DATA SCIENCE PROYECT
THE SPACE RACE

Miguel Angel Ortiz del Burgo

13.04.2025

IBM **SkillsBuild**

OUTLINE

- Executive Summary
- Introduction
- Methodology
- Conclusion
- Appendix

EXECUTIVE SUMMARY

- Summary of methodologies
    - Data Collection API / Web Scraping
    - Data Wrangling
    - Exploratory Data Analysis with SQL and Visualization
    - Interactive Visual Analysis with Folium
    - Machine Learning Prediction
- Summary of all results
    - Exploratory Data Analysis result
    - Interactive Analysis in screenshots
    - Predictive Analytics result

INTRODUCTION

- **Background**

SpaceX advertises Falcon 9 rocket launches on its website at a cost of 62 million dollars, while other providers charge upwards of 165 million dollars per launch. A major part of these savings comes from SpaceX's ability to **reuse the first stage of the rocket**, significantly reducing costs.

- **Business Problem**

SpaceY wants to **compete with SpaceX** in the commercial spaceflight market. To do so, we need to answer some key questions:

- What factors make a rocket more likely to **land successfully**?
- Can we **predict the success rate** of Falcon 9 rocket landings using machine learning?
- Are Falcon 9 rockets **landing more successfully now** compared to when they first started?

METHODOLOGY

- Data Colection
- Data Wrangling
- EDA with SQL results
- EDA with data Visualization
- Visualitation using Folium and Dash

DATA COLLECTION WITH APIS (SPACEX)

We used the **provided SpaceX API** to collect, process, and clean the data in order to obtain relevant information about Falcon 9 rocket launches.
The data was retrieved in JSON format and then transformed into structured **dataframes** for further analysis.
Below, we illustrate the process of retrieving the data and loading it into a pandas dataframe for wrangling and exploration

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

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

```
[9]: data = pd.json_normalize(response.json())
```

After looking at the data we can observe that the data is still unclear so we have to extract the meaings of its columns into something a bit more clear for us

```
[10]: data.head(2)
```

[10]:

	static_fire_date_utc	static_fire_date_unix	tbd	net	window	rocket	success	details	crew	ships	capsules	payloads
0	2006-03-17T00:00:00.000Z	1.142554e+09	False	False	0.0	5e9d0d95eda69955f709d1eb	False	Engine failure at 33 seconds and loss of vehicle	[]	[]	[]	[5eb0e4b5b6c3bb0006eeb1e1]
1	None	NaN	False	False	0.0	5e9d0d95eda69955f709d1eb	False	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 to recover first stage	[]	[]	[]	[5eb0e4b6b6c3bb0006eeb1e2]

```
[12]: data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]
data = data[data['cores'].map(len)==1]
data = data[data['payloads'].map(len)==1]
data['cores'] = data['cores'].map(lambda x : x[0])
data['payloads'] = data['payloads'].map(lambda x : x[0])
data['date'] = pd.to_datetime(data['date_utc']).dt.date
data = data[data['date'] <= datetime.date(2020, 11, 13)]
```

```
[21]: getBoosterVersion(data)
```

```
[22]: getLaunchSite(data)
```

```
[23]: getPayloadData(data)
```

```
[24]: getCoreData(data)
```

After processing the data we have this clear frame that we can sort to select only hte BoosterVersion "Falcon 9"

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial
0	1	2006-03-24	Falcon 1	20.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0	Merlin1A
1	2	2007-03-21	Falcon 1	NaN	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0	Merlin2A
2	4	2008-09-28	Falcon 1	165.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0	Merlin2C
3	5	2009-07-13	Falcon 1	200.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0	Merlin3C
4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0003

Heres a link to the notebook : Notebook - APIs

DATA COLLECTION WITH WEBSCRAPING (WIKIPEDIA)

In this step, we used **BeautifulSoup4** to web scrape the Falcon 9 Wikipedia page that contains **landing outcomes** for various launches.
The goal was to extract additional data not available through the SpaceX API, specifically related to **booster landing success**, landing types, and sites.
Below, we show the process of identifying and extracting the target HTML table, as well as retrieving its column structure.

```
[5]: response = requests.get(static_url)
```

```
[6]: soup = BeautifulSoup(response.content, 'html.parser')
```

```
[8]: html_tables = soup.find_all('table')
```

```
[9]: first_launch_table = html_tables[2]
```

```
[10]: column_names = []

for column in first_launch_table.find_all("th"):
    result = extract_column_from_header(column)
    if result is not None and len(result) > 0:
        column_names.append(result)
```

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

After identifying the column headers, we proceeded to **extract the table content** and convert it into a structured **dataframe** using pandas.

```
[13]: launch_dict= dict.fromkeys(column_names)

del launch_dict['Date and time ( )']
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'] = []
launch_dict['Version Booster']=[]
launch_dict['Booster landing']=[]
launch_dict['Date']=[]
launch_dict['Time']=[]
```

```
[15]: df= pd.DataFrame({ key:pd.Series(value) for key, value in launch_dict.items() })
```

Heres the link to check the notebook:
[Notebook - Webscraping]()

```
extracted_row = 0
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders collapsible")):
    for rows in table.find_all("tr"):
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
        else:
            flag=False
        row=rows.find_all('td')
        if flag:
            extracted_row += 1
            launch_dict["Flight No."].append( flight_number) # Flight Number value
            datatimelist=date_time(row[0])

            date = datatimelist[0].strip(',') # Date value
            launch_dict["Date"].append(date)

            time = datatimelist[1] # Time value
            launch_dict["Time"].append(time)

            bv=booster_version(row[1]) # Booster version
            if not(bv):
                bv=row[1].a.string
            launch_dict["Version Booster"].append(bv)

            launch_site = row[2].a.string # Launch Site
            launch_dict["Launch site"].append(launch_site)

            payload = row[3].a.string # Payload
            launch_dict["Payload"].append(payload)

            payload_mass = get_mass(row[4]) # Payload Mass
            launch_dict["Payload mass"].append(payload_mass)

            orbit = row[5].a.string # Orbit
            launch_dict["Orbit"].append(orbit)

            if row[6].a == None:
                customer = "Various"
            else :
                customer = row[6].a.string
            launch_dict["Customer"].append(customer) # Customer
```

DATA WRANGLING

After obtaining the data, we were able to **clean and transform it into a usable format**.

As a first step, we explored the structure and contents of each column to understand the **data types** and what each feature represents.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 90 entries, 0 to 89
Data columns (total 17 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   FlightNumber    90 non-null     int64
 1   Date            90 non-null     object
 2   BoosterVersion  90 non-null     object
 3   PayloadMass     90 non-null     float64
 4   Orbit           90 non-null     object
 5   LaunchSite      90 non-null     object
 6   Outcome         90 non-null     object
 7   Flights         90 non-null     int64
 8   GridFins        90 non-null     bool
 9   Reused          90 non-null     bool
 10  Legs            90 non-null     bool
 11  LandingPad      64 non-null     object
 12  Block           90 non-null     float64
 13  ReusedCount     90 non-null     int64
 14  Serial          90 non-null     object
 15  Longitude       90 non-null     float64
 16  Latitude        90 non-null     float64
dtypes: bool(3), float64(4), int64(3), object(7)
memory usage: 10.2+ KB
```

After reviewing the dataset, we focused on the key **categorical features (**`Launch Site`, `Orbit`, and `Outcome`**)** which play a central role in our analysis.

```
[5]: df.groupby("LaunchSite")["LaunchSite"].value_counts()
```

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

```
[6]: df.groupby("Orbit")["Orbit"].value_counts()
```

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

To prepare the data for classification, we **simplified the landing outcome** column by mapping all possible results into a binary value:

- `1` → Successful landing
- `0` → Unsuccessful or no landing

This binary outcome will serve as the **target variable** for our machine learning model.

```
landing_class = df['Outcome'].apply(lambda outcome: 0 if outcome in bad_outcomes else 1)
```

```
[7]: landing_outcomes = df.groupby("Outcome")["Outcome"].value_counts()
     landing_outcomes
```

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

```
[12]: df['Class']=landing_class
      df[['Outcome','Class']].head(8)
```

[12]:

	Outcome	Class
0	None None	0
1	None None	0
2	None None	0
3	False Ocean	0
4	None None	0
5	None None	0
6	True Ocean	1
7	True Ocean	1

To check the notebook : [Notebook Wrangling]

EDA WITH SQL

We used **SQL queries** to explore and summarize the dataset

```
%sql SELECT AVG(PAYLOAD_MASS__KG_) FROM SPACEXTABLE WHERE Booster_Version LIKE "%F9 v1.1%" LIMIT 1
```

```
 * sqlite:///my_data1.db
Done.
```

AVG(PAYLOAD_MASS__KG_)
2534.6666666666665

```
%sql SELECT Landing_Outcome,Count(Landing_Outcome) FROM SPACEXTABLE GROUP BY Landing_Outcome
```

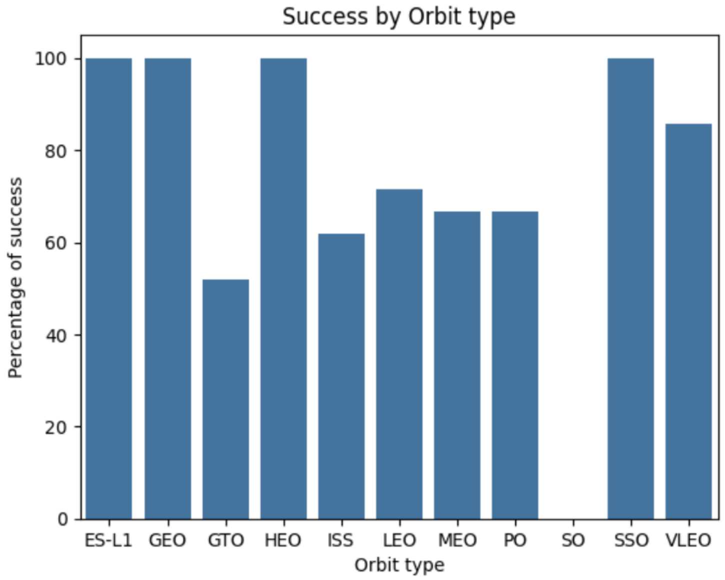
```
 * sqlite:///my_data1.db
Done.
```

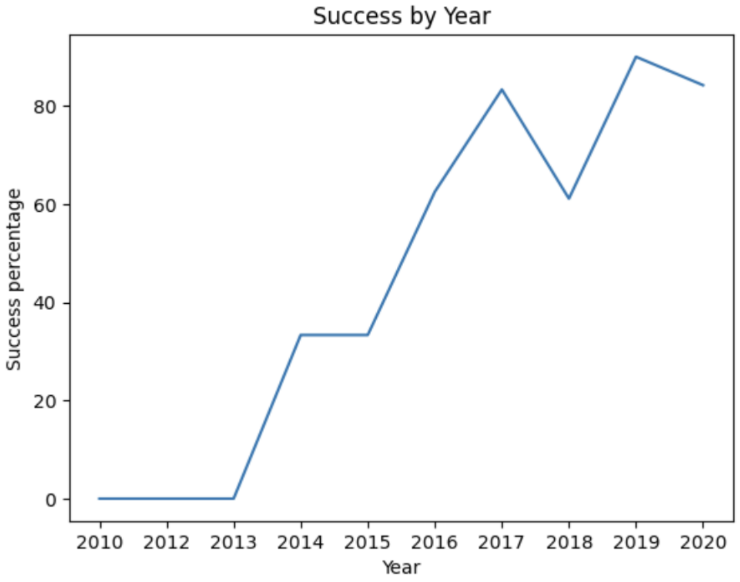
Landing_Outcome	Count(Landing_Outcome)
Controlled (ocean)	5
Failure	3
Failure (drone ship)	5
Failure (parachute)	2
No attempt	21
No attempt	1
Precluded (drone ship)	1
Success	38
Success (drone ship)	14
Success (ground pad)	9
Uncontrolled (ocean)	2

To observe the rest of the querys check the notebook :
Notebook - SQL

EDA WITH DATA VISUALIZATION

We used **Jupyter Notebook** along with libraries such as **Matplotlib**, and **Pandas** to explore and visualize the dataset.
Here we can visualize the % of success by year and % of success by type of orbit.



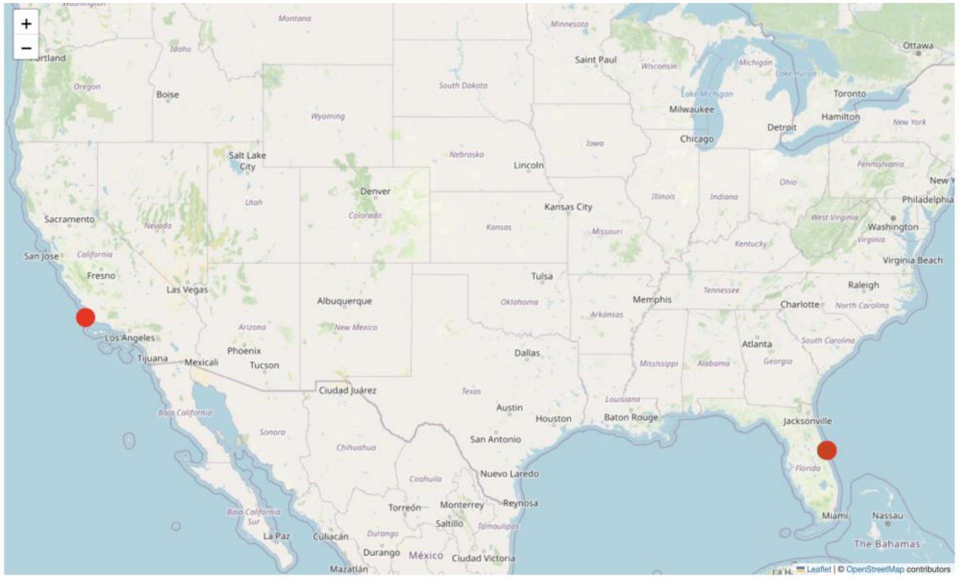


To observe the rest of the plots check the notebook : [Notebook - Visualization]

VISUALITATION USING FOLIUM

Using **Folium**, we created **interactive maps** to visualize:

- The geographical locations of Falcon 9 launches
- The landing outcomes at each site



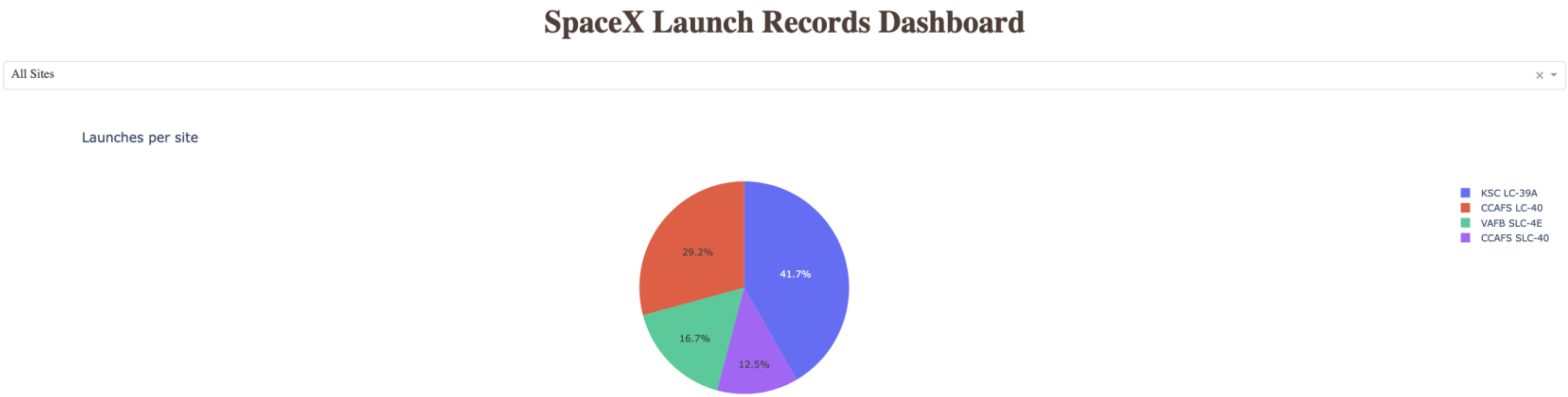
To observe the rest of the maps check the notebook :
[Notebook - Folium]

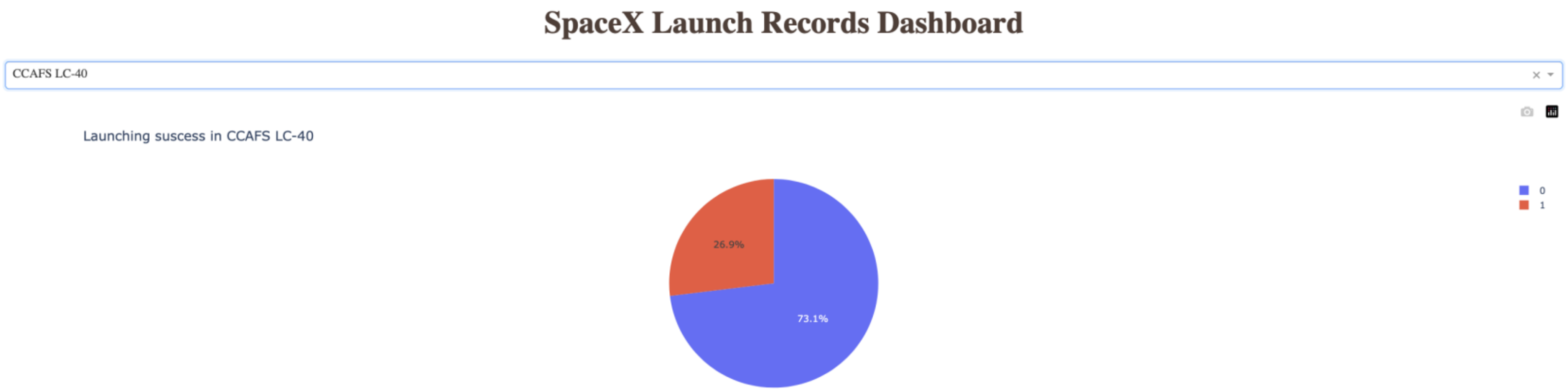
VISUALITATION USING DASH

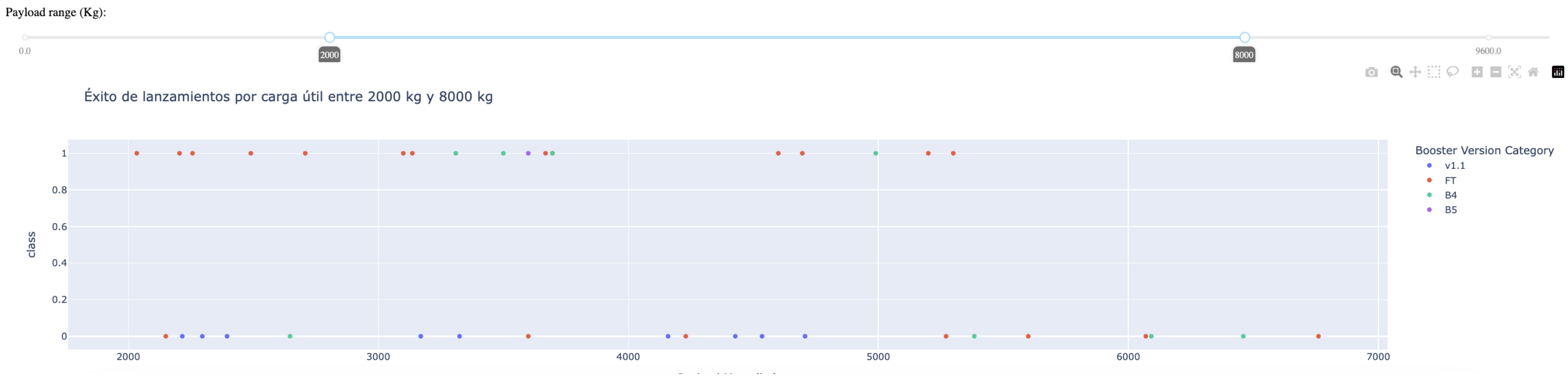
Using **Plotly Dash** to create an interactive dashboard for data exploration.

The dashboard allows users to:

- Select launch sites and view success rates
- Explore the relationship between **payload mass** and **landing outcome**
- Filter and compare launch data using interactive components like dropdowns and sliders







To observe the pyhton check the link :
[Dash app]

MACHINE LEARNING PREDICTION

After exploring and preparing the data, we built a **predictive model** to estimate the **success probability of a rocket landing**.

We tested four classification algorithms:

- Decision Tree
- Random Forest Classifier
- Support Vector Machine (SVM)
- Logistic Regression

Each model was trained and evaluated to determine which provided the most accurate predictions.

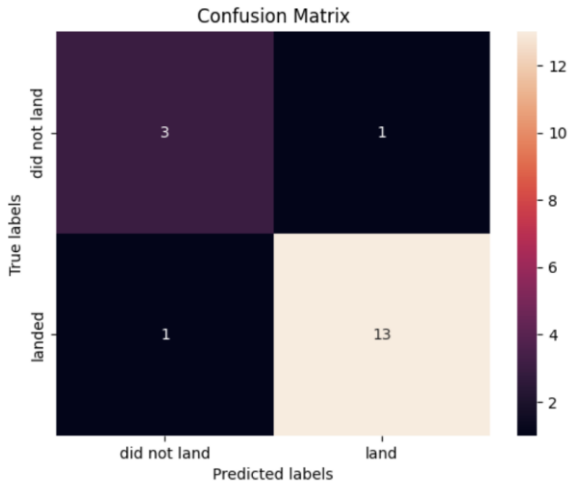
After testing all models we found the SVM as the best model with an acuracy of 88.88%

```
print("The best method performance is", best_method, " with a performance of ",np.round((best_score * 100),2), "%." )
print ("Logistic Regression : ",log_prediction)
print ("SVM Regression : ",svm_prediction)
print ("Tree Classifier Regression : ",tree_prediction)
print ("Knn Classifier : ",knn_prediction)
```

```
The best method performance is SVM  with a performance of  88.89 %.
Logistic Regression :  0.8178571428571427
SVM Regression :  0.8888888888888888
Tree Classifier Regression :  0.7222222222222222
Knn Classifier :  0.5
```

To observe the notebook check the link :
[Notebook Machine Learning]

```
yhat_svm=svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat_svm)
```



CONCLUSION

After looking at the slides we can finally answer those questions we made previusly .

- What factors make a rocket more likely to **land successfully**?

  Rockets with an orbit type of ES-L1, GEO , HEO or SSO are more likely to land,

- Can we **predict the success rate** of Falcon 9 rocket landings using machine learning?

  Yes , with a SVM Model that predicts with an 88% acuracy

- Are Falcon 9 rockets **landing more successfully now** compared to when they first started?

  Yes , over the years the landing success rate has increased up to 85% aprox