IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

<Marcus A Bryant>
<2024-10-01>

Outline

- Executive Summary
- Introduction
- Methodology
- Results
- Conclusion
- Appendix

Executive Summary

- Summary of methodologies – Data Collection Via API, EDA With Data Visualization, EDA With SQL, Interactive Map with Folium, Dashboards with Plotly Dash, Predictive Analysis
- Summary of all results – EDA Results, Maps and Dashboards, Predictive Results

Introduction

- Project background and context
  - To determine the price of each launch.
  - Gather information about Space X and creating dashboards based off that information.
  - You will also determine if SpaceX will reuse the first stage.
  - Instead of using rocket science to determine if the first stage will land succesfully, I will train a machine learning model and use public information to predict if SpaceX.

- Problems you want to find answers
  - What makes information do I have on successful or failed landings?
  - Are there certain independent variables that will determine my dependent variable(Successful or Failed).
  - How can the influence of my independent variable give me the best result (Successful)?

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - I collected my data using two things: The SpaceX Rest API and Web Scrapping
- Perform data wrangling
  - One Hot Encoding And Beautiful Soup
- Perform exploratory data analysis (EDA) using visualization and SQL
  - SQL
- Perform interactive visual analytics using Folium and Plotly Dash
  - Folium and Plotly
- Perform predictive analysis using classification models
  - How to build, tune, evaluate classification models

Data Collection

- Describe how data sets were collected.
    - I collected data by using the Rest SpaceX API and WebScrapping. I received information about rockets, cost, and the types of lanuches(Successful or Failed)
- You need to present your data collection process use key phrases and flowcharts

API Call:

SpaceX API Makes A Call -> SpaceX API Returns JSON File -> Create an Empty DataFrame With Columns -> Create Another Data Frame And Export The JSON Data Into It -> Clean Date -> CONCAT DataFrame With The JSON Data With The Empty Data Frame With The Columns.

Web Scrapping:

HTML Response Using The URL -> Use BeatifulSoup To Extract Data -> Create DataFrame – Export Data

Data Collection – SpaceX API

Now let's start requesting rocket launch data from SpaceX API with the following URL:

```
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.imgbox.com/94/f2/NN6Ph45r_o.png","large":"https://images2.imgbox.com/5b/02/QcxHUb5V_o.png"},"reddit":{"campaign":null,"launch":null,"media":null,"recovery":null},"flickr":{"small":[],"original":[]},"presskit":null,"webcast":"https://www.youtube.com/watch?v=0a_00nJ_Y88","youtube_id":"0a_00nJ_Y88","article":"https://www.space.com/2196-spacex-inaugural-falcon-1-rocket-lost-launch.html","wikipedia":"https://en.wikipedia.org/wiki/DemoSat"},"static_fire_date_utc":"2006-03-17T00:00:00.000Z","static_fire_date_unix":1142553600,"net":false,"window":0,"rocket":"5e9d0d95eda69955f709d1eb","success":false,"failures":[{"time":33,"altitude":null,"reason":"merlin engine failure"}],"details":"Engine failure at 33 seconds and loss of vehicle","crew":[],"ships":[],"capsules":[],"payloads":["5eb0e4b5b6c3bb0006eeb1e1"],"launchpad":"5e9e4502f5090995de566f8
```

You will notice that a lot of the data are IDs. For example the rocket column has no information about the rocket just an identification number.

We will now use the API again to get information about the launches using the IDs given for each launch. Specifically we will be using columns `rocket`, `payloads`, `launchpad`, and `cores`.

```
# Lets take a subset of our dataframe keeping only the features we want and the flight number, and date_utc.
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]

# We will remove rows with multiple cores because those are falcon rockets with 2 extra rocket boosters and rows that have
data = data[data['cores'].map(len)==1]
data = data[data['payloads'].map(len)==1]

# Since payloads and cores are lists of size 1 we will also extract the single value in the list and replace the feature.
data['cores'] = data['cores'].map(lambda x : x[0])
data['payloads'] = data['payloads'].map(lambda x : x[0])

# We also want to convert the date_utc to a datetime datatype and then extracting the date leaving the time
data['date'] = pd.to_datetime(data['date_utc']).dt.date

# Using the date we will restrict the dates of the launches
data = data[data['date'] <= datetime.date(2020, 11, 13)]
```

Task 1: Request and parse the SpaceX launch data using the GET request

To make the requested JSON results more consistent, we will use the following static response object for this project:

```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/AP
```

We should see that the request was successfull with the 200 status response code

```
response.status_code
```

```
200
```

Now we decode the response content as a Json using `.json()` and turn it into a Pandas dataframe using `.json_normalize()`

```
# 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
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	[]	[]

```
BoosterVersion
```

Now, let's apply `getBoosterVersion` function method to get the booster version

```
# Call getBoosterVersion
getBoosterVersion(data)
```

the list has now been update

```
BoosterVersion[0:5]
```

```
['Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 9']
```

we can apply the rest of the functions here:

```
# Call getLaunchSite
getLaunchSite(data)
```

```
# Call getPayloadData
getPayloadData(data)
```

```
# Call getCoreData
getCoreData(data)
```

Finally lets construct our dataset using the data we have obtained. We we combine the columns into a dictionary.

```
launch_dict = {'FlightNumber': list(data['flight_number']),
'Date': list(data['date']),
'BoosterVersion':BoosterVersion,
'PayloadMass':PayloadMass,
'Orbit':Orbit,
'LaunchSite':LaunchSite,
'Outcome':Outcome,
'Flights':Flights,
'GridFins':GridFins,
'Reused':Reused,
'Legs':Legs,
'LandingPad':LandingPad,
'Block':Block,
'ReusedCount':ReusedCount,
'Serial':Serial,
'Longitude': Longitude,
'Latitude': Latitude}
```

Then, we need to create a Pandas data frame from the dictionary launch_dict.

Data Collection - Scraping

TASK 1: Request the Falcon9 Launch Wiki page from its URL

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
data = 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
soup = BeautifulSoup(data, "html.parser")
```

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

```
# Use soup.title attribute
print(soup.title)
```

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

TASK 2: Extract all column/variable names from the HTML table header

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 = soup.find_all('table')
```

TASK 3: Create a data frame by parsing the launch HTML tables

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']=[]
```

After you have fill in the parsed launch record values into `launch_dict`, you can create a dataframe from it.

```
#df=pd.DataFrame({ key:pd.Series(value) for key, value in launch_dict.items() })
headings = []
for key,values in dict(launch_dict).items():
    if key not in headings:
        headings.append(key)
    if values is None:
        del launch_dict[key]

def pad_dict_list(dict_list, padel):
    lmax = 0
    for lname in dict_list.keys():
        lmax = max(lmax, len(dict_list[lname]))
    for lname in dict_list.keys():
        ll = len(dict_list[lname])
        if ll < lmax:
            dict_list[lname] += [padel] * (lmax - ll)
    return dict_list

pad_dict_list(launch_dict,0)

df = pd.DataFrame.from_dict(launch_dict)
df.head()
```

	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	<generator object Tag._all_strings at 0x7f5a62...	Success\n	F9 v1.0B0003.18	Failure	4 June 2010	18:45
1	2	CCAFS	Dragon	0	LEO	<generator object Tag._all_strings at 0x7f5a64...	Success	F9 v1.0B0004.18	Failure	8 December 2010	15:43
2	3	CCAFS	Dragon	525 kg	LEO	<generator object Tag._all_strings at 0x7f5a62...	Success	F9 v1.0B0005.18	No attempt\n	22 May 2012	07:44
3	4	CCAFS	SpaceX CRS-1	4,700 kg	LEO	<generator object Tag._all_strings at 0x7f5a62...	Success\n	F9 v1.0B0006.18	No attempt	8 October 2012	00:35
4	5	CCAFS	SpaceX CRS-2	4,877 kg	LEO	<generator object Tag._all_strings at 0x7f5a62...	Success\n	F9 v1.0B0007.18	No attempt\n	1 March 2013	15:10

Code Link:

Data Wrangling

Use the method `value_counts()` on the column `LaunchSite` to determine the number of launches on each site

```
[10]: # Apply value_counts() on column LaunchSite
      df['LaunchSite'].value_counts()
```

```
[10]: LaunchSite
      CCAFS SLC 40    55
      KSC LC 39A      22
      VAFB SLC 4E     13
      Name: count, dtype: int64
```

TASK 3: Calculate the number and occurence of mission outcome of the orbits

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

```
[12]: # landing_outcomes = values on Outcome columnsd
      landing_outcomes = df['Outcome'].value_counts()
      landing_outcomes
```

```
[12]: Outcome
      True ASDS      41
      None None      19
      True RTLS      14
      False ASDS      6
      True Ocean      5
      False Ocean     2
      None ASDS       2
      False RTLS      1
      Name: count, dtype: int64
```

TASK 2: Calculate the number and occurrence of each orbit

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

```
[11]: df['Orbit'].value_counts()
```

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

TASK 4: Create a landing outcome label from Outcome column

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` :

```
[17]: # landing_class = 0 if bad_outcome
      # landing_class = 1 otherwise
      landing_class = []
      for outcome in df['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

```
[18]: 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

```
[19]: df.head(5)
```

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False

EDA with Data Visualization

- Graphs:
  - Scatter Graphs:
    - Flight # Vs Payload
    - Flight # Vs Launch Site
    - Payload vs Launch Site
    - F light # Vs Orbit
  - Bar Graphs:
    - Success Rate Vs Orbit
  - Line Graphs:
    - Success Rate Vs Year

EDA with SQL

- SQL Used:
  - Total Number of successful and failure missions
  - The names of the unique sites
  - The dates of landing outcome
  - Using Rank system to count # of successful landing between a given point
  - Average Payload Mass Per Booster
  - Total Payload Mass Per Booster
  - Names of Booster which have success

Build an Interactive Map with Folium

- I used Folium map to center on the NASA Space Center
- (Folium.Circle, Folium.map.Marker) - To draw a red circle at the Nasa Space Center
- (Folium.Circle.foloium.map.Marker,folium.features) - To Draw Red Circles At Each Site
- (Folium.plugins.MarkerCluster) - To Group Points In A Cluster
- (Folium.map.Marker, folium.Icon) - To Show Successful and Failed Landings
- (Folium.map.Marker,folium.PolyLine,folium.features) - To show each distance for each site.
- Using this code allows me to view the full picture of my data and to have a better understanding of where things are on my map.

Build a Dashboard with Plotly Dash

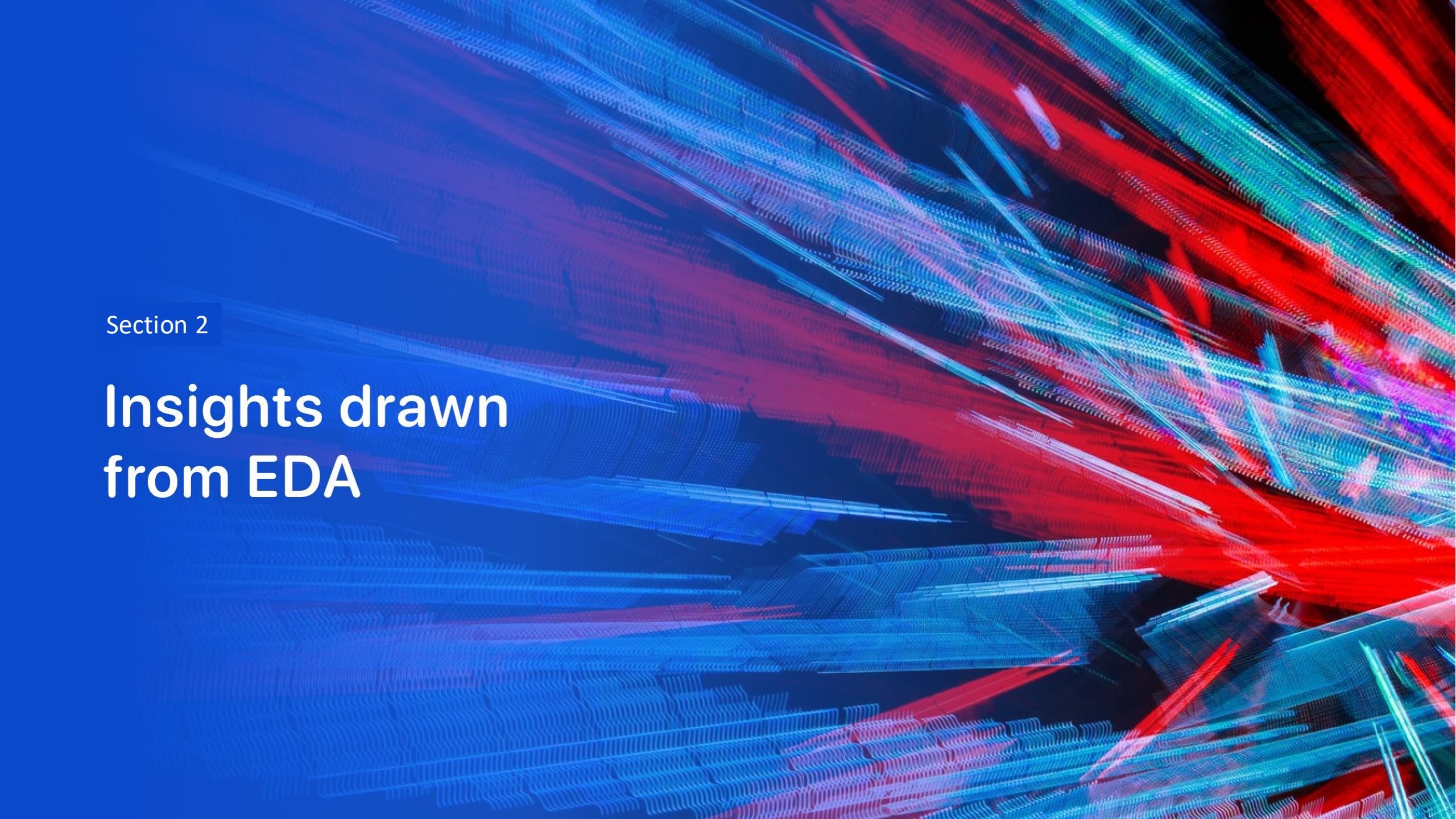
- I added multiple dashboards that range from pie charts and scatter plots.
- (dash_core_components.dropdown) - Allows myself to choose which site I would like to view.
- (plotly.express.pie) - Creates my Pie Chart to review the total number of success and failed per the launch site
- (Plotly.express.scatter) - Creates my Scatterplit to review the relationship between my two variables

Predictive Analysis (Classification)

- I built my classification model using four steps: Data Prep, Model Prep, Model Evaluation and Model Comparison.
- Data Prep– Loading My Dataset , Normalizing The Data. Splitting data into test and training sets. Fitting the data
- Model Prep – Selecting a MLA, Use GridSearch, Train my model using the data from data prep stage
- Model Evaluation – MSE, R2, Accuracy
- Model Comparison – Determine which model has the best accuracy for what I want to accomplish.

Results

- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results

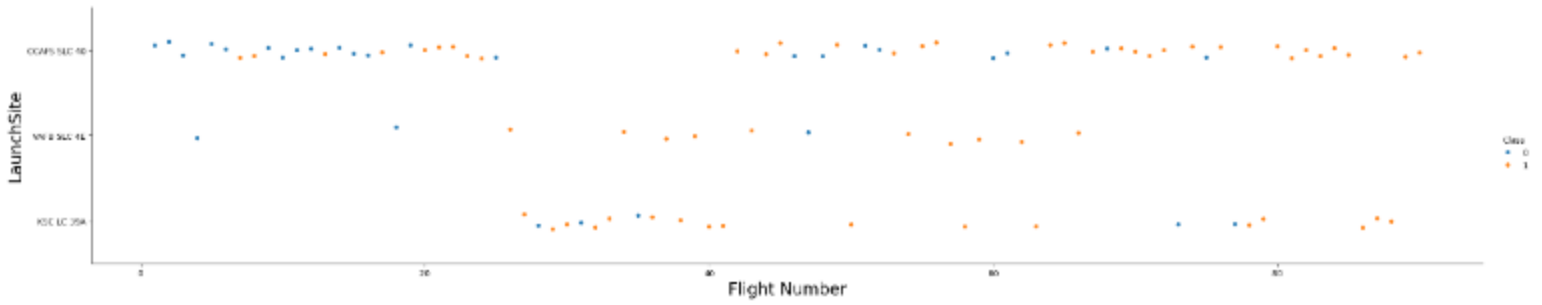

Section 2
Insights drawn from EDA

Flight Number vs. Launch Site

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'`

[5]:
```
# Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to
sns.catplot(y="LaunchSite", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("LaunchSite",fontsize=20)
plt.show()
```

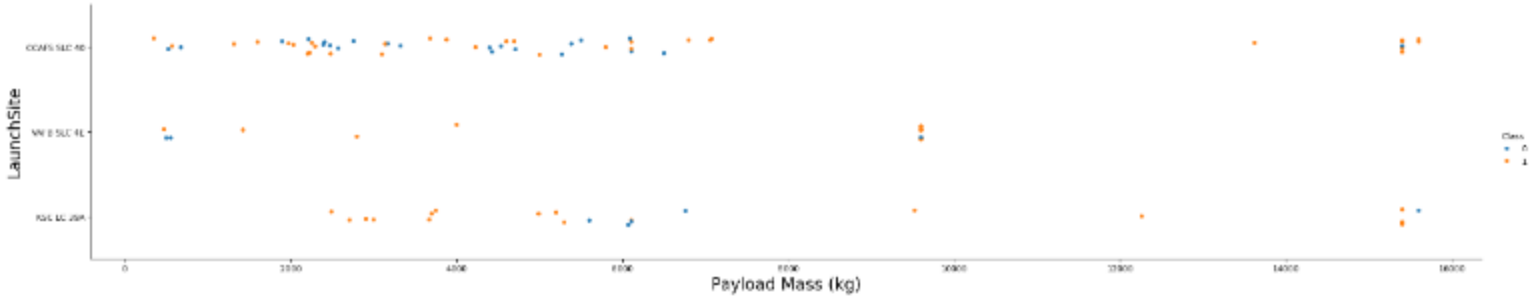


Payload vs. Launch Site

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.

[6]:
```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch site, and hu
sns.catplot(y="LaunchSite", x="PayloadMass", hue="Class", data=df, aspect = 5)
plt.xlabel("Payload Mass (kg)",fontsize=20)
plt.ylabel("LaunchSite",fontsize=20)
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).

Success Rate vs. Orbit Type

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

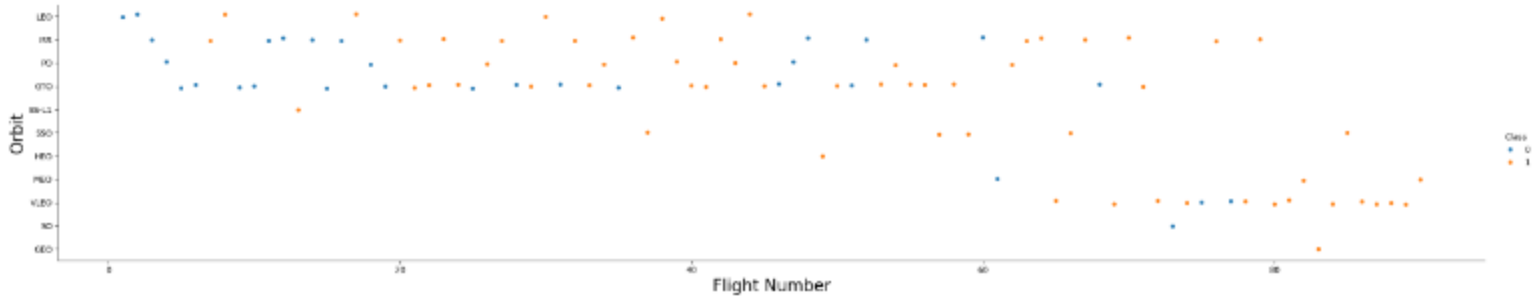
```
# HINT use groupby method on Orbit column and get the mean of Class column
df.groupby(['Orbit']).mean()['Class'].plot(kind='bar')
plt.xlabel("Orbit",fontsize=20)
plt.ylabel("Sucess Rate",fontsize=20)
plt.show()
```

Flight Number vs. Orbit Type

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.

[9]:
```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the
sns.catplot(y="Orbit", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```



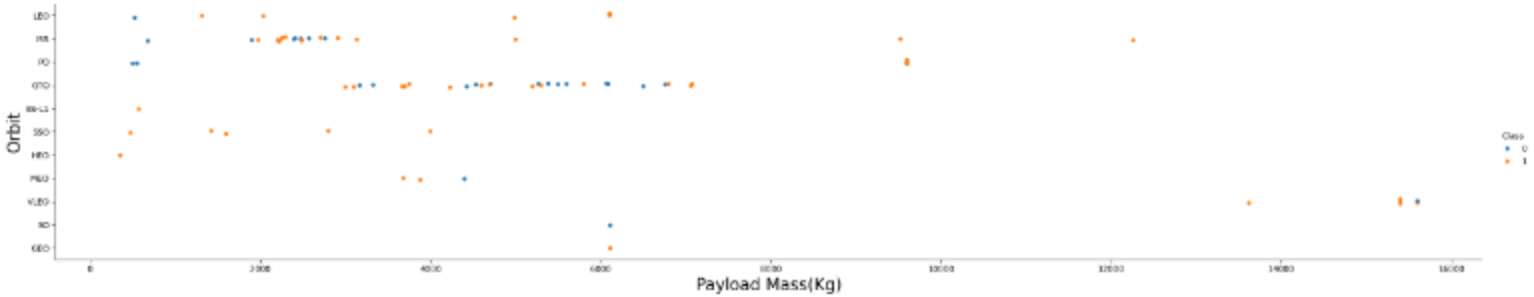
You can observe that in the LEO orbit, success seems to be related to the number of flights. Conversely, in the GTO orbit, there appears to be no relationship between flight number and success.

Payload vs. Orbit Type

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

[10]:
```
# Plot a scatter point chart with x axis to be Payload Mass and y axis to be the Orbit, and hue to be the
sns.catplot(y="Orbit", x="PayloadMass", hue="Class", data=df, aspect = 5)
plt.xlabel("Payload Mass(Kg)",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
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.

Launch Success Yearly Trend

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:

[13]:
```
# 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()
```

[13]:

	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

All Launch Site Names

- SELECT DISTINCT "launch_site" FROM SPACEXTBL
- Sites:
  - CCAFS LC-40
  - VAFD SLC-4E
  - KSC LC-39A
  - CCAFS SLC-40

Launch Site Names Begin with 'CCA'

- Find 5 records where launch sites begin with `CCA`
- SELECT * FROM SPACEXTBL WHERE "LAUNCH_SITE" LIKE '%CCA%' LIMIT 5

4	5	3170.000000	GTO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B1004
5	6	3325.000000	GTO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B1005
6	7	2296.000000	ISS	CCAFS SLC 40	1	False	False	True	NaN	1.0	0	B1006
7	8	1316.000000	LEO	CCAFS SLC 40	1	False	False	True	NaN	1.0	0	B1007
8	9	4535.000000	GTO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B1008

Total Payload Mass

- Calculate the total payload carried by boosters from NASA
- Present your query result with a short explanation here

Average Payload Mass by F9 v1.1

- Calculate the average payload mass carried by booster version F9 v1.1
- Present your query result with a short explanation here

First Successful Ground Landing Date

- Find the dates of the first successful landing outcome on ground pad
- Present your query result with a short explanation here

Successful Drone Ship Landing with Payload between 4000 and 6000

- List the names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000

- Present your query result with a short explanation here

Total Number of Successful and Failure Mission Outcomes

- Calculate the total number of successful and failure mission outcomes
- Present your query result with a short explanation here

Boosters Carried Maximum Payload

- List the names of the booster which have carried the maximum payload mass
- Present your query result with a short explanation here

2015 Launch Records

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

- Present your query result with a short explanation here

Rank Landing Outcomes Between 2010-06-04 and 2017-03-20

- 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

- Present your query result with a short explanation here

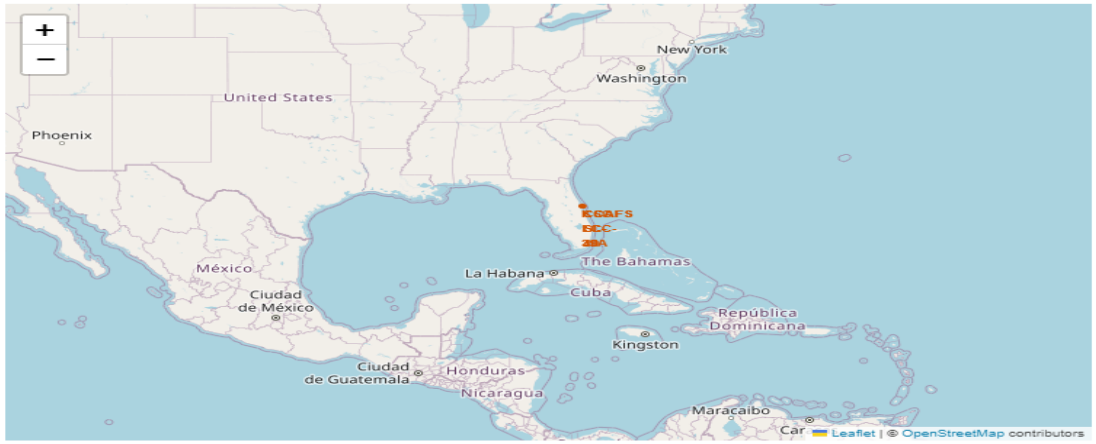
Section 3

Launch Sites Proximities Analysis

<Folium Map Screenshot 1>

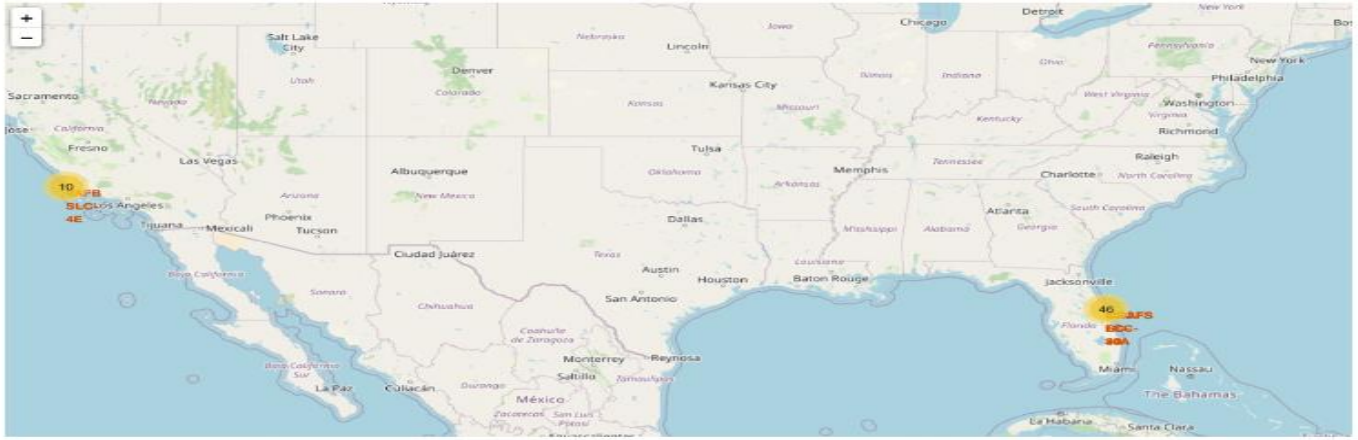
```
[9]: # Initial the map
site_map = folium.Map(location=nasa_coordinate, zoom_start=5)
# For each launch site, add a Circle object based on its coordinate (Lat, Long) values. In addition, add L
for idx,row in launch_sites_df.iterrows():
    circle = folium.Circle([row['Lat'],row['Long']], radius=1000, color='#d35400', fill=True).add_child(fo
    marker = folium.map.Marker(
        [row['Lat'],row['Long']],
        # Create an icon as a text label
        icon=DivIcon(
            icon_size=(20,20),
            icon_anchor=(0,0),
            html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % row['Launch Site'],
            )
        )
    site_map.add_child(circle)
    site_map.add_child(marker)
site_map
```

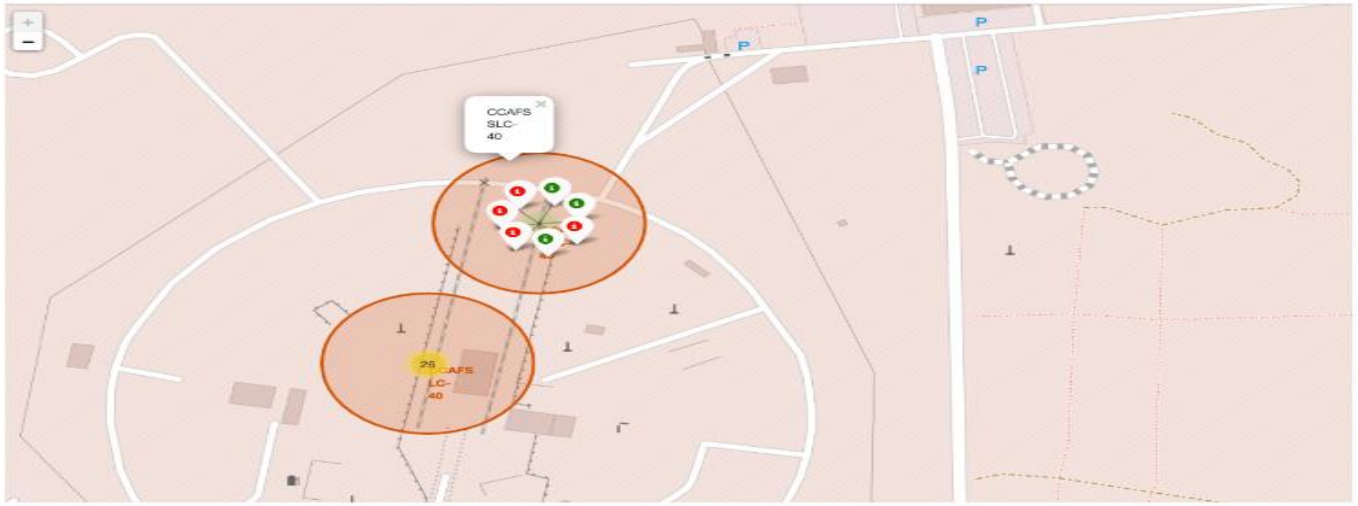
[9]:



All Of The Sites Are Located
In Florida

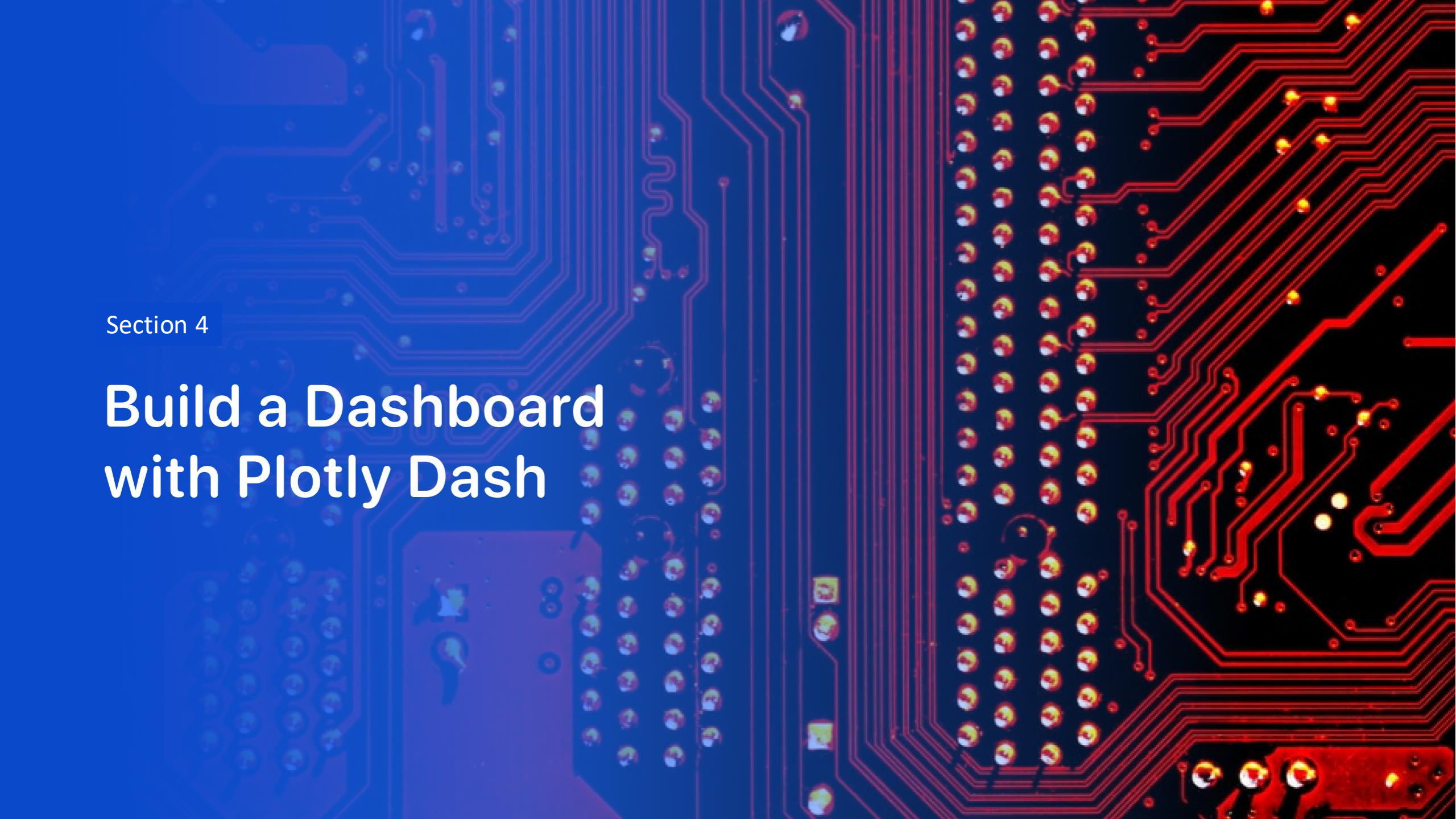
<Folium Map Screenshot 2>





From the color-labeled markers in marker clusters, you should be able to easily identify which launch sites have relatively high success rates

<Folium Map Screenshot 3>


Section 4
Build a Dashboard with Plotly Dash

<Dashboard Screenshot 1>

- Replace <Dashboard screenshot 1> title with an appropriate title
- Show the screenshot of launch success count for all sites, in a piechart
- Explain the important elements and findings on the screenshot

<Dashboard Screenshot 2>

- Replace <Dashboard screenshot 2> title with an appropriate title
- Show the screenshot of the piechart for the launch site with highest launch success ratio
- Explain the important elements and findings on the screenshot

<Dashboard Screenshot 3>

- Replace <Dashboard screenshot 3> title with an appropriate title
- Show screenshots of Payload vs. Launch Outcome scatter plot for all sites, with different payload selected in the range slider
- Explain the important elements and findings on the screenshot, such as which payload range or booster version have the largest success rate, etc.

Section 5

Predictive Analysis (Classification)

Classification Accuracy

- Visualize the built model accuracy for all built classification models, in a bar chart

- Find which model has the highest classification accuracy

Confusion Matrix

- Show the confusion matrix of the best performing model with an explanation

Conclusions

- Point 1
- Point 2
- Point 3
- Point 4
- …

Appendix

- Include any relevant assets like Python code snippets, SQL queries, charts, Notebook outputs, or data sets that you may have created during this project

Thank you!