

# Winning Space Race with Data Science

Rahman Azari  
Oct 10, 2023



# Outline

---

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

# Executive Summary

---

- **Summary of methodologies**

This project implemented various data science techniques including data loading and reading, data wrangling, Exploratory Data Analysis with Data Visualization, Exploratory Data Analysis with SQL, data visualization, mapping, and creating Dashboards, and predictive analysis using various machine learning models.

- **Summary of all results**

The results illustrated the locations of space experiments with different landing outcomes, studied the patterns of data across launching sites, and demonstrated the predictive performance of different machine learning models with best hyperparameters and accuracy results.

# Introduction

---

- **Project background and context**

SpaceX has become the most successful commercial space company by making space travel affordable through reusing of the rockets. Falcon 9 rocket in particular has led to significant reduction of the costs by enabling the reuse of the rocket in the first stage. Therefore, the prediction of the outcome of the first stage (i.e., failure or success in landing) can help determine the cost of a launch. This project will use publicly available data and machine learning techniques to predict if SpaceX will reuse the first stage.

- **Problems you want to find answers**

This project aims to examine how variables such as launch site, payload mass, number of flights, and orbits affect the success of the first stage landing and identify the predictive machine learning algorithms to best predict the class outcome.

Section 1

# Methodology

# Methodology

---

## Executive Summary

- **Data collection methodology:**

Data were collected from two main sources including SpaceX REST API endpoints and web scraping related Wikipages.

- **Perform data wrangling**

Data wrangling in this project included creating dummy variables (one hot encoding) to convert categorical variables into multiple binary variables and identifying and handling of missing and null values.

# Methodology

## Executive Summary

- Perform exploratory data analysis (EDA) using visualization and SQL

Bar graphs, scatterplots, and descriptive statistics (such as minimum and sum) were used in this part to explore the links between success rate and other variables such as payload mass and launch sites.

# Methodology

## Executive Summary

- Perform interactive visual analytics using Folium and Plotly Dash

Using Folium and Plotly Dash, geographical maps and dashboards were constructed in this part to communicate the patterns in data with regard to launch site, etc.

## Perform predictive analysis using classification models

- In this part, different machine learning classification models were tuned and fit to data to predict the outcome of launch.

# Data Collection

---

Data were collected from two main sources including SpaceX REST API endpoints and web scraping related Wikipages.

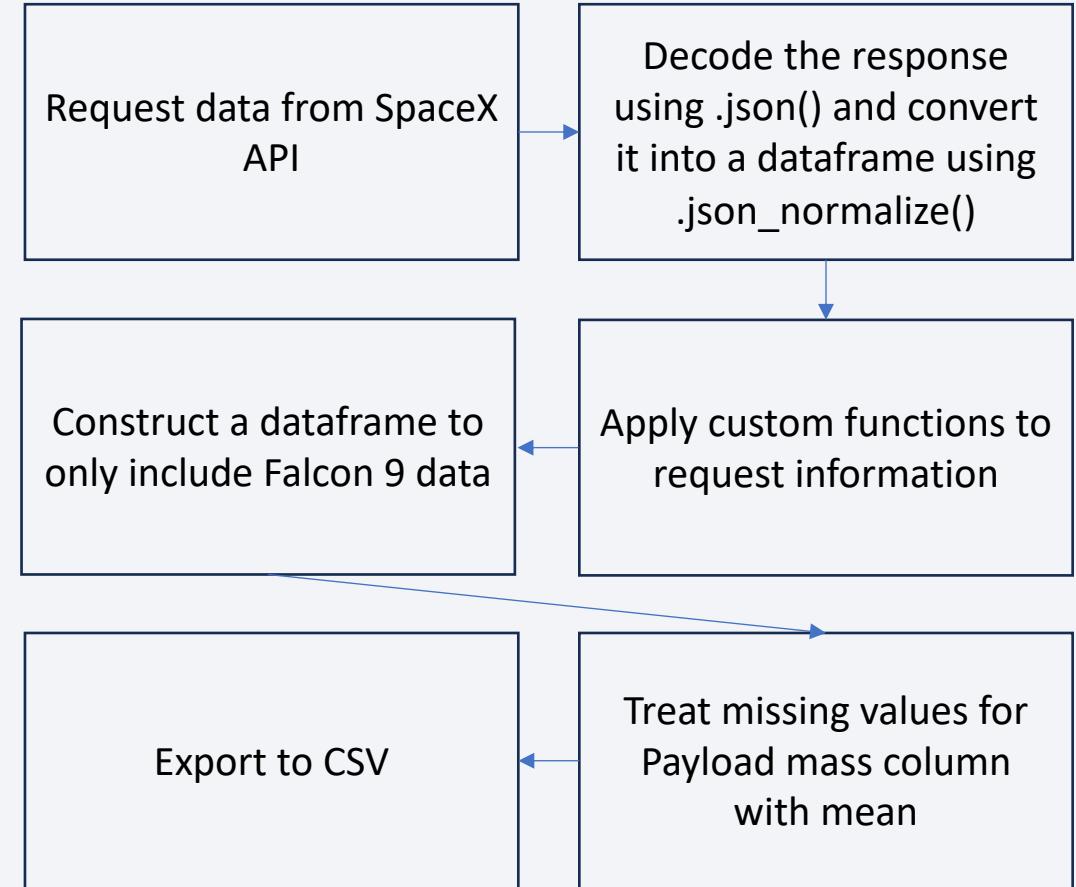
From SpaceX REST API the following data items were collected: FlightNumber, Date, BoosterVersion, PayloadMass, Orbit, LaunchSite, Outcome, Flights, GridFins, Reused, Legs, LandingPad, Block, ReusedCount, Serial, Longitude, Latitude

From HTML Web Scraping the following data were collected: Flight No., Launch site, Payload, PayloadMass, Orbit, Customer, Launch outcome, Version Booster, Booster landing, Date, Time

# Data Collection – SpaceX API

---

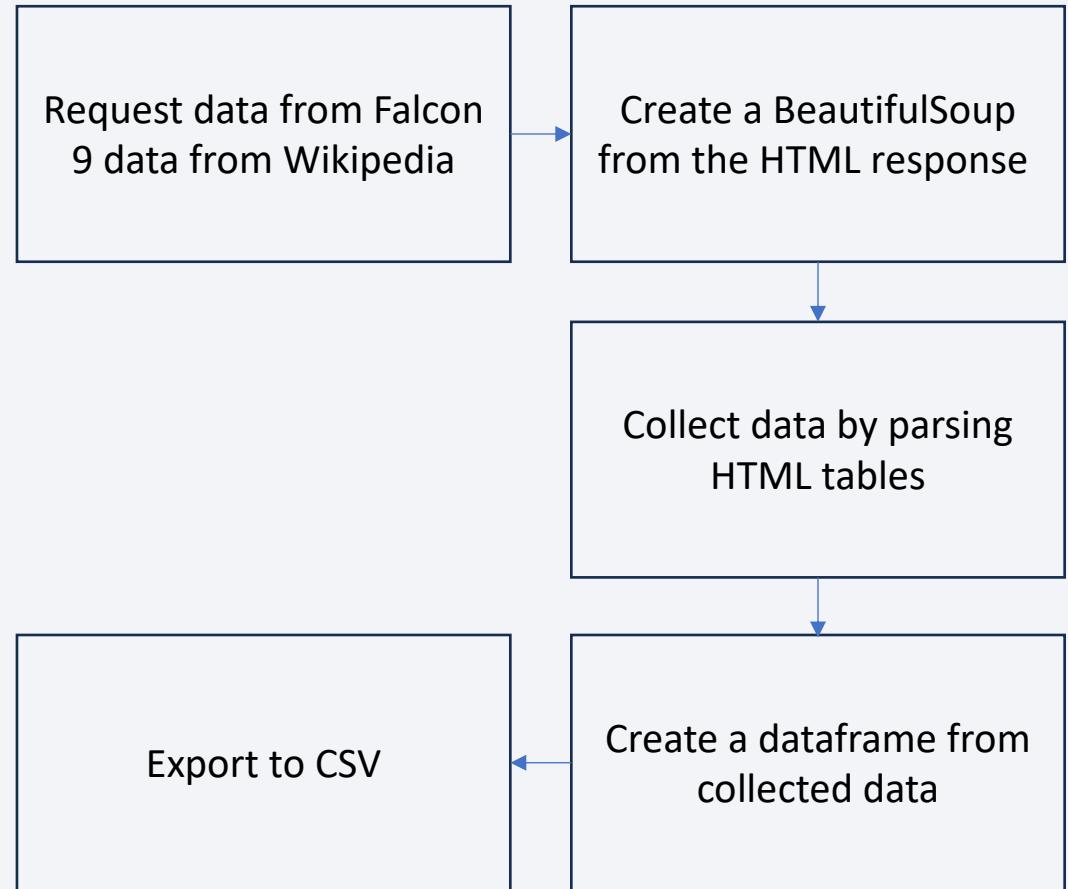
[Link to Github](#)



# Data Collection - Scraping

---

[Link to Github](#)

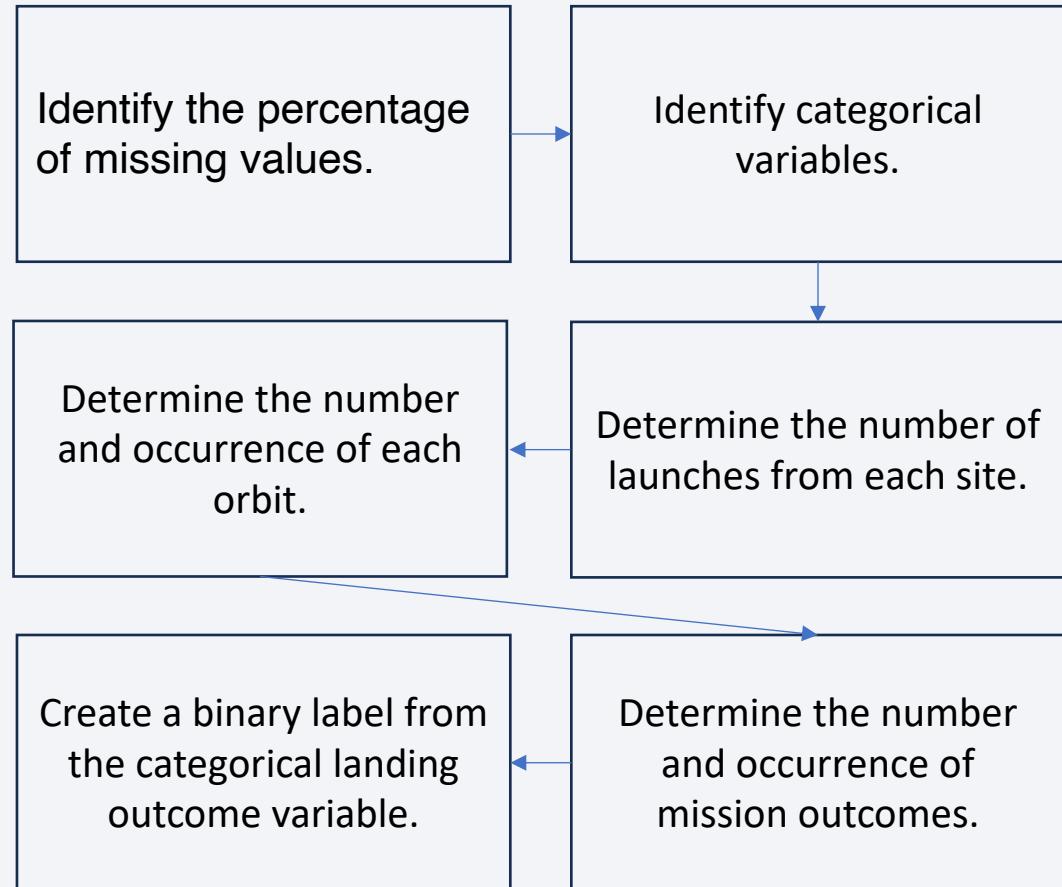


# Data Wrangling

---

The main task in this part was to calculate the frequency and percentages of null variables, identify the variable types, calculate the number and occurrence of main mission outcomes, and label the landing outcomes data in “Outcome” to a binary ‘Class’ variable to present 0 for failure and 1 for success.

[Link to Github](#)



# EDA with Data Visualization

---

In this task, scatter plots, bar charts, and trend line charts were plotted to illustrate the relationships between variables: Flight Number vs. Payload Mass (scatterplot), Flight Number vs. Launch Site (scatterplot, Payload Mass vs. Launch Site (scatterplot), Orbit Type vs. Success Rate (bar chart), Flight Number vs. Orbit Type (scatterplot), Payload Mass vs Orbit Type (scatterplot), and Success Rate (trend).

[Link to Github](#)

# EDA with SQL

---

The following tasks were completed in this part:

- Display the names of the unique launch sites in the space mission
- Display 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 successful landing outcome in ground pad was achieved
- 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
- List the failed landing outcomes, their booster versions and launch site names for 2015
- Rank the count of landing outcomes between 2010-06-04 and 2017-03-20 in descending order.

# Build an Interactive Map with Folium

---

In this task, the geographical location of the launch sites were illustrated on the folium map with the text, circles and markers, using their geographical coordinates (latitude and longitude). Also, proximity of the launch sites to Equator and coastlines were explored. Also, colored markers (green for success, red for failure) were used to illustrate and identify the launch sites with high success rates. Finally, colored Lines were added to show distances between a launch site and its proximities (Railway, Highway, Coastline and Closest City). More specifically, the geographical map of the launch site KSC LC-39A was marked with the lines that connected it to the closest railway, highway and city (Titusville), and coastline.

[Link to Github](#)

# Build a Dashboard with Plotly Dash

---

This task involved constructing a launch sites dropdown list. This was completed by adding a dropdown list to enable launch site selection. Then, a pie chart was constructed to show success launches rates for different launch sites. This pie chart showed the total successful launches count for all sites and the success vs. failed counts for specific launch site selections. This task involved constructing a launch sites dropdown list.

[Link to Github](#)

# Predictive Analysis (Classification)

---

In this task, data were split into training and test datasets based on 0.8/0.2 ratio. Then, different models were used to fit the data. The models included logistic regression, tree classifier, SVM, and K-nearest neighbor (KNN) classifiers. Finally, the best hyperparameters and the accuracy of the models were determined.

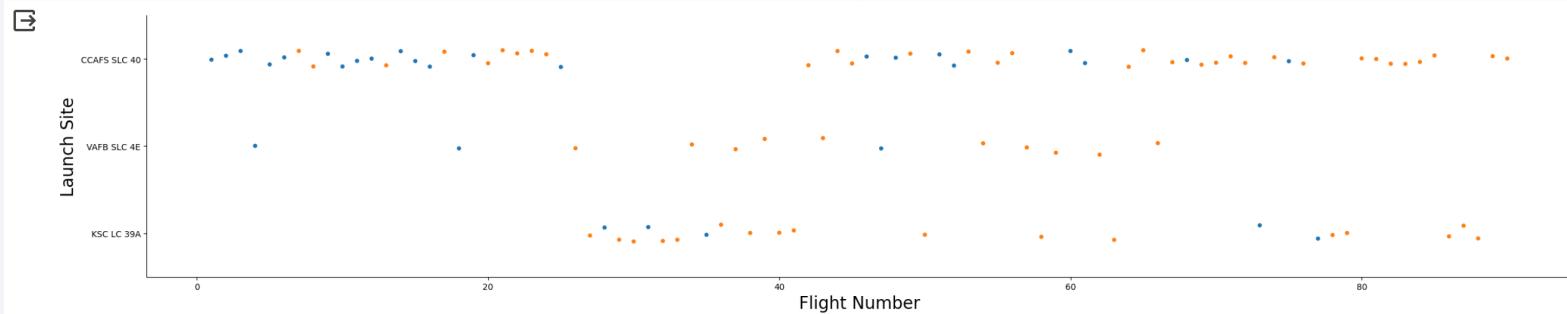
[Link to Github](#)

# Results - Exploratory data analysis

## TASK 1: Visualize the relationship between Flight Number and Launch Site

Use the function `catplot` to plot `FlightNumber` VS `LaunchSite`, set the parameter `x` parameter to `FlightNumber`, set the `y` to `LaunchSite` and set the parameter `hue` to 'class'

```
▶ # Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to be the class value  
sns.catplot(y="LaunchSite", x="FlightNumber", hue="Class", data=df, aspect = 5)  
plt.xlabel("Flight Number", fontsize=20)  
plt.ylabel("Launch Site", fontsize=20)  
plt.show()
```

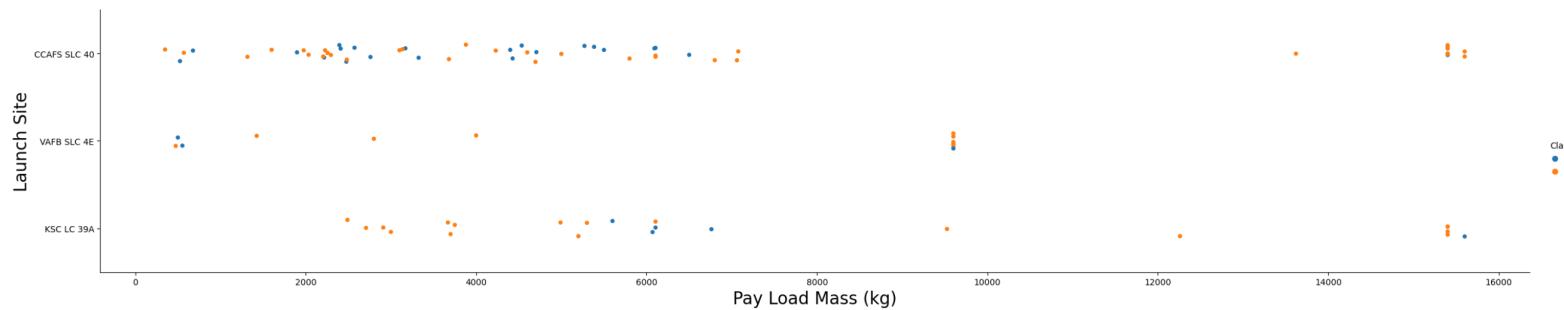


# Results - Exploratory data analysis

TASK 2: Visualize the relationship between Payload and Launch Site

We also want to observe if there is any relationship between launch sites and their payload mass.

```
[ ] # Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch site, and hue to be the class value  
sns.catplot(y="LaunchSite", x="PayloadMass", hue="Class", data=df, aspect = 5)  
plt.xlabel("Pay Load Mass (kg)", fontsize=20)  
plt.ylabel("Launch Site", fontsize=20)  
plt.show()
```



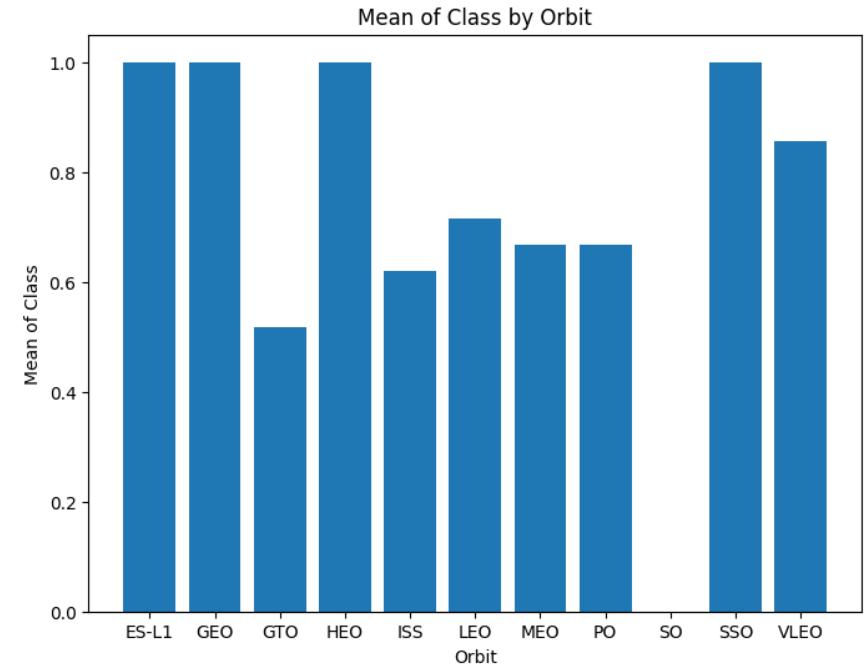
# Results - Exploratory data analysis

## TASK 3: Visualize the relationship between success rate of each orbit type

Next, we want to visually check if there are any relationship between success rate and orbit type.

Let's create a bar chart for the sucess rate of each orbit

```
▶ # HINT use groupby method on Orbit column and get the mean of Class column  
# Group by 'Orbit' and calculate the mean of 'Class'  
orbit_means = df.groupby('Orbit')['Class'].mean().reset_index()  
  
# Create a bar chart to visualize the means  
plt.figure(figsize=(8, 6))  
plt.bar(orbit_means['Orbit'], orbit_means['Class'])  
plt.xlabel('Orbit')  
plt.ylabel('Mean of Class')  
plt.title('Mean of Class by Orbit')  
plt.show()
```

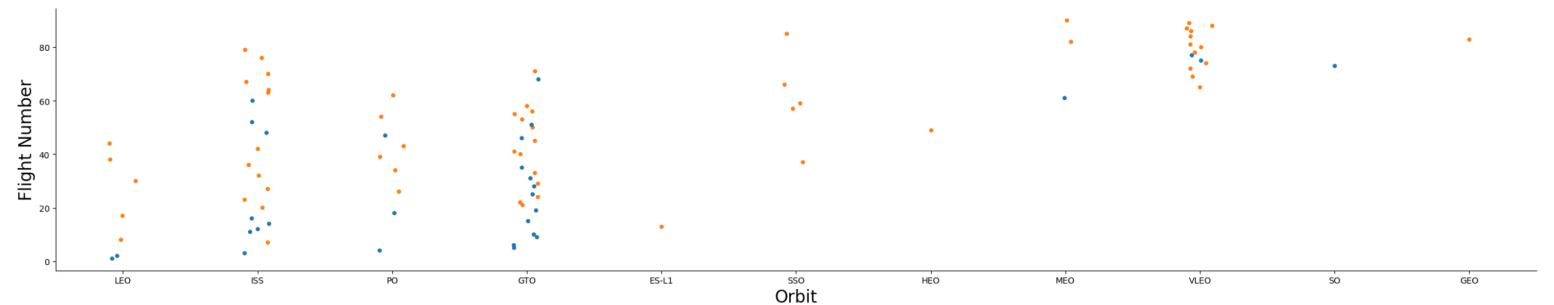


# Results - Exploratory data analysis

TASK 4: Visualize the relationship between FlightNumber and Orbit type

For each orbit, we want to see if there is any relationship between FlightNumber and Orbit type.

```
[ ] # Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class value  
sns.catplot(y="FlightNumber", x="Orbit", hue="Class", data=df, aspect = 5)  
plt.xlabel("Orbit", fontsize=20)  
plt.ylabel("Flight Number", fontsize=20)  
plt.show()
```

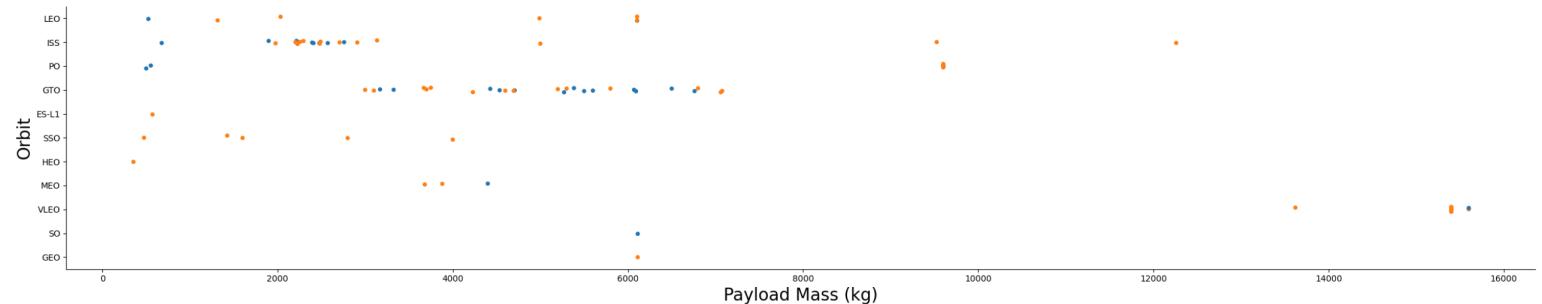


# Results - Exploratory data analysis

TASK 5: Visualize the relationship between Payload and Orbit type

Similarly, we can plot the Payload vs. Orbit scatter point charts to reveal the relationship between Payload and Orbit type

```
[ ] # Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, and hue to be the class value
sns.catplot(x = 'PayloadMass', y = 'Orbit', hue = 'Class', data = df, aspect = 5)
plt.xlabel('Payload Mass (kg)', fontsize = 20)
plt.ylabel('Orbit', fontsize = 20)
plt.show()
```



# Results - Exploratory data analysis

## TASK 6: Visualize the launch success yearly trend

You can plot a line chart with x axis to be `Year` and y axis to be average success rate, to get the average launch success trend.

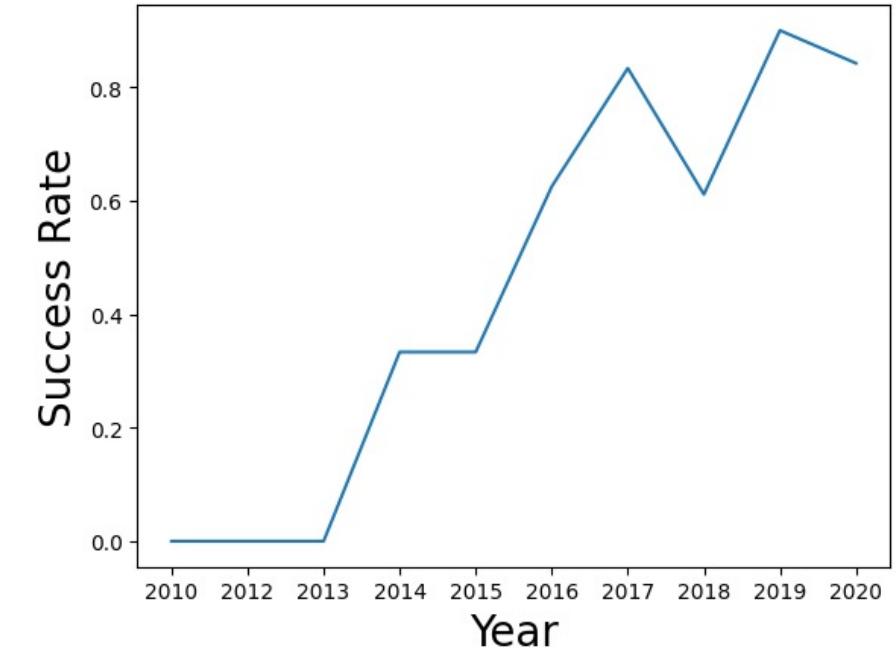
The function will help you get the year from the date:

```
[ ] # A function to Extract years from the date
year=[]
def Extract_year(date):
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year
Year1 = df[year]

[ ] # Plot a line chart with x axis to be the extracted year and y axis to be the success rate
years = df.groupby(Extract_year(df['Date'])).mean()['Class']

# Create a line chart to visualize the trend

sns.lineplot(x = years.index, y = years)
plt.xlabel('Year', fontsize = 20)
plt.ylabel('Success Rate', fontsize = 20)
plt.show()
```



# Results - Exploratory data analysis with SQL

## Task 1

Display the names of the unique launch sites in the space mission

```
[ ] %sql select distinct launch_site from SPACEXTABLE;  
* sqlite:///my_data1.db  
Done.  
Launch_Site  
CCAFS LC-40  
VAFB SLC-4E  
KSC LC-39A  
CCAFS SLC-40
```

## Task 2

Display 5 records where launch sites begin with the string 'CCA'

```
[ ] %sql select * from SPACEXTABLE where launch_site like 'CCA%' limit 5;  
* sqlite:///my_data1.db  
Done.  


| 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) | 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          |


```

# Results - Exploratory data analysis with SQL

## ▼ Task 3

Display the total payload mass carried by boosters launched by NASA (CRS)

```
[ ] *sql select sum(payload_mass_kg_) as total_payload_mass from SPACEXTABLE where customer = 'NASA (CRS)';

* sqlite:///my_data1.db
Done.
total_payload_mass
45596
```

## ▼ Task 4

Display 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%';

* sqlite:///my_data1.db
Done.
average_payload_mass
2534.6666666666665
```

## Task 5

List the date when the first succesful landing outcome in ground pad was acheived.

*Hint:Use min function*

```
[ ] *sql select min(date) as first_successful_landing from SPACEXTABLE where Mission_Outcome = 'Success';

* sqlite:///my_data1.db
Done.
first_successful_landing
2010-04-06
```

## Task 6

List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000

```
[ ] *sql select booster_version from SPACEXTABLE where Mission_Outcome = 'Success (drone ship)' and payload_mass_kg_ between 4000 and 6000;

* sqlite:///my_data1.db
Done.
Booster_Version
```

# Results - Exploratory data analysis with SQL

## ▼ Task 7

List the total number of successful and failure mission outcomes

```
[ ] %sql select Mission_Outcome, count(*) as total_number from SPACEXTABLE group by Mission_Outcome;  
* sqlite:///my_data1.db  
Done.  


| Mission_Outcome                  | total_number |
|----------------------------------|--------------|
| Failure (in flight)              | 1            |
| Success                          | 98           |
| Success                          | 1            |
| Success (payload status unclear) | 1            |


```

## ▼ Task 8

List the names of the booster\_versions which have carried the maximum payload mass. Use a subquery

```
▶ %sql select booster_version from SPACEXTABLE where payload_mass_kg_ = (select max(payload_mass_kg_) from SPACEXTABLE);  
* sqlite:///my_data1.db  
Done.  


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


```

# Results - Exploratory data analysis with SQL

## ▼ Task 9

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.

**Note: SQLite does not support monthnames. So you need to use substr(Date, 6,2) as month to get the months and substr(Date,0,5)='2015' for year.**

```
▶ %%sql select monthname(date) as month, date, booster_version, launch_site, Mission_Outcome from SPACEXTABLE  
      where Mission_Outcome = 'Failure (drone ship)' and year(date)=2015;
```

```
⠼ * sqlite:///my_data1.db  
    (sqlite3.OperationalError) no such function: monthname  
    [SQL: select monthname(date) as month, date, booster_version, launch_site, Mission_Outcome from SPACEXTABLE  
         where Mission_Outcome = 'Failure (drone ship)' and year(date)=2015;]  
    (Background on this error at: http://sqlalch.me/e/e3q8)
```

## ▼ Task 10

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 Mission_Outcome, count(*) as count_outcomes from SPACEXTABLE  
      where date between '2010-06-04' and '2017-03-20'  
      group by Mission_Outcome  
      order by count_outcomes desc;
```

```
* sqlite:///my_data1.db  
Done.  


| Mission_Outcome     | count_outcomes |
|---------------------|----------------|
| Success             | 31             |
| Failure (in flight) | 1              |


```

# Results - Exploratory data analysis with SQL

## ▼ Task 9

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.

**Note: SQLite does not support monthnames. So you need to use substr(Date, 6,2) as month to get the months and substr(Date,0,5)='2015' for year.**

```
▶ %%sql select monthname(date) as month, date, booster_version, launch_site, Mission_Outcome from SPACEXTABLE  
      where Mission_Outcome = 'Failure (drone ship)' and year(date)=2015;
```

```
✖ * sqlite:///my_data1.db  
  (sqlite3.OperationalError) no such function: monthname  
  [SQL: select monthname(date) as month, date, booster_version, launch_site, Mission_Outcome from SPACEXTABLE  
      where Mission_Outcome = 'Failure (drone ship)' and year(date)=2015;]  
  (Background on this error at: http://sqlalch.me/e/e3q8)
```

## ▼ Task 10

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 Mission_Outcome, count(*) as count_outcomes from SPACEXTABLE  
      where date between '2010-06-04' and '2017-03-20'  
      group by Mission_Outcome  
      order by count_outcomes desc;
```

```
* sqlite:///my_data1.db  
Done.  


| Mission_Outcome     | count_outcomes |
|---------------------|----------------|
| Success             | 31             |
| Failure (in flight) | 1              |

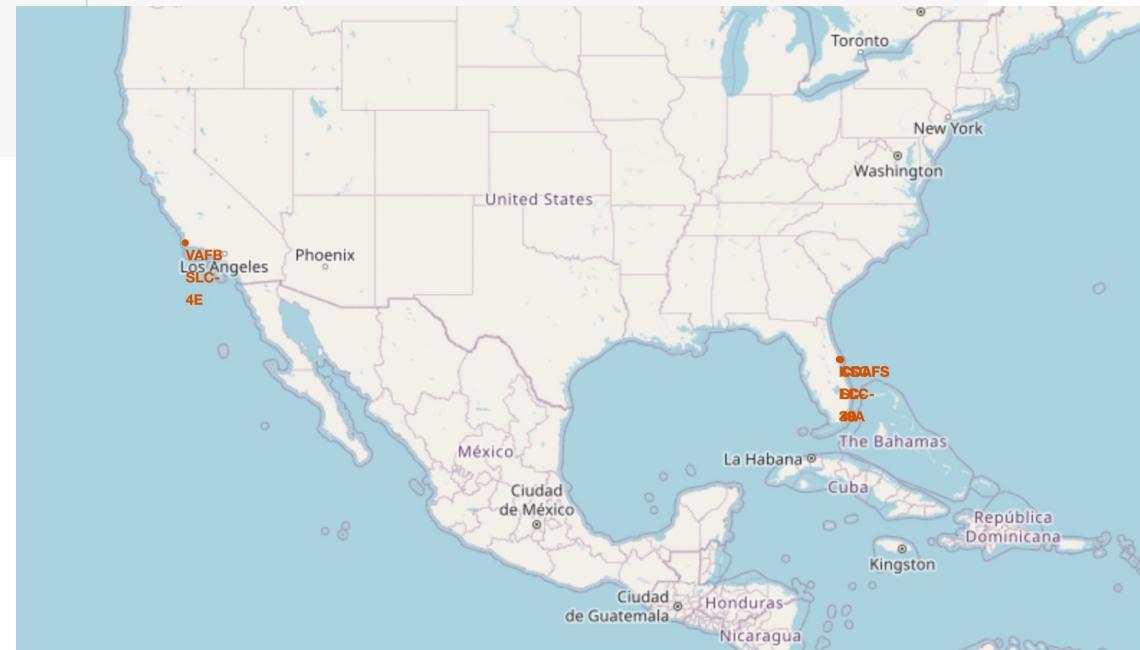

```

# Results - Interactive analytics demo in screenshots

```
▶ # 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
for launch_site, site_lat, site_long in zip(launch_sites_df['Launch Site'], launch_sites_df['Lat'], launch_sites_df['Long']):
    site_coordinate = [site_lat, site_long]

    circle = folium.Circle(site_coordinate, radius=1000, color='#d35400', fill=True).add_child(folium.Popup(launch_site))

    marker = folium.map.Marker(
        site_coordinate,
        # Create an icon as a text label
        icon=DivIcon(
            icon_size=(20, 20),
            icon_anchor=(0, 0),
            html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % launch_site,
        )
    )
    site_map.add_child(circle)
    site_map.add_child(marker)
site_map
```



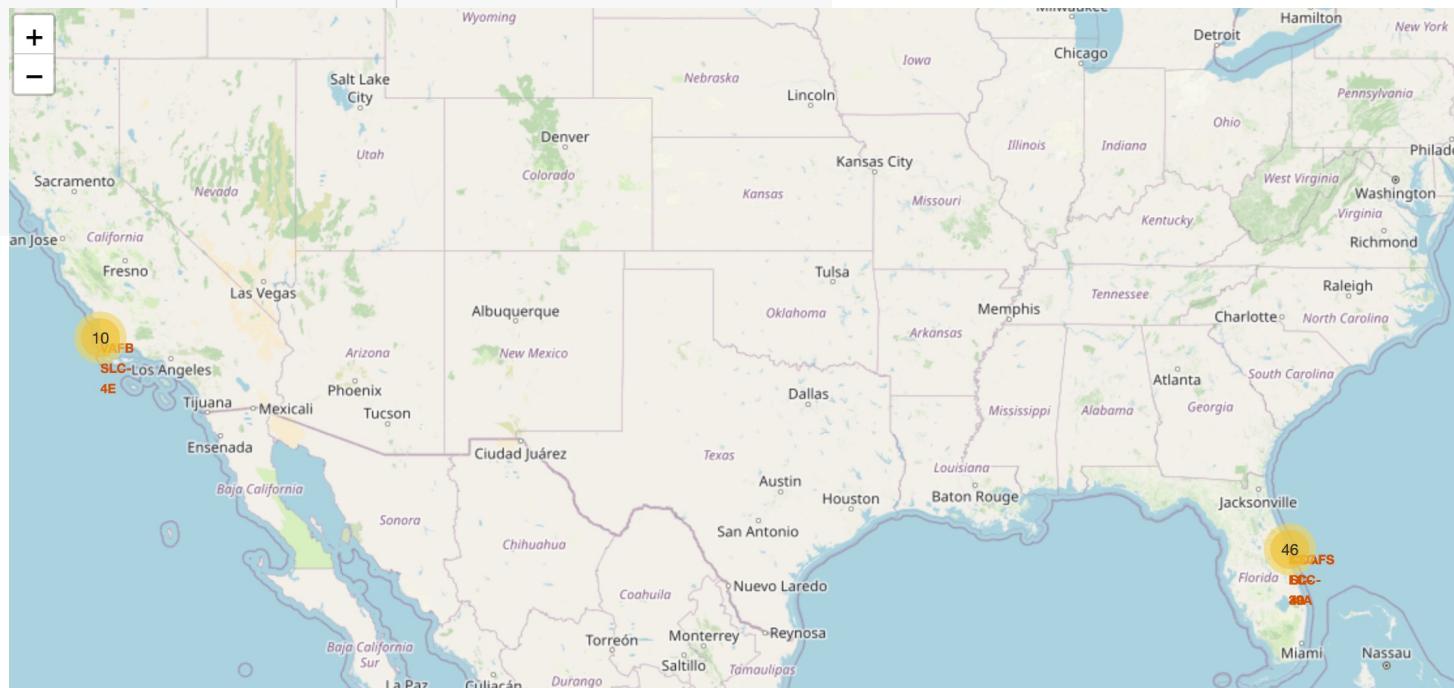
# Results - Interactive analytics demo in screenshots

TODO: For each launch result in `spacex_df` data frame, add a `folium.Marker` to `marker_cluster`

```
[ ] # 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 site_lat, site_long, marker_color in zip(spacex_df['Lat'], spacex_df['Long'], spacex_df['marker_color']):
    site_coordinate = [site_lat, site_long]
    marker = folium.map.Marker(
        site_coordinate,
        # Create an icon as a text label
        icon=folium.Icon(color='white',
                          icon_color=marker_color)
    )
    marker.add_to(marker_cluster)

site_map
```



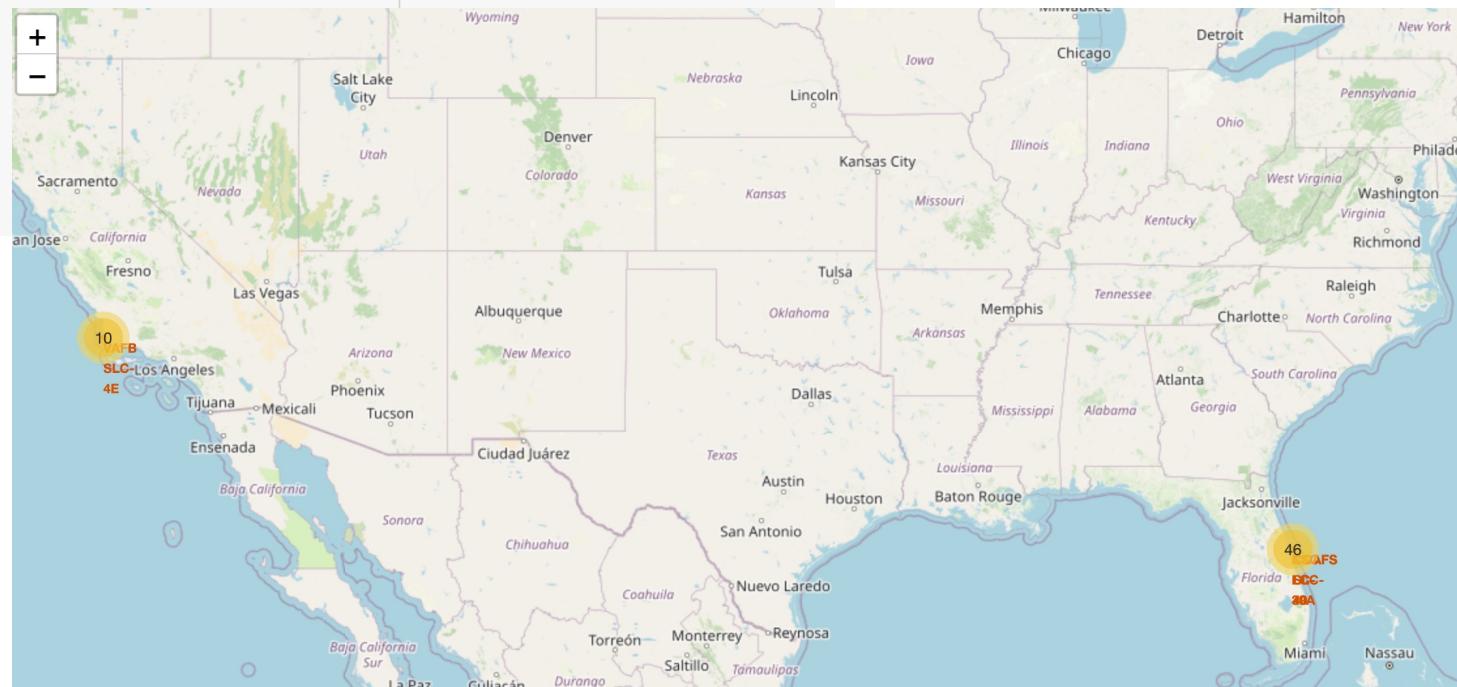
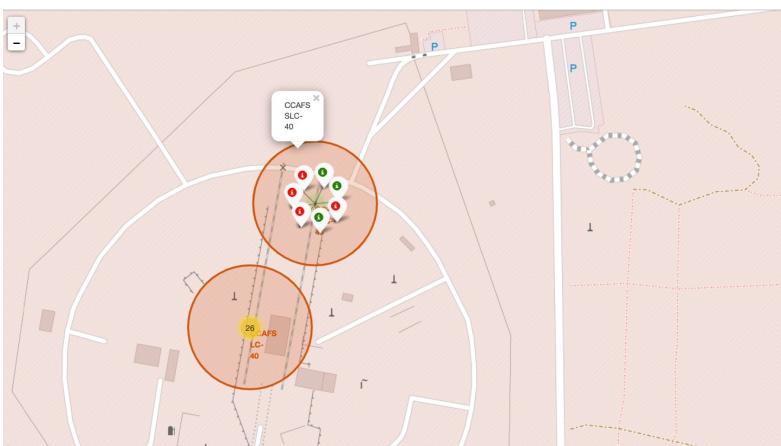
# Results - Interactive analytics demo in screenshots

TODO: For each launch result in `spacex_df` data frame, add a `folium.Marker` to `marker_cluster`

```
[ ] # 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 site_lat, site_long, marker_color in zip(spacex_df['Lat'], spacex_df['Long'], spacex_df['marker_color']):
    site_coordinate = [site_lat, site_long]
    marker = folium.map.Marker(
        site_coordinate,
        # Create an icon as a text label
        icon=folium.Icon(color='white',
                          icon_color=marker_color)
    )
    marker.add_to(marker_cluster)

site_map
```



# Results - Interactive analytics demo in screenshots

```
[ ] # find coordinate of the closest coastline
# e.g.: Lat: 28.56367 Lon: -80.57163
# distance_coastline = calculate_distance(launch_site_lat, launch_site_lon, coastline_lat, coastline_lon)
coastline_marker = [28.56293, -80.56803]
launch_coordinate = [28.57337, -80.64669]
distance_coastline = calculate_distance(coastline_marker[0], coastline_marker[1], launch_coordinate[0], launch_coordinate[1])
distance_coastline # distance in km

7.771346579504573
```

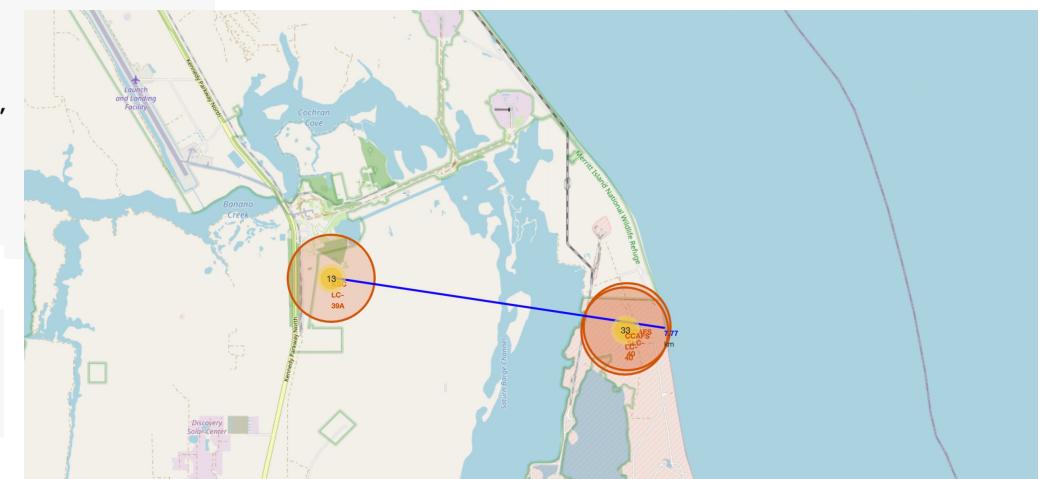
TODO: After obtained its coordinate, create a folium.Marker to show the distance

```
▶ # Create and add a folium.Marker on your selected closest coastline point on the map
# Display the distance between coastline point and launch site using the icon property
# for example
# distance_marker = folium.Marker(
#     coordinate,
#     icon=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),
#     )
# )
marker = folium.map.Marker(
    coastline_marker,
    # Create an icon as a text label
    icon=DivIcon(
        icon_size=(400, 400),
        icon_anchor=(0, 0),
        html='<div style="font-size:400; color:#0c10f2;"><b>%s</b></div>' % str(round(distance_coastline, 2))+' km',
    )
)
marker.add_to(site_map)

site_map
```

TODO: Draw a PolyLine between a launch site to the selected coastline point

```
▶ # Create a `folium.PolyLine` object using the coastline coordinates and launch site coordinate
# lines=folium.PolyLine(locations=coordinates, weight=1)
folium.PolyLine([coastline_marker, launch_coordinate], color='blue').add_to(site_map)
site_map
```



# Results - Interactive analytics demo in screenshots

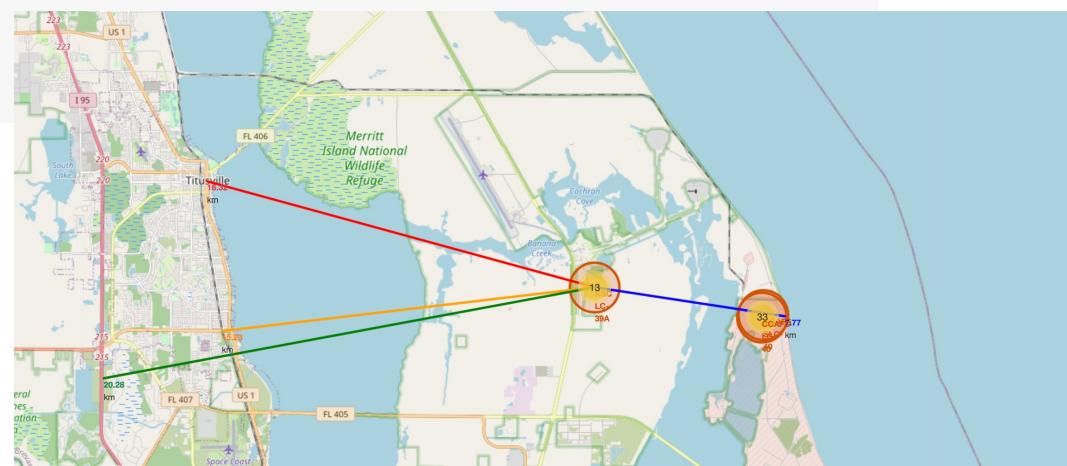


```
# Create a marker with distance to a closest city, railway, highway, etc.
# Draw a line between the marker to the launch site
city    = [28.61200, -80.80788]
railway = [28.55752, -80.80155]
highway = [28.5402, -80.85079]

city_distance = calculate_distance(city[0], city[1], launch_coordinate[0], launch_coordinate[1])
railway_distance = calculate_distance(railway[0], railway[1], launch_coordinate[0], launch_coordinate[1])
highway_distance = calculate_distance(highway[0], highway[1], launch_coordinate[0], launch_coordinate[1])

colors = ['red','orange','green']
html_colors = ['#dc3545','#fd7e14','#198754']

for coordinate ,distance, color, html_color in zip([city, railway, highway], [city_distance, railway_distance, highway_distance], colors, html_colors):
    marker = folium.map.Marker(
        coordinate,
        # Create an icon as a text label
        icon=DivIcon(
            icon_size=(20,20),
            icon_anchor=(0,0),
            html='<div style="font-size: 12; color:' +html_color+';"><b>%s</b></div>' % str(round(distance, 2)) + 'km',
        )
    )
    marker.add_to(site_map)
    folium.PolyLine([coordinate, launch_coordinate], color=color).add_to(site_map)
site_map
```



# Results - Exploratory data analysis with Dashboards

```
[ ] # Create an app layout
app.layout = html.Div(children=[html.H1('SpaceX Launch Records Dashboard',
                                         style={'textAlign': 'center', 'color': '#503D36',
                                                 'font-size': 40}),
# TASK 1: Add a dropdown list to enable Launch Site selection
# The default select value is for ALL sites
dcc.Dropdown(id='site-dropdown', options=[{'label': 'All Sites', 'value': 'All Sites'}, {'label': 'CCAFS LC-40', 'value': 'CCAFS LC-40'}, {'label': 'VAFB SLC-4E', 'value': 'VAFB SLC-4E'}, {'label': 'KSC LC-39A', 'value': 'KSC LC-39A'}, {'label': 'CCAFS SLC-40', 'value': 'CCAFS SLC-40'}]),
# TASK 2: Add a pie chart to show the total successful launches count for all sites
# If a specific launch site was selected, show the Success vs. Failed counts for the site
html.Div(dcc.Graph(id='success-pie-chart')),
html.Br(),
html.P("Payload range (Kg):"),
# TASK 3: Add a slider to select payload range
dcc.RangeSlider(id='payload-slider',
min=0,
max=10000,
step=1000,
marks={i: '{}'.format(i) for i in range(0, 10001, 1000)},
value=[min_payload, max_payload]),
# TASK 4: Add a scatter chart to show the correlation between payload and launch success
html.Div(dcc.Graph(id='success-payload-scatter-chart')),
])
```



# Results - Predictive analysis results

## ▼ TASK 1

Create a NumPy array from the column `Class` in `data`, by applying the method `to_numpy()` then assign it to the variable `y`, make sure the output is a Pandas series (only one bracket `df['name of column']`).

```
[ ] y = pd.Series(data['Class'].to_numpy())
y.head(10)
```

```
0    0
1    0
2    0
3    0
4    0
5    0
6    1
7    1
8    0
9    0
dtype: int64
```

# Results - Predictive analysis results

## ▼ TASK 2

Standardize the data in `x` then reassign it to the variable `x` using the transform provided below.

```
[ ] # students get this  
transform = preprocessing.StandardScaler()  
  
[ ] X = transform.fit(X).transform(X)
```

We split the data into training and testing data using the function `train_test_split`. The training data is divided into validation data, a second set used for training data; then the models are trained and hyperparameters are selected using the function `GridSearchCV`.

## ▼ TASK 3

Use the function `train_test_split` to split the data `X` and `Y` into training and test data. Set the parameter `test_size` to 0.2 and `random_state` to 2. The training data and test data should be assigned to the following labels.

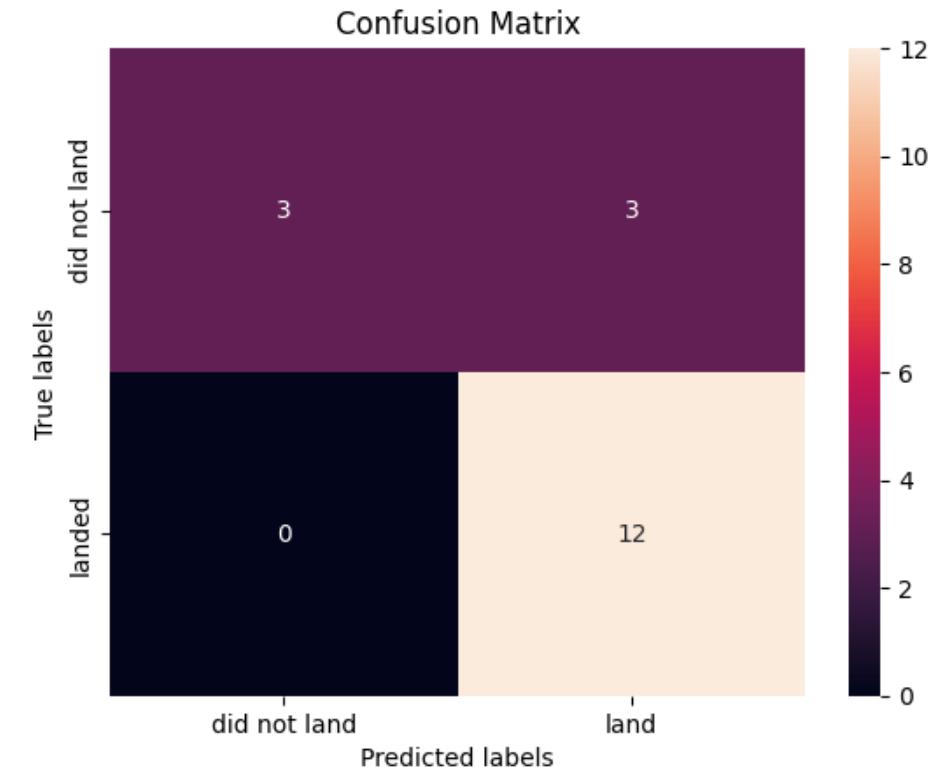
```
X_train, X_test, Y_train, Y_test  
  
[ ] X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
```

# Results - Predictive analysis results

## ▼ TASK 4

Create a logistic regression object then create a GridSearchCV object `logreg_cv` with `cv = 10`. Fit the object to find the best parameters from the dictionary `parameters`.

```
[ ] parameters ={'C':[0.01,0.1,1],  
    'penalty':['l2'],  
    'solver':['lbfgs']}  
  
[ ] parameters ={"C": [0.01, 0.1, 1], 'penalty': ['l2'], 'solver': ['lbfgs']}# 11 lasso 12 ridge  
lr=LogisticRegression()  
  
logreg_cv=GridSearchCV(lr, parameters, cv=10)  
logreg_cv.fit(X_train, Y_train)  
  
▶     GridSearchCV  
▶ estimator: LogisticRegression  
    LogisticRegression
```



## ▼ TASK 5

Calculate the accuracy on the test data using the method `score`:

```
[ ] logreg_accuracy = logreg_cv.score(X_test, Y_test)  
logreg_accuracy  
  
0.8333333333333334
```

Lets look at the confusion matrix:

```
▶ logreg_yhat=logreg_cv.predict(X_test)  
plot_confusion_matrix(Y_test, logreg_yhat)
```

# Results - Predictive analysis results

## TASK 6

Create a support vector machine object then create a `GridSearchCV` object `svm_cv` with `cv = 10`. Fit the object to find the best parameters from the dictionary `parameters`.

```
[ ] parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
                 'C' : np.logspace(-3, 3, 5),
                 'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

```
[ ]  svm_cv = GridSearchCV(svm, param_grid)
      svm_cv.fit(X_train, Y_train)
```

```
print("tuned hyperparameters : (best parameters) ",svm_cv.best_params_)
print("accuracy : ",svm_cv.best_score_)
```

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

▼ TASK 7

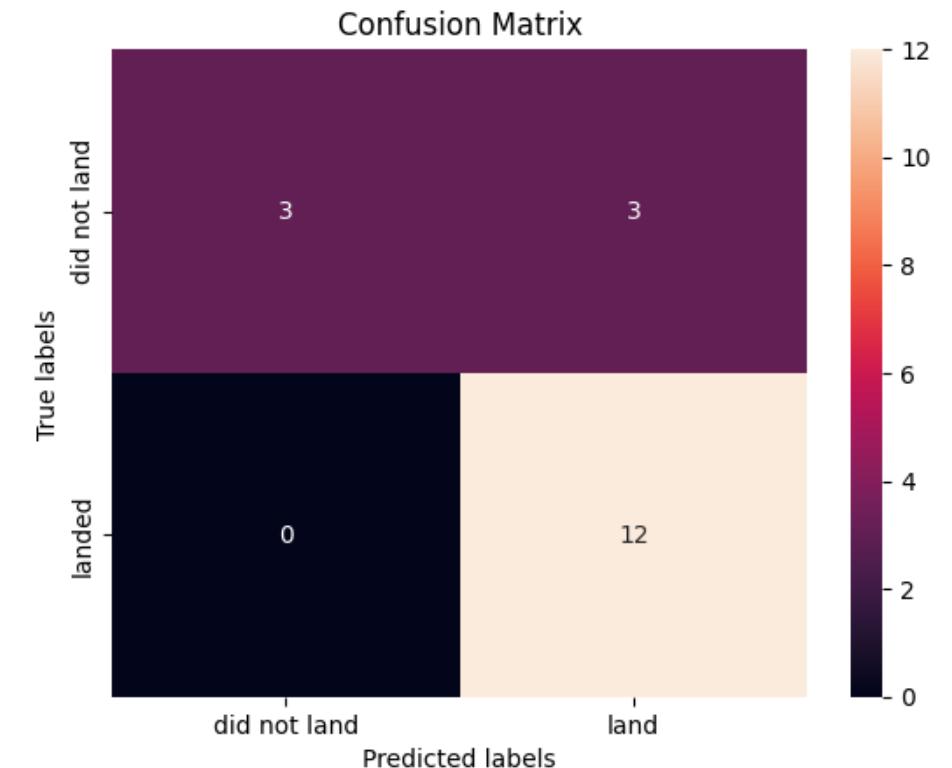
Calculate the accuracy on the test data using the method `score`:

```
[ ] svm_accuracy = svm_cv.score(X_test, Y_test)  
svm_accuracy
```

0.8333333333333334

We can plot the confusion matrix

```
svm_yhat=svm_cv.predict(X_test)  
plot_confusion_matrix(Y_test, svm_yhat)
```



# Results - Predictive analysis results

## TASK 8

Create a decision tree classifier object then create a `GridSearchCV` object `tree_cv` with `cv = 10`. Fit the object to find the best parameters from the dictionary `parameters`.

```
[ ] parameters = {'criterion': ['gini', 'entropy'],
   'splitter': ['best', 'random'],
   'max_depth': [2*n for n in range(1,10)],
   'max_features': ['auto', 'sqrt'],
   'min_samples_leaf': [1, 2, 4],
   'min_samples_split': [2, 5, 10]}

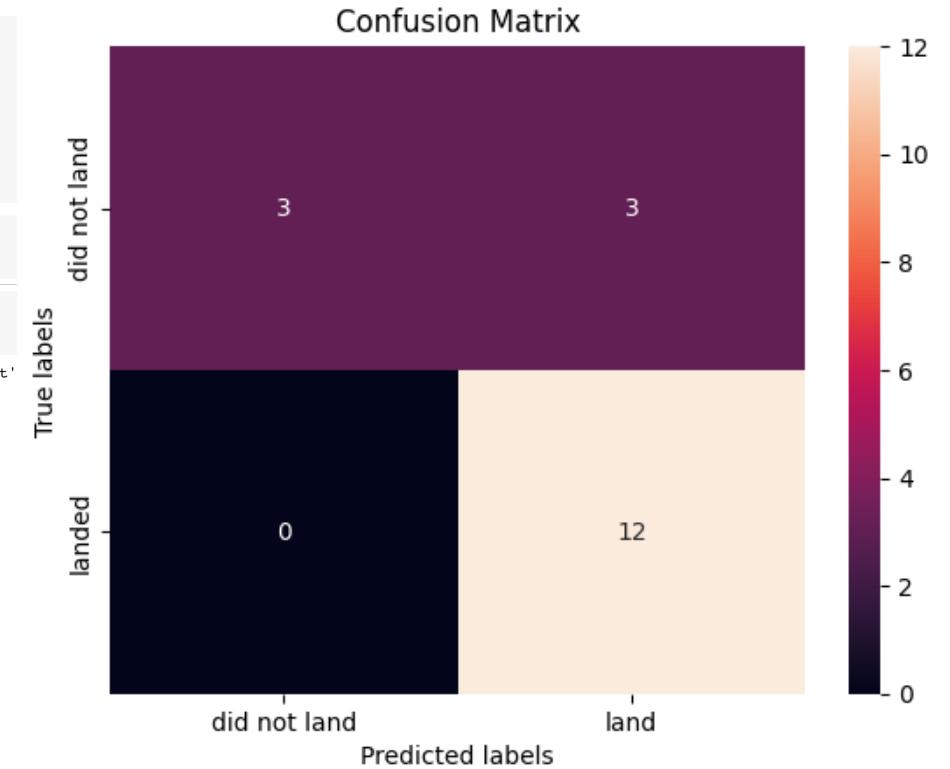
tree = DecisionTreeClassifier()

▶ tree_cv = GridSearchCV(tree, parameters, cv=10)
tree_cv.fit(X_train, Y_train)

[ ] print("tuned hyperparameters : (best parameters) ",tree_cv.best_params_)
print("accuracy : ",tree_cv.best_score_)

tuned hyperparameters : (best parameters)  {'criterion': 'entropy', 'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 4, 'min_samples_split': 2}
accuracy : 0.8732142857142857
```

+ Code + Text



## TASK 9

Calculate the accuracy of `tree_cv` on the test data using the method `score`:

```
[ ] tree_accuracy = tree_cv.score(X_test, Y_test)
tree_accuracy

0.7222222222222222
```

We can plot the confusion matrix

```
▶ tree_yhat = svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,tree_yhat)
```

# Results - Predictive analysis results

## TASK 10

Create a k nearest neighbors object then create a `GridSearchCV` object `knn_cv` with `cv = 10`. Fit the object to find the best parameters from the dictionary `parameters`.

```
[ ] parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],  
                 'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],  
                 'p': [1,2]}\n\nKNN = KNeighborsClassifier()
```

```
▶ knn_cv = GridSearchCV(KNN, parameters, cv=10)  
knn_cv.fit(X_train, Y_train)
```

```
► GridSearchCV  
► estimator: KNeighborsClassifier  
  ► KNeighborsClassifier
```

```
[ ] print("tuned hyperparameters :(best parameters) ",knn_cv.best_params_)\nprint("accuracy : ",knn_cv.best_score_)\n\n tuned hyperparameters :(best parameters) {'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}\naccuracy : 0.8482142857142858
```

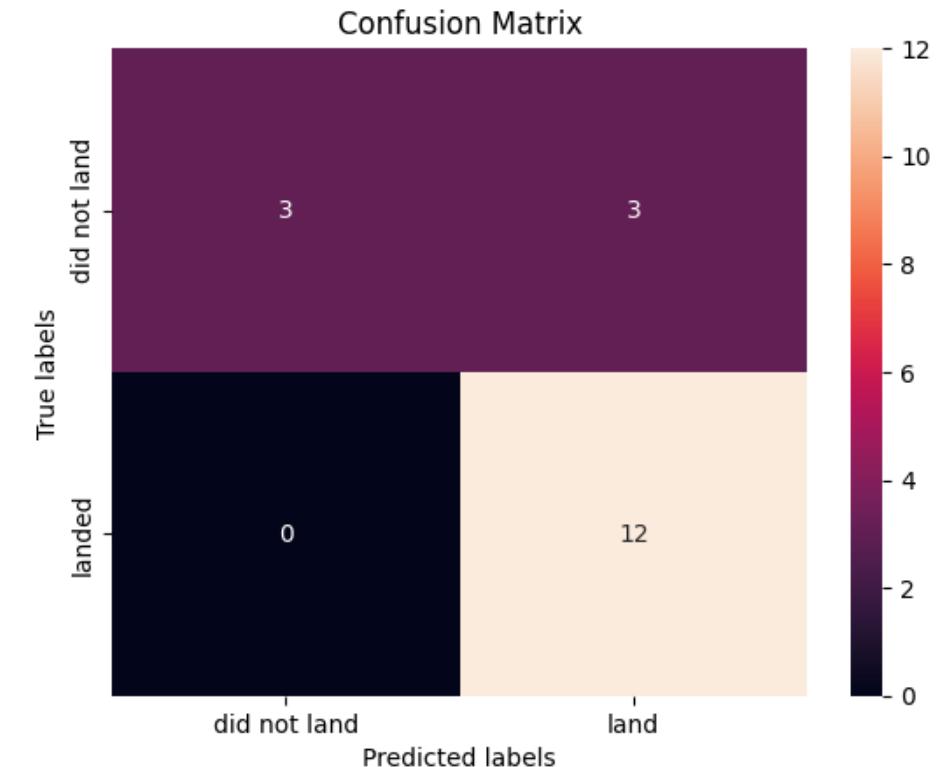
## TASK 11

Calculate the accuracy of `tree_cv` on the test data using the method `score`:

```
[ ] knn_accuracy = knn_cv.score(X_test, Y_test)\n\nknn_accuracy\n\n0.8333333333333334
```

We can plot the confusion matrix

```
▶ knn_yhat = knn_cv.predict(X_test)\nplot_confusion_matrix(Y_test, knn_yhat)
```

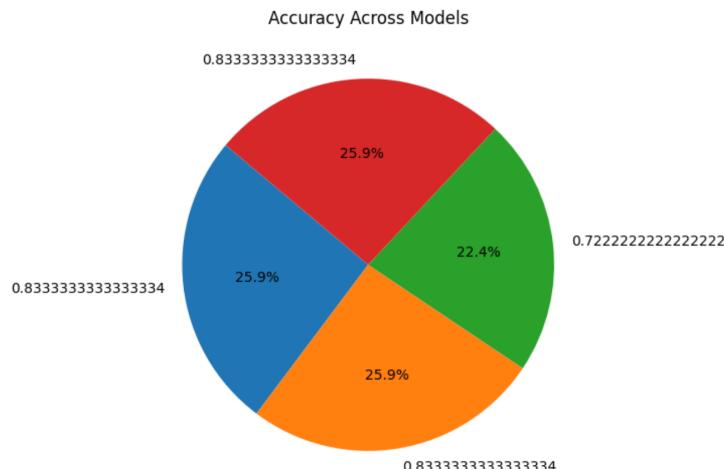


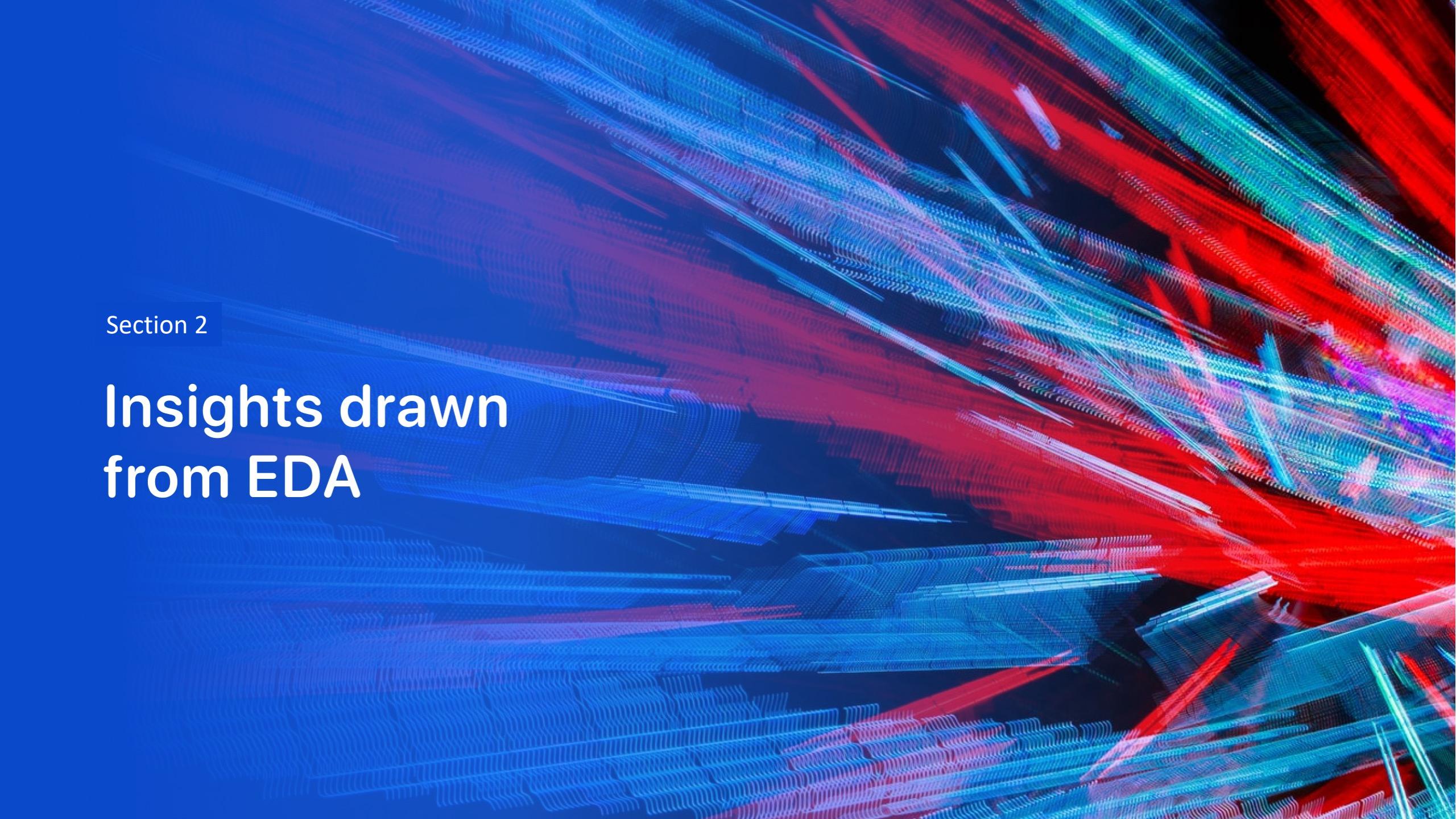
# Results - Predictive analysis results

```
▶ from sklearn.metrics import jaccard_score, f1_score  
  
jaccard_scores = [  
    jaccard_score(Y_test, logreg_yhat, average='binary'),  
    jaccard_score(Y_test, svm_yhat, average='binary'),  
    jaccard_score(Y_test, tree_yhat, average='binary'),  
    jaccard_score(Y_test, knn_yhat, average='binary'),  
]  
  
f1_scores = [  
    f1_score(Y_test, logreg_yhat, average='binary'),  
    f1_score(Y_test, svm_yhat, average='binary'),  
    f1_score(Y_test, tree_yhat, average='binary'),  
    f1_score(Y_test, knn_yhat, average='binary'),  
]  
  
accuracy = [logreg_accuracy, svm_accuracy, tree_accuracy, knn_accuracy]  
  
scores = pd.DataFrame(np.array([jaccard_scores, f1_scores, accuracy]), index=['Jaccard_Score', 'F1_Score', 'Accuracy'], columns=['LogReg', 'SVM', 'Tree', 'KNN'])  
scores
```

	LogReg	SVM	Tree	KNN
Jaccard_Score	0.800000	0.800000	0.800000	0.800000
F1_Score	0.888889	0.888889	0.888889	0.888889
Accuracy	0.833333	0.833333	0.722222	0.833333

```
import matplotlib.pyplot as plt  
  
# Create a pie chart  
plt.figure(figsize=(6, 6)) # Set the figure size  
plt.pie(accuracy, labels=accuracy, autopct='%.1f%%', startangle=140) # 'autopct' adds percentage labels  
plt.title('Accuracy Across Models')  
  
# Display the pie chart  
plt.show()
```

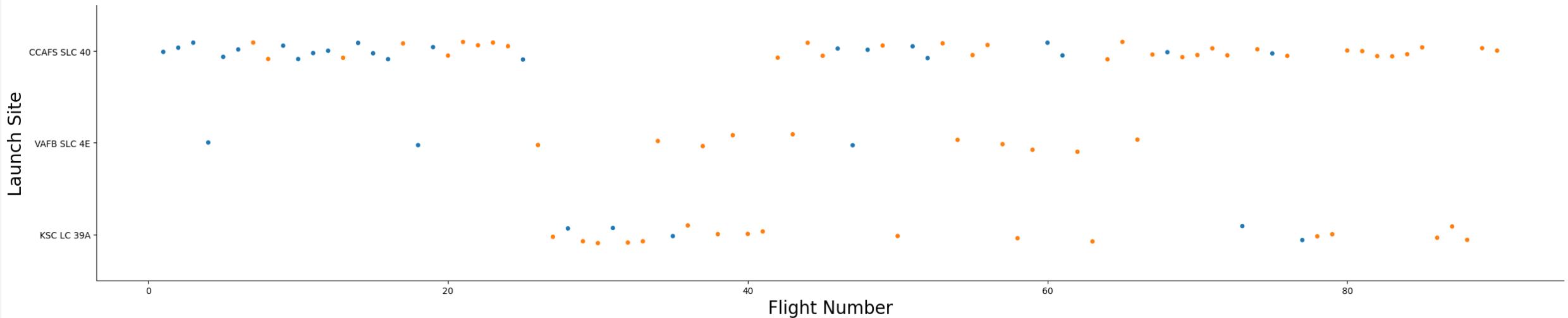


The background of the slide features a complex, abstract digital visualization. It consists of numerous thin, glowing lines that create a sense of depth and motion. The lines are primarily blue and red, with some green and purple highlights. They form a grid-like structure that curves and twists across the frame, resembling a three-dimensional space or a network of data points. The overall effect is futuristic and dynamic.

Section 2

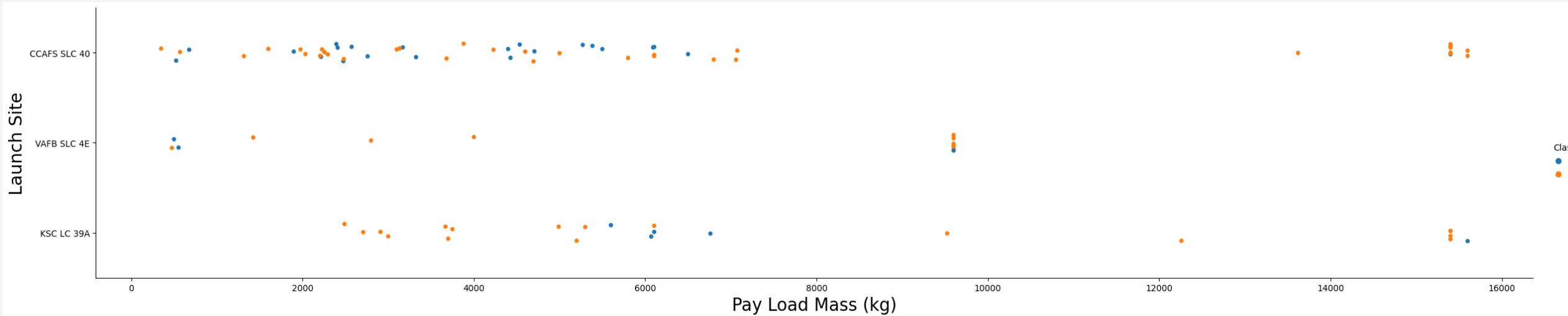
## Insights drawn from EDA

# Flight Number vs. Launch Site



We learn that the increased flight numbers have lead to increased success rate across the launch sites.

# Payload vs. Launch Site

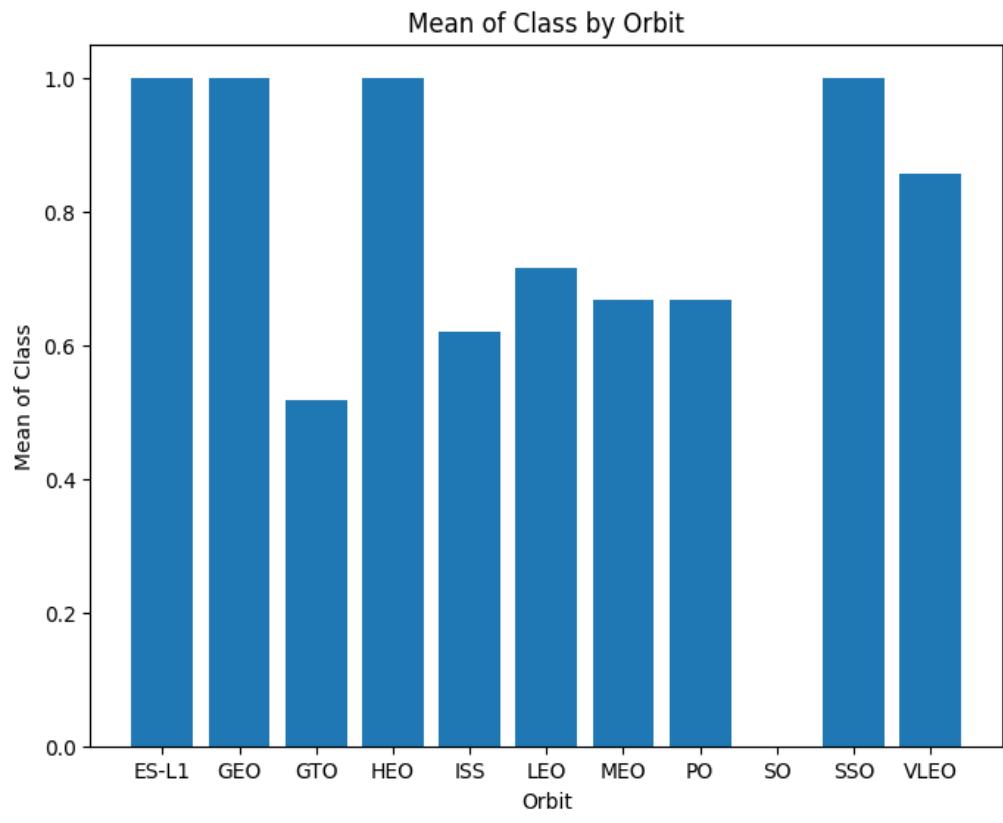


We learn that CCAFS SLC 40 is associated with higher success rates for high payload masses (such as greater than 15,000 kg). It is also clear that KSC LC 39A has experienced smaller failure rates for both small to high pay load masses.

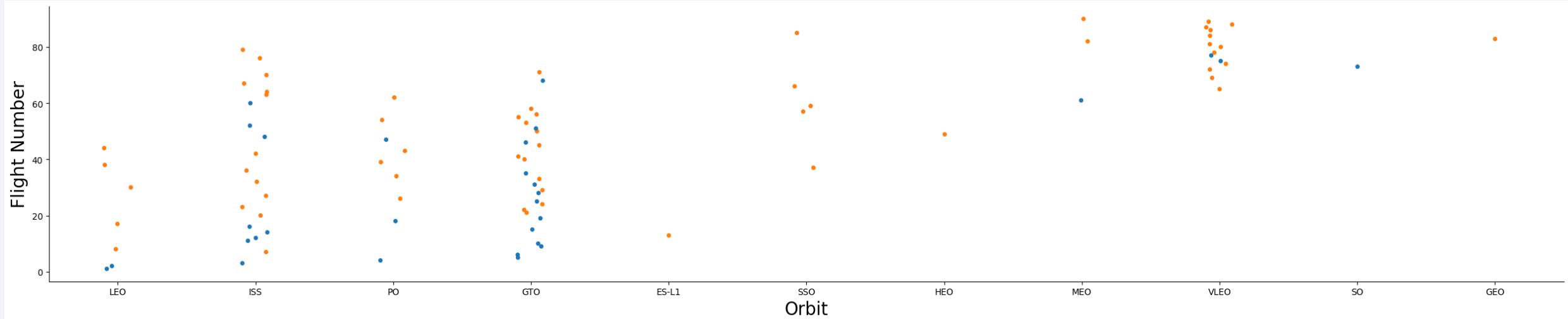
# Success Rate vs. Orbit Type

---

This bar chart shows ES-L1, SSO, GEO and HEO are the orbits associated with the highest success rates.

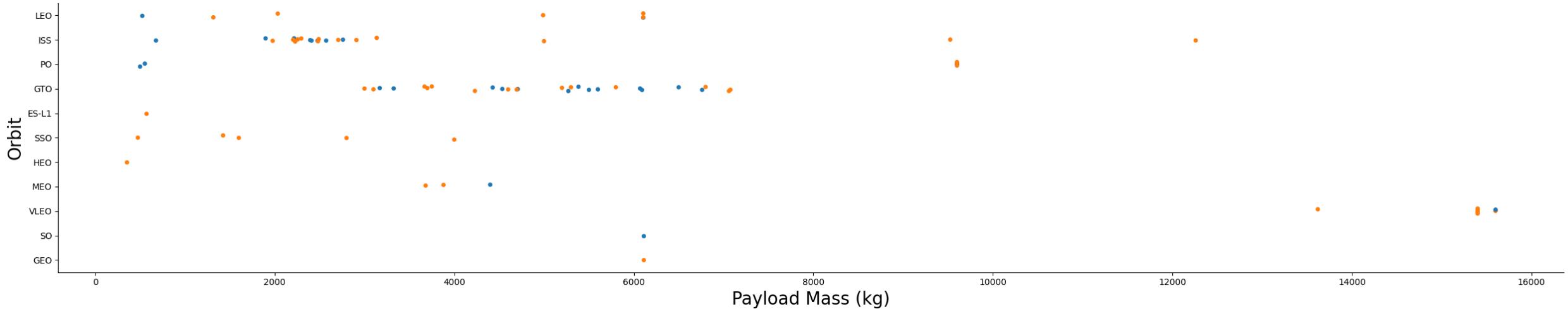


# Flight Number vs. Orbit Type



We learn that VLEO is associated with very high success rates with great flight numbers. The pattern in most other orbits is scattered.

# Payload vs. Orbit Type

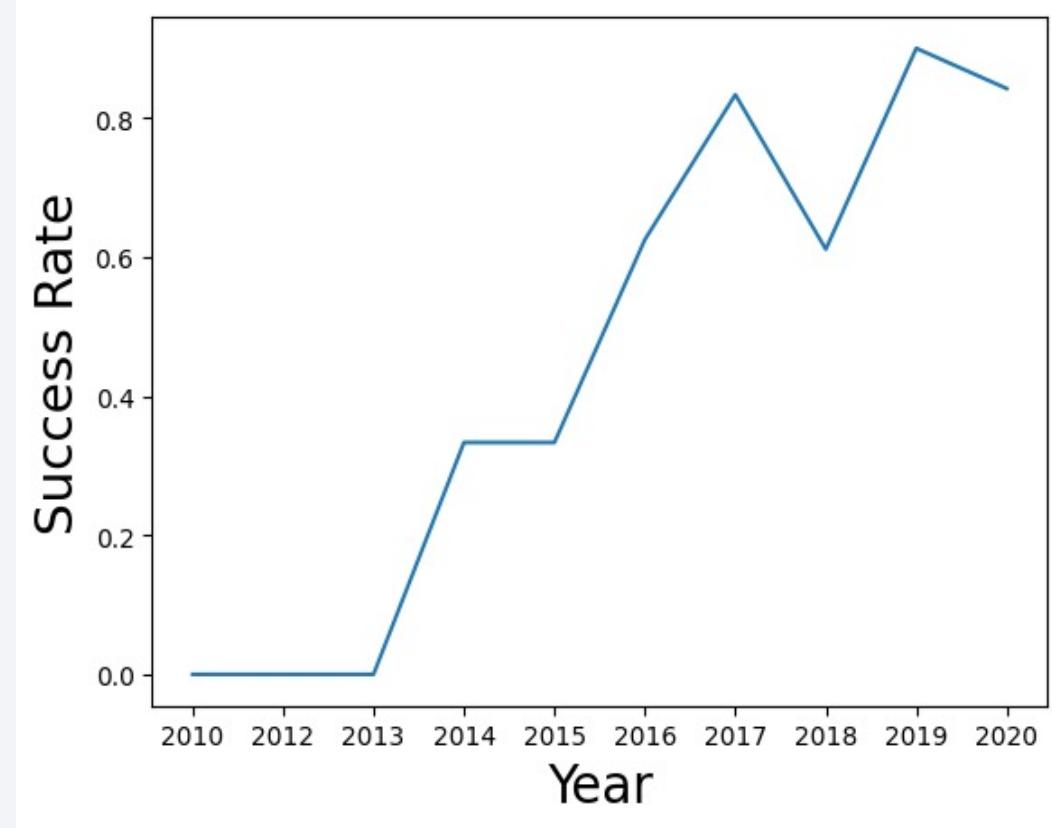


We learn that for loads above 9000 kg, ISS and PO and VLEO represent success. Also, for loads between 4000 and 6000, LEO, ISS represent full success. For loads lesser than 4000 kg, MEO, HEO, SSO are the best success rate options.

# Launch Success Yearly Trend

---

This trend chart show that the success rate has significantly improved in recent years.



# All Launch Site Names

---

- Find the names of the unique launch sites

```
[ ] # 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
```

This value\_count query shows the frequency of launches across three diverse launch sites.

# Launch Site Names Begin with 'CCA'

- Find 5 records where launch sites begin with `CCA`.

Display 5 records where launch sites begin with the string 'CCA'

```
[ ] %sql select * from SPACEXTABLE where launch_site like 'CCA%' limit 5;
```

```
* sqlite:///my_data1.db
Done.
```

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

This SQL SELECT query identifies 5 records with launch sites starting with CCA.

# 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)';

* sqlite:///my_data1.db
Done.
total_payload_mass
45596
```

This SQL SUM query calculates the total payload to be 45,596 kg.

# 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%';  
* sqlite:///my_data1.db  
Done.  
average_payload_mass  
2534.6666666666665
```

This SQL AVG query calculates the average payload to be 2,534.66 kg.

# First Successful Ground Landing Date

---

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

```
[ ] %sql select min(date) as first_successful_landing from SPACEXTABLE where Mission_Outcome = 'Success';  
* sqlite:///my_data1.db  
Done.  
first_successful_landing  
2010-04-06
```

This SQL MIN query determines the first date for successful landing outcome to be 2010-04-06.

## 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

```
[ ] %sql select booster_version from SPACEXTABLE where Mission_Outcome = 'Success (drone ship)' and payload_mass_kg_ between 4000 and 6000;  
* sqlite:///my_data1.db  
Done.  
Booster_Version
```

This SQL SELECT conditional query shows the boosters meeting the criteria.

# Total Number of Successful and Failure Mission Outcomes

---

- Calculate the total number of successful and failure mission outcomes.

```
[ ] %sql select Mission_Outcome, count(*) as total_number from SPACEXTABLE group by Mission_Outcome;  
* sqlite:///my_data1.db  
Done.  


| Mission_Outcome                  | total_number |
|----------------------------------|--------------|
| Failure (in flight)              | 1            |
| Success                          | 98           |
| Success                          | 1            |
| Success (payload status unclear) | 1            |


```

This SQL Count query calculates the frequency of success and failure mission outcomes to be 100 success cases and 1 failure.

# Boosters Carried Maximum Payload

---

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

```
[ ] *sql select booster_version from SPACEXTABLE where payload_mass_kg_ = (select max(payload_mass_kg_) from SPACEXTABLE);

* sqlite:///my_data1.db
Done.
Booster_Version
F9 B5 B1048.4
F9 B5 B1049.4
F9 B5 B1051.3
F9 B5 B1056.4
F9 B5 B1048.5
F9 B5 B1051.4
F9 B5 B1049.5
F9 B5 B1060.2
F9 B5 B1058.3
F9 B5 B1051.6
F9 B5 B1060.3
F9 B5 B1049.7
```

This SQL conditional query determines the names of the booster meeting the criteria.

# 2015 Launch Records

---

- List the failed landing\_outcomes in drone ship, their booster versions, and launch site names for in year 2015

```
In [12]: %%sql select monthname(date) as month, date, booster_version, launch_site, landing_outcome from SPACEXDATASET  
where landing_outcome = 'Failure (drone ship)' and year(date)=2015;  
  
* ibm_db_sa://wzf08322:***@0c77d6f2-5da9-48a9-81f8-86b520b87518.bs2io90108kqb1od8lcg.databases.appdomain.cloud:31198/bludb  
Done.  
Out[12]:
```

MONTH	DATE	booster_version	launch_site	landing_outcome
January	2015-01-10	F9 v1.1 B1012	CCAFS LC-40	Failure (drone ship)
April	2015-04-14	F9 v1.1 B1015	CCAFS LC-40	Failure (drone ship)

This SQL query determines the failed landing\_outcomes in 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 Mission_Outcome, count(*) as count_outcomes from SPACEXTABLE
      where date between '2010-06-04' and '2017-03-20'
      group by Mission_Outcome
      order by count_outcomes desc;

* sqlite:///my_data1.db
Done.

Mission_Outcome count_outcomes
Success          31
Failure (in flight) 1
```

This SQL Count query determines the frequency of success and failure cases between the specified dates to be 31 success and 1 failure cases.

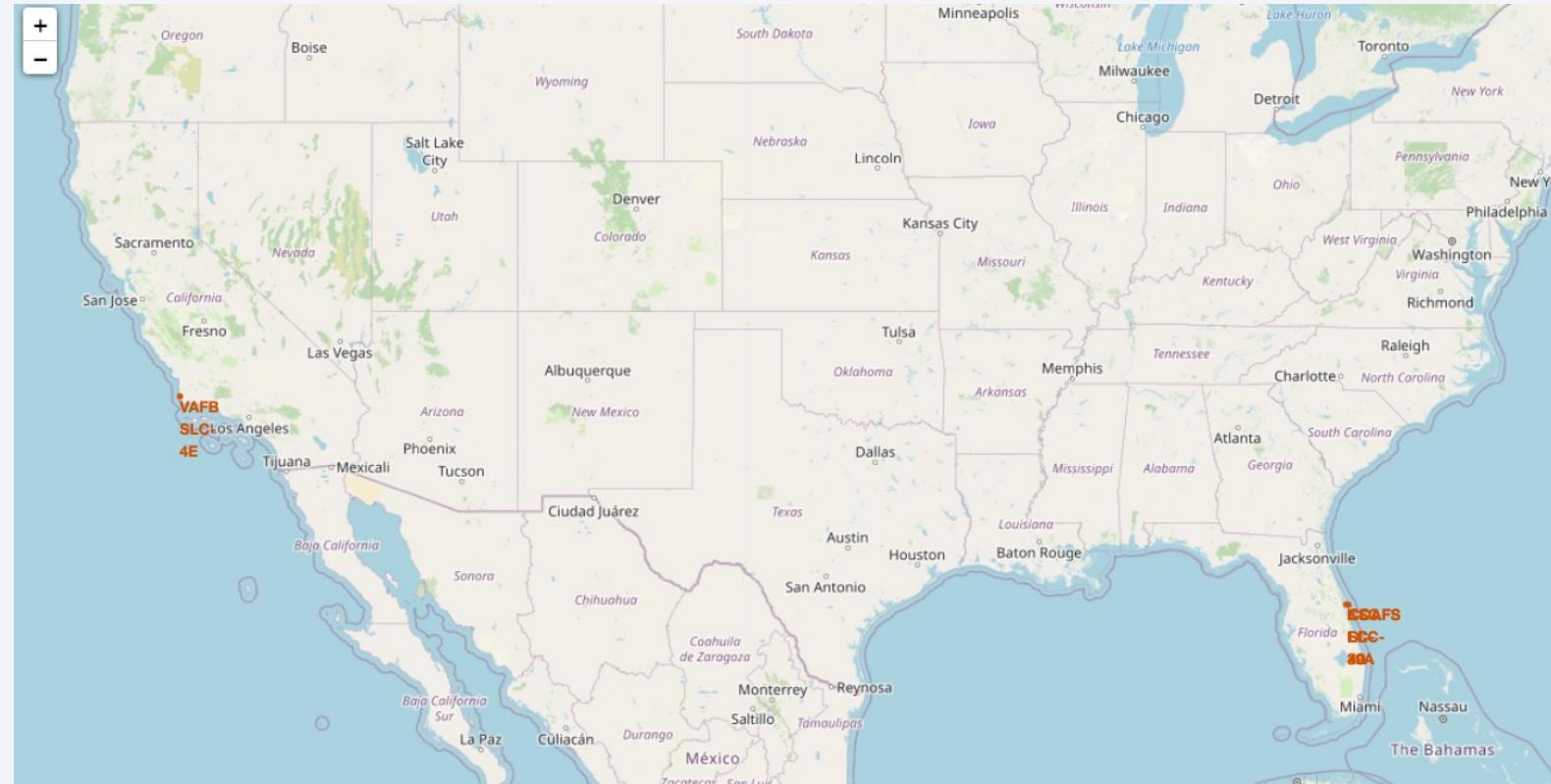
The background of the slide is a photograph taken from space at night. It shows the curvature of the Earth against a dark blue-black void of space. City lights are visible as numerous small white and yellow dots, primarily concentrated in the lower right quadrant where the United States appears. In the upper right, the green and yellow glow of the aurora borealis is visible. The atmosphere of the Earth is thin and hazy, appearing as a light blue band near the horizon.

Section 3

# Launch Sites Proximities Analysis

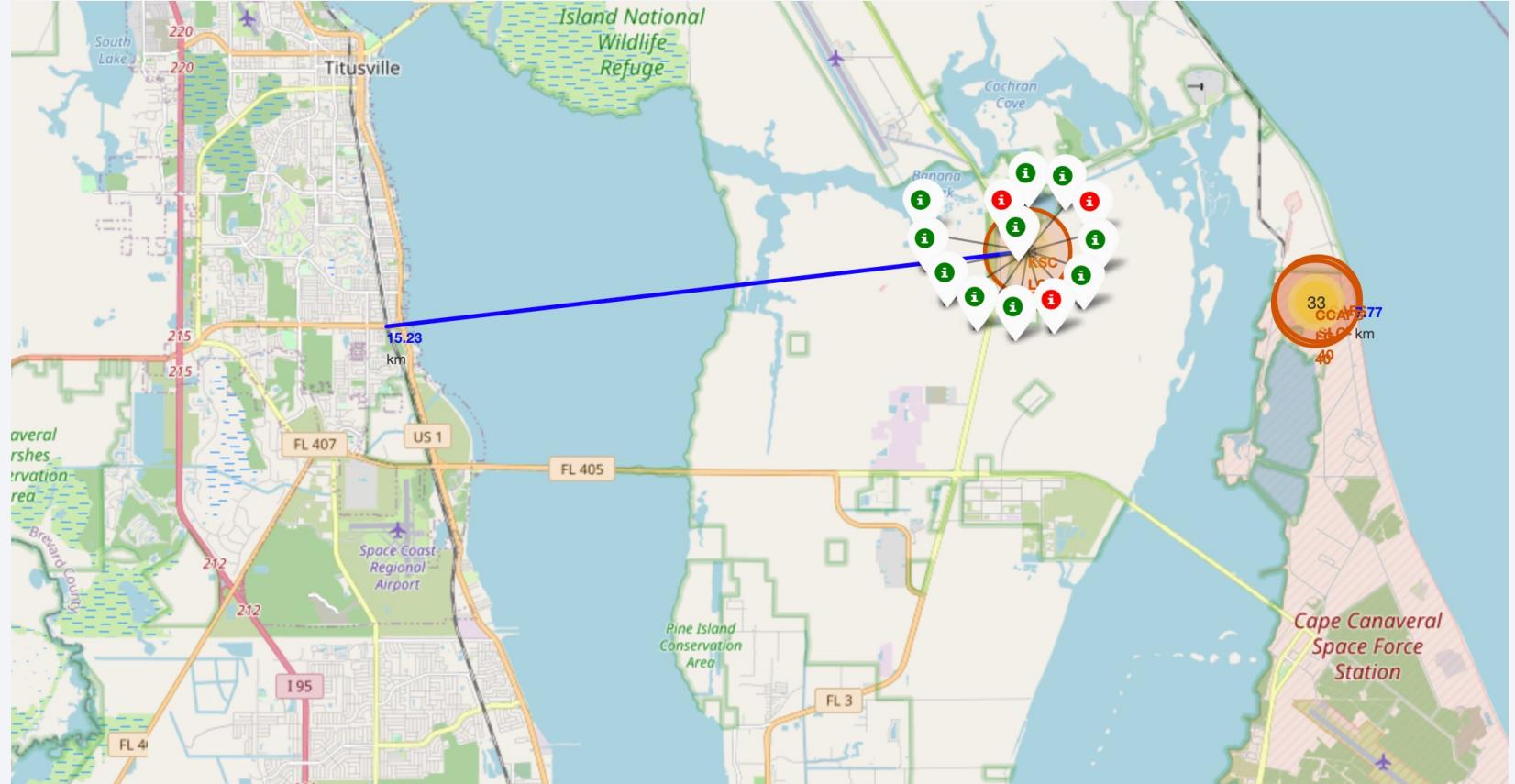
# Geographical Location of Launch Sites

- The Launch sites are in very close proximity to the coastline on both east and west coasts of the United States.



# Success of Launches for KSC LC-39A

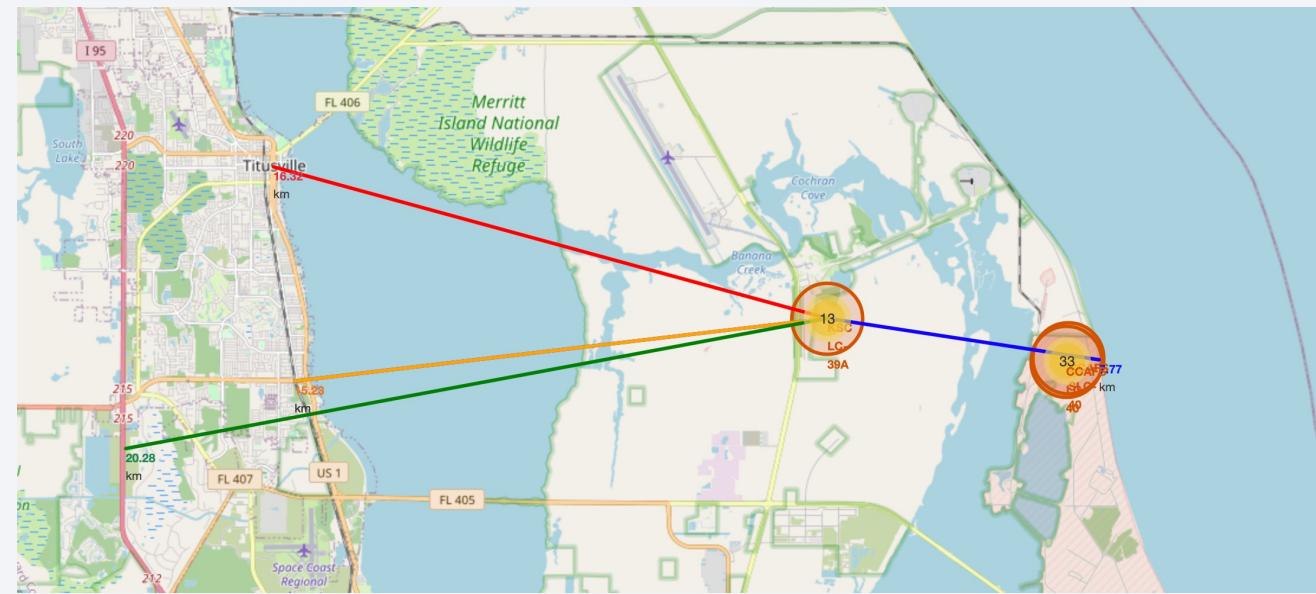
- This map shows successful (green) and failed launches for KSC LC-39A.



# Proximity to Infrastructure

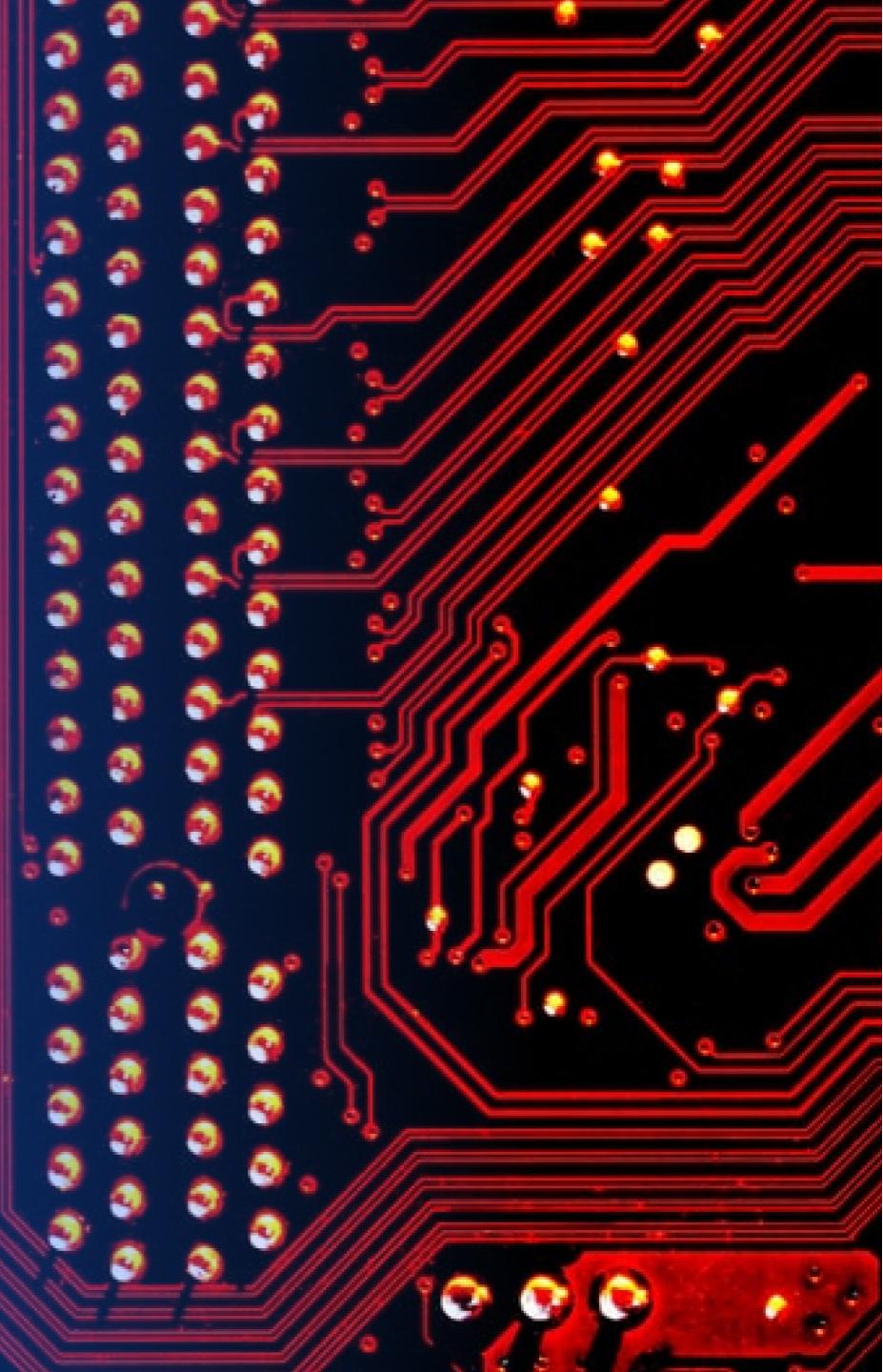
---

- The map of the launch site KSC LC-39A shows that closest railway, highway and city (Titusville), coastline are 15.23 km, 20.28 km, 7.77 km, and 16.32 km away.



Section 4

# Build a Dashboard with Plotly Dash

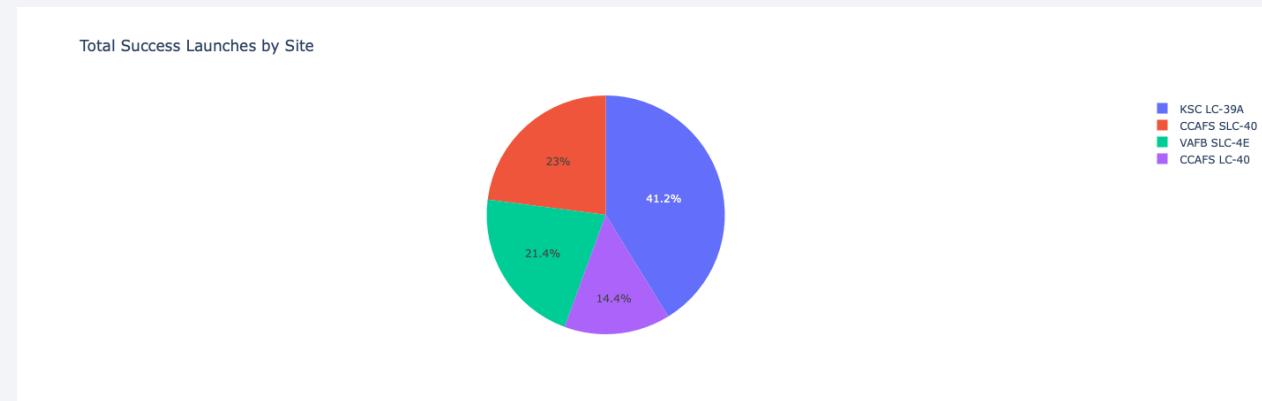


# Dashboard for all launche sites

---

- Show the screenshot of launch success count for all sites, in a piechart

The chart clearly shows that KSC LC-39A has the most successful launches (41.2%).

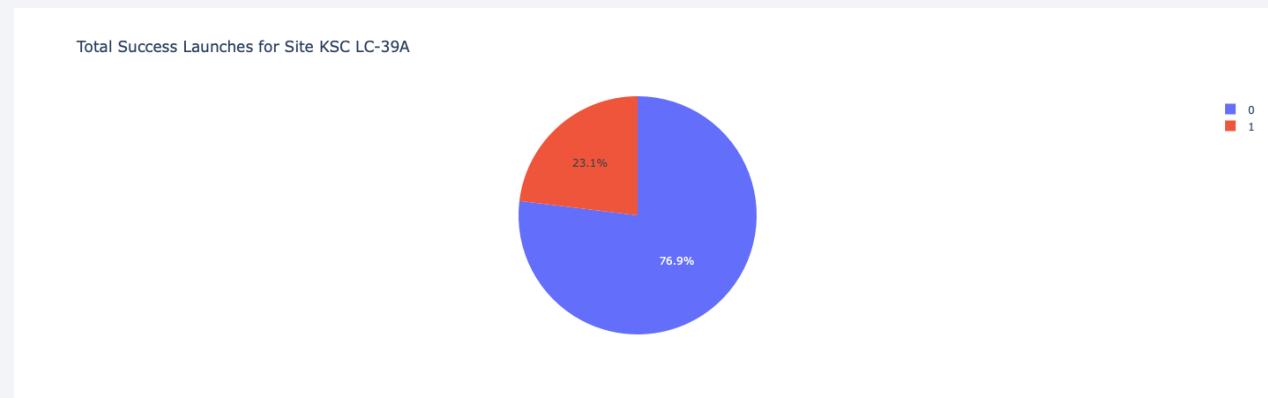


# Dashboard for the launch site with highest success

---

- Show the screenshot of the piechart for the launch site with highest launch success ratio

This chart shows that KSC LC-39A has the highest launch success rate (76.9%).



The background of the slide features a dynamic, abstract design. It consists of several thick, curved lines that transition from a bright yellow at the top right to a deep blue at the bottom left. These lines create a sense of motion and depth, resembling a tunnel or a stylized landscape. The overall effect is modern and professional.

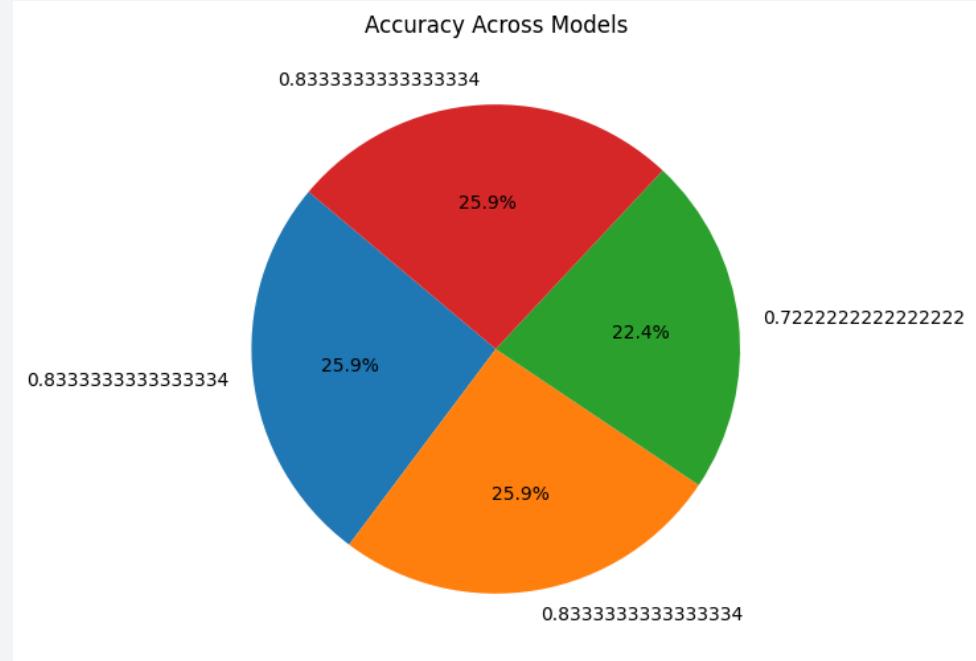
Section 5

# Predictive Analysis (Classification)

# Classification Accuracy

- Visualize the built model accuracy for all built classification models, in a bar chart. Find which model has the highest classification accuracy.

The results show that logistic regression, SVM and KNN have the highest performance.

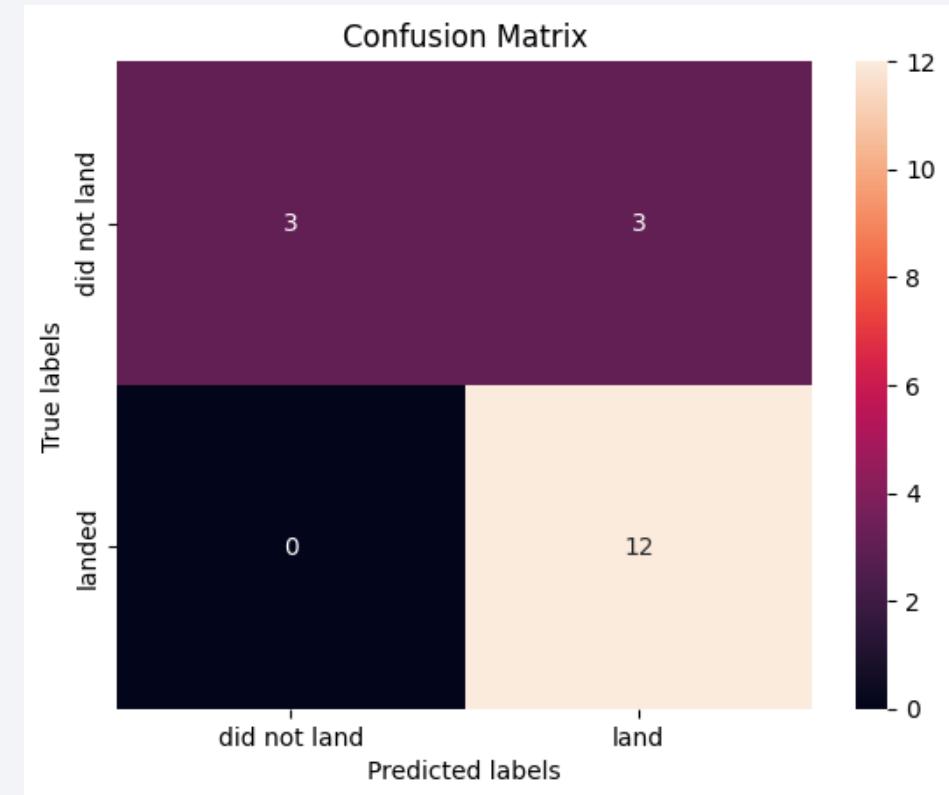


index	LogReg	SVM	Tree	KNN
Jaccard_Score	0.8	0.8	0.8	0.8
F1_Score	0.88888888888889	0.88888888888889	0.88888888888889	0.88888888888889
Accuracy	0.8333333333333334	0.8333333333333334	0.7222222222222222	0.8333333333333334

# Confusion Matrix

- Show the confusion matrix of the best performing model with an explanation

The confusion Matrix for a high-performing model such as KNN shows 12 cases are true positives, 3 are true negatives, 3 are false positives and zero are false negatives.



# Conclusions

---

- Logistic regression, SVM and KNN models performed equally well and high for this dataset.
- Decision Tree Model is the lowest performing model for this dataset.
- Launches associated with a low payload mass demonstrate better results than launches with a larger payload mass.
- Most of launch sites are in proximity to the Equator line and all the sites are in very close proximity to the coastline.
- The success rate of launches has significantly increased over the years.
- KSC LC-39A has the highest success rate of the launches from all the sites.

Thank you!

