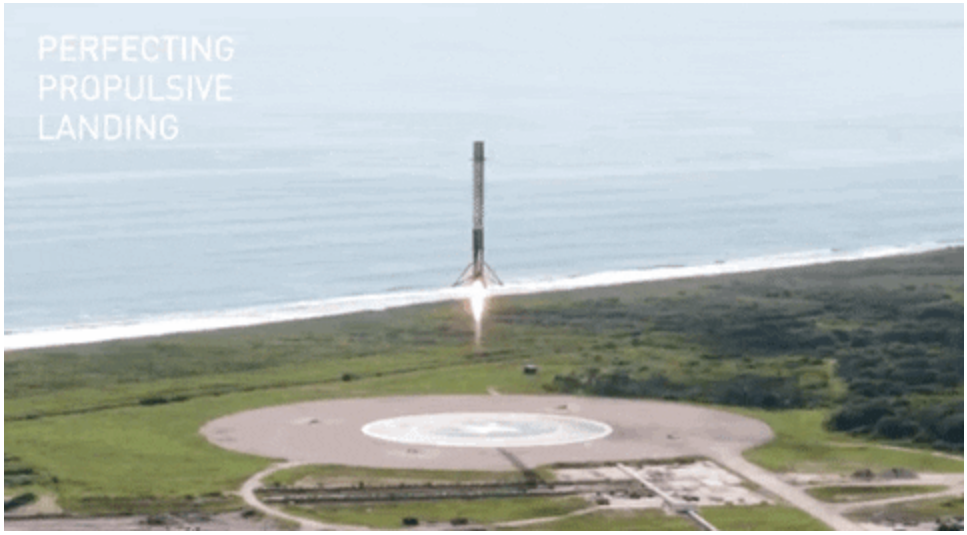
Sricharan Nallan Chakravarthula ¶

Space X Falcon 9 First Stage Landing Prediction

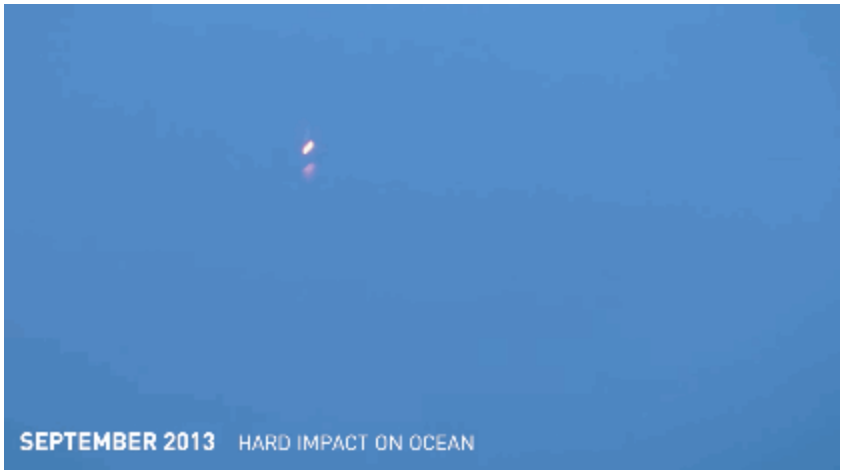
Lab 1: Collecting the data

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

- Get data from SpaceX API
- Clean/Wrangle/Format the data set

Import Libraries

In [1]:
```
import requests
import pandas as pd
import numpy as np
import datetime
pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth', None)
print("All libraries have been imported.")
```

```
All libraries have been imported.
```

Define Useful Functions

> **Note**
> These code bits were provided.

In [2]:
```
# NOTE: This code was provided.
# Takes the dataset and uses the rocket column to call the API and append
def getBoosterVersion(data):
    for x in data['rocket']:
       if x:
        response = requests.get("https://api.spacexdata.com/v4/rockets/"+s
        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 [3]:
```
# NOTE: This code was provided.
# Takes the dataset and uses the launchpad column to call the API and appe
def getLaunchSite(data):
    for x in data['launchpad']:
       if x:
         response = requests.get("https://api.spacexdata.com/v4/launchpads
         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 [4]:
```
# NOTE: This code was provided.
# Takes the dataset and uses the payloads column to call the API and appen
def getPayloadData(data):
    for load in data['payloads']:
       if load:
        response = requests.get("https://api.spacexdata.com/v4/payloads/"+
        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 [5]:

```
# NOTE: This code was provided.
# Takes the dataset and uses the cores column to call the API and append t
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cor
                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['land
            Flights.append(core['flight'])
            GridFins.append(core['gridfins'])
            Reused.append(core['reused'])
            Legs.append(core['legs'])
            LandingPad.append(core['landpad'])
```

Start Here

DataFrame of Launch Data - All

Task 1

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

In [6]:
```
# Convert JSON file into DataFrame
static_json_url = 'https://cf-courses-data.s3.us.cloud-object-storage.appd
response = requests.get(static_json_url)
response_json = response.json()
data_initial = pd.json_normalize(response_json)
data_initial.head(1)
```

Out[6]:

	static_fire_date_utc	static_fire_date_unix	tbd	net	window	rocke
0	2006-03-17T00:00:00.000Z	1.142554e+09	False	False	0.0	5e9d0d95eda69955f709d1el

In [7]:
```
data_initial.shape
```

Out[7]:
```
(107, 42)
```

In [8]:

```
# View column names
pd.DataFrame(data_initial.columns)
```

Out[8]:

	0
0	static_fire_date_utc
1	static_fire_date_unix
2	tbd
3	net
4	window
5	rocket
6	success
7	details
8	crew
9	ships
10	capsules
11	payloads
12	launchpad
13	auto_update
14	failures
15	flight_number
16	name
17	date_utc
18	date_unix
19	date_local
20	date_precision
21	upcoming
22	cores
23	id
24	fairings.reused
25	fairings.recovery_attempt
26	fairings.recovered
27	fairings.ships
28	links.patch.small
29	links.patch.large
30	links.reddit.campaign
31	links.reddit.launch
32	links.reddit.media
33	links.reddit.recovery
34	links.flickr.small
35	links.flickr.original

	0
36	links.presskit
37	links.webcast
38	links.youtube_id
39	links.article
40	links.wikipedia
41	fairings

DataFrame of Launch Data - Selected Information

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 [9]:
```
# Lets take a subset of our dataframe keeping only the features we want an
data = data_initial[['rocket', 'payloads', 'launchpad', 'cores', 'flight_n

# We will remove rows with multiple cores because those are falcon rockets
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 si
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 ext
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 [10]:
```
# Set global variables to be empty lists
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
ReusedCount = []
Serial = []
Longitude = []
Latitude = []
```

In [11]:
```
# Confirm list to be empty
BoosterVersion
```

Out[11]:
```
[]
```

In [12]:
```
# Call getBoosterVersion
getBoosterVersion(data)
```

In [13]:
```
# Call getLaunchSite
getLaunchSite(data)
```

In [14]:
```
# Call getPayloadData
getPayloadData(data)
```

In [15]:
```
# Call getCoreData
getCoreData(data)
```

In [16]:
```
# The lists has now been updated
BoosterVersion[0:5]
```

Out[16]:
```
['Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 9']
```

In [17]:
```
# 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}
```

In [18]:
```
# Create a DataFrame from launch_dict
launch_df = pd.DataFrame(launch_dict)
launch_df.head(3)
```

Out[18]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights
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

In [19]:
```
launch_df.shape
```

Out[19]:
```
(94, 17)
```

Task 2

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

In [20]:
```
# Quantify types of booster versions.
launch_df['BoosterVersion'].value_counts()
```

Out[20]:
```
Falcon 9    90
Falcon 1     4
Name: BoosterVersion, dtype: int64
```

In [21]:
```
# Exclude all launches except those with the Falcon 9 booster.
data_falcon_9 = launch_df.loc[launch_df['BoosterVersion'].isin(['Falcon 9'
data_falcon_9.head(2)
```

Out[21]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights
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

In [22]:
```
# Confirm that only the Falcon 9 booster is included.
data_falcon_9['BoosterVersion'].value_counts()
```

Out[22]:
```
Falcon 9    90
Name: BoosterVersion, dtype: int64
```

In [23]:
```
# Reset the FlightNumber column
data_falcon_9.loc[:,'FlightNumber'] = list(range(1, data_falcon_9.shape[0]
data_falcon_9.head(2)
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/pandas/cor
e/indexing.py:1773: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  self._setitem_single_column(ilocs[0], value, pi)
```

Out[23]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights
4	1	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1
5	2	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1

In [24]:
```
data_falcon_9.shape
```

Out[24]:
```
(90, 17)
```

In [25]:
```
data_falcon_9.describe()
```

Out[25]:

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Longitude	Latit
count	90.000000	85.000000	90.000000	90.000000	90.000000	90.000000	90.000
mean	45.500000	6123.547647	1.788889	3.500000	3.188889	-86.366477	29.449
std	26.124701	4870.916417	1.213172	1.595288	4.194417	14.149518	2.141
min	1.000000	350.000000	1.000000	1.000000	0.000000	-120.610829	28.561
25%	23.250000	2482.000000	1.000000	2.000000	0.000000	-80.603956	28.561
50%	45.500000	4535.000000	1.000000	4.000000	1.000000	-80.577366	28.561
75%	67.750000	9600.000000	2.000000	5.000000	4.000000	-80.577366	28.608
max	90.000000	15600.000000	6.000000	5.000000	13.000000	-80.577366	34.632

Data Wrangling

In [26]:
```
# There are some missing values in the dataset
data_falcon_9.isnull().sum()
```

Out[26]:
```
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
```

Task 3

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

used

In [27]:
```
# Calculate the mean value of the values in the PayloadMass column and rep
mean = data_falcon_9['PayloadMass'].mean()
data_falcon_9['PayloadMass'].replace(np.nan, mean, inplace=True)
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/pandas/cor
e/generic.py:6619: 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-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  return self._update_inplace(result)
```

In [28]:
```
# There are now no missing values for 'PayLoadMass'. We keep the 'None' va
data_falcon_9.isnull().sum()
```

Out[28]:
```
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
```

Export DataFrame to .CSV

Note
dataset_part_1.csv

In [29]:
```
# Export DataFrame as .csv
data_falcon_9.to_csv('dataset_part_1.csv', index=False)
```

End Here