jupyter-labs-spacex-data-collection-api

June 15, 2022

1 SpaceX Falcon 9 first stage Landing Prediction

2 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.

2.1 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

2.2 Import Libraries and Define Auxiliary Functions

We will import the following libraries into the lab

[2]:
```
# 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 manipulation and analysis.
import pandas as pd
# NumPy is a library for the Python programming language, adding support for␣
 ↪large, multi-dimensional arrays and matrices, along with a large collection␣
 ↪of high-level mathematical functions to operate on these arrays
```

```
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.

[3]:
```
# Takes the dataset and uses the rocket column to call the API and append the␣
 ↪data to the list
def getBoosterVersion(data):
    for x in data['rocket']:
        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.

[4]:
```
# Takes the dataset and uses the launchpad column to call the API and append␣
 ↪the data to the list
def getLaunchSite(data):
    for x in data['launchpad']:
        response = requests.get("https://api.spacexdata.com/v4/launchpads/
 ↪"+str(x)).json()
        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.

[5]:
```
# Takes the dataset and uses the payloads column to call the API and append the␣
 ↪data to the lists
def getPayloadData(data):
    for load in data['payloads']:
        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.

[6]:
```
# Takes the dataset and uses the cores column to call the API and append the␣
 ↪data to the lists
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cores/
 ↪"+core['core']).json()
                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:

[7]:
```
spacex_url="https://api.spacexdata.com/v4/launches/past"
```

[8]:
```
response = requests.get(spacex_url)
```

Check the content of the response

[ ]:
```
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.

2.2.1 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:

[10]:
```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.
 ↪cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_call_spacex_api.json'
```

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

[11]:
```
response.status_code
```

[11]: 200

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

[12]:
```
# Use json_normalize meethod to convert the json result into a dataframe
response2 = requests.get(static_json_url)
data = pd.json_normalize(response2.json())
```

Using the dataframe data print the first 5 rows

[13]:
```
# Get the head of the dataframe
data.head()
```

[13]:

	static_fire_date_utc	static_fire_date_unix	tbd	net	window	rocket	success	details	crew	ships	capsules	payloads
0	2006-03-17T00:00:00.000Z	1.142554e+09	False	False	0.0	5e9d0d95eda69955f709d1eb	False	Engine failure at 33 seconds and loss of vehicle	[]	[]	[]	[5eb0e4b5b6c3bb0006eeb1e1]
1	None	NaN	False	False	0.0	5e9d0d95eda69955f709d1eb	False	Successful first stage burn and transition to second stage, maximum altitude 289 km, Premature engine shutdown at T+7 min 30 s, Failed to reach orbit, Failed to recover first stage	[]	[]	[]	[5eb0e4b6b6c3bb0006eeb1e2]
2	None	NaN	False	False	0.0	5e9d0d95eda69955f709d1eb	False	Residual stage 1 thrust led to collision between stage 1 and stage 2	[]	[]	[]	[5eb0e4b6b6c3bb0006eeb1e3, 5eb0e4b6b6c3bb0006eeb1e4]
3	2008-09-20T00:00:00.000Z	1.221869e+09	False	False	0.0	5e9d0d95eda69955f709d1eb	True	Ratsat was carried to orbit on the first successful orbital launch of any privately funded and developed, liquid-propelled carrier rocket, the SpaceX Falcon 1	[]	[]	[]	[5eb0e4b7b6c3bb0006eeb1e5]
4	None	NaN	False	False	0.0	5e9d0d95eda69955f709d1eb	True	None	[]	[]	[]	[5eb0e4b7b6c3bb0006eeb1e6]

```
                  launchpad  auto_update  \
0  5e9e4502f5090995de566f86         True
1  5e9e4502f5090995de566f86         True
2  5e9e4502f5090995de566f86         True
3  5e9e4502f5090995de566f86         True
4  5e9e4502f5090995de566f86         True

                           failures  \
0                                                [{'time': 33, 'altitude': None,
'reason': 'merlin engine failure'}]
1            [{'time': 301, 'altitude': 289, 'reason': 'harmonic oscillation
leading to premature engine shutdown'}]
2  [{'time': 140, 'altitude': 35, 'reason': 'residual stage-1 thrust led to
collision between stage 1 and stage 2'}]
3
[]
4
[]

   flight_number         name                  date_utc   date_unix  \
0              1    FalconSat  2006-03-24T22:30:00.000Z  1143239400
1              2      DemoSat  2007-03-21T01:10:00.000Z  1174439400
2              3  Trailblazer  2008-08-03T03:34:00.000Z  1217734440
3              4       RatSat  2008-09-28T23:15:00.000Z  1222643700
4              5     RazakSat  2009-07-13T03:35:00.000Z  1247456100

                  date_local date_precision  upcoming  \
0  2006-03-25T10:30:00+12:00           hour     False
1  2007-03-21T13:10:00+12:00           hour     False
2  2008-08-03T15:34:00+12:00           hour     False
3  2008-09-28T11:15:00+12:00           hour     False
4  2009-07-13T15:35:00+12:00           hour     False

                              cores  \
0  [{'core': '5e9e289df35918033d3b2623', 'flight': 1, 'gridfins': False, 'legs':
False, 'reused': False, 'landing_attempt': False, 'landing_success': None,
'landing_type': None, 'landpad': None}]
1  [{'core': '5e9e289ef35918416a3b2624', 'flight': 1, 'gridfins': False, 'legs':
False, 'reused': False, 'landing_attempt': False, 'landing_success': None,
'landing_type': None, 'landpad': None}]
2  [{'core': '5e9e289ef3591814873b2625', 'flight': 1, 'gridfins': False, 'legs':
False, 'reused': False, 'landing_attempt': False, 'landing_success': None,
'landing_type': None, 'landpad': None}]
3  [{'core': '5e9e289ef3591855dc3b2626', 'flight': 1, 'gridfins': False, 'legs':
False, 'reused': False, 'landing_attempt': False, 'landing_success': None,
'landing_type': None, 'landpad': None}]
```

```
4  [{'core': '5e9e289ef359184f103b2627', 'flight': 1, 'gridfins': False, 'legs':
False, 'reused': False, 'landing_attempt': False, 'landing_success': None,
'landing_type': None, 'landpad': None}]
```

```
                        id fairings.reused fairings.recovery_attempt  \
0  5eb87cd9ffd86e000604b32a          False                     False
1  5eb87cdaffd86e000604b32b          False                     False
2  5eb87cdbffd86e000604b32c          False                     False
3  5eb87cdbffd86e000604b32d          False                     False
4  5eb87cdcffd86e000604b32e          False                     False

  fairings.recovered fairings.ships  \
0              False             []
1              False             []
2              False             []
3              False             []
4              False             []

                                links.patch.small  \
0  https://images2.imgbox.com/3c/0e/T8iJcSN3_o.png
1  https://images2.imgbox.com/4f/e3/I0lkuJ2e_o.png
2  https://images2.imgbox.com/3d/86/cnu0pan8_o.png
3  https://images2.imgbox.com/e9/c9/T8CfiSYb_o.png
4  https://images2.imgbox.com/a7/ba/NBZSw3Ho_o.png

                                links.patch.large links.reddit.campaign  \
0  https://images2.imgbox.com/40/e3/GypSkayF_o.png                  None
1  https://images2.imgbox.com/be/e7/iNqsqVYM_o.png                  None
2  https://images2.imgbox.com/4b/bd/d8UxLh4q_o.png                  None
3  https://images2.imgbox.com/e0/a7/FNjvKlXW_o.png                  None
4  https://images2.imgbox.com/8d/fc/0qdZMWWx_o.png                  None

  links.reddit.launch links.reddit.media links.reddit.recovery  \
0                None               None                  None
1                None               None                  None
2                None               None                  None
3                None               None                  None
4                None               None                  None

  links.flickr.small links.flickr.original  \
0                 []                    []
1                 []                    []
2                 []                    []
3                 []                    []
4                 []                    []

           links.presskit  \
```

```
0
None
1
None
2
None
3
None
4  http://www.spacex.com/press/2012/12/19/spacexs-falcon-1-successfully-
delivers-razaksat-satellite-orbit

                                 links.webcast links.youtube_id  \
0  https://www.youtube.com/watch?v=0a_00nJ_Y88      0a_00nJ_Y88
1  https://www.youtube.com/watch?v=Lk4zQ2wP-Nc      Lk4zQ2wP-Nc
2  https://www.youtube.com/watch?v=v0w9p3U8860      v0w9p3U8860
3  https://www.youtube.com/watch?v=dLQ2tZEH6G0      dLQ2tZEH6G0
4  https://www.youtube.com/watch?v=yTaIDooc8Og      yTaIDooc8Og

                                                                links.article
\
0  https://www.space.com/2196-spacex-inaugural-falcon-1-rocket-lost-launch.html
1      https://www.space.com/3590-spacex-falcon-1-rocket-fails-reach-orbit.html
2       http://www.spacex.com/news/2013/02/11/falcon-1-flight-3-mission-summary
3                                          https://en.wikipedia.org/wiki/Ratsat
4                       http://www.spacex.com/news/2013/02/12/falcon-1-flight-5

                                       links.wikipedia  fairings
0                  https://en.wikipedia.org/wiki/DemoSat       NaN
1                  https://en.wikipedia.org/wiki/DemoSat       NaN
2  https://en.wikipedia.org/wiki/Trailblazer_(satellite)       NaN
3                   https://en.wikipedia.org/wiki/Ratsat       NaN
4                 https://en.wikipedia.org/wiki/RazakSAT       NaN
```

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.

[14]:
```
# Lets take a subset of our dataframe keeping only the features we want and the␣
 ↪flight number, and date_utc.
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 rocket boosters and rows that have multiple payloads in a single␣
 ↪rocket.
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 the list and replace the feature.
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 date leaving the time
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.

[15]:
```
#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:

[16]:
```
BoosterVersion
```

[16]:
```
[]
```

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

[17]:
```
# Call getBoosterVersion
getBoosterVersion(data)
```

the list has now been update

[18]:
```
BoosterVersion[0:5]
```

[18]:
```
['Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 9']
```

we can apply the rest of the functions here:

[19]:
```
# Call getLaunchSite
getLaunchSite(data)
```

[20]:
```
# Call getPayloadData
getPayloadData(data)
```

[21]:
```
# Call getCoreData
getCoreData(data)
```

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

[22]:
```
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.

[24]:
```
# Create a data from launch_dict
df = pd.DataFrame(launch_dict)
```

Show the summary of the dataframe

[25]:
```
# Show the head of the dataframe
df.head()
```

[25]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude
0	1	2006-03-24	Falcon 1	20.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0	Merlin1A	167.743129	9.047721
1	2	2007-03-21	Falcon 1	NaN	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0	Merlin2A	167.743129	9.047721
2	4	2008-09-28	Falcon 1	165.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0	Merlin2C	167.743129	9.047721
3	5	2009-07-13	Falcon 1	200.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0	Merlin3C	167.743129	9.047721
4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0003	-80.577366	28.561857

2.2.2 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.

[27]:
```
# Hint data['BoosterVersion']!='Falcon 1'
df = df[ df['BoosterVersion'] =='Falcon 9']
```

[28]:
```
df.head()
```

[28]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block
4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0
5	8	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0
6	10	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False	False	False	None	1.0
7	11	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None	1.0
8	12	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1.0

```
4    None None        1     False   False  False       None    1.0
5    None None        1     False   False  False       None    1.0
6    None None        1     False   False  False       None    1.0
7  False Ocean        1     False   False  False       None    1.0
8    None None        1     False   False  False       None    1.0

   ReusedCount Serial   Longitude   Latitude
4            0  B0003  -80.577366  28.561857
5            0  B0005  -80.577366  28.561857
6            0  B0007  -80.577366  28.561857
7            0  B1003 -120.610829  34.632093
8            0  B1004  -80.577366  28.561857
```

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

[29]:
```
data_falcon9 = df
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9
```

[29]:
```
    FlightNumber        Date BoosterVersion  PayloadMass Orbit    LaunchSite  \
4              1  2010-06-04       Falcon 9          NaN   LEO  CCSFS SLC 40
5              2  2012-05-22       Falcon 9        525.0   LEO  CCSFS SLC 40
6              3  2013-03-01       Falcon 9        677.0   ISS  CCSFS SLC 40
7              4  2013-09-29       Falcon 9        500.0    PO   VAFB SLC 4E
8              5  2013-12-03       Falcon 9       3170.0   GTO  CCSFS SLC 40
..           ...         ...            ...          ...   ...           ...
89            86  2020-09-03       Falcon 9      15600.0  VLEO    KSC LC 39A
90            87  2020-10-06       Falcon 9      15600.0  VLEO    KSC LC 39A
91            88  2020-10-18       Falcon 9      15600.0  VLEO    KSC LC 39A
92            89  2020-10-24       Falcon 9      15600.0  VLEO  CCSFS SLC 40
93            90  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
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           11  B1060  -80.603956  28.608058
90    5.0           11  B1058  -80.603956  28.608058
91    5.0           11  B1051  -80.603956  28.608058
92    5.0           11  B1060  -80.577366  28.561857
93    5.0            6  B1062  -80.577366  28.561857

[90 rows x 17 columns]
```

2.3 Data Wrangling

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

[30]:
```
data_falcon9.isnull().sum()
```

[30]:
```
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.

2.3.1 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.

[32]:
```
# Calculate the mean value of PayloadMass column
data_falcon9['PayloadMass'].mean()
```

```
# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'].fillna(data_falcon9['PayloadMass'].mean(), inplace
 ↪= True)
```

[33]:
```
data_falcon9.isnull().sum()
```

[33]:
```
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.

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

2.4 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.

2.5 Change Log

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

Copyright © 2021 IBM Corporation. All rights reserved.