Getting Data from SpaceX API

Objectives

Make a get request to the SpaceX API as well as basic data wrangling and formating.

- Request to the SpaceX API
- Clean the requested data

Import Libraries and Define Auxiliary Functions

In [49]:
```
import requests # make HTTP requests which we will use to get data from an API
import pandas as pd # data manipulation and analysis
import numpy as np # matrix operations and high-level math for arrays
import datetime # represent dates

pd.set_option('display.max_columns', None) # print all columns in df
pd.set_option('display.max_colwidth', None) # show all of the data for the fea
ture
```

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

In [50]:
```
# Takes the dataset and uses the rocket column to call the API and append the
 data to the list
def getBoosterVersion(data):
    for x in data['rocket']:
        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 [51]:
```
# Takes the dataset and uses the launchpad column to call the API and append t
he data to the list
def getLaunchSite(data):
    for x in data['launchpad']:
        response = requests.get("https://api.spacexdata.com/v4/launchpads/"+st
r(x)).json()
        Longitude.append(response['longitude'])
        Latitude.append(response['latitude'])
        LaunchSite.append(response['name'])
```

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

In [52]:
```
# Takes the dataset and uses the payloads column to call the API and append th
e data to the lists
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 [53]:
```
# Takes the dataset and uses the cores column to call the API and append the d
ata to the lists
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cores/"
+core['core']).json()
                Block.append(response['block'])
                ReusedCount.append(response['reuse_count'])
                Serial.append(response['serial'])
            else:
                Block.append(None)
                ReusedCount.append(None)
                Serial.append(None)
            Outcome.append(str(core['landing_success'])+' '+str(core['landing_
type']))
            Flights.append(core['flight'])
            GridFins.append(core['gridfins'])
            Reused.append(core['reused'])
            Legs.append(core['legs'])
            LandingPad.append(core['landpad'])
```

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

In [54]:
```
spacex_url="https://api.spacexdata.com/v4/launches/past"
```

In [55]:
```
response = requests.get(spacex_url)
```

Check the content of the response

In [56]:
```
#print(response.content)
```

The response contains massive information about SpaceX launches.

Next, discover some more relevant information for this project.

Request and parse the SpaceX launch data using the GET request

Static response object:

In [57]:
```
static_json_url= 'https://cf-courses-data.s3.us.cloud-object-storage.appdomai
n.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_call_spacex_api.json'
```

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

In [58]:
```
response.status_code
```

Out[58]: 200

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

In [59]:
```
# Use json_normalize meethod to convert the json result into a dataframe
data = pd.json_normalize(response.json())
```

Using the dataframe `data` print the first few rows

In [60]:
```
data.head(1)
```

Out[60]:

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

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 [61]:
```
# Lets take a subset of our dataframe keeping only the features we want and th
e flight number, and date_utc.
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'dat
e_utc']]

# We will remove rows with multiple cores because those are falcon rockets wit
h 2 extra rocket boosters and rows that have multiple payloads in a single roc
ket.
data = data[data['cores'].map(len)==1]
data = data[data['payloads'].map(len)==1]

# Since payloads and cores are lists of size 1 we will also extract the single
value in the list and replace the feature.
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 extract
ing the date leaving the time
data['date'] = pd.to_datetime(data['date_utc']).dt.date

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

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

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

In [62]:
```
#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 [63]:
```
BoosterVersion
```

Out[63]:
```
[]
```

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

In [64]:
```
# Call getBoosterVersion
getBoosterVersion(data)
```

the list has now been update

In [65]:
```
BoosterVersion[0:5]
```

Out[65]:
```
['Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 9']
```

we can apply the rest of the functions here:

In [66]:
```
getLaunchSite(data)
```

In [67]:
```
getPayloadData(data)
```

In [68]:
```
getCoreData(data)
```

construct dataset

In [69]:
```
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 [70]:
```
data_falcon9 = pd.DataFrame(launch_dict)
```

Show the summary of the dataframe

In [71]:
```
data_falcon9.head()
```

Out[71]:

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

Filter the dataframe to only include `Falcon 9` launches

Finally we will remove the Falcon 1 launches keeping only the Falcon 9 launches. Filter the data dataframe using the `BoosterVersion` column to only keep the Falcon 9 launches. Save the filtered data to a new dataframe called `data_falcon9` .

In [72]:
```
data_falcon9 = data_falcon9[data_falcon9.BoosterVersion !='Falcon 1']

#This will also work
#data_falcon9.drop(data_falcon9.loc[data_falcon9['BoosterVersion'] !='Falcon
 1'].index, inplace=True)

data_falcon9.head()
```

Out[72]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	Gric
4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	
5	8	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	
6	10	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	
7	11	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	
8	12	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	

Data Wrangling

Still has missing values...

In [73]:
```
data_falcon9.isnull().sum()
```

Out[73]:
```
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
```

Note: The `LandingPad` column will retain None values to represent when landing pads were not used.

Dealing with Missing Values

Calculating the mean for the `PayloadMass` using the `.mean()` .

Then useing mean and the `.replace()` function to replace `np.nan` values in the data with the mean.

In [74]:
```
# Calculate the mean value of PayloadMass column
PayloadMass_mean = data_falcon9['PayloadMass'].mean()
print("PayloadMass mean: ", PayloadMass_mean)

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

```
PayloadMass mean:  6123.547647058824
```

Out[74]:
```
4      6123.547647
5       525.000000
6       677.000000
7       500.000000
8      3170.000000
           ...
89    15600.000000
90    15600.000000
91    15600.000000
92    15600.000000
93     3681.000000
Name: PayloadMass, Length: 90, dtype: float64
```

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` .

Export to CSV

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)
```