Winning Space Race with Data Science

Rizzi Cabrera
June 4, 2022

Outline

1. Executive Summary
2. Introduction
3. Methodology
4. Results
5. Conclusion
6. Appendix

Executive Summary

Methodologies	Results
• Data Collection (REST API and Web Scraping) • Data Wrangling • Exploratory Data Analysis (SQL, Data Visualization, Visual Analytics via Folium) • Machine Learning Prediction	• Exploratory Data Analysis result • Interactive analytics in screenshots • Predictive analytics results from machine learning lab

Introduction

Project Background

SpaceX advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upward of 165 million dollars 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.

Questions to Answer

1. How does payload affect the landing outcome?
2. What factors determine the success of the first stage's landing?

Section 1

Methodology

Methodology

Data collection methodology:

Data was collected from SpaceX API and via Web Scraping on Wikipedia

Perform data wrangling

Data was processed using one-hot encoding for categorical features

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, and evaluate classification models

Data Collection

SpaceX REST API

This API will give us the data about the launches
(e.g., rocket used, payload, specifications, landings, etc.)

Web Scraping via `BeautifulSoup`

For extracting the Falcon 9 launch records from Wikipedia

Data Collection – SpaceX API

1. `GET()` request for specific targets (JSON response) to get the past launch data about:
   - Rocket (booster name)
   - Payload – payload mass
   - Launchpad – launch sites
   - Cores - landing outcomes
2. Normalize JSON response using `json_normalize` to convert to a DataFrame
3. Create a new DataFrame from the get response

[GitHub link]

In [2]:
```
# Takes the dataset and uses the rocket column to call the API and append the data to the list
def getBoosterVersion(data):
    for x in data['rocket']:
        response = requests.get("https://api.spacexdata.com/v4/rockets/"+str(x)).json()
        BoosterVersion.append(response['name'])
```

In [6]:
```
spacex_url="https://api.spacexdata.com/v4/launches/past"
```

In [7]:
```
response = requests.get(spacex_url)
```

In [9]:
```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-Skill
```

In [11]:
```
# Use json_normalize meethod to convert the json result into a dataframe
data = pd.json_normalize(response.json())
```

In [13]:
```
# Lets take a subset of our dataframe keeping only the features we want and the flight number, and dat
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]

# We will remove rows with multiple cores because those are falcon rockets with 2 extra rocket booster
data = data[data['cores'].map(len)==1]
data = data[data['payloads'].map(len)==1]

# Since payloads and cores are lists of size 1 we will also extract the single value in the list and r
data['cores'] = data['cores'].map(lambda x : x[0])
data['payloads'] = data['payloads'].map(lambda x : x[0])

# We also want to convert the date_utc to a datetime datatype and then extracting the date leaving the
data['date'] = pd.to_datetime(data['date_utc']).dt.date

# Using the date we will restrict the dates of the launches
data = data[data['date'] <= datetime.date(2020, 11, 13)]
```

Data Collection – SpaceX API

4

Create a new DataFrame from the `GET()` response

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

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

5

Filtered to Falcon 9 launches to remove all the Falcon 1 data

```
In [24]:
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = df_launch[df_launch['BoosterVersion']!='Falcon 1']
data_falcon9.head()
```

[GitHub link]

Data Collection – SpaceX API

6

Check the summary of the dataset along with `isnull()` to see if there are missing values

7

Replace the missing values with the mean of `PayloadMass`

```
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
dtype: int64
```

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

[GitHub link]

Data Collection – Scraping

`GET()` request for the Wikipedia page

```
static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid
```

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

Create a `BeautifulSoup` object from the response

```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(data.text,'html.parser')
```

[GitHub link]

Data Collection – Scraping

Extract all the tables then, append all the column names and put them on an empty list

Fill in launch_dict dictionary with record values from the table rows. Create a new DataFrame from it

```
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:
```

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

[GitHub link]

Data Wrangling

Goal: Find patterns in the data and determine the label for supervised learning

Identify the data types and missing values

```
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
dtype: object
```

Data Types

```
df.isnull().sum()/df.count()*100
```

```
FlightNumber     0.000
Date             0.000
BoosterVersion   0.000
PayloadMass      0.000
Orbit            0.000
LaunchSite       0.000
Outcome          0.000
Flights          0.000
GridFins         0.000
Reused           0.000
Legs             0.000
LandingPad      40.625
Block            0.000
ReusedCount      0.000
Serial           0.000
Longitude        0.000
Latitude         0.000
dtype: float64
```

Missing Values

[GitHub link]

Data Wrangling

Goal: Find patterns in the data and determine the label for supervised learning

Calculate the numbers of:

- Launch per site
- Occurrence
- Mission outcome per orbit

```
# 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
```

Launch per Site

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

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

Occurrence

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

```
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
```

Mission outcome per orbit

[GitHub link]

Data Wrangling

Goal: Find patterns in the data and determine the label for supervised learning

Convert landing outcomes into training labels

Label	Outcome
1	Successful
0	Failure

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

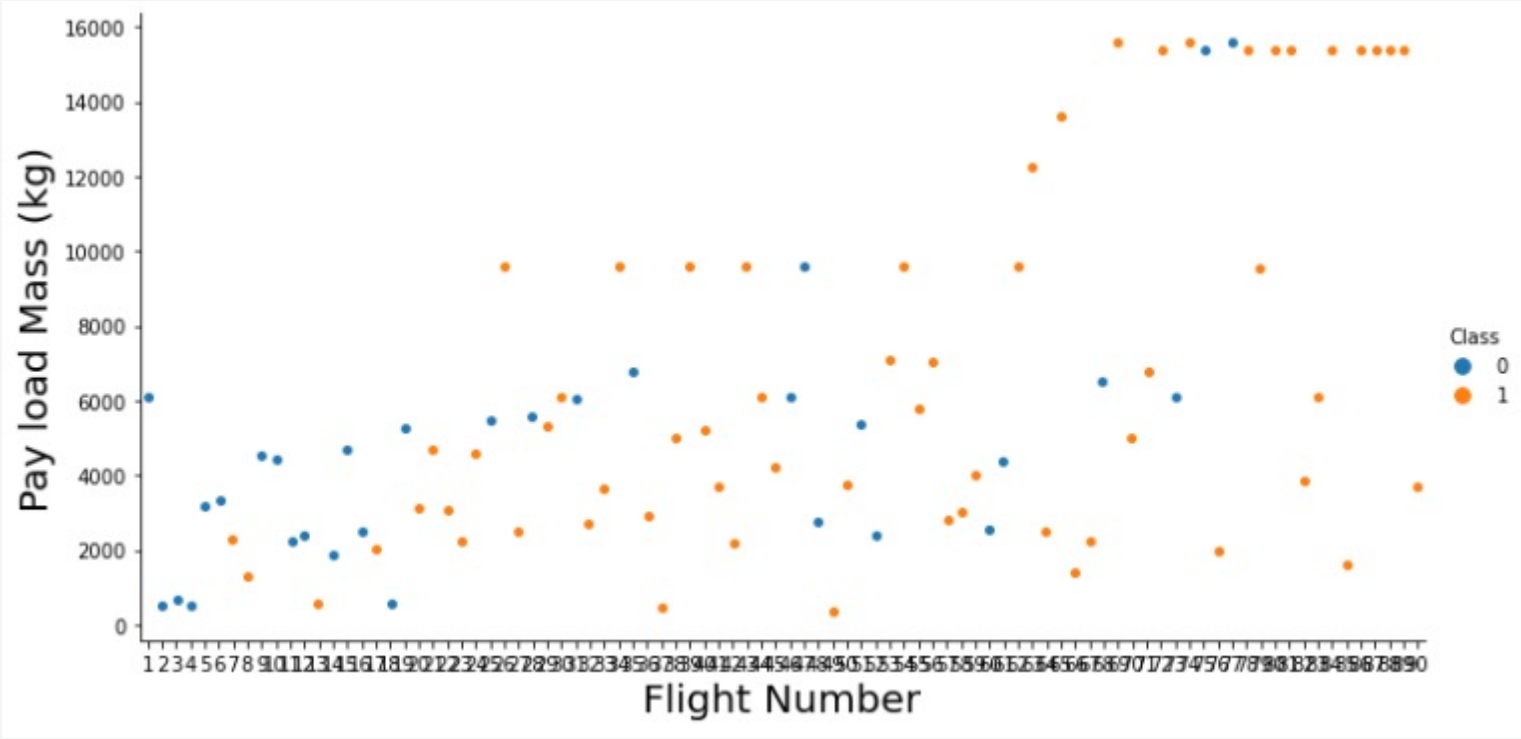
```
df['Class']=landing_class
df[['Class']].value_counts()
```

```
Class
1        60
0        30
dtype: int64
```

[GitHub link]

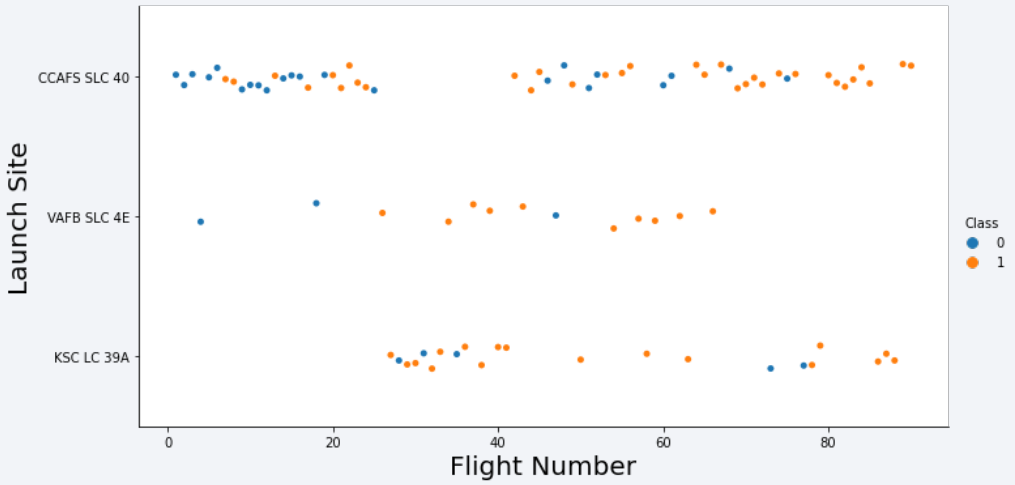
EDA with Data Visualization

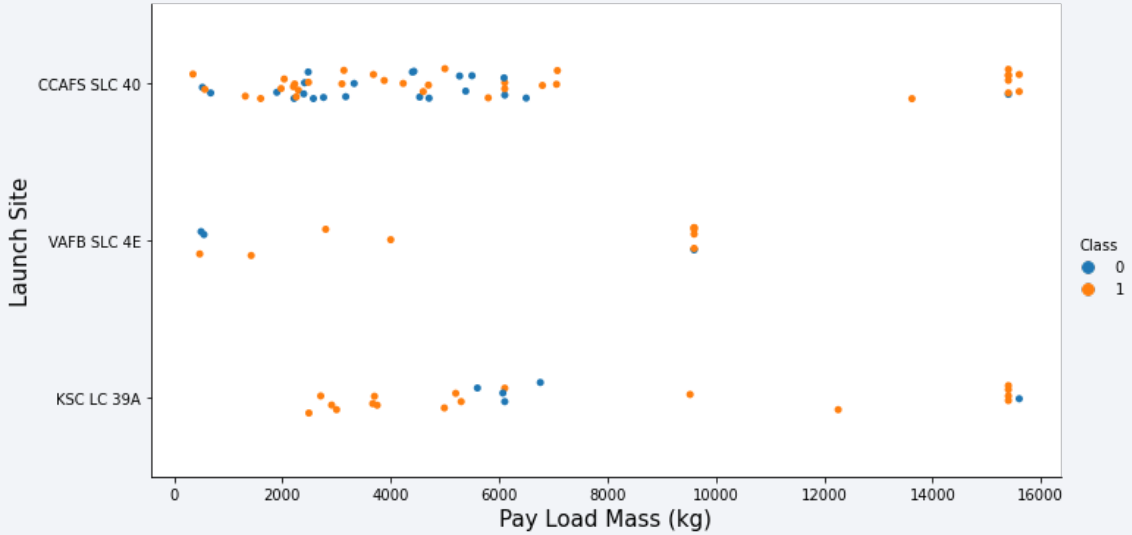
Overview of the success rate between the Payload Mass and the Flight (per launch site) to get a general hint about the success rates for all of the launch sites



[GitHub link]

EDA with Data Visualization

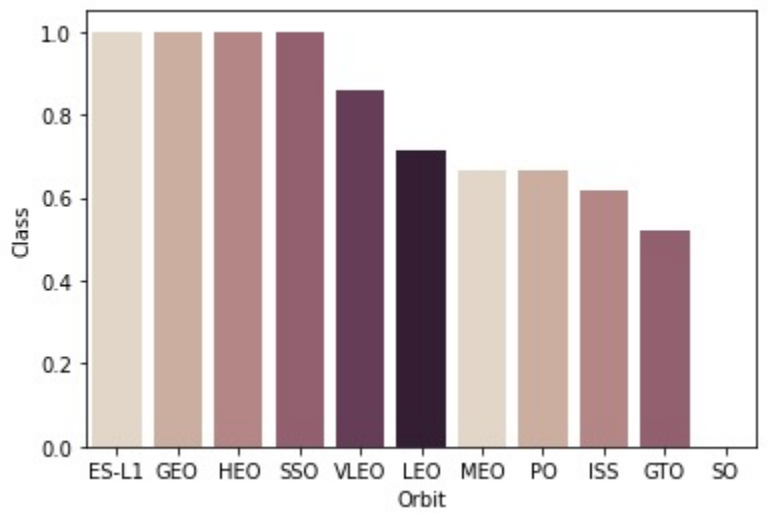




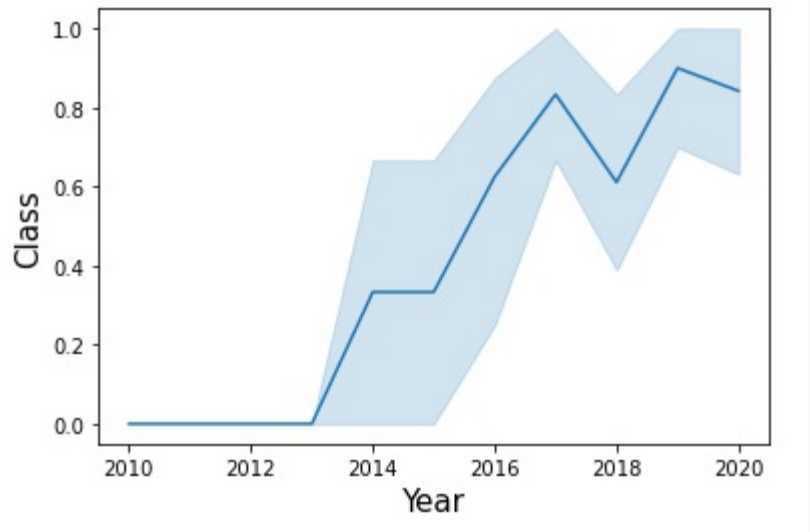
Another general view of the plotted scatterplots in order to get a general hint of how the Flight Number, Launch Site, and Payload Mass relates to each other

[GitHub link]

EDA with Data Visualization



A bar graph is plotted to easily see the success rate for each orbit type with a single glance



A line graph is plotted in order to see the yearly trend for the success rate of the all the launches

[GitHub link]

EDA with SQL

- Distinct names of launch sites
- Records of launch sites with names that start with ‘CCA’
- Total Payload Mass carried by booster from NASA (CRS)
- Average Payload Mass by booster version F9 v1.1
- Date of the first successful landing on a ground pad
- Boosters with a Payload weighing between 4,000 and 6,000 that succeeded on landing on a drone ship

[GitHub link]

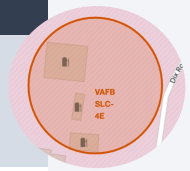
EDA with SQL

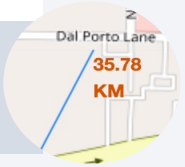
- Total number of mission outcomes (Success and failures)
- Booster versions that has carried the maximum payload
- Booster version and launch site that failed a drone ship landing from 2015
- Descending order of all the landing outcomes between June 6, 2010 and March 20, 2017

[GitHub link]

Build an Interactive Map with Folium

Object	Description
`Folium.Circle`	Circles and highlights the coordinates of launch sites
`Folium.Marker`	For marking and labeling a location on the map (launch sites, location, etc.)
`MarkerCluster()`	a Folium plugin that lets us cluster a location that contains a lot of markers
`MousePosition`	A folium plugin that lets us get the coordinates for a mouse over a point on the map
`Folium.PolyLine`	Lets us draw a line from one coordinate to another

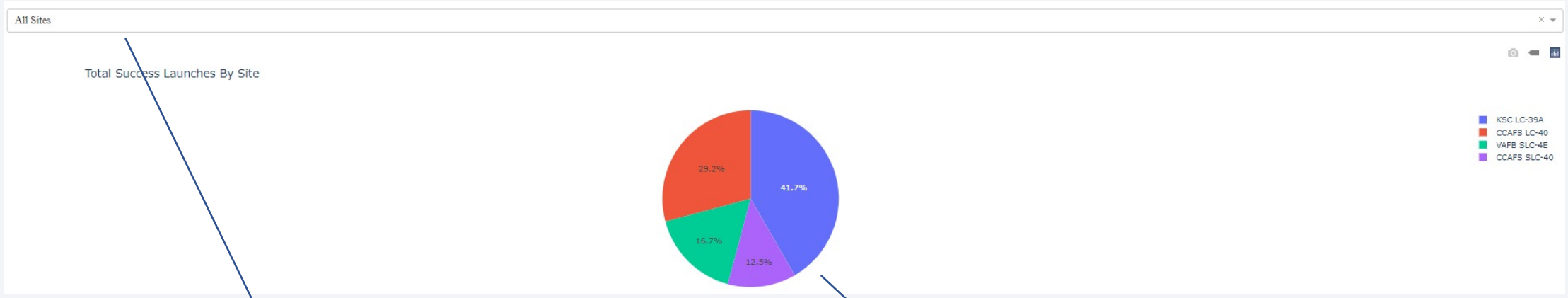




[via NB viewer link]

[GitHub link]

Build a Dashboard with Plotly Dash

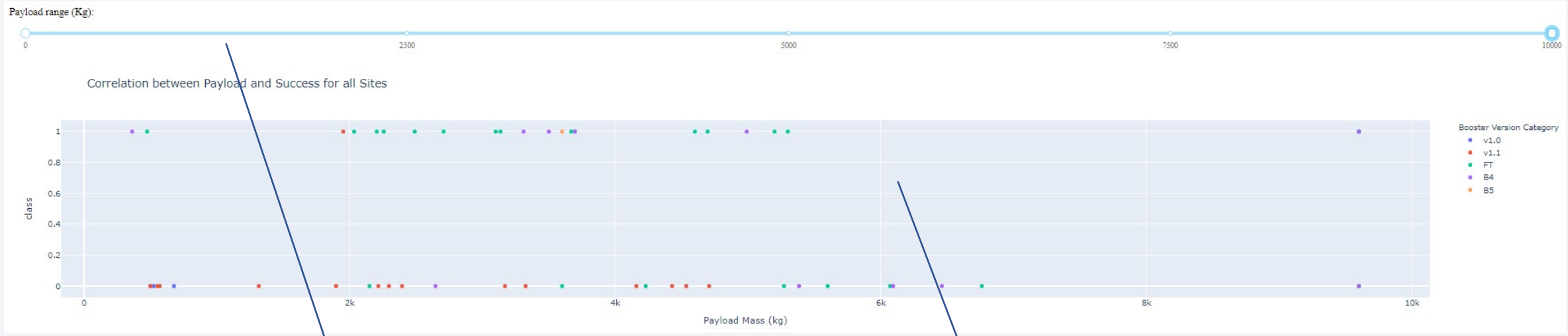


Dropdown
For sorting the different outcomes for the pie chart

Pie Chart
A chart for the launch results of each launch sites

[GitHub link]

Build a Dashboard with Plotly Dash



Range Slider
For selecting different ranges of payload in correlation with the mission outcome

Scatter Plot
A plot that shows the correlation between the Payload and the Success of a mission (categorized by booster version)

[GitHub link]

Predictive Analysis (Classification)

Model Building

1. Data transformation (Numpy and Pandas)
2. Data fitting
3. Creating train and test sets

Model Evaluation

1. Use of Grid Search to find the optimal hyperparameters of a model which results with the most accurate predictions

`best_score_` data attribute - For testing the data accuracy on the validation data

`score()` method - for determining the accuracy of the model using the test data

Model Improvement

Feature Engineering and Algorithm Tuning

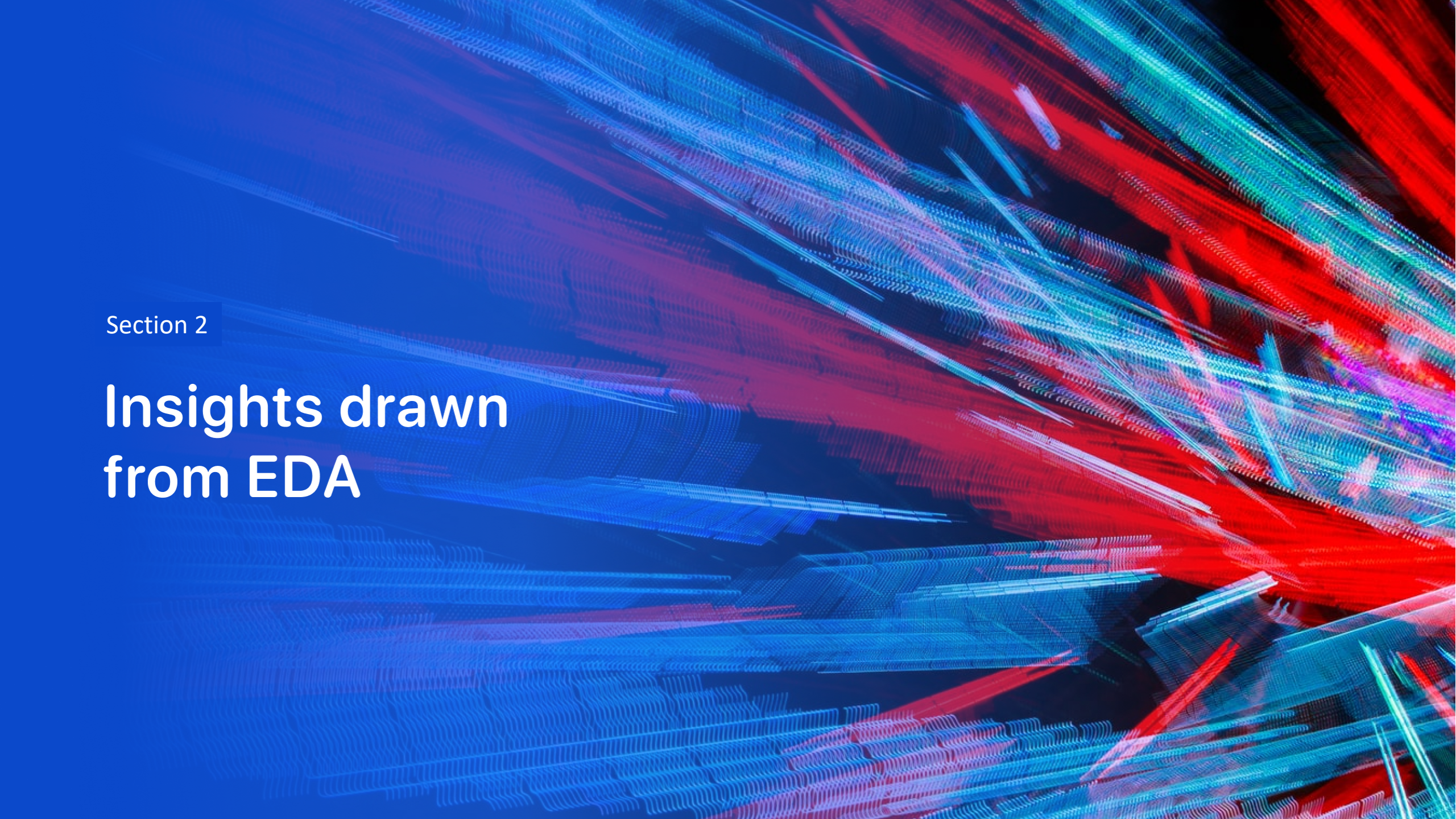
Determining the best model

The model with the best score when used on a test data

[GitHub link]

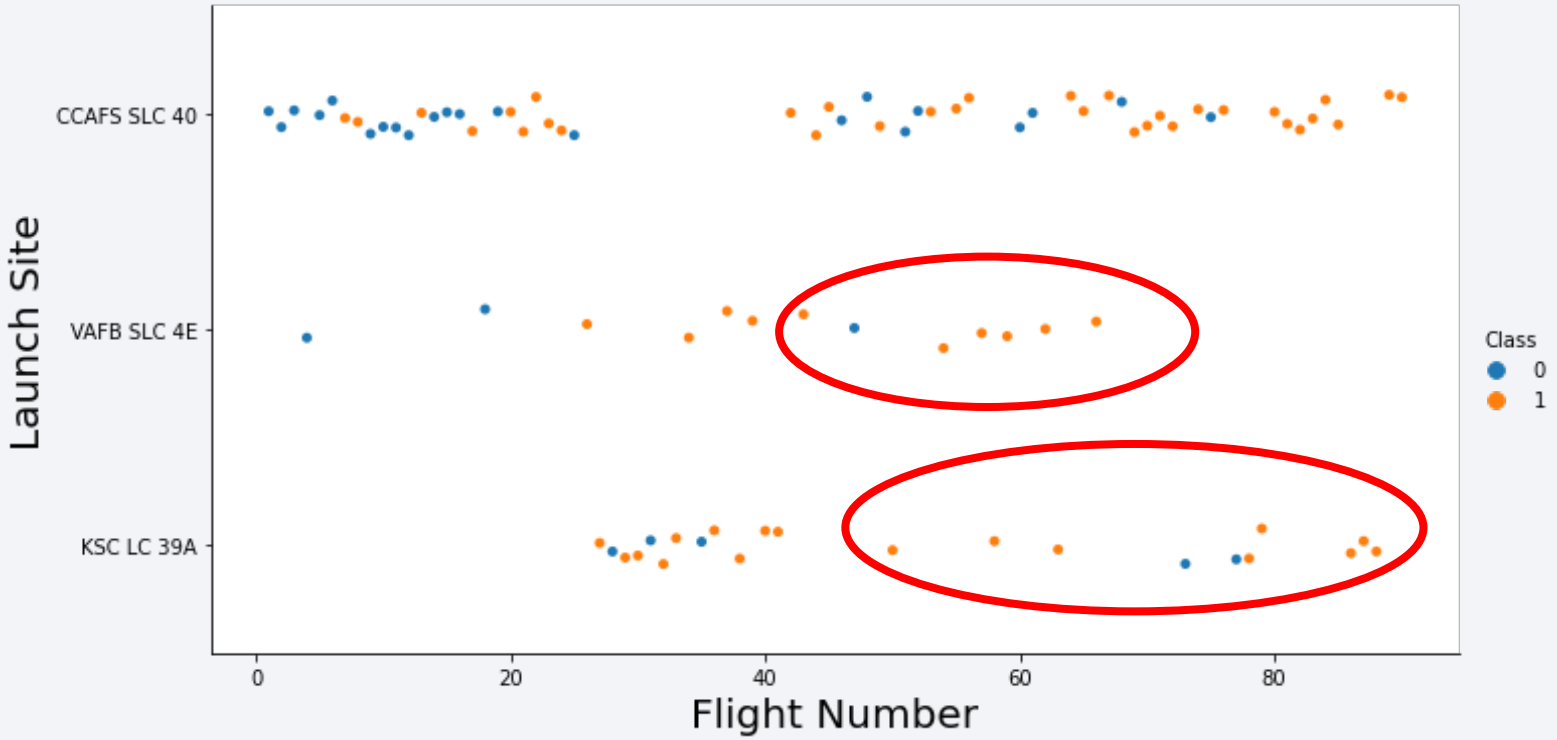
Results

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


Section 2
Insights drawn from EDA

Flight Number vs. Launch Site

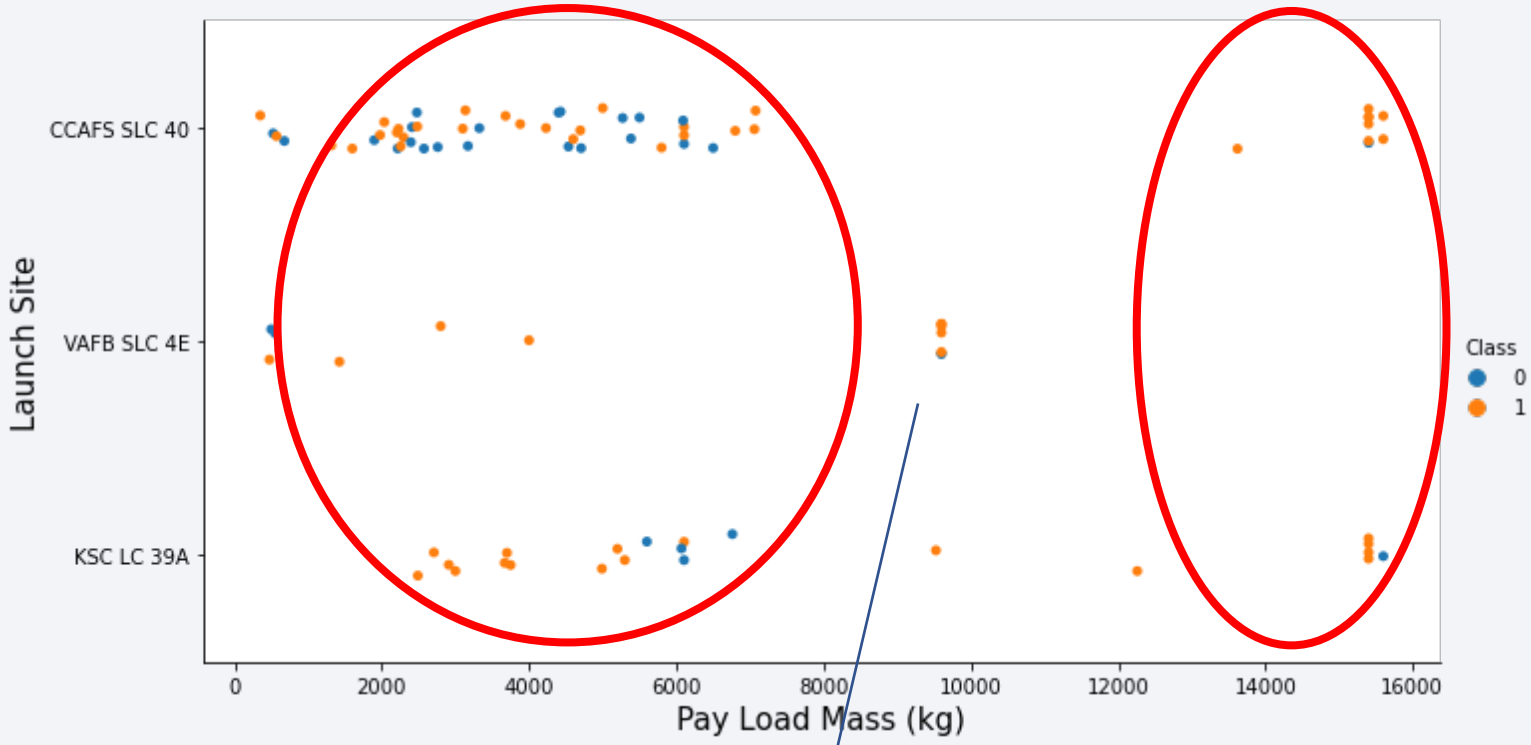
Launch Sites **VAFB SLC 4E and KSC LC 39A** shows promising **positive mission outcomes** the more flights are made.



Meanwhile, it's hard to distinguish the correlation for CCAFS SLC 40 as their mission outcome results are in the same range.

Payload vs. Launch Site

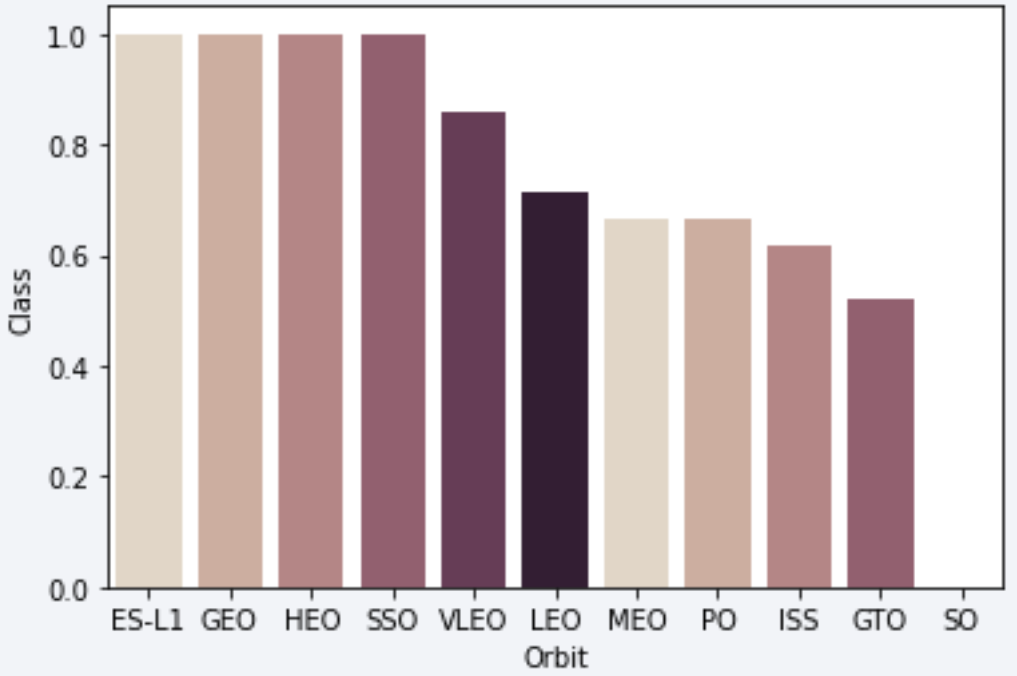
There seems to be a slight trend of successful mission outcomes for payloads heavier than 10,000kg. Payloads lesser than 8,000kg seem to have mixed results with their mission outcomes.



In addition, VAFB-SLC has not launched any rockets with payloads greater than 10,000kg.

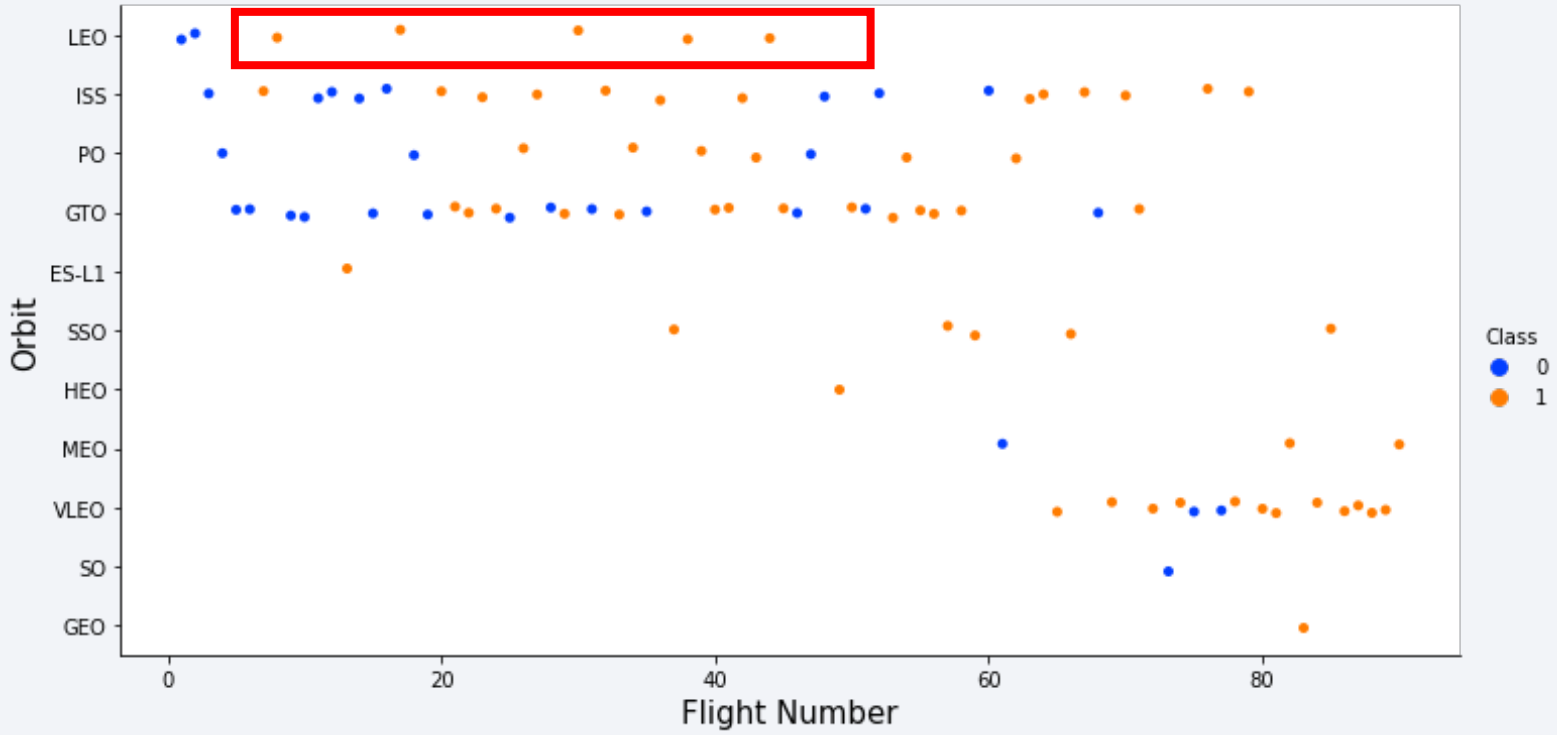
Success Rate vs. Orbit Type

Missions that orbited **ES-L1, GEO, HEO, and SSO** all had a **100% success**.
However, the **data is not enough** as some of the orbits like SSO (3 missions) and HEO (2 missions) do not have enough missions on their orbits.



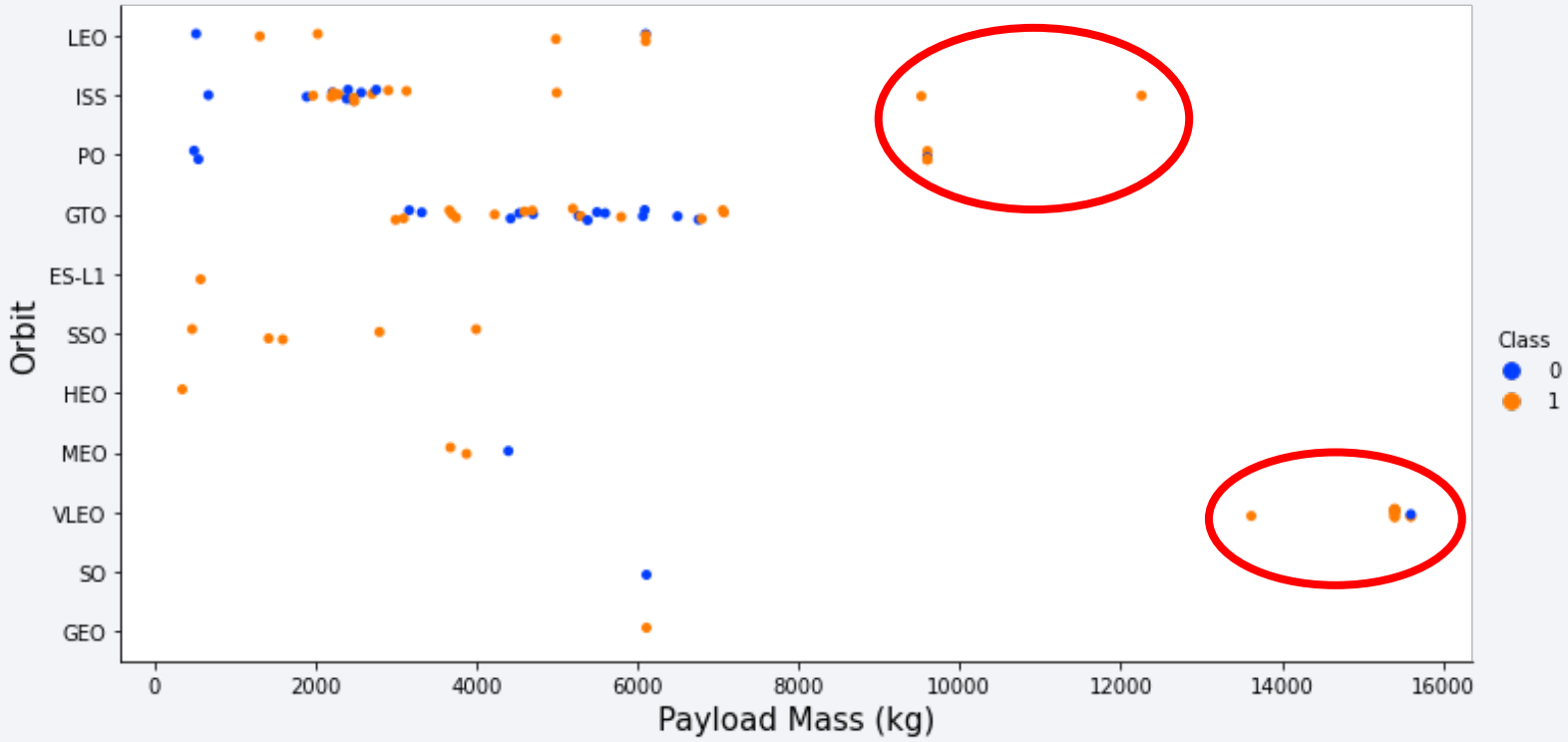
Flight Number vs. Orbit Type

The success of the **LEO orbit** appears to be **related to the number of flights**. Meanwhile, there seems to be **no relationship** between the number of flights and the **GTO orbit**.

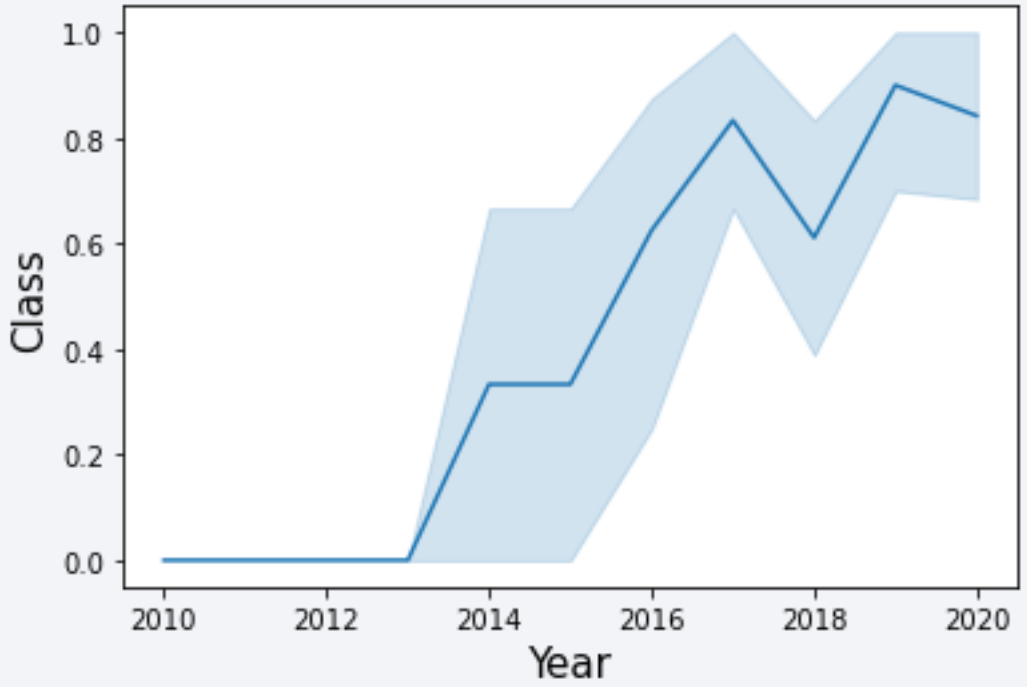


Payload vs. Orbit Type

The orbits of **Polar, LEO, and ISS** appear to have a **higher rate of successful landings with heavy payloads**. Meanwhile, the landing rate for the GTO orbit cannot be distinguished as both of the results are in the same range.



Launch Success Yearly Trend



Since 2013, there has been a positive and **steady increase in successful landings**.

All Launch Site Names

SELECT DISTINCT statement was used to fetch the distinct Launch Site names from the table

```
%sql SELECT DISTINCT launchsite FROM spacex
```

```
 * postgresql://postgres:***@localhost/SpaceX
4 rows affected.
```

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

Launch Site Names Begin with 'CCA'

The `LIKE` operator was used along with the `WHERE` clause in order to search for Launch Sites that specifically starts with ‘CCA’. The `LIMIT` clause constrains the number of returned rows.

```
%%sql
SELECT launchsite
FROM spacex
WHERE launchsite LIKE 'CCA%' LIMIT 5
```

```
 * postgresql://postgres:***@localhost/SpaceX
5 rows affected.
```

launchsite
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40

Total Payload Mass

WHERE clause was used in order to filter the customer to only return the data for NASA (CRS). The **SUM()** function was used to calculate NASA (CRS)'s total payload as **45,596**

```
%%sql
SELECT SUM(payload_mass_kg) as TOTAL_SUM
FROM spacex
WHERE customer='NASA (CRS)'
```

```
 * postgresql://postgres:***@localhost/SpaceX
1 rows affected.
```

total_sum
45596

Average Payload Mass by F9 v1.1

Query was filtered to only select the F9 v1.1 booster version. The **AVG()** function was then used find the average Payload Mass of the selected version which is **2,534.67kg**

```
%%sql
SELECT AVG(payload_mass_kg) as F9_v1_1_AVG
FROM spacex
WHERE booster_version LIKE 'F9 v1.1%'
```

```
 * postgresql://postgres:***@localhost/SpaceX
1 rows affected.
```

f9_v1_1_avg
2534.6666666666666667

First Successful Ground Landing Date

Query was filtered to only return the rows for successful landings on the ground pad. The `MIN()` function was used to get the earliest date.

The very first successful landing on a ground pad was on **December 22, 2015**

```
%%sql
SELECT MIN(l_date)
FROM spacex
WHERE landing_outcome='Success (ground pad)'
```

```
 * postgresql://postgres:***@localhost/SpaceX
1 rows affected.
```

min
2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

Query was filtered to only return the rows from the successful landings on a drone ship. **BETWEEN** and **AND** were used to specifically target the returned rows with a payload mass between 4,000kg and 6,000kg

```
%%sql
SELECT booster_version, payload_mass_kg, landing_outcome
FROM spacex
WHERE landing_outcome='Success (drone ship)' AND
payload_mass_kg BETWEEN 4000 AND 6000
```

```
 * postgresql://postgres:***@localhost/SpaceX
4 rows affected.
```

booster_version	payload_mass_kg	landing_outcome
F9 FT B1022	4696	Success (drone ship)
F9 FT B1026	4600	Success (drone ship)
F9 FT B1021.2	5300	Success (drone ship)
F9 FT B1031.2	5200	Success (drone ship)

Total Number of Successful and Failure Mission Outcomes

```
%%sql
SELECT SUM(CASE WHEN mission_outcome LIKE 'Success%' THEN 1 END) AS success,
SUM(CASE WHEN mission_outcome LIKE 'Fail%' THEN 1 END) AS failure
FROM spacex;
```

```
 * postgresql://postgres:***@localhost/SpaceX
1 rows affected.
```

success	failure
100	1

Mission outcomes were tallied by summing the total returned integers from two **CASE** statements which used the wildcards 'Success%' and 'Fail%' as its conditions.

Boosters Carried Maximum Payload

```
%%sql
SELECT booster_version
FROM spacex
WHERE payload_mass_kg = (SELECT MAX(payload_mass_kg) FROM spacex)
```

```
 * postgresql://postgres:***@localhost/SpaceX
12 rows affected.
```

The query was filtered according to their Payload Mass, then it was furthered filtered by only selecting the boosters with the maximum payload via the `MAX()` function

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

The query was filtered by their landing date, then furthered filtered by specifically using **BETWEEN** and **AND** operators to fetch the failed drone ship landing made from throughout the year 2015.

```
%%sql
SELECT booster_version, launchsite, landing_outcome
FROM spacex
WHERE l_date BETWEEN '2015-01-01' AND '2015-12-31'
AND landing_outcome ='Failure (drone ship)'
```

```
 * postgresql://postgres:***@localhost/SpaceX
2 rows affected.
```

booster_version	launchsite	landing_outcome
F9 v1.1 B1012	CCAFS LC-40	Failure (drone ship)
F9 v1.1 B1015	CCAFS LC-40	Failure (drone ship)

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

```
%%sql
SELECT landing_outcome, COUNT(landing_outcome)
FROM spacex
WHERE l_date BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY landing_outcome
ORDER BY COUNT(landing_outcome) DESC
```

```
 * postgresql://postgres:***@localhost/SpaceX
 8 rows affected.
8 rows affected.
```

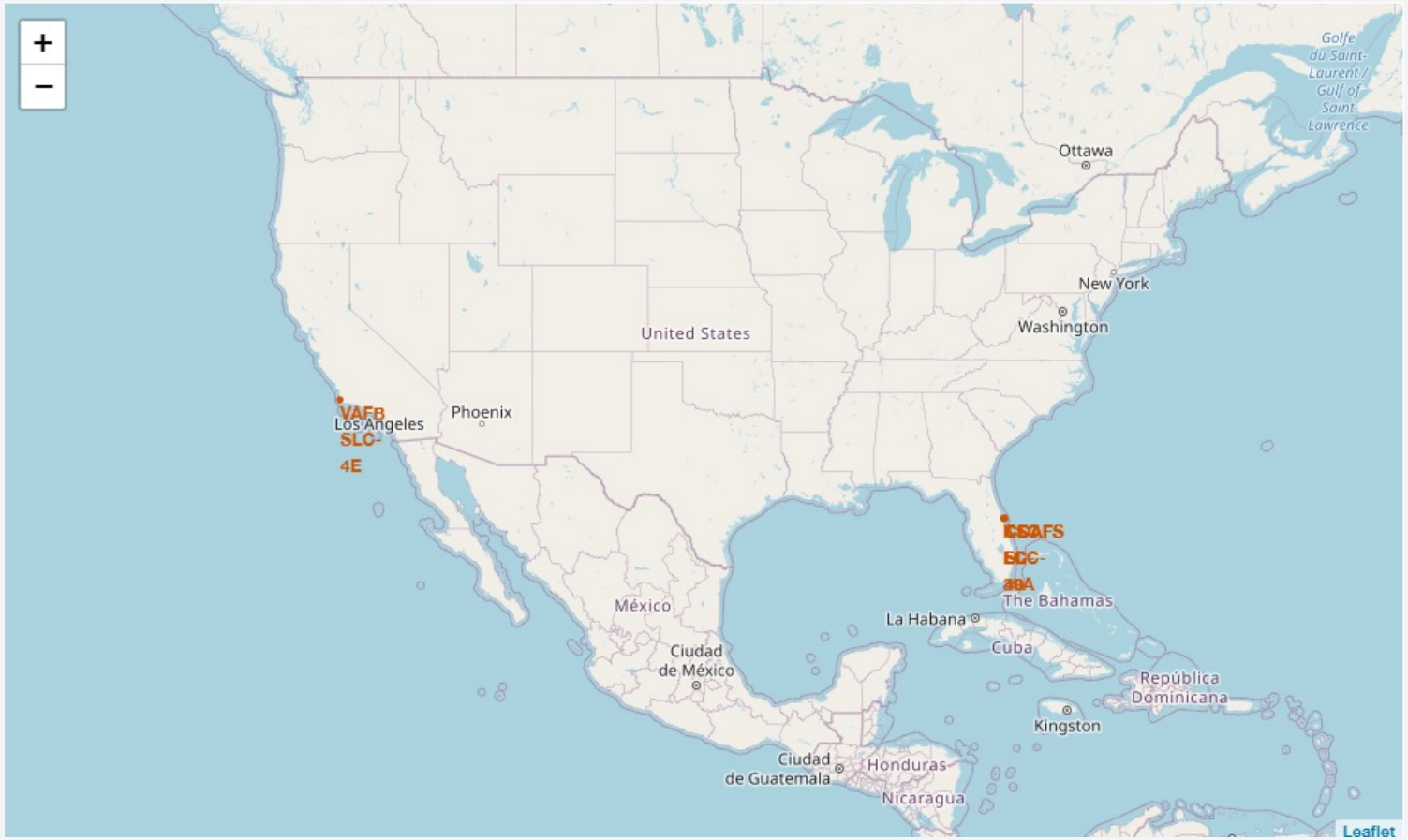
The query was filtered by their launch date, then furthered filtered to fetch the results between the launch dates June 04, 2010 and March 20, 2017. The fetch results were counted and grouped by their outcomes, then ranked in descending order.

8 rows affected.

landing_outcome	count
No attempt	10
Success (drone ship)	6
Success (ground pad)	5
Failure (drone ship)	5
Controlled (ocean)	3
Uncontrolled (ocean)	2
Failure (parachute)	1
Precluded (drone ship)	1


Section 3
Launch Sites
Proximities Analysis

Launch site locations

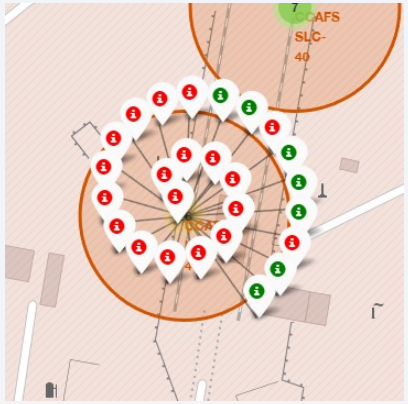


Most launch sites are located in **Florida**, while VAFB SLC-4E is the only site located in **California**

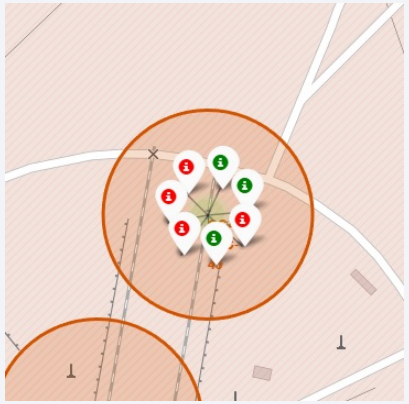
Color-labeled markers for landing outcomes

Color	Meaning
	Success
	Fail

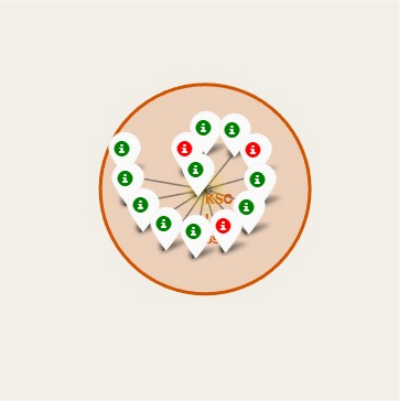
KSC LC-39A has a relatively **higher success rate** compared to the other launch sites



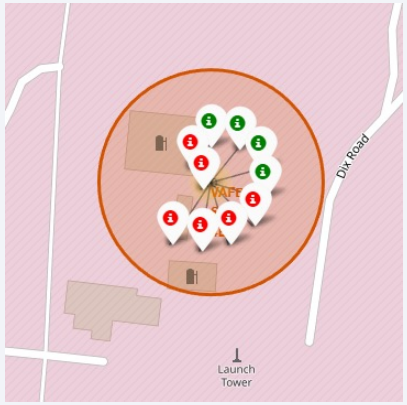
CCAFS-LC-40



CCAFS-SLC-40



KSC LC-39A



VAFB-SLC-4E

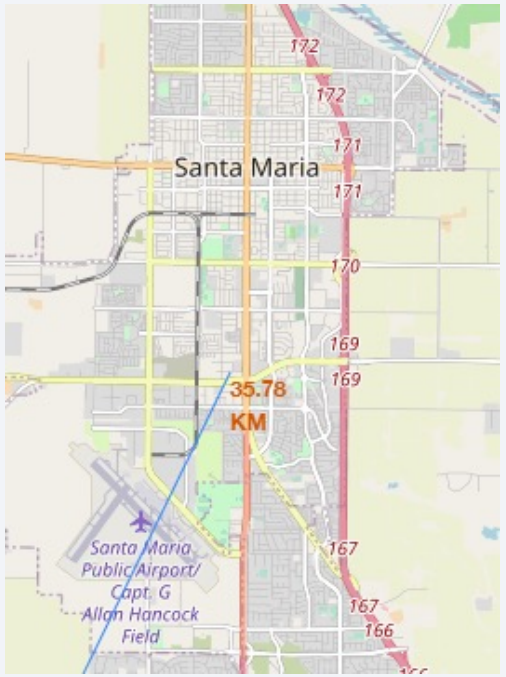
Nearest landmarks from VAFB SLC-4E

Nearest coastline and Santa Barbara Subdivision MT1 Railway

Launch sites are **relatively far from the nearest possible place a human can be** (~1km at minimum)

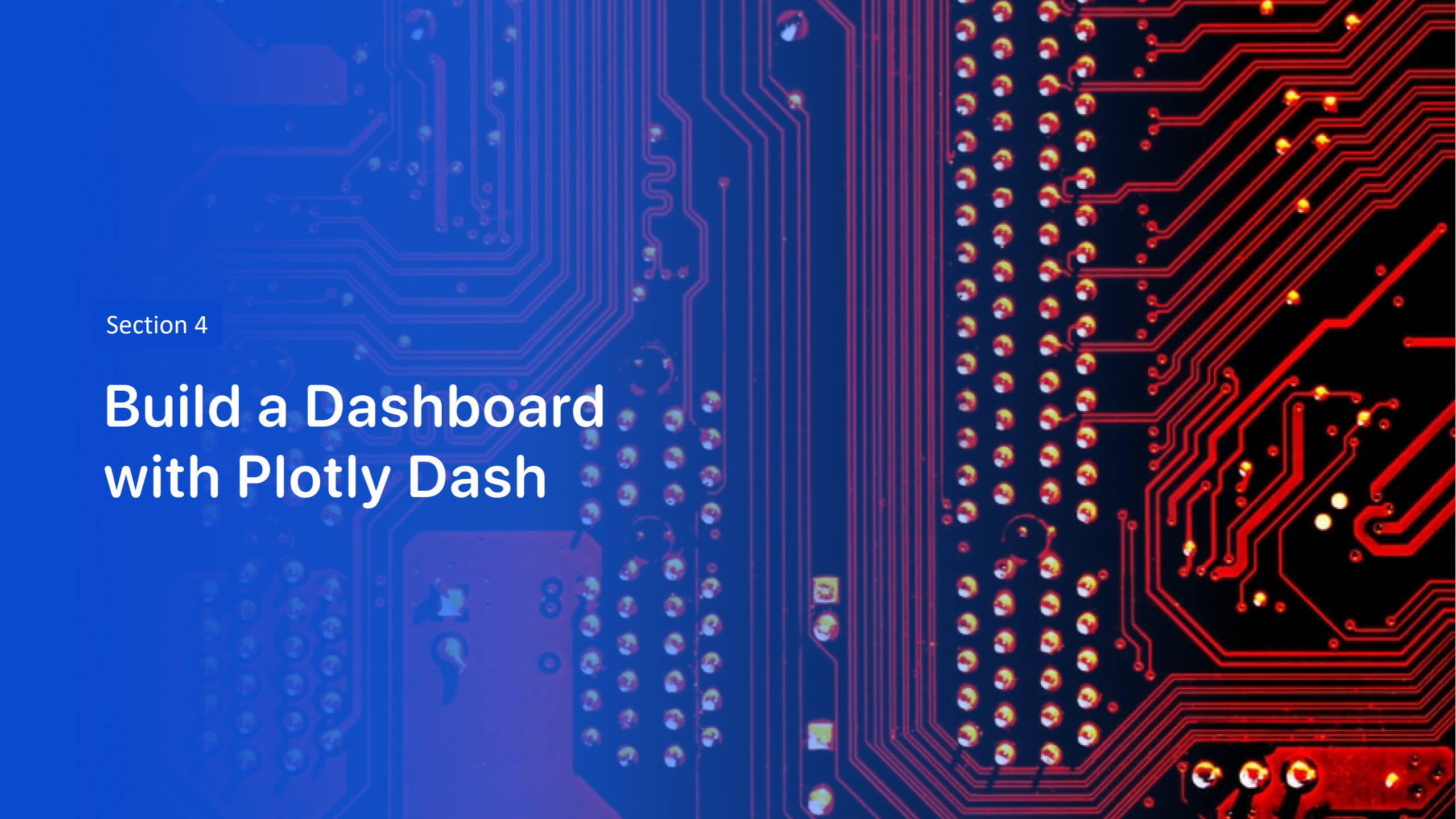
Nearest landmarks from VAFB SLC-4E

California State Route 246

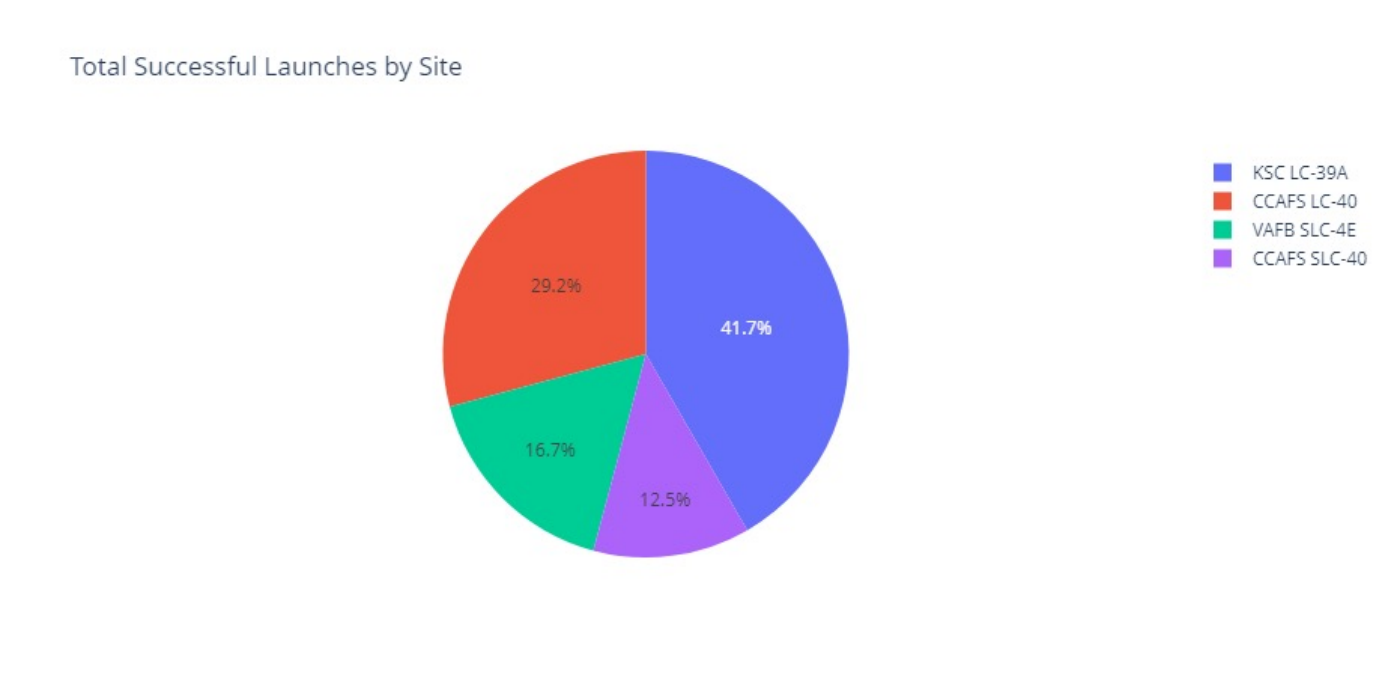


Santa Maria City, CA

Populated areas such as highways and cities are conveniently **very far from a launch site**

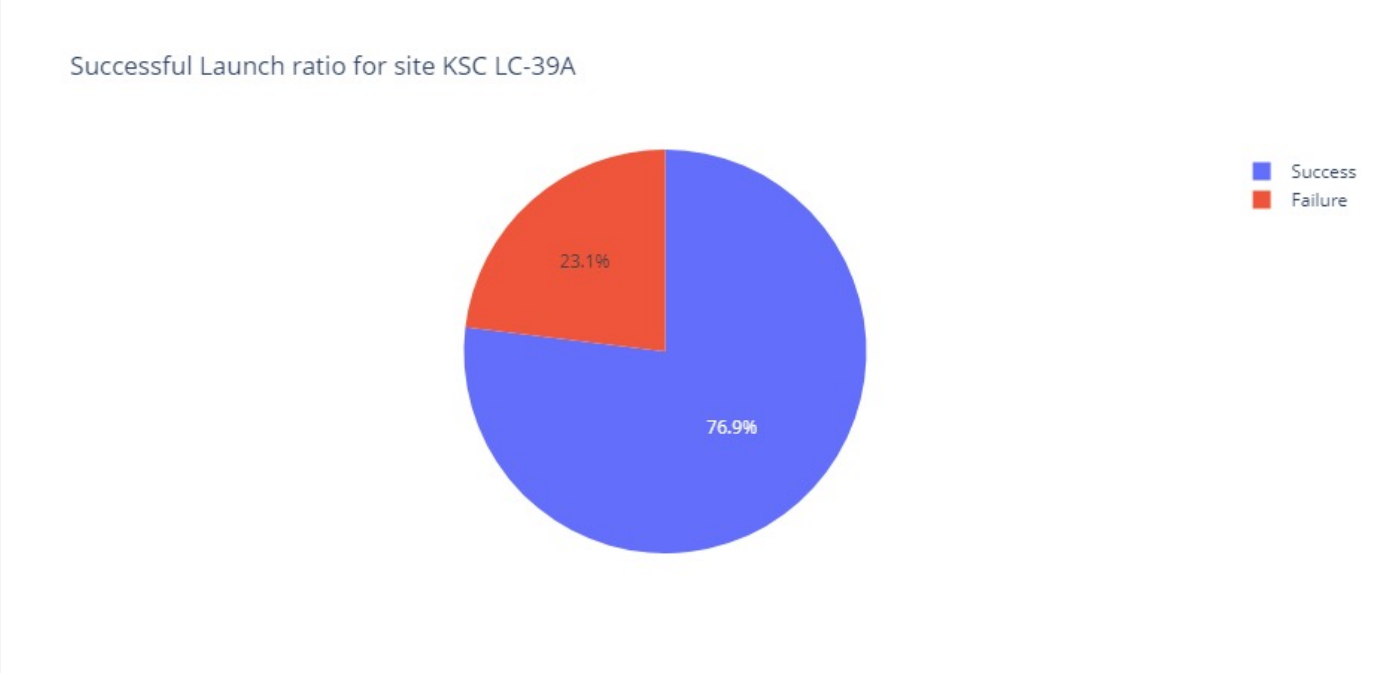

Section 4
Build a Dashboard with Plotly Dash

Success rate distribution per site



KSC LC-39A relatively has the highest launch success rate out of all the sites. Meanwhile, **CCAFS SLC-40** has the lowest success rate.

Success-Failure ratio for KSC LC-39A



KSC LC-39A has a huge success rate of **76.9%** and a low failure rate of **23.1%**

Launch sites' correlation between Payload and Success Rate

Boosters with payloads weighing **between 2000kg and 4000kg** have seen a reasonably **high success rate** compared to those weighing below 1000kg and those above 4000kg

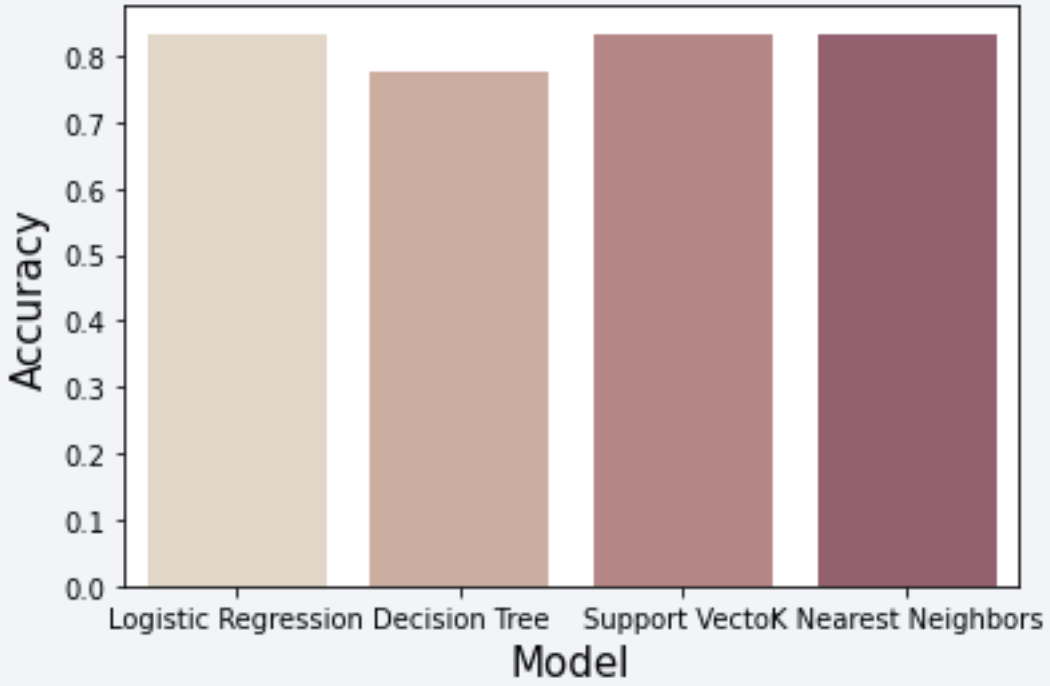
Launch sites' correlation between Payload and Success Rate

Launching a booster that weighs **higher than 5000kg** seems **risky** as there's a relatively low success rate for payloads of this weight

Section 5

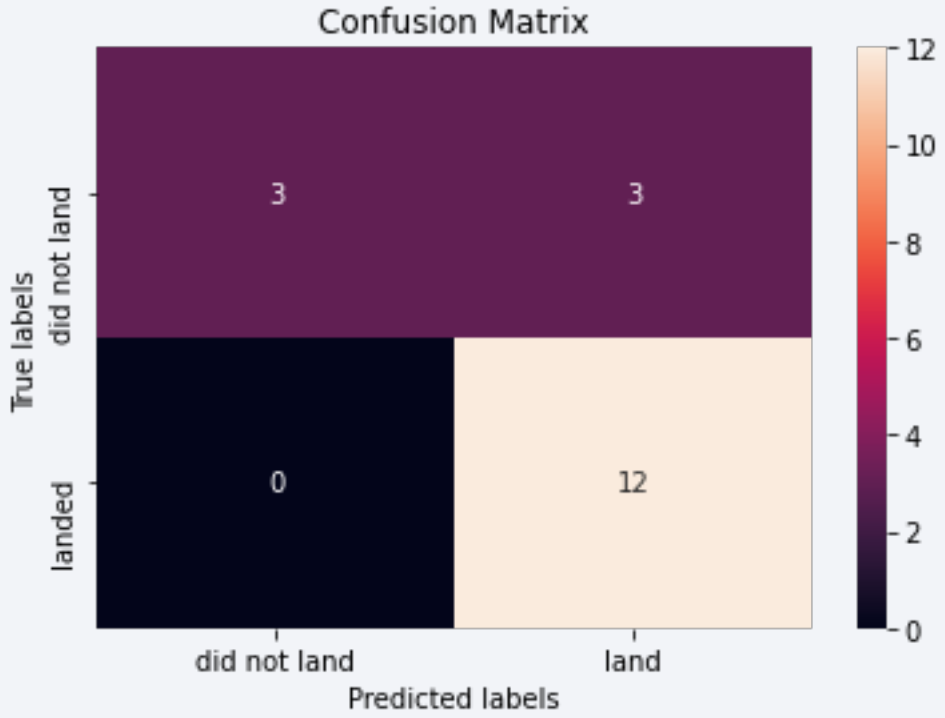
Predictive Analysis (Classification)

Classification Accuracy



The bar chart shows that **Logistic Regression, Support Vector and KNN** have shown the highest (and equal) **accuracy of 83%** when used on the test data.

Confusion Matrix



Logistic Regression, Support Vector, and KNN has classified a total of:

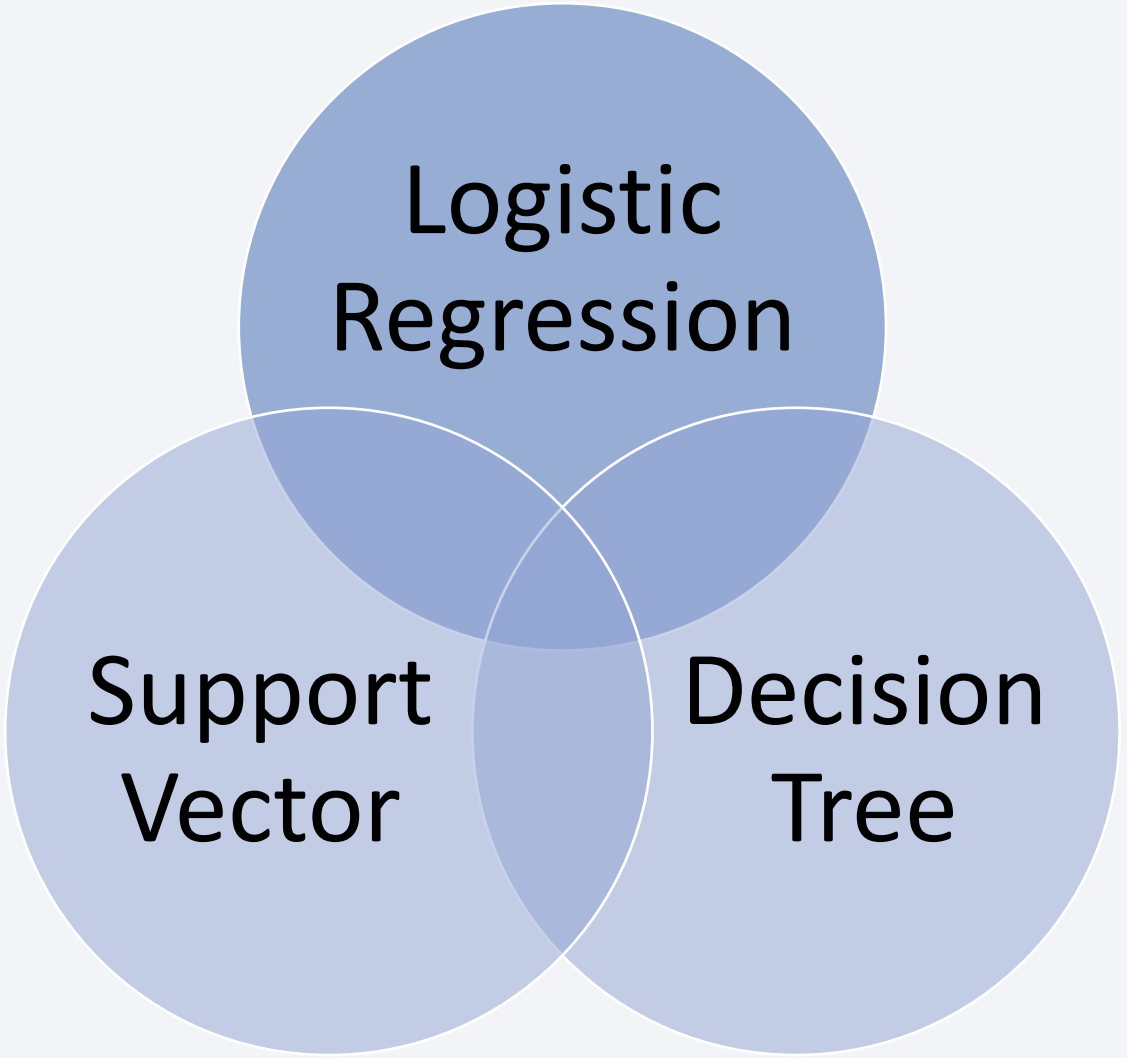
Correct prediction	
3	True Positive
12	True Negative

False prediction	
3	False Positive
0	False Negative

Conclusions

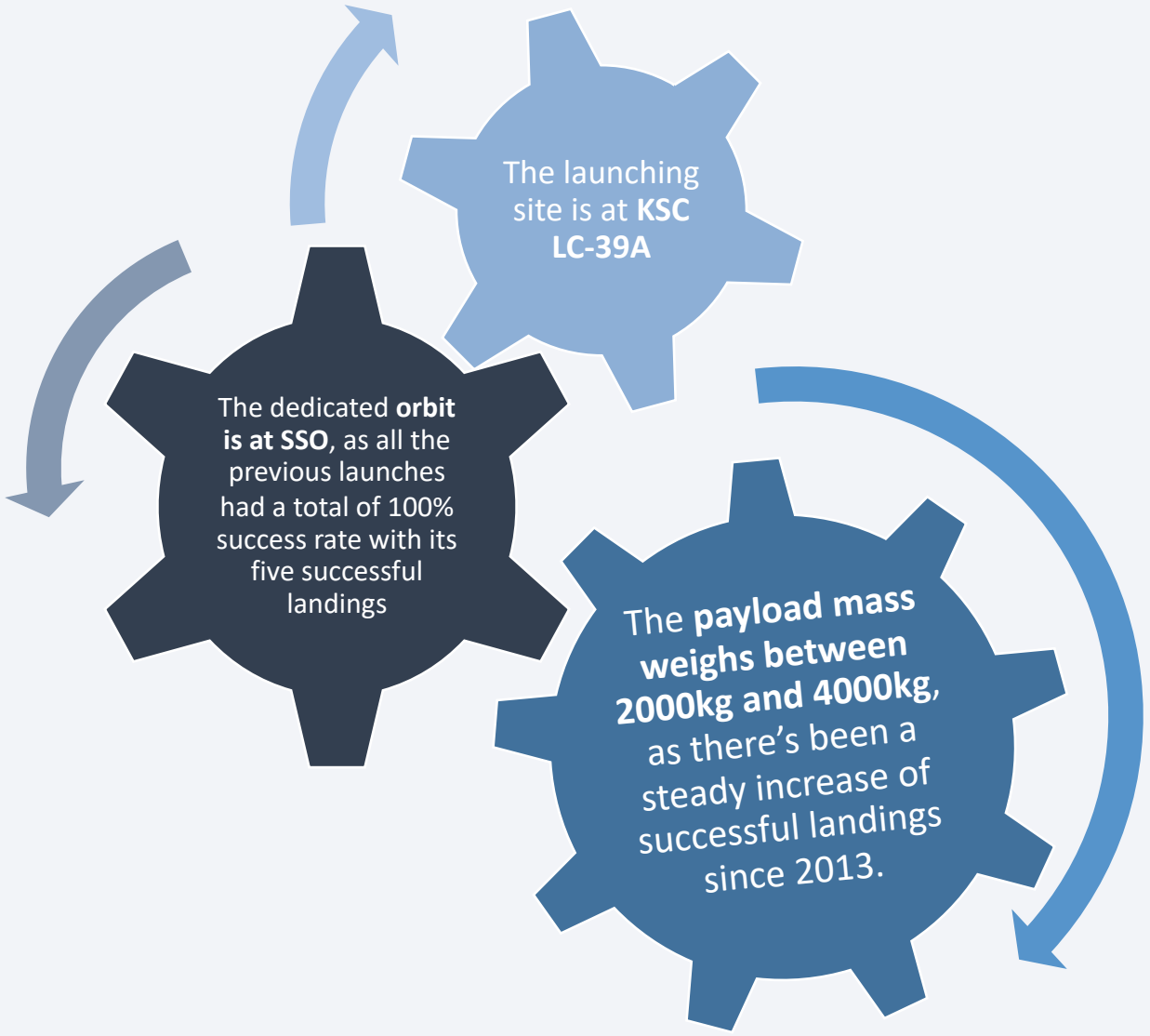
The following are the classification models that can (be used to) predict the first stage's landing outcome for the next launch

These three models have shown an equal amount of accuracy towards the test data



Conclusions

There’s a high chance of SpaceX reusing the first stage of its Falcon9 rocket if one more of the following conditions are met:



Appendix

Query Editor | Query History

```
1  SELECT orbit, COUNT(orbit)
2  FROM spacex
3  GROUP BY orbit
4  ORDER BY COUNT(orbit) DESC;
```

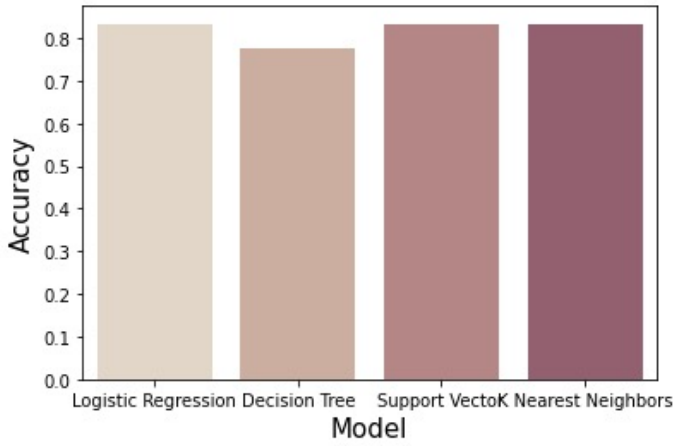
Data Output | Explain | Messages

	orbit character varying (50)	count bigint
1	GTO	30
2	LEO (ISS)	26
3	LEO	25
4	Polar LEO	8
5	SSO	6
6	MEO	3
7	HEO	2
8	Sub-orbital	1

```
Accuracy_Chart = pd.DataFrame(list(methods.items()),columns = ['model','accuracy'])
Accuracy_Chart
```

	model	accuracy
0	Logistic Regression	0.833333
1	Decision Tree	0.777778
2	Support Vector	0.833333
3	K Nearest Neighbors	0.833333

```
import matplotlib.pyplot as plt
import seaborn as sns
sequential_colors = sns.color_palette("ch:.25")
sns.barplot(x='model',y='accuracy', palette=sequential_colors, data=Accuracy_Chart)
plt.xlabel('Model', fontsize=15)
plt.ylabel('Accuracy', fontsize=15)
plt.show()
```



Thank you!