Winning Space Race with Data Science

Ziwen Zhong
2023.05.06

Outline

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

Executive Summary

Summary of methodologies:

- Data collection
- Data wrangling
- Exploratory data analysis
- Interactive visual analytics
- Predictive analysis

Summary of all results

- EDA with visualization results
- EDA with SQL results
- Interactive map with Folium results
- Plotly Dash dashboard results
- Predictive analysis (classification) results

Introduction

- 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.
- We aim to determine if the first stage will land, and then we can determine the cost of a launch.
- This information can be used if an alternate company wants to bid against SpaceX for a rocket launch.

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - Get a request to the SpaceX API
  - Web Scrapping
- Perform data wrangling
  - Use the mean value for PayloadMass to impute the missing value
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - Use sklearn to build, use GridSearchCV to tune, assess accuracy to evaluate classification models

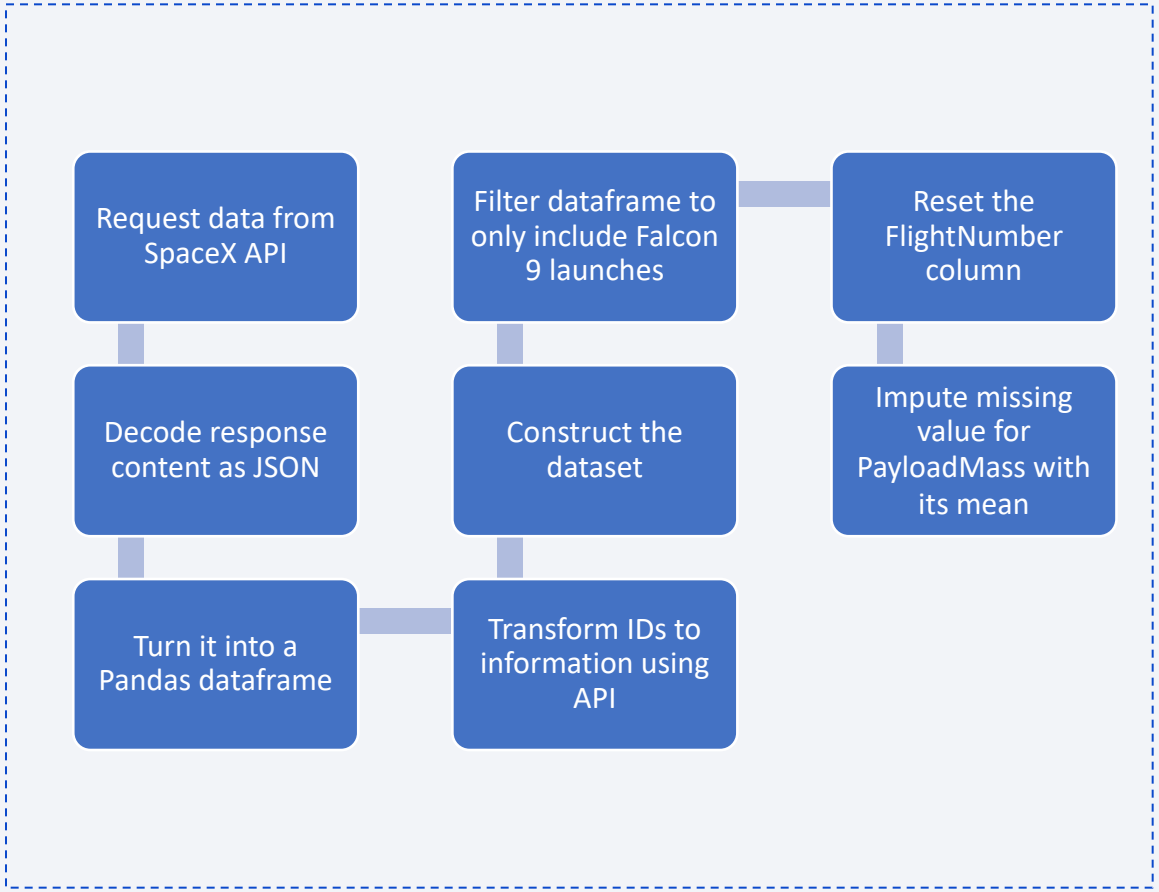
Data Collection

- Describe how data sets were collected.
- You need to present your data collection process use key phrases and flowcharts

Data Collection – SpaceX API

- The progress of data collection with SpaceX REST calls is presented as right using key phrases and flowcharts

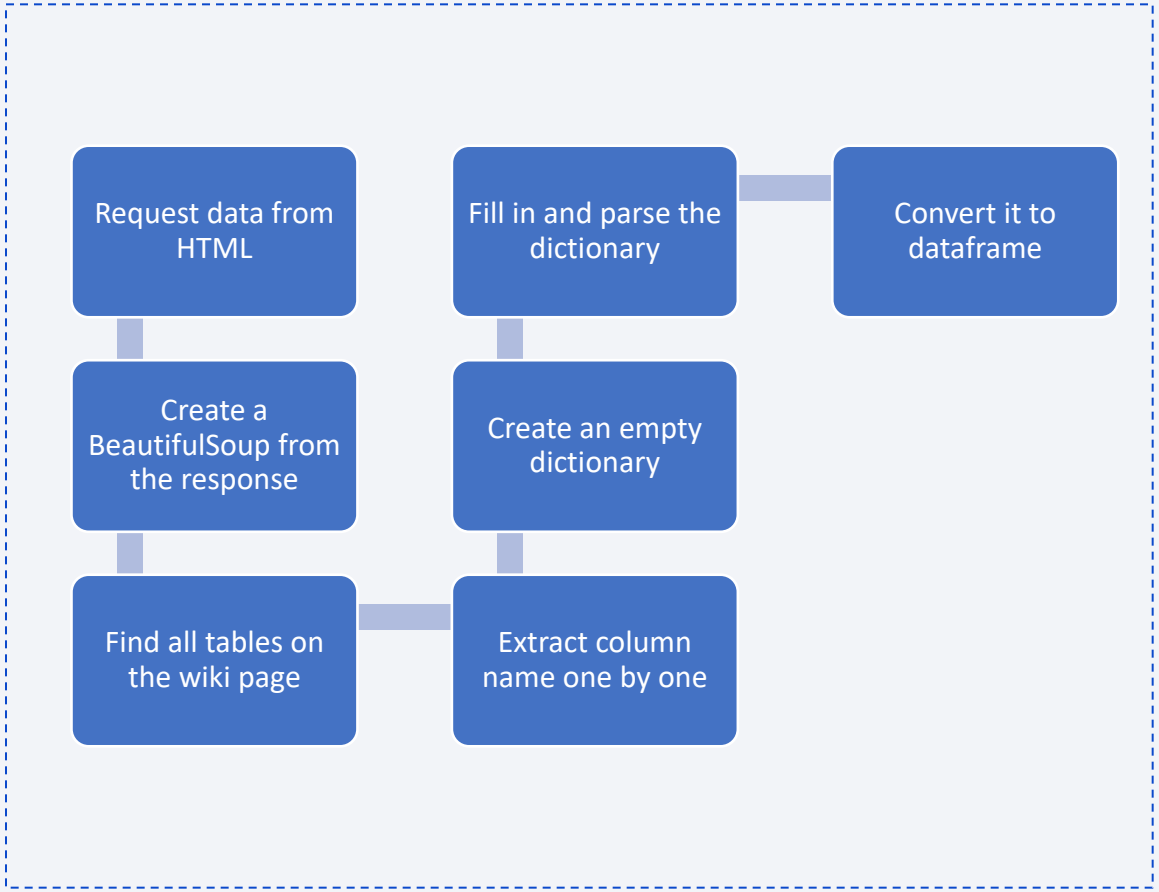
- The GitHub URL of the completed SpaceX API calls notebook with the completed code cell and outcome cell is: https://github.com/zzw-math/Winning-Space-Race/blob/main/jupyter-labs-spacex-data-collection-api.ipynb



Data Collection - Scraping

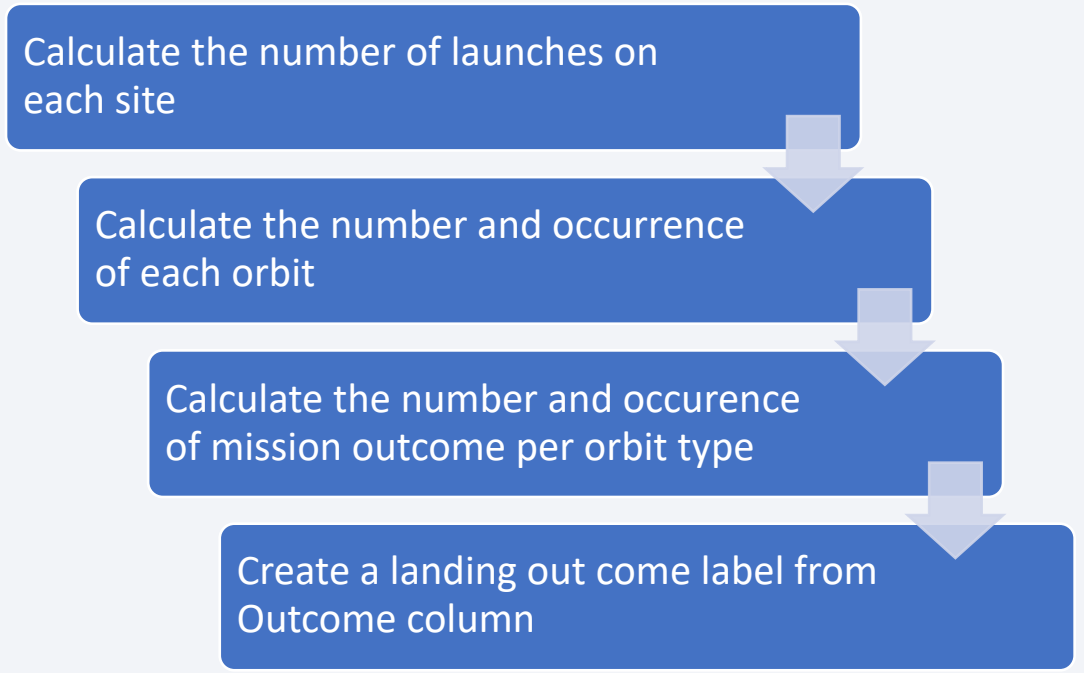
- The progress of the web scraping is presented as right using key phrases and flowcharts

- The GitHub URL of the completed web scraping notebook is: https://github.com/zzw-math/Winning-Space-Race/blob/main/jupyter-labs-webscraping.ipynb



Data Wrangling

- We use value_counts in Pandas to calculate 3 things and convert outcome column to one-hot
- The data wrangling process is presented as right using key phrases and flowcharts
- The GitHub URL of completed data wrangling-related notebooks is: https://github.com/zzw-math/Winning-Space-Race/blob/main/labs-jupyter-spacex-data_wrangling_jupyterlite.jupyterlite.ipynb



EDA with Data Visualization

- Scatter plots were produced to visualise the relationship between flight number, payload and launch site, orbit type
- A bar plot was induced to visualise the success rates for different orbit type
- A line plot was conducted to show the success rate change by year
- The GitHub URL of the completed EDA with data visualization notebook is https://github.com/zzw-math/Winning-Space-Race/blob/main/jupyter-labs-eda-dataviz.ipynb.jupyterlite.ipynb

EDA with SQL

- Use distinct to get the name of the unique launch sites in the space mission
- Use where and like to get 5 records whose launch sites begin with 'CCA'
- Use sum, where and like to obtain the total payload mass launched by NASA
- Use avg, where and like to obtain the average payload mass carried by booster version F9 v1.1
- Use min to get the date of the first successful landing outcome in the ground pad was achieved
- Use where and like to obtain the names of the boosters which drone ship successfully and have payload mass between 4000 and 6000
- Use count and group by to obtain the total number of successful and failed mission outcomes
- Use subquery to obtain the names of booster_versions which have carried the maximum payload mass
- Use substr, where, and like to obtain the records which fail to drone ship in 2015
- Use count, where and group by to obtain the ranking of the count of successful landing between 2010-06-04 and 2017-03-20
- The GitHub URL of the completed EDA with SQL notebook is https://github.com/zzw-math/Winning-Space-Race/blob/main/jupyter-labs-eda-sql-coursera_sqllite.ipynb

Build an Interactive Map with Folium

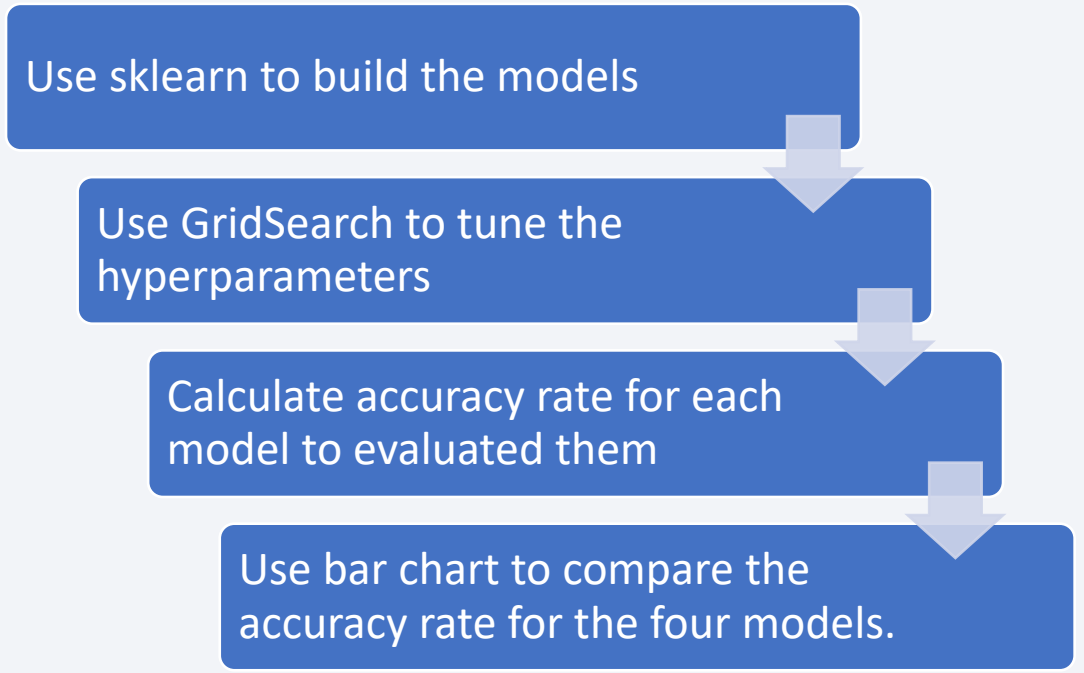
- Mark all launch sites on a map using markers, green for success, red for failed
- Use mark clusters (circle) to show many markers at the same location
- Calculate the distances between a launch site to its proximities and add lines in the map to represented by them
- The GitHub URL of the completed interactive map with Folium map is: https://github.com/zzw-math/Winning-Space-Race/blob/main/lab_jupyter_launch_site_location.jupyterlite.ipynb

Build a Dashboard with Plotly Dash

- Use pie charts to show each launch site's proportion of successful launches among total successful launches and the proportion of successful launches among all launches for each launch site.
- Use the scatter plots to show the correlation between outcome and payload mass
- The GitHub URL of the completed Plotly Dash lab is: https://github.com/zzw-math/Winning-Space-Race/blob/main/spacex_dash_app.py

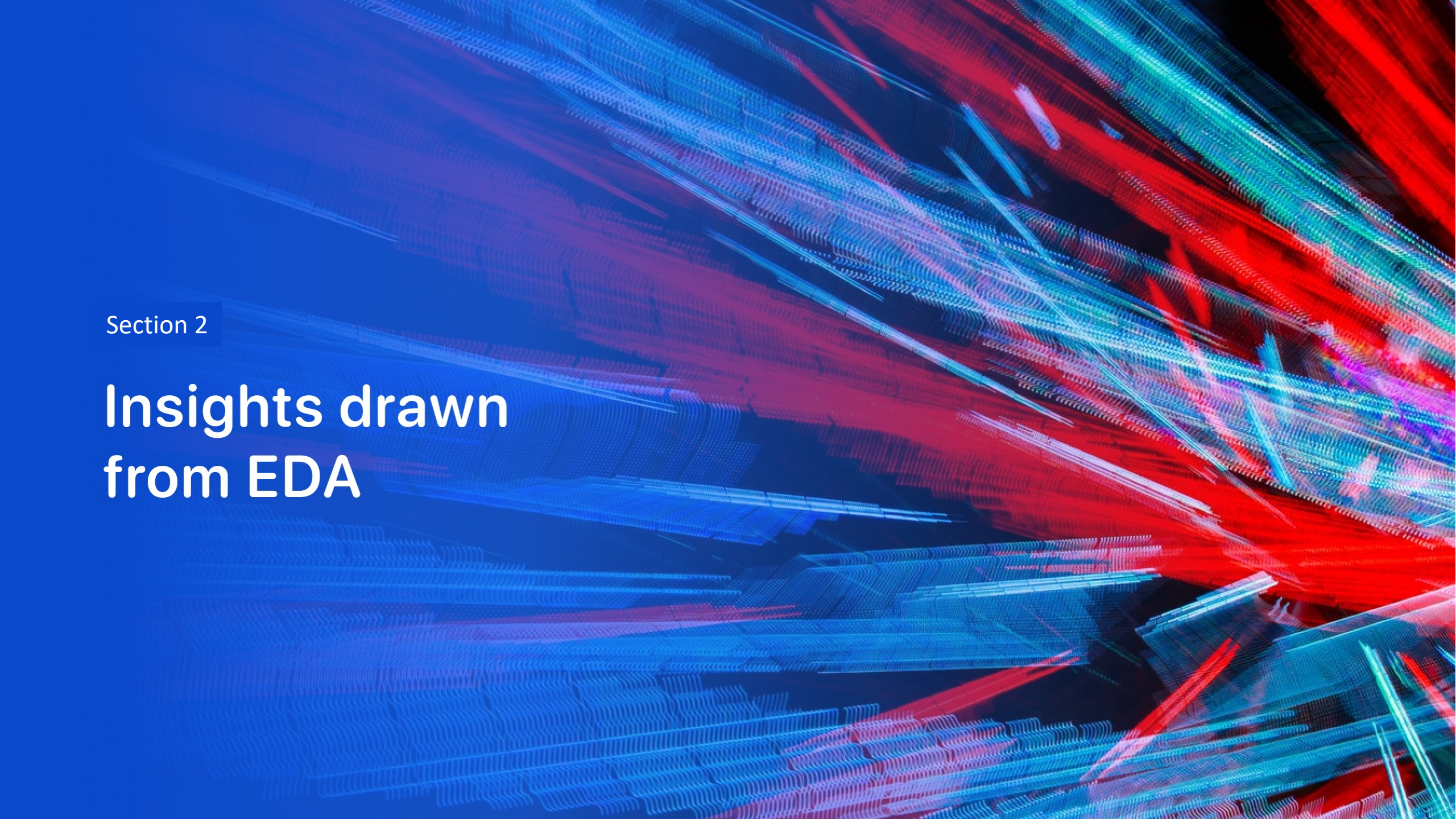
Predictive Analysis (Classification)

- The model development process is presented as right using key phrases and a flowchart
- The GitHub URL of the completed predictive analysis lab is: https://github.com/zzw-math/Winning-Space-Race/blob/main/SpaceX_Machine_Learning_Prediction_Part_5.jupyterlite.ipynb



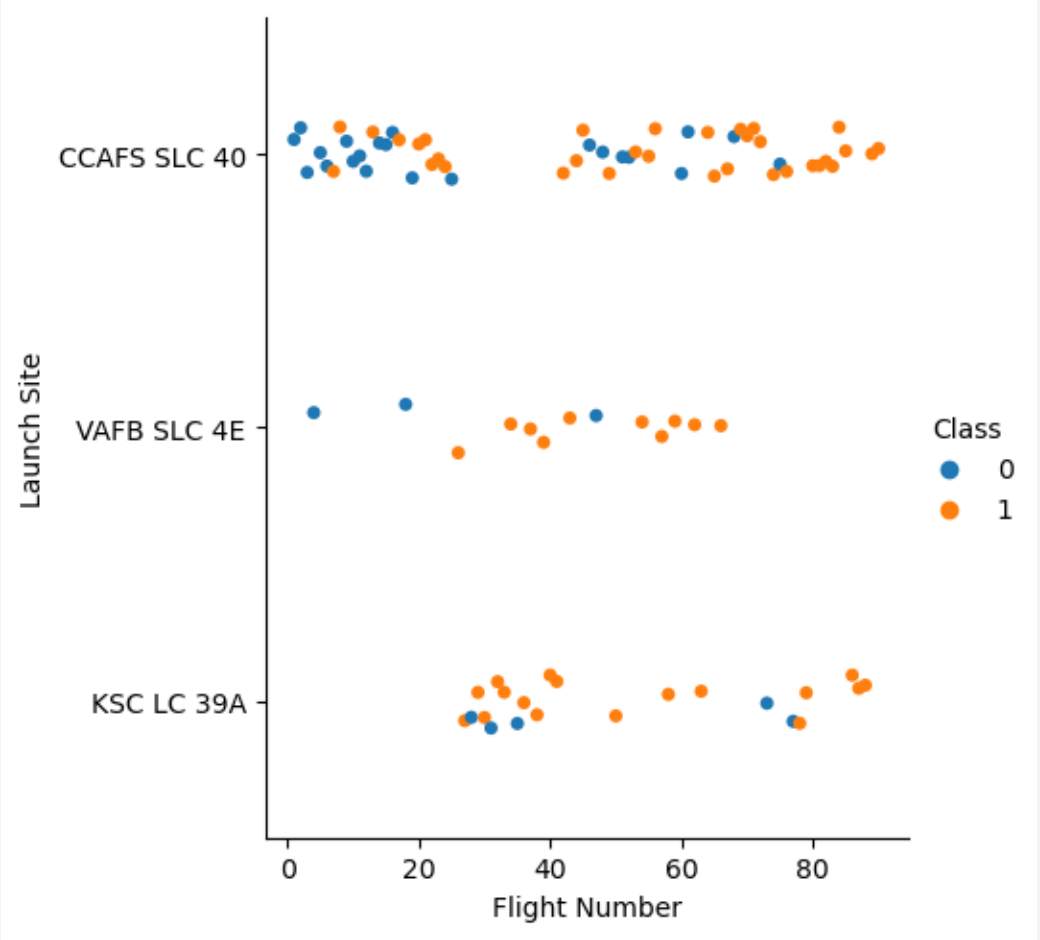
Results

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


Section 2
Insights drawn from EDA

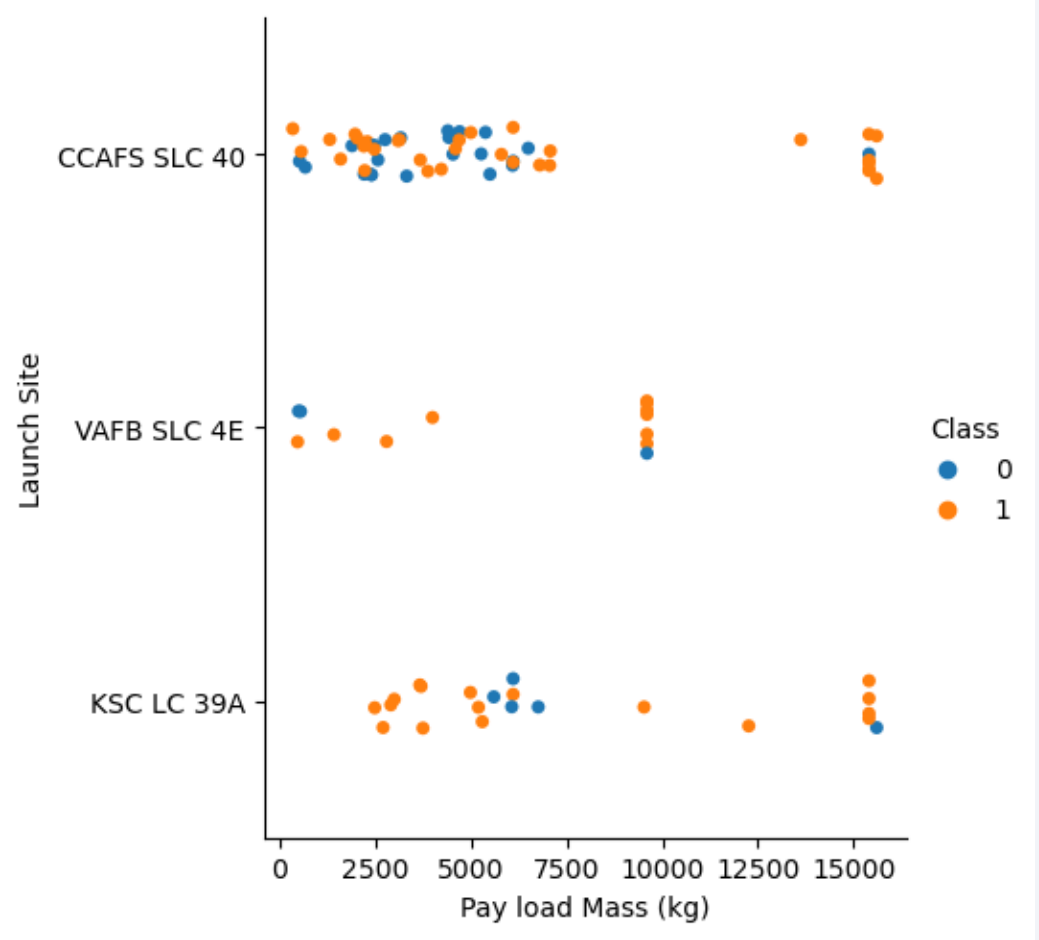
Flight Number vs. Launch Site

- The scatter plot of Flight Number vs. Launch Site is as right
- Flights with flight numbers less than 25 were mostly launched at CCAFS SLC 40
- Flights with higher flight numbers have a higher successful launch rate
- CCAFS SLC 40 has the lowest successful launch rate



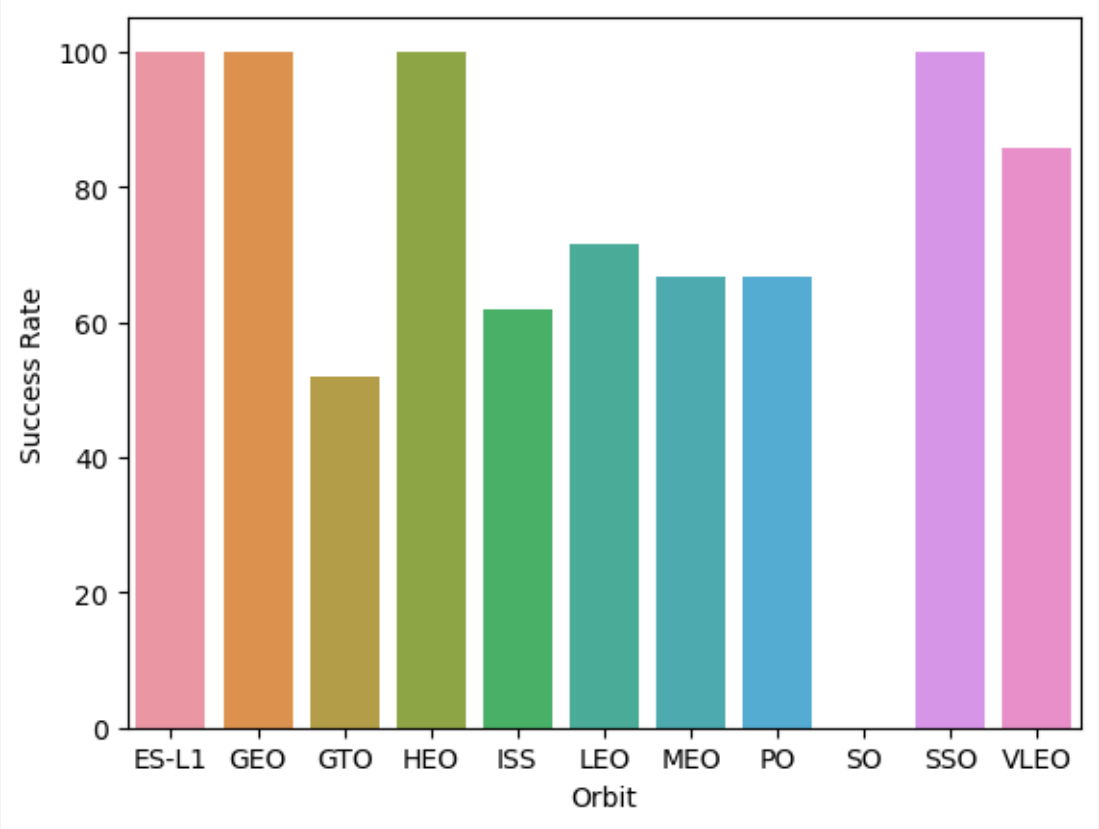
Payload vs. Launch Site

- The scatter plot of Payload vs. Launch Site is as right
- CCAFS SLC 40 has the lowest successful launch rate
- Most flights' payload masses are below 7500
- Flights with payload mass higher than 7500 has a higher successful launch rate



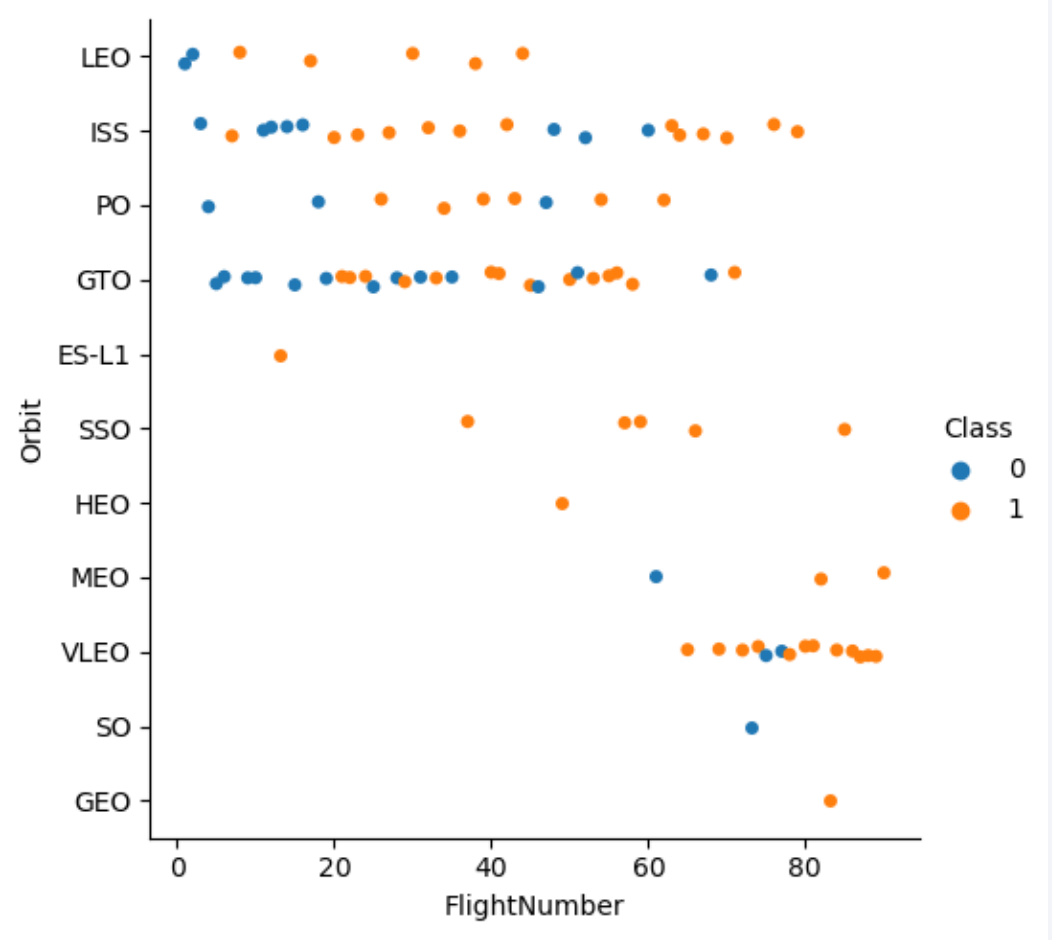
Success Rate vs. Orbit Type

- The bar chart for the success rate of each orbit type is as right
- ES-L1, EGO, HEO, SSO, VLEO has the highest success rate higher than 80%
- The success rates for GTO, ISS, LEO, MEO, PO are around 50%
- The success rate for SO is 0%



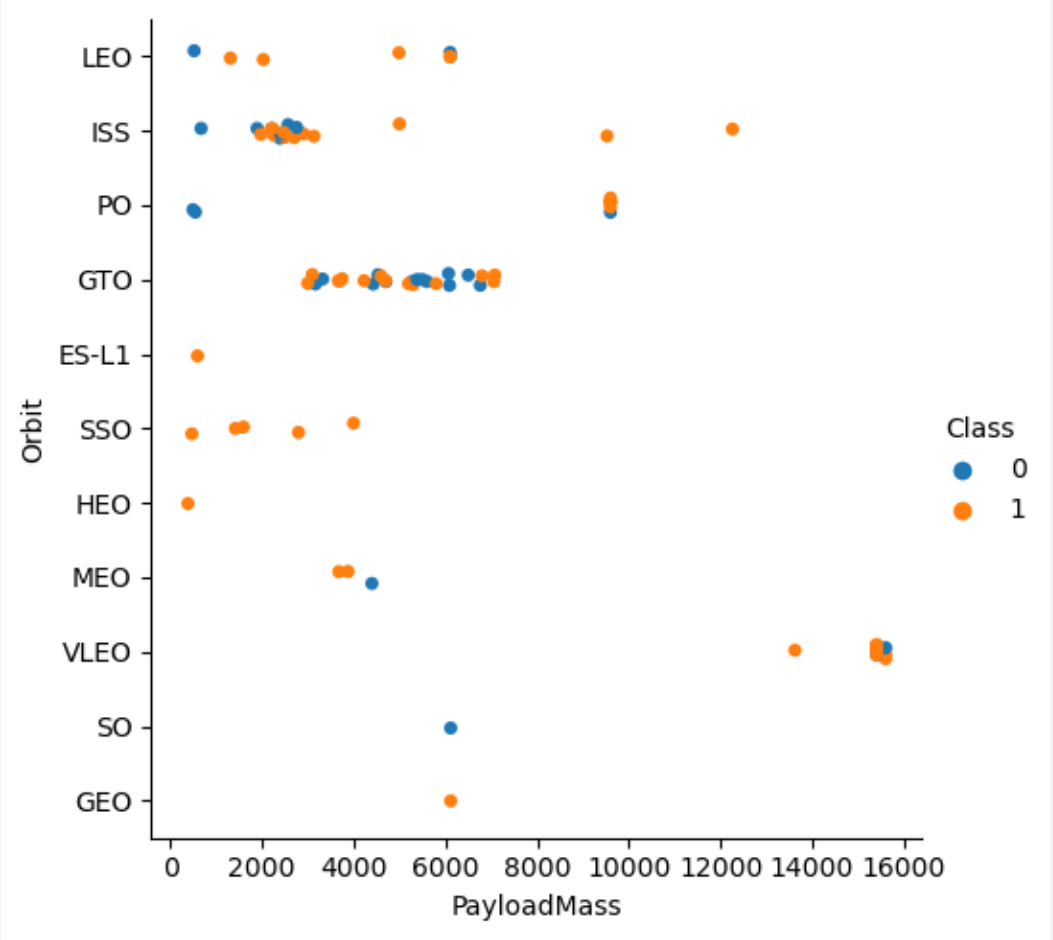
Flight Number vs. Orbit Type

- The scatter point of Flight number vs. Orbit type is as right
- The successful launch rate increase as flight number increase
- ES-L1, HEO, SO, GEO only have one sample, so the success rate for them is not impressive



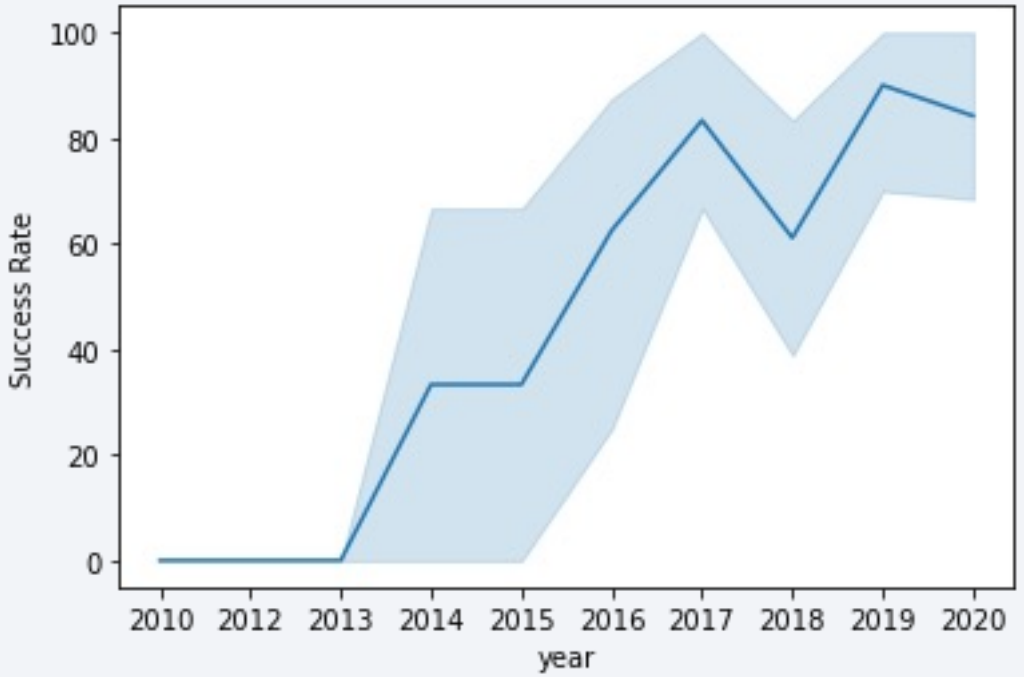
Payload vs. Orbit Type

- The scatter point of payload vs. orbit type is as right
- The payload mass of VLEO is mostly higher than 15000
- For LEO, ISS, and PO flights with higher payload mass have a higher success rate



Launch Success Yearly Trend

- The line chart of the yearly average success rate is as right
- Before 2013, all launched failed
- After 2013, the success rate increased, with small dips in 2018 and 2020



All Launch Site Names

- Find the names of the unique launch sites
- The word DISTINCT returns only unique values from the Launch_Site column of the SPACEXTBL table.

Task 1

Display the names of the unique launch sites in the space mission

```
%sql select DISTINCT LAUNCH_SITE from SPACEXTBL;
```

[11] ✓ 0.0s Python

```
* 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'

- Find 5 records where launch sites begin with `CCA`
- LIMIT 5 fetches only 5 records, and the LIKE keyword is used with the wild card ‘CCA%’ to retrieve string values beginning with ‘CCA’.

Task 2

Display 5 records where launch sites begin with the string 'CCA'

```
%sql select * from SPACEXTBL where launch_site like 'CCA%' limit 5;
```

[12] ✓ 0.0s Python

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

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing__Outcome
2010-04-06	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-08-12	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2012-05-22	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-08-10	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-01-03	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Total Payload Mass

- Calculate the total payload carried by boosters from NASA
- The SUM keyword is used to calculate the total of the LAUNCH column, and the SUM keyword (and the associated condition) filters the results to only boosters from NASA (CRS).

Task 3

Display the total payload mass carried by boosters launched by NASA (CRS)

```
%sql select sum(payload_mass__kg_) as sum from SPACEXTBL where customer like 'NASA (CRS)';
```

[13] ✓ 0.0s Python

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

sum
45596

Average Payload Mass by F9 v1.1

- Calculate the average payload mass carried by booster version F9 v1.1
- The AVG keyword is used to calculate the average of the PAYLOAD_MASS__KG_ column, and the WHERE keyword (and the associated condition) filters the results to only the F9 v1.1 booster version.

Task 4

Display average payload mass carried by booster version F9 v1.1

```
%sql select avg(payload_mass__kg_) as Average from SPACEXTBL where booster_version like 'F9 v1.1%';
```

[78] Python

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

Average
2534.6666666666665

First Successful Ground Landing Date

- Find the dates of the first successful landing outcome on ground pad
- The MIN keyword is used to calculate the minimum of the DATE column, i.e. the first date, and the WHERE keyword (and the associated condition) filters the results to only the successful ground pad landings.

Task 5

List the date when the first succesful landing outcome in ground pad was acheived.

Hint:Use min function

```
%sql select min(date) as Date from SPACEXTBL where mission_outcome like 'Success';
```

[10] ✓ 0.0s Python

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

Date
2010-04-06

Successful Drone Ship Landing with Payload between 4000 and 6000

- List the names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000
- The WHERE keyword is used to filter the results to include only those that satisfy both conditions in the brackets (as the AND keyword is also used). The BETWEEN keyword allows for 4000 < x < 6000 values to be selected.

Task 6

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

```
%sql select booster_version from SPACEXTBL \
    where (landing__outcome = 'Success (drone ship)') & (payload_mass__kg_ > 4000) & (payload_mass__kg_ < 6000)
```

[28] ✓ 0.0s Python

```
 * 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

- Calculate the total number of successful and failure mission outcomes
- The COUNT keyword is used to calculate the total number of mission outcomes, and the GROUPBY keyword is also used to group these results by the type of mission outcome.

Task 7

List the total number of successful and failure mission outcomes

```
%sql SELECT mission_outcome, count(*) as Count FROM SPACEXTBL GROUP by mission_outcome;
```

[20] ✓ 0.0s Python

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

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

Boosters Carried Maximum Payload

- List the names of the booster which have carried the maximum payload mass
- A subquery is used here. The SELECT statement within the brackets finds the maximum payload, and this value is used in the WHERE condition. The DISTINCT keyword is then used to retrieve only distinct /unique booster

Task 8

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

```
%sql select booster_version from SPACEXTBL where payload_mass__kg_=(select max(payload_mass__kg_) from SPACEXTBL)
```

[82] Python

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

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

- List the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015
- The WHERE keyword is used to filter the results for only failed landing outcomes, AND only for the year of 2015.

Task 9

List the records which will display the month names, failure landing_outcomes in drone ship ,booster versions, launch_site for the months in year 2015.

Note: SQLLite does not support monthnames. So you need to use substr(Date, 4, 2) as month to get the months and substr(Date,7,4)='2015' for year.

```
%sql select substr(Date, 6, 2) as Month, booster_version, launch_site from SPACEXTBL \
    where (DATE like '2015%') & (landing__outcome like 'Failure (drone ship)');
```

[118] Python

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

Month	Booster_Version	Launch_Site
10	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

- Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order
- The WHERE keyword is used with the BETWEEN keyword to filter the results to dates only within those specified. The results are then grouped and ordered, using the keywords GROUP BY and ORDER BY, respectively, where DESC is used to specify the descending order.

Task 10

Rank the count of successful landing_outcomes between the date 04-06-2010 and 20-03-2017 in descending order.

```
%sql select landing__outcome, count(*) as count from SPACEXTBL \
    where Date >= '2010-06-04' AND Date <= '2017-03-20' GROUP by landing__outcome ORDER BY count Desc
```

[21] ✓ 0.0s Python

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

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

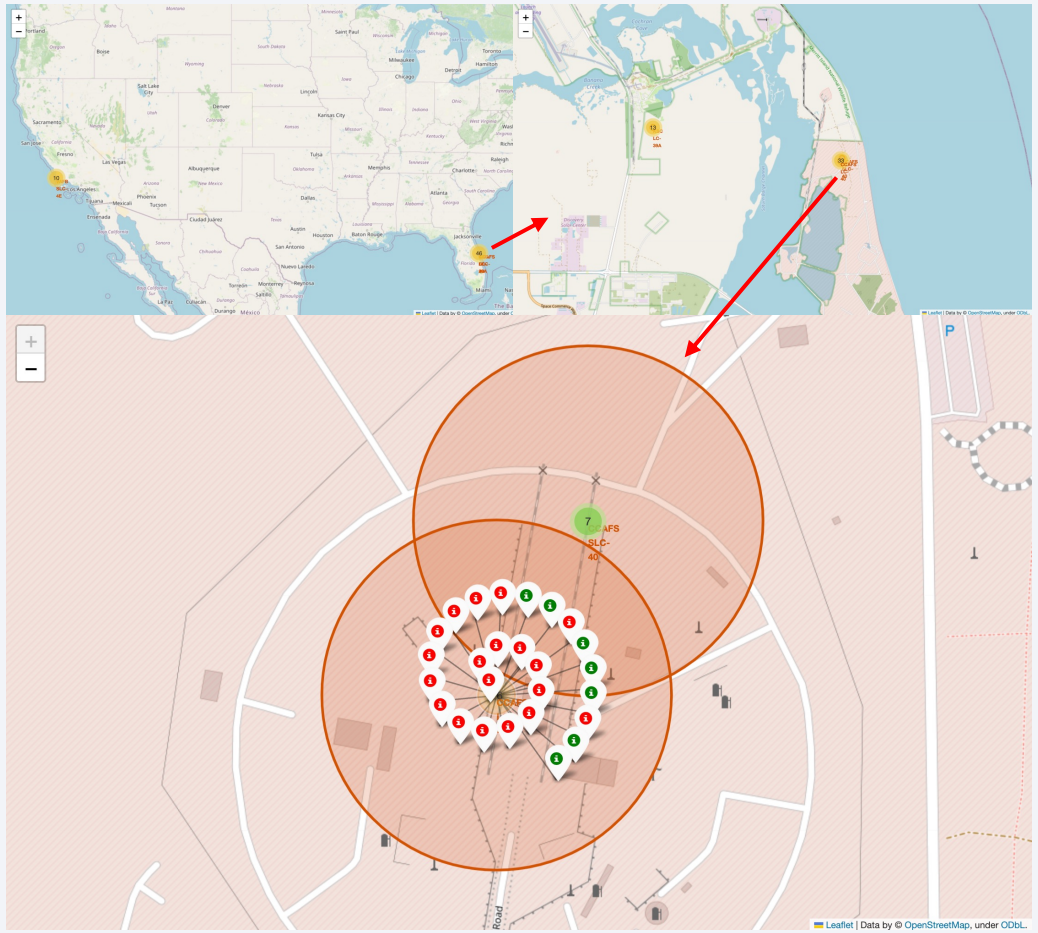

Section 3
Launch Sites
Proximities Analysis

All Launch Sites on a Map

- All SpaceX launch sites are on coasts of the United States of America, specifically Florida and California.

Success/Failed Launches for each Site

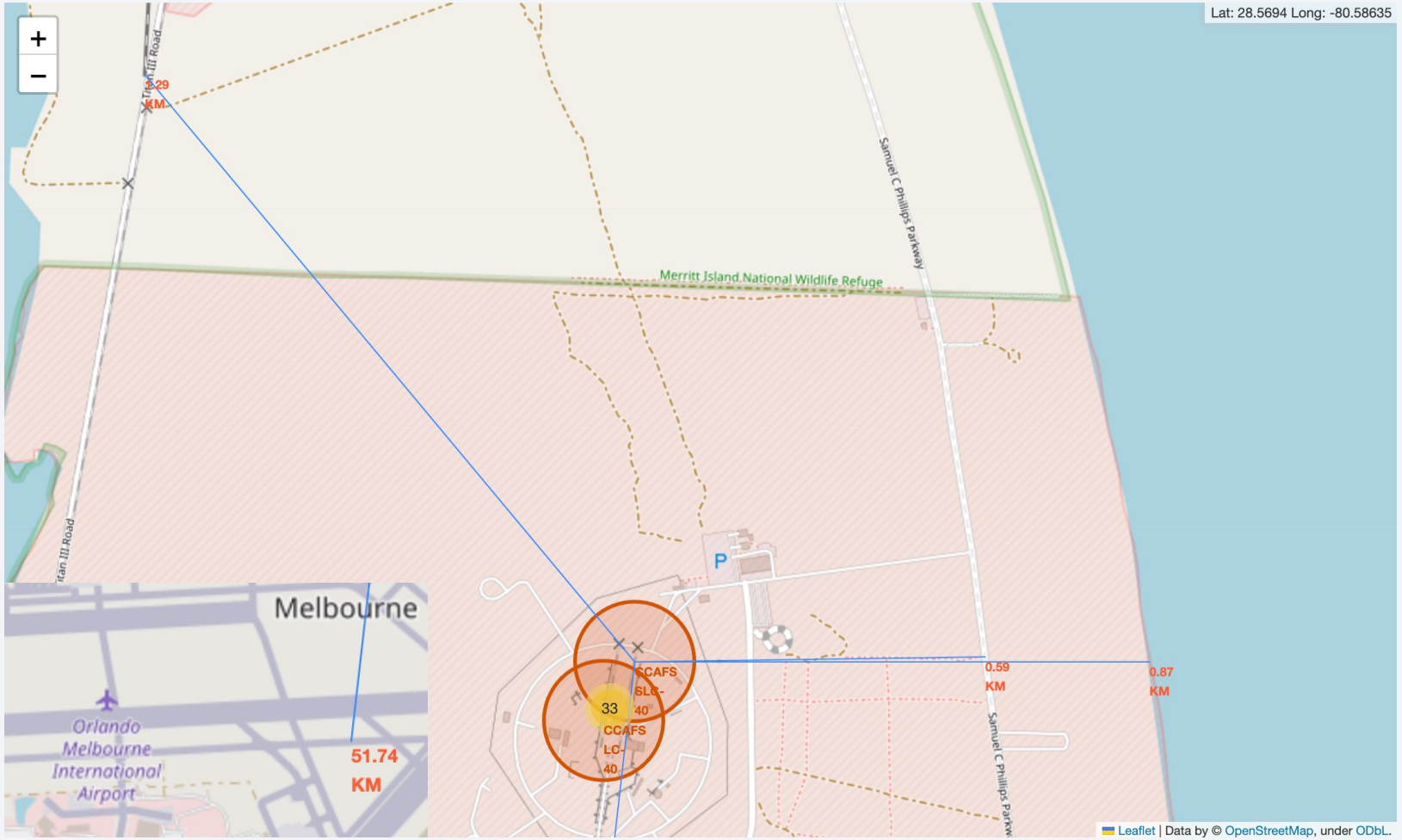
- Launches have been grouped into clusters, and annotated with green icons for successful launches, and red icons for failed launches.

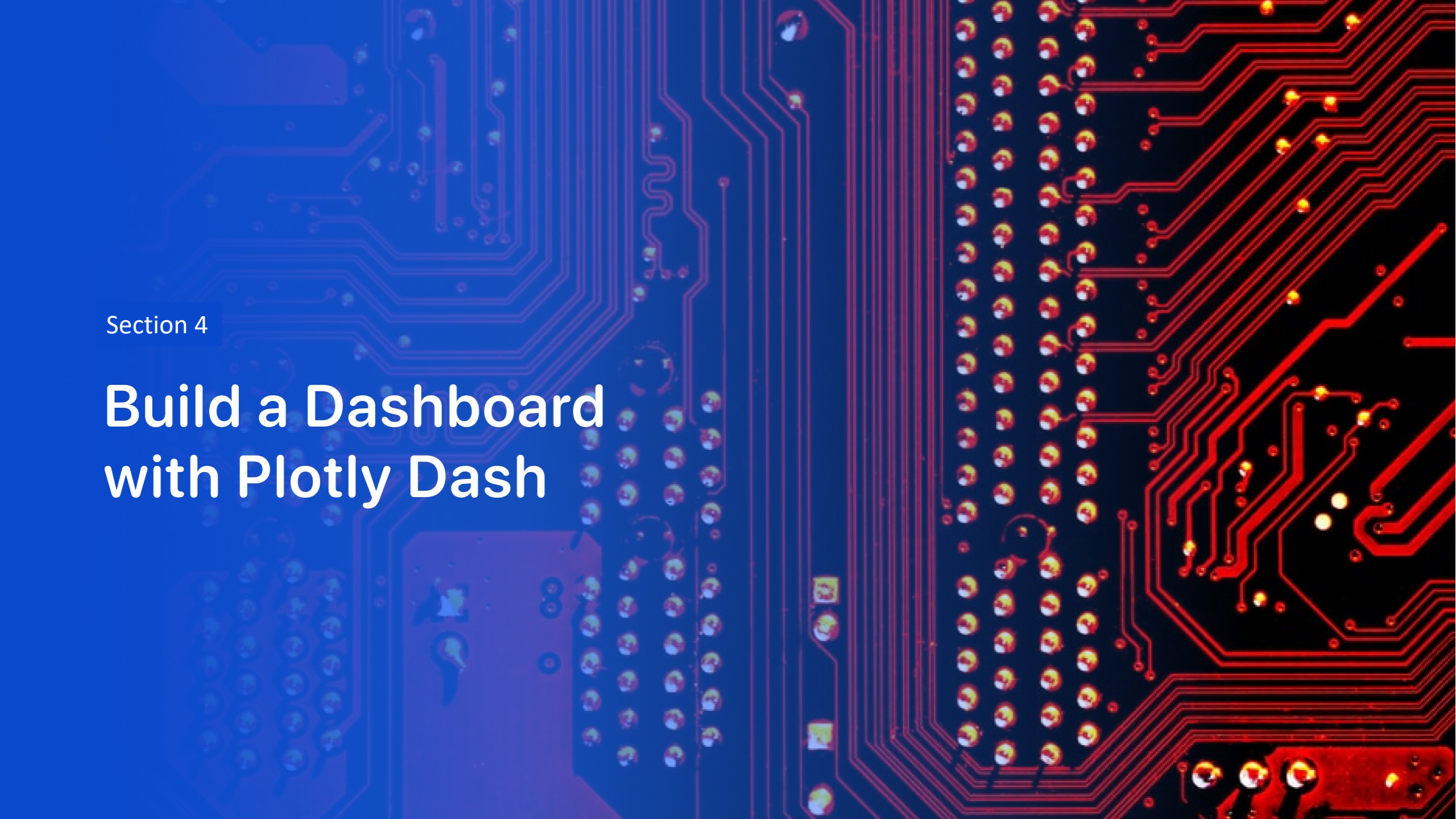


Proximity of Launch Sites to Other Points of Interest

From the folium map we know that for CCAFS SLC-40,

- The nearest railway is 1.29 km away,
- The nearest highway is 0.59 km away,
- The nearest coastline is 0.87 km away,
- The nearest airport is 51.74 km away



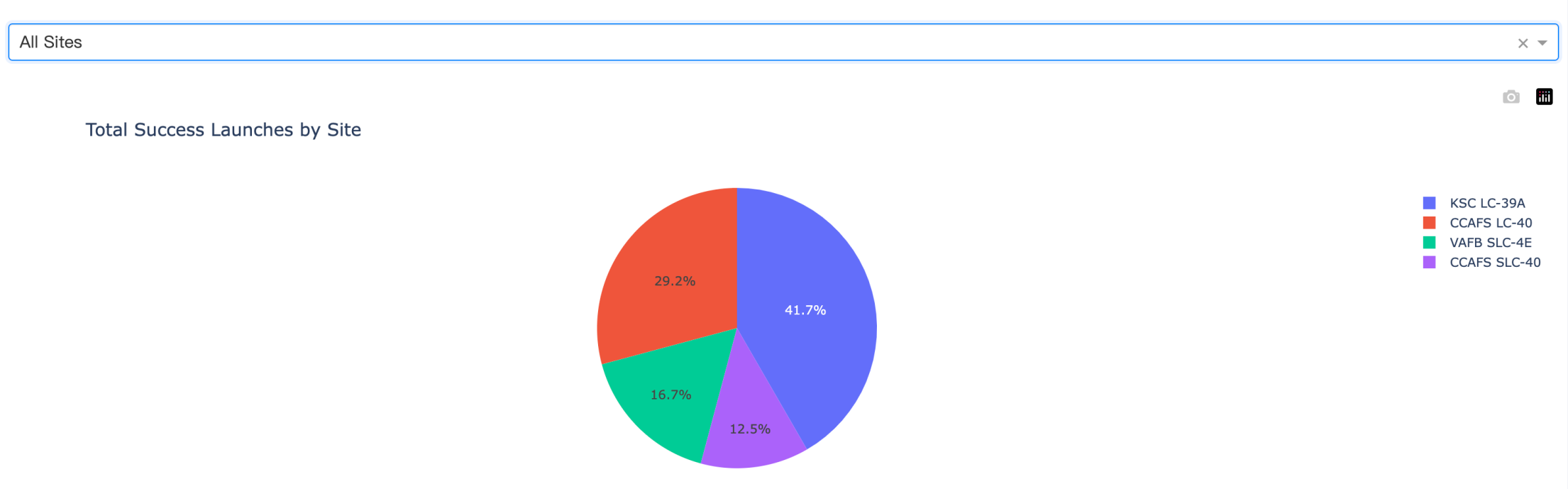


Section 4

Build a Dashboard with Plotly Dash

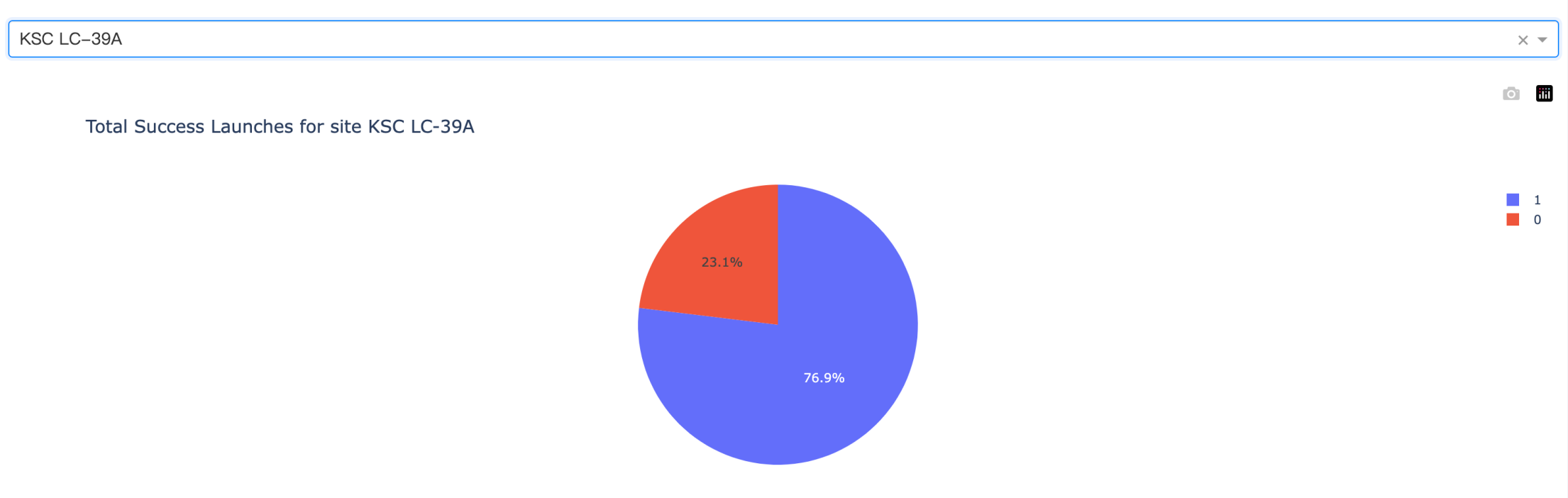
Launch Success Count for All Sites

- The launch site KSC LC-39 A had the most successful launches, with 41.7% of the total successful launches.



Pie Chart for the Launch Site with Highest Launch Success Ratio

- The launch site KSC LC-39 A also had the highest rate of successful launches, with a 76.9% success rate.



Launch Outcome VS. Payload Scatter Plot for All Sites

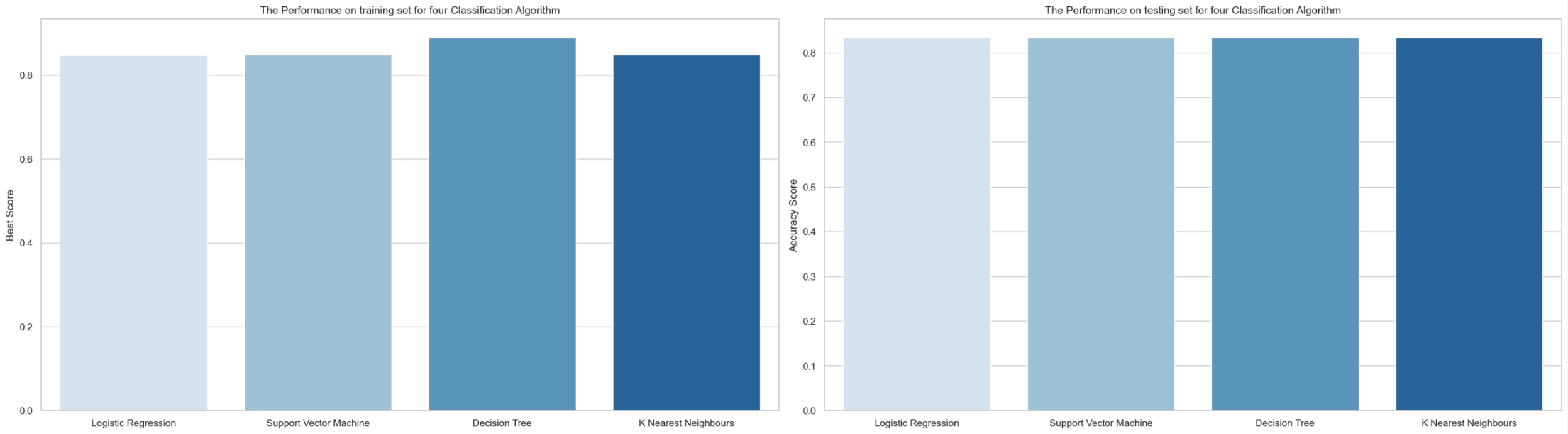
- Payloads mass between 3000 and 4000 kg have the largest success rate
- Booster Version is FT and payloads mass between 2000 and 3000 kg have the largest success rate

Section 5

Predictive Analysis (Classification)

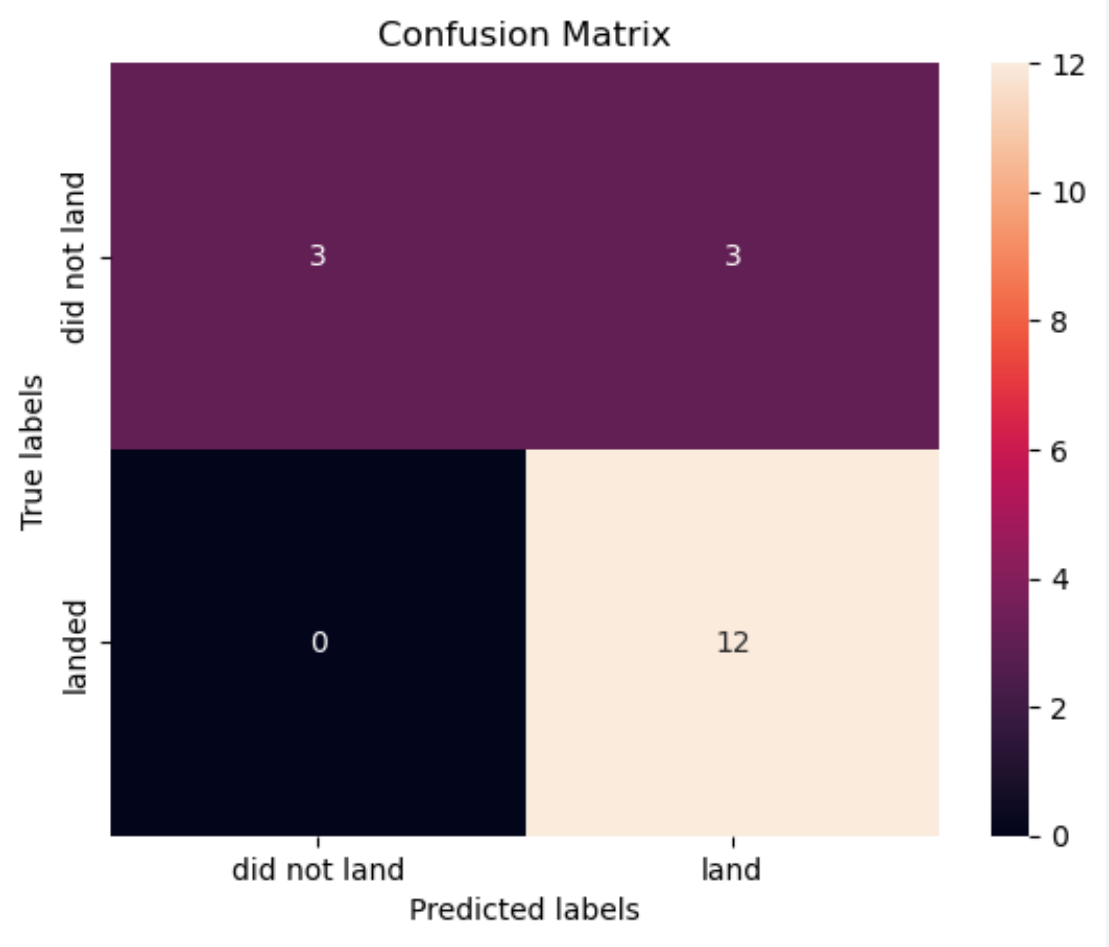
Classification Accuracy

- From the bar chart we can observe that decision tree and KNN has the highest classification accuracy



Confusion Matrix

- The confusion matrix of the best performing model – Decision Tree is as right
- 3 results were classified incorrectly
- 15 results were classified correctly
- For labels “did not land”, the correctly classified rate is 50%
- For labels “land”, the correctly classified rate is 100%



Conclusions

- Flights with flight numbers less than 25 were mostly launched at CCAFS SLC 40
- CCAFS SLC 40 has the lowest successful launch rate
- The successful launch rate increase as flight number increase
- Flights with payload mass higher than 7500 has a higher successful launch rate
- Orbit types VLEO and SSO have the highest and most impressive success rate higher than 80%
- For LEO, ISS, and PO flights with higher payload mass have a higher success rate
- Before 2013, all launched failed
- After 2013, the success rate increased, with small dips in 2018 and 2020
- The best-performing classification model for this dataset is the Decision Tree model

Appendix

- In Data Collection – SpaceX API
- define a series of helper functions that will help us use the API to extract information using identification numbers in the launch data.

Below we will define a series of helper functions that will help us use the API to extract information using identification numbers in the launch data.

From the `rocket` column we would like to learn the booster name.

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

[64] Python

From the `launchpad` we would like to know the name of the launch site being used, the logitude, and the latitude.

```
# Takes the dataset and uses the launchpad column to call the API and append the data to the list
def getLaunchSite(data):
    for x in data['launchpad']:
       if x:
        response = requests.get("https://api.spacexdata.com/v4/launchpads/"+str(x)).json()
        Longitude.append(response['longitude'])
        Latitude.append(response['latitude'])
        LaunchSite.append(response['name'])
```

[37] Python

From the `payload` we would like to learn the mass of the payload and the orbit that it is going to.

```
# Takes the dataset and uses the payloads column to call the API and append the data to the lists
def getPayloadData(data):
    for load in data['payloads']:
       if load:
        response = requests.get("https://api.spacexdata.com/v4/payloads/"+load).json()
        PayloadMass.append(response['mass_kg'])
        Orbit.append(response['orbit'])
```

[38] Python

From `cores` we would like to learn the outcome of the landing, the type of the landing, number of flights with that core, whether gridfins were used, wheter the core is reused, wheter legs were used, the landing pad used, the block of the core which is a number used to seperate version of cores, the number of times this specific core has been reused, and the serial of the core.

```
# Takes the dataset and uses the cores column to call the API and append the data to the lists
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cores/"+core['core']).json()
                Block.append(response['block'])
                ReusedCount.append(response['reuse_count'])
                Serial.append(response['serial'])
            else:
                Block.append(None)
                ReusedCount.append(None)
                Serial.append(None)
            Outcome.append(str(core['landing_success'])+' '+str(core['landing_type']))
            Flights.append(core['flight'])
            GridFins.append(core['gridfins'])
            Reused.append(core['reused'])
            Legs.append(core['legs'])
            LandingPad.append(core['landpad'])
```

[39] Python

Appendix

- In Data Collection – Scraping
- Define code to clean the data from the wiki

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

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

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

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

```
# Launch Site
# TODO: Append the bv into launch_dict with key `Launch Site`
launch_site = row[2].a.string
launch_dict['Launch site'].append(launch_site)
print(launch_site)

# Payload
# TODO: Append the payload into launch_dict with key `Payload`
payload = row[3].a.string
launch_dict['Payload'].append(payload)
print(payload)

# Payload Mass
# TODO: Append the payload_mass into launch_dict with key `Payload mass`
payload_mass = get_mass(row[4])
launch_dict['Payload mass'].append(payload_mass)
print(payload)

# Orbit
# TODO: Append the orbit into launch_dict with key `Orbit`
orbit = row[5].a.string
launch_dict['Orbit'].append(orbit)
print(orbit)

# Customer
# TODO: Append the customer into launch_dict with key `Customer`
if row[6].a!=None:
    customer = row[6].a.string
else:
    customer='None'
launch_dict['Customer'].append(customer)
print(customer)

# Launch outcome
# TODO: Append the launch_outcome into launch_dict with key `Launch outcome`
launch_outcome = list(row[7].strings)[0]
launch_dict['Launch outcome'].append(launch_outcome)
print(launch_outcome)

# Booster landing
# TODO: Append the launch_outcome into launch_dict with key `Booster landing`
booster_landing = landing_status(row[8])
launch_dict['Booster landing'].append(booster_landing)
print(booster_landing)
```

Thank you!