IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Alan Trapenard
January 30, 2024

Outline

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

Executive Summary

- Summary of methodologies
  - Data Collection from SpaceX API
  - Data Collection through Web Scraping
  - Data Wrangling
  - EDA with SQL
  - EDA with Data Viz
  - Geospatial Analytics with Folium
  - Machine Learning Predictive Modelling
- Summary of all results
  - EDA Result
  - Interactive Analytics
  - Predictive Analysis Result

Introduction

- Project background and context
  - SpaceX advertises Falcon 9 rocket launches on its website with a cost of $62 Million; other providers cost upward of $165 Million each, much of the savings is because SpaceX 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 sued if an alternate company wants to bid against SpaceX for a rocket launch
- Problems you want to find answers
  - What factors determine if the rocket will land successfully?
  - The interaction amongst various features that determine the success rate of first stage landing
  - What operating conditions need to be in place to ensure a successful landing.

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - Data was collected using the SpaceX API
  - Data was also collected, using Web Scraping techniques, from Wikipedia.
- Perform data wrangling
  - One-hot encoding (dummy coding) was applied to categorical feature

Methodology

Executive Summary

- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - Split data into training and testing batches
  - Built models (Decision Tree, SVM, and Logistic Regression)
  - Utilized Cross Validation in a Grid Search style to check for best hyperparameters
  - Select model with best accuracy score

Data Collection

- Data Collection Method 1: SpaceX API and SQL
  - GET Request to SpaceX API
  - Decoded response as JSON Object using .json() function -> Turn into Pandas DataFrame using .json_normalize()
  - Necessary data cleaning (check for missing values and fill in missing data)
- Data Collection Method 2: Web Scraping
  - Wikipedia Web Scraping for Falcon 9 launch records using BeautifulSoup
  - Objective: Extract launch records, parse the table objects, convert to DataFrame for further analysis/visualization

Data Collection – SpaceX API

- GET Request to SpaceX API -> Basic Wrangling/Formatting -> Further filtering/cleaning to isolate necessary data

- The link to the notebook is: https://github.com/ATrapenard/IBM_AppliedDataScience_Capstone/blob/main/jupyter-labs-spacex-data-collection-api.ipynb

Now let's start requesting rocket launch data from SpaceX API with the following URL:

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

```
response = requests.get(spacex_url)
```

```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = data[data['BoosterVersion']!='Falcon 1']
```

Now that we have removed some values we should reset the FlgihtNumber column

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

Calculate below the mean for the `PayloadMass` using the `.mean()`. Then use the mean and the `.replace()` function to replace `np.nan` values in the data with the mean you calculated.

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

```
4      6123.547647
5       525.000000
6       677.000000
7       500.000000
8      3170.000000
          ...
89    15600.000000
90    15600.000000
91    15600.000000
92    15600.000000
93     3681.000000
Name: PayloadMass, Length: 90, dtype: float64
```

```
data_falcon9.to_csv('dataset_part_1.csv', index=False)
```

Data Collection - Scraping

- Wikipedia Scraped HTML -> BeautifulSoup Object -> Parse Tables to Dict-> Pandas DataFrame
- The link to the notebook is: https://github.com/ATrapenard/IBM_AppliedDataScience_Capstone/blob/main/jupyter-labs-webscraping.ipynb

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

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

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

```
# 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')
```

```
launch_dict= dict.fromkeys(column_names)

# Remove an irrelvant column
del launch_dict['Date and time ( )']

# Let's initial the launch_dict with each value to be an empty list
launch_dict['Flight No.'] = []
launch_dict['Launch site'] = []
launch_dict['Payload'] = []
launch_dict['Payload mass'] = []
launch_dict['Orbit'] = []
launch_dict['Customer'] = []
launch_dict['Launch outcome'] = []
# Added some new columns
launch_dict['Version Booster']=[]
launch_dict['Booster landing']=[]
launch_dict['Date']=[]
launch_dict['Time']=[]
```

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

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

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

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

Data Wrangling

- Summary information on Orbits and Launch Sites -> Summary information on Landing Outcomes -> Create new 'Class' Feature for subsequent analysis
- The link to the notebook is: https://github.com/ATrapenard/IBM_AppliedDataScience_Capstone/blob/main/labs-jupyter-spacex-Data%20wrangling.ipynb

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

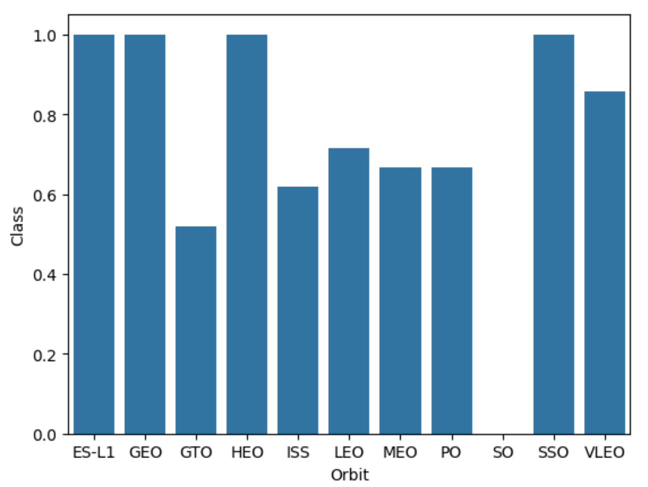
```
# landing_outcomes = values on Outcome column
landing_outcomes = df['Outcome'].value_counts()
landing_outcomes
```

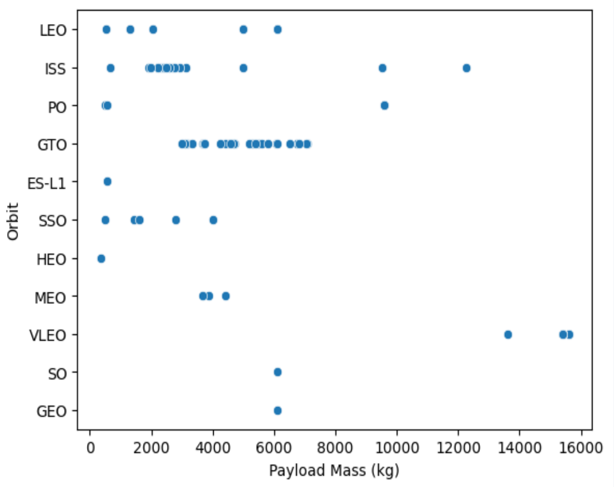
```
True ASDS      41
None None      19
True RTLS      14
False ASDS      6
True Ocean      5
False Ocean     2
None ASDS       2
False RTLS      1
```

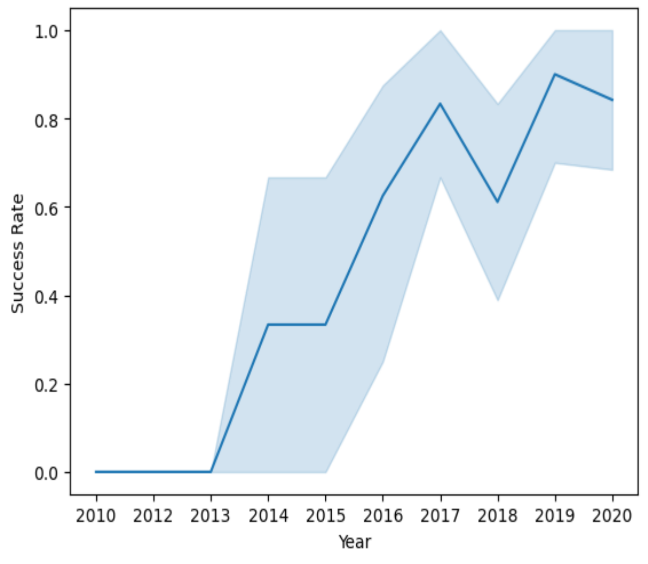
```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
landing_class= []
for i in df['Outcome']:
    if i in bad_outcomes:
        landing_class.append(0)
    else:
        landing_class.append(1)
landing_class
```

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

EDA with Data Visualization







- Our initial visualizations explored relationships between Flight Number vs. Launch Site, Payload vs. Launch Site, Success vs. Orbit, Flight Number vs. Orbit, and the Yearly Launch Success Trends

- The link to the notebook is: https://github.com/ATrapenard/IBM_AppliedDataScience_Capstone/blob/main/jupyter-labs-eda-dataviz.ipynb.jupyterlite.ipynb

EDA with SQL

The link to the notebook is:
https://github.com/ATrapenard/IBM_AppliedDataScience_Capstone/blob/main/jupyter-labs-eda-sql-coursera_sqllite.ipynb

- The SpaceX Data was loaded into a SQLite .db file to create the dataset we are going to reference
- EDA Query Objectives:
    - Names of unique Launch Sites
    - “CCA” Launch Site Records
    - Total Payload Mass carried by boosters with NASA as the associated Customer
    - Average Payload Mass carried by Booster Version F9 v1.1
    - List Booster Versions that have successfully landed on a Drone Ship with designated Payload Mass
    - Total number of successful and failure mission outcomes
    - Booster Versions recorded to carry maximum Payload Mass
    - 2015 Launch Records by Month

Build an Interactive Map with Folium

- We marked all Launch Sites, nearby coastlines, nearby cities, nearby railways, and nearby highways.
- We added Markers to our Launch Sites to be able to visualize the distribution/location
- We used colored marker clusters to identify the Launch Outcomes from each Site
- We used PolyLines to visualize the distance of each Launch Site to their closest coastlines, cities, railways, and highways
- These PolyLines were used to answer the questions
    - Are launch sites near railways? Highways? Coastlines?
    - Do Launch Sites keep a certain distance away from cities?
- The link to this notebook is: https://github.com/ATrapenard/IBM_AppliedDataScience_Capstone/blob/main/lab_jupyter_launch_site_location.jupyterlite.ipynb

Build a Dashboard with Plotly Dash

- Utilized Plotly Dash to create an interactive dashboard for Launch Site statistics
- We used Pie Charts to visualize Launch Outcomes by Site or for all Sites
- We created a Payload Range Slider to dictate the range for the subsequent Scatter Plot
- We used Scatter Plots to visualize the relationship between Launch Site, Launch Outcomes, and Booster Version based on a given Payload Range
- The link to the notebook is: https://github.com/ATrapenard/IBM_AppliedDataScience_Capstone/blob/main/spacex_dash_app.py

Predictive Analysis (Classification)

- After loading the data using Numpy and Pandas, we transformed the data using a Scaler, then split it into Training and Testing groups
- After transforming our cleaned data we created 4 different models (Logistic Regression, Decision Tree, SVM, and KNN)
- These models were put through a GridSearchCV method to extract the best hyperparameters for each model.
- These hyperparameters were then input into each respective model and the accuracies were compared to extract the best performing model
- The link to this notebook is: https://github.com/ATrapenard/IBM_AppliedDataScience_Capstone/blob/main/SpaceX_Machine_Learning_Prediction_Part_5.jupyterlite.ipynb

```
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])
```

```
Best model is DecisionTree with a score of 0.875
```

```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()
logreg_cv = GridSearchCV(lr, parameters, cv=10)
logreg_cv.fit(X_train, Y_train)
```

```
GridSearchCV(cv=10, estimator=LogisticRegression(),
             param_grid={'C': [0.01, 0.1, 1], 'penalty': ['l2'],
                         'solver': ['lbfgs']})
```

```
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, parameters, cv=10)
svm_cv.fit(X_train, Y_train)
```

```
GridSearchCV(cv=10, estimator=SVC(),
             param_grid={'C': array([1.00000000e-03, 3.16227766e-02, 1.00000000e+00, 3.16227766e+01,
       1.00000000e+03]),
                         'gamma': array([1.00000000e-03, 3.16227766e-02, 1.00000000e+00, 3.16227766e+01,
       1.00000000e+03]),
                         'kernel': ('linear', 'rbf', 'poly', 'rbf', 'sigmoid')})
```

```
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)
```

```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18],
                         'max_features': ['auto', 'sqrt'],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'splitter': ['best', 'random']})
```

Results

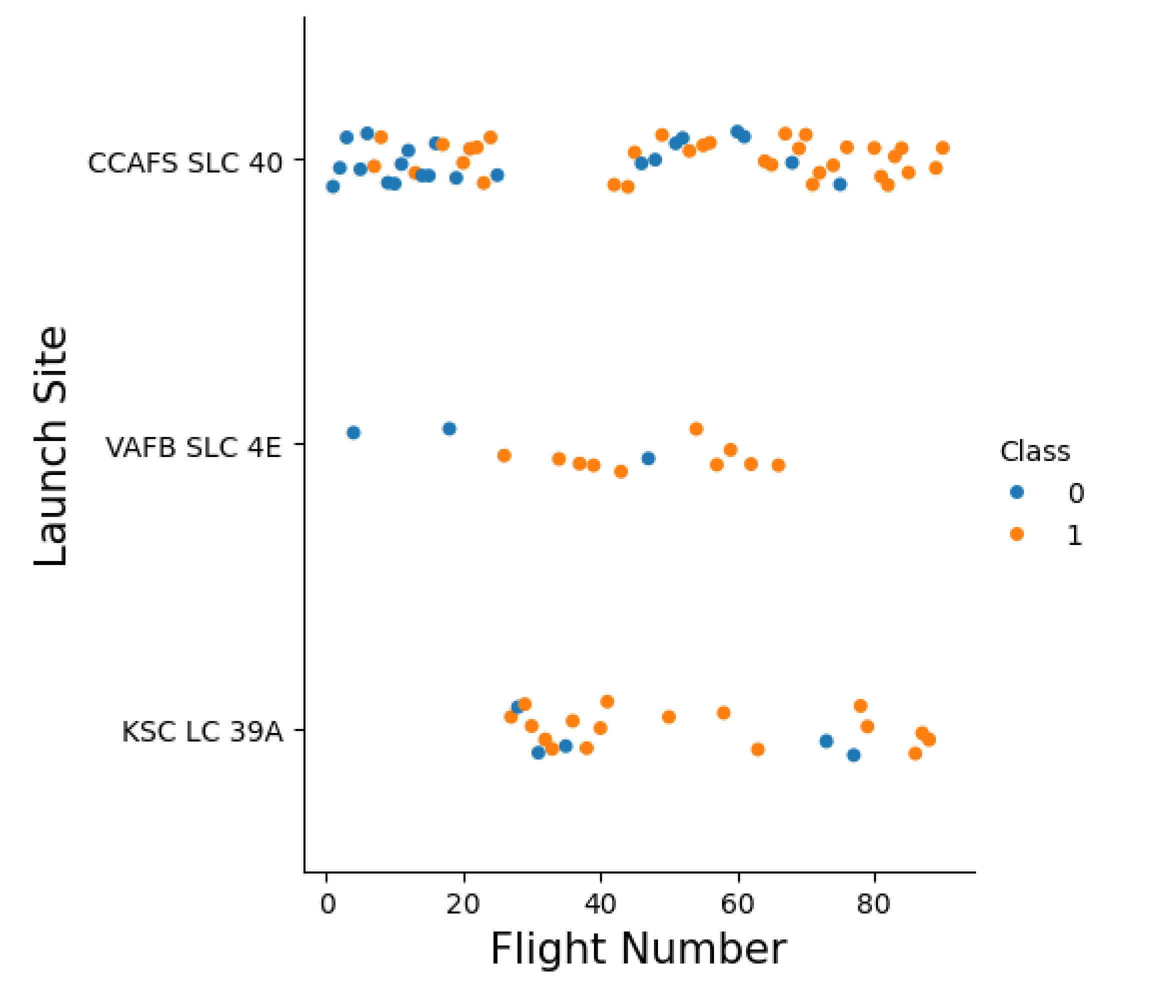
- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results

Section 2

Insights drawn from EDA

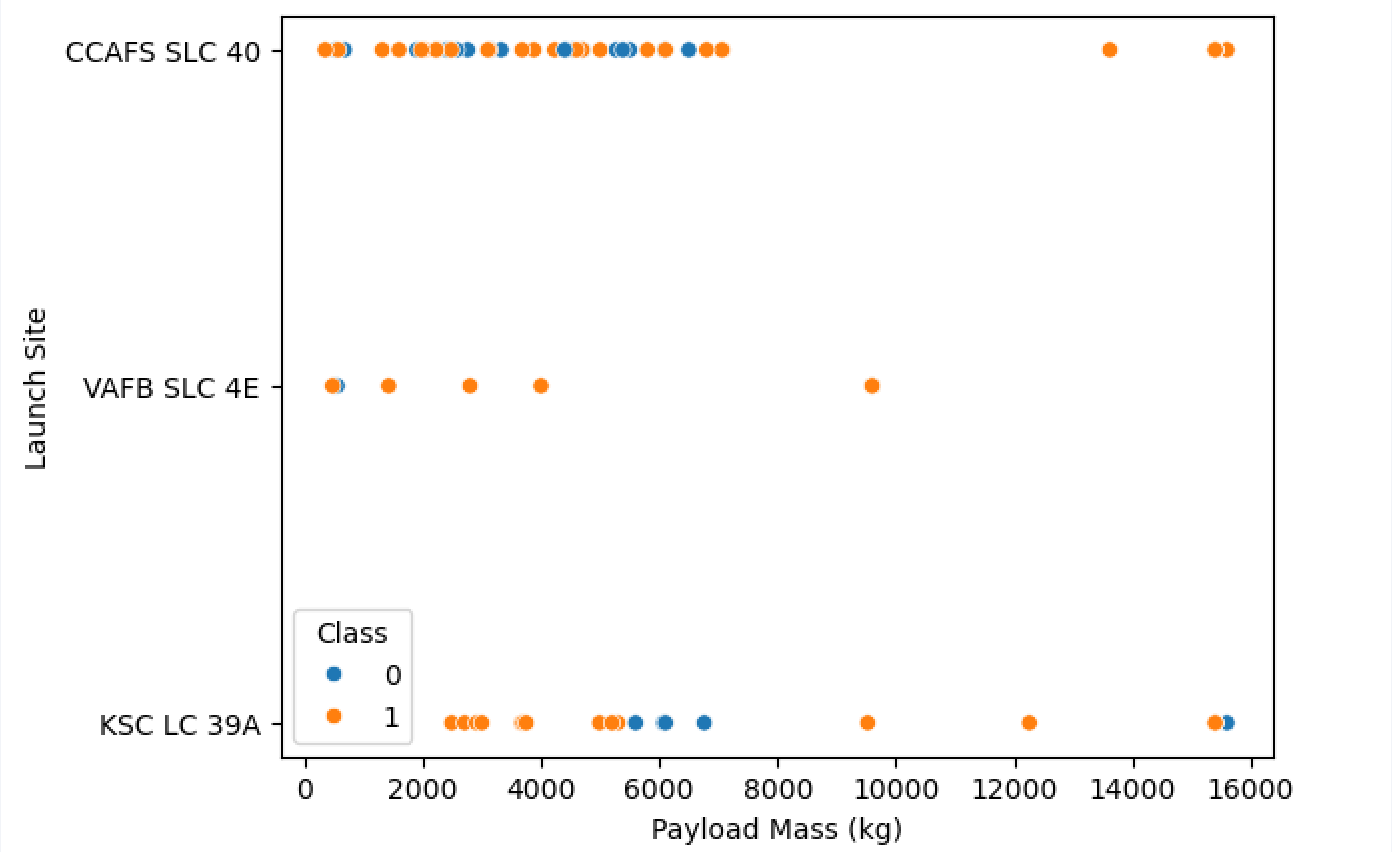
Flight Number vs. Launch Site

- Here we can visualize two main interactions
  - The VAFB Launch Site has experienced the least launches
  - The Launch Sites that have the lowest success ratio have the fewest flights out of them



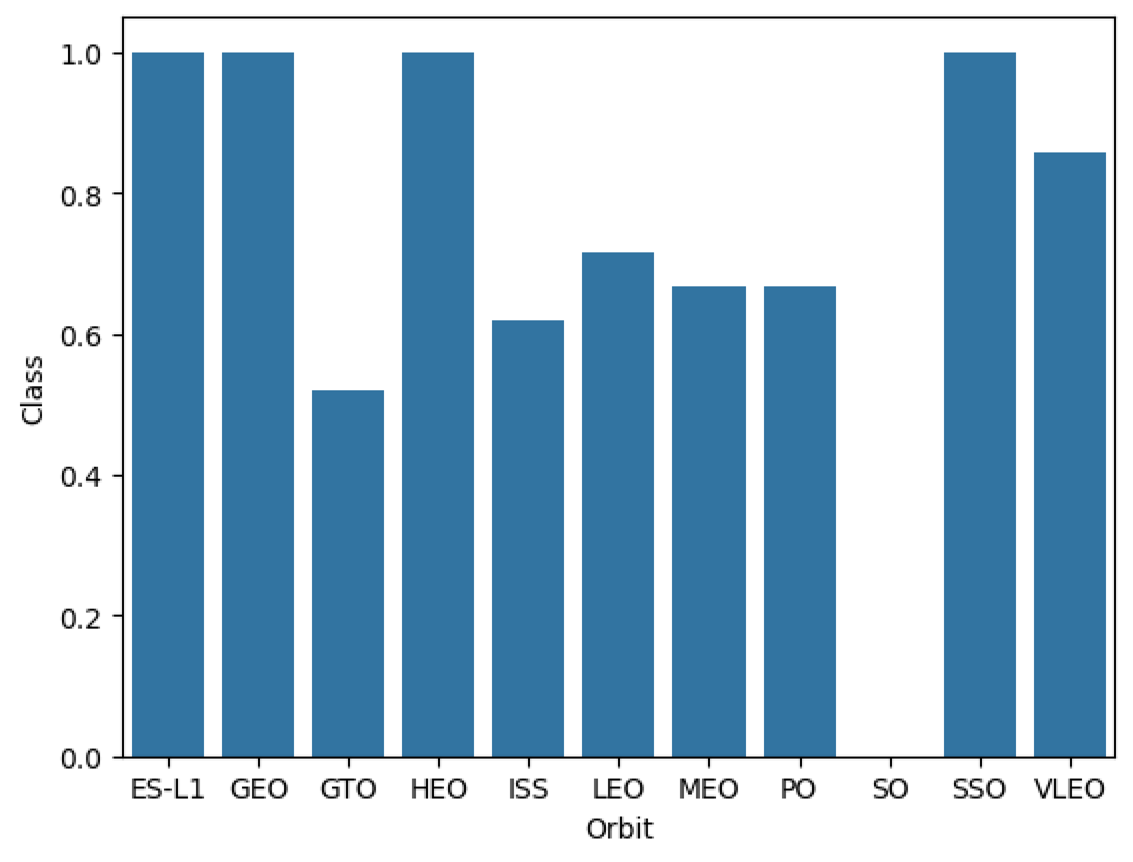
Payload vs. Launch Site

- Two main takeaways from this graph
  - The greater the mass of the payload at Site CCAFS SLC 40 the higher the success rate
  - Site VAFB SLC 4E has never launched a rocket with a payload over 10000kg



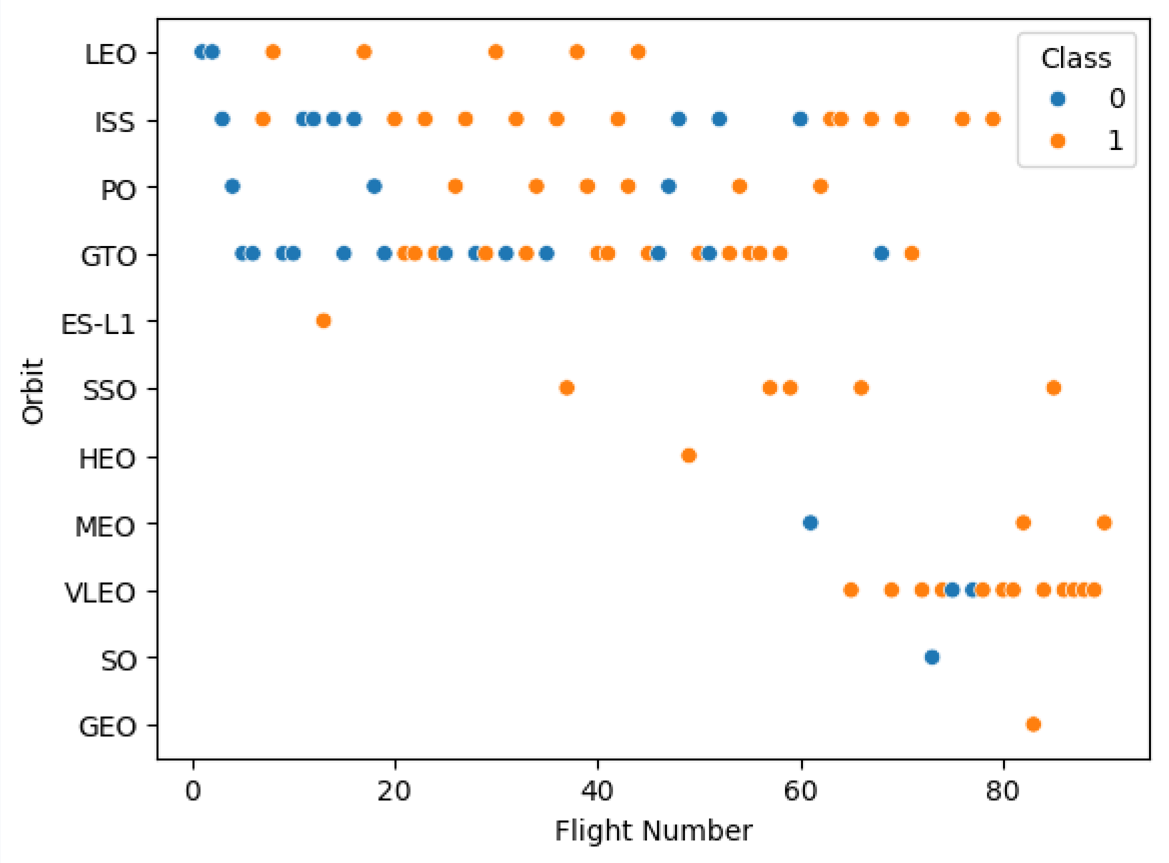
Success Rate vs. Orbit Type

- Main takeaways from this graph:
  - Highest success rate orbits: ES-L1, GEO, HEO, SSO, and VLEO
  - There seems to be an interaction based on Distance from Earth



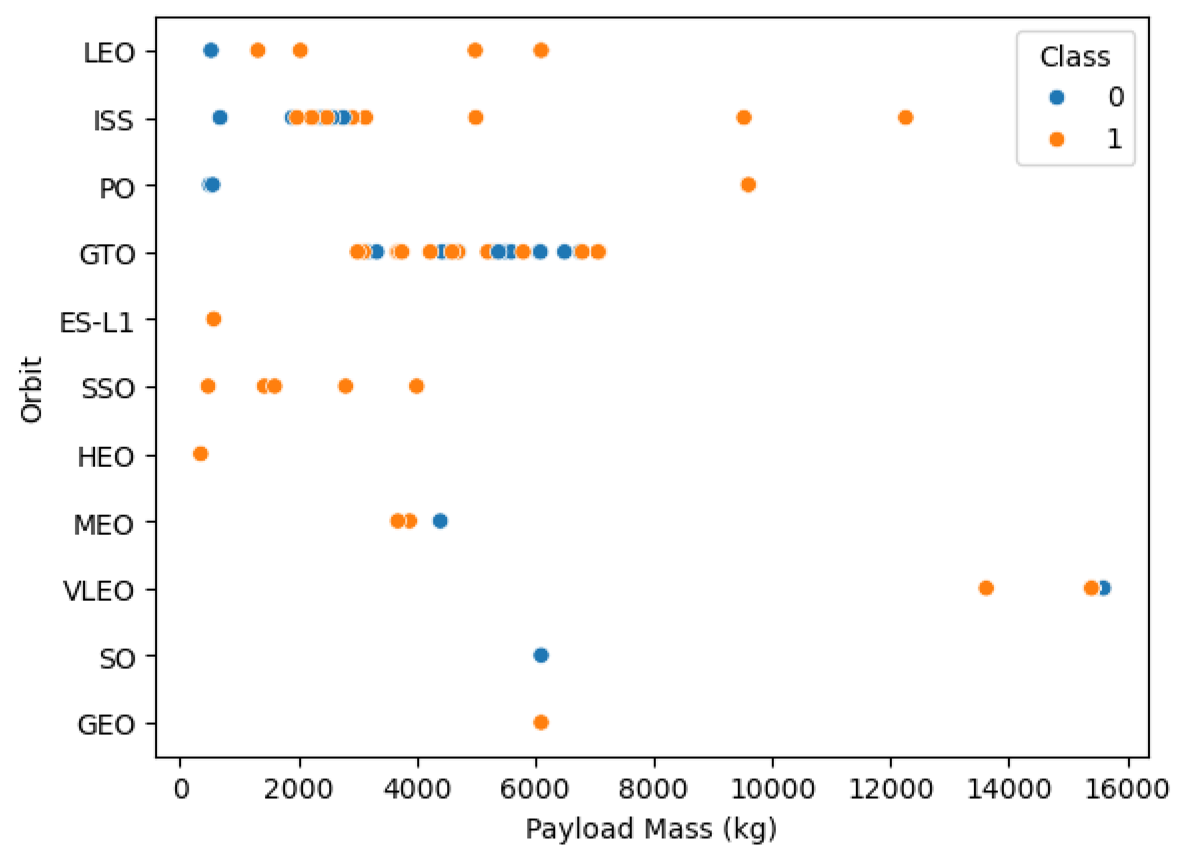
Flight Number vs. Orbit Type

- Main takeaways from this graph:
  - In LEO Orbit, the success rate seems influenced by the number of flights
  - This trend seems to follow until you reach GTO where there doesn't seem to be any interaction



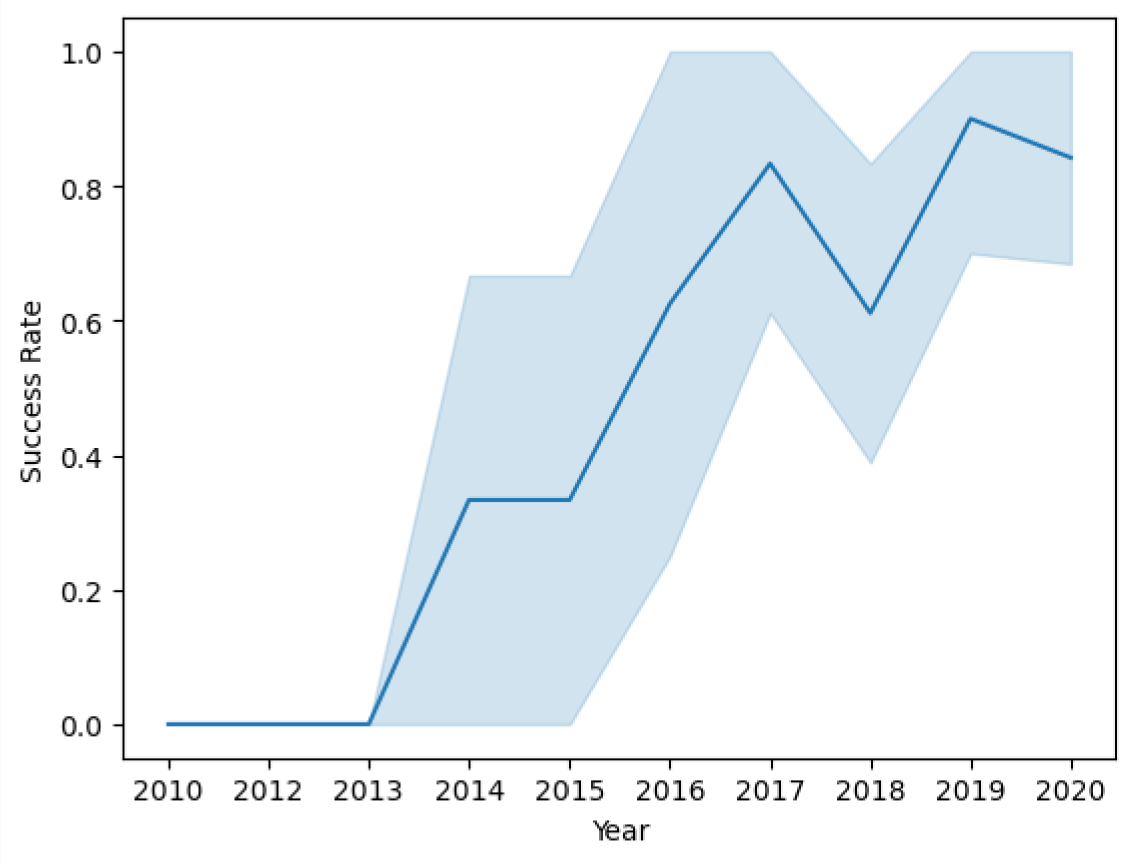
Payload vs. Orbit Type

- Main takeaways from this graph:
  - Polar, LEO, and ISS experienced higher success rates the heavier their Payload was
  - For GTO, this trend is difficult to analyze because the ratio of successes to failures is exactly 50/50



Launch Success Yearly Trend

- Main takeaways from this graph:
  - We can visualize steady increases in success rate from 2013 to 2020



All Launch Site Names

- We used the DISTINCT operator to show the unique Launch Sites from the SpaceX Table we previously created

```
In [8]: %sql select distinct("Launch_Site") from SPACEXTABLE

 * 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'

- We used the LIKE and LIMIT keywords to output the 5 records where the Launch Site contains the string ‘CCA%’

In [9]:
```
%sql select * from SPACEXTABLE where "Launch_Site" like 'CCA%' limit 5
```

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

Out[9]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Total Payload Mass

- We used the SUM() sub-query alongside LIKE to extract the sum of the payload masses for boosters were launched by NASA (CRS)

```
In [10]: %sql select sum(PAYLOAD_MASS__KG_) from SPACEXTABLE where "Customer" like '%NASA (CRS)%'

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

Out[10]:

sum(PAYLOAD_MASS__KG_)
48213

Average Payload Mass by F9 v1.1

- We used the AVG() function to calculate the average payload mass for F9 v1.1 Boosters

```
In [11]: %sql select avg(payload_mass__kg_) from SPACEXTABLE where "Booster_Version" like 'F9 v1.1%'

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

Out[11]:

avg(payload_mass__kg_)
2534.6666666666665

First Successful Ground Landing Date

```
[19]: %sql select "Date" from spacextable where "Date" = (select min("Date") from spacextable where "Landing_Outcome" like "Success (ground pad)")

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

[19]:

Date
2015-12-22

- Here we use the MIN() function to extract the minimum date from the DATETIME objects that satisfy the Success (ground pad) outcome

Successful Drone Ship Landing with Payload between 4000 and 6000

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

In [13]:
```
%sql select "Booster_Version" from spacextable where "Landing_Outcome" = "Success (drone ship)" and payload_mass__kg_>4000 a
```

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

Out[13]:

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

- We used several AND statements to include the criteria for the payload to be within 4000 and 6000
- We also used the WHERE operator to extract only the columns where Landing_Outcome = Success (drone ship)

Total Number of Successful and Failure Mission Outcomes

```
[30]: %sql select count("Landing_Outcome") as SuccessCount from spacextable where "Landing_Outcome" like 'Success%'

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

[30]:

SuccessCount
61

```
[31]: %sql select count("Landing_Outcome") as FailureCount from spacextable where "Landing_Outcome" like 'Failure%'

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

[31]:

FailureCount
10

- We used the COUNT() function as the main method to count the number of Successes and Failures respectively

Boosters Carried Maximum Payload

```
In [15]: %sql select "Booster_Version" from spacextable where payload_mass__kg_ = (select max(payload_mass__kg_) from spacextable)

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

Out[15]:

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

- We used a subquery to extract the Max payload
- We then filtered the spacextable by Booster Version for only those who have records carrying the Max Payload

2015 Launch Records

```
In [16]: %sql select substr(Date, 6,2) as Month, "Landing_Outcome" = 'Failure (drone ship)', "Booster_Version", "Launch_Site" from sp

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

Out[16]:

Month	"Landing_Outcome" = 'Failure (drone ship)'	Booster_Version	Launch_Site
01	1	F9 v1.1 B1012	CCAFS LC-40
02	0	F9 v1.1 B1013	CCAFS LC-40
03	0	F9 v1.1 B1014	CCAFS LC-40
04	1	F9 v1.1 B1015	CCAFS LC-40
04	0	F9 v1.1 B1016	CCAFS LC-40
06	0	F9 v1.1 B1018	CCAFS LC-40
12	0	F9 FT B1019	CCAFS LC-40

- We used the substr() method to extract the Month value from our Dates
- We then filtered the dataset to only include those with year values from 2015
- Finally, we listed the required categories by filtering the dataset even further

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

In [17]:
```
%sql select "Landing_Outcome", count("Landing_Outcome") from spacextable where "Date" between '2010-06-04' and '2017-03-20'
```

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

Out[17]:

Landing_Outcome	count("Landing_Outcome")
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

- We used the BETWEEN operator and the AND operator to filter this set. BETWEEN was used to filter for our dates and AND was used to join the two DATE values we're trying to filter based on

Section 3

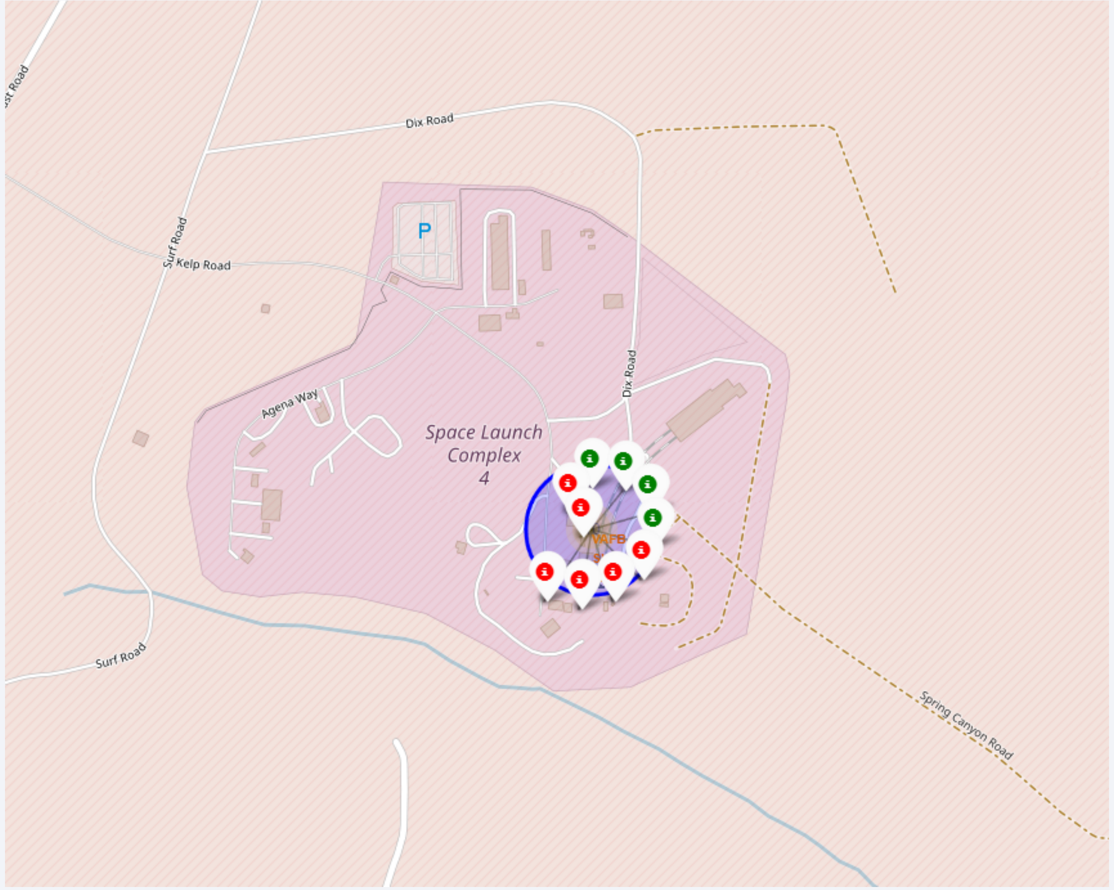
Launch Sites Proximities Analysis

Global Launch Sites

- Here we can see that the SpaceX Launch Sites are in the United States clustered on the coasts of California and Florida

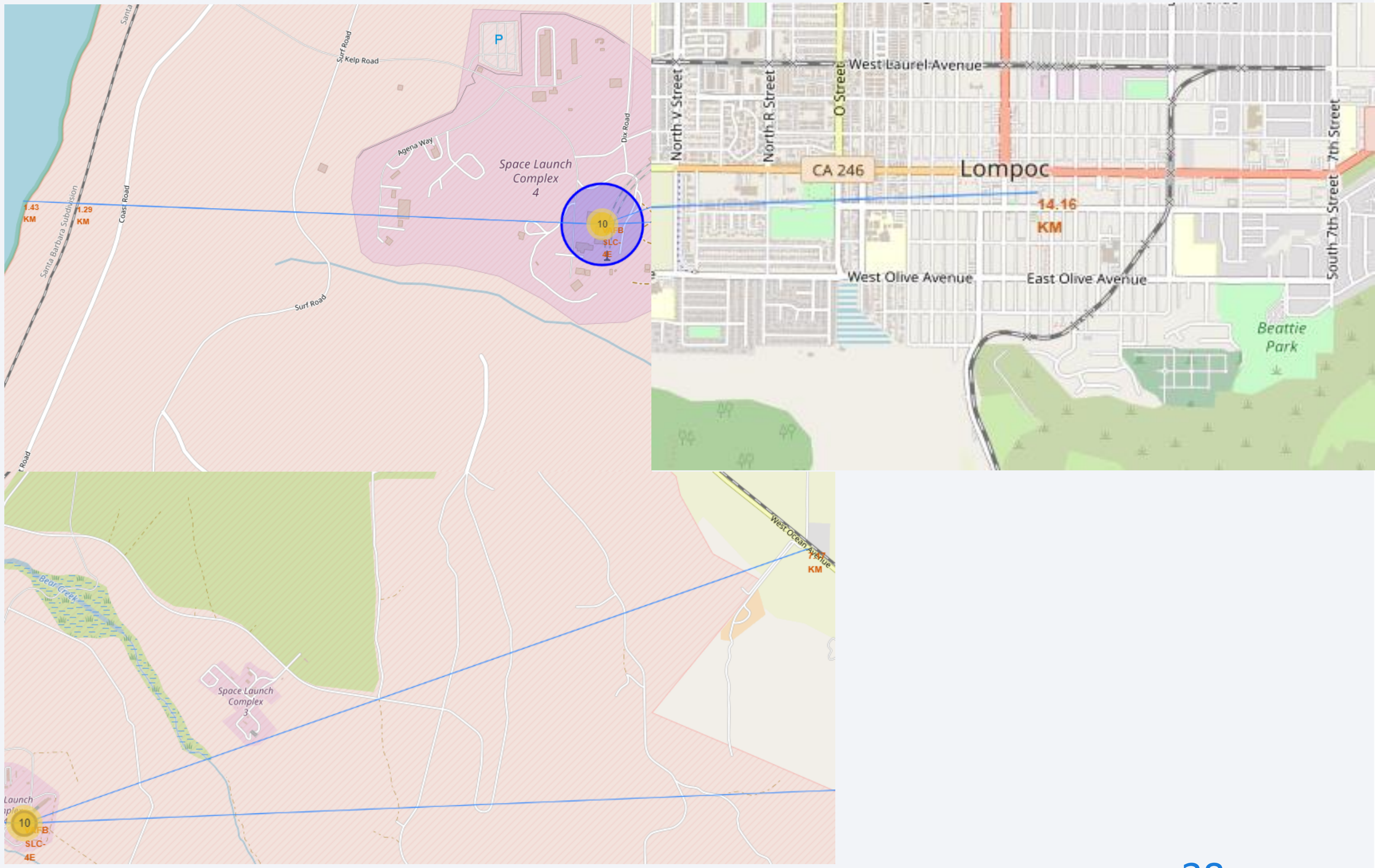
Color Labeled Launch Outcome Marker Clusters

- California Launch Site as example
- Green Markers show successful Launches
- Red Markers show failed Launches



Closest Infrastructure to Launch Sites

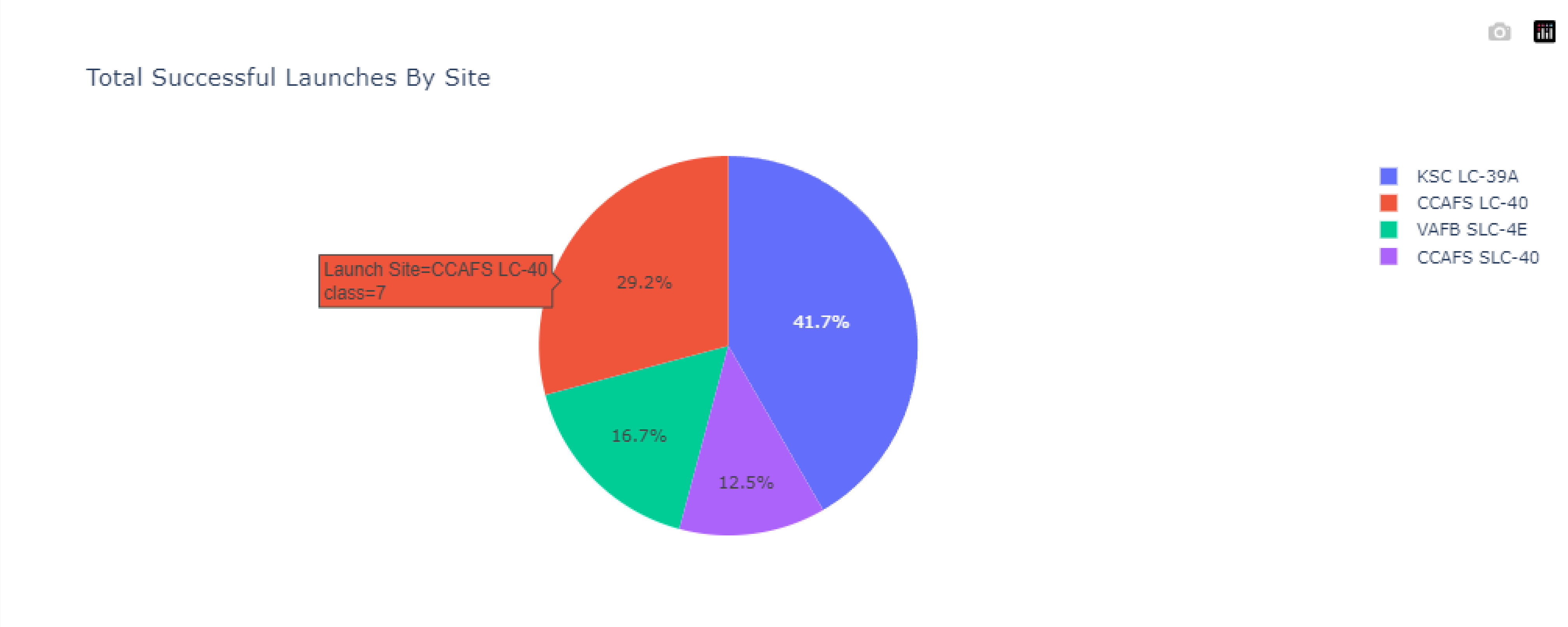
- Visualized here are the proximity lines that show the distance from the Launch Site to the nearest City, Railway, Coastline, and Highway to answer the questions:
  - Are launch sites in close proximity to railways? The one in California is
  - Are launch sites in close proximity to highways? No
  - Are launch sites in close proximity to coastline? Yes
  - Do launch sites keep certain distance away from cities? Yes



Section 4

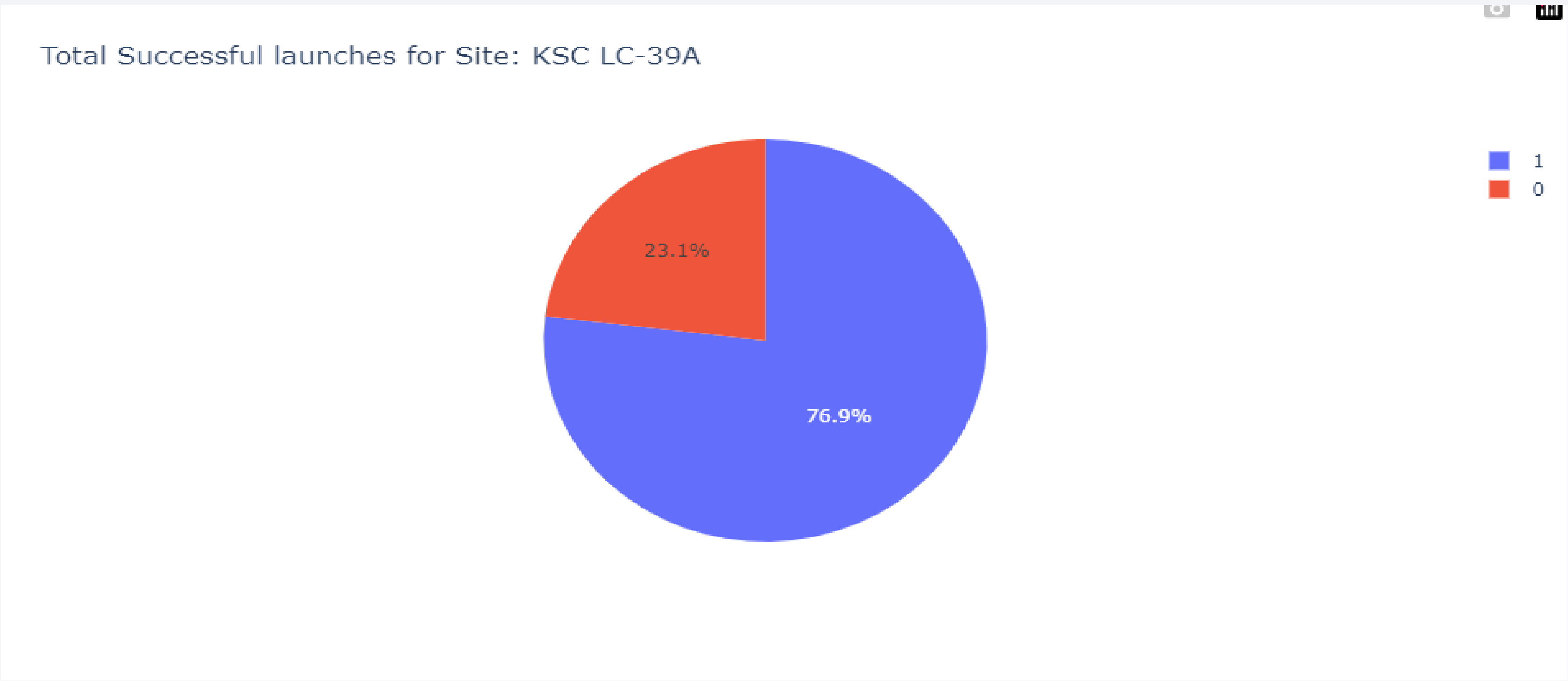
Build a Dashboard with Plotly Dash

Launch Success Ratio by Launch Site



- Here we can see that KSC LC-39A has the highest Launch Success Ratio out of all the Launch Sites
- CCAFS SLC-40 has the lowest

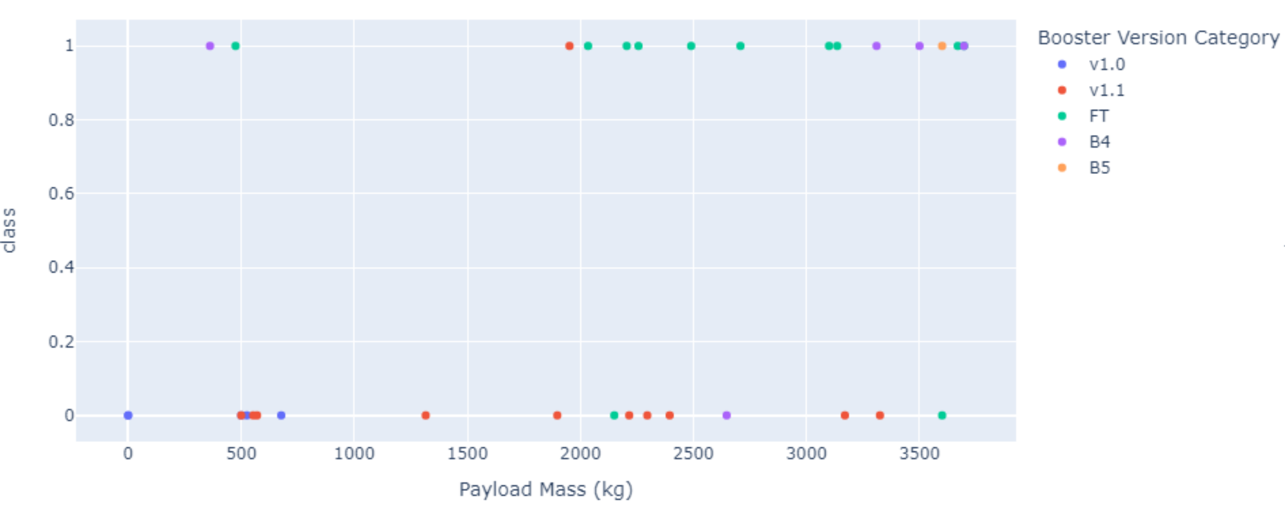
Highest Launch Success Ratio



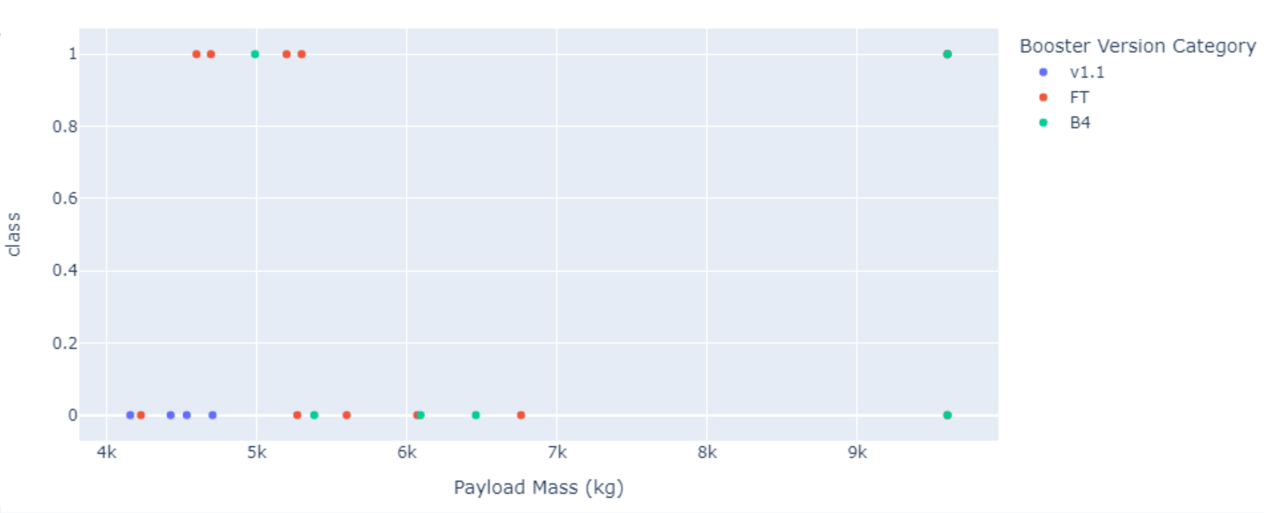
- KSC LC-39A achieved a 76.9% Launch Success Rate and a 23.1% Launch Failure Rate giving it the highest Success Rate among the Launch Sites

Payload Range vs. Launch Outcome

Light Payload Range: 0-4000kg



Heavy Payload Range: 4000-10000kg



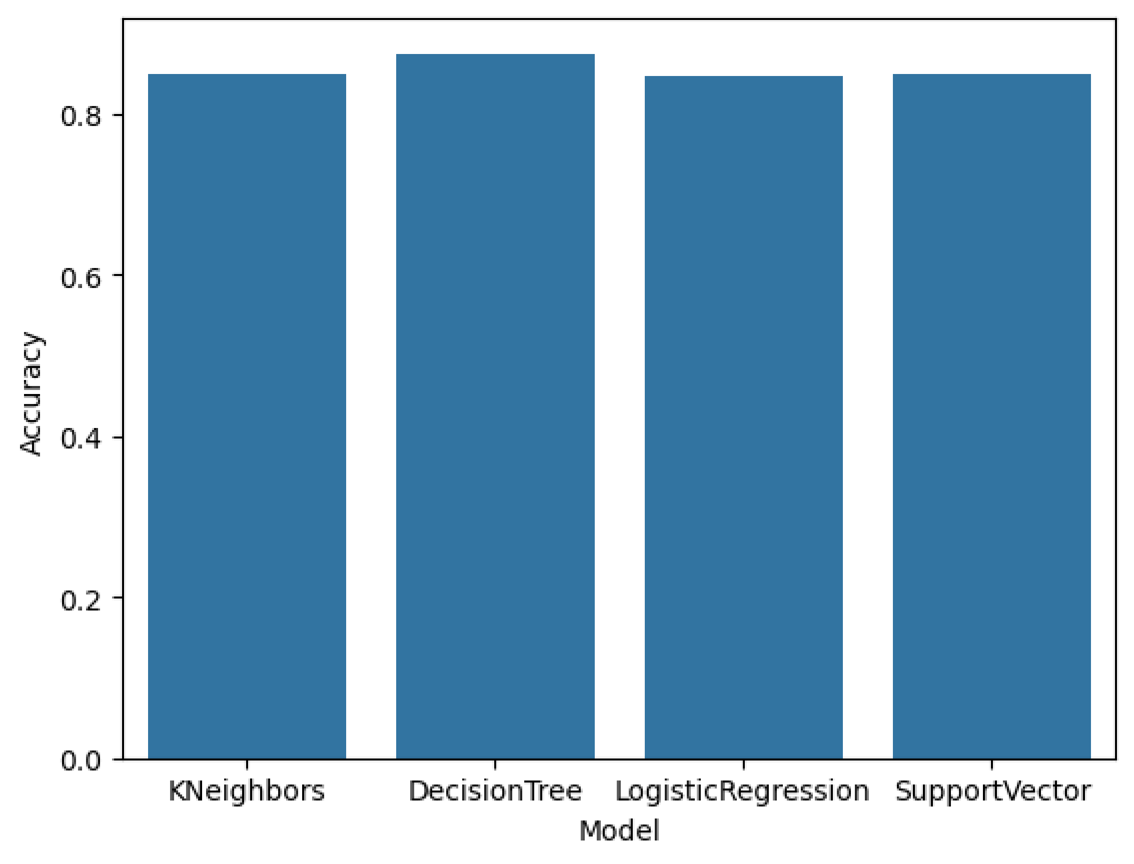
- Here we can see that the heavier the mass is, the lower the chance of a Launch Success
- We can also see that the FT Booster at low Payload mass has a high success rate but an average success rate at higher Payload masses.

Section 5

Predictive Analysis (Classification)

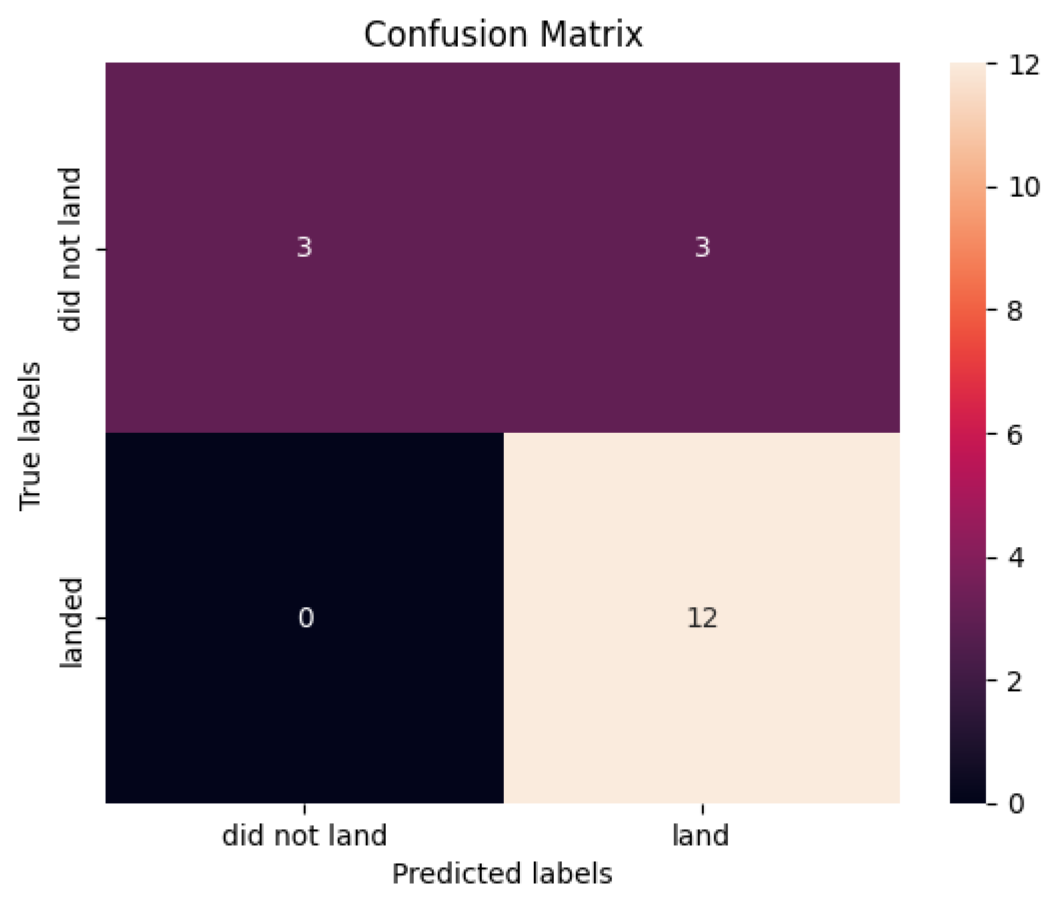
Classification Accuracy

- After building, training, and testing the models this bar chart visualizes the Model Accuracy for each tested model
- We can see that while all retain a high (80%+) average accuracy, the DecisionTree model has the highest.



Confusion Matrix

- This model performs well on classifying True Negatives and True Positives but lacks predictive power when it comes to the False Positives
- We can see that the model predicts 1/5th of the landings as Successful that were failures



Conclusions

- The lower the Payload Mass (kg) the higher the Success Rate
- Yearly Launch Success Rate increased steadily from 2013 to 2020 after which it began to fall off
- KSC LC-39A appears to be the most successful Launch Site
- Utilizing a DecisionTree classifier is the best tested Machine Learning Algorithm for this predicting Launch Success
- Orbits ES-L1, GEO, HEO, SSO, and VLEO had the highest success rate

Thank you!