Skills Network

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 [1]:
```
# Requests allows us to make HTTP requests which we will use to get data from an AP
import requests
# Pandas is a software library written for the Python programming language for data
import pandas as pd
# NumPy is a library for the Python programming language, adding support for large,
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 [2]:

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

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

```
# Takes the dataset and uses the payloads column to call the API and append the dat
def getPayloadData(data):
    for load in data['payloads']:
       if load:
        response = requests.get("https://api.spacexdata.com/v4/payloads/"+load).jso
        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]:

```
# Takes the dataset and uses the cores column to call the API and append the data t
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cores/"+core
                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 [6]:
```
spacex_url="https://api.spacexdata.com/v4/launches/past"
```

In [7]:
```
response = requests.get(spacex_url)
```

Check the content of the response

In [29]:
```
#uncomment to print out the content
#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 [8]:
```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud
```

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

In [9]:
```
response.status_code
responses=requests.get(static_json_url)
#responses.json()
```

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

In [10]:
```
# Use json_normalize meethod to convert the json result into a dataframe
data=pd.json_normalize(responses.json())
```

Using the dataframe `data` print the first 5 rows

In [11]:
```
# Get the head of the dataframe
data.head()
```

Out[11]:

	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
1	None	NaN	False	False	0.0	5e9d0d95eda69955f709d1el
2	None	NaN	False	False	0.0	5e9d0d95eda69955f709d1el
3	2008-09-20T00:00:00.000Z	1.221869e+09	False	False	0.0	5e9d0d95eda69955f709d1el

	static_fire_date_utc	static_fire_date_unix	tbd	net	window	rocke
4	None	NaN	False	False	0.0	5e9d0d95eda69955f709d1el

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 [12]:
```
# Lets take a subset of our dataframe keeping only the features we want and the fli
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc

# We will remove rows with multiple cores because those are falcon rockets with 2 e
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 valu
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 t
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 [13]:

```
#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 [14]:

```
BoosterVersion
```

Out[14]:

```
[]
```

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

In [15]:

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

the list has now been update

In [16]:

```
BoosterVersion[0:5]
```

Out[16]:

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

we can apply the rest of the functions here:

In [17]:

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

In [18]:

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

In [19]:

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

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

In [20]:

```
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 [21]:

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

Show the summary of the dataframe

In [22]:

```
# Show the head of the dataframe
print(df.info())
print(df.describe(include='all'))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 94 entries, 0 to 93
Data columns (total 17 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   FlightNumber    94 non-null     int64  
 1   Date            94 non-null     object 
 2   BoosterVersion  94 non-null     object 
 3   PayloadMass     88 non-null     float64
 4   Orbit           94 non-null     object 
 5   LaunchSite      94 non-null     object 
 6   Outcome         94 non-null     object 
 7   Flights         94 non-null     int64  
 8   GridFins        94 non-null     bool   
 9   Reused          94 non-null     bool   
 10  Legs            94 non-null     bool   
 11  LandingPad      64 non-null     object 
 12  Block           90 non-null     float64
 13  ReusedCount     94 non-null     int64  
 14  Serial          94 non-null     object 
 15  Longitude       94 non-null     float64
 16  Latitude        94 non-null     float64
dtypes: bool(3), float64(4), int64(3), object(7)
memory usage: 10.7+ KB
None
        FlightNumber        Date BoosterVersion   PayloadMass Orbit  \
count      94.000000          94             94     88.000000    94   
unique           NaN          94              2           NaN    11   
top              NaN  2006-03-24       Falcon 9           NaN   GTO   
freq             NaN           1             90           NaN    27   
mean       54.202128         NaN            NaN   5919.165341   NaN   
std        30.589048         NaN            NaN   4909.689575   NaN   
min         1.000000         NaN            NaN     20.000000   NaN   
25%        28.250000         NaN            NaN   2406.250000   NaN   
50%        52.500000         NaN            NaN   4414.000000   NaN   
75%        81.500000         NaN            NaN   9543.750000   NaN   
max       106.000000         NaN            NaN  15600.000000   NaN   

          LaunchSite    Outcome    Flights GridFins Reused  Legs  \
count             94         94  94.000000       94     94    94   
unique             4          8        NaN        2      2     2   
top     CCSFS SLC 40  True ASDS        NaN     True  False  True   
freq              55         41        NaN       70     57    71   
mean             NaN        NaN   1.755319      NaN    NaN   NaN   
std              NaN        NaN   1.197544      NaN    NaN   NaN   
min              NaN        NaN   1.000000      NaN    NaN   NaN   
25%              NaN        NaN   1.000000      NaN    NaN   NaN   
50%              NaN        NaN   1.000000      NaN    NaN   NaN   
75%              NaN        NaN   2.000000      NaN    NaN   NaN   
max              NaN        NaN   6.000000      NaN    NaN   NaN   

                      LandingPad      Block  ReusedCount Serial   Longitude  \
count                         64  90.000000    94.000000     94   94.000000   
unique                         5        NaN          NaN     57         NaN   
top     5e9e3032383ecb6bb234e7ca        NaN          NaN  B1049         NaN   
freq                          35        NaN          NaN      6         NaN   
```

```
mean                          NaN   3.500000     3.053191    NaN  -75.553302
std                           NaN   1.595288     4.153938    NaN   53.391880
min                           NaN   1.000000     0.000000    NaN -120.610829
25%                           NaN   2.000000     0.000000    NaN  -80.603956
50%                           NaN   4.000000     1.000000    NaN  -80.577366
75%                           NaN   5.000000     4.000000    NaN  -80.577366
max                           NaN   5.000000    13.000000    NaN  167.743129

         Latitude
count   94.000000
unique        NaN
top           NaN
freq          NaN
mean    28.581782
std      4.639981
min      9.047721
25%     28.561857
50%     28.561857
75%     28.608058
max     34.632093
```

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 [23]:

```python
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9=df[df['BoosterVersion'].str.contains('Falcon 9')]
```

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

In [24]:

```python
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9.to_csv('dataset_part_trial.csv', index=False)
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/pandas/core/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-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
```

In [26]:

```python
!pip install graphviz
from graphviz import Digraph

# Create a new directed graph
dot = Digraph(comment='SpaceX Data Collection Flowchart')
dot.attr(rankdir='TB')

# Add nodes
```

```
dot.node('A', 'Import Libraries')
dot.node('B', 'Define Auxiliary Functions')
dot.node('C', 'Make API Request')
dot.node('D', 'Parse JSON Response')
dot.node('E', 'Extract Specific Data')
dot.node('F', 'Construct Dataset')

dot.node('G', 'getBoosterVersion')
dot.node('H', 'getLaunchSite')
dot.node('I', 'getPayloadData')
dot.node('J', 'getCoreData')

# Add edges
dot.edge('A', 'B')
dot.edge('B', 'C')
dot.edge('C', 'D')
dot.edge('D', 'E')
dot.edge('E', 'F')

dot.edge('B', 'G')
dot.edge('B', 'H')
dot.edge('B', 'I')
dot.edge('B', 'J')

dot.edge('G', 'E')
dot.edge('H', 'E')
dot.edge('I', 'E')
dot.edge('J', 'E')

# Render the graph
dot.render('spacex_data_collection_flowchart', format='png', cleanup=True)
print("Flowchart has been created as 'spacex_data_collection_flowchart.png'")
```

```
Collecting graphviz
  Downloading graphviz-0.20.1-py3-none-any.whl (47 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 47.0/47.0 kB 8.4 MB/s eta 0:00:00
Installing collected packages: graphviz
Successfully installed graphviz-0.20.1
Flowchart has been created as 'spacex_data_collection_flowchart.png'
```

Data Wrangling

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

In [29]:

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

```
Out[29]: 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 [36]: # Calculate the mean value of PayloadMass column
         payload_mass_mean=data_falcon9['PayloadMass'].mean()
         payload_mass_mean
         # Replace the np.nan values with its mean value
         data_falcon9['PayloadMass'] = data_falcon9['PayloadMass'].fillna(payload_mass_mean)
         data_falcon9.isnull().sum()
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/ipykernel_launcher.p
y:5: 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-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  """
```

Out[36]:

```
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
```

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.

In [38]:

```
#<code>
data_falcon9.to_csv('dataset_part_1.csv', index=False)
#</code>
```

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.