Executive Summary slide

- Introduction slide
- data collection and data wrangling methodology
- EDA and interactive visual analytics methodology
- predictive analysis methodology related slides
- EDA with visualization results slides
- EDA with SQL results slides
- interactive map with Folium results slides
- Plotly Dash dashboard results slides
- predictive analysis (classification)
- Conclusion slide

Introduction

- Project content:
  - the goal for this project is to predict the Falcon 9 first stage will successfully landing
  - it cost 165 million $ for rockest
  - Useful data for other company who interested in race with spaceX
- Problems we should solve to get answers:
  - The important fact that make landing successful or failing
  - True and Different relationship between rockets and success and fail rates

Methodology

- Data collection Methodology:
  - Web scrapping from Wikipedia and spaceX API
- Data Wrangling
  - Cleaning data and delete unuseful(with good relation) data
- Exploratory data analysis using SQL and Visualization
- Visual analytics and folium and plotly dash
- Predictive analysis using classification models

Data Collection

- collecting datasets from spaceX API:
  - SpaceX API
  - Json file
  - Converting to DataFrame
  - Wrangling nad cleaning data

- Web scrapping from Wikipedia:
- Html data from Wikipedia
- Making soup object by BeautifulSoup
- Making DatFrame
- Explore data

SpaceX API

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

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']:
        response = requests.get("https://api.spacexdata.com/v4/launchpads/"+str(x)).json()
        Longitude.append(response['longitude'])
        Latitude.append(response['latitude'])
        LaunchSite.append(response['name'])
```

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']:
        response = requests.get("https://api.spacexdata.com/v4/payloads/"+load).json()
        PayloadMass.append(response['mass_kg'])
        Orbit.append(response['orbit'])
```

```
# 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'])
```

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.json'
```

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

Using the dataframe `data` print the first 5 rows

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

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

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Lo
0	1.0	2006-03-24	Falcon 1	20.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0	Merlin1A	16
1	2.0	2007-03-21	Falcon 1	NaN	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0	Merlin2A	16
2	4.0	2008-09-28	Falcon 1	165.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0	Merlin2C	16
3	5.0	2009-07-13	Falcon 1	200.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0	Merlin3C	16
4	6.0	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0003	-8

Data Wrangling

We can see below that some of the rows are missing values in our dataset.

```
data_falcon9.isnull().sum()
```

```
FlightNumber       0
Date              90
BoosterVersion     0
PayloadMass       10
Orbit              0
LaunchSite         0
Outcome            0
Flights            0
GridFins           0
Reused             0
Legs               0
LandingPad        52
Block              0
ReusedCount        0
Serial             0
Longitude          0
Latitude           0
dtype: int64
```

Before we can continue we must deal with these missing values. The `LandingPad` column will retain None values to represent when landing pads were not used.

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` values in the data with the mean you calculated.

```
# Calculate the mean value of PayloadMass column
mean_payload_mass = data_falcon9['PayloadMass'].mean()

# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'] = data_falcon9['PayloadMass'].replace(np.nan, mean_payload_mass)
data_falcon9.isnull().sum()

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

Collecting data from Wikipedia

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

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

Create a `BeautifulSoup` object from the HTML `response`

```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(response.text, "html5lib")
```

Print the page title to verify if the `BeautifulSoup` object was created properly

```
# Use soup.title attribute
soup.title
```

```
<title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

Next, we just need to iterate through the `<th>` elements and apply the provided `extract_column_from_header()` to extract column name one by one

```
column_names = []

# Apply find_all() function with `th` element on first_launch_table
# Iterate each th element and apply the provided extract_column_from_header() to get a column name
# Append the Non-empty column name (`if name is not None and len(name) > 0`) into a list called column_names

for th in first_launch_table.find_all('th'):
    name = extract_column_from_header(th)
    if name is not None and len(name) > 0 :
        column_names.append(name)
```

Check the extracted column names

```
print(column_names)
```

```
['Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome']
```

We will create an empty dictionary with keys from the extracted column names in the previous task. Later, this dictionary will be converted into a Pandas dataframe

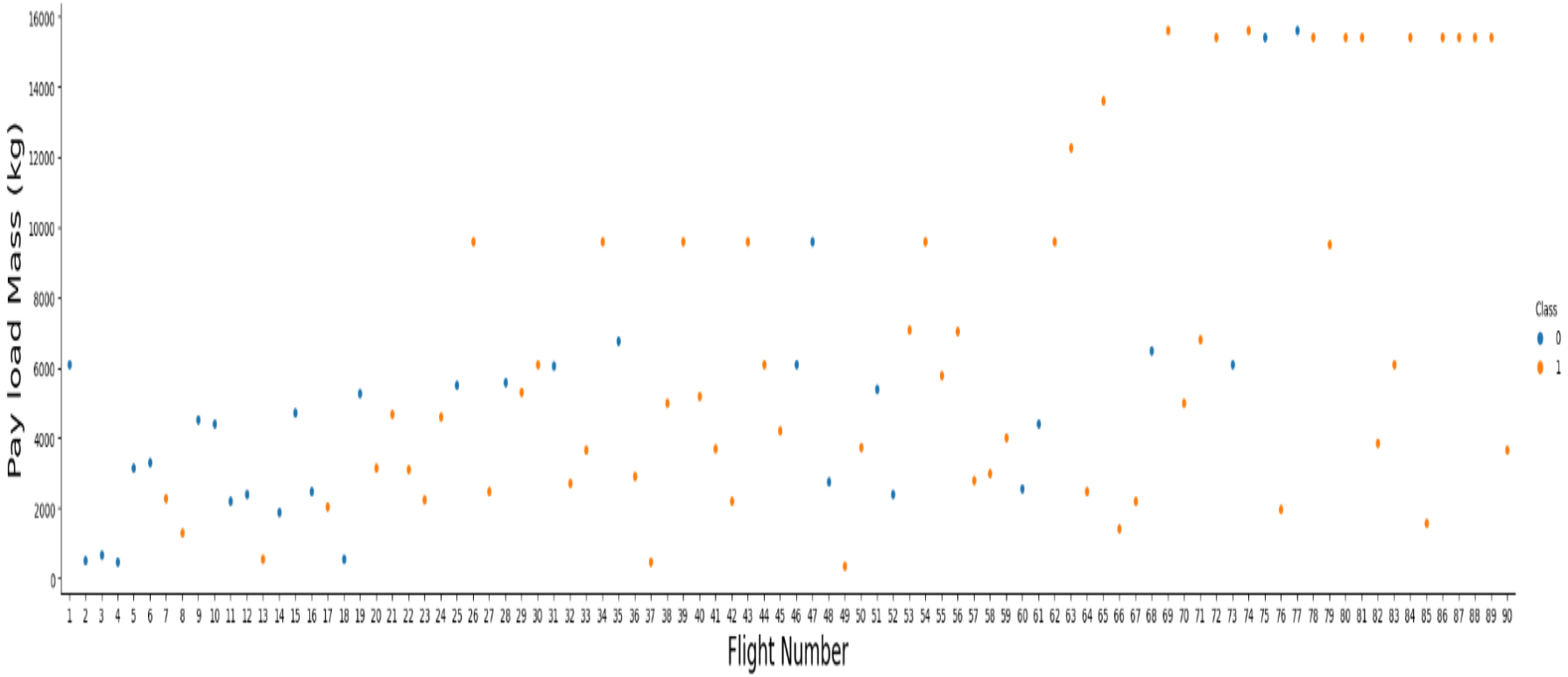
```
launch_dict= dict.fromkeys(column_names)

# 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']=[]
```

	Flight No.	Launch site	Payload	Payload mass	Orbit	Customer	Launch outcome	Version Booster	Booster landing	Date	Time
0	1	CCAFS	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success\n	F9 v1.0B0003.1	Failure	4 June 2010	18:45
1	2	CCAFS	Dragon	0	LEO	.mw-parser-output .plainlist ol,.mw-parser-out...	Success	F9 v1.0B0004.1	Failure	8 December 2010	15:43
2	3	CCAFS	Dragon	525 kg	LEO	NASA (COTS)	Success	F9 v1.0B0005.1	No attempt\n	22 May 2012	07:44
3	4	CCAFS	SpaceX CRS-1	4,700 kg	LEO	NASA (CRS)	Success\n	F9 v1.0B0006.1	No attempt	8 October 2012	00:35
4	5	CCAFS	SpaceX CRS-2	4,877 kg	LEO	NASA (CRS)	Success\n	F9 v1.0B0007.1	No attempt\n	1 March 2013	15:10

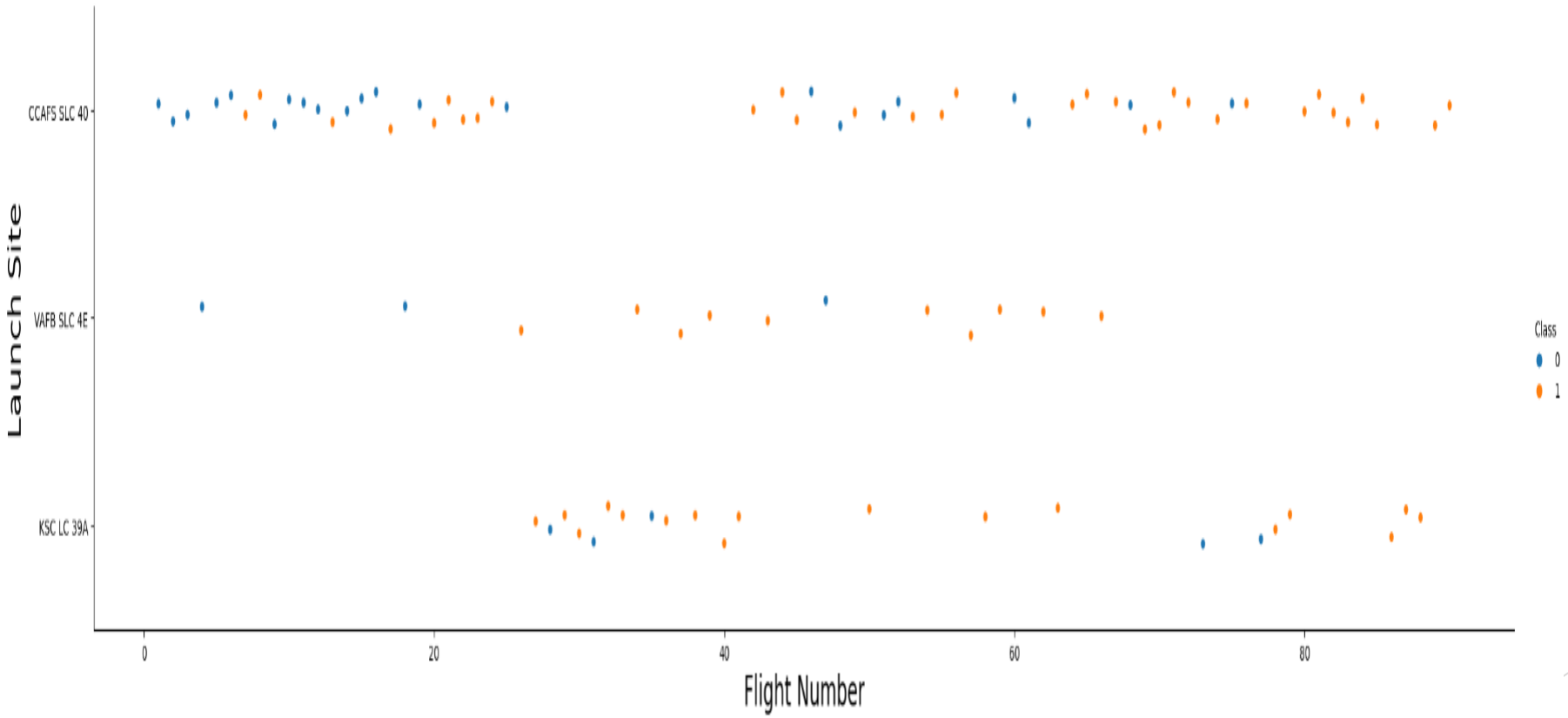
EDA with data visualization

- Bar graph: (for between Categorical data and numeric data)
  - Success rate for different rockets
  - Success rate vs orbit
- Scatter graphs: (for relationship between variables)
  - Payload and lunch site
  - Payload and orbit type
  - Payload and orbit
  - Payload mass and flight number
  - Flight number and lunch site
- Line graph:
  - Success rate and year

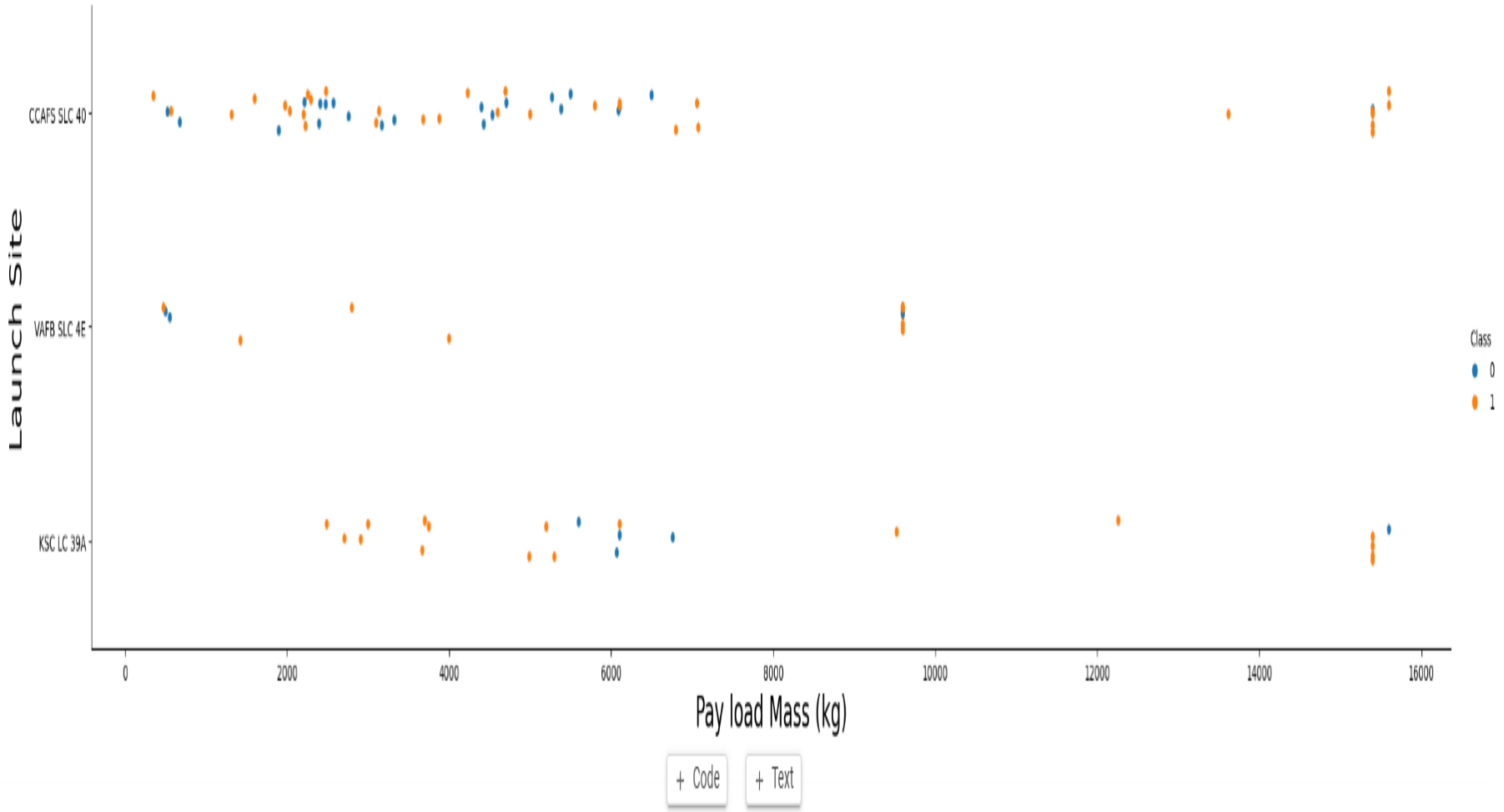
FlightNumber vs. PayloadMass



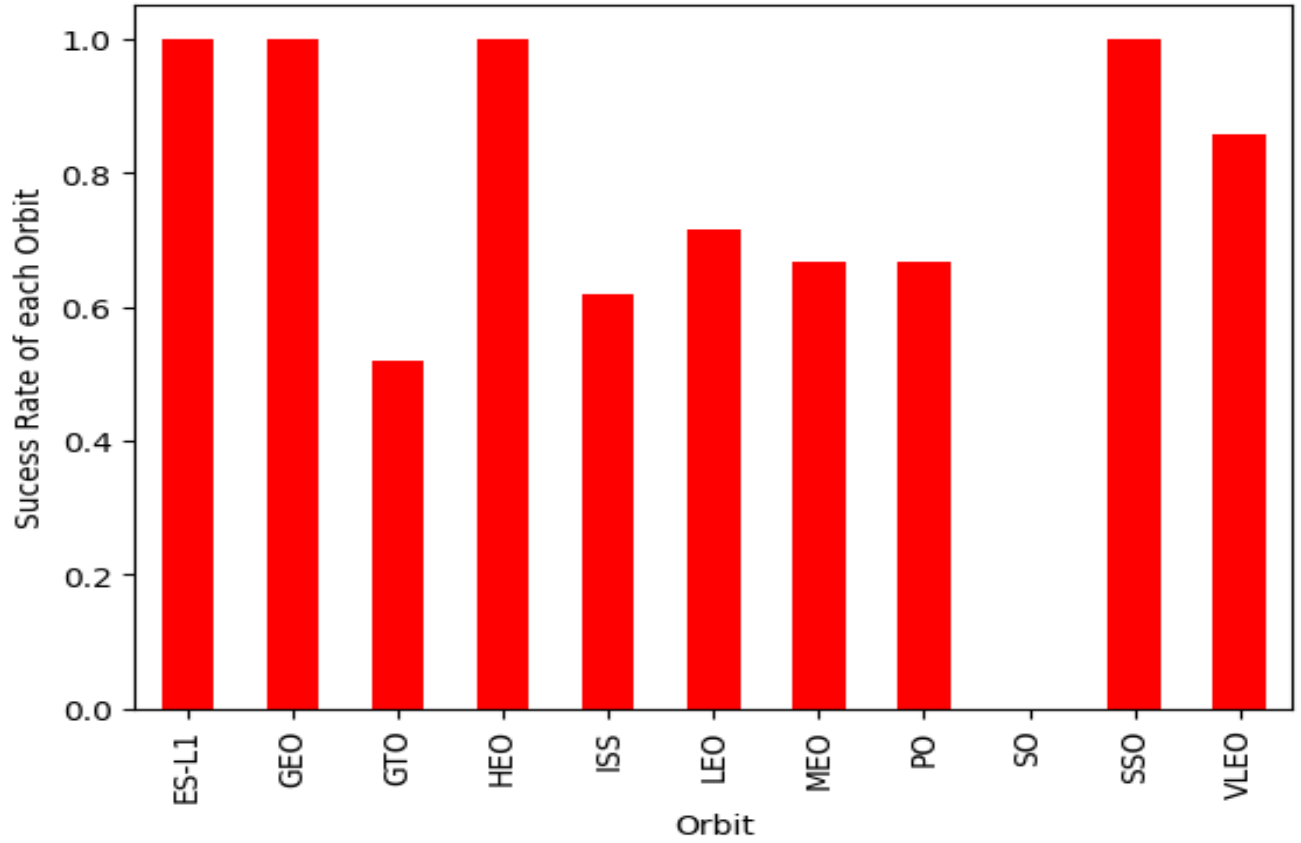
Flight Number and Launch Site



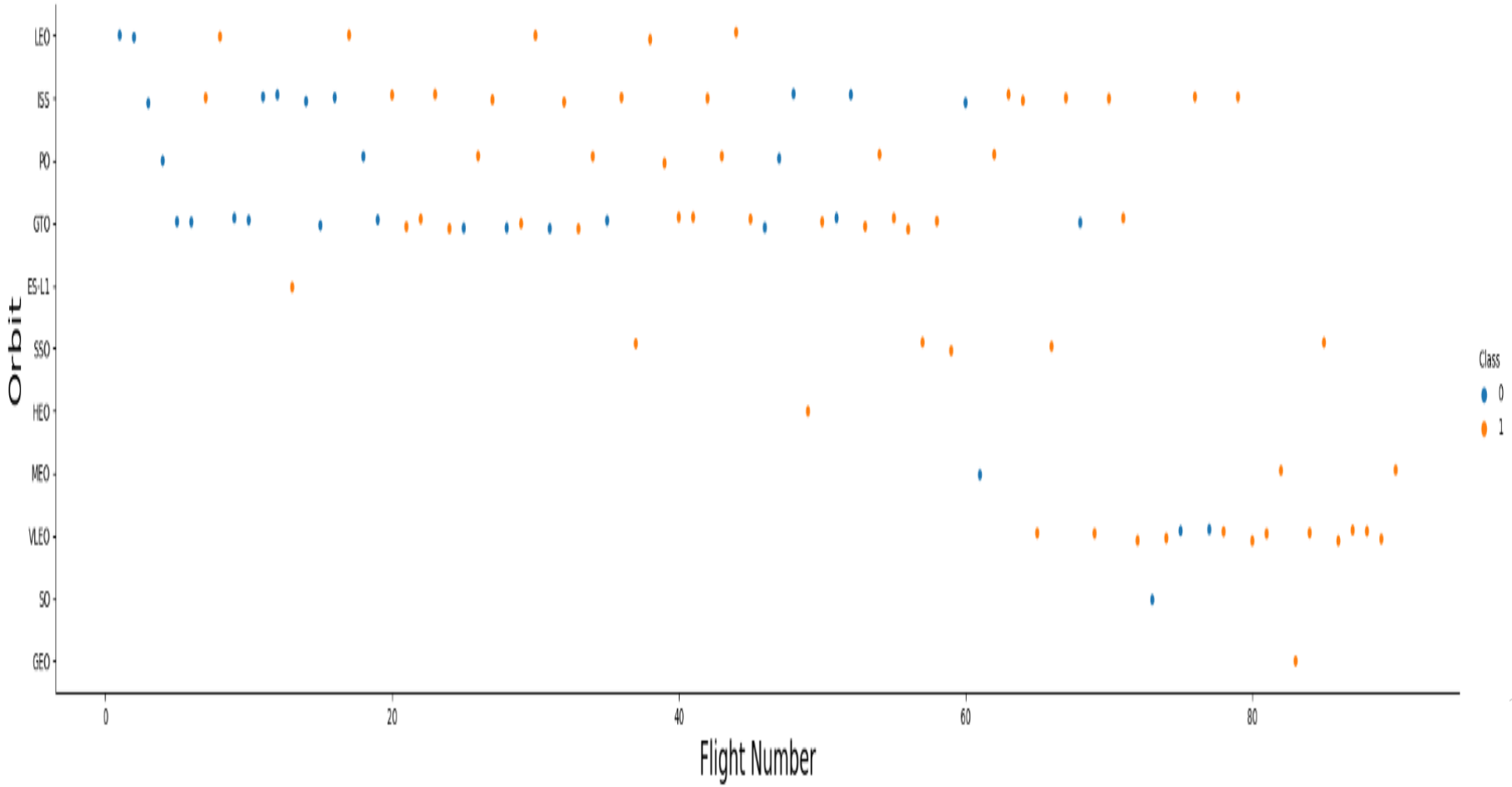
Payload and Launch Site



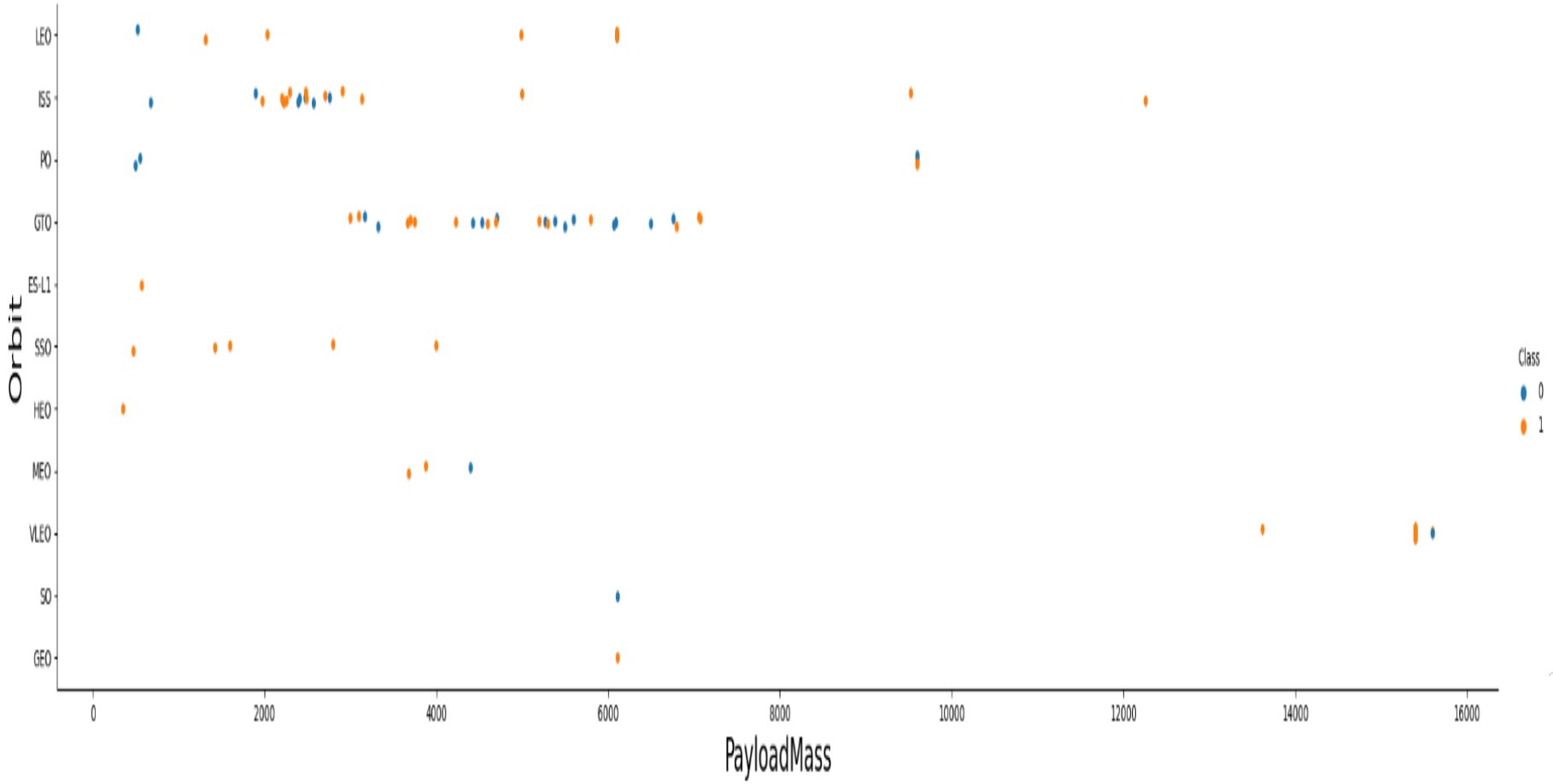
success rate of each orbit type



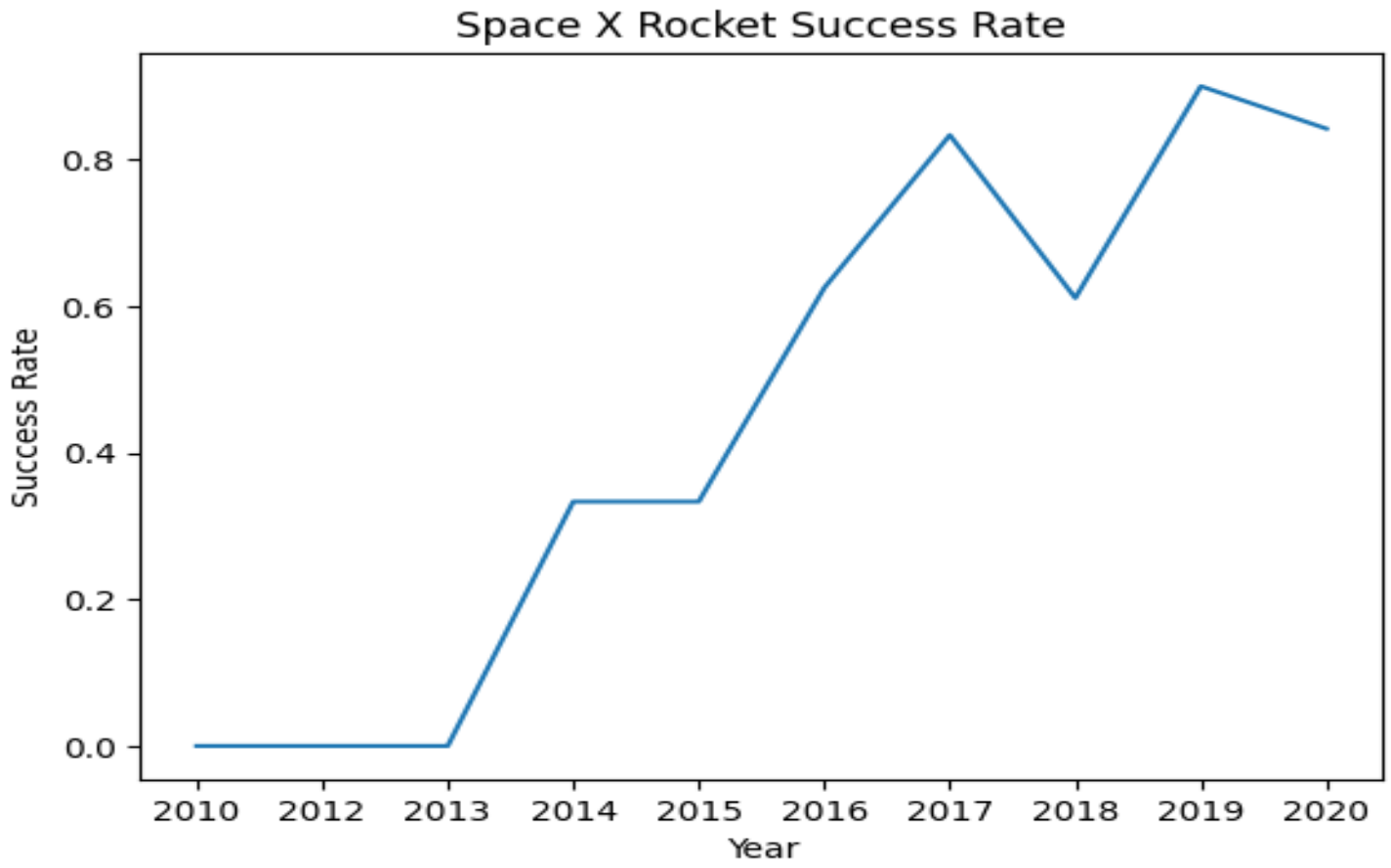
FlightNumber and Orbit type



Payload and Orbit type



launch success yearly trend



EDA with SQL

- Using SQL queries to find dataset better and understand datas
  - Show 5 records where lunch sites begin with ‘CCA’
  - show total payload mass carried by NASA ‘CRS’
  - Show average payload mass carried by F9 v1.1
  - Show first successful landing in ground
  - Show names of booster success in drone ship that have mass payload bigger than 4000 and lower than 6000
  - Show total number of successful and failed landings
  - And …

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

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS_KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
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

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

total payload mass carried by boosters launched by NASA (CRS)
45596.0

average payload mass carried by booster version F9 v1.1

average payload mass carried by booster version F9 v1.1
2534.6666666666665

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

- **22/12/2015**

succesful landing outcome in ground pad
22/12/2015
19/02/2017
18/07/2016
15/12/2017
14/08/2017
09/07/2017
06/03/2017
05/01/2017
01/08/2018

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

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

List the total number of successful and failure mission outcomes

SUCCESS	FAILURE
100	1

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

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

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

month	Landing_Outcome	Booster_Version	Launch_Site
10	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
04	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

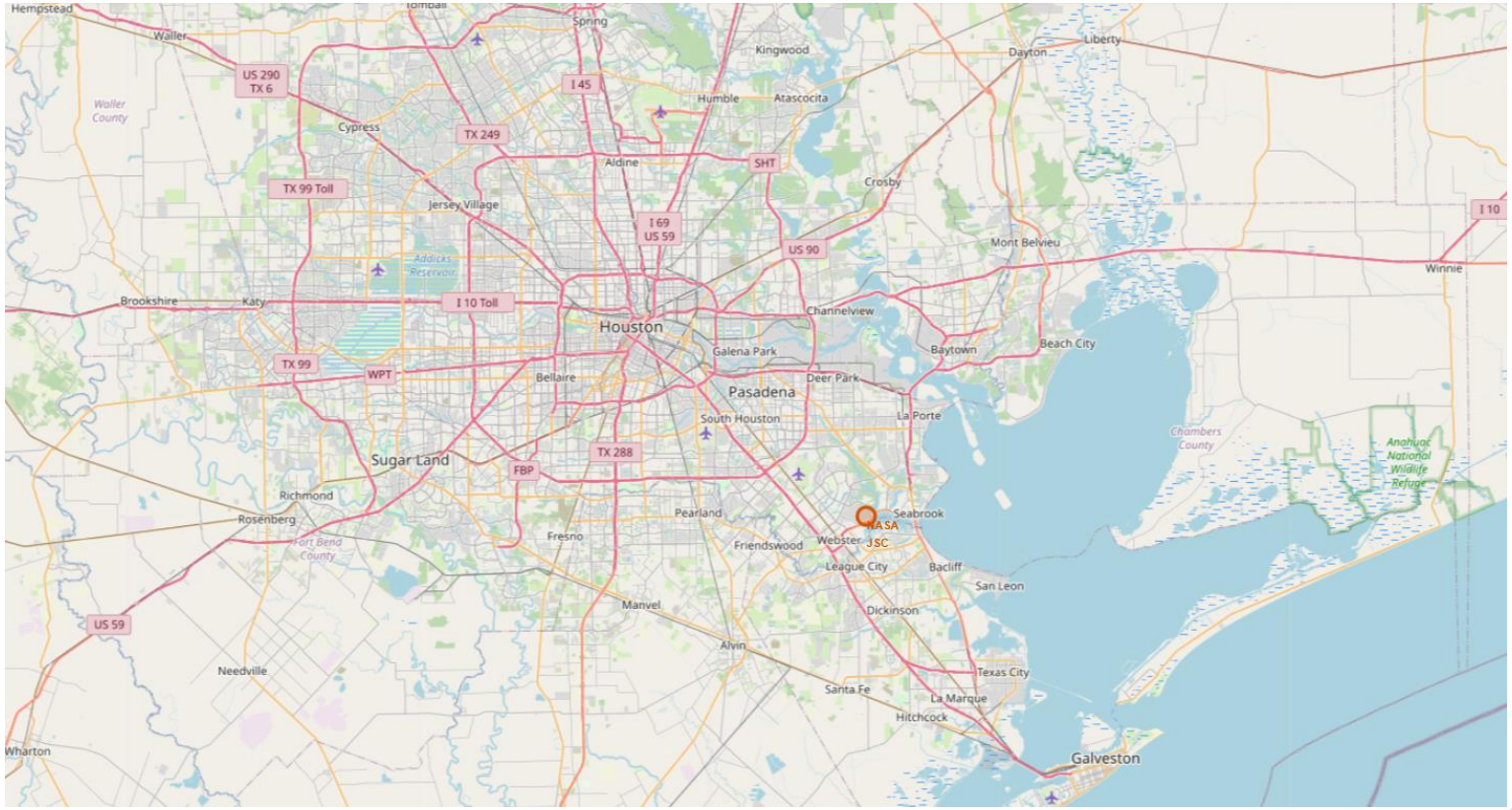
Rank the count of successful landing_outcomes between the date 04-06-2010 and 20-03-2017 in descending order.

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

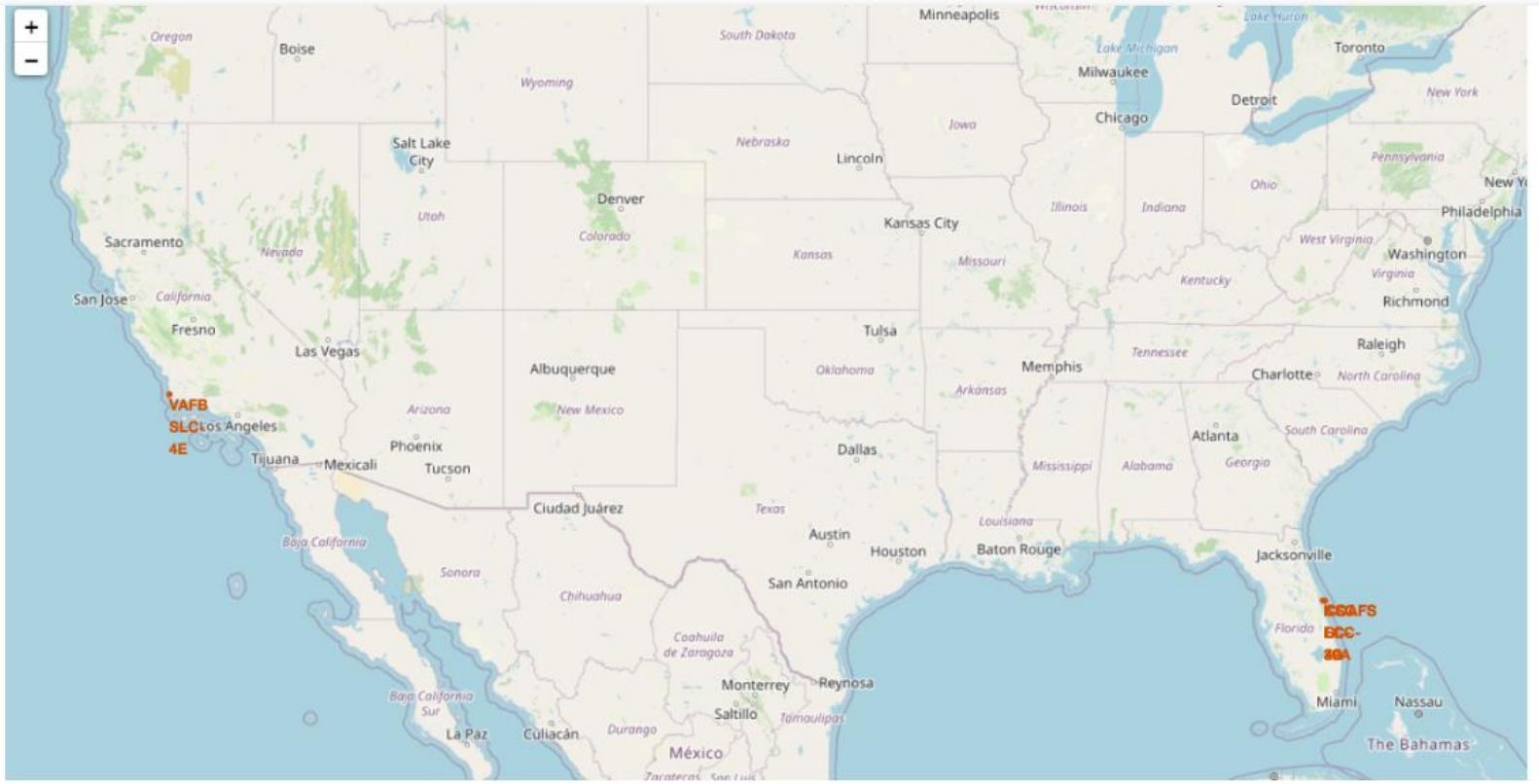
interactive map with Folium results

- Using folium map on NASA Johson space at Texas Houson
- `NASA coordinate = [29.559684888503615, -95.08309719307 59]`
  - Red circles on NASA Johnson space center's coordinate showing name
  - Red circles on each lunch site coordinate showing labels and lunch names
  - Markers shows successful and failed landings.
    - Green color for successful landings
    - Red color for failed landings
  - Markers shows distance between lunch site to locations(cities, highways …) and plot line between them

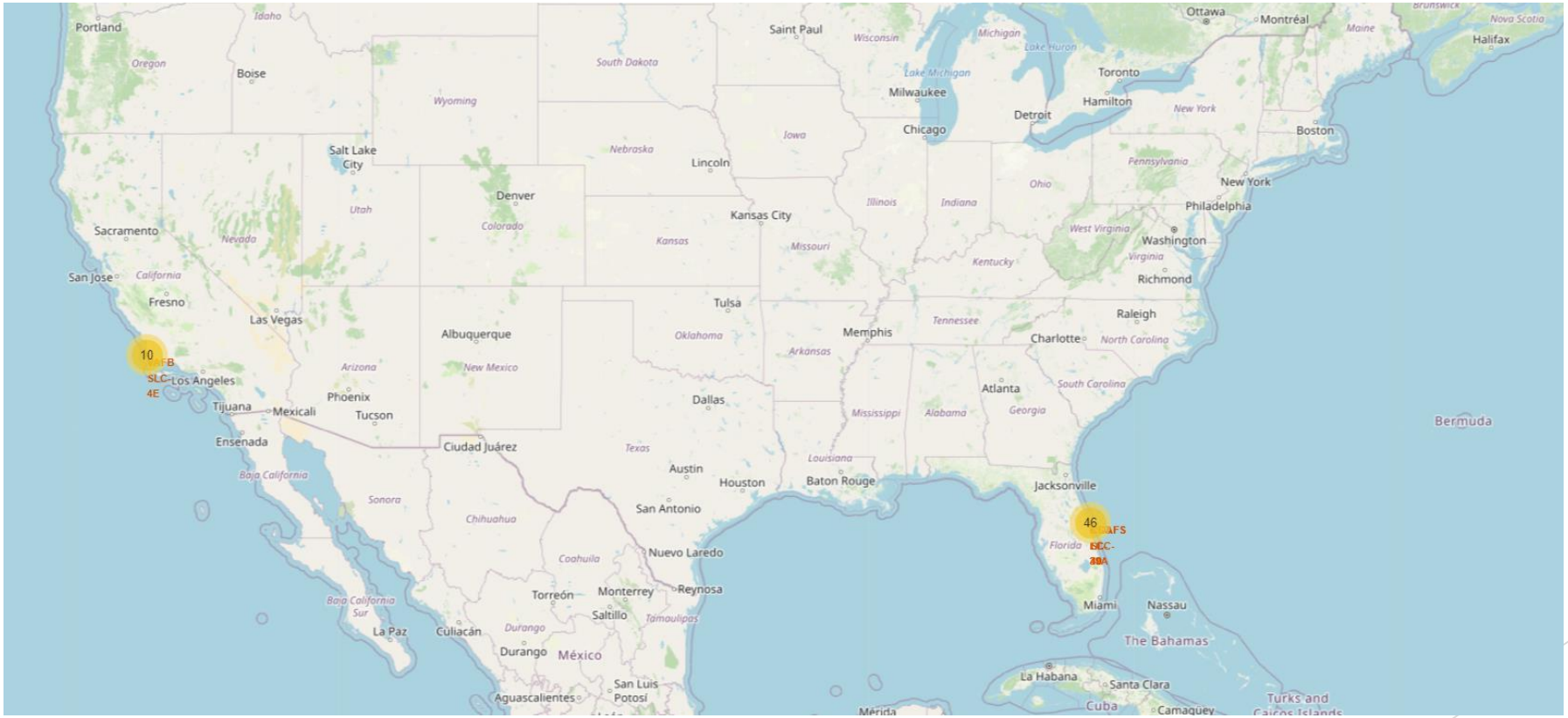
Mark all launch sites on a map



Mark all launch sites on a map

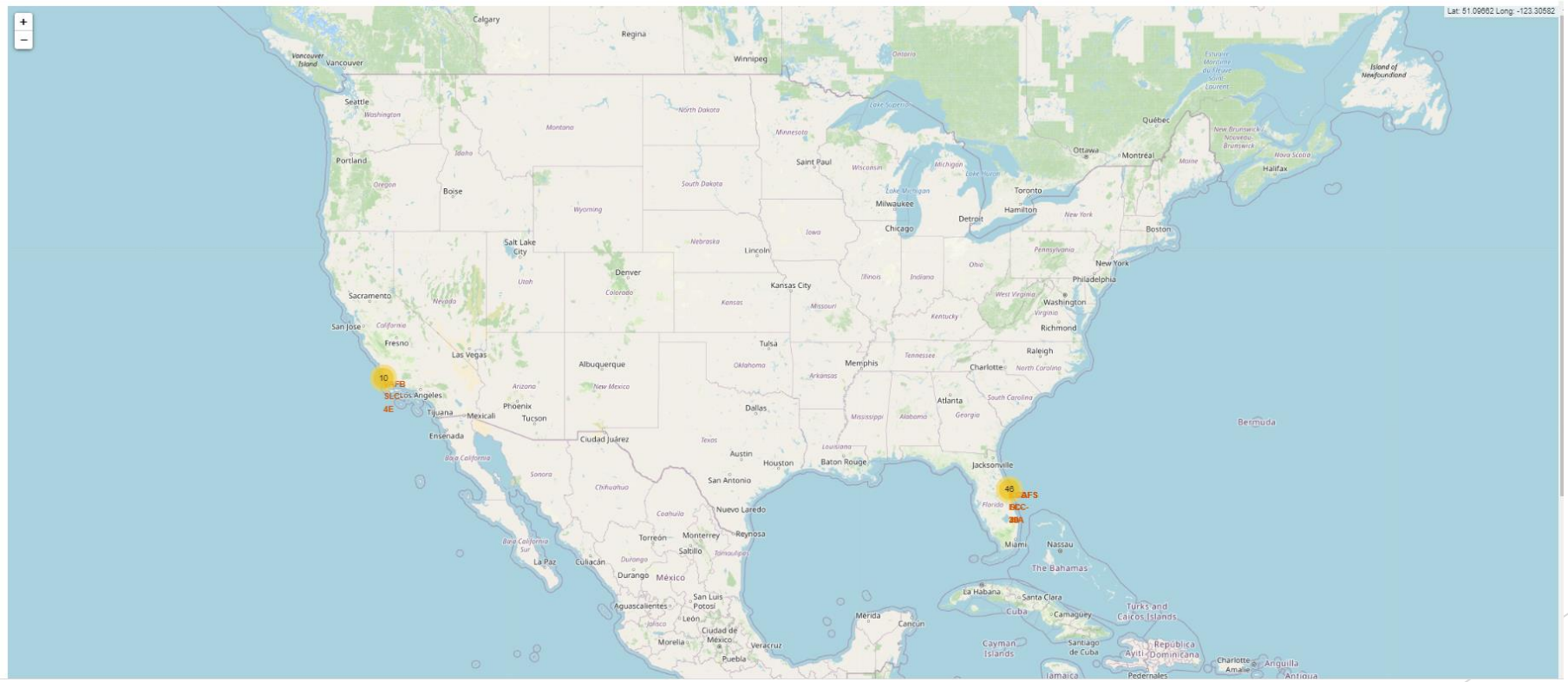


Mark the success/failed launches for each site on the map




CCAFS
SLC-
40
P
P
26
CAFS
LC-
40

Distances


Lat: 51.09662 Long: -123.30582
Calgary
Regina
Winnipeg
Vancouver
Seattle
Portland
Boise
Saint Paul
Québec
Ottawa
Montréal
Halifax
Toronto
Milwaukee
Hamilton
Detroit
Chicago
Boston
Salt Lake City
Denver
Lincoln
New York
Philadelphia
Sacramento
Kansas City
Washington
Richmond
San Jose
Fresno
Tulsa
Las Vegas
Albuquerque
Memphis
Raleigh
Charlotte
Los Angeles
Phoenix
Tijuana
Mexicali
Tucson
Dallas
Atlanta
Ensenada
Ciudad Juárez
Austin
Houston
Baton Rouge
Jacksonville
Bermuda
San Antonio
Nuevo Laredo
Torreón
Monterrey
Reynosa
Miami
Nassau
La Paz
Culiacán
Saltillo
The Bahamas
Durango
México
La Habana
Santa Clara
Aguascalientes
San Luis Potosí
Cuba
Camagüey
Turks and Caicos Islands
Mérida
Cancún
León
Ciudad de México
Morelia
Veracruz
Puebla
Cayman Islands
Santiago de Cuba
República Dominicana
Haití
Jamaica
Pedernales
Charlotte Amalie
Anguilla
Antigua
10
FB
SLC
4E
46
AFS
ECC
20A


P
P
P
7
CCAFS
SLC-
40
26
CCAFS
LC-
40
0.90
KM
Samuel C Phillips Pkwy
Centaur Raod

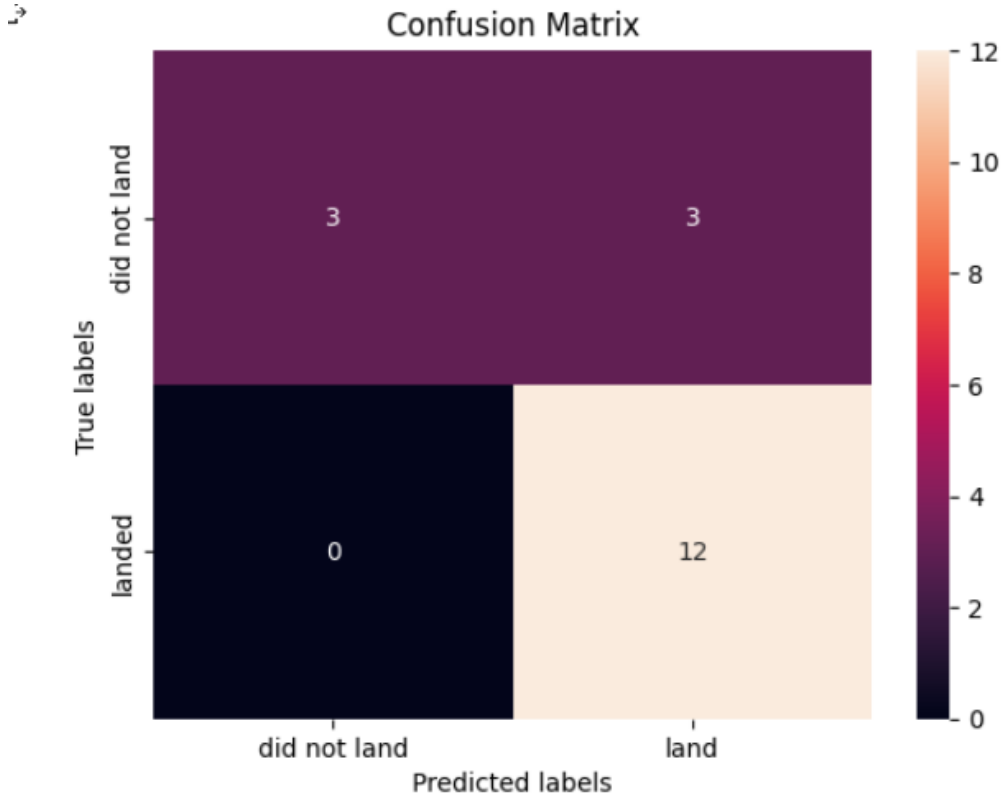
Plotly Dash Dashboard results slides

- Using Dashboard dropdown and pie chart, scatter plots, rangeslider
  - Dropdown give user to choose lunch sites or all lunch sites
  - Pie chart show total success and total failed for the lunch sites with dropdown options (plotly.express.pie)
  - Scatter plot show relationship between variables, like success and payload mass, like failes and payload mass (plotly.express.scatter)
  - Rangeslider give user to selecet payload mass in range

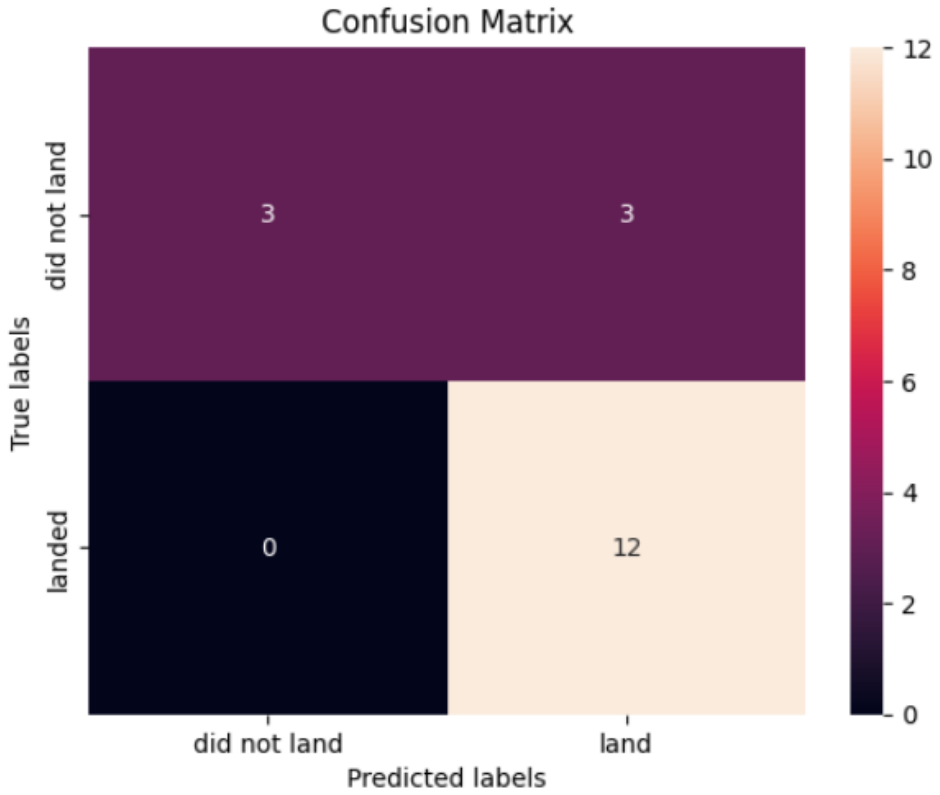
predictive analysis (classification)

- Data preparation:
  - Loading data
  - Normalize
  - And split to training and testing samples
- Model Preparation:
  - Choosing machine learning algorithms
  - GridSearchCV and giving parameters to algorithms
- Model evaluation:
  - Finding best hyperparameters for each models
  - Compare accuracy with testing data
- Model comparing:
  - Compare models with their accuracy
  - Choosing exact accuracy for model

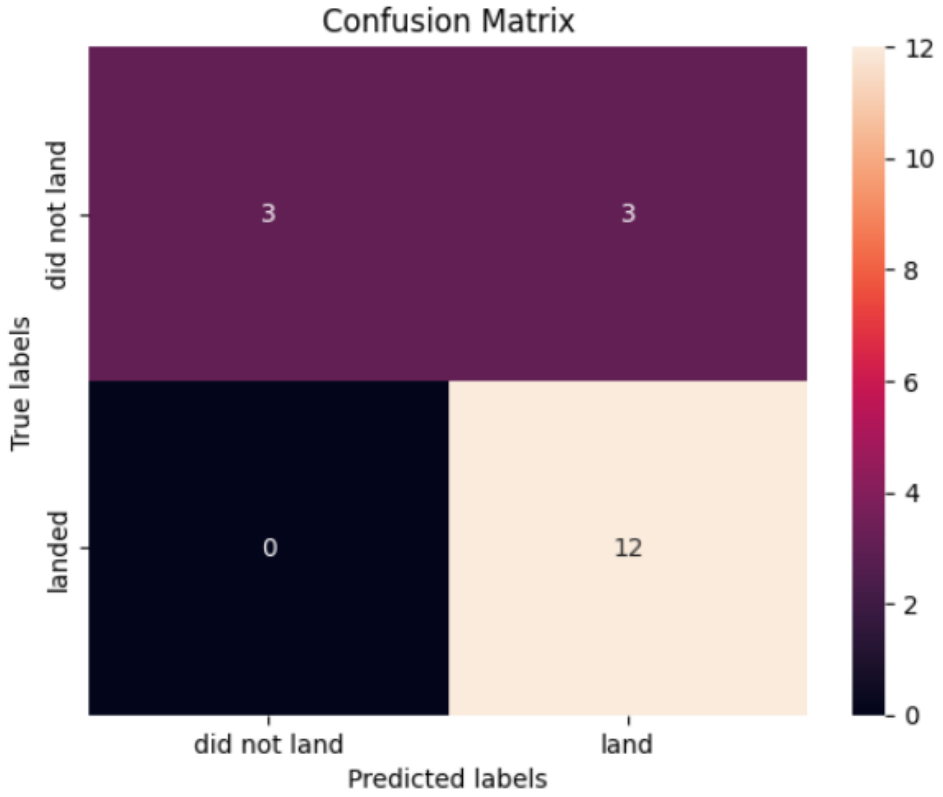
Logistic regression



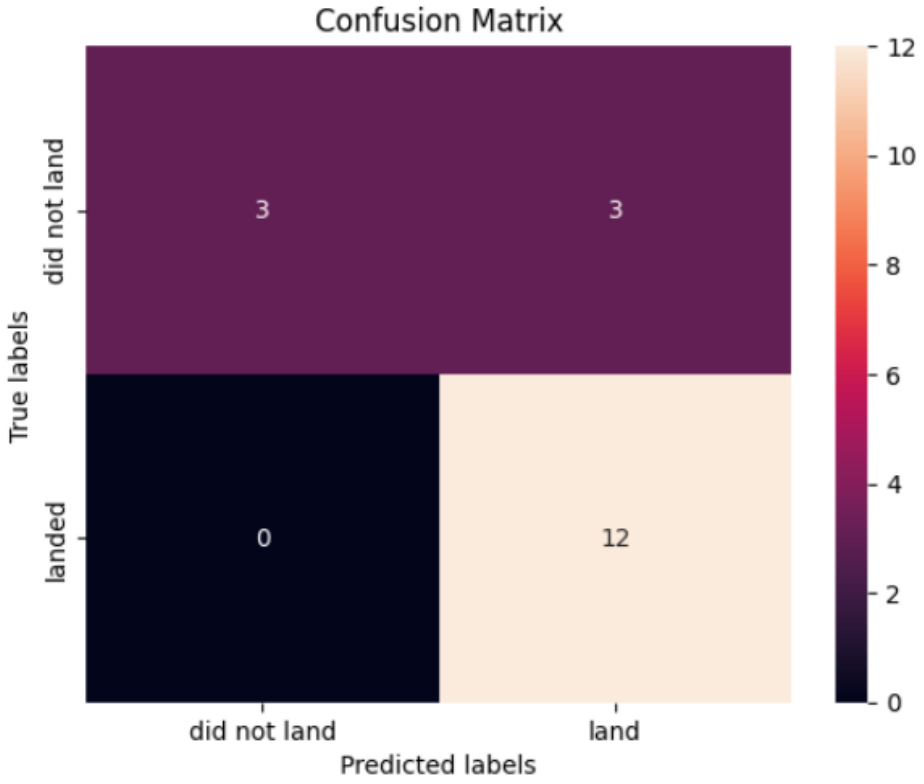
Decision Tree

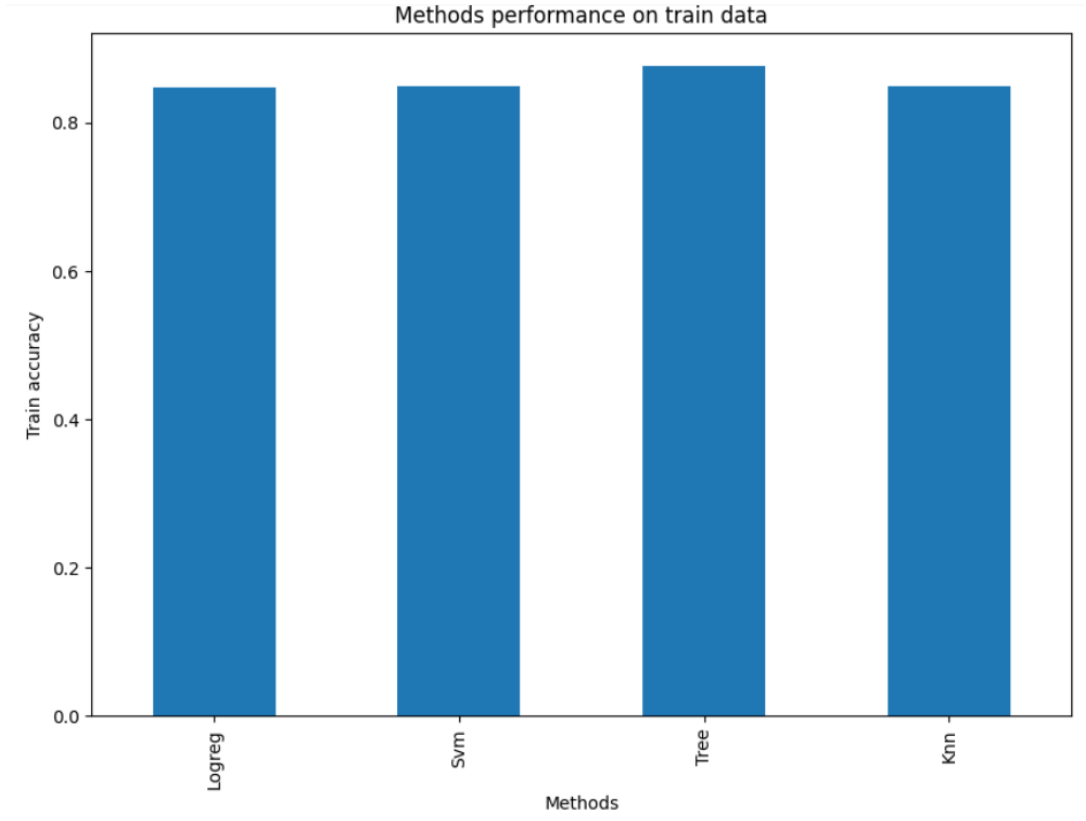


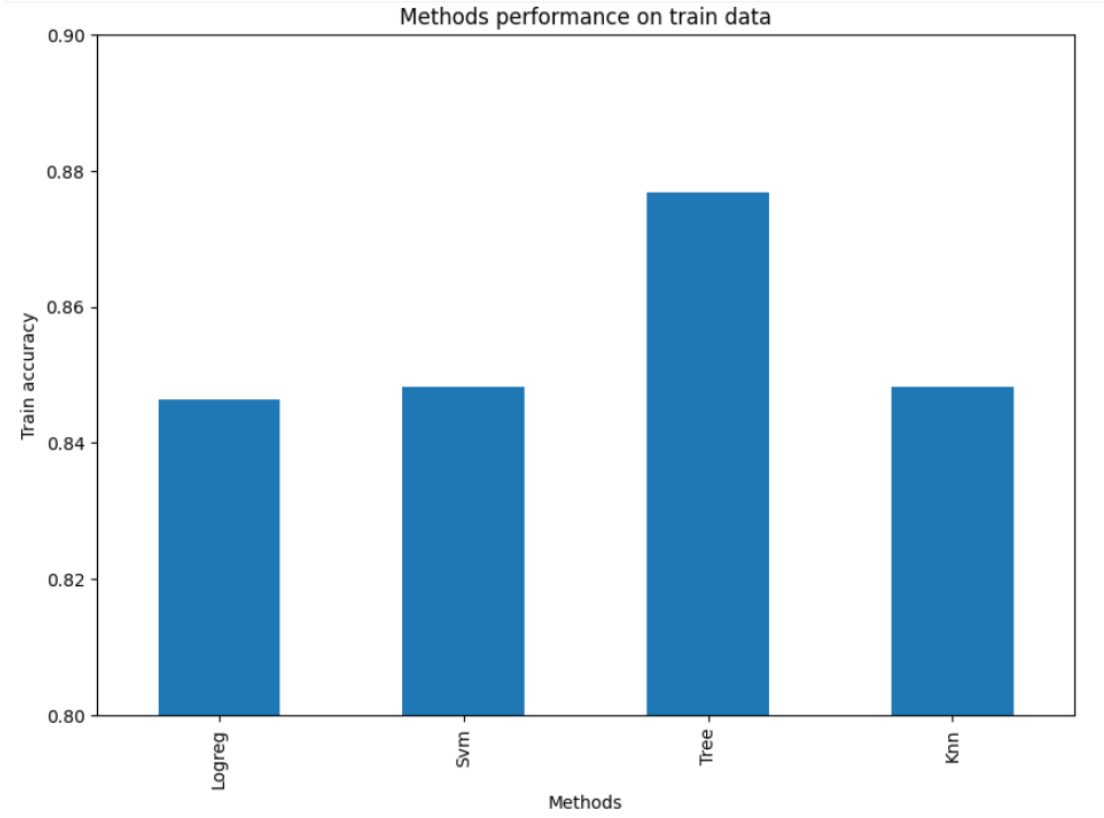
kNN

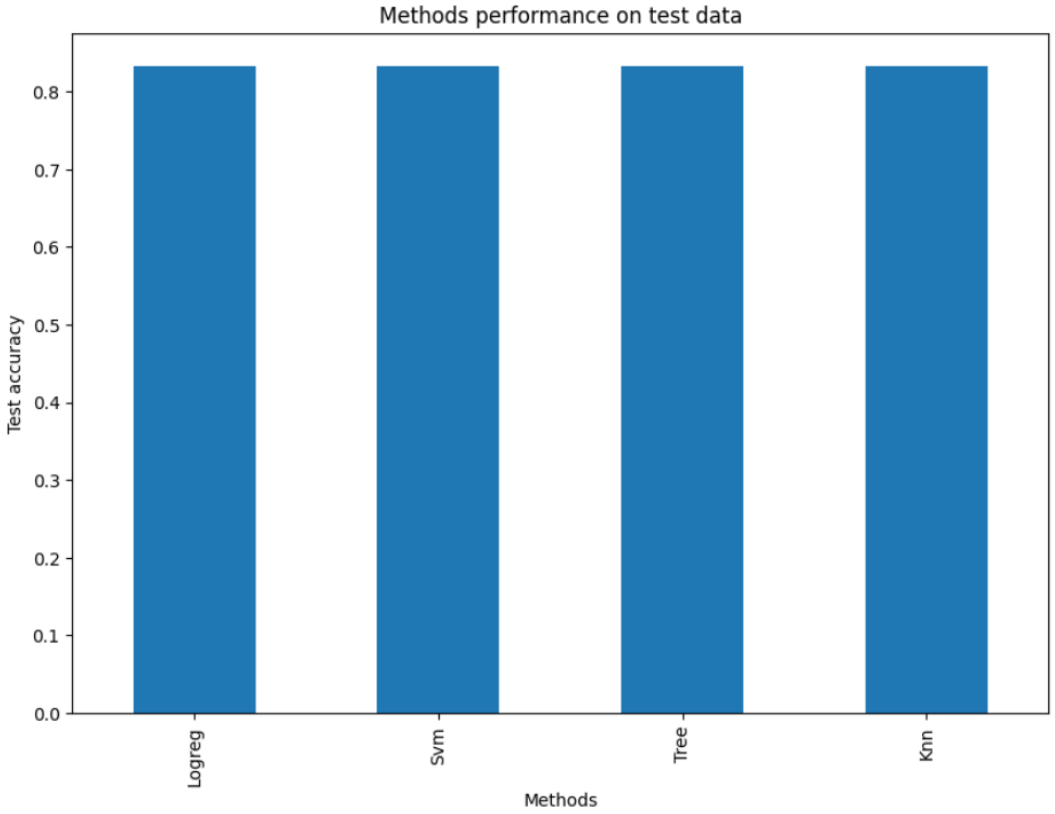

Confusion Matrix
True labels
did not land
landed
3
3
0
12
did not land
land
Predicted labels
12
10
8
6
4
2
0

SVM


Confusion Matrix
3
3
0
12
did not land
landed
True labels
did not land
land
Predicted labels
12
10
8
6
4
2
0


Methods performance on train data
Train accuracy
0.0
0.2
0.4
0.6
0.8
Logreg
Svm
Tree
Knn
Methods


Methods performance on train data
0.90
0.88
0.86
0.84
0.82
0.80
Train accuracy
Logreg
Svm
Tree
Knn
Methods

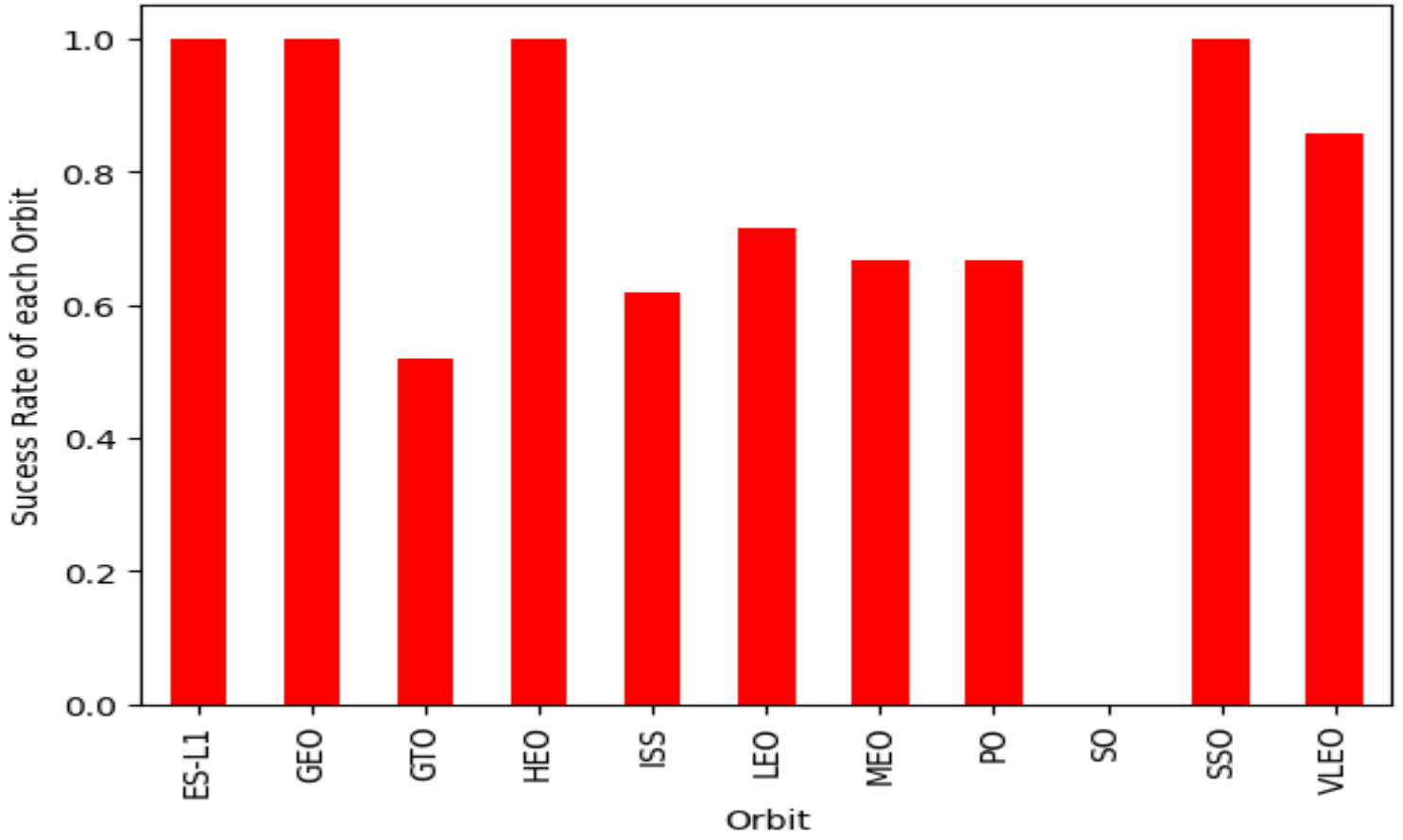

Methods performance on test data
Test accuracy
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Logreg
Svm
Tree
Knn
Methods

Conclusion

- All codes are in github.com .ipynb file

improve the presentation beyond the template

- **Visualize the relationship between success rate of each orbit type**



- **dummy variables to categorical columns**

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	ES-L1	GEO	...	B1048	B1049	B1050	B1051	B1054	B1056	B1058	B1059	B1060	B1062
0	1	6104.959412	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	2	525.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	3	677.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	4	500.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	5	3170.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 80 columns

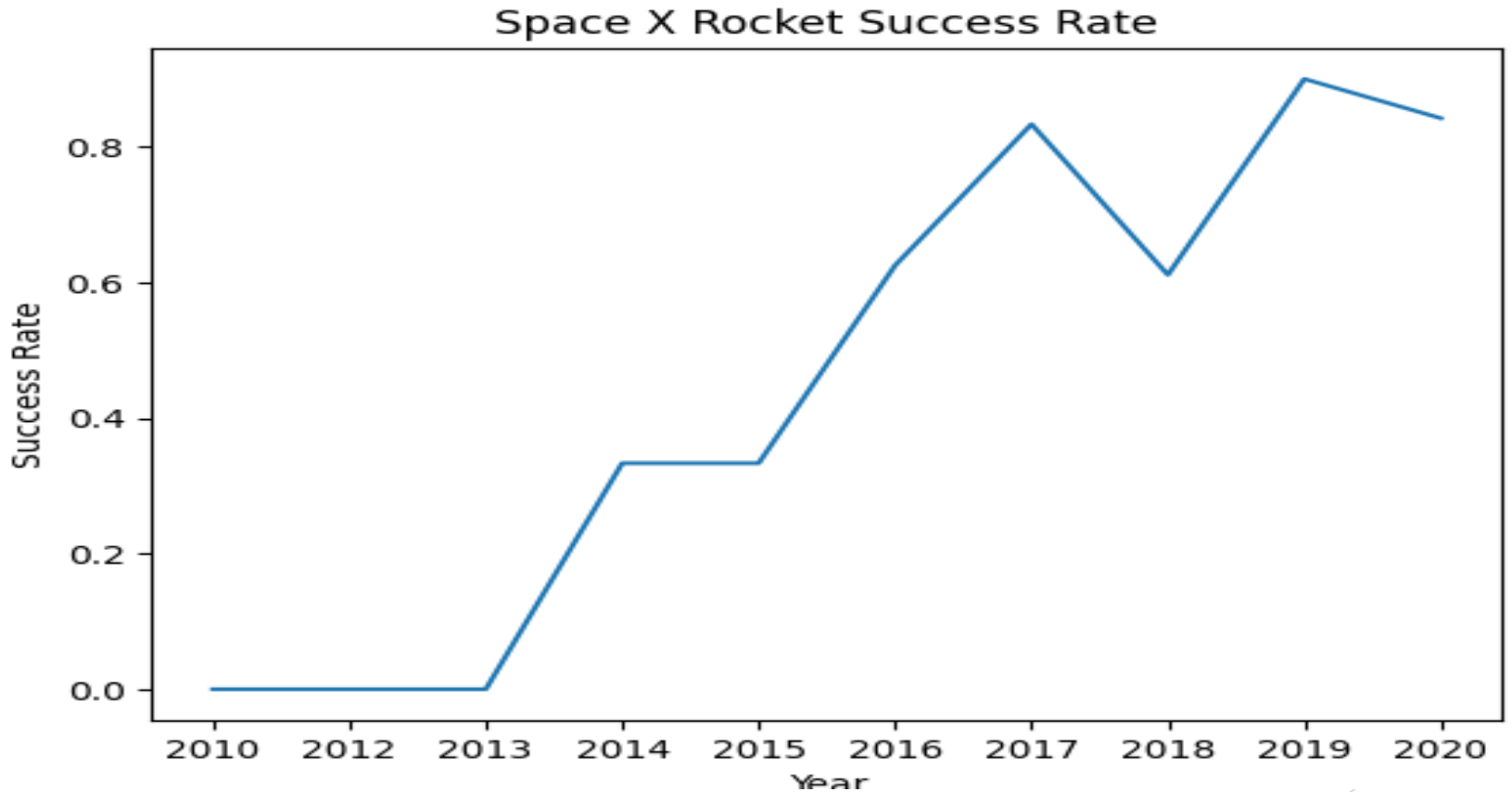
Displayed any innovative insights

- all numeric columns to float64

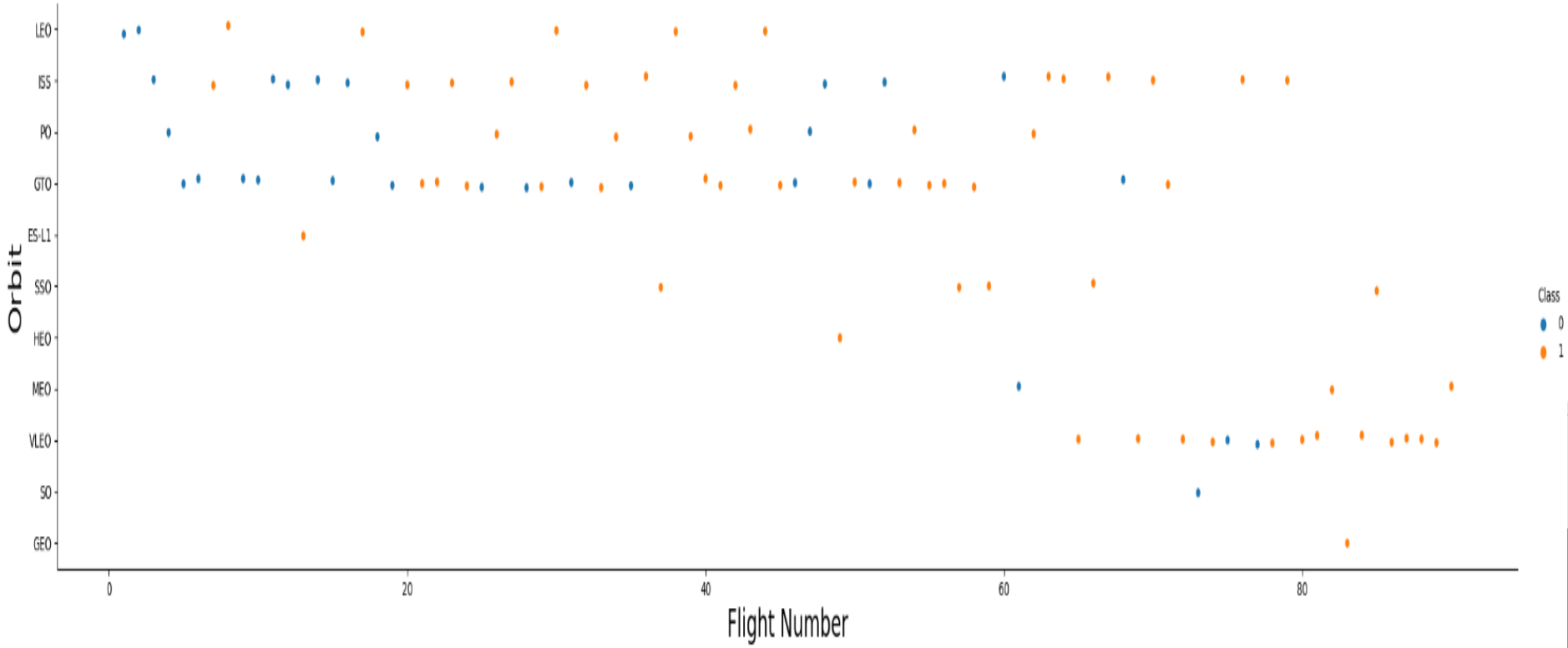
	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	ES-L1	GEO	...	B1048	B1049	B1050	B1051	B1054	B1056	B1058	B1059	B1060	B1062
0	1.0	6104.959412	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	2.0	525.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	3.0	677.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	4.0	500.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	5.0	3170.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
85	86.0	15400.000000	2.0	1.0	1.0	1.0	5.0	2.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
86	87.0	15400.000000	3.0	1.0	1.0	1.0	5.0	2.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
87	88.0	15400.000000	6.0	1.0	1.0	1.0	5.0	5.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
88	89.0	15400.000000	3.0	1.0	1.0	1.0	5.0	2.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
89	90.0	3681.000000	1.0	1.0	0.0	1.0	5.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

90 rows × 80 columns

- Space X Rocket Success Rate



- **Visualize the relationship between FlightNumber and Orbit type**



- **Visualize the relationship between Payload and Orbit type**