Winning Space Race with Data Science

Lukas Mendes
11/09/2021

Outline

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

Executive Summary

In this presentation I want to answer questions regarding the success or failure of rocket launch, specifically, Space X's Falcon 9 rockets.

Is there a way we can improve the success percentage of a rocket landing correctly, or even making predictions about the outcome landing of rockets through data?

In this presentation we will see that yes, it is possible!

Let's see how the magic of data science works.

Introduction

Qustions:

- What factors can influence the success or failure of a rocket mission?
- How to discover factors that really influence the result?
- Is there a way we can improve the success percentage of a rocket landing correctly?
- Is there a way we can make predictions about the outcome landing through data?

Data Science:

Data is like a raw gem. Data can often be a mess, and we can often get lost trying to make sense of it.

But data often hide great treasures, and it takes an adventurer to go after them.

In this analogy, the adventurer is a Data Scientist. The data scientist is the one who has the right tools to explore data, which is like treasure maps.

In this analogy, the adventurer is a Data Scientist. The data scientist is the one who has the right tools to explore data, which is like treasure maps.

It is necessary to explore alternative paths, understand the best tools for the job, perform analysis, and then go on discovering the secrets of the map, or insights into the data.

As an adventurer, having the right knowledge and tools is not enough. It takes determination, curiosity, passion and to be a successful data scientist, this is critical. I will be the data scientist on this project, and together we will get our treasure, which is nothing more than answering the questions above.

The Process:

In this project we have access to data provided by Space X itself, with the records of several Falcon 9 rocket missions, which contains the results of the missions, the version of the rocket, whether the rocket landed successfully or not, among other information.

We will make a series of procedures to clean the data and sort it. Then we will analyze this data statistically and with graphics and go through the insights that we will have.

Finally, we will see the results of a machine learning model that gives us a prediction whether a rocket will land successfully or not.

The entire process was done in Python through Jupyter Notebooks.

It will be quite a journey, I hope you enjoy it as much as I do!

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:

SpaceX has an API that provides data from different areas. We collected data from the Falcon rockets, which is our focus on this project.

Through Python's Requests library, we make a request to receive the data. We transform the received data into a dataframe through the pandas library.

The result is a dataset with several records and each one containing the following informations about the mission:

Flight Number, Date, Booster Version, Payload Mass, Orbit, Launch Site, Outcome, Flights, Grid Fins, Reused, Legs, Landing Pad, Block, Reused Count, Serial, Longitude and Latitude.

- Data Wrangling:

In this step we start cleaning the dataset, and doing pre analysis.

First we remove the date of the missions from the dataset as it is irrelevant to our purpose and we also handle null values, that is, records with blank information.

Finally we created a new column called 'Class' that summarizes the outcomes to 1 or 0, with 1 being for success and 0 for landing failure.

With that we were able to calculate the average of the results and we saw that the average of overall success in the landing outcomes is 66%.

- Exploratory Data Analysis (EDA) using visualization and SQL:

After creating a dataset with the data already modified, we inserted it in a relational database (IBM Db2) to do some analysis, and custom queries through SQL queries. We got a lot of useful information here like the heavier Falcon 9 versions, the types of crashes when the landing didn't work, and so on.

Then we went back to python and used the matplotlib and plotly libraries to see graphs showing the relationship between variables and which of them were relevant or influencing in the outcome landing.

For example, we use a bar graph to visually check if there is any relationship between success rate and orbit type. We use the scatter chart, histogram, bar charts, among others.

We also collect some metrics about some variables with dispersion and position measurements, like mean, median, standard deviation, etc.

- Interactive visual analytics with Folium and Plotly Dash:

In this step we did a very interactive analysis.

We use the Folium library that allows us to create worldwide graphics of different types. We used a map that started in the United States.

Then we were informing the latitude and longitude of each launch site used in the Falcon 9 rocket missions and marking it on the map. Then we mark within each launch site the number of missions in each of them and each number has a color representing a successful landing or failure. Finally, we set the distance from a launch site to the ocean, as launch sites are usually close to the ocean.

After that we created a web application with the Dash library.

We created an interactive environment with graphics for queries like a pie chart with all launch sites and the percentage of success in each one of them. You can also view a pie chart and a scatter chart for each launch site individually.

- # Predictive analysis using a classification model:

This was the final step, where we built a classification model that based on the dataset information can tell us if a new rocket will succeed when landing or not.

We first need to provide real and historical information (which has already happened) to "train" the model, after training we test the model to see how it does by predicting information that it has never seen, that was not in the dataset it used to train. In this process we use a tool that takes a part of the dataset for testing and the rest is used to train the model.

At the end we compared the models to see which had the highest accuracy rate.

We make an evaluation of several classification algorithms with the GridSearchCV library to then find the best one for our case study.

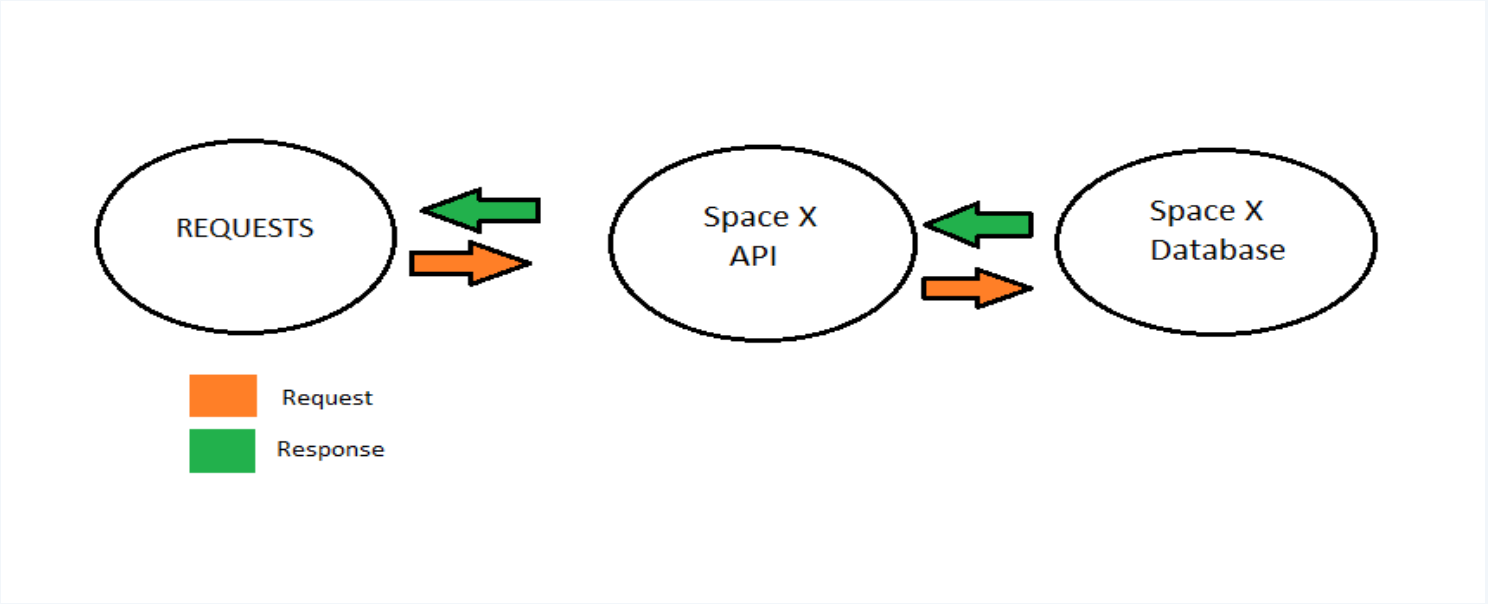
The tested algorithms were:

- **Logistic Regression**
- **Decision Tree**
- **Support Vector Machine**
- **KNeighbors**

Data Collection

Request Library:

The process for collecting the data was quite simple. Thanks to a Space X API that allows us to make a simple request (Request) via code with Python's Requests library. The information is returned to us via HTML. See in the flowchart below.



We convert the response into a JSON file and then convert it into a dataframe with Pandas.

Web Scrapping:

We don't always have this facility to collect data with an API. Congratulations to Space X for this tool that facilitates the work of researchers around the world. It's not just for the ease, but also because we collect reliable data, because we get them through Space X itself.

Well, and for those cases where we have to collect data on our own?

There is a technique called web scraping which is nothing more than extracting data from web pages.

For example, there's a Wikipedia page with the same data we've collected through the Space X API.

See the table at this link:

List of Falcon 9 and Falcon Heavy launches

To carry out this process, you need to use a Python library called Beautiful Soup.

First we use the Requests library to get the HTML code of the page in question. Then we pass the data to a BeautifulSoup object. With BeautifulSoup we can easily extract the information we need from the HTML code of the page.

With that, we extract the table with the Falcon 9 mission data. The same one we get with the API.

In the end, we turned it into a dataframe through Pandas.

Data Collection – SpaceX API

First we need to import the necessary libraries.

```
In [2]: # Requests allows us to make HTTP requests which we will use to get data from an API
        import requests
        # 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 large collection of high-level mathematical functions to operate on these arrays
        import numpy as np
        # Datetime is a library that allows us to represent dates
        import datetime
```

Then we make a request with just one line of code.

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

```
In [7]: spacex_url="https://api.spacexdata.com/v4/launches/past"
```

```
In [8]: response = requests.get(spacex_url)
```

The response is something like this:

```
In [9]: print(response.content)
```


```
b'[{"fairings":{"reused":false,"recovery_attempt":false,"recovered":false,"ships":[]},"links":{"patch":{"small":"https://images
2.imgbox.com/3c/0e/T8iJcSN3_o.png","large":"https://images2.imgbox.com/40/e3/GypSkayF_o.png"},"reddit":{"campaign":null,"launc
h":null,"media":null,"recovery":null},"flickr":{"small":[],"original":[]},"presskit":null,"webcast":"https://www.youtube.com/wa
tch?v=0a_00nJ_Y88","youtube_id":"0a_00nJ_Y88","article":"https://www.space.com/2196-spacex-inaugural-falcon-1-rocket-lost-launc
h.html","wikipedia":"https://en.wikipedia.org/wiki/DemoSat"},"static_fire_date_utc":"2006-03-17T00:00:00.000Z","static_fire_dat
e_unix":1142553600,"net":false,"window":0,"rocket":"5e9d0d95eda69955f709d1eb","success":false,"failures":[{"time":33,"altitud
e":null,"reason":"merlin engine failure"}],"details":"Engine failure at 33 seconds and loss of vehicle","crew":[],"ships":[],"c
apsules":[],"payloads":["5eb0e4b5b6c3bb0006eeb1e1"],"launchpad":"5e9e4502f5090995de566f86","flight_number":1,"name":"FalconSa
t","date_utc":"2006-03-24T22:30:00.000Z","date_unix":1143239400,"date_local":"2006-03-25T10:30:00+12:00","date_precision":"hou
r","upcoming":false,"cores":[{"core":"5e9e289df35918033d3b2623","flight":1,"gridfins":false,"legs":false,"reused":false,"landin
g_attempt":false,"landing_success":null,"landing_type":null,"landpad":null}],"auto_update":true,"tbd":false,"launch_library_i
d":null,"id":"5eb87cd9ffd86e000604b32a"},{"fairings":{"reused":false,"recovery_attempt":false,"recovered":false,"ships":[]},"li
nks":{"patch":{"small":"https://images2.imgbox.com/4f/e3/I0lkuJ2e_o.png","large":"https://images2.imgbox.com/be/e7/iNqsqVYM_o.p
ng"},"reddit":{"campaign":null,"launch":null,"media":null,"recovery":null},"flickr":{"small":[],"original":[]},"presskit":nul
l,"webcast":"https://www.youtube.com/watch?v=Lk4zQ2wP-Nc","youtube_id":"Lk4zQ2wP-Nc","article":"https://www.space.com/3590-spac
ex-falcon-1-rocket-fails-reach-orbit.html","wikipedia":"https://en.wikipedia.org/wiki/DemoSat"},"static_fire_date_utc":null,"st
atic_fire_date_unix":null,"net":false,"window":0,"rocket":"5e9d0d95eda69955f709d1eb","success":false,"failures":[{"time":301,"a
ltitude":289,"reason":"harmonic oscillation leading to premature engine shutdown"}],"details":"Successful first stage burn and
transition to second stage, maximum altitude 289 km, Premature engine shutdown at T+7 min 30 s, Failed to reach orbit, Failed t
o recover first stage","crew":[],"ships":[],"capsules":[],"payloads":["5eb0e4b6b6c3bb0006eeb1e2"],"launchpad":"5e9e4502f5090995
de566f86","flight_number":2,"name":"DemoSat","date_utc":"2007-03-21T01:10:00.000Z","date_unix":1174439400,"date_local":"2007-03
-21T13:10:00+12:00","date_precision":"hour","upcoming":false,"cores":[{"core":"5e9e289ef35918416a3b2624","flight":1,"gridfins":
false,"legs":false,"reused":false,"landing_attempt":false,"landing_success":null,"landing_type":null,"landpad":null}],"auto_upd
ate":true,"tbd":false,"launch_library_id":null,"id":"5eb87cdaffd86e000604b32b"},{"fairings":{"reused":false,"recovery_attempt":
false,"recovered":false,"ships":[]},"links":{"patch":{"small":"https://images2.imgbox.com/3d/86/cnu0pan8_o.png","large":"http
s://images2.imgbox.com/4b/bd/d8UxLh4q_o.png"},"reddit":{"campaign":null,"launch":null,"media":null,"recovery":null},"flickr":
{"small":[],"original":[]},"presskit":null,"webcast":"https://www.youtube.com/watch?v=v0w9p3U8860","youtube_id":"v0w9p3U886
0","article":"http://www.spacex.com/news/2013/02/11/falcon-1-flight-3-mission-summary","wikipedia":"https://en.wikipedia.org/wi
```

It can be quite confusing to look at a picture like this. This is pure HTML code.
To extract what we need, it takes a few lines of code and the goal is to transform this "mess" of HTML code into a Pandas DataFrame.
The end result is this:

In [24]:
```
# Show the head of the dataframe
df_launch.head()
```

Out[24]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedC
0	1	2006-03-24	Falcon 1	20.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0
1	2	2007-03-21	Falcon 1	NaN	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0
2	4	2008-09-28	Falcon 1	165.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0
3	5	2009-07-13	Falcon 1	200.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0
4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0

- GitHub URL of the completed SpaceX API calls notebook:

Data Collection via Space X API

Go to my GitHub profile and see all notebooks. This one above, is the entire data collection process via Space X API.

Data Collection - Scraping

For Web Scrapping we need a library called Beautiful Soup and the others we've already mentioned. See below:

First let's import required packages for this lab

In [1]:
```
!pip3 install beautifulsoup4
!pip3 install requests
```

```
Requirement already satisfied: beautifulsoup4 in /opt/conda/envs/Python-3.8-main/lib/python3.8/site-packages (4.9.3)
Requirement already satisfied: soupsieve>1.2 in /opt/conda/envs/Python-3.8-main/lib/python3.8/site-packages (from beautifulsoup
4) (2.2.1)
Requirement already satisfied: requests in /opt/conda/envs/Python-3.8-main/lib/python3.8/site-packages (2.25.1)
Requirement already satisfied: idna<3,>=2.5 in /opt/conda/envs/Python-3.8-main/lib/python3.8/site-packages (from requests) (2.
8)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /opt/conda/envs/Python-3.8-main/lib/python3.8/site-packages (from reque
sts) (1.26.6)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/envs/Python-3.8-main/lib/python3.8/site-packages (from request
s) (2021.5.30)
Requirement already satisfied: chardet<5,>=3.0.2 in /opt/conda/envs/Python-3.8-main/lib/python3.8/site-packages (from requests)
(3.0.4)
```

In [2]:
```
import sys

import requests
from bs4 import BeautifulSoup
import re
import unicodedata
import pandas as pd
```

Then we basically do the same steps. As with the API, we receive the information in HTML.

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

In [6]:
```
# use requests.get() method with the provided static_url
# assign the response to a object

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

Out[6]:
```
<Response [200]>
```

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

In [8]:
```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(response.content, 'html5lib')
```

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

In [11]:
```
# Use soup.title attribute
soup.title
```

Out[11]:
```
<title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

The big difference here is where the BeautifulSoup library comes in. With it, we can browse the HTML code through the TAGS.

For example, on the wikipedia page our objective is to extract the table that contains the rocket data. With BeautifulSoup we can access elements according to the correct tag, in this case we use the '<th>' tag which is used to create tables. If we wanted to extract the title of the page for example, the tag would be '<H1>' of header. This makes it very easy to extract the desired content.

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

In [28]:
```
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
result = first_launch_table.find_all('th')

for i in result:
    columns = extract_column_from_header(i)
    if columns is not None and len(columns) > 0:
        column_names.append(columns)
    else:
        pass
```

Check the extracted column names

In [29]:
```
print(column_names)
```

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

After a few lines of code we got the same result, a Dataframe with records and information about Falcon 9 rocket missions.

```
In [105]: df=pd.DataFrame(launch_dict)
          df
```

Out[105]:

	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

Wikipedia Falcon 9 List

- GitHub URL of the completed Web Scrapping Process notebook:

Web Scrapping notebook

Data Wrangling

In this step we start working on the collected data, checking null values, inconsistencies, cleaning the dataset, adding a new feature called 'Class' and of course, some pre analysis.

Identifying null values:

Identify and calculate the percentage of the missing values in each attribu

```
In [3]: df.isnull().sum()/df.count()*100
```

```
Out[3]: FlightNumber      0.000
        Date              0.000
        BoosterVersion    0.000
        PayloadMass       0.000
        Orbit             0.000
        LaunchSite        0.000
        Outcome           0.000
        Flights           0.000
        GridFins          0.000
        Reused            0.000
        Legs              0.000
        LandingPad       40.625
        Block             0.000
        ReusedCount       0.000
        Serial            0.000
        Longitude         0.000
        Latitude          0.000
        dtype: float64
```

- 40% of the records in the 'Landing Pad' column were null.

The data contains 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'.

Next, let's see the number of launches for each site:

```
In [8]: # Apply value_counts() on column LaunchSite
        df['LaunchSite'].value_counts()

Out[8]: CCAFS SLC 40    55
        KSC LC 39A      22
        VAFB SLC 4E     13
        Name: LaunchSite, dtype: int64
```

We can see that the launch site **CCAFS SLC 40** was much more used than the others. It's probably the oldest.

ORBITS:

We see that 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), or with at least 11.25 periods per day (an orbital period of 128 minutes or less) and an eccentricity less than 0.25. Most of the manmade objects in outer space are in LEO.
- **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.
- **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.

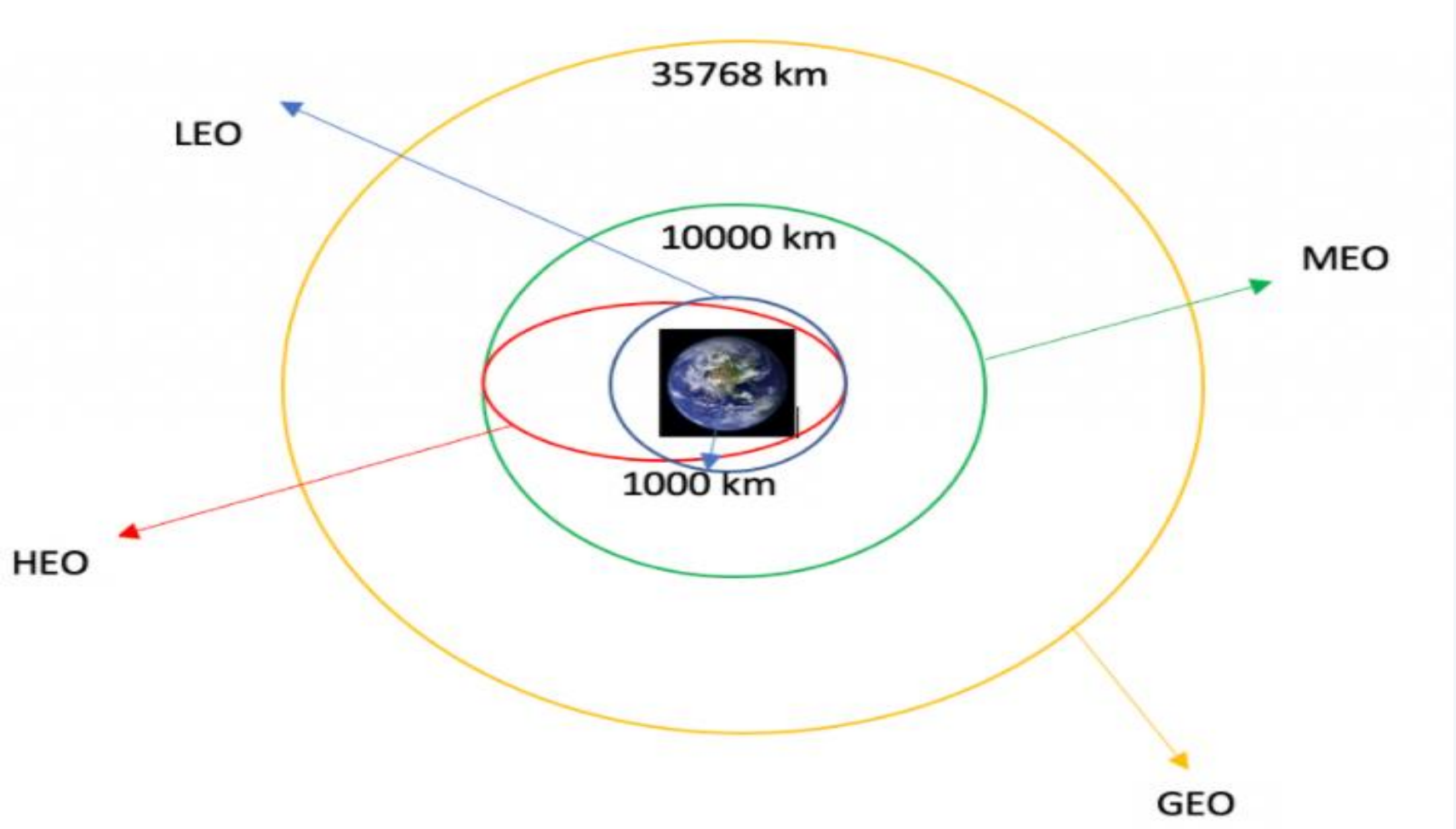
- **SSO (or SO) -** It is a Sun-
  synchronous orbit also called a heliosynchronous orbit is a nearly polar orbit around a planet, in which the satellite passes
  over any given point of the planet's surface at the same local mean solar time.

- **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 c enter of mass of the large bodies. L1 is one such point between the sun and the earth.

- **HEO** A highly elliptical orbit, is an elliptic orbit with high eccentricity, usually referring to one around Earth.

- **ISS -** A modular space station (habitable artificial satellite) in low Earth orbit. It is a multinational collaborative project between five participating sp ace agencies: NASA (United States), Roscosmos (Russia), JAXA (Japan), ESA (Europe), and CSA (Canada).
- **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 kno wn 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.
- **HEO -**Geocentric orbits above the altitude of geosynchronous orbit (35,786 km or 22,236 mi).
- **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.
- **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.

Here is an illustration:



Here is the number of launches for each orbit:

```
In [9]: # Apply value_counts on Orbit column
        df['Orbit'].value_counts()

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

Here are the types of landing outcomes:

```
In [36]: # landing_outcomes = values on Outcome column
         landing_outcomes = df['Outcome'].value_counts()
         landing_outcomes

Out[36]: 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: Outcome, dtype: int64
```

Finally, we made a column that simplifies landing outcomes into just two classes, success or failure. Being "1" for success and "0" for failure.

```
In [31]: df['Class']=landing_class
         df[['Class']].head(8)
```

Out[31]:

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

With this new 'Class' column, we were able to average overall success.

```
In [34]: df["Class"].mean()

Out[34]: 0.6666666666666666
```

The result of the DataFrame with the column 'Class':

```
In [33]: df.head(10)
```

Out[33]:

yloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
04.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577366	28.561857	0
5.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366	28.561857	0
7.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.577366	28.561857	0

- GitHub URL of the completed Data Wrangling notebook:

[Data Wrangling notebook]

EDA with Data Visualization

In this step we use several graphs to visualize variables and find correlation between them. We use Scatter charts, Bar charts, histogram, and Line charts.

For example, we try to see how the FlightNumber (indicating the continuous launch attempts.) and Payload variables would affect the launch outcome.

We plot a scatter chart out the FlightNumber vs. PayloadMassand to overlay the outcome of the launch. We see that 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.

We will see the graphics in the results section.

We also visualized, through a bar graph, the relationship between success rate of each orbit type. Lastly we saw the success rate over the years with a line graph.

Also at this stage, we did a process called "Features Engineering", which consists of better preparing the Dataframe for the future use of machine learning models.

See the entire process on my GitHub profile.

- GitHub URL of the completed Data EAD Visualization notebook:

Data EAD Visualization

EDA with SQL

Databases are not just for storing data. We can get very interesting insights through custom SQL queries.

We use the IBM Db2 relational database to carry out this process.

First we upload the dataset to the database. Then through a Jupyter notebook we even made a connection with the database.

For this process, the following libraries are needed:

sqlalchemy, **ibm_db_sa** and **ipython-sql**.

```
In [1]: !pip install sqlalchemy==1.3.9
        !pip install ibm_db_sa
        !pip install ipython-sql
```

After connecting to the database using my credentials, it is possible to make SQL queries on a Jupyter notebook. See below some of the surveys made and our findings.

The total payload mass carried by boosters launched by NASA (CRS):

```
In [10]: %sql SELECT SUM(payload_mass__kg_) FROM spacexdataset WHERE customer = 'NASA (CRS)';
 * ibm_db_sa://lcn03492:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:32731/BLUDB
Done.
Out[10]:
1
45596
```

The average payload mass carried by booster version F9 v1.1:

```
In [14]: %sql SELECT AVG(payload_mass__kg_) FROM spacexdataset WHERE booster_version = 'F9 v1.1';
 * ibm_db_sa://lcn03492:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.c1ogj3sd0tgtu0lqde00.dat
Done.
Out[14]:
1
2928
```

The date when the first successful landing outcome in ground pad was aquived:

```
In [16]: %sql SELECT min(date) FROM spacexdataset WHERE landing__outcome = 'Success (ground pad)';
 * ibm_db_sa://lcn03492:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.c1ogj3sd0tgtu0lqde00.data
Done.
```

Out[16]:

1
2015-12-22

The total number of successful mission outcomes:

```
In [32]: %sql SELECT COUNT(landing__outcome) FROM spacexdataset WHERE landing__outcome LIKE '%Success%';
 * ibm_db_sa://lcn03492:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.c1ogj3sd0tgtu0lqde00.databases.
Done.
```

Out[32]:

1
61

The names of the booster versions which have carried the maximum payload mass:

In [36]:
```
%sql SELECT BOOSTER_VERSION, payload_mass__kg_ FROM spacexdataset WHERE  payload_mass__kg_ = (SELECT MAX(payload_mass__kg_) FROM spacexdataset);
```

```
 * ibm_db_sa://lcn03492:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:32731/BLUDB
Done.
```

Out[36]:

booster_version	payload_mass__kg_
F9 B5 B1048.4	15600
F9 B5 B1049.4	15600
F9 B5 B1051.3	15600
F9 B5 B1056.4	15600
F9 B5 B1048.5	15600
F9 B5 B1051.4	15600
F9 B5 B1049.5	15600
F9 B5 B1060.2	15600
F9 B5 B1058.3	15600
F9 B5 B1051.6	15600
F9 B5 B1060.3	15600
F9 B5 B1049.7	15600

The rank of the number 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:

In [56]:
```
%sql SELECT COUNT(landing__outcome), landing__outcome FROM spacexdataset WHERE (date BETWEEN '2010-06-04' AND '2017-03-20') GROU
P BY landing__outcome ORDER BY COUNT(landing__outcome) DESC;
```

```
 * ibm_db_sa://lcn03492:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:32731/BLUDB
Done.
```

Out[56]:

1	landing__outcome
10	No attempt
5	Failure (drone ship)
5	Success (drone ship)
3	Controlled (ocean)
3	Success (ground pad)
2	Failure (parachute)
2	Uncontrolled (ocean)
1	Precluded (drone ship)

- GitHub URL of the completed Data EAD Visualization with SQL notebook:

[Data EAD with SQL]

Interactive Map with Folium

The Folium library allows us to visualize through interactive maps. First we need to install the libraries.

```
In [1]: !pip3 install folium
        !pip3 install wget
```

We started by creating a map that starts on top of NASA Johnson Space Center. For this you need to provide the latitude and longitude of the location.

```
In [6]: # Start location is NASA Johnson Space Center
        nasa_coordinate = [29.559684888503615, -95.0830971930759]
        site_map = folium.Map(location=nasa_coordinate, zoom_start=10)
```

We had access to the latitude and longitude of each launch site used on the Falcon 9 rocket missions.

Out[5]:

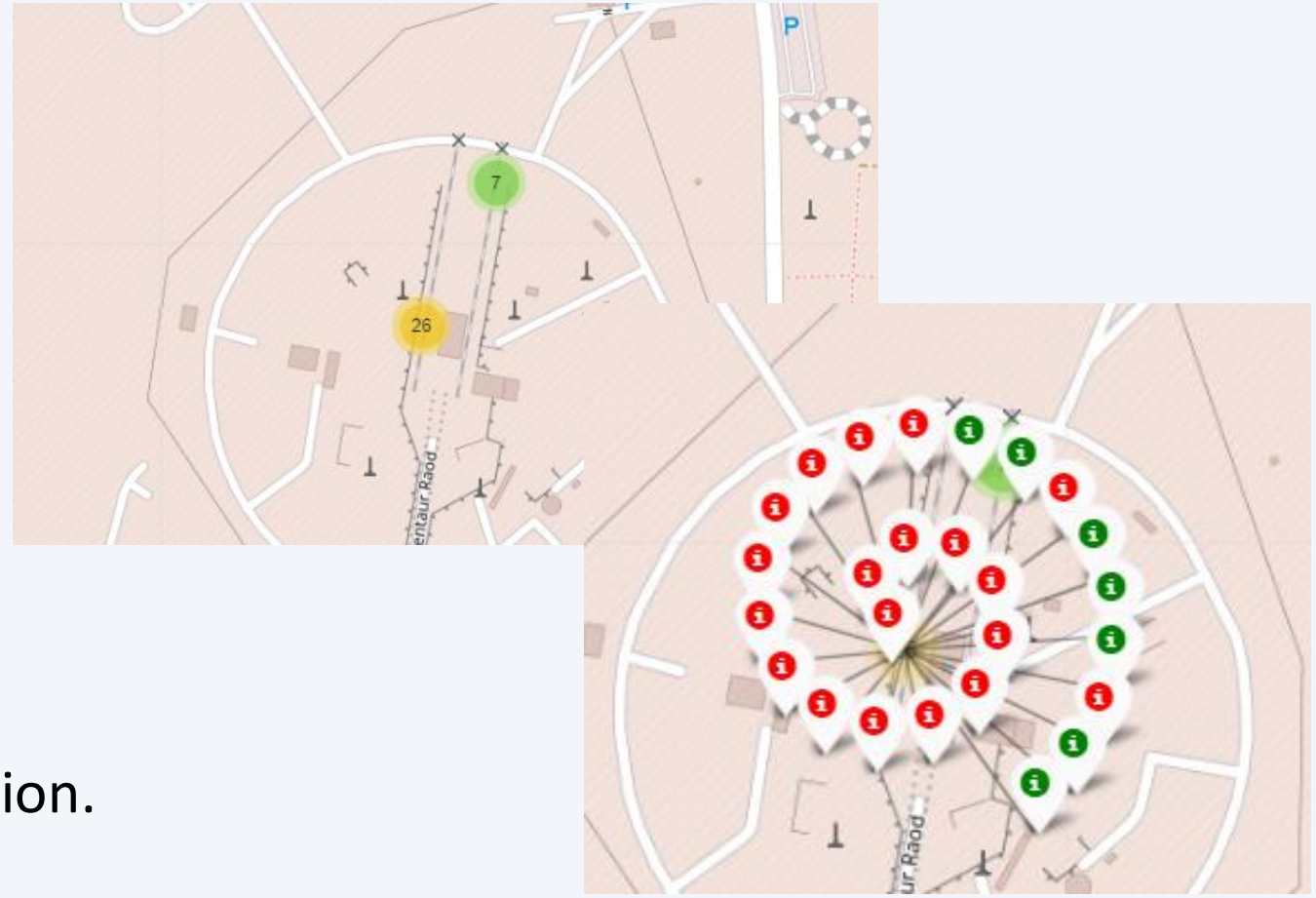
	Launch Site	Lat	Long
0	CCAFS LC-40	28.562302	-80.577356
1	CCAFS SLC-40	28.563197	-80.576820
2	KSC LC-39A	28.573255	-80.646895
3	VAFB SLC-4E	34.632834	-120.610746

We mark each of them with a circle using the Circle() function and providing the latitude and longitude values.

In [8]:
```
# Initial the map
nasa_coordinate1 = [28.562302, -80.577356]
nasa_coordinate2 = [28.563197, -80.576820 ]
nasa_coordinate3 = [28.573255, -80.646895]
nasa_coordinate4 = [34.632834, -120.610746]
site_map1 = folium.Map(location=nasa_coordinate1, zoom_start=5)
```

After creating a circle around each launch site, we created a MarkerCluster, a Folium function that marks several markers in the same designated place. With that, we mark each mission of each launch site in a cluster. We still gave the color green to those who were successful and the color red to those who failed.

The result was like this:

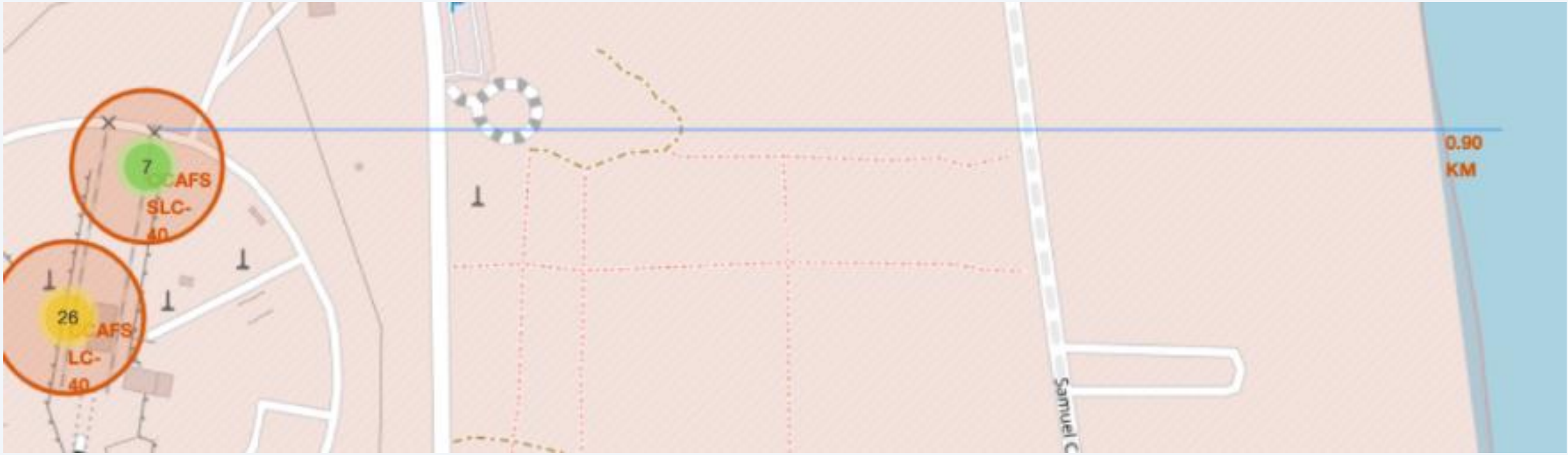


You will see more in the result session.

Looking at this map, we have the opportunity to see the environment around the Launch sites. We noticed for example that all launch sites are very close to the ocean.

We could see the distance between them and the ocean. We marked the beginning of the ocean and then we draw a line between the marker to the launch site through a distance marking, a function called MousePoint.

See that it was possible to know from a distance:



Folium is great and it give us a big picture about the proximities of the launch sites.

After we plot distance lines to the proximities, we can answer the following questions easily:

1. Are launch sites in close proximity to railways?
2. Are launch sites in close proximity to highways?
3. Are launch sites in close proximity to coastline?
4. Do launch sites keep certain distance away from cities?

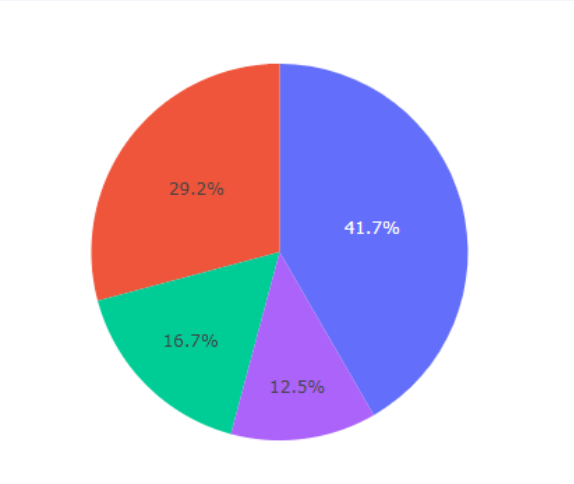
- GitHub URL of the completed Data Visualization with Folium notebook:

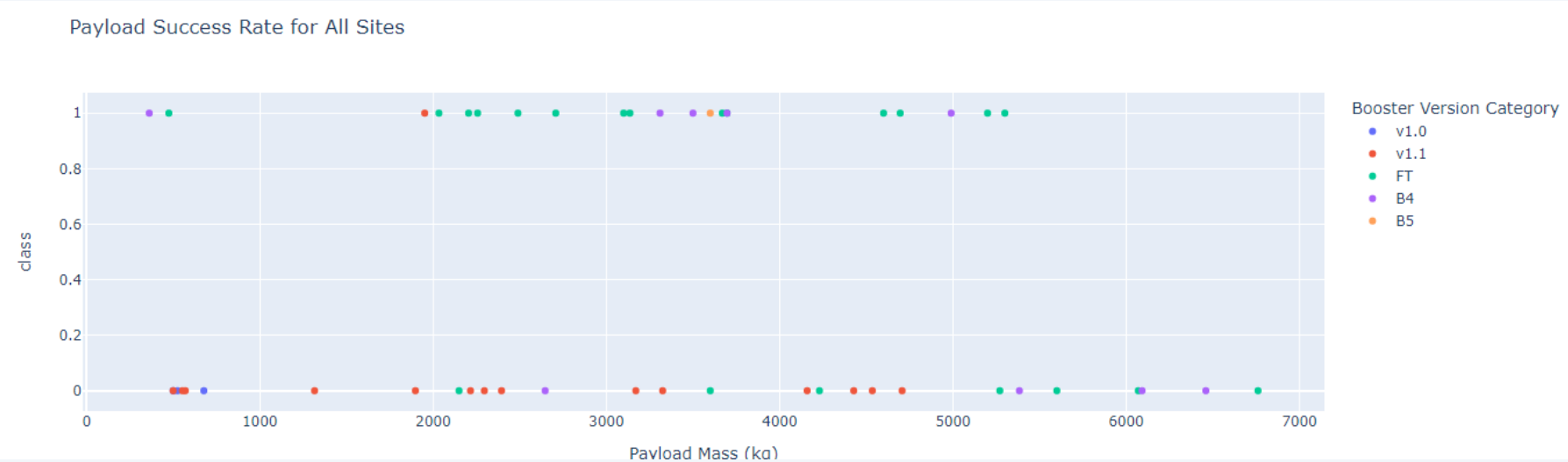
Data visualization with Folium

Build a Dashboard with Plotly Dash

When it comes to visualizing graphs over data, we have to use a very special tool, the Dash.

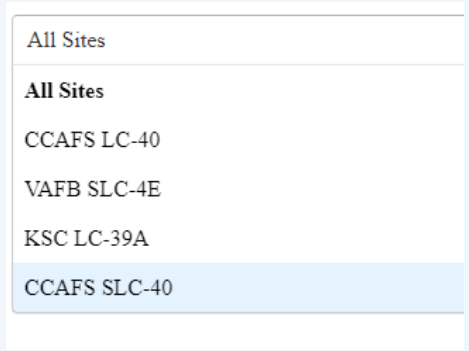
Dash is a library that allows us to create a web application with interactive graphics from matplotlib library. We created a dash application that contains a pie chart and a scatter chart.





Our pie chart shows the percentage of success on each launch site.

Below the pie chart we have a scatter chart that shows us the Payload success rate for each launch site or for all of them, you choose!



The scatter chart shows us the relationship between classes (0 and 1, success and failure) and Payload Mass(Kg).

Let's explore our web application and graphics further in the results section.

- GitHub URL of the completed Data Visualization with Dash notebook:

Dash notebook

Dash Application Web

Predictive Analysis (Classification)

This is the final step of this project. We build a classification model that will predict whether with given characteristics, the rocket will land successfully or not.

In this process, the first step was to test some classification machine learning algorithms. We use classification models because we want to classify a new rocket between two classes, 0 or 1, as we saw earlier.

The algorithms we are going to test are:

- **`Logistic Regression`**
- **`Decision Tree`**
- **`Support Vector Machine`**
- **`KNeighbors`**

Each of these models has strengths and weaknesses, but they are all interesting in this case study. For each of them there are several parameters that we can change and we need to find the correct ones so that the model is as accurate as possible.

For this task, there is a library called GridSearchCV, where it creates models with different parameter settings, and then it test until it finds the best combination for the given case. See the required library imports for this step:

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 large collection of high-level mathematica
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a MatLab like plotting framework. We will use this in our plotter function to plot data.
import matplotlib.pyplot as plt
#Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics
import seaborn as sns
# Preprocessing allows us to standarsize our data
from sklearn import preprocessing
# Allows us to split our data into training and testing data
from sklearn.model_selection import train_test_split
# Allows us to test parameters of classification algorithms and find the best one
from sklearn.model_selection import GridSearchCV
# Logistic Regression classification algorithm
from sklearn.linear_model import LogisticRegression
# Support Vector Machine classification algorithm
from sklearn.svm import SVC
# Decision Tree classification algorithm
from sklearn.tree import DecisionTreeClassifier
# K Nearest Neighbors classification algorithm
from sklearn.neighbors import KNeighborsClassifier
```

Sklearn is a library for machine learning. It contains almost all regression, cluster and classification models.

- **Preprocessing:**

Let's take a step back for a moment. Before creating and using predict models, we need to prepare the data for this. All algorithms work with mathematical calculations, and almost all of them need the data to be in numerical form, otherwise the algorithm will return an error.

See how our data is currently:

Out[3]:

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

Note that we have several categorical attributes, ie words. We have "BoosterVersion", "Orbit" among others. So the first step is to apply a technique called One Hot Encoding.

This technique consists of replacing words with numbers, but each different word will have a different number. For example, if we only have 3 different launch sites, with this technique we will change each launch site by a number, it will be like CCAFS SLC 40 will be replaced by number 1, VAFB SLC 4E will be replaced by number 2 and so on.

In the case of the Orbit column, we create sub-columns for each orbit, so the orbit column was divided into a column for each orbit, and if this orbit is the target of the mission, the column will have a 1, if it does not have a 0. See the result below:

Out[16]:

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	...	Serial_B1058	Serial_B
0	1.0	6104.959412	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
1	2.0	525.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
2	3.0	677.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	0.0
3	4.0	500.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
4	5.0	3170.000000	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
85	86.0	15400.000000	2.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
86	87.0	15400.000000	3.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0	...	1.0	0.0
87	88.0	15400.000000	6.0	5.0	5.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
88	89.0	15400.000000	3.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
89	90.0	3681.000000	1.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0

90 rows × 83 columns

The last step of preparing the data for use in the models is called the StandScaler. Notice in the figure above how the scale of values is very disproportionate.

The "Payload Mass" column has values up to 6104.959412 Kg, while other columns such as orbits vary between 0 and 1.

This is harmful because at the time of mathematical calculations, the Pay load column will exert a very large influence compared to the others. The values extracted from this column will be disproportionate compared to the others. In the end, the algorithm may interpret this column as being more important than all the others and thus harming the final result.

For that we use this tool called Standard Scale, which will put all numbers on the same scale. Thus, for the algorithm, all columns will have the same influence or the same "weight" in the calculation phase. See how it turned out:

```
Out[30]: array([[-1.71291154e+00, -5.29526321e-17, -6.53912840e-01, ...,
                 -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
                [-1.67441914e+00, -1.19523159e+00, -6.53912840e-01, ...,
                 -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
                [-1.63592675e+00, -1.16267307e+00, -6.53912840e-01, ...,
                 -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
                ...,
                [ 1.63592675e+00,  1.99100483e+00,  3.49060516e+00, ...,
                  1.19684269e+00, -5.17306132e-01,  5.17306132e-01],
                [ 1.67441914e+00,  1.99100483e+00,  1.00389436e+00, ...,
                  1.19684269e+00, -5.17306132e-01,  5.17306132e-01],
                [ 1.71291154e+00, -5.19213966e-01, -6.53912840e-01, ...,
                 -8.35531692e-01, -5.17306132e-01,  5.17306132e-01]])
```

Now that the data is in numerical form and on the same scale, we've separated the so-called "predictive features" from our "taget".

What is that? What do we want to predict here?

We want to predict whether a rocket will land successfully or not, that is, we want to designated it with 1 or 0 in the "Class" column. So the "Class" column is our target also called the dependent variable.

All other columns are our predictive features or also called independent variables.

That's because we're going to use them as information that will tell us the result of the classification.

Commonly, we separate independent variables into a single variable called "X". And we put our dependent variable or target in a variable called "Y".

In the end, the X variable will contain all the columns of the dataset, except the "Class" column, which will be inside the Y variable. The X variable you saw above with equal scale values, the Y variable looks like this:

```
In [5]: y = data['Class'].to_numpy()
        y

Out[5]: array([0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1,
               1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
               1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
               1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
               1, 1])
```

- **Train and test split:**

In this step, we train our algorithm with the data. We use the train_test_split tool which takes a piece of data for testing and the rest is used for training. I'll explain why this is.

If we passed all the data to the algorithm to train with them, and then we took a record of the data and wanted a prediction from it, the model would get it right almost 100% of the time.

Why that?

Why aren't we giving him new information, he already trains with this data. It's the same as if a teacher gives you an exercise and at the time of the test the same exercise is there. It makes no sense and there is no way to measure student learning. It's the same thing with our model.

So what we do is take a part of the data, usually about 20% to 30% depending on the size of the data, and separate it, so after the model trains with 80% of the data, we make a prediction with the test data. That way we can see how the model is behaving with new data, that is, data it has never seen.

The data division process is like this:

The data division process is like this:

```
X_train, X_test, Y_train, Y_test
```

```
In [19]: X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.2, random_state=2)
```

We divide variable X and y into 2 variables.

The y variables are 20% of the data, and the X variables are 80% of the data.

Variable "X_train" contains the predictor features and variable "X_test" contains the target feature. Same thing with the y variables, the difference is that they have less data.

So later on we will train the algorithm with the variables X_train and y_train.

That's because in algorithmic training, we need to provide as much information as the "answer" so to speak.

The algorithm needs to know the classes of each record that was used in your training.

That's why the y_train variable enters. It is the answer to the X_train variable.

At test time we only give the model the variable X_test and ask for the forecast.

Once we have the prediction, we compare it with the y-test variable, which is the answer, and see how well the algorithm got right.

- Grid Search:

The variables we split will be used later. Now is the step of figuring out which algorithm is best for our data and which parameters are best for each of them.

This is what Grid Search does. First we create a variable of the dictionary type containing several parameter settings that we want the grid to test, and this is for each algorithm.

The first algorithm we tested was Logistic regression.

What grid search does is take an algorithm, take a set of parameter settings then train the algorithm and measure the accuracy score.

Each test round, he changes the algorithm's parameters and compares the results.

In the end, it returns what were the best parameters and the best result.

See the code to carry out this process:

```
In [23]: parameters ={'C':[0.01,0.1,1],
                      'penalty':['l2'],
                      'solver':['lbfgs']}

         logreg_cv = GridSearchCV(estimator=LogisticRegression(), param_grid=parameters)
         logreg_cv.fit(X, y)
         best_parameters_lr = logreg_cv.best_params_    # Best parameters obtained
         best_result_lr = logreg_cv.best_score_        # Best accuracy obtained
         print(f'The best parameters are {best_parameters_lr}')
         print(f"The best_result/accuracy of this model is: {best_result_lr}")

         The best parameters are {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
         The best_result/accuracy of this model is: 0.8222222222222222
```

After that, just create the Logistical Regression model with the parameters that GridSearch returned.

```
In [33]: parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
         lr=LogisticRegression(C=0.01, penalty='l2', solver='lbfgs')
         lr.fit(X_train, Y_train)
         lr_predict = lr.predict(X_test)
```

We do this process for each algorithm and create a model based on the best parameters.

In the end, we compare the results between them to choose which is the best among them.

We use the Accuracy Score metric, which returns the algorithm's average accuracy.

In other words, this metric compares the prediction values with the actual values, and returns the accuracy score of each model.

- Accuracy Score:

The score of the models is:

1. Logistic Regression = 83%

```
In [13]: lr.score(X_test, Y_test)
Out[13]: 0.8333333333333334
```

2. Decision Tree = 88%

```
In [36]: tree.score(X_test, Y_test)
Out[36]: 0.8888888888888888
```

3. Support Vector Machine = 83%

```
In [18]: svm.score(X_test, Y_test)
Out[18]: 0.8333333333333334
```

4. KNeighbors = 83%

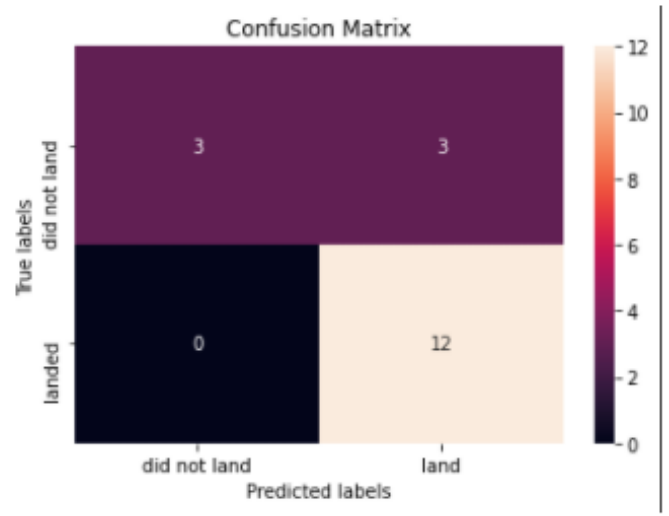
```
In [28]: KNN.score(X_test, Y_test)
Out[28]: 0.8333333333333334
```

- Confusion Matrix:

We also analyzed a graph called "Confusion Matrix". They show us the actual values of each class, and the error of the predictions by class.

It's a good way to see if the model hits one class better than another. See a example of a confusion matrix of the Logistic Regression model:

```
In [14]: yhat=lr.predict(X_test)
         plot_confusion_matrix(Y_test,yhat)
```



- The model classified 3 instances as "did not land" and which actually were "did not land".
- The model classified 3 instances as "land" and which actually were "did not land".
- The model classified 0 instances as "did not land" that were actually "land".
- The model classified 12 instances as "land" that were actually "land".

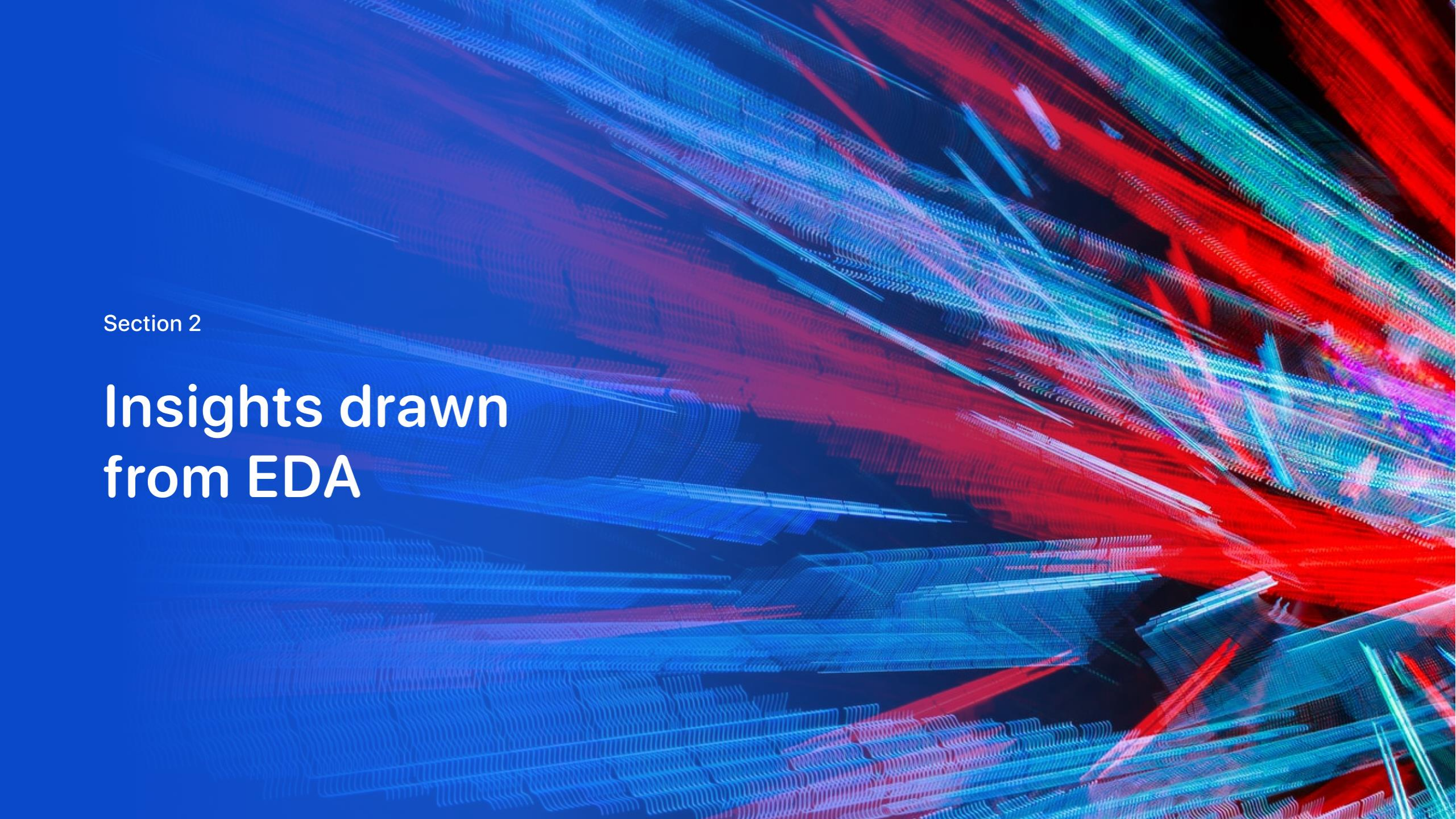
It is noticed that the Logistic Regression model had a better accuracy in classifying the "land" class or the success(1) class, than the failure(0) class.

In the results section we will analyze the decision tree model matrix.

See the entire process of creating the algorithms in my GitHub profile.

- GitHub URL of the completed Machine Learning models notebook:

Machine learning process



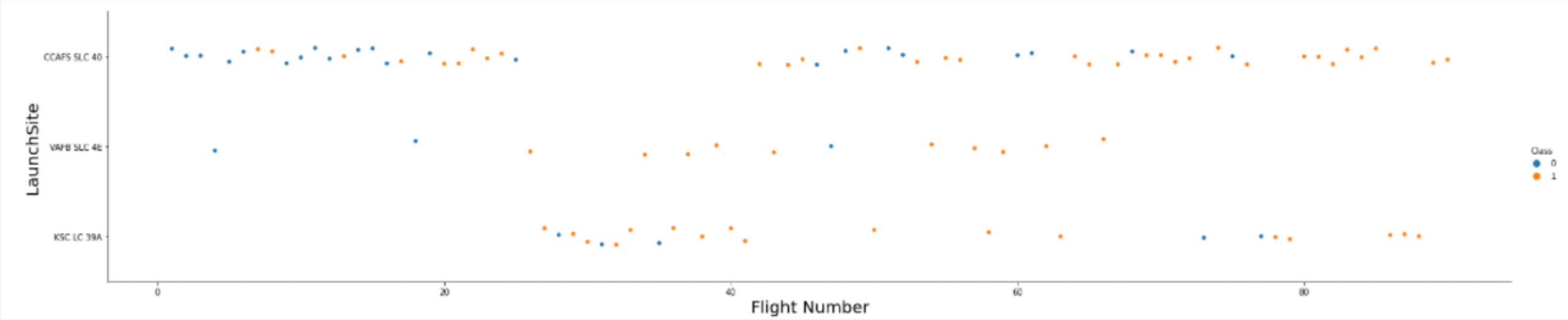
Section 2

Insights drawn from EDA

Results

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

Flight Number vs. Launch Site



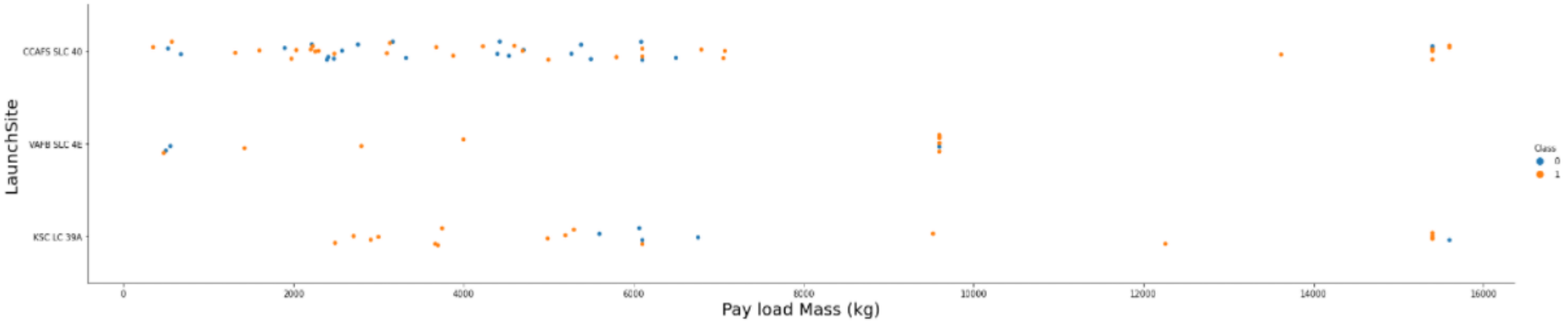
Here we clearly see that there is a positive linear correlation between these two variables.

In all Launch Sites, the success rate increases as the number of Flights increases.

We designate yellow for successful missions and blue for failed missions.

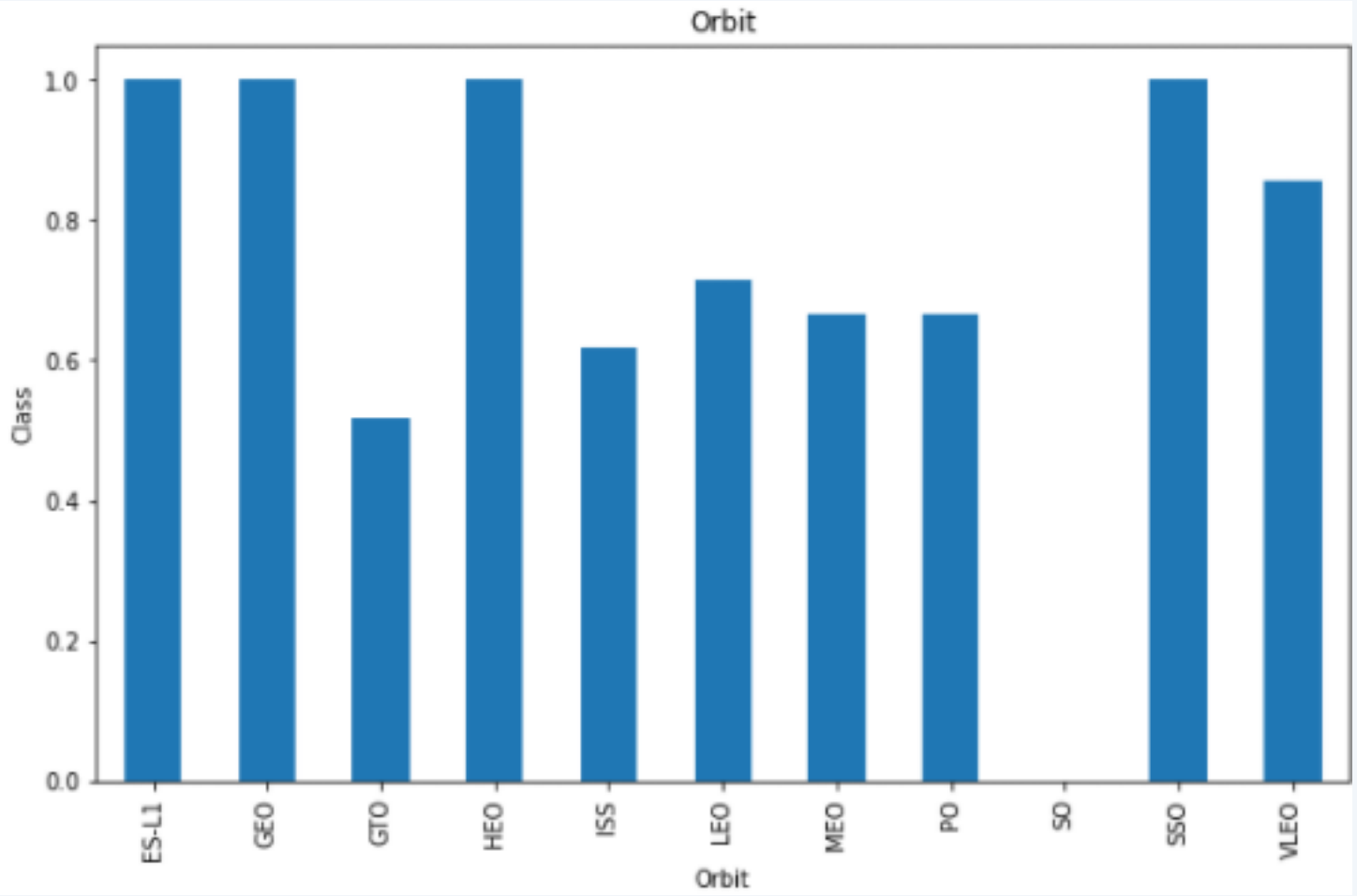
See that as the number of flights increases the more yellow dots we have in all de Launch sites. This tells us that they are on the right path and increasingly improving their rockets.

Payload vs. Launch Site



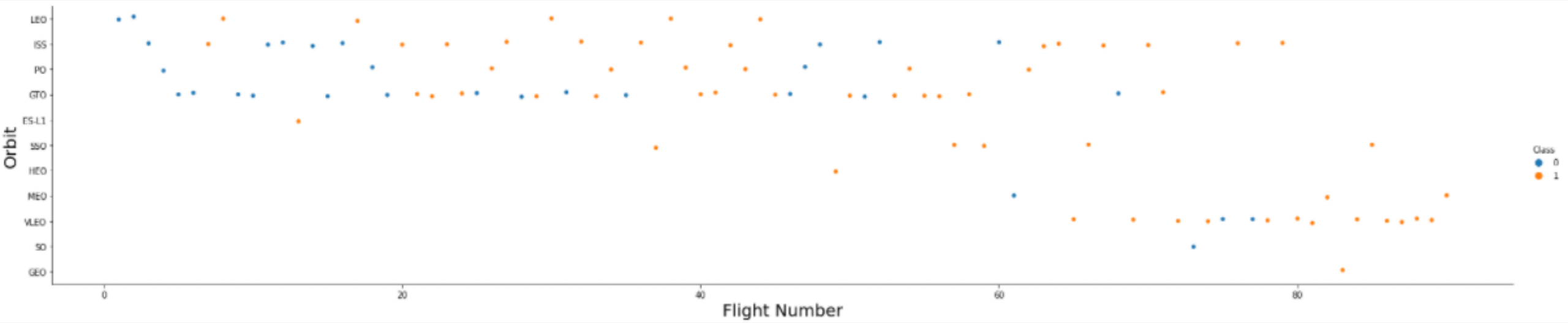
1. We definitely see that the bigger the Payload Mass the greater the chance of success on the VAFB SLC 4E launch site.
2. On the launch site KSC LC-39A, we have a Payload Mass fax that did not work, approximately between 5700 Kg and 6400 Kg we have this window with almost 100% failure. Smaller Payload and Bigger Payload than that, we have a very good success rate!
3. On the CCAFS SLC 40 launch site, we have a lot of variation in the results between 1000kg and 6000kg Payload Mass. But from 6000kg to approximately 16000kg we have a great success rate.

Success Rate vs. Orbit Type



1. We have the ES-L1, GEO, HEO, SSO orbits with 100% success!
2. The worst orbit is the GTO with about 50% success, same as randomness.

Flight Number vs. Orbit Type

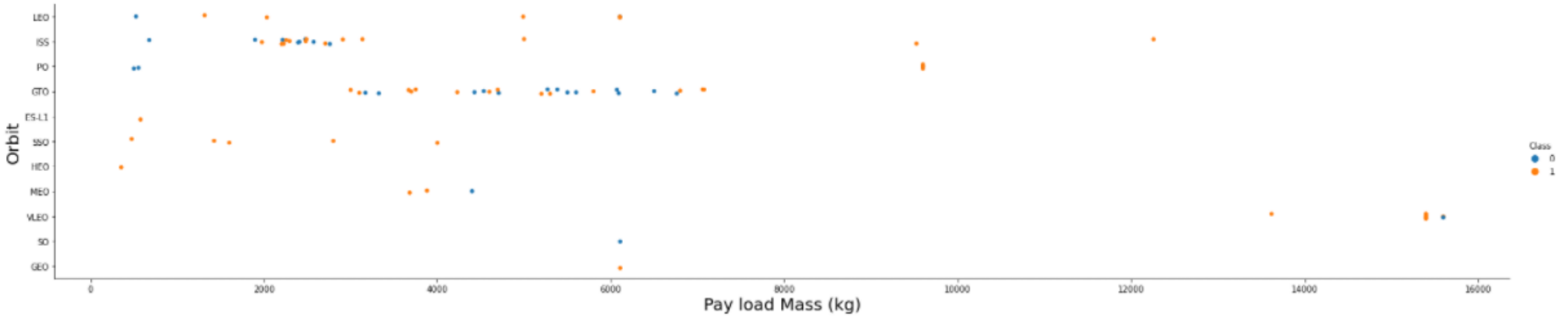


For each orbit the success clearly increases with each Flight number.

Clearly progress throughout the missions.

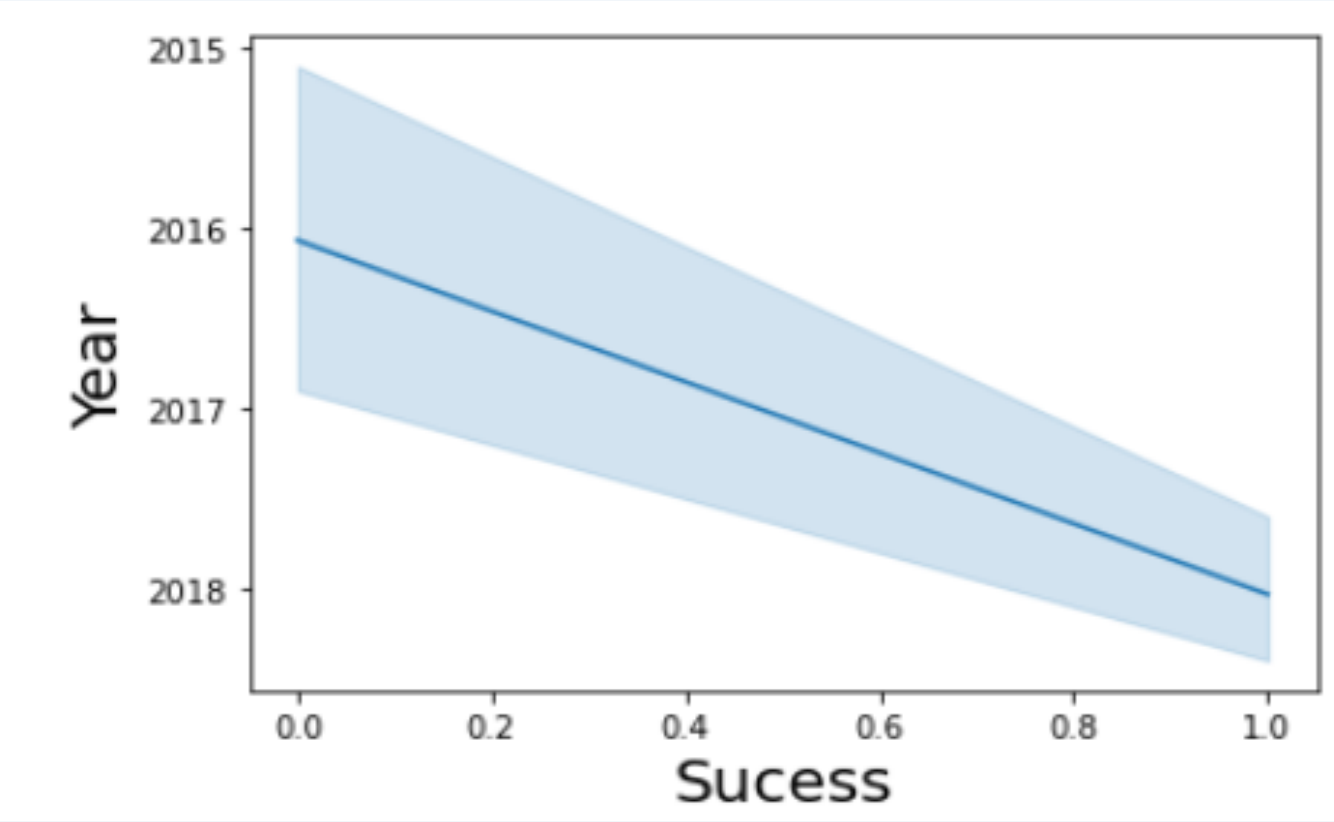
Except for the GTO orbit, which looks completely random.

Payload vs. Orbit Type



- For ES-L1, SSO and HEO orbits, it has a negative linear correlation. In other words, the smaller the Payload Mass, the higher the success rate!
- As for the LEO, ISS, PO and VLEO orbits, we see a positive correlation, where the higher the Payload Mass, the higher the success rate.
- The GTO orbit continues in randomness.

Launch Success Yearly Trend



We could see that progress actually happened in each mission. Here we prove it.

We have a linearly positive relationship between success rate and years.

Each year the success rate increases.

They are definitely on the right track. But you can always improve, isn't it?

All Launch Site Names

Out[5]:

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

With SQL queries, we were able to know the name of all launch sites used in Falcon 9 rocket missions.

Launch Site Names Begin with 'CCA'

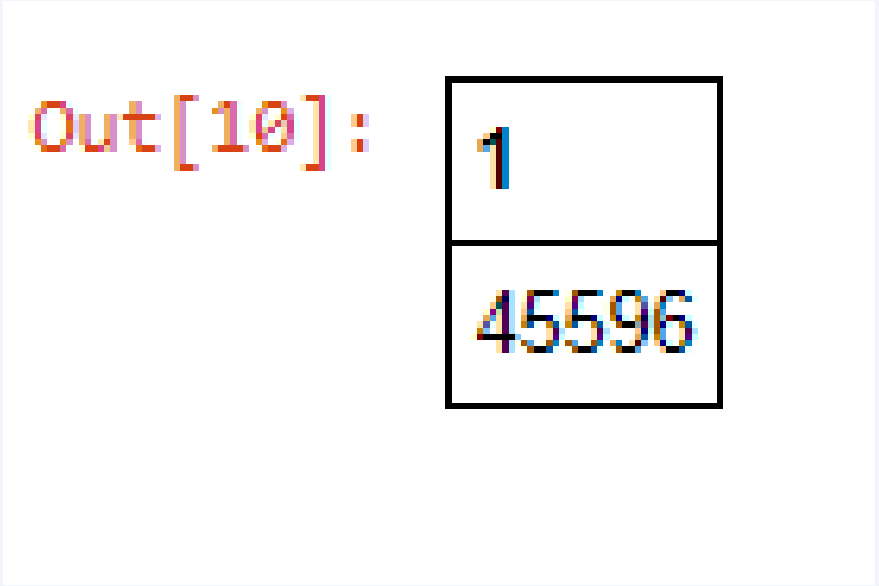
Out[6]:

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	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	00:35:00	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

Here we look at the first 5 missions through the launch sites starting with "CCA".

We realized that despite the 5 first missions being successful, the landing was not.

Total Payload Mass



This is the payload total that NASA boosters loaded on Falcon 9 missions.

45.596 kg!

Average Payload Mass by F9 v1.1

Out[14]:

1	
2928	

This is the total average Payload that the F9 v1.1 version boosters have carried.

2.928 kg.

First Successful Ground Landing Date

Out[16]:

1
2015-12-22

The date when the first successful landing outcome in ground pad was achieved.

Successful Drone Ship Landing with Payload between 4000 and 6000

The names of the boosters which have success in drone ship and have Payload mass greater than 4000 but less than 6000.

Out[26]:

booster_version
F9 v1.1
F9 v1.1 B1011
F9 v1.1 B1014
F9 v1.1 B1016
F9 FT B1020
F9 FT B1022
F9 FT B1026
F9 FT B1030
F9 FT B1021.2
F9 FT B1032.1
F9 B4 B1040.1
F9 FT B1031.2
F9 FT B1032.2
F9 B4 B1040.2
F9 B5 B1046.2
F9 B5 B1047.2
F9 B5 B1046.3
F9 B5B1054
F9 B5 B1048.3
F9 B5 B1051.2
F9 B5B1060.1
F9 B5 B1058.2
F9 B5B1062.1

Total Number of Successful and Failure Mission Outcomes

Out[31]:

Total	mission_outcome
99	Success
1	Failure (in flight)
1	Success (payload status unclear)

Here is the result of missions with Falcon 9 rockets.

Note that it is not the landing that is at issue, but the mission.

Only one mission failed.

Boosters Carried Maximum Payload

This is the list of booster versions that carried the biggest Payload Mass.

No Booster Versions have carried more than this value of 15.600Kg.

Out[11]:

booster_version	payload_mass__kg_
F9 B5 B1048.4	15600
F9 B5 B1049.4	15600
F9 B5 B1051.3	15600
F9 B5 B1056.4	15600
F9 B5 B1048.5	15600
F9 B5 B1051.4	15600
F9 B5 B1049.5	15600
F9 B5 B1060.2	15600
F9 B5 B1058.3	15600
F9 B5 B1051.6	15600
F9 B5 B1060.3	15600
F9 B5 B1049.7	15600

2015 Launch Records

Out[12]:

DATE	booster_version	launch_site	landing__outcome
2015-01-10	F9 v1.1 B1012	CCAFS LC-40	Failure (drone ship)
2015-04-14	F9 v1.1 B1015	CCAFS LC-40	Failure (drone ship)

These are the missions that failed to land in 2015.

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

Here is the count of landing outcomes between the date 2010-06-04 and 2017-03-20, in descending order.

Out[32]:

Total	Landing outcomes
10	No attempt
5	Failure (drone ship)
5	Success (drone ship)
3	Controlled (ocean)
3	Success (ground pad)
2	Failure (parachute)
2	Uncontrolled (ocean)
1	Precluded (drone ship)

Section 4

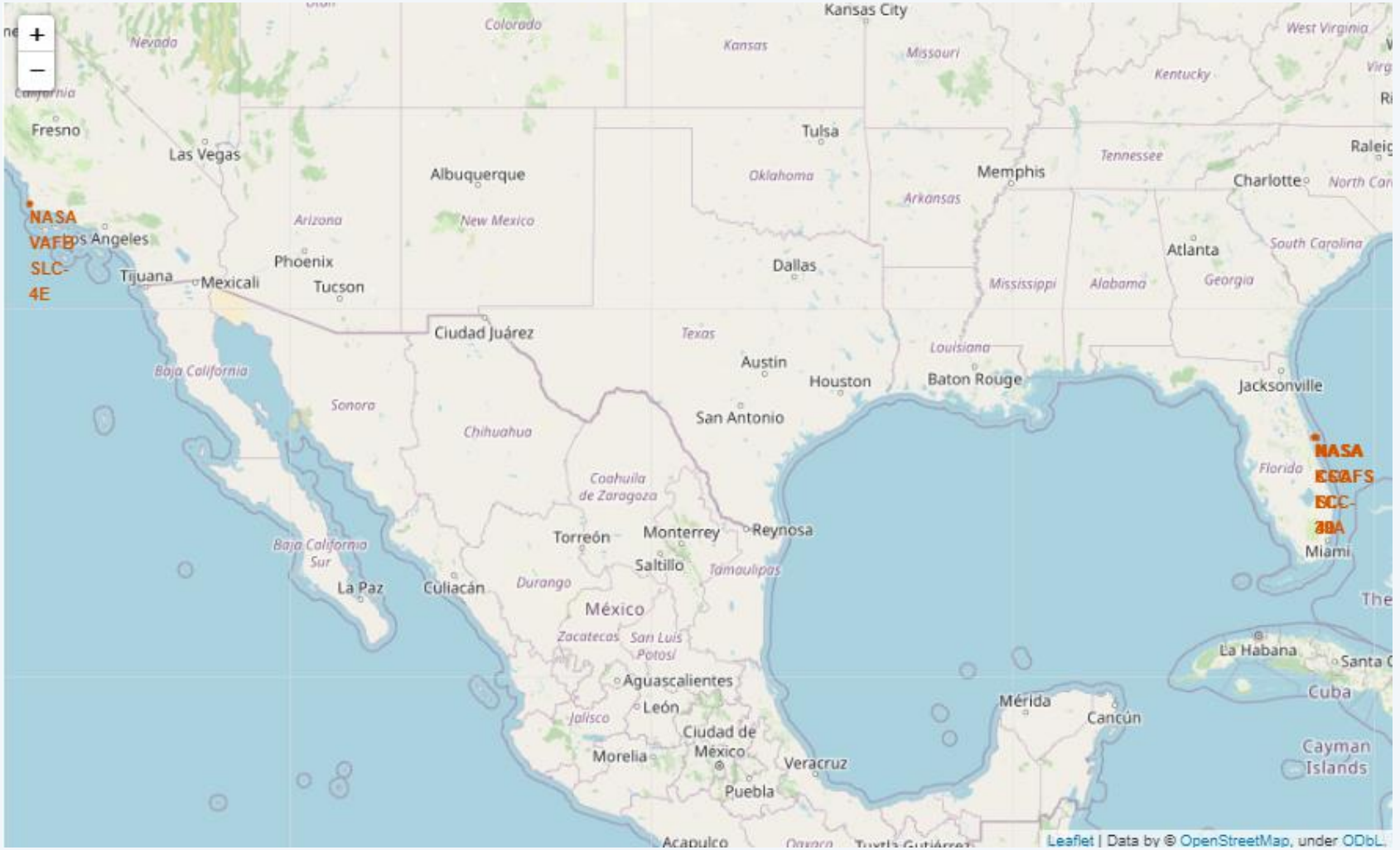
Launch Sites Proximities Analysis

Folium Map with Launch Sites

Here we see the Launch sites marked.

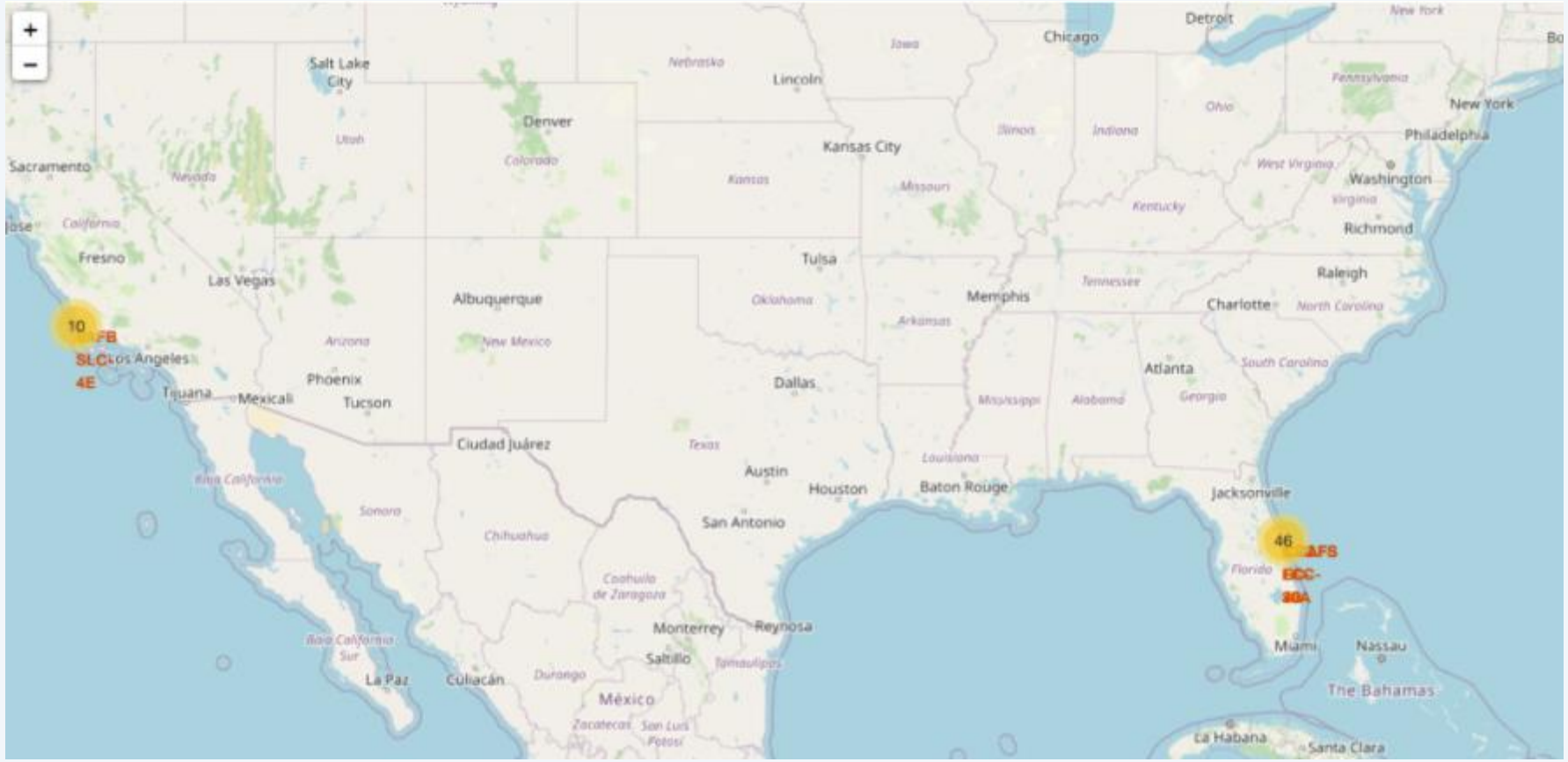
Two of them in the state of Florida, and one in the state of California.

We could already see that they are near the coast.



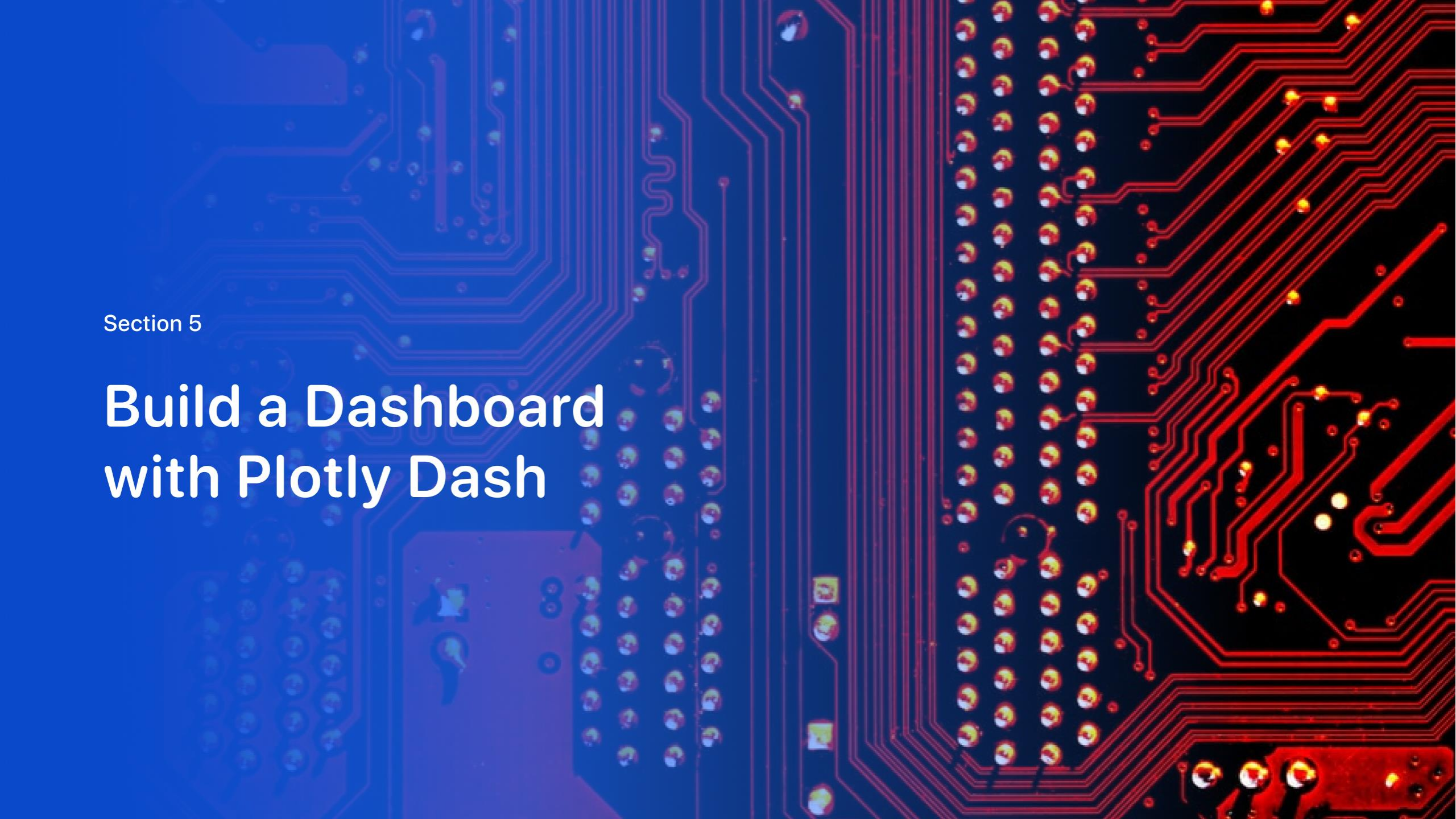
Folium Map with Clusters of the missions.

- Here we put clusters on each of the launch sites containing the number of missions on each of them.
- The launch site in california has the fewest missions.



Folium Map with missions number colored

- Missions where the landing was successful are colored green, missions where the landing was a failure are colored red.



Section 5

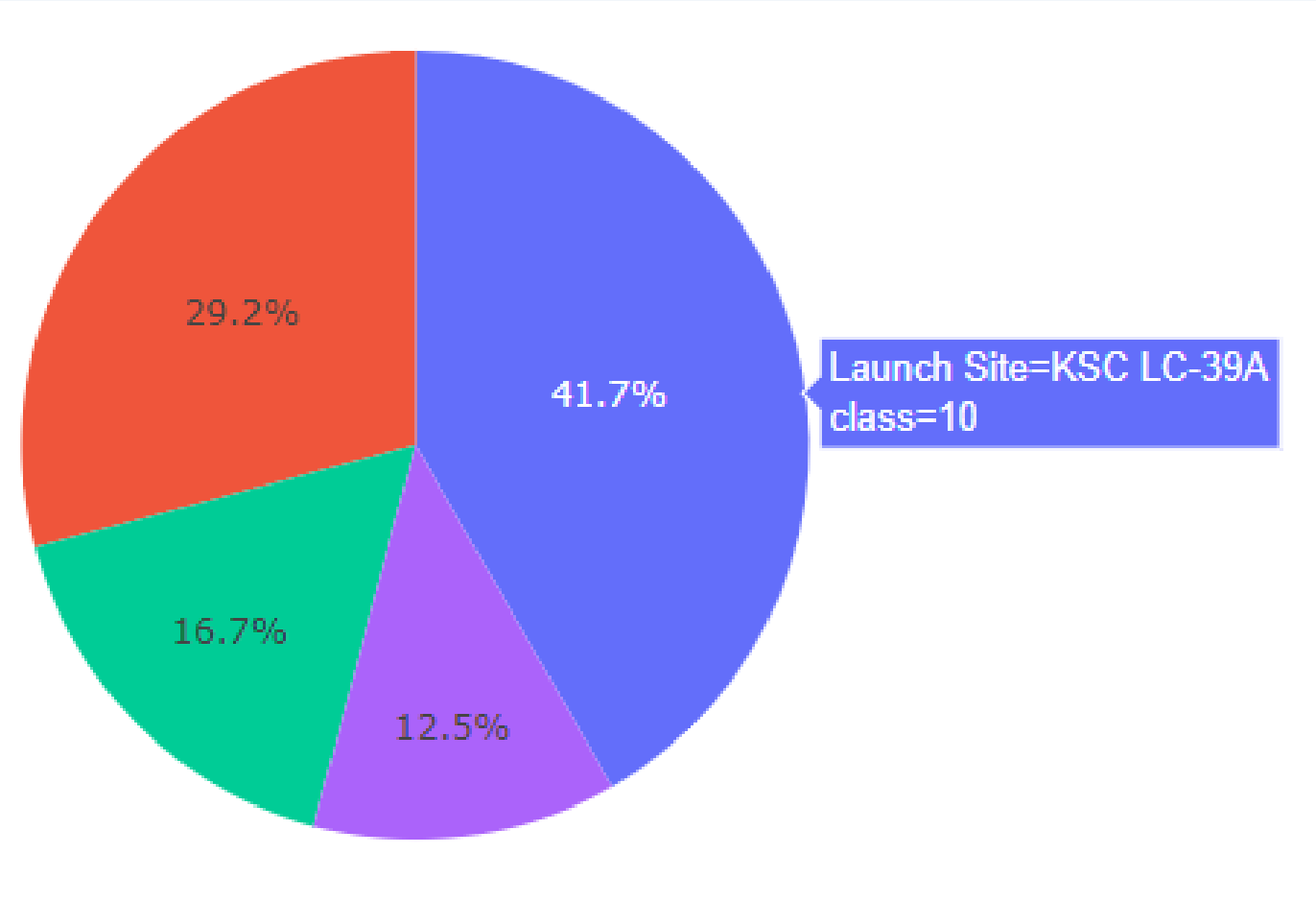
Build a Dashboard with Plotly Dash

Dashboard Success Count

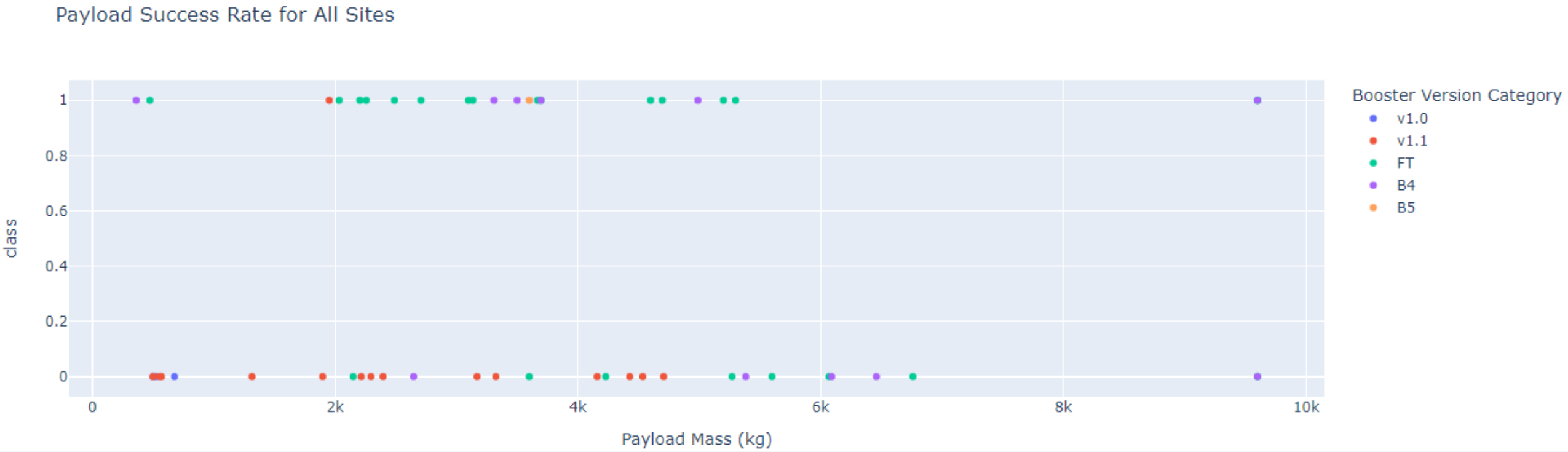
- Here we have the landing success percentage for each launch site.
- Note that here is the breakdown of overall success for each launch site.

Dashboard: Highest success

The launch site with the highest success rate is KSC LC-39A, with 10 missions where the landing was successful.



Dashboard Scatter Plot



Note that the most successful booster version category with a low Payload Mass was the FT category.

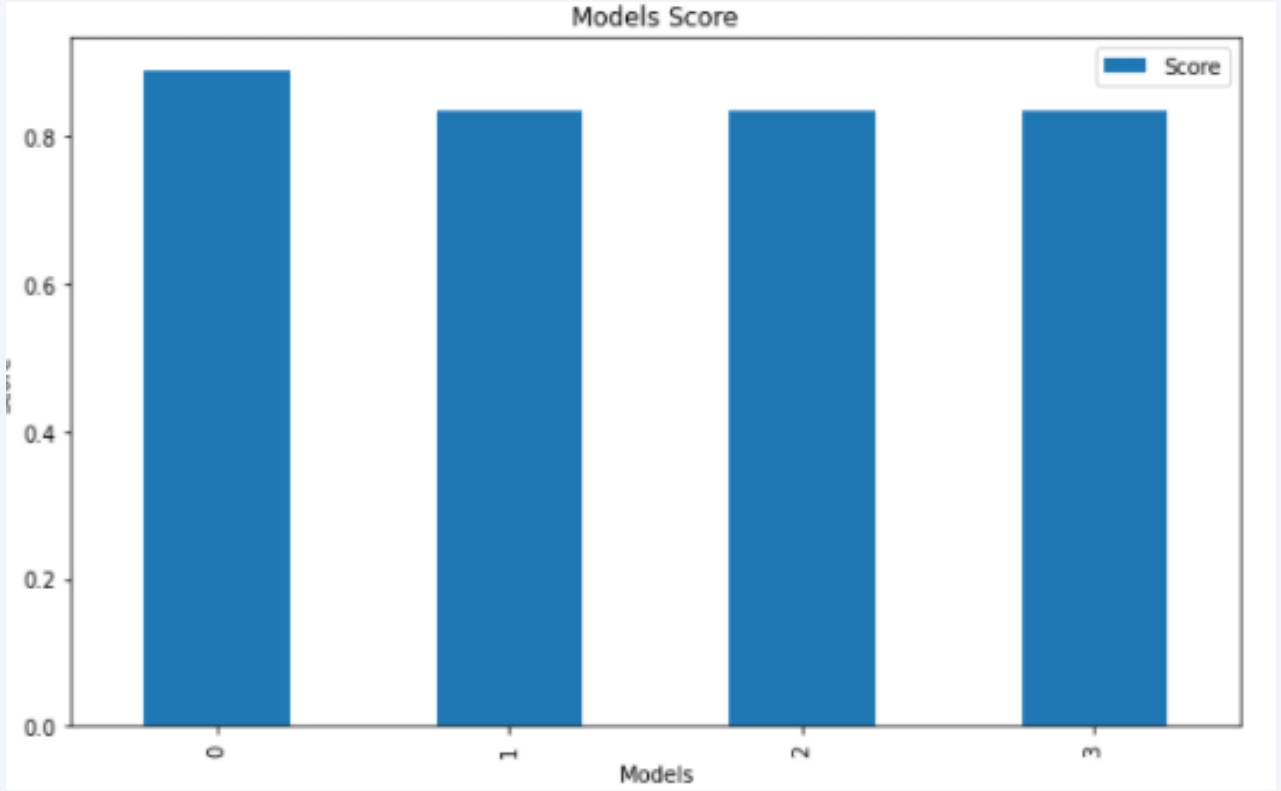
Section 6

Predictive Analysis (Classification)

Classification Accuracy

Out[47]:

	Models	Score
0	Decision Tree	0.888889
1	Logistic Regression	0.833333
2	KNN	0.833333
3	SVM	0.833333



- See the accuracy score of each model used.
- We conclude that the best classification model for this case study is the Decision Tree algorithm with 88% of accuracy.

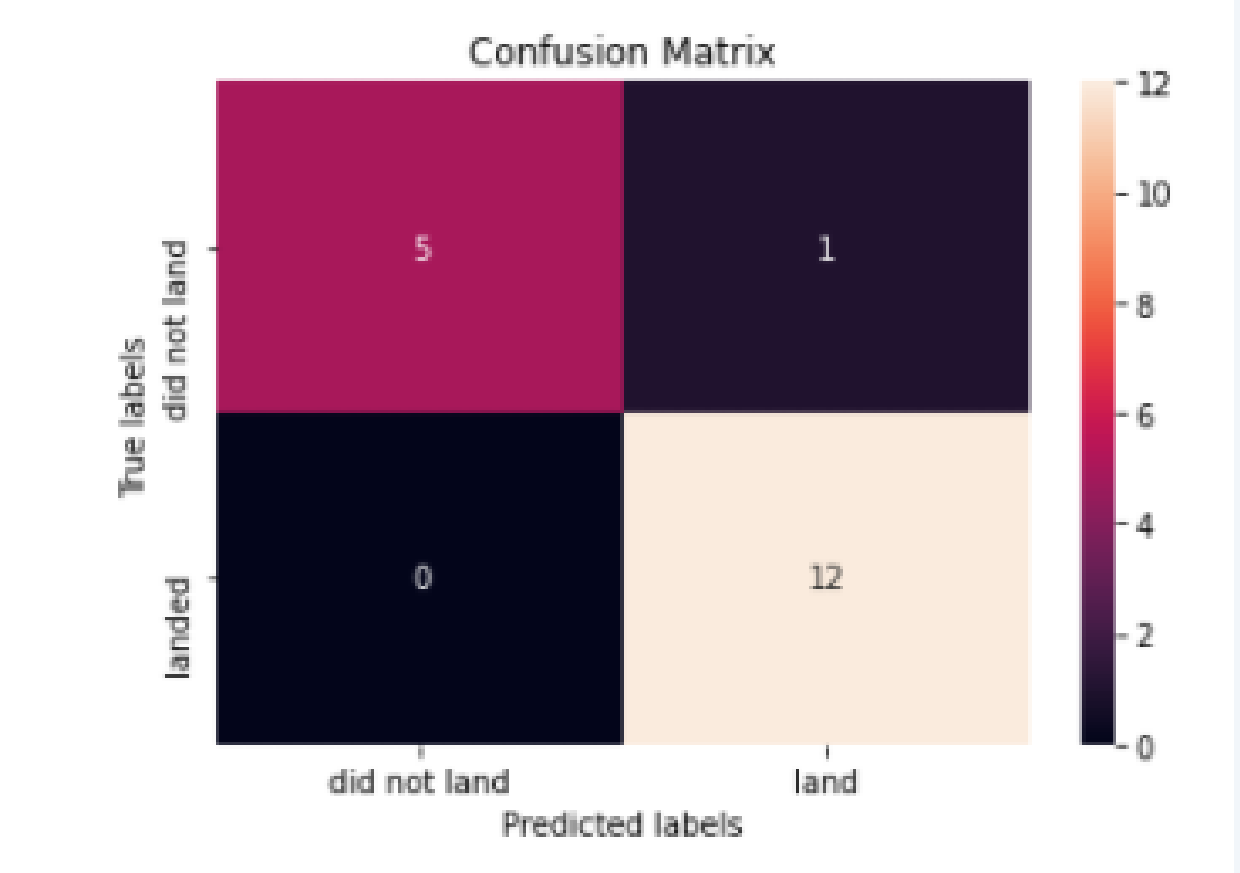
Confusion Matrix: Decision Tree

The model classified 5 instances as "did no t land" and which actually were "did not l and".

The model classified 1 instances as "land" and which actually were "did not land".

The model classified 0 instances as "did not land" that were actually "land".

The model classified 12 instances as "land" that were actually "land".



Our model did very well for both classes, missing only one record for class 0.

Conclusions

- It was quite clear that in the overall, rocket landing success is increasing every year.
- Which orbit the rocket is going to influences the landing outcome a lot.
- All launch sites had a high success percentage depending on Payload Mass.
- For each Launch site, there is a Payload that increases the chances of landing success.
- Pay attention to the orbits that had 100% landing success.
- The GEO orbit had a very poor performance, being easier to flip a coin to know if the landing will be successful or not.
- Droneship failure was the most common among landing failures and also the most common among landing successes.

GitHub URL of the completed project:

Data Science complete project

Thank you!