SpaceX Falcon 9 first stage Landing Prediction

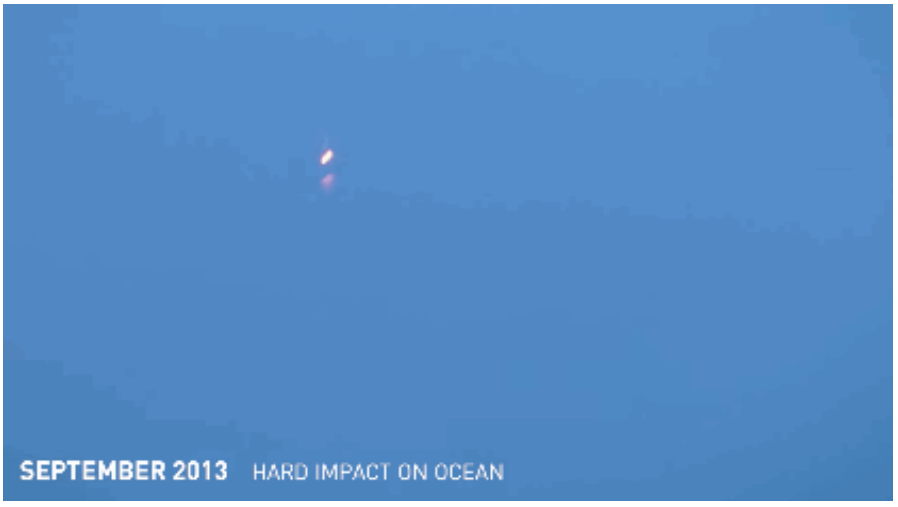
Lab 1: Collecting the data

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 [6]:
```
# Requests allows us to make HTTP
import requests
# Pandas is a software library wri
import pandas as pd
# NumPy is a library for the Pytho
import numpy as np
# Datetime is a library that allow
import datetime

# Setting this option will print o
pd.set_option('display.max_columns
# Setting this option will print o
pd.set_option('display.max_colwidt
```

In [9]:
```
#Below we will define a series of

#From the <code>rocket</code> colu
```

In [10]:
```
# Takes the dataset and uses the r
```

```
def getBoosterVersion(data):
    for x in data['rocket']:
        response = requests.get("h
        BoosterVersion.append(resp
```

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

In [11]:
```
# Takes the dataset and uses the l
def getLaunchSite(data):
    for x in data['launchpad']:
        response = requests.get("h
        Longitude.append(response[
        Latitude.append(response['
        LaunchSite.append(response
```

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

In [12]:
```
# Takes the dataset and uses the p
def getPayloadData(data):
    for load in data['payloads']:
        response = requests.get("h
```

```
        PayloadMass.append(respons
        Orbit.append(response['orb
```

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 [13]:
```
# Takes the dataset and uses the c
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != Nor
                response = request
                Block.append(respc
                ReusedCount.append
                Serial.append(resp
            else:
```

```
            Block.append(None)
            ReusedCount.append
            Serial.append(None
        Outcome.append(str(cor
        Flights.append(core['f
        GridFins.append(core['
        Reused.append(core['re
        Legs.append(core['legs
        LandingPad.append(core
```

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

In [14]:
```
spacex_url="https://api.spacexdata
```

In [15]:
```
response = requests.get(spacex_url
```

Check the content of the response

In [16]:
```
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","large":"https://images2.imgbox.com/40/e3/GypSkayF_o.png"},"reddit":{"campaign":null,"launch":null,"media":null,"recovery":null},"flickr":{"small":[],"original":[]},"presskit":null,"webcast":"https://www.youtube.com/watch?v=0a_00nJ_Y88","youtube_id":"0a_00nJ_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,"net":false,"window":0,"rocket":"5e9d0d95eda69955f709d1eb","success":false,"failures":[{"time":33,"altitude":null,"reason":"merlin engine failure"}],"details":"Engine failure at 33 seconds and loss of vehicle","crew":[],"ships":[],"capsules":[],"payloads":["5eb0e4b5b6c3bb0006eeb1e1"],"launchpad":"5e9e4502f5090995

In [37]:
```
# Get the head of the dataframe
data.head()
```

Out[37]:

	static_fire_date_utc	static_fire_date_un
0	2006-03-17T00:00:00.000Z	1.142554e+
1	None	Na

	static_fire_date_utc	static_fire_date_un
2	None	Na

	static_fire_date_utc	static_fire_date_un
3	2008-09-20T00:00:00.000Z	1.221869e+
4	None	Na

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 [38]:
```
# Lets take a subset of our datafr
data = data[['rocket', 'payloads',

# We will remove rows with multipl
data = data[data['cores'].map(len)
data = data[data['payloads'].map(l

# Since payloads and cores are lis
data['cores'] = data['cores'].map(
data['payloads'] = data['payloads'
```

```
# We also want to convert the date
data['date'] = pd.to_datetime(data

# Using the date we will restrict
data = data[data['date'] <= dateti
```

- 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 [39]:
```
#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 [40]:
```
BoosterVersion
```

Out[40]:
```
[]
```

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

In [41]:
```
# Call getBoosterVersion
getBoosterVersion(data)
```

the list has now been update

In [42]:
```
BoosterVersion[0:5]
```

Out[42]:
```
['Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 9']
```

we can apply the rest of the functions here:

In [43]:
```
# Call getLaunchSite
getLaunchSite(data)
```

In [44]:
```
# Call getPayloadData
getPayloadData(data)
```

In [45]:
```
# Call getCoreData
getCoreData(data)
```

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

In [53]:
```
launch_dict = {
```

```
'FlightNumber': list(data['flight_
'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 [54]:
```
# Create a data from launch_dict
df = pd.DataFrame.from_dict(launch
```

Show the summary of the dataframe

In [55]:
```
# Show the head of the dataframe
df.head()
```

Out[55]:

	FlightNumber	Date	BoosterVersion
0	1	2006-03-24	Falcon 1
1	2	2007-03-21	Falcon 1
2	4	2008-09-28	Falcon 1
3	5	2009-07-13	Falcon 1
4	6	2010-06-04	Falcon 9

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 [56]:
```
# Hint data['BoosterVersion']!='Fc
data_falcon9 = df[df['BoosterVersi
```

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

In [ ]:
```
data_falcon9.loc[:,'FlightNumber']
data_falcon9
```

Data Wrangling

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

dataset.

In [57]:
```
data_falcon9.isnull().sum()
```

Out[57]:
```
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 [59]:
```
# Calculate the mean value of Payl
payloadmassavg = data_falcon9['Pay
# Replace the np.nan values with i
data_falcon9['PayloadMass'].replac

# Replace the np.nan values with i
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/pandas/core/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-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return self._update_inplace(result)
```

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_pa
 index=False)
```

In [60]:
```
data_falcon9.isnull().sum()
```

Out[60]:
```
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
```

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

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