IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Luis Vivar
February, 2024

OUTLINE

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

EXECUTIVE SUMMARY

- **Summary of methodologies:**
  - Data Collection through SpaceX API and Web Scraping
  - Data Wrangling
  - Exploratory Data Analysis with SQL & DataVisualization
  - Interactive Visual Analytics with Folium
  - Machine Learning Prediction

- **Summary of all results**
  - Exploratory Data Analysis result
  - Interactive Analysis graphs
  - Predictive Analysis result

INTRODUCTION

Since SpaceX can recycle its initial stage, the Falcon 9 rockets' launch comes at a price of 62 million dollars, a significant saving compared to the 165 million dollars charged by some other providers. Consequently, being able to forecast the success of the first stage landing allows us to estimate launch costs. This insight becomes crucial for potential competitors wishing to challenge SpaceX's bids for rocket launches.

Can we predict the outcome of a new launch based on past data regarding the first stage's landing success? And can we determine the optimal strategy for achieving a successful launch?

Section 1

Methodology

METHODOLOGY

Executive Summary

- Data Collection methodology:
  - SpaceX API & Web Scraping
- Data Wrangling:
  - Converted categorical features into numerical values through “One-Hot Encoding”
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models:
  - Used GridSearchCV method from the SciKit library to find the best model by using cross-validation

DATA COLLECTION

SPACEX API

1. The data was collected through the *SpaceX* API and the use of the *requests* library.
2. The response content was decoded into *JSON* format via the use of the json() function.
3. The JSON was turned into a *pandas* dataframe using the json_normalize() function.
4. The data was cleaned and checked for missing values.

SpaceX REST API → **JSON** → **DataFrame**

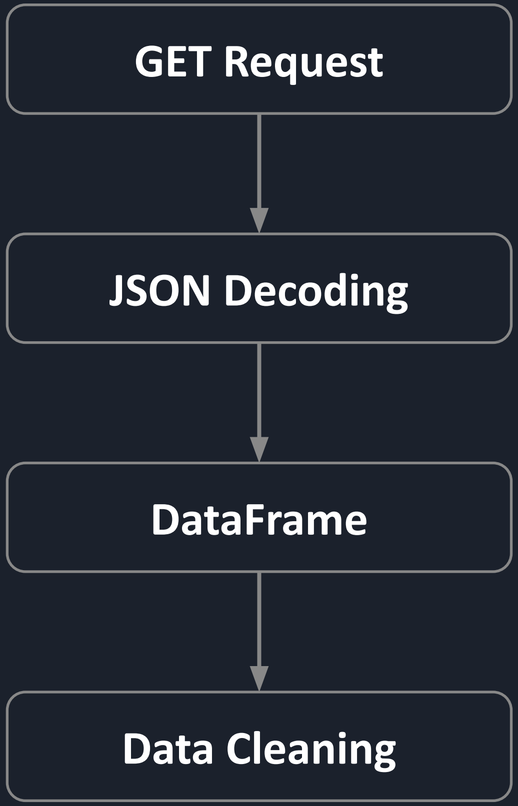
WEB

1. The data was collected through Web Scraping off Wikipedia on Falcon 9 launch records with the help of *BeautifulSoup* library.
2. The launch records were extracted as an HTML table, parsed, and converted into a pandas *dataframe*.

Wikipedia → **HTML** → **DataFrame**

Data Collection - SpaceX API

See on GitHub




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

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

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

✓ 0.0s

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

# Replace the np.nan values with its mean value
data['PayloadMass'] = data['PayloadMass'].replace(np.nan, payloadMass_mean)
data
```

✓ 0.0s

Data Collection - Web Scraping

See on GitHub

GET Request

BeautifulSoup

Extract Columns

(cont.)

```
# uses requests.get() method with the provided static_url
# assigns the response to an object
response = requests.get(static_url)
response
```

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

```
column_names = []

# Applies find_all() function with `th` element on first_launch_table
# Iterates each th element and apply the provided extract_column_from_header() to get a column name
# Appends the Non-empty column name (`if name is not None and len(name) > 0`) into a list called column_names
all_th = first_launch_table.find_all('th')

for th in all_th:
    name = extract_column_from_header(th)
    if name != None and len(name) > 0:
        column_names.append(name)
```

Data Collection - Web Scraping (cont.)

Parsing

```python
launch_dict= dict.fromkeys(column_names)

# Removes irrelevant 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']=[]
```

```python
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders collapsible")):

    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
            datatimelist=date_time(row[0])
            launch_dict['Flight No.'].append(flight_number)

            # Date value
            date = datatimelist[0].strip(',')
            launch_dict['Date'].append(date)

            # Time value
            time = datatimelist[1]
            launch_dict['Time'].append(time)

            # Booster version
            bv=booster_version(row[1])
            launch_dict['Version Booster'].append(bv)

            if not(bv):
                bv=row[1].a.string
            print(bv)

            # Launch Site
            launch_site = row[2].a.string
            launch_dict['Launch site'].append(launch_site)

            # Payload
            payload = row[3].a.string
            launch_dict['Payload'].append(payload)

            # Payload Mass
            payload_mass = get_mass(row[4])
            launch_dict['Payload mass'].append(payload_mass)

            # Orbit
            orbit = row[5].a.string
            launch_dict['Orbit'].append(orbit)

            # Customer
            customer = row[6].a
            launch_dict['Customer'].append(customer)

            # Launch outcome
            launch_outcome = list(row[7].strings)[0]
            launch_dict['Launch outcome'].append(launch_outcome)

            # Booster landing
            booster_landing = landing_status(row[8])
            launch_dict['Booster landing'].append(booster_landing)
```

(cont.)

Data Collection - Web Scraping (cont.)

DataFrame

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

CSV Export

```
df.to_csv('spacex_web_scraped.csv', index=False)
```

Data Wrangling

See on GitHub

We will convert categorical features into numerical values. Mainly, the data found in the *outcome* column. If the landing was successful, we assign it a 1; otherwise, a 0 for unsuccessful landings.

First, we load the CSV we exported from our web scraping.

We load the SpaceX dataset from last section.

```python
df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_1.csv")
df.head(10)
```

Python

Data Wrangling (cont.)

Next, we identify and calculate the percentage of the missing values in each attribute.

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

```
FlightNumber       0.000000
Date               0.000000
BoosterVersion     0.000000
PayloadMass        0.000000
Orbit              0.000000
LaunchSite         0.000000
Outcome            0.000000
Flights            0.000000
GridFins           0.000000
Reused             0.000000
Legs               0.000000
LandingPad        28.888889
Block              0.000000
ReusedCount        0.000000
Serial             0.000000
Longitude          0.000000
Latitude           0.000000
dtype: float64
```

Data Wrangling (cont.)

Now, we identify which columns are numerical and categorical:

```
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 Wrangling (cont.)

We use the .value_counts() method to determine the number and occurrence of each orbit in the column *Orbit*

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

Python

```
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 (cont.)

We use the .value_counts() method on the *Outcome* column to determine the number of landing_outcomes. Then, we assign it to the *landing_outcomes* variable.

```
# 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
```

Data Wrangling (cont.)

- *'True Ocean'* means the mission outcome was successfully landed to a specific region of the ocean while '*False Ocean'* means the mission outcome was unsuccessfully landed to a specific region of the ocean.

- *'True RTLS'* means the mission outcome was successfully landed to a ground pad '*False RTLS'* means the mission outcome was unsuccessfully landed to a ground pad.

- *'True ASDS'* means the mission outcome was successfully landed to a drone ship '*False ASDS'* means the mission outcome was unsuccessfully landed to a drone ship.

- *'None ASDS'* and '*None None'* these represent a failure to land.

```
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
```

Data Wrangling (cont.)

We create a set of outcomes where the second stage did not land successfully:

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

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

Now, we create a *Landing Outcome* label from the *Outcome* column.

Using the *Outcome* column, we create a list where the element is 0 if the corresponding row in *Outcome* is in the set *bad_outcome*; otherwise, it's 1. Then, we assign it to the variable *landing_class*.

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

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

landing_class
```

Data Wrangling (cont.)

This variable below will represent the classification variable that represents the outcome of each launch.

If the value is 0, the first stage did not land successfully; 1 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

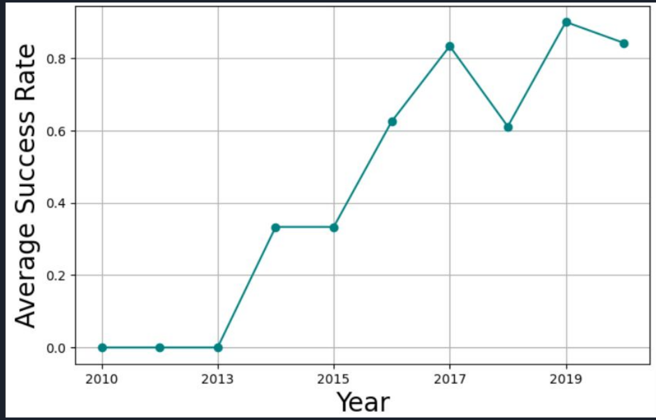
See on GitHub

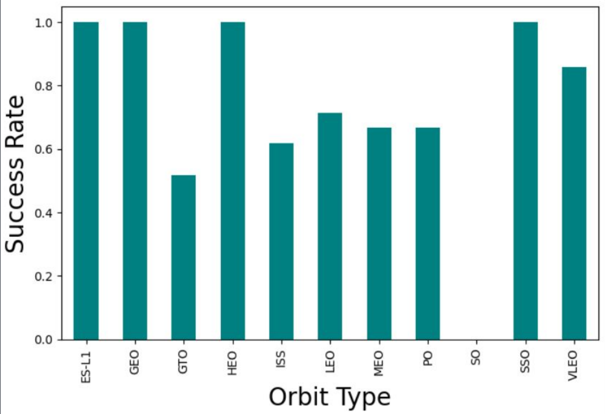
We used the following graphs to visualize the data:

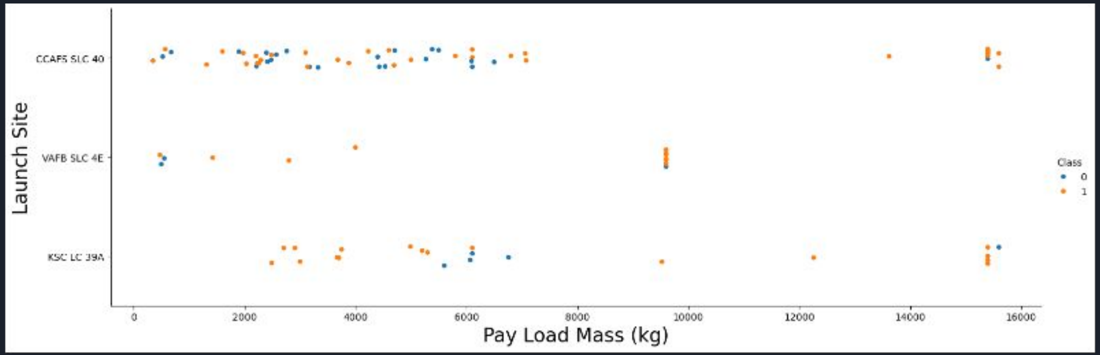
- **Scatter Plot:** to understand the relationship between the records' features (e.g. orbit type and success rate)

- **Line Chart:** to graph the yearly trend of success rate

- **Bar Plot:** to plot the rate of orbit type success







EDA with SQL

See on GitHub

We executed the following SQL queries:

- Names of each unique launch site
- Total Payload Mass carried by boosters launched by NASA
- Average Payload mass carried by booster version F9 v1.1
- Data of first successful landing outcome in ground pad
- Names of booters that succeeded in drone ship and whose payload mass falls between two quantities
- Total number of successful and failed mission outcomes
- Names of booster versions that have carried the maximum payload mass
- Count Rank of landing outcomes between two dates sorted in descending order

```
unique_launch_sites = """
SELECT DISTINCT "Launch_Site"
FROM SPACEXTABLE;
"""

cur.execute(unique_launch_sites)

rows = cur.fetchall()

for row in rows:
    print(row)
```

```
('CCAFS LC-40',)
('VAFB SLC-4E',)
('KSC LC-39A',)
('CCAFS SLC-40',)
```

Interactive Map with Folium

See on GitHub

The interactive map has the following icons and guides:

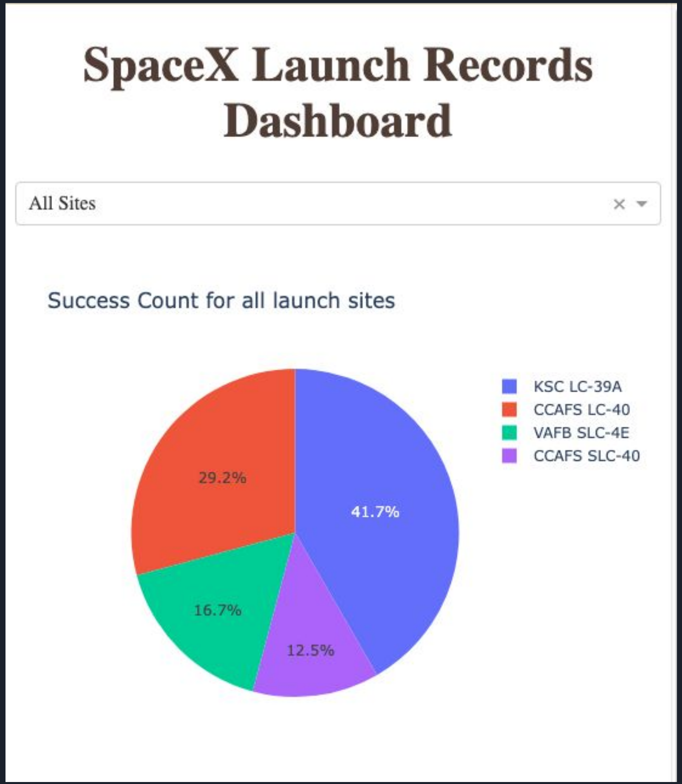
- Circles represent launch sites
- Markers represent labels
- MarkerCluster for Successful and Failed launches
- Lines represent and calculate the distance between launch sites and their proximities

Dashboard with Plotly Dash

See on GitHub

This interactive dashboards features:

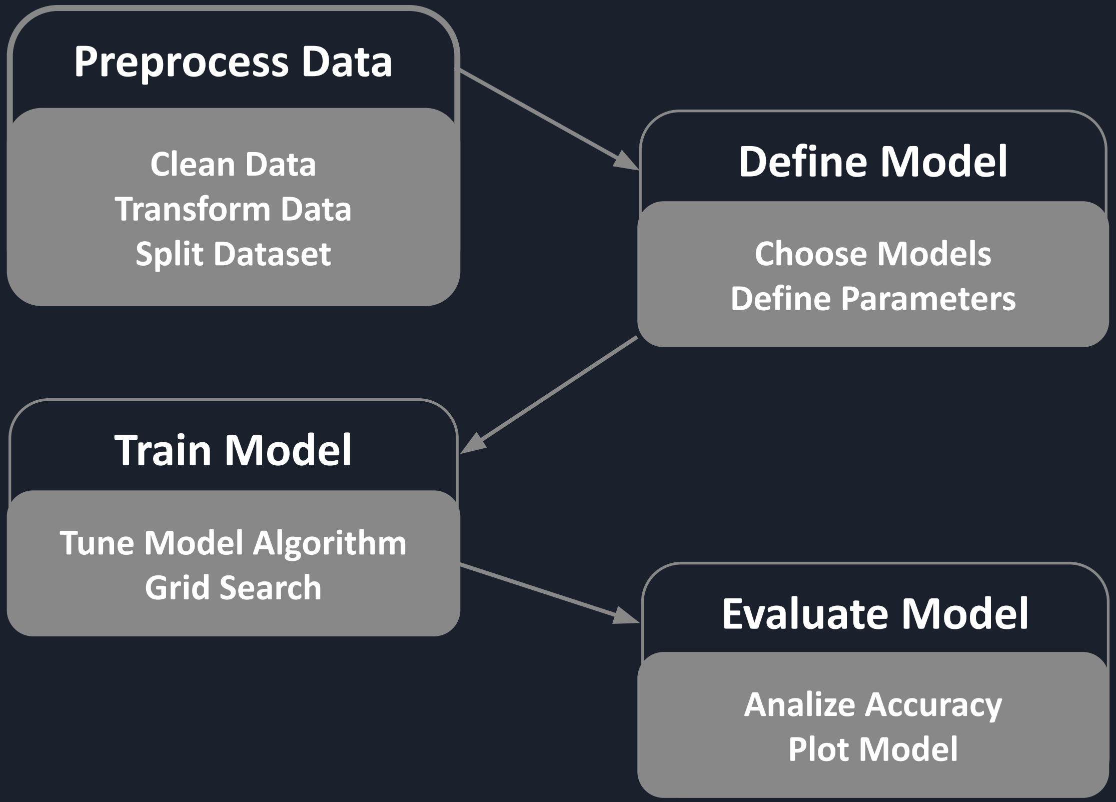
- A **drop down menu** and a **pie chart**, to reflect successful launches by launch site.
- A **range slider** and a **scatter plot**, to analyze the correlation between payload and success by each launch site.



Predictive Analysis (Classification)

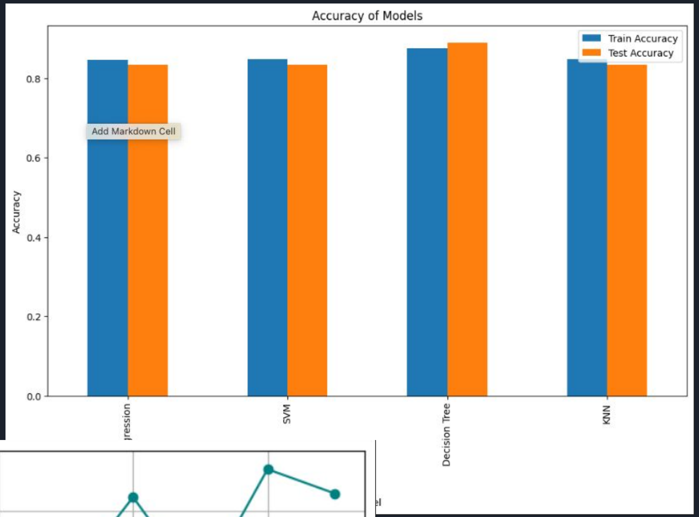
See on GitHub

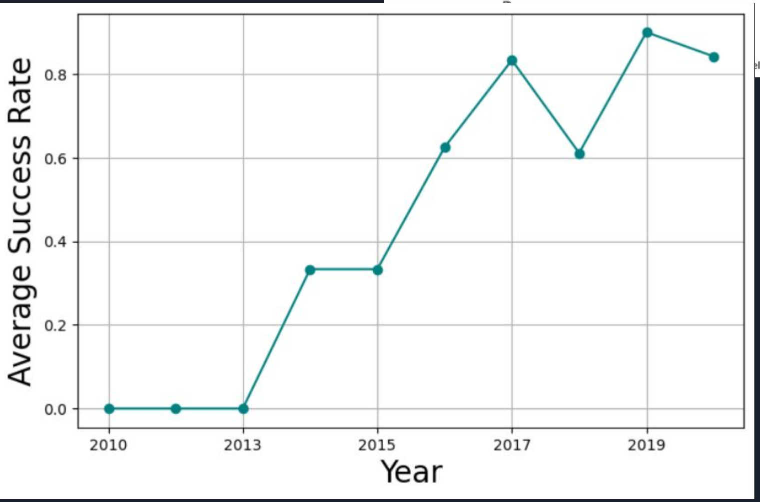
- Prepared the data
- Created a column for the class
- Standardized the data
- Split the data into training data and test data
- Defined Machine Learning models
- Trained the models and performed a Grid Search
- Improved the model using feature engineering and algorithm tuning
- Evaluated the best model



RESULTS

- The *Decision Tree* model was the best for predicting launches' accuracy at 88.8%
- Low-weighted payloads perform better than the heavier payloads
- The orbit types *GEO, HEO, SSO,* and *ES-L1* were the most successful
- The launch site *KSC LC 39A* has the most successful launches
- The success rate for launches is directly proportional to time in years



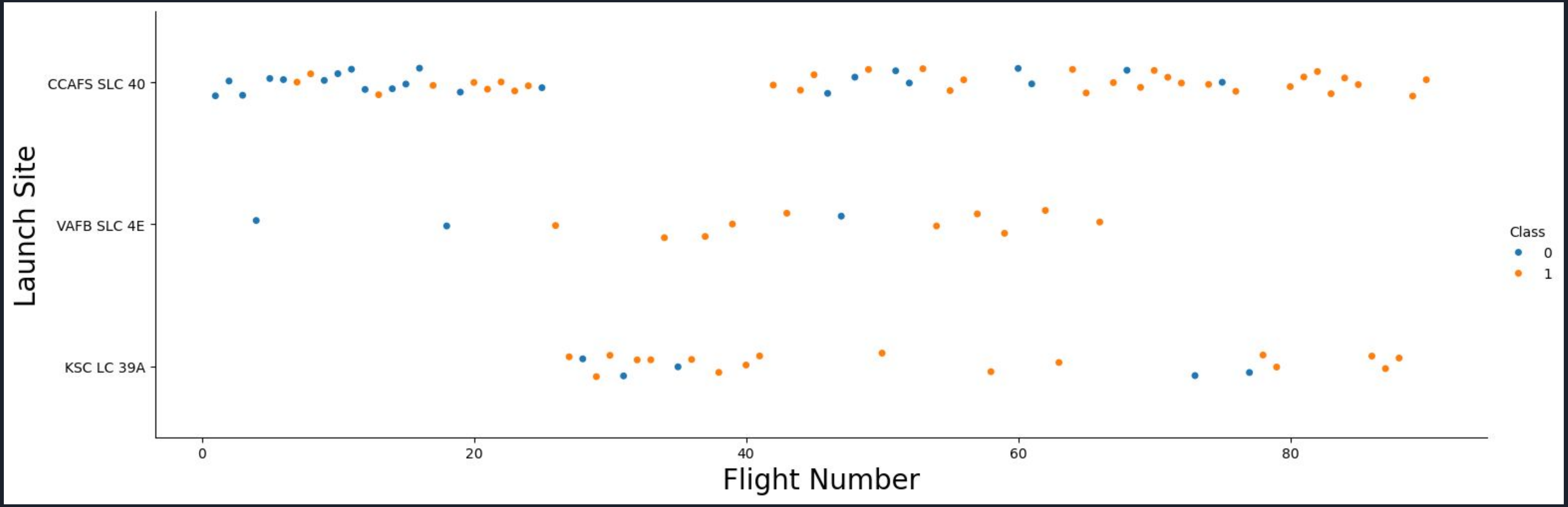


Section 2

Insights drawn from EDA

Flight Number vs. Launch Site

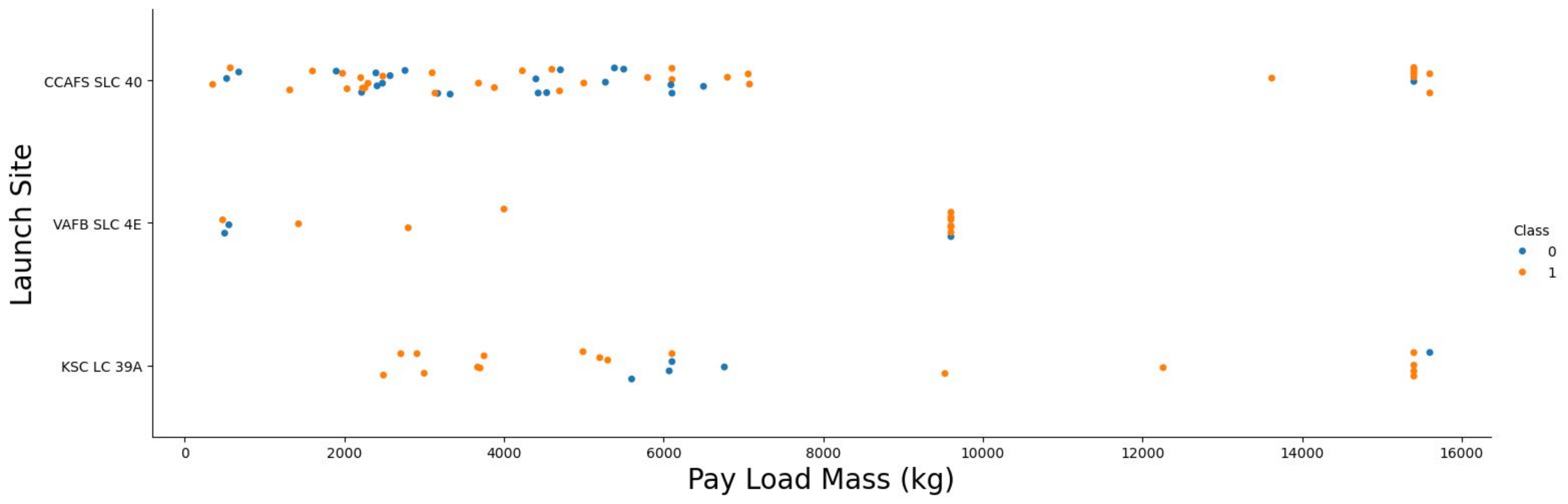
In the graph below, we can see that the launching site known as *CCAFS SLC 40* has had the most successful launches; in addition, it has also had the most number of launches.



Scatter plot showing the relationship between Flight Number and Launch Site

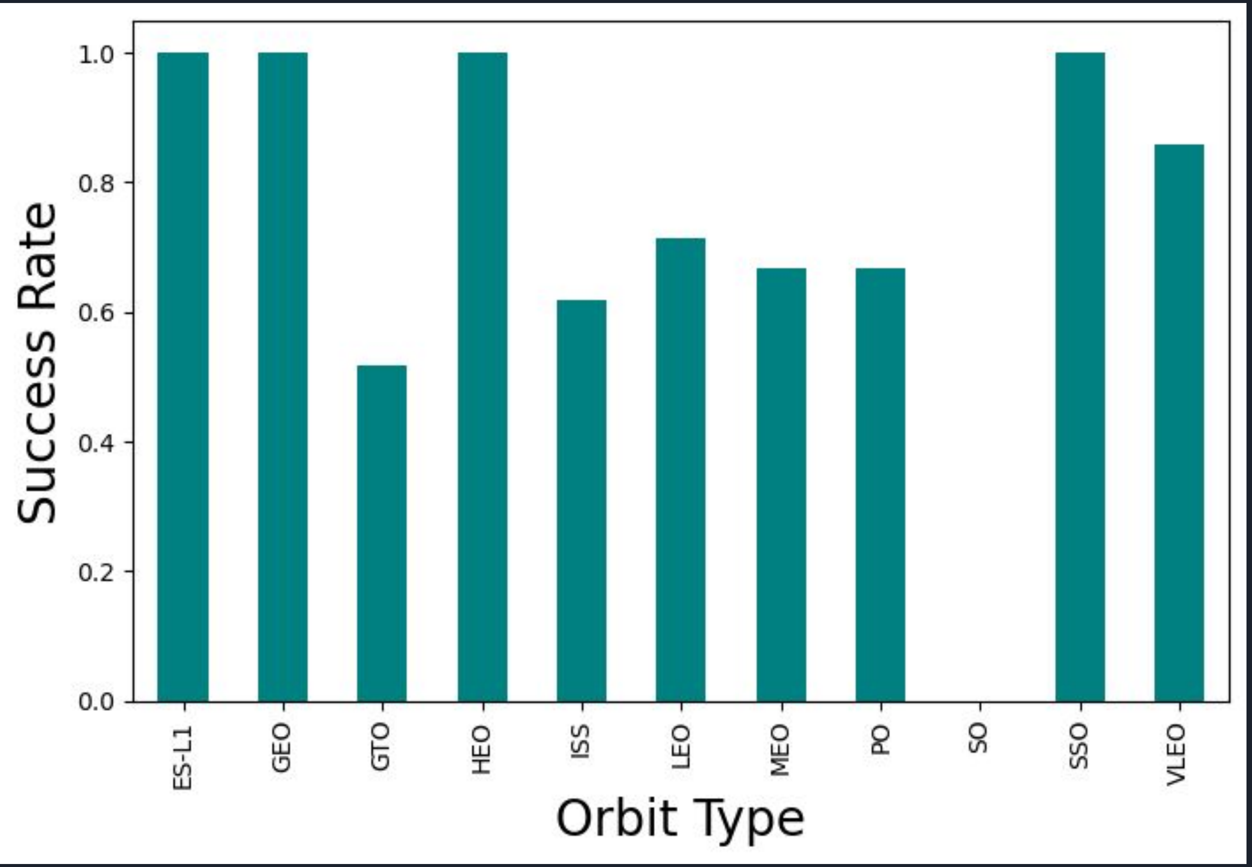
Payload vs. Launch Site

- The launching site known as CCAFS SLC 40 has had the most launches under 80,000 kg of payload mass
- The highest success rate is found in launches whose payload mass is over 9000 kg
  (i.e. the higher the payload mass, the higher the success rate)



Scatter plot showing the relationship between Payload Mass and Launch Site

Success Rate vs. Orbit Type



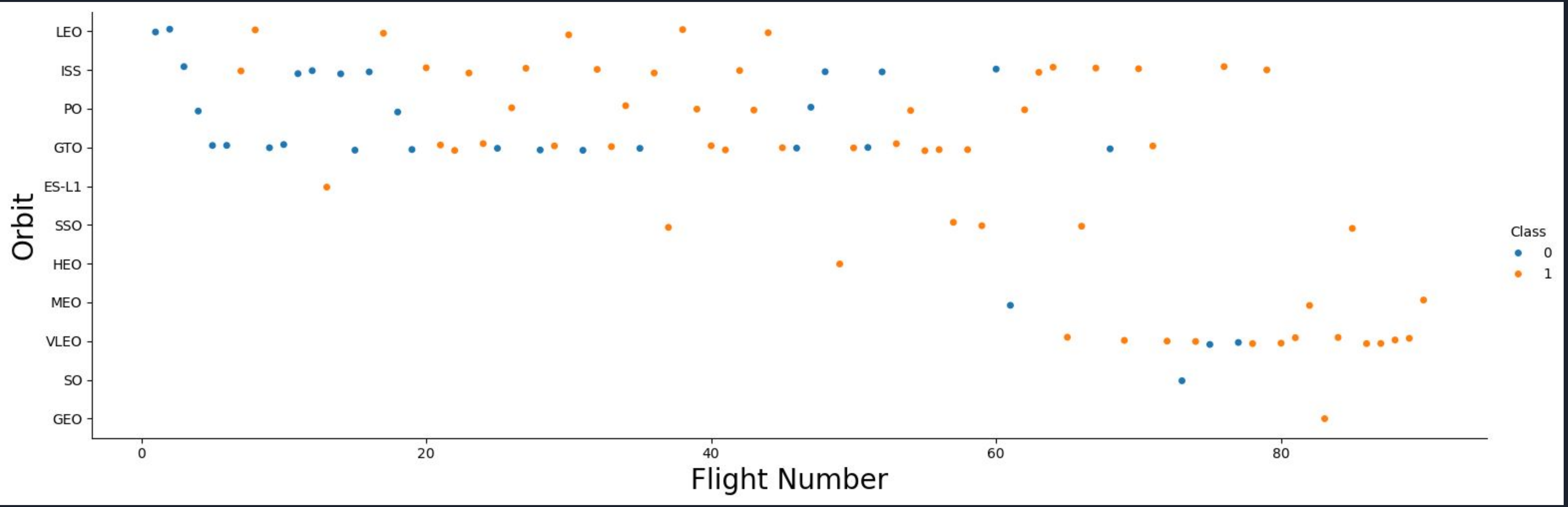
Bar plot showing the relationship between Success Rate and Orbit Type

- The most successful orbit types were ES-L1, GEO, HEO, and SSO with a 100% success rate

- The most unsuccessful orbit type was SO with 0% of success.

Flight Number vs. Orbit Type

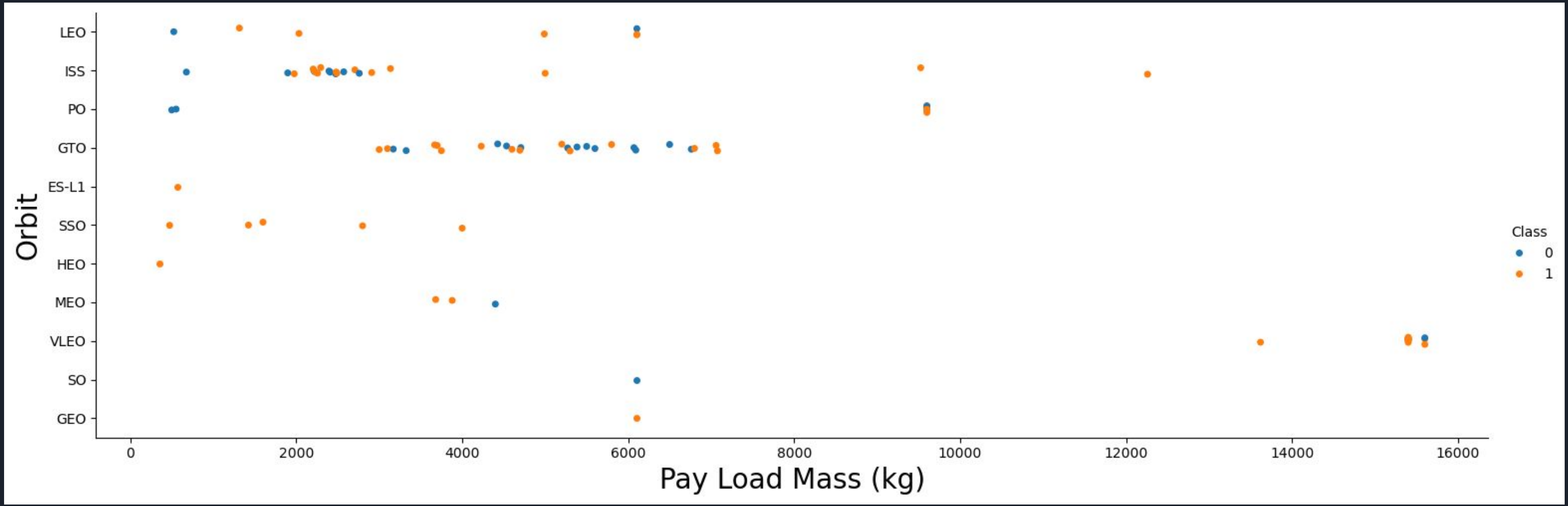
- The orbit types SO and GEO have only one launch each
- Orbit type VLEO seems to be more used in recent years compared to its counterparts
- The LEO orbit type has a direct relation between its number of flights and its success



Scatter plot showing the relationship between Flight Number and Orbit Type

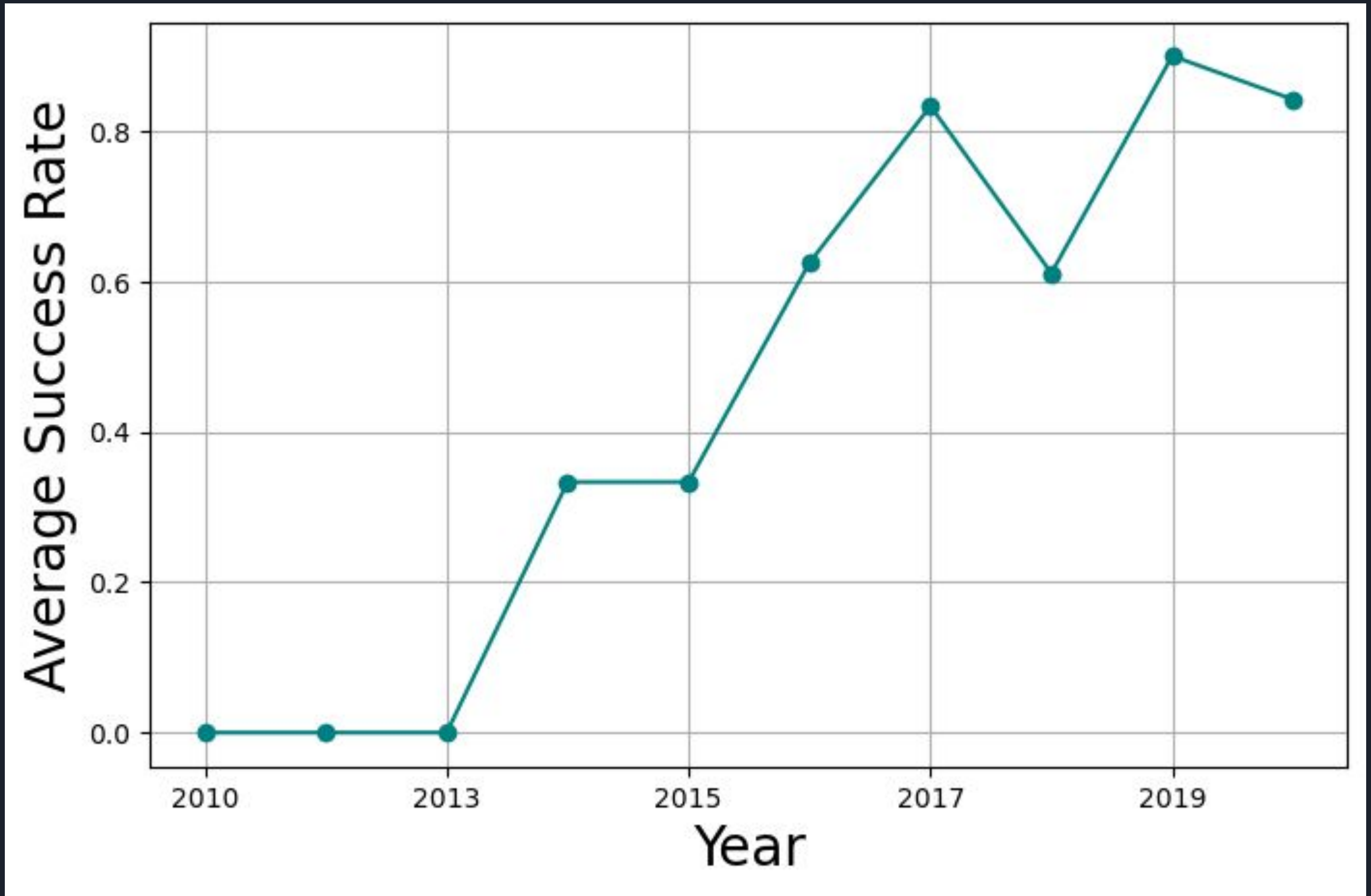
Payload vs. Orbit Type

The orbyt types LEO, ISS, and PO seem to more successful the heavier the payload mass



Scatter plot showing the relationship between Payload Mass and Orbit Type

Launch Success Yearly Trend



Line chart showing the relationship between the pass of years and the average success rate

We can observe how there is a direct relationship between the pass of time and the success rate of launches

All Launch Site Names

```
unique_launch_sites = """
SELECT DISTINCT "Launch_Site"
FROM SPACEXTABLE;
"""

cur.execute(unique_launch_sites)

rows = cur.fetchall()

for row in rows:
    print(row)
```

```
('CCAFS LC-40',)
('VAFB SLC-4E',)
('KSC LC-39A',)
('CCAFS SLC-40',)
```

The DISTINCT statement will allow us to retrieve only different values found in the *Launch Site* column.

(i.e. only unique names, no duplicates)

Launch Site Names Begin with 'CCA'

```
CCA_5 = """
SELECT *
FROM SPACEXTABLE
WHERE "Launch_Site" LIKE "CCA%"
LIMIT 5;
"""

column_names = [description[0] for description in cur.description]
print("Column Names: ", column_names)

cur.execute(CCA_5)
rows = cur.fetchall()

for row in rows:
    print(row)
```

```
Column Names:  ['Date', 'Time (UTC)', 'Booster_Version', 'Launch_Site', 'Payload',
('2010-06-04', '18:45:00', 'F9 v1.0  B0003', 'CCAFS LC-40', 'Dragon Spacecraft Qual
('2010-12-08', '15:43:00', 'F9 v1.0  B0004', 'CCAFS LC-40', 'Dragon demo flight C1,
('2012-05-22', '7:44:00', 'F9 v1.0  B0005', 'CCAFS LC-40', 'Dragon demo flight C2',
('2012-10-08', '0:35:00', 'F9 v1.0  B0006', 'CCAFS LC-40', 'SpaceX CRS-1', 500, 'LE
('2013-03-01', '15:10:00', 'F9 v1.0  B0007', 'CCAFS LC-40', 'SpaceX CRS-2', 677, 'L
```

The LIMIT statement allows us to retrieve only the number of records specified from our queried column.

In addition, the LIKE operator will search for a specified pattern in our queried column.

(i.e. 5 records that contain ‘CCA’ in the Launch_Site column)

Total Payload Mass

```
total_payload_mass_CRS = """
SELECT SUM("PAYLOAD_MASS__KG_")
FROM SPACEXTABLE
WHERE "Customer" LIKE "%NASA (CRS)%"
"""

cur.execute(total_payload_mass_CRS)
total_payload_mass_CRS = cur.fetchone()
total_payload_mass_CRS[0]
```

```
48213
```

The total payload mass carried by boosters from NASA(CRS) is 48213 kg

Average Payload Mass by F9 v1.1

```
average_payload_mass_F9 = """
SELECT AVG("PAYLOAD_MASS__KG_")
FROM SPACEXTABLE
WHERE "Booster_Version" LIKE "%F9 v1.1%"
"""

cur.execute(average_payload_mass_F9)
average_payload_mass_F9 = cur.fetchone()
average_payload_mass_F9[0]
```

```
2534.6666666666665
```

The average payload mass carried by booster version F9 v1.1 is 2534.67 kg

First Successful Ground Landing Date

```
first_successful_landing_date = """
SELECT MIN("Date")
FROM SPACEXTABLE
WHERE "Landing_Outcome" = "Success (ground pad)"
"""

cur.execute(first_successful_landing_date)
first_successful_landing_date = cur.fetchone()
first_successful_landing_date[0]
```

```
'2015-12-22'
```

The first successful landing outcome on ground pad was on December the 22nd, 2015

Successful Drone Ship Landing with Payload between 4000 and 6000

```
successful_drone_ship_names = """
SELECT Booster_Version FROM SPACEXTABLE
WHERE "Landing_Outcome" = "Success (drone ship)"
    AND "PAYLOAD_MASS__KG_" > 4000
    AND "PAYLOAD_MASS__KG_" < 6000;
"""

cur.execute(successful_drone_ship_names)
rows = cur.fetchall()

for row in rows:
    print(row)
```

```
('F9 FT B1022',)
('F9 FT B1026',)
('F9 FT  B1021.2',)
('F9 FT  B1031.2',)
```

All boosters that have successfully landed on drone ships and have had a payload mass between 4000 and 6000 kg

Total Number of Successful and Failure Mission Outcomes

```
successful_outcomes = """
SELECT COUNT('*')
FROM SPACEXTABLE
WHERE "Mission_Outcome" LIKE "Success%"
"""

cur.execute(successful_outcomes)
successful_missions = cur.fetchone()[0]


failed_outcomes = """
SELECT COUNT ('*')
FROM SPACEXTABLE
WHERE "Mission_Outcome" LIKE "Failure%"
"""

cur.execute(failed_outcomes)
failed_missions = cur.fetchone()[0]


successful_missions, failed_missions
```

```
(100, 1)
```

- The total number of successful mission outcomes is 100
- The total number of failed mission outcomes is 1

Boosters Carried Maximum Payload

```
max_payload_mass = """
SELECT "Booster_Version"
FROM SPACEXTABLE
WHERE "PAYLOAD_MASS__KG_" = (
    SELECT MAX("PAYLOAD_MASS__KG_")
    FROM SPACEXTABLE
);
"""

cur.execute(max_payload_mass)
rows = cur.fetchall()

for row in rows:
    print(row)
```

```
('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 ',)
```

All boosters that have carried the maximum payload mass

2015 Launch Records

```
More... ax_payload_mass = """
    SELECT strftime('%m', "Date") as Month, "Landing_Outcome", "Booster_Version", "Launch_Site"
    FROM SPACEXTABLE
    WHERE "Landing_Outcome" = 'Failure (drone ship)' AND strftime('%Y', "Date") = '2015';
    """

    cur.execute(max_payload_mass)
    rows = cur.fetchall()

    for row in rows:
        print(row)
```

```
('01', 'Failure (drone ship)', 'F9 v1.1 B1012', 'CCAFS LC-40')
('04', 'Failure (drone ship)', 'F9 v1.1 B1015', 'CCAFS LC-40')
```

All failed landing in drone ship outcomes in 2015, including their booster version and launch site names

Rank of Landing Outcomes Between 2010/06/04 and 2017/03/20

```
landing_outcomes_ranking = """
SELECT "Landing_Outcome", COUNT(*) as Count
FROM SPACEXTABLE
WHERE "Date" BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY "Landing_Outcome"
ORDER BY Count DESC;
"""

cur.execute(landing_outcomes_ranking)
rows = cur.fetchall()

for row in rows:
    print(row)
```

```
('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)
```

Rank of landing outcomes between the dates 2010-06-04 and 2017-03-20, in descending order

Section 3

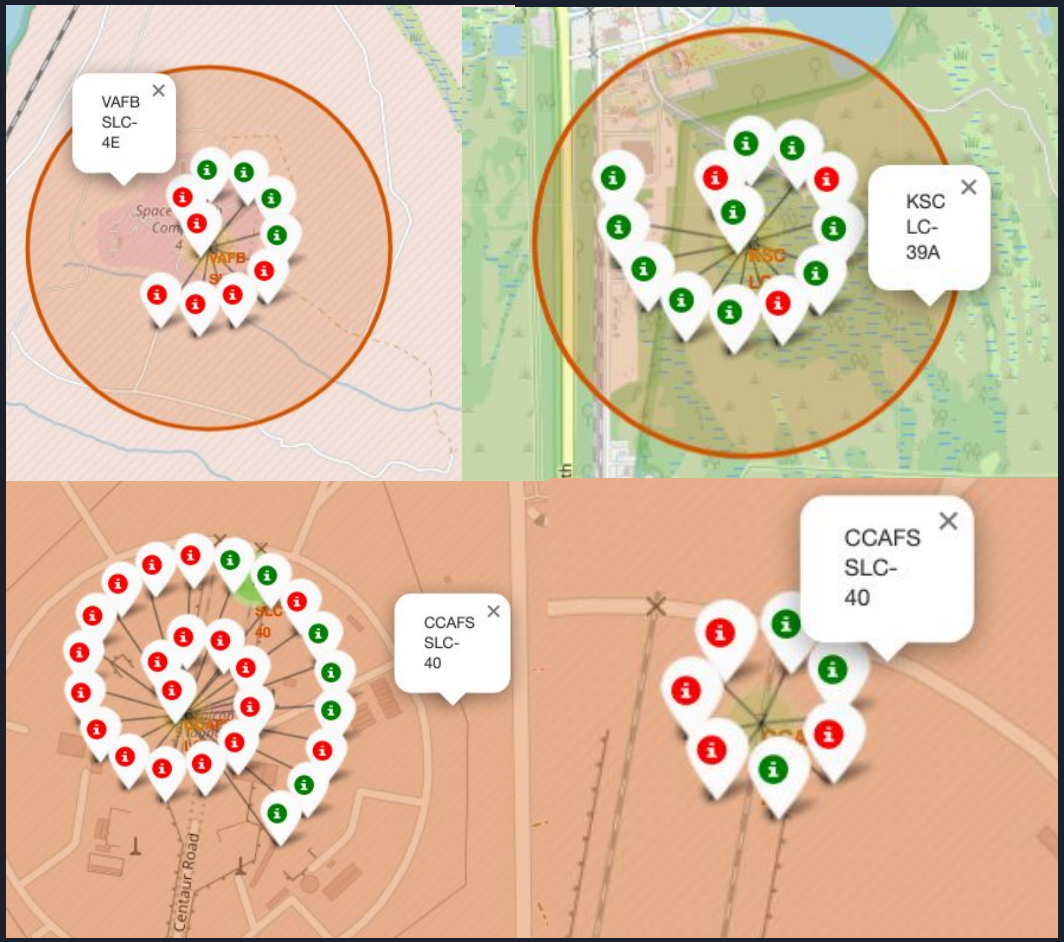
Launch Sites Proximities Analysis

Location of All Launch Sites

- 1 location in California, part of the US west coast
- 3 locations in Florida, parts of the US east coast
- All in the southern part of the country

```
('CCAFS LC-40',)
('VAFB SLC-4E',)
('KSC LC-39A',)
('CCAFS SLC-40',)
```

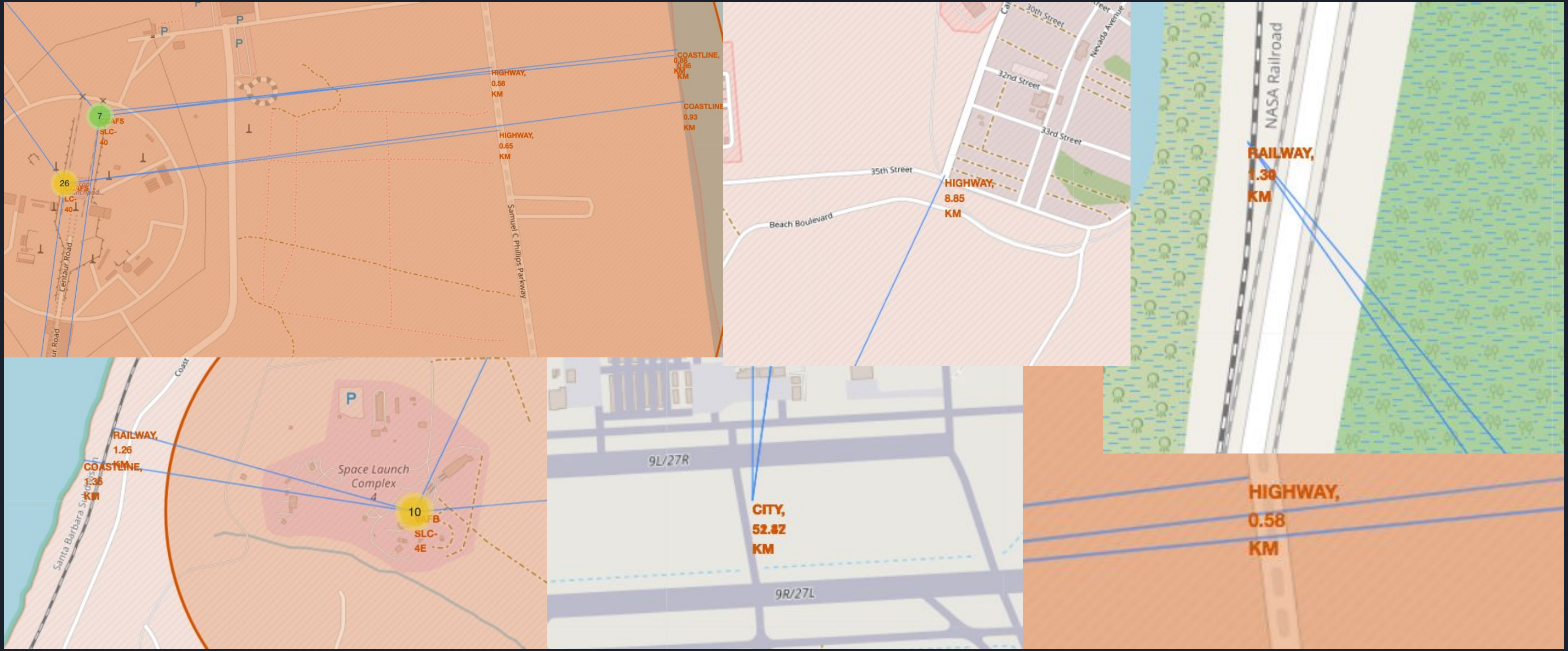
Launch Outcomes



Green markers show successful launches; red markers show failed launches

The highest success rate belongs to the launch site KSC LC-39A

Launch Site Proximities

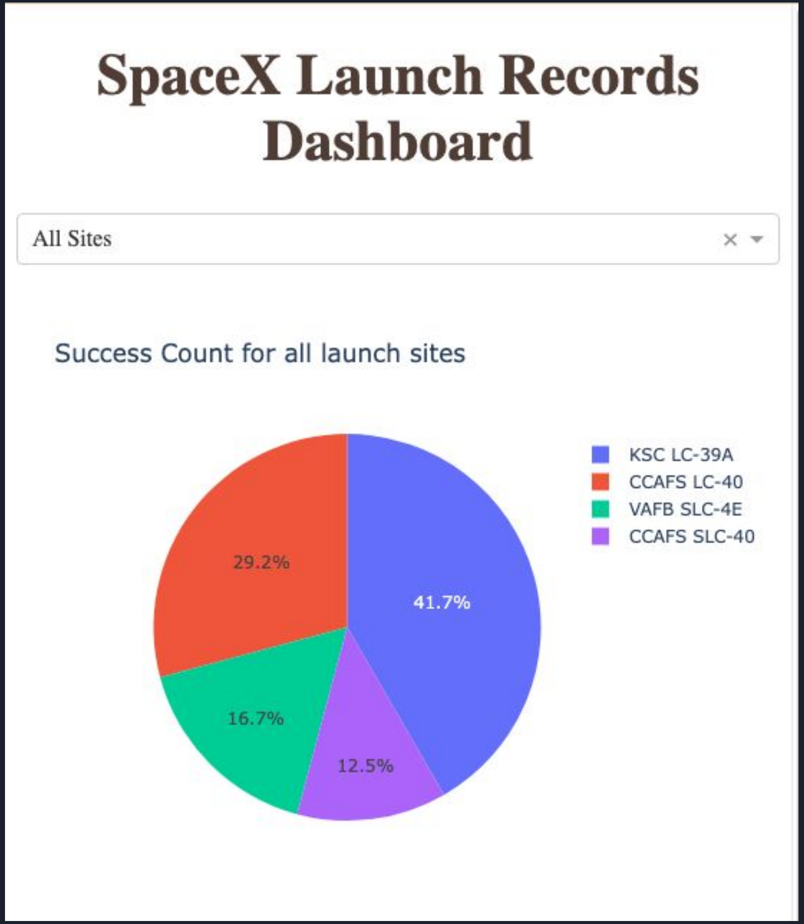


Information about the closest proximities from each launch site (cities, highways, railways and coastlines)

Section 4

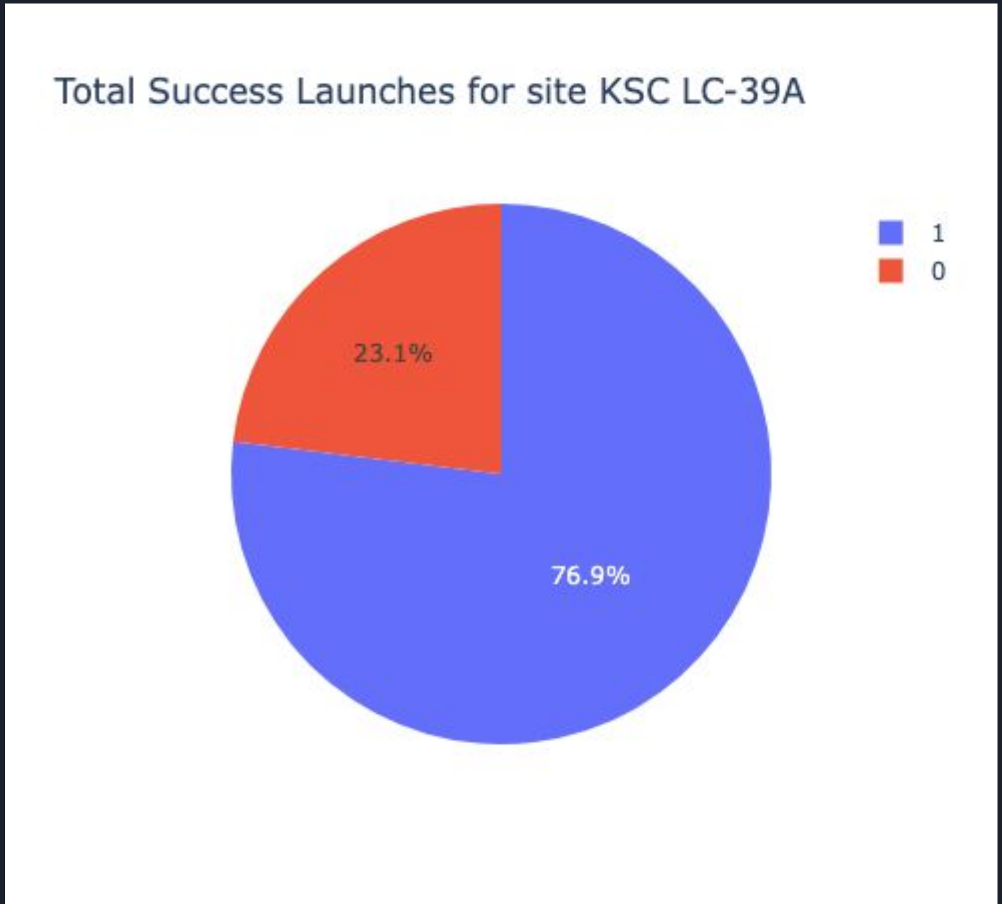
Build a Dashboard with Plotly Dash

Success Rate for All Location Sites



- Of all sites, KSC LC-39A was the most successful one with a success rate of 41.7% out of every success
- Conversely, CCAFS SLC-40 was the least successful one with a success rate of 12.5% out of every success

Launching Site with Highest Success Rate



- KSC LC-39A was the most successful launching site with a success rate of 41.7% out of all successes
- As far as the location goes, out of all launches, 76.9% of them resulted in successful launches
- As for failed launches, this location had a rate of 23.1%

Relation between Payload mass and Success

- The booster version *FT* has the highest success rate
- Lower payloads have a higher success than heavier payloads
- The highest success rate is found between the payloads of 3,000 and 4,000 kg

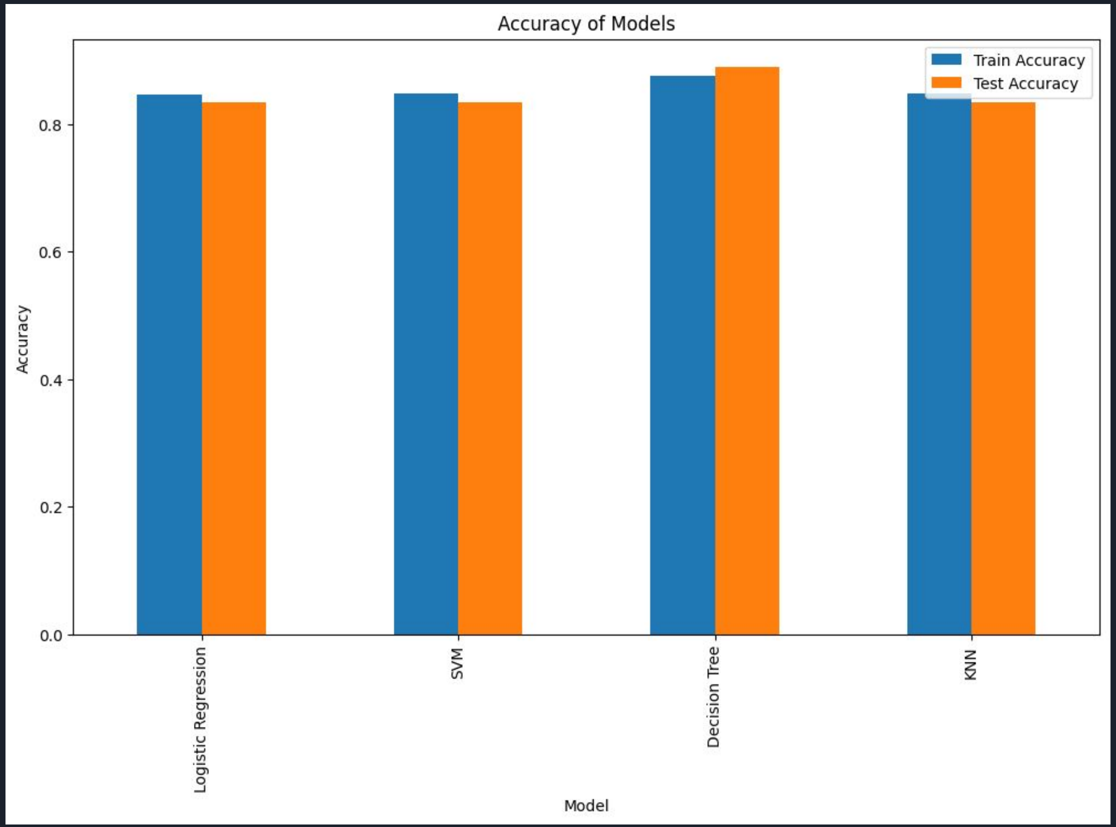
Section 5

Predictive Analysis (Classification)

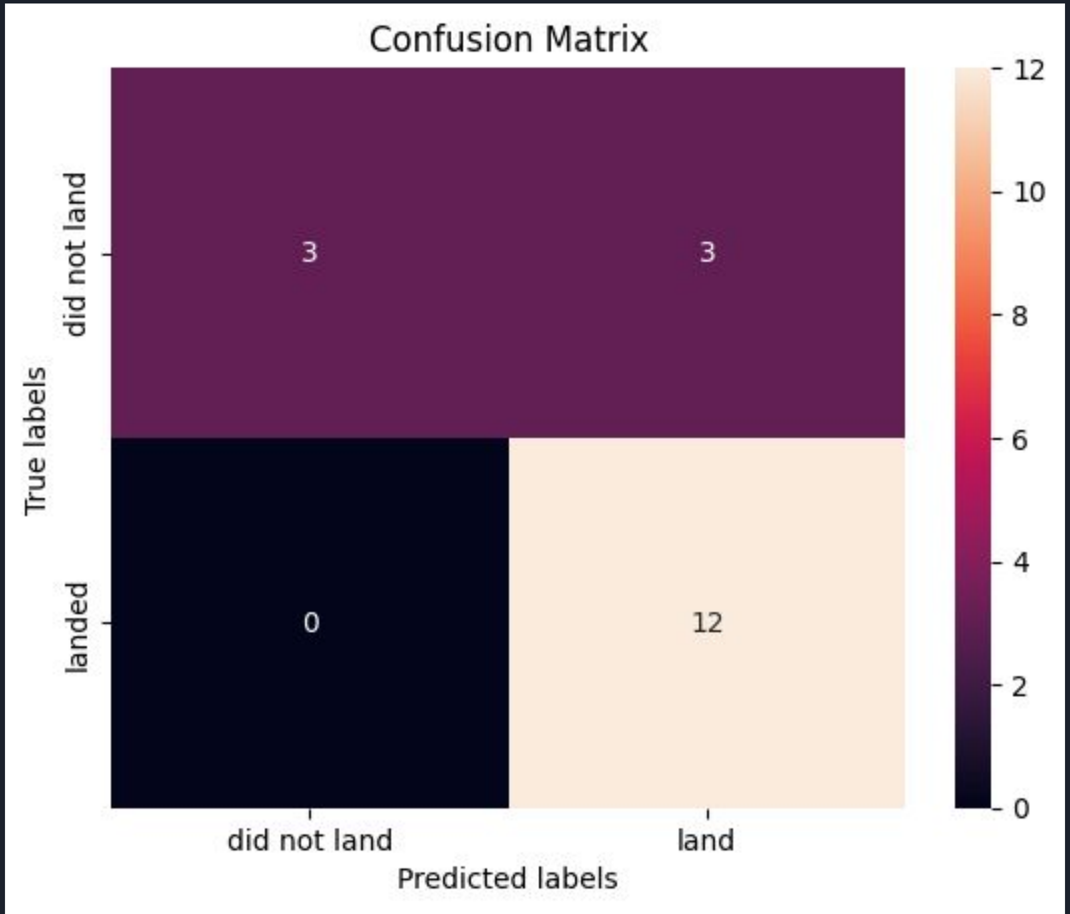
Classification Accuracy

- The *decision tree* model has the highest level of accuracy at 88.9%

- All the other models fall at the same rate of 83.3% of accuracy



Confusion Matrix



Confusion matrix for the decision tree model

- This confusion matrix distinguishes between the different classes present
- We must be cautious with false positives as the classifier could mark successful landings as unsuccessful and vice versa

CONCLUSIONS

- The more flights at a launch site, the higher the success rate at such launch site
- KSC LC-39A was the most successful launches site
- There is a direct relation between the pass of time and the success rate of missions
- Orbit types ES-L1, GEO, HEO, SSO were the most successful ones
- Low payload launches have a higher success rate than those with a heavy payload mass
- The Decision tree classifier is the best model for predicting landing outcomes in this dataset

Thank you!