IBM Developer
SKILLS NETWORK

SpaceX Falcon 9 first stage Landing Prediction

Lab 1: Collecting the data

Estimated time needed: **45** minutes

In this capstone, we will predict if the Falcon 9 first stage will land successfully. SpaceX advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is because SpaceX can reuse the first stage. Therefore if we can determine if the first stage will land, we can determine the cost of a launch. This information can be used if an alternate company wants to bid against SpaceX for a rocket launch. In this lab, you will collect and make sure the data is in the correct format from an API. The following is an example of a successful and launch.

Several examples of an unsuccessful landing are shown here:

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

Objectives

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

- Request to the SpaceX API
- Clean the requested data

Import Libraries and Define Auxiliary Functions

We will import the following libraries into the lab

In [1]:
```
# Requests allows us to make HTTP requests which we will use to get data from an API
import requests
# Pandas is a software library written for the Python programming language for data
import pandas as pd
# NumPy is a library for the Python programming language, adding support for large,
import numpy as np
# Datetime is a library that allows us to represent dates
import datetime

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

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

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

In [44]:
```
# Takes the dataset and uses the rocket column to call the API and append the data t
def getBoosterVersion(data):
    for x in data['rocket']:
        response = requests.get("https://api.spacexdata.com/v4/rockets/"+str(x)).jso
        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 [43]:
```
# Takes the dataset and uses the launchpad column to call the API and append the dat
def getLaunchSite(data):
    for x in data['launchpad']:
        response = requests.get("https://api.spacexdata.com/v4/launchpads/"+str(x)).
        Longitude.append(response['longitude'])
        Latitude.append(response['latitude'])
        LaunchSite.append(response['name'])
```

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

In [45]:
```
# Takes the dataset and uses the payloads column to call the API and append the data
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 [46]:
```
# Takes the dataset and uses the cores column to call the API and append the data to
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cores/"+core[
                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 [27]:
```
spacex_url="https://api.spacexdata.com/v4/launches/past"
```

In [28]:
```
response = requests.get(spacex_url)
```

Check the content of the response

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

```
b'[{"fairings":{"reused":false,"recovery_attempt":false,"recovered":false,"ships":
[]},"links":{"patch":{"small":"https://images2.imgbox.com/3c/0e/T8iJcSN3_o.png","lar
ge":"https://images2.imgbox.com/40/e3/GypSkayF_o.png"},"reddit":{"campaign":null,"la
unch":null,"media":null,"recovery":null},"flickr":{"small":[],"original":[]},"pressk
it":null,"webcast":"https://www.youtube.com/watch?v=0a_00nJ_Y88","youtube_id":"0a_00
nJ_Y88","article":"https://www.space.com/2196-spacex-inaugural-falcon-1-rocket-lost-
launch.html","wikipedia":"https://en.wikipedia.org/wiki/DemoSat"},"static_fire_date_
utc":"2006-03-17T00:00:00.000Z","static_fire_date_unix":1142553600,"tbd":false,"ne
t":false,"window":0,"rocket":"5e9d0d95eda69955f709d1eb","success":false,"details":"E
ngine failure at 33 seconds and loss of vehicle","ships":[],"capsules":[],"payload
s":["5eb0e4b5b6c3bb0006eeb1e1"],"launchpad":"5e9e4502f5090995de566f86","auto_updat
e":true,"launch_library_id":null,"failures":[{"time":33,"altitude":null,"reason":"me
rlin engine failure"}],"crew":[],"flight_number":1,"name":"FalconSat","date_utc":"20
06-03-24T22:30:00.000Z","date_unix":1143239400,"date_local":"2006-03-25T10:30:00+12:
00","date_precision":"hour","upcoming":false,"cores":[{"core":"5e9e289df35918033d3b2
623","flight":1,"gridfins":false,"legs":false,"reused":false,"landing_attempt":fals
e,"landing_success":null,"landing_type":null,"landpad":null}],"id":"5eb87cd9ffd86e00
0604b32a"},{"fairings":{"reused":false,"recovery_attempt":false,"recovered":false,"s
hips":[]},"links":{"patch":{"small":"https://images2.imgbox.com/4f/e3/I0lkuJ2e_o.pn
g","large":"https://images2.imgbox.com/be/e7/iNqsqVYM_o.png"},"reddit":{"campaign":n
ull,"launch":null,"media":null,"recovery":null},"flickr":{"small":[],"original":
[]},"presskit":null,"webcast":"https://www.youtube.com/watch?v=Lk4zQ2wP-Nc","youtube
_id":"Lk4zQ2wP-Nc","article":"https://www.space.com/3590-spacex-falcon-1-rocket-fail
```

In [54]:
```
response.status_code
```

Out[54]:
```
200
```

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

In [71]:
```
# Use json_normalize meethod to convert the json result into a dataframe
object_ = requests.get(static_json_url)
json = object_.json()
data = pd.json_normalize(json)
```

Using the dataframe `data` print the first 5 rows

In [56]:
```
# Get the head of the dataframe
data.head()
```

Out[56]:

	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
1	None	NaN	False	False	0.0	5e9d0d95eda69955f709d1eb	False
2	None	NaN	False	False	0.0	5e9d0d95eda69955f709d1eb	False

	static_fire_date_utc	static_fire_date_unix	tbd	net	window	rocket	success
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 [72]:
```
# Lets take a subset of our dataframe keeping only the features we want and the flig
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc'

# We will remove rows with multiple cores because those are falcon rockets with 2 ex
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
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 th
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 [73]:
```
#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 [51]:
```
BoosterVersion
```

Out[51]:
```
[]
```

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

In [74]:
```
# Call getBoosterVersion
getBoosterVersion(data)
```

the list has now been update

In [60]:
```
BoosterVersion[0:5]
```

Out[60]:
```
['Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 9']
```

we can apply the rest of the functions here:

In [75]:
```
# Call getLaunchSite
getLaunchSite(data)
```

In [76]:
```
# Call getPayloadData
getPayloadData(data)
```

In [77]:
```
# Call getCoreData
getCoreData(data)
```

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

In [78]:
```
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 [79]:
```
# Create a data from launch_dict
data = pd.DataFrame(launch_dict)
```

Show the summary of the dataframe

In [80]:
```
# Show the head of the dataframe
data.head()
```

Out[80]:

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

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 [81]:
```
# Hint data['BoosterVersion']!='Falcon 1'

data_falcon9 = data[data['BoosterVersion']!='Falcon 1']
```

In [83]:
```
import pandas as pd
pd.options.mode.chained_assignment = None  # default='warn'
```

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

In [84]:
```
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9
```

Out[84]:

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

90 rows × 17 columns

Data Wrangling

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

In [85]:
```
data_falcon9.isnull().sum()
```

Out[85]:
```
FlightNumber      0
Date              0
BoosterVersion    0
PayloadMass       5
Orbit             0
LaunchSite        0
```

```
Outcome          0
Flights          0
GridFins         0
Reused           0
Legs             0
LandingPad      26
Block            0
ReusedCount      0
Serial           0
Longitude        0
Latitude         0
dtype: int64
```

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

Task 3: Dealing with Missing Values

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

In [86]:
```
# Calculate the mean value of PayloadMass column
avg_PayloadMass = data_falcon9["PayloadMass"].astype("float").mean(axis=0)
print("Average:", avg_PayloadMass)

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

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

Out[86]:
```
Average: 6123.547647058824
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)
```

In [88]:
```
data_falcon9.to_csv('dataset_1.csv', index=False)
```

Authors

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

Change Log

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

Objectives

Web scrap Falcon 9 launch records with `BeautifulSoup`:

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

First let's import required packages for this lab

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

In [1]:
```
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 [2]:
```
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.str
    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 [3]:
```
static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon
```

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

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

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

In [12]:
```
# use requests.get() method with the provided static_url
# assign the response to a object
response = requests.get(static_url)
response.status_code
```

Out[12]:
```
200
```

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

In [13]:
```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(response.content, 'html.parser')
```

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

In [16]:
```
# Use soup.title attribute
soup.title
```

Out[16]:
```
<title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

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

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

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

In [23]:
```
# Use the find_all function in the BeautifulSoup object, with element type `table`
# Assign the result to a list called `html_tables`
html_tables= list(soup.find_all('tr'))
```

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

In [24]:
```
# Let's print the third table and check its content
first_launch_table = html_tables[2]
print(first_launch_table)
```

```
<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="c
ite_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="#ci
te_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 f
irst-stage landing tests">Booster<br/>landing</a>
</th></tr>
```

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 [27]:
```
column_names = []

# Apply find_all() function with `th` element on first_launch_table
lst = list(first_launch_table.find_all('th'))

# Iterate each th element and apply the provided extract_column_from_header() to get
for th in lst:
    name = extract_column_from_header(th)

# Append the Non-empty column name (`if name is not None and len(name) > 0`) into a
    if name is not None and len(name) > 0:
        column_names.append(name)
```

Check the extracted column names

In [28]:
```
print(column_names)
```

```
['Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbi
t', 'Customer', 'Launch outcome']
```

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

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

In [36]:
```
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 [37]:

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

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

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

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

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

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

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

```
            #print(payload)

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

            # Customer
            # TODO: Append the customer into launch_dict with key `Customer`

            customer = row[6].a
            if customer is None:
                pass
            else:
                customer = customer.string
            launch_dict['Customer'].append(customer)

            #print(customer)

            # Launch outcome
            # TODO: Append the launch_outcome into launch_dict with key `Launch outc
            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 lan
            booster_landing = landing_status(row[8])
            launch_dict['Booster landing'].append(booster_landing)
            #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 [38]:
```
df=pd.DataFrame(launch_dict)
```

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

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

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

In [40]:
```
df.to_csv('dataset_2.csv', index=False)
```

Authors

Yan Luo

Nayef Abou Tayoun

Change Log

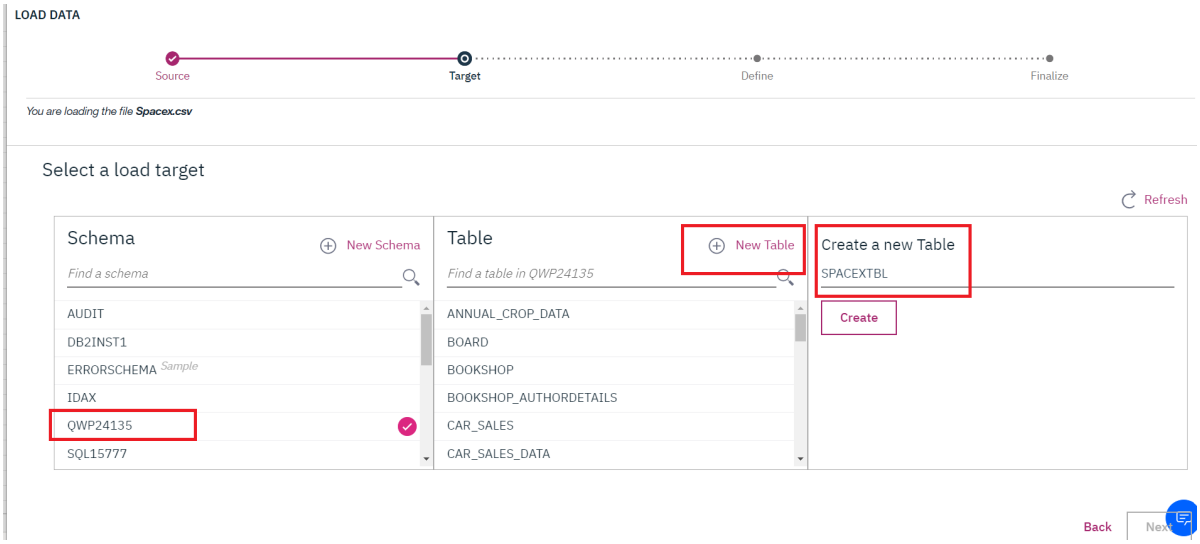
Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-06-09	1.0	Yan Luo	Tasks updates
2020-11-10	1.0	Nayef	Created the initial version

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.



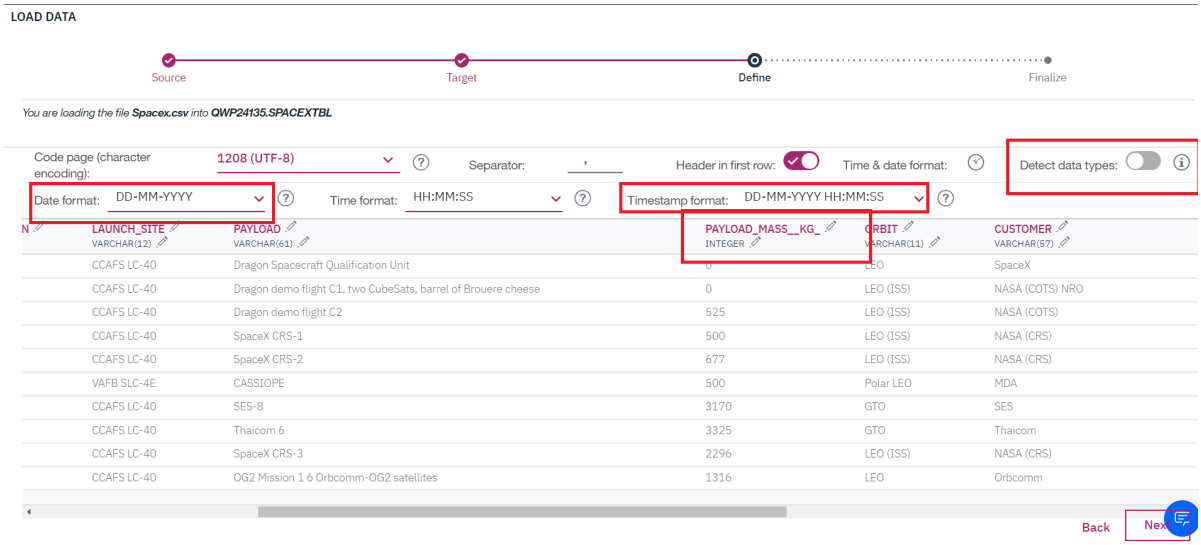
Now open the Db2 console, open the LOAD tool, Select / Drag the .CSV file for the dataset, Next create a New Table, and then follow the steps on-screen instructions to load the data. Name the new table as follows:

SPACEXDATASET

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

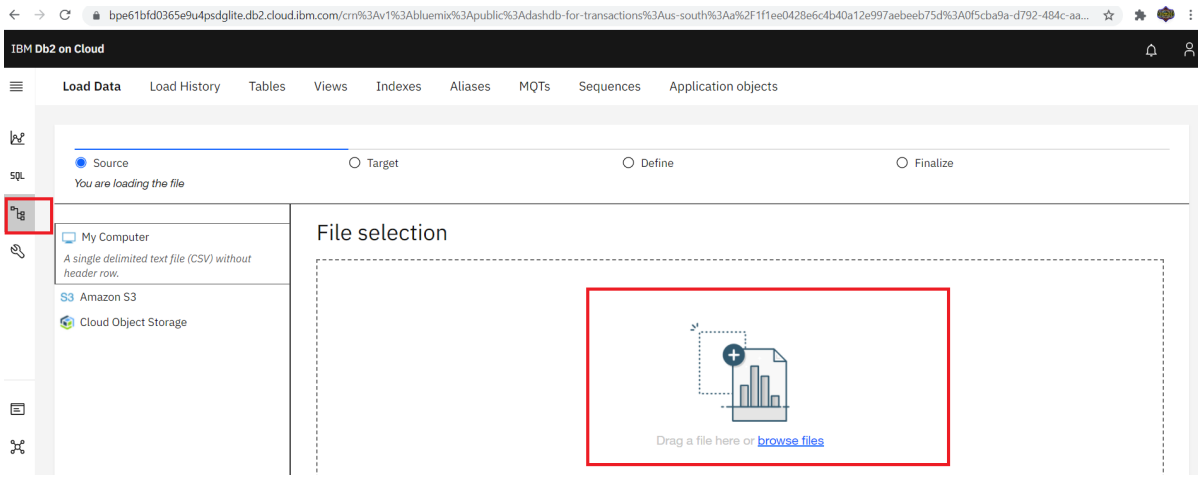
Note:While loading Spacex dataset, ensure that detect datatypes is disabled. Later click on the pencil icon(edit option).

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.

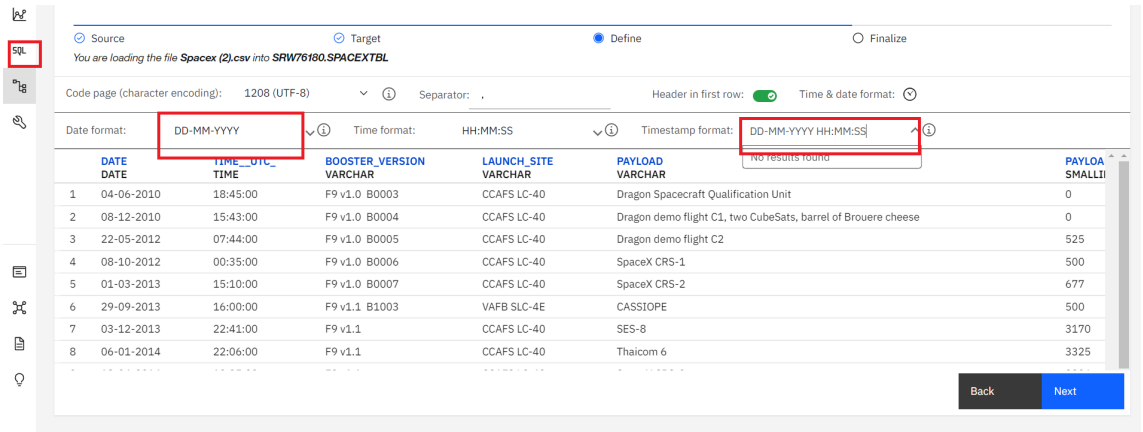


Changes to be considered when having DB2 instance with the new UI having Go to UI screen

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

```
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
Requirement already satisfied: sqlalchemy==1.3.9 in c:\users\kerry\appdata\local\pro
grams\python\python39\lib\site-packages (1.3.9)
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
Requirement already satisfied: ibm_db_sa in c:\users\kerry\appdata\local\programs\py
thon\python39\lib\site-packages (0.3.7)
Requirement already satisfied: sqlalchemy>=0.7.3 in c:\users\kerry\appdata\local\pro
grams\python\python39\lib\site-packages (from ibm_db_sa) (1.3.9)
Requirement already satisfied: ibm_db>=2.0.0 in c:\users\kerry\appdata\local\program
s\python\python39\lib\site-packages (from ibm_db_sa) (3.0.4)
```

Connect to the database

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

In [2]:

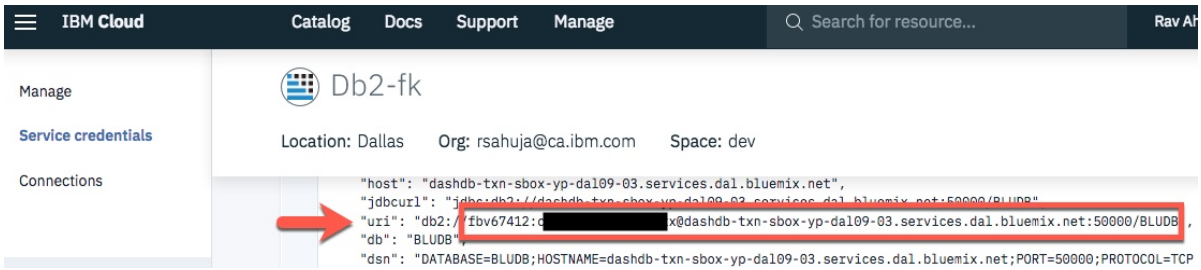
```
%load_ext sql
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_43568/3202754126.py in <module>
----> 1 get_ipython().run_line_magic('load_ext', 'sql')

c:\users\kerry\appdata\local\programs\python\python39\lib\site-packages\IPython\core
\interactiveshell.py in run_line_magic(self, magic_name, line, _stack_depth)
   2346                 kwargs['local_ns'] = self.get_local_scope(stack_depth)
   2347             with self.builtin_trap:
-> 2348                 result = fn(*args, **kwargs)
   2349             return result
   2350

c:\users\kerry\appdata\local\programs\python\python39\lib\site-packages\decorator.py
in fun(*args, **kw)
    230             if not kwsyntax:
    231                 args, kw = fix(args, kw, sig)
--> 232             return caller(func, *(extras + args), **kw)
    233     fun.__name__ = func.__name__
    234     fun.__doc__ = func.__doc__

c:\users\kerry\appdata\local\programs\python\python39\lib\site-packages\IPython\core
\magic.py in <lambda>(f, *a, **k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188
    189         if callable(arg):

c:\users\kerry\appdata\local\programs\python\python39\lib\site-packages\IPython\core
\magics\extension.py in load_ext(self, module_str)
     31         if not module_str:
     32             raise UsageError('Missing module name.')
---> 33         res = self.shell.extension_manager.load_extension(module_str)
     34
     35         if res == 'already loaded':

c:\users\kerry\appdata\local\programs\python\python39\lib\site-packages\IPython\core
\extensions.py in load_extension(self, module_str)
     78             if module_str not in sys.modules:
     79                 with prepended_to_syspath(self.ipython_extension_dir):
---> 80                     mod = import_module(module_str)
     81                     if mod.__file__.startswith(self.ipython_extension_dir):
     82                         print(("Loading extensions from {dir} is deprecated.
"
```

```
c:\users\kerry\appdata\local\programs\python\python39\lib\importlib\__init__.py in i
mport_module(name, package)
    125                 break
    126             level += 1
--> 127     return _bootstrap._gcd_import(name[level:], package, level)
    128
    129

c:\users\kerry\appdata\local\programs\python\python39\lib\importlib\_bootstrap.py in
_gcd_import(name, package, level)

c:\users\kerry\appdata\local\programs\python\python39\lib\importlib\_bootstrap.py in
_find_and_load(name, import_)

c:\users\kerry\appdata\local\programs\python\python39\lib\importlib\_bootstrap.py in
_find_and_load_unlocked(name, import_)

ModuleNotFoundError: No module named 'sql'
```

DB2 magic in case of old UI service credentials.

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



in the following format

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

DB2 magic in case of new UI service credentials.

- Use the following format.

- Add security=SSL at the end

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

In [3]:
```
import ibm_db
import pandas
import ibm_db_dbi

##
dsn_hostname = "ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databases.
dsn_uid = "nmc92672"        # e.g. "abc12345"
dsn_pwd = "XSH8unF0oocZTavQ"      # e.g. "7dBZ3wWt9XN6$o0J"

dsn_driver = "{IBM DB2 ODBC DRIVER}"
dsn_database = "BLUDB"            # e.g. "BLUDB"
dsn_port = "31321"                # e.g. "32733"
dsn_protocol = "TCPIP"            # i.e. "TCPIP"
dsn_security = "SSL"              #i.e. "SSL"

dsn = (
    "DRIVER={0};"
    "DATABASE={1};"
    "HOSTNAME={2};"
    "PORT={3};"
    "PROTOCOL={4};"
    "UID={5};"
    "PWD={6};"
    "SECURITY={7};").format(dsn_driver, dsn_database, dsn_hostname, dsn_port, dsn_pr

try:
    conn = ibm_db.connect(dsn, "", "")
    print ("Connected to database: ", dsn_database, "as user: ", dsn_uid, "on host:

except:
    print ("Unable to connect: ", ibm_db.conn_errormsg() )

##
```

```
Connected to database:  BLUDB as user:  nmc92672 on host:  ba99a9e6-d59e-4883-8fc0-d
6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud
```

In [ ]:
```
%sql ibm_db_sa://
```

Tasks

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

Task 1

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

In [5]:
```
pconn = ibm_db_dbi.Connection(conn)
selectQuery = "select distinct Launch_Site from SPACE"
pandas.read_sql(selectQuery, pconn)
```

Out[5]:

	LAUNCH_SITE

	LAUNCH_SITE
0	CCAFS LC-40
1	CCAFS SLC-40
2	KSC LC-39A
3	VAFB SLC-4E

Task 2

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

In [6]:
```
selectQuery = "select * from SPACE where Launch_Site like 'CCA%' limit 5"
pandas.read_sql(selectQuery, pconn)
```

Out[6]:

	DATE	TIME__UTC_	BOOSTER_VERSION	LAUNCH_SITE	PAYLOAD	PAYLOAD_MASS__KG_	ORBIT
0	2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO
1	2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of...	0	LEO (ISS)
2	2012-05-22	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)
3	2012-10-08	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)
4	2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)

Task 3

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

In [13]:
```
selectQuery = "select SUM(PAYLOAD_MASS__KG_) from SPACE where CUSTOMER like 'NASA (C
pandas.read_sql(selectQuery, pconn)
```

Out[13]:

	1
0	45596

Task 4

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

In [ ]:
```

```

Task 5

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

Hint:Use min function

In [ ]:

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 [ ]:

Task 7

List the total number of successful and failure mission outcomes

In [ ]:

Task 8

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

In [ ]:

Task 9

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

In [ ]:

Task 10

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

In [ ]:

Reference Links

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

Author(s)

Lakshmi Holla

Other Contributors

Rav Ahuja

Change log

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