IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Dylan
November 1, 2022

Slide Outline

Executive Summary

- Data was collected using SpaceX API and webscraping
- Data wrangling included accounting for missing values and encoding categorical data
- Exploratory data analysis included visualization with charts and sql queries
- Interactive visual analysis was done with Folium maps and a Plotly Dashboard
- Predictive analysis was done with multiple machine learning methods

Introduction

Project Background

A new rocket company, Space Y, would like to compete with SpaceX. To aid in estimating costs for each launch, we developed a machine learning model that uses launch parameters to predict how likely it is for SpaceX to reuse the first stage of a rocket (aka “booster”). Elon Musk has stated that fabrication of the first stage accounts for 60% of a rocket’s manufacturing cost, but refurbishing a successfully recovered booster reduces this cost to less than 10%. Identifying which launch parameters generate the highest likelihood of booster recovery can generate considerable cost savings for Space Y.

Slide titles that are underlined contain hyperlinks to individual notebooks. Full link here: https://github.com/ibmDataScienceCourse/finalProject

Section 1

Methodology

Methodology

- Data collection:
  - SpaceX API and webscraping
- Data Wrangling:
  - Replaced missing values with mean of column and encoded categorical variables
- Performed exploratory data analysis (EDA) using visualization and SQL
- Performed interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - Used GridSearchCv on multiple learning methods and choose method with best score

Data Collection – SpaceX API

- Requested static JSON of SpaceX launch data from SkillsNetwork API
  - Normalized response into pandas DataFrame (df) and selected features of interest
  - Most data were ID numbers, rather than information
- Passed IDs through helper functions to extract required information from SpaceX API
  - Appended data for features into individual lists
  - Created df from dictionary of lists

```python
# Example helper function. Initialized in later code: BoosterVersion = []
def getBoosterVersion(data):
    for x in data['rocket']:    # data['rocket'] contains IDs for booster versions
       if x:
        response = requests.get("https://api.spacexdata.com/v4/rockets/"+str(x)).json()
        BoosterVersion.append(response['name'])
```

Data Collection – SpaceX API

1. Request Static JSON

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

2. Normalize Response

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

3. Select Features of Interest

```
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]
data = data[data['cores'].map(len)==1]
data = data[data['payloads'].map(len)==1]
data['cores'] = data['cores'].map(lambda x : x[0])
data['payloads'] = data['payloads'].map(lambda x : x[0])
data['date'] = pd.to_datetime(data['date_utc']).dt.date
data = data[data['date'] <= datetime.date(2020, 11, 13)]
```

4. Initialize Lists

```
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
ReusedCount = []
Serial = []
Longitude = []
Latitude = []
```

5. Call Functions

```
getBoosterVersion(data)
getLaunchSite(data)
getPayloadData(data)
getCoreData(data)
```

6. Create DataFrame

```
launch_dict = {'FlightNumber': list(data['flight_number']),
'Date': list(data['date']),
'BoosterVersion':BoosterVersion,
'PayloadMass':PayloadMass,
'Orbit':Orbit,
'LaunchSite':LaunchSite,
'Outcome':Outcome,
'Flights':Flights,
'GridFins':GridFins,
'Reused':Reused,
'Legs':Legs,
'LandingPad':LandingPad,
'Block':Block,
'ReusedCount':ReusedCount,
'Serial':Serial,
'Longitude': Longitude,
'Latitude': Latitude}

data = pd.DataFrame(launch_dict)
```

7. Remove Falcon 1 Launches

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

Data Collection – Scraping

- Requested Wiki page from static URL
- Created BeautifulSoup (bs4) object from HTML response
  - Created list of all table elements with .find_all() method
  - Assigned target table to a variable with list indexing
- Extracted column names by iterating over table headers in target table
- Created dictionary with column names as keys
  - Values were initialized as empty lists
  - Irrelevant column was removed
  - Additional columns were added
- Parsed soup object to fill dictionary
  - DataFrame was created from dictionary

Data Collection – Scaping

1. Request Wiki page from static URL

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

2. Create BeautifulSoup object

```
soup = BeautifulSoup(response.content, 'html.parser')
```

3. Find all table elements

```
html_tables = soup.find_all('table')
```

4. Identify target table

```
first_launch_table = html_tables[2]
print(first_launch_table)
```

5. Iterate to extract column names

```
column_names = []
for header in first_launch_table.find_all('th'):
    name = extract_column_from_header(header)
    if name != None and len(name)>0:
        column_names.append(name)
```

6. Initialize dictionary

```
launch_dict = dict.fromkeys(column_names)
# Remove irrelvant column
del launch_dict['Date and time ( )']
# Initialize dict with empty lists
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'] = []
# Add additional columns
launch_dict['Version Booster']=[]
launch_dict['Booster landing']=[]
launch_dict['Date']=[]
launch_dict['Time']=[]
```

7. Parse soup object to fill dictionary (partial code below)

```
extracted_row = 0
for table_number,table in enumerate\
(soup.find_all('table',"wikitable plainrowheaders collapsible")):
    for rows in table.find_all("tr"):
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
        else:
            flag=False
        row=rows.find_all('td')
        if flag:
            extracted_row += 1
            launch_dict['Flight No.'].append(flight_number)
            datatimelist=date_time(row[0])
            date = datatimelist[0].strip(',')
```

8. Convert to DataFrame

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

Data Wrangling – Missing Values

- Identified columns with missing values using df.isnull().sum()
  - Returns a pandas Series object with number of NaN values in each column
  - Missing values for PayloadMass were filled with column's mean

1. Identify columns with missing data

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

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

2. Calculate mean of column

```
payload_mean = data_falcon9['PayloadMass'].mean()
```

3. Replace missing values with mean

```
data_falcon9['PayloadMass'] = data_falcon9['PayloadMass'].replace(np.nan, payload_mean)
```

Data Wrangling – Encoding Categorical Data

- Identified data types for each column with df.dtypes
- Assigned .value_counts() of Outcome column to variable
  - Method returns a Series object with counts for each unique value
  - Created set containing labels of unsuccessful outcomes by indexing their keys
- Iterated over Outcome column to compare elements to set
  - Initialized empty list to hold encoded data
  - If row's element was in set, 0 was appended to list
  - If element was not in set, 1 was appended
- List added to frame as column Class to represent if a launch was successful or not

Data Wrangling – Encoding Categorical Data

1. Identify data types of columns

```
df.dtypes
```

```
FlightNumber        int64
Date               object
BoosterVersion     object
PayloadMass       float64
Orbit              object
LaunchSite         object
Outcome            object
Flights             int64
GridFins             bool
Reused               bool
Legs                 bool
LandingPad         object
Block             float64
ReusedCount         int64
Serial             object
Longitude         float64
Latitude          float64
```

2. Get .value_counts() of Outcome

```
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
```

3. Get indices of keys

```
for i,outcome in enumerate(landing_outcomes.keys()):
    print(i,outcome)
```

```
0 True ASDS
1 None None
2 True RTLS
3 False ASDS
4 True Ocean
5 False Ocean
6 None ASDS
7 False RTLS
```

4. Create set of unsuccessful outcomes

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

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

5. Compare to Outcome column

```
landing_class = []
for key, value in df['Outcome'].items():
    if value in bad_outcomes:
        landing_class.append(0)
    else:
        landing_class.append(1)
```

6. Add to DataFrame

```
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

- Used multiple types of plots to visualize relationships between variables
- Categorical Plots – for plotting relationships between numeric variable(s) and at least one categorical variable
- Bar Chart – uses rectangular bars to represent categorical data with heights proportional to a numeric value
- Line Chart – plots a series of data points connected by a line to display changes over time

EDA with Data Visualization – Categorical Plots

hue = ‘Class’ was set as categorical variable for all charts

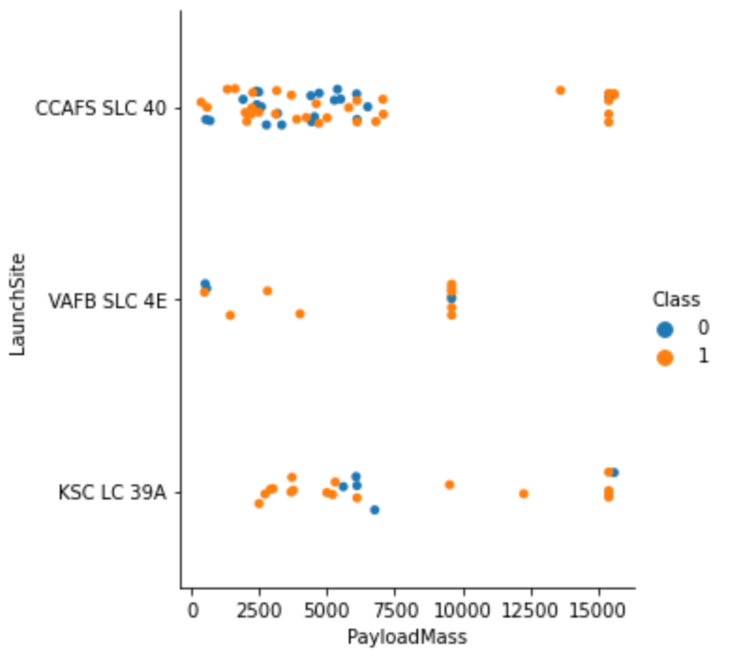
- Showed whether the relationships between variables effected landing success
- Launch Site or Orbit were additional categorical variables on some charts
- Relationships plotted:

X		Y
Flight Number	and	Payload Mass
Flight Number	and	Launch Site
Payload Mass	and	Launch Site
Flight Number	and	Orbit
Payload Mass	and	Orbit

Example Question:
Does SpaceX have different success with launching certain payload masses from particular launch sites?

```
sns.catplot(data=df, x="PayloadMass", y="LaunchSite", hue="Class")
```

```
<seaborn.axisgrid.FacetGrid at 0x2bdaf560408>
```

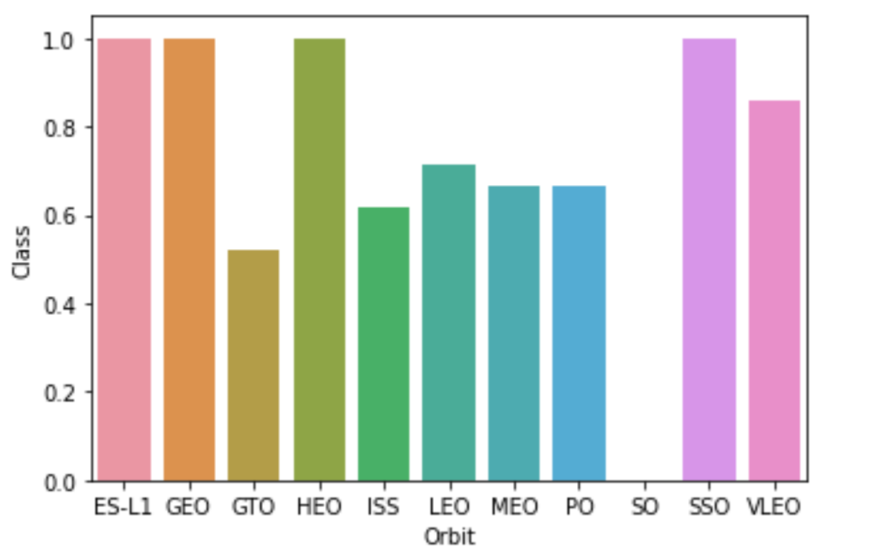


EDA with Data Visualization – Bar & Line Chart

Bar Chart – Visualized relationships between type of orbit and success rate

```
sns.barplot(data=df.groupby('Orbit', as_index=False)\
            ['Class'].mean(), x='Orbit', y='Class')
```

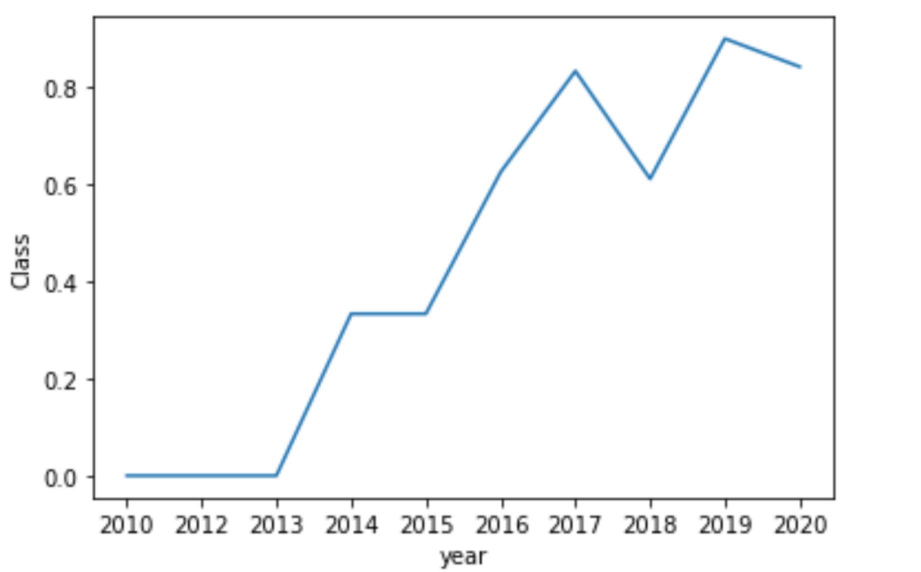
```
<AxesSubplot:xlabel='Orbit', ylabel='Class'>
```



Line Chart – Showed yearly trend in landing success rate

```
sns.lineplot(data=df2.groupby('year', as_index=False)\
             ['Class'].mean(), x='year', y='Class')
```

```
<AxesSubplot:xlabel='year', ylabel='Class'>
```



EDA with SQL – Summary of Queries

- Names of the unique launch sites
- 5 records where launch sites started with 'CCA’
- Total payload mass carried by boosters launched by NASA (CRS)
- Average payload mass carried by booster version F9 v1.1
- Date of first successful ground pad landing
- Names of boosters that have successfully landed on a drone ship with a payload mass between 4000 and 6000 kilograms
- Total number of successful and failed mission outcomes
- Names of booster versions that have carried the maximum payload mass
- Records of failed drone ship landings in 2015 and their booster versions, launch sites, and months launched
- Ranked types of landing outcomes by their count of successful launches between June 4, 2010 and March 20, 2017 from greatest to least

Build an Interactive Map with Folium

1. Folium Map Object – initialized base map with location of NASA Johnson Space Center and zoom level of 5. Marker for location was not added to main map, but it served as a good midpoint between launch sites to initialize Map object with.
2. Folium.Circle – feature to add circular overlays on map. Placed at coordinates for each launch site, with Popup child to display the launch site's name when clicked.
3. Folium.Marker – feature to mark names and HTML on map, which was used to display site names without needing to click. Marked coordinates and set icon parameter to folium.DivIcon feature to display site names as lightweight html elements, rather than images.

Build an Interactive Map with Folium

5. MarkerCluster Object – plugin to add animated marker clustering. Added color-coordinated markers to each site that indicate outcomes of their launches, with green being successful and red being failed. Cluster displays total number of launches from a particular area. Clusters combine when zoomed out. Clicking on cluster will zoom into the individual clusters. Clicking on individual clusters show the color-coordinated markers for that site.
6. MousePosition – added field in top right to display coordinates for where user's mouse pointer is on map.
7. Folium.Marker for distances – marked the nearest coast line, railroad, highway, and city using coordinates from MousePosition. Obtained distance between launch site and individual markers using function, displayed each on map.
8. PolyLine – drew line connecting launch site and distance marker

Build a Dashboard with Plotly Dash

Interactions

Dropdown: Allows user to choose data for either a specific launch site or all sites to be displayed on charts.

RangeSlider: Allows user specify the range of payload masses for scatter chart.

Charts

Pie Chart: For seeing how success rates differed between launch sites. If a specific site was selected, it shows what percent of their launches were successful. If all sites selected, it shows the share of total successful launches that each site is responsible for.

Scatter Chart: To see if a correlation between payload mass and booster version effected success rate. Booster version determined the point's color, which was plotted with payload mass along the X axis and class along the Y axis.

Predictive Analysis (Classification)

Building
- Split data into testing and training sets
- Performed GridSearchCV on multiple learning methods to test different parameters

Evaluation
- Found best parameters and best score for each GridSearchCV object
- Plotted confusion matrix for each model

Selecting Best Model
- Added model names and best score dictionary, selected model with max score

Predictive Analysis Flowchart - Preparation

1. Load data set as X

```
X = pd.read_csv('https://cf-courses-
```

2. Initialize Y as array

```
Y = data['Class'].to_numpy()
```

3. Standardize and transform X

```
X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))
```

4. Split into training and testing sets

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

Predictive Analysis Flowchart – Development

5. Perform GridSearches

```
logreg_cv = grid_search_lr.fit(X_train,Y_train)
svm_cv = grid_search_svm.fit(X_train,Y_train)
tree_cv = grid_search_tree.fit(X_train, Y_train)
knn_cv = grid_search_knn.fit(X_train, Y_train)
```

6. Find best parameters and score

```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
print("tuned hpyerparameters :(best parameters) ",svm_cv.best_params_)
print("accuracy :",svm_cv.best_score_)
print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)
print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
print("accuracy :",knn_cv.best_score_)
```

7. Calculate accuracy for test data

```
logreg_cv.score(X_test, Y_test)
svm_cv.score(X_test, Y_test)
tree_cv.score(X_test, Y_test)
knn_cv.score(X_test, Y_test)
```

8. Plot Confusion Matrix

```
yhat_lr=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat_lr)
yhat_svm=svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat_svm)
yhat_tree = svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat_tree)
yhat_knn = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat_knn)
```

9. Find method that performs best

```
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 0.8767857142857143
```

Results

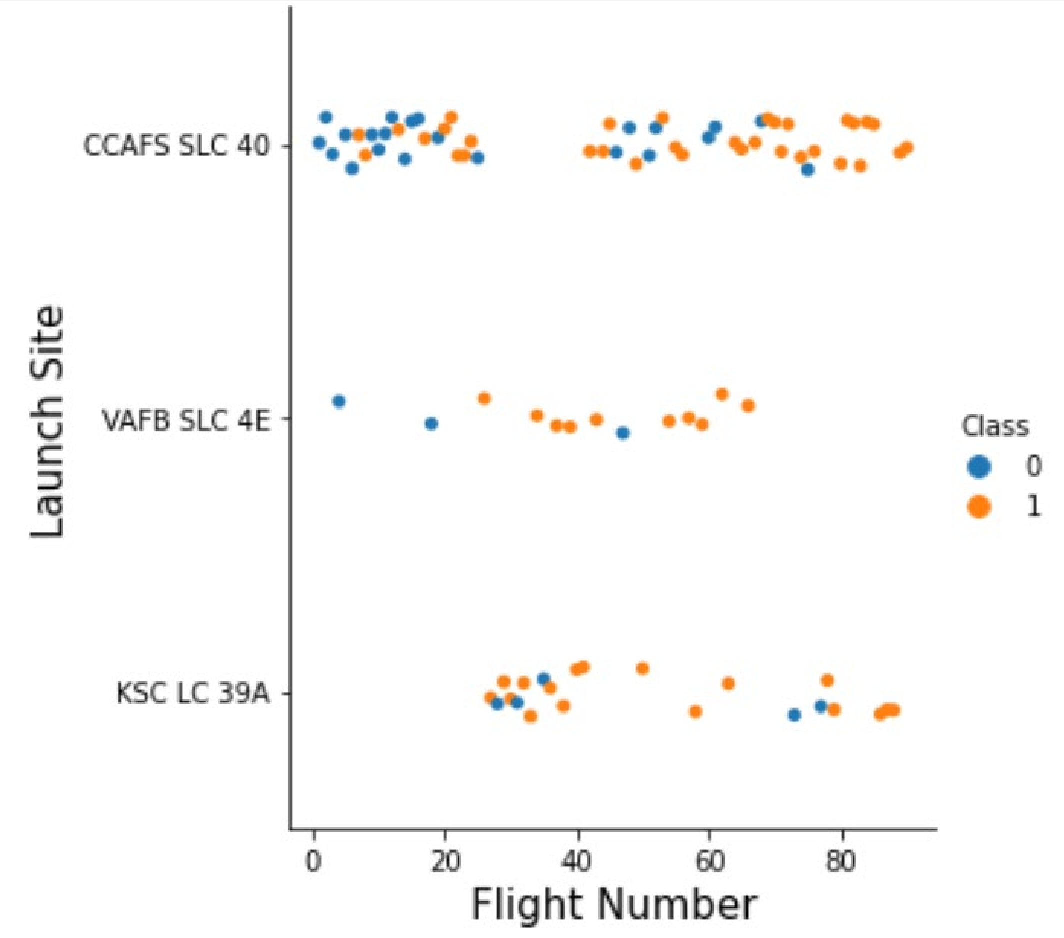
- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results

Section 2

Insights drawn from EDA

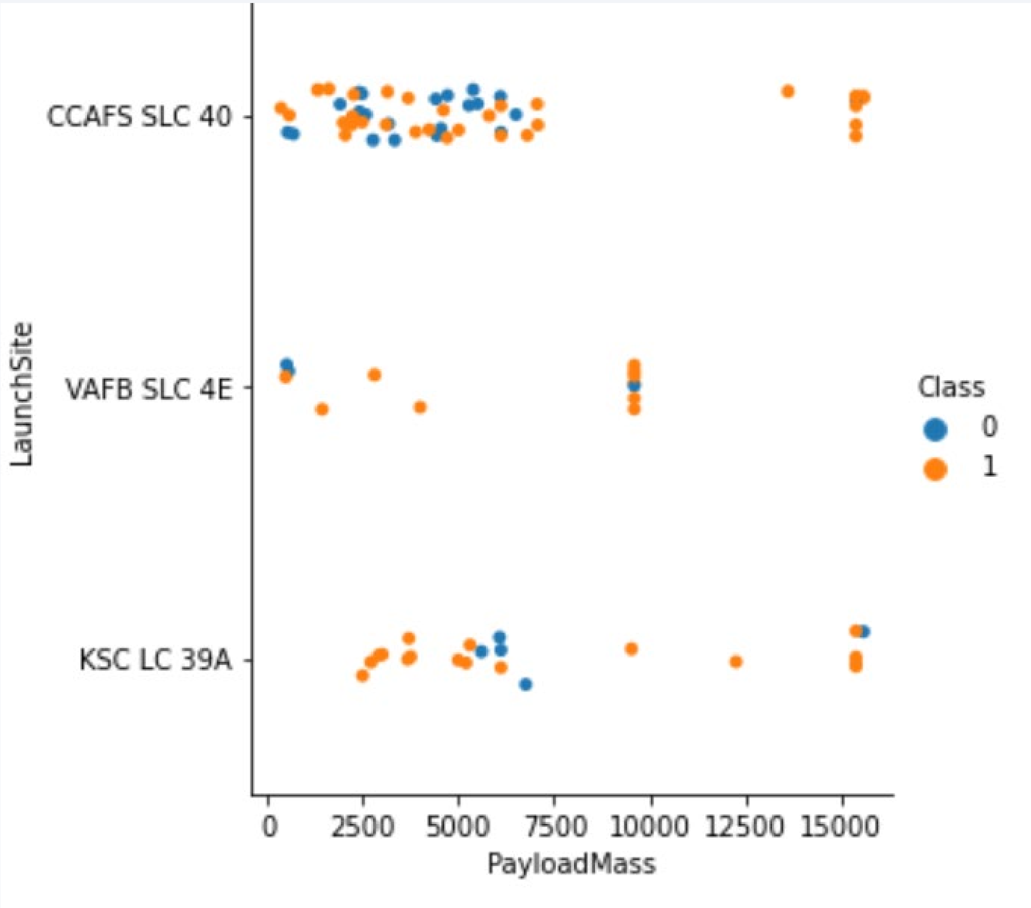
Flight Number vs. Launch Site

- Early fights were mainly at CCAFS SLC 40
- These flights were also in lower orbits, so this launch site may have been more equipped for those types of launches



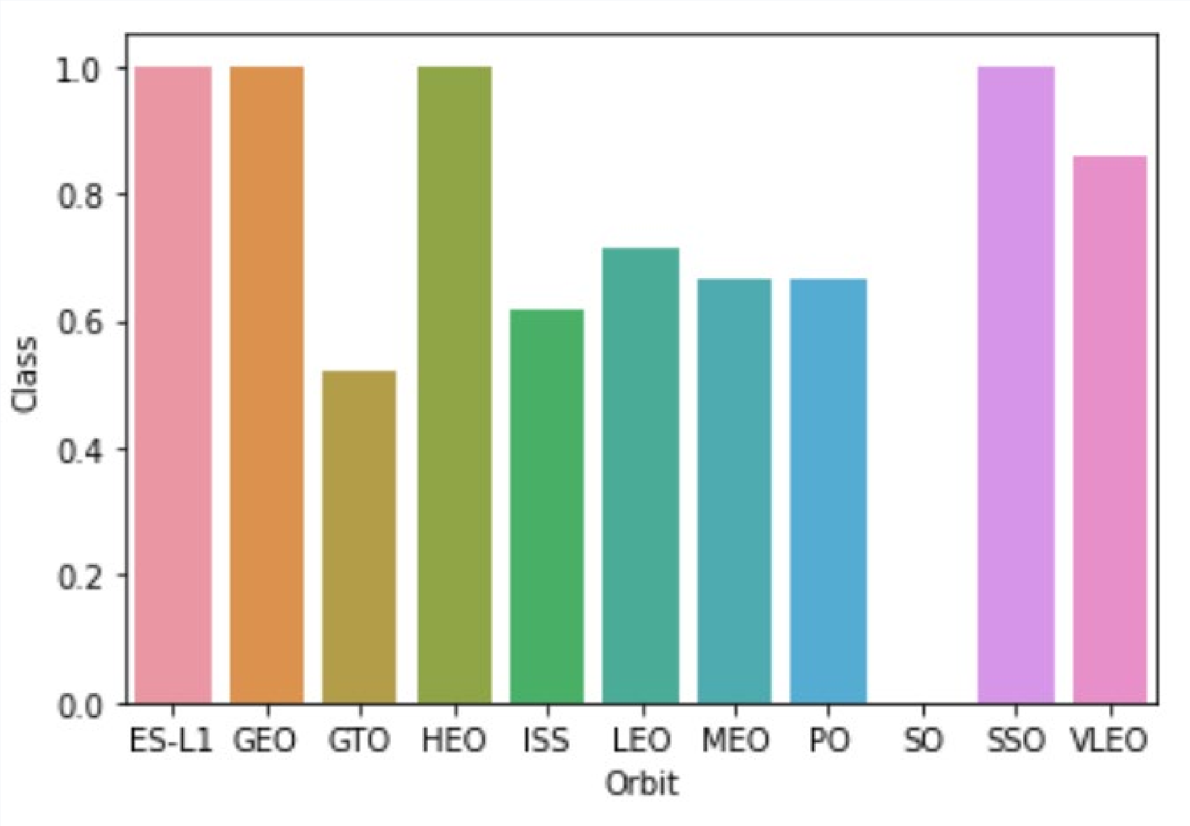
Payload vs. Launch Site

- CCAFS SLC 40 is more preferred for low payload masses
- VAFB SLC 4E is not used for any payload masses over 10,000 kg
- Most masses launched from KSC LC 39A were on the lower end, but those may just be from early test flights



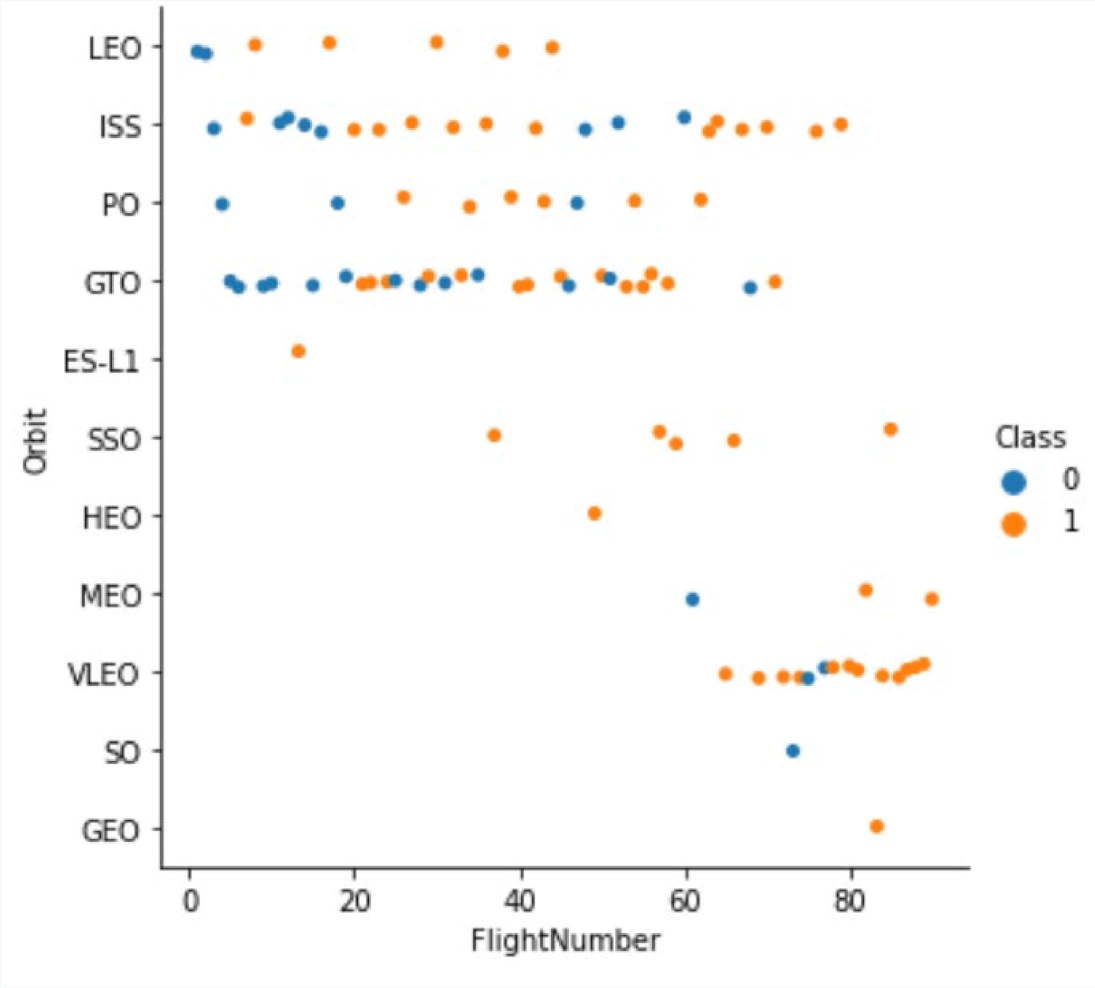
Success Rate vs. Orbit Type

- Some orbit types have higher success rates than other, but there is no correlation between the altitude of orbits and success rate



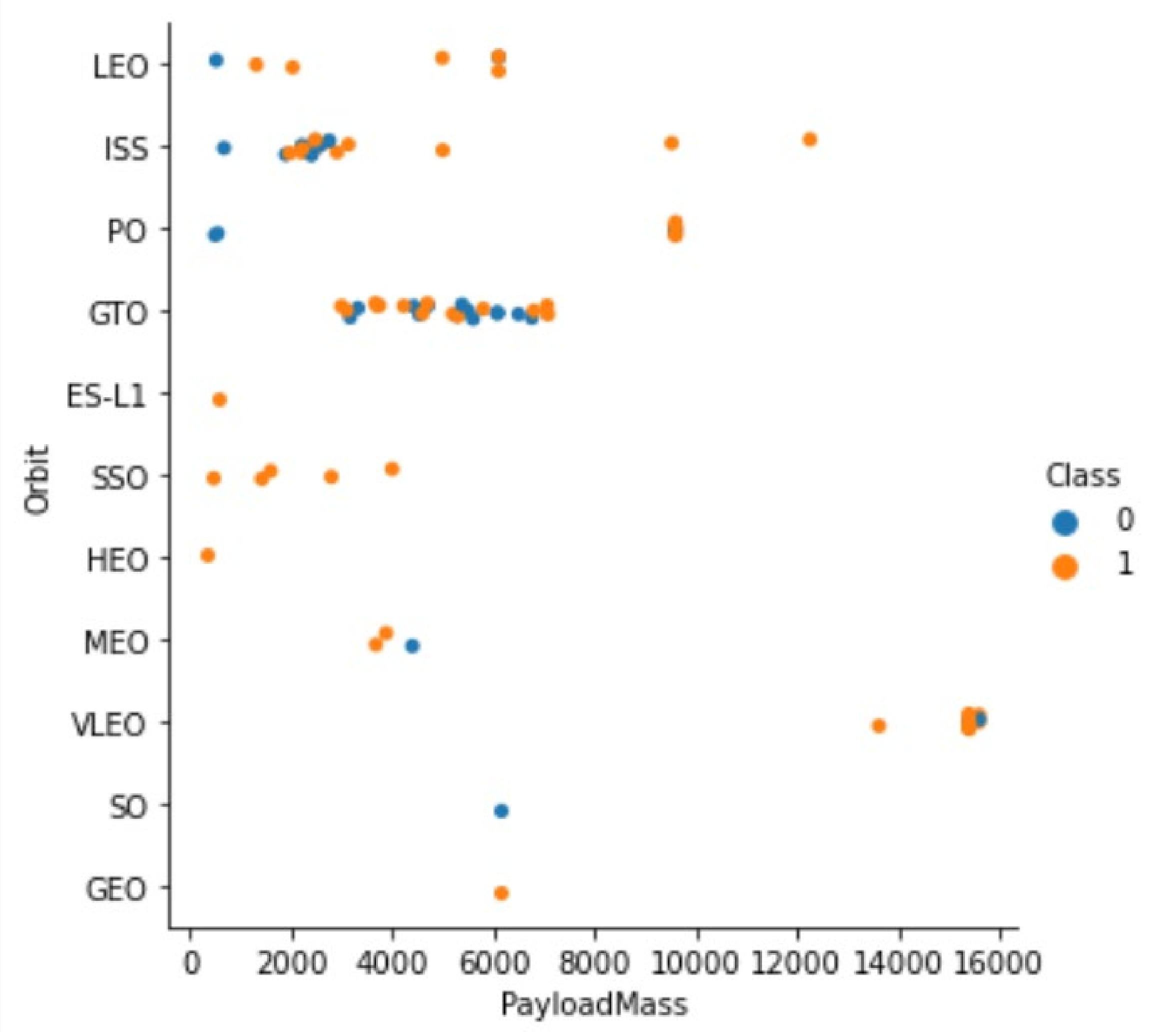
Flight Number vs. Orbit Type

- Success rate appears to correlated with flight number for LEO
- There are a higher number of failures for early GTO launches, but flight number still not appear to be correlated to success rate
- If an orbit type had their first launch after the first 30 flights, they had a greater success rate
- Nearly all of the first 20 flights failed



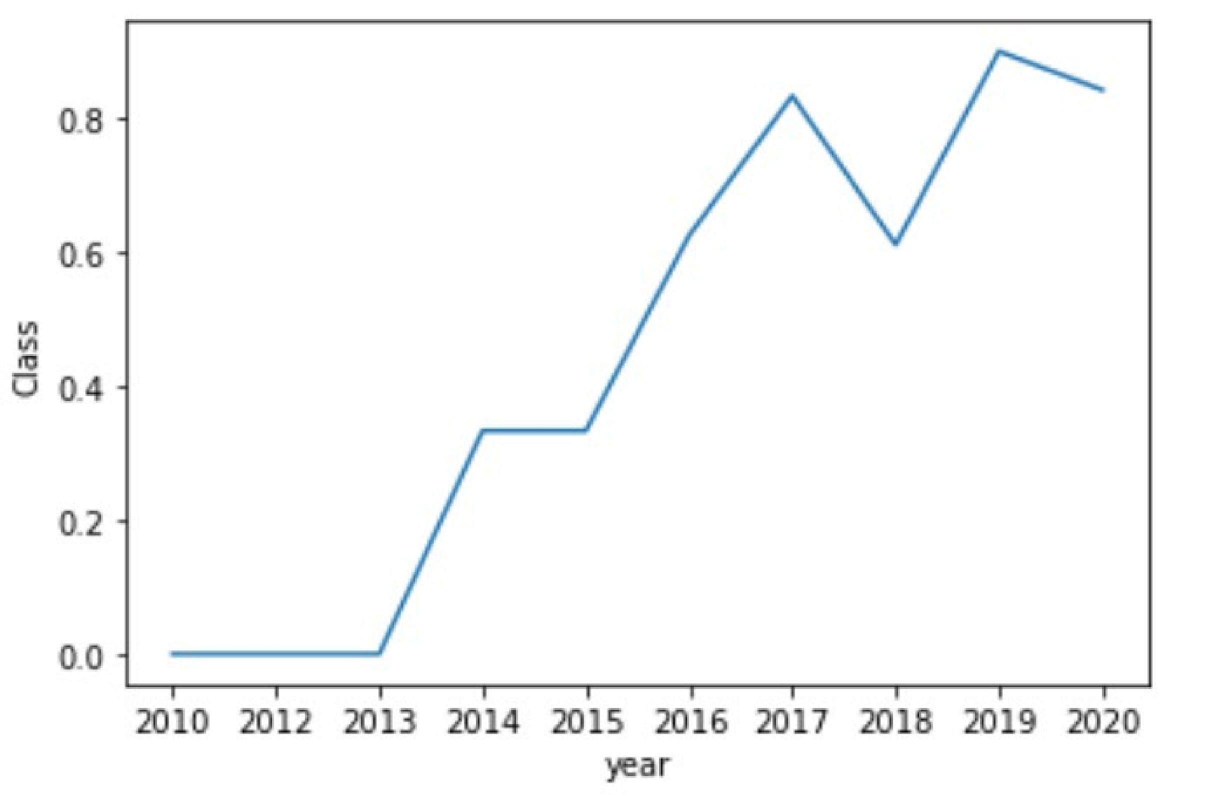
Payload vs. Orbit Type

- LEO, ISS, and Polar orbits are more successful with heavier payloads
- No correlation for GTO



Launch Success Yearly Trend

- All early flights failed
- Success rate has steadily climbed with experience
- Needs Further Exploration:
  - Plateau 2014-15
  - Large Dip 2017-18
  - Small Dip 2019-20



All Launch Site Names

- Distinct retrieves only unique records
- CCAFS LC-40 was renamed to CCAFS SCL-40. Name kept in records for launches pre-change.

```
%%sql
select distinct launch_site from spacextbl
```

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

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

Launch Site Names Begin with 'CCA'

```
%%sql
select * from spacextbl
    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
04-06-2010	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
08-12-2010	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)
22-05-2012	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
08-10-2012	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
01-03-2013	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

% is wildcard symbol. Matches any string of 0 or more characters.

In this instance, it will match anything that comes after “CCA”

Since the only records that begin with “CCA” are CCAFS LC-40 and CCAFS SLC-40, the full query would return all records for that site.

Total Payload Mass

- Returns sum of data in payload_mass__kg_ column for entries that have 'NASA (CRS)' in their customer column
- This is the total weight of payloads launched for NASA

```
%%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

- Returns average of payload_mass__kg_ column for rows who's booster_version entries begin with "F9 v1.1"
- There are additional characters after the booster version to denote flight number, so we use % to ignore these

```
%%sql
select avg(payload_mass__kg_) from spacextbl
    where booster_version like "F9 v1.1%"
```

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

avg(payload_mass__kg_)
2534.6666666666665

First Successful Ground Landing Date

```
%%sql
select min(date) from spacextbl
    where "Landing _Outcome" = "Success (ground pad)"
```

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

min(date)
01-05-2017

- Earlier dates are treated as lower values and recent dates are treated as higher values
- Applying min() to date will retrieve the earliest record, in this case for entries that have “Success (ground pad)” in the “Landing _Outcome” column

Successful Drone Ship Landing with Payload between 4000 and 6000

The FT booster version is the only booster to have a successful drone ship landing after being launched with a payload mass within this range

```
%%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

- Only one mission had a failed outcome
- Note that a launch can have a failed landing, but the mission still be deemed successful

```
%%sql
select mission_outcome, count(mission_outcome) from spacextbl
    group by mission_outcome
```

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

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

Boosters Carried Maximum Payload

- Where clause contains subquery to select records for payload mass that are equal to the maximum value in the payload mass column
- Only the B5 booster has carried the maximum payload mass

```
%%sql
select booster_version, PAYLOAD_MASS__KG_ from spacextbl
    where PAYLOAD_MASS__KG_ = (select max(PAYLOAD_MASS__KG_) from spacextbl)
```

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

Booster_Version	PAYLOAD_MASS__KG_
F9 B5 B1048.4	15600
F9 B5 B1049.4	15600
F9 B5 B1051.3	15600
F9 B5 B1056.4	15600
F9 B5 B1048.5	15600
F9 B5 B1051.4	15600
F9 B5 B1049.5	15600
F9 B5 B1060.2	15600
F9 B5 B1058.3	15600
F9 B5 B1051.6	15600
F9 B5 B1060.3	15600
F9 B5 B1049.7	15600

2015 Launch Records

```
%%sql
select substr(Date,4,2), "Landing _Outcome", booster_version, launch_site from spacextbl
where substr(Date,7,4)='2015' and "Landing _Outcome"="Failure (drone ship)"
```

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

substr(Date,4,2)	Landing _Outcome	Booster_Version	Launch_Site
01	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
04	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

- Failed drone ship landings from 2015, with month, booster version, and launch site
- Data in Date column are actually being stored as strings, since SQLite doesn't have built-in DateTime storage.
- substr(Date,4,2) returns a substring from the Date column, that starts at position 4 (index starts at 1, not 0), and has a length of 2 characters. Substr(Date,7,4) returns a 4 character substring, from the Date column, that starts at position 7.

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

- Landings were not attempted for most of these launches. They may decide to not attempt it if chances of recovery are too low
- When landing are attempted, most are successful
- The first entry in the data set is for 2010-06-04, so specifying for dates to be greater this is unnecessary

```
%%sql
select "Landing _Outcome", count("Landing _Outcome") from spacextbl
group by "Landing _Outcome"
having substr(Date,7,4)||'-'||substr(Date,4,2)||'-'||substr(Date,1,2) < '2017-03-20'
order by count("Landing _Outcome") Desc
```

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

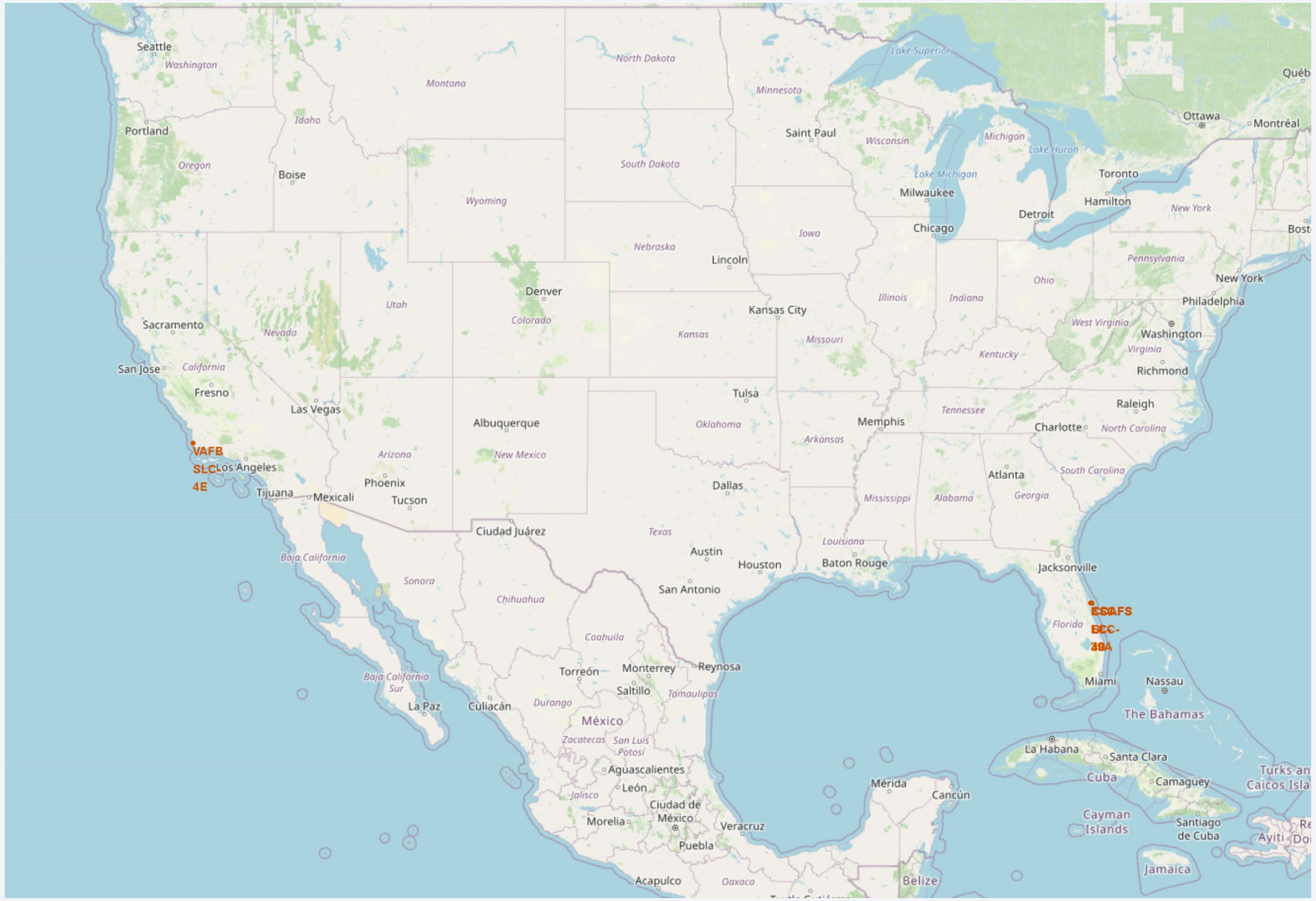
Landing _Outcome	count("Landing _Outcome")
No attempt	21
Success (drone ship)	14
Success (ground pad)	9
Failure (drone ship)	5
Controlled (ocean)	5
Uncontrolled (ocean)	2
Failure (parachute)	2
Precluded (drone ship)	1

Section 3

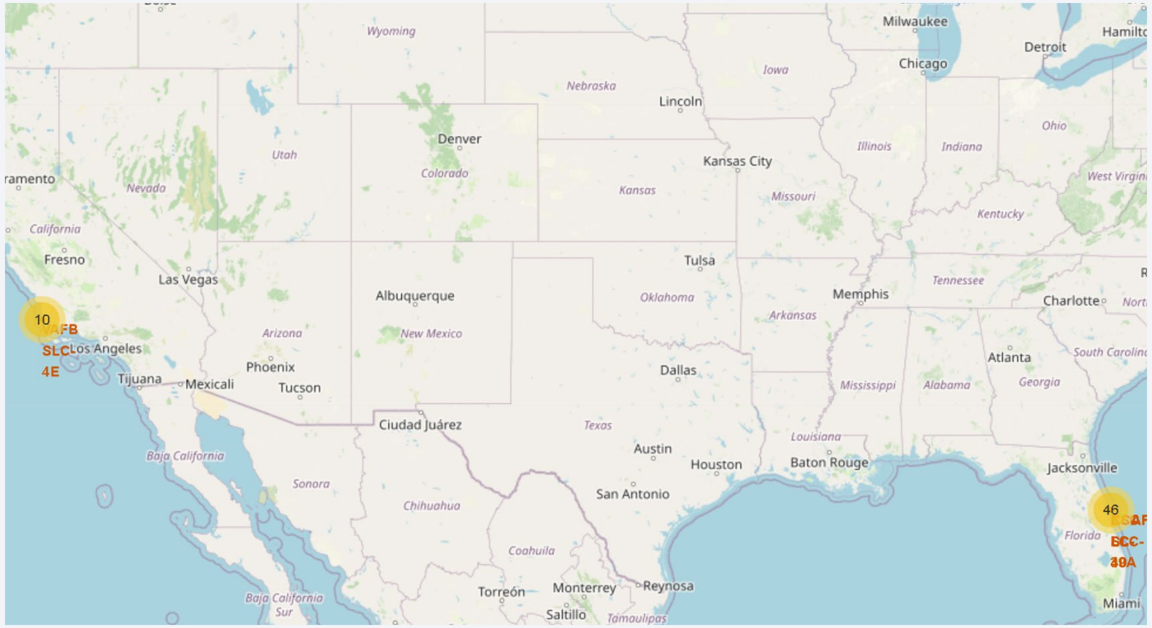
Launch Sites Proximities Analysis

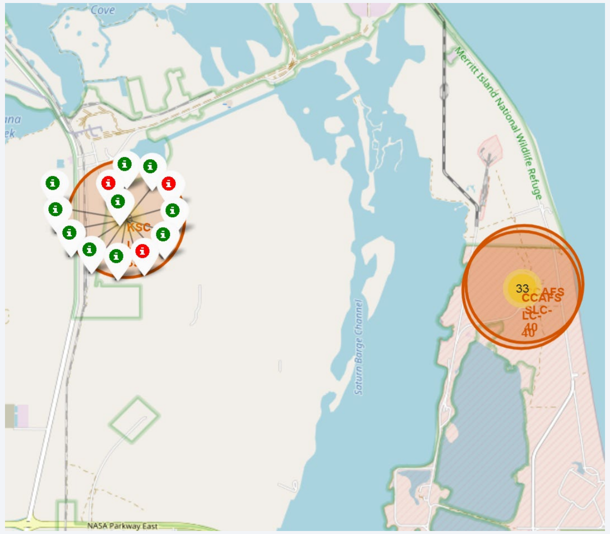
Folium Map – All Launch Sites

- Launch sites are located on coastlines, near oceans



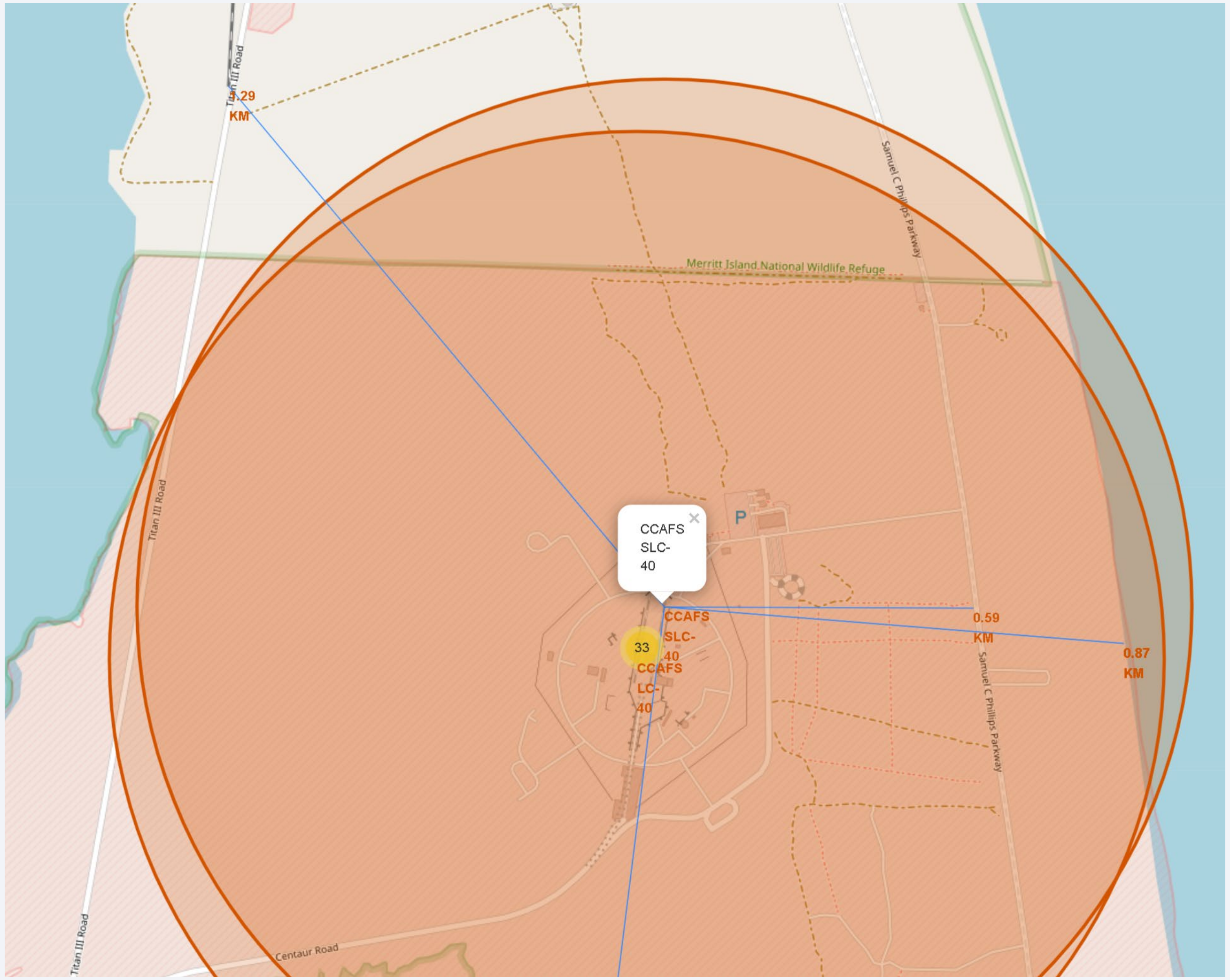
Folium Map – Launch Outcomes

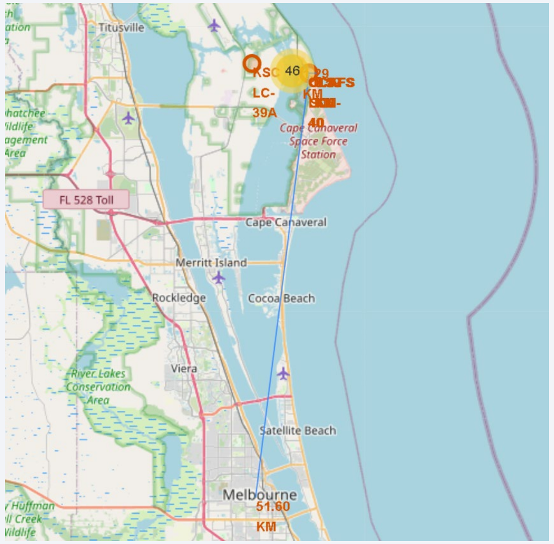




- Marker clusters show total number of launches from a particular area
- Clicking on a cluster will zoom in to show it’s constituents
- Clicking on an individual constituent will show color-coordinated breakdown of outcomes

Folium Map - Proximities



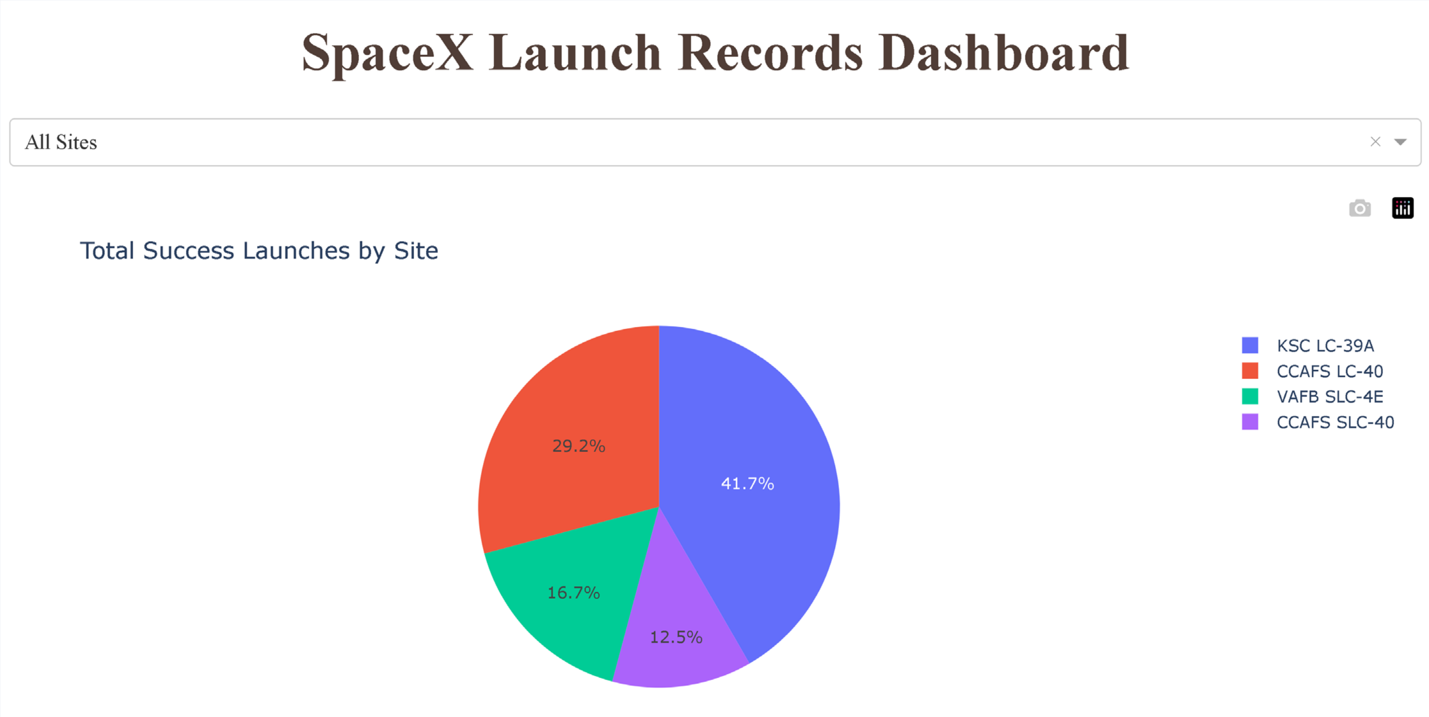


- As well as being located near oceans for launches, sites are also close distribution routes for better logistics
- They are also a fair distance away from nearby cities

Section 4

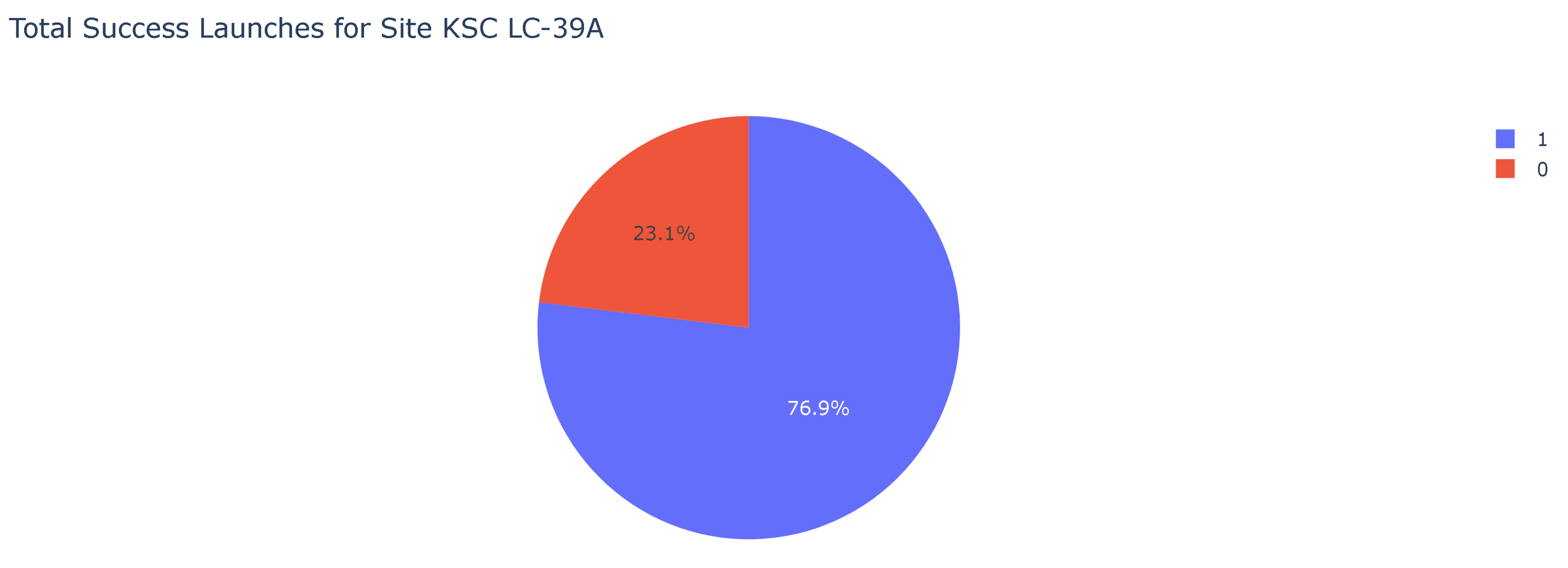
Build a Dashboard with Plotly Dash

Plotly Dash – Share of Successful Launches by Site



KSC LC-39A is responsible for most of SpaceX's successful launches

Plotly Dash – Ratio of Successful Launches



KSC LC-39A also had the highest ratio of successful launches for individual launch sites

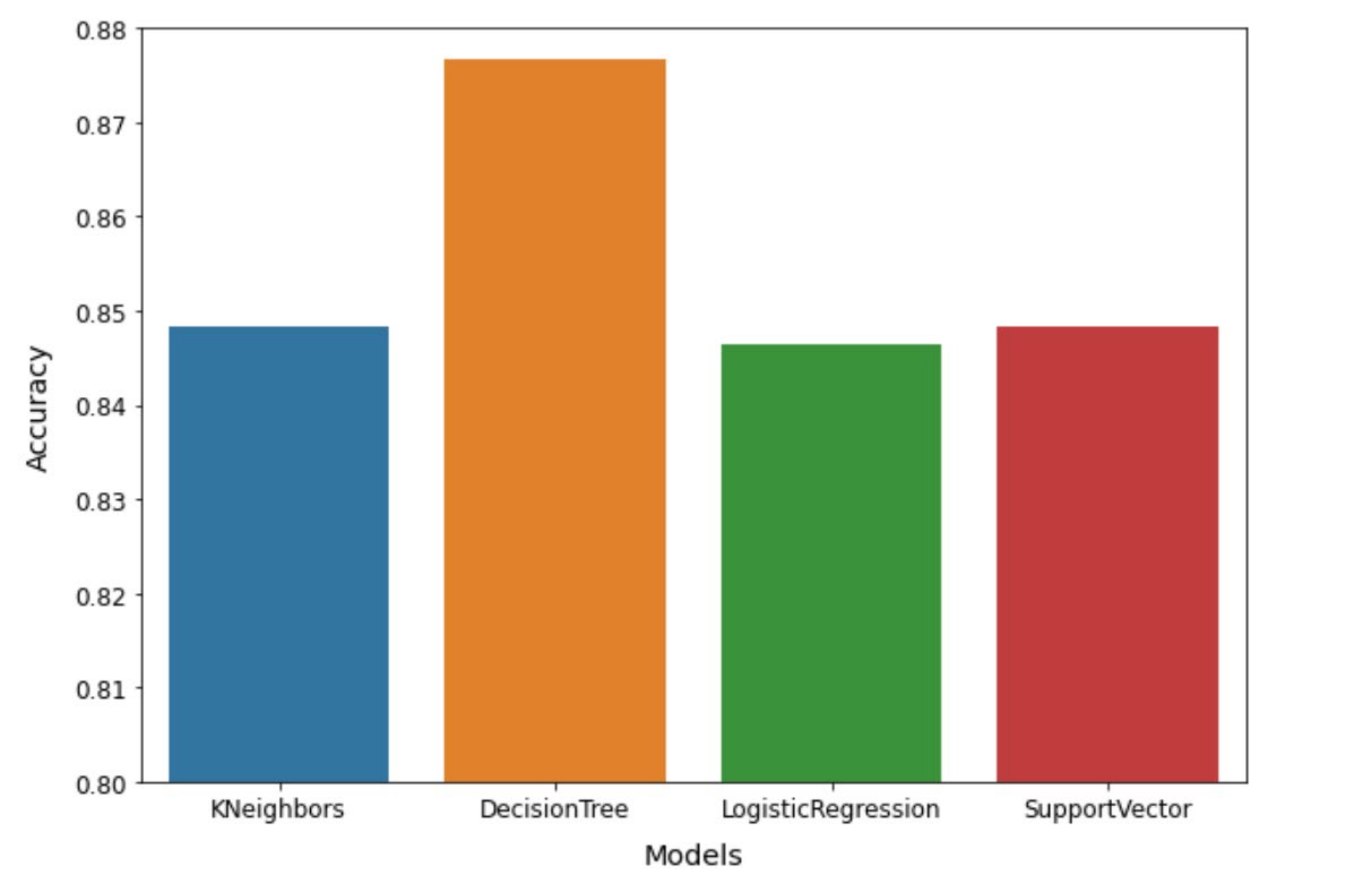
Plotly Dash – Payload Range and Booster Success

- Booster v1.1 was tested for a wide range of payloads and was unsuccessful nearly 100% of the time
- Booster FT looks to have the highest success rate out of boosters that have been launched multiple times

Section 5

Predictive Analysis (Classification)

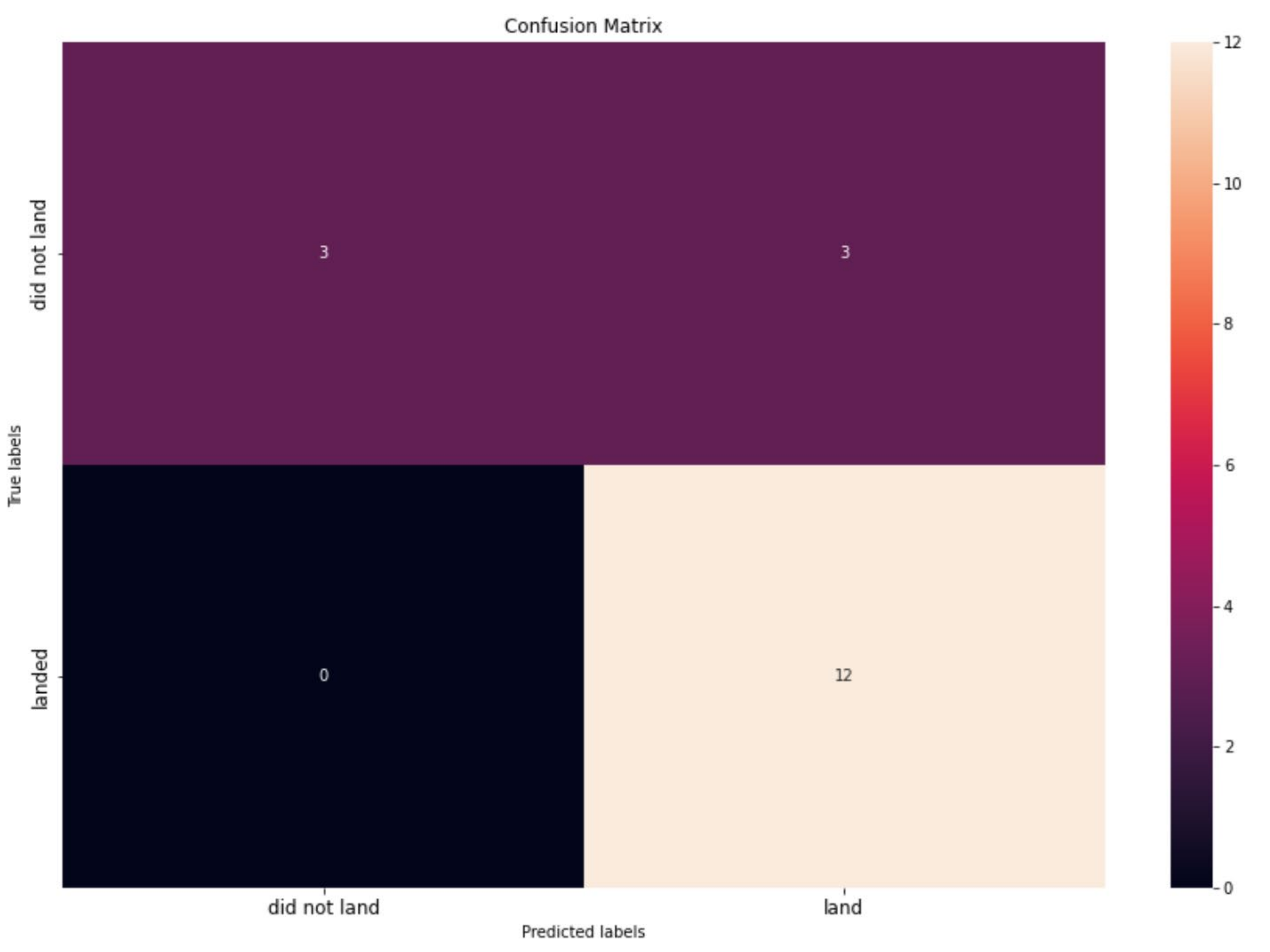
Classification Accuracy



The bar chart shows that the decision tree model has the highest accuracy

Confusion Matrix

- The decision tree model can accurately predict landing outcomes for the majority of the launch data
- There are no instances of false negatives
- There were instances of false positives, where it predicted successful landings that were unsuccessful in reality



Conclusions

- Booster FT had the highest success rate
  - Replicate booster FT launches for early launches
  - Keep early payload masses <6k kg
- Expect that the first ~20 landings may be unsuccessful
  - Account for this when calculating costs
- Launch sites must be located near coastlines
  - Must be in logistically appropriate locations
  - Far enough away from densely populated areas
- DecisionTree is best classification model

Thank you!