1 SpaceX Falcon 9 First Stage Landing Prediction¶

1.1 Assignment: Exploring and Preparing Data¶

Estimated time needed: **70** minutes

In this assignment, 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 due to the fact that SpaceX can reuse the first stage.

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

Falcon 9 first stage will land successfully

Several examples of an unsuccessful landing are shown here:

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

1.2 Objectives¶

Perform exploratory Data Analysis and Feature Engineering using `Pandas` and `Matplotlib`

- Exploratory Data Analysis
- Preparing Data Feature Engineering

1.2.1 Import Libraries and Define Auxiliary Functions¶

We will import the following libraries the lab

In [2]:
```
# andas 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 matr
ices, along with a large collection of high-level mathematical functions to operate on these arrays
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a MatLab like plotting framework. We will use this
in our plotter function to plot data.
import matplotlib.pyplot as plt
#Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing
attractive and informative statistical graphics
import seaborn as sns
```

1.3 Exploratory Data Analysis¶

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

In [4]:
```
df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datase
ts/dataset_part_2.csv")

# If you were unable to complete the previous lab correctly you can uncomment and load this csv

# df = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DS0
701EN-SkillsNetwork/api/dataset_part_2.csv')

df.head(5)
```

Out[4]:

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

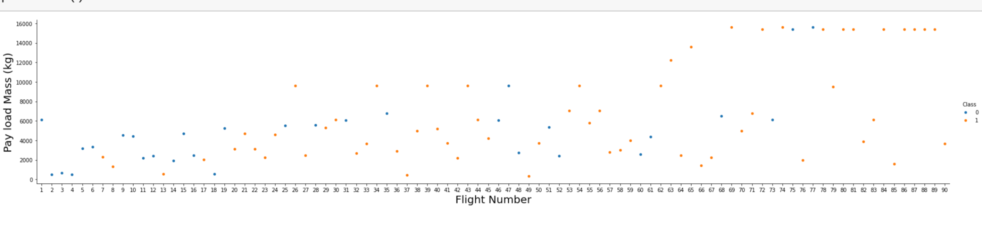
In [ ]:
```
df = pd.read_csv('dataset_part_2.csv')

df.head(5)
```

In [3]:
```
df.to_csv('dataset_part_2.csv', index = None)
```

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.

In [4]:
```
sns.catplot(y="PayloadMass", x="FlightNumber", hue="Class", data=df, aspect = 5) #seaborn's catplot() to create a ca
tegorical plot analyzing cateforical variables
plt.xlabel("Flight Number",fontsize=20) #aspect ratio to 5, meaning the width will be 5 times the height.
plt.ylabel("Pay load Mass (kg)",fontsize=20)
plt.show()
```

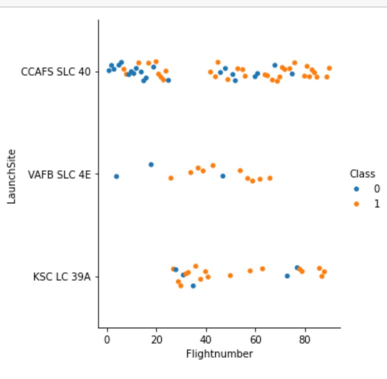


Next, let's drill down to each site visualize its detailed launch records.

1.3.1 TASK 1: Visualize the relationship between Flight Number and Launch Site¶

Use the function `catplot` to plot `FlightNumber` vs `LaunchSite`, set the parameter `x` parameter to `FlightNumber`,set the `y` to `Launch Site` and set the parameter `hue` to `'class'`

In [12]:
```
# Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to be the cla
ss value

sns.catplot(x = 'FlightNumber', y = 'LaunchSite',hue = 'Class', data = df)

plt.xlabel('Flightnumber')
plt.ylabel('LaunchSite')
plt.show()
```

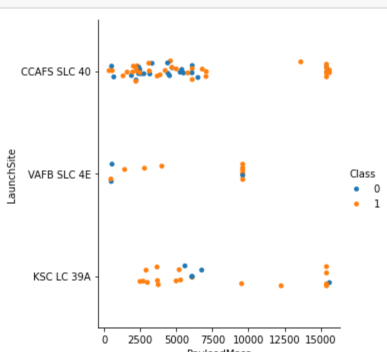


Now try to explain the patterns you found in the Flight Number vs. Launch Site scatter point plots.

1.3.2 TASK 2: Visualize the relationship between Payload and Launch Site¶

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

In [14]:
```
# 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 th
e class value

sns.catplot(x = 'PayloadMass', y = 'LaunchSite', hue = 'Class', data = df )
plt.xlabel('PayloadMass')
plt.ylabel('LaunchSite')
plt.show()
```



In [ ]:

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

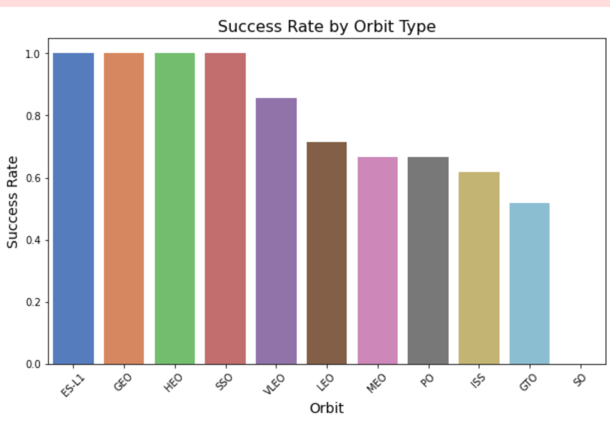
1.3.3 TASK 3: Visualize the relationship between success rate of each orbit type¶

Next, we want to visually check if there are any relationship between success rate and orbit type.

Let's create a `bar chart` for the sucess rate of each orbit

In [23]:
```
# HINT use groupby method on Orbit column and get the mean of Class column

df_orbit_success_rate = df.groupby('Orbit')['Class'].mean().reset_index()
#In this case, the 'Orbit' column becomes the index. However, using reset_index()
#after the grouping operation flattens the index and converts the 'Orbit' values back into a regular column.
#This makes it easier to work with the DataFrame, especially when visualizing the data or performing further analysi
s.

df_orbit_success_rate = df_orbit_success_rate.sort_values(by='Class', ascending=False)
# Sort the DataFrame by the 'Class' column (success rate) in descending order

plt.figure(figsize=(10, 6))  # Set the figure size
sns.barplot(x='Orbit', y='Class', data=df_orbit_success_rate, palette='muted')

plt.xlabel('Orbit', fontsize=14)
plt.ylabel('Success Rate', fontsize=14)
plt.title('Success Rate by Orbit Type', fontsize=16)
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.show()
```

```
/var/folders/1c/w8qh2r6x0q7_sb813g8j8rvh0000gn/T/ipykernel_26151/3829626987.py:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
ue` and set `legend=False` for the same effect.

  sns.barplot(x='Orbit', y='Class', data=df_orbit_success_rate, palette='muted')
```

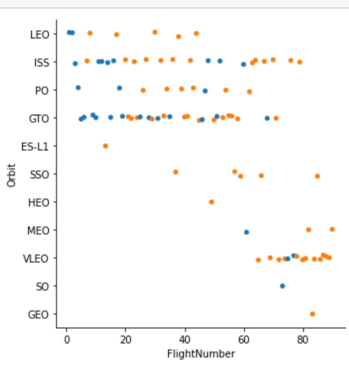


Analyze the ploted bar chart try to find which orbits have high sucess rate.

1.3.4 TASK 4: Visualize the relationship between FlightNumber and Orbit type¶

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

In [25]:
```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class valu
e
sns.catplot(x = 'FlightNumber', y = 'Orbit', hue = 'Class', data = df)
plt.show()
```

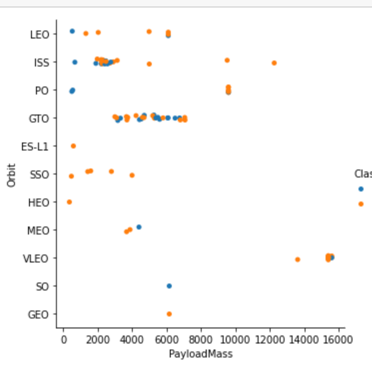


You should see that in the LEO orbit the Success appears related to the number of flights; on the other hand, there seems to be no relationship between flight number when in GTO orbit.

1.3.5 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

In [27]:
```
# 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(x = 'PayloadMass', y = 'Orbit', hue = 'Class', data = df)
plt.show()
```



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

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

1.3.6 TASK 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:

In [93]:
```
# Create series and define the total index beforehand
year= pd.Series(index=range(len(df['Date'])))
def Extract_year(date):
    for i in range(len(date)):
        year[i] =date[i].split("-")[0]
    return year

df['year'] = Extract_year(df['Date'])
df['year']
```

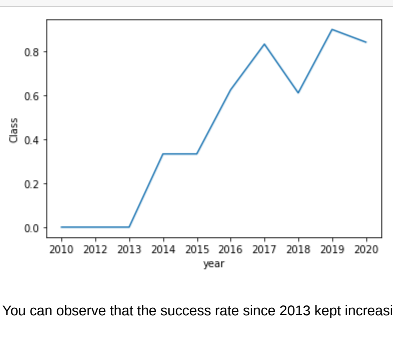
```
/var/folders/1c/w8qh2r6x0q7_sb813g8j8rvh0000gn/T/ipykernel_26151/3192594650.py:1: DeprecationWarning: The default dty
pe for empty Series will be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to silence
this warning.
  year= pd.Series(index=range(len(df['Date'])))
```

Out[93]:
```
0     2010.0
1     2012.0
2     2013.0
3     2013.0
4     2013.0
       ...
85    2020.0
86    2020.0
87    2020.0
88    2020.0
89    2020.0
Name: year, Length: 90, dtype: float64
```

In [15]:
```
# A function to Extract years from the date
year= [] #Create year list
def Extract_year(date):
    for i in date:
        year.append(i.split("-")[0])
    return year


df['year'] = Extract_year(df['Date'])


df_year_success1 = df.groupby('year')['Class'].mean().reset_index
#If you don't reset the index, after using groupby in pandas
#the 'year' column becomes the index of the resulting DataFrame
# reseting index can adjust year column into regular column
```

Out[15]:
```
<bound method Series.reset_index of year
2010    0.000000
2012    0.000000
2013    0.000000
2014    0.333333
2015    0.333333
2016    0.625000
2017    0.833333
2018    0.611111
2019    0.900000
2020    0.842105
Name: Class, dtype: float64>
```

In [8]:
```
# Plot a line chart with x axis to be the extracted year and y axis to be the success rates

sns.lineplot(x = 'year', y = 'Class' , data = df_year_success)
plt.show()
```



You can observe that the success rate since 2013 kept increasing till 2017 (stable in 2014) and after 2015 it started increasing.

1.4 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.

In [16]:
```
features = df[['FlightNumber', 'PayloadMass', 'Orbit', 'LaunchSite', 'Flights', 'GridFins', 'Reused', 'Legs', 'Landi
ngPad', 'Block', 'ReusedCount', 'Serial']]
features.head()
```

Out[16]:

	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

1.4.1 TASK 7: Create dummy variables to categorical columns¶

Use the function `get_dummies` and `features` dataframe to apply OneHotEncoder to the column `Orbits`, `LaunchSite`, `LandingPad`, and `Serial`. Assign the value to the variable `features_one_hot`, display the results using the method head. Your result dataframe must include all features including the encoded ones.

In [20]:
```
# 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()
```

Out[20]:

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

5 rows × 80 columns

1.4.2 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`

In [26]:
```
# HINT: use astype function
features_one_hot = features_one_hot.astype('float64')
features_one_hot.dtypes.head()
```

Out[26]:
```
FlightNumber    float64
PayloadMass     float64
Flights         float64
GridFins        float64
Reused          float64
dtype: object
```

In [29]:
```
features_one_hot.to_csv('dataset_part_3.csv', index=False)
```

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)`

1.5 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.

Nayef Abou Tayoun is a Data Scientist at IBM and pursuing a Master of Management in Artificial intelligence degree at Queen's University.

1.6 Change Log¶

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-10-12	1.1	Lakshmi Holla	Modified markdown
2020-09-20	1.0	Joseph	Modified Multiple Areas
2020-11-10	1.1	Nayef	updating the input data