IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

<PHOO PHOO MON MYAT THU>
<22 Jan 2024>

OUTLINE

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

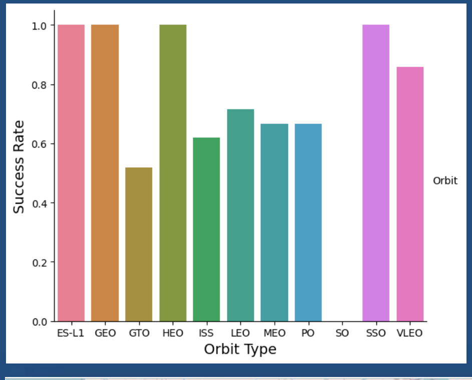
EXECUTIVE SUMMARY

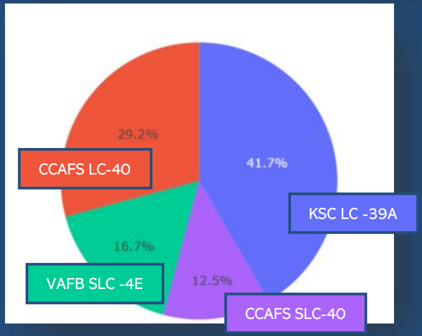
Summary of methodologies

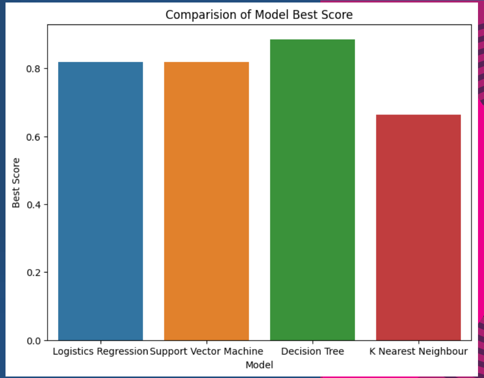
- Data collection
- Data wrangling
- Exploratory data analysis using SQL
- Exploratory data analysis using pandas and matplotlib
- Interactive visual analytics and dashboard
- Predictive analysis(classification)

Summary of All results

- Exploratory Data Analysis (EDA) results
- Geospatial Analysis
- Interactive Dashboard
- Predictive analysis of classification models







PROJECT BACKGROUND AND CONTEXT

SpaceX offers Falcon 9 rocket launches on its website for $62 million, which is significantly less expensive than other providers, costing upwards of $165 million for each launch.

The primary reason for this is that SpaceX reuses the rocket's first stage. The cost of a launch can be determined by whether the first stage will land or not, which can be used by other companies to compete with SpaceX for a rocket launch bid.

This project aims to predict the successful landing of the first stage of the Falcon 9 rocket.

Section 1

Methodology

METHADOLOGY

1

Data collection

- Request to SpaceX API by using the GET request function
- Web scraping Falcon 9 and Falcon Heavy Launches Records from Wikipedia

2

Data wrangling

- Dealing the missing value
- Extract Launch sites, orbits and responsible numbers
- Calculate the landing outcome successful/unsuccessful

3

Exploratory Analysis

- Manipulate and Evaluate dataset by using SQL
- Determine the correlation between variables and plot the pattern

4

Interactive visual analytics

Geospatial analysis using Folium

Creating interactive dashboard with Plotly Dash

5

Data modeling and Evaluation

- Split target variable and feature variable
- Standardized variables by using scikit learn library
- Use prediction model such as Logistics Regression, SVM, Decision Tree and K Nearest Neighbor
- Tune the parameter by using GridSearchCV
- Calculate the Accuracy score and determine which model is the best

DATA COLLECTION - SPACE X API

Retrieving launch data using the SpaceX API, including rocket details, payload, launch specifications, and landing information.

- Create Custom Function to retrieve necessary data (Appendix)

1

- Use GET request to parse the SpaceX launch data
- Decode response content as JSON and convert it to a Pandas data frame using .json_normalize( )

2

- Retrieve required information by using the custom logic

1

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

2

```
data = data[['rocket', 'payloads', 'launchpad',
'cores', 'flight_number', 'date_utc']]

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)]
```

DATA COLLECTION - SPACE X API

3

- Create Global Variable
- Use the Function created in first step to retrieve the required information and store in Global variables
- Create a tuple dictionary using those global variables

4

- Filter the data frame to only include ' Falcon 9' Launches
- Finding the missing value and fill the missing value with the mean
- Export data to CSV File

```
#Global variables
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
ReusedCount = []
Serial = []
Longitude = []
Latitude = []
```

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

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

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

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

```
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}
```

3

```
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9
data_falcon9.isnull().sum()
# Calculate the mean value of PayloadMass column
mean=data_falcon9['PayloadMass'].mean()
# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'].replace(np.nan,mean, inplace=True)
data_falcon9.isnull().sum()
data_falcon9.to_csv('dataset_part_1.csv', index=False)
```

4

DATA COLLECTION – WEB SCRAPING

Collect historical Falcon 9 launch records from a Wikipedia page titled List of Falcon 9 and Falcon Heavy launches using web scraping.

- Create helper functions for you to process web scraped HTML table(Appendix)

1

- Use GET request to parse Falcon9 Launch Wiki page
- Create BeautifulSoup Object from HTML response

2

- Extract all column/variable names from the HTML table header

- Use the column names as keys in a dictionary (See Appendix)
- Use custom functions and logic to parse all launch tables (see Appendix) to fill the dictionary values

3

- Convert the dictionary to a Pandas Data Frame ready for export

1

```
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)
soup = BeautifulSoup(response.text)
soup.title
html_tables = soup.find_all('table')
```

2

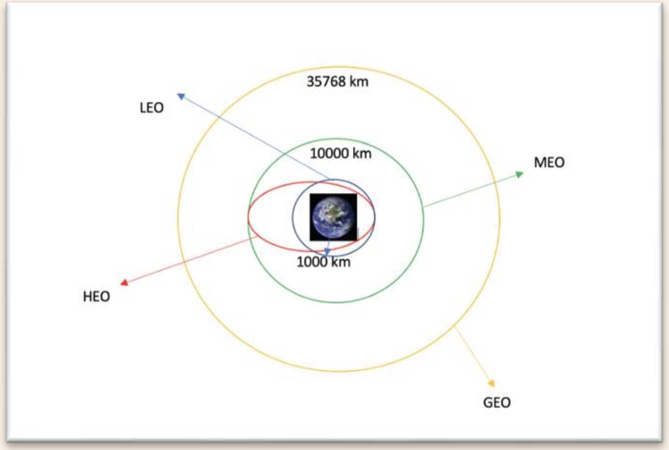
```
column_names = []
for row in first_launch_table.find_all('th'):
    name = extract_column_from_header(row)
    if(name != None and len(name) > 0):
        column_names.append(name)
```

3

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

DATA WRANGLING - I

Space X data set contains different launch sites, each dedicated to different orbits. In the figure, some of the orbits can be seen.



Retrieving the data by using value_counts( ) function

1. The number of launch sites
2. The name of orbits and the number of occurrences for each orbit
3. The number of outcomes and landing outcomes per orbit type.

1

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

2

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

3

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

DATA WRANGLING - II

Interpreting the meaning of outcomes:

- **True Ocean**: the mission outcome was successfully landed in a specific region of the ocean
- **False Ocean**: The mission outcome was an unsuccessful landing in a particular ocean region.
- **True RTLS**: the mission outcome was successfully landed on a ground pad
- **False RTLS**: the mission outcome was unsuccessfully landed on a ground pad.
- **True ASDS**: the mission outcome was successfully landed on a drone ship
- **False ASDS**: the mission outcome was unsuccessfully landed on a drone ship.
- **None ASDS** and **None None** represent a failure to land.

For the target variable (outcome), the landing outcome must be determined as successful = 0 and unsuccessful = 1. The outcome column will be manipulated accordingly.

1. put the label to each outcome type by using the enumerate( ) function
2. Extract the data set that the second landing is unsuccessful as bad_outcome.
3. Label the successful landing as 1 and the bad_outcome as 0.

1

```
for i,outcome in enumerate(landing_outcomes.keys()):
    print(i,outcome)
```

0 True ASDS
1 None None
2 True RTLS
3 False ASDS
4 True Ocean
5 False Ocean
6 None ASDS
7 False RTLS

2

```
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
bad_outcomes
```

3

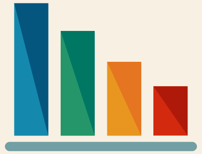
```
landing_class = []

for outcome in df['Outcome']:
    if outcome in bad_outcomes:
        landing_class.append(0)
    else:
        landing_class.append(1)
```

EDA WITH DATA VISUALIZATION

SCATTER CHART

Show the correlation between two variables, helping gain insights into their relationship and interaction.



BAR CHARTS

Compare a numerical value with a categorical variable. Horizontal or vertical bar charts can be used depending on the data size.

LINE CHARTS

Display the change of a variable over time, with numerical values on both axes.

Scatter charts were produced to visualize the relationships between:

- **Flight Number and Launch Site**
- **Payload and Launch Site**
- **Orbit Type and Flight Number**
- **Payload and Orbit Type**

A bar chart was produced to visualize the relationship between:

- **Success Rate and Orbit Type**

Line charts were produced to visualize the relationships between:

- **Success Rate and Year (i.e., the launch success yearly trend)**

EXPLORATORY ANALYSIS USING SQL

To obtain information about the dataset, SQL queries were executed.

1. Display the names of the unique launch sites in the space mission
2. Display five records where launch sites begin with the string 'CCA.'
3. Display the total payload mass carried by boosters launched by NASA (CRS)
4. Display the average payload mass carried by booster version F9 v1.1
5. List the date when the first successful landing outcome on a ground pad was achieved
6. List the names of the boosters that had success on a drone ship and a payload mass between 4000 and 6000 kg
7. List the total number of successful and failed mission outcomes
8. List the names of the booster versions that have carried the maximum payload mass
9. List the failed landing outcomes on drone ships, their booster versions, and launch site names for 2015
10. Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the dates 2010-06-04 and 2017-03-20, in descending order

BUILD AN INTERACTIVE MAP WITH FOLIUM

Marking launch sites and their successful or failed launches on a map:

1. Mark all the launch sites on a map:
   - Begin by initializing the map using a Folium Map object.
   - Use the groupby() function to extract unique launch sites and their coordinates.
   - For each launch site on the map, add a folium.Circle() and folium.Marker().
2. Mark each site's successful/failed launches on a map:
   - Cluster the launch sites with the exact coordinates using the MarkerCluster() function.
   - Determine a green marker color for successful launches (class=1) and a red marker color for failed launches (class=0) before clustering.
   - Create an icon as a text label and assign the icon_color as the marker_color determined previously.
3. Calculate the distances between a launch site and its proximities:
   - To explore the proximities of launch sites, calculate the distances between points using the Lat and Long values.
   - Create a folium using the Lat and Long values after marking a point.Marker( ) object to show the distance.
   - To display the distance line between two points, draw a folium.PolyLine and add it to the map.

BUILD A DASHBOARD WITH PLOTLY DASH

The Plotly Dash dashboard now has two interactive plots to visualize the data.

1. A pie chart (px.Pie()) that displays the percentage of the number of successful launches per site.
   - This chart makes it easy to identify the most successful sites.
   - A filterable chart by using a DCC.Dropdown() object to see a specific site's success/failure ratio.
2. A scatter graph (px.Scatter()) that shows the correlation between the outcome (success or failure) and the payload mass (in kg).
   - A filterable graph using a rangeslider() object to select different ranges of payload masses or by booster version.

PREDICTIVE ANALYSIS(CLASSIFICATION)

Model Development

Prepare the data set for development

- Load the data
- .Separate the target variable from the data set
- Transform the data using NumPy, Standard Scalar, and preprocessing
- Split the data into train set and test set
- Decide the appropriate machine-learning algorithm for the dataset

Create a Parameters dictionary and model for each algorithm for **GridSerachCV()** to tune the parameter

Fit the data into the model by using the training set

Use the fitted model to estimate the predicted outcome by using the testing set

Model Evaluation

For each chosen algorithm:

By using the output GridSearch object, we will determine

- Check the tuned parameter (best_params_)
- Check the accuracy (score and best score)

Plot and examine the confusion matrix

Model assessment

Review the accuracy scores for

all chosen algorithms

The model with the highest

accuracy score is determined as

the best-performing model

RESULTS

- Exploratory Data Analysis
- Interactive Analytics
- Predictive Analysis

Section 2

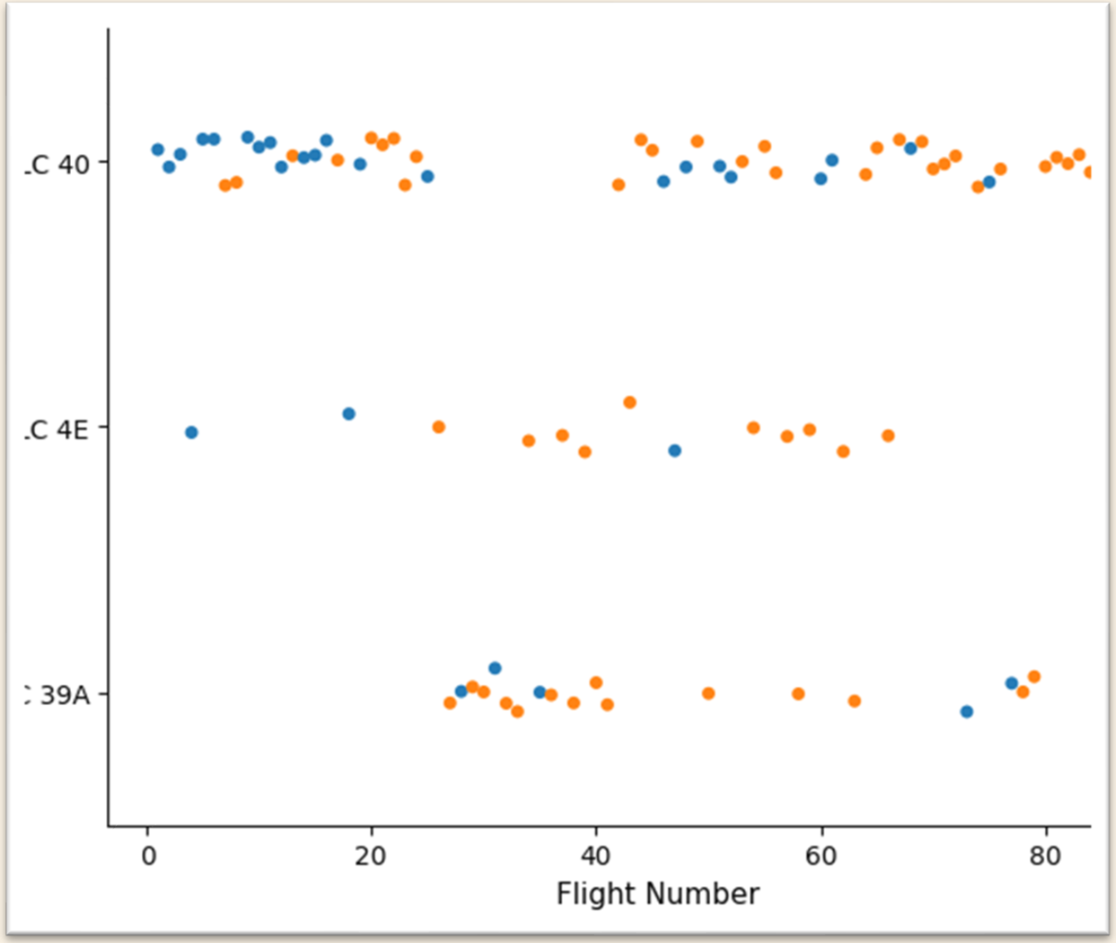
Insights drawn from EDA

EDA WITH - VISUALIZATION

FLIGHT NUMBER VS. LAUNCH SITE

The scatter plot of Flight Number and Launch Site indicates that:

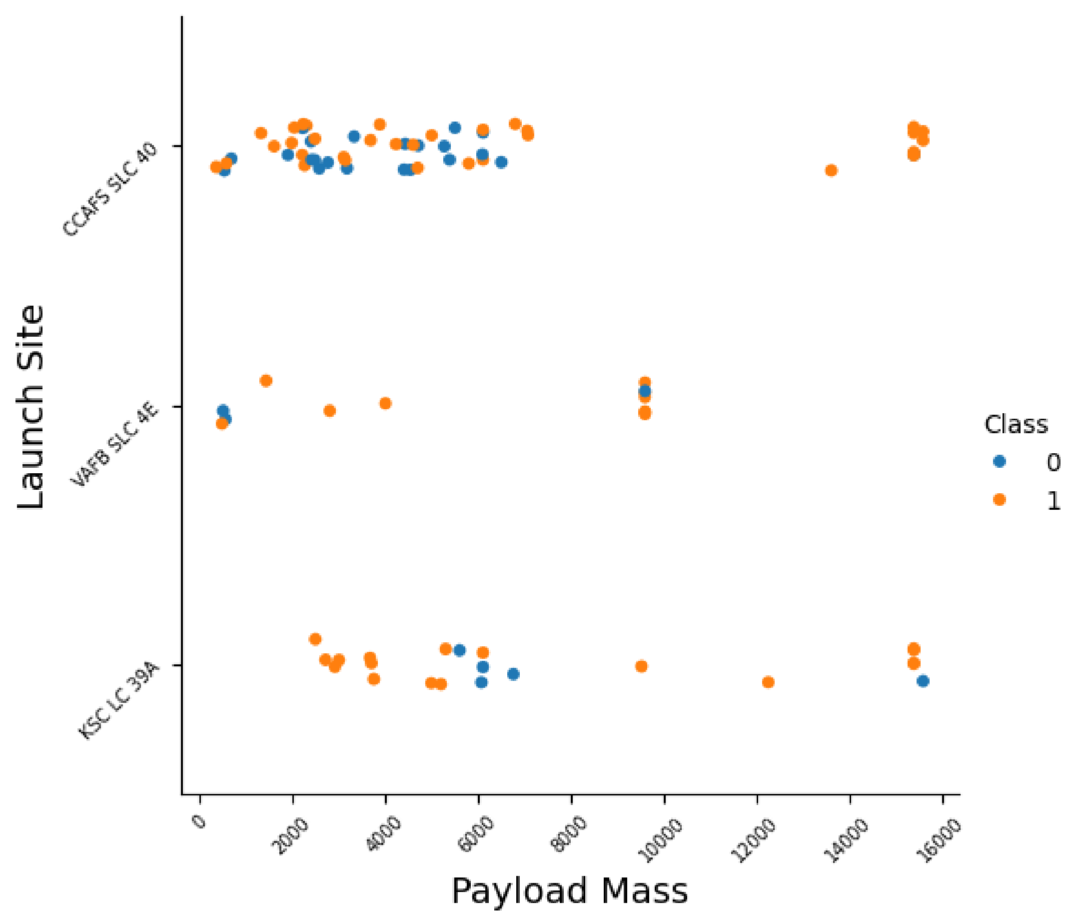
1. The higher flight number results in a higher first-stage landing success rate, while the success rate for flights with less than 30 attempts is lower.
2. Most flights are at the launch site (CCAFS SLC 40). So, we can see the mixing for both successful and unsuccessful attempts.
3. No launching from KSCLC 39A can be seen for the flight less than 30.
4. A few flights are launched at VAFB SLC 4E, but with the higher success rate.



PAYLOAD VS. LAUNCH SITE

The scatter plot of Payload Mass and Launch Site indicates that:

1. There is no exact correlation between the payload mass and the launch site.
2. At CCAFS SLC 40, landings have been successful for payloads weighing less than 7000 kg. At VAFB SLC-4E, launches of payloads weighing 10000 kg have a high success rate. However, at KSC LC 39A, around 6000 kg payloads have been less successful than other launches.
3. The outliers are on the launch site CCAFS SLC 40 and KSC LC 39A.



SUCCESS RATE VS. ORBIT TYPE

The bar plot of orbit type success rate indicates that:

1. There are 4 types of orbit with a 100% successful rate.

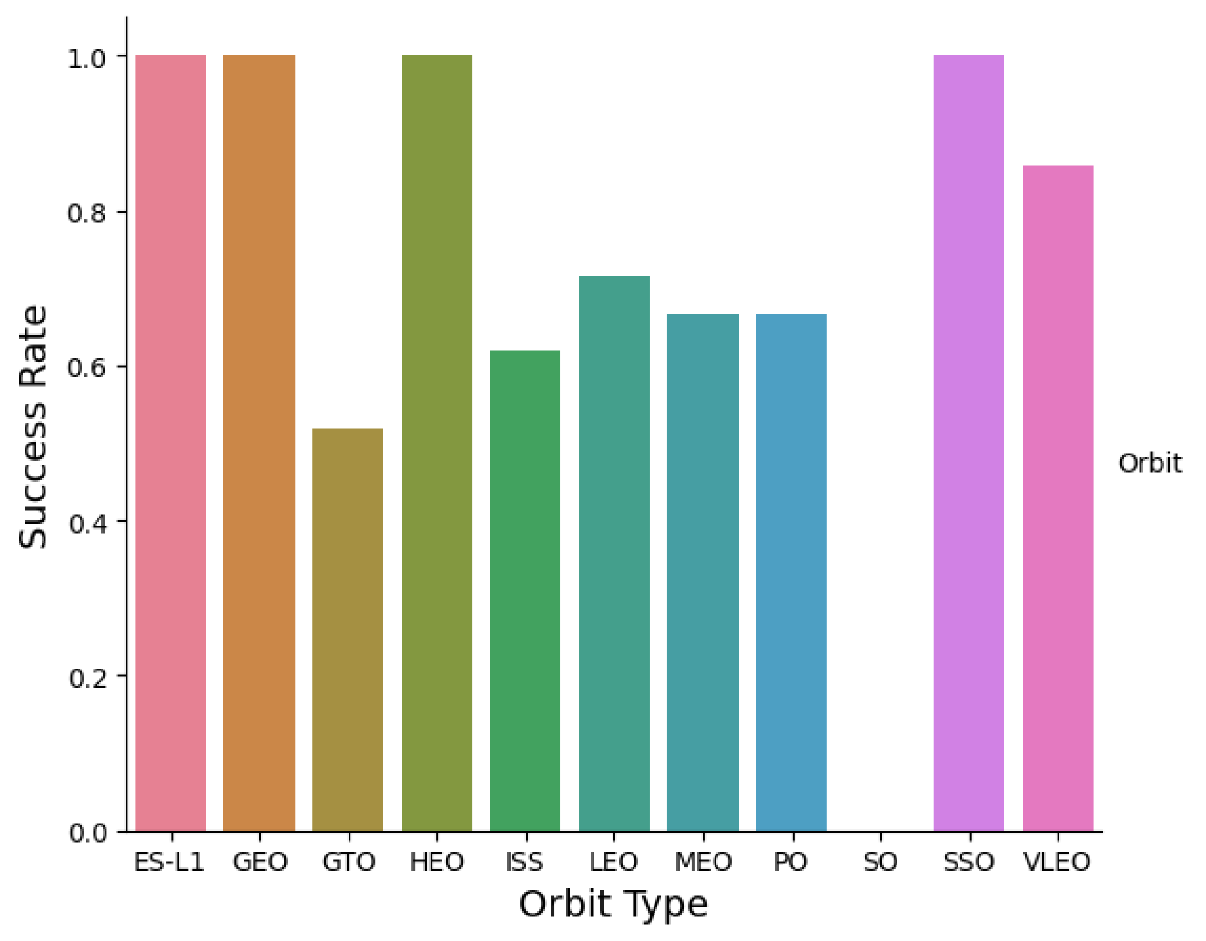
   ES-L1

   GEO

   HEO

   SSO

2. It appears that no launches have been made for the orbit type SO.
3. The success rate of other orbits is between 50% and 70%.



FLIGHT NUMBER VS. ORBIT TYPE

The scattered plot of flight number and orbit type indicates that:

1. The following orbit type has a 100% successful rate in that graph, too.

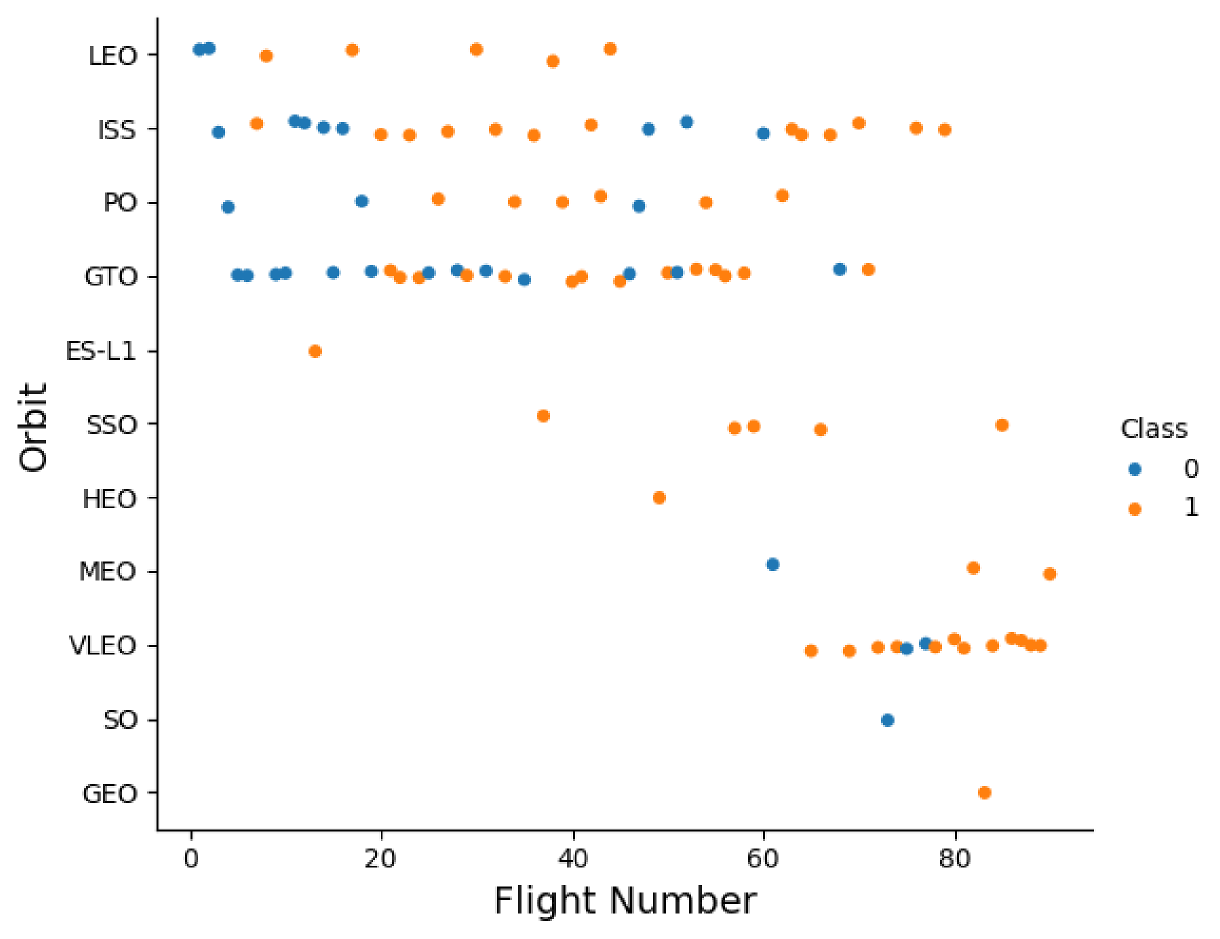
   ES-L1

   GEO

   HEO

   SSO

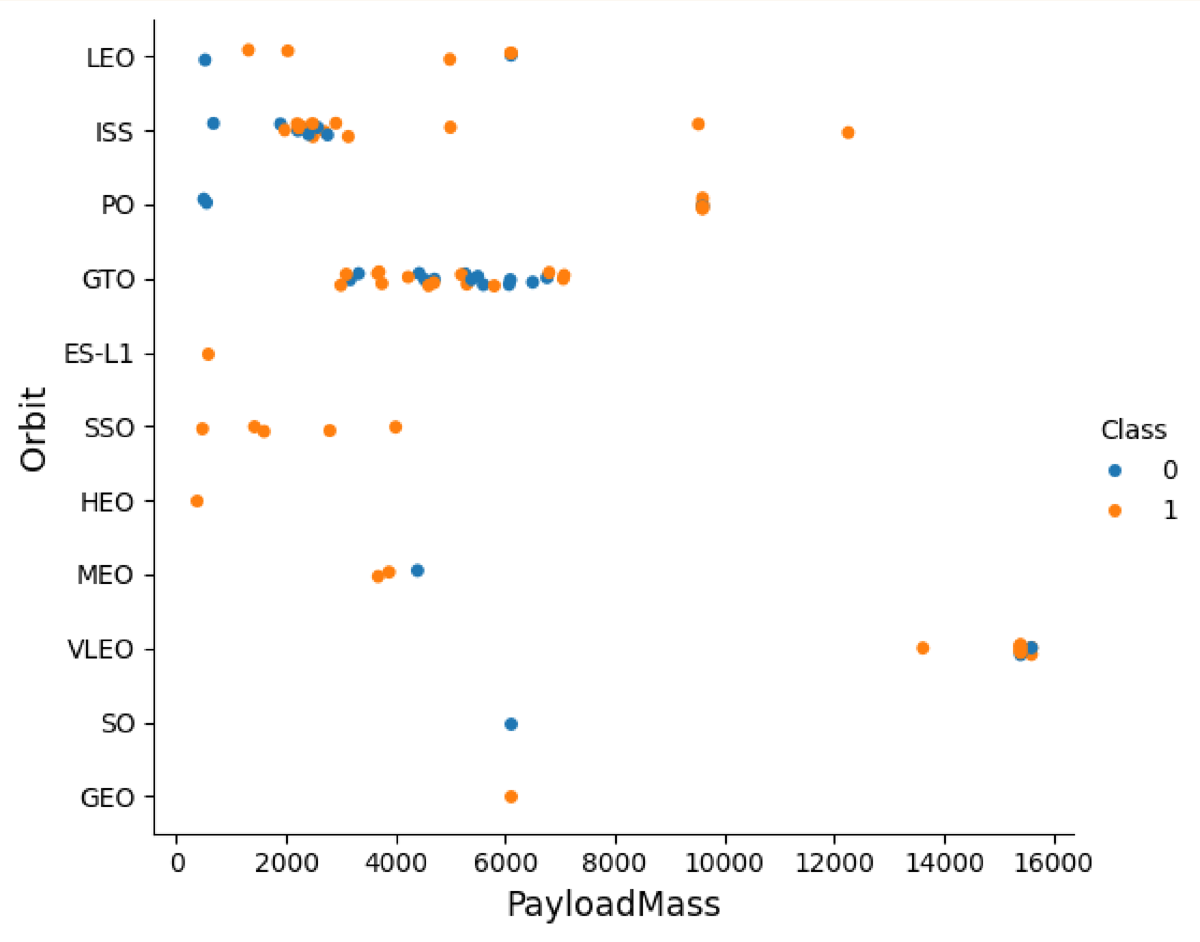
2. The correlation of successful rate correlation between the flight number and the orbit type can only be seen on LEO. And the others have no significant correlation.



PAYLOAD VS. ORBIT TYPE

The scattered plot of payload and orbit type indicates that:

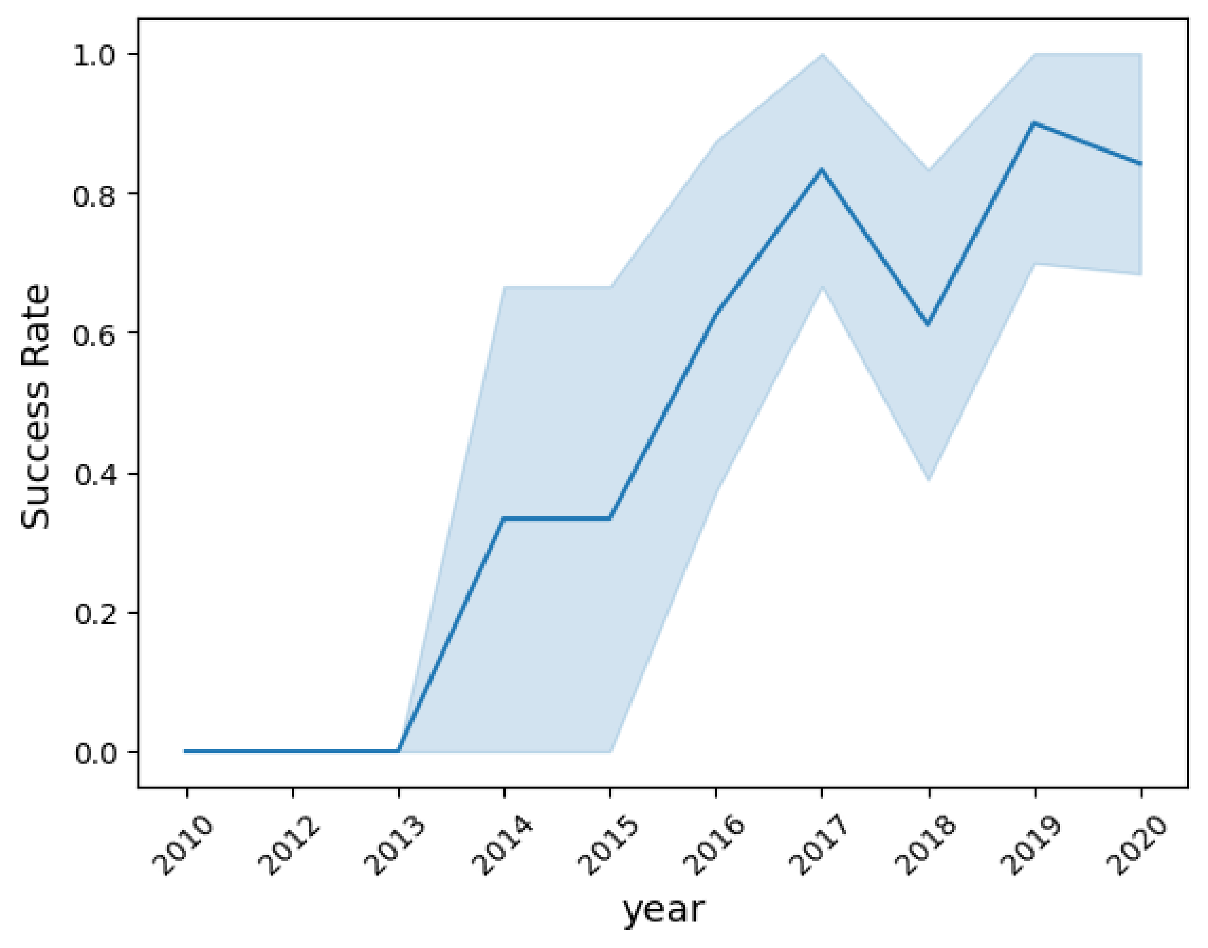
1. For Polar, LEO, and ISS orbits with heavy payloads, the successful or positive landing rate is higher
2. It's challenging to distinguish between success and failure in GTO landing
3. The correlation between the successful landing rate for flight number and orbit type is only significant for LEO. There is no significant correlation for the other orbits.



LAUNCH SUCCESS YEARLY TREND

The Line chart of the yearly launch success rate indicates that:

1. Between 2010 and 2013, there were no successful landings, as the success rate was 0%.
2. Since 2013, the success rate has generally increased, with slight dips in 2018 and 2020.
3. After 2016, the probability of success was always greater than 50%.



EDA – WITH SQL

ALL LAUNCH SITE NAMES

Find the names of the unique launch sites

```
%sql select  distinct Launch_Site from SPACEXTABLE LIMIT 3
```

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

The word distinct returns unique values from the LAUNCH_SITE column of the SPACEXTBL table.

LAUNCH SITE NAMES BEGIN WITH 'CCA'

Find 5 records where launch sites begin with `CCA`

```
%sql select * from SPACEXTABLE where Launch_Site LIKE 'CCA%' LIMIT 5
```

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	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	0:35:00	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

The LIKE keyword is used with the wild card ' to retrieve string values beginning with 'CCA' and LIMIT 5 fetches only 5 records.

TOTAL PAYLOAD MASS

Calculate the total payload carried by boosters from NASA

```
%sql SELECT SUM(PAYLOAD_MASS__KG_) AS TOTAL_PAYLOAD_MASS
FROM SPACEXTABLE WHERE CUSTOMER = 'NASA (CRS)'
```

TOTAL_PAYLOAD_MASS
45596

The SUM keyword summates payload_mass_kg and the associate function filters boosters only to 'NASA(CRS)'.

AVERAGE PAYLOAD MASS BY F9 V1.1

Calculate the average payload mass carried by booster version F9 v1.1

```
%sql select AVG(PAYLOAD_MASS__KG_) AS AVERAGE_PAYLOAD_MASS
FROM SPACEXTABLE WHERE Booster_Version LIKE 'F9 v1.1%'
```

AVERAGE_PAYLOAD_MASS
2534.6666666666665

The AVG keyword calculates the average payload_mass_kg and the association functions filiters boosters only to ‘NASA(CRS)’.

FIRST SUCCESSFUL GROUND LANDING DATE

Find the dates of the first successful landing outcome on the ground pad

```sql
%sql select min(Date) from SPACEXTABLE where
Landing_Outcome = 'Success (ground pad)'
```

min(Date)
2015-12-22

The DATE column's minimum is calculated using MIN. The results are filtered using WHERE to show successful ground pad landings only.

SUCCESSFUL DRONE SHIP LANDING WITH PAYLOAD BETWEEN 4000 AND 6000

List the names of boosters that have successfully landed on drone ships and had payload mass more significant than 4000 but less than 6000

```
%sql select Booster_Version from SPACEXTABLE where
Landing_Outcome= 'Success (drone ship)' AND
PAYLOAD_MASS__KG_ > 4000 AND PAYLOAD_MASS__KG_ <6000
```

Mission_Outcome	COUNT
Failure (in flight)	1
Success	98
Success	1
Success (payload status unclear)	1

The WHERE keyword filters the results to include only those that satisfy both conditions in the brackets (as the AND keyword is also used). The two conditions x> 4000 and x < 6000 values are fulfilled by using AND keywords.

BOOSTERS CARRIED MAXIMUM PAYLOAD

List the names of the booster which have carried the maximum payload mass

```
%sql select  DISTINCT Booster_Version  from SPACEXTABLE
where   PAYLOAD_MASS__KG_ = (select MAX(PAYLOAD_MASS__KG_)
from SPACEXTABLE )
```

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

By using subquery, the SELECT statement within the brackets finds the maximum payload, and this value is used in the WHERE condition. The DISTINCT keyword is then used to retrieve only distinct/unique booster versions.

TOTAL NUMBER OF SUCCESSFUL AND FAILURE MISSION OUTCOMES

Calculate the total number of successful and failure mission outcomes

```
%sql select Mission_Outcome, count(Mission_Outcome) as
COUNT from SPACEXTABLE group by Mission_Outcome
```

Mission_Outcome	COUNT
Failure (in flight)	1
Success	98
Success	1
Success (payload status unclear)	1

The COUNT keyword calculates the total outcomes of missions and GROUPBY groups them by outcome type.

2015 LAUNCH RECORDS

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

```
%sql select substr(Date,6,2) as MONTH,Booster_Version,Launch_Site,Payload,
PAYLOAD_MASS__KG_,Orbit, Customer,Mission_Outcome from SPACEXTABLE where
Landing_Outcome = 'Failure (drone ship)' AND substr(Date,0,5) = '2015'
```

MONTH	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome
01	F9 v1.1 B1012	CCAFS LC-40	SpaceX CRS-5	2395	LEO (ISS)	NASA (CRS)	Success
04	F9 v1.1 B1015	CCAFS LC-40	SpaceX CRS-6	1898	LEO (ISS)	NASA (CRS)	Success

Select a substring of the Date column starting from the 6th character, take two characters, and label it MONTH. From SPACEXTABLE, select Booster_Version, Launch_Site, Payload, PAYLOAD_MASS__KG_, Orbit, Customer, Mission_Outcome. Filter by Failure(drone ship) and year 2015.

RANK LANDING OUTCOMES BETWEEN 2010-06-04 AND 2017-03-20

Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order

```
%sql select  Landing_Outcome, count(Landing_Outcome) as
COUNT from SPACEXTABLE where Date between '2010-06-04' and
'2017-03-20' group by Landing_Outcome order by COUNT DESC
```

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

Use WHERE and BETWEEN to filter dates, then GROUP BY and ORDER BY to sort. Use DESC for descending order.

Section 3

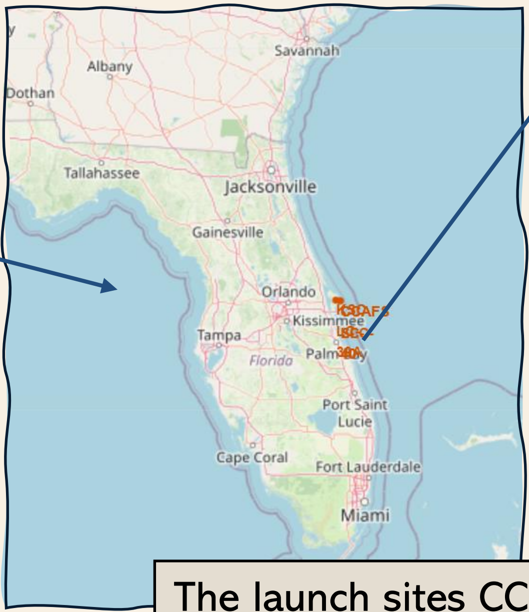
Launch Sites Proximities Analysis

MARK ALL LAUNCH SITES ON MAP

The launch sites exist in the coastal areas of two states, California, and Florida. One in California and three in Florida.

VAFB SLC 4E

Three sites in Florida

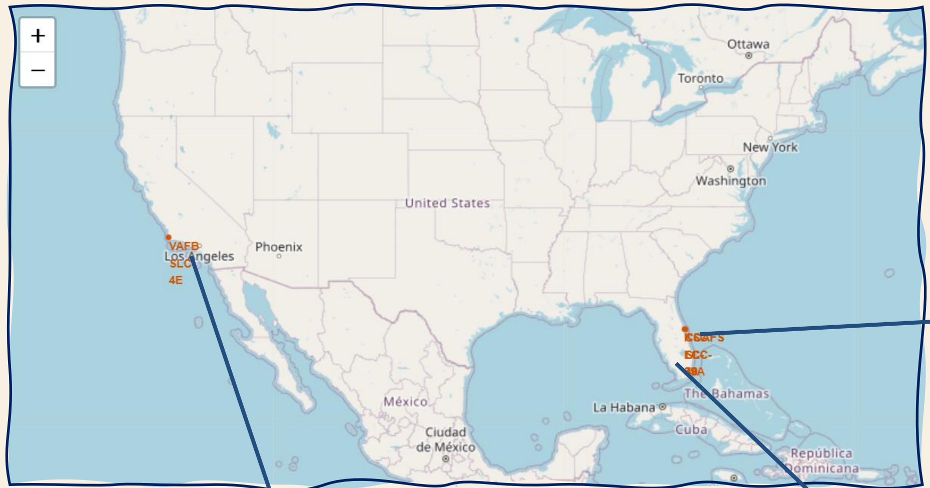


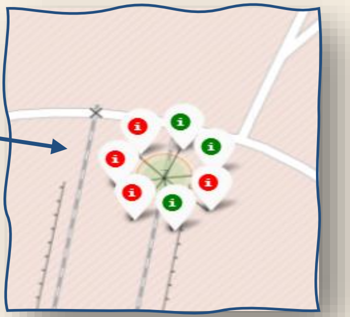
KSC LC-39A

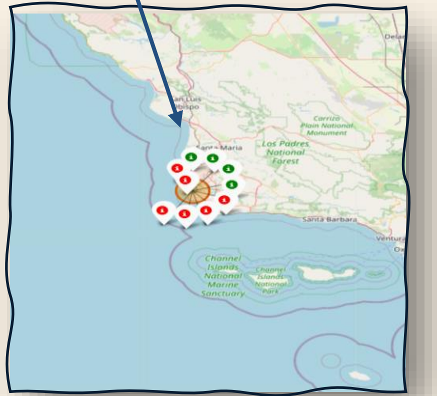
The launch sites CCAFS SL 40 and CCAFS FSL 40 are in the same place: Cape Canaveral Space Force Station.

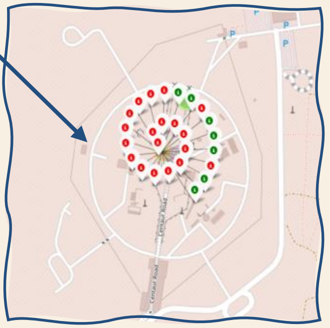
MARK THE SUCCESSFUL/FAILED LAUNCHES FOR EACH SITE ON THE MAP

Launch Sites were grouped into clusters and annotated with Green icon for successful launches, and red icon for failure lunch sites.

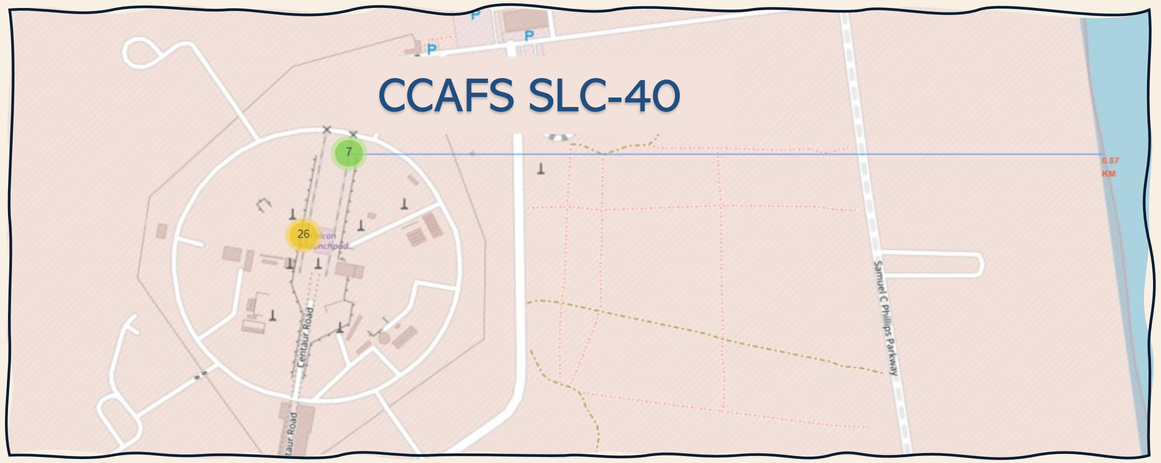


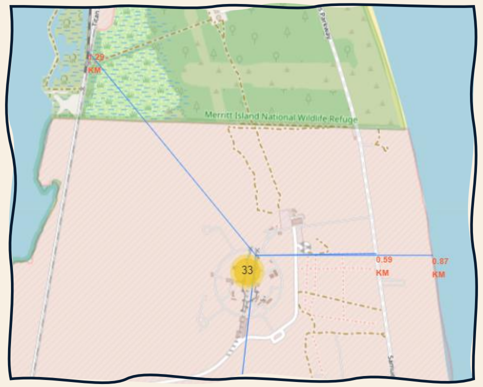


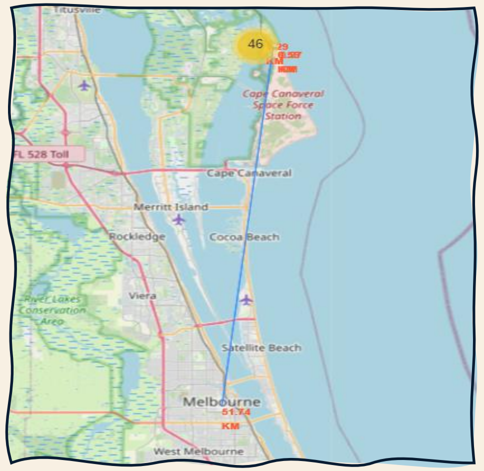




THE DISTANCES BETWEEN A LAUNCH SITE AND ITS PROXIMITIES







The analysis answers the following questions using the CCAFS SLC-40 launch site as a sample site.

Is launch site in close proximity to railways?

Yes. The nearest railway is 1.29 Km.

Is launch site in close proximity to highways?

Yes, the nearest highway is 0.59 km.

Is launch site in close proximity to coastline?

Yes, the nearest coastal line is 0.87 km.

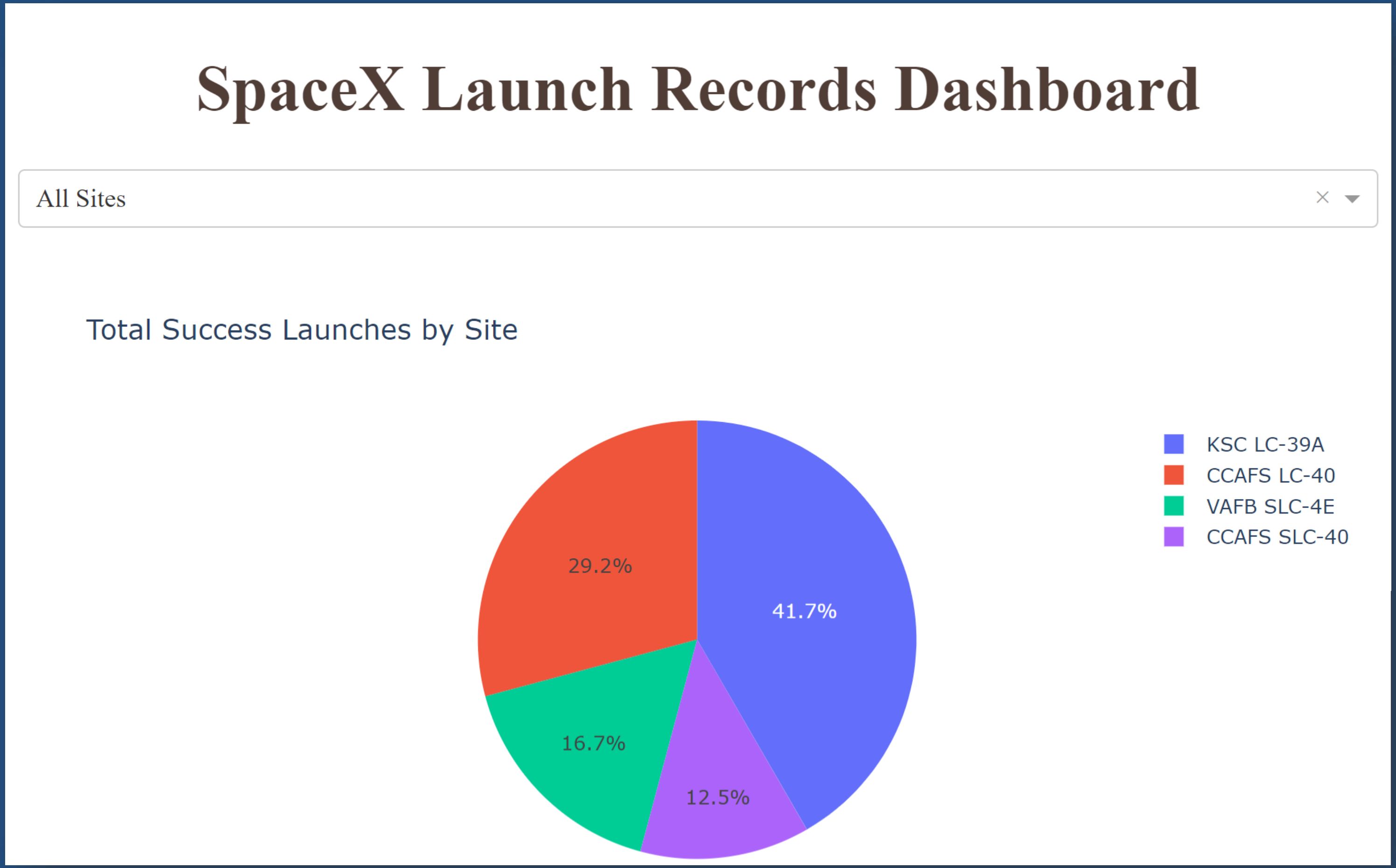
Does launch sites keep certain distance away from cities?

Yes, the nearest city is Melbourne (51.74cm)

Section 4

Build a Dashboard with Plotly Dash

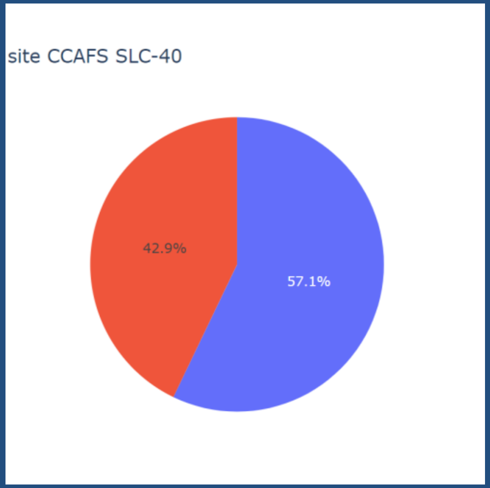
THE DISTANCES BETWEEN A LAUNCH SITE AND ITS PROXIMITIES



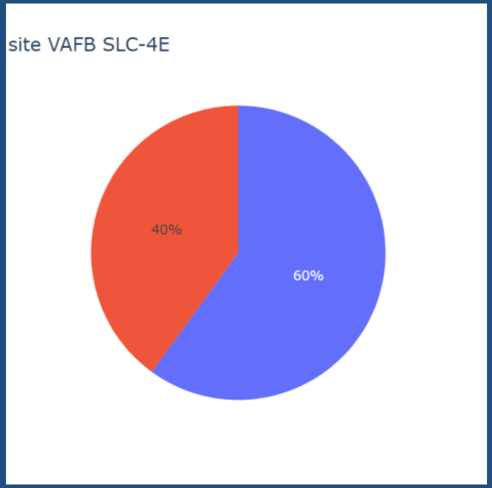
The launch site KSC LC -39A in Florida has the highest success rate, with 41.7%

ANNOTATE SITE WITH THE HIGHEST LAUNCH RATE

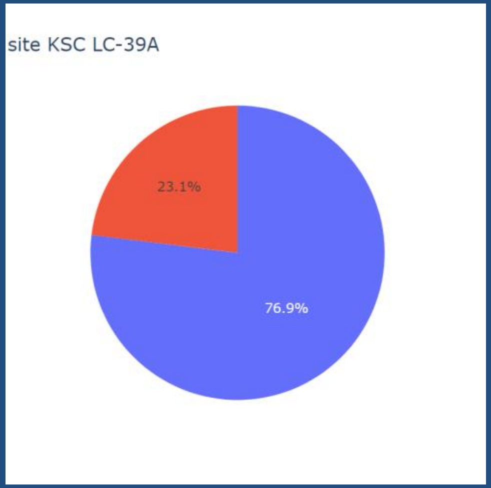
failure success



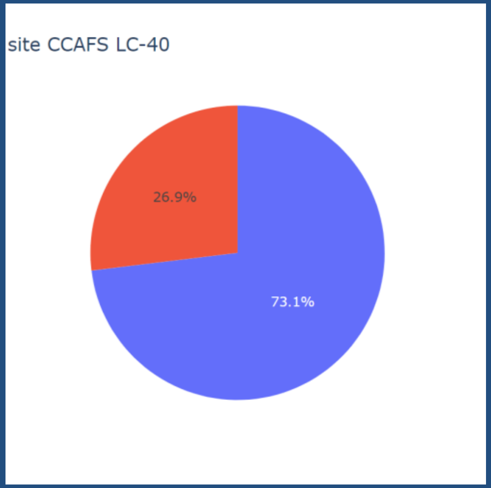
CCAFS LC -40



VAFB SLC-40



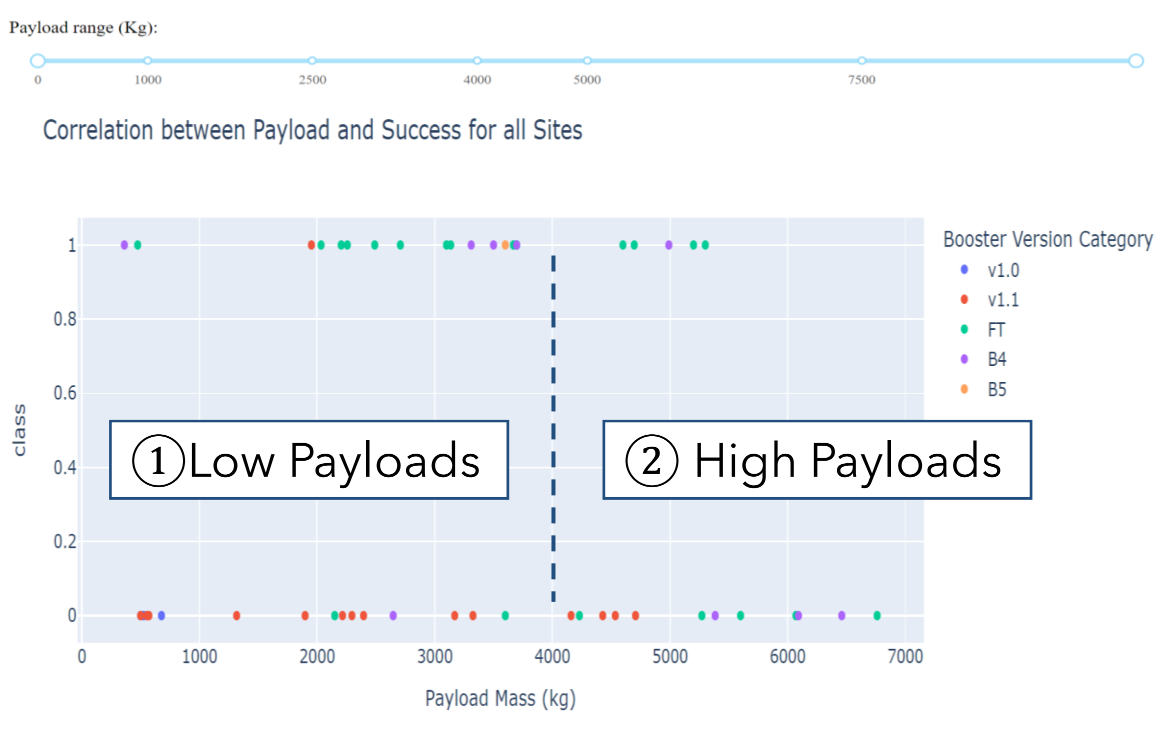
KSCLC-39A

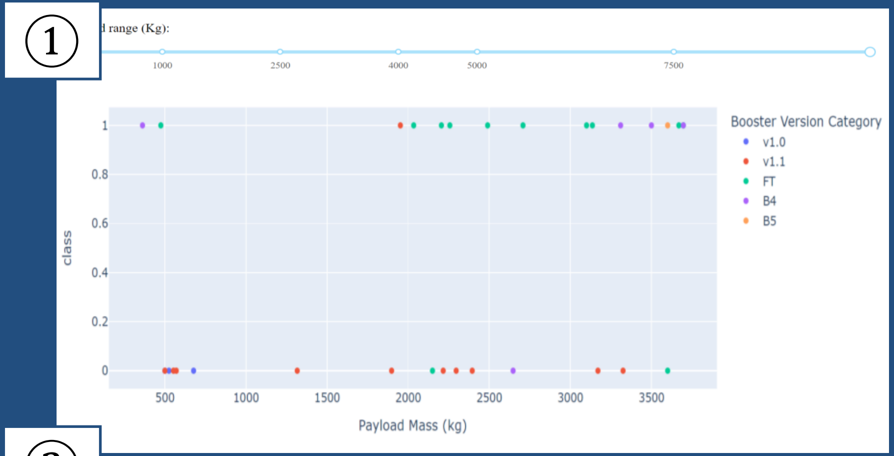


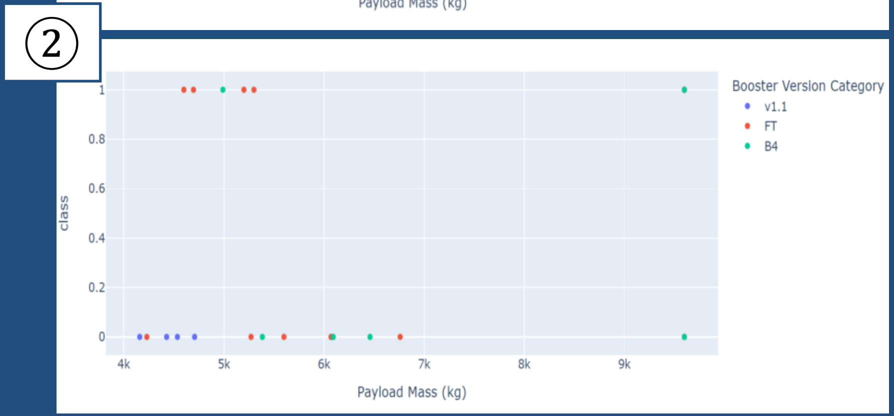
CCAFS LC-40

The pie charts indicate that KSCLC – 39 A has the highest success rate at 76.9%.

LAUNCH OUTCOME VS. PAYLOAD SCATTER PLOT FOR ALL SITES







LAUNCH OUTCOME VS. PAYLOAD SCATTER PLOT FOR ALL SITES

1st quarter
0~1999 Kg

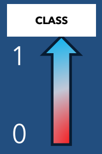
2nd quarter
2000~3999Kg

3rd quarter
4000~5999Kg

4th quarter
6000kg and above

We will divide the data range into four quarters because there is a significant gap for payload mass at 2000, 4000, and 6000 kg.

COMPARISON OF DIFFERENT RANGES



0 failure
1 success

The success rate for each quarter was calculated by dividing successful outcomes by the total outcomes. The following insights can be drawn:

- Highest success rate : 2nd quarter and lowest success rate : 4th quarter
- The Falcon 9 with massive payload mass has a lower success rate.

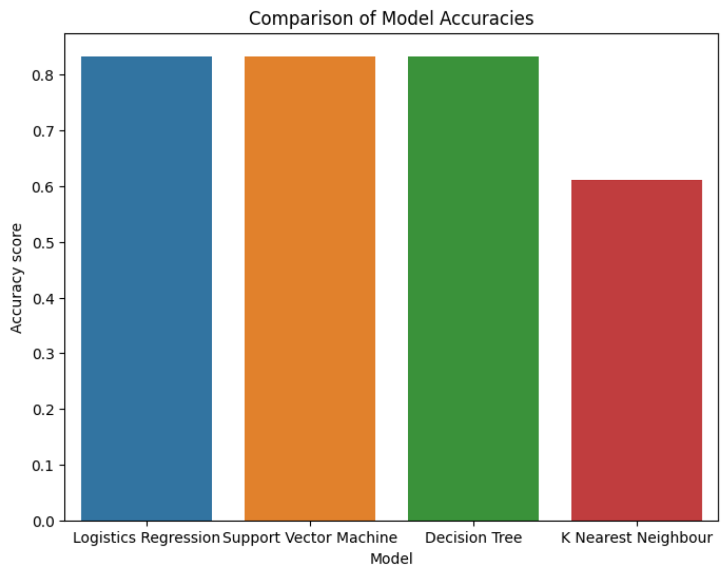
Section 5

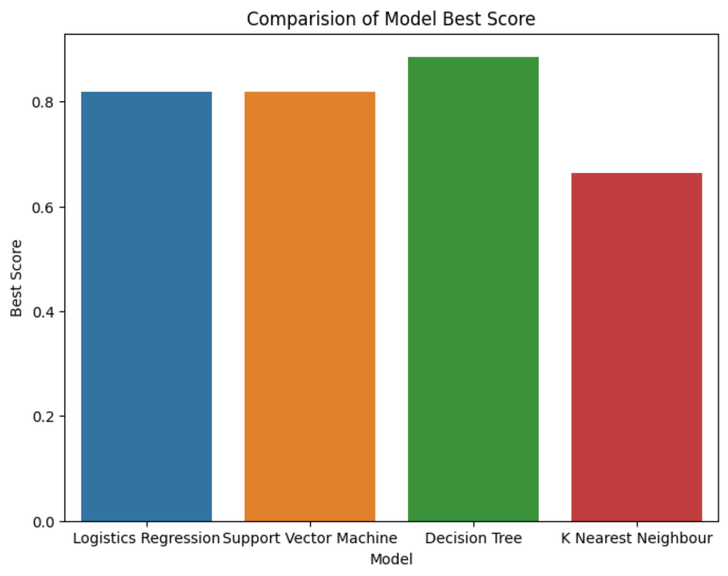
Predictive Analysis (Classification)

CLASSIFICATION ACCURACY

Plotting the Accuracy Score and Best Score for each classification algorithm produces the The following result:

- **The Decision Tree model has the highest classification accuracy**
- **The Accuracy Score is 88.6%**
- **The Best Score is 88.3%**



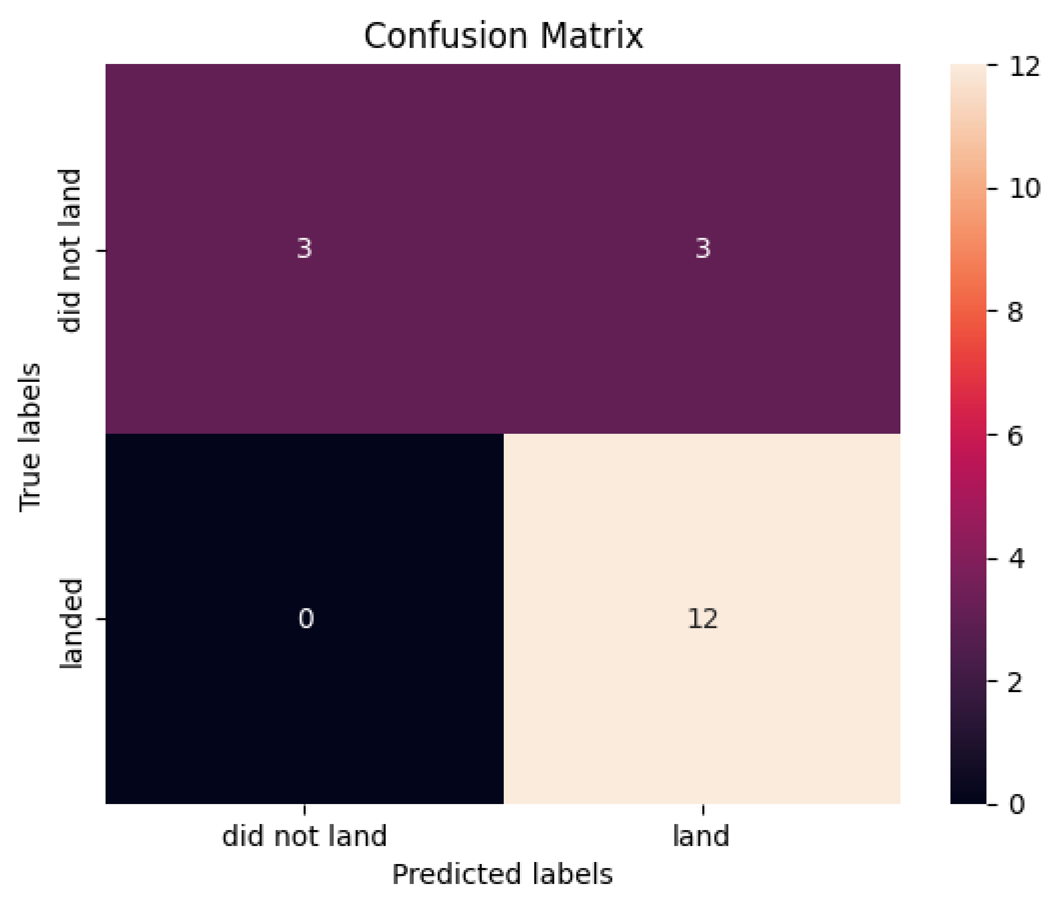


	model	Best_Score	Accuracy Score
0	Logistics Regression	82.0%	83.3%
1	Support Vector Machine	81.9%	83.3%
2	Decision Tree	88.6%	83.3%
3	K Nearest Neighbour	66.4%	61.1%

CONFUSION MATRIX

The best performing classification model is Decision Tree model, with an accuracy of 88.6%.

- **The confusion matrix explains this, which shows only 3 out of 18 total results classified incorrectly (a false positive, shown in the top right corner).**
- **The other 17 results are correctly classified (3 did not land, 12 did land).**



CONCLUSION

CONCLUSION

Flight Number Analysis

- **The higher the number of flights, the higher the success rate** at a launch site, with most early flights being unsuccessful. I.e., with more experience, the success rate increases.

Timeline Analysis

- Between 2010 and 2013, the success rate for landings was 0%. Since then, the rate has generally increased, with slight dips in 2018 and 2020.

Orbit Type Analysis and payload analysis

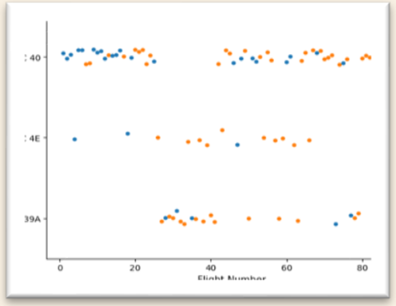
- The higher the payload mass(Kg), the lower the success rate.
- The orbit types ES-L1, GEO, HEO, and SSO have a 100% success rate.
- The orbit types ES-L1, GEO, and HEO have only one space launch, whereas SSO has 5 out of 5 successful Falcon 9 launches.
- The orbit types FT and B4 are used from the lowest payload to the heavier payload with the higher success rate for FT.

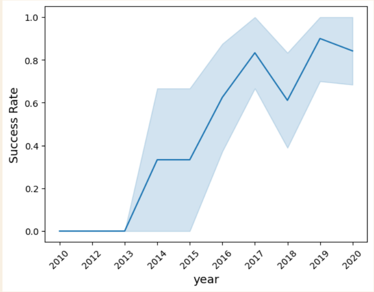
Launch Site Analysis

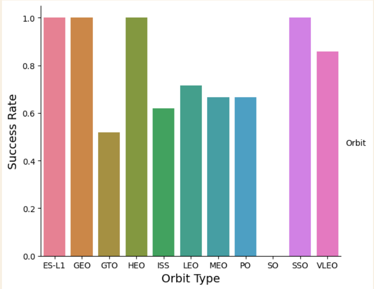
- KSC LC 39A has the highest success rate (41.7%)within the four launch sites, with a success rate of 76.9%.

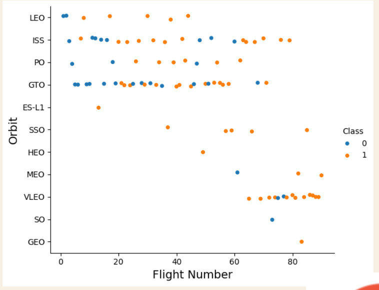
Model Estimation

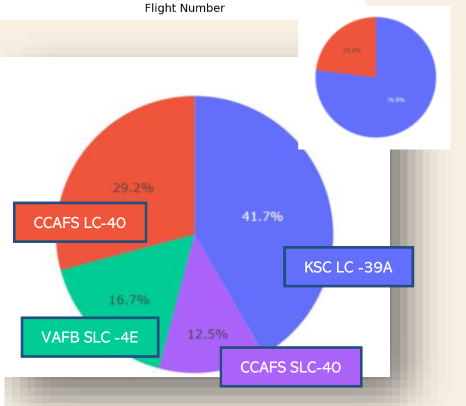
- The best-performing classification model is the Decision Tree model, with an accuracy of 88.6%.

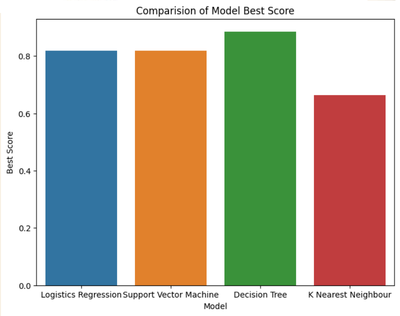












APPENDIX

DATA COLLECTION - SPACE X API

Creating Custom Function to retrieve necessary data

```
# Takes the dataset and uses the rocket column to call the API and
append the data to the list
def getBoosterVersion(data):
    for x in data['rocket']:
       if x:
        response =
requests.get("https://api.spacexdata.com/v4/rockets/"+str(x)).json(
)
        BoosterVersion.append(response['name’])
# Takes the dataset and uses the launchpad column to call the API
and append the data to the list
def getLaunchSite(data):
    for x in data['launchpad']:
       if x:
         response =
requests.get("https://api.spacexdata.com/v4/launchpads/"+str(x)).js
on()
         Longitude.append(response['longitude'])
         Latitude.append(response['latitude'])
         LaunchSite.append(response['name'])
# Takes the dataset and uses the payloads column to call the API
and append the data to the lists
def getPayloadData(data):
    for load in data['payloads']:
       if load:
        response =
requests.get("https://api.spacexdata.com/v4/payloads/"+load).json()
        PayloadMass.append(response['mass_kg'])
        Orbit.append(response['orbit’])
```

```
# Takes the dataset and uses the cores column to call the API and
append the data to the lists

def getCoreData(data):
    for core in data['cores']:
        if core['core'] != None:
            response =
requests.get("https://api.spacexdata.com/v4/cores/"+core['core']).json()
            Block.append(response['block'])
            ReusedCount.append(response['reuse_count'])
            Serial.append(response['serial'])
        else:
            Block.append(None)
            ReusedCount.append(None)
            Serial.append(None)
        Outcome.append(str(core['landing_success'])+'
'+str(core['landing_type']))
        Flights.append(core['flight'])
        GridFins.append(core['gridfins'])
        Reused.append(core['reused'])
        Legs.append(core['legs'])
        LandingPad.append(core['landpad'])
```

DATA COLLECTION - WEB SCRAPING

Creating helper functions for you to process web scraped HTML table

```
def date_time(table_cells):
    """
    This function returns the data and time from the
HTML  table cell
    Input: the  element of a table data cell extracts extra row
    """
    return [data_time.strip() for data_time in
list(table_cells.strings)][0:2]

def booster_version(table_cells):
    """
    This function returns the booster version from the
HTML  table cell
    Input: the  element of a table data cell extracts extra row
    """
    out=''.join([booster_version for i,booster_version in
enumerate( table_cells.strings) if i%2==0][0:-1])
    return out

def landing_status(table_cells):
    """
    This function returns the landing status from the HTML
table cell
    Input: the  element of a table data cell extracts extra row
    out=[i for i in table_cells.strings][0]
    return out
```

```
def get_mass(table_cells):
    mass=unicodedata.normalize("NFKD",table_cells.text).strip()
    if mass:
        mass.find("kg")
        new_mass=mass[0:mass.find("kg")+2]
    else:
        new_mass=0
    return new_mass

def extract_column_from_header(row):
    """
    This function returns the landing status from the HTML table
cell
    Input: the  element of a table data cell extracts extra row
    """
    if (row.br):
        row.br.extract()
    if row.a:
        row.a.extract()
    if row.sup:
        row.sup.extract()

    column_name = ' '.join(row.contents)
    # Filter the digit and empty names
    if not(column_name.strip().isdigit()):
        column_name = column_name.strip()
        return column_name
```

DATA COLLECTION - WEB SCRAPING

Use the column names as keys in a dictionary

```
launch_dict= dict.fromkeys(column_names)
# Remove an irrelevant 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['Time']=[]
```

DATA COLLECTION - WEB SCRAPING

Use custom functions and logic to parse all launch tables to fill the dictionary values

```
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders collapsible")):
   # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number corresponding to launch a number
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
        else:
            flag=False
        #get table element
        row=rows.find_all('td')
        #if it is number save cells in a dictonary
        if flag:
            extracted_row += 1
            # Flight Number value
            # Append the flight_number into launch_dict with key `Flight No.`
            launch_dict["Flight No."].append(flight_number)

            # Date value
            #Append the date into launch_dict with key `Date`
            datatimelist=date_time(row[0])
            date = datatimelist[0].strip(',')
            launch_dict["Date"].append(date)

            # Time value
            #Append the time into launch_dict with key `Time`
            time = datatimelist[1]
            launch_dict["Time"].append(time)

            # Booster version
            #Append the bv into launch_dict with key `Version Booster`
            bv=booster_version(row[1])
            if not(bv):
                bv=row[1].a.string
            launch_dict["Version Booster"].append(bv)
```

```
            # Launch Site
            #Append the bv into launch_dict with key `Launch site`
            launch_site = row[2].a.string
            launch_dict['Launch site'].append(launch_site)

            # Payload
            #Append the payload into launch_dict with key `Payload`
            payload = row[3].a.string
            launch_dict['Payload'].append(payload)

            # Payload Mass
            #Append the payload_mass into launch_dict with key `Payload mass`
            payload_mass = get_mass(row[4])
            launch_dict['Payload mass'].append(payload_mass)

            # Orbit
            #Append the orbit into launch_dict with key `Orbit`
            orbit = row[5].a.string
            launch_dict['Orbit'].append(orbit)

            # Customer
            #Append the customer into launch_dict with key `Customer`
            if row[6].a != None:
                customer = row[6].a.string
            else:
                customer = 'None'

            launch_dict['Customer'].append(customer)

            # Launch outcome
            #Append the launch_outcome into launch_dict with key `Launch outcome`
            launch_outcome = list(row[7].strings)[0]
            launch_dict['Launch outcome'].append(launch_outcome)

            # Booster landing
            #Append the booster landing into launch_dict with key `Booster landing`
            booster_landing = landing_status(row[8])
            launch_dict['Booster landing'].append(booster_landing)


            print(f"Flight Number: {flight_number}, Date: {date}, Time: {time} \n \
            Booster Version {bv}, Launch Site: {launch_site} \n \
            Payload: {payload}, Orbit: {orbit} \n \
            Customer: {customer}, Launch Outcome: {launch_outcome}\
            Booster Landing: {booster_landing} \n \
            *** ")
```

THANK YOU

PHOO PHOO MON MYAT THU

Linkedin - https://www.linkedin.com/in/phoo-phoo-mon-myat-thu-17461a9b/