IBM Developer
SKILLS NETWORK

# Winning Space Race with Data Science

Pavel Makarov
2023-06-25

# Outline

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

# Executive Summary

- Summary of methodologies
  - Collect the required data and do data wrangling
  - Data exploratory analysis and visualization using SQL, Pandas, and Matplotlib
  - Build Interactive Visual analytics tools  and Dashboards
- Summary of all results
  - Results of Exploratory Data Analysis
  - Dashboards, graphs, and tables
  - Results of applied ML

# Introduction

- Project background and context
  - Commercial space travel is becoming more accessible as companies like Virgin Galactic, Rocket Lab, Blue Origin, and SpaceX enter the market. Virgin Galactic offers suborbital spaceflights, while Rocket Lab specializes in small satellite launches. Blue Origin manufactures reusable sub-orbital and orbital rockets. SpaceX stands out as the most successful, having achieved numerous milestones such as sending spacecraft to the International Space Station, deploying the Starlink satellite internet constellation, and conducting manned missions to space. One key advantage of SpaceX is its cost-effective rocket launches, with Falcon 9 launches priced at $62 million compared to competitors' prices of $165 million or more, thanks to the reuse of the first stage. Determining the successful landing of the first stage is crucial in estimating launch costs. As data scientists working for Space Y, we gather information about SpaceX, create dashboards, and develop a machine-learning model to predict whether the first stage will be reusable.
- Problems you want to find answers
  - Use programming to determine the characteristics of successful and failed landings
  - Determine what we need to know about successful launches to apply it at SpaceY

Section 1

# Methodology

# Methodology

## Executive Summary

- Data collection methodology:
  - SpaceX REST API
  - Web Scrapping and Data mining
- Perform data wrangling
  - Cleaning up the data by dropping unnecessary columns
  - One Hot Encoding for models
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - How to build, tune, evaluate classification models

# Data Collection

- Datasets are collected from Rest SpaceX API and webscrapping Wikipedia
  - Required information from SpaceX REST API is the rocket, launchpads, payloads, cores.
  - Space X REST API URL is api.spacexdata.com/v4/


  - The information obtained by the webscrapping of Wikipedia is – flight number, launch site, payload, payload mass, orbit, customer, launch outcome, version of the booster, booster landing, date and time
  - URL is https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922

# Data Collection – SpaceX API

- https://github.com/PavelM90/Final-Project/blob/4194d5c800386fadd824bf230c6efc22b3140639/spacex-data-collection-api.ipynb

Make an API call and return JSON

```
def getBoosterVersion(data):
    for x in data['rocket']:
       if x:
        response = requests.get("https://api.spacexdata.com/v4/rockets/"+str(x)).json()
        BoosterVersion.append(response['name'])
```

Transform JSON into data frame using functions

```
## Fucntions
getCoreData
getBoosterVersion
getLaunchSite
getPayloadData
```

Create a dictionary and data frame

```
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}
```

Then, we need to create a Pandas data frame from the dictionary launch_dict.

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

Wrangle the data and export into .csv

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

# Data Collection - Scraping

- https://github.com/PavelM90/Final-Project/blob/ccc88deec53711ebae57932e12808e9f11a5a60d/jupyter-labs-webscraping.ipynb

Get response from HTML

```
static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"

response = requests.get(static_url)
```

Find tables

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

Create a dictionary and empty lists

```
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']=[]
```

Wrangle the data and export into .csv

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

# Data Wrangling

- https://github.com/PavelM90/Final-Project/blob/cf62f401b4b464a427e07a259e40c4b4524c93d2/-data_wrangling_jupyterlite.jupyterlite.ipynb

Identify the percentage of missing values

```
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
```

Calculate the number of lunches for each site

```
CCAFS SLC 40    55
KSC LC 39A      22
VAFB SLC 4E     13
Name: LaunchSite, dtype: int64
```

Calculate the number of outcomes per orbit

```
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
```

Calculate the number of occurrences for each orbit

```
GTO      27
ISS      21
VLEO     14
PO        9
LEO       7
SSO       5
MEO       3
ES-L1     1
HEO       1
SO        1
GEO       1
```

Create a “Class” parameters for outcomes

```
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

- https://github.com/PavelM90/Final-Project/blob/a253ad3cdbfc84c69ec0124f110eb08d10f761dd/Final_assignment_visualization.ipynb

Scatter plots

- Payload Mass vs Flight number
- Flight Number vs Lunch Site
- Payload Mass vs Launch Site
- Flight Number vs Orbit
- Payload Mass vs Orbit

Bar plots

- Mean Class by Orbit

Line plot

- Success / Failure by year

# EDA with SQL

- https://github.com/PavelM90/Final-Project/blob/adfe8823a63ed7256e37c9522052194e002f610f/Final_Project_SQL.ipynb
- Displayed the names of the unique launch sites in the space mission
- Displayed 5 records where launch sites begin with the string 'CCA’
- Displayed the total payload mass carried by boosters launched by NASA (CRS)
- Displayed average payload mass carried by booster version F9 v1.1
- Listed the date when the first successful landing outcome in the ground pad was achieved.
- Listed the names of the boosters which have success in drone ships and have payload mass greater than 4000 but less than 6000
- Listed the total number of successful and failure mission outcomes
- Listed the records which will display the month names, failure landing_outcomes in drone ship ,booster versions, launch_site for the months in year 2015.
- Ranked the count of successful landing_outcomes between the date 04-06-2010 and 20-03-2017 in descending order.

# Build an Interactive Map with Folium

- https://github.com/PavelM90/Final-Project/blob/131fe046a8b8a91b81d9eac64b2fc0c7277beb73/Fina_Project_Folium.ipynb

***The purpose of creating the map objects is to visualize the launch sites geographically and map successful outcomes on it***

The folium map object is a map centered on NASA Space Center in Houston, TX

- Red circle at NASA Johnson Space Center's coordinate with a label showing its name *(folium.Circle, folium.map.Marker)*
- Red circles at each launch site coordinates with a label showing the launch site name *(folium.Circle, folium.map.Marker, folium.features.DivIcon)*
- The grouping of points in a cluster to display multiple and different information for the same coordinates *(folium.plugins.MarkerCluster)*
- Markers to show successful and unsuccessful landings. Green for successful landing and Red for unsuccessful landing. *(folium.map.Marker, folium.Icon)*
- Markers to show the distance between the launch site to key locations (railway, highway, coastway, city) and plot a line between them. *(folium.map.Marker, folium.PolyLine, folium.features.DivIcon)*

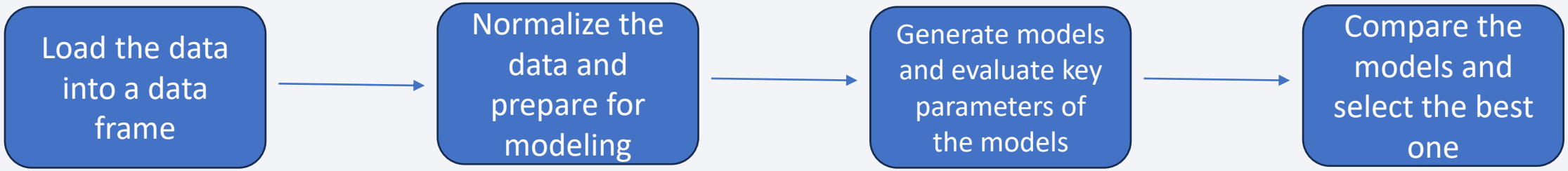
# Build a Dashboard with Plotly Dash

- https://github.com/PavelM90/Final-Project/blob/a8069cfd9f567b162c037961a2a2504bdec442be/Final_Project_Dashboard.py

***This interactive dashboard helps us better understand the relationships between different parameters of the successful/failed launches via visualization with pie charts and scatter plots***

- **Dropdown** (dash_core_components.Dropdown)
  - **Pie charts** (plotly.express.pie)
- **Range slider** was used to select a payload mass in the fixed range (dash_core_components.RangeSlider)
- **Scatter chart** *Successful Outcome vs Payload Mass* (plotly.express.scatter)

# Predictive Analysis (Classification)




- Load dataset
- Normalize data
- Split data into training and test sets

- Select the type of ML and use algorithms
- Set parameters
- Train the models and test on the test data set

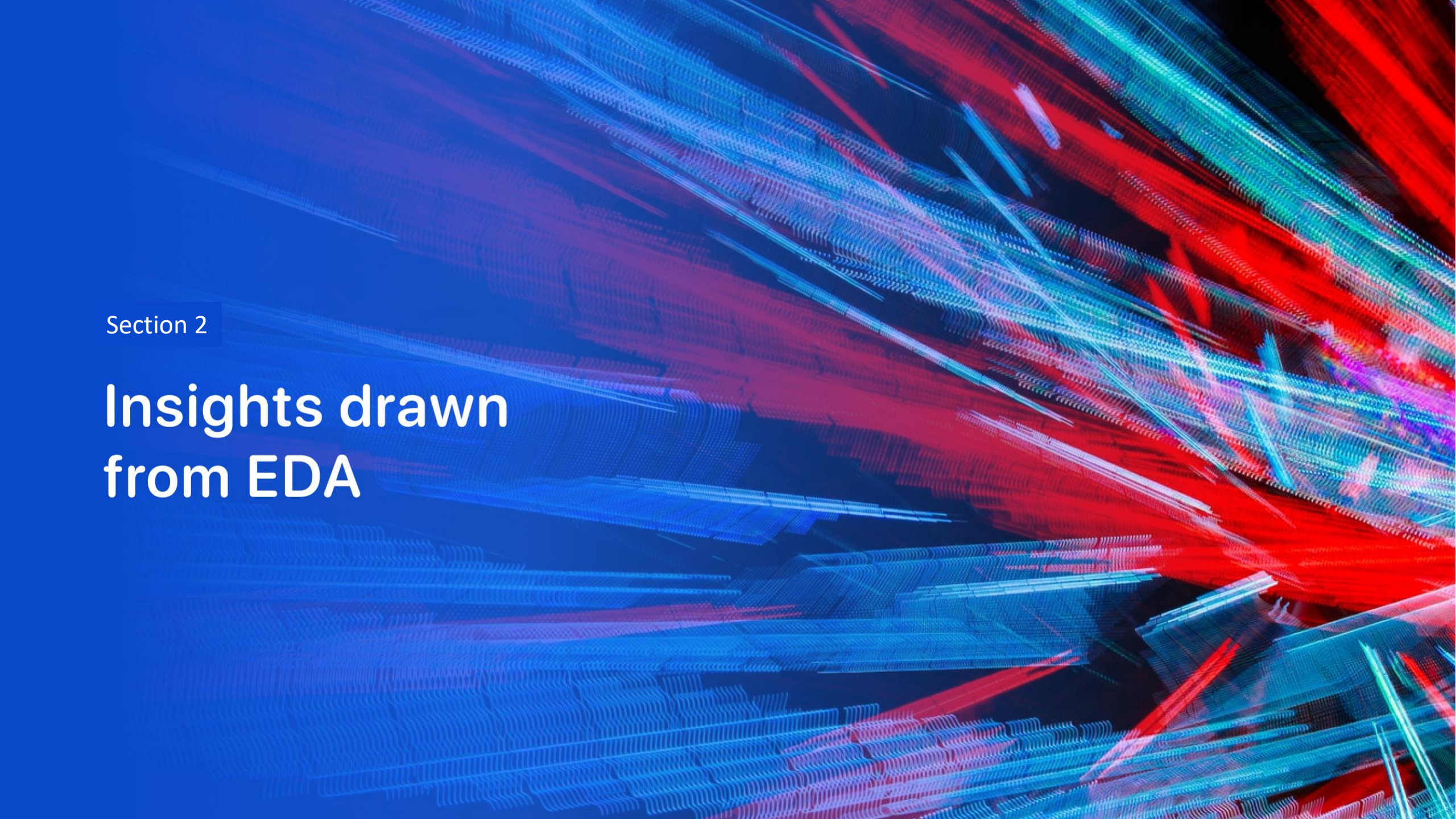
- Get the most optimal parameters for each model
- Generate confusion matrix for each

- Compare all types of algorithms' key parameters and select the best model

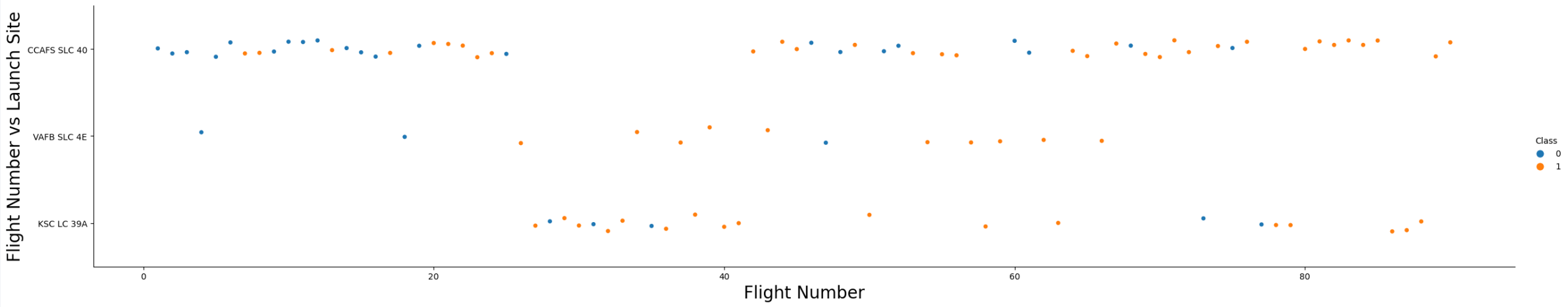
- https://github.com/PavelM90/Final-Project/blob/76140e51e928d37175c989e6e0d7fcbc8589fac8/Machine%20%20Learning%20PArt.ipynb

# Results

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

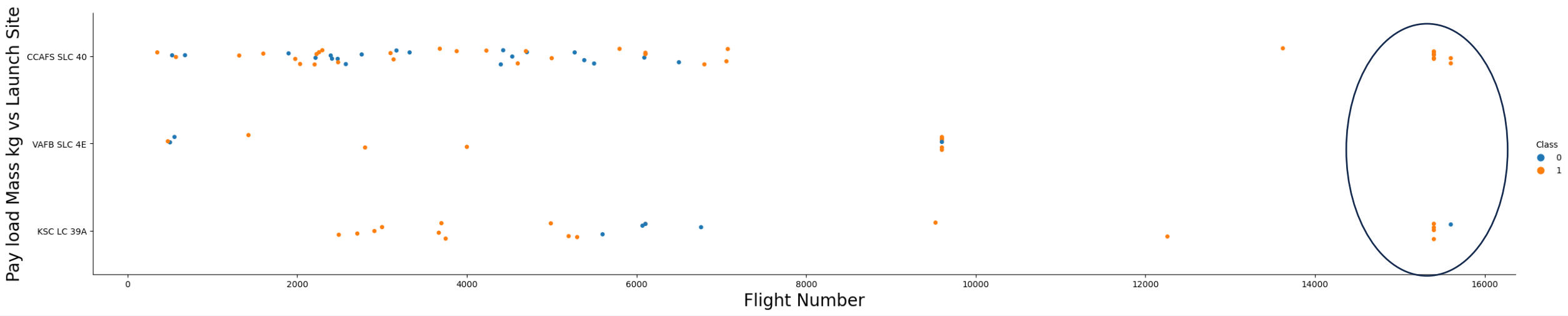

Section 2
Insights drawn from EDA

# Flight Number vs. Launch Site



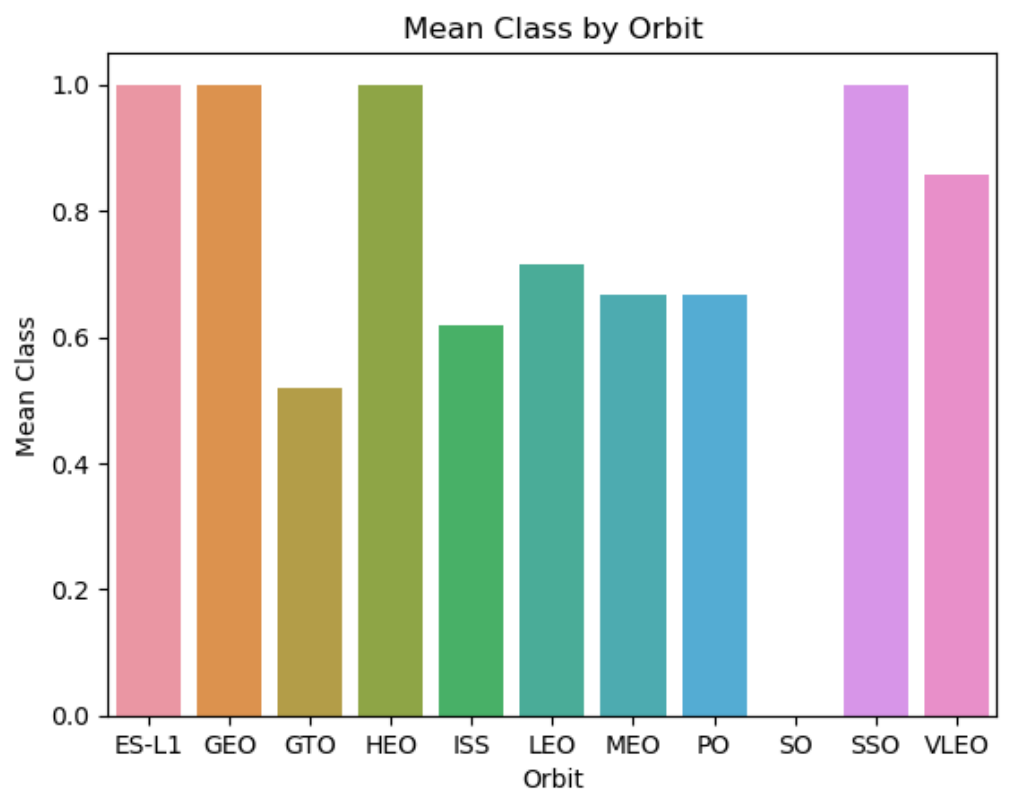

- We can visualize the number of flights from each line site with this scatter plot and the with an increasing number of flights the success rate is increasing

# Payload vs. Launch Site



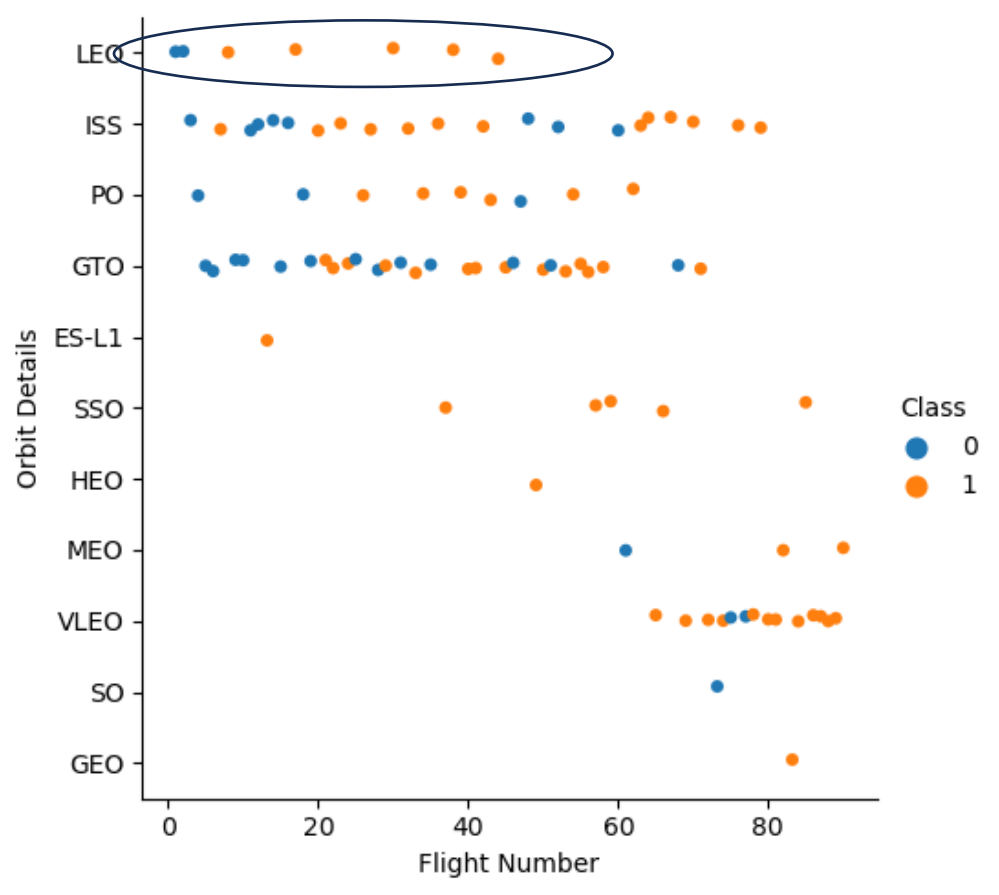

- CCAFS SLC 40 and KSC LC 39A have several successful launches with the heaviest payload masses

# Success Rate vs. Orbit Type



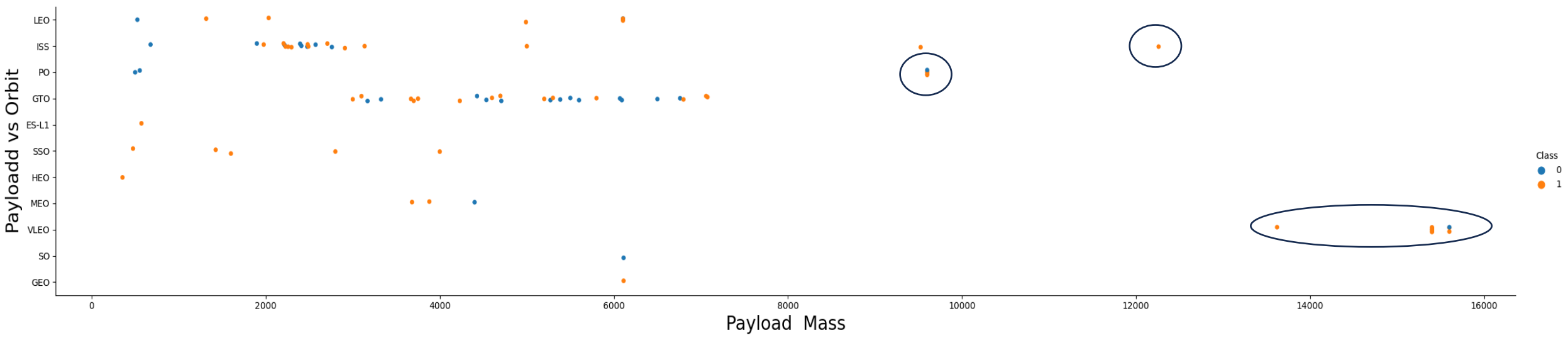

- There are four orbits with the highest success rate – ES-L1, GEO, HEO, SSO

# Flight Number vs. Orbit Type



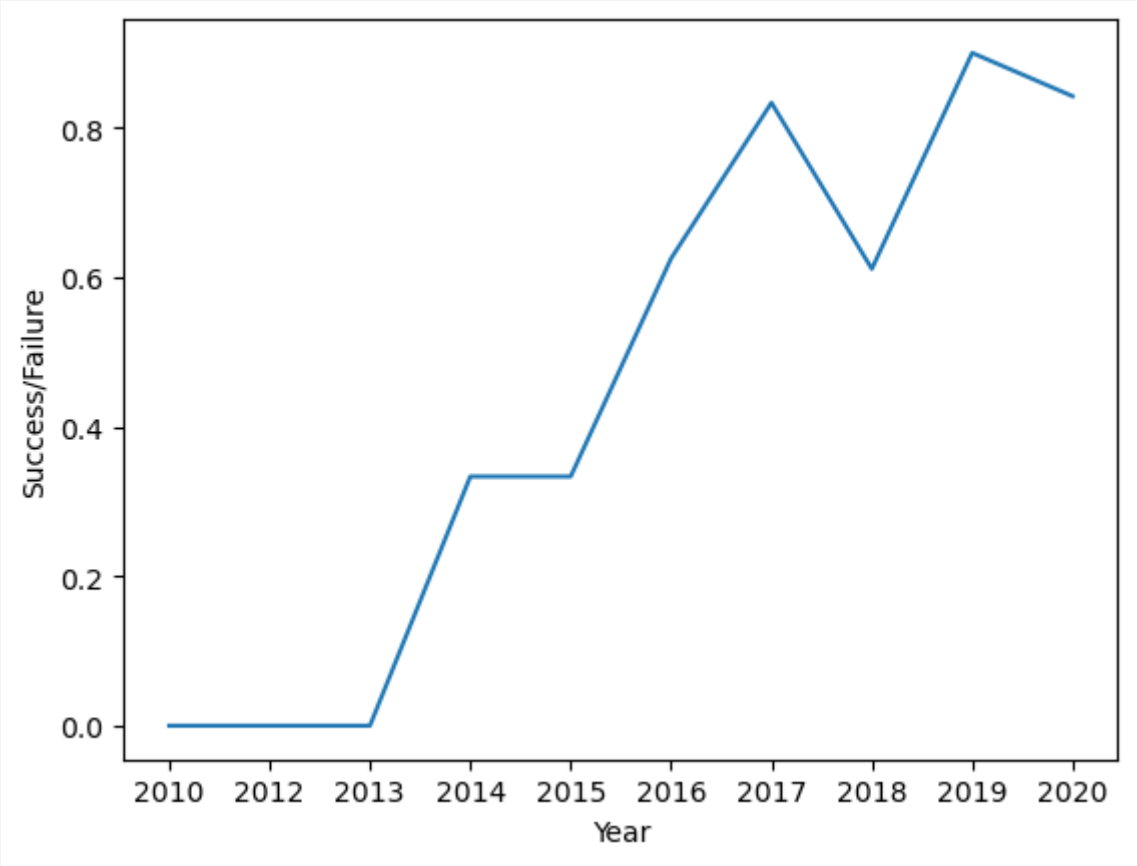

- From this chart, we can see that despite the high success rate ES-L1 orbit has one successful launch and LEO orbit success is increasing with an increasing number of flights.

# Payload vs. Orbit Type




- With heavy payloads, the successful landing or positive landing rate is more for Polar, LEO and ISS

# Launch Success Yearly Trend




- Since 2013, the company has had a continuous increase in successful launches

# All Launch Site Names

- We found all distinct launch sites to have a number of all possible lunch sites

```
cur.execute("SELECT DISTINCT launch_site FROM SPACEXTBL")
launch_sites = cur.fetchall()
for site in launch_sites:
    print(site[0])
```

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

# Launch Site Names Begin with 'CCA'

```
launch_site_pattern = 'CCA%'
cur.execute("SELECT * FROM SPACEXTBL WHERE launch_site LIKE ? LIMIT 5", (launch_site_pattern,))
records = cur.fetchall()
records
```

```
[('06/04/2010',
  '18:45:00',
  'F9 v1.0  B0003',
  'CCAFS LC-40',
  'Dragon Spacecraft Qualification Unit',
  0.0,
  'LEO',
  'SpaceX',
  'Success',
  'Failure (parachute)'),
 ('12/08/2010',
  '15:43:00',
  'F9 v1.0  B0004',
  'CCAFS LC-40',
  'Dragon demo flight C1, two CubeSats, barrel of Brouere cheese',
  0.0,
  'LEO (ISS)',
  'NASA (COTS) NRO',
  'Success',
  'Failure (parachute)'),
 ('22/05/2012',
  '7:44:00',
  'F9 v1.0  B0005',
  'CCAFS LC-40',
  'Dragon demo flight C2',
  525.0,
  'LEO (ISS)',
  'NASA (COTS)',
  'Success',
  'No attempt'),
 ('10/08/2012',
  '0:35:00',
  'F9 v1.0  B0006',
  'CCAFS LC-40',
  'SpaceX CRS-1',
  500.0,
  'LEO (ISS)',
  'NASA (CRS)',
  'Success',
  'No attempt'),
 ('03/01/2013',
  '15:10:00',
  'F9 v1.0  B0007',
  'CCAFS LC-40',
  'SpaceX CRS-2',
  677.0,
  'LEO (ISS)',
  'NASA (CRS)',
  'Success',
  'No attempt')]
```

This query was used to demonstrate the results of launches where launch site starts with “CCA”

# Total Payload Mass

```
from prettytable import PrettyTable
cur.execute("SELECT * FROM SPACEXTBL")

# Fetch the column names from the cursor description
column_names = [description[0] for description in cur.description]

# Fetch all the rows returned by the query
rows = cur.fetchall()

# Create a PrettyTable object
table = PrettyTable(column_names)

# Populate the table with data
for row in rows:
    table.add_row(row)

# Set the table alignment
table.align = "l"

# Display the table
print(table)

cur.execute("""
    SELECT SUM(PAYLOAD_MASS__KG_) AS total_payload_mass
    FROM SPACEXTBL
    WHERE Customer = 'NASA (CRS)'
""")
NASA_total_payload_mass = cur.fetchall()
NASA_total_payload_mass
```

We sued this query to find the total payload mass  carried by boosters launched by NASA

Result- **[(45596.0,)]**

# Average Payload Mass by F9 v1.1

```
cur.execute("""
    SELECT AVG(PAYLOAD_MASS__KG_) AS average_payload_mass
    FROM SPACEXTBL
    WHERE Booster_Version =  "F9 v1.1"
""")
average_from_F9 = cur.fetchall()
average_from_F9
```

```
[(2928.4,)]
```

- We calculated the average mass F9 v 1.1 booster carried – 2928.4 kg

# First Successful Ground Landing Date

```
cur.execute("""
    SELECT MIN(Date) AS first_date_success
    FROM SPACEXTBL
    WHERE Landing_Outcome = "Success (ground pad)"
""")
first_success = cur.fetchall()
first_success
```

```
[('01/08/2018',)]
```

- Then we found when the first successful landing was in the ground pad. It was on January 8th, 2018

# Successful Drone Ship Landing with Payload between 4000 and 6000

```
cur.execute("""
    SELECT DISTINCT Booster_Version
    FROM SPACEXTBL
    WHERE Landing_Outcome = "Success (drone ship)"
    AND PAYLOAD_MASS__KG_ > 4000
    AND PAYLOAD_MASS__KG_ < 6000
""")
Booster_name = cur.fetchall()
Booster_name
```

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

- We found all types of boosters which had successful landings on drone ship with payload mass between 4000 and 6000 kg

# Total Number of Successful and Failure Mission Outcomes

```
cur.execute("""
    SELECT Mission_Outcome, COUNT(*) AS total_landings
    FROM SPACEXTBL
    WHERE Mission_Outcome IN ('Success', 'Failure (in flight)')
    GROUP BY Mission_Outcome
""")
counts = cur.fetchall()
counts
### Task 8



##### List the   names of the booster_versions which have carried the maximum payload mass. Use a subquery
```

```
[('Failure (in flight)', 1), ('Success', 98)]
```

- We used SQL query to count total successful and failed mission outcomes

# Boosters Carried Maximum Payload

```
cur.execute("""
    SELECT Booster_Version
    FROM SPACEXTBL
    WHERE PAYLOAD_MASS__KG_ = (
        SELECT MAX(PAYLOAD_MASS__KG_)
        FROM SPACEXTBL
    )
""")
booster_versions = cur.fetchall()

# Process the fetched booster versions
for booster_version in booster_versions:
    print(booster_version[0])
```

```
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
```

- Then we found what type of boosters had the launches with maximum payload, and it was F9 B5

# 2015 Launch Records

```
cur.execute("""
    SELECT
        CASE substr(Date, 4, 2)
            WHEN '01' THEN 'January'
            WHEN '02' THEN 'February'
            WHEN '03' THEN 'March'
            WHEN '04' THEN 'April'
            WHEN '05' THEN 'May'
            WHEN '06' THEN 'June'
            WHEN '07' THEN 'July'
            WHEN '08' THEN 'August'
            WHEN '09' THEN 'September'
            WHEN '10' THEN 'October'
            WHEN '11' THEN 'November'
            WHEN '12' THEN 'December'
            ELSE 'Invalid Month'
        END AS month_name,
        Mission_Outcome,
        Booster_Version,
        Launch_Site
    FROM SPACEXTBL
    WHERE substr(Date, 7, 4) = '2015'
        AND Landing_Outcome  = 'Failure (drone ship)'
""")

# Fetch all the records returned by the query
records = cur.fetchall()

# Process the fetched records
for record in records:
    print("Month:", record[0])
    print("Landing Outcome:", record[1])
    print("Booster Version:", record[2])
    print("Launch Site:", record[3])
    print()  # Adding a newline for readability
```

- Then we used query search criteria to find failed landings in drone ships, booster versions, and launch sites for the months in 2015

| MONTH | Booster_Version | Launch_Site |
|---|---|---|
| 01 | F9 v1.1 B1012 | CCAFS LC-40 |
| 04 | F9 v1.1 B1015 | CCAFS LC-40 |

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

```
successful_landings = 'Sucess%'
cur.execute("""
    SELECT landing_outcome, COUNT(*) AS count
    FROM SPACEXTBL
    WHERE Date BETWEEN '2010-06-04' AND '2017-03-20'
        AND landing_outcome = ?
    GROUP BY landing_outcome
    ORDER BY count DESC,(successful_landings
""")

# Fetch all the ranked results
ranked_results = cursor.fetchall()

# Process the fetched results
for rank, result in enumerate(ranked_results, start=1):
    landing_outcome = result[0]
    count = result[1]
    print("Rank:", rank)
    print("Landing Outcome:", landing_outcome)
    print("Count:", count)
    print()  # Adding a newline for readability

# Close the cursor and connection
cursor.close()
conn.close()
```

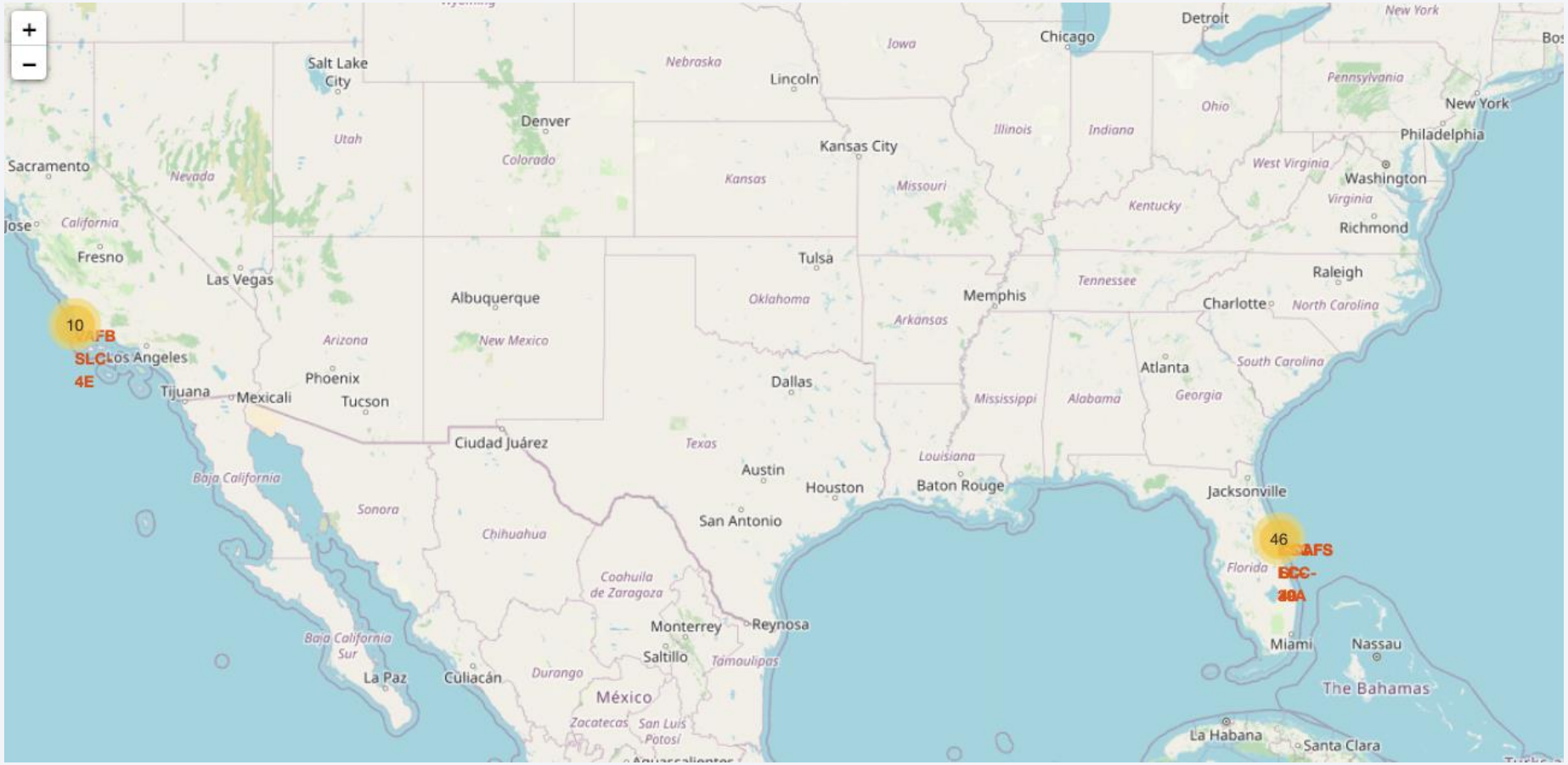
- In the specific time range between 2010-06-04 and 2017-03-20 we found 20 successful landings where 8 was on the drone ship and 6 on the ground pad

| Landing _Outcome | COUNT("LANDING _OUTCOME") |
|---|---|
| Success | 20 |
| Success (drone ship) | 8 |
| Success (ground pad) | 6 |

Section 3

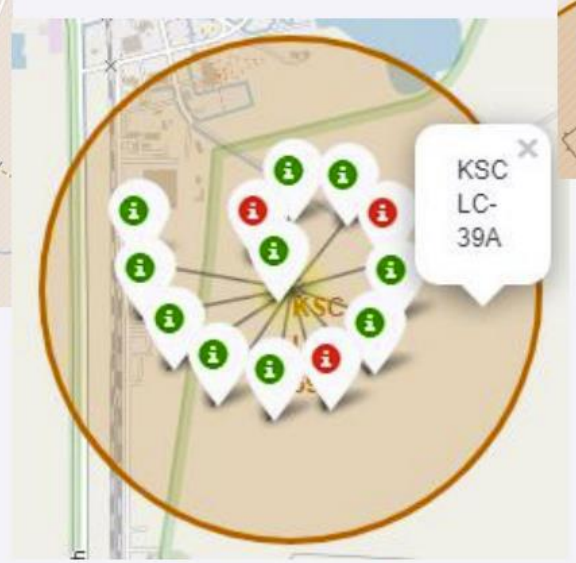
# Launch Sites Proximities Analysis

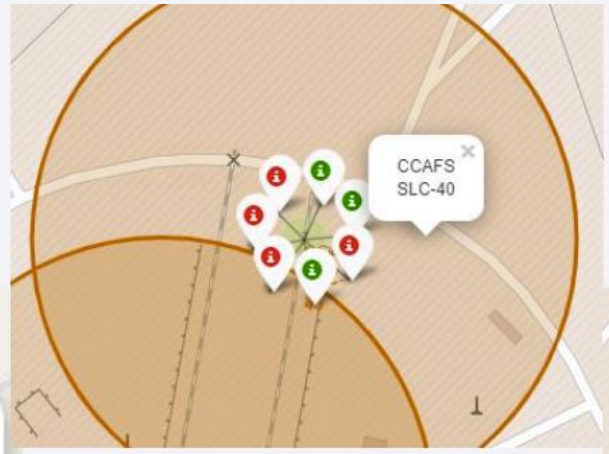
# Location of launch sites on the map

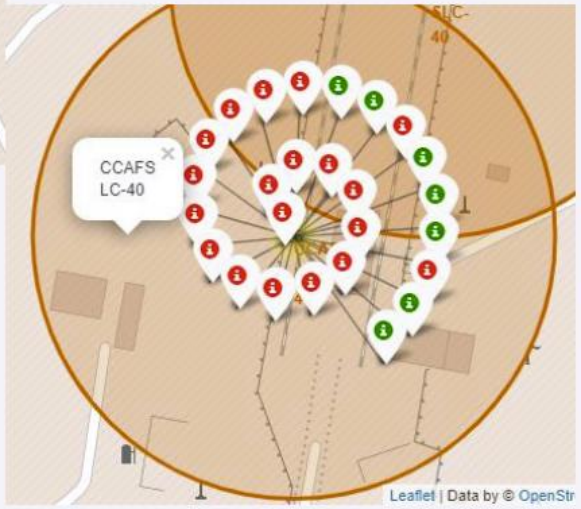



We can see where geographically the launch site is located and launch records for each location

# Location of the launch sites with success/failure labels

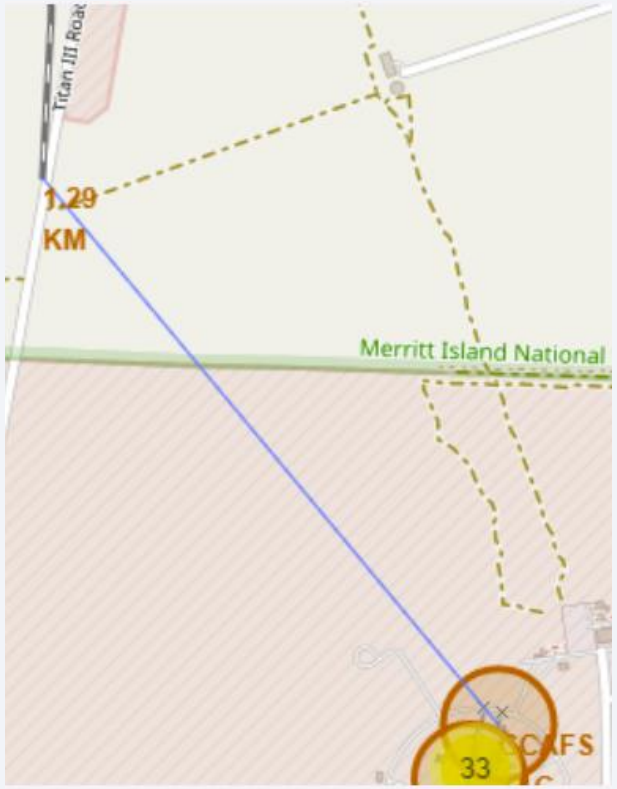



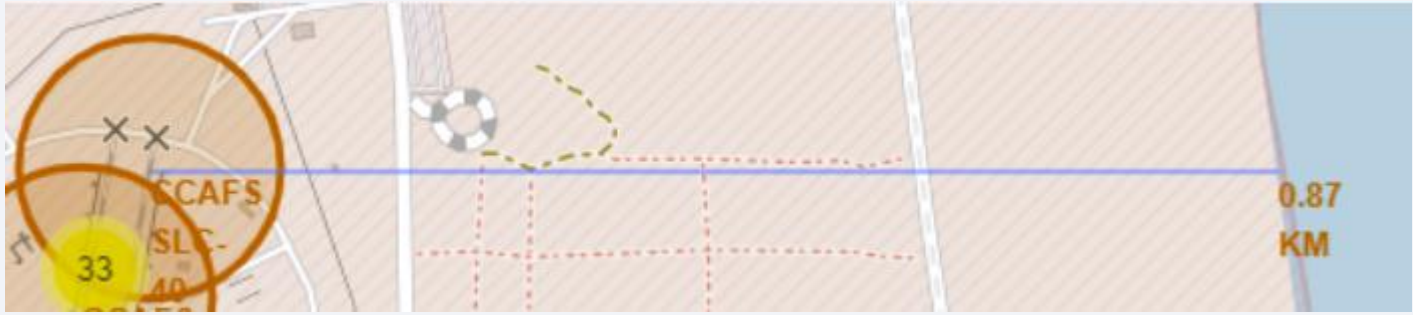



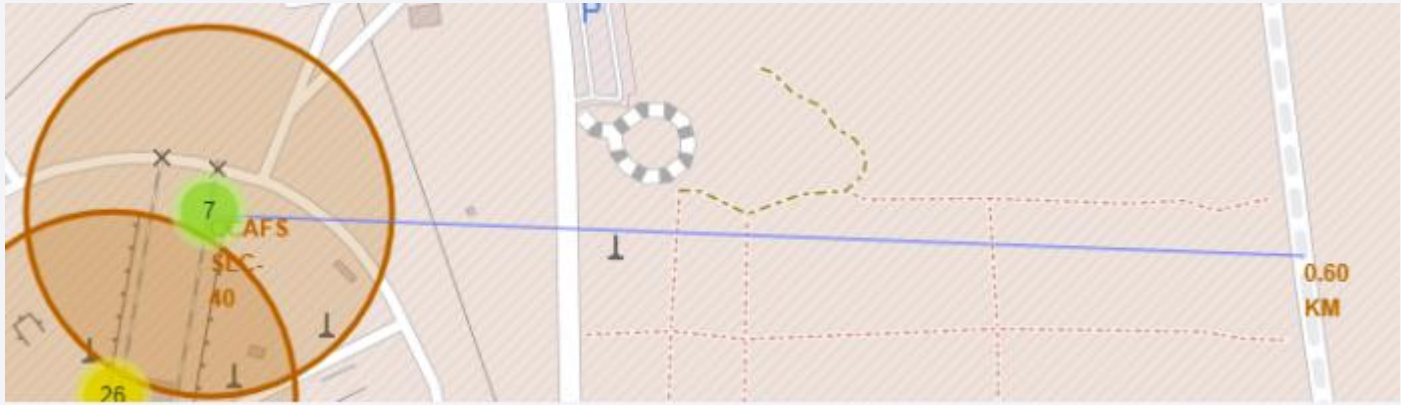




Green – Success
Red -Failure

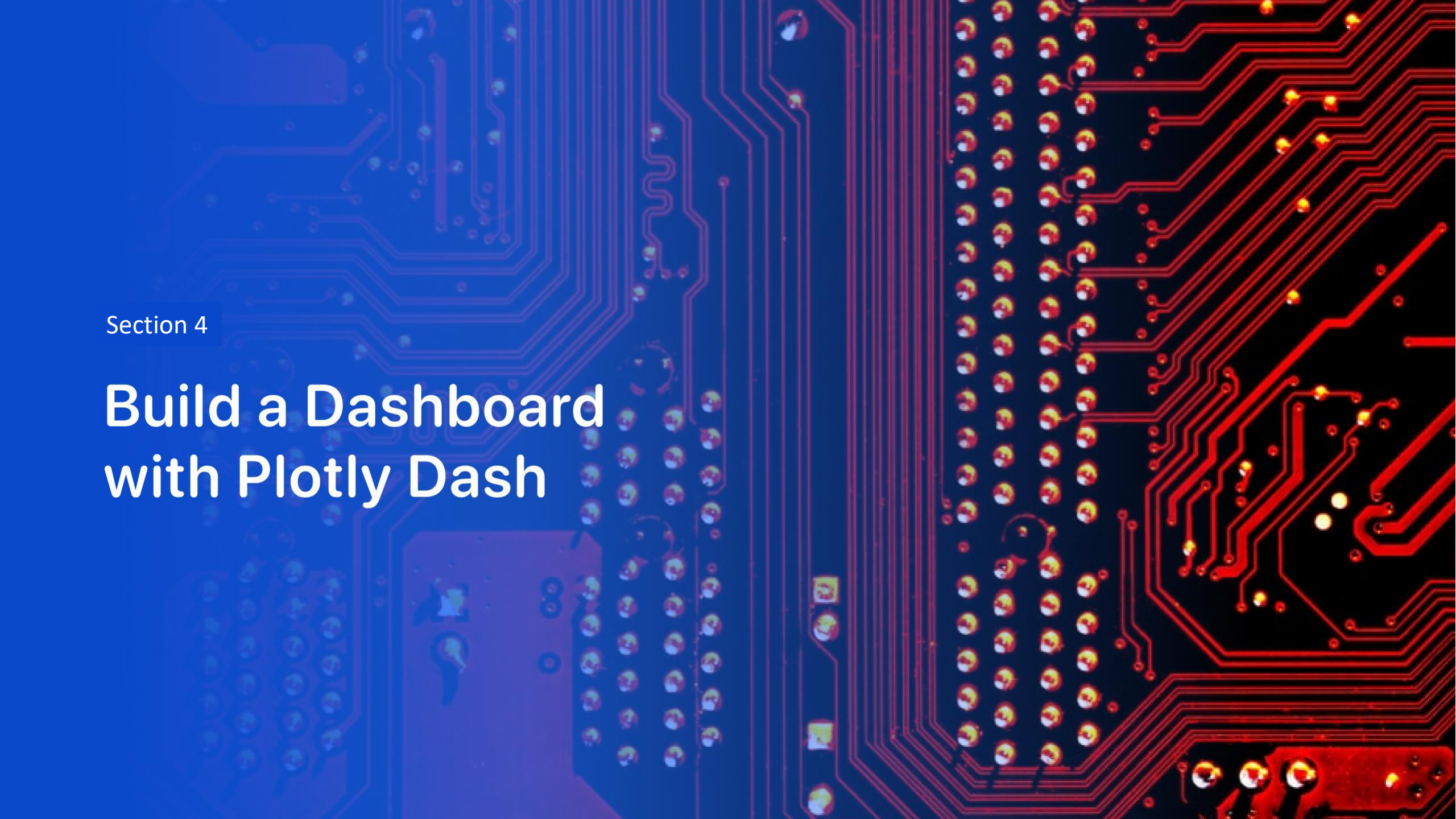
# CCASF-SLC-40 proximity



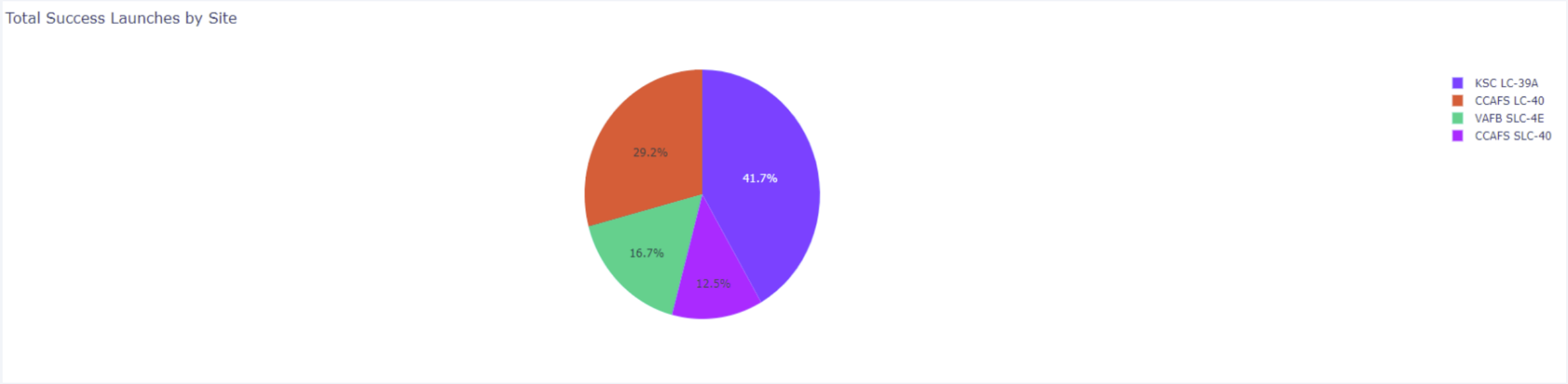




- CCASF SLC-40 proximities to coastline, railways, and highways.
- From the map this launch site is relatively close to all these objects

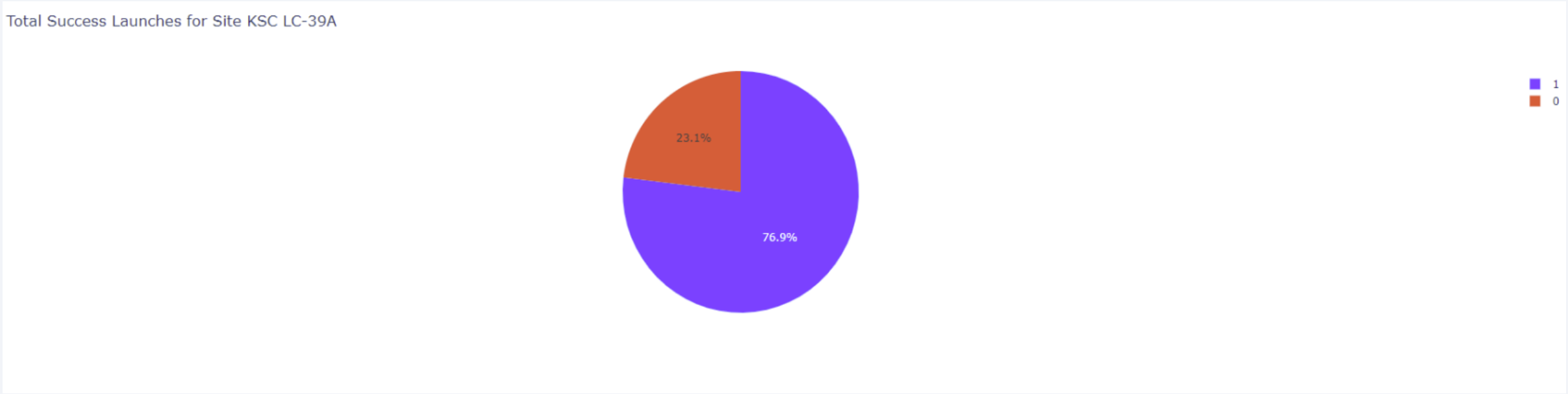

Section 4
Build a Dashboard with Plotly Dash

# Pie chart for total successful launches by launch site



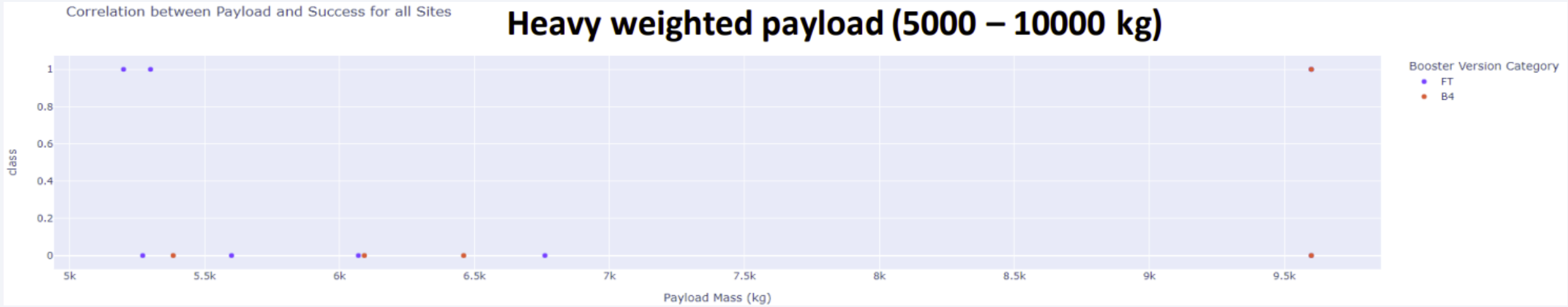

Among all launch site, KSC LC-39A has the highest relative percentage of success

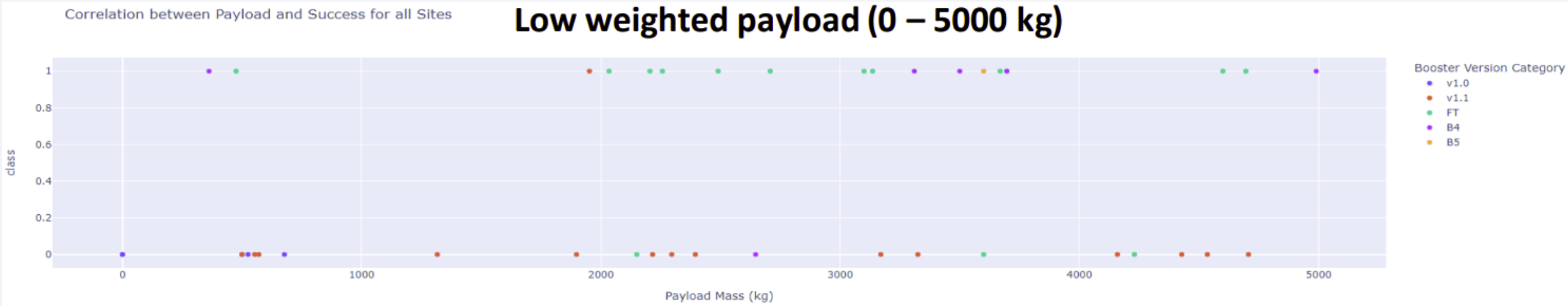
# Total successful launches from KSC LC-39A




We can see that 77% of launches from this site were successful

# Correlation between success rate and payload mass





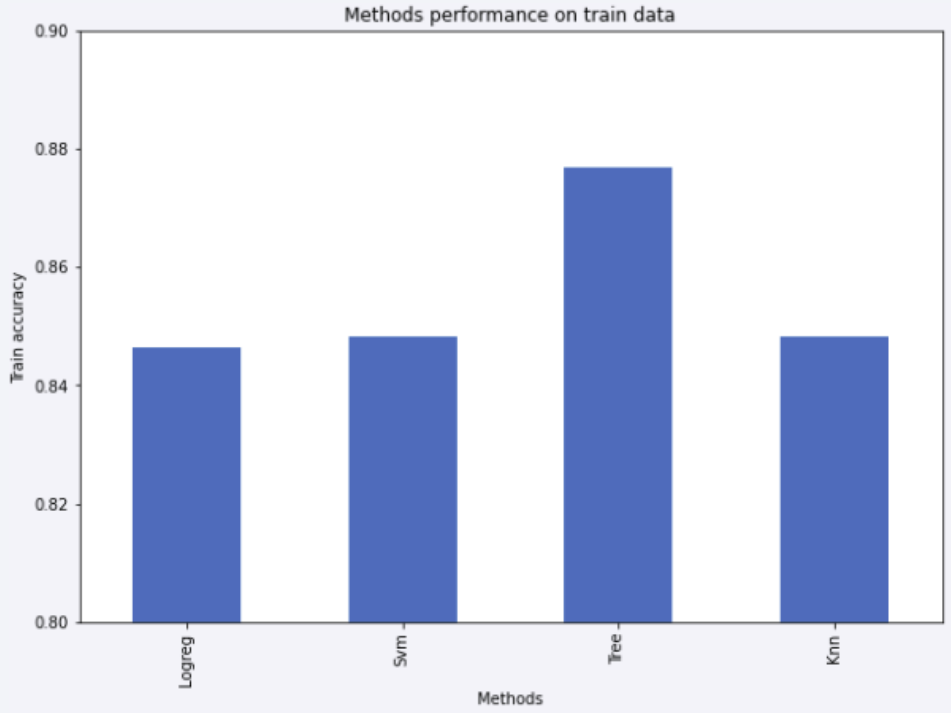

Low-weight payloads with FT, B4, and V1.0 have the highest success rate

Section 5

# Predictive Analysis (Classification)

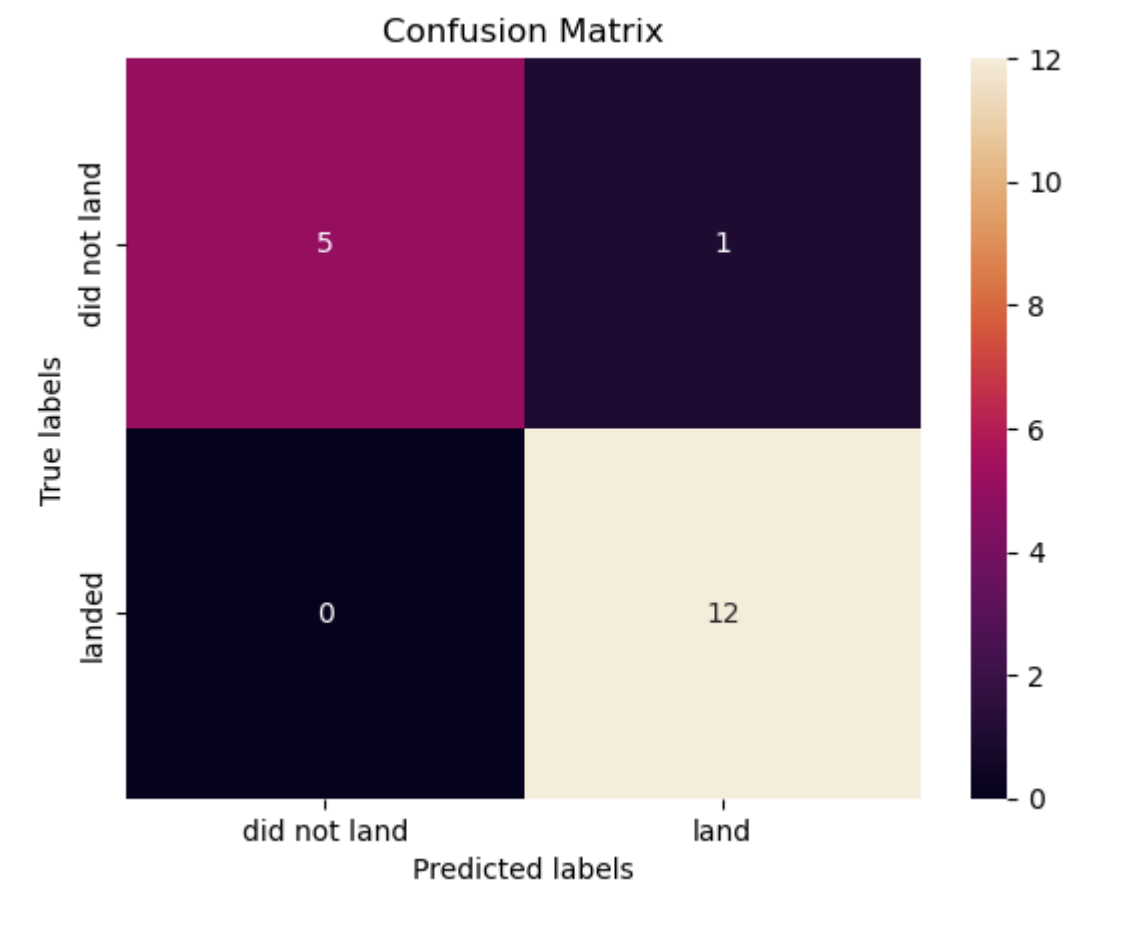
# Classification Accuracy

- The decision tree model has the best accuracy among all tested models




The calculated accuracy was 0.875

# Confusion Matrix




The decision tree model has the highest number of true labels (12) and predicted labels for landed and true and predicted (5) did not land

# Conclusions

- We used our study on the example of analysis of historical data of Space X company.
- We used visualization tools, SQL queries, and data wrangling to find that the most successful launch sites were GEO, HEO, SSO, ES-L1. The sites are close to railways, highways, the city, and the coast.
- We visualized the results for different booster and payload masses. The results showed that average light-weight payloads with booster versions FT, B4, and V1.0 have higher success rates.
- When we applied Machine Learning algorithms to build predictive models, we determined that decision tree model has the highest accuracy

# Appendix

- Link to GitHub repository
- https://github.com/PavelM90/Final-Project.git

# Thank you!