EDA With Data Visualization

Objectives

- Exploratory Data Analysis
- Preparing Data Feature Engineering

Import Libraries and Define Auxiliary Functions

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt # plotting library and pyplot gives plotting framework
import seaborn as sns #data visualization based on matplotlib. Provides level interface for pretty graphs.
```

Exploratory Data Analysis

In [3]:
```
df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_2.csv")

df.head(3)
```

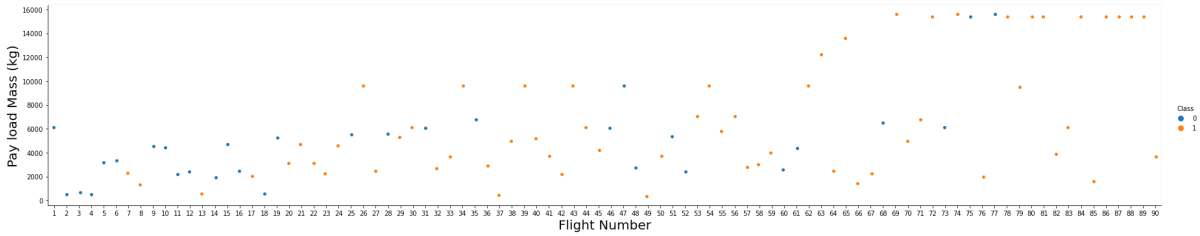
Out[3]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	Gric
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	

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]:
```
#catplot: Figure-level interface for drawing categorical plots onto a FacetGrid
#Class is whether it was a successful landing or not
sns.catplot(x="FlightNumber", y="PayloadMass", data=df, hue="Class",  aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Pay load Mass (kg)",fontsize=20)
plt.show()
```



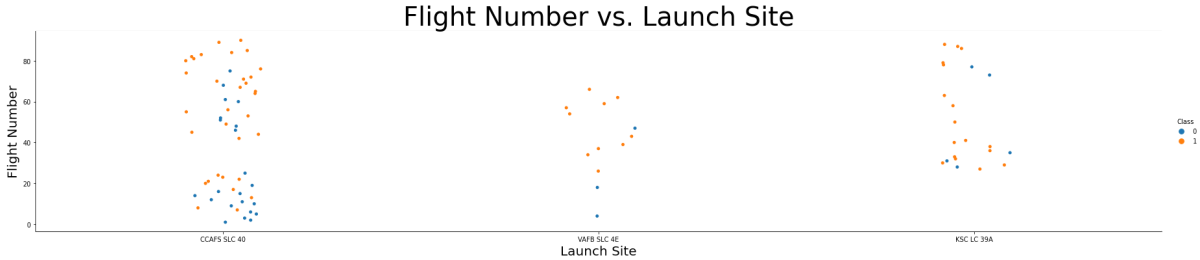
We see that different launch sites have different success rates. `CCAFS LC-40` , has a success rate of 60 %, while `KSC LC-39A` and `VAFB SLC 4E` has a success rate of 77%.

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

Visualize the relationship between Flight Number and Launch Site

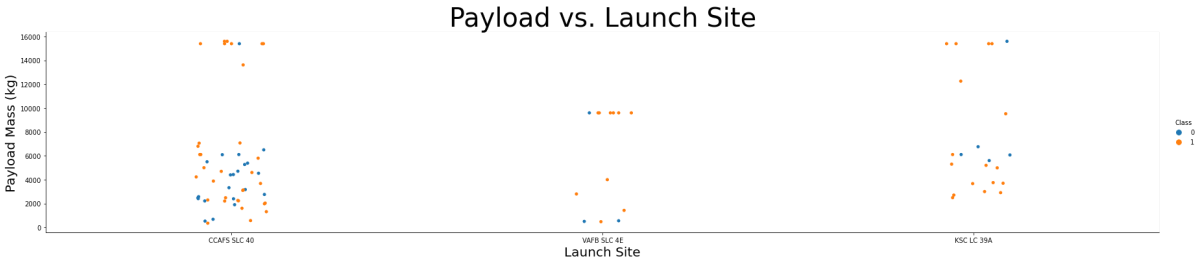
Use the function `catplot` to plot `FlightNumber` vs `LaunchSite` , set the parameter `x` parameter to `FlightNumber` ,set the `y` to `Launch Site` and set the parameter `hue` to `'class'`

In [6]:
```
sns.catplot(x="LaunchSite", y="FlightNumber", data=df, hue="Class", aspect=5)
plt.xlabel("Launch Site",fontsize=20)
plt.ylabel("Flight Number",fontsize=20)
plt.title("Flight Number vs. Launch Site", fontsize = 40)
plt.show()
```



Visualize the relationship between Payload and Launch Site

In [7]:
```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis t
o be the launch site, and hue to be the class value
sns.catplot(x="LaunchSite", y="PayloadMass", data=df, hue="Class", aspect=5) #
aspect draws out to wide angle
plt.xlabel("Launch Site", fontsize=20)
plt.ylabel("Payload Mass (kg)", fontsize=20)
plt.title("Payload vs. Launch Site", fontsize = 40)
plt.show()
```



Heaviest payloads were launched from KSC and CCAFS, and were mostly successful. Most unsuccessful payloads were at CCAFS and were between 2000-6000kg at CCAFS.

Visualize the relationship between success rate of each orbit type

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

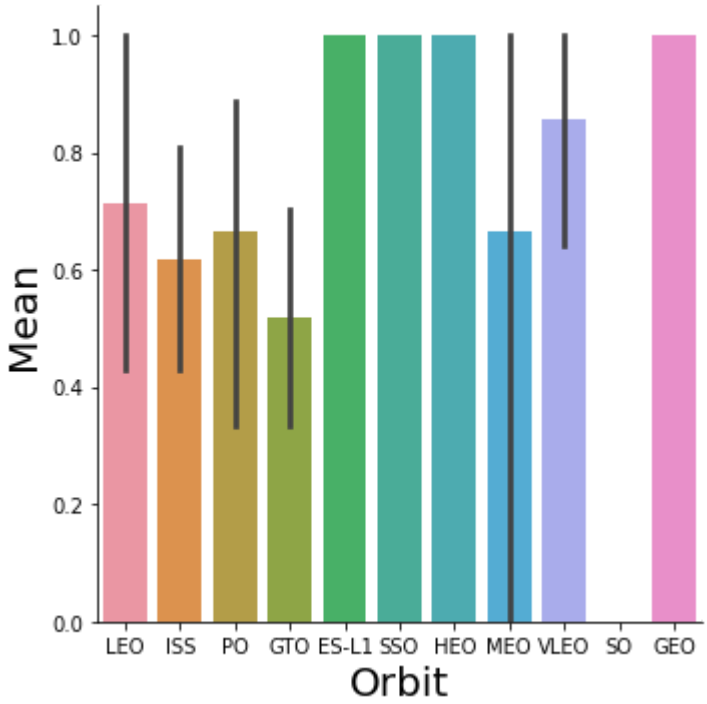
In [106]:
```
#df['Orbit'].value_counts()

df.groupby(['Orbit']).mean()
```

Out[106]:

Orbit	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCo
ES-L1	13.000000	570.000000	1.000000	1.000000	0.000000	1.000000	1.000000	0.000(
GEO	83.000000	6104.959412	2.000000	1.000000	1.000000	1.000000	5.000000	2.000(
GTO	35.037037	5011.994444	1.407407	0.629630	0.333333	0.629630	3.037037	0.962
HEO	49.000000	350.000000	1.000000	1.000000	0.000000	1.000000	4.000000	1.000(
ISS	39.142857	3279.938095	1.238095	0.809524	0.238095	0.857143	3.142857	1.285
LEO	20.000000	3882.839748	1.000000	0.571429	0.000000	0.714286	2.142857	0.428
MEO	77.666667	3987.000000	1.000000	0.666667	0.000000	0.666667	5.000000	0.666(
PO	36.333333	7583.666667	1.333333	0.888889	0.333333	0.777778	3.222222	1.555
SO	73.000000	6104.959412	4.000000	0.000000	1.000000	0.000000	5.000000	3.000(
SSO	60.800000	2060.000000	2.400000	1.000000	0.800000	1.000000	4.600000	3.200(
VLEO	78.928571	15315.714286	3.928571	1.000000	1.000000	1.000000	5.000000	3.928

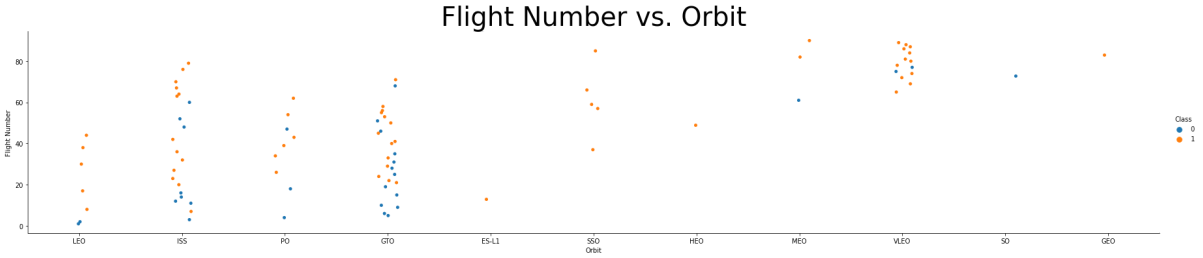
Analyze the ploted bar chart try to find which orbits have high sucess rate.

In [12]:
```
sns.catplot(x="Orbit",y="Class", kind="bar",data=df)
plt.xlabel("Orbit",fontsize=20)
plt.ylabel("Mean",fontsize=20)
plt.show()
```



Visualize the relationship between FlightNumber and Orbit type

In [10]:
```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be t
he Orbit, and hue to be the class value
sns.catplot(x="Orbit", y="FlightNumber", data=df, hue="Class", aspect=5)
plt.xlabel("Orbit")
plt.ylabel("Flight Number")
plt.title("Flight Number vs. Orbit", fontsize = 40)
plt.show()
```

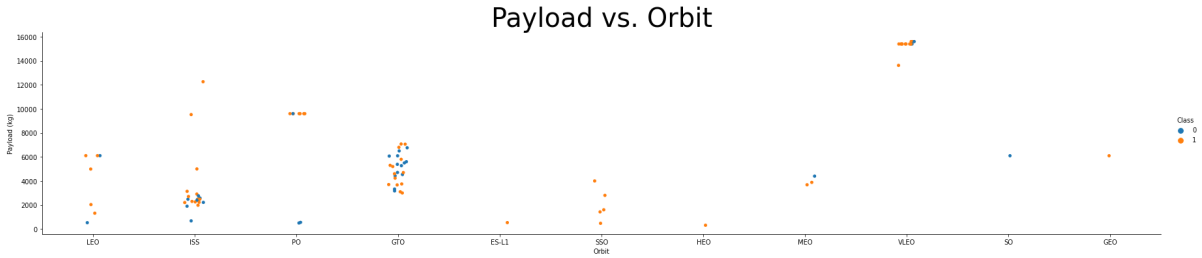


LEO orbit the Success appears related to the number of flights; on the other hand, there seems to be no relationship between flight number when in GTO orbit.

because LEO is evenly spaced

Visualize the relationship between Payload and Orbit type

In [16]:
```
# 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="Orbit", y="PayloadMass", data=df, hue="Class", aspect=5)
plt.xlabel("Orbit")
plt.ylabel("Payload (kg)")
plt.title("Payload vs. Orbit", fontsize=40)
plt.show()
```

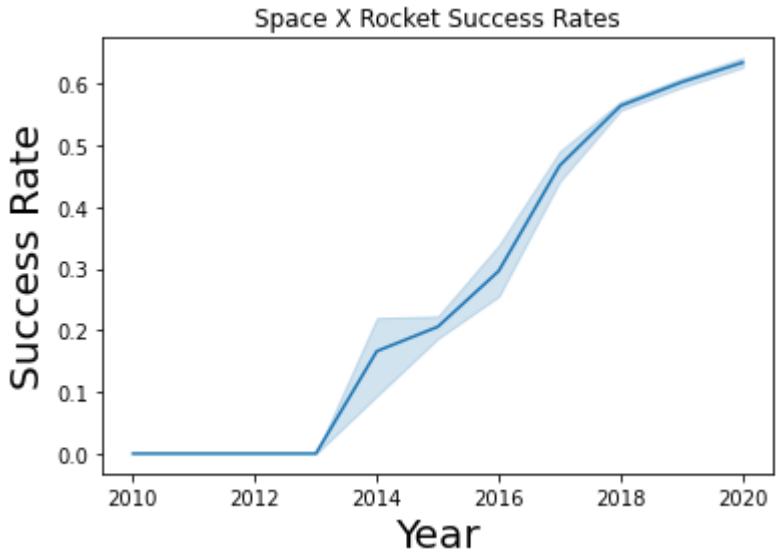


You should observe that Heavy payloads have a negative influence on GTO orbits and positive on GTO and Polar LEO (ISS) orbits.

Visualize the launch success yearly trend

The function will help get the year from the date:

In [105]:
```
#Create an array of the years
year = pd.DatetimeIndex(df['Date']).year
year = np.array(list(year))


#Initialize variables for successrate calc
successratelist = []
successrate = 0.00
runningcount = 1
data = 0

for x in df['Class']:
    data = x + data
    successrate = data/runningcount
    successratelist.append(successrate)
    runningcount= runningcount +1

#Create an array for successrate
successratelist = np.array(successratelist)
#Turn the two arrays into a dictionary
d = {'successrate':successratelist,'year':year}


sns.lineplot(x="year", y="successrate", data=d)
plt.xlabel("Year",fontsize=20)
plt.title('Space X Rocket Success Rates')
plt.ylabel("Success Rate",fontsize=20)
plt.show()
```



The sucess rate since 2013 kept increasing till 2020

Features Engineering

Obtain preliminary insights about how each important variable would affect the success rate. 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', 'ReusedCount', 'Serial'
]]
features.head()
```

Out[85]:

	FlightNumber	PayloadMass	Orbit	LaunchSite	Flights	GridFins	Reused	Legs	LandingPad
0	1	6104.959412	LEO	CCAFS SLC 40	1	False	False	False	NaN
1	2	525.000000	LEO	CCAFS SLC 40	1	False	False	False	NaN
2	3	677.000000	ISS	CCAFS SLC 40	1	False	False	False	NaN
3	4	500.000000	PO	VAFB SLC 4E	1	False	False	False	NaN
4	5	3170.000000	GTO	CCAFS SLC 40	1	False	False	False	NaN

In [34]:
```
features['Orbit'].value_counts()
```

Out[34]:
```
GTO      27
ISS      21
VLEO     14
PO        9
LEO       7
SSO       5
MEO       3
ES-L1     1
HEO       1
SO        1
GEO       1
Name: Orbit, dtype: int64
```

In [88]:
```
#features['LaunchSite'].value_counts()
```

In [87]:
```
#features['LandingPad'].value_counts()
```

In [86]:
```
#features['Serial'].value_counts()
```

Encode categoricals to numerical assignments

The following columns need to be converted to numerical equivalents: `Orbits` , `LaunchSite` , `LandingPad` , and `Serial` .

For binary data, we use the function `get_dummies` to apply OneHotEncoder.

However, since this is not binary, we use `LabelEncoder` to the feature matrix.

In [91]:
```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
```

In [93]:
```
label_encoder.fit(features['Orbit'])
Orbit_mapping = dict(zip(label_encoder.classes_, label_encoder.transform(label
_encoder.classes_)))

features['Orbit'] = label_encoder.fit_transform(features['Orbit'])

Orbit_mapping
```

```
/opt/conda/envs/Python-3.7-OpenCE/lib/python3.7/site-packages/ipykernel/__mai
n__.py:4: 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/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
```

Out[93]:
```
{'ES-L1': 0,
 'GEO': 1,
 'GTO': 2,
 'HEO': 3,
 'ISS': 4,
 'LEO': 5,
 'MEO': 6,
 'PO': 7,
 'SO': 8,
 'SSO': 9,
 'VLEO': 10}
```

In [94]:
```
label_encoder.fit(features['LaunchSite'])
LaunchSite_mapping = dict(zip(label_encoder.classes_, label_encoder.transform(
label_encoder.classes_)))

features['LaunchSite'] = label_encoder.fit_transform(features['LaunchSite'])

LaunchSite_mapping
```

```
/opt/conda/envs/Python-3.7-OpenCE/lib/python3.7/site-packages/ipykernel/__mai
n__.py:4: 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/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
```

Out[94]:
```
{'CCAFS SLC 40': 0, 'KSC LC 39A': 1, 'VAFB SLC 4E': 2}
```

In [96]:
```
label_encoder.fit(features['LandingPad'].astype(str))
LandingPad_mapping = dict(zip(label_encoder.classes_, label_encoder.transform(
label_encoder.classes_)))

features['LandingPad'] = label_encoder.fit_transform(features['LandingPad'].as
type(str))
```

```
/opt/conda/envs/Python-3.7-OpenCE/lib/python3.7/site-packages/ipykernel/__mai
n__.py:4: 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/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
```

In [97]:
```
label_encoder.fit(features['Serial'].astype(str))
Serial_mapping = dict(zip(label_encoder.classes_, label_encoder.transform(labe
l_encoder.classes_)))

features['Serial'] = label_encoder.fit_transform(features['Serial'])
```

```
/opt/conda/envs/Python-3.7-OpenCE/lib/python3.7/site-packages/ipykernel/__mai
n__.py:4: 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/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
```

Confirm feature matrix was properly encoded and keep track of encodings

Now that our `features_one_hot` dataframe only contains numbers cast the entire dataframe to variable type `float64`

In [98]:
```
#features.drop(columns=['Orbit_d', 'Serial_d', 'LandingPad_d', 'LaunchSite_
d'], inplace=True)
features.head()
```

Out[98]:

	FlightNumber	PayloadMass	Orbit	LaunchSite	Flights	GridFins	Reused	Legs	LandingPad
0	1	6104.959412	5	0	1	False	False	False	5
1	2	525.000000	5	0	1	False	False	False	5
2	3	677.000000	4	0	1	False	False	False	5
3	4	500.000000	7	2	1	False	False	False	5
4	5	3170.000000	2	0	1	False	False	False	5

In [103]:
```
# Orbit_mapping,Serial_mapping,LandingPad_mapping,LaunchSite_mapping
```

Export to CSV

```
features.to_csv('dataset_part\_3.csv', index=False)
```