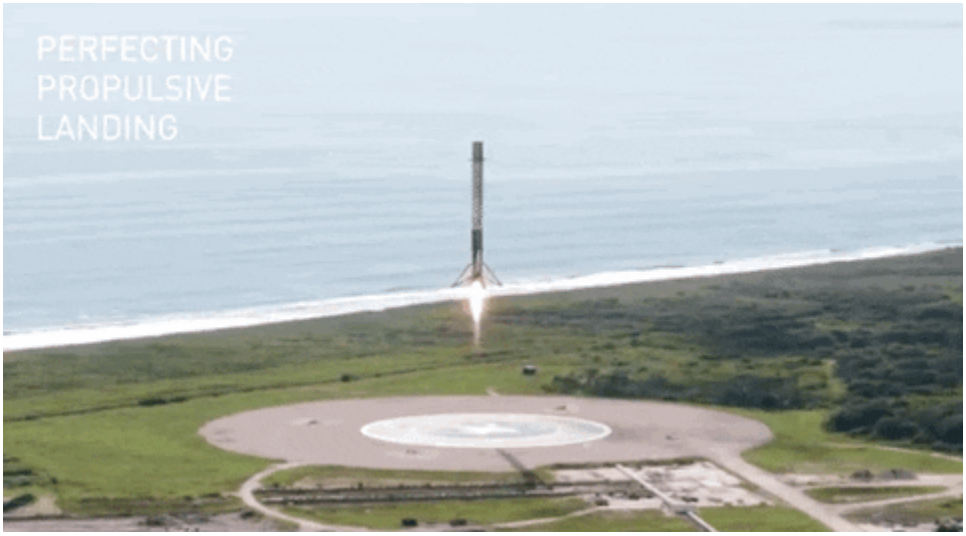
SpaceX Falcon 9 first stage Landing Prediction

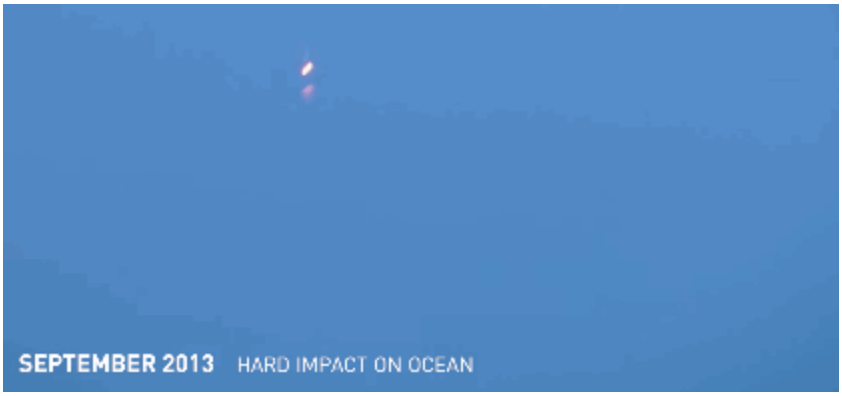
Lab 1: Collecting the data

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

- Request to the SpaceX API
- Clean the requested data

- Import Libraries

```
# 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 ma
import pandas as pd
# NumPy is a library for the Python programming language, adding support for large, mu
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)
```

Define a series of helper functions that will help in extractin information using identification numbers from the launch data.

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

```
# Takes the dataset and uses the rocket column to call the API and append the data to
```

```
def getBoosterVersion(data):
    for x in data['rocket']:
       if x:
        response = requests.get("https://api.spacexdata.com/v4/rockets/"+str(x)).json(
        BoosterVersion.append(response['name'])
```

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

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

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

From `cores` we would like to learn the outcome of the landing, the type of the landing, number of flights with that core, whether gridfins were used, wheter the core is reused, wheter legs were used, the landing pad used, the block of the core which is a number used to seperate version of cores, the number of times this specific core has been reused, and the serial of the core.

```
# Takes the dataset and uses the cores column to call the API and append the data to t
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cores/"+core['c
                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']))
```

```
14             Flights.append(core['flight'])
15             GridFins.append(core['gridfins'])
16             Reused.append(core['reused'])
17             Legs.append(core['legs'])
18             LandingPad.append(core['landpad'])
```

Start requesting rocket launch data from SpaceX API with the following URL:

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

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

Check the content of the response

```
1 print(response.content)
```

```
b'[{"fairings":{"reused":false,"recovery_attempt":false,"recovered":false,"ships":[]}
```

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

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

```
1 response.status_code
```

```
200
```

```
1 # convert the json result into a dataframe using json_normalize method
2 data=pd.json_normalize(response.json())
```

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

	static_fire_date_utc	static_fire_date_unix	net	window	roc
0	2006-03-17T00:00:00.000Z	1.142554e+09	False	0.0	5e9d0d95eda69955f709

1	None	NaN	False	0.0	5e9d0d95eda69955f709
2	None	NaN	False	0.0	5e9d0d95eda69955f709
3	2008-09-20T00:00:00.000Z	1.221869e+09	False	0.0	5e9d0d95eda69955f709
4	None	NaN	False	0.0	5e9d0d95eda69955f709

```
1 data.shape
```

```
(187, 43)
```

```
1 pd.DataFrame(data.columns) # View columns
```

	0
0	static_fire_date_utc
1	static_fire_date_unix
2	net
3	window
4	rocket
5	success
6	failures
7	details
8	crew
9	ships
10	capsules
11	payloads
12	launchpad
13	flight_number
14	name
15	date_utc
16	date_unix
17	date_local
18	date_precision
19	upcoming
20	cores
21	auto_update
22	tbd
23	launch_library_id

23	launch_library_id
24	id
25	fairings.reused
26	fairings.recovery_attempt
27	fairings.recovered
28	fairings.ships
29	links.patch.small
30	links.patch.large
31	links.reddit.campaign
32	links.reddit.launch
33	links.reddit.media
34	links.reddit.recovery
35	links.flickr.small
36	links.flickr.original
37	links.presskit
38	links.webcast
39	links.youtube_id
40	links.article
41	links.wikipedia
42	fairings

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.

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`.

```
# Lets take a subset of our dataframe keeping only the features we want and the flight r
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 da
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.

```
#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:

```
BoosterVersion # Verify the BoosterVersion is empty
```

```
[]
```

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

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

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

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

```
BoosterVersion[0:5] # Check the lists have been updated
```

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

Construct a dataset using the data we have obtained. We we combine the columns into a dictionary.

```
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}

```

```
# Create a data from launch_dict
launch_data=pd.DataFrame(launch_dict)
```

Show the summary of the dataframe

```
1 # Show the head of the dataframe
2 launch_data.head()
```

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

Next steps: Generate code with `launch_data` View recommended plots

Task 2: Filter the dataframe to only include `Falcon 9` launches

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`.

```
1 # Hint data['BoosterVersion']!='Falcon 1'
2 data_falcon9=launch_data[launch_data['BoosterVersion']!='Falcon 1']
3 print(data_falcon9)
```

```
    FlightNumber        Date BoosterVersion  PayloadMass Orbit    LaunchSite  \
4              6  2010-06-04       Falcon 9          NaN   LEO  CCSFS SLC 40
5              8  2012-05-22       Falcon 9        525.0   LEO  CCSFS SLC 40
6             10  2013-03-01       Falcon 9        677.0   ISS  CCSFS SLC 40
7             11  2013-09-29       Falcon 9        500.0    PO   VAFB SLC 4E
8             12  2013-12-03       Falcon 9       3170.0   GTO  CCSFS SLC 40
..           ...         ...            ...          ...   ...           ...
89           102  2020-09-03       Falcon 9      15600.0  VLEO    KSC LC 39A
90           103  2020-10-06       Falcon 9      15600.0  VLEO    KSC LC 39A
91           104  2020-10-18       Falcon 9      15600.0  VLEO    KSC LC 39A
92           105  2020-10-24       Falcon 9      15600.0  VLEO  CCSFS SLC 40
93           106  2020-11-05       Falcon 9       3681.0   MEO  CCSFS SLC 40

       Outcome  Flights  GridFins  Reused   Legs               LandingPad  \
4     None None        1     False   False  False                     None
```

```
4     None None        1     False   False  False                        None
5     None None        1     False   False  False                        None
6     None None        1     False   False  False                        None
7    False Ocean       1     False   False  False                        None
8     None None        1     False   False  False                        None
..         ...      ...       ...     ...    ...                         ...
89    True ASDS        2      True    True   True  5e9e3032383ecb6bb234e7ca
90    True ASDS        3      True    True   True  5e9e3032383ecb6bb234e7ca
91    True ASDS        6      True    True   True  5e9e3032383ecb6bb234e7ca
92    True ASDS        3      True    True   True  5e9e3033383ecbb9e534e7cc
93    True ASDS        1      True   False   True  5e9e3032383ecb6bb234e7ca

    Block  ReusedCount Serial   Longitude   Latitude
4     1.0            0  B0003  -80.577366  28.561857
5     1.0            0  B0005  -80.577366  28.561857
6     1.0            0  B0007  -80.577366  28.561857
7     1.0            0  B1003 -120.610829  34.632093
8     1.0            0  B1004  -80.577366  28.561857
..    ...          ...    ...         ...        ...
89    5.0           12  B1060  -80.603956  28.608058
90    5.0           13  B1058  -80.603956  28.608058
91    5.0           12  B1051  -80.603956  28.608058
92    5.0           12  B1060  -80.577366  28.561857
93    5.0            8  B1062  -80.577366  28.561857

[90 rows x 17 columns]
```

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

```
1 data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
2 data_falcon9.head(5)
3 data_falcon9.describe()
4
```

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Longitude	Lati
count	90.000000	85.000000	90.000000	90.000000	90.000000	90.000000	90.0(
mean	45.500000	6123.547647	1.788889	3.500000	3.188889	-86.366477	29.4(
std	26.124701	4870.916417	1.213172	1.595288	4.194417	14.149518	2.1(
min	1.000000	350.000000	1.000000	1.000000	0.000000	-120.610829	28.5(
25%	23.250000	2482.000000	1.000000	2.000000	0.000000	-80.603956	28.5(
50%	45.500000	4535.000000	1.000000	4.000000	1.000000	-80.577366	28.5(
75%	67.750000	9600.000000	2.000000	5.000000	4.000000	-80.577366	28.6(
max	90.000000	15600.000000	6.000000	5.000000	13.000000	-80.577366	34.6(

Data Wrangling

Data Wrangling

Some of the rows are missing values in our dataset and we need to deal with missing data

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

```
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.

```
1 # Calculate the mean value of PayloadMass column
2 # Replace the np.nan values with its mean value
3 # # Replacing the np.nan values with mean value of PayloadMass column
4 x=data_falcon9['PayloadMass'].mean()
5 data_falcon9['PayloadMass'].replace(np.nan,x, inplace=True)
6 data_falcon9.isnull().sum()
```

```
<ipython-input-29-20c06846807a>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  data_falcon9['PayloadMass'].replace(np.nan,x, inplace=True)
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
```

```
1 x
```

```
6123.547647058824
```

```
1 data_falcon9.head()
```

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

```
1 # Calculate the mean value of PayloadMass column
2 payloadmassavg = data_falcon9['PayloadMass'].mean()
3 # Replace the np.nan values with its mean value
4 data_falcon9['PayloadMass'].replace(np.nan, payloadmassavg, inplace=True)
```

```
<ipython-input-32-2980b6e868f8>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  data_falcon9['PayloadMass'].replace(np.nan, payloadmassavg, inplace=True)
```

Import data into dataset_part_1 CSV file

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

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

```
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
```