Winning Space Race with Data Science

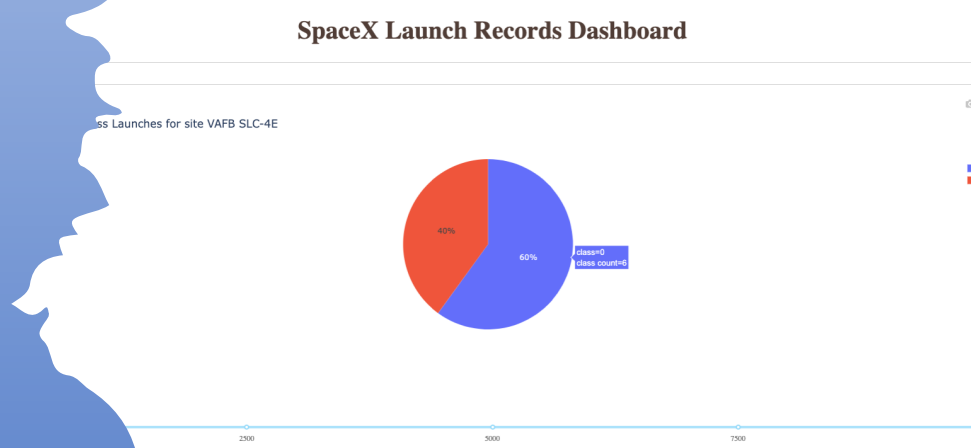
Vamshi Teja Vallala
August 30, 2023

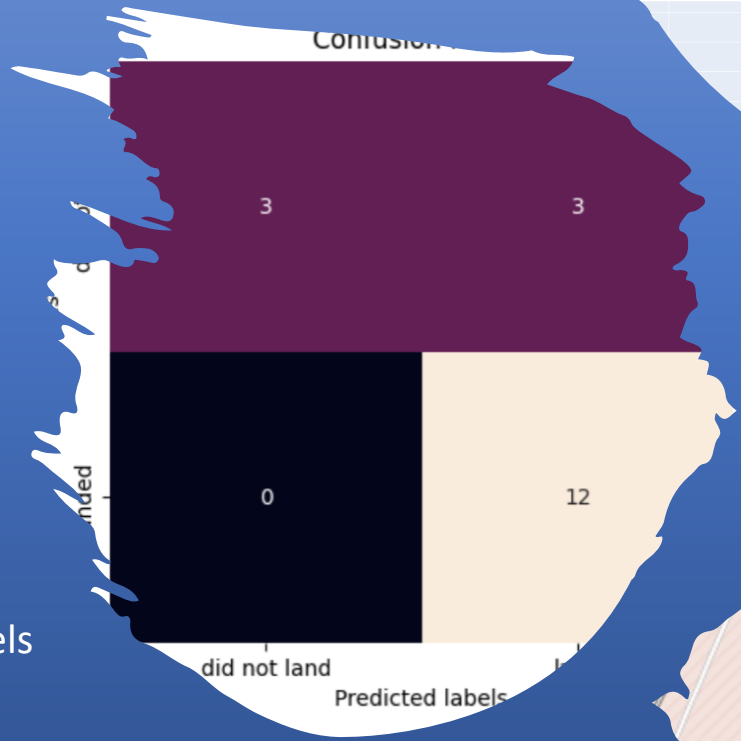
Outline

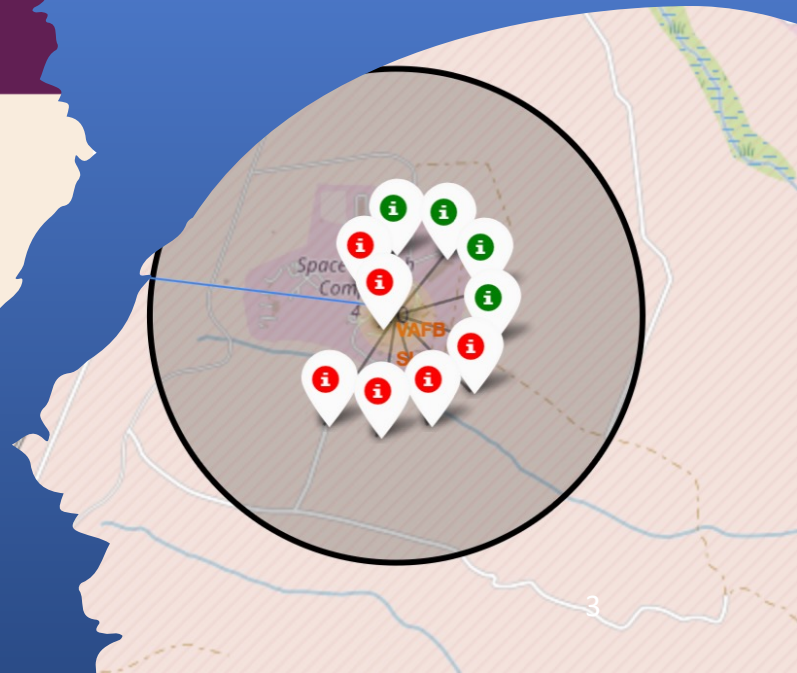
- Executive Summary
- Introduction
- Methodology
- Results
- Conclusion
- Appendix

EXECUTIVE SUMMARY

- This Project Consists of the Following Methodologies:
  - Data Collection
  - Data Wrangling
  - Data Analysis
  - Data Visualization
  - Predictive Analysis
- Outcomes of this Project are:
  - Exploratory Data Analysis Results
  - Interactive Dashboards
  - Predictive Classification and Regression Models
  - Geo Spatial Analysis







INTRODUCTION

Embark on an extraordinary data-driven odyssey exploring the frontier of commercial space exploration. This comprehensive data science project seamlessly navigates through the intricacies of data collection, meticulous wrangling, insightful exploratory analysis, predictive modeling, and the captivating realm of geo-spatial analysis. Our ultimate mission? To unravel the intriguing puzzle of whether SpaceX will opt to reuse its first stage in its rocket launches.

With each data point meticulously gathered and refined, we embark on an intellectual expedition to predict the pivotal decision of first stage reusability using advanced machine learning techniques. This venture is more than data; it's an innovation symphony where human intellect and technology harmonize. In this journey, every line of code and every insightful revelation propels us closer to unlocking the cosmos of SpaceX's reusability strategy.

Section 1

Methodology

Methodology

1. Data Collection
   - Making GET requests to the SpaceX REST API
   - Web Scraping
2. Data Wrangling
   - Using the `.fillna()` method to remove NaN values
   - Using the `.value_counts()` method to determine the following:
     - Number of launches on each site
     - Number and occurrence of each orbit
     - Number and occurrence of mission outcome per orbit type
   - Creating a landing outcome label that shows the following:
     - 0 when the booster did not land successfully
     - 1 when the booster did land successfully
3. Exploratory Data Analysis
   - Using SQL queries to manipulate and evaluate the SpaceX dataset
   - Using Pandas and Matplotlib to visualize relationships between variables, and determine patterns
4. Interactive Visual Analytics
   - Geospatial analytics using Folium
   - Creating an interactive dashboard using Plotly Dash
5. Data Modelling and Evaluation
   - Using Scikit-Learn to:

     Pre-process (standardize) the data
     Split the data into training and testing data using `train_test_split`
     Train different classification models
     Find hyperparameters using `GridSearchCV`

   Plotting confusion matrices for each classification model
   Assessing the accuracy of each classification model

Using the SpaceX API to retrieve data about launches, including information about the rocket used, payload delivered, launch specifications, landing specifications, and landing outcome.

- Make a GET response to the SpaceX REST API
- Convert the response to a .json file then to a Pandas DataFrame

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

response = requests.get(spacex_url)
```

```
# Use json_normalize method to convert the json result into a dataframe
data = pd.json_normalize(response.json())
```

- Use custom logic to clean the data
- Define lists for data to be stored in
- Call custom functions to retrieve data and fill the lists
- Use these lists as values in a dictionary and construct the dataset

```
#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}
```

- Create a Pandas DataFrame from the constructed dictionary dataset

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

- Filter the DataFrame to only include Falcon 9 launches
- Reset the FlightNumber column
- Replace missing values of PayloadMass with the mean PayloadMass value

```
data_falcon9 = df[df['BoosterVersion']!='Falcon 1']
```

```
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
```

```
# Calculate the mean value of PayloadMass column and Replace the np.nan values with its mean value
data_falcon9 = data_falcon9.fillna(value={'PayloadMass': data_falcon9['PayloadMass'].mean()})
```

Github Link

Data Collection - Scraping

We collect data from a Wikipedia page that lists all Falcon 9 missions.

1. We request HTML object from a URL and save it in a response.
2. Create a Beautiful Soup Object and then collect all column names found in the HTML Page.
3. Use the column names as keys in a dictionary
4. Use custom functions and logic to parse all launch tables to fill the dictionary values.
5. Convert the dictionary to a Pandas DataFrame ready for export

```
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
response = requests.get(static_url)
# assign the response to a object
data = response.text
```

```
soup = BeautifulSoup(data, 'html5lib')
html_tables = soup.find_all('table')
```

```
column_names = []

# Apply find_all() function with `th` element on first_launch_table
# Iterate each th element and apply the provided extract_column_from_header() to get a column name
# Append the Non-empty column name (`if name is not None and len(name) > 0`) into a list called 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)
```

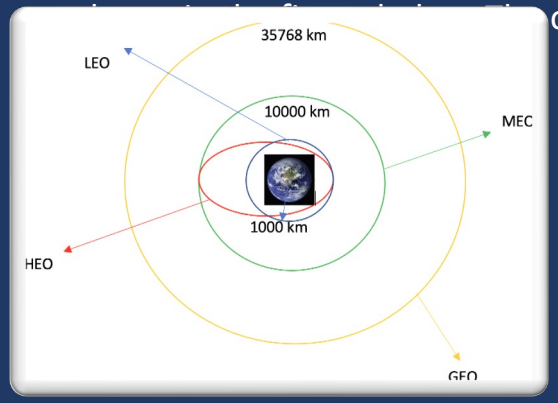
```
launch_dict= dict.fromkeys(column_names)

# 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']=[]
```

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

Data Wrangling - GITHUB

Context:

- The SpaceX dataset contains several Space X launch facilities, and each location is in the LaunchSite column.
- Each launch aims to a dedicated orbit, and some of the common orbit types orbit type is in the Orbit column.



Initial Data Exploration:

- Using the .value_counts() method to determine the following:
  1. Number of launches on each site
  2. Number and occurrence of each orbit
  3. Number and occurrence of landing outcome per orbit type

```
# Apply value_counts() on column LaunchSite
df['LaunchSite'].value_counts()
```

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

```
# Apply value_counts on Orbit column
df['Orbit'].value_counts()
```

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

```
# landing_outcomes = values on Outcome column
landing_outcomes = df['Outcome'].value_counts()
landing_outcomes
```

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

Data Wrangling - GITHUB

Data Wrangling:

- To determine whether a booster will successfully land, it is best to have a binary column, i.e., where the value is 1 or 0, representing the success of the landing.
- This is done by:
  1. Defining a set of unsuccessful (bad) outcomes, `bad_outcome`
  2. Creating a list, `landing_class`, where the element is 0 if the corresponding row in `Outcome` is in the set `bad_outcome`, otherwise, it's 1.
  3. Create a `Class` column that contains the values from the list `landing_class`
  4. Export the DataFrame as a .csv file.

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

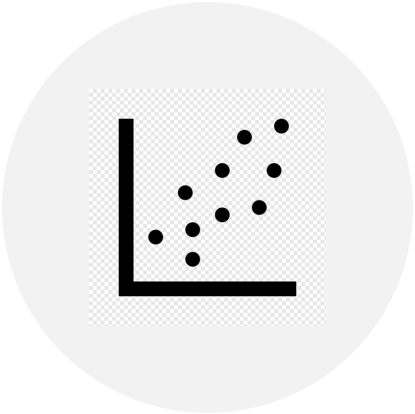
```
{'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise

landing_class = []

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

```
df['Class']=landing_class
```

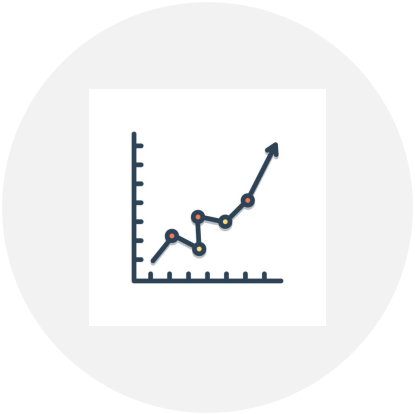
```
df.to_csv("dataset_part\_2.csv", index=False)
```

EDA with Data Visualization - GITHUB



SCATTER PLOTS

SCATTER PLOTS ARE USED TO OBSERVE CORRELATIONS. WE USED IT FOR FLIGHT NUMBER VS LAUNCH SITE, PAYLOAD VS ORBIT TYPE FOR EXAMPLE.



LINE CHARTS

LINE CHARTS ARE USED TO COMPARE NUMERICAL VALUES, IN OUR CASE, WE USED IT FOR SUCCESS RATE VS YEAR (FOR YEARLY TRENDS)

BAR CHARTS

BAR CHARTS ARE USED TO COMPARE CATEGORICAL VALUES AND, IN OUR CASE, THE SUCCESS RATE VS ORBIT TYPE

EDA with SQL - GITHUB

SQL QUERIES PERFORMED TO GATHER DATA ARE:

- **Display the names of the unique launch sites in the space mission**
- **Display 5 records where launch sites begin with the string 'CCA'**
- **Display the total payload mass carried by boosters launched by NASA (CRS)**
- **Display average payload mass carried by booster version F9 v1.1**
- **List the date when the first succesful landing outcome in ground pad was acheived.**
- **List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000**
- **List the total number of successful and failure mission outcomes**
- **List the names of the booster_versions which have carried the maximum payload mass. Use a subquery**
- **List the records which will display the month names, failure landing_outcomes in drone ship ,booster versions, launch_site for the months in year 2015.**
- **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.**

Build an Interactive Map with Folium - GITHUB

- The following steps were taken to visualize the launch data on an interactive map:

1. Mark all launch sites on a map
   - Initialise the map using a Folium `Map` object
   - Add a `folium.Circle` and `folium.Marker` for each launch site on the launch map

2. Mark the success/failed launches for each site on a map
   - As many launches have the same coordinates, it makes sense to cluster them together.
   - Before clustering them, assign a marker colour of successful (class = 1) as green, and failed (class = 0) as red.
   - To put the launches into clusters, for each launch, add a `folium.Marker` to the `MarkerCluster()` object.
   - Create an icon as a text label, assigning the `icon_color` as the `marker_colour` determined previously.

3. Calculate the distances between a launch site to its proximities
   - To explore the proximities of launch sites, calculations of distances between points can be made using the `Lat` and `Long` values.
   - After marking a point using the `Lat` and `Long` values, create a `folium.Marker` object to show the distance.
   - To display the distance line between two points, draw a `folium.PolyLine` and add this to the map.

Build a Dashboard with Plotly Dash - GITHUB

- The following plots were added to a Plotly Dash dashboard to have an interactive visualisation of the data:

1. Pie chart (`px.pie()`) showing the total successful launches per site
   - This makes it clear to see which sites are most successful
   - The chart could also be filtered (using a `dcc.Dropdown()` object) to see the success/failure ratio for an individual site

2. Scatter graph (`px.scatter()`) to show the correlation between outcome (success or not) and payload mass (kg)
   - This could be filtered (using a `RangeSlider()` object) by ranges of payload masses
   - It could also be filtered by booster version

Predictive Analysis (Classification) - GITHUB

MODEL DEVELOPMENT

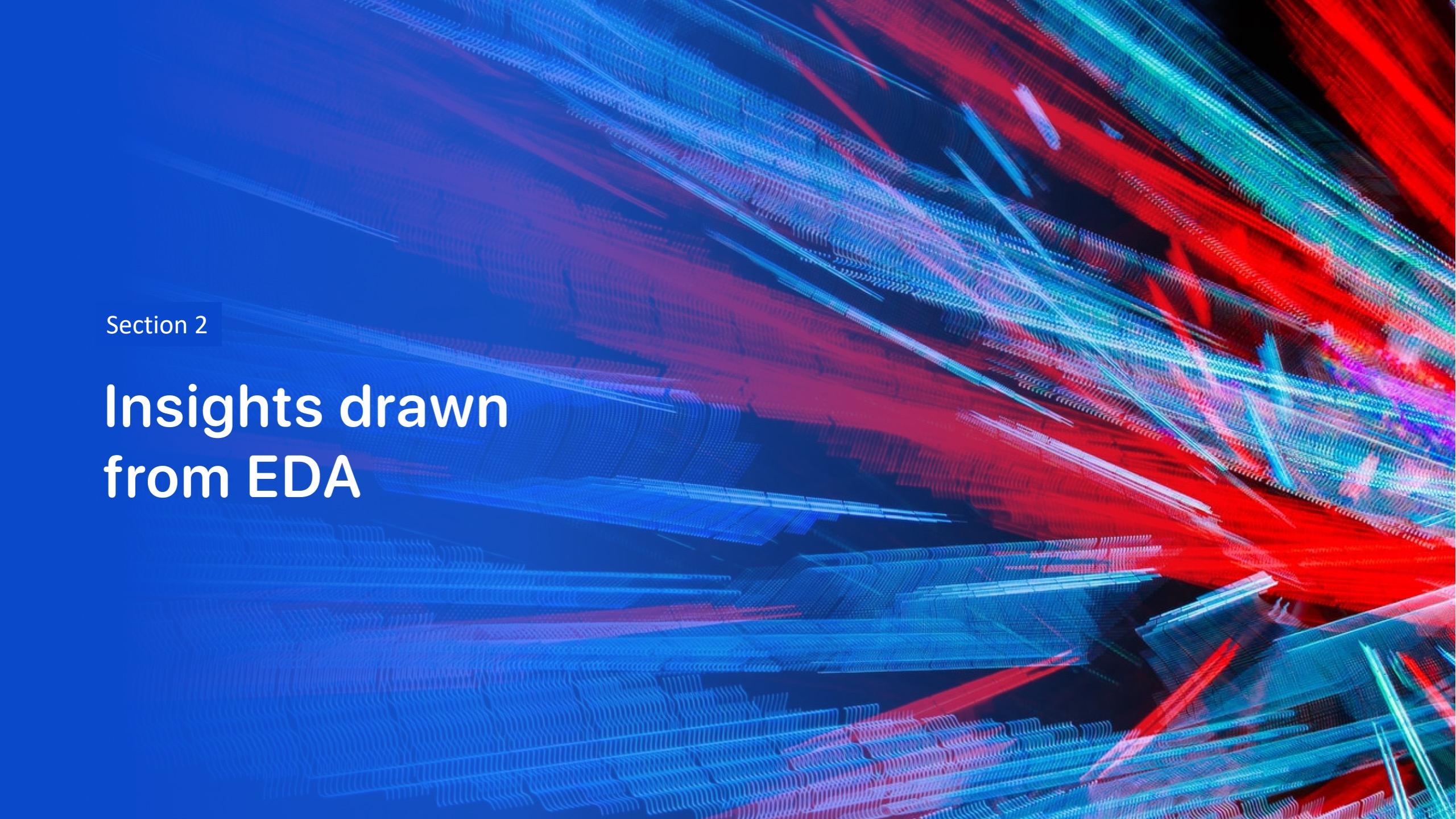
- After Dataset was cleaned and transformed with preprocessing, data was then split into train and test sets with train_test_split().
- Models were built using Linear Reg, Logistic Reg, SVM, KNN, and Decision Trees.
- To choose the parameters, the models are then run through GridSearchCV.

MODEL EVALUATION

- For model evaluation, several metrics were chosen. Accuracy, F1 score, MSE, MAE were the main metrics.
- The model was run through GridSearchCV for choosing the best hyperparameters.
- A validation set is used to choose the best parameters for a model with the highest accuracy and minimal loss.

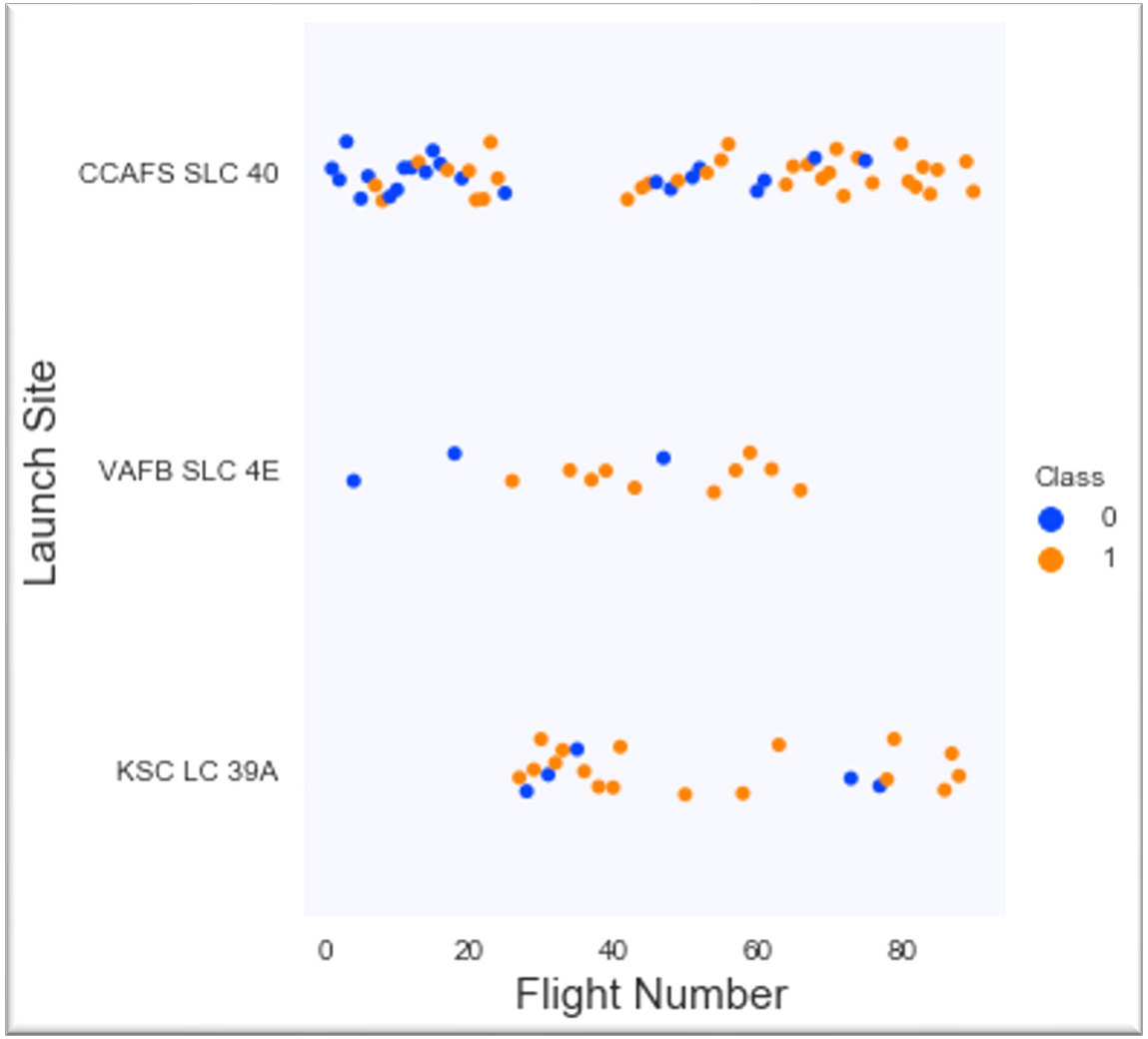
FINAL MODEL SELECTION

- We plot the confusion matrix and and examine the performance.
- We choose the best accuracy model.


Section 2
Insights drawn from EDA

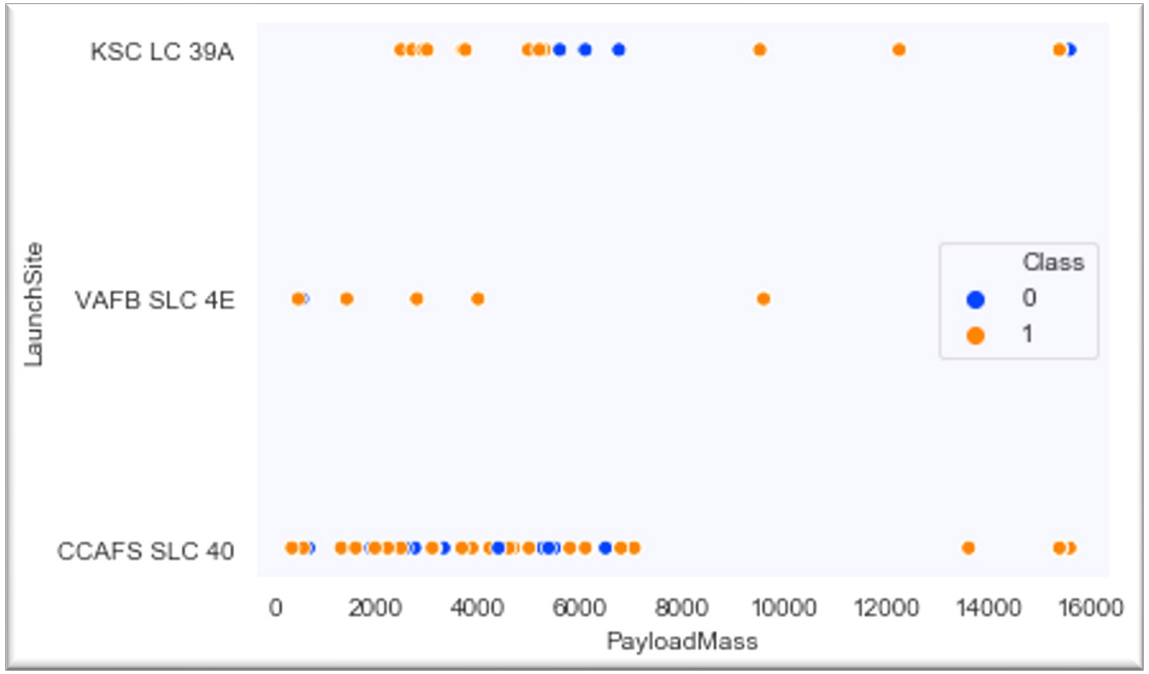
Flight Number vs. Launch Site

- We can see that most early flights from the launch site CCAFS SLC 40 were unsuccessful.
- As the flight number increased, as more and more flights were launched, the success rate increased as well.
- VAFB SLC 4E was scarcely used compared to the others.



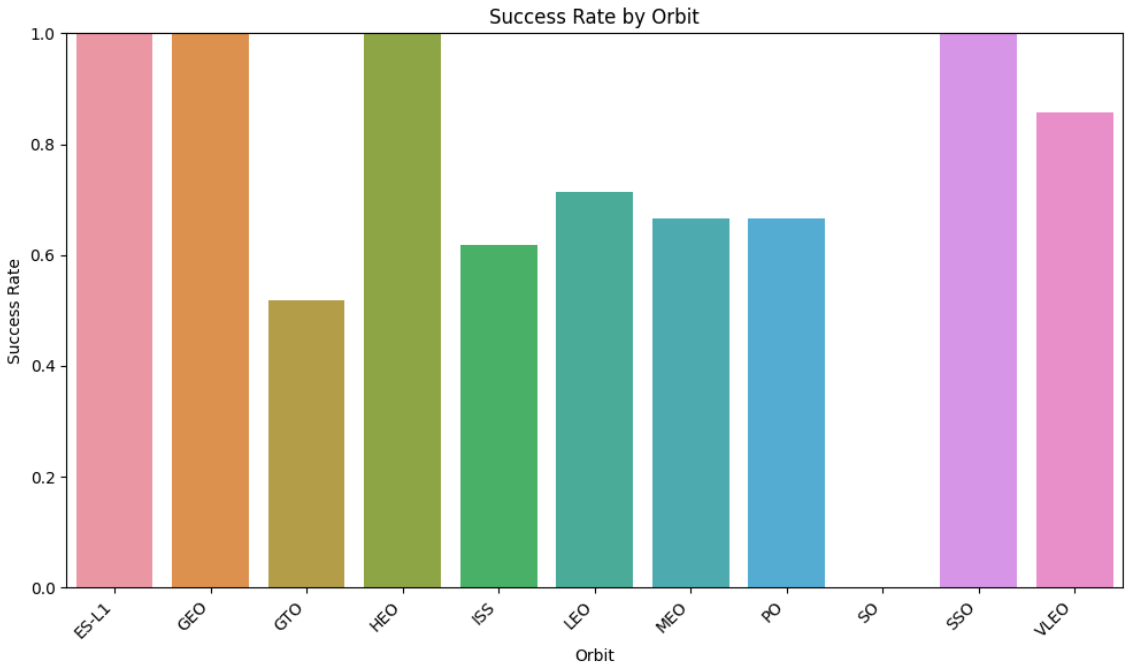
Payload vs. Launch Site

- I see no correlation between these launch sites and payload mass.
- Payload mass > 7000 has very few failures.



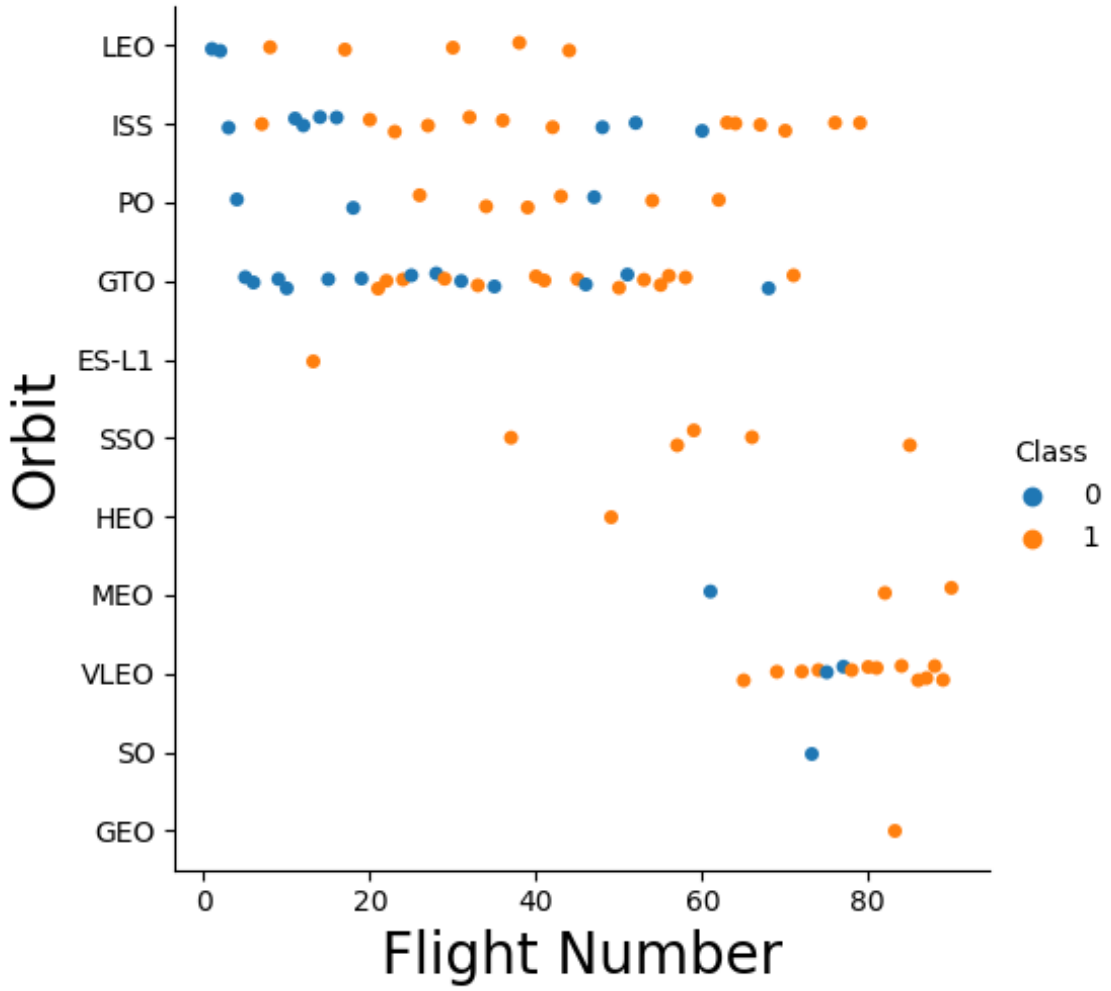
Success Rate vs. Orbit Type

- We can draw the conclusion that success rate for all orbits except SO was over 50%.
- SO only had one launch and it was a failure.
- The success rate for Orbits ES-L1, GEO, HEO, and SSO were 100%.



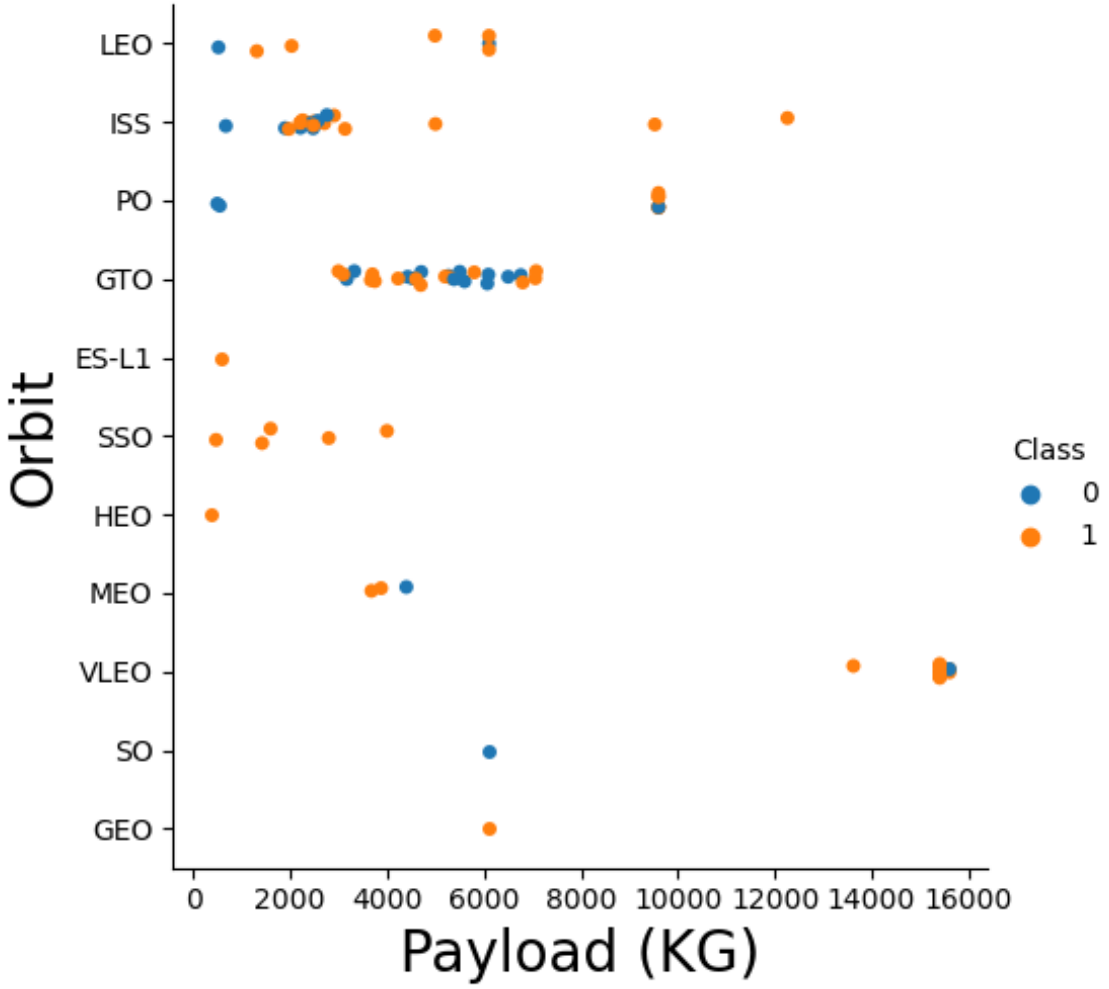
Flight Number vs. Orbit Type

- Most initial launches were done into orbits LEO, ISS, PO, GTO.
- Possibly because these are low risk orbits. As we can see, most flight numbers < 40 were failures.
- These 4 orbits were tested until 60 flights until success was more often, then they shifted to other orbits like SSO, HEO, MEO, VLEO etc.



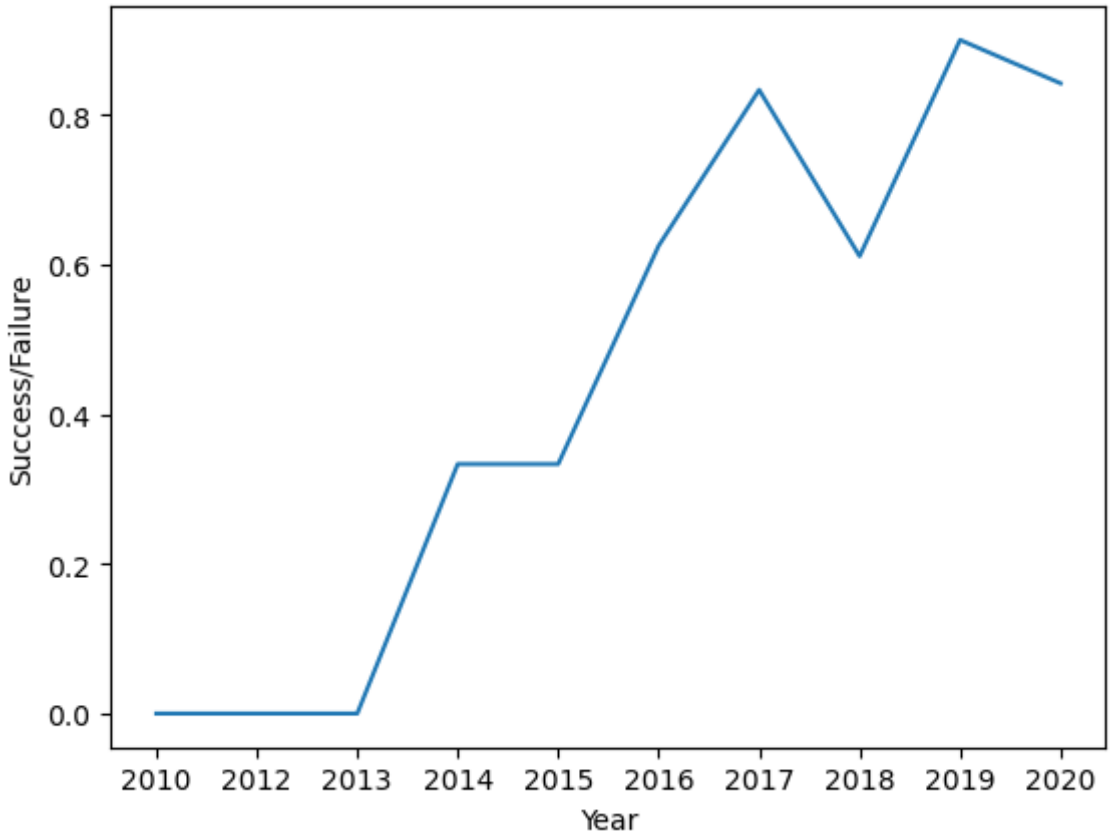
Payload vs. Orbit Type

- Initial, lower payloads were launched into orbits SSO, HEO, ISS, LEO. These orbits were tested for success.
- From payload > 3000, Orbit GTO was extensively used, and its success rate is as much as its failure.
- From payload > 8000, GTO was relieved and other orbits were proving greater success.



Launch Success Yearly Trend

- The yearly trend for success is almost an upward trend.
- Except for the drop in 2018, Space X has seen tremendous upward success in their launches.



All Launch Site Names

```
%sql SELECT DISTINCT Launch_Site FROM SPACEXTABLE;
```

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

- The query SELECT DISTINCT will show unique names in Launch_Site column in the SPACEXTABLE;

Launch Site Names 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
2010-04-06	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

- The query shows all rows of the table SPACEXTABLE where Launch_Site column contains a string expression ‘CCA’

Total Payload Mass

```
%%sql

SELECT SUM(PAYLOAD_MASS__KG_) AS TOTAL_PAYLOAD_MASS_KG
FROM SPACEXTABLE
WHERE Customer = 'NASA (CRS)';
```

TOTAL_PAYLOAD_MASS_KG
45596

- The query sums the whole payload mass and presents it in a column name “TOTAL_PAYLOAD_MASS_KG” for the customer NASA(CRS).
- This helps us identify the payload for a specific customer of interest.

Average Payload Mass by F9 v1.1

```
%%sql

SELECT AVG(PAYLOAD_MASS__KG_) AS AVG_PAYLOAD_MASS_KG
FROM SPACEXTABLE
WHERE Booster_Version LIKE 'F9 v1.1%';
```

AVG_PAYLOAD_MASS_KG
2534.6666666666665

- The query now averages the payload mass for a specific booster version F9 v1.1.

First Successful Ground Landing Date

```
%%sql

SELECT MIN(Date) as EARLIEST_SUCCESS FROM SPACEXTABLE WHERE Landing_Outcome LIKE 'SUC%';
```

EARLIEST_SUCCESS
2015-12-22

- The query shows the earliest Date (MIN) where the landing outcome column shows success(of any kind).

Successful Drone Ship Landing with Payload between 4000 and 6000

```
%%sql
SELECT Booster_Version
FROM SPACEXTABLE WHERE Landing_Outcome='Success (drone ship)'
AND PAYLOAD_MASS__KG_ > 4000 AND PAYLOAD_MASS__KG_ < 6000
```

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

- The query shows us the Booster Versions that have successful drone ship landing outcome.
- It further narrows down for Payload Mass between 4000 and 6000 kgs.

Total Number of Successful and Failure Mission Outcomes

```
%%sql
SELECT Mission_Outcome, COUNT(*) 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 query here groups the mission outcomes by successes and failures using the COUNT keyword.
- In the table, we have different kinds of success hence 3 different groups.

Boosters Carried Maximum Payload

```
%%sql

SELECT 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

- The query here shows us the Booster Version that have carried the maximum Payload.
- To calculate the maximum Payload_mass, we use a subquery SELECT MAX().

2015 Launch Records

```
%%sql
SELECT
    Booster_Version,
    Launch_Site,
    substr(Date, 4, 2) AS Month,
    Landing_Outcome
FROM
    SPACEXTABLE
WHERE
    Landing_Outcome = 'Failure (drone ship)' AND substr(Date, 7, 4) = '2015';
```

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

- The query lists the failed landing outcomes in drone ship, in the year 2015.

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

```
%%sql
SELECT Landing_Outcome, COUNT(*) AS COUNT_LAUNCHES
FROM SPACEXTABLE WHERE Date BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY Landing_Outcome
ORDER BY COUNT_LAUNCHES DESC;
```

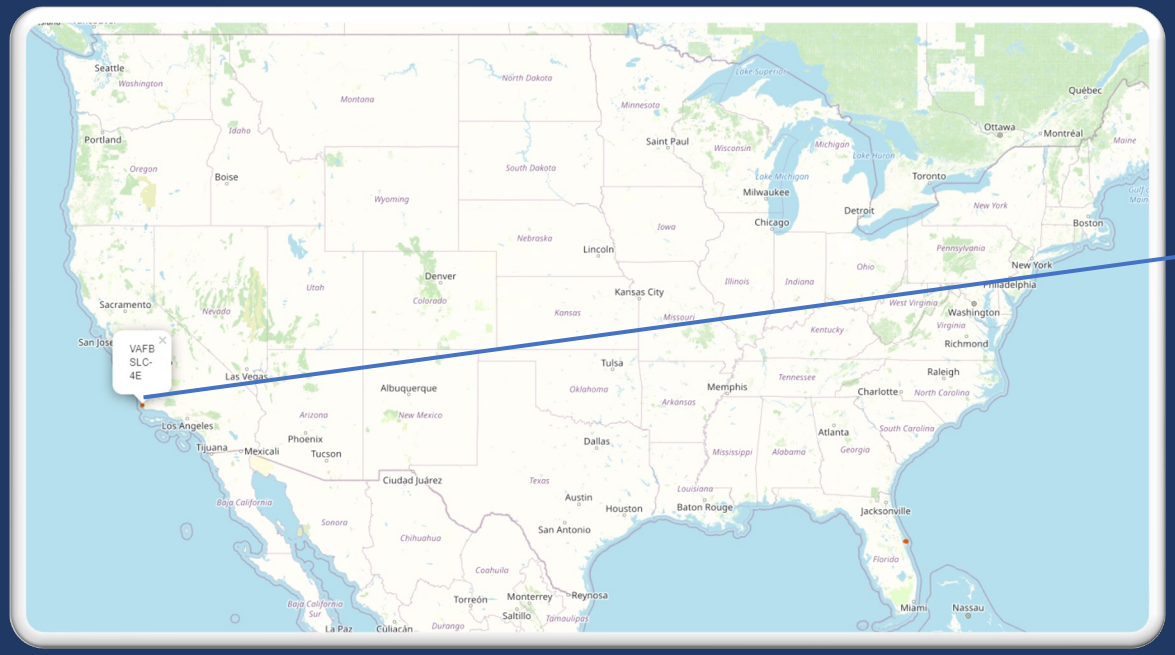
Landing_Outcome	COUNT_LAUNCHES
No attempt	10
Success (ground pad)	5
Success (drone ship)	5
Failure (drone ship)	5
Controlled (ocean)	3
Uncontrolled (ocean)	2
Precluded (drone ship)	1
Failure (parachute)	1

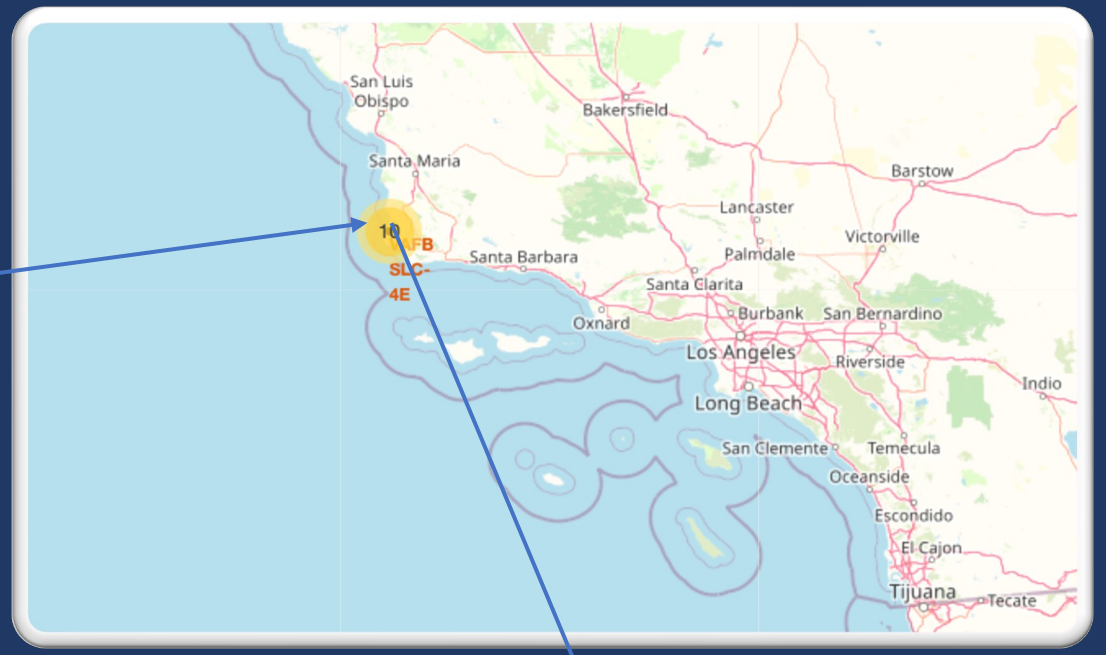
- The query sorts (ranks) the Landing Outcome groups (such as Failure (drone ship) or Success (ground pad)) based on the number of counts.
- The query narrows the search to be between June 4th, 2010, and March 20th, 2017.


Section 3
Launch Sites
Proximities Analysis

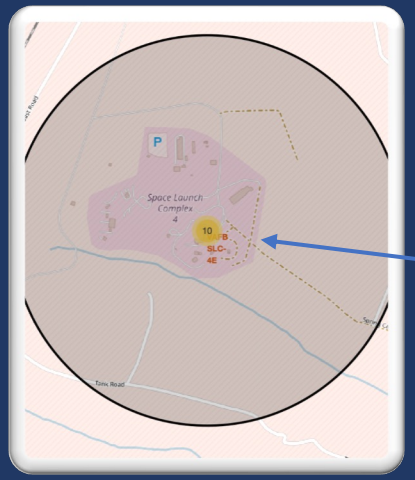
<Folium Map Screenshot 1>

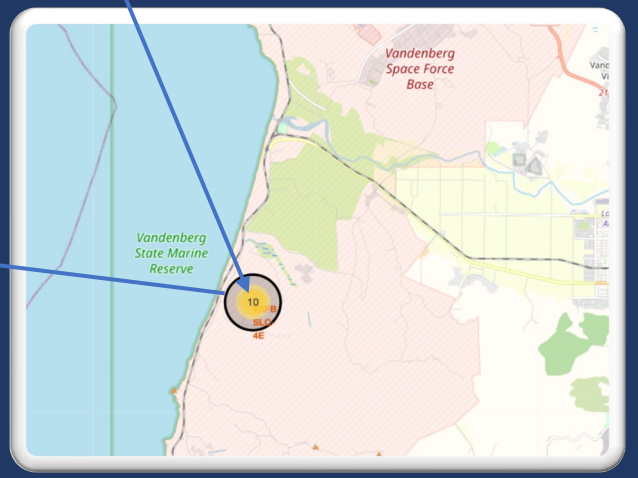
- Replace <Folium map screenshot 1> title with an appropriate title
- Explore the generated folium map and make a proper screenshot to include all launch sites' location markers on a global map
- Explain the important elements and findings on the screenshot



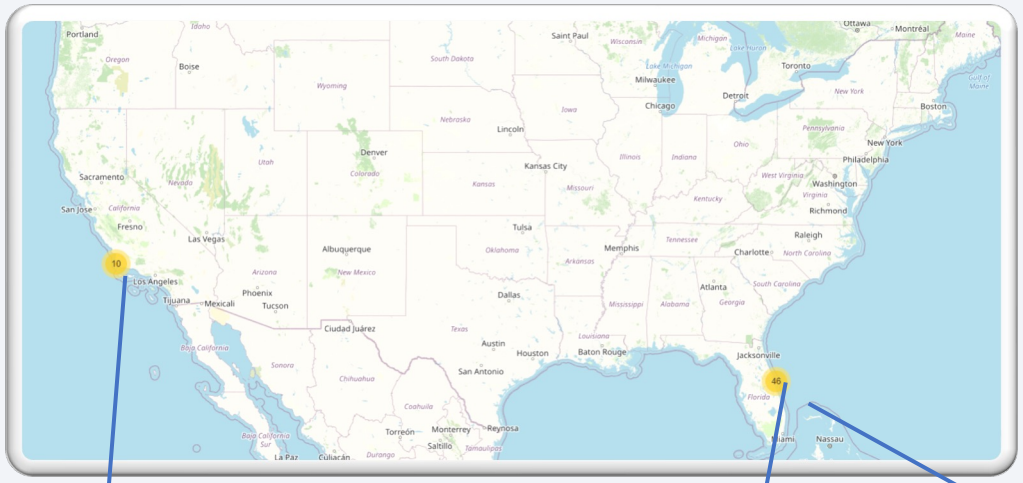


- All SpaceX launch sites are on coasts of the United States of America, specifically Florida and California.



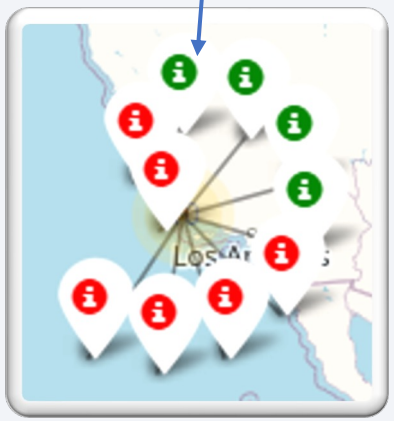


SUCCESS AND FAILURES BY SITES

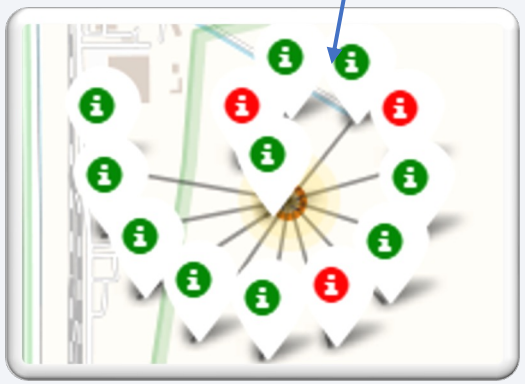


- Launches have been grouped into clusters, and annotated with green icons for successful launches, and red icons for failed launches.

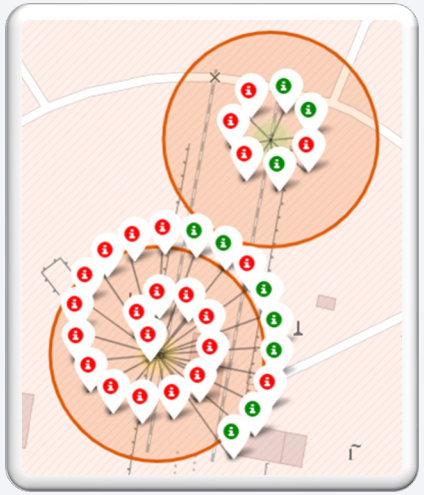
VAFB SLC-4E



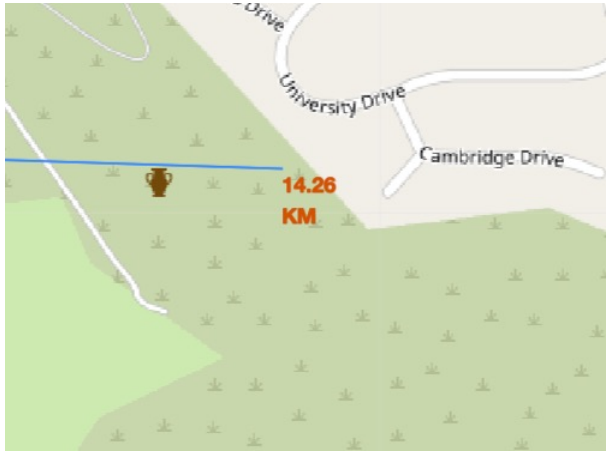
KSC LC-39A

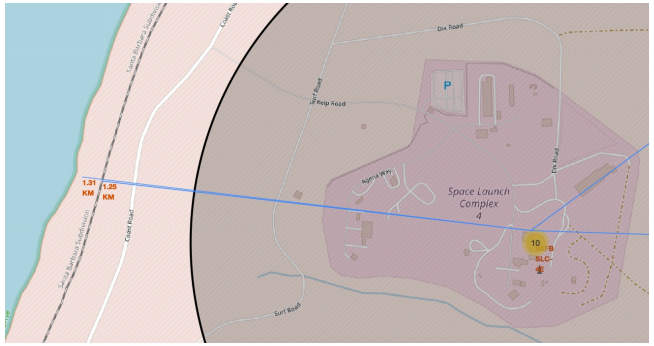


CCAFS SLC-40 and CCAFS LC-40



FOLIUM MAP PROXIMITIES





- Using the VAFB SLC-4E launch site as an example site, we can understand more about the placement of launch sites.

Are launch sites near railways?

- YES. The coastline is only 1.31 km due West.

Are launch sites near highways?

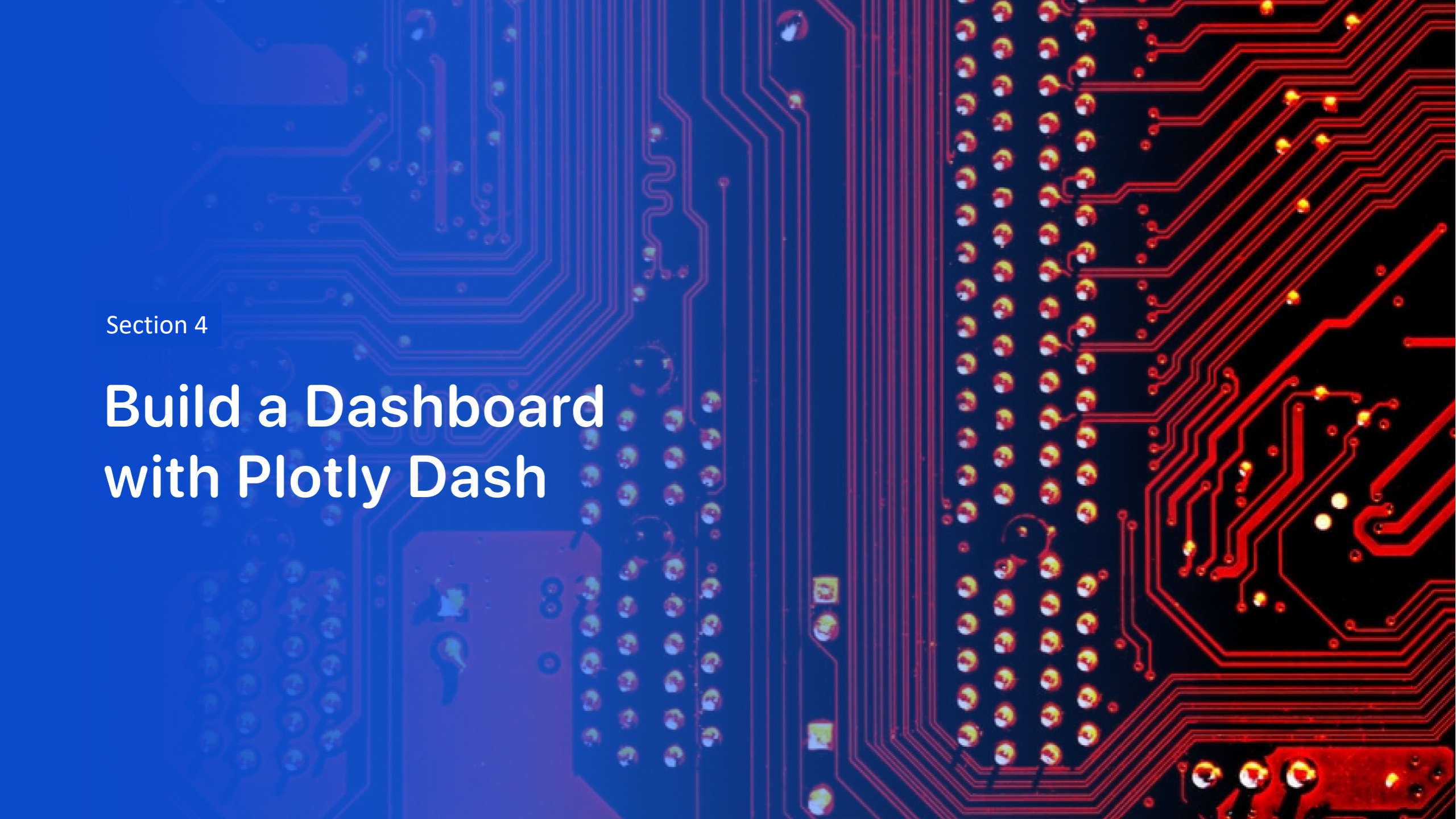
- YES. The nearest highway is only 14.06 km away.

Are launch sites near to railways?

- YES. The nearest railway is only 1.25 km away.

Do launch sites keep certain distance away from cities?

- YES. The nearest city is 14.26 km away.

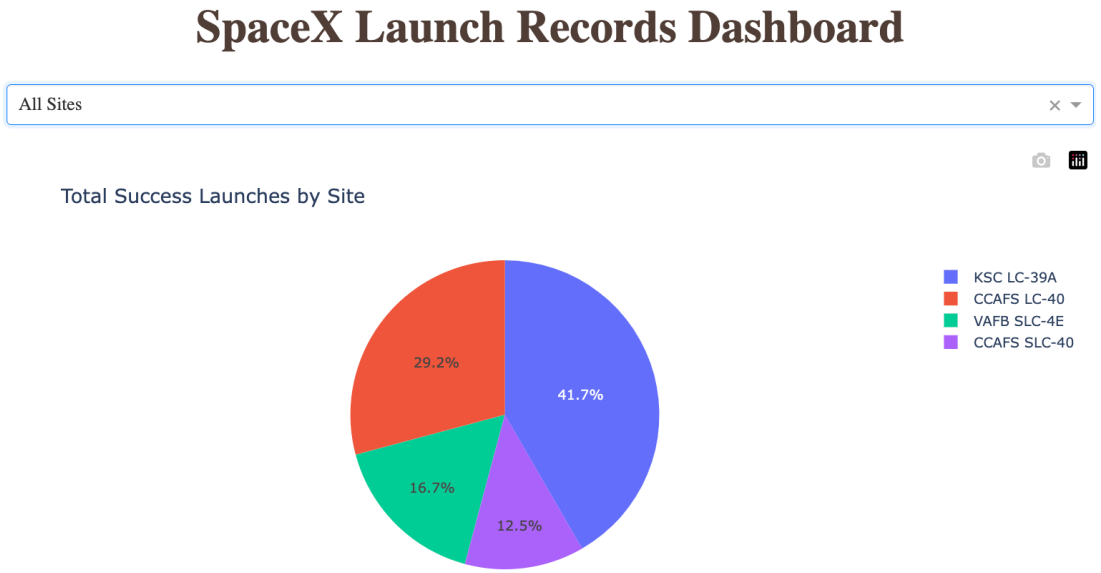


Section 4

Build a Dashboard with Plotly Dash

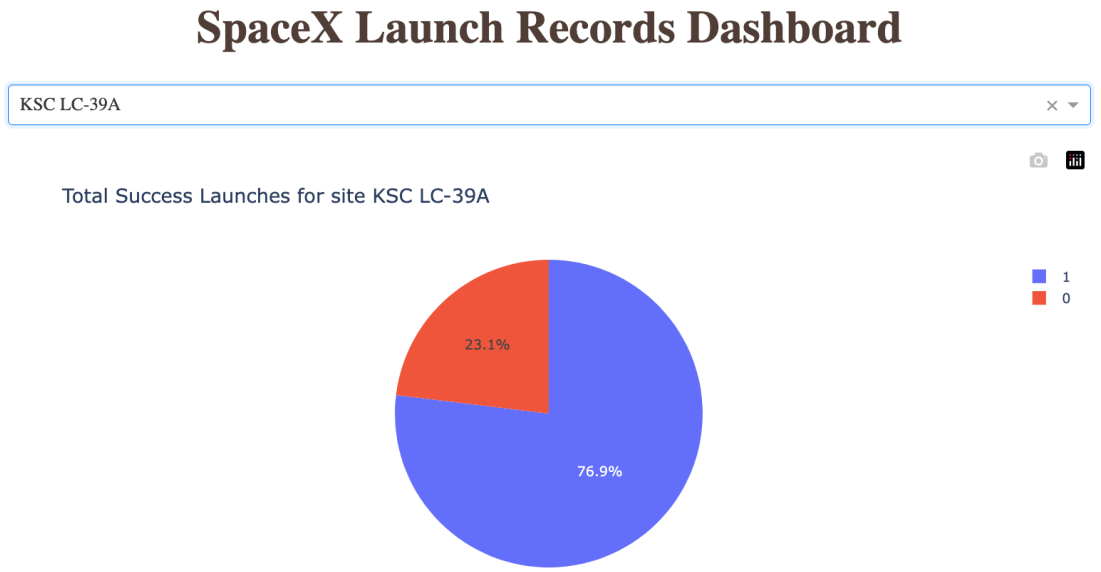
PIE CHART FOR ALL LAUNCH SITES

- In the pie chart for all launch sites, we can see the KSC LC-39A has the highest success percentage at 41.7%.

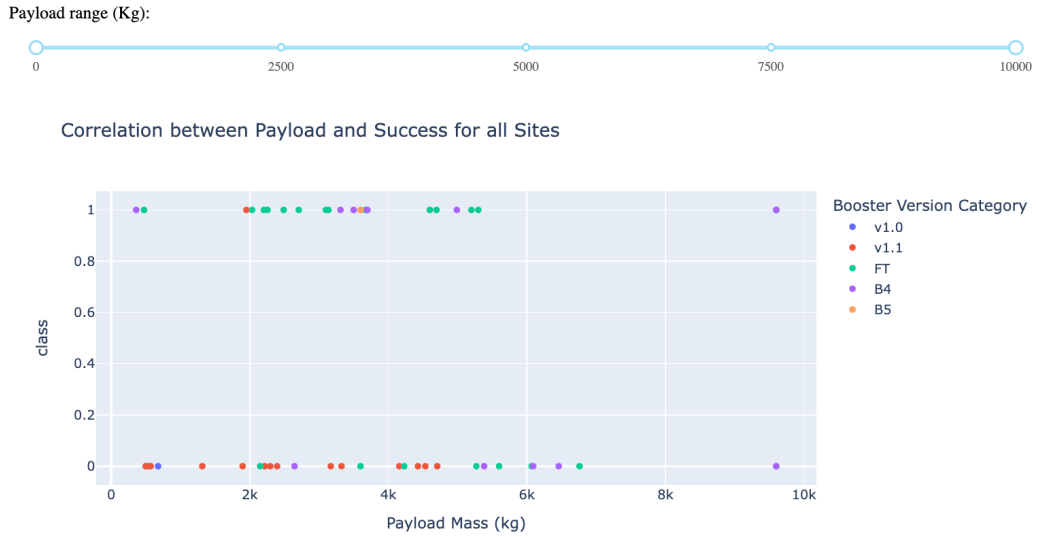


PIE CHART FOR LAUNCH SITE WITH HIGHEST SUCCESS

- KSC LC-39A is the Launch Site with the highest success rate at 76.9%.



LAUNCH OUTCOME VS PAYLOAD MASS FOR ALL SITES



- Plotting the launch outcome vs. payload for all sites shows a gap around 4000 kg, so it makes sense to split the data into 2 ranges:
  - 0 – 4000 kg (low payloads)
  - 4000 – 10000 kg (massive payloads)
- From these 2 plots, it can be shown that the success for massive payloads is lower than that for low payloads.
- It is also worth noting that some booster types (v1.0 and B5) have not been launched with massive payloads.

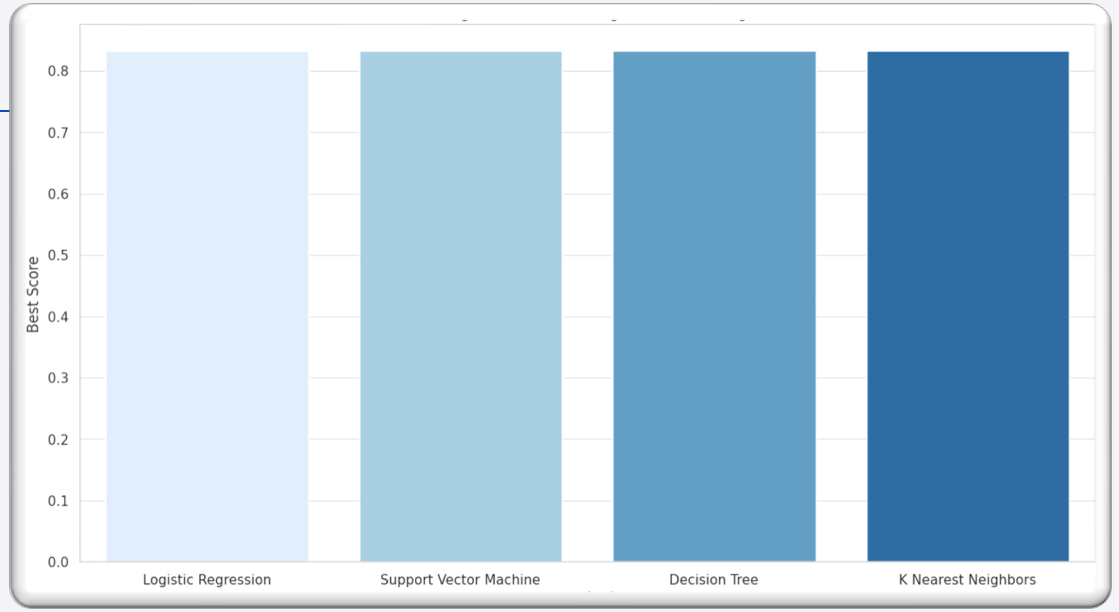
Section 5

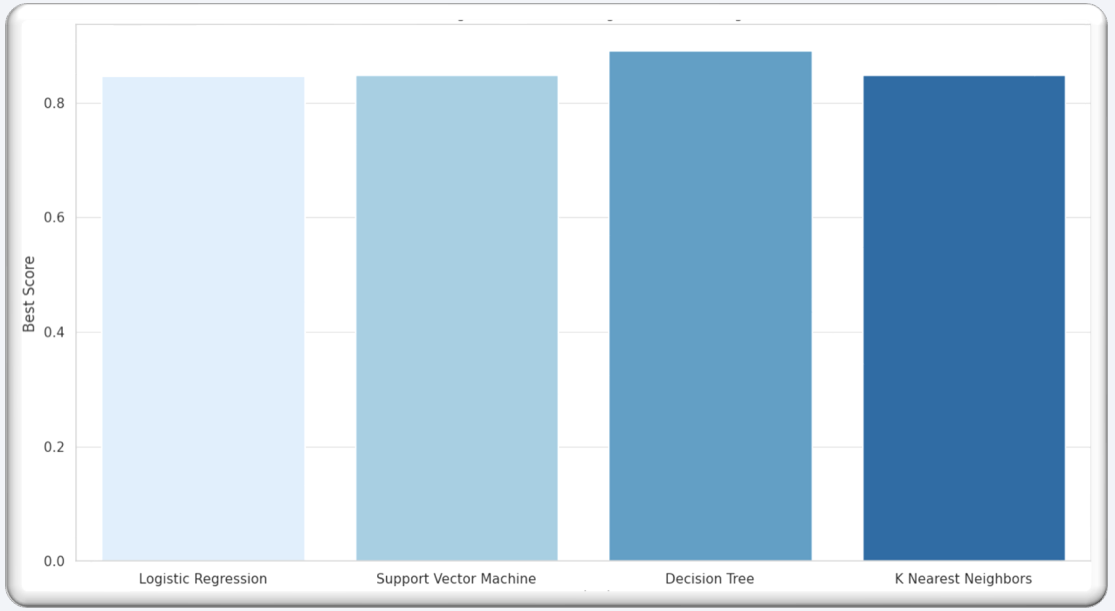
Predictive Analysis (Classification)

Classification Accuracy

- Plotting the Accuracy Score and Best Score for each classification algorithm produces the following result:
- ALL models have the same classification accuracy.
  - The Accuracy Score is 83.33%
  - The Best Score is 89.12%

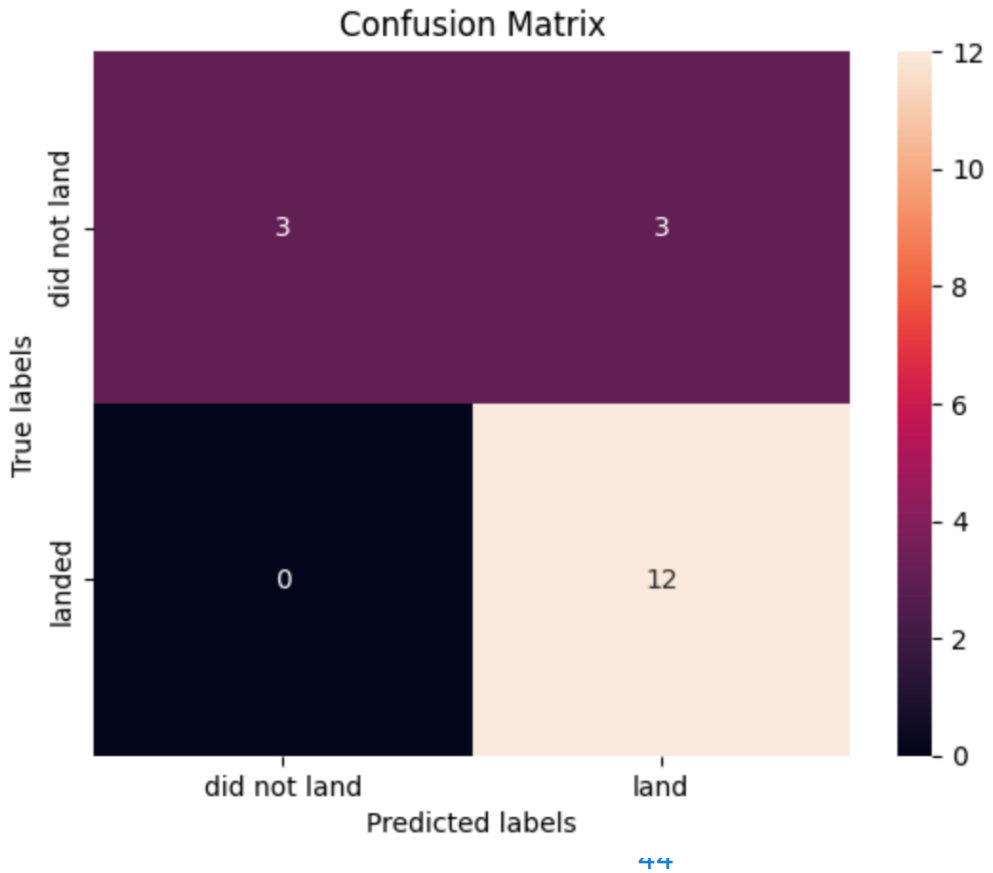
	Method	Accuracy	Best Score
0	Logistic Regression	0.833333	0.846429
1	Support Vector Machine	0.833333	0.848214
2	Decision Tree	0.833333	0.891071
3	K Nearest Neighbors	0.833333	0.848214





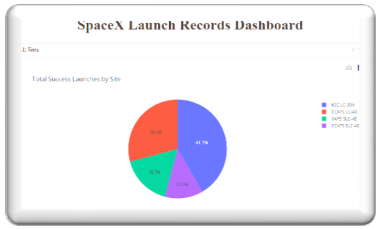
Confusion Matrix

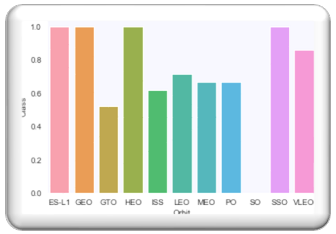
- As shown previously, all models classified with the same accuracy at 83.33%.
- This is explained by the 3 false negatives we have. The other categories are all correctly classified.
- The other 15 results are correctly classified (3 did not land, 12 did land, 3 were false positives).

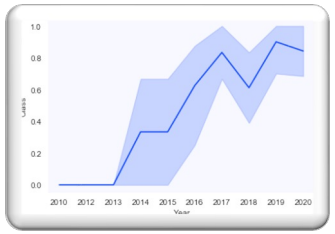


CONCLUSIONS

- As the number of flights increases, the rate of success at a launch site increases, with most early flights being unsuccessful. i.e. with more experience, the success rate increases.
  - Between 2010 and 2013, all landings were unsuccessful (as the success rate is 0).
  - After 2013, the success rate generally increased, despite small dips in 2018 and 2020.

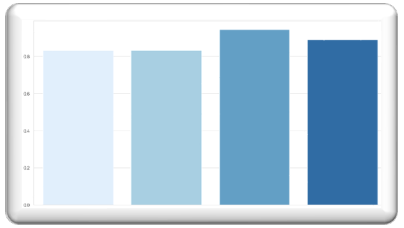


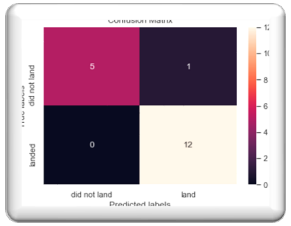




- Orbit types ES-L1, GEO, HEO, and SSO, have the highest (100%) success rate.
  - The 100% success rate of GEO, HEO, and ES-L1 orbits can be explained by only having 1 flight into the respective orbits.
  - The orbit types PO, ISS, and LEO, have more success with heavy payloads:
  - VLEO (Very Low Earth Orbit) launches are associated with heavier payloads, which makes intuitive sense.

- The launch site KSC LC-39 A had the most successful launches, with 41.7% of the total successful launches, and also the highest rate of successful launches, with a 76.9% success rate.





- As payload gets over 4000kgs, the success increases as well for all Booster Versions.

- All the models performed the same in accuracy with 83.33%

Appendix

- In the Data Collection – REST API, there are some customs logics used in Data Collection and Web Scraping which are worth noting:

From the `rocket` column we would like to learn the booster name.

```
# 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'])
```

From the `launchpad` we would like to know the name of the launch site being used, the logitude, and the latitude.

```
# 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'])
```

From the `payload` we would like to learn the mass of the payload and the orbit that it is going to.

```
# 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

- In the Data Collection – Web Scraping, there are some customs logics used in Data Collection and Web Scraping which are worth noting:

```
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
```

Thank you!