IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Created by Chueshie Xiong
CAO: 23FEB2024

Outline

- Executive Summary
- Introduction
- Methodology
- Results
- Conclusion

Executive Summary

- **Summary of Methodologies**
  - Data Collection through API
  - Data Collection with Web Scraping
  - Data Wrangling
  - Exploratory Data Analysis with SWL
  - Exploratory Data Analysis with Data Visualization
  - Interactive Visual Analytics with Folium
  - Machine Learning Prediction
- **Summary of all results**
  - Exploratory Data Analysis result
  - Interactive analytics in screenshots
  - Predictive Analytics result from Machine Learning Lab

Introduction

Background

SpaceX has gained worldwide attention for a series of historic milestones. It is the only private company ever to return a spacecraft from low-earth orbit, which it first accomplished in December 2010. SpaceX advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars where as other providers cost upward of 165 million dollars each, much of the savings is because Space X can reuse the first stage. Therefore, if we can determine if the first stage will land, we can determine the cost of a launch. This information can be used if an alternate company wants to bid against space X for a rocket launch. The goal of this project is to create a machine learning pipeline to predict if the first stage will land given the data from the preceding labs.

Challenges

- Identify factors influencing the landing outcome
- Discover relationships between different variables such as payload mass, orbit, and other factors
- Find the best condition needed to increase the probability of successful landing

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - Make a get request to the SpaceX API → Request and Parse the Launch Data
  - Perform web scraping techniques to collect Falcon 9 historical launch records
- Perform data wrangling
  - Load the data, handle missing values, ID the types of Data, Perform Calculations, Create outcome labels, and apply one-hot encoding to prepare data for analysis and modeling
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - How to build, tune, evaluate classification models

Data Collection – API

Data Collection Process

1. **Import** Libraries and Define Auxiliary Functions
2. **Define** a series of helper functions to help with the API in order to extract information using ID Numbers in the launch data
3. **Request** rocket launch data from SpaceX API w/ the provided URL
4. **Decode** the response content as a Json and turn it into a Pandas dataframe
5. **Apply** getBoosterVersion function method (refer to link for more in depth explanation of function)
6. **Perform** Data Wrangling Operations to clean, transform, and handle missing values
7. **Export** data to csv file

Data Collection – SpaceX API

Collecting the Data

1. Request and parse the SpaceX launch data using the GET Request
2. Decode the response content as Json and turn it into a Pandas dataframe using .json_normalize()
3. Filter the dataframe to only include Falcon launches
4. Conduct further data cleaning to check for missing values and any other data wrangling necessary

https://github.com/cx2206/SpaceX-Capstone/blob/main/jupyter-labs-spacex-data-collection-api.ipynb

1

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

response = requests.get(spacex_url)
```

Check the content of the response

```
print(response.content)
```

2

Now we decode the response content as a Json using `.json()` and turn it into a Pandas dataframe using `.json_normalize()`

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

3

```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = launch_data[launch_data['BoosterVersion']!='Falcon 1']
data_falcon9.head()
```

4

```
# Calculate the mean value of PayloadMass column
mean=data_falcon9['PayloadMass'].mean()

# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'].replace(np.nan,mean, inplace=True)

data_falcon9.isnull().sum()
```

Dealing w/ Missing Values

FlightNumber	0	FlightNumber	0
Date	0	Date	0
BoosterVersion	0	BoosterVersion	0
PayloadMass	5	PayloadMass	0
Orbit	0	Orbit	0
LaunchSite	0	LaunchSite	0
Outcome	0	Outcome	0
Flights	0	Flights	0
GridFins	0	GridFins	0
Reused	0	Reused	0
Legs	0	Legs	0
LandingPad	26	LandingPad	26
Block	0	Block	0
ReusedCount	0	ReusedCount	0
Serial	0	Serial	0
Longitude	0	Longitude	0
Latitude	0	Latitude	0
dtype: int64		dtype: int64	

Data Collection – Web Scraping

Web Scraping Process

1. **Utilize** the BeautifulSoup to extract the launch records as HTML table
2. **Request** the Falcon9 Launch Wiki page from its URL
3. **Parse** the table and **Convert** it to a pandars dataframe for further analysis
4. **Export** data to csv file

Data Collection – Web Scraping

Web Scraping Process

1. **Utilize** the BeautifulSoup to extract the launch records as HTML table
2. **Request** the Falcon9 Launch Wiki page from its URL
3. **Parse** the table and **Convert** it to a pandars dataframe for further analysis
4. **Export** data to csv file

https://github.com/cx2206/SpaceX-Capstone/blob/main/jupyter-labs-webscraping.ipynb

1

TASK 1: Request the Falcon9 Launch Wiki page from its URL

First, let's perform an HTTP GET method to request the Falcon9 Launch HTML page, as an HTTP response.

```
# use requests.get() method with the provided static_url
html_data = requests.get(static_url)
# assign the response to a object
html_data.status_code
```

```
200
```

Create a `BeautifulSoup` object from the HTML `response`

```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(html_data.text)
```

Print the page title to verify if the `BeautifulSoup` object was created properly

```
# Use soup.title attribute
soup.title
```

```
<title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

2

TASK 2: Extract all column/variable names from the HTML table header

Next, we want to collect all relevant column names from the HTML table header

Let's try to find all tables on the wiki page first. If you need to refresh your memory about `BeautifulSoup`, please check the external reference link towards the end of this lab

```
# Use the find_all function in the BeautifulSoup object, with element type `table`
# Assign the result to a list called `html_tables`
html_tables = soup.find_all('table')
```

Starting from the third table is our target table contains the actual launch records.

```
# Let's print the third table and check its content
first_launch_table = html_tables[2]
print(first_launch_table)
```

3

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

	Flight No.	Launch site	Payload	Payload mass	Orbit	Customer	Launch outcome	Version Booster	Booster landing	Date	Time
0	1	CCAFS	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success\n	F9 v1.0B0003.1	Failure	4 June 2010	18:45
1	2	CCAFS	Dragon	0	LEO	NASA	Success	F9 v1.0B0004.1	Failure	8 December 2010	15:43
2	3	CCAFS	Dragon	525 kg	LEO	NASA	Success	F9 v1.0B0005.1	No attempt\n	22 May 2012	07:44
3	4	CCAFS	SpaceX CRS-1	4,700 kg	LEO	NASA	Success\n	F9 v1.0B0006.1	No attempt	8 October 2012	00:35
4	5	CCAFS	SpaceX CRS-2	4,877 kg	LEO	NASA	Success\n	F9 v1.0B0007.1	No attempt\n	1 March 2013	15:10

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

Data Wrangling

Data Wrangling Process

- Perform EDA
- Determine Training Labels
- Load data from last section
- Clean Data (Examples: Missing Values/Data Types)

Tasks

1. Calculate the number of launches on each site
2. Calculate the number and occurrence of each orbit
3. Calculate the number and occurrence of mission outcome of the orbits
4. Create a landing outcome label from Outcome column

TASK 1: Calculate the number of launches on each site

```
# 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
```

TASK 2: Calculate the number and occurrence of each orbit

Use the method `.value_counts()` to determine the number and occurrence of each orbit in the column `Orbit`

```
# 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
HEO       1
SO        1
GEO       1
Name: Orbit, dtype: int64
```

Data Wrangling Continued…

Data Wrangling Process

- Perform EDA
- Determine Training Labels
- Load data from last section
- Clean Data (Examples: Missing Values/Data Types)

Tasks

1. Calculate the number of launches on each site
2. Calculate the number and occurrence of each orbit
3. Calculate the number and occurrence of mission outcome of the orbits
4. Create a landing outcome label from Outcome column

https://github.com/cx2206/SpaceX-Capstone/blob/main/labs-jupyter-spacex-Data%20wrangling.ipynb

TASK 3: Calculate the number and occurence of mission outcome of the orbits

Use the method `.value_counts()` on the column `Outcome` to determine the number of `landing_outcomes` .Then assign it to a variable landing_outcomes.

```
# 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
False Ocean     2
None ASDS       2
False RTLS      1
Name: Outcome, dtype: int64
```

Landing Outcome (Reference)

- True Ocean: mission outcome had a successful landing to a specific region of the ocean
- False Ocean: represented an unsuccessful landing to a specific region of ocean
- True RTLS: meant the mission had a successful landing on a ground pad
- False RTLS: represented an unsuccessful landing on a ground pad
- True ASDS: meant the mission outcome had a successful landing on a drone ship
- False ASDS: represented an unsuccessful landing on drone ship • Outcomes converted into 1 for a successful landing and 0 for an unsuccessful landing

TASK 4: Create a landing outcome label from Outcome column

Using the `Outcome`, create a list where the element is zero if the corresponding row in `Outcome` is in the set `bad_outcome`; otherwise, it's one. Then assign it to the variable `landing_class`:

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

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

This variable will represent the classification variable that represents the outcome of each launch. If the value is zero, the first stage did not land successfully; one means the first stage landed Successfully

```
df['Class']=landing_class
df[['Class']].head(8)
```

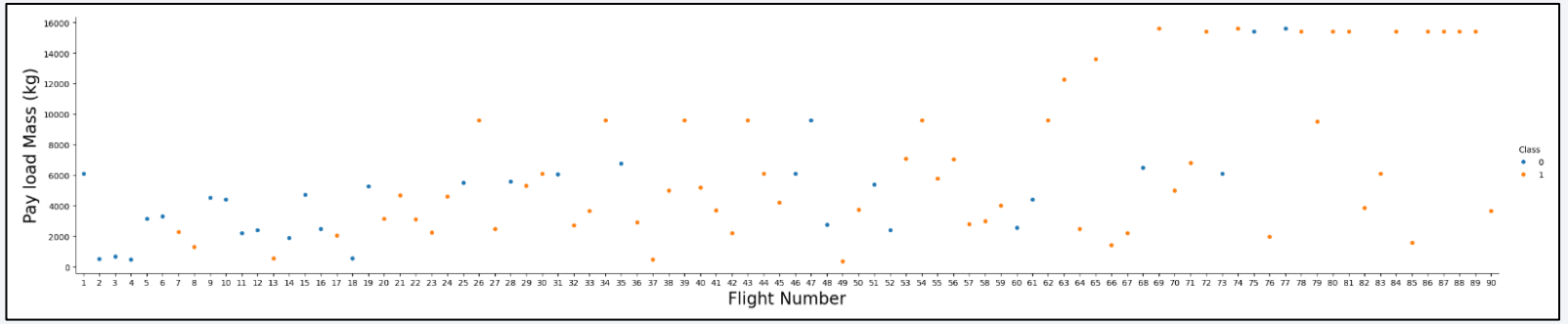
	Class
0	0
1	0
2	0
3	0
4	0
5	0
6	1
7	1

EDA with Data Visualization

Charts Plotted

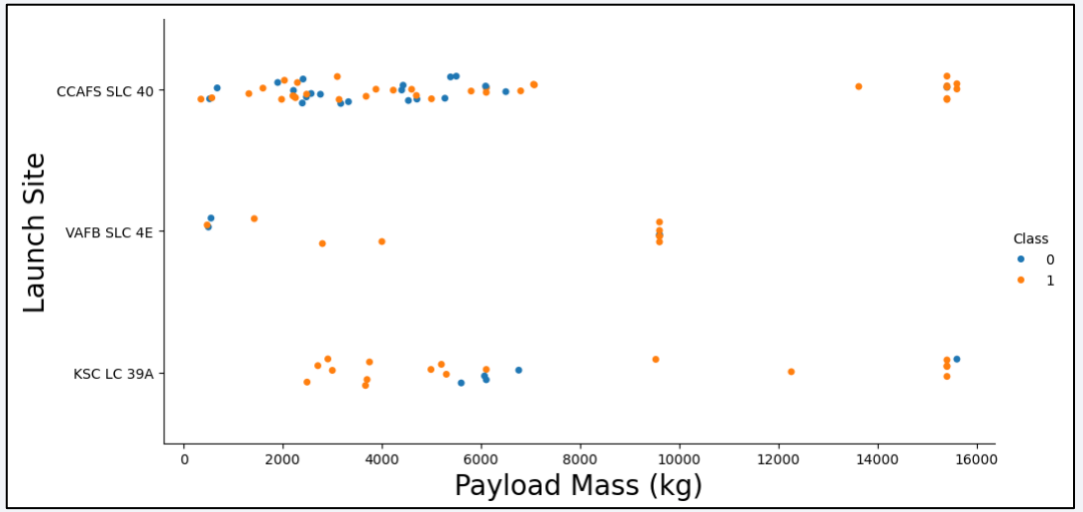
- Flight Number vs. Payload
- Flight Number vs. Launch Site
- Payload Mass (kg) vs. Launch Site
- Payload Mass (kg) vs. Orbit type

* Not all charts are presented on the slide
For more examples click on the link below!



Scatter plots were used to discover relationships between the different attributes (Payload, Flight Number, Launch Site, and Orbit Type).

Scatter plots can help viewers with observing patterns.



https://github.com/cx2206/SpaceX-Capstone/blob/main/jupyter-labs-eda-dataviz.ipynb.jupyterlite.ipynb

EDA with Data Visualization

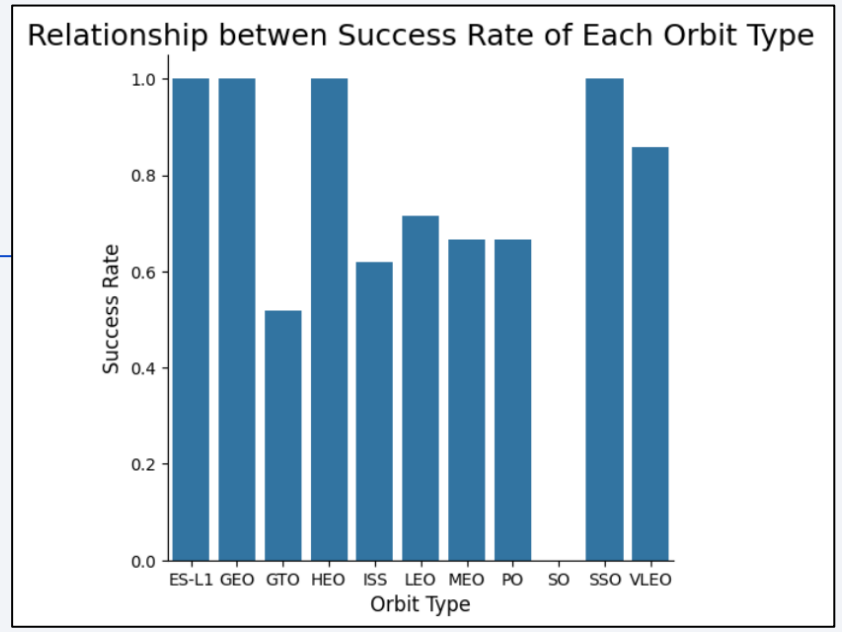
Charts Plotted

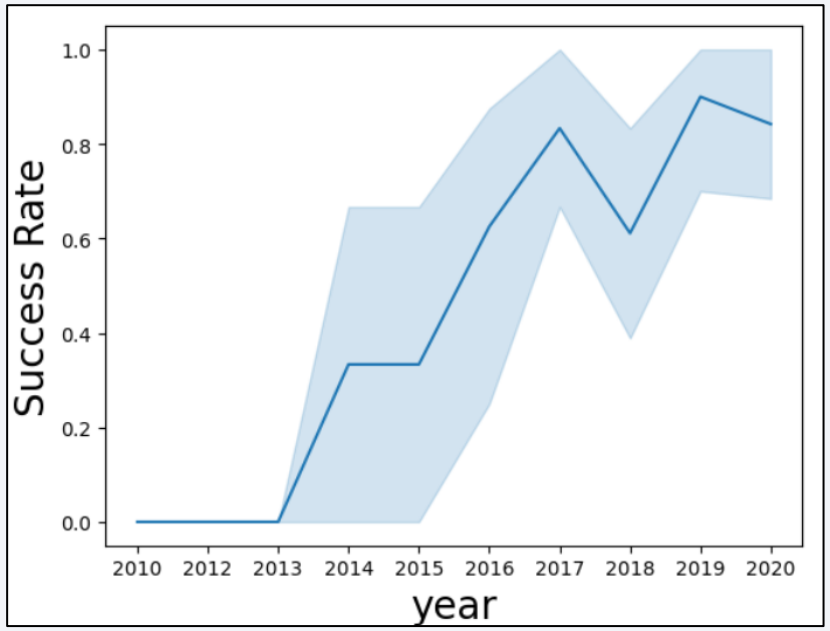
- Orbit Type vs Success Rate
- Year vs Success Rate

* Not all charts are presented on the slide
For more examples click on the link below!

Bar graphs were used to interpret the relationship between the attributes. We used the bar graph to determine which orbits had the highest probability of success rate.

A line graph was plotted for the Year vs Success rate from 2010 to 2017. We can observe that throughout 2010 to 2020 there was an increase of success rate. There was a drop in 2018, but a year later the success rate continued to raise again until 2020.





https://github.com/cx2206/SpaceX-Capstone/blob/main/jupyter-labs-eda-dataviz.ipynb.jupyterlite.ipynb

EDA with SQL

- Using SQL, we had performed many queries to get better understanding of the dataset, Ex:
- Displaying the names of the launch sites.
- Displaying 5 records where launch sites begin with the string ‘CCA’.
- Displaying the total payload mass carried by booster launched by NASA (CRS).
- Displaying the average payload mass carried by booster version F9 v1.1.
- Listing the date when the first successful landing outcome in ground pad was achieved.
- Listing the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000.
- Listing the total number of successful and failure mission outcomes.
- Listing the names of the booster_versions which have carried the maximum payload mass.
- Listing the failed landing_outcomes in drone ship, their booster versions, and launch sites names for in year 2015.
- Rank the count of landing outcomes or success between the date 2010-06-04 and 2017-03-20, in descending order.

Click on the link to view the EDA w/ SQL:
https://github.com/cx2206/SpaceX-Capstone/blob/main/jupyter-labs-eda-sql-coursera_sqllite.ipynb

Build an Interactive Map with Folium

Summarize what map objects such as markers, circles, lines, etc. you created and added to a folium map

- **Marked all launch sites on a map using the site's latitude and longitude coordinates with the Launch Site name**
  - Created a circle w/ a text label marker of the Launch Site Name
- **Marked the success/failed launches for each site on the map**
  - If a lunch was **successful (class=1)**, then we use a **green marker** and if a launch was **failed**, we used a **red marker (class=0)**
- **Calculate the distances between a launch site to its proximities**
  - Utilize Haversine's formula to calculate the distance of the launch sites to various landmark such as the nearest coastline, railway, highway, and the city.

https://github.com/cx2206/SpaceX-Capstone/blob/main/lab_jupyter_launch_site_location.ipynb

Build a Dashboard with Plotly Dash

- **Dropdown list w/ Launch Sites:** Allow users to select all launch sites or specific sites
- **Pie Chart Showing Successful Launches:** Displays successful and unsuccessful launches in a % total
- **Slider of Payload Mass Range:** Allow user to select payload mass vs range
- **Scatter Chart (Payload Mass vs Success Rate by Booster Version):** Shows a correlation between the two variables (if there is any)

Predictive Analysis (Classification)

- Import necessary libraries
- Def plot_confusion_maxtric to plot the confusion matrix
- Load the datagrame
- 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']).
- Standardize the data in X then reassign it to the variable X using the transform provided below.
- 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.
- 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.

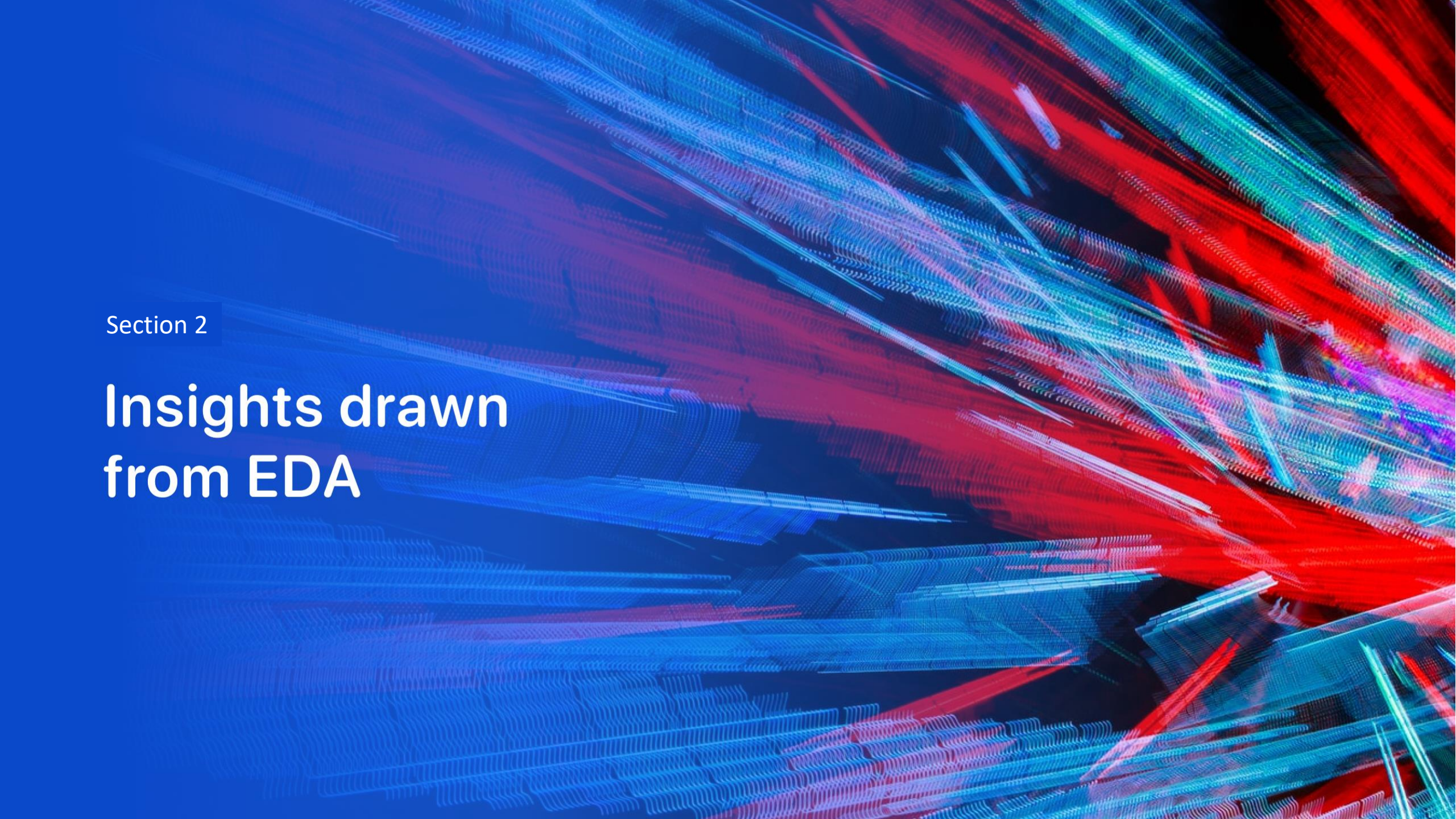
- Calculate the accuracy on the test data using the method score
- Create a support vector machine object then create a GridSearchCV object svm_cv with cv - 10. Fit the object to find the best parameters from the dictionary parameters
- Calculate the accuracy on the test data using the method score
- 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.
- Calculate the accuracy of tree_cv on the test data using the method score

- 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.
- Calculate the accuracy of knn_cv on the test data using the method score
- Calculate the accuracy of knn_cv on the test data using the method score

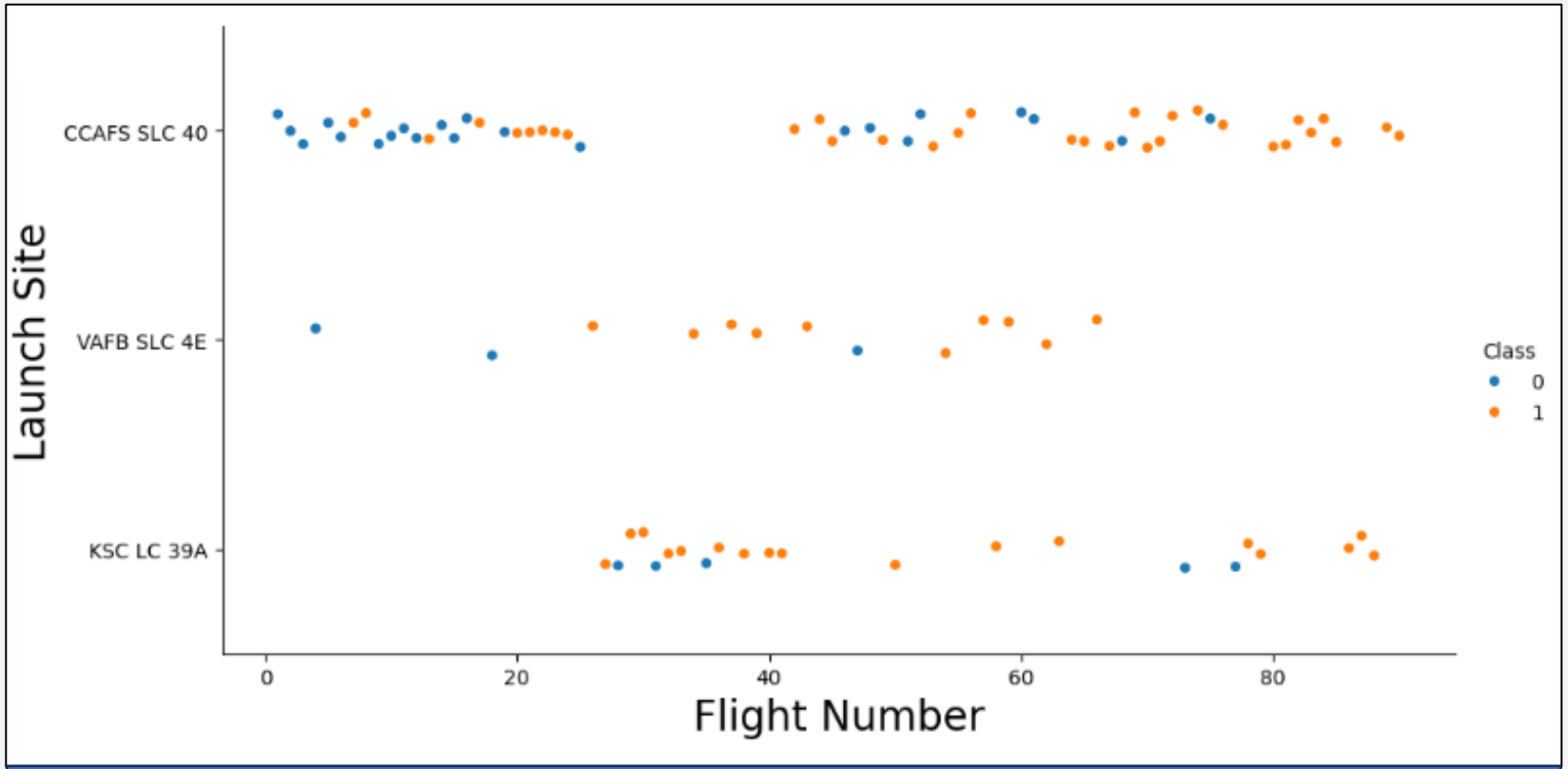
https://github.com/cx2206/SpaceX-Capstone/blob/main/SpaceX_Machine_Learning_Prediction_Part_5.jupyterlite.ipynb

Results

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


Section 2
Insights drawn from EDA

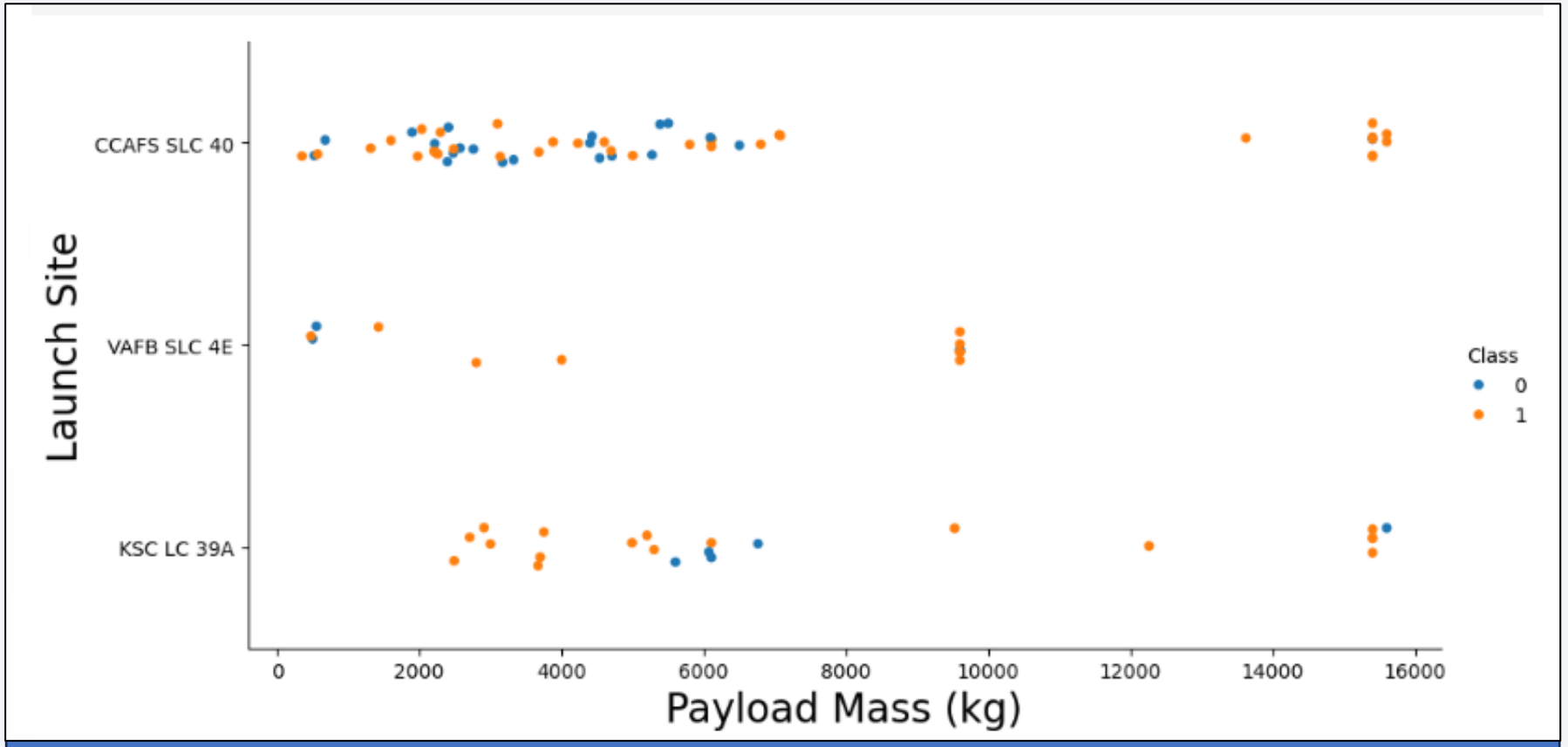
Flight Number vs. Launch Site



Insights Drawn from EDA:

- Earlier flights had lower success rate (indicated in blue)
- Later flights (indicated in orange) had a higher success rate.
- VAFB SLC 4E and KSC LC 39A had less flight numbers but a higher success rate ratio compared to CCAFS SLC 40

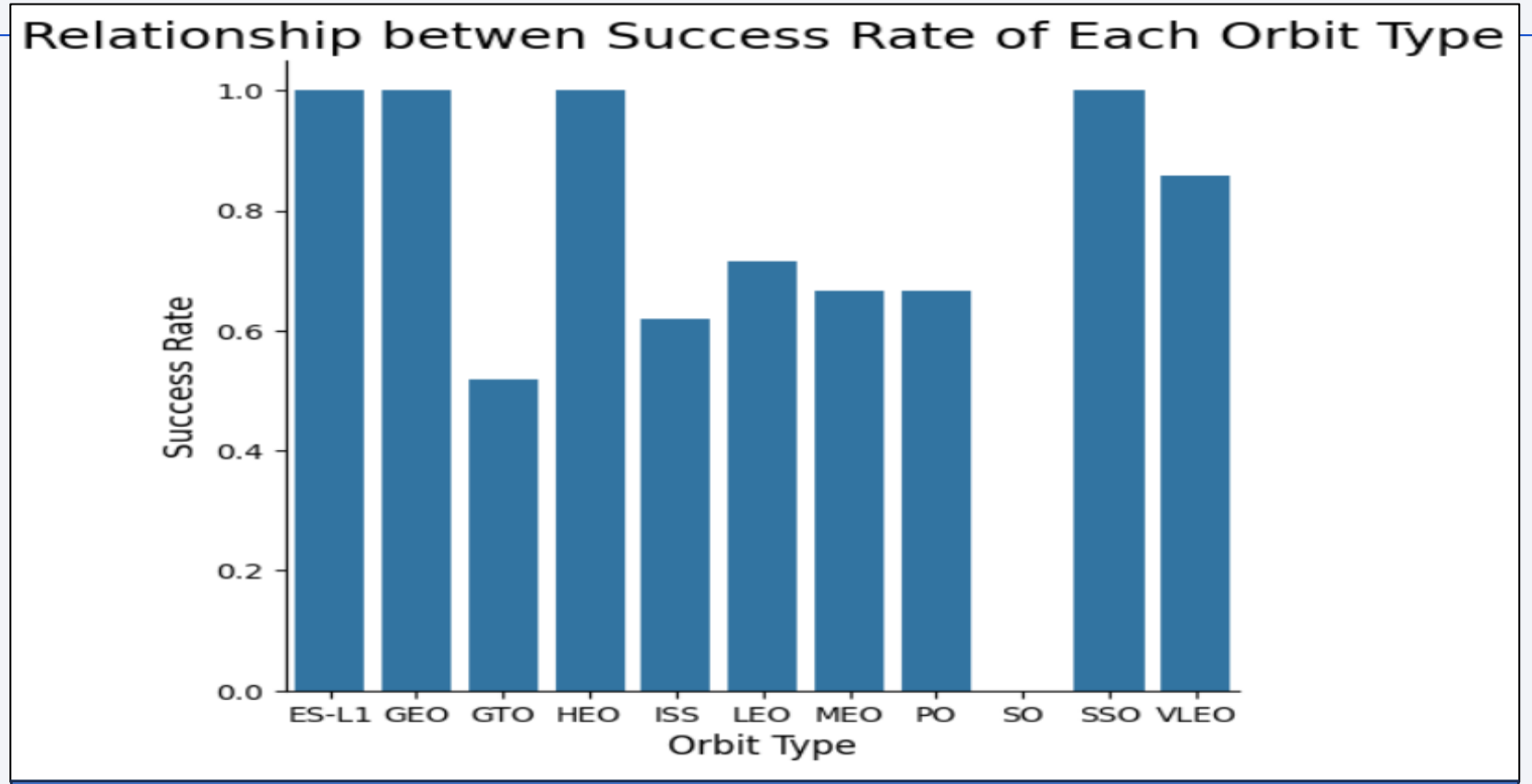
Payload vs. Launch Site



Insights Drawn from EDA:

- Higher payloads greater than 7,000 kg had a higher probability of success rate
- KSC LC 39A is likely to have a higher success rate with payloads less than 5,000 kg

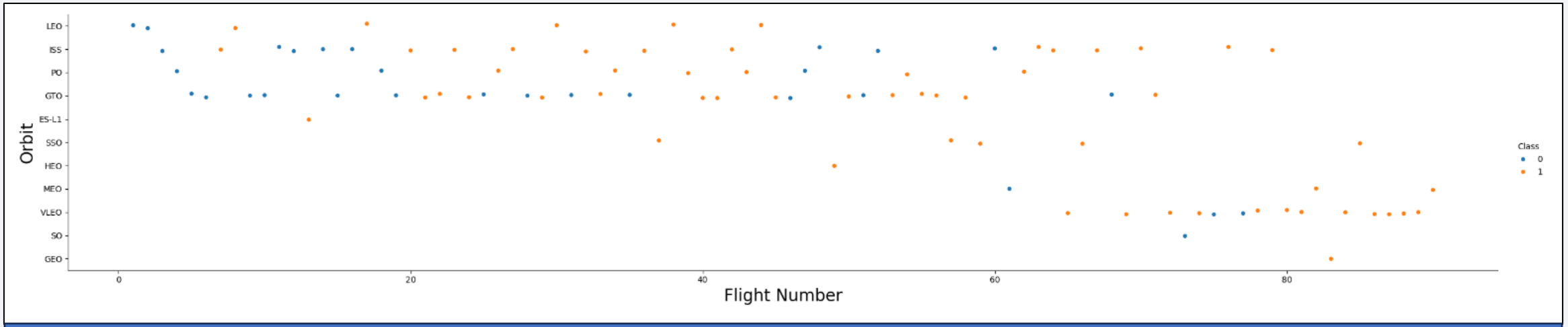
Success Rate vs. Orbit Type



Insights Drawn from EDA:

- Success rate of 100%: ES-L1, GEO, HEO, SSO, and VLEO had a GTO, ISS, LEO,
  Success rate between 50%-80%: MEO, PO, VLEO
- 0% success rate: SO

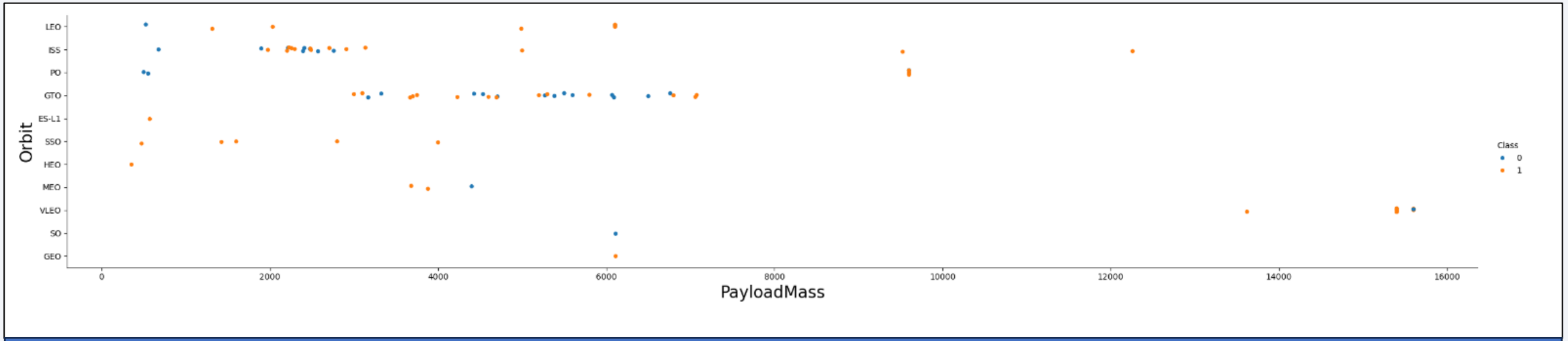
Flight Number vs. Orbit Type



Insights Drawn from EDA:

- LEO, ISS, PO, SSO, VLEO success rate is higher with the increase flight number
- GTO has inconsistent pattern of success rate with the increase of flight number
- HEO, MEO, SO, GEO have limited data to make a strong inference

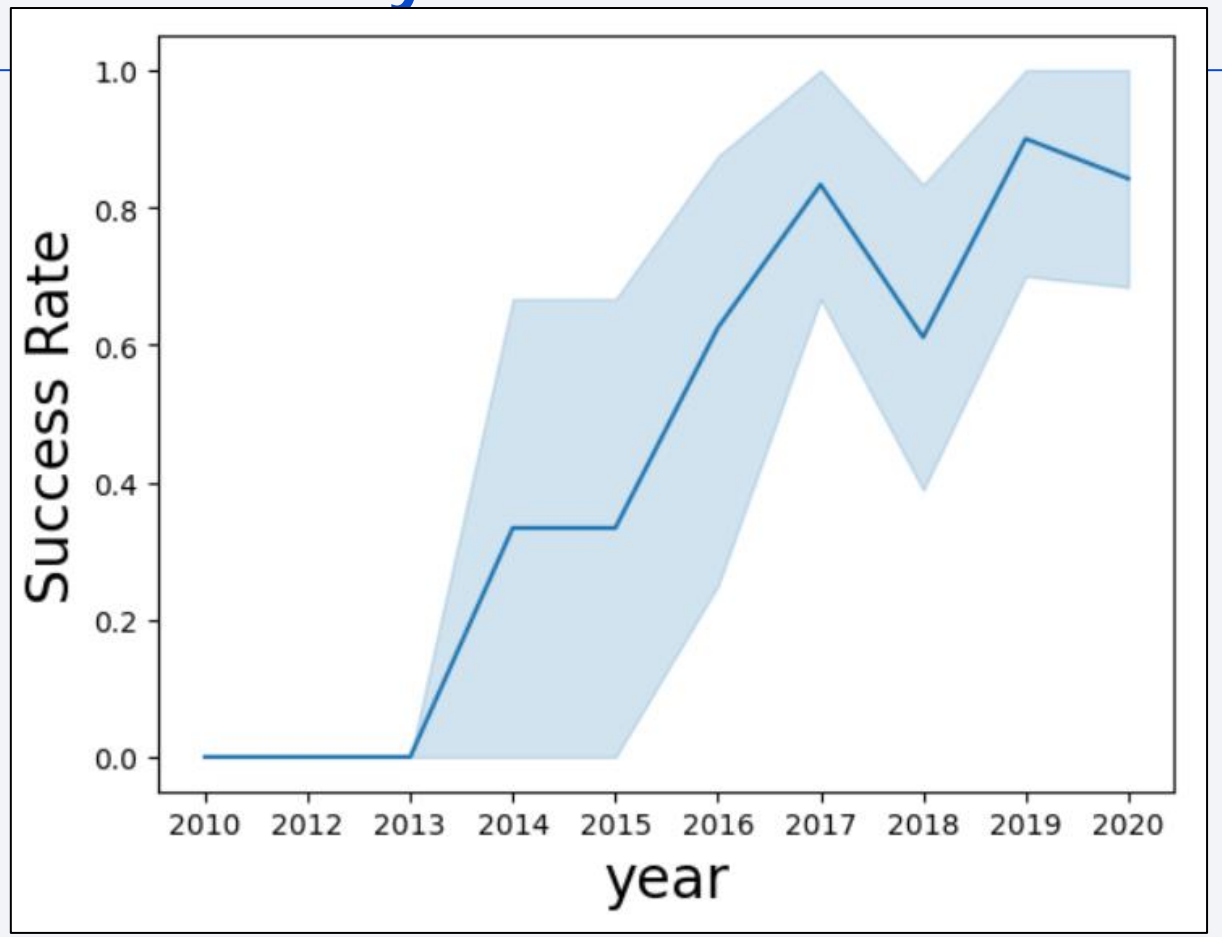
Payload vs. Orbit Type



Insights Drawn from EDA:

- SSO payload mass from 500kg-4000kg has a high success rate
- LEO payload mass from 1500kg up to 6000kg has a higher success rate
- ISS and GTO have inconsistent success rates when looking at the payload mass

Launch Success Yearly Trend



Insights Drawn from EDA:

- 0% Success Rate from 2010 – 2013
- Success Rate increased from 2017-2018 and 2018-2019
- Success Rate decreased from 2017-2018 and 2019-2020
- Overall Success Rate has improved from 2013-2020

All Launch Site Names

"distinct" was used to show only unique launch sites from the SpaceX Data.

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

In [8]:
```
%sql select distinct launch_site from SPACEXTBL
```

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

Out[8]:

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

Launch Site Names Begin with 'CCA'

LIKE ‘CCA%’ LIMIT 5 was used in the query to display 5 records where launch sites begin with the string ‘CCA”

Task 2

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

In [12]:
```
%sql SELECT * \
    FROM SPACEXTBL \
    WHERE LAUNCH_SITE LIKE'CCA%' LIMIT 5;
```

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

Out[12]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_O
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	

Total Payload Mass

The total payload carried by boosters from NASA is 45,596 KG. On the right is the query that was used to find the Sum of the Payload.

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

```
%sql SELECT SUM(PAYLOAD_MASS__KG_) \
    FROM SPACEXTBL \
    WHERE CUSTOMER = 'NASA (CRS)';
```

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

SUM(PAYLOAD_MASS__KG_)
45596

Average Payload Mass by F9 v1.1

The AVE payload carried by boosters version F9 v1.1 is 2,928.4 KG. On the right is the query that was used to find the Sum of the Payload.

Display average payload mass carried by booster version F9 v1.1

```
%sql SELECT avg(PAYLOAD_MASS__KG_) \
FROM SPACEXTBL \
WHERE booster_version like 'F9 v1.1'
```

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

avg(PAYLOAD_MASS__KG_)
2928.4

First Successful Ground Landing Date

First successful landing outcome in ground pad was achieved on: 2015-12-22. The query on the right was used to find the date.

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

Hint:Use min function

```
%sql SELECT MIN(DATE) AS FIRST_SUCCESS_GP FROM SPACEXTBL WHERE LANDING_OUTCOME = 'Success (ground p
```

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

FIRST_SUCCESS_GP
2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

Query on the right shows how I was able to set the conditions and ranges to answer the task.

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 SPACEXTBL \
WHERE LANDING_OUTCOME = 'Success (drone ship)' \
AND PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000;
```

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

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

Total Number of Successful and Failure Mission Outcomes

On the right, the image shows the total of successful and failure mission outcomes. The query above counts the total number of failure flights and successes.

Total Success: 100
Total Failure: 1

List the total number of successful and failure mission outcomes

```
%sql SELECT MISSION_OUTCOME, COUNT(*) as total_number \
FROM SPACEXTBL \
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

Boosters Carried Maximum Payload

On the right, the image shows the names of the booster which have carried the maximum payload mass. The query used is displayed with specific conditions set.

```
%sql SELECT BOOSTER_VERSION \
FROM SPACEXTBL \
WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTBL);
```

```
 * 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

2015 Launch Records

```
%sql SELECT substr(Date,6,2) as month, DATE,BOOSTER_VERSION, LAUNCH_SITE, [Landing_Outcome] \
FROM SPACEXTBL \
where [Landing_Outcome] = 'Failure (drone ship)' and substr(Date,0,5)='2015';
```

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

month	Date	Booster_Version	Launch_Site	Landing_Outcome
01	2015-01-10	F9 v1.1 B1012	CCAFS LC-40	Failure (drone ship)
04	2015-04-14	F9 v1.1 B1015	CCAFS LC-40	Failure (drone ship)

The where clause and substr(Date,_,_) was used to filter for failed landing outcomes in drone ship, booster version, and launch site names in the year 2015.

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

To rank the count of landing outcomes or success between the dates specified on the right, in desc order, I utilize Count, Where, Group, and order by to get the results on the right.

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

```
%sql SELECT LANDING_OUTCOME, \
COUNT(LANDING_OUTCOME) AS Landing_Count FROM SPACEXTBL \
WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20' \
GROUP BY LANDING_OUTCOME \
ORDER BY COUNT(LANDING_OUTCOME) DESC;
```

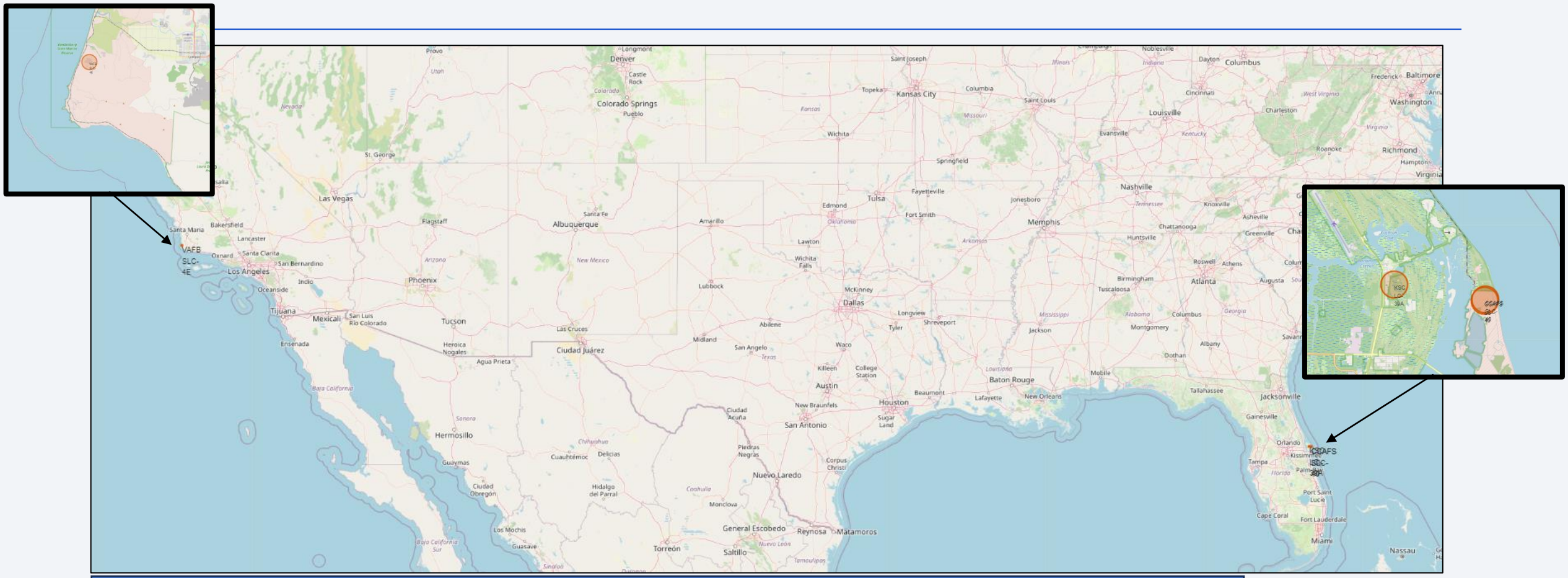
```
 * sqlite:///my_data1.db
Done.
```

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

Section 3

Launch Sites Proximities Analysis

Location of Launch Sites

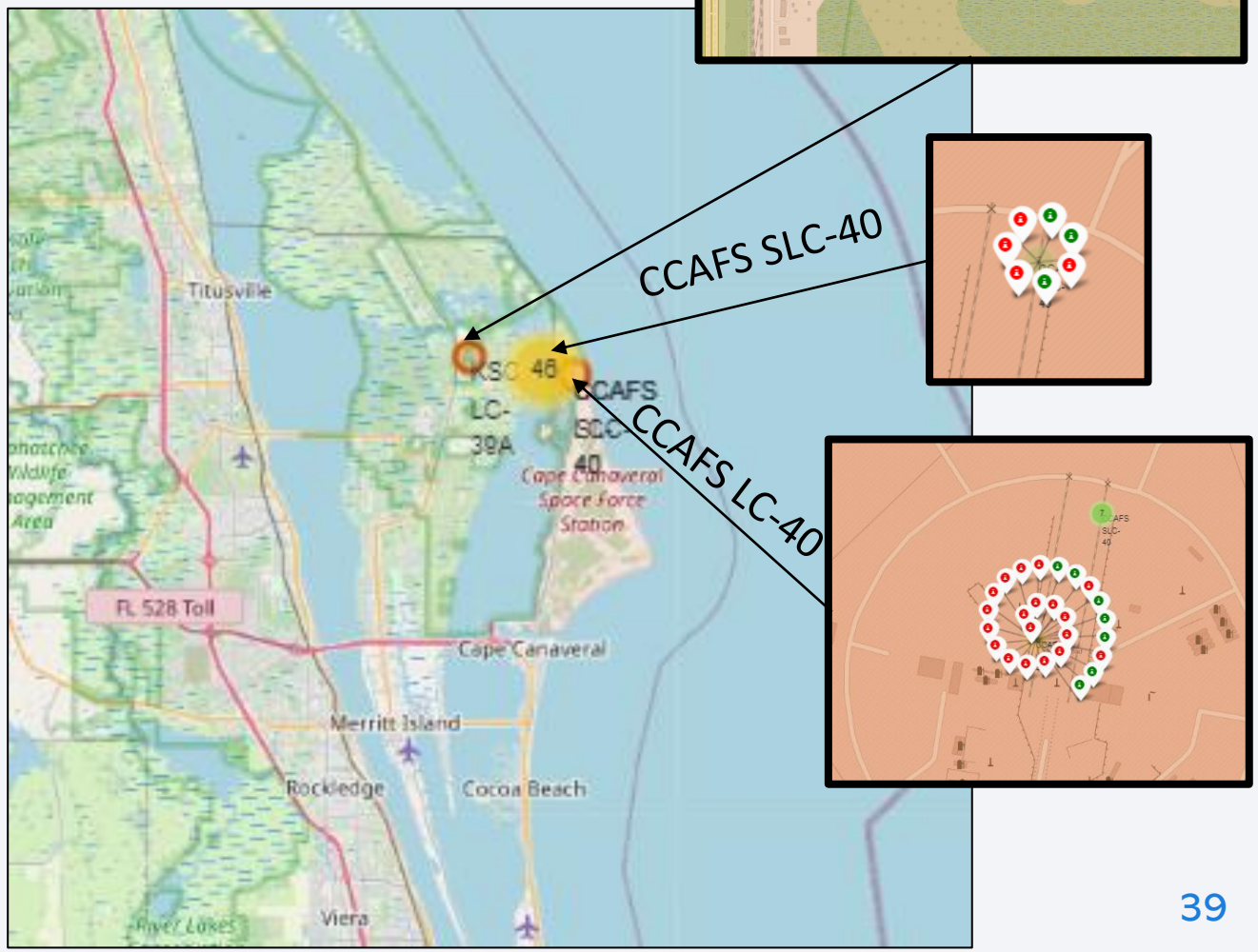


All of the launch sites are on the coast and located in the United States. The launch sites are also near the Equator due to increase the increase of launching capabilities.

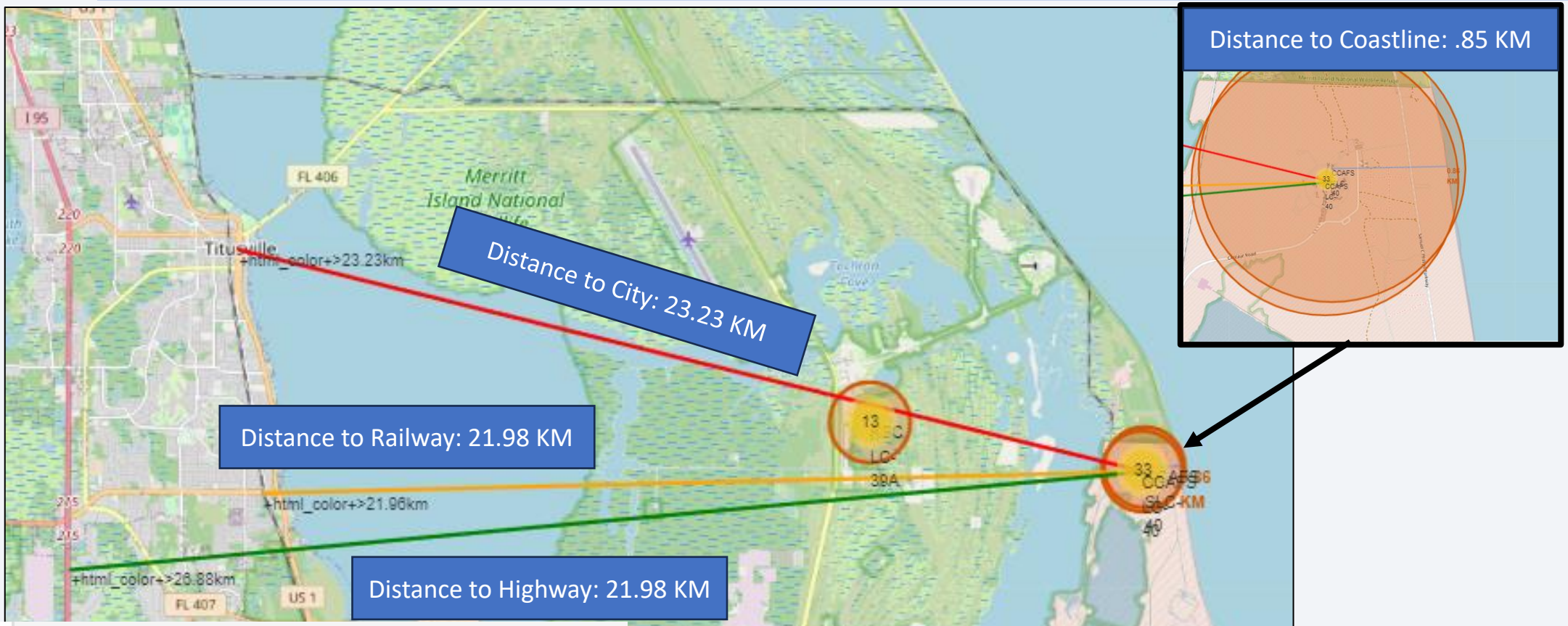
Launch Sites with Success Markers

Florida Launch Sites
- KSC LC39A
- CCAFS SLC-40
- CCAFS LC-40

- **Green Marker** shows **successful Launches**
- **Red Marker** show failuress



Distance from Launch Sites to Landmarks




```
City Distance 23.234752126023245
Railway Distance 21.961465676043673
Highway Distance 26.880385696814 92
Coastline Distance 0.8627671182499878
```

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

Section 5

Predictive Analysis (Classification)

Classification Accuracy

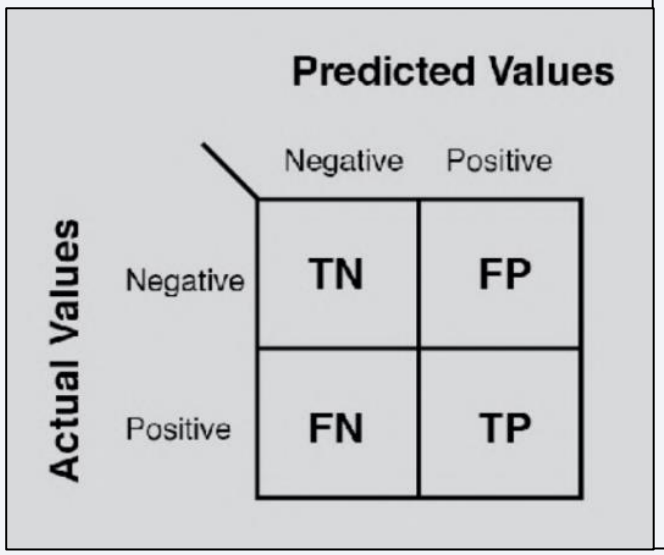
```
models = {'KNeighbors':knn_cv.best_score_,
              'DecisionTree':tree_cv.best_score_,
              'LogisticRegression':logreg_cv.best_score_,
              'SupportVector': svm_cv.best_score_}

bestalgorithm = max(models, key=models.get)
print('Best model is', bestalgorithm,'with a score of', models[bestalgorithm])
if bestalgorithm == 'DecisionTree':
    print('Best params is :', tree_cv.best_params_)
if bestalgorithm == 'KNeighbors':
    print('Best params is :', knn_cv.best_params_)
if bestalgorithm == 'LogisticRegression':
    print('Best params is :', logreg_cv.best_params_)
if bestalgorithm == 'SupportVector':
    print('Best params is :', svm_cv.best_params_)
```

```
Best model is DecisionTree with a score of 0.8767857142857143
Best params is : {'criterion': 'gini', 'max_depth': 4, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 5, 'splitter': 'best'}
```

The highest classification accuracy model is **the Decision Tree model**.
Logreg_cv score: .84
Svm_cv score: 0.84
Tree_cv score: 0.87
Knn_cv_score: 0.83

Confusion Matrix

1.True Positive (TP): The model correctly predicted instances of the positive class.
2.True Negative (TN): The model correctly predicted instances of the negative class.
3.False Positive (FP): The model incorrectly predicted instances as belonging to the positive class when they actually belong to the negative class. Also known as Type I error.
4.False Negative (FN): The model incorrectly predicted instances as belonging to the negative class when they actually belong to the positive class. Also known as Type II error.

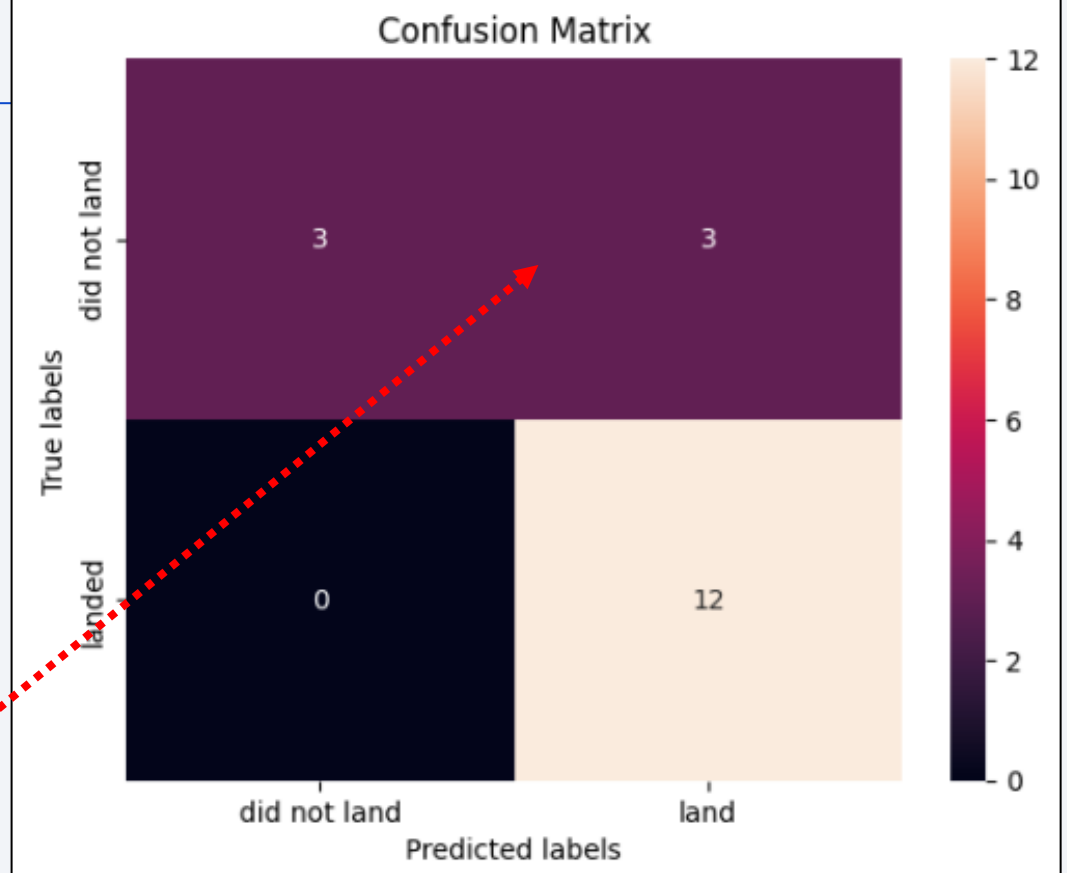


Performance Summary

Confusion Matrix

- 12 TP
- 3 TN
- 3 FP
- 0 FN

```
yhat = tree_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



The Decision Tree Model was the most accurate. All of the confusion matrix were similar. The 3 False positive is alarming because it is an error.

Conclusions

- **Model Performance:** The Decision Tree model performed the best out of the models. The other models were similar in accuracy at around 0.83.
- **Launch Success:** From 2013 to 2020 the success rates for SpaceX launches has increased
- **Orbits:**
  - **Success rate of 100%:** ES-L1, GEO, HEO, SSO, and VLEO had a GTO, ISS, LEO
  - Success rate between 50%-80%: MEO, PO, VLEO
  - 0% success rate: SO
- **Payload Mass vs Success Rate:** There is a higher probability of success rates with payloads over 7,000 kg
- **Launch Site Location:** Near Coastlines and equator

Thank you!