Winning the Space race with Data Science

Morten Skaar

22/08/2023

Outline

Executive Summary

Methodologies:

Data Collection

Data Wrangling.

Exploratory Data Analysis

Interactive Visual Analytics

Predictive Analysis using machine learning

Results:

EDA insights into data patterns.

Geospatial data representation.

Interactive dashboard.

Visualization of machine learning outcome

Introduction

- SpaceX offers rocket launches at a competitive price of $62m, significantly lower than competitors who charge upwards of $165m. A key factor in this price difference is SpaceX's capability to reuse rockets. By accurately predicting the success rate of rocket launches, we can position ourselves advantageously in contract bids. In this presentation, we'll analyze publicly available data and use machine learning to identify factors determining the success of a stage 1 rocket landing.

Methodology

Methodology

- Executive Summary
- Data collection methodology:
  - Data was gathered both through web scraping Wikipedia and through SpaceX API
- Perform data wrangling
  - The data was standardized through one shot encoding
- Performed exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - Performed a grid search for all models to obtain the best parameter and compare these for each model

Data Collection – SpaceX API

1) Gather the initial data from the SpaceX API and perform some transformations.

2) Generate list objects and execute custom functions to gather data from other api endpoints. And combine the lists into a dictionary

3) Convert the dictionary into a pandas DataFrame

4) Reviewing missing datapoints and Cleaning the dataset by; replacing null with mean, resetting the Flight numbers to a continuous range

GitHub Link

1)

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

response = requests.get(spacex_url)

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

2)

```
#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)

```
# Create a data from launch_dict
Data = pd.DataFrame(launch_dict)
```

4)

```
data_falcon9.isnull().sum()
#Remove rows with Falcon 9 refrences
data_falcon9 = Data[Data['BoosterVersion'] != 'Falcon 1']
#Reset the Flightnumber column to make a single continous list
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))

# Calculate the mean value of PayloadMass column
mean payloadmass = data falcon9['PayloadMass'].mean()
# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'] = data_falcon9['PayloadMass'].replace(np.nan, mean_payloadmass)
```

Data Collection – Wiki Web scraping

1) Set the URL to the site to be scraped. Use the get function to access the data. Parse the data using Beautiful Soup and save the content to object soup.

2) Locate the table content contained in the HTML code.

3) Generate a list object and by iteration populate it based on the <th> tag within the HTML code. Generate a Dictionary from the list and generate list object within the dictionary.

4) Populate the dictionary with custom functions (see Appendix) and convert the resulting dictionary into a pandas DataFrame.

GitHub Link

1)

```
# Define an object containg the URL to be used
static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"

# use requests.get() method with the provided static_url
result = requests.get(static_url)

# assign the response to a object
response = result.content

# Create a Beutiful Soup Object from response
soup = BeautifulSoup(response,'lxml')
soup.prettify()
```

2)

```
# Use the find_all function in the BeautifulSoup object, with element type `table`
table_content = soup.find_all('table')
#Assigne the table contet into object html_table
html_tables = table_content
```

3)

```
# Create a list object
column_names = []

headers = first_launch_table.find_all('th')

for header in headers:
    header_text = extract_column_from_header(header)
    if header_text and len(header_text)>0:
        column_names.append(header_text)
```

```
#Convert the list into a dictionary
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'] = []

launch_dict['Version Booster']=[]
launch_dict['Booster landing']=[]
launch_dict['Date']=[]
launch_dict['Time']=[]
```

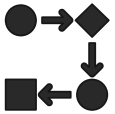
4)

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

Data Wrangling

Review the structure of the data; missing datapoints, data types and types of categories within

Review the different outcomes and generate a set containing the values for bad outcomes.



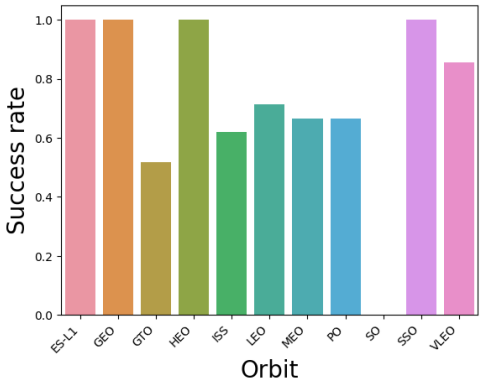
Create an empty list to contain the classes of the launches. Iterate through all the Outcomes and assign a value based on the result of the landing (1 = Success, 0 = Failure)

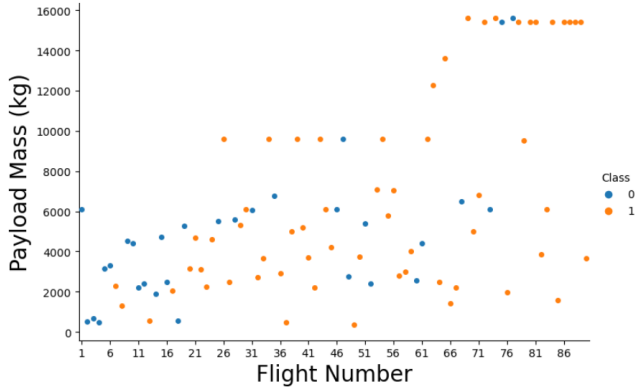
Assigned a new column (Class) into the existing pandas Data frame.

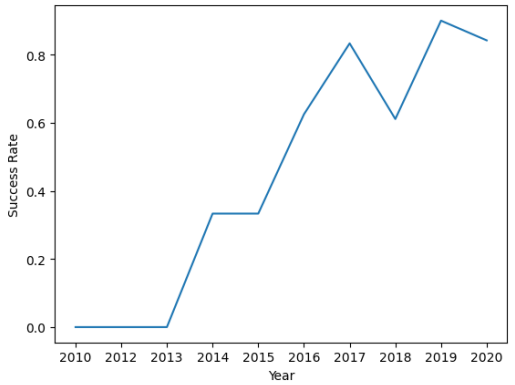
GitHub Link

```
# Review missing datapoints as a percentage
df.isnull().sum()/df.shape[0]*100

#Review the datatypes in the set
df.dtypes

# Review the  Launch Sites and number of launches at each
df['LaunchSite'].value_counts()

# Review the types and the occurance of Orbit
df['Orbit'].value_counts()



# Review the different landing types and their occurance
landing_outcomes = df['Outcome'].value_counts()

#Loop through the result and index the different outcomes
for i,outcome in enumerate(landing_outcomes.keys()):
    print(i,outcome)

# Generate a set of the "failiure landings" i.e. bad_outcome
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])



#Generate a list object to contain the Class of landing
landing_class = []

# Iterate through all landings and output [1,0] 1 = success, 0 = failiure
for value in df['Outcome']:
    if value in bad_outcomes:
        landing_class.append(0)
    else:
        landing_class.append(1)


#Join the landing_class list as a column in the dataframe
df['Class']=landing_class
```

EDA with Data Visualization







Bar Chart

Review relationship between orbits and respective success rate.

Categorical- Scatter plots

Review relationship between:

- Payload and flight number
- Launch site and flight number
- Launch site and Payload
- Orbit type and Flight Number
- Orbit type and Payload

Line plots

Reviewed the relationship (or trend) between how success rate have evolved over the years.

GitHub Link

EDA with SQL

Exploratory analysis performed with SQL

- Created Table SPACEXTABLE from the csv
- Reviewed distinct sites and sites containing “CCA” i.e., Cape Canaveral Space Station
- Reviewed total payload carried for NASA (CRS)
- Found the average Payload carried by Falcon 9
- Found First Successful landing
- Found all Booster versions successfully carrying between 4000-6000k kgs
- Summarized the mission outcomes and their count
- Listed the booster versions carrying max payload
- Listed the Month, Booster version and Launch site, for failed landings in 2015
- Listed the outcome and their count for launches between 2010 and 2017

GitHub Link

Build an Interactive Map with Folium

- Added circle points and markers for each launch site with corresponding name labels.
- Implemented a marker cluster for each launch with green icons for successful launches and red icons for failed launches.
- Integrated a mouse position feature to display real-time coordinates.
- Collected coordinates for the nearest;
  - Coastline
  - Trainline
  - Highway
  - City
- Illustrated lines indicating the distance to each of the points, accompanied by callouts with distance.

GitHub Link

Build a Dashboard with Plotly Dash

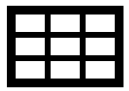
1) Slicers used dropdown(dcc.Dropdown() object) and Range selector (dcc.RangeSlider() object).
- Dropdown to be able to select specific site or all.
- Range selector where you can specify a range of payload mass.

2) Pie Chart (px.pie)
- While all is selected all sites along with their successive launches are displayed.
- While a single site is selected proportion of successful vs. failure are displayed.
- Highlights the sites with the most successful launches (all sites) and the proportions (specific site)

3) Scatterplot (px.scatter)
- Plot is filtered on both launch site dropdown and payload slider.
- Scatter displays Success rate (1,0) (y-axis), Payload Mass (x-axis) and points are colored based on the booster version.
- This will help to highlight the booster version and weight that correlate to a successful launch.

GitHub Link

Predictive Analysis (Classification)



Model Development

- Load Dataset
- Standardize the data using ***StandardScaler()*** and fit the features ***Train_test_split()*** the data into training and testing sets for X,Y
- Perform a ***Gridsearch()*** to find the optimal parameters for the model.

Model Evaluation

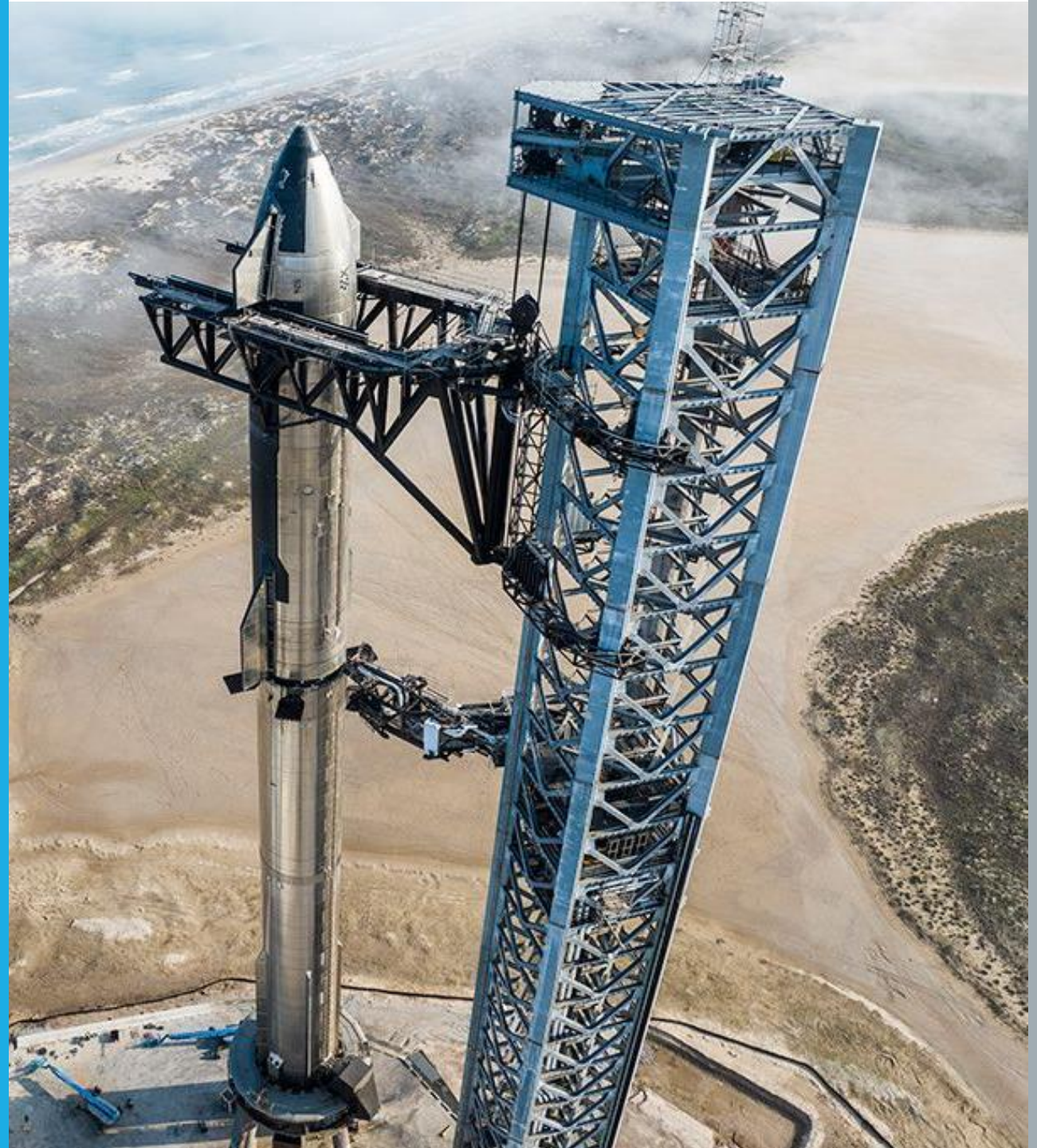
- Use the optimal parameters and using evaluating the model with .score()
- Calculate the predicted $\hat{Y}$ and generate a confusion matrix to display the predicted vs observed results

Finding the best classification model

- Compared the outcome of all the models based on their out of sample accuracy

GitHub Link

Results



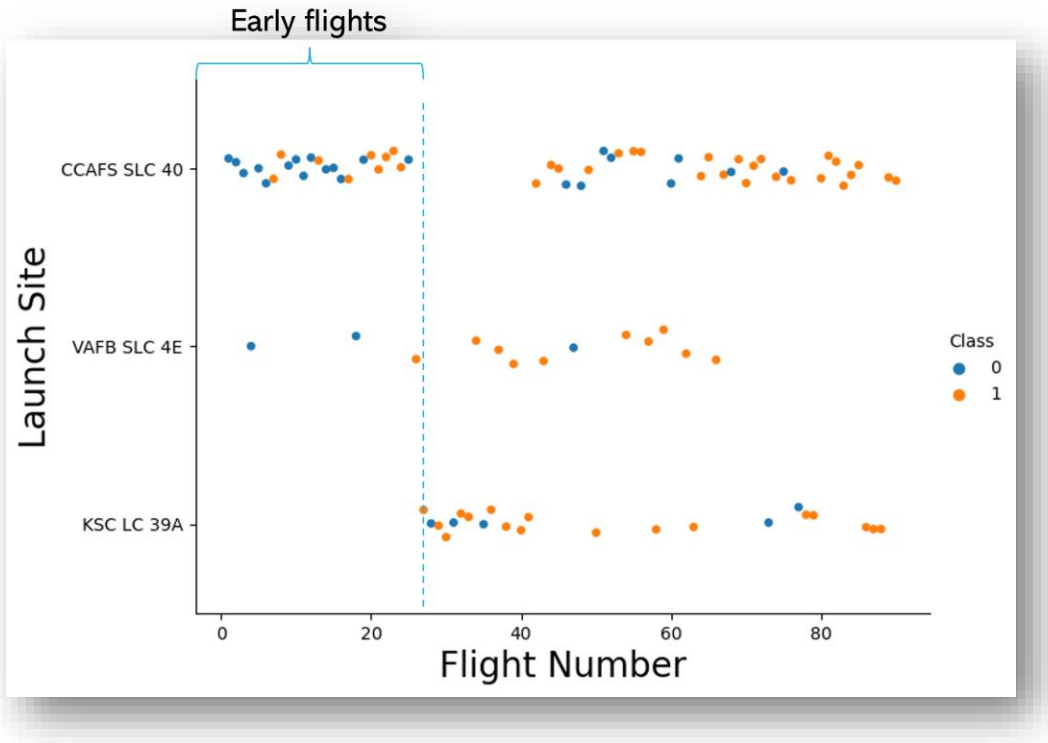
Exploratory Data Analysis

Flight Number vs. Launch Site

Observations:

- We can see that for the first ~25 launches sites was completed at CCAFS and VAFB.
- First 6 launches resulted in class 0 i.e., failure.
- Prior to launch ~25 KSC LC launch site were not used.
- Post launch 25 the prevalence of failure fall drastically.

In summary we can state that there is a high correlation between failure and low flight numbers. There is a higher correlation between launch site VAFB and CCAFS and failure compared to KSC LC

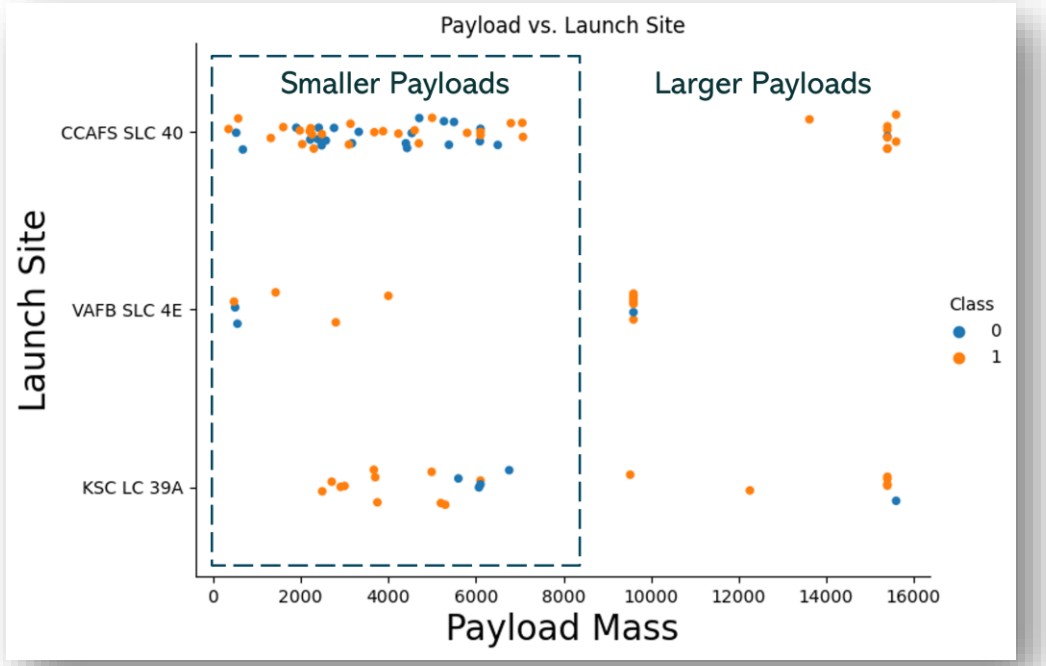


Payload vs. Launch Site

Observations:

- From the scatterplot we can observe at least two areas. Smaller payloads (dotted area) and Large Payloads.
- We can see that CCAFS has the greatest number of launches of “smaller payloads”. CCAFS also has the greatest range of payloads while VAFB and KSC have smaller [~0 - ~4000] and larger [~2000 – 7000] accordingly.
- In the larger payload range, we can observe that KSC has the biggest range while VAFB maxes out around 10000kg and CCAFS only have payloads in the range 13000 – 16000.

We can also observe that most of the failures occur in the “Smaller payload” range. Hinting at a higher correlation between failure and smaller payloads.

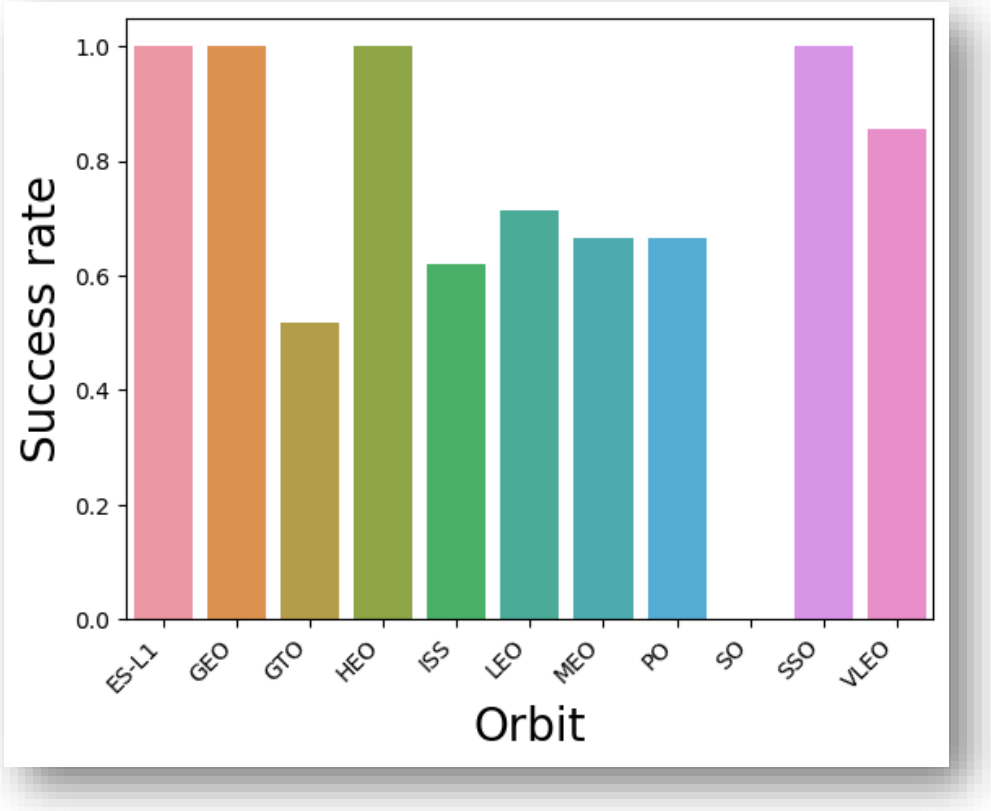


Success Rate vs. Orbit Type

Observations:

- We can instantly see that the following orbits have a 100% success rate:
  - ES-L1
  - GEO
  - HEO
  - SSO
- Further we can see VLEO has close to 90% success rate.
- GTO, OSS, LEO, MEO and PO has success rates between 50-70%.
- Lastly, we can see that SO has a zero-success rate.

From this we can extrapolate that Orbit ES-L1, GEO, HEO and SSO and VLEO has a very high correlation with success. While the remaining orbits while high has lower correlation with success and SO has had no successful landings.

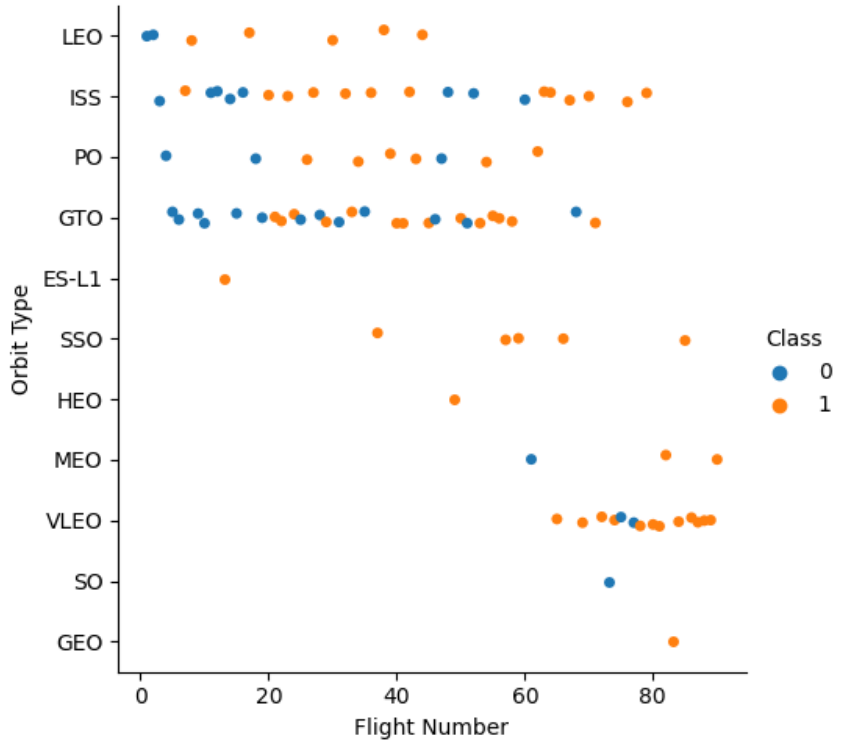


Flight Number vs. Orbit Type

Observations:

- LEO, ISS, PO, and GTO have the most failures and the widest range of launches by flight number.
- Orbits with 100% success, except for SSO (5 times), have mostly been attempted once.
- SSO, HEO, MEO, VLEO, SO, and GEO were first launched around the 40th flight.
- We can also observe that the orbit that in the last slide exhibited a zero-success rate (SO) have only been attempted once.

From this we can surmise that comparatively LEO, ISS, PO and GTO will exhibit a higher correlation for failure. We can also see that SO, ES-L1, GEO, and HEO will have extreme effects as these only have one observation. Hence, their probability will be either 0% or 100% failure. Considering removing these single-attempt data points might reduce data noise.Finally, we can also observe a trend of success increases as flight number increases which inutility would map on given no planned failures.

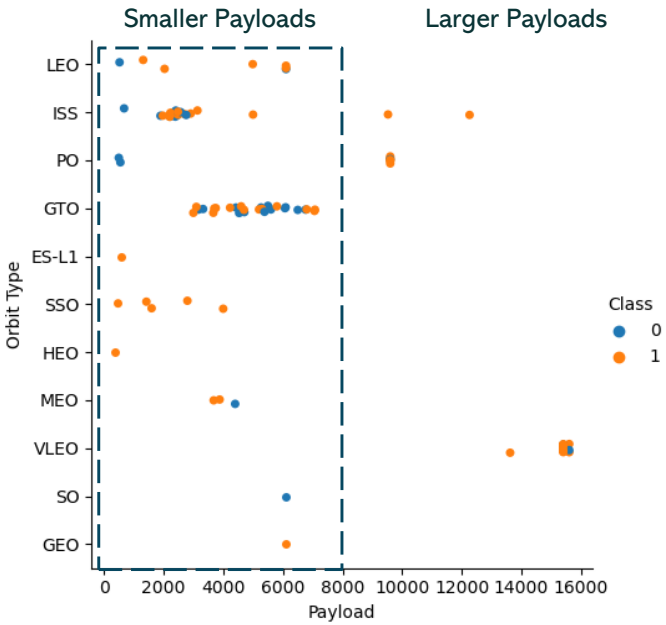


Payload vs. Orbit Type

Observations:

- Utilizing the same categorizations as Payload vs. Launch Site plot we can see that only three orbits have payloads in the “larger Payload category.
- ISS and PO occur in both ranges and both display failures but mostly with smaller payloads
- From the plot it seems like there is strong correlation between larger payloads and successful landings. Based on the plot it seems like VLEO and PO are the only larger payload with failures, and this seems to only be one.

In summary it seems that smaller payload has lower correlation with successful outcome. Except in the case of SSO if we disregard the single observation orbits. Further it seems like larger payloads have a much stronger correlation with successful outcome.

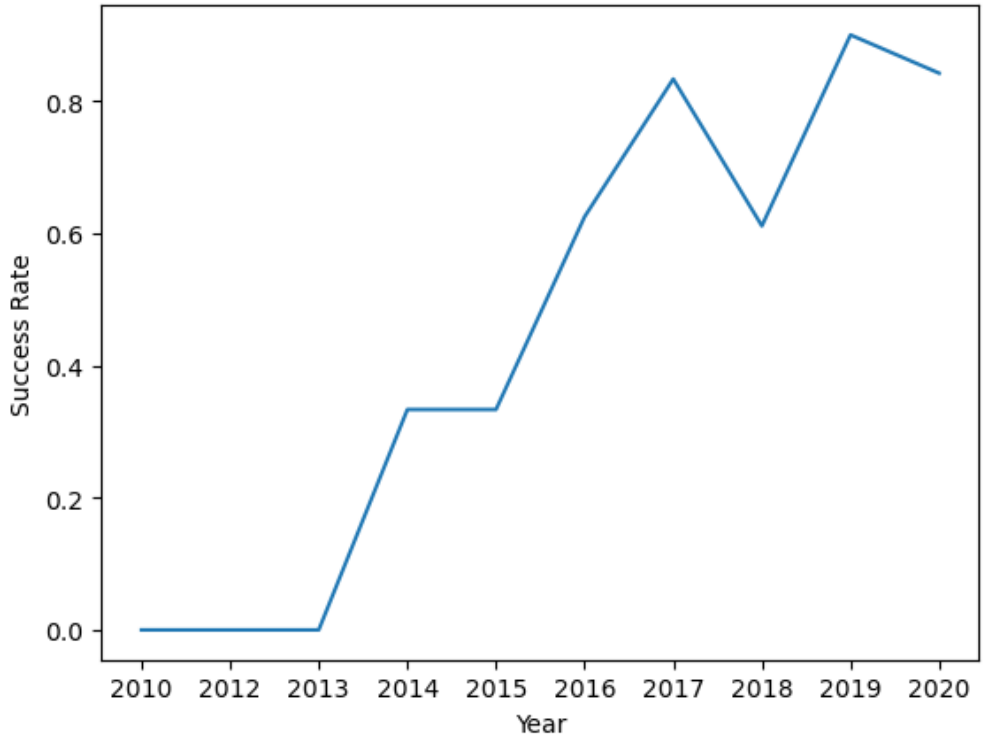


Launch Success Yearly Trend

Observations:

- We can observe that for the first three years there were no successful landings.
- From 2013 to 2019 there has been a steady increase except between 2014-2015 where the success rate remained the same and in 2018 taking a sharp dip.
- We can see that 2019 was the year with the highest success rate followed be a slight decrease in 2020.

In general, we could say that there is a strong correlation between Year and success rate. With a few exceptions we say that as Year increases so does Success rate.



Launch Site info

From the query we can see that there are four different launch sites.

We can also see that Cape Canaveral space station has two launch sites namely CCAFS LC-40 and CCAFS SLC-40.

The other locations Vandenberg and Kennedy Space Center only have one site namely VAFB SLC-4E and KSC LC-39A

Query

```
%sql \
SELECT \
DISTINCT(Launch_Site)\
FROM SPACEXTABLE
```

Result

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

Launch Site Names Begin with 'CCA'

- The below code returns all column (Select *) for the first 5 rows (limit 5) given the filter condition that Launch_Site starts with “CCA” (“CCA%”).

Query

```
%sql Select * FROM SPACEXTABLE where Launch_Site like "CCA%" limit 5
```

Result

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2010-04-06	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-08-12	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
2012-08-10	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-01-03	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Total Payload Mass

- In the query we use the sum() function this will sum the values in the column. We also use an alias for the output i.e., “Total Payload”.
- In the WHERE statement we specify that the customer should be “NASA (CRS)”.
- Total Payload carried for NASA(CRS) is 45.596 kg

Query

```
%sql\
SELECT\
sum(PAYLOAD_MASS__KG_) as 'Total Payload'\
FROM SPACEXTABLE \
WHERE Customer = 'NASA (CRS)'
```

Result

Total Payload
45596

Average Payload Mass by F9 v1.1

- In the query we use the AVG() function for payload mass. This will calculate the average value of payloads.
- In the WHERE statement we specify that Booster Version is like 'F9 v1.1%'.
  This will return any rows where Booster version starts with F9 v1.1 i.e., F9 v1.1B….. Etc.
- For all the F9 v1.1 launches the average was 2534.67 kg

Query

```
%sql\
SELECT\
AVG(PAYLOAD_MASS__KG_) as 'Average Payload'\
FROM SPACEXTABLE\
WHERE Booster_Version like 'F9 v1.1%'
```

Result

Average Payload
2534.6666666666665

First Successful Ground Landing Date

Using the min(Date) function we can locate the oldest date. Combining this with the WHERE clause of Landing outcome equals ‘Success (ground pad) will yield the first achieved landing

Query

```
%sql SELECT \
min(a.Date) as 'First Achived Landing'\
FROM SPACEXTABLE a\
WHERE Landing_Outcome = 'Success (ground pad)'
```

Result

First Achived Landing
2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

- List the names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000

Query

```
%sql \
SELECT Booster_Version \
FROM SPACEXTABLE \
WHERE Mission_Outcome = 'Success'\
AND Landing_Outcome like "%drone%"\
AND PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000
```

Result

Booster_Version
F9 FT B1020
F9 FT B1022
F9 FT B1026
F9 FT B1021.2
F9 FT B1031.2

Total Number of Successful and Failure Mission Outcomes

- In this Query we select the Mission Outcome From the SPACEXTABLE.
- With the Count(Mission_Outcome) combined with GROUP BY we can summarize the number of occurrences Each “type of success/Failure” occurred.
- Note that this relates to the mission not the first stage landing or not. There are only 1 failure. However, there are a total of 100 Success observations spread in 3 categories.
- The difference between the first two success in the table is that one contains a space (‘Success ‘) after. SQL will therefor treat these as different from each other.

Query

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

```
%sql SELECT Mission_Outcome FROM SPACEXTABLE WHERE Mission_Outcome = 'Success '
```

Result

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

Mission_Outcome
Success

Boosters Carried Maximum Payload

- **In the query we select the Booster_Version. In the WHERE statement we filter PAYLOAD_MASS_KG_ equal to the return value of the subquery.**
- **In the subquery we use the MAX() function this will only return a single value. Hence the WHERE function can use only "=" rather than a "IN"**

Query

```
sub ="""
SELECT Booster_Version
FROM SPACEXTABLE
WHERE PAYLOAD_MASS__KG_= (
    SELECT MAX(PAYLOAD_MASS__KG_)
    FROM SPACEXTABLE)
ORDER BY Booster_Version
"""
%sql $sub
```

Result

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

2015 Launch Records

- In this query I have utilized the CASE and SUBSTR() function.
- The SUBSTR() (Substring) function extracts a string from a value based on the index position within a cell. So, this will extract from index 6 and output two numbers.
- The CASE function works like an if/switch function so if the result of the substring function equals '01' then the function will output January etc. The function ends with an ELSE which if no other values match, then the Output will be December (This can be problematic if there were a Null value as the way it's been written here it would output as December.

Query

```
%sql SELECT \
CASE substr(Date, 6, 2)\
    WHEN '01' THEN "January"\
    WHEN '02' THEN "February"\
    WHEN '03' THEN "March"\
    WHEN '04' THEN "April"\
    WHEN '05' THEN "May"\
    WHEN '06' THEN "June"\
    WHEN '07' THEN "July"\
    WHen '08' THEN "August"\
    WHEN '09' THEN "September"\
    WHEN '10' THEN "October"\
    WHEN '11' THEN "November"\
    ELSE "December" END as Month,\
Booster_Version,\
Launch_Site,\
Landing_Outcome\
FROM SPACEXTABLE\
WHERE Landing_Outcome = 'Failure (drone ship)'\
    AND substr(Date,1,4) = '2015'
```

Result

Month	Booster_Version	Launch_Site	Landing_Outcome
October	F9 v1.1 B1012	CCAFS LC-40	Failure (drone ship)
April	F9 v1.1 B1015	CCAFS LC-40	Failure (drone ship)

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

- In the query we return the Landing Outcome and the amount (count of occurrence) of each outcome.
- In the WHERE we use the operator <= and >= to get dates in the period 2010-06-04 and 2017-03-20. We could also use Between instead (Date BETWEEN '2010-06-04' AND '2017-03-20') as both would return the same.
- We then use the GROUP BY to only get one occurrence of each value in Landing_Outcome
- Finally, we use the ORDER BY with count() to rank the count in Descending (DESC) order.

Query

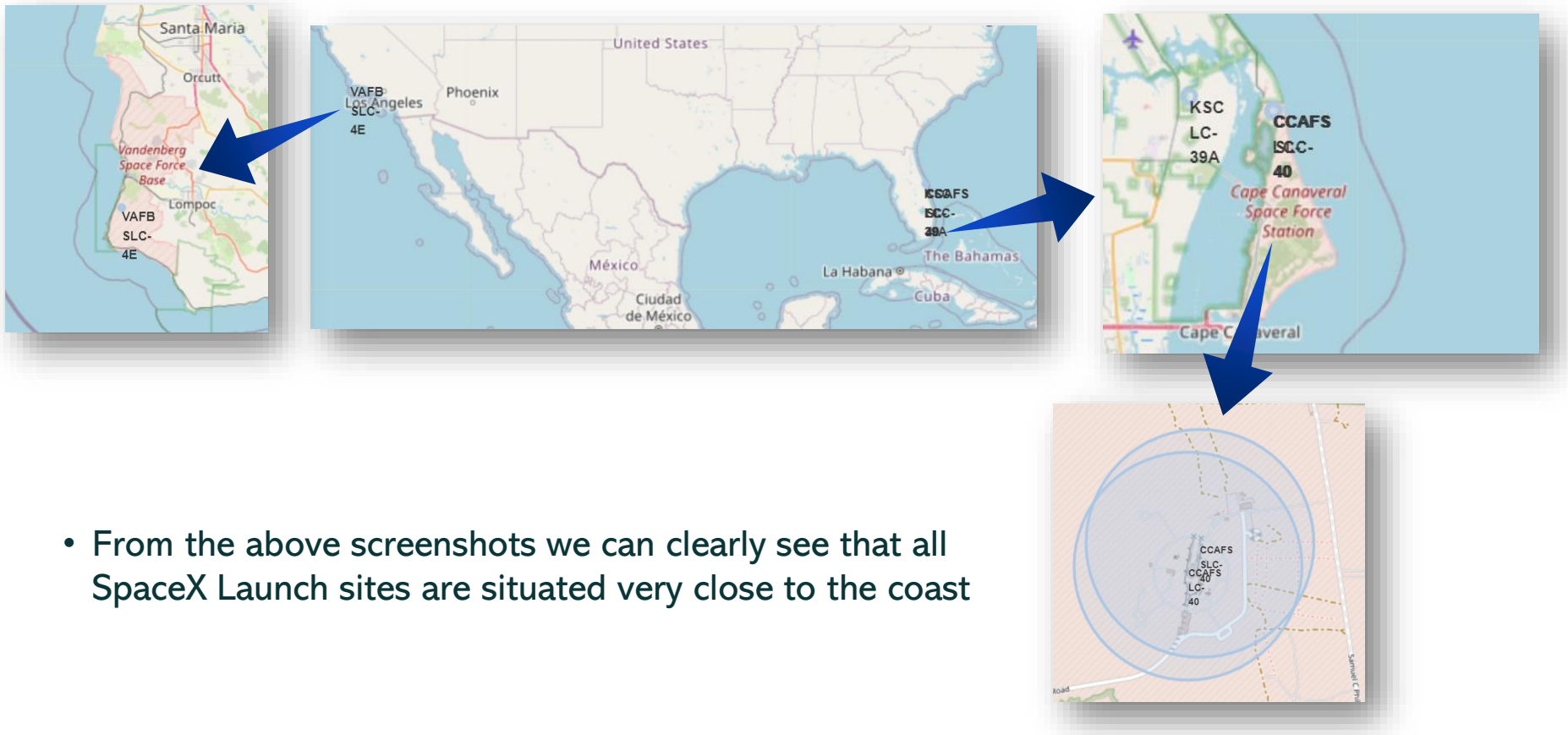
```
rank ="""
SELECT Landing_Outcome,
count(Landing_Outcome) as 'AMOUNT'
FROM SPACEXTABLE
WHERE Date >= '2010-06-04'
    AND Date <= '2017-03-20'
GROUP BY Landing_Outcome
ORDER BY count(Landing_Outcome) DESC
"""
%sql $rank
```

Result

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

Launch Site Proximity Analysis

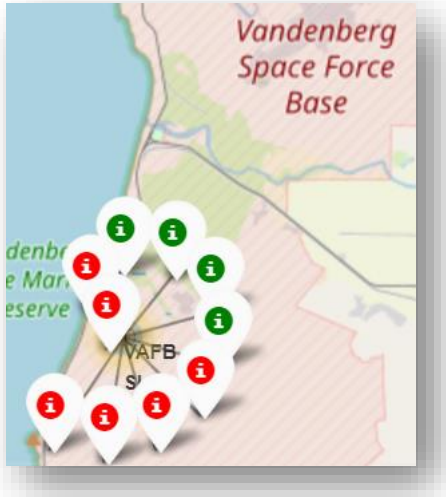
SpaceX Launch Sites

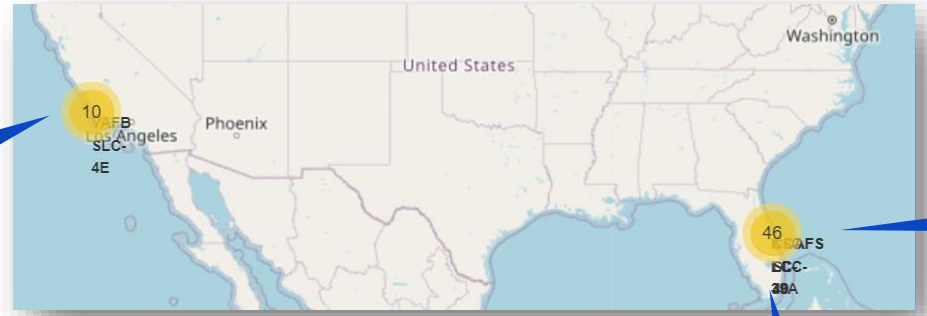


- From the above screenshots we can clearly see that all SpaceX Launch sites are situated very close to the coast

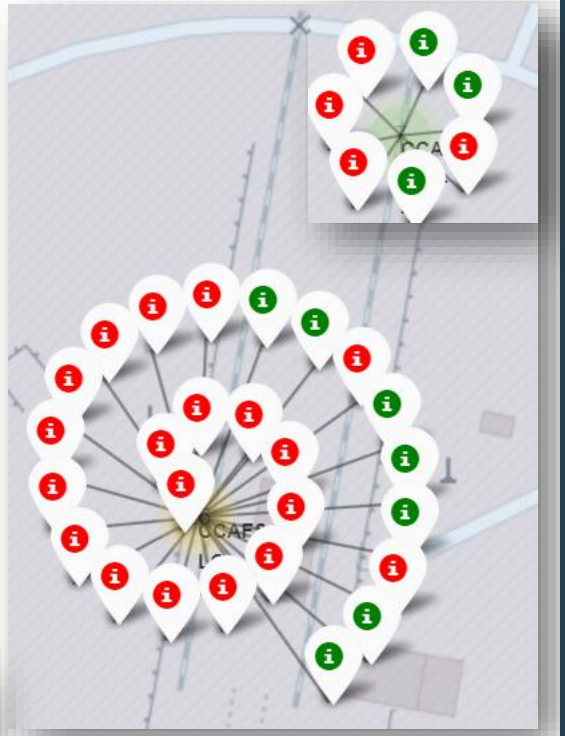
SpaceX successful and unsuccessful launches

VAFB SLC-4E

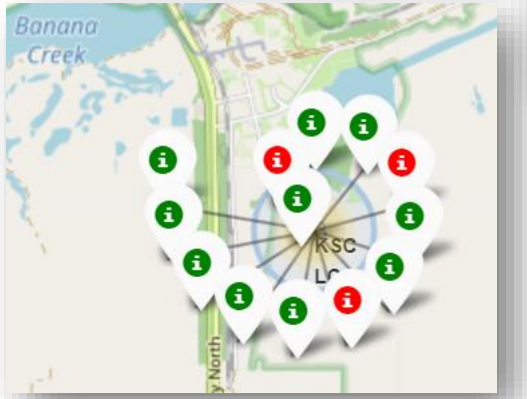




CCAFS SLC-40 & CCAFS LC-40

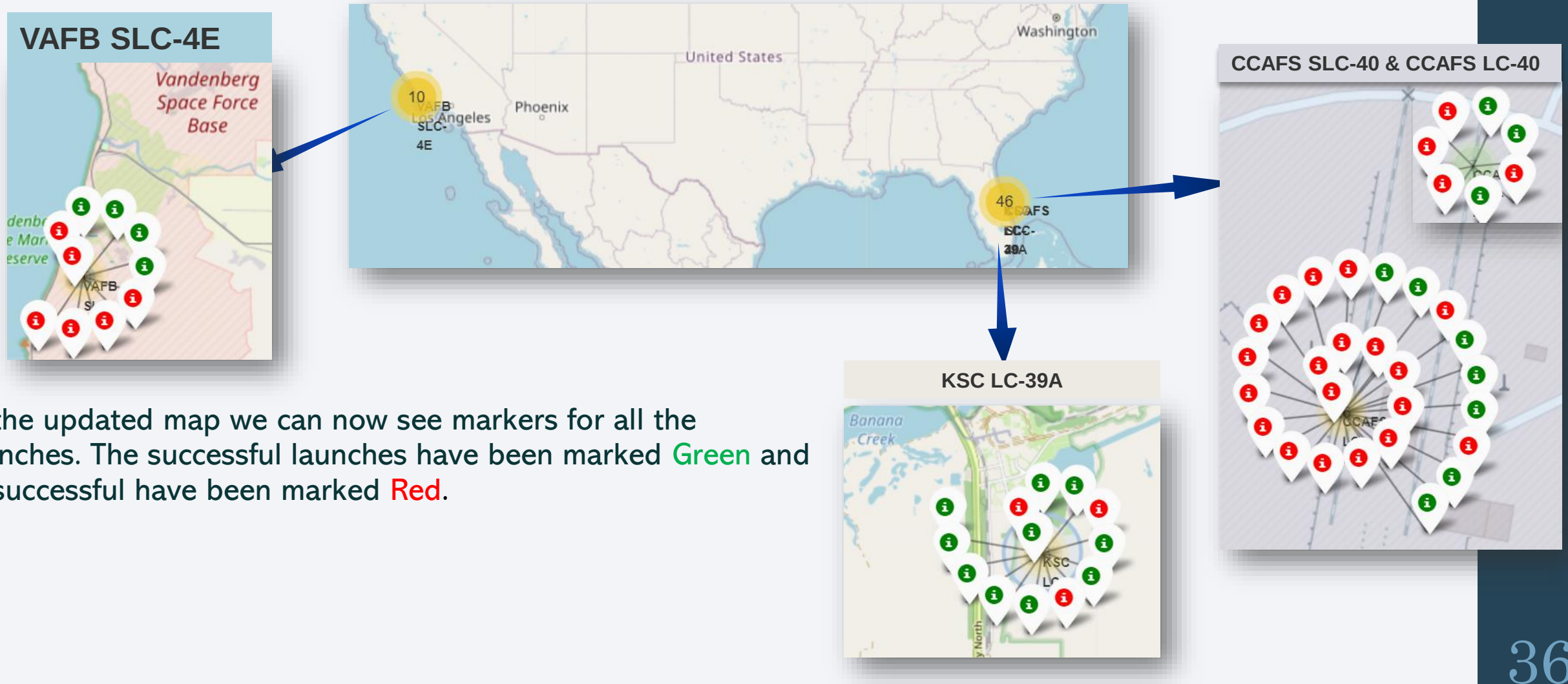


KSC LC-39A



- On the updated map we can now see markers for all the launches. The successful launches have been marked Green and unsuccessful have been marked Red.
- We can see that proportionally Kennedy Space Center has highest Success rate

SpaceX successful and unsuccessful launches

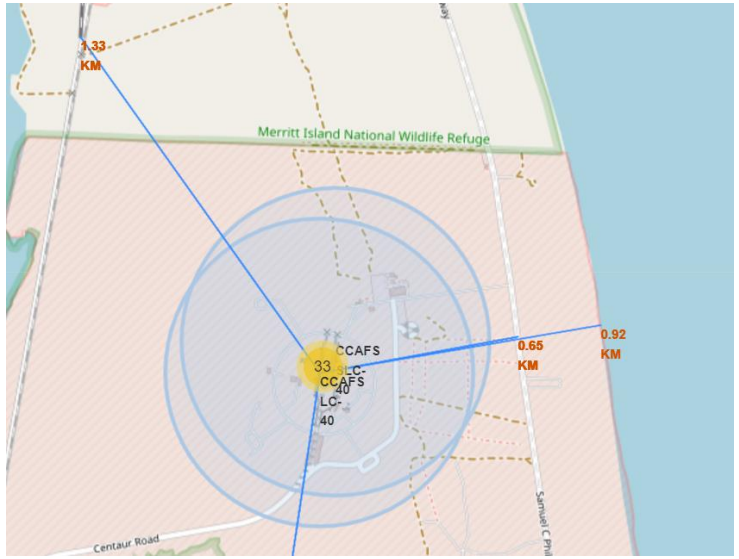


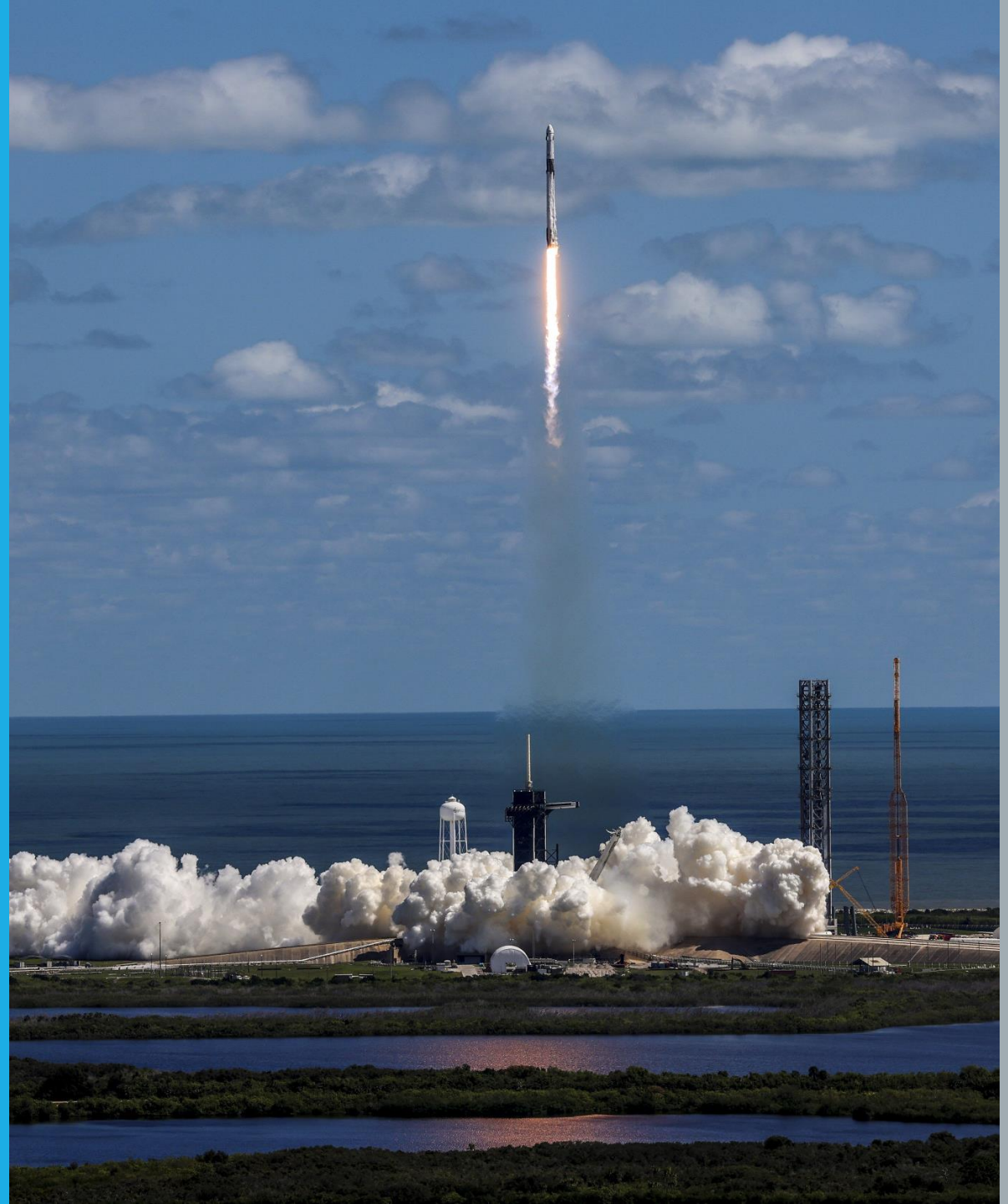
In the updated map we can now see markers for all the launches. The successful launches have been marked Green and unsuccessful have been marked Red.

Infrastructure proximity to Launch Site

Infrastructure proximity to Cape Canaveral Space Station (CCAFS LC-40)

- Are launch sites in close proximity to railways?
  - Yes, there is a railway approx. 1.3 km away from the launch site
- Are launch sites in close proximity to highways?
  - Samuel C Phillips Parkway is located 650 meters away from the launch site
- Are launch sites in close proximity to coastline?
  - The coastline is approx. 900 meters away from the launch site
- Do launch sites keep certain distance away from cities?
  - The closest city to the launch site is Cape Canaveral

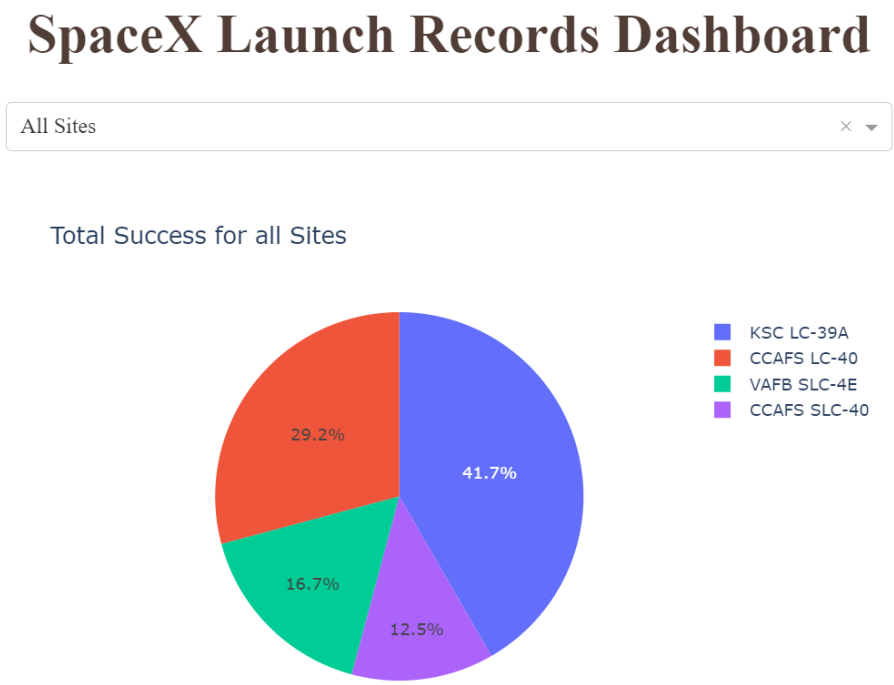




Section 4 Build a Dashboard with Plotly Dash

Total Success for all SpaceX sites

From the Pie chart we can see that Kennedy Space Centre (KSC LC-40) has the greatest number of successful landings, followed by CCAFS LC-40, VAFB SLC-4E and CCAFS SLC-40.

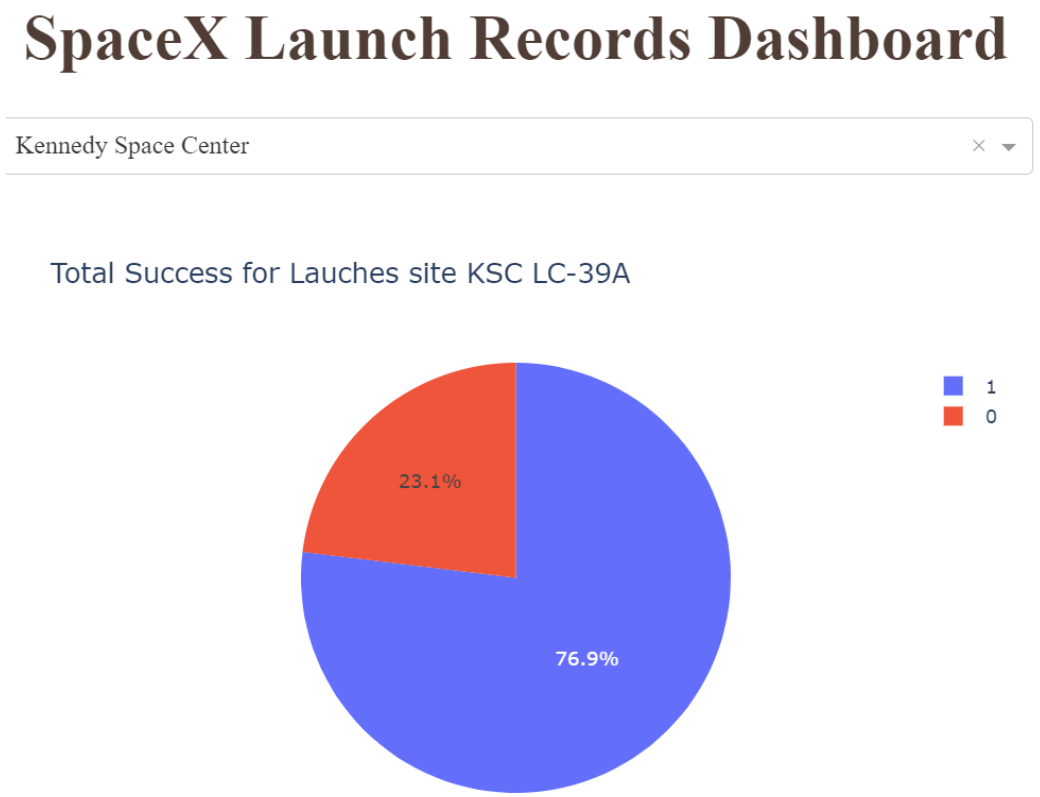


Kennedy Space Centre

From the chart we can see that Kennedy space centre is the launch site with the greatest ratio as well.

Notably we can see that CCAFS LC-40 comes in a close second and it's the site with the most launches and their success Rate is quite close to KSC LC-39A.

Site	Observations	Success	Failure	Success Rate
KSC LC-39A	13	10	3	77%
CCAFS LC-40	26	19	7	73%
VAFB SLC-4E	10	6	4	60%
CCAFS SLC-40	7	4	3	57%



Payloads effect on success rate

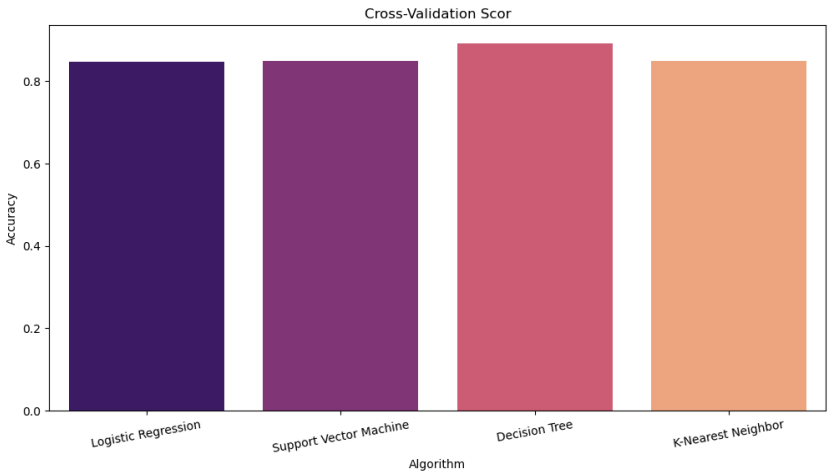
The plots to the right display the graphs for the payload weight :
0 - 5000 kg (upper).
5000 – 10 000 (lower).

- From the graph we can see that v1.0, v1.1, B5 has only carried payloads in the lower range.
- The FT and B4 occur in both tables with the min values among the smallest payloads and the largest.
- FT seems to be one of the most successful in the lower range specifically in the 2-5k range while in the upper range it mostly result in failure.
- B4 has the highest payload that landed successfully. Although the outcome in the upper bound seems relatively unsuccessful (4 unsuccessful and 1 successful outcomes).
- We can see that v.1.0 and v1.1 based on the graph don't have any successful landings and B5 seem to only have a single observation which is successful

Based on the above we can extrapolate that a for v.1.0 and v.1.1 will most likely result in a failure. Excluding these we can see that a lower payload will have a higher successrate.

Predictive Analysis (Classification)

Classification Accuracy



The bar plot displays the Cross-Validation-Score for all the models.

The Cross-Validation-Score shows us that all the models performed quite well however, Decision tree has performed best.

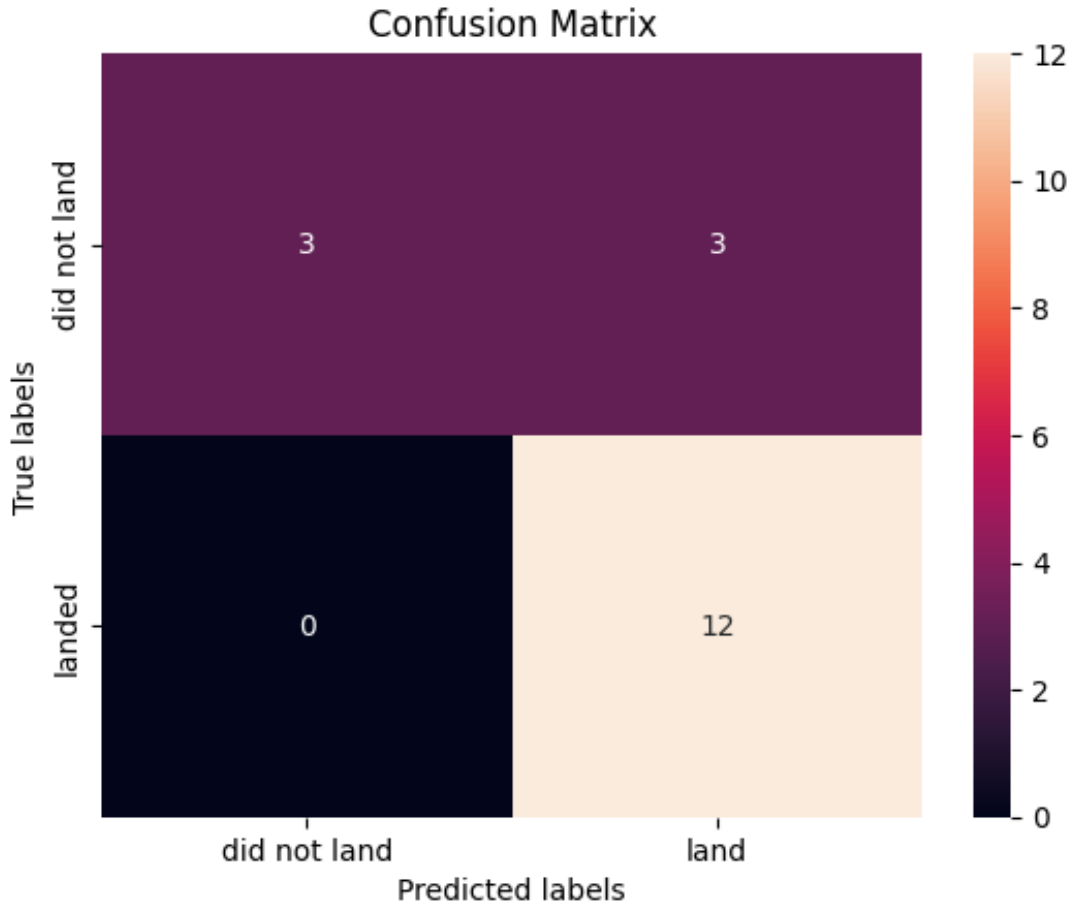
Interestingly we can see from the table that all the models generalized the unseen data equally.

From running the decision tree model multiple times there were occasions where it performed both better and worse with Out-of sample accuracies ranging from around 60 to 90%

Algorithm	Cross-Validation Score	Out-of Sample Score
Logistic Regression	0.846429	0.833334
Support Vector Machine	0.848214	0.833334
Decision Tree	0.891071	0.833334
K-nearest Neighbor	0.848214	0.833334

Confusion Matrix

In the Confusion matrix we can see that the model is good at predicting the true label (predicted landed when true label was land). However, there is an issue with false positives (upper right) where for 3 observations the model predicted that the stage 1 landed when it in fact didn't. The model also predicted 3 true false i.e., predicted that the stage 1 didn't land when true label was "did not land".



Conclusion

Conclusions

Orbits:

- *Earth Sun Lagrange Point 1* (ES-L1), *Geostationary* (GEO), *Highly Elliptical Orbit* (HEO) & *Sun-Synchronous Orbit* (SSO). Have the highest success rate although SSO is the only one of these that has multiple launches so the others can be viewed as noise. International Space Station (ISS) and Very Low Earth Orbit also show high success rate.

Performance

- There is a clear increase in success rate over time.

Location:

- All sites have a close proximity to the coast.
- Kennedy Space Center (KSC LC-39A) has the highest success rate of all locations. Although launches did not occur here until around the 25th launch.
- It is worth noting that Cape Canaveral Space Station (CCAFS – LC40 & SLC-40) has by far the largest number of launches and success rate for LC-40 is only 4 percentage points below that of KSC.

Payload:

- Cape Canaveral and Kennedy Space station has lauches with payloads in the entire weight range, while Vanderberg Space Force Base has launches where payload maxes out around 10.000 kg.

Booster Version

- v.1.0 and v.1.1show the lowest success rate
- B4 has the heaviest payload with successful landing.
- FT is the most successful in the lower weight range

Conclusions

Considerations

- The Dataset is limited and with ex. some orbits having only a single launch orbits weight on the model can be overstated if these orbits are even in the training set.
- Since we would want to predict future launches keeping the early-stage launches could potentially lower predicted success chance. There is a clear trend that success has risen with time. Although as mentioned that dataset is quite limited.


Appendix

Appendix - API

- Functions used for extracting data from the API
- First function extracts the launch data.
- Remaining extracts the details.

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

response = requests.get(spacex_url)
```

```
# 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)).json()
         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'])
```

Appendix - Web Scraping

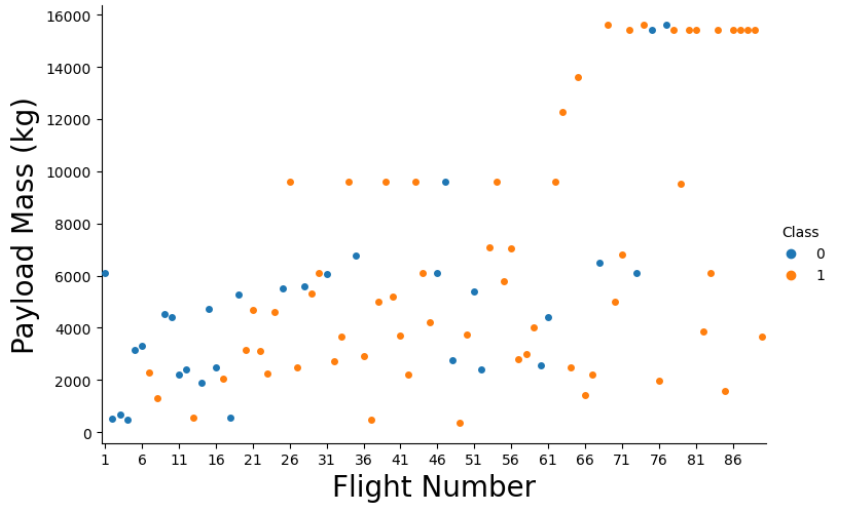
- Functions to extract various launch data from the html code

```
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()

    colunm_name = ' '.join(row.contents)

    # Filter the digit and empty names
    if not(colunm_name.strip().isdigit()):
        colunm_name = colunm_name.strip()
        return colunm_name
```

Appendix – Visual EDA

- Graph plot to show flight number and Payload mass

- Review of the distinct values for Orbit, Launch Sites, Landing Pas and Serial.




```
columns_to_check = ['Orbit', 'LaunchSite', 'LandingPad', 'Serial']
unique_values = features[columns_to_check].nunique()
print(unique_values)
```

```
Orbit         11
LaunchSite     3
LandingPad     5
Serial        53
dtype: int64
```

Appendix - Web Scraping II

- Code that uses the function from last slide and appends it into the dictionary shown below.

```
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'] = []

launch_dict['Version Booster']=[]
launch_dict['Booster landing']=[]
launch_dict['Date']=[]
launch_dict['Time']=[]
```

```
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
            launch_dict['Flight No.'].append(flight_number)
            # TODO: Append the flight_number into launch_dict with key `Flight No.`
            datatimelist=date_time(row[0])

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

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

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

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

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

            # Customer
            # TODO: Append the customer into launch_dict with key `Customer`

            if row[6].a:
                customer = row[6].a.string
                launch_dict['Customer'].append(customer)
            else:
                launch_dict['Customer'].append("missing customer")

            # Launch outcome
            # TODO: 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
            # TODO: Append the launch_outcome into launch_dict with key `Booster Landing`
            booster_landing = landing_status(row[8])
            launch_dict['Booster landing'].append(booster_landing)
```

Appendix - Folium

- Code to generate the map and to populate the markers and circles for the sites.
- Adding Clusters for each site/ launch
- Function to add lines with distances to each site of interest.

```
# Initial the map
site_map = folium.Map(location=nasa_coordinate, zoom_start=5)
# For each launch site, add a Circle object based on its coordinate (Lat, Long) values.
# In addition, add Launch site name as a popup label
labels = folium.map.FeatureGroup(name = 'Labels')

for lat, long, name in zip(launch_sites_df.Lat, launch_sites_df.Long, launch_sites_df['Launch Site']):
    labels.add_child(
        folium.Circle(
            [lat,long],
            radius = 1000,
            color = '#000000',
            fill = True
        ).add_child(folium.Popup(name))
    ),
    labels.add_child(
        folium.map.Marker(
            [lat,long],
            icon = DivIcon(icon_size=(20,20),
                           icon_anchor = (0,0),
                           html='<div style = "font-size: 12; color:#d35400;"><b>%s</b></div>' % name,
                          )
        )
    )

site_map.add_child(labels)
```

```
def add_marker_and_line(map_obj, point_name, point_lat, point_lon, launch_lat, launch_lon):
    # Calculate distance
    distance = calculate_distance(launch_lat, launch_lon, point_lat, point_lon)

    # Create marker
    marker = folium.Marker(
        [point_lat, point_lon],
        icon=folium.DivIcon(
            icon_size=(20,20),
            icon_anchor=(0,0),
            html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:10.2f} KM".format(distance)
        )
    ).add_to(map_obj)

    # Draw line between the point and launch site
    line_coords = [(point_lat, point_lon), (launch_lat, launch_lon)]
    folium.PolyLine(locations=line_coords, weight=1).add_to(map_obj)

# Coordinates

Railway_lat, Railway_lon = 28.57207, -80.58528
Road_lat, Road_lon = 28.56328, -80.58695
City_lat, city_lon = 28.40138, -80.60462

# Add markers and lines for each point
add_marker_and_line(site_map, 'Railway', Railway_lat, Railway_lon, launch_site_lat, launch_site_lon)
add_marker_and_line(site_map, 'Road', Road_lat, Road_lon, launch_site_lat, launch_site_lon)
add_marker_and_line(site_map, 'City', City_lat, city_lon, launch_site_lat, launch_site_lon)
```

```
# Add marker_cluster to current site_map
site_map.add_child(marker_cluster)

# for each row in spacex_df data frame
# create a Marker object with its coordinate
# and customize the Marker's icon property to indicate if this launch was successed or failed,
# e.g., icon=folium.Icon(color='white', icon_color=row['marker_color']
for index, record in spacex_df.iterrows():
    # TODO: Create and add a Marker cluster to the site map
    # marker = folium.Marker(...)
    marker = folium.map.Marker(
        location = [record['Lat'], record['Long']],
        icon=folium.Icon(color='white', icon_color=record['marker_color'])
    )
    marker_cluster.add_child(marker)

site_map
```

Appendix – Dash (Slicers)

- Slices generated with actual names for the Space stations instead of codes.
- Range slider with ticks generated to easier see what range has been selected in increment of thousands.

```
dcc.Dropdown(id='site-dropdown',
    options=[
        {'label': 'All Sites', 'value': 'ALL'},
        {'label': 'Cape Canaveral LC-40', 'value': 'CCAFS LC-40'},
        {'label': 'Cape Canaveral SLC-40', 'value': 'CCAFS SLC-40'},
        {'label': 'Kennedy Space Center', 'value': 'KSC LC-39A'},
        {'label': 'Vandenberg AFB', 'value': 'VAFB SLC-4E'}
    ],
    value='ALL',
    placeholder="All Sites",
    searchable=True
    ),

html.Br(),
```

```
dcc.RangeSlider(id='payload-slider',
                min=0, max=10000, step=1000,
                marks={0: '0', 1000: '1000',2000: '2000',3000: '3000',4000: '4000',5000: '5000',6000: '6000',7000: '7000',8000: '8000',9000: '9000',10000: '1000
                 value=[min_payload, max_payload]),
```

Appendix – Dash (Graphs)

- Function to generate the graphs with logic functions to slice the data according to filters set.

```python
@app.callback(Output(component_id='success-pie-chart', component_property='figure'),
              Input(component_id='site-dropdown', component_property='value'))
def get_pie_chart(entered_site):
    filtered_df = spacex_df
# If all is selected show success based on location
    if entered_site == 'ALL':
        filtered_df = filtered_df[filtered_df['class']==1]

        fig = px.pie(filtered_df, values='class',
                     names = 'Launch Site',
                     title = "Total Success for all Sites")

# If specific site is selected then show pie as a percentage success and percentage failiure
    else:
        filtered_df=filtered_df[filtered_df['Launch Site']==entered_site]
        filtered_df = filtered_df['class'].value_counts().reset_index()
        filtered_df.columns = ['Site','count']

        fig = px.pie(filtered_df,
                     values='count',
                     names = 'Site',
                     labels = {0: 'Failiure', 1: 'Success'},
                     color_discrete_map={0: 'red', 1: 'green'},
                     title = f"Total Success for Lauches site {entered_site}")

    return fig
```

```python
@app.callback(Output(component_id = 'success-payload-scatter-chart', component_property = 'figure'),
              Input(component_id='site-dropdown',                    component_property='value'),
              Input(component_id='payload-slider',                   component_property='value'))

def get_scatterplot(entered_site, entered_range):
    filtered_df = spacex_df
    if entered_site != 'ALL':
        filtered_df = filtered_df[filtered_df['Launch Site']==entered_site]

    min_value, max_value = entered_range
    filtered_df = filtered_df[(filtered_df['Payload Mass (kg)']>= min_value) & (filtered_df['Payload Mass (kg)']<= max_value)]

    fig = px.scatter(filtered_df, x = 'Payload Mass (kg)', y = 'class', color = 'Booster Version Category' )
    fig.update_layout(
        title = 'Success by Payload and Booster Version',
        xaxis_title = 'Payload Mass (kgs)',
        yaxis_title = 'Result'
    )

    return fig
```

Appendix – Machine Learning I

- Function to create Confusion Matrices

```
def plot_confusion_matrix(y,y_predict):
    "this function plots the confusion matrix"
    from sklearn.metrics import confusion_matrix

    cm = confusion_matrix(y, y_predict)
    ax= plt.subplot()
    sns.heatmap(cm, annot=True, ax = ax); #annot=True to annotate cells
    ax.set_xlabel('Predicted labels')
    ax.set_ylabel('True labels')
    ax.set_title('Confusion Matrix');
    ax.xaxis.set_ticklabels(['did not land', 'land']); ax.yaxis.set_ticklabels(['did not land', 'landed'])
    plt.show()
```

- Tuned Hyperparameter of each model.

KNN

```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}
accuracy : 0.8482142857142858
```

Logistic regression

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

SVM

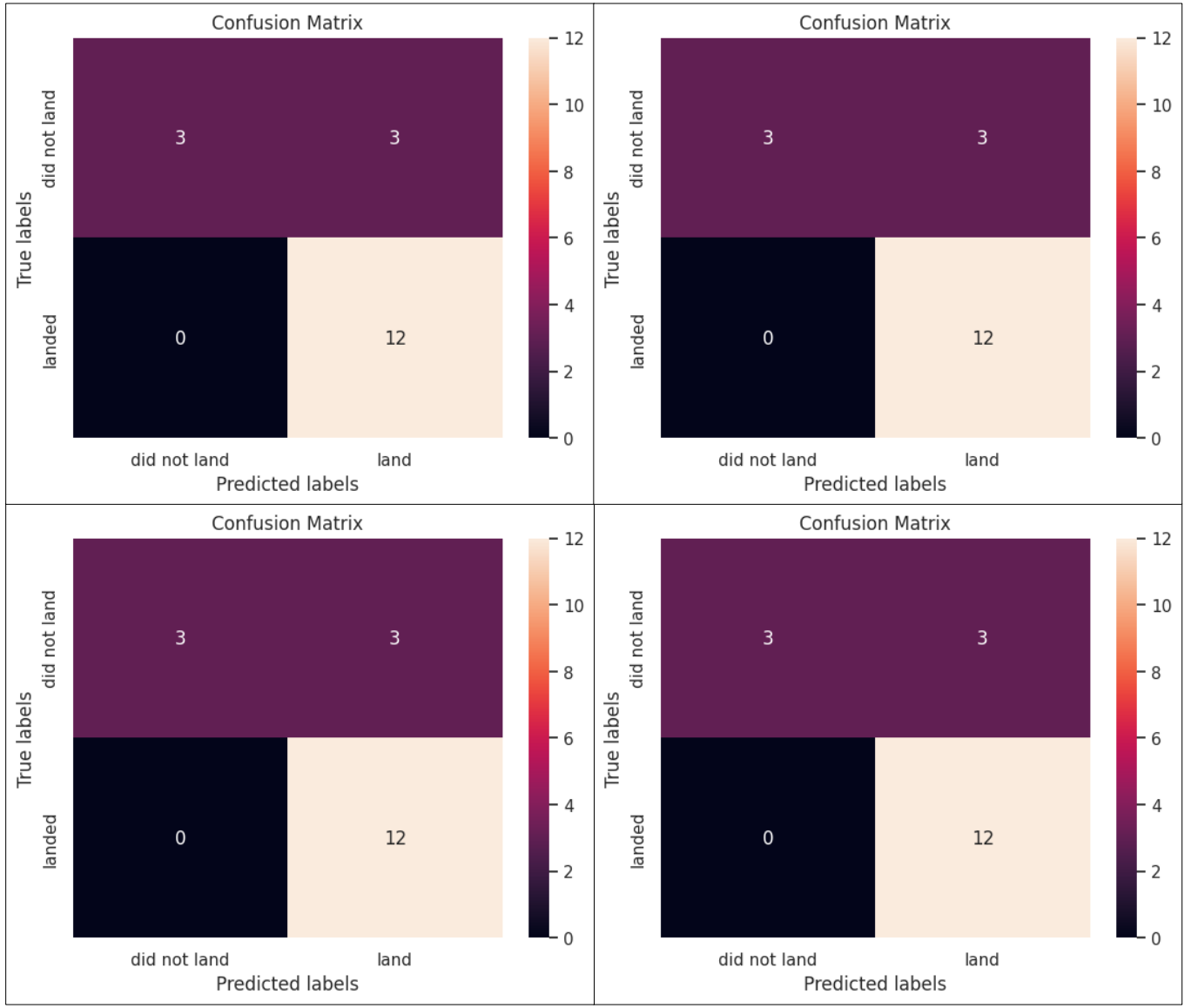
```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
accuracy : 0.8482142857142856
```

Decision Tree

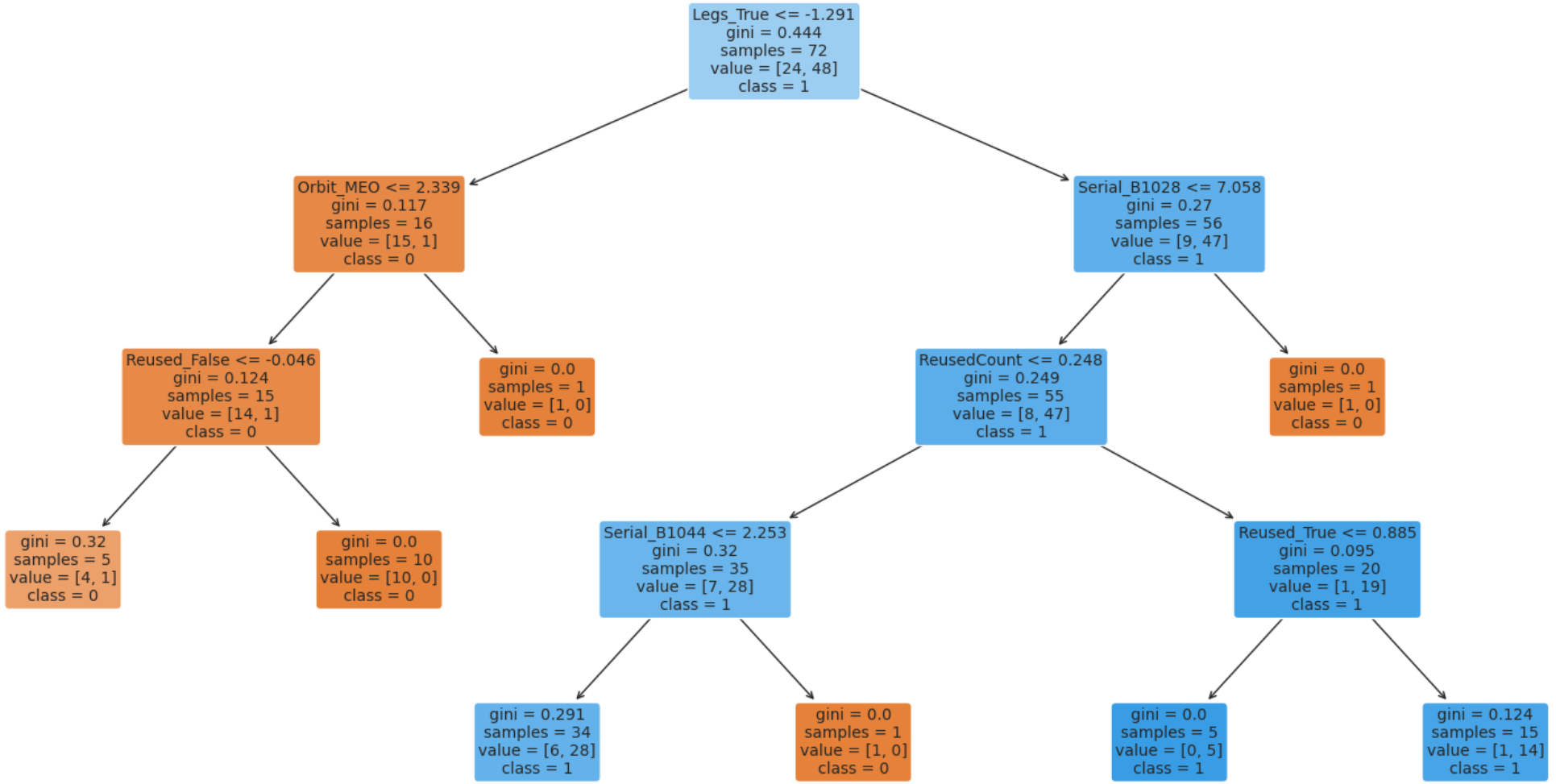
```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 4, 'max_features': 'auto', 'min_samples_leaf':
1, 'min_samples_split': 10, 'splitter': 'random'}
accuracy : 0.8875
```

Appendix – Machine Learning II
Confusion Matrices

- Confusion Matrix for all the Models.



Appendix – Machine Learning III
Decision tree




Thank You