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 [34]:
```
# Requests allows us to make HTTP requests which we will use to get data fro
import requests
# Pandas is a software library written for the Python programming language f
import pandas as pd
# NumPy is a library for the Python programming language, adding support for
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 [35]:
```
# Takes the dataset and uses the rocket column to call the API and append th
def getBoosterVersion(data):
    for x in data['rocket']:
       if x:
        response = requests.get("https://api.spacexdata.com/v4/rockets/"+str
        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 [38]:
```
# Takes the dataset and uses the launchpad column to call the API and append
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 [39]:
```
# Takes the dataset and uses the payloads column to call the API and append
def getPayloadData(data):
    for load in data['payloads']:
       if load:
        response = requests.get("https://api.spacexdata.com/v4/payloads/"+lo
        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 [40]:
```
# Takes the dataset and uses the cores column to call the API and append the
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cores
                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['landir
            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 [41]:
```
spacex_url="https://api.spacexdata.com/v4/launches/past"
```

In [42]:
```
response = requests.get(spacex_url)
```

Check the content of the response

In [43]:
```
#print(response.content)
```

You should see the response contains massive information about SpaceX launches. Next, let's try to discover some more relevant information for this project.

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

To make the requested JSON results more consistent, we will use the following static response object for this project:

In [44]:
```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomai
```

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

In [45]:
```
response=requests.get(static_json_url)
```

In [46]:
```
response.status_code
```

Out[46]:
```
200
```

In [16]:
```
response_json = response.json()
#print(response_json)
```

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

In [17]:
```
# Use json_normalize meethod to convert the json result into a dataframe
import pandas as pd

data = pd.json_normalize(response_json)
```

Using the dataframe `data` print the first 5 rows

In [51]:
```
# Get the head of the dataframe
#print(data.head())
```

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 [52]:
```
# Lets take a subset of our dataframe keeping only the features we want and
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'c

# We will remove rows with multiple cores because those are falcon rockets w
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 sing
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 extra
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 [24]:
```
#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 [56]:
```
#BoosterVersion
```

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

In [57]:
```
# Call getBoosterVersion
getBoosterVersion(data)
```

the list has now been update

In [58]:
```
BoosterVersion[0:5]
```

Out[58]:
```
['Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 9']
```

we can apply the rest of the functions here:

In [59]:
```
# Call getLaunchSite
getLaunchSite(data)
```

In [60]:
```
# Call getPayloadData
getPayloadData(data)
```

In [61]:
```
# Call getCoreData
getCoreData(data)
```

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

In [31]:
```
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 [66]:
```
# Create a data from launch_dict
launch_df = pd.DataFrame(launch_dict)   # not data.DataFrame(...)
```

Show the summary of the dataframe

In [67]:
```
#Show the head of the dataframe
launch_df.head()
```

Out[67]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	I
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	

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 [68]:
```
# Hint data['BoosterVersion']!='Falcon 1'
# starting from the table you just built
data_falcon9 = launch_df[launch_df["BoosterVersion"] == "Falcon 9"].copy()
```

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

In [69]:
```
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9
```

Out[69]:

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

90 rows × 17 columns

Data Wrangling

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

In [70]:
```
data_falcon9.isnull().sum()
```

Out[70]:
```
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 [73]:
```
# Calculate the mean value of PayloadMass column

# Replace the np.nan values with its mean value

# Ensure numeric dtype first (sometimes comes as object/str)
data_falcon9["PayloadMass"] = pd.to_numeric(data_falcon9["PayloadMass"], err

# 1) compute the mean (ignoring NaNs)
pm_mean = data_falcon9["PayloadMass"].mean(skipna=True)
pm_mean

# 2) replace NaNs with the mean
data_falcon9["PayloadMass"] = data_falcon9["PayloadMass"].fillna(pm_mean)

# 3) verify the fix
data_falcon9.isnull().sum()[["PayloadMass","LandingPad"]]
# expect: PayloadMass -> 0, LandingPad -> 26 (unchanged)
```

Out[73]:
```
PayloadMass     0
LandingPad     26
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)
```

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.