Lab 5 - SpaceX Falcon 9 First Stage Landing Prediction

#Coursera #IBMDataScienceProfessionalCertificate #AppliedDataScienceCapstone

Blog: https://blog.stackademic.com/web-scraping-with-python-55de0118bcac

----

Lab 4: EDA with Visualization Lab

Assignment: Exploring and Preparing Data

In this assignment, we will predict if the Falcon 9 first stage will and successfully. SpaceX advertises Falcon 9 rocket launches on tis website with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is due to the fact that that SpaceX can reuse the first stage.

In this lab, you will perform Exploratory Data Analysis and Feature Engineering.

Most unsuccessful landings are planned. Space X performs a controlled landing in the oceans.

Objectives

Perform Exploratory Data Analysis and Feature Engineering using *Pandas* and *Matplotlib*.

- Exploratory Data Analysis
- Preparing Data Feature Engineering

Import Libraries and Define Auxiliary Functions

```
import piplite
await piplite.install(['numpy'])
await piplite.install(['pandas'])
await piplite.install(['seaborn'])

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

- Pandas: software library written for Python programming language for

data manipulation and analysis.
- Numpy: 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 those arrays
- Matplotlib: plotting library for Python and pyplot gives us a MatLab like plotting framework. We will use this in our plotter function to plot data.
- Seaborn: Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

```
## Exploratory Data Analysis
```

First, let's read the SpaceX dataset into a Pandas dataframe and print it summary

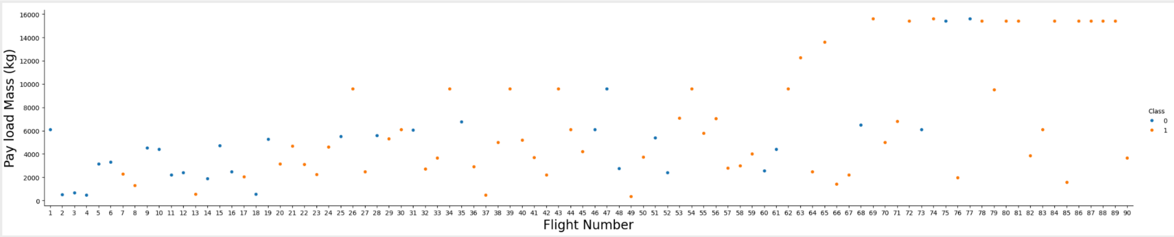
```
from js import fetch
import io

URL = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_2.csv"
resp = await fetch(URL)
dataset_part_2_csv = io.BytesIO((await resp.arrayBuffer()).to_py())
df=pd.read_csv(dataset_part_2_csv)
df.head(5)
```

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577366	28.561857	0
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366	28.561857	0
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.577366	28.561857	0
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	0	B1003	-120.610829	34.632093	0
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561857	0

First, let's try to see how the *FlightNumber* (indicating the continuous launch attempts) and *Payload* variables would affect the launch outcome.

We can plot out the *FlightNumber* vs *PayloadMass* and overlay the outcome of the launch. We see that as the flight number increases, the first stage is more likely to land successfully. The payload mass is also important; it seems the more massive the payload, the less likely the first stage will return.

```
sns.catplot(y = 'PayloadMass', x = 'FlightNumber', hue = 'Class', data = df, aspect = 5)
plt.xlabel("Flight Number",fontsize = 20)
plt.ylabel("Pay load Mass (kg)",fontsize = 20)
plt.show()
```



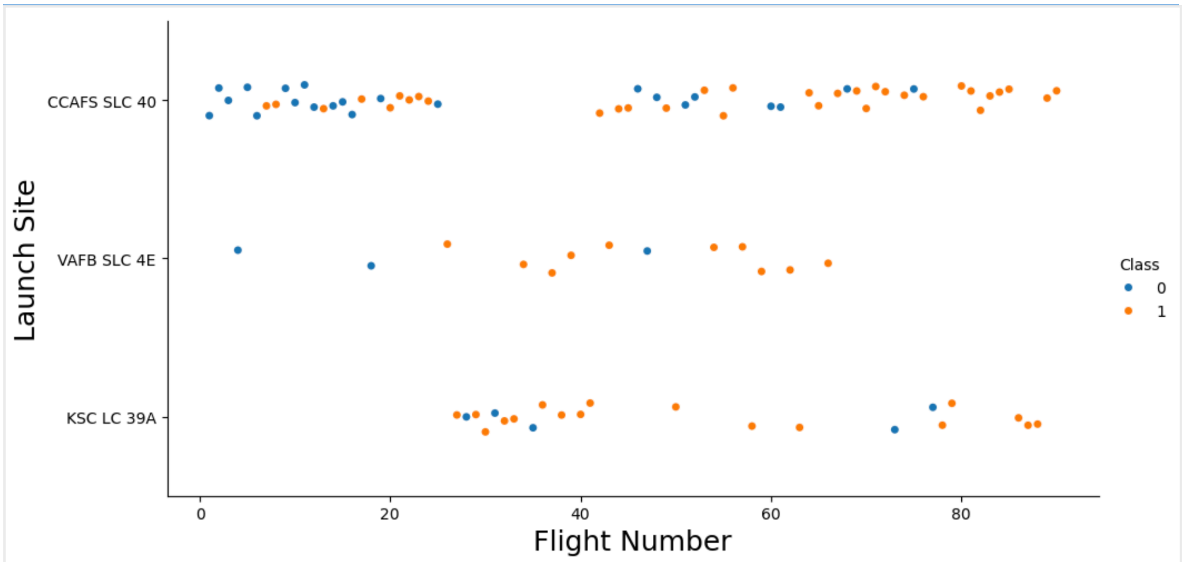
We see that the different launch sites have different success rates. *CCAFS LC-40* has a success rate of 60%, while *KSC LC-39A* and *VAFB SLC 4E* has a success rate of 77%.

Next, let's drill down to each site visualize its detailed launch records.

Task 1: Visualize the relationship between Flight Number and Launch Site

Use the function *caplot* to plot *FlightNumber* vs *LaunchSite*, set the parameter *x* to *FlightNumber*, set the *y* to *LaunchSite* and set the parameter *hue* to '*class*'

```
# Plot a scatter point chart with x axis to be FlightNumber
and y axis to be LaunchSite,
# and hue to be the class value
sns.catplot(y = 'LaunchSite',
            x = 'FlightNumber',
            hue = 'Class',
            aspect = 2,
            data = df)
plt.xlabel('Flight Number',
           fontsize = 18)
plt.ylabel('Launch Site',
           fontsize = 18)
plt.show()
```

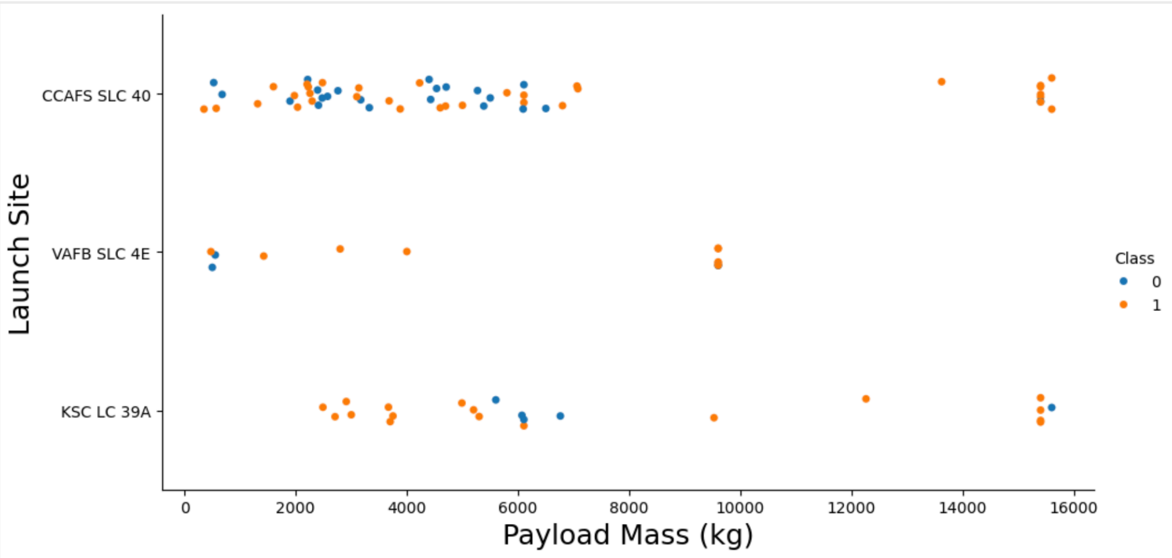


Now try to explain the patterns you found in the FlightNumber vs LaunchSite scatter point plots.

Task 2: Visualize the relationship between Payload and LaunchSite

We also want to observe if there is any relationship between launch sites and their payload mass.

```
# Plot a scatter point chart with x axis to be Pay Load Mass
(kg) and
# y axis to be the launch site, and hue to be the class
value
sns.catplot(y = 'LaunchSite',
            x = 'PayloadMass',
            hue = 'Class',
            aspect = 2,
            data = df)
plt.xlabel('Payload Mass (kg)',
           fontsize = 18)
plt.ylabel('Launch Site',
           fontsize = 18)
plt.show()
```

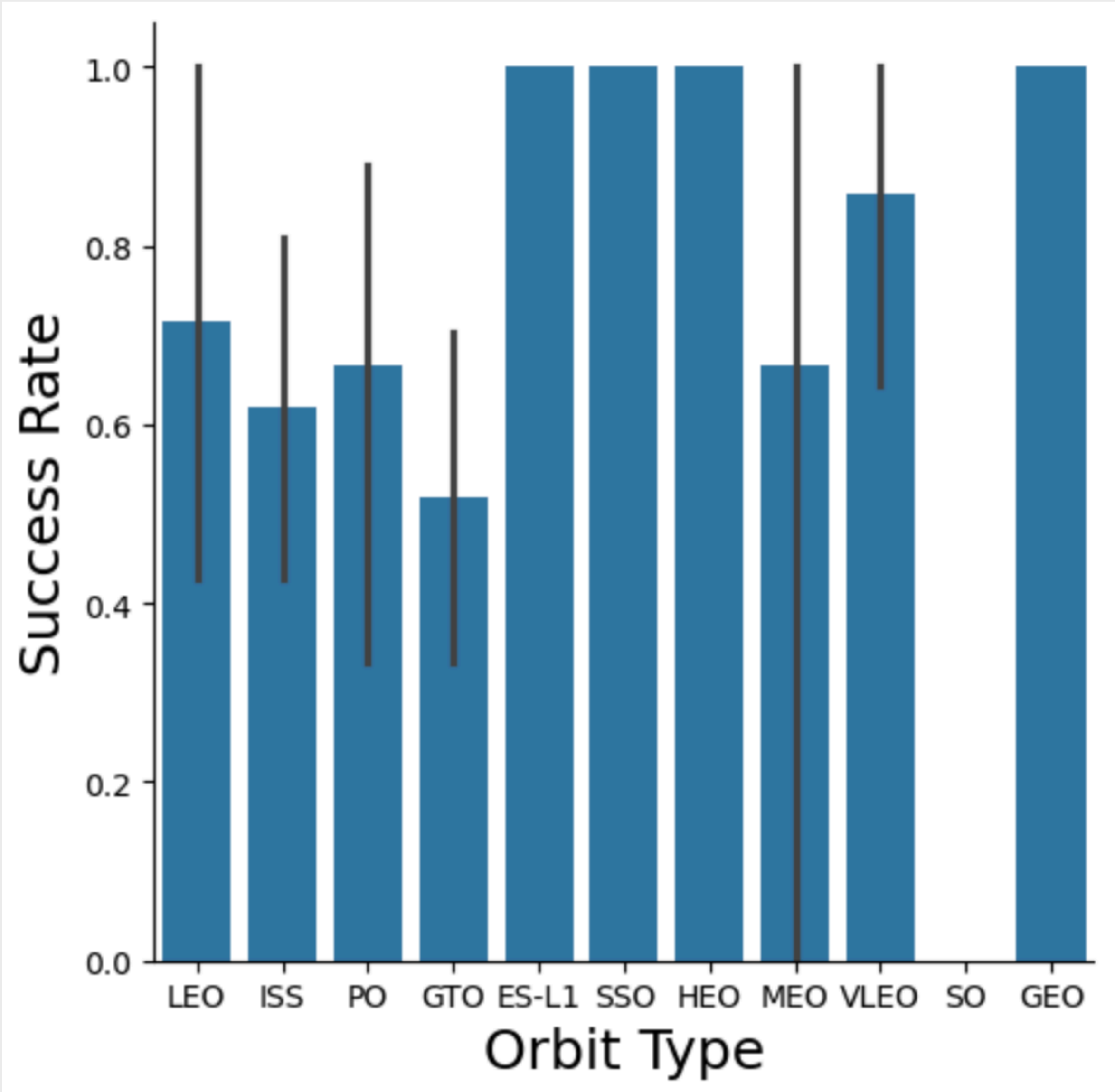


Now if you observer Payload vs Launch Site scatter point chart you will find for the VAFB-SLC launchsite there are no rockets launched for heavy payload mass (greater than 10000).

Task 3: Visualize the relationship between success rate of each orbit type

```
# HINT use groupby method on Orbit column and get the mean
```

```
of Class column
sns.catplot(x = 'Orbit',
            y = 'Class',
            data = df,
            kind = 'bar')
plt.xlabel('Orbit Type',
           fontsize = 18)
plt.ylabel('Success Rate',
           fontsize = 18)
plt.show()
```

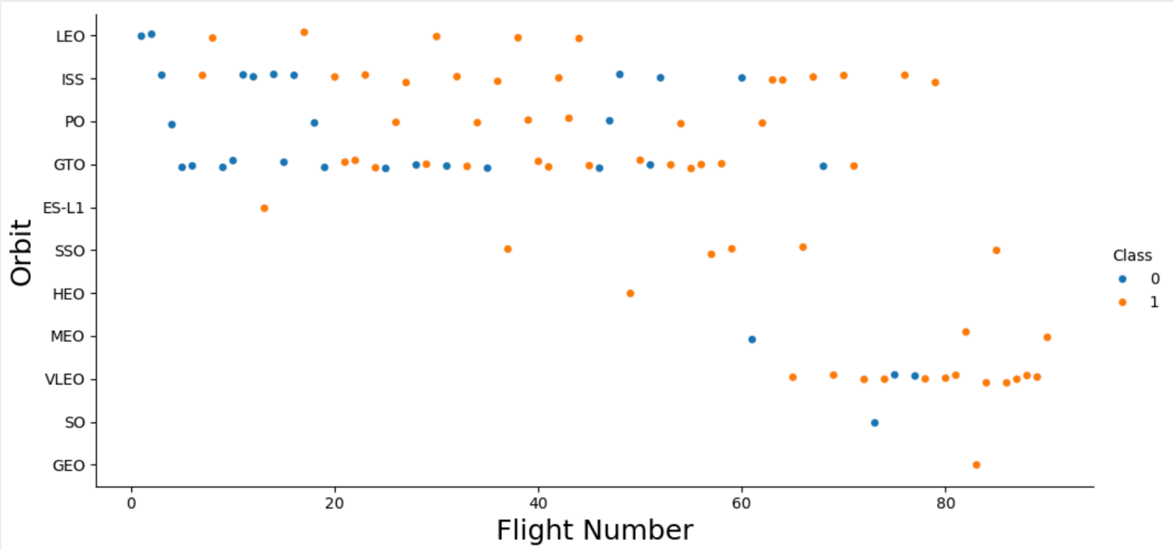


Analyze the plotted bar chart try to find which orbits have high success rate.

Task 4: Visualize the relationship between FlightNumber and Orbit type

For each orbit, we want to see if there is any relationship between FlighNumber and Orbit type.

```
# Plot a scatter point chart with x axis to be FlightNumber
and y axis to be the Orbit, and hue to be the class value
sns.catplot(y = 'Orbit',
            x = 'FlightNumber',
            hue = 'Class',
            data = df,
            aspect = 2)
plt.xlabel('Flight Number',
           fontsize = 18)
plt.ylabel('Orbit',
           fontsize = 18)
plt.show()
```



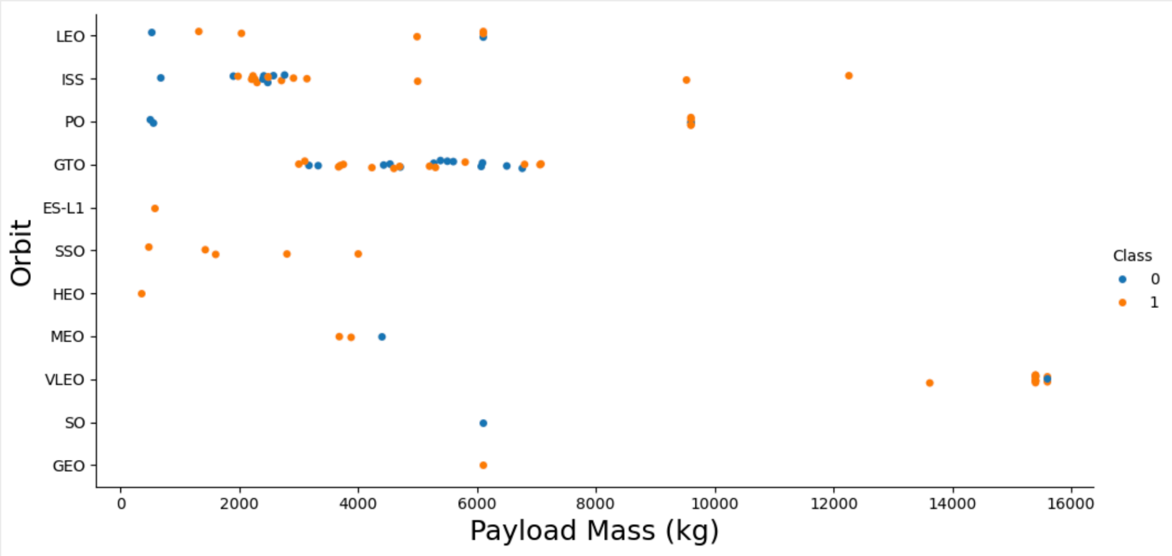
We should see that in the LEO orbit, the Success appears related to the number of flights; on the other hand, there seems to be no relationship between flight number when in GTO orbit.

Task 5: Visualize the relationship between Payload and Orbit Type.

Similarly, we can plot the Payload vs Orbit scatter point charts to reveal the relationship between Payload and Orbit type.

```
# Plot a scatter point chart with x axis to be Payload and y
axis to be the Orbit, and hue to be the class value
sns.catplot(y = 'Orbit',
            x = 'PayloadMass',
            hue = 'Class',
            data = df,
            aspect = 2)
plt.xlabel('Payload Mass (kg)',
           fontsize = 18)
plt.ylabel('Orbit',
```

```
            fontsize = 18)
plt.show()
```



With heavy payloads the successful landing or positive landing rate are more for Polar, LEO and ISS.

However, for GTO, we cannot distinguished this well as both positive landing rate and negative landing (unsuccessful mission) are both there here.

Taks 6: Visualize the launch success yearly trend

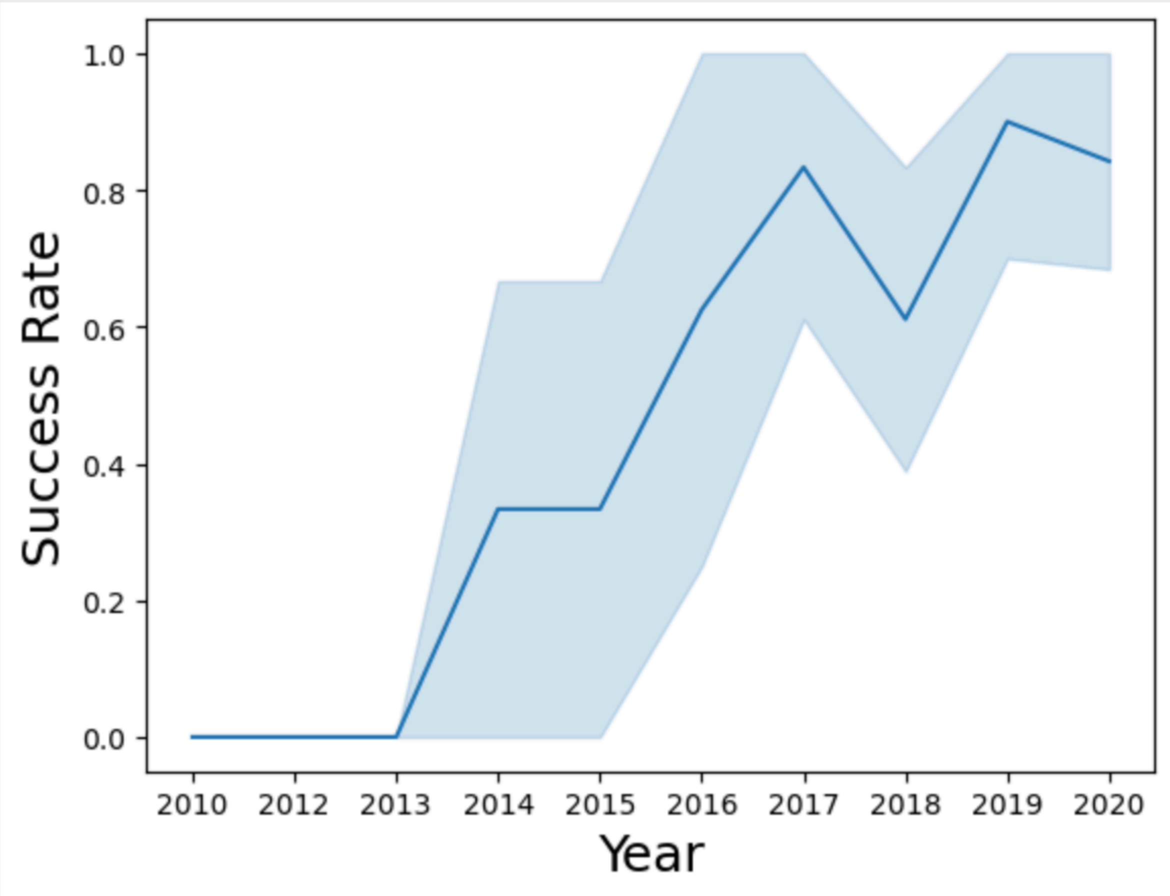
You can plot a line chart with x axis to be *Year* and y axis to be average success rate, to get the average launch success trend.

The function will help you get the year from the date:

```
# A function to Extract years from the date

year = []

def Extract_year():
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year
Extract_year()
df['Date'] = year

df.head()
```

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
0	1	2010	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577366	28.561857	0
1	2	2012	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366	28.561857	0
2	3	2013	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.577366	28.561857	0
3	4	2013	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	0	B1003	-120.610829	34.632093	0
4	5	2013	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561857	0

```
# Plot a line chart with x axis to be the extracted year and
y axis to be the success rate
sns.lineplot(data = df,
             x = 'Date',
             y = 'Class')
plt.xlabel('Year',
           fontsize = 18)
plt.ylabel('Success Rate',
           fontsize = 18)
plt.show()
```



You can observe that the success rate since 2013 kept increasing till 2020.

Features Engineering

By now, you should obtain some preliminary insights about how each important variable would affect the success rate, we will select the features that will be used in success prediction in the future module.

```
features = df[['FlightNumber',
               'PayloadMass',
               'Orbit',
               'LaunchSite',
               'Flights',
               'GridFins',
               'Reused',
               'Legs',
               'LandingPad',
               'Block',
               'ReusedCount',
               'Serial']]
features.head()
```

	FlightNumber	PayloadMass	Orbit	LaunchSite	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial
0	1	6104.959412	LEO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B0003
1	2	525.000000	LEO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B0005
2	3	677.000000	ISS	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B0007
3	4	500.000000	PO	VAFB SLC 4E	1	False	False	False	NaN	1.0	0	B1003
4	5	3170.000000	GTO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B1004

Task 7: Create dummy variables to categorical columns

Use the function *get_dummies* and *features* dataframe to apply OneHotEncoder to the columns *Orbit*, *LaunchSite*, *LandingPad*, and *Serial*. Assign the values to the variable *features_one_hot*, display the results using the method head. Your result dataframe must include all features including the encoded ones.

```
# HINT: Use get_dummies() function on the categorical
columns
features_one_hot = pd.get_dummies(features,
                                  columns = ['Orbit',
                                             'LaunchSite',
                                             'LandingPad',
                                             'Serial'])
features_one_hot.head()
```

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	...	Serial_B1048	Serial_B1049	Serial_B1050	Serial_B1051	Serial_B1054	Serial_B1056	Serial_B1058	Seria
0	1	6104.959412	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	
1	2	525.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	
2	3	677.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	
3	4	500.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	
4	5	3170.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	

5 rows × 80 columns

Task 8: Cast all numeric columns to *float64*

Now that our *features_one_hot* dataframe only contains numbers, cast the entire dataframe to variable type *float64*.

```
# HINT: use astype function
features_one_hot.astype(float)
```

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	...	Serial_B1048	Serial_B1049	S
0	1.0	6104.959412	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	
1	2.0	525.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	
2	3.0	677.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	
3	4.0	500.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	
4	5.0	3170.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
85	86.0	15400.000000	2.0	1.0	1.0	1.0	5.0	2.0	0.0	0.0	...	0.0	0.0	
86	87.0	15400.000000	3.0	1.0	1.0	1.0	5.0	2.0	0.0	0.0	...	0.0	0.0	
87	88.0	15400.000000	6.0	1.0	1.0	1.0	5.0	5.0	0.0	0.0	...	0.0	0.0	
88	89.0	15400.000000	3.0	1.0	1.0	1.0	5.0	2.0	0.0	0.0	...	0.0	0.0	
89	90.0	3681.000000	1.0	1.0	0.0	1.0	5.0	0.0	0.0	0.0	...	0.0	0.0	

90 rows × 80 columns

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.

```
features_one_hot.to_csv('dataset_part_3.csv', index = False)
```