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

```python
# Requests allows us to make HTTP requests which we will use to get data from an
import requests
# Pandas is a software library written for the Python programming language for d
import pandas as pd
```

```python
# NumPy is a library for the Python programming language, adding support for lar
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]:
```python
# Takes the dataset and uses the rocket column to call the API and append the da
def getBoosterVersion(data):
    for x in data['rocket']:
        if x:
            response = requests.get("https://api.spacexdata.com/v4/rockets/"+str(x))
            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]:
```python
# Takes the dataset and uses the launchpad column to call the API and append the
def getLaunchSite(data):
    for x in data['launchpad']:
        if x:
            response = requests.get("https://api.spacexdata.com/v4/launchpads/"+str
            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]:
```python
# Takes the dataset and uses the payloads column to call the API and append the
def getPayloadData(data):
    for load in data['payloads']:
        if load:
            response = requests.get("https://api.spacexdata.com/v4/payloads/"+load).
            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]:
```python
# Takes the dataset and uses the cores column to call the API and append the dat
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
```

```
            response = requests.get("https://api.spacexdata.com/v4/cores/"+c
            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_ty
        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]:
```python
spacex_url="https://api.spacexdata.com/v4/launches/past"
```

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

Check the content of the response

In [8]:
```python
response
```

Out[8]:
```
<Response [200]>
```

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 [9]:
```python
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cl
```

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

In [10]:
```python
response.status_code
```

Out[10]:
```
200
```

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

In [14]:
```python
# Use json_normalize meethod to convert the json result into a dataframe

data = pd.json_normalize(response.json())
```

Using the dataframe `data` print the first 5 rows

In [15]:
```python
# Get the head of the dataframe

data.head()
```

Out[15]:

	static_fire_date_utc	static_fire_date_unix	net	window	rocket
0	2006-03-17T00:00:00.000Z	1.142554e+09	False	0.0	5e9d0d95eda69955f709d1eb
1	None	NaN	False	0.0	5e9d0d95eda69955f709d1eb
2	None	NaN	False	0.0	5e9d0d95eda69955f709d1eb
3	2008-09-20T00:00:00.000Z	1.221869e+09	False	0.0	5e9d0d95eda69955f709d1eb

static_fire_date_utc	static_fire_date_unix	net	window	rocket

4	None	NaN	False	0.0	5e9d0d95eda69955f709d1eb

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 [16]:
```
# Lets take a subset of our dataframe keeping only the features we want and the
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_

# We will remove rows with multiple cores because those are falcon rockets with
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 v
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 extractin
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 [17]:   #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 [18]:   BoosterVersion
```

```
Out[18]:   []
```

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

```
In [19]:   # Call getBoosterVersion
           getBoosterVersion(data)
```

the list has now been update

```
In [20]:   BoosterVersion[0:5]
```

```
Out[20]:   ['Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 9']
```

we can apply the rest of the functions here:

```
In [21]:   # Call getLaunchSite
           getLaunchSite(data)
```

```
In [22]:   # Call getPayloadData
           getPayloadData(data)
```

```
In [23]:   # Call getCoreData
           getCoreData(data)
```

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

```
In [24]:   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 [25]:
```python
# Create a data from launch_dict
max_len = max(len(lst) for lst in launch_dict.values())

for key, value in launch_dict.items():
    if len(value) < max_len:
        launch_dict[key] = value + [None] * (max_len - len(value))

df = pd.DataFrame(launch_dict)
```

Show the summary of the dataframe

In [26]:
```python
# Show the head of the dataframe

df.head()
```

Out[26]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Fli
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 [27]:
```python
# Hint data['BoosterVersion']!='Falcon 1'
```

```
data_falcon9=df.loc[df['BoosterVersion']!='Falcon 1']
print("Number of Falcon 9 launches:", data_falcon9.shape[0])
```

Number of Falcon 9 launches: 90

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

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

```
    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           12  B1060  -80.603956  28.608058
90    5.0           13  B1058  -80.603956  28.608058
91    5.0           12  B1051  -80.603956  28.608058
92    5.0           12  B1060  -80.577366  28.561857
93    5.0            8  B1062  -80.577366  28.561857

[90 rows x 17 columns]
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/pandas/core/indexi
ng.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/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
```

Data Wrangling

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

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

Out[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.

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 [31]:
```python
data_falcon9['PayloadMass'] = pd.to_numeric(data_falcon9['PayloadMass'], errors=

# Calculate the mean value of PayloadMass column
print(data_falcon9['PayloadMass'].mean())

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

```
6123.547647058824
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/ipykernel_launche
r.py:1: 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/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/pandas/core/generi
c.py:6392: 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/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  return self._update_inplace(result)
```

In [32]:
```python
data_falcon9.isnull().sum()
```

Out[32]:
```
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.

```python
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.

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

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