Winning Space Race with Data Science

Pablo Martin
24th of January 2022

Outline

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

Executive Summary

- **Summary of methodologies**
  - Data collection through an **API** and **Web Scraping**
  - Data wrangling
  - Exploratory data analysis (EDA) with **SQL** and **Data Visualization**
  - Interactive visual analytics with **Folium**
  - Dashboard with **Plotly Dash**
  - **Machine Learning prediction**
- **Summary of all results**
  - Exploratory data analysis results
  - Interactive analytics dashboard screenshots
  - Predictive analysis results

Introduction

- **Project background and context**

The key of SpaceX's success is because its rocket launches are relatively inexpensive (62 million $) compared to the other providers that its cost goes up to 165 million $.

Much of the savings is because SpaceX can reuse the part of the rocket that is used on the first stage of the launch.

Therefore, if we can determine if the first stage will land, we can determine the cost of a launch.

- **Problems you want to find answers**
  - **What price will it be for each launch?**

  And if the mission parameters allow us to land the first stage:

  - **What variables and conditions increase the chance of a successful landing?**

Section 1

Methodology

Methodology

- **Data collection methodology:**
  - SpaceX REST API
  - Wikipedia (using Web Scraping)
- **Perform data wrangling:**
  - Firstly, cleanse the data (when using web scraping).
  - Secondly, drop columns that don't contain relevant information that helps tackle the problem.
  - Thirdly, if necessary, apply One-Hot Encoding for transforming categorical variables into numeric ones.
- **Perform exploratory data analysis (EDA) using visualization and SQL.**
- **Perform interactive visual analytics using Folium and Plotly Dash.**
- **Perform predictive analysis using classification models.**
  - When created all the classification models that are going to be used we apply **GridSearchCV** for tunning the hyperparameters and search for the best model that fits the test data.

Data Collection – SpaceX API

1. Get response from API.

```
spacex_url="https://api.spacexdata.com/v4/launches/past"
response = requests.get(spacex_url)
```

2. Transform the response to a JSON and apply it to a dataframe.

```
data = pd.json_normalize(response.json())
```

3. Apply functions to make the data even and meaningful.

```
# 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.
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)]
# Call custom functions
getBoosterVersion(data)
getLaunchSite(data)
getPayloadData(data)
getCoreData(data)
```

4. Add the lists to a dictionary and apply it to a dataframe.

```
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}
```

GitHub [Data Collection Notebook]

Data Collection - Scraping

1. Get HTML response from wikipedia and parse it into a Soup object.

```
static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"
response = requests.get(static_url)
```

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

2. Extract and list column headers from the table.

```
column_names = []
html_tables = soup.find_all('table')[2]
for x in first_launch_table.find_all('th'):
    x = extract_column_from_header(x)
    if x != None and len(x) > 0:
        column_names.append(x)
```

3. Create a dictionary of column names.

```
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']=[]
launch_dict
```

4. Iterate through all the rows of the Soup object and add its elements to their corresponding part of the dictionary which, after the iteration, will be transformed to a pandas dataframe.

(See the 13th block of code of the notebook)

GitHub [Data Collection with Web Scraping Notebook]

Data Wrangling

The data collected directly from the sources used in this project is mostly structured and useful, but in the following cases data needed to be treated:

- **Data Collection Notebook** *(2-6, 13, 16, 18-21,24,25,27):* The date is converted, values retrieved and assigned to the corresponding lists, rows filtered to only show Falcon9 launches, multiple core rows removed and replaced ***NaN*** values with the mean of its column...
- **Data Collection with Web Scraping** *(3,6,8-10, 12, 13):* Parse the data to a soup object, locate the first table, extract its columns, iterate through the table processing the data and adding it to the data frame.
- **Exploratory Data Analysis** *(9, 10):* Create an additional column (Class) with a simplified landing outcome [0 or 1].
- **EDA with Data Visualization** *(6,9, 10, 12, 13):* Created columns to represent % of the successes on each orbit and year, created dummy variables and changed its column type.
- **Machine Learning Prediction** *(6, 7):* Dataset X is standardized for its future fitting of the models.
- **Interactive Visual Analytics with Folium** *(11):* Created additional column to assign color depending on the mission success.

Data Collection Notebook

Data Collection with Web Scraping Notebook

Exploratory Data Analysis Notebook

EDA with Data Visualization Notebook

Machine Learning Prediction

Interactive Visual Analytics with Folium Notebook

GitHub

EDA with Data Visualization

Scatter Plot

- **Flight nº** *vs* **Payload mass**
- **Flight nº** *vs* **Launch site**
- **Payload mass** *vs* **Launch site**
- **Flight nº** *vs* **Orbit**
- **Payload mass** *vs* **Orbit**

A scatter plot allows us to see the frequency/relation of the variable X through Y.

Bar Chart

- **Orbit** *vs* **Success Rate**

A bar chart allows us to compare easily different groups of numerical or categorical variables to a numerical axis.

Line Chart

- **Year** *vs* **Success Rate**

A line chart shows the tendency of a numeric variable mostly through a period of time.

GitHub [EDA with Data Visualization Notebook]

EDA with SQL

- Retrieve all the *unique* launch sites.
- Get 5 records of launches which launch sites start with *CCA.*
- Get the total payload mass carried by all *NASA (CRS)* launches.
- Calculate the average payload mass carried by boosters version *F9 v. 1. 1*.
- List the date on when the *landing* was *successful* on a *ground pad*.
- Retrieve the versions of the boosters that *landed successfully* on a *drone ship* and its *mass* is between *4000* and *6000 Kg*.
- List all the possible *mission outcomes* and the *nº of times* that happened.
- Get the booster versions that *carried* the *maximum payload mass*.
- Retrieve all the *failed drone ship landing outcomes* in *2015*.
- Rank all the types of *landing outcomes* and *count* how many times happened *each one* between *04-06-2010* and *20-03-2017*.

EDA with SQL Notebook

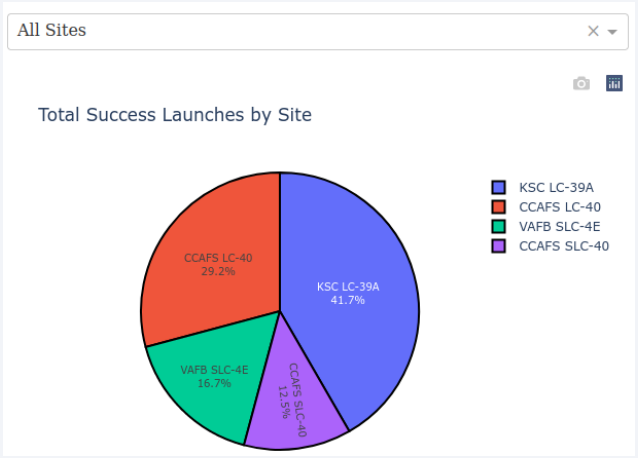
Build an Interactive Map with Folium

- Firstly, we established the markers using the coordinates of the launch sites and created a circle around them indicating the name of the launch site.
- Secondly, we created a cluster of markers in each launch site, containing each launch with an icon color depending if it was successful or not (Green or Red).
- Finally, added custom coordinates of certain locations (Los Angeles, Bernina Express, coastline and Route 66) and linked them with the CCAFS LC-40 launch site while showing the distance of each point from it.

Interactive Visual Analytics with Folium Notebook

Build a Dashboard with Plotly Dash

- **Dropdown:** Allows you to choose what launch site's data want to visualize.
- **Pie chart:** Visualize the percentage of successful launches of all launch sites and the rate of successful/failed launches by site.



GitHub

[Dashboard Plotly Dash]

- **Slider:** Lets you filter the launches on the scatter plot by payload mass.
- **Scatter plot:** At a glance you can easily identify which launches failed or not, how much payload they were carrying and its booster version.

Predictive Analysis (Classification)

1. Choose the Model, pass it and the parameters that you want to test into the GridSearch and fit the data into it to train it.

```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}
lr=LogisticRegression()
logreg_cv = GridSearchCV(lr, parameters, cv=10)
logreg_cv.fit(X_train, Y_train)
```

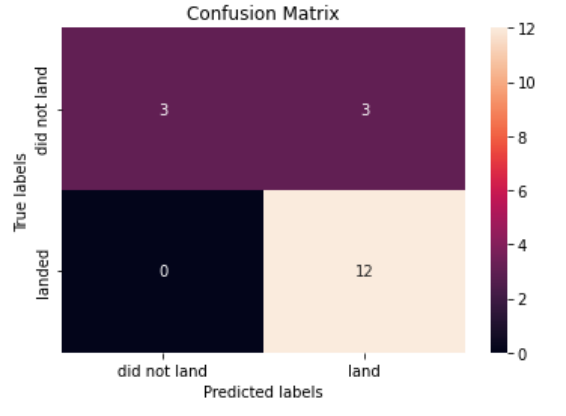
2. See the best score and the parameters used.

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

```
tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8464285714285713
```

3. Predict the target variables and plot a confusion matrix to find the false positive/negatives found between the real and predicted labels.

```
yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



4. Iterate through each model and find the max accuracy.

```
methods = [logreg_cv, svm_cv, tree_cv, knn_cv]
scores = {str(method.estimator)[:-2]: float(method.score(X_test, Y_test)) for method in methods}
print("The best method is {} with an accuracy of {}".format(max(scores, key=scores.get),max(scores.values())))
df=pd.DataFrame.from_dict(scores, orient="index").reset_index().rename(columns={'index':'Model', 0:'Accuracy'})
df.sort_values(by='Accuracy', ascending=False).head().style.hide_index()
```

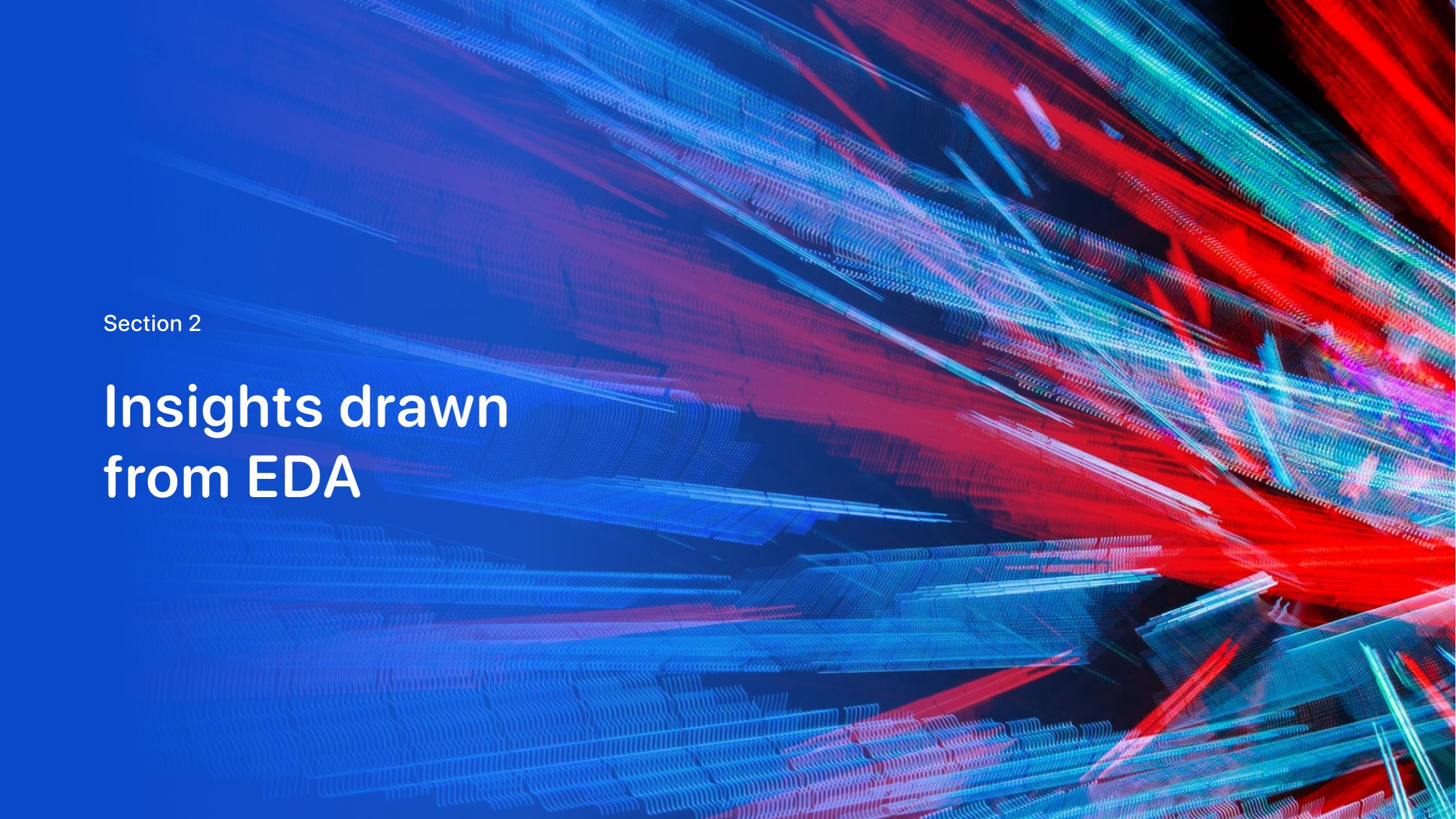
```
The best method is DecisionTreeClassifier with an accuracy of 0.8888888888888888
```

Model	Accuracy
DecisionTreeClassifier	0.888889
LogisticRegression	0.833333
SVC	0.833333
KNeighborsClassifier	0.833333

GitHub [Machine Learning Prediction]

Results

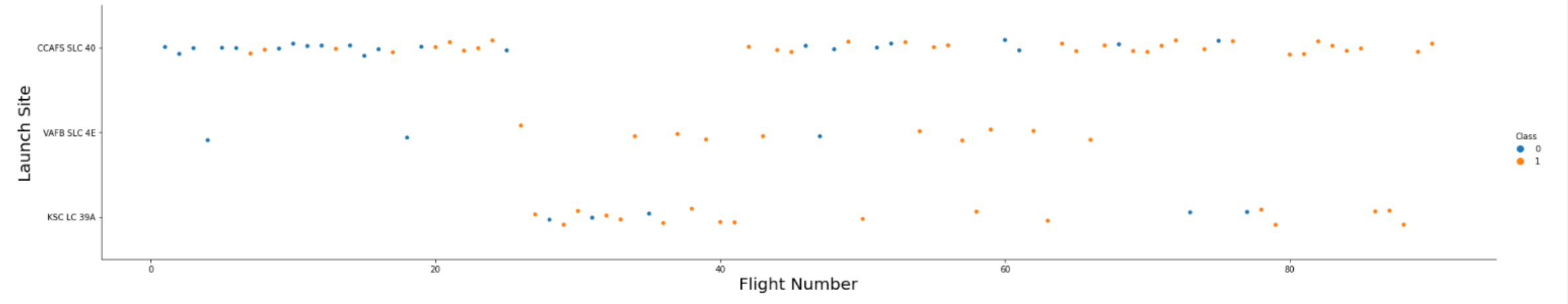
- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results



Section 2

Insights drawn from EDA

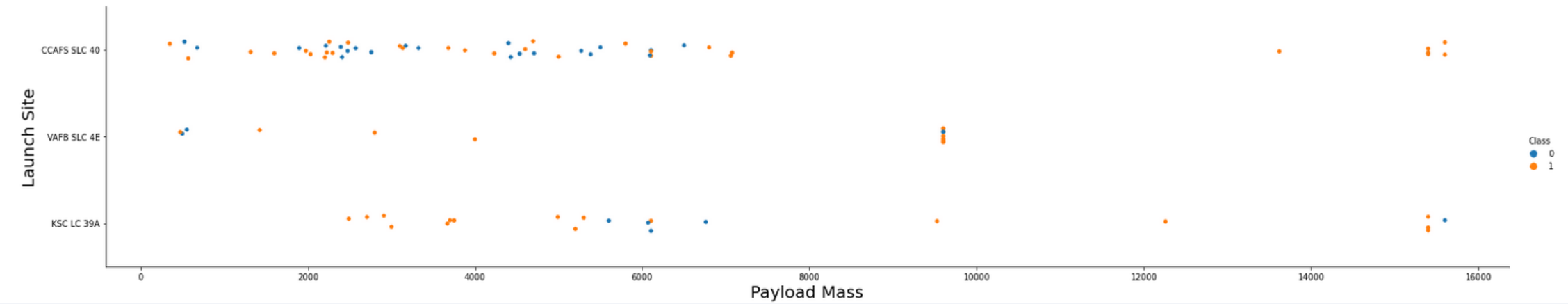
Flight Number vs. Launch Site



We can see that everytime a launch is made on a launch site, it increases that chances of success for the next one.

Due to experience that is gained through every launch and working on that determined launch site.

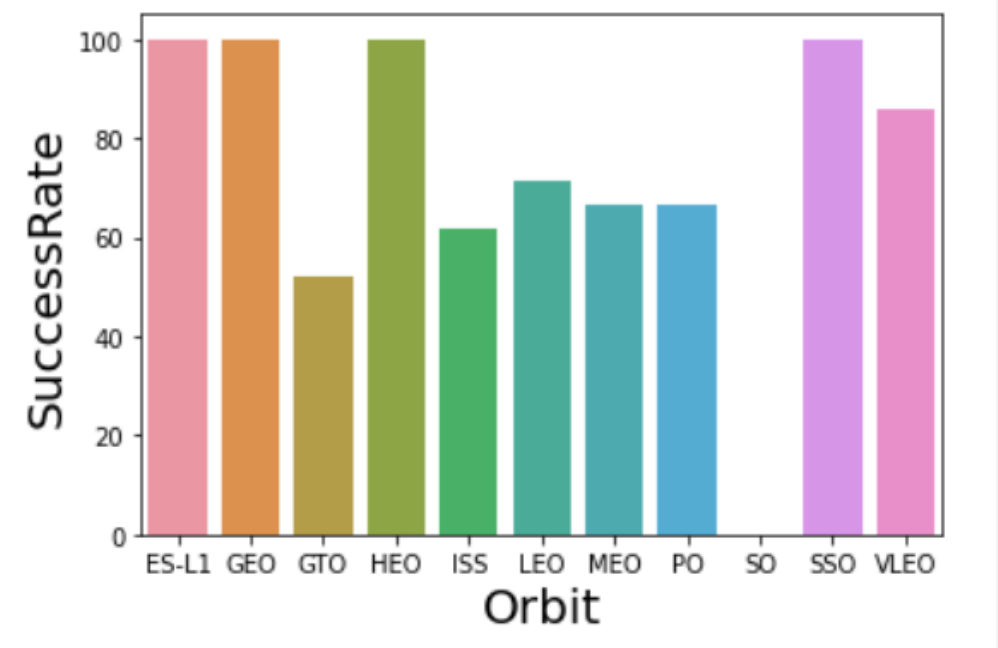
Payload vs. Launch Site



If we compare this table to the one before, we can see that most of the failed launches in *CCAFS SLC 40* were at the initial launches and here we can see that in the same launch site, the failed ones are mostly with a payload between 2200 and 5250.

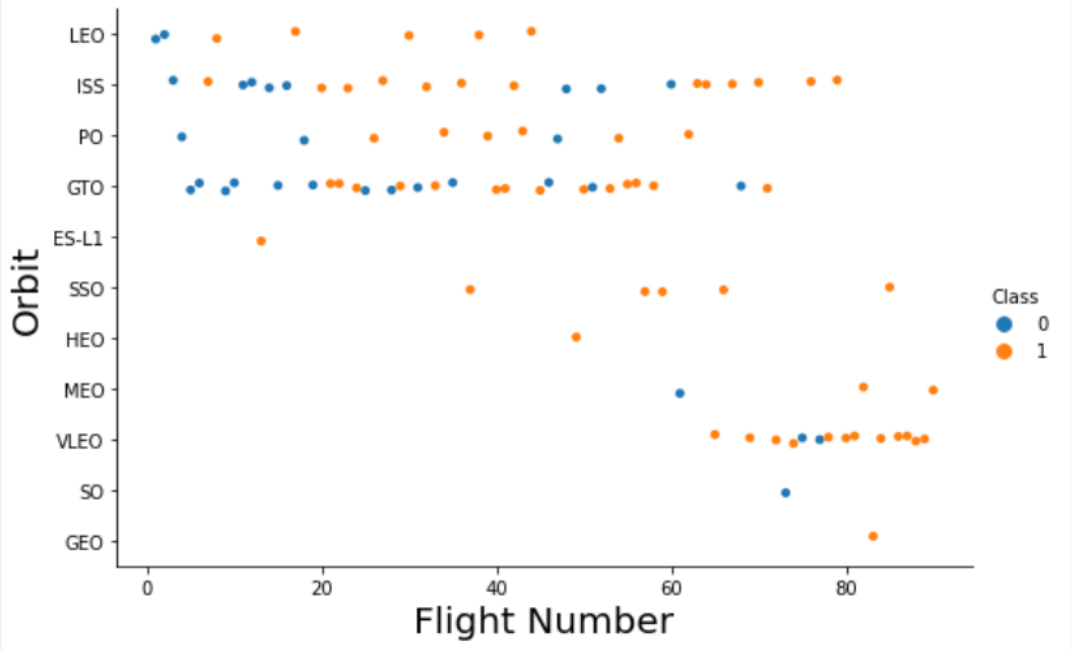
So we can deduce that they started with a greater payload and then decided to reduce it, focusing themselves on landing first and then increasing its payload.

Success Rate vs. Orbit Type



The orbits with the highest success rate are SSO, ES-L1, GEO, HEO.

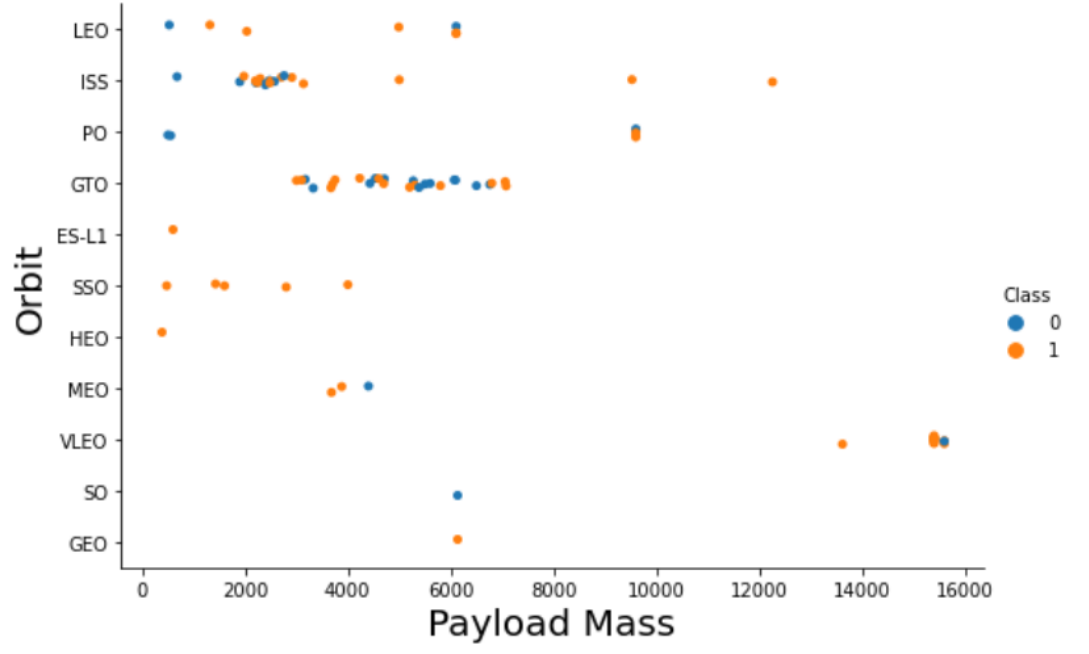
Flight Number vs. Orbit Type



We can see that the orbits with most launches are the closest to the Earth such as:

- International Space Station *(ISS)*
- Low Earth Orbit *(LEO)*
- Very Low Earth Orbit (VLEO)
- Geostationary Transfer Orbit (GTO)

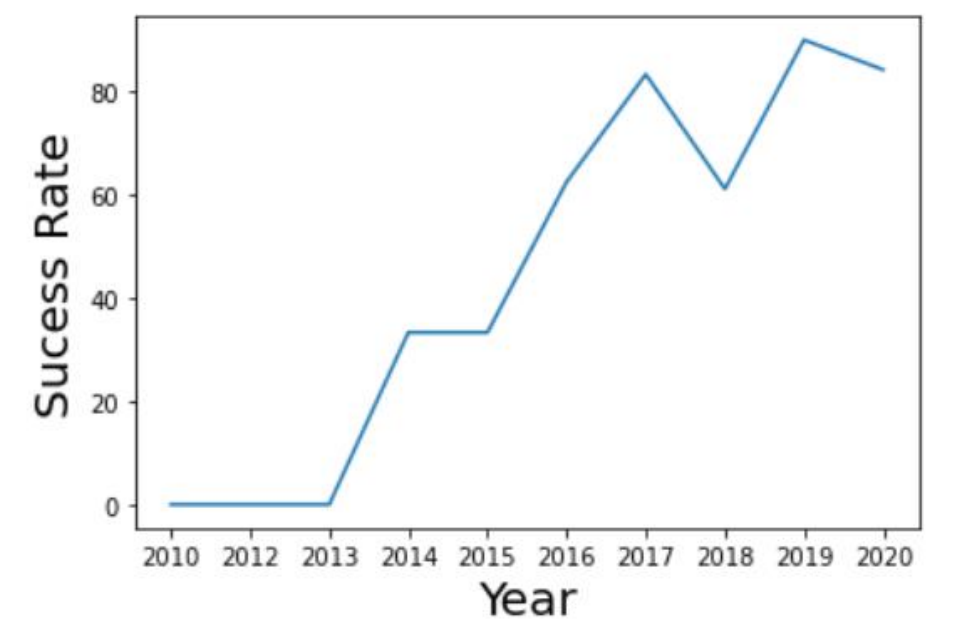
Payload vs. Orbit Type



The outlier with the greatest payload mass by far is in the VLEO orbit.

Meaning, such a heavy payload can only be transported to a short distance at this moment.

Launch Success Yearly Trend



We can see an exponential increase up to almost 100% success rate.

This is due to the accumulated experience of try and error over the initial years.

All Launch Site Names

```
%sql select distinct(launch_site) from spacex
```

Done.

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

Retrieved the launch sites using *DISTINCT.*

The keyword distinct is used for selecting only once the non-repeatable values from the column.

Launch Site Names Begin with 'CCA'

```
%sql select * from spacex where launch_site like 'CCA%' limit 5
```

Done.

DATE	time__utc_	booster_version	launch_site	payload	payload_mass__kg_	orbit	customer	mission_outcome	landing__outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2012-05-22	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

Retrieved all the columns of which the launch site start its name with the letters *CCA*.

The keyword *LIKE* is used to work with operators and strings, the symbol *%* is used as a wildcard and the *LIMIT* is used to retrieve only the specified amount of rows.

Total Payload Mass

```
%sql select sum(payload_mass__kg_) as "Total mass carried by 'NASA (CRS)'" from spacex where customer like 'NASA (CRS)'
```

Done.

Total mass carried by 'NASA (CRS)'
45596

Did a sum of all the payloads of the column payload mass which its customer is NASA.

In this case we needed to apply the keyword *SUM* that does the sum of all values on the applied column.

Average Payload Mass by F9 v1.1

```
%sql select avg(payload_mass__kg_) as "Average payload mass carried by booster 'F9 v.1.1'" from spacex where booster_version like 'F9 v1.
```

Done.

Average payload mass carried by booster 'F9 v.1.1'
2928

Did the average of the payload mass carried by the boosters version *F9 v.1.1*.

Here we applied the keyword *AVG* that does the sum of the values on the row and divides it by the number of rows. Alse we needed to apply the LIKE operator.

First Successful Ground Landing Date

```
%sql select min(date) as "First time success landing in ground pad" from spacex where landing__outcome like 'Success (ground pad)'
```

Done.

First time success landing in ground pad
2015-12-22

Retrieved the first successful landing outcome date on ground pad.

The *MIN* keyword can be used on integers but in this case it can be also used on the date and it retrieves the oldest date.

Successful Drone Ship Landing with Payload between 4000 and 6000

select booster_version as "Booster version" from spacex where landing__outcome like 'Success (drone ship)' and payload_mass__kg_ between 4000 and 6000.

```
%sql select booster_version as "Booster version" from spacex where landing__outcome like 'Success (drone ship)' and payload_mass__kg_ bet
```

Done.

Booster version
F9 FT B1022
F9 FT B1026
F9 FT B1021.2
F9 FT B1031.2

Retrieved the version of the boosters where its landing on a drone ship was successful and its payload mass its between 4000 and 6000.

In this case the used the *LIKE* keyword, the keyword *AND* to concatenate conditions and the keyword *BETWEEN* to filter between two integer values.

Total Number of Successful and Failure Mission Outcomes

```
%sql select mission_outcome as "Mission Outcome", count(mission_outcome) as "Nº of times" from spacex group by mission_outcome
```

Done.

Mission Outcome	Nº of times
Failure (in flight)	1
Success	99
Success (payload status unclear)	1

Grouped the mission outcomes and counted each outcome.

The keyword *GROUP BY* is used to group the equal variables on a column and this allow us apply almost any operation keyword on the select part of the statement. In this case I applied the *COUNT* keyword.

Boosters Carried Maximum Payload

```
%sql select booster_version from spacex where payload_mass__kg_=(select max(payload_mass__kg_) from spacex)
```

Done.

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

Used a subquery to select the maximum payload and retrieved the booster versions that carried that payload.

In this case I needed to apply another query that retrieved the max payload found on the same table and choose those which had the same exact payload.

2015 Launch Records

```
%sql select booster_version, launch_site from spacex where year(date)=2015 and landing__outcome like 'Failure (drone ship)'
```

Done.

booster_version	launch_site
F9 v1.1 B1012	CCAFS LC-40
F9 v1.1 B1015	CCAFS LC-40

Retrieved the booster version and the launch site of the failed drone ship landings.

In this case I used the *YEAR* keyword that when applied on a date type column it selects the year inside the date.

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

select landing__outcome as "Landing Outcome", count(landing__outcome) as "Nº of times" from spacex where date between '2010-06-04' and '2017-03-20' group by landing__outcome order by 2 desc

```
%sql select landing__outcome as "Landing Outcome", count(landing__outcome) as "Nº of times" from spacex where date between '2010-06-04' a
```

Done.

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

Grouped all the landing outcomes, counted and sorted them by the number of times that happened.

This time I used the *ORDER BY* keyword to sort the 2nd column by descending order

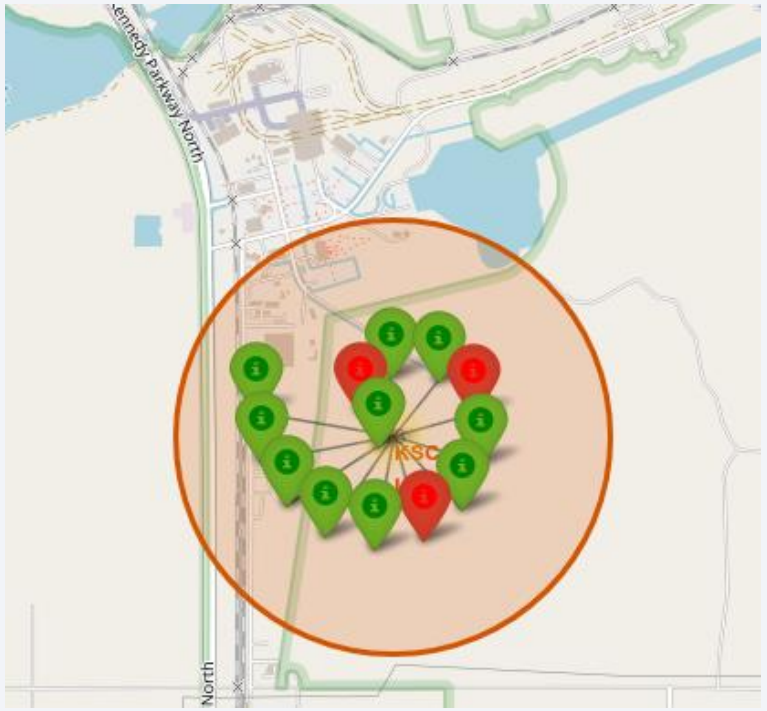
Section 4

Launch Sites Proximities Analysis

Folium Map (Launch Sites)

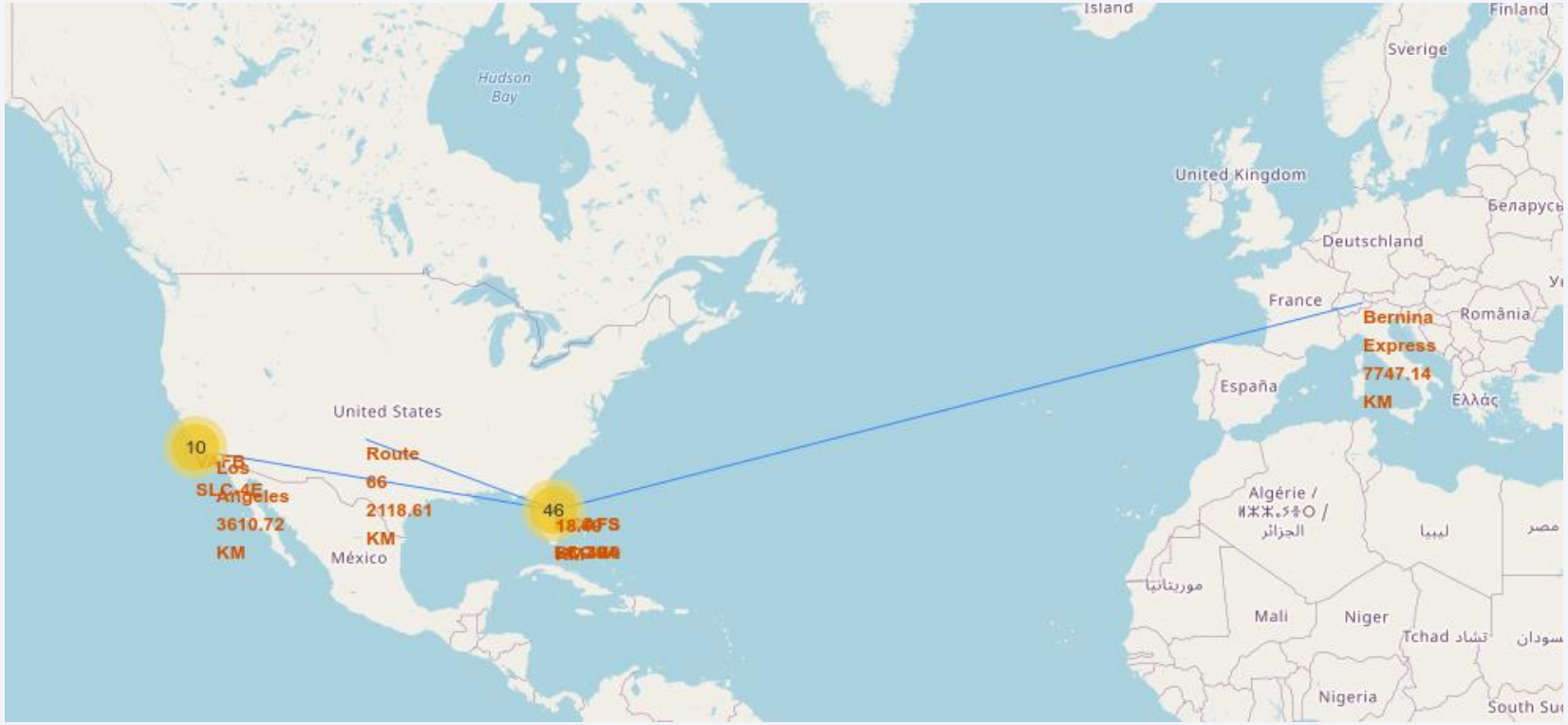
In the map we can see that all the launch sites are very close to the sea/ocean and at opposing coasts.

Folium Map (Launch Outcomes)

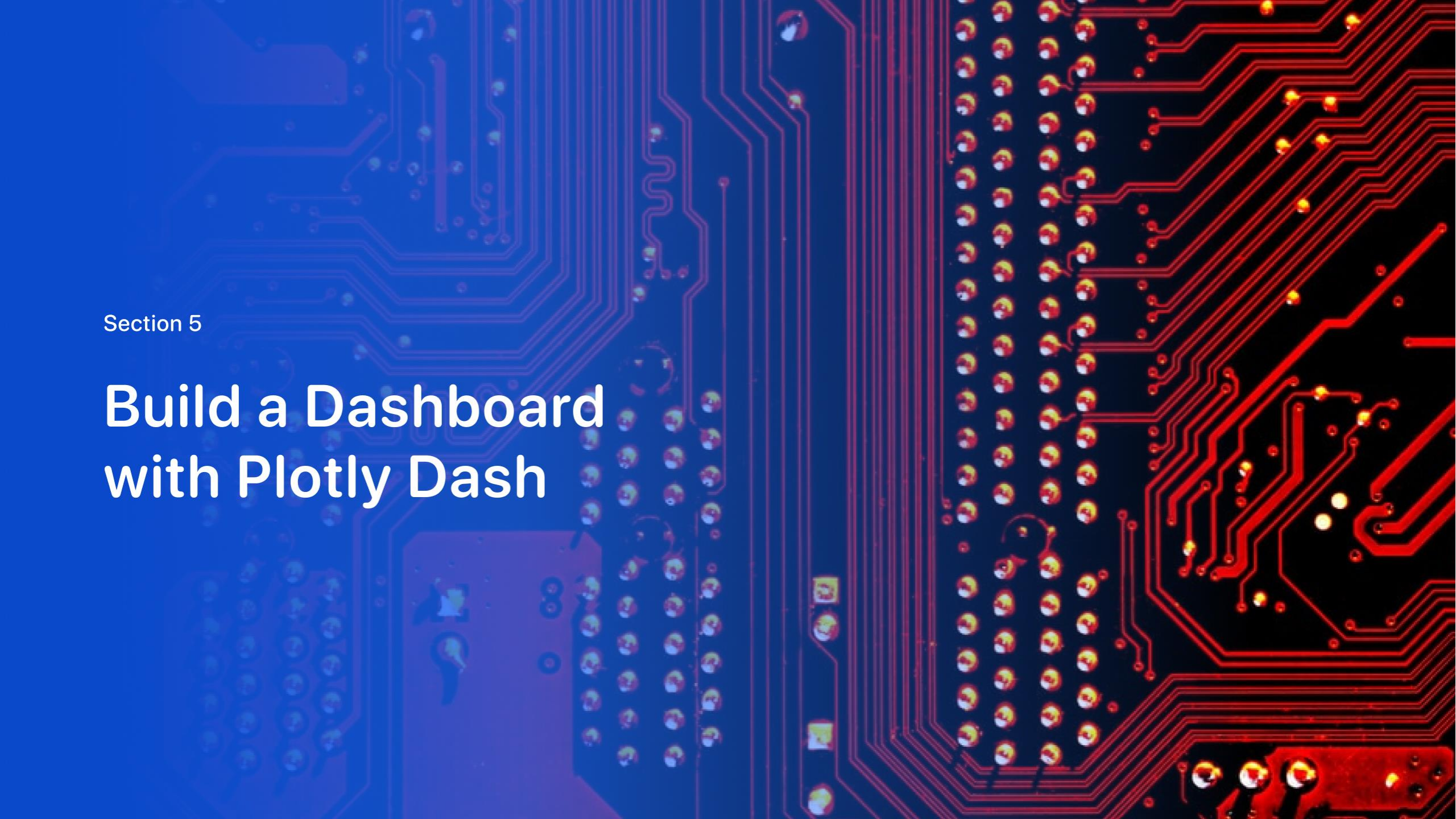


Here we can see the launch outcomes *(successful in green, failed in red)* of the *KSC LC-39A* launch site.

Folium Map (Locations Distance)



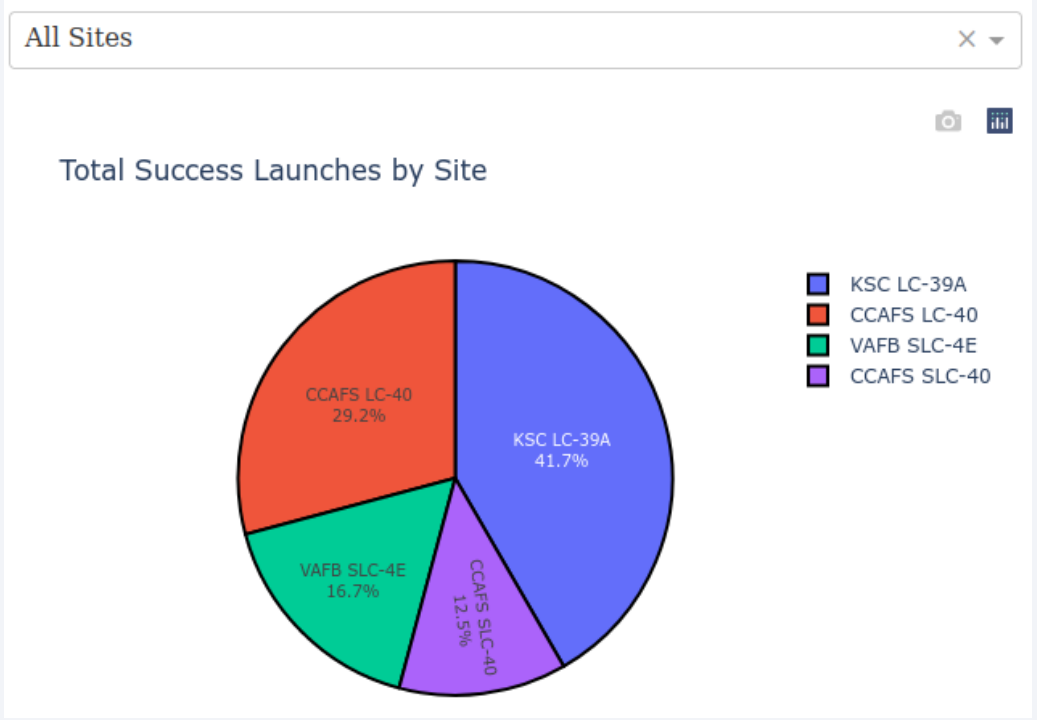
We can see the distance to some locations such as a highway *(Route 66)*, a city *(Los Angeles)* and a railway *(Bernina Express)*.



Section 5

Build a Dashboard with Plotly Dash

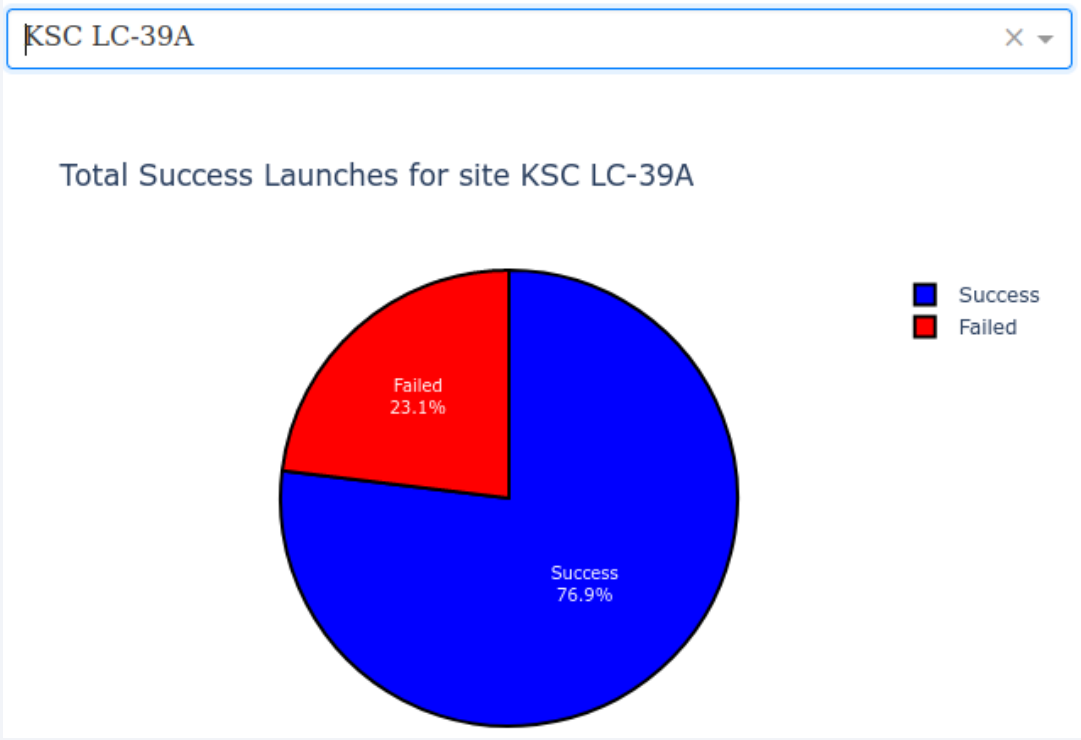
Pie Chart (All Sites)



This pie chart counts all the successful launches and represents them with a percentage from they were launched.

On the right side there is the legend of the pie chart.

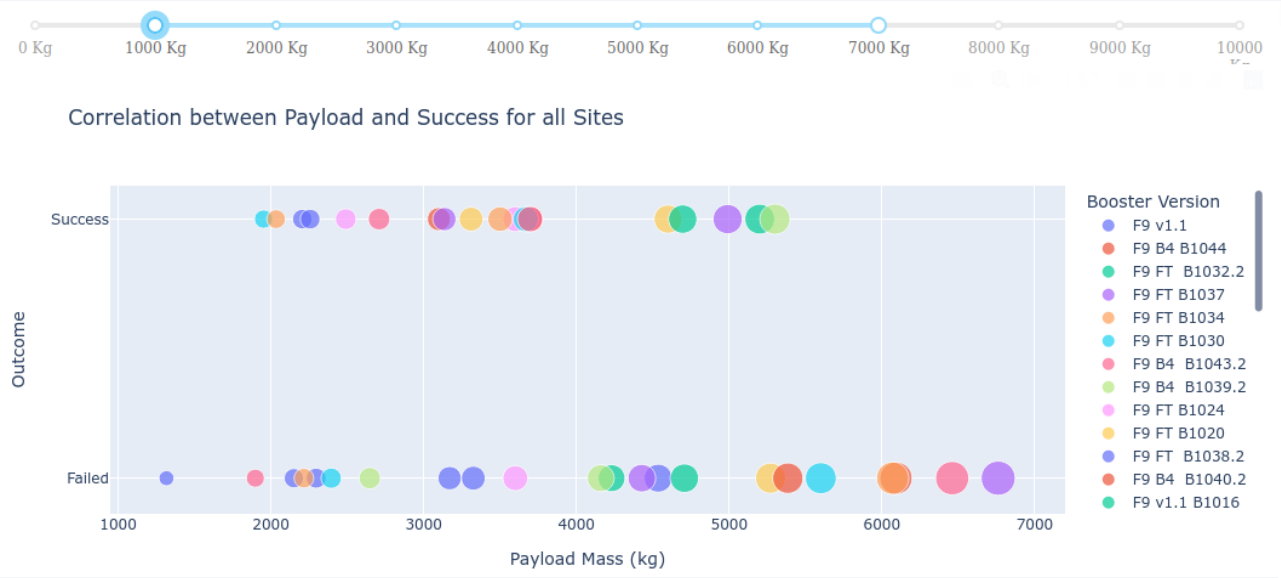
Pie Chart (KSC LC-39A)



This pie chart shows the percentage of successful launches vs failed ones on the *KSC LC-39A* site.

On the right side there is the legend of the pie chart.

Scatter Plot (All Sites)



The scatter plot divides the successful launches (top) and the failed ones (bottom) with its X axis representing the payload mass This pie chart shows the percentage of successful launches vs failed ones on the *KSC LC-39A* site.

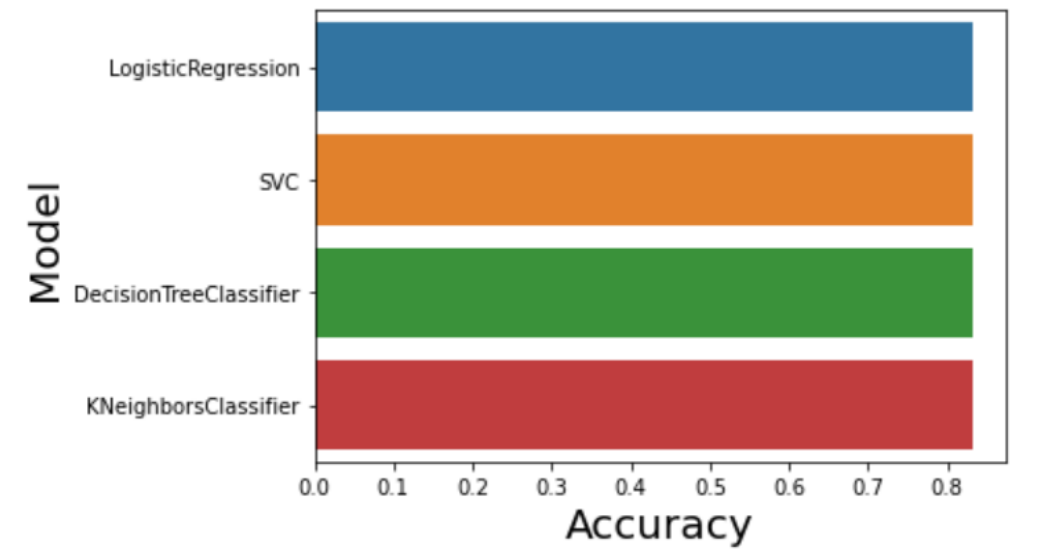
On the right side there is the legend of the pie chart.

We can see that booster carrying lower payloads have higher success rate than heavier ones.

Section 6

Predictive Analysis (Classification)

Classification Accuracy

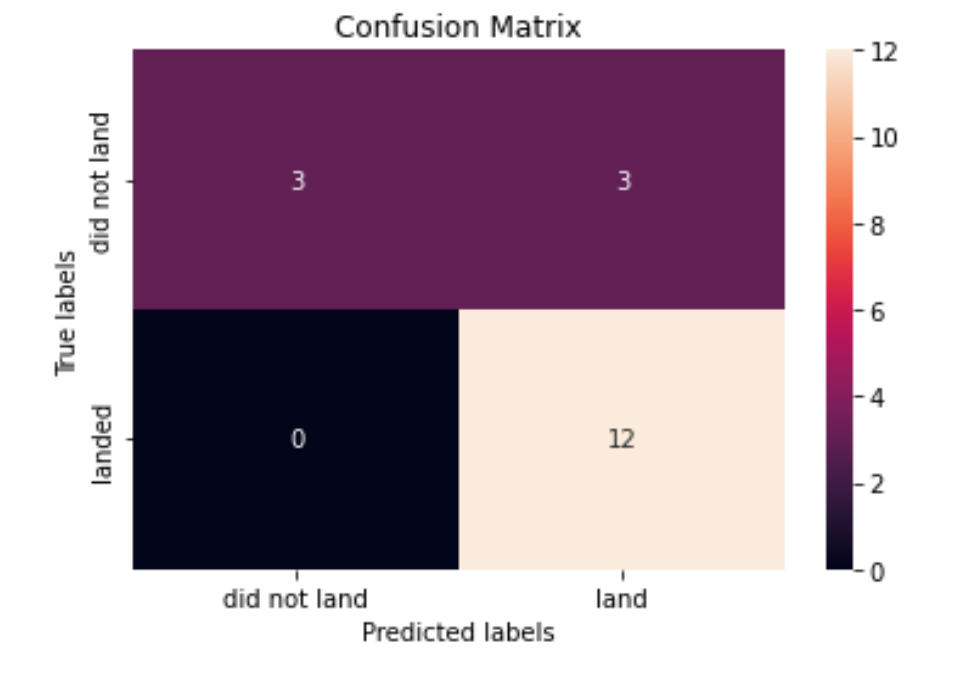


The predictive analysis was made with a test size of the 20% of all data.

Model	Accuracy
LogisticRegression	0.833333
SVC	0.833333
DecisionTreeClassifier	0.833333
KNeighborsClassifier	0.833333

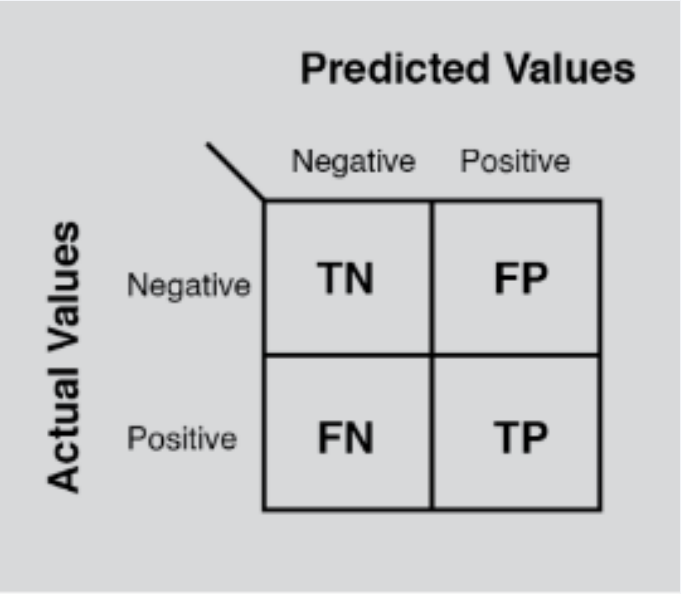
After a couple of tries we can see that all the tests have the same accuracy score, except the *TreeClassifier* which in some cases had outline values such as 0.71 or 0.94.

Confusion Matrix



Due to all the accuracy values of the models being the exact same, I decided to choose the confusion matrix of the tree classifier.

We can see that most of the values are accurate either in True Positive or True Negative but we can see that there are 3 values that are false positives.



Conclusions

- **Launch sites are located close to a sea or ocean and its far away from big cities** so in case the launch goes wrong the objects that fall don't harm anyone.
- Meanwhile years pass SpaceX learns from its mistakes and tries to correct them for the next launch, leading to an **increase in its success rate over the years**.
- **Most of the successful launches** are in **close range orbits** and with **light payloads**, meaning they're **not fully ready to launch a heavy payload** booster to the **GEO** orbit.
- The launch site with the highest success rate by far is **KSC LC-39A**. We can extract the conclusion that the first launch on this site is much later than the other launch sites (so it already has the experience) and it launches boosters with a light payload.
- Three out of the four models ***(KNN, Logistic regression, SVC)*** are **very 'stable'** in terms of predicting the values but on the other hand the **DecisionTree's accuracy varies too much**.

Appendix

- The database that its used is IBM db2 cloud and to make the connection I created a .env file where you can put your credentials without showing them on the notebook.
- Created a copy of all the data used that self-updates when calling the SpaceX API, this allows you use the notebooks without an online connection or if the API is no longer available.

Thank you!