# SpaceX Falcon 9 first stage Landing Prediction

## Hands-on Lab: Complete the Data Collection API Lab

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

---

Install the below libraries

```
!pip install requests
!pip install pandas
!pip install numpy
```

```
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (2.32.4)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests) (2026.1.4)
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (2.2.2)
Requirement already satisfied: numpy>=1.26.0 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (2.0.2)
```

## Import Libraries and Define Auxiliary Functions

We will import the following libraries into the lab

```
# 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 mathemat
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.

```
# 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']:
       if x:
        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.

```
# 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']:
       if x:
         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.

```
# 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']:
       if load:
        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.

```
# 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:

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

```
response = requests.get(spacex_url)
```

Check the content of the response

```
print(response.content)
```

```
b'[{"fairings":{"reused":false,"recovery_attempt":false,"recovered":false,"ships":[]},"links":{"patch":{"small":"https://images2.imgbox.com/94/f2/NN6Ph45r_o.png","large":"https
```

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:

```
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

```
response.status_code
```

```
200
```

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

```
# Use json_normalize meethod to convert the json result into a dataframe
response = requests.get(static_json_url)
data = response.json()
data = pd.json_normalize(data)
```

Using the dataframe `data` print the first 5 rows

```
# Get the head of the dataframe
data.head()
```

| | 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] | 5e9e4502f509( |
| **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] | 5e9e4502f509( |
| **2** | None | NaN | False | False | 0.0 | 5e9d0d95eda69955f709d1eb | False | Residual stage 1 thrust led to collision between stage 1 and stage 2 | [] | [] | [] | [5eb0e4b6b6c3bb0006eeb1e3, 5eb0e4b6b6c3bb0006eeb1e4] | 5e9e4502f509( |
| **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] | 5e9e4502f509( |
| **4** | None | NaN | False | False | 0.0 | 5e9d0d95eda69955f709d1eb | True | None | [] | [] | [] | [5eb0e4b7b6c3bb0006eeb1e6] | 5e9e4502f509( |

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`.

```
# 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.

```
#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:

```
BoosterVersion
```

```
[]
```

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

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

the list has now been update

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

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

we can apply the rest of the functions here:

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

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

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

```
Legs[0:5]
```

```
[False, False, False, False, False]
```

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

```
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.

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

Show the summary of the dataframe

```
# Show the head of the dataframe
data.head()
```

| | 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 |

Next steps: Generate code with `data` New interactive sheet

## 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`.

```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9=data[data['BoosterVersion']==('Falcon 9')]
data_falcon9.head()
```

| | FlightNumber | Date | BoosterVersion | PayloadMass | Orbit | LaunchSite | Outcome | Flights | GridFins | Reused | Legs | LandingPad | Block | ReusedCount | Serial | Longitude | Latitude |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 |
| **5** | 8 | 2012-05-22 | Falcon 9 | 525.0 | LEO | CCSFS SLC 40 | None None | 1 | False | False | False | None | 1.0 | 0 | B0005 | -80.577366 | 28.561857 |
| **6** | 10 | 2013-03-01 | Falcon 9 | 677.0 | ISS | CCSFS SLC 40 | None None | 1 | False | False | False | None | 1.0 | 0 | B0007 | -80.577366 | 28.561857 |

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

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

| | FlightNumber | Date | BoosterVersion | PayloadMass | Orbit | LaunchSite | Outcome | Flights | GridFins | Reused | Legs | LandingPad | Block | ReusedCount | Serial | Longitu |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **4** | 1 | 2010-06-04 | Falcon 9 | NaN | LEO | CCSFS SLC 40 | None None | 1 | False | False | False | None | 1.0 | 0 | B0003 | -80.5773 |
| **5** | 2 | 2012-05-22 | Falcon 9 | 525.0 | LEO | CCSFS SLC 40 | None None | 1 | False | False | False | None | 1.0 | 0 | B0005 | -80.5773 |
| **6** | 3 | 2013-03-01 | Falcon 9 | 677.0 | ISS | CCSFS SLC 40 | None None | 1 | False | False | False | None | 1.0 | 0 | B0007 | -80.5773 |
| **7** | 4 | 2013-09-29 | Falcon 9 | 500.0 | PO | VAFB SLC 4E | False Ocean | 1 | False | False | False | None | 1.0 | 0 | B1003 | -120.6108 |
| **8** | 5 | 2013-12-03 | Falcon 9 | 3170.0 | GTO | CCSFS SLC 40 | None None | 1 | False | False | False | None | 1.0 | 0 | B1004 | -80.5773 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **89** | 86 | 2020-09-03 | Falcon 9 | 15600.0 | VLEO | KSC LC 39A | True ASDS | 2 | True | True | True | 5e9e3032383ecb6bb234e7ca | 5.0 | 12 | B1060 | -80.6039 |
| **90** | 87 | 2020-10-06 | Falcon 9 | 15600.0 | VLEO | KSC LC 39A | True ASDS | 3 | True | True | True | 5e9e3032383ecb6bb234e7ca | 5.0 | 13 | B1058 | -80.6039 |

## Data Wrangling

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

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

| | 0 |
|---|---|
| **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.

```
# Calculate the mean value of PayloadMass column
avg_PayloadMass=data_falcon9["PayloadMass"].astype("float").mean(axis=0)
# Replace the np.nan values with its mean value
data_falcon9["PayloadMass"].replace(np.nan,avg_PayloadMass,inplace=True)
```

```
/tmp/ipython-input-1917437548.py:4: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the oper


  data_falcon9["PayloadMass"].replace(np.nan,avg_PayloadMass,inplace=True)
/tmp/ipython-input-1917437548.py:4: 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
  data_falcon9["PayloadMass"].replace(np.nan,avg_PayloadMass,inplace=True)
```

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

| | 0 |
|---|---|
| **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)
```

## 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.

Copyright © 2021 IBM Corporation. All rights reserved.