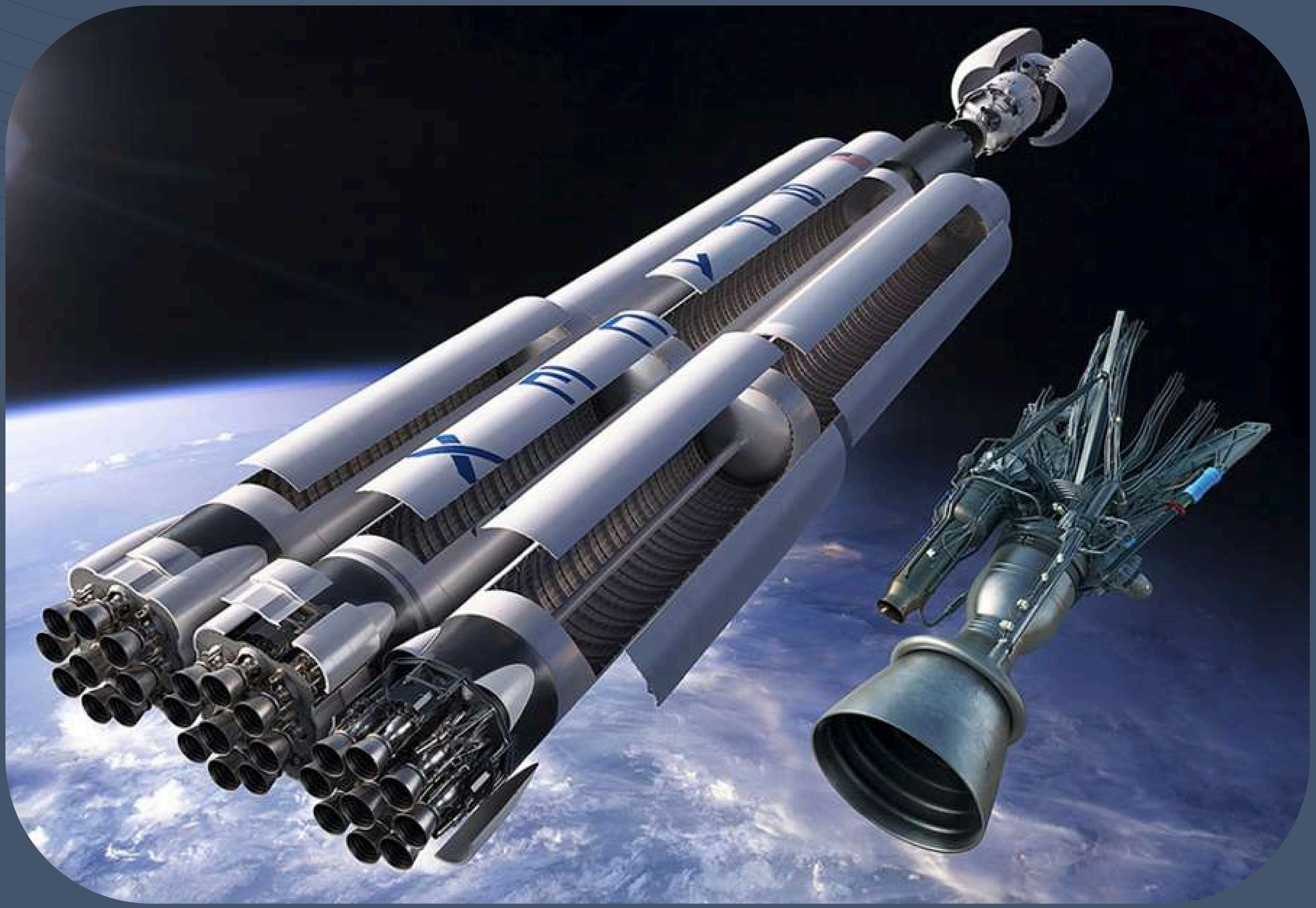
IBM DATA SCIENCE CAPSTONE PROJECT - SPACE X

Rodrigo Chemite Barbosa



ABOUT THE CAPSTONE

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

EXECUTIVE SUMMARY

• Summary of methodologies

- Data collection
- Data wrangling
- EDA with data visualization
- EDA with SQL
- Building an interactive map with Folium
- Building a Dashboard with Plotly Dash
- Predictive analysis (Classification)

• Summary of all results

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

INTRODUCTION

• Project background

The main purpose of this project is predicting if the Falcon 9 first stage will land successfully.
SpaceX advertises Falcon 9 rocket launches on its website, with a cost of 62 million dollars, in which much of the savings is due to SpaceX capability of recovering the basis of the rocket, also know as the first stage, throught an integrated system with GIS locations.
So, determining where the first stage will land is essencial for the launching costs.

• Common problems that needed solving.

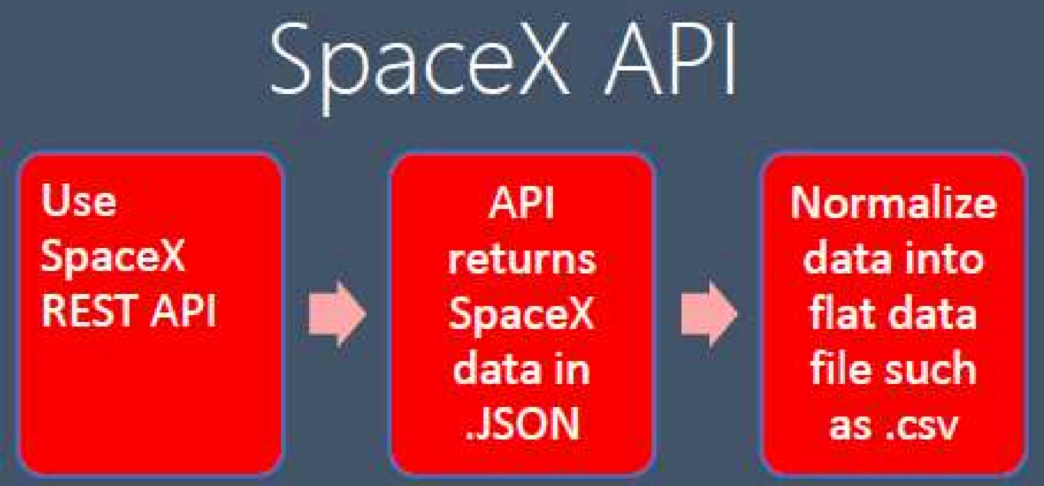
- What are the variables that makes the rocket landing successful?
- How those variables communicate between itselves and interfere on the success rate of the lauch?
- What conditions does SpaceX have to achieve to get the best results and ensure the landing will be sucessful?.

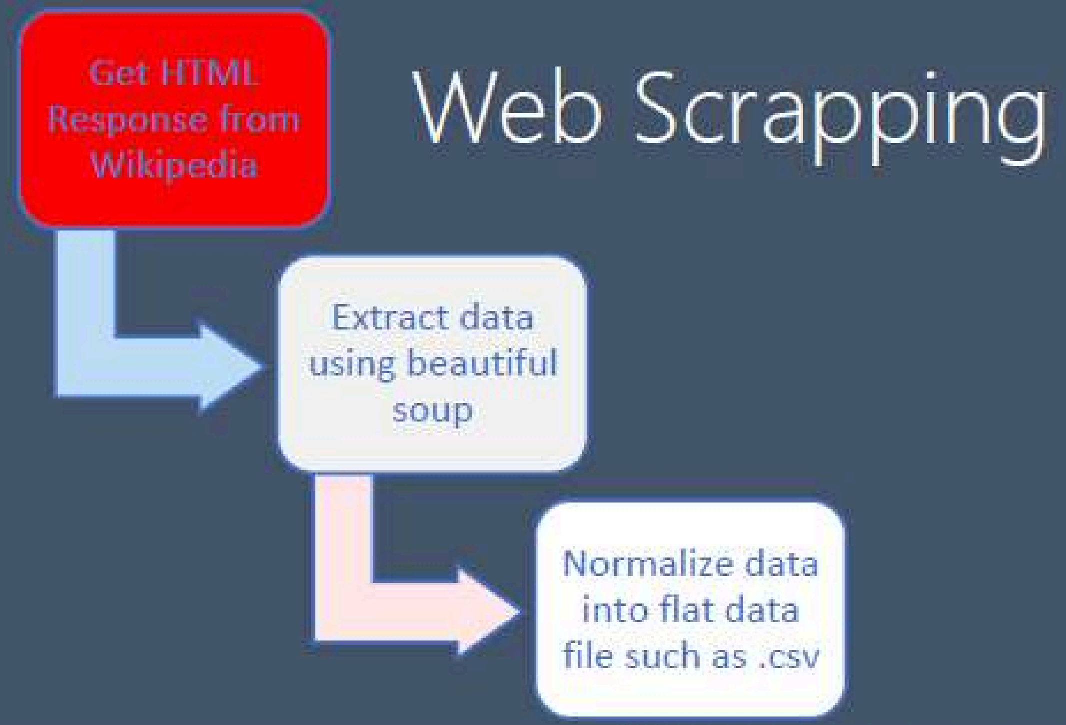
METHODOLOGY

- Data collection methodology; using SpaceX Rest API and Web Scraping (Wikipedia as a source)
- Performed data wrangling (Transforming data for Machine Learning)
- Performed exploratory data analysis (EDA) using visualization and SQL
- Performed interactive visual analytics using Folium and Plotly Dash
- Performed predictive analysis using classification models

METHODOLOGY

The datasets of the Falcon 9 were collected throught the SpaceX REST API, using it to retrieve data about the rocket launchings. Our goal is to use this data to predict either a launching will not or will be sucessful. The date was obtained using BeautifulSoup command in python language, throught the API, web scripting it using Wikipedia as a source.





DATA COLLECTING

1 .Getting Response from API

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

2. Converting Response to a .json file

```
response = requests.get(static_json_url).json()
data = pd.json_normalize(response)
```

3. Apply custom functions to clean data

```
getLaunchSite(data)
getPayloadData(data)
getCoreData(data)
```

```
getBoosterVersion(data)
```

4. Assign list to dictionary then 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}
```

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

5. Filter dataframe and export to flat file (.csv)

```
data_falcon9 = df.loc[df['BoosterVersion']!="Falcon 1"]
```

```
data_falcon9.to_csv('dataset_part_1.csv', index=False)
```

WEB SCRAPING

1 .Getting Response from HTML

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

2. Creating BeautifulSoup Object

```
soup = BeautifulSoup(page.text, 'html.parser')
```

3. Finding tables

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

4. Getting column names

```
column_names = []
temp = soup.find_all('th')
for x in range(len(temp)):
    try:
     name = extract_column_from_header(temp[x])
     if (name is not None and len(name) > 0):
        column_names.append(name)
    except:
     pass
```

5. Creation of dictionary

```
launch_dict= dict.fromkeys(column_names)

# Remove an irrelvant column
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']=[]
```

6. Appending data to keys (refer) to notebook block 12

```
In [12]: extracted_row = 0
         #Extract each table
         for table_number,table in enumerate(
            # get table row
             for rows in table.find_all("tr"):
                 #check to see if first table
```

7. Converting dictionary to dataframe

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

8. Dataframe to .CSV

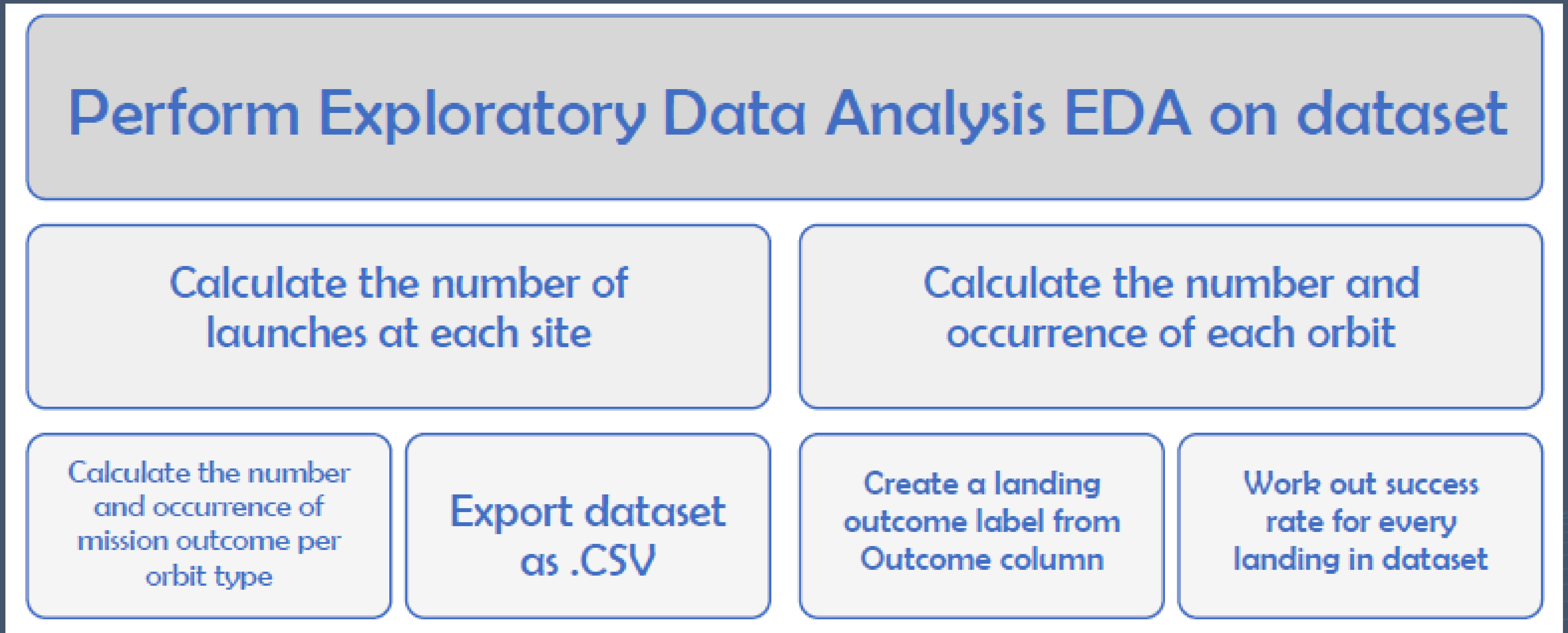
```
df.to_csv('spacex_web_scraped.csv', index=False)
```

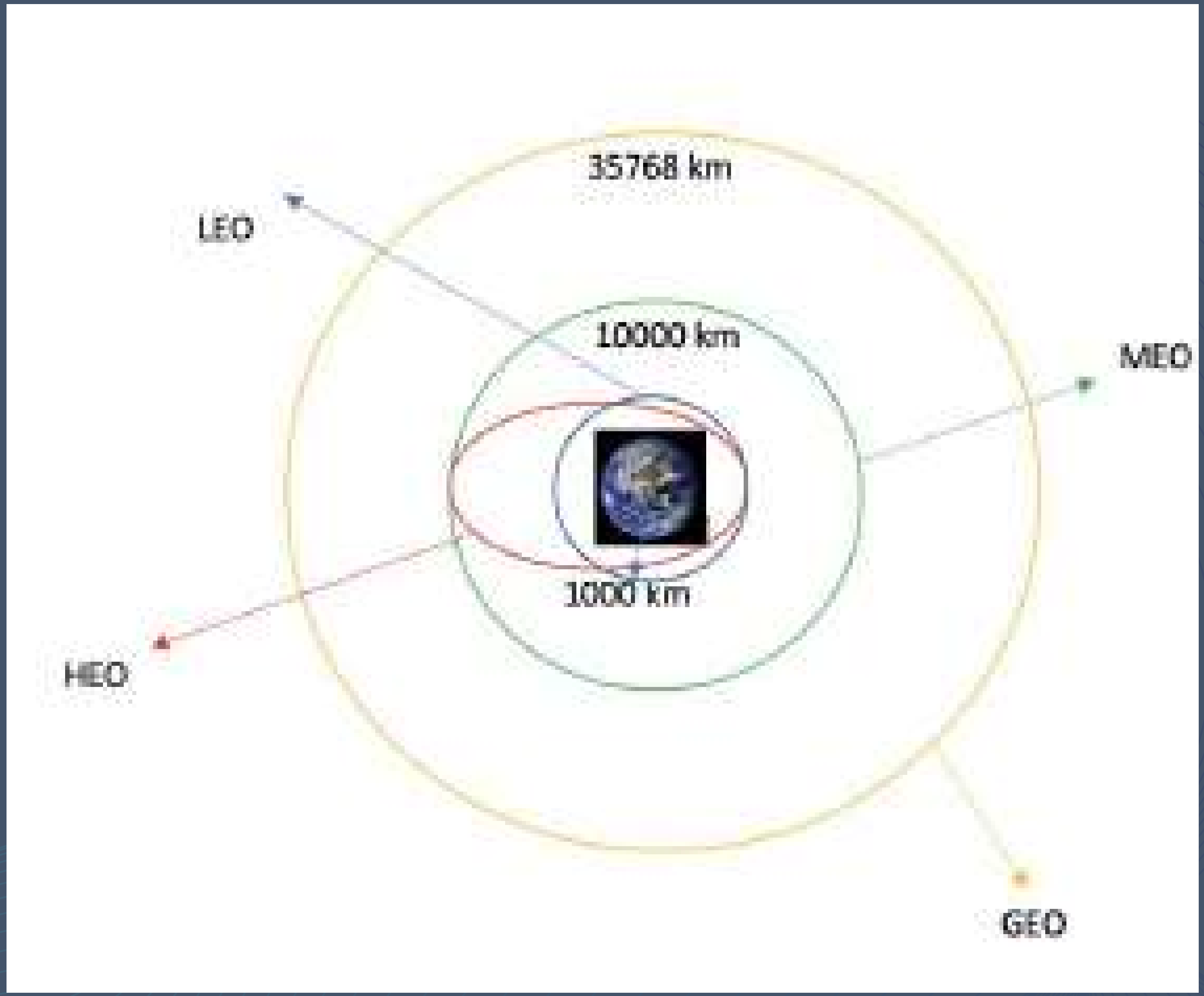
DATA WRANGLING

In the data set, there are several different cases toi determine either a launch was sucessful or not. Each code relates to the situation status of the respective flight.

- True_Ocean means the mission outcome was successfully landed to a specific region of the ocean
- False_Ocean means the mission outcome was unsuccessfully landed to a specific region of the ocean.
- True_RTLS means the mission outcome was successfully landed to a ground pad
- False_RTLS means the mission outcome was unsuccessfully landed to a ground pad.
- True_ASDS means the mission outcome was successfully landed on a drone ship
- False_ASDS means the mission outcome was unsuccessfully landed on a drone ship.

We use Training llabels to describe the situation status of each lauch, to determine if it is sucessful.





EDA WITH DATA VISUALIZATION

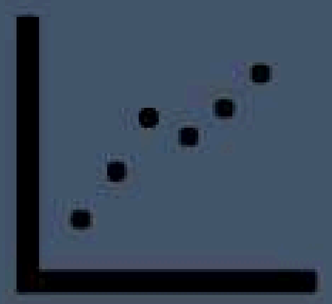
Scatter Graphs being drawn:

Flight Number VS. Payload Mass

Flight Number VS. Launch Site

Payload VS. Launch Site

Orbit VS. Flight Number



Payload VS. Orbit Type

Orbit VS. Payload Mass

Scatter plots show how much one variable is affected by another. The relationship between two variables is called their correlation . Scatter plots usually consist of a large body of data.

Bar Graph being drawn:

Mean VS. Orbit

A bar diagram makes it easy to compare sets of data between different groups at a glance. The graph represents categories on one axis and a discrete value in the other. The goal is to show the relationship between the two axes. Bar charts can also show big changes in data over time.

Line Graph being drawn:

Success Rate VS. Year

Line graphs are useful in that they show data variables and trends very clearly and can help to make predictions about the results of data not yet recorded

EDA WITH SQL

Using SQL queries to gather information about the dataset, for example of some questions we were asked about the data we needed information about. The following SQL queries to get the answers in the dataset are shown below:

- Displaying the names of the unique launch sites in the space mission
- Displaying 5 records where launch sites begin with the string 'KSC'
- Displaying the total payload mass carried by boosters launched by NASA (CRS)
- Displaying average payload mass carried by booster version F9 v1.1
- Listing the date where the successful landing outcome in drone ship was achieved.
- Listing the names of the boosters which have success in ground pad and have payload mass greater than 4000 but less th 6000
- Listing the total number of successful and failure mission outcomes
- Listing the names of the booster_versions which have carried the maximum payload mass.
- Listing the records which will display the month names, successful landing_outcomes in ground pad ,booster versions, launc for the months in year 2017
- Ranking the count of successful landing_outcomes between the date 2010-06-04 and 2017-03-20 in descending order.

BUILDING AN INTERACTIVE MAP USING FOLIUM

We created an interactive map to visualize launch data. Each launch site is marked with a circle, labeled by its name, using its latitude and longitude coordinates. Launch outcomes (failures and successes) are represented by color-coded markers (green for success, red for failure) within a MarkerCluster(). Additionally, we used Haversine's formula to calculate and display distances from launch sites to nearby landmarks, revealing geographical patterns and trends.

To visualize the launch data interactively, we developed a map with the following features:

- Launch Site Markers: Each launch site is represented by a circular marker, precisely positioned using its latitude and longitude, and labeled with its name.

- Outcome Visualization: Launch outcomes (failures and successes, mapped to classes 0 and 1 respectively) are displayed using color-coded markers (green for success, red for failure) within a MarkerCluster().

- Geographical Trend Analysis: Haversine's formula was applied to calculate and draw lines on the map, showing distances from launch sites to various landmarks. This helps in identifying patterns related to the surrounding geography.

BUILDING AN INTERACTIVE DASHBOARD WITH FLASK AND DASH

Used Python Anywhere to host the website so the data can be displayed live. The Dashboard is build with Flask and Dash web framework.

- Pie Chart showing the total launches by a certain site/allsites.
- Display relative proportions of multiple classes of data.
- Size of the circle can be made proportional to the total.

Also, Scatter Graph shows the relationship with Outcome and Payload Mass (Kg) for the different Booster Versions.

- It shows the relationship between two variables.
- It is the best method to show you a non-linear pattern.
- The range of data flow, i.e. maximum and minimum value, can be determined.
- Observation and reading are straightforward.

PREDICTIVE ANALYSIS (CLASSIFICATION)

Building model

- Load our dataset into NumPy and Pandas
- Transform Data
- Split our data into training and test data sets
- Check how many test samples we have
- Decide which type of machine learning algorithms we want to use
- Set our parameters and algorithms to GridSearchCV
- Fit our datasets into the GridSearchCV objects and train our dataset.

Evaluating model

- Check accuracy for each model
- Get tuned hyperparameters for each type of algorithms
- Plot Confusion Matrix

Improving model

- Feature Engineering
- Algorithm Tuning

Finding the best performing classification model

- The model with the best accuracy score wins the best performing model
- In the notebook there is a dictionary of algorithms with scores at the bottom of the notebook.

PREDICTIVE ANALYSIS (CLASSIFICATION)

Building model

- Load our dataset into NumPy and Pandas
- Transform Data
- Split our data into training and test data sets.
- Check how many test samples we have.
- Decide which type of machine learning algorithms we want to use.
- Set our parameters and algorithms to GridSearchCV
- Fit our datasets into the GridSearchCV objects and train our dataset.

Evaluating model

- Check accuracy for each model.
- Get tuned hyperparameters for each type of algorithms.
- Plot Confusion Matrix.

Improving model

- Feature Engineering.
- Algorithm Tuning.

Finding the best performing classification model

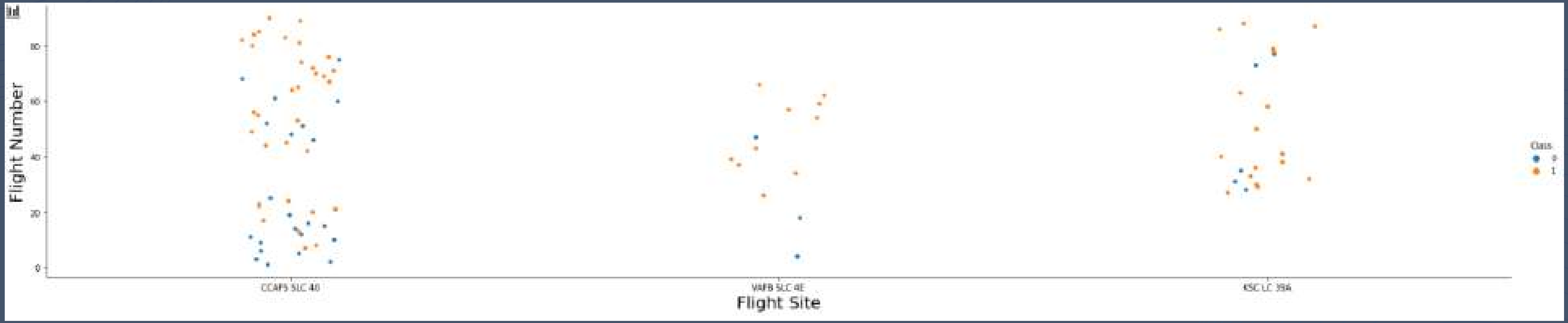
- The model with the best accuracy score wins the best performing model.
- In the notebook there is a dictionary of algorithms with scores at the bottom of the notebook.

RESULTS

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

EDA WITH VISUALIZATION

Flight number vs. Flight Site



Conclusion: The greater the amount of flights at a launch site the greater is the success rate at the site.

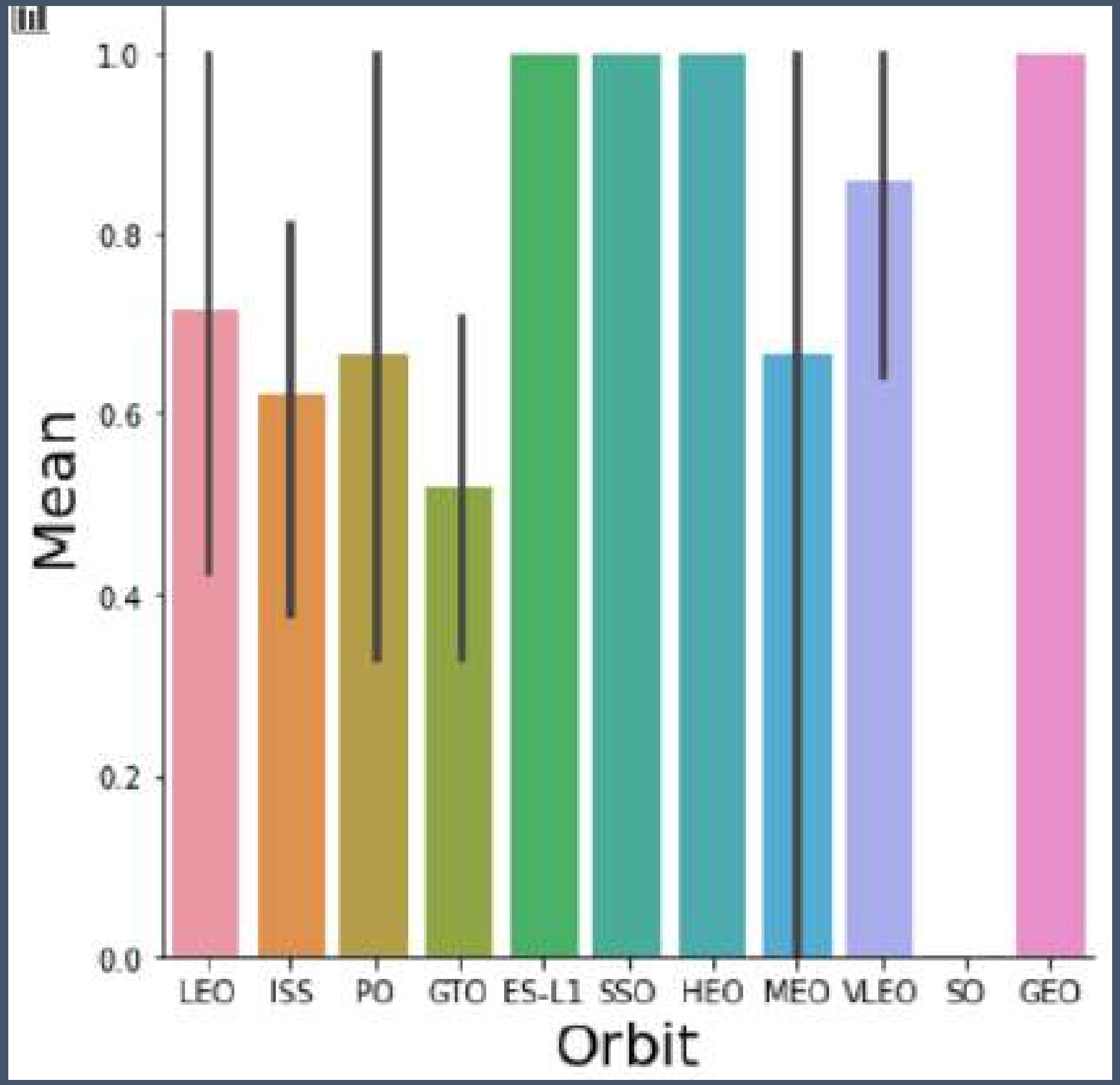
EDA WITH VISUALIZATION

Payload Mass vs. Launch Site

Conclusion: The greater the payload mass for Launch Site CCAFS SLC 40 the higher the success rate for the rocket launch. There is no quite clear pattern to be found using this visualization to make a decision if the Launch Site is dependant on Pay Load Mass for a success launch.

EDA WITH VISUALIZATION

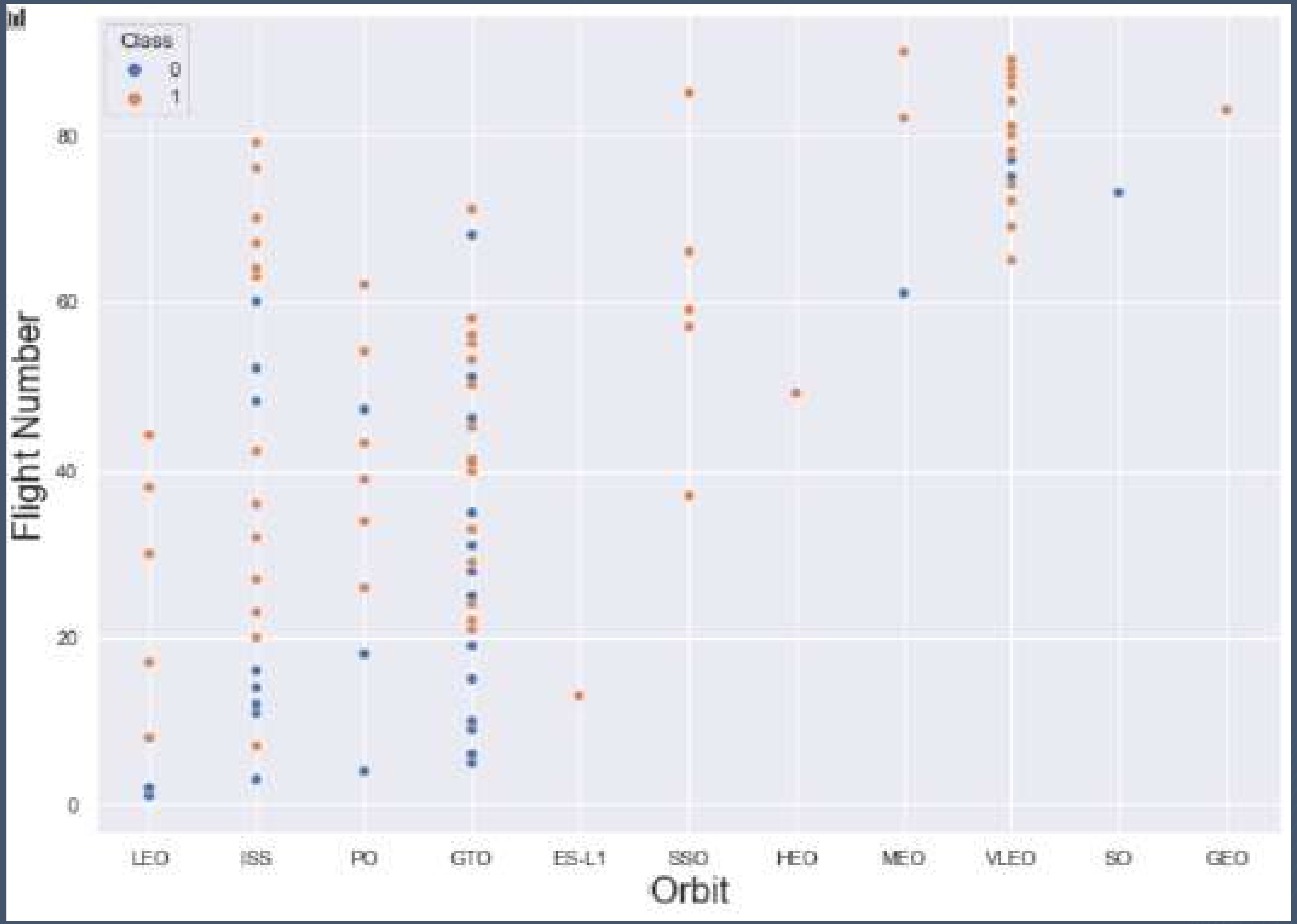
Success rate vs. Orbit type



Conclusion: Orbit GEO,HEO,SSO,ES-L1 have the best Success Rate between all the analyzed Orbit types.

EDA WITH VISUALIZATION

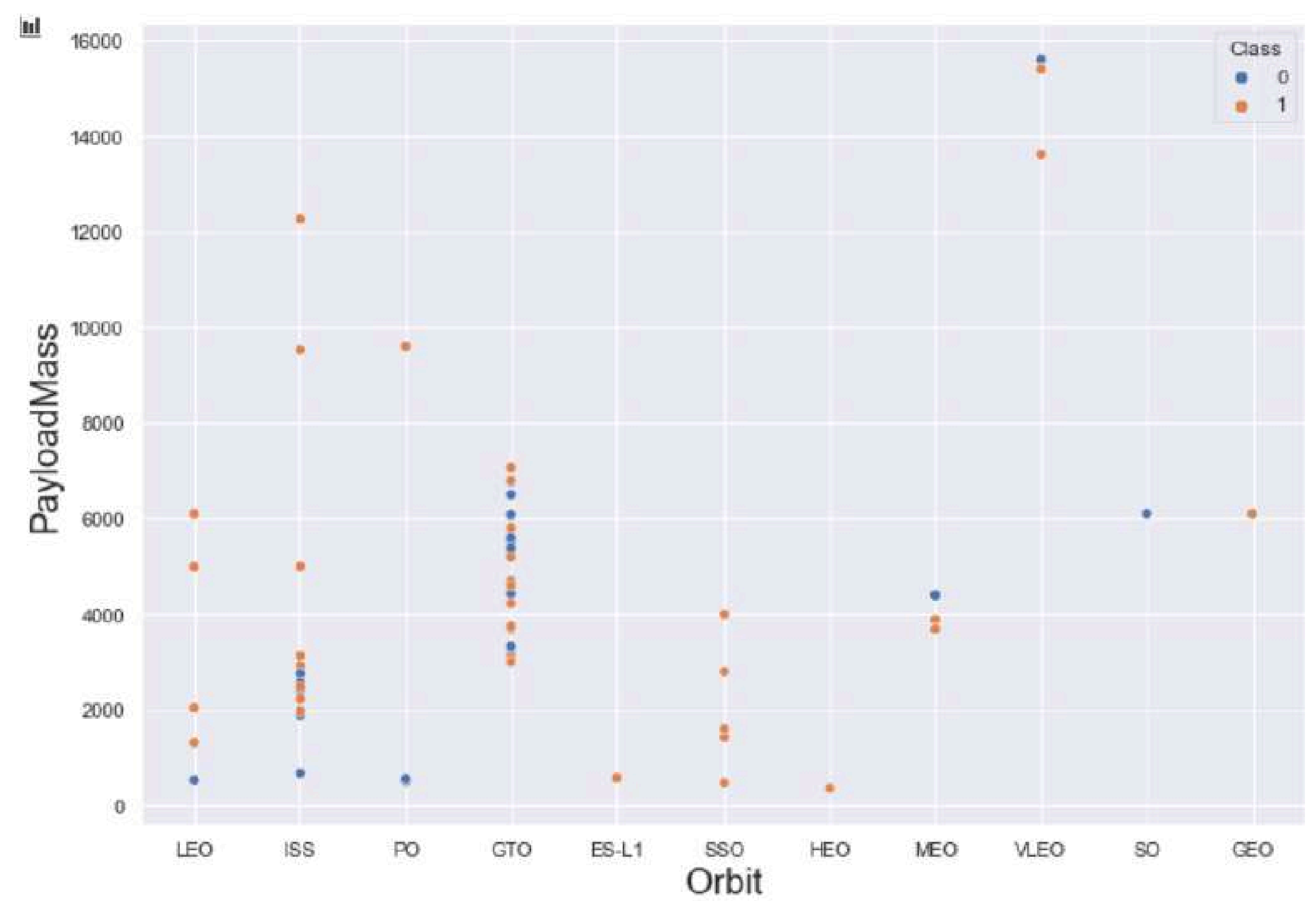
Flight Number vs. Orbit type



Conclusion: There is no relationship between flight number when the orbit is related GTO. The launches at Leo orbit have performed succesfully at major part of its launches.

EDA WITH VISUALIZATION

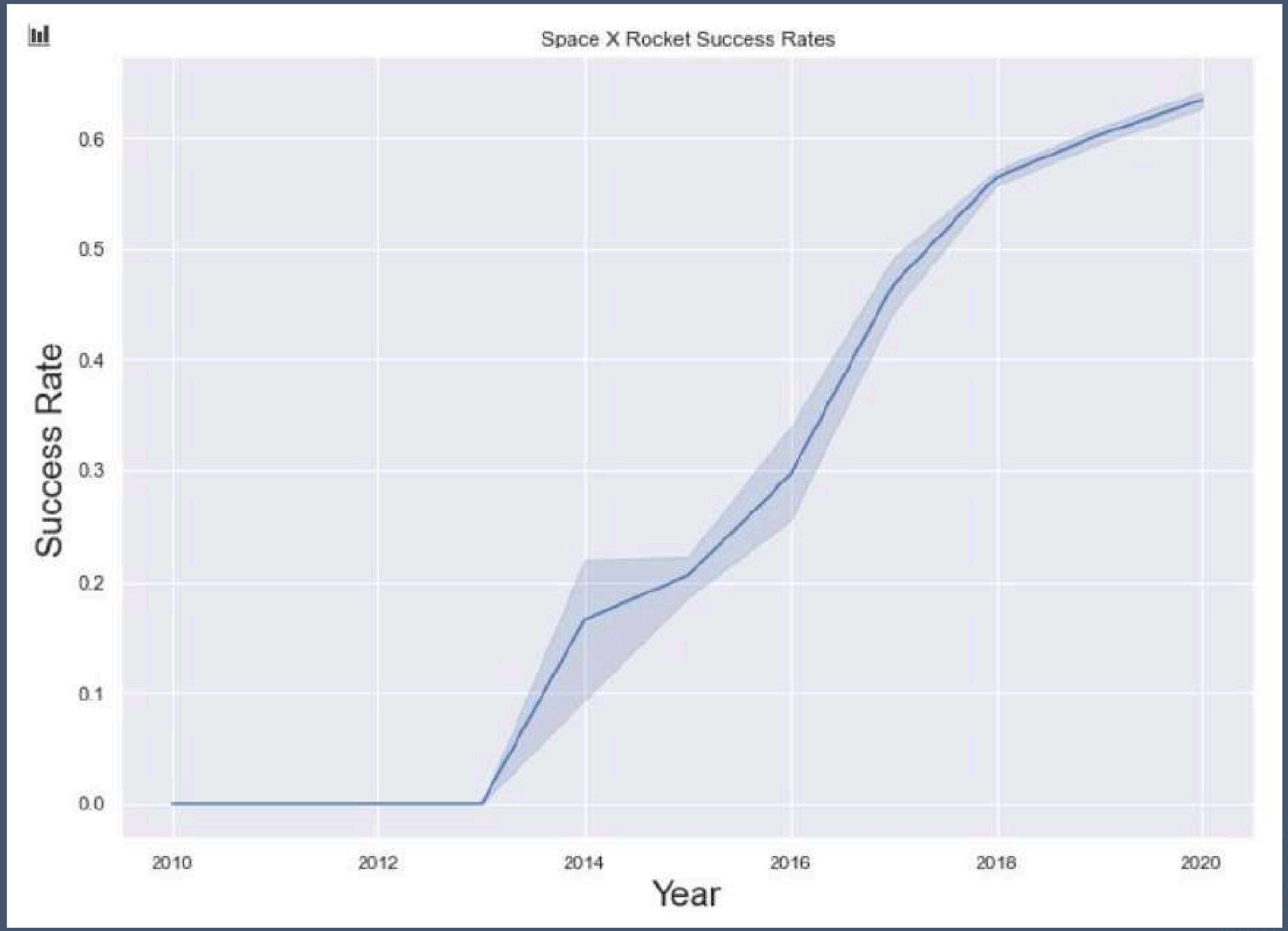
Payload vs. Orbit type



Conclusion: Heavy payloads have a negative influence on GTO orbits and positive on GTO and Polar LEO (ISS) orbits.

EDA WITH VISUALIZATION

Launch success yearly trend



Conclusion: There is an increase of the success rate since 2013 up to 2020

EDA WITH SQL

Using the word DISTINCT in the query means that it will only show Unique values in the Launch_Site column from tblSpaceX

select DISTINCT Launch_Site from tblSpaceX

Unique Launch Sites
CCAFS LC-40
CCAFS SLC-40
CCAFS SLC-40
KSC LC-39A
VAFB SLC-4E

EDA WITH SQL

Using the word ***TOP 5*** in the query means that it will only show 5 records from ***tblSpaceX*** and ***LIKE*** keyword has a wild card with the words ***KSC%*** the percentage in the end suggests that the Launch_Site name must start with KSC.

select TOP 5 * from tblSpaceX WHERE Launch_Site LIKE '

	Date	Time_UTC	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS_KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
0	19-02-2017	2021-07-02 14:39:00.0000000	F9 FT B1031.1	KSC LC-39A	SpaceX CRS-10	2490	LEO (ISS)	NASA (CRS)	Success	Success (ground pad)
1	16-03-2017	2021-07-02 06:00:00.0000000	F9 FT B1030	KSC LC-39A	EchoStar 23	5600	GTO	EchoStar	Success	No attempt
2	30-03-2017	2021-07-02 22:27:00.0000000	F9 FT B1021.2	KSC LC-39A	SES-10	5300	GTO	SES	Success	Success (drone ship)
3	01-05-2017	2021-07-02 11:15:00.0000000	F9 FT B1032.1	KSC LC-39A	NROL-76	5300	LEO	NRO	Success	Success (ground pad)
4	15-05-2017	2021-07-02 23:21:00.0000000	F9 FT B1034	KSC LC-39A	Inmarsat-5 F4	6070	GTO	Inmarsat	Success	No attempt

EDA WITH SQL

Using the function ***SUM*** summates the total in the column ***PAYLOAD_MASS_KG_***.
The ***WHERE*** clause filters the dataset to only perform calculations on ***Customer NASA (CRS).***

select SUM(PAYLOAD_MASS_KG_) TotalPayloadMass from tblSpaceX
where Customer =='NASA (CRS)''','TotalPayloadMass'

	Total Payload Mass
0	45596

Using the function ***AVG*** works out the average in the column ***PAYLOAD_MASS_KG_.***
The ***WHERE*** clause filters the dataset to only perform calculations on ***Booster_version F9 v1.1***

select AVG(PAYLOAD_MASS_KG_) AveragePayloadMass from tblSpaceX
where Booster_Version =='F9 v1.1'

	Average Payload Mass
0	2928

EDA WITH SQL

Using the function ***MIN*** works out the minimum date in the column ***Date*** .
The ***WHERE*** clause filters the dataset to only perform calculations on ***Landing_Outcome Success (drone ship)***.

select MIN(Date) SLO from tblSpaceX where Landing_Outcome ==”Success (drone ship)”

	Date which first Successful landing outcome in drone ship was acheived.
0	06-05-2016

Selecting only ***Booster_Version***
The ***WHERE*** clause filters the dataset to ***Landing_Outcome =Success (drone ship)***
The ***AND*** clause specifies additional filter conditions ***Payload_MASS_KG 4000*** AND ***Payload_MASS_KG 6000***.

select Booster_Version from tblSpaceX where Landing_Outcome = 'Success (ground pad)’ AND Payload_MASS_KG_ > 4000 AND Payload_MASS_KG_ < 6000

	Date which first Successful landing outcome in drone ship was acheived.
0	F9 FT B1032.1
1	F9 B4 B1040.1
2	F9 B4 B1043.1

EDA WITH SQL

The LIKE '%foo%' wildcard shows that in the record the foo phrase is in any part of the string in the records for example.

SELECT(SELECT Count(Mission_Outcome from tblSpaceX where Mission_Outcome
LIKE '%Success%') as Successful_Mission_Outcomes
(SELECTCount(Mission_Outcome from tblSpaceX where Mission_Outcome
LIKE"%F ailure%') as Failure_Mission _Coutcomes

	Successful_Mission_Outcomes	Failure_Mission_Outcomes
0	100	1

EDA WITH SQL

Using the word ***DISTINCT*** in the query means that it will only show Unique values in the ***Booster_Version*** column from tblSpaceX
GROUP BY puts the list in order set to a certain condition.
DESC means its arranging the dataset into descending order

```
SELECT DISTINCT Booster_Version, MAX(PAYLOAD_MASS
_KG_) AS [Maximum Payload Mass]
FROM tblSpaceX GROUP BY Booster_Version
ORDER BY [Maximum Payload Mass] DESC
```

	Booster_Version	Maximum Payload Mass
0	F9 B5 B1048.4	15600
1	F9 B5 B1048.5	15600
2	F9 B5 B1049.4	15600
3	F9 B5 B1049.5	15600
4	F9 B5 B1049.7	15600
...	...	...
92	F9 v1.1 B1003	500
93	F9 FT B1038.1	475
94	F9 B4 B1045.1	362
95	F9 v1.0 B0003	0
96	F9 v1.0 B0004	0

97 rows × 2 columns

EDA WITH SQL

Date fields in SQL Server stored as ***NVARCHAR*** the ***MONTH*** function returns name month. The function CONVERT converts ***NVARCHAR*** to ***Date***.

SELECT DATENAME(month, DATEADD(month,MONTH(CONVERT(date, Date, 105)), 0) 1) AS Month, Booster_Version, Launch_Site, Landing_Outcome FROM tblSpaceX WHERE (Landing_Outcome LIKEN'%Success %') AND (YEAR(CONVERT(date, Date, 105)) = '2017'

Month	Booster_Version	Launch_Site	Landing_Outcome
January	F9 FT B1029.1	VAFB SLC-4E	Success (drone ship)
February	F9 FT B1031.1	KSC LC-39A	Success (ground pad)
March	F9 FT B1021.2	KSC LC-39A	Success (drone ship)
May	F9 FT B1032.1	KSC LC-39A	Success (ground pad)
June	F9 FT B1035.1	KSC LC-39A	Success (ground pad)
June	F9 FT B1029.2	KSC LC-39A	Success (drone ship)
June	F9 FT B1036.1	VAFB SLC-4E	Success (drone ship)
August	F9 B4 B1039.1	KSC LC-39A	Success (ground pad)
August	F9 FT B1038.1	VAFB SLC-4E	Success (drone ship)
September	F9 B4 B1040.1	KSC LC-39A	Success (ground pad)
October	F9 B4 B1041.1	VAFB SLC-4E	Success (drone ship)
October	F9 FT B1031.2	KSC LC-39A	Success (drone ship)
October	F9 B4 B1042.1	KSC LC-39A	Success (drone ship)
December	F9 FT B1035.2	CCAFS SLC-40	Success (ground pad)

EDA WITH SQL

Function ***COUNT*** counts records in column ***WHERE*** filters data.

```
SELECT COUNT(Landing_Outcome) FROM tblSpaceX WHERE (Landing_Outcome LIKE '%Success%') AND (Date > '04 06 2010')AND(Date < '20 03 2017')
```

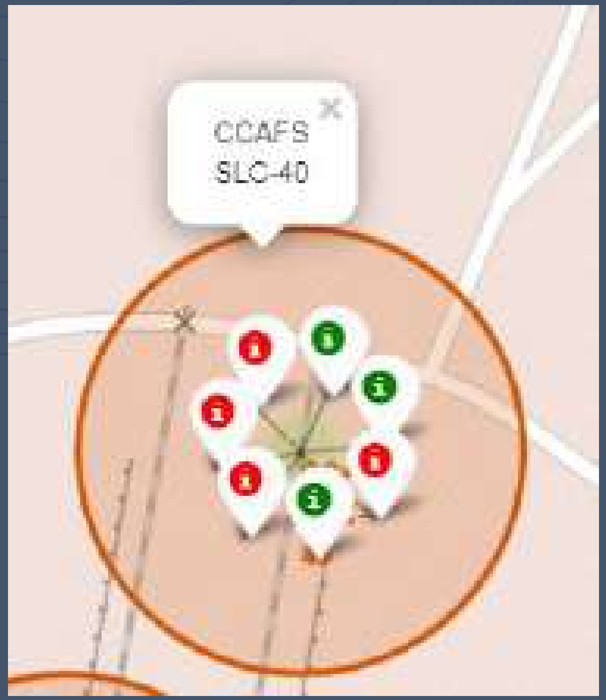
	Successful Landing Outcomes Between 2010-06-04 and 2017-03-20
0	34

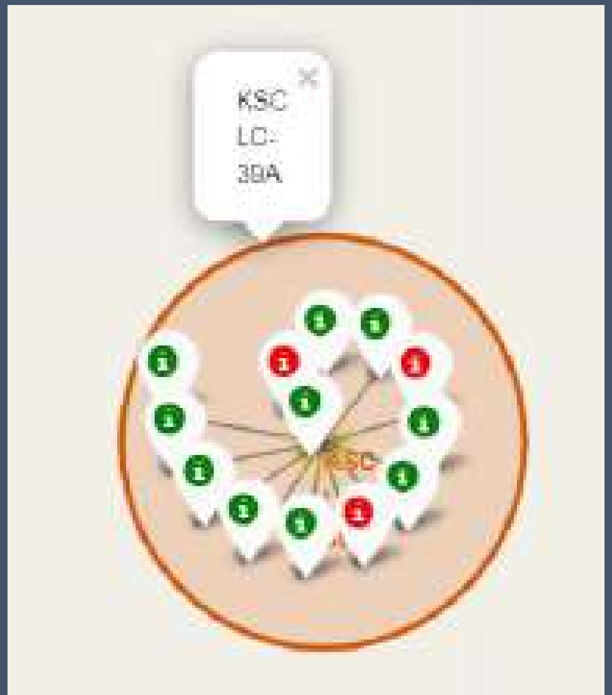
INTERACTIVE MAP WITH FOLIUM

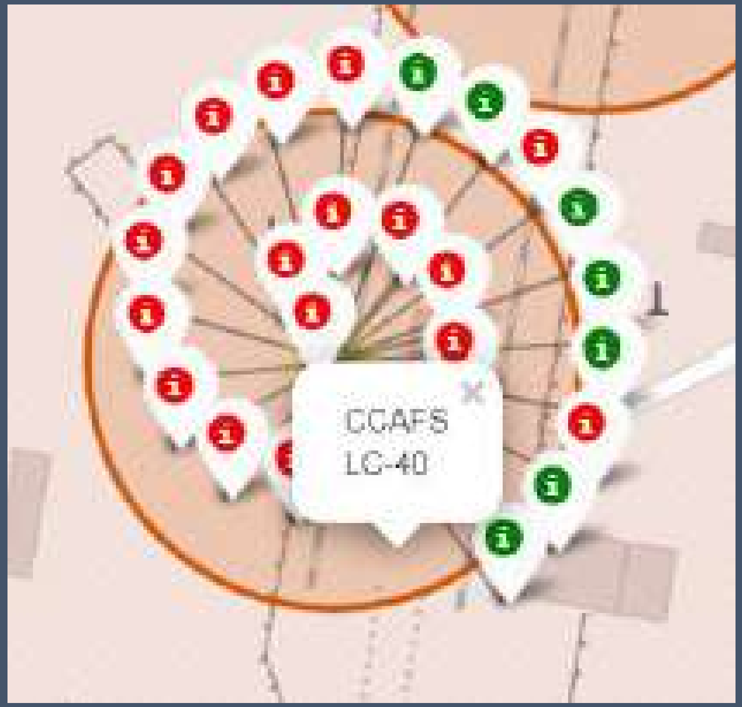
Every launch sites in the globe as map marker clusters.

INTERACTIVE MAP WITH FOLIUM

Green markers shows succesful launches.
Red markers shows unsucessful launches.





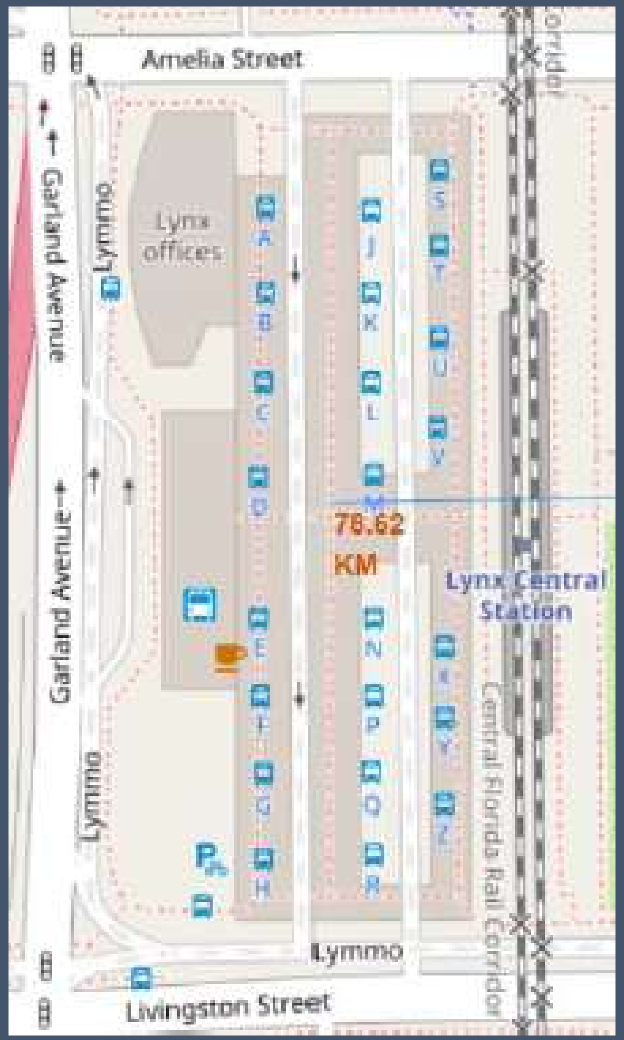


INTERACTIVE MAP WITH FOLIUM

Launch sites distance to landmarkers to find trends with Haversine formula using CCAFS-SLC-40 as a reference.

- Are launch sites in close proximity to railways? No
- Are launch sites in close proximity to highways? No
- Are launch sites in close proximity to coastline? Yes
- Do launch sites keep certain distance away from cities? Yes

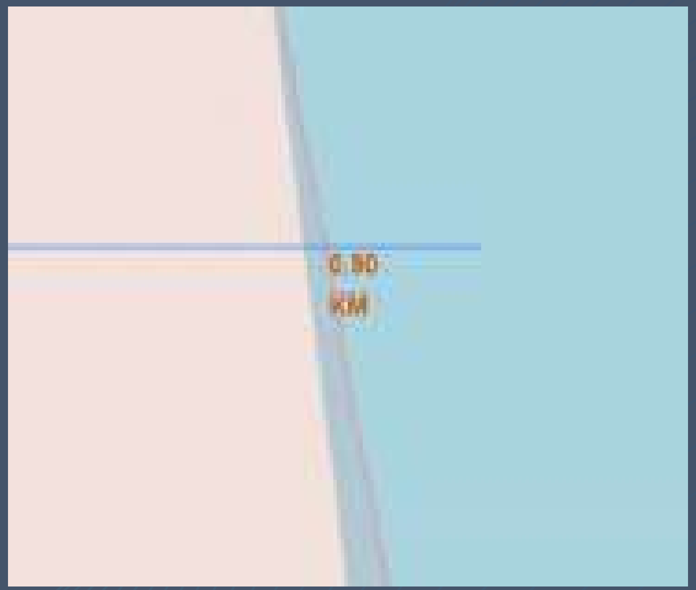
Distance to Railway Station



Distance to closest Highway

Distance to closest coast

Distance to coastlinet

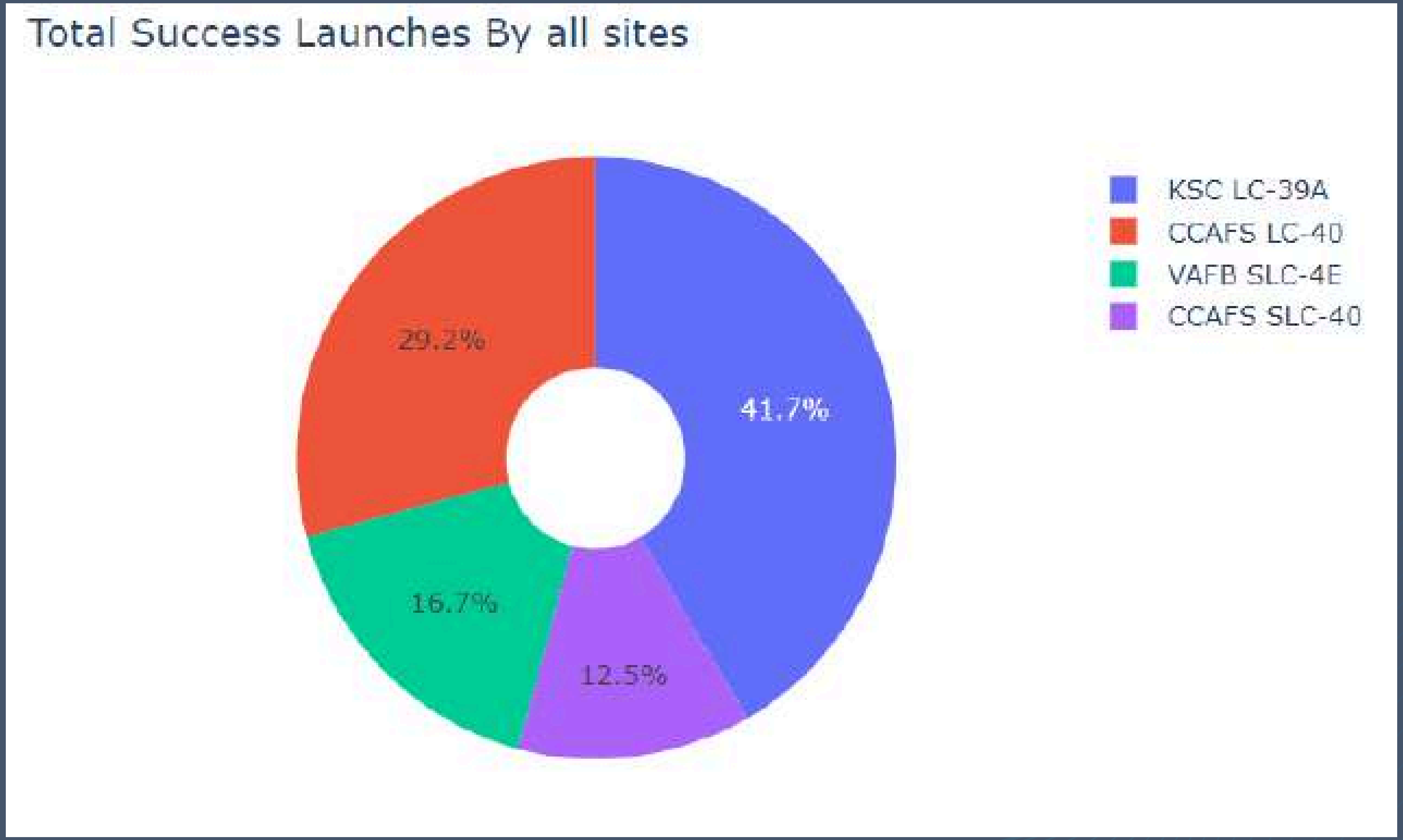


Distance to city

DASHBOARD WITH PLOTLY DASH

DASHBOARD – Pie chart showing the success percentage achieved by each launch site.

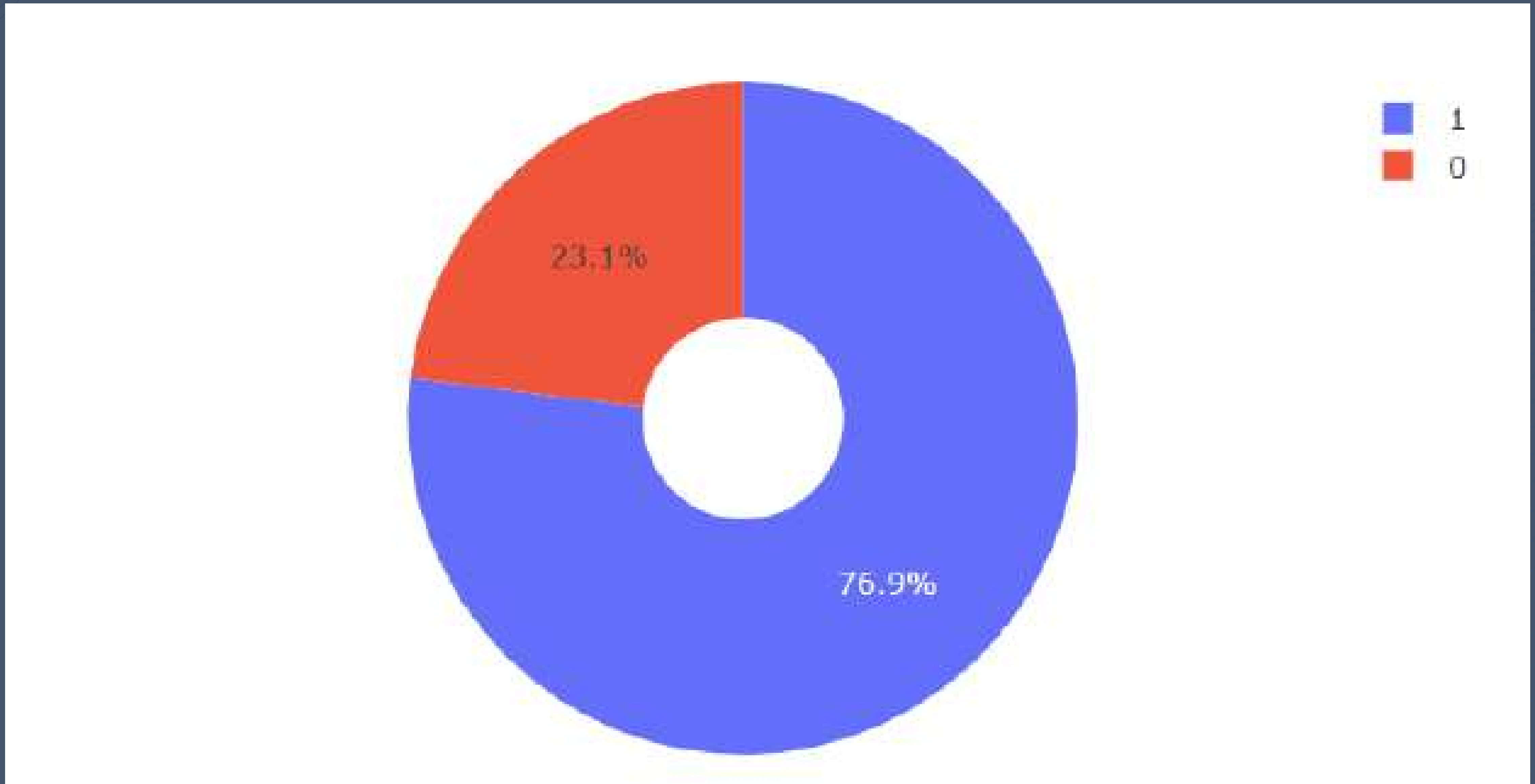
It is possible to check that KSC-LC39-A had the most succesful launches from all the sites.



DASHBOARD WITH PLOTLY DASH

DASHBOARD – Pie chart showing the lauch site with the highest launch success ratio.

KSC LC-39A achived a 76,9% success rate while getting a 23.1% failure rate.

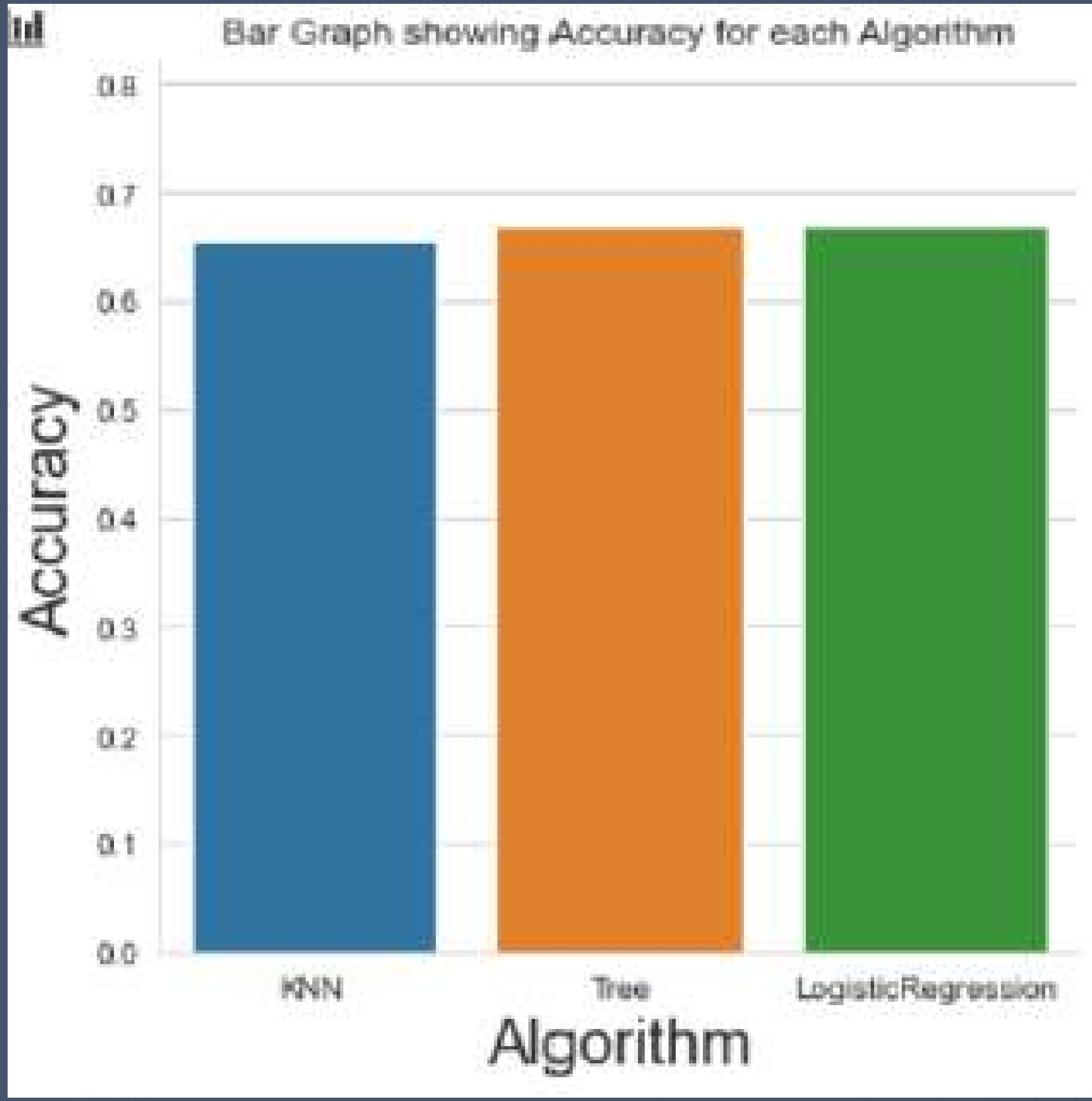


PREDICTIVE ANALYSIS (CLASSIFICATION)

Selecting the best hyperparameters for the decision tree classifier using the validation data, the obtained accuracy is 83,34%

```
bestalgorithm = max(algorithms, key=algorithms.get)
```

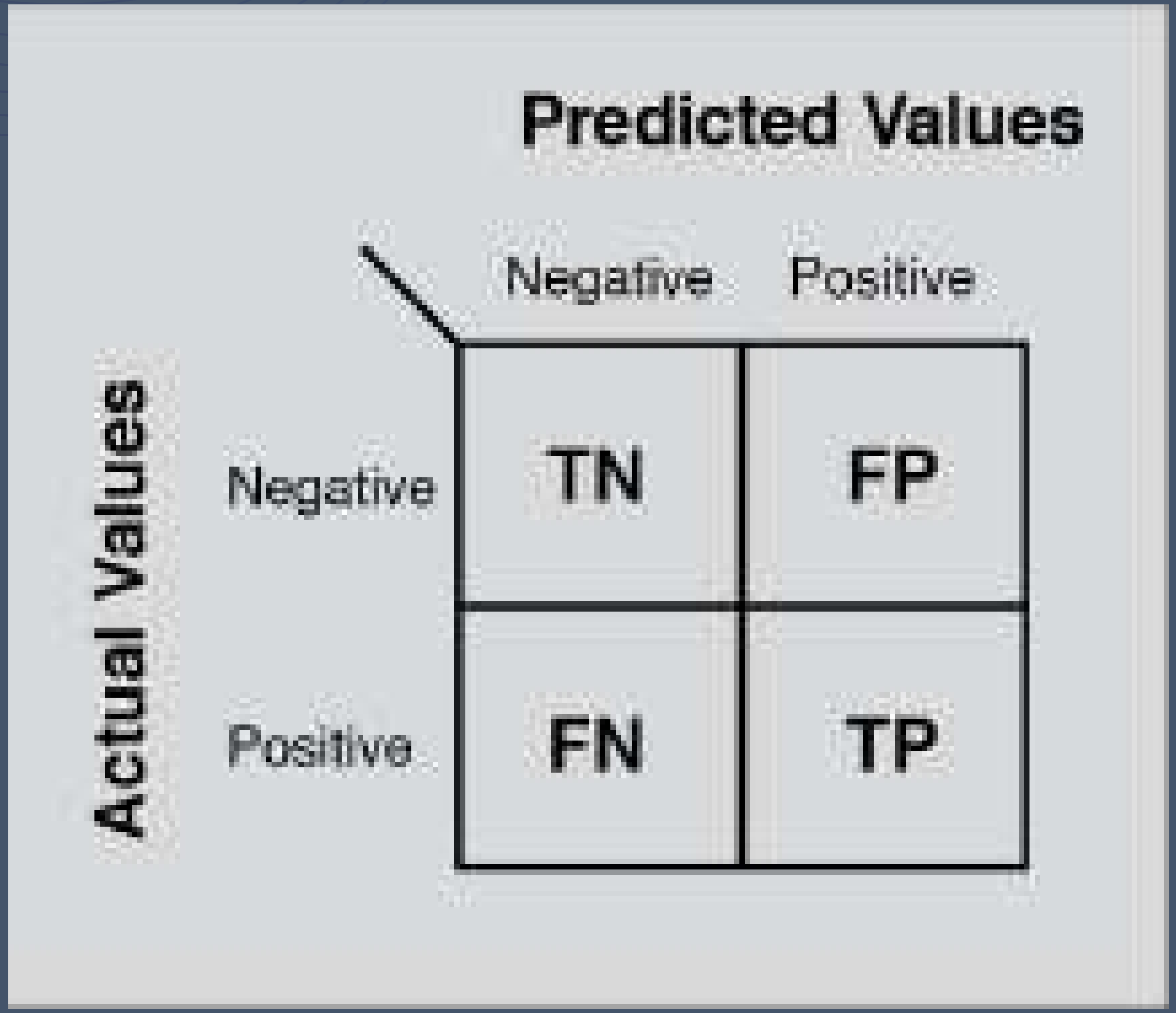
	Accuracy	Algorithm
0	0.653571	KNN
1	0.667857	Tree
2	0.667857	LogisticRegression

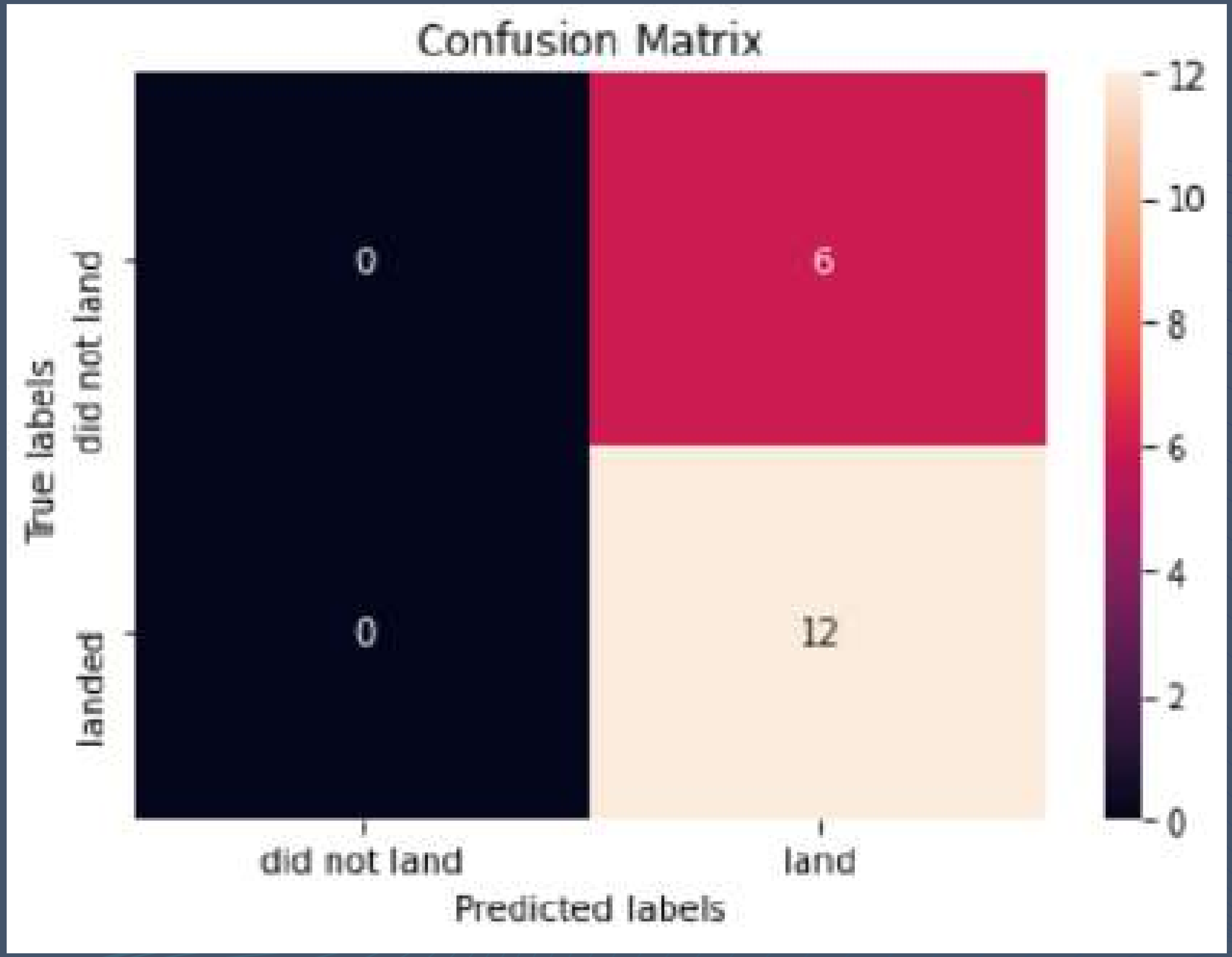



```
Best Algorithm is Tree with a score of 0.6678571428571429
Best Params is : {'criterion': 'gini', 'max_depth': 2, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 2, 'splitter': 'best'}
```

PREDICTIVE ANALYSIS (CLASSIFICATION)

While examinating the confusion matrix, it is possible to see that Tree can distinguish between different classes. The problem with most major problem is false positive.





CONCLUSION

- The Tree Classifier Algorithm is the best for ML for this dataset.
- Low weighted payloads perform better than heavier payloads.
- The success rates for SpaceX lauches is directly proportional to the time in years the launches perform perfectly.
- KSC LC-39A had the most succesful lauches from all sites.
- Obits GEO, HEO, SSO, ES-1 has the best succes rate between the launches by Orbit Type

APPENDIX

- Haversine formula
- ADGGoogleMaps Module (not used but created)
- Module SQLSERVER(ADGSQLSERVER)
- PythonAnywhere 24/7 dashboard

APPENDIX

- Haversine formula
- ADGGoogleMaps Module (not used but created)
- Module SQLSERVER(ADGSQLSERVER)
- PythonAnywhere 24/7 dashboard

APPENDIX

- Haversine formula

The haversine formula describes the great-circle distance between two points as an sphere, given their longitudes and latitudes measures. The method was adopted considering to analyze plain distance between given point, even know earth is shaped as a geoid, the approximation method considers earth as an elipsoid, giving its results as a reliable mathmatical approximation.

$$a = \sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos\varphi 1 \cdot \cos\varphi 2 \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right)$$

$$c = 2 \cdot \operatorname{atan2}\left(\sqrt{a}, \sqrt{(1-a)}\right)$$

$$d = R \cdot c$$

```
#Variables
d is the distance between the two points along a great circle of the sphere (see spherical distance),
r is the radius of the sphere.
φ1, φ2 are the latitude of point 1 and latitude of point 2 (in radians),
λ1, λ2 are the longitude of point 1 and longitude of point 2 (in radians).
```

APPENDIX

- ADGGoogleMaps Module (not used but created)

For obtaining coordinates, using Googl API Secret, we signed up for a Google Geocoding API key, heading to library under APIs and Services adding Geocoding API, the following command was used to obtain a list of coordinates:

```
import ADGGoogleMaps
map = ADGGoogleMaps.ADGGoogleMaps("google_api_secret_key","your_address")
cords = map.GetCordsFromAddress()
cords
```

After declaring the mapclass using the code above, we used the code below to obtain a Folium map in the Jupyter Notebook:

```
map.ReturnMap(20)
```

APPENDIX

- Module SQLSERVER(ADGSQLSERVER)

The implementation to pull data from collumns, extract records, run stored procedures, etc. are examplified below:

```
import sqlserver as ss

#(ip,portnumber,databasename,username,password)
db = ss.sqlserver('localhost','1433','CVs','','')

#(query,columnname)
db.GetRecordsOfColumn('select * from tblUsers','personid')
```

APPENDIX

- PythonAnywhere Live Dashboard

The live dashboard “PythonAnywhere” can be executed using the URL “www.pythonanywhere.com” on the dock Linux container.