IBM Data Science Capstone Project

Studies and findings about SPACE X's FALCON 9

Leandro Norcini

30-04-2023

Link to the GitHub Repository: https://github.com/sacrosK11/IBM---Space-X---Data-Science-Capstone-Project.git

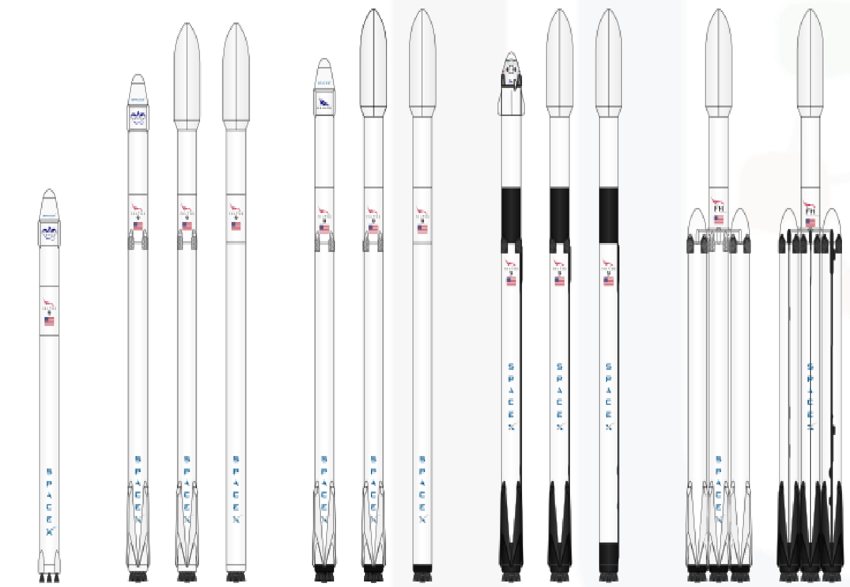
OUTLINE

- Executive Summary
- Introduction
- Methodology
- Data Collection
- Data Wrangling
- Visualization – Charts
- Dashboard
- Predictive Data Analysis
- Findings & Implications
- Conclusion

EXECUTIVE SUMMARY

- Methodologies
  - Data Collection
  - Data Wrangling
  - EDA with visual analytics
  - EDA with SQL
  - Folium interactive map
  - Plotly Dash Dashboard
  - Predictive Analysis (classification)

INTRODUCTION



- **Abstract**

Space X 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 Space X 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 space X for a rocket launch.

- **The Challange**

Collecting, cleaning and storing the data to find the best Machine Learning model.
Then we will test the model and try to make a prediction on unseen data. Finally we'll demostrate the results in a visual format to see if we can spot an insight.

METHODOLOGY

- Data Collection were made through:
  - SpaceX REST API
  - Web Scraping (source: https://en.wikipedia.org/wiki/List_of_Falcon_9_and_Falcon_Heavy_launches)
- Data Wrangling
  - Exploratory Data Analysis (EDA) to find some patterns in the data and determine what would be the label for training a Machine Learning supervised model
- EDA with Visual Analytics
  - Using Python Libraries to plot charts and better understand data and present them in an intuitive Dashboard
- EDA with SQL
  - SQL queries executed in a Jupyter Notebook to clean, filter and extract data from the Data Frame
- Predictive Analysis
  - Building, testing and presenting a Machine Learning model that best fits the needs for this project

IBM Developer
SKILLS NETWORK

DATA COLLECTION

```
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.im
unch":null,"media":null,"recovery":null},"flickr":{"small":[],"original":[]},"presskit":null,"webcast":"https://www.youtube.com/wat
t-launch.html","wikipedia":"https://en.wikipedia.org/wiki/DemoSat"},"static_fire_date_utc":"2006-03-17T00:00:00.000Z","static_fire_
3,"altitude":null,"reason":"merlin engine failure"}],"details":"Engine failure at 33 seconds and loss of vehicle","crew":[],"ships"
e":"FalconSat","date_utc":"2006-03-24T22:30:00.000Z","date_unix":1143239400,"date_local":"2006-03-25T10:30:00+12:00","date_precisio

Working with Python and the Requests module we retrieved the rocket launch data from the SpaceX API, then the response was converted into a Json file and transformed into a DataFrame using the Pandas library.

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

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	net	window	rocket	success	failures	details	crew	ships	capsules	payloads	launchpad	flight_number	name	date_utc	date_unix	date_local	date
0	2006-03-17T00:00:00.000Z	1.142554e+09	False	0.0	5e9d0d95eda69955f709d1eb	False	[{'time': 33, 'altitude': None, 'reason': 'merlin engine failure'}]	Engine failure at 33 seconds and loss of vehicle	[]	[]	[]	[5eb0e4b5b6c3bb0006eeb1e1]	5e9e4502f5090995de566f86	1	FalconSat	2006-03-24T22:30:00.000Z	1143239400	2006-03-25T10:30:00+12:00	
								Successful first stage burn and											

IBM Developer

SKILLS NETWORK

DATA COLLECTION

At this stage we needed to clean the data and filter the dataframe to minimize the variables at only the meaningful ones.
Finally this was the dataframe format:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude
4	1	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	2	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	3	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
7	4	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None	1.0	0	B1003	-120.610829	34.632093
8	5	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B1004	-80.577366	28.561857
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
89	86	2020-09-03	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	2	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	12	B1060	-80.603956	28.608058
90	87	2020-10-06	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	3	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	13	B1058	-80.603956	28.608058
91	88	2020-10-18	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	6	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	12	B1051	-80.603956	28.608058
92	89	2020-10-24	Falcon 9	15600.0	VLEO	CCSFS SLC 40	True ASDS	3	True	True	True	5e9e3033383ecbb9e534e7cc	5.0	12	B1060	-80.577366	28.561857
93	90	2020-11-05	Falcon 9	3681.0	MEO	CCSFS SLC 40	True ASDS	1	True	False	True	5e9e3032383ecb6bb234e7ca	5.0	8	B1062	-80.577366	28.561857

90 rows × 17 columns

Notice how we reduced the variables to 17 columns and the number of records to 90. All rows, representing a launch, were cast to the correct data type and filtered for the Booster Version as "Falcon 9" only.
However, we encountered some missing values in the PayloadMass column ('NaN' visible in the first row). To handle null values, when there are few, a common approach is to replace them with the mean of the column.
Since we had only 5 missing values, that is exactly what we did.

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

```
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
```

```
# Calculate the mean value of PayloadMass column
payloadmass_mean = data_falcon9['PayloadMass'].mean()
# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'].replace(np.nan, payloadmass_mean, inplace=True)
data_falcon9.isnull().sum()
```

```
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
```

NB. The ".isnull().sum()" method, showed us 26 Null values in the LandingPad column too but they're not representing a miss of data. The LandingPad retains None values to represent when landing pads were not used, for this reason we didn't manage the missing values.

DATA COLLECTION

WEB SCRAPING

Finally we built the remaining part of our dataframe with the use of Web Scraping on the Wikipedia website.

First, let's perform an HTTP GET method to request the Falcon9 Launch HTML page, as an HTTP response.

```
# use requests.get() method with the provided static_url
# assign the response to a object

response = requests.get(static_url).text
```

Create a `BeautifulSoup` object from the HTML `response`

```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
bs = BeautifulSoup(response, parser='html.parser')
```

Print the page title to verify if the `BeautifulSoup` object was created properly

```
# Use soup.title attribute
bs.title
```

```
<title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

Next, we want to collect all relevant column names from the HTML table header

Let's try to find all tables on the wiki page first. If you need to refresh your memory about `BeautifulSoup`, please check the external reference link towards the end of this lab

```
# Use the find_all function in the BeautifulSoup object, with element type `table`
# Assign the result to a list called `html_tables`

html_tables = bs.findAll('table')
```

Starting from the third table is our target table contains the actual launch records.

```
# Let's print the third table and check its content
first_launch_table = html_tables[2]
print(first_launch_table)
```

```
<table class="wikitable plainrowheaders collapsible" style="width: 100%;">
<tbody><tr>
<th scope="col">Flight No.
</th>
<th scope="col">Date and<br/>time (<a href="/wiki/Coordinated_Universal_Time" title="Coordinated Universal Time">UTC</a>)
</th>
<th scope="col"><a href="/wiki/List_of_Falcon_9_first-stage_boosters" title="List of Falcon 9 first-stage boosters">Version,<br/>Booster</a> <sup class="
</th>
<th scope="col">Launch site
</th>
<th scope="col">Payload<sup class="reference" id="cite_ref-Dragon_12-0"><a href="#cite_note-Dragon-12">[c]</a></sup>
</th>
<th scope="col">Payload mass
</th>
```

IBM Developer

DATA COLLECTION

WEB SCRAPING

Next, we just need to iterate through the `<th>` elements and apply the provided `extract_column_from_header()` to extract column name one by one

```
column_names = []

# Apply find_all() function with `th` element on first_launch_table
# Iterate each th element and apply the provided extract_column_from_header() to get a column name
# Append the Non-empty column name (`if name is not None and len(name) > 0`) into a list called column_names

for i in first_launch_table.find_all('th'):
    x = extract_column_from_header(i)
    if x != None and len(x) > 0:
        column_names.append(x)
```

Check the extracted column names

```
print(column_names)
```

```
['Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome']
```

We will create an empty dictionary with keys from the extracted column names in the previous task. Later, this dictionary will be converted into a Pandas dataframe

```
launch_dict= dict.fromkeys(column_names)

# Remove an irrelvant column
del launch_dict['Date and time ( )']

# Let's initial the launch_dict with each value to be an empty list
launch_dict['Flight No.'] = []
launch_dict['Launch site'] = []
launch_dict['Payload'] = []
launch_dict['Payload mass'] = []
launch_dict['Orbit'] = []
launch_dict['Customer'] = []
launch_dict['Launch outcome'] = []
# Added some new columns
launch_dict['Version Booster']=[]
launch_dict['Booster landing']=[]
launch_dict['Date']=[]
launch_dict['Time']=[]
```

	Flight No.	Launch site	Payload	Payload mass	Orbit	Customer	Launch outcome	Version Booster	Booster landing	Date	Time
0	1	CCAFS	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success\n	F9 v1.0B0003.1	Failure	4 June 2010	18:45
1	2	CCAFS	Dragon	0	LEO	NASA	Success	F9 v1.0B0004.1	Failure	8 December 2010	15:43
2	3	CCAFS	Dragon	525 kg	LEO	NASA	Success	F9 v1.0B0005.1	No attempt\n	22 May 2012	07:44
3	4	CCAFS	SpaceX CRS-1	4,700 kg	LEO	NASA	Success\n	F9 v1.0B0006.1	No attempt	8 October 2012	00:35
4	5	CCAFS	SpaceX CRS-2	4,877 kg	LEO	NASA	Success\n	F9 v1.0B0007.1	No attempt\n	1 March 2013	15:10
...	...	...	...	...	...	...	...	...	...	...	...
100	101	KSC	SpaceX CRS-21	2,972 kg	LEO	NASA	Success\n	F9 B5 Δ	Success	6 December 2020	16:17:08
101	102	CCSFS	SXM-7	7,000 kg	GTO	Sirius XM	Success\n	F9 B5 Δ	Success	13 December 2020	17:30:00
102	103	KSC	NROL-108	C	LEO	NRO	Success\n	F9 B5 Δ	Success	19 December 2020	14:00:00
103	104	CCSFS	Türksat 5A	3,500 kg	GTO	Türksat	Success\n	F9 B5	Success	8 January 2021	02:15
104	105	KSC	Starlink	15,600 kg	LEO	SpaceX	Success\n	F9 B5B1051.8	Success	20 January 2021	13:02

DATA WRANGLING

In this part, we performed some Exploratory Data Analysis (EDA) to find some patterns in the data and determine what would be the label for training supervised models.

In the data set, there were several different cases where the booster did not land successfully.
Sometimes a landing was attempted but failed due to an accident; for example, True Ocean means the mission outcome was successfully landed to a specific region of the ocean while False Ocean means the mission outcome was unsuccessfully landed to a specific region of the ocean. True RTLS means the mission outcome was successfully landed to a ground pad False RTLS means the mission outcome was unsuccessfully landed to a ground pad.True ASDS means the mission outcome was successfully landed on a drone ship False ASDS means the mission outcome was unsuccessfully landed on a drone ship.
In this section we mainly converted those outcomes into Training Labels with 1 means the booster successfully landed 0 means it was unsuccessful.

Use the method `.value_counts()` on the column `Outcome` to determine the number of `landing_outcomes` .Then assign it to a variable landing_outcomes.

```
# landing_outcomes = values on Outcome column
landing_outcomes = df['Outcome'].value_counts()
```

`True Ocean` means the mission outcome was successfully landed to a specific region of the ocean while `False Ocean` means the mission outcome was unsuccessfully landed to a specific region of the ocean. `True RTLS` means the mission outcome was successfully landed to a ground pad `False RTLS` means the mission outcome was unsuccessfully landed to a ground pad. `True ASDS` means the mission outcome was successfully landed to a drone ship `False ASDS` means the mission outcome was unsuccessfully landed to a drone ship. `None ASDS` and `None None` these represent a failure to land.

```
for i,outcome in enumerate(landing_outcomes.keys()):
    print(i,outcome)
```

```
0 True ASDS
1 None None
2 True RTLS
3 False ASDS
4 True Ocean
5 False Ocean
6 None ASDS
7 False RTLS
```

We create a set of outcomes where the second stage did not land successfully:

```
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
bad_outcomes
```

```
{'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

DATA WRANGLING

Using the `Outcome`, create a list where the element is zero if the corresponding row in `Outcome` is in the set `bad_outcome`; otherwise, it's one. Then assign it to the variable `landing_class`:

```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise

landing_class = []
for outcome in df.loc[:,'Outcome']:
    if outcome in bad_outcomes:
        landing_class.append(0)
    else:
        landing_class.append(1)
```

This variable will represent the classification variable that represents the outcome of each launch. If the value is zero, the first stage did not land successfully; one means the first stage landed Successfully

```
df['Class']=landing_class
df[['Class']].head(8)
```

	Class
0	0
1	0
2	0
3	0
4	0
5	0
6	1
7	1

```
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

We can use the following line of code to determine the success rate:

```
df["Class"].mean()
```

```
0.6666666666666666
```

DATA WRANGLING

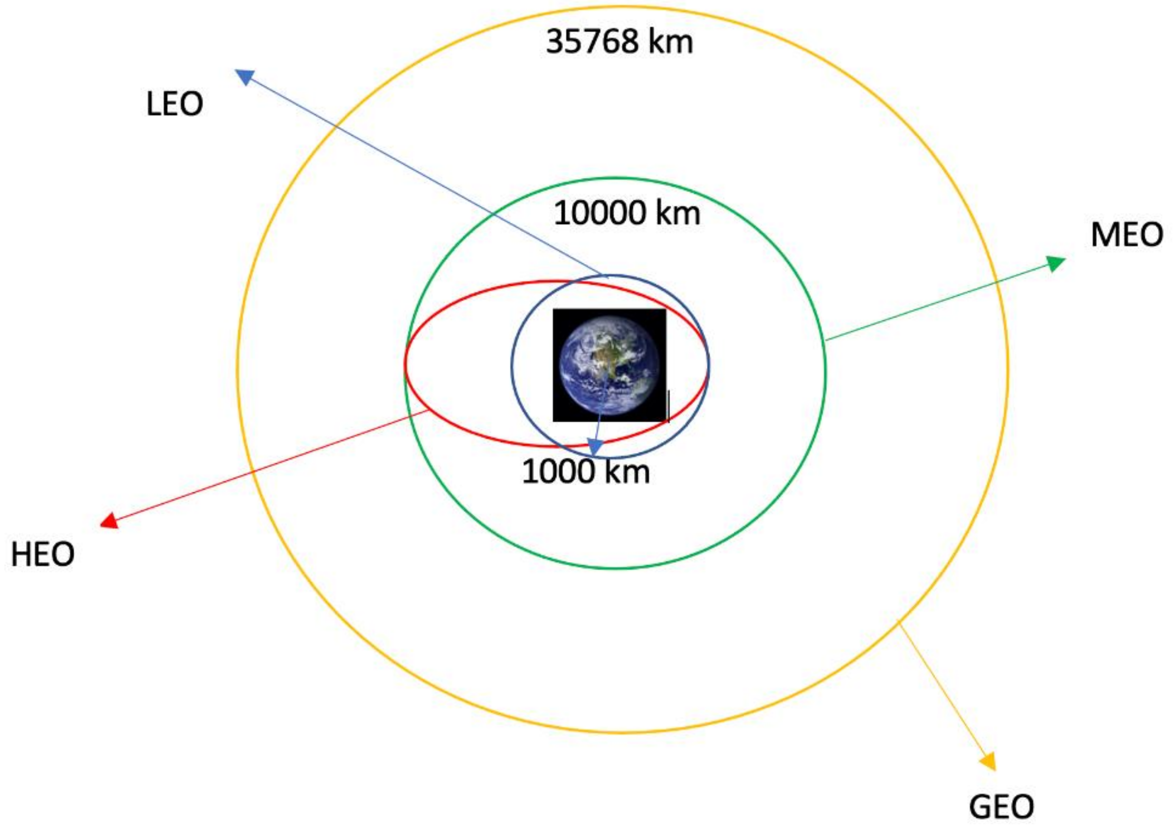
The data contained several Space X launch facilities: Cape Canaveral Space Launch Complex 40 **VAFB SLC 4E** , Vandenberg Air Force Base Space Launch Complex 4E **(SLC-4E)**, Kennedy Space Center Launch Complex 39A **KSC LC 39A** .The location of each Launch Is placed in the column LaunchSite.

Each launch aims to an dedicated orbit, and here are some common orbit types:

- **LEO**: Low Earth orbit (LEO)is an Earth-centred orbit with an altitude of 2,000 km (1,200 mi) or less (approximately one-third of the radius of Earth),[1] or with at least 11.25 periods per day (an orbital period of 128 minutes or less) and an eccentricity less than 0.25.[2] Most of the manmade objects in outer space are in LEO [1].
- **VLEO**: Very Low Earth Orbits (VLEO) can be defined as the orbits with a mean altitude below 450 km. Operating in these orbits can provide a number of benefits to Earth observation spacecraft as the spacecraft operates closer to the observation[2].
- **GTO** A geosynchronous orbit is a high Earth orbit that allows satellites to match Earth's rotation. Located at 22,236 miles (35,786 kilometers) above Earth's equator, this position is a valuable spot for monitoring weather, communications and surveillance. Because the satellite orbits at the same speed that the Earth is turning, the satellite seems to stay in place over a single longitude, though it may drift north to south," NASA wrote on its Earth Observatory website [3] .
- **SSO (or SO)**: It is a Sun-synchronous orbit also called a heliosynchronous orbit is a nearly polar orbit around a planet, in which the satellite passes over any given point of the planet's surface at the same local mean solar time [4] .
- **ES-L1** :At the Lagrange points the gravitational forces of the two large bodies cancel out in such a way that a small object placed in orbit there is in equilibrium relative to the center of mass of the large bodies. L1 is one such point between the sun and the earth [5] .
- **HEO** A highly elliptical orbit, is an elliptic orbit with high eccentricity, usually referring to one around Earth [6].
- **ISS** A modular space station (habitable artificial satellite) in low Earth orbit. It is a multinational collaborative project between five participating space agencies: NASA (United States), Roscosmos (Russia), JAXA (Japan), ESA (Europe), and CSA (Canada) [7]
- **MEO** Geocentric orbits ranging in altitude from 2,000 km (1,200 mi) to just below geosynchronous orbit at 35,786 kilometers (22,236 mi). Also known as an intermediate circular orbit. These are "most commonly at 20,200 kilometers (12,600 mi), or 20,650 kilometers (12,830 mi), with an orbital period of 12 hours [8]
- **HEO** Geocentric orbits above the altitude of geosynchronous orbit (35,786 km or 22,236 mi) [9]
- **GEO** It is a circular geosynchronous orbit 35,786 kilometres (22,236 miles) above Earth's equator and following the direction of Earth's rotation [10]
- **PO** It is one type of satellites in which a satellite passes above or nearly above both poles of the body being orbited (usually a planet such as the Earth [11]

DATA WRANGLING

Some are shown in the following plot:



DATA WRANGLING

Use the method `.value_counts()` to determine the number and occurrence of each orbit in the column `Orbit`

```
# Apply value_counts on Orbit column
df['Orbit'].value_counts()
```

```
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
```

EDA WITH VISUAL ANALYTICS

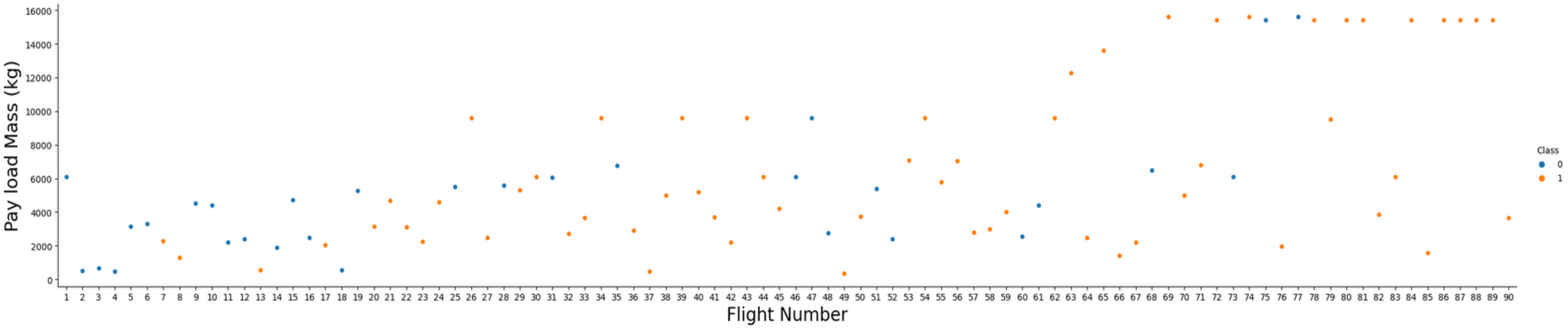
Visualization of the data helps us to immediately spot trends and correlations between the variables.
To reach the goal we made use of famous Python plotting libraries as "Matplotlib" and "Seaborn", let's see some of the charts we tested:

Flight Number VS Pay Load Mass (Kg)

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()
```



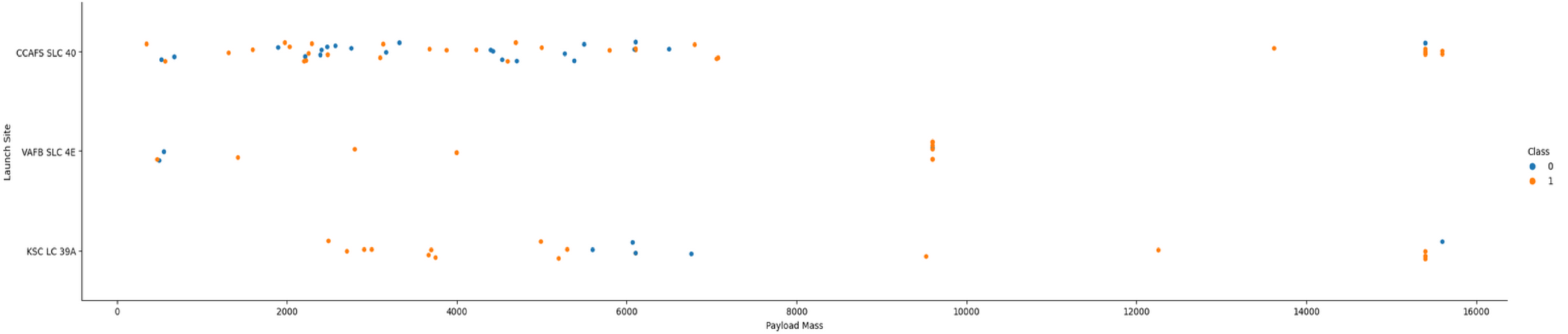
We see that different launch sites have different success rates. `CCAFS LC-40`, has a success rate of 60 %, while `KSC LC-39A` and `VAFB SLC 4E` has a success rate of 77%.

EDA WITH VISUAL ANALYTICS

Pay Load Mass (Kg) VS Launch Site

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

[9]:
```
# 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=5)
plt.xlabel('Payload Mass')
plt.ylabel('Launch Site')
plt.show()
```



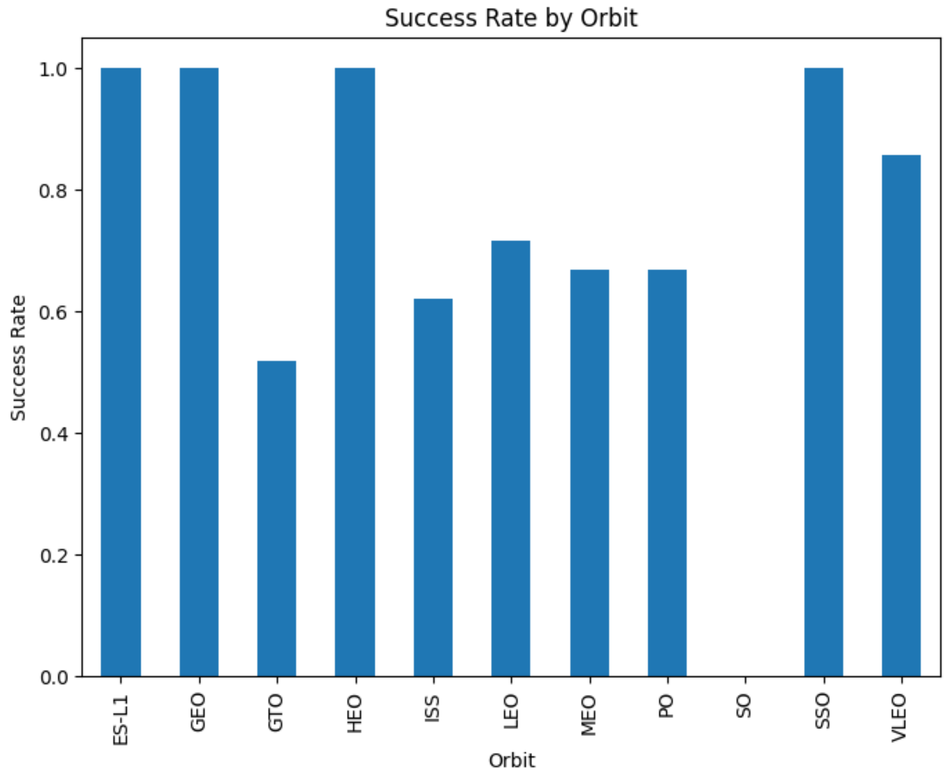
Now if you observe Payload Vs. Launch Site scatter point chart you will find for the VAFB-SLC launchsite there are no rockets launched for heavypayload mass(greater than 10000).

EDA WITH VISUAL ANALYTICS

Orbit Success Rate

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

```
# HINT use groupby method on Orbit column and get the mean of Class column
success_rate = df.groupby('Orbit')['Class'].mean()
success_rate.plot(kind='bar', figsize=(8, 6))
plt.title('Success Rate by Orbit')
plt.xlabel('Orbit')
plt.ylabel('Success Rate')
plt.show()
```

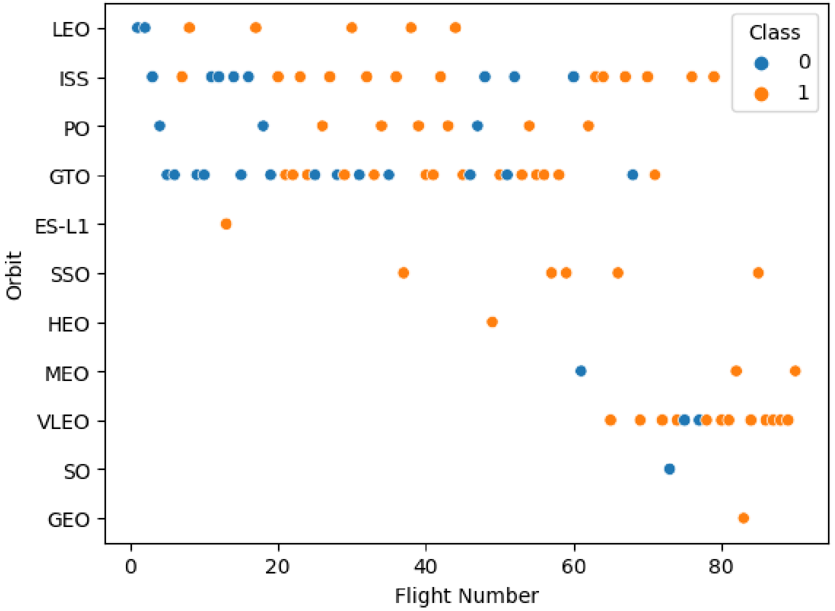


EDA WITH VISUAL ANALYTICS

Flight Number VS Orbit

For each orbit, we want to see if there is any relationship between FlightNumber 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.scatterplot(y="Orbit", x="FlightNumber", hue="Class", data=df)
plt.xlabel('Flight Number')
plt.ylabel('Orbit')
plt.show()
```



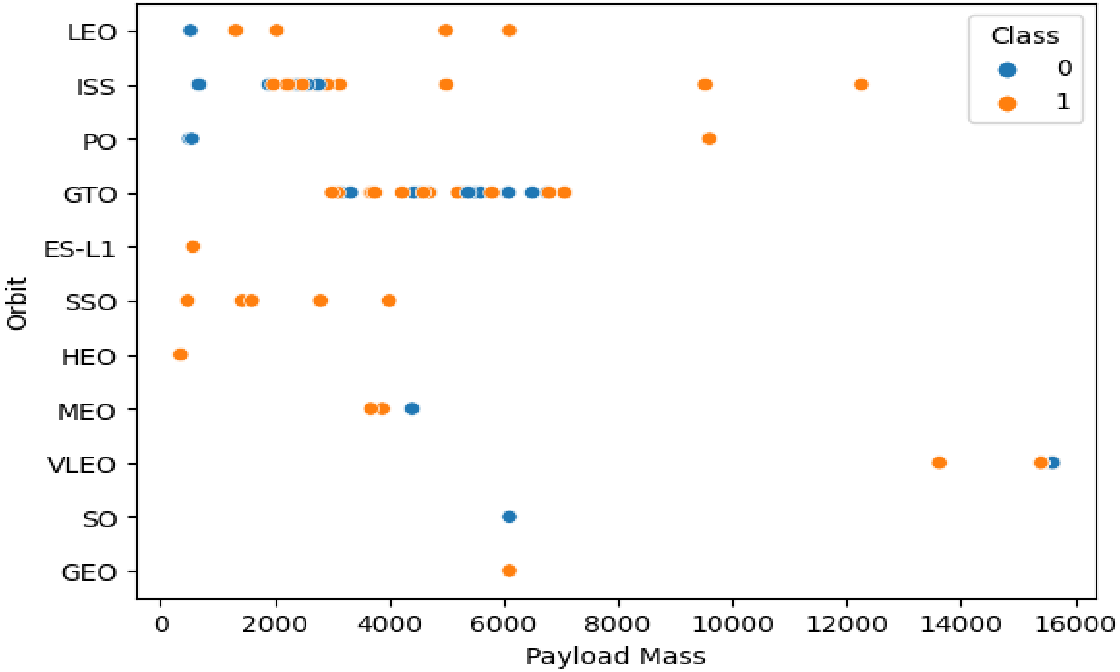
You should see that in the LEO orbit the Success appears related to the number of flights; on the other hand, there seems to be no relationship between flight number when in GTO orbit.

EDA WITH VISUAL ANALYTICS

Pay Load Mass (Kg) VS Orbit

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.scatterplot(y="Orbit", x="PayloadMass", hue="Class", data=df)
plt.xlabel('Payload Mass')
plt.ylabel('Orbit')
plt.show()
```



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

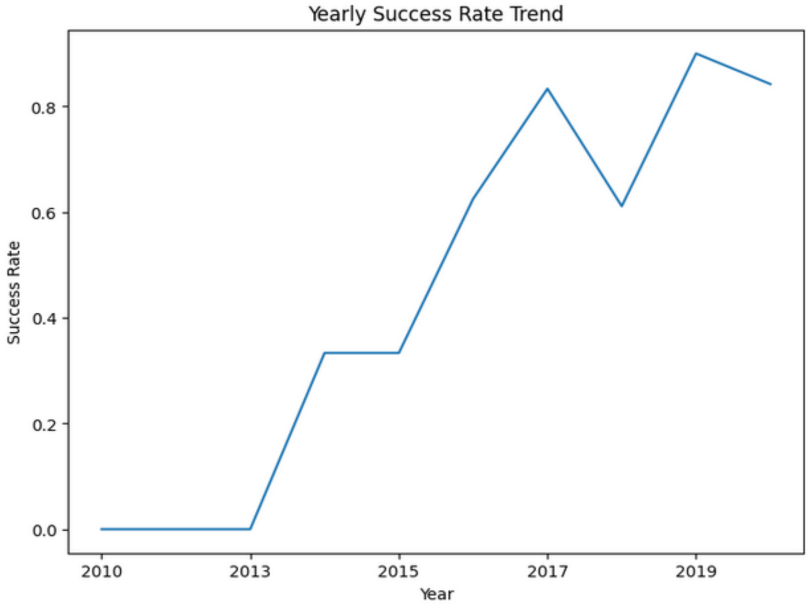
EDA WITH VISUAL ANALYTICS

Yearly Success Rate

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

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

avg_y= df.groupby(Extract_year(df))['Class'].mean()
avg_y.plot(kind='line', figsize=(8,6))
plt.title('Yearly Success Rate Trend')
plt.xlabel('Year')
plt.ylabel('Success Rate')
plt.show()
```



you can observe that the sucess rate since 2013 kept increasing till 2020

IBM Developer

SKILLS NETWORK

EDA WITH VISUAL ANALYTICS

At this stage we had enough statistics to work on and some preliminary insights about how each important variable would affect the success rate, we selected the features that were used in success prediction.

```
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

Finally we used the "get_dummies()" function to convert categorical data into binary values:

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.

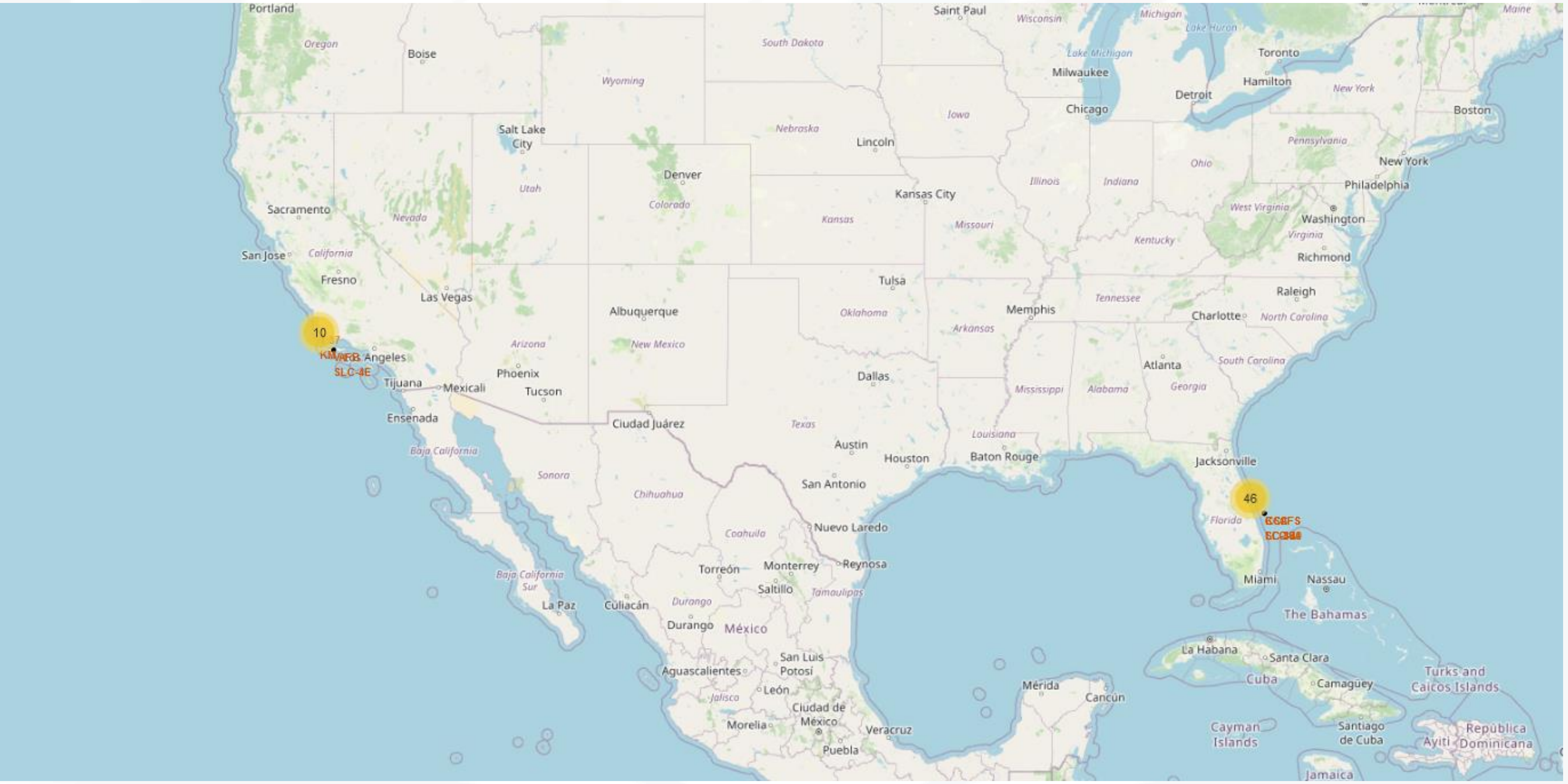
```
# HINT: Use get_dummies() function on the categorical columns
features_one_hot = pd.get_dummies(features[['Orbit','LaunchSite','LandingPad','Serial']])
features.drop(['Orbit','LaunchSite','LandingPad','Serial'], axis=1, inplace=True)
features_one_hot = pd.concat([features, features_encoded], axis=1)
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	Serial_B1059	Serial_B1060	Serial_B1062
0	1	6104.959412	1	False	False	False	1.0	0	0	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	0	0	0
2	3	677.000000	1	False	False	False	1.0	0	0	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	0	0	0
4	5	3170.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 80 columns

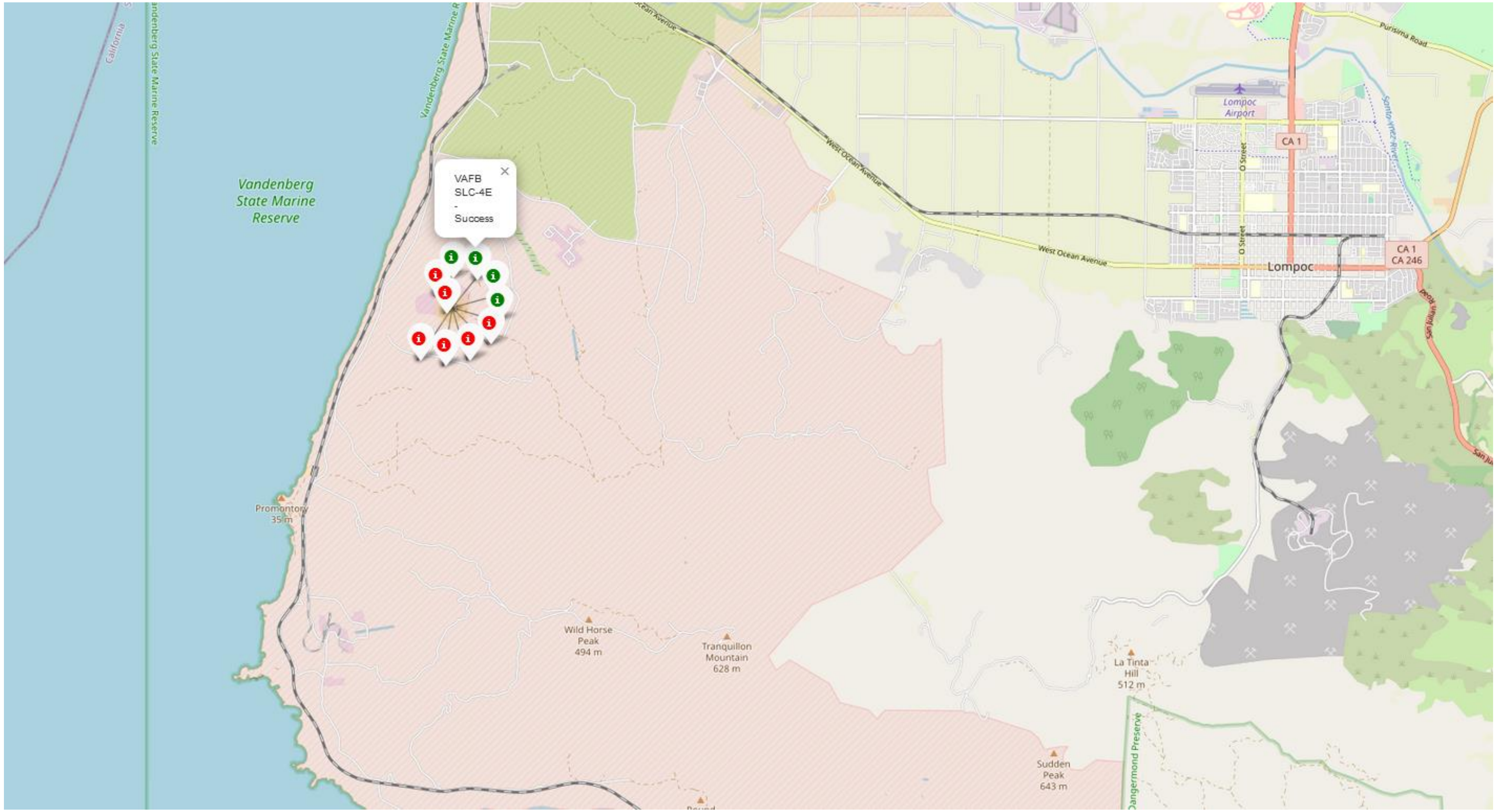
EDA WITH VISUAL ANALYTICS
Folium

To spot geographical insights for our model we used the Folium Python library and we noticed how almost all the launch sites are in proximity to the Equator line and very close to the coast line. Moreover Folium allowed us to better visualize and locate successful and failed launches for each launch site.



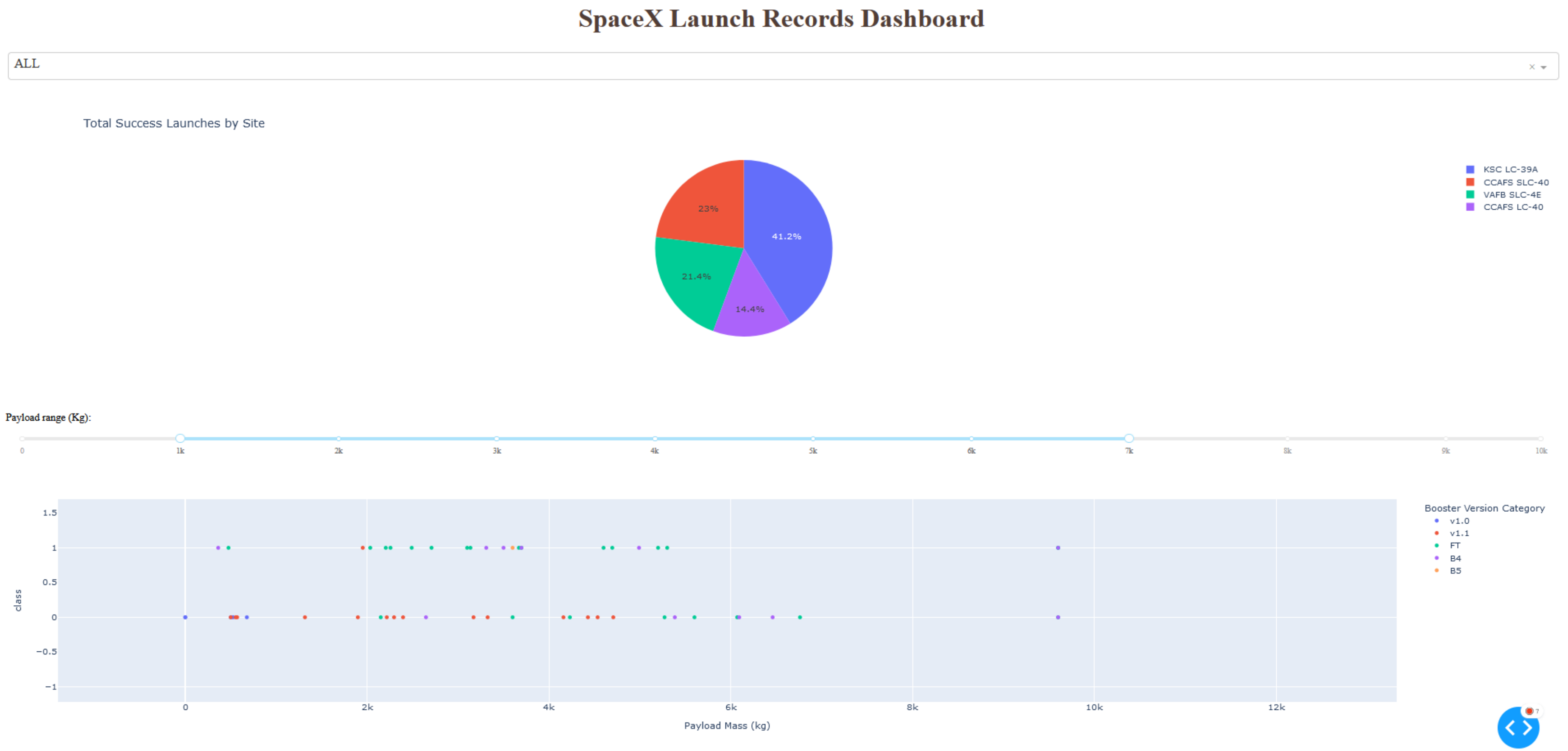
EDA WITH VISUAL ANALYTICS
Folium

EDA WITH VISUAL ANALYTICS
Folium



EDA WITH VISUAL ANALYTICS
Plotly Dash dashboard

Plotly Dash is a powerful tool to create dashboards and keep them running on a server, it gives us the possibility to rapidly switch to different charts accessing to a menu.
Here an example of the result:



IBM Developer

SKILLS NETWORK

EDA WITH SQL

At this stage we had enough statistics to work on and some preliminary insights about how each important variable would affect the success rate, we selected the features that were used in success prediction.

```
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

Finally we used the "get_dummies()" function to convert categorical data into binary values:

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.

```
# HINT: Use get_dummies() function on the categorical columns
features_one_hot = pd.get_dummies(features[['Orbit','LaunchSite','LandingPad','Serial']])
features.drop(['Orbit','LaunchSite','LandingPad','Serial'], axis=1, inplace=True)
features_one_hot = pd.concat([features, features_encoded], axis=1)
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	Serial_B1059	Serial_B1060	Serial_B1062
0	1	6104.959412	1	False	False	False	1.0	0	0	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	0	0	0
2	3	677.000000	1	False	False	False	1.0	0	0	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	0	0	0
4	5	3170.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 80 columns

EDA WITH SQL

SQL queries were performed on the dataframe to filter and extract data. Here are shown all of them:

Task 1

Display the names of the unique launch sites in the space mission

```
%sql SELECT launch_site FROM spacex GROUP BY launch_site
```

```
* ibm_db_sa://zxr33210:***@815fa4db-dc03-4c70-869a-a9cc13f33084.bs2io90108kqb1od8lcg.databases.appdomain.cloud:30367/bludb;security=SSL
Done.
```

launch_site
CCAFS LC-40
CCAFS SLC-40
KSC LC-39A
VAFB SLC-4E

Task 2

Display 5 records where launch sites begin with the string 'CCA'

```
%sql SELECT * FROM spacex WHERE launch_site LIKE 'CCA%' LIMIT 5;
```

```
* ibm_db_sa://zxr33210:***@815fa4db-dc03-4c70-869a-a9cc13f33084.bs2io90108kqb1od8lcg.databases.appdomain.cloud:30367/bludb;security=SSL
Done.
```

DATE	time_utc	booster_version	launch_site	payload	payload_mass_kg_	orbit	customer	mission_outcome	landing_outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Task 3

Display the total payload mass carried by boosters launched by NASA (CRS)

```
%sql SELECT SUM(payload_mass_kg_) FROM spacex WHERE customer LIKE '%NASA (CRS)%';
```

```
* ibm_db_sa://zxr33210:***@815fa4db-dc03-4c70-869a-a9cc13f33084.bs2io90108kqb1od8lcg.databases.appdomain.cloud:30367/bludb;security=SSL
Done.
```

1
48213

Task 4

Display average payload mass carried by booster version F9 v1.1

```
%sql SELECT AVG(payload_mass_kg_) AS "Average_Payload_Mass_F9v1_1" FROM spacex WHERE booster_version = 'F9 v1.1';
```

```
* ibm_db_sa://zxr33210:***@815fa4db-dc03-4c70-869a-a9cc13f33084.bs2io90108kqb1od8lcg.databases.appdomain.cloud:30367/bludb;security=SSL
Done.
```

average_payload_mass_f9v1_1
2928

Task 5

List the date when the first successful landing outcome in ground pad was acheived.

Hint:Use min function

```
%sql SELECT MIN(DATE) FROM spacex WHERE landing_outcome LIKE '%Success%';
```

```
* ibm_db_sa://zxr33210:***@815fa4db-dc03-4c70-869a-a9cc13f33084.bs2io90108kqb1od8lcg.databases.appdomain.cloud:30367/bludb;security=SSL
Done.
```

1
2015-12-22

EDA WITH SQL

Task 6

List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000

```
%sql SELECT booster_version, payload_mass_kg_, landing_outcome FROM spacex WHERE landing_outcome LIKE '%Success%' AND payload_mass_kg_ BETWEEN 4000 AND 6000;
```

```
 * ibm_db_sa://zxr33210:***@815fa4db-dc03-4c70-869a-a9cc13f33084.bs2io90108kqb1od8lcg.databases.appdomain.cloud:30367/bludb;security=SSL
Done.
```

booster_version	payload_mass_kg_	landing_outcome
F9 FT B1022	4696	Success (drone ship)
F9 FT B1026	4600	Success (drone ship)
F9 FT B1021.2	5300	Success (drone ship)
F9 FT B1032.1	5300	Success (ground pad)
F9 B4 B1040.1	4990	Success (ground pad)
F9 FT B1031.2	5200	Success (drone ship)
F9 B4 B1043.1	5000	Success (ground pad)
F9 B5 B1046.2	5800	Success
F9 B5 B1047.2	5300	Success
F9 B5 B1046.3	4000	Success
F9 B5 B1048.3	4850	Success
F9 B5 B1051.2	4200	Success
F9 B5B1060.1	4311	Success
F9 B5 B1058.2	5500	Success
F9 B5B1062.1	4311	Success

Task 7

List the total number of successful and failure mission outcomes

```
%sql SELECT COUNT(mission_outcome) AS "Successful_Missions", (SELECT COUNT(mission_outcome) AS "Failed_Missions" FROM spacex WHERE mission_outcome LIKE '%Fail%') FROM spacex WHERE mission_outcome LIKE '%Success%';
```

```
 * ibm_db_sa://zxr33210:***@815fa4db-dc03-4c70-869a-a9cc13f33084.bs2io90108kqb1od8lcg.databases.appdomain.cloud:30367/bludb;security=SSL
Done.
```

successful_missions	failed_missions
100	1

Task 8

List the names of the booster_versions which have carried the maximum payload mass. Use a subquery

```
%sql SELECT booster_version, payload_mass_kg_ FROM spacex WHERE (SELECT MAX(payload_mass_kg_) FROM spacex) ORDER BY payload_mass_kg_ DESC LIMIT 10;
```

```
 * ibm_db_sa://zxr33210:***@815fa4db-dc03-4c70-869a-a9cc13f33084.bs2io90108kqb1od8lcg.databases.appdomain.cloud:30367/bludb;security=SSL
Done.
```

booster_version	payload_mass_kg_
F9 B5 B1048.4	15600
F9 B5 B1051.6	15600
F9 B5 B1058.3	15600
F9 B5 B1060.2	15600
F9 B5 B1049.5	15600
F9 B5 B1051.4	15600
F9 B5 B1048.5	15600
F9 B5 B1056.4	15600
F9 B5 B1051.3	15600
F9 B5 B1049.4	15600

Task 9

List the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015

```
%sql SELECT DATE, booster_version, landing_outcome, launch_site FROM spacex WHERE DATE BETWEEN '2015-01-01' AND '2015-12-31';
```

```
 * ibm_db_sa://zxr33210:***@815fa4db-dc03-4c70-869a-a9cc13f33084.bs2io90108kqb1od8lcg.databases.appdomain.cloud:30367/bludb;security=SSL
Done.
```

DATE	booster_version	landing_outcome	launch_site
2015-01-10	F9 v1.1 B1012	Failure (drone ship)	CCAFS LC-40
2015-02-11	F9 v1.1 B1013	Controlled (ocean)	CCAFS LC-40
2015-03-02	F9 v1.1 B1014	No attempt	CCAFS LC-40
2015-04-14	F9 v1.1 B1015	Failure (drone ship)	CCAFS LC-40
2015-04-27	F9 v1.1 B1016	No attempt	CCAFS LC-40
2015-06-28	F9 v1.1 B1018	Precluded (drone ship)	CCAFS LC-40
2015-12-22	F9 FT B1019	Success (ground pad)	CCAFS LC-40

EDA WITH SQL

2010-06-04### Task 10

Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order

```
%sql SELECT landing_outcome, DENSE_RANK() OVER(ORDER BY COUNT(landing_outcome) DESC) Rank FROM spacex WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20' GROUP BY landing_outcome;
```

```
 * ibm_db_sa://zxr33210:***@815fa4db-dc03-4c70-869a-a9cc13f33084.bs2io90108kqb1od8lcg.databases.appdomain.cloud:30367/bludb;security=SSL
Done.
```

landing_outcome	RANK
No attempt	1
Failure (drone ship)	2
Success (drone ship)	2
Controlled (ocean)	3
Success (ground pad)	3
Failure (parachute)	4
Uncontrolled (ocean)	4
Precluded (drone ship)	5

PREDICTIVE ANALYSIS

The final step of our study is to build and test the models. For this reason, we needed to standardize the data and then split it into training and test data. Finally, we had to find the best hyperparameters for SVM, Classification Trees, and Logistic Regression and compare the results to determine which method performs best using the test data.

Standardize the data in `X` then reassign it to the variable `X` using the transform provided below.

```
# students get this
transform = preprocessing.StandardScaler()
```

```
transform.fit(X)
X = pd.DataFrame(X, columns=X.columns)
```

We split the data into training and testing data using the function `train_test_split`. The training data is divided into validation data, a second set used for training data; then the models are trained and hyperparameters are selected using the function `GridSearchCV`.

Use the function train_test_split to split the data X and Y into training and test data. Set the parameter test_size to 0.2 and random_state to 2. The training data and test data should be assigned to the following labels.

`X_train, X_test, Y_train, Y_test`

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
```

we can see we only have 18 test samples.

```
Y_test.shape
```

```
(18,)
```

PREDICTIVE ANALYSIS

Let's go through the creation and training of all the models:

LINEAR REGRESSION MODEL

TASK 4

Create a logistic regression object then create a GridSearchCV object `logreg_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

```
parameters = {'C': [0.01, 0.1, 1],
              'penalty': ['l2'],
              'solver': ['lbfgs']}
```

```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()
# create GridSearchCV object
logreg_cv = GridSearchCV(lr, parameters, cv=10)

# fit GridSearchCV object to the data
logreg_cv.fit(X_train, Y_train)
```

We output the `GridSearchCV` object for logistic regression. We display the best parameters using the data attribute `best_params_` and the accuracy on the validation data

```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 1, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8053571428571429
```

TASK 5

Calculate the accuracy on the test data using the method `score`:

```
lr_accuracy = logreg_cv.score(X_test, Y_test)
print("Accuracy on test data: {:.2f}%".format(lr_accuracy*100))
```

```
Accuracy on test data: 83.33%
```

SVC MODEL

Create a support vector machine object then create a `GridSearchCV` object `svm_cv` with cv - 10. Fit the object to find the best parameters from the dictionary `parameters`.

```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

```
n_estimators = 10
svm_cv  = OneVsRestClassifier(BaggingClassifier(SVC(kernel='linear', probability=True), max_samples=1.0 / n_estimators, n_estimators=n_estimators))
svm_cv.fit(X_train, Y_train)
proba = svm_cv.predict_proba(X)
```

```
C:\Users\leand\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\calibration.py:835: RuntimeWarning: invalid value encountered in multiply
  return (sample_weight * loss).sum()
```

```
print ("Bagging SVC", svm_cv.score(X,Y))
proba = svm_cv.predict_proba(X)
```

```
Bagging SVC 0.6888888888888889
```

svm_cv = GridSearchCV(svm, parameters, cv=10) svm_cv.fit(X_train, Y_train)

print("tuned hpyerparameters :(best parameters) ",svm_cv.best_params_) print("accuracy :",svm_cv.best_score_)

TASK 7

Calculate the accuracy on the test data using the method `score`:

```
svm_accuracy = svm_cv.score(X_test, Y_test)
print("Accuracy on test data: {:.2f}%".format(svm_accuracy*100))
```

```
Accuracy on test data: 72.22%
```

PREDICTIVE ANALYSIS

DECISION TREE MODEL

Create a decision tree classifier object then create a `GridSearchCV` object `tree_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

```
parameters = {'criterion': ['gini', 'entropy'],
     'splitter': ['best', 'random'],
     'max_depth': [2*n for n in range(1,10)],
     'max_features': ['auto', 'sqrt'],
     'min_samples_leaf': [1, 2, 4],
     'min_samples_split': [2, 5, 10]}

tree = DecisionTreeClassifier()
```

```
tree_cv = GridSearchCV(tree, parameters, cv=10)
tree_cv.fit(X_train, Y_train)
```

```
print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', 'max_depth': 10, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split
accuracy : 0.9035714285714287
```

TASK 9

Calculate the accuracy of tree_cv on the test data using the method `score`:

```
tree_accuracy = tree_cv.score(X_test, Y_test)
print("Accuracy on test data: {:.2f}%".format(tree_accuracy*100))
```

```
Accuracy on test data: 77.78%
```

KNN MODEL

Create a k nearest neighbors object then create a `GridSearchCV` object `knn_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

```
parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'p': [1,2]}

KNN = KNeighborsClassifier()
```

```
knn_cv = GridSearchCV(KNN, parameters, cv=10)
knn_cv.fit(X_train, Y_train)
```

```
GridSearchCV
estimator: KNeighborsClassifier
KNeighborsClassifier
```

```
print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
print("accuracy :",knn_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 3, 'p': 1}
accuracy : 0.6642857142857143
```

CONCLUSION

Linear Regression was found to be the best model for predicting the outcome of the first phase landing, and I conducted further tests to confirm these results.
We can observe that the "KSC LC-39 A" site has the highest success rate for launches, and that low-weighted payloads tend to perform better than heavier ones.
It is important to consider that SpaceX's launch success rate has been increasing each year as they continue to perfect their techniques.

```
accuracy_score Scores:
Linear Regression: 0.83
SVM: 0.72
Decision Tree: 0.78
KNN: 0.61

precision_score Scores:
Linear Regression: 0.80
SVM: 0.71
Decision Tree: 0.79
KNN: 0.73

recall_score Scores:
Linear Regression: 1.00
SVM: 1.00
Decision Tree: 0.92
KNN: 0.67

f1_score Scores:
Linear Regression: 0.89
SVM: 0.83
Decision Tree: 0.85
KNN: 0.70

_daal_roc_auc_score Scores:
Linear Regression: 0.75
SVM: 0.58
Decision Tree: 0.71
KNN: 0.58
```

From the results we can observe how Linear Regression Model performs better over the others.

Link to the GitHub Repository: https://github.com/sacrosK11/IBM---Space-X---Data-Science-Capstone-Project.git