SpaceX Falcon 9 First Stage Landing Prediction

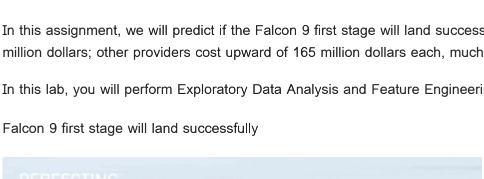
Assignment: Exploring and Preparing Data

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.

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

We will import the following libraries the lab

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

In [1]:
```
# 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
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 fu
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 inf
import seaborn as sns
```

Exploratory Data Analysis

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

In [4]:
```
import requests
import io

URL = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_2.csv"
response = requests.get(URL)
dataset_part_2_csv = io.BytesIO(response.content)
df = pd.read_csv(dataset_part_2_csv)

df.head(5)
```

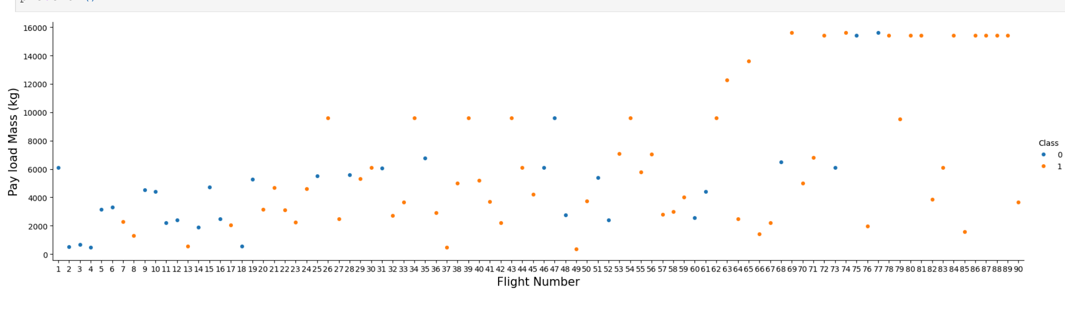
Out[4]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Lo
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.
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.
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.
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.
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.

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 also appears to be a factor; even with more massive payloads, the first stage often returns successfully.

In [54]:
```
sns.catplot(y="PayloadMass", x="FlightNumber", hue="Class", data=df, aspect = 3.7)
plt.xlabel("Flight Number",fontsize=15)
plt.ylabel("Pay load Mass (kg)",fontsize=15)
plt.show()
```

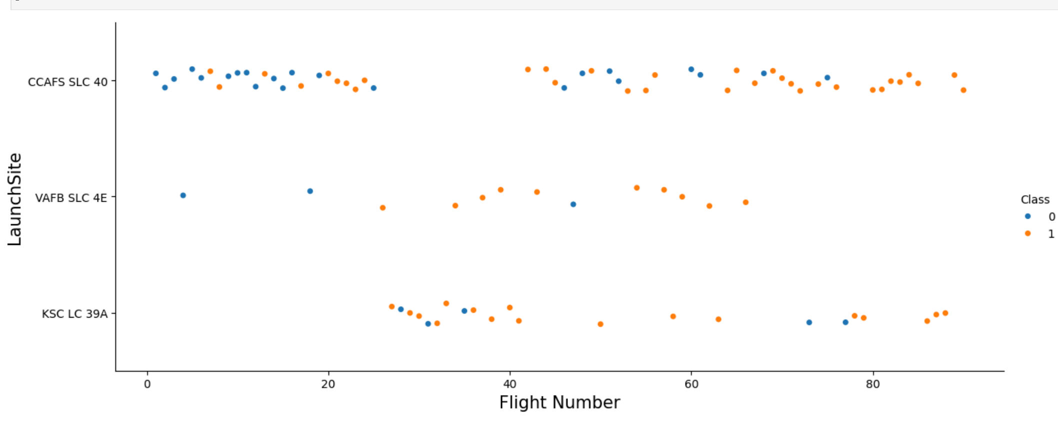


Next, let's drill down to each site visualize its detailed launch records.

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

Use the function `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 [53]:
```
# 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 class
sns.catplot(y="LaunchSite", x="FlightNumber", hue="Class", data=df, aspect = 2.5)

plt.xlabel("Flight Number",fontsize=15)
plt.ylabel("LaunchSite",fontsize=15)

plt.show()
```

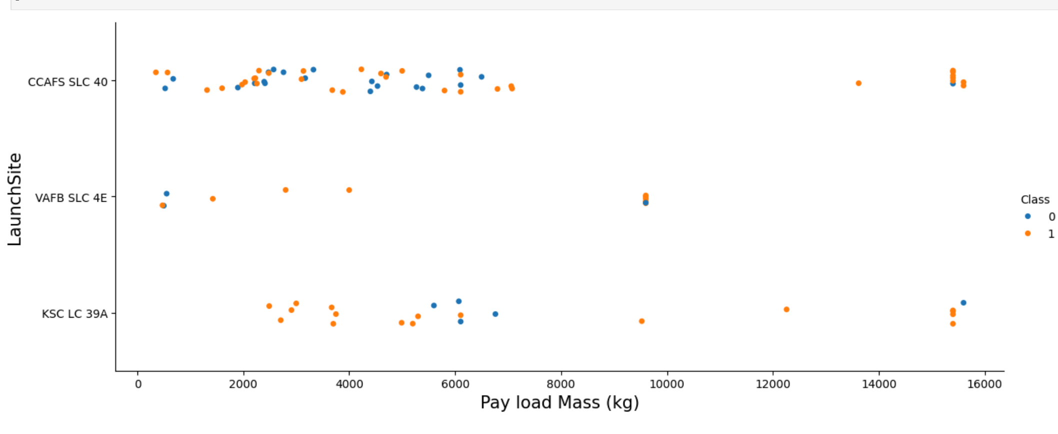


Now try to explain the patterns you found in the Flight Number vs. Launch Site scatter point plots.

TASK 2: Visualize the relationship between Payload Mass and Launch Site

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

In [58]:
```
# 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", data=df, aspect =2.5)

plt.xlabel("Pay load Mass (kg)",fontsize=15)
plt.ylabel("LaunchSite",fontsize=15)

plt.show()
```



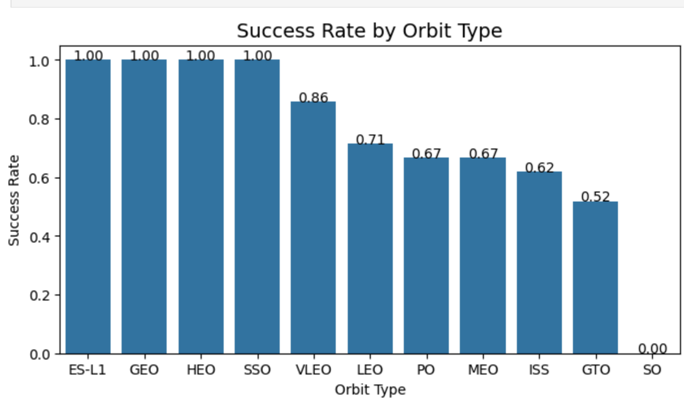
Now if you observe Payload Mass Vs. Launch Site scatter point chart you will find for the VAFB-SLC launchsite there are no rockets launched for heavypayload mass(greater than 10000).

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 [40]:
```
# HINT use groupby method on Orbit column and get the mean of Class column

# Set the figure size
plt.figure(figsize=(8, 4))

# Group by Orbit and calculate the mean of Class
orbit_success_rate = df.groupby('Orbit')['Class'].mean().reset_index()

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

# Create a bar chart
barplot = sns.barplot(x='Orbit', y='Class', data=orbit_success_rate)

plt.xlabel('Orbit Type', fontsize=10)
plt.ylabel('Success Rate', fontsize=10)

plt.title('Success Rate by Orbit Type', fontsize=14)
#plt.xticks(rotation=180)

# Add bar labels for all bars
for p in barplot.patches:
    barplot.annotate(f'{p.get_height():.2f}', (p.get_x() + p.get_width() / 2., p.get_height()), ha='center')

plt.show()
```

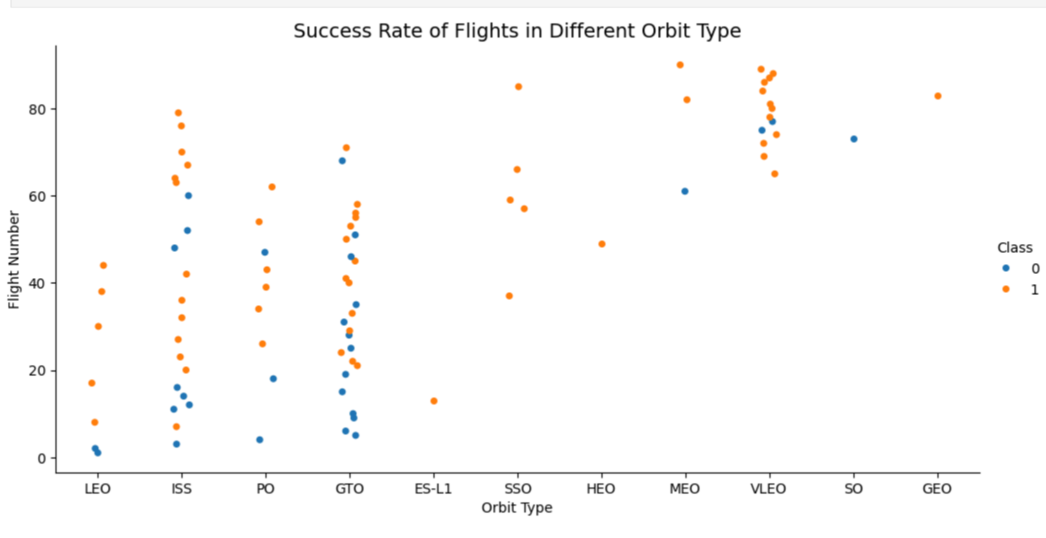


Analyze the plotted bar chart to identify which orbits have the highest success rates.

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 [66]:
```
# 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

# Set the figure size
#plt.figure(figsize=(12, 8))

sns.catplot(y="FlightNumber", x="Orbit", hue="Class", data=df, aspect=2)

plt.xlabel("Orbit Type",fontsize=10)
plt.ylabel("Flight Number",fontsize=10)

# Plot itle
plt.title('Success Rate of Flights in Different Orbit Type', fontsize=14)

plt.show()
```

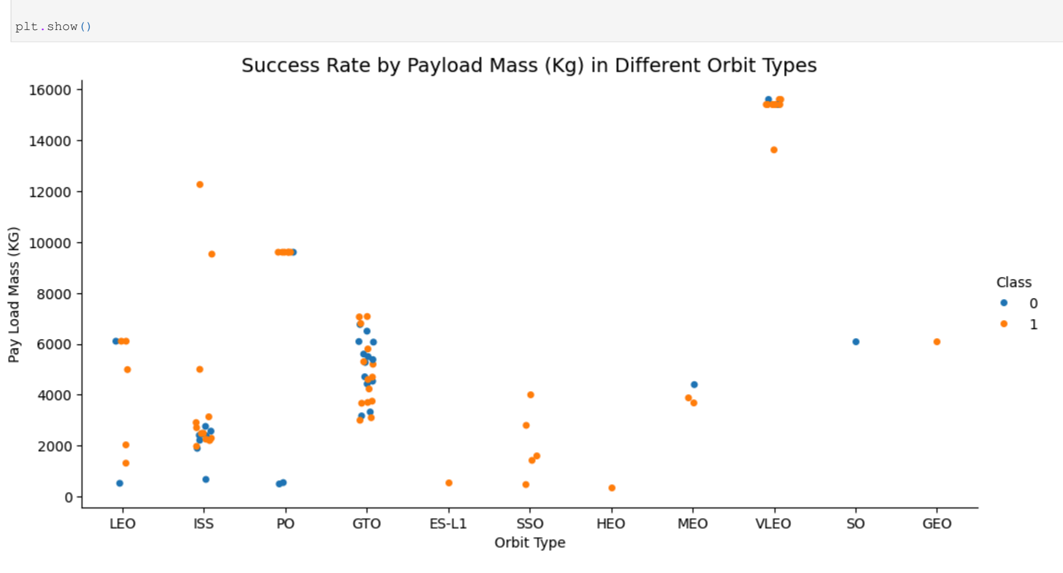


You can observe that in the LEO orbit, success seems to be related to the number of flights. Conversely, in the GTO orbit, there appears to be no relationship between flight number and success.

TASK 5: Visualize the relationship between Payload Mass and Orbit type

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

In [64]:
```
# Plot a scatter point chart with x axis to be Payload Mass and y axis to be the Orbit, and hue to be the class value

sns.catplot(y="PayloadMass", x="Orbit", hue="Class", data=df, aspect=2)

plt.xlabel("Orbit Type",fontsize=10)
plt.ylabel("Pay Load Mass (KG)",fontsize=10)

# Plot itle
plt.title('Success Rate by Payload Mass (Kg) in Different Orbit Types', fontsize=14)

plt.show()
```



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

However, for GTO, it's difficult to distinguish between successful and unsuccessful landings as both outcomes are present.

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.

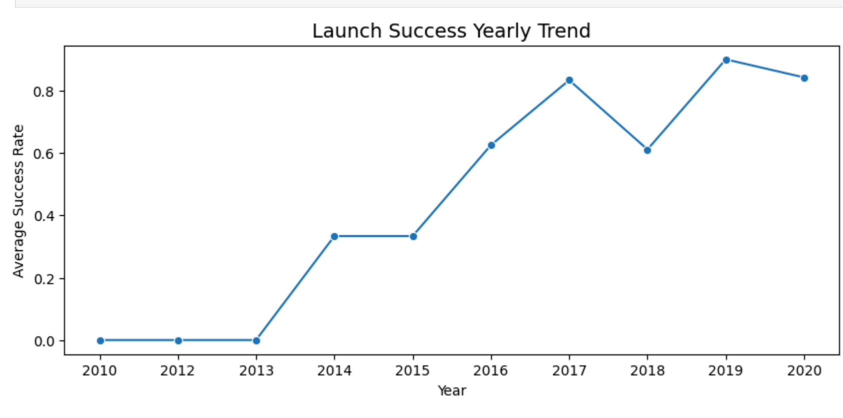
The function will help you get the year from the date:

In [67]:
```
# 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()
```

Out[67]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Lo
0	1	2010	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.
1	2	2012	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.
2	3	2013	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.
3	4	2013	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	0	B1003	-120.
4	5	2013	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.

In [75]:
```
# Plot a line chart with x axis to be the extracted year and y axis to be the success rate

# Extract years from the Date column
df['Year'] = df['Date'].apply(lambda x: x.split('-')[0])

# Group by Year and calculate the average success rate
yearly_success_rate = df.groupby('Year')['Class'].mean().reset_index()

# Set the figure size
plt.figure(figsize=(10, 4))

# Plot the line chart
sns.lineplot(x='Year', y='Class', data=yearly_success_rate, marker='o')

plt.xlabel('Year', fontsize=10)
plt.ylabel('Average Success Rate', fontsize=10)

plt.title('Launch Success Yearly Trend', fontsize=14)

plt.show()
```



you can observe that the sucess 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.

In [85]:
```
features = df[['FlightNumber', 'PayloadMass', 'Orbit', 'LaunchSite', 'Flights', 'GridFins', 'Reused', 'Legs', 'LandingPad', 'Block', '
features
```

Out[85]:

	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
...	...	...	...	...	...	...	...	...	...	...	...	...
85	86	15400.000000	VLEO	KSC LC 39A	2	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	2	B1060
86	87	15400.000000	VLEO	KSC LC 39A	3	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	2	B1058
87	88	15400.000000	VLEO	KSC LC 39A	6	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	5	B1051
88	89	15400.000000	VLEO	CCAFS SLC 40	3	True	True	True	5e9e3033383ecbb9e534e7cc	5.0	2	B1060
89	90	3681.000000	MEO	CCAFS SLC 40	1	True	False	True	5e9e3032383ecb6bb234e7ca	5.0	0	B1062

90 rows × 12 columns

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 [86]:
```
# To create dummy variables for the categorical columns Orbit, LaunchSite, LandingPad, and Serial, using the pd.get_dummies function i
# This will apply one-hot encoding to these columns and include the encoded features in the resulting DataFrame.

# Select the columns to be encoded
categorical_columns = ['Orbit', 'LaunchSite', 'LandingPad', 'Serial']

# Apply one-hot encoding to the selected columns
features_one_hot = pd.get_dummies(features, columns=categorical_columns)

# Display the results using the head method
print(features_one_hot.head())
```

```
   FlightNumber  PayloadMass  Flights  GridFins  Reused   Legs  Block  \
0             1  6104.959412        1     False   False  False    1.0
1             2   525.000000        1     False   False  False    1.0
2             3   677.000000        1     False   False  False    1.0
3             4   500.000000        1     False   False  False    1.0
4             5  3170.000000        1     False   False  False    1.0

   ReusedCount  Orbit_ES-L1  Orbit_GEO  ...  Serial_B1048  Serial_B1049  \
0            0        False      False  ...         False         False
1            0        False      False  ...         False         False
2            0        False      False  ...         False         False
3            0        False      False  ...         False         False
4            0        False      False  ...         False         False

   Serial_B1050  Serial_B1051  Serial_B1054  Serial_B1056  Serial_B1058  \
0         False         False         False         False         False
1         False         False         False         False         False
2         False         False         False         False         False
3         False         False         False         False         False
4         False         False         False         False         False

   Serial_B1059  Serial_B1060  Serial_B1062
0         False         False         False
1         False         False         False
2         False         False         False
3         False         False         False
4         False         False         False

[5 rows x 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`

In [87]:
```
# HINT: use astype function
# Cast all numeric columns to float64
features_one_hot = features_one_hot.astype('float64')
```

In [90]:
```
# Display the results using the head method
features_one_hot
```

Out[90]:

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	...	Serial_B1048	Serial_B1049	Serial_B1050	Serial_
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	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	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	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	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	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	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	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	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	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	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.

In [92]:
```
features_one_hot.to_csv('dataset_part_3.csv', index=False)
```

Authors

Pratiksha Verma