

```
# create dummy variables
features_one_hot = pd.get_dummies(data = features, columns = ['Orbit', 'LaunchSite', 'L
```

In [92]:

```
# view dataframe
features_one_hot.head()
```

Out[92]:

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Orbit_ES-L1	Orbit_(
0	1	6104.959412	1	False	False	False	1.0	0	0	
1	2	525.000000	1	False	False	False	1.0	0	0	
2	3	677.000000	1	False	False	False	1.0	0	0	
3	4	500.000000	1	False	False	False	1.0	0	0	
4	5	3170.000000	1	False	False	False	1.0	0	0	

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 [93]:

```
# HINT: use astype function
features_one_hot.astype(float)
```

Out[93]:

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Orbit_ES-L1	Orbit
0	1.0	6104.959412	1.0	0.0	0.0	0.0	1.0	0.0	0.0	
1	2.0	525.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	
2	3.0	677.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	
3	4.0	500.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	
4	5.0	3170.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	
85	86.0	15400.000000	2.0	1.0	1.0	1.0	5.0	2.0	0.0	

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Orbit_ES-L1	Orbit
86	87.0	15400.000000	3.0	1.0	1.0	1.0	5.0	2.0	0.0	
87	88.0	15400.000000	6.0	1.0	1.0	1.0	5.0	5.0	0.0	
88	89.0	15400.000000	3.0	1.0	1.0	1.0	5.0	2.0	0.0	
89	90.0	3681.000000	1.0	1.0	0.0	1.0	5.0	0.0	0.0	

90 rows × 80 columns

Week 3: Interactive Visual Analytics and Dashboards

Lab: Interactive Visual Analytics with Folium

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`.

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.

In [94]:
```
# !pip3 install folium
# !pip3 install wget
```

In [95]:
```
import folium
import wget
import pandas as pd
```

In [96]:
```
# 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
```

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 [97]:
```
# Download and read the `spacex_launch_geo.csv`
spacex_csv_file = wget.download('https://cf-courses-data.s3.us.cloud-object-storage.app
spacex_df=pd.read_csv(spacex_csv_file)
```

```
100% [..........................................................................]
8966 / 8966
```

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

In [98]:
```
# Select relevant sub-columns: `Launch Site`, `Lat(Latitude)`, `Long(Longitude)`, `clas
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[98]:

	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	-120.610746

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 [99]:
```
# 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 [100…
```
# Create a blue circle at NASA Johnson Space Center's coordinate with a popup label sho
circle = folium.Circle(nasa_coordinate, radius=1000, color='#d35400', fill=True).add_ch
# Create a blue circle at NASA Johnson Space Center's coordinate with a icon showing it
marker = folium.map.Marker(
    nasa_coordinate,
    # Create an icon as a text label
    icon=DivIcon(
        icon_size=(20,20),
```

```
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % 'NASA JSC',
        )
    )
site_map.add_child(circle)
site_map.add_child(marker)
```

Out[100… 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

In [101…

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

#display the map
site_map
```

Out[101… Make this Notebook Trusted to load map: File -> Trust Notebook

In [102…

```
# For each launch site, add a Circle object based on its coordinate (Lat, Long) values.
for i in range(0, len(launch_sites_df)):
    folium.Circle(
    location = [launch_sites_df.iloc[i]['Lat'], launch_sites_df.iloc[i]['Long']],
    radius = 1000,
    color = 'blue',
    popup = launch_sites_df.iloc[i]['Launch Site']
    ).add_to(site_map)
```

In [103…

```
site_map
```

Out[103… Make this Notebook Trusted to load map: File -> Trust Notebook

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 [104…

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

Out[104…

	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.

In [105…

```
# create MarkerCluster object
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 [106…

```
# 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

launch_sites_df['marker_color'] = ""
```

In [107…

```
launch_sites_df
```

Out[107…

	Launch Site	Lat	Long	marker_color
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	-120.610746	

In [108…

```
# 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[108…

	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 [109…

```
# Add the Marker cluster to the site map
site_map.add_child(marker_cluster)

# for each row in spacex_df data frame
# create a Marker object with its coordinate
from folium import plugins

# instantiate a mark cluster object for the incidents in the dataframe
incidents = plugins.MarkerCluster().add_to(site_map)

# and customize the Marker's icon property to indicate if this launch was successed or 
marker_cluster = MarkerCluster()

for index, row in spacex_df.iterrows():

      folium.map.Marker((row['Lat'], row['Long']), icon=folium.Icon(color='white', icon

site_map.add_child(marker_cluster)
```

Out[109… Make this Notebook Trusted to load map: File -> Trust Notebook

Task 3: Calculate 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 [110…

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

site_map.add_child(mouse_position)
site_map
```

Out[110… 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 [111…

```
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 railway using MousePosition and calculate the distance between the railway point to the launch site.

In [112…

```
# distance_railway = calculate_distance(lat1, lon1, lat2, lon2)
distance_railway = calculate_distance(28.59281, -80.64637, 28.56312, -80.57629)
distance_railway
```

Out[112… 7.600267590603478

In [113…

```
# create and add a folium.Marker on your selected closest railway point on the map
marker_railway = folium.map.Marker(
    [28.59281, -80.64637],
    # Create an icon as a text label
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:10.2f} K
    )
)
# show the distance to the launch site using the icon property
```

In [114…

```
# Create a `folium.PolyLine` object using the railway point coordinate and launch site
# Create a marker with distance to a closest city, coastline, highway, etc.
# Draw a line between the marker to the launch site
latlongs = [[28.57205, -80.58527],[28.562302, -80.577356]]
nearest_rail_line = folium.PolyLine([latlongs],weight=1)
site_map.add_child(nearest_rail_line)
```

Out[114… Make this Notebook Trusted to load map: File -> Trust Notebook

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.

Week 4: Predictive Analysis (Classification)

Machine Learning Prediction:

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.

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

We will import the following libraries for the lab

In [115…

```
# Pandas is a software library written for the Python programming language for data man
import pandas as pd
# NumPy is a library for the Python programming language, adding support for large, mul
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a MatLab like plottin
import matplotlib.pyplot as plt
#Seaborn is a Python data visualization library based on matplotlib. It provides a high
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 [116…

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

    cm = confusion_matrix(y, y_predict)
    ax= plt.subplot()
    sns.heatmap(cm, annot=True, ax = ax); #annot=True to annotate cells
    ax.set_xlabel('Predicted labels')
    ax.set_ylabel('True labels')
    ax.set_title('Confusion Matrix');
    ax.xaxis.set_ticklabels(['did not land', 'land']); ax.yaxis.set_ticklabels(['did no
```

In [117…

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

data.head()
```

Out[117…

FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins

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

In [118…

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

X.head(100)
```

Out[118…

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_I
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 (one bracket []).

In [120…

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

Out[120…

```
array([0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1,
       1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
        1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
        1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1], dtype=int64)
```

Task 2: Standardize the data in X, then reassign it to the variable X using the transform below

In [121…

```
# students get this
transform = preprocessing.StandardScaler()
```

In [122…

```
# standardize the data in X
scaler = preprocessing.StandardScaler().fit(X)

# reassign it to variable X
X = scaler.transform(X)
```

Task 3: Use the function train_test_split to split the data into training and test data

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`.

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 [123…

```
# split data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = .2, random_state
print('Train set: ', X_train.shape, Y_train.shape)
print('Test set: ', X_test.shape, Y_test.shape)
# note we have only 18 test samples
```

```
Train set:  (72, 83) (72,)
Test set:  (18, 83) (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 [148…

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

In [149…

```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge

# create logistic regression object
lr=LogisticRegression()

# create gridsearchcv object
```

```
logreg_cv = GridSearchCV(lr, parameters, cv = 10)

# fit the model
logreg_cv.fit(X_train, Y_train)
```

Out[149…

```
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 [150…

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

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

Task 5: Calculate the accuracy on the test data using method score

In [151…

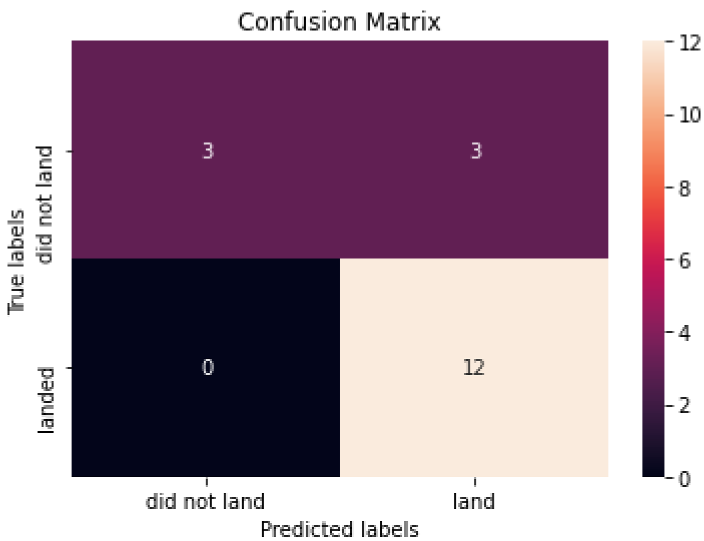
```
# calculate accuracy on test data
logreg_cv.score(X_test, Y_test)
```

Out[151…

```
0.8333333333333334
```

In [152…

```
# view confusion matrix
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.

In [153…

```
from sklearn import metrics

print("LR jaccard train score: ", metrics.accuracy_score(Y_train,logreg_cv.predict(X_tr
print("LR jaccard test score: ", metrics.accuracy_score(Y_test, yhat))
```

```
print("LR R-squared score: ", logreg_cv.score(X_test, Y_test))
print("LR gridsearch score", logreg_cv.best_score_)
```

```
LR jaccard train score:  0.875
LR jaccard test score:  0.8333333333333334
LR R-squared score:  0.8333333333333334
LR gridsearch score 0.8464285714285713
```

Task 6: Create a support vector machine object, then create a gridsearccv object svm_cv with cv = 10.

Fit the object to find the best parameters from the dictionary parameters.

In [154…

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

# create svm object
svm = SVC()

# create gridsearchcv object
svm_cv = GridSearchCV(svm, parameters, cv = 10)

# fit the model
svm_cv.fit(X_train, Y_train)
```

Out[154…

```
GridSearchCV(cv=10, estimator=SVC(),
             param_grid={'C': array([1.00000000e-03, 3.16227766e-02, 1.00000000e+00, 3.1
6227766e+01,
       1.00000000e+03]),
                         'gamma': array([1.00000000e-03, 3.16227766e-02, 1.00000000e+00,
3.16227766e+01,
       1.00000000e+03]),
                         'kernel': ('linear', 'rbf', 'poly', 'rbf', 'sigmoid')})
```

In [155…

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

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

Task 7: Calculate the accuracy on the test data with method score

In [156…

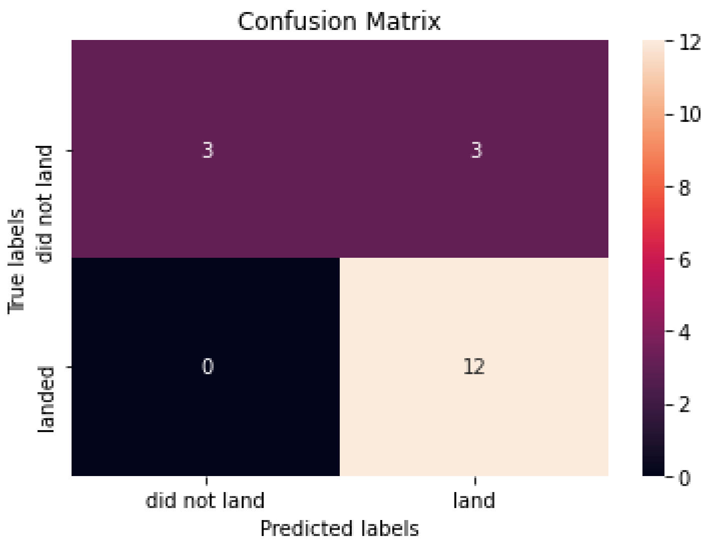
```
# calculate accuracy
svm_cv.score(X_test, Y_test)
```

Out[156…

```
0.8333333333333334
```

In [157…

```
# plot confusion matrix
yhat=svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



In [158…

```
print("LR jaccard train score: ", metrics.accuracy_score(Y_train,svm_cv.predict(X_train
print("LR jaccard test score: ", metrics.accuracy_score(Y_test, yhat))
print("LR R-squared score: ", svm_cv.score(X_test, Y_test))
print("LR gridsearch score", svm_cv.best_score_)
```

```
LR jaccard train score:  0.8888888888888888
LR jaccard test score:  0.8333333333333334
LR R-squared score:  0.8333333333333334
LR gridsearch score 0.8482142857142856
```

SVM can distinguish between classes, but the major problem is false positives.

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 [159…

```
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]}

# create tree object
tree = DecisionTreeClassifier()

# create gridsearchcv object
tree_cv = GridSearchCV(tree, parameters, cv = 10)

# fit the model
tree_cv.fit(X_train, Y_train)
```

Out[159…

```
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 [160…

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

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 4, 'max_fea
tures': 'auto', 'min_samples_leaf': 4, 'min_samples_split': 2, 'splitter': 'random'}
accuracy : 0.8767857142857143
```

Task 9: test the accuracy of tree_cv on test data using score method

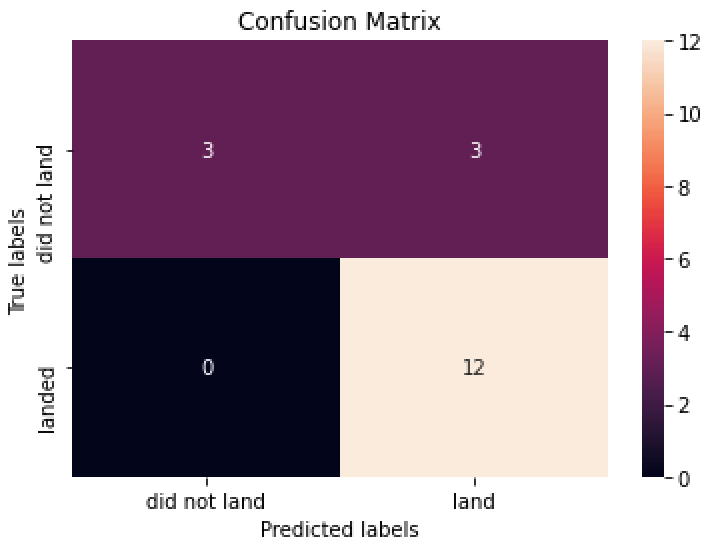
In [163…

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

Out[163… 0.8333333333333334

In [164…

```
# plot confusion matrix
yhat = tree_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



In [165…

```
print("LR jaccard train score: ", metrics.accuracy_score(Y_train,tree_cv.predict(X_trai
print("LR jaccard test score: ", metrics.accuracy_score(Y_test, yhat))
print("LR R-squared score: ", tree_cv.score(X_test, Y_test))
print("LR gridsearch score", tree_cv.best_score_)
```

```
LR jaccard train score:  0.8611111111111112
LR jaccard test score:  0.8333333333333334
LR R-squared score:  0.8333333333333334
LR gridsearch score 0.8767857142857143
```

Task 10: Create a KNN object, then create a gridsearchcv object knn_cv with cv = 10.

Fit the object to find the best parameters from the dictionary parameters.

In [170…

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

# create KNN object
KNN = KNeighborsClassifier()
```

```
# create gridsearchcv object
knn_cv = GridSearchCV(KNN, parameters, cv = 10)

# fit the model
knn_cv.fit(X_train, Y_train)
```

Out[170…
```
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                         'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'p': [1, 2]})
```

In [171…
```
print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
print("accuracy :",knn_cv.best_score_)
```

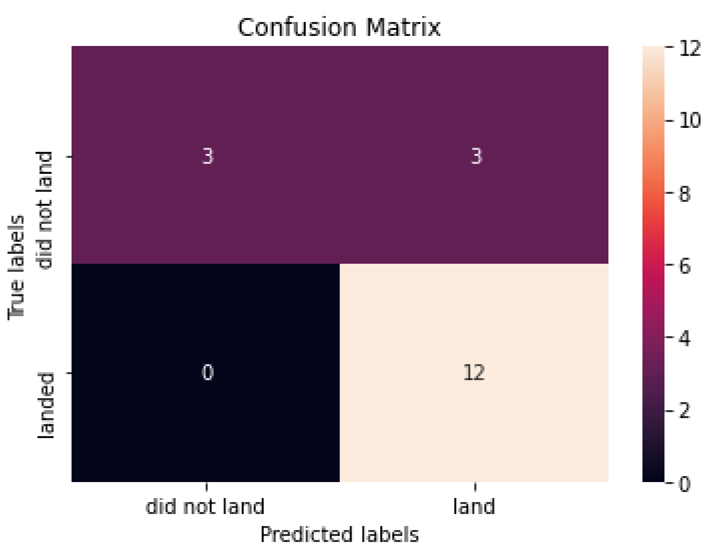
```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 10, 'p': 
1}
accuracy : 0.8482142857142858
```

Task 11: Calculate the accuracy of knn_CV on the test data using method score.

In [172…
```
# accuracy
knn_cv.score(X_test, Y_test)
```

Out[172…
```
0.8333333333333334
```

In [173…
```
# plot confusion matrix
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



In [174…
```
print("LR jaccard train score: ", metrics.accuracy_score(Y_train,knn_cv.predict(X_train
print("LR jaccard test score: ", metrics.accuracy_score(Y_test, yhat))
print("LR R-squared score: ", knn_cv.score(X_test, Y_test))
print("LR gridsearch score", knn_cv.best_score_)
```

```
LR jaccard train score:  0.8611111111111112
LR jaccard test score:  0.8333333333333334
LR R-squared score:  0.8333333333333334
LR gridsearch score 0.8482142857142858
```

Task 12: Find the method that performs best

ALGORITHM	JACCARD TRAIN ACCURACY SCORE (.accuracy_score())	JACCARD TEST ACCURACY SCORE (.accuracy_score)	TEST ACCURACY R-SQUARED SCORE (.score())	GRIDSEARCHCH Score (.BEST_SCORE)
Logistic Regression	.875	.8334	.8334	.8464
Support Vector Machine	.889	.8334	.8334	.8484
Decision Tree	.8611	.8333	.8333	.8768
K-Nearest Neighbors	.8611	.8334	.8334	.848

All the algorithms give roughly the same results and accuracy (.score()) ~ 83.3%

Therefore, all methods perform practically the same.

Week 5: Present Your Data-Driven Insights

Outline:

- cover page
- executive summary (briefly explain details of project; stand-alone document)
- table of contents
- introduction (explain the nature of the analysis, states the problem, gives the questions that are answered by the analysis)
- methodology (explains data sources used in analysis and outlines the plan for collected data)
- results (explains in detail of the data collection, how it was organized, how it was analyzed. This section also contains charts and graphs that substantiate the results)
- discussion (engage with the audience in a discussion of the implications drawn in the research)
- conclusion (reiterate the problem given in the intro and give summary of the findings)
- appendix (info that doesn't fit in the main body of the report, but still important enough to include)

```
In [ ]:
```