IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

<Genchang Peng>

<Nov. 29th,2024>

Outline

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

Executive Summary

- Summary of methodologies
  - ➢Data Collection Through API (Module 1)
  - ➢Data Collection with Web Scraping (Module 1)
  - ➢Data Wrangling (Module 1)
  - ➢Exploratory Data Analysis with SQL (Module 2)
  - ➢Exploratory Data Analysis with Data Visualization (Module 2)
  - ➢Interactive Visual Analytics with Folium (Module 3)
  - ➢Machine Learning Prediction (Module 4)
- Summary of all results
  - ➢Exploratory Data Analysis Results (Data Collection and Wrangling)
  - ➢Interactive Analytics Results (Data Visualization and Dashboard Interaction)
  - ➢Predictive Machine Learning Results (Classification Models)

Introduction

- Project background and context

  ➢Space X 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 Space X can reuse the first stage.**

  ➢ **We (Space Y) as a competitor, need to determine if the first stage will land, we can determine the cost of a launch.**

  ➢ This information can be used if we (Space Y) wants to bid against Space X for a rocket launch.

  ➢ The goal of this project is to **create a machine learning pipeline to predict if the first stage will land successfully**.

- Problems you want to find answers

  ➢ What **factors in the Space X dataset (e.g., payload mass, launch site, etc.)** determine the rocket will land successfully?

  ➢ How **can we visualize these factors** in an easy and explicit way?

  ➢ Can we **build the optimal machine learning model** to predict the successful rate?

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - We collect data using the **Space X API and Web Scrapping** from Wikipedia
- Perform data wrangling
  - We use **pandas Dataframe** to **replace missing values** in specific columns by their means()
- Perform exploratory data analysis (EDA) using **Python visualization and SQL (inline magic)**
- Perform interactive visual analytics using **Folium and Plotly Dash**
- Perform predictive analysis using classification models
  - Building four models: **logistic regression (LR), SVM, decision-tree and KNN**
  - Designing **cross-validation** by splitting training and testing data as 80% : 20%
  - Using **GridSearchCV** to find the optimal parameters of a model

Data Collection

- Describe how data sets were collected.

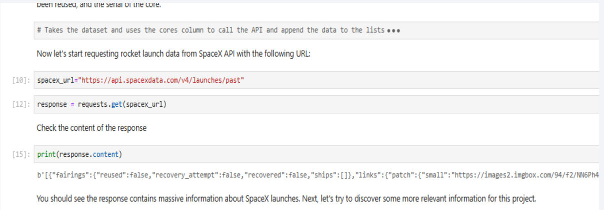
  1. **API Request**: Using requests.get(spacex-url) to request the Space X API
  2. **JSON to DataFrame:** Used .json() function to call the response, turn into pd.DataFrame using pd.json_normalize()
  3. **Data Wrangling:** Replacing the nan values by the mean() of PayloadMass
  4. **WEB Scraping:** Using BeautifulSoup() to requesting the HTML page of Falcon9 launch WIKI page
  5. **HTML to DataFrame:** Parsing the HTML table into pd.DataFrame using html5lib

Data Collection – SpaceX API

- Our basic steps for API data collection:
  - ➢Use **request.get()** to request the Space X API, get response
  - ➢ Use **json_normalize()** to get json response **into pandas DataFrame**
  - ➢ Use **me** columns **an()** to replace the nan in some (e.g., PayloadMass)

- GitHub URL: [(Git location: data collection via APIs)]()

1. Request the Space X


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 •••
```

Now let's start requesting rocket launch data from SpaceX API with the following URL:

```
[10]: spacex_url="https://api.spacexdata.com/v4/launches/past"
[12]: response = requests.get(spacex_url)
```

Check the content of the response

```
[15]: print(response.content)
```

b'[{"fairings":{"reused":false,"recovery_attempt":false,"recovered":false,"ships":[]},"links":{"patch":{"small":"https://images2.imgbox.com/94/f2/NN6Ph4

You should see the response contains massive information about SpaceX launches. Next, let's try to discover some more relevant information for this project.

2. Use json_normalize() to decode and turn into pd.DataFrame


Task 1: Request and parse the SpaceX launch data using the GET request

To make the requested JSON results more consistent, we will use the following static response object for this project:

```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_call_spacex_api.js
```

We should see that the request was successfull with the 200 status response code

```
response.status_code
```

200

Now we decode the response content as a Json using `.json()` and turn it into a Pandas dataframe using `.json_normalize()`

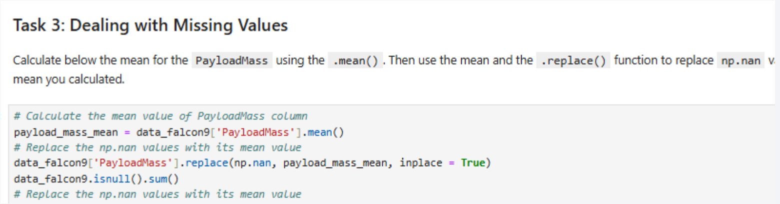
```
# Use json_normalize meethod to convert the json result into a dataframe
data = pd.json_normalize(response.json())
```

Using the dataframe `data` print the first 5 rows

```
# Get the head of the dataframe
data.head()
```

	rocket	payloads	launchpad	cores	flight_number	date_ut
				{'core': '5e9e289df35918033d3b2623', 'flight': 1, 'gridfins': False, 'legs': False,		2006-03

3. Used mean() of PayloadMass to replace the nan value


Task 3: Dealing with Missing Values

Calculate below the mean for the `PayloadMass` using the `.mean()`. Then use the mean and the `.replace()` function to replace `np.nan` v mean you calculated.

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

Data Collection - Scraping

- Our basic steps for Scraping data collection:
  - ➢ Use **request.get()** to get the Space X HTML
  - ➢ Use **beautifulsoup()** to extract the **HTML table**
  - ➢ Sparse the **HTML table into DataFrame**

- GitHub URL: [(Github location: data collection scraping)]()

1. Request the Space X HTML

TASK 1: Request the Falcon9 Launch Wiki page from its URL

First, let's perform an HTTP GET method to request the Falcon9 Launch HTML page, as an HTTP resp

```
# use requests.get() method with the provided static_url
# assign the response to a object
## 1, we get the Falcon9 Launch wiki page
html_data = requests.get(static_url)
html_data.status_code
```

```
200
```

2. Extract the HTML Table by beautifulsoup()

```
# Assign the result to a list called `html_tables`
## 2, sparsing the table
html_tables = soup.find_all('table')
```

Starting from the third table is our target table contains the actual launch

```
# Let's print the third table and check its content
first_launch_table = html_tables[2]
print(first_launch_table)
```

```
<table class="wikitable plainrowheaders collapsible" style="wic
```

You should able to see the columns names embedded in the table heade

```
``` •••
```

Next, we just need to iterate through the `&lt;th&gt;` elements and apply the

```
column_names = []

# Apply find_all() function with `th` element on first_launch_t
# Iterate each th element and apply the provided extract_columr
# Append the Non-empty column name (`if name is not None and le

element = soup.find_all('th')
for row in range(len(element)):
    try:
        name = extract_column_from_header(element[row])
        if (name is not None and len(name) > 0):
            column_names.append(name)
    except:
        pass
```

3. Sparsing table into DataFrame

After you have fill in the parsed launch record values into `launch_dict`, you can create

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

We can now export it to a **CSV** for the next section, but to make the answers consistent

Following labs will be using a provided dataset to make each lab independent.

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

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

Data Wrangling

- Our data wrangling steps:
  - ➢Use **isnull().sum** to identify the percentage of missing values
  - ➢Use **value_counts()** to calculate the number of the orbits, launches, mission outcomes, etc..
  - ➢ Create a landing outcome label
  - ➢ Use **mean()** to replace the nan in some columns (e.g., PayloadMass)
- GitHub URL: [(Github location: data wrangling)]()

1. Identify the missing values

```
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
```

2. Calculate number of launches

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

```
LaunchSite
```

3. Calculate number of orbits

Use the method `.value_counts()` to determin

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

```
Orbit
GTO     27
ISS     21
VLEO    14
PO       9
LEO      7
```

4. Creating a landing outcome label

Using the `Outcome`, create a list where the element
variable `landing_class`:

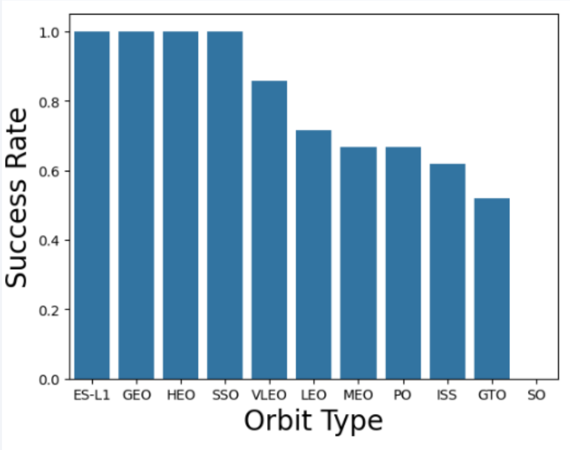
```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
### this one takes my time!!
landing_class = []
for key, value in df['Outcome'].items():
    if value in bad_outcomes:
        landing_class.append(0)
    else:
        landing_class.append(1)
```

This variable will represent the classification variable
first stage landed Successfully

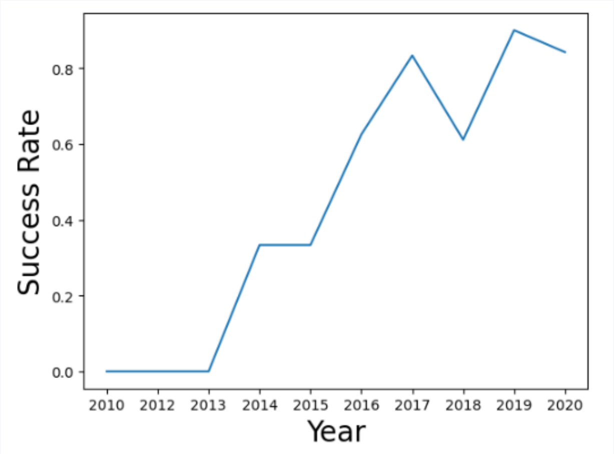
10

EDA with Data Visualization

- We use **scatter plot** to visualize the relationship among
  - ➢ Flight Number & Launch Site
  - ➢ Payload & Launch Site
  - ➢ Flight Number & Orbit Type
  - ➢ Payload & Orbit Type



- We use **bar plot (upper right)** to visualize the Success Rate of Orbit Type
- We use **line plot (lower right)** to visualize the trend between Year and Success Rate



- GitHub URL (Github location: data visualization)

EDA with SQL

- We use sqlite to connect to our database my_data1.db, read Spacex.csv, created the TABLE SPACEXTABLE, and perform the following sql queries ([Github location: sqlite]()):
  - Names of the unique launch sites in the space mission
  - Display 5 records where launch sites begin with the string 'CCA',
  - Display the total payload mass carried by boosters launched by NASA (CRS)
  - Display average payload mass carried by booster version F9 v1.1
  - List the date when the first succesful landing outcome in ground pad was achieved
  - List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000
  - List the total number of successful and failure mission outcomes
  - List the names of the booster_versions which have carried the maximum payload mass. Use a subquery
  - 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.
  - 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.

Build an Interactive Map with Folium

- We marked launch sites, and added map objects like **markers, circles, lines** to mark the success or failure of launches for each site on the folium map
- We assigned the feature launch outcomes (failure or success) to class 0 and 1.i.e., 0 for failure, and 1 for success.
- Using the **color-labeled marker clusters**, we identify which launch sites have relatively high successful rate
- We calculate the distances between a launch site to its proximities.
- GIT location: (Github location: Folium location)

Build a Dashboard with Plotly Dash

- We build an interactive dashboard via **Plotly Dash**
- We plot the **pie chart** showing the total launches by a certain site
- We plot **scatter plot** showing the relationship between Outcome and Payload Mass for different booster version
- GIT location: (Github location: Dashboard)

Predictive Analysis (Classification)

- We load the data using **numpy and pandas**, transformed the data, split into training and testing
- We build four models, **Logistic Regression, SVM, Decision Tree and KNN**
- We use **GridSearchCV** to find the best parameters
- We use accuracy as the metric, using feature engineering and algorithm tuning, find Decision Tree as optimal model
- GIT location: (Github location: machine learning)

Results

- Exploratory data analysis results
  - Successfully collect SpaceX launching data via API and Web scaping
  - Perform data wrangling, and use SQL query to read the data information
- Interactive analytics demo in screenshots
  - Visualize the relationship between success rate and different factors (e.g., Payload Mass)
  - Visualize the location of launch site, create the dashboard
- Predictive analysis results
  - Designed machine learning pipelines with four different models
  - Perform classification and find the optimal results

Section 2

Insights drawn from EDA

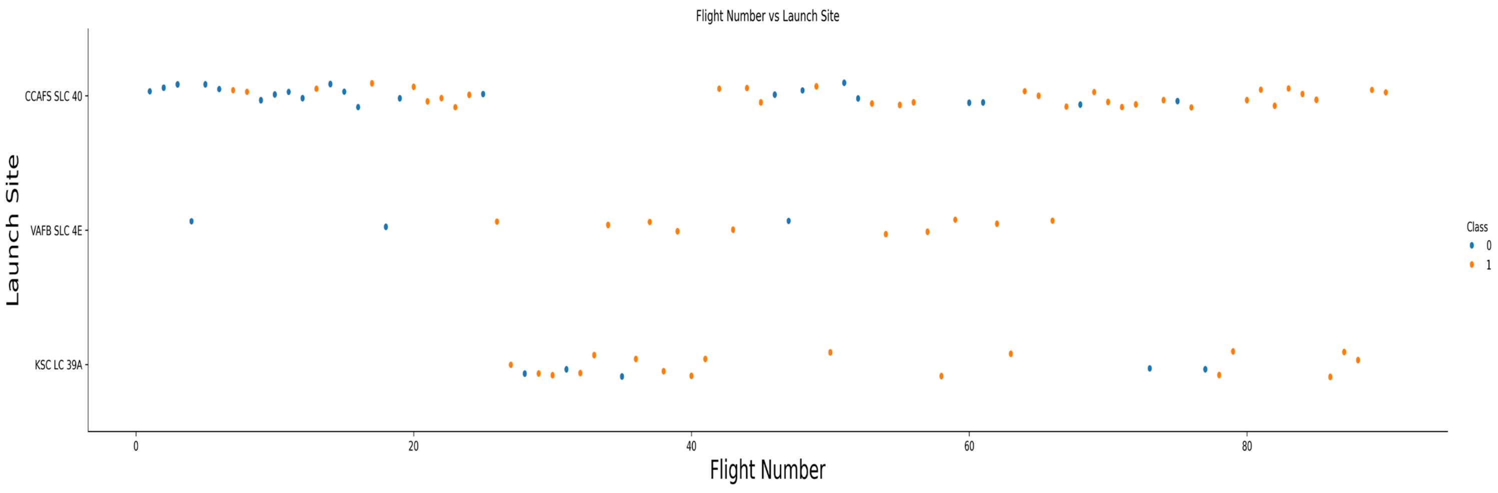
Flight Number vs. Launch Site

- A scatter plot of Flight Number vs. Launch Site
- 0 is failure, 1 is success
- We noticed that:

  ➢The **later year, more likely to success**

  ➢CCAFS SLC 40 launch site has the most launches, but only half success rate

  ➢KSC LC 39 A and VAFB SLC 30, while less launches, have higher success ful rate than CCAFS SLC 40

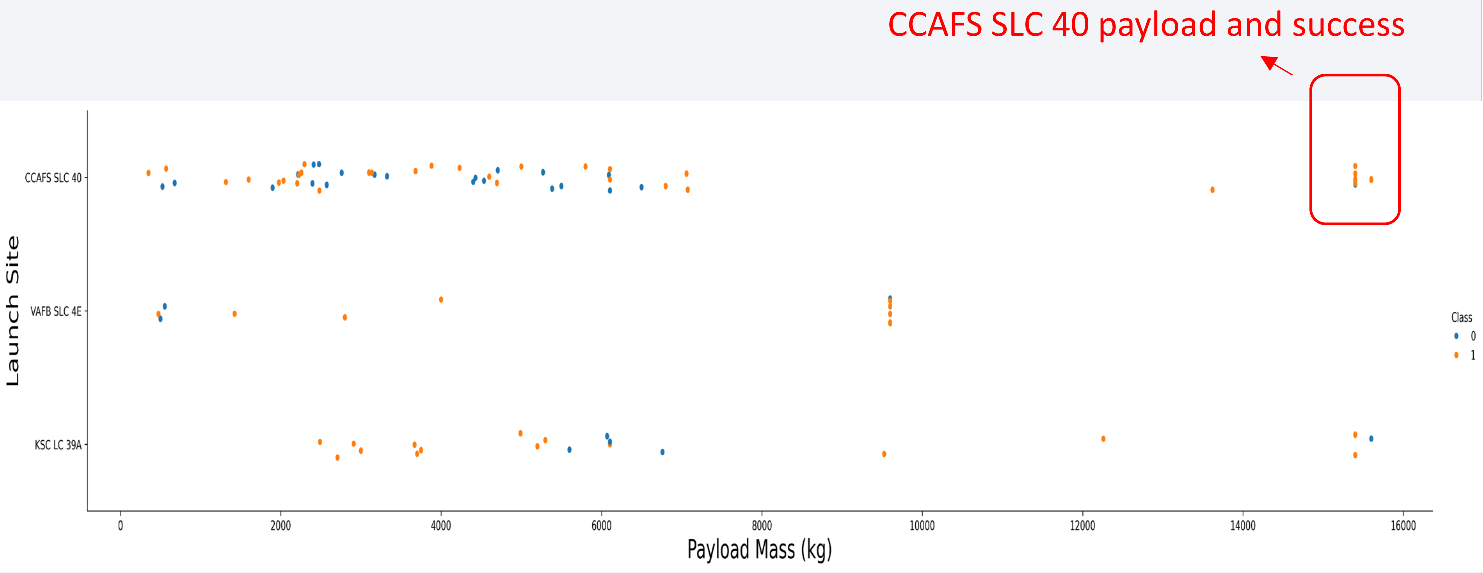


Payload vs. Launch Site

- A scatter plot of Payload vs Launch Site
- 0 is failure, 1 is success
- We noticed that:

  ➢For CCAFS SLC 40, the greater payload mass, the more success rate (see the red box)

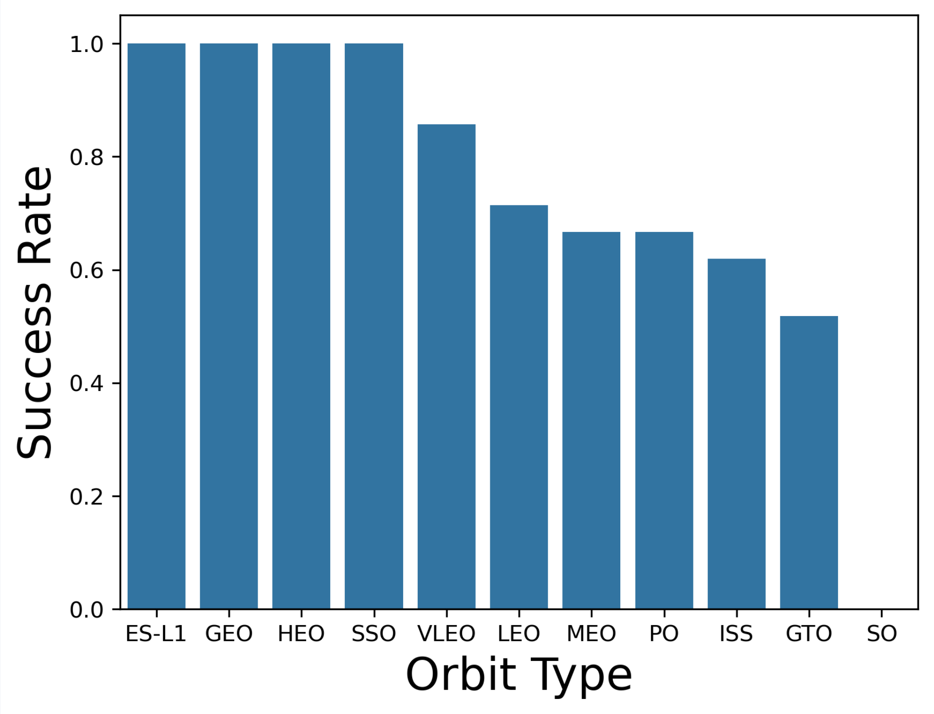
  ➢KSC LC 39 A and VAFB SLC 30, due to less launches, cannot be surely stated about such observation



Success Rate vs. Orbit Type

- A bar plot to show the Success Rate vs Orbit Type
- They are shown in descending order of Success Rate
- We noticed that:

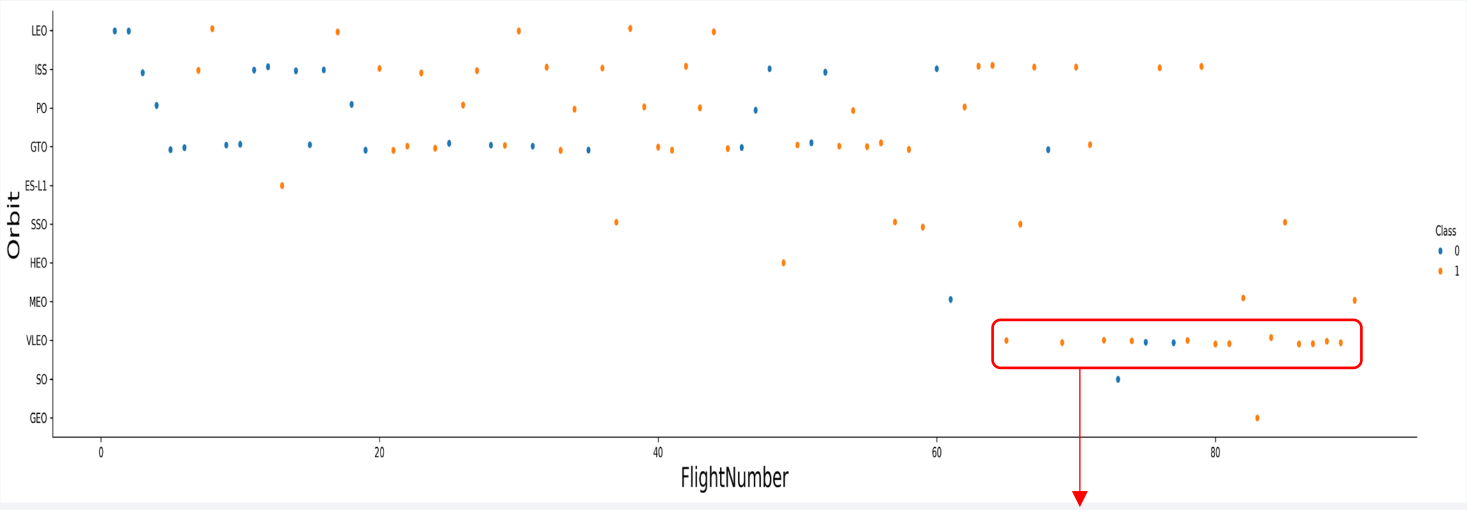
➢For **ES-L1, GEO, HEO, SSO and VLEO**, they have the most Success Rate



Flight Number vs. Orbit Type

- A scatter plot of Flight Number vs Orbit Type
- 0 is failure, 1 is success
- We noticed that:

  ➢For VLEO orbit, high Flight Number, more Success Rate (see the red box)



VLEO orbit flight number and success

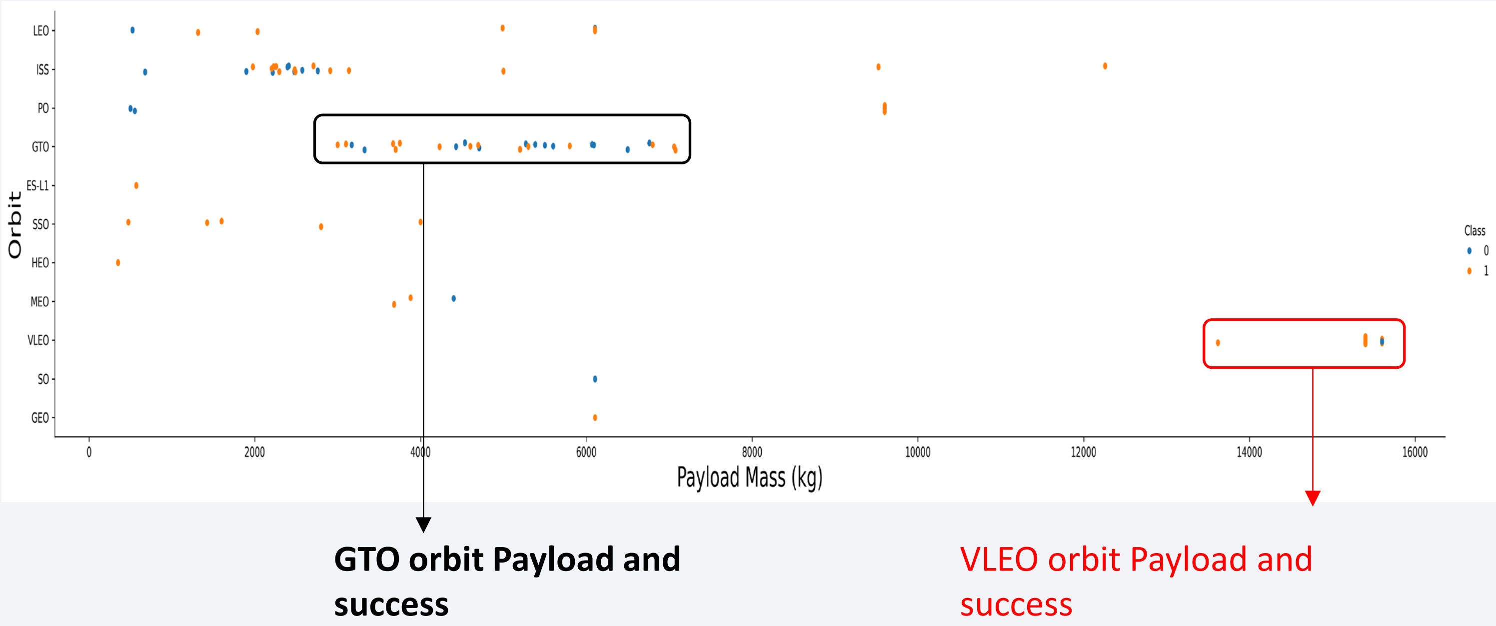
Payload vs. Orbit Type

- A scatter plot of Payload vs Orbit Type
- 0 is failure, 1 is success
- We noticed that:

  ➢Most Payload Mass (kg) is between 2000 to 6000

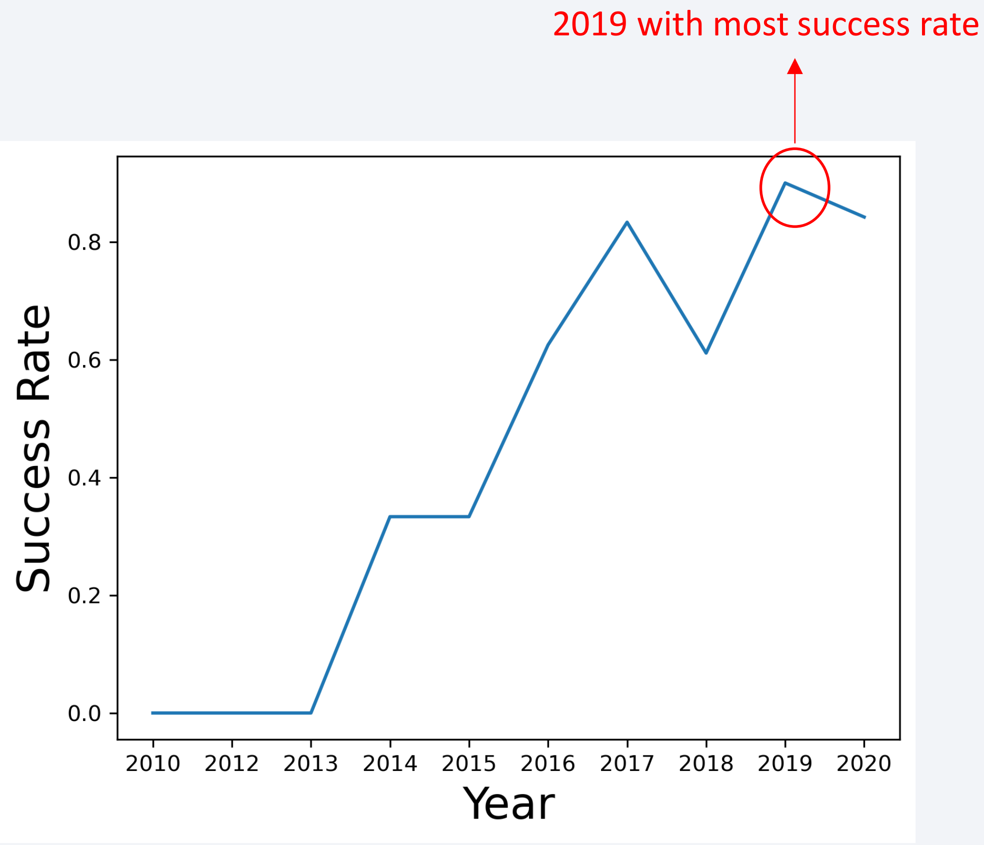
  ➢VLEO orbit sees the hightest payload, with 3 Success (see red box)

  ➢GTO, with most launches, the Success is not very satisfactory (see black box)



Launch Success Yearly Trend

- A line plot of Launch Success and Year
- We noticed that:
  - ➢In general, the launce success increases with new years
  - ➢2019 sees the highest success rate (marked in red)



All Launch Site Names

- We use SELECT DISTINCE launch_site, from SPACEXTABLE
- We notice there are **four** launch site names
  - ➢CCAFS LC-40
  - ➢VAFB SLC-4E
  - ➢KSC LC-39A
  - ➢CCAFS SLC-40

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

```
[57]: %sql SELECT DISTINCT launch_site from SPACEXTABLE;

 * sqlite:///my_data1.db
Done.
[57]:
```

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

Launch Site Names Begin with 'CCA'

- We use SELECT * to select from table
- We use **WHERE launch_site like ‘CCA%’** as condition to select only ‘CCA’
- We use LIMIT 5 to show the 5 records

```
%sql SELECT * from SPACEXTABLE where launch_site like 'CCA%' limit 5;
```

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

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	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	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Total Payload Mass

- We use **SUM(PAYLOAD_MASS_KG)** to calculate the total Payload Mass
- We use **WHERE Customer = ‘NASA (CRS)’** as condition to select those launched by NASA (CRS)
- Total Payload Mass(kg): 45596

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

```
%sql SELECT SUM(PAYLOAD_MASS__KG_) AS total_payload_mass FROM SPACEXTABLE WHERE Customer = 'NASA (CRS)'
```

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

total_payload_mass
45596

Average Payload Mass by F9 v1.1

- We use **AVG(PAYLOAD_MASS_KG)** to calculate the average Payload Mass
- We use **WHERE Booster_Version = ‘F9 v1.1’** as the condition to select those by F9 V1.1
- Average Payload Mass(kg): 2928.4

```
]: %sql SELECT AVG(PAYLOAD_MASS__KG_) AS average_payload_mass FROM SPACEXTABLE WHERE Booster_Version = 'F9 v1.1'

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

average_payload_mass
2928.4

First Successful Ground Landing Date

- We use **MIN(Date)** to find the first (min) landing date
- We use **WHERE Landing_Outcome= ‘Success (Ground pad)’** as the condition to select the success
- First successful ground landing data: 2015-12-22

```
%sql SELECT MIN(Date) AS first_successful_landing FROM SPACEXTABLE WHERE Landing_Outcome = 'Success (ground pad)'
```

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

first_successful_landing
2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

- In WHERE, we have two conditions below:
  - ➢Success (drone ship)
  - ➢PAYLOAD_MASS_KG BETWEEN 4000 AND 6000, as the condition of payload range
- Four boost versions:
  - ➢F9 FT B1022
  - ➢F9 FT B1026
  - ➢F9 FT B1021.2
  - ➢F9 FT B1031.2

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 SPACEXTABLE WHERE Landing_Outcome = 'Success (drone ship)' AND PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000
```

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

Booster_Version
F9 FT B1022
F9 FT B1026
F9 FT B1021.2
F9 FT B1031.2

Total Number of Successful and Failure Mission Outcomes

- We **SELECT Mission_Outcome, Count(*)** to count the number of outcomes
- We use **GROUP BY Mission_Outcome** to group the outcomes by success and failure
- Results are:
  - ➢ Success: 98+1+1=100
  - ➢ Failure: 1

Task 7

List the total number of successful and failure mission outcomes

```
%sql SELECT Mission_Outcome, COUNT(*) FROM SPACEXTABLE GROUP BY Mission_Outcome
```

```
 * 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

- We have **subquery**:
  - We use **SELECT MAX(PAYLOAD_MASS_KG_) FROM SPACEXTABLE**, to select the maximum payload, as subquery
  - In the outside, we **SELECT Booster_Version FROM SPACEXTABLE**, and put the **subquery** after WHERE
- There are **12 booster_versions**

Task 8

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

```
%sql SELECT Booster_Version FROM SPACEXTABLE WHERE PAYLOAD_MASS__KG_ =(SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTABLE)
```

```
 * 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

- In Python SQLLite inline querey, we use **SUBSTR(Date, 6, 2)** as Month
- We use **SUBSTR(Date, 0, 5)=2015** as year
- Two records are:
  - ➢ 2015-01-01
  - ➢ 2015-04-14

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, 6,2) as month to get the months and substr(Date,0,5)='2015' for year.

```
%sql SELECT SUBSTR(Date, 6,2) AS Month, Date, Booster_Version, launch_site, Landing_Outcome FROM Landing_Outcome = 'Failure (drone ship)' and SUBSTR(Date
```

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

Month	Date	Booster_Version	Launch_Site	Landing_Outcome
01	2015-01-10	F9 v1.1 B1012	CCAFS LC-40	Failure (drone ship)
04	2015-04-14	F9 v1.1 B1015	CCAFS LC-40	Failure (drone ship)

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

- We use **WHERE Date BETWEEN** as condition
- We use **GROUP BY Landing_Outcomes** to group by each outcomes
- We have **COUNT(*)** to count the outcomes
- We rank them by **DESC**
- Eight outcomes in descending order:
  - ➢ No attempt is the most number
  - ➢ Precluded (drop ship) is the least

Task 10

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.

```
AS count_landing_outcomes FROM SPACEXTABLE WHERE Date BETWEEN '2010-06-04' AND '2017-03-20' GROUP BY Landing_Outcome ORDER BY count_landing_outcomes DESC
```

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

Landing_Outcome	count_landing_outcomes
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

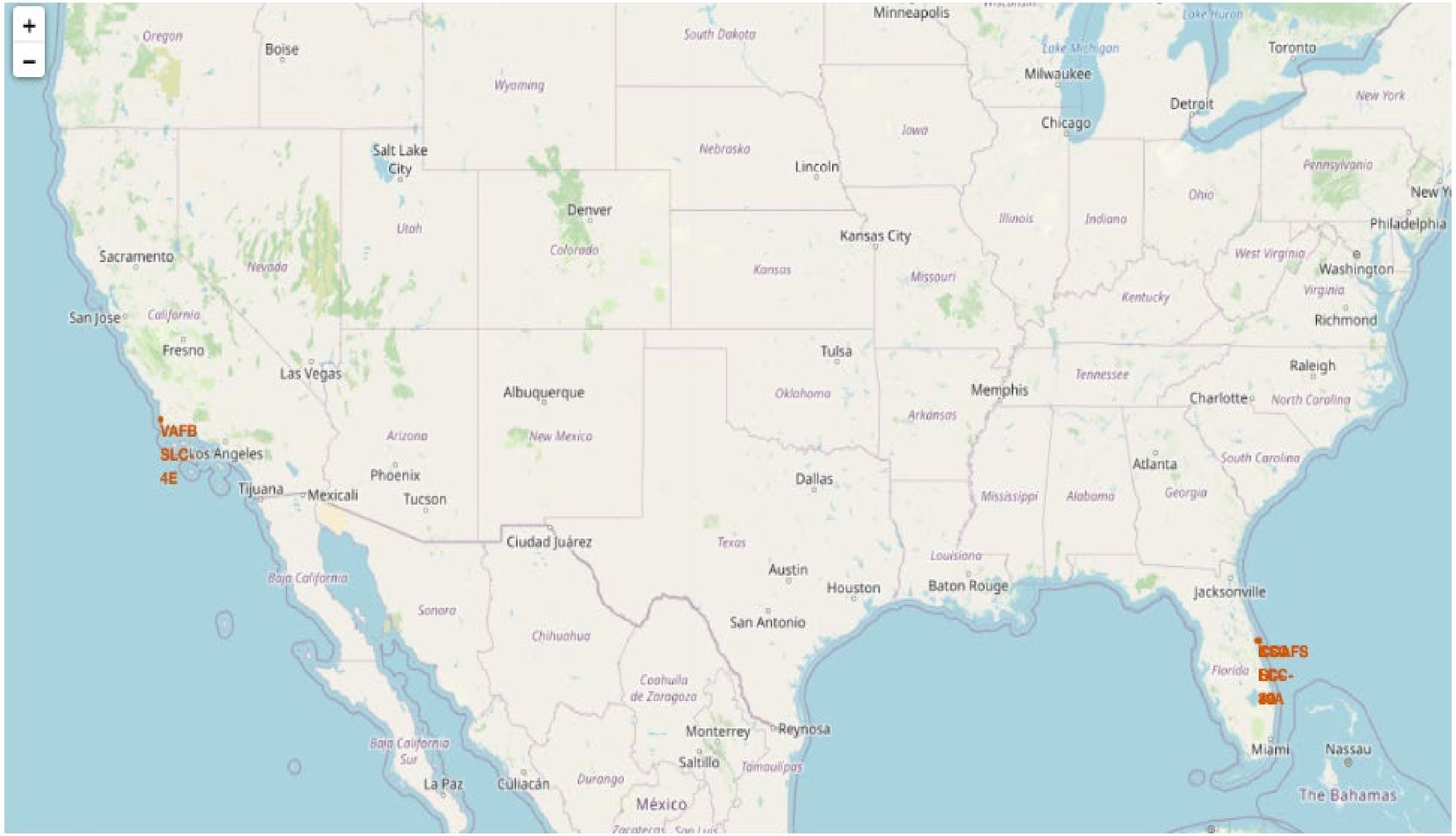
Section 3

Launch Sites Proximities Analysis

Global Map Markers of All Launch Sites

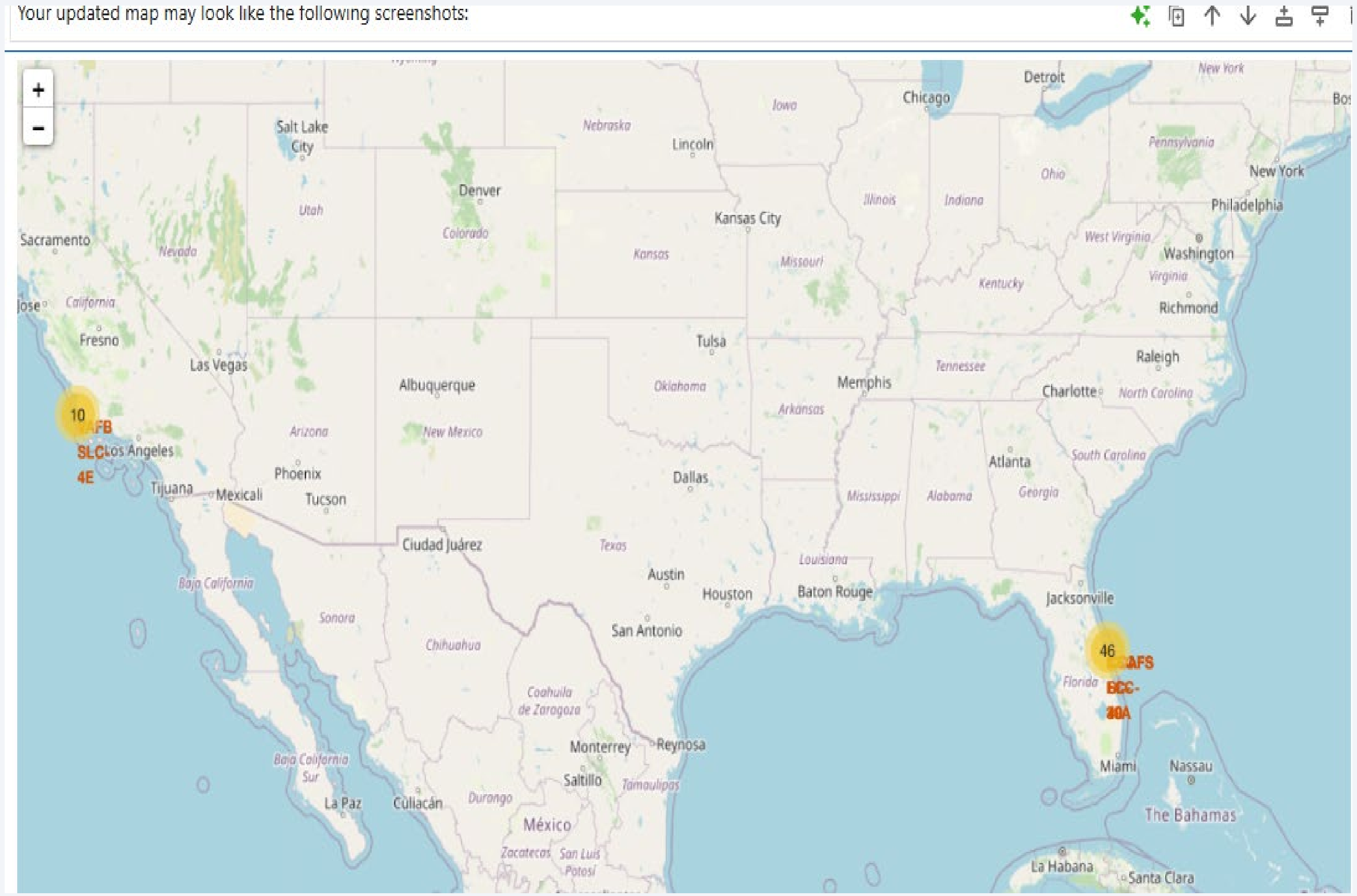
- We note there are two sites, one in East lower latitude, another in West higher latitude

The generated map with marked launch sites should look similar to the following:



Success/Failed Launches of Each Site

- East has 46 launches, West has 10 launches



<Folium Map Screenshot 3>

- Replace <Folium map screenshot 3> title with an appropriate title

- Explore the generated folium map and show the screenshot of a selected launch site to its proximities such as railway, highway, coastline, with distance calculated and displayed

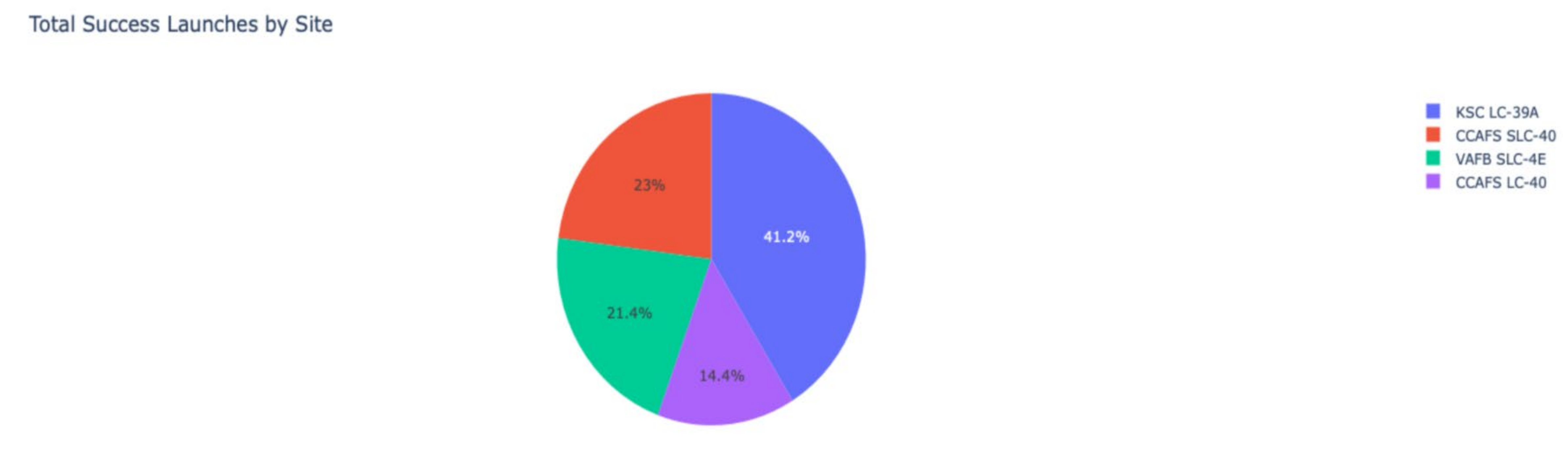
- Explain the important elements and findings on the screenshot

Section 4

Build a Dashboard with Plotly Dash

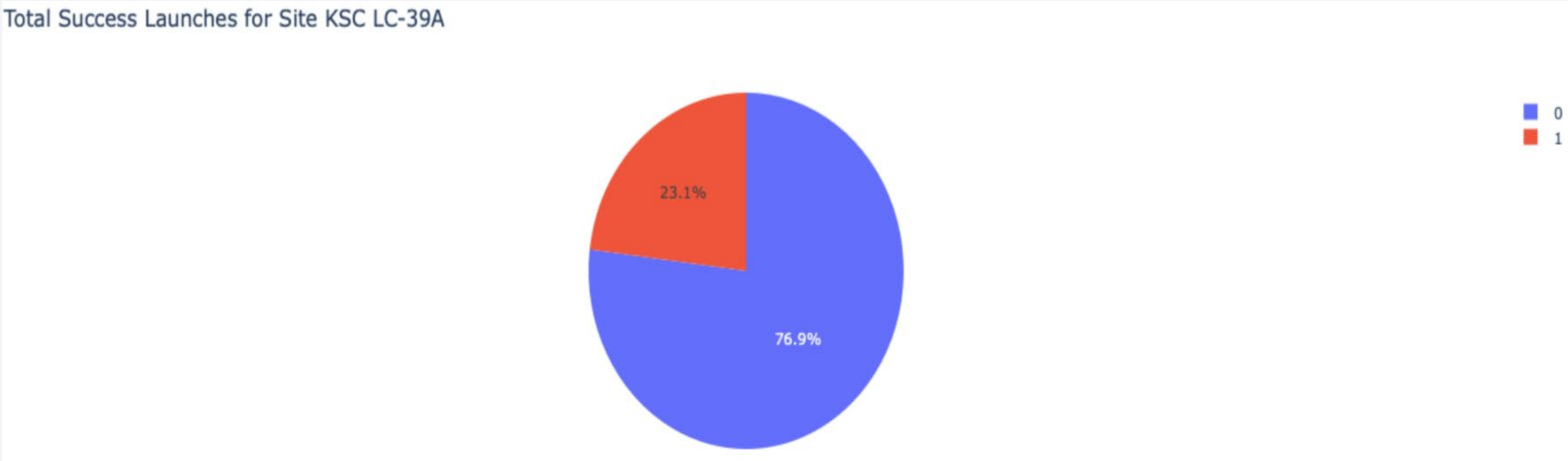
Success Percentage of Each Launch Site

- KSC LC-39A is most success launches



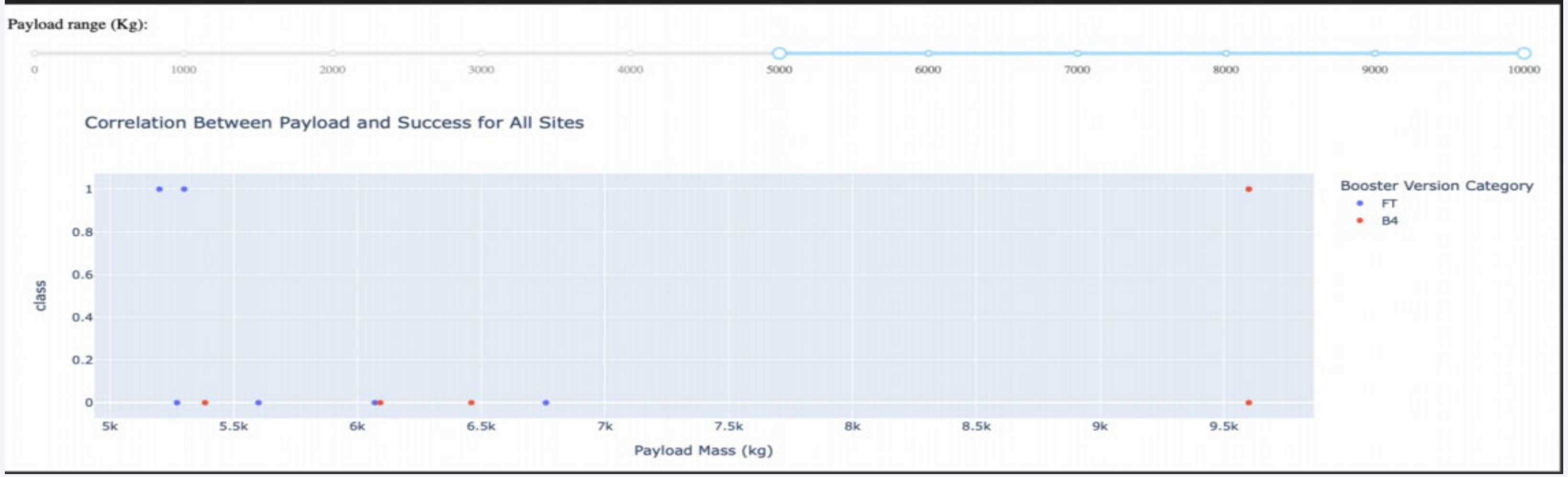
Success Ratio of KSC LC-39A

- KSC LC-39A, as the most success launch site, has the success ratio of 76.9%



Scatter Plot of Payload vs Launch Outcomes

- Success rate in light-weighted payload (lower than 5000kg, left) is higher than heavy-weighted payload (higher than 5000kg, right)

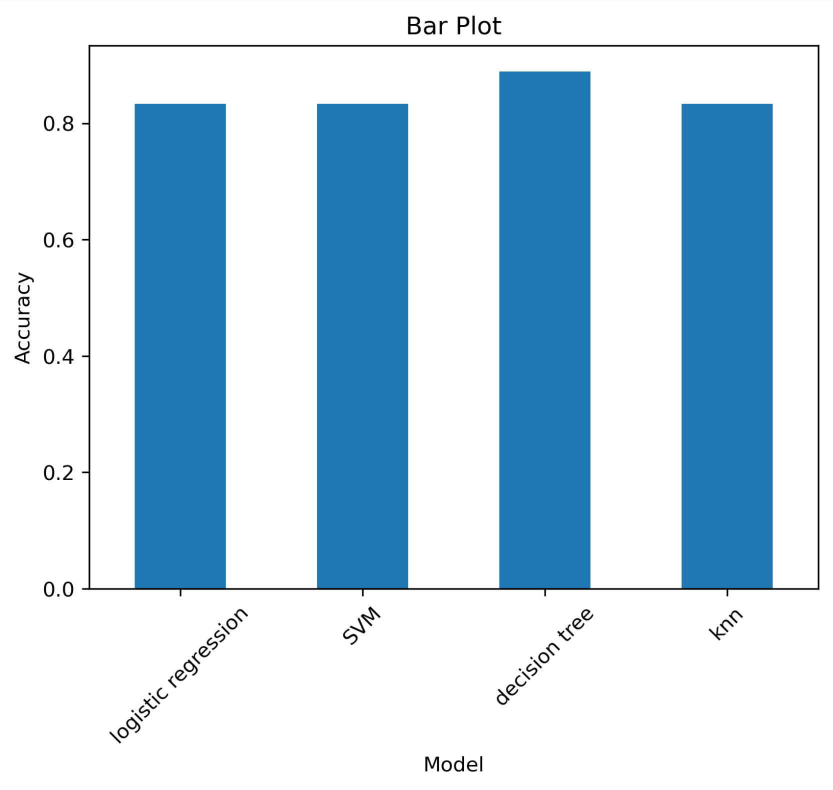


Section 5

Predictive Analysis (Classification)

Classification Accuracy

- By comparing the accuracy of four models as barplot, we find **Decision-Tree** has the highest accuracy of 0.88 (or 88.89%, marked in red)

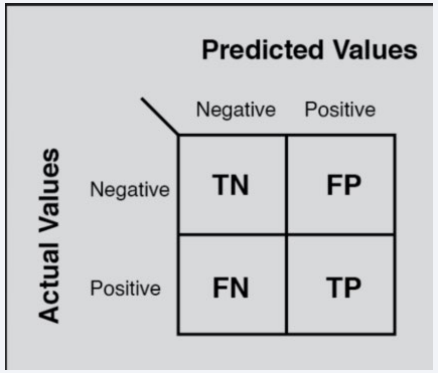


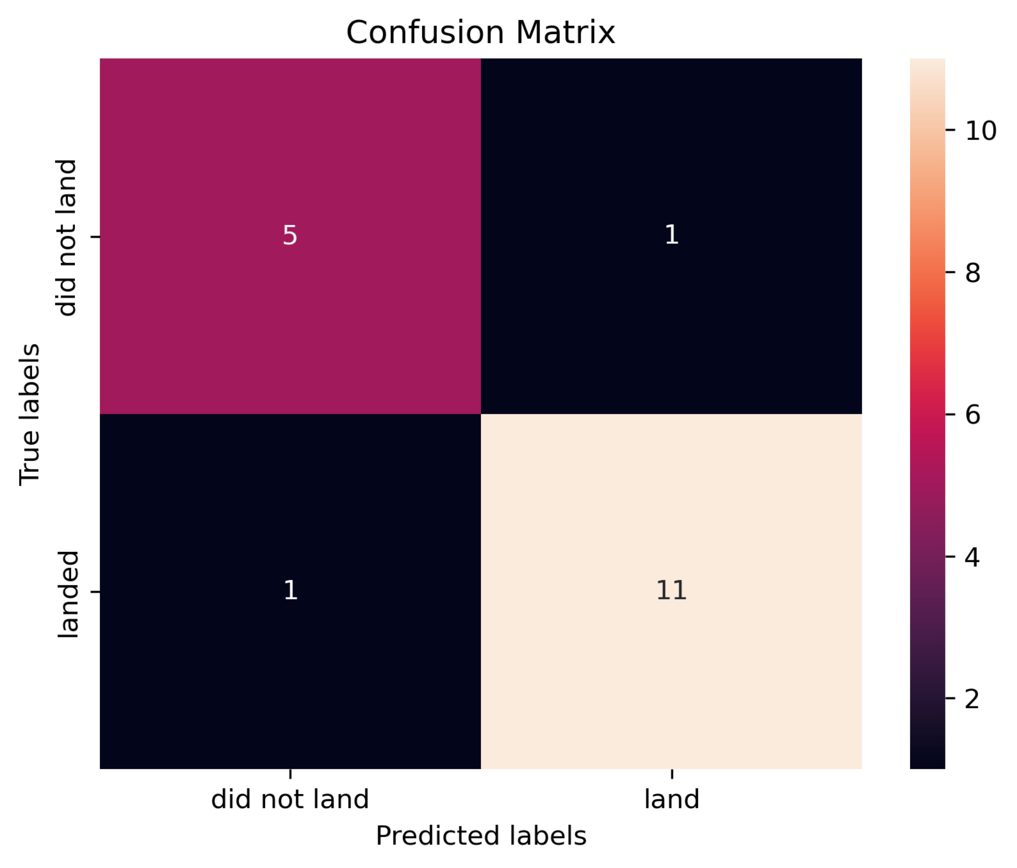
	Accuracry
logistic regression	0.833333
SVM	0.833333
decision tree	0.888889
knn	0.833333

Decision tree has highest accuracy

Confusion Matrix

- We draw the confusion matrix of decision-tree
  - ➢ It successfully classify the 'did not land (true negative/TN)' and 'land (true positive/TP)', which can be found in the diagonal
  - ➢ Also, the false negative (FN, lower left) and false positive (FP, upper right) are pretty small, only 1 in each case
  - ➢ The data is imbalanced (land > did not land), but decision tree still performs good





Conclusions

- **Data Collection and Wrangling:**
  - ➢ In Python, we use **API and HTML Sparse** to collect the online data, and store them into pandas DataFrame
  - ➢ We pre-process the DataFrame, **replacing the nan or missing values** by others, e.g., the mean value of one column
- **Insight Drawn from EDA:**
  - ➢ We visualize the relationship among different factors, and find the **increasing trend between success rate and years**
  - ➢ We use SQL to query data, and **summarize the statistics of different columns**, e.g., average payload mass, landing records in 2015 year, etc
- …

Conclusions – Cont.

- **Launching Sites Proximities Analysis:**
  - ➢ Using Folium, we locate the **two different launch sites**, and their locations
- **Build a Dashboard with Plotly Dash:**
  - ➢ We draw pie chart to find the **launch site KSC LC-39A has the highest success rate** of 76.9%
- **Predictive Analysis (Classification):**
  - ➢ We **compare LR, SVM, decision-tree and KNN**, using GridSearchCV + 10-fold cross-validation
  - ➢ We find **decision-tree has the highest accuracy of 0.88 (or 88.89%)**

Appendix

- In the function plot_confusion_matrix() of Machine Learning module, I have added a line to save the figure (Github location: machine learning)
- In SQL query module (Github location: sqlite), I implemented all SQL query via line-magic of SQLLite in Python

Thank you!