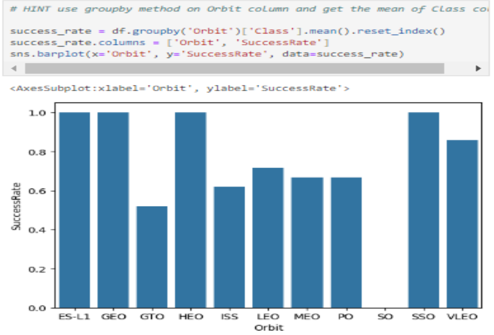
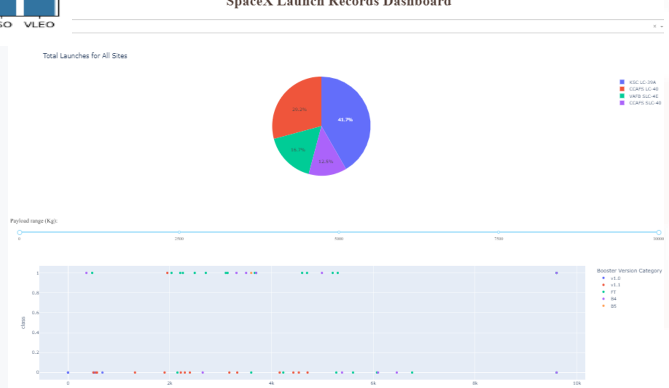
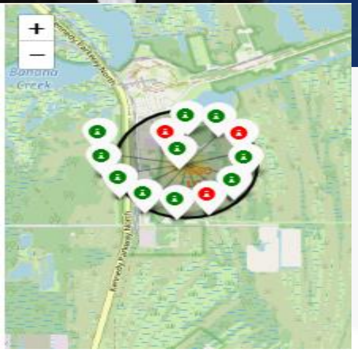
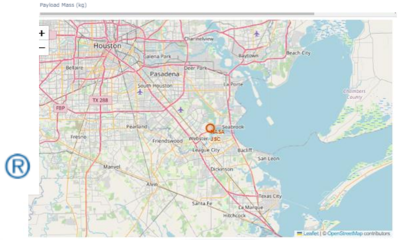






IBM Data Science Capstone Project – Space X

Silvia Calzada Gil

Sep 2024

https://github.com/Statninja/IBM-DataScience-SpaceX-Capstone/tree/main

IBM Developer

SKILLS NETWORK

OUTLINE

- Executive Summary
- Introduction
- Methodology
- Results
  - EDA Visualization – Charts
  - EDA with SQL
  - Interactive Maps with Folium
  - Plotly Dash Dashboard
  - Predictive Analytics
  - Summary of Findings
- Conclusion

EXECUTIVE SUMMARY

Summary of methodologies

- Data collection
- Data wrangling
- Exploratory Data Analysis (EDA) with data visualization
- EDA with SQL
- Building an interactive map with Folium
- Building a Dashboard with Plotly Dash
- Predictive analysis (Classification)

INTRODUCTION

- The project focuses on predicting the success of the Falcon 9 first stage landings by SpaceX. Falcon 9 launches are priced at $62 million, significantly lower than competitors, who charge over $165 million per launch. This cost advantage is largely due to SpaceX’s ability to reuse the first stage of the rocket. By accurately predicting whether the first stage will successfully land, the project can provide valuable insights into launch costs, potentially aiding competitors in forming bids against SpaceX.

Key Objectives

- Analyze how payload mass, launch site, number of flights, and orbits impact first-stage landing success.
- Examine the rate of successful landings over time.
- Determine the best predictive model for landing success.

METHODOLOGY Details

- **Data Collection**:
  - Use the SpaceX REST API and web scraping to gather data on rocket launches.
- **Data Wrangling**:
  - Clean and prepare the data, creating a variable that indicates whether each launch was successful or failed.
- **Exploratory Data Analysis**:
  - Visualize and explore data considering factors like payload, launch site, flight number, and yearly trends.
- **SQL Analysis**:
  - Calculate key statistics:
    - Total payload.
    - Payload range for successful launches.
    - Total number of successful and failed launches.
- **Launch Site Analysis**:
  - Examine the success rates of different launch sites and their proximity to important geographical markers.
- **Data Visualization**:
  - Visualize:
    - Launch sites with the highest success rates.
    - Payload ranges for successful launches.
- **Modeling**:
  - Build models to predict landing outcomes using:
    - Logistic Regression.
    - Support Vector Machine (SVM).
    - Decision Tree.
    - K-Nearest Neighbor (KNN).

METHODOLOGY & Results of Main Tasks

- Objectives
- Perform Exploratory Data Analysis (EDA).
- Determine training labels for predictive modeling.
- Data Analysis Steps
- **Import Libraries:**
  - Imported essential libraries and more:
    - **Pandas** for data manipulation and analysis.
    - **NumPy** for numerical operations.
- **Request to the SpaceX API:**
  - response = requests.get("https://api.spacexdata.com/v4/rockets/"+str(x)).json()
- **Missing Values Analysis:**
    - Most attributes had 0% missing values, except for `LandingPad (40%) -` Hands-on Lab: Data Wrangling
- **Data Types Identification:**
  - Identified numerical and categorical columns using `dtypes` method to understand the data structure (int65, object,float64, bool)

METHODOLOGY & Results of Main Tasks

- Data Analysis Steps
- **Launch Site Analysis:**
  - Used the `value_counts()` method on the `LaunchSite` column to calculate the number of launches at each site:
    - Cape Canaveral Space Launch Complex 40: 55 launches
    - Kennedy Space Center Launch Complex 39A: 22 launches
    - Vandenberg Air Force Base Space Launch Complex 4E: 13 launches
- **Orbit Type Analysis:**
  - Analyzed orbit types using `value_counts()` to determine the occurrence of each orbit:
    - GTO: 30%
    - ISS: 23.33%
    - VLEO: 15.56%
    - Others: Varying percentages
- **Mission Outcome Analysis:**
  - Analyzed mission outcomes using `value_counts()` on the `Outcome` column to categorize success and failure types.
  - Defined a set of bad outcomes representing unsuccessful landings.
- **Creating Landing Outcome Labels:**
  - Created a binary classification variable, `landing_class`, to label the landing outcome:
    - 0: Unsuccessful landing
    - 1: Successful landing
- **Success Rate Calculation:**
  - Calculated the success rate of landings by averaging the `Class` column:
    - Success rate: 66.67%
- **Data Export:**
  - Exported the modified dataset with the new `Class` column to a CSV file for future analysis.

Data Collection - API

Request SpaceX API.

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

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

Check the content of the response

```
print(response.content)
```

```
response.status_code
```

```
200
```

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Out
0	1	2006-03-24	Falcon 1	20.0	LEO	Kwajalein Atoll	
1	2	2007-03-21	Falcon 1	NaN	LEO	Kwajalein Atoll	
2	4	2008-09-28	Falcon 1	165.0	LEO	Kwajalein Atoll	
3	5	2009-07-13	Falcon 1	200.0	LEO	Kwajalein Atoll	
4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	

```
# Show the head of the dataframe
df.info()
df.describe()
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 94 entries, 0 to 93
Data columns (total 17 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   FlightNumber    94 non-null     int64
 1   Date            94 non-null     object
 2   BoosterVersion  94 non-null     object
 3   PayloadMass     88 non-null     float64
 4   Orbit           94 non-null     object
 5   LaunchSite      94 non-null     object
 6   Outcome         94 non-null     object
 7   Flights         94 non-null     int64
 8   GridFins        94 non-null     bool
 9   Reused          94 non-null     bool
 10  Legs            94 non-null     bool
 11  LandingPad      64 non-null     object
 12  Block           90 non-null     float64
 13  ReusedCount     94 non-null     int64
 14  Serial          94 non-null     object
 15  Longitude       94 non-null     float64
 16  Latitude        94 non-null     float64
dtypes: bool(3), float64(4), int64(3), object(7)
memory usage: 10.7+ KB
```

Data wrangling and formatting.

https://github.com/Statninja/IBM-DataScience-SpaceX-Capstone/tree/main/Complete%20the%20Data%20Collection%20API%20Lab

launches. Filter the data dataframe using the BoosterVers to only keep the Falcon 9 launches. Save the filtered data to dataframe called `data_falcon9`.

```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = df[df['BoosterVersion'] == 'Falcon 9']
```

	FlightNumber	Date	BoosterVersion	PayloadMa
4	1	2010-06-04	Falcon 9	N
5	2	2012-05-22	Falcon 9	52
6	3	2013-03-01	Falcon 9	67
7	4	2013-09-29	Falcon 9	50
8	5	2013-12-03	Falcon 9	317
...	...	...	...	
89	86	2020-09-03	Falcon 9	1560
90	87	2020-10-06	Falcon 9	1560
91	88	2020-10-18	Falcon 9	1560
92	89	2020-10-24	Falcon 9	1560
93	90	2020-11-05	Falcon 9	368

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

```
FlightNumber       0
Date               0
BoosterVersion     0
PayloadMass        5
Orbit              0
LaunchSite         0
Outcome            0
Flights            0
GridFins           0
Reused             0
Legs               0
LandingPad        26
Block              0
ReusedCount        0
Serial             0
Longitude          0
Latitude           0
dtype: int64
```

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

```
# Replace the np.nan values with its mean value
data_falcon9.loc[data_falcon9['PayloadMass'].isnull(),'PayloadMass']= mean_
```

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

```
FlightNumber       0
Date               0
BoosterVersion     0
PayloadMass        0
Orbit              0
LaunchSite         0
Outcome            0
Flights            0
GridFins           0
Reused             0
Legs               0
LandingPad        26
Block              0
ReusedCount        0
Serial             0
Longitude          0
Latitude           0
dtype: int64
```

Performing web scraping to collect Falcon 9 historical launch records from a Wikipedia page

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

```
# use requests.get() method with the provided static_url
response = requests.get(static_url)
```

```
soup = BeautifulSoup(response.text, 'html.parser')
```

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

TASK 2: Extract all column/variable names from the HTML table header

```
# Use the find_all function in the BeautifulSoup object, with element
html_tables = soup.find_all('table')
# Assign the result to a list called `html_tables`
print(html_tables)
```

```
# 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="width: 10
0%;">
<tbody><tr>
<th scope="col">Flight No.
</th>
<th scope="col">Date and<br/>time (<a href="/wiki/Coordinated_Universa
```

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

TASK 3: Create a data frame by parsing the launch HTML tables

```
1
4 June 2010
18:45
F9 v1.07B0003.18
CCAFS
Dragon Spacecraft Qualification Unit
Dragon Spacecraft Qualification Unit
LEO
SpaceX
Success

Failure
2
8 December 2010
15:43
F9 v1.07B0004.18
CCAFS
Dragon
Dragon
LEO
NASA
Success
Failure
3
22 May 2012
07:44
F9 v1.07B0005.18
CCAFS
Dragon
```

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

Data Wrangling - EDA with Visualization

determine what would be the label for training supervised models.

Import Libraries (pandas, numpy)

Data Analysis df.head(10)

```
df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object
df.head(10)
```

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	La
0	1	2010-06-04	Falcon 9	6104.959412	LEO	C
1	2	2012-05-22	Falcon 9	525.000000	LEO	C
2	3	2013-03-01	Falcon 9	677.000000	ISS	C
3	4	2013-09-29	Falcon 9	500.000000	PO	
4	5	2013-12-03	Falcon 9	3170.000000	GTO	C
5	6	2014-01-06	Falcon 9	3325.000000	GTO	C
6	7	2014-04-18	Falcon 9	2296.000000	ISS	C

Identify and calculate the percentage of the missing values in each attribute

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

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

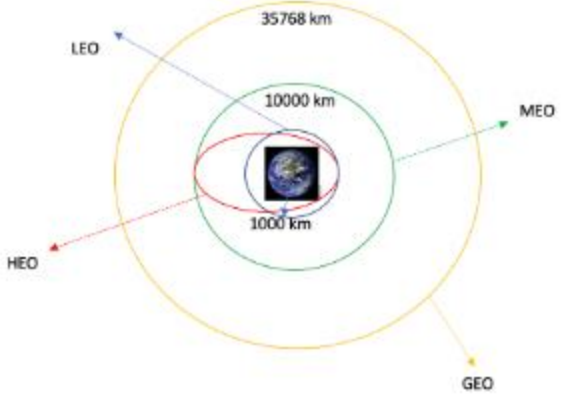
```
df.dtypes
```

```
FlightNumber        int64
Date               object
BoosterVersion     object
PayloadMass       float64
Orbit              object
LaunchSite         object
Outcome            object
Flights             int64
GridFins             bool
Reused               bool
Legs                 bool
LandingPad         object
Block             float64
ReusedCount         int64
Serial             object
Longitude         float64
Latitude          float64
dtype: object
```

TASK 1: Calculate the number of launches on each site

```
df['LaunchSite'].value_counts()
```

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



TASK 2: Calculate the number and occurrence of each orbit

```
df['Orbit'].value_counts()
```

```
Orbit
GTO      27
ISS      21
VLEO     14
PO        9
LEO       7
SSO       5
MEO       3
HEO       1
ES-L1     1
SO        1
GEO       1
Name: count, dtype: int64
```

TASK 3: Calculate the number and occurrence of mission outcome of the orbits

```
df['Outcome'].value_counts()
landing_outcomes=df['Outcome'].value_counts()
landing_outcomes
```

```
Outcome
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: count, dtype: int64
```

```
for i,outcome in enumerate(landing_outcomes.keys()):
    print(i,outcome)
```

```
0 True ASDS
1 None None
2 True RTLS
3 False ASDS
4 True Ocean
5 False Ocean
6 None ASDS
7 False RTLS
```

IBM Developer

LLS NETWORK

Data Wrangling - EDA with Visualization

TASK 4: Create a landing outcome label from Outcome column

```
# landing_outcomes = values on Outcome column
df['Outcome'].value_counts()
landing_outcomes=df['Outcome'].value_counts()
landing_outcomes
```

```
Outcome
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: count, dtype:
```

```
for i,outcome in enumerate(landing_outcomes.keys()):
    print(i,outcome)
```

```
0 True ASDS
1 None None
2 True RTLS
3 False ASDS
4 True Ocean
5 False Ocean
6 None ASDS
7 False RTLS
```

We can use the following line of code to determine the success rate:

```
df["Class"].mean()
```

```
np.float64(0.6666666666666666)
```

66.6% Success Rate

TASK 4: Create a landing outcome label from Outcome column

Using the `Outcome`, create a list where the element is zero if the corresponding row in `Outcome` is in the set `bad_outcome`; otherwise, it's one. Then assign it to the variable `landing_class`:

```
# landing_class = 0 if bad_outcome
landing_class = [0 if outcome in bad_outcomes else 1 for outcome in df
df['landing_class'] = landing_class
# landing_class = 1 otherwise
print(landing_class)
```

```
[0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1,
1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1,
1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

EDA with SQL

Import Libraries sqlalchemy, ipython-sql, csv, sqlite3

remove blank rows from table

Create a table

TASK 1: Display the names of the unique launch sites in the space mission

```
%sql select DISTINCT (LAUNCH_SITE) from SPACEXTBL
```

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

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

Task 2

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

```
%sql SELECT * from SPACEXTBL where (LAUNCH_SITE) LIKE 'CCA%' LIMIT
```

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

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MAS
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	5
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	5
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	6

Task 3

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

```
G_) AS total_payload_mass FROM SPACEXTBL WHERE Customer = 'NASA (CR
```

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

total_payload_mass
45596

Task 4

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

```
%sql select avg(PAYLOAD_MASS__KG_) from SPACEXTBL as payloadmass WHERE
```

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

avg(PAYLOAD_MASS__KG_)
2928.4

Task 5

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

Hint:Use min function

```
%sql SELECT MIN(DATE) \
FROM SPACEXTBL \
WHERE LANDING_OUTCOME = 'Success (ground pad)'
```

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

MIN(DATE)
2015-12-22

EDA with SQL

Import Libraries sqlalchemy, ipython-sql, csv, sqlite3

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

```
rom SPACEXTBL where LANDING_OUTCOME='Success (drone ship)' and PAYLOAD
```

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

Payload
JCSAT-14
JCSAT-16
SES-10
SES-11 / EchoStar 105

Task 7

List the total number of successful and failure mission outcomes

```
%sql select count(MISSION_OUTCOME) as missionoutcomes from SPACEXTBL GI
```

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

missionoutcomes
1
98
1
1

Task 8

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

```
N as boosterversion from SPACEXTBL where PAYLOAD_MASS__KG_=(select ma
```

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

boosterversion
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

Task 9

Complete the EDA with SQL

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_SI
FROM SPACEXTBL \
where [Landing_Outcome] = 'Failure (drone ship)' and substr(Date,0,5)=
```

```
 * 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)

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.

```
%sql SELECT Landing_Outcome, COUNT(*) AS count_outcomes FROM SPACEXTBL
```

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

Landing_Outcome	count_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

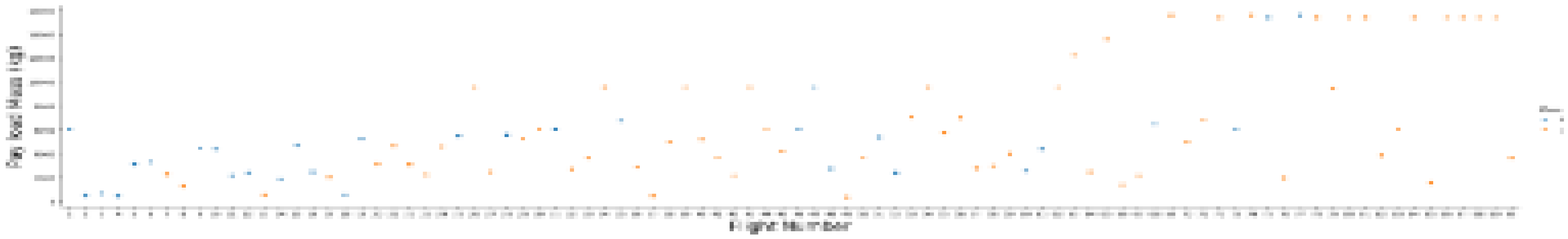
https://github.com/Statninja/IBM-DataScience-SpaceX-Capstone/blob/main/Complete%20the%20EDA%20with%20SQL/

EDA with Visualization Lab

Import Libraries
matplotlib.pyploy, seaborn

Flight Number vs. Payload Mass

```
sns.catplot(y="PayloadMass", x="FlightNumber", hue="Class", data=df, a
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Pay load Mass (kg)",fontsize=20)
plt.show()
```

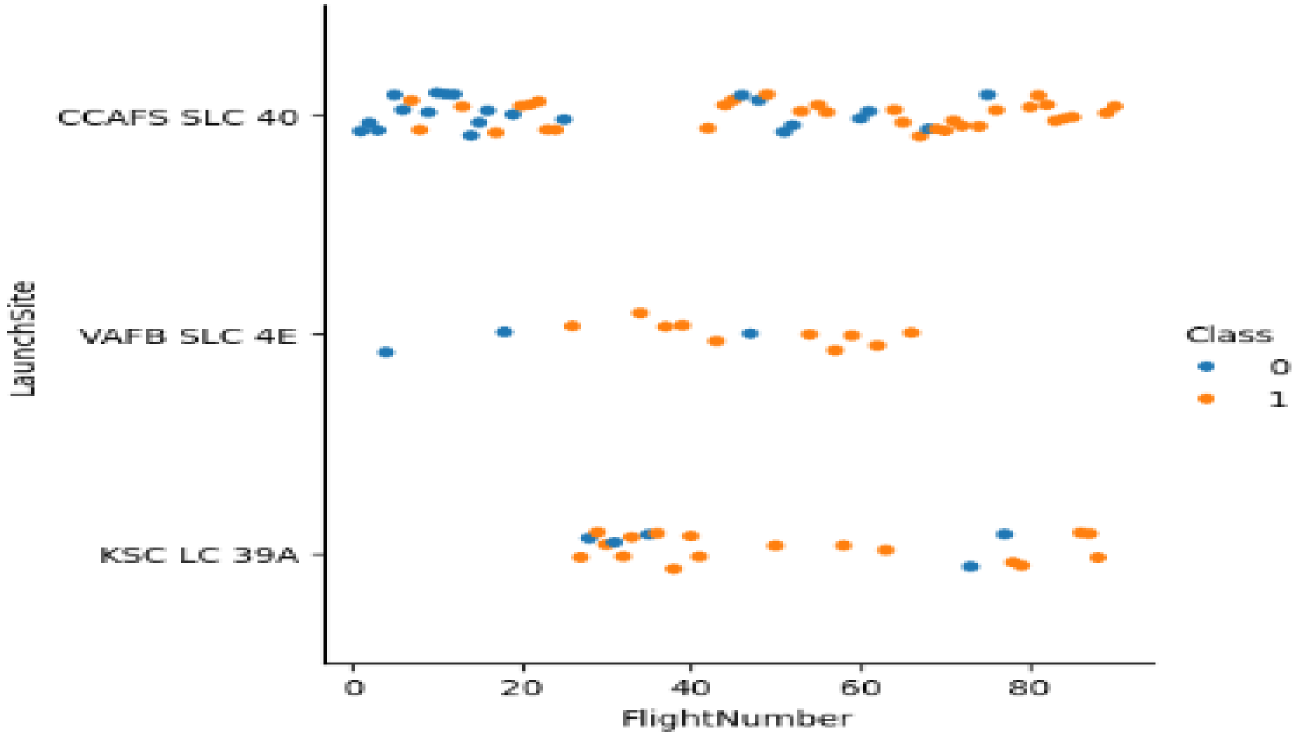


EDA with Visualization Lab

Flight Number vs. Launch Site

```
# Plot a scatter point chart with x axis to be Flight Number and y axi
sns.catplot(
     data=df,
     x='FlightNumber',
     y='LaunchSite',
     hue='Class',
     kind='strip'
)
```

<seaborn.axisgrid.FacetGrid at 0x64cbbd8>

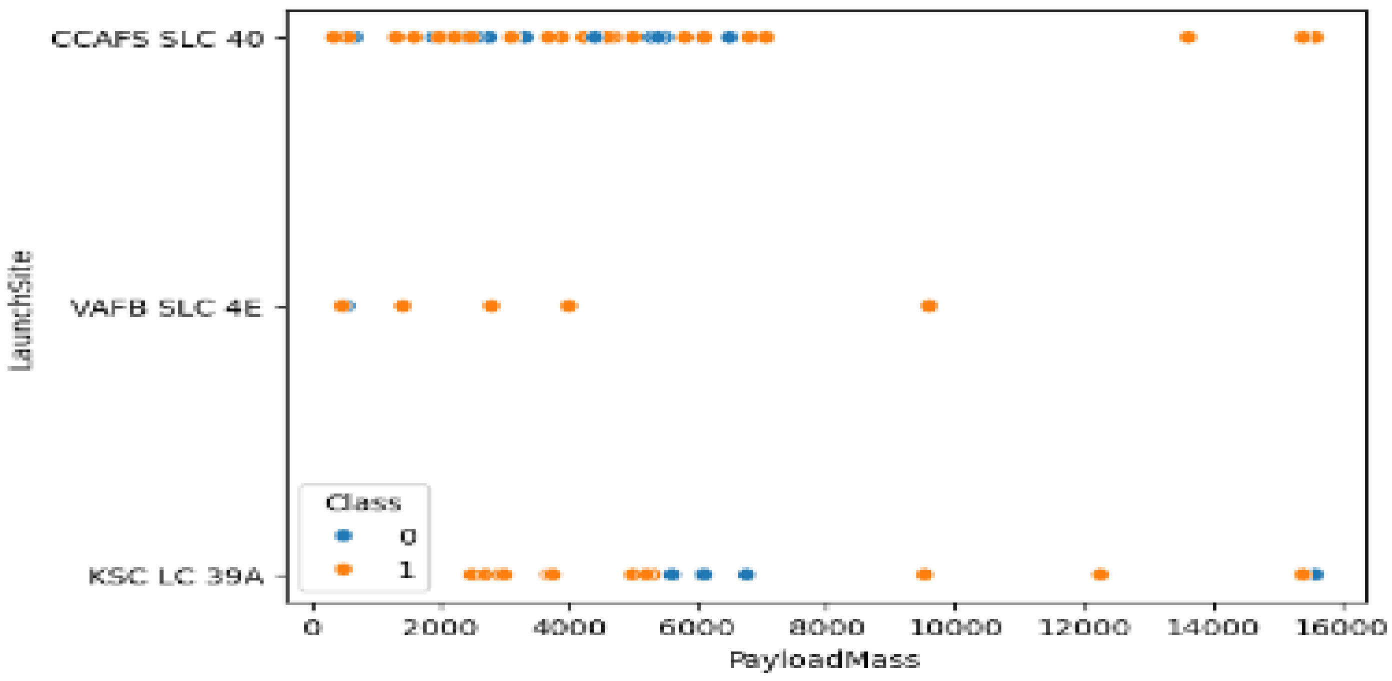


EDA with Visualization Lab

Launch Sites vs. Payloads Mass

```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y
sns.scatterplot(data=df, x="PayloadMass", y="LaunchSite", hue='Class')
```

```
<AxesSubplot:xlabel='PayloadMass', ylabel='LaunchSite'>
```



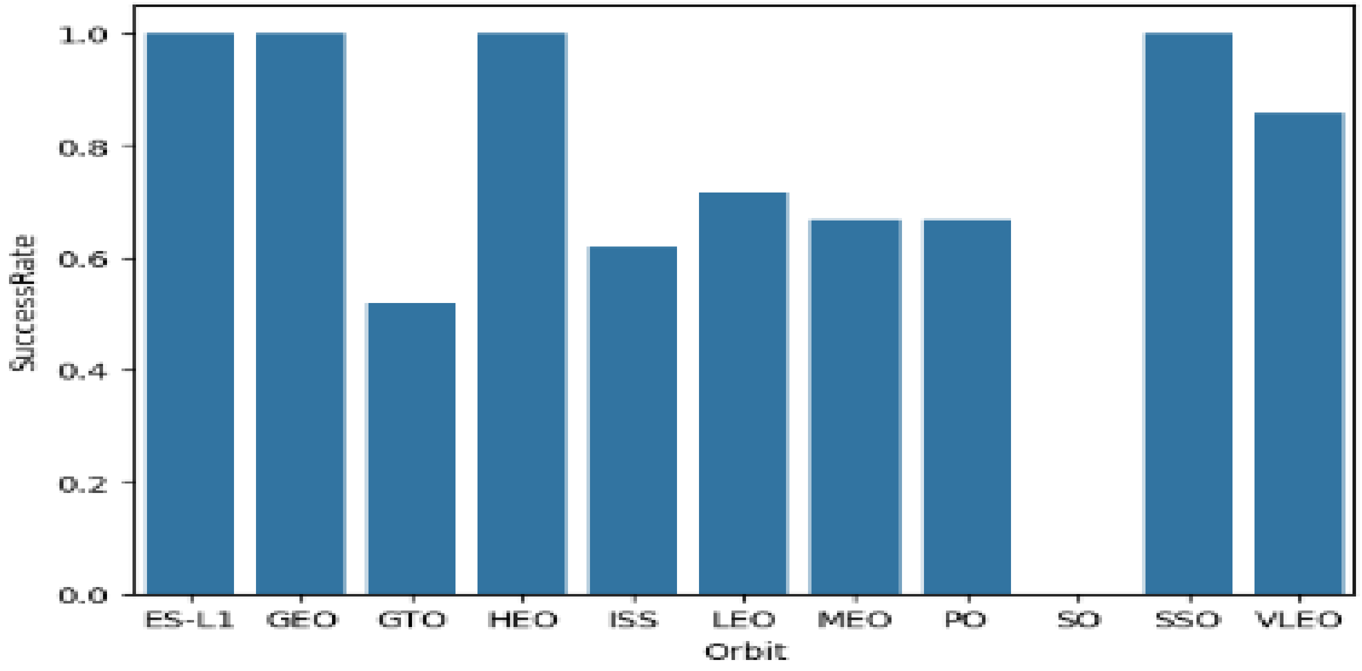
EDA with Visualization Lab

Success Rate for each orbit

```
# HINT use groupby method on Orbit column and get the mean of Class co

success_rate = df.groupby('Orbit')['Class'].mean().reset_index()
success_rate.columns = ['Orbit', 'SuccessRate']
sns.barplot(x='Orbit', y='SuccessRate', data=success_rate)
```

```
<AxesSubplot:xlabel='Orbit', ylabel='SuccessRate'>
```

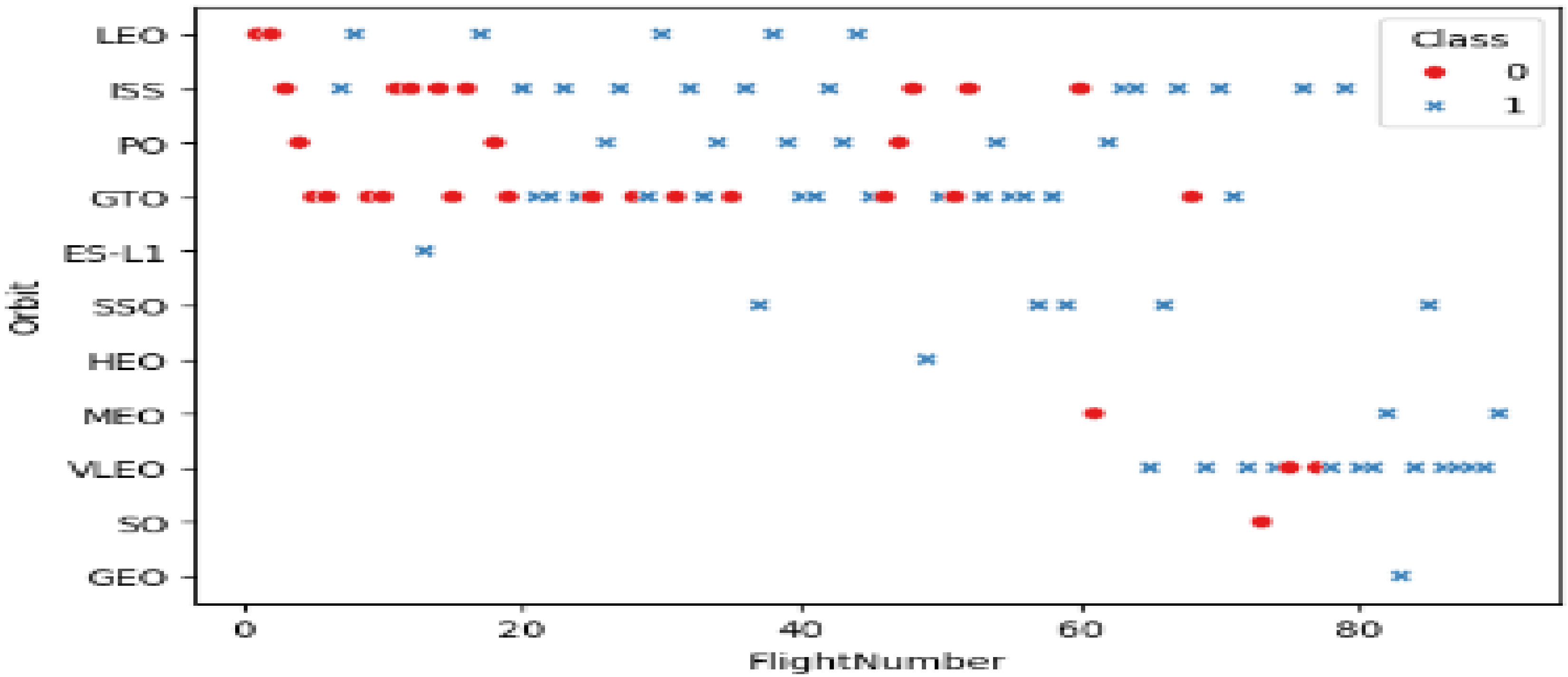


EDA with Visualization Lab

Flight Number vs. Orbit Type

```
# Plot a scatter point chart with x axis to be FlightNumber and y axis
sns.scatterplot(data=df, x='FlightNumber', y='Orbit', hue='Class', pal
```

<AxesSubplot:xlabel='FlightNumber', ylabel='Orbit'>

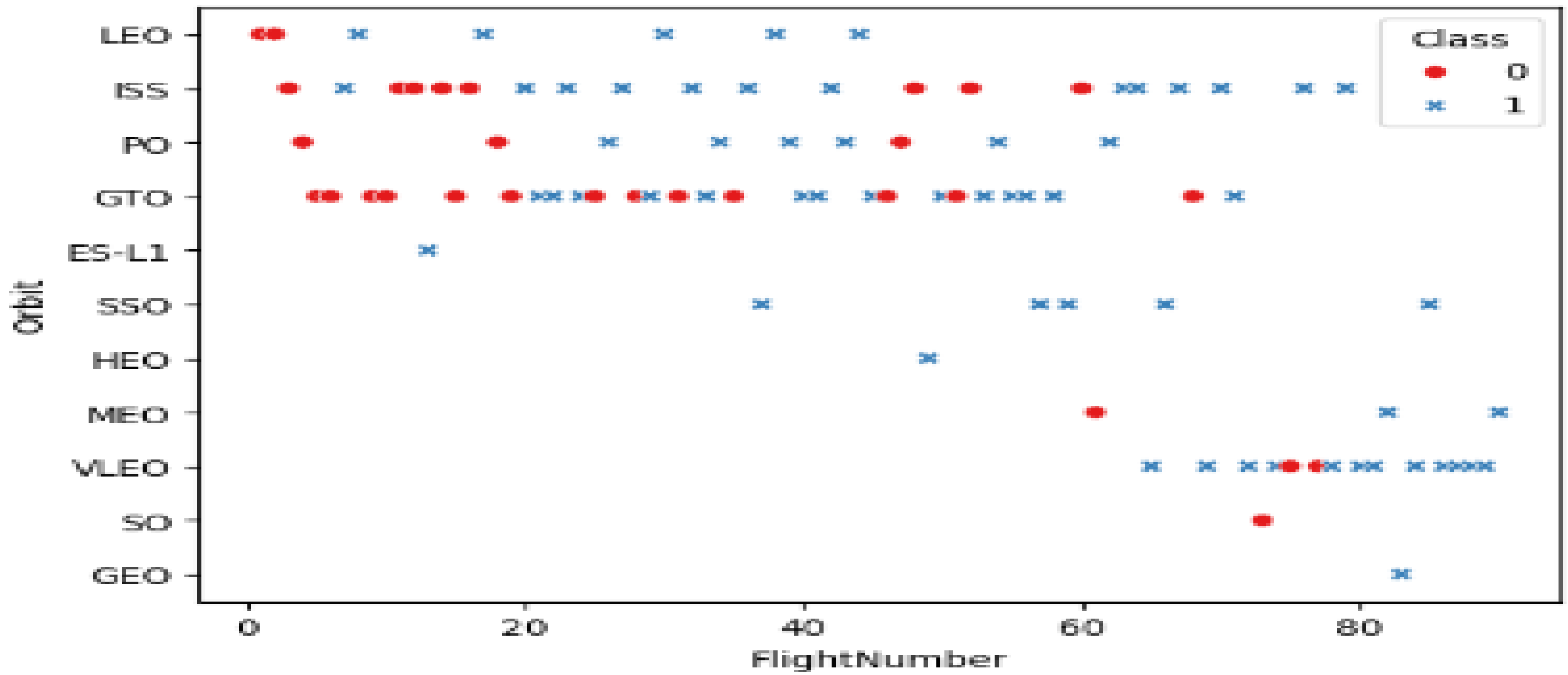


EDA with Visualization Lab

Flight Number vs. Orbit Type

```
# Plot a scatter point chart with x axis to be FlightNumber and y axis
sns.scatterplot(data=df, x='FlightNumber', y='Orbit', hue='Class', pal
```

```
<AxesSubplot:xlabel='FlightNumber', ylabel='Orbit'>
```

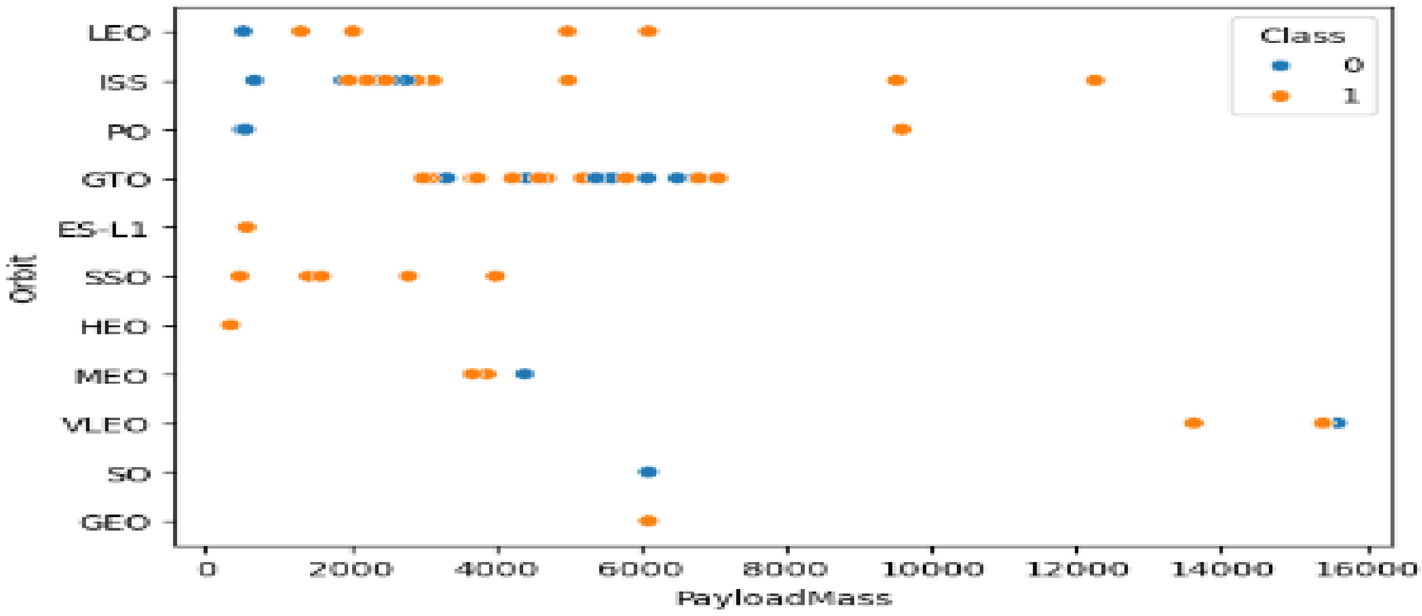


EDA with Visualization Lab

Payload Mass vs. Orbit

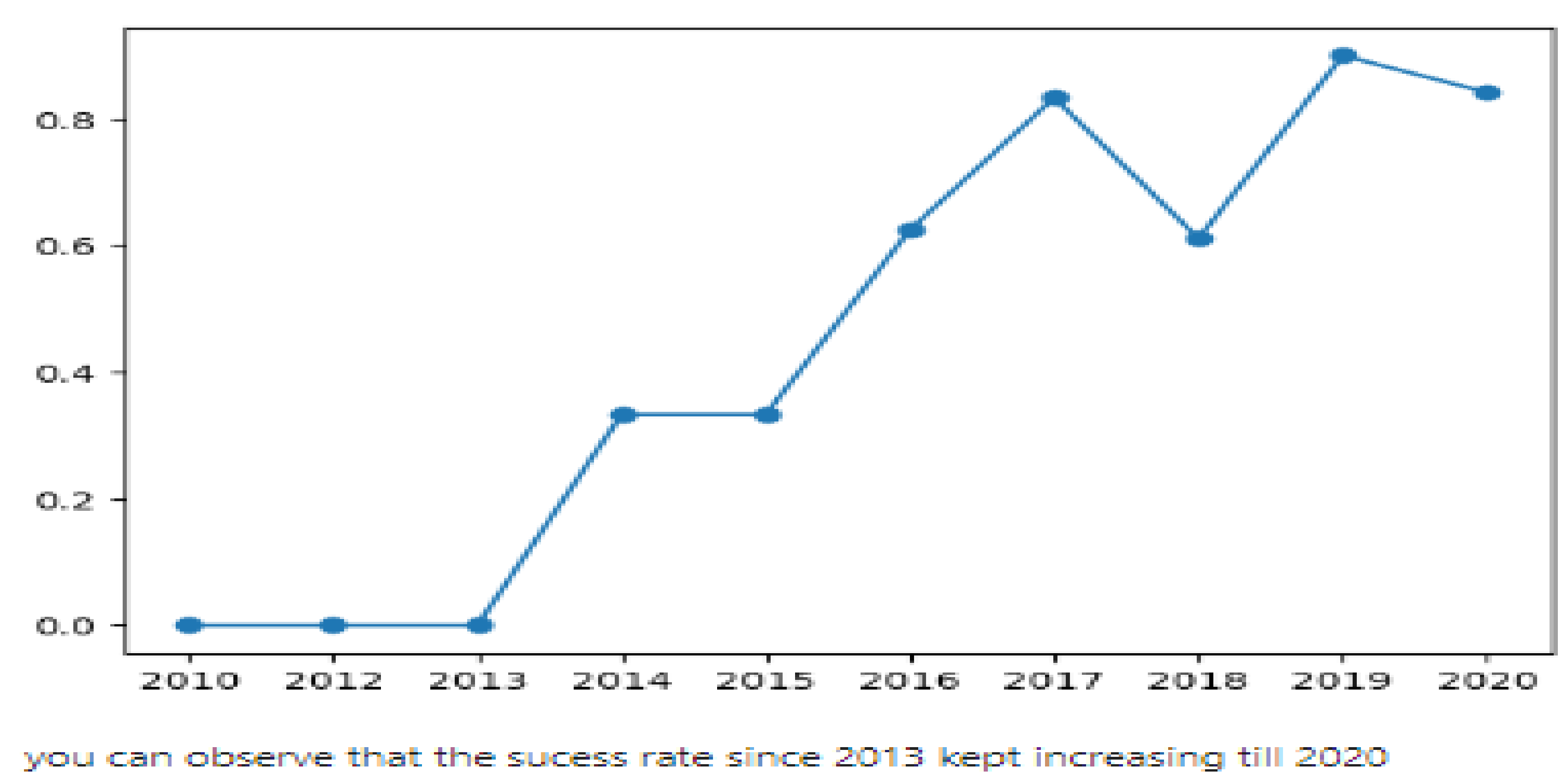
```
# Plot a scatter point chart with x axis to be Payload Mass and y axis
#chart with x axis to be FlightNumber and y axis to be the Orbit, and
sns.scatterplot(data=df, x='PayloadMass', y='Orbit', hue='Class')
```

```
<AxesSubplot:xlabel='PayloadMass', ylabel='Orbit'>
```



EDA with Visualization Lab

Success Rate since 2010. Since 2013 kept increasing till 2020



Interactive Visual Analytics with Folium lab

- **TASK 1:** Mark all launch sites on a map
- **TASK 2:** Mark the success/failed launches for each site on the map
- **TASK 3:** Calculate the distances between a launch site to its proximities

- https://github.com/Statninja/IBM-DataScience-SpaceX-Capstone/tree/main/Interactive%20Visual%20with%20Folium%20Lab

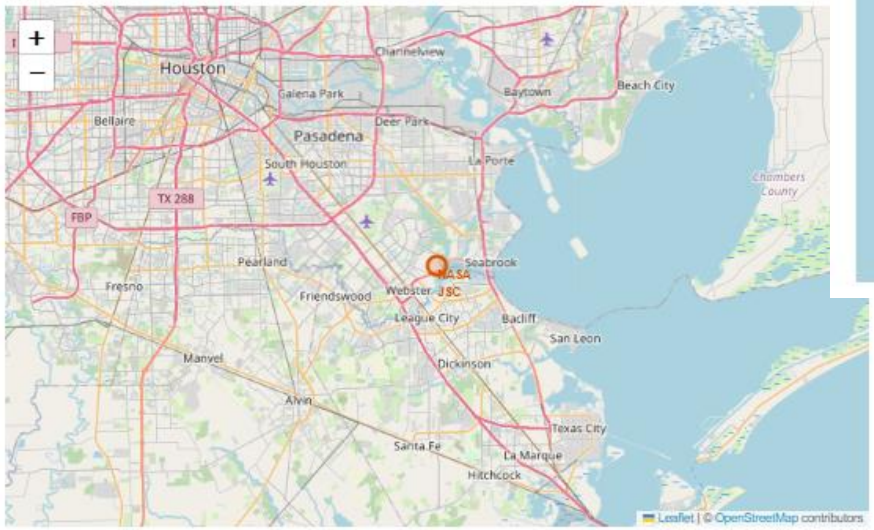
Interactive Visual Analytics with Folium lab

TASK 1: Mark all launch sites on a map

- Import Libraries Folium,Pandas

```
# Select relevant sub-columns: `Launch Site`, `Lat(Latitude)`, `Long(Longitude)`, `class`
spacex_df = spacex_df[['Launch Site', 'Lat', 'Long', 'class']]
launch_sites_df = spacex_df.groupby(['Launch Site'], as_index=False).first()
launch_sites_df = launch_sites_df[['Launch Site', 'Lat', 'Long']]
launch_sites_df
```

	Launch Site	Lat	Long
0	CCAFS LC-40	28.562302	-80.577356
1	CCAFS SLC-40	28.563197	-80.576820
2	KSC LC-39A	28.573255	-80.646895
3	VAFB SLC-4E	34.632834	-120.610745



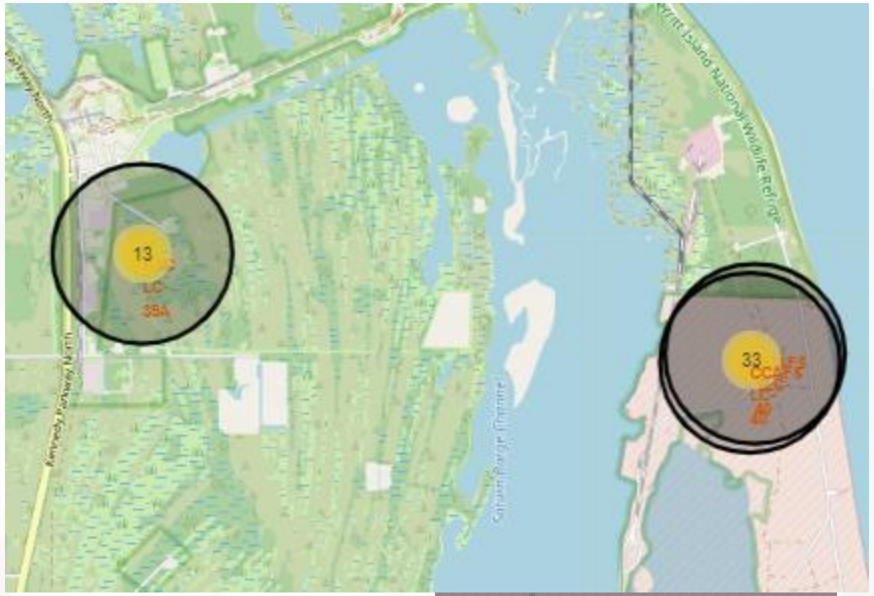

```
# Initial the map
site_map = folium.Map(location=nasa_coordinate, zoom_start=5)
# For each launch site, add a Circle object based on its coordinate (Lat, Long) values. In addition, a
for index, row in launch_sites_df.iterrows():
    coordinate = [row['Lat'], row['Long']]
    folium.Circle(coordinate, radius=1000, color='#000000', fill=True).add_child(folium.Popup(row['Lau
    folium.map.Marker(coordinate, icon=DivIcon(icon_size=(20,20),icon_anchor=(0,0), html='<div style="
site_map
```

Interactive Visual Analytics with Folium lab

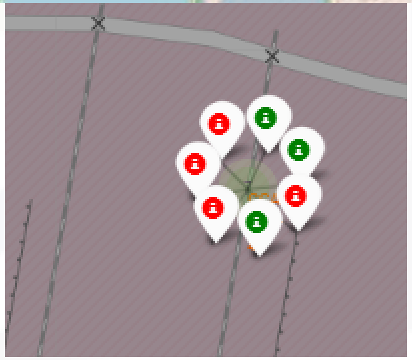
TASK 2: Mark the success/failed launches for each site on the map

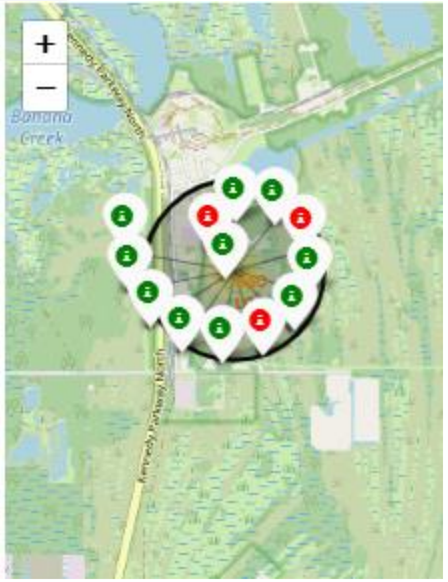
Import Libraries Folium,Pandas

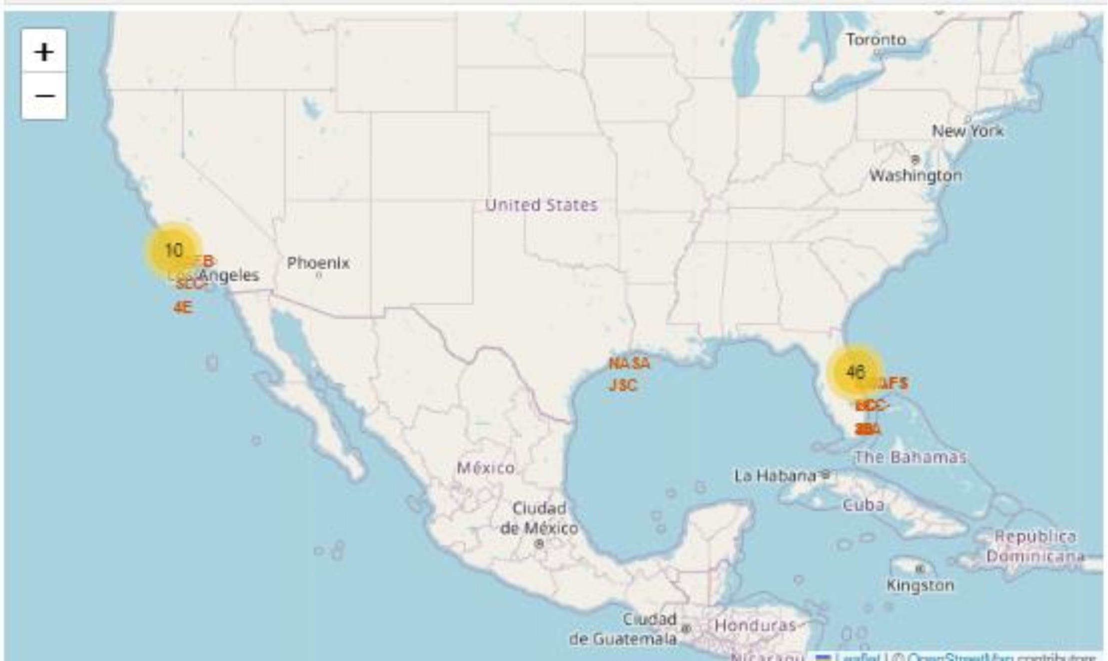
For each launch result in `spacex_df` data frame, add a `folium.Marker` to `marker_cluster`



Successful Laung = Green Marker



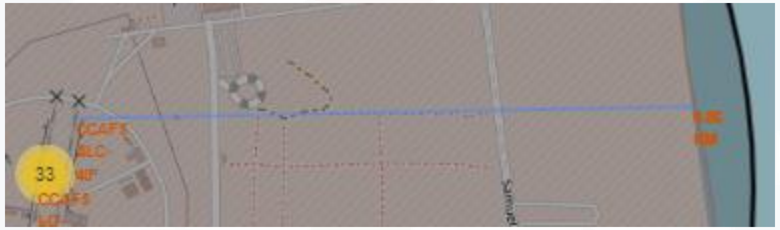




SKILLS NETWORK

Interactive Visual Analytics with Folium lab

TASK 3: Calculate the distances between a launch site to its proximities

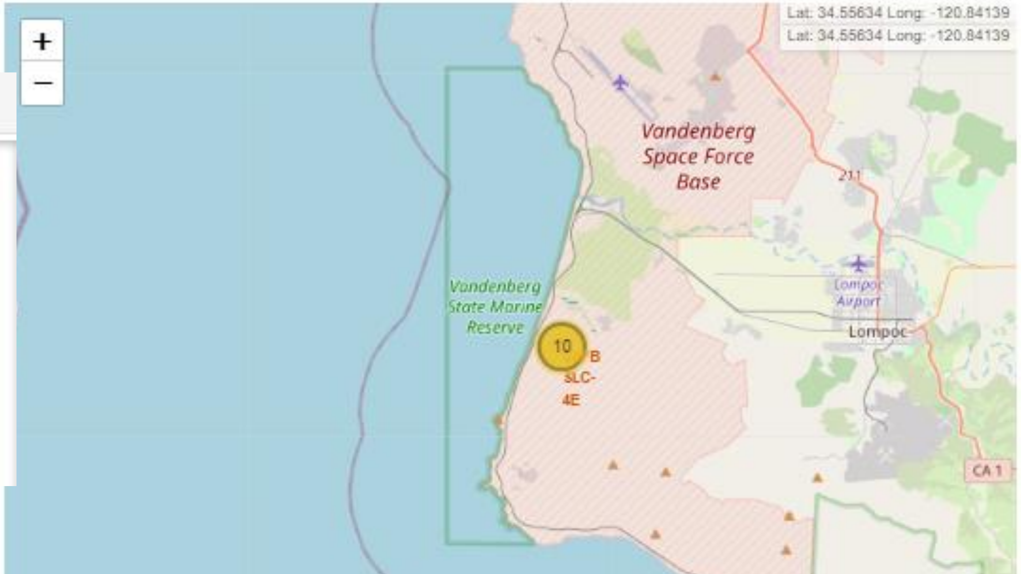



```python
# Create a marker with distance to a closest city, railway, highway, etc.
# Draw a line between the marker to the launch site
closest_highway = 28.56335, -80.57085
closest_railroad = 28.57206, -80.58525
closest_city = 28.10473, -80.64531
```

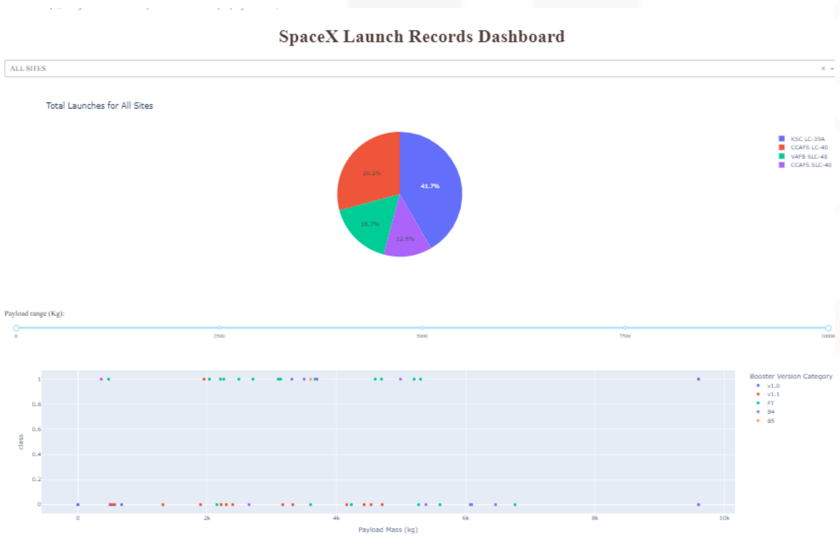
```python
distance_highway = calculate_distance(launch_site_lat, launch_site_lon, closest_
print('distance_highway =',distance_highway, ' km')
distance_railroad = calculate_distance(launch_site_lat, launch_site_lon, closes
print('distance_railroad =',distance_railroad, ' km')
distance_city = calculate_distance(launch_site_lat, launch_site_lon, closest_ci
print('distance_city =',distance_city, ' km')
```

```
distance_highway = 0.5834695366934144  km
distance_railroad = 1.2845344718142522  km
distance_city = 51.434169995172326  km
```

```python
# Create a `folium.PolyLine` object using the coastline coordinates and launch site coordinate
# lines=folium.PolyLine(locations=coordinates, weight=1)
coordinates = [[launch_site_lat,launch_site_lon],[coastline_lat,coastline_lon]]
lines=folium.PolyLine(locations=coordinates, weight=1)
site_map.add_child(lines)
```



DASHBOARD - Interactive analytics demo in screenshots

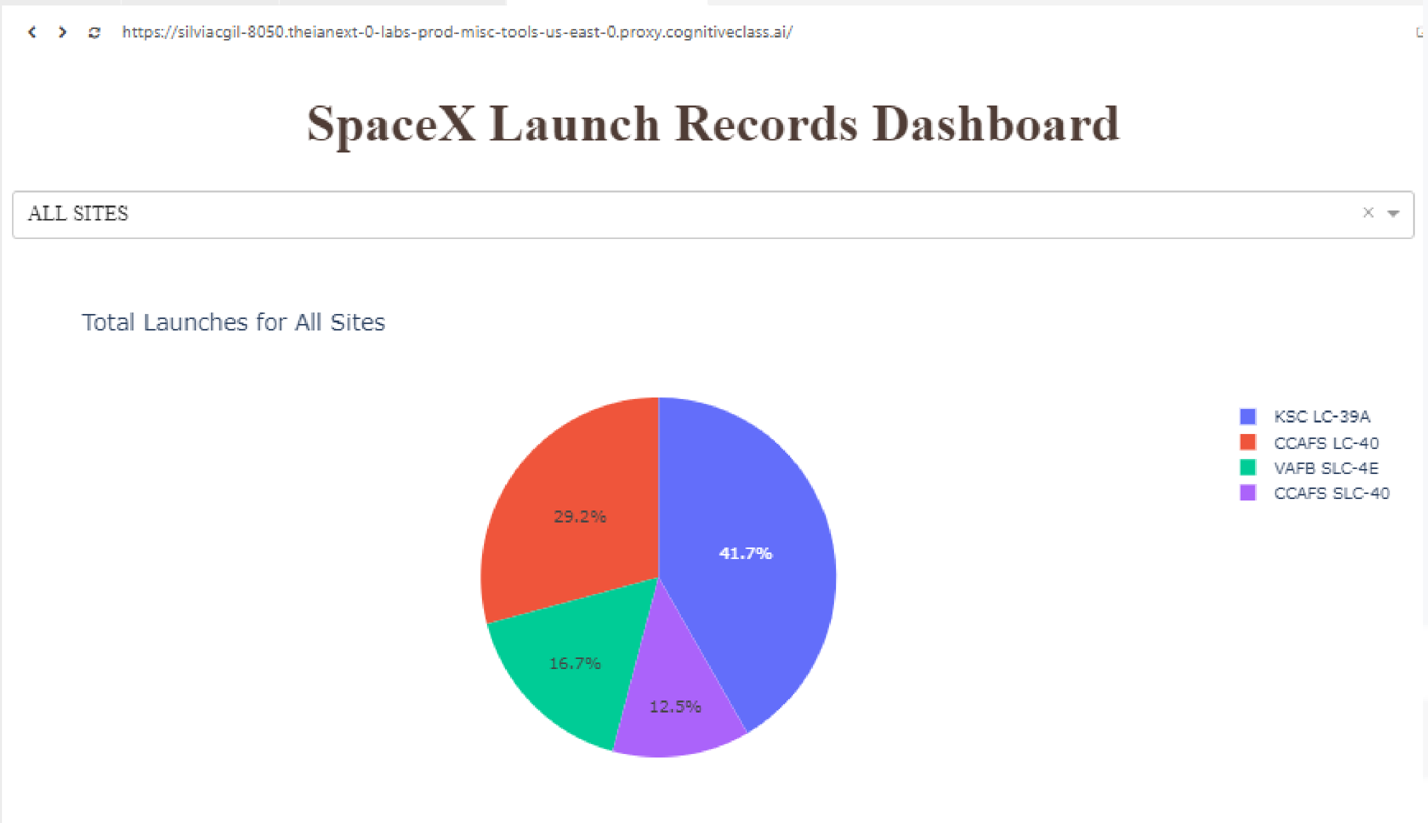


- TASK 1: Add a Launch Site Drop-down Input Component
- TASK 2: Add a callback function to render success-pie-chart based on selected site dropdown
- TASK 3: Add a Range Slider to Select Payload
- TASK 4: Add a callback function to render the success-payload-scatter-chart scatter plot

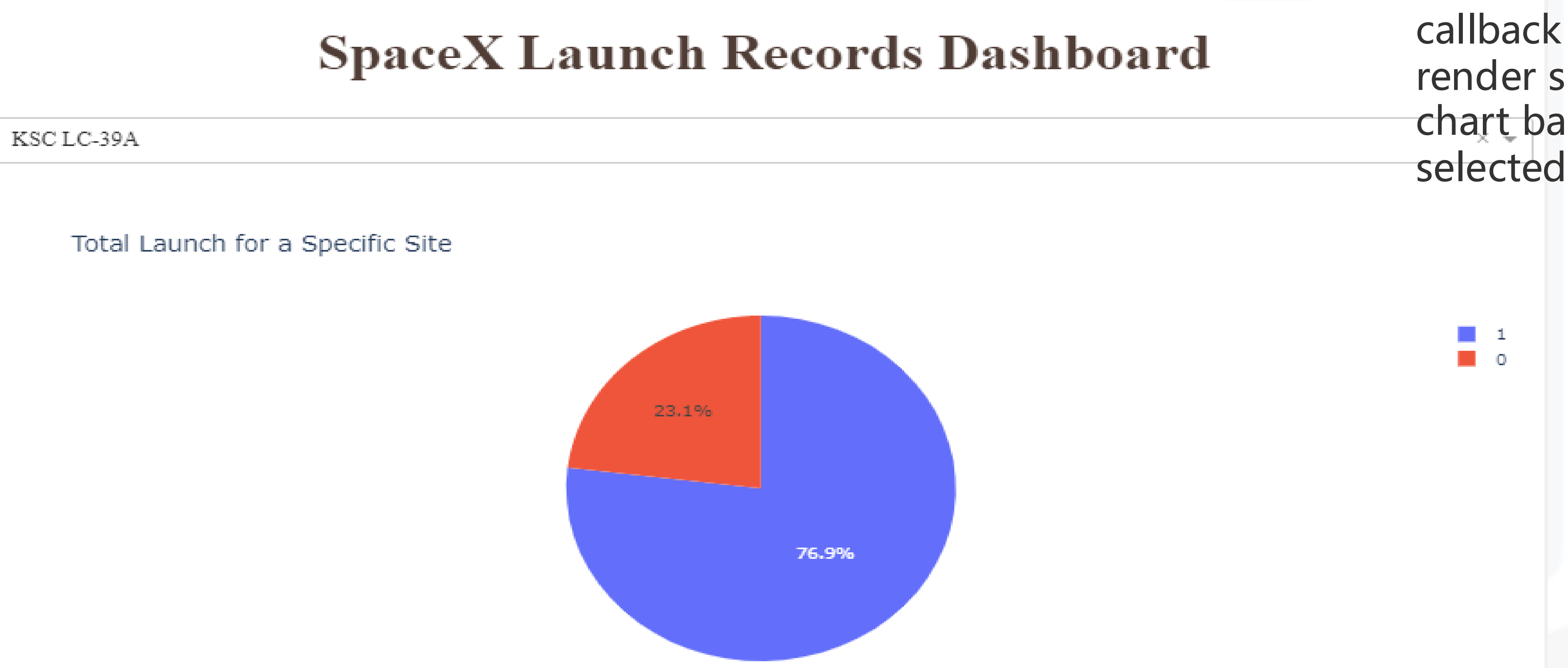
https://github.com/Statninja/IBM-DataScience-SpaceX-Capstone/tree/main/SpaceXLaunchDashboard

DASHBOARD TAB 1

TASK 1: Add a Launch Site Drop-down Input Component



DASHBOARD TAB 2

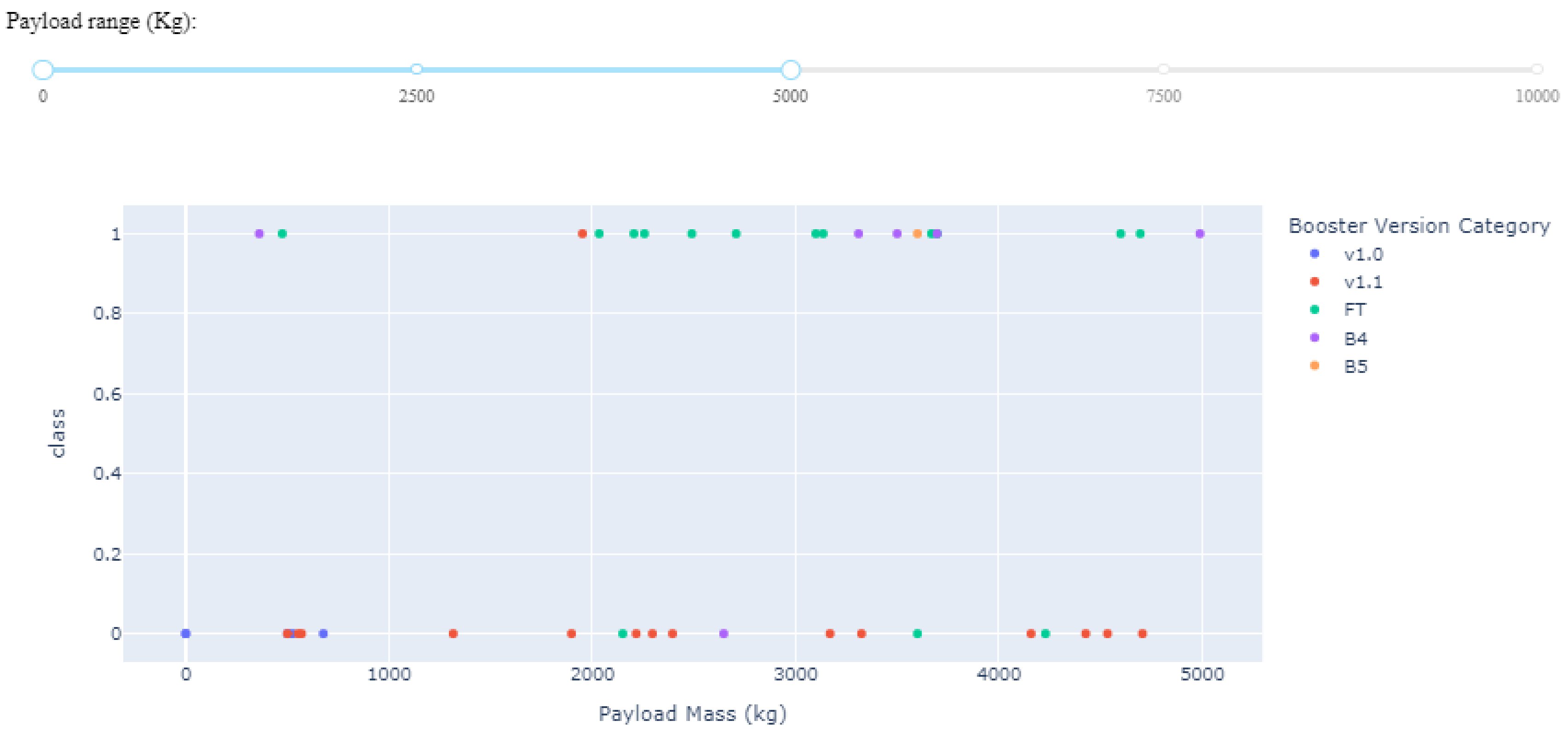


- TASK 2: Add a callback function to render success-pie-chart based on selected site dropdown

DASHBOARD TAB 3

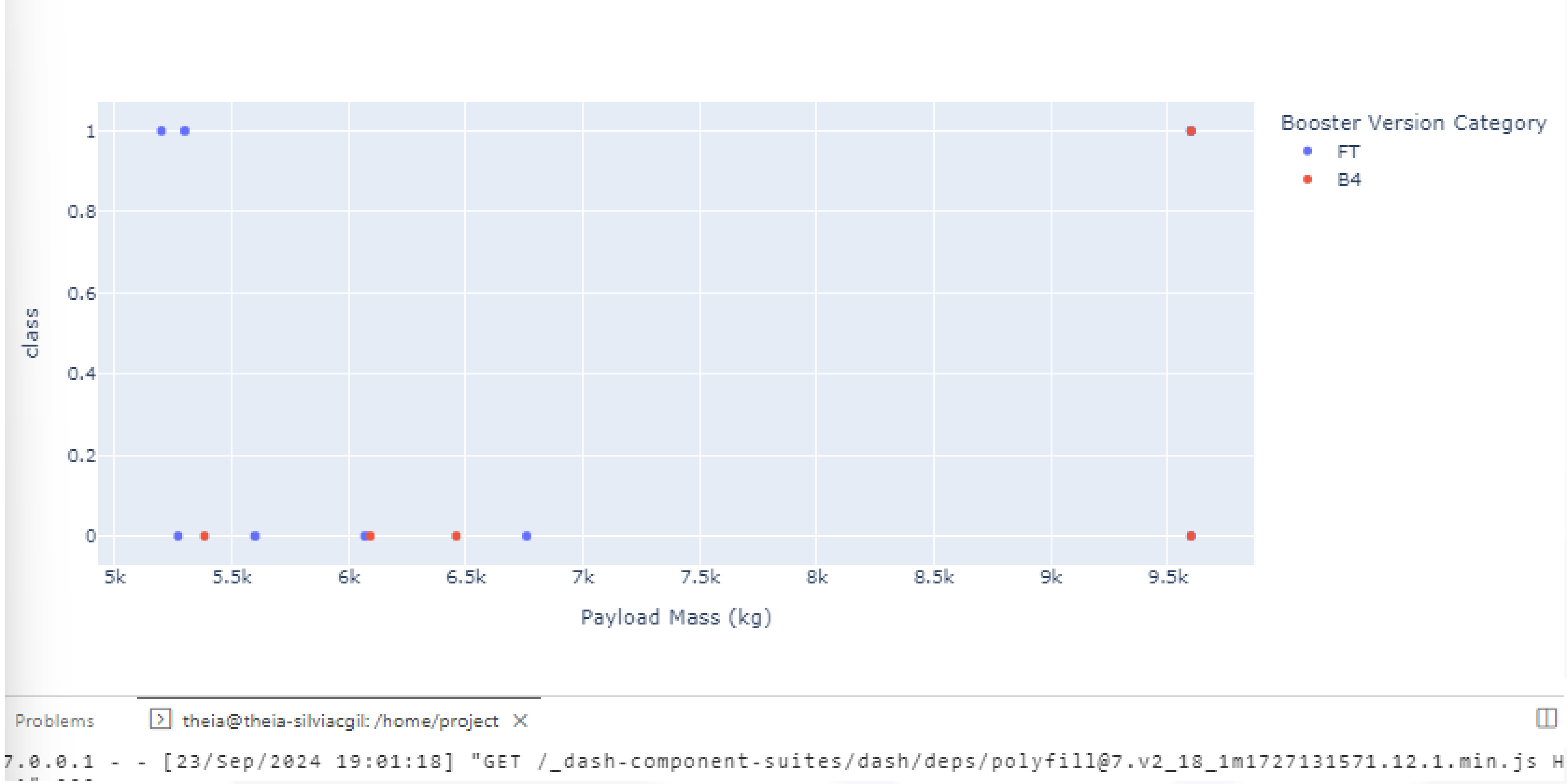
TASK 3: Add a Range Slider to Select Payload

DASHBOARD TAB 4

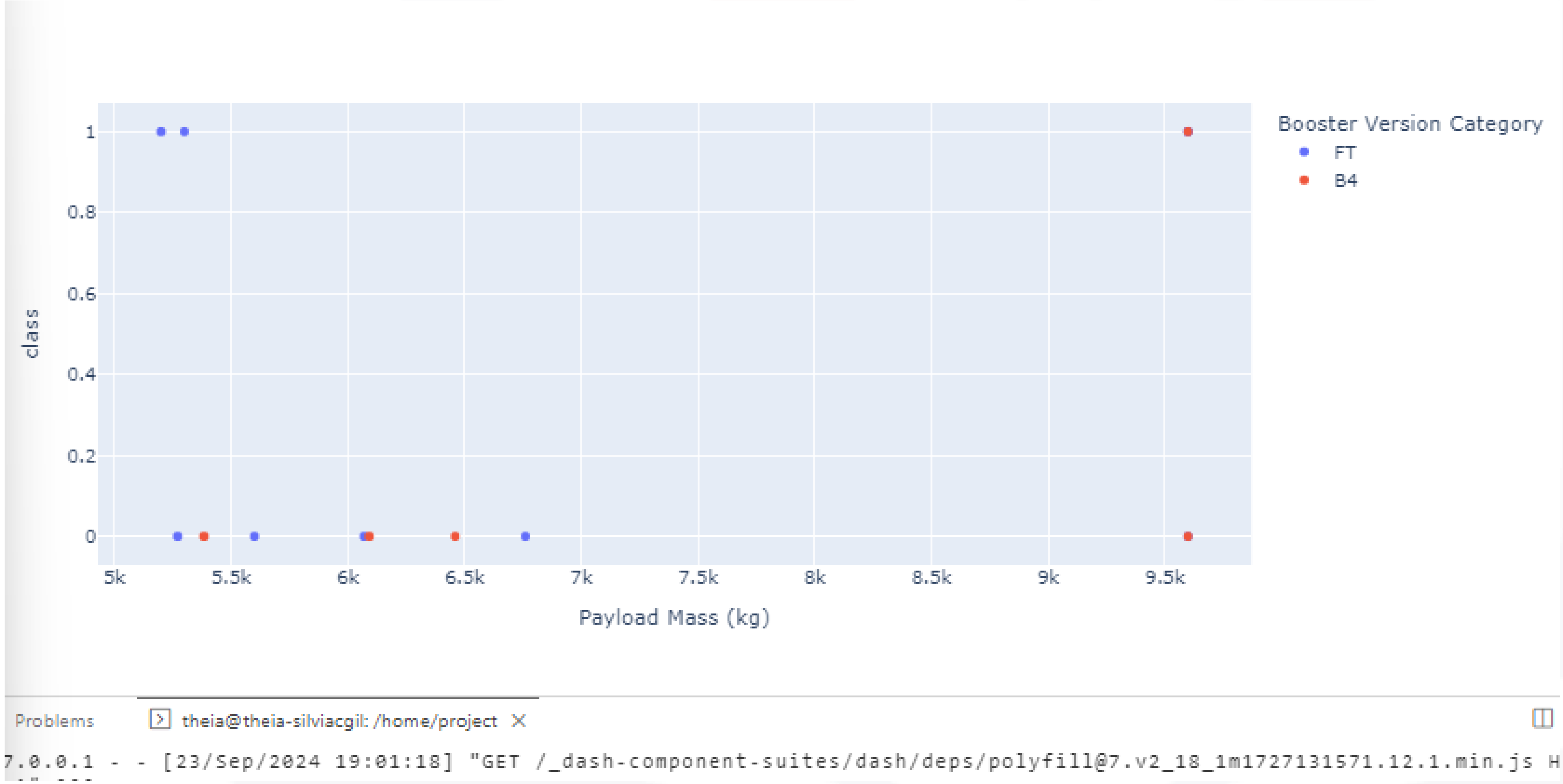


- TASK 4: Add a callback function to render the success-payload-scatter-chart scatter plot

DASHBOARD TAB 5



DASHBOARD TAB 5



Machine Learning Prediction lab

Tasks

Perform exploratory Data Analysis and determine Training Labels

1. create a column for the class
2. Standardize the data
3. Split into training data and test data
4. Find best Hyperparameter for SVM, Classification Trees and Logistic Regression
5. Find the method performs best using test data

Machine Learning Prediction lab

- Task 1. Create a column for the class
- Import libraries sklearn.svm,tree,neighbors

TASK 1

Create a NumPy array from the column `Class` in `data`, by applying the method `to_numpy()` then assign it to the variable `Y`,make sure the output is a Pandas series (only one bracket df['name of column']).

```
[96]: Y=data['Class'].to_numpy()
      #Y=data['Class']
      Y.dtype
      type(Y)
```

```
[96]: numpy.ndarray
```

- Task 2. Standardize the data

TASK 2

Standardize the data in `X` then reassign it to the variable `X` using the transform provided below.

```
# students get this
#transform = preprocessing.StandardScaler()
X= preprocessing.StandardScaler().fit(X).transform(X)
X
```

```
array([[-1.71291154e+00, -1.94814463e-16, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       [-1.67441914e+00, -1.19523159e+00, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       [-1.63592675e+00, -1.16267307e+00, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       ...,
       [ 1.63592675e+00,  1.99100483e+00,  3.49060516e+00, ...,
         1.19684269e+00, -5.17306132e-01,  5.17306132e-01],
       [ 1.67441914e+00,  1.99100483e+00,  1.00389436e+00, ...,
         1.19684269e+00, -5.17306132e-01,  5.17306132e-01],
       [ 1.71291154e+00, -5.19213966e-01, -6.53912840e-01, ...,
        -8.35531692e-01, -5.17306132e-01,  5.17306132e-01]])
```

Machine Learning Prediction lab

- Task 3. Split into training data and test data

TASK 3

Use the function train_test_split to split the data X and Y into training and test data. Set the parameter t
and random_state to 2. The training data and test data should be assigned to the following labels.

`X_train, X_test, Y_train, Y_test`

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
print ('Train set:', X_train.shape,  Y_train.shape)
print ('Test set:', X_test.shape,  Y_test.shape)
```

```
Train set: (72, 83) (72,)
Test set: (18, 83) (18,)
```

we can see we only have 18 test samples.

```
Y_test.shape
```

```
(18,)
```

- Task 4. Create a logistic regression object then create a GridSearchCV object logreg_cv with cv = 10. Fit the object to find the best parameters from the dictionary parameters.

TASK 4

Create a logistic regression object then create a GridSearchCV object `logreg_cv` with cv = 10. Fit the object
the best parameters from the dictionary `parameters`.

```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()
logreg_cv = GridSearchCV(estimator=lr, cv=10, param_grid=parameters).fit(X_train, Y_train)
```

We output the `GridSearchCV` object for logistic regression. We display the best parameters using the data a
`best_params_` and the accuracy on the validation data using the data attribute `best_score_`.

```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8464285714285713
```

Machine Learning Prediction lab

- Calculate the accuracy on the logreg test data using the method score

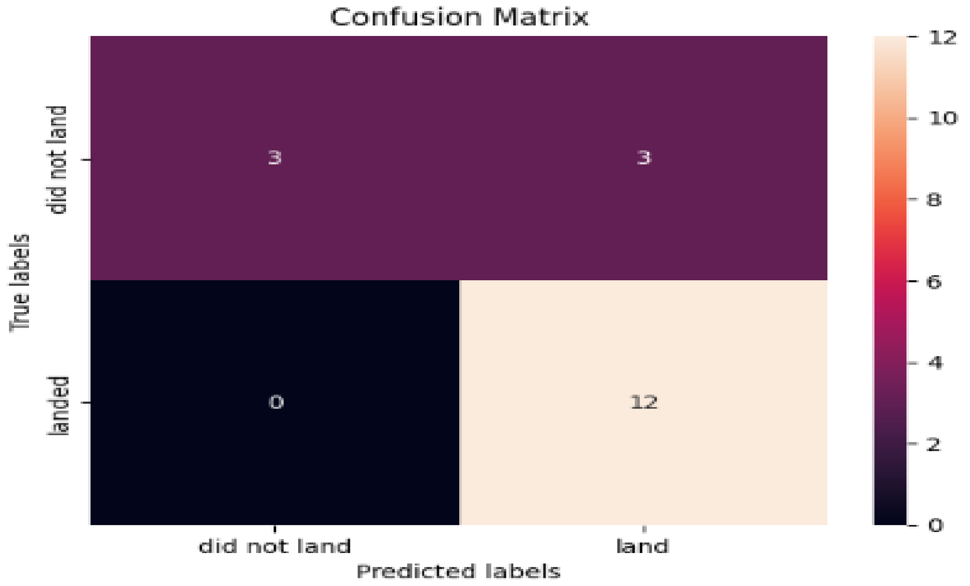
Calculate the accuracy on the test data using the method `score` :

```
logreg_score = logreg_cv.score(X_test, Y_test)
print("score :", logreg_score)
```

```
score : 0.8333333333333334
```

Lets look at the confusion matrix:

```
logreg_yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



- Create a support vector machine object then create a GridSearchCV object svm_cv with cv = 10. Fit the object to find the best parameters from the dictionary parameters.

```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
svm_cv = GridSearchCV(estimator=svm, cv=10, param_grid=parameters).fit(X_train, Y_train)
```

```
print("tuned hpyerparameters :(best parameters) ",svm_cv.best_params_)
print("accuracy :",svm_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
accuracy : 0.8482142857142856
```

SKILLS NETWORK

Machine Learning Prediction lab

- Calculate the accuracy on the svm_cv test data using the method score

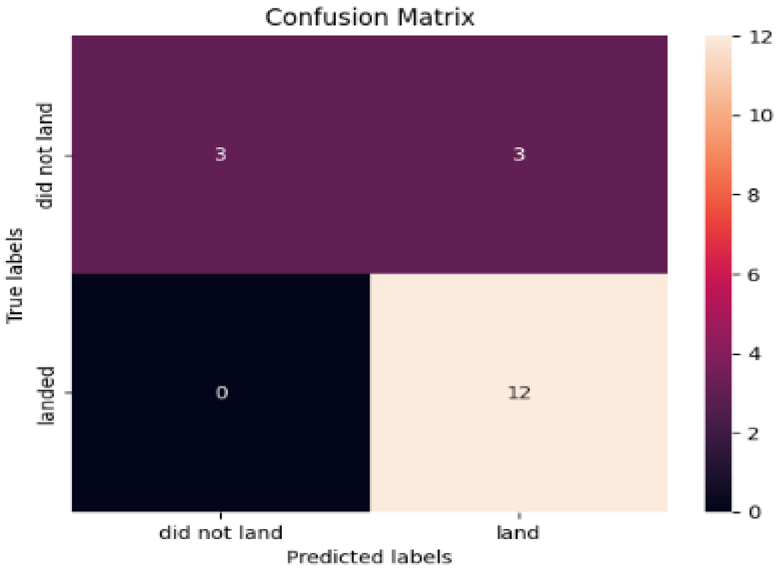
Calculate the accuracy on the test data using the method `score` :

```
svm_cv_score = svm_cv.score(X_test, Y_test)
print("score :",svm_cv_score)
```

```
score : 0.8333333333333334
```

We can plot the confusion matrix

```
svm_yhat=svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



- Create a support vector machine object then create a GridSearchCV object svm_cv with cv = 10. Fit the object to find the best parameters from the dictionary parameters.

Create a decision tree classifier object then create a `GridSearchCV` object `tree_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

```
parameters = {'criterion': ['gini', 'entropy'],
     'splitter': ['best', 'random'],
     'max_depth': [2*n for n in range(1,10)],
     'max_features': ['auto', 'sqrt'],
     'min_samples_leaf': [1, 2, 4],
     'min_samples_split': [2, 5, 10]}

tree = DecisionTreeClassifier()
```

```
grid_search = GridSearchCV(tree, parameters, cv=10)
tree_cv = grid_search.fit(X_train, Y_train)
```

```
print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 8, 'max_features': 'sqrt'
'min_samples_leaf': 2, 'min_samples_split': 2, 'splitter': 'random'}
accuracy : 0.8892857142857145
```

Machine Learning Prediction lab

- Calculate the accuracy on the tree_cv test data using the method score

Calculate the accuracy of tree_cv on the test data using the method `score`:

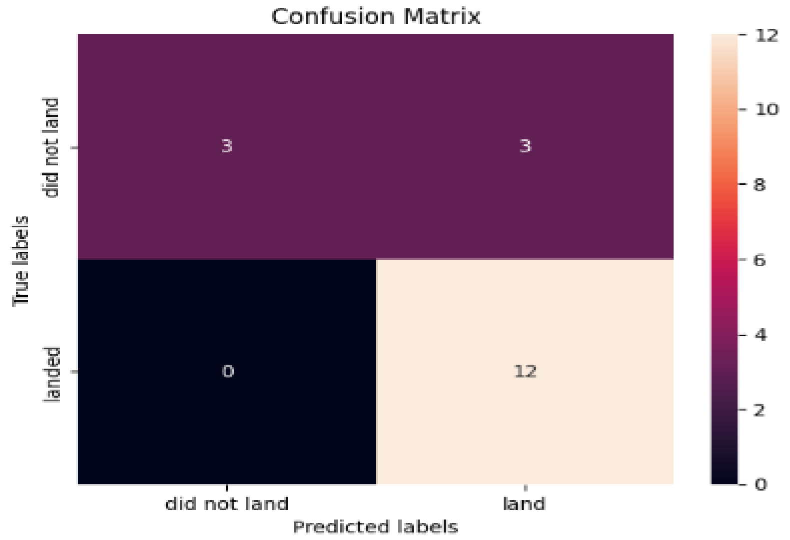
```
tree_cv_score=tree_cv.score(X_test, Y_test)

print("score :",tree_cv_score)
```

```
score : 0.8333333333333334
```

We can plot the confusion matrix

```
tree_yhat = tree_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



- Create a k nearest neighbors object then create a GridSearchCV object knn_cv with cv = 10. Fit the object to find the best parameters from the dictionary parameters.

Create a k nearest neighbors object then create a `GridSearchCV` object `knn_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

```
parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'p': [1,2]}

KNN = KNeighborsClassifier()
knn_cv = GridSearchCV(estimator=KNN, cv=10, param_grid=parameters).fit(X_train, Y_train)
```

```
print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
print("accuracy :",knn_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}
accuracy : 0.8482142857142858
```

Machine Learning Prediction lab

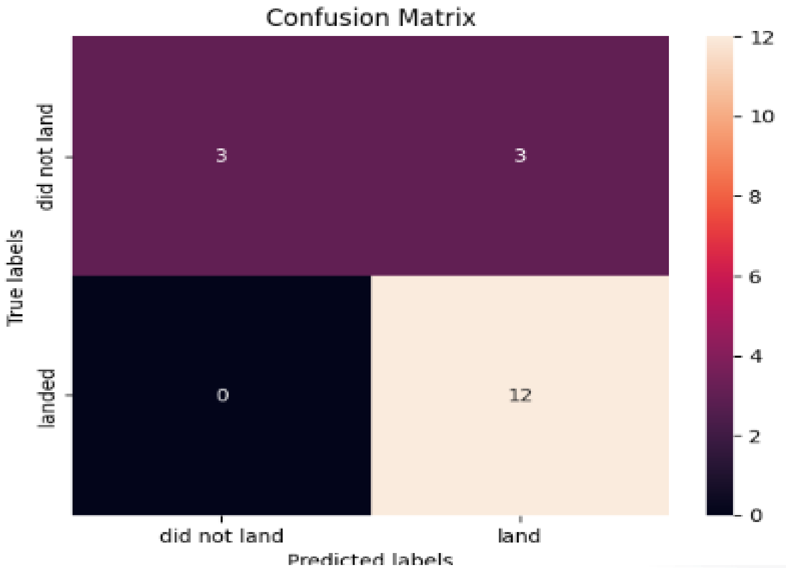
- Calculate the accuracy of knn_cv on the test data using the method score

```
knn_cv_score = knn_cv.score(X_test, Y_test)
print("score :",knn_cv_score)
```

```
score : 0.8333333333333334
```

We can plot the confusion matrix

```
knn_yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



- Find the method performs best

```
accuracy = [svm_cv_score, logreg_score, knn_cv_score, tree_cv_score]
accuracy = [i * 100 for i in accuracy]

method = ['Support Vector Machine', 'Logistic Regression', 'K Nearest Neighbour', 'Decision Tree']
models = {'ML Method':method, 'Accuracy Score (%)':accuracy}

ML_df = pd.DataFrame(models)
ML_df
print('Accuracy for Logistics Regression method:', logreg_cv.score(X_test, Y_test))
print( 'Accuracy for Support Vector Machine method:', svm_cv.score(X_test, Y_test))
print('Accuracy for Decision tree method:', tree_cv.score(X_test, Y_test))
print('Accuracy for K nearsdt neighbors method:', knn_cv.score(X_test, Y_test))
```

```
Accuracy for Logistics Regression method: 0.8333333333333334
Accuracy for Support Vector Machine method: 0.8333333333333334
Accuracy for Decision tree method: 0.9444444444444444
Accuracy for K nearsdt neighbors method: 0.8333333333333334
```

https://github.com/Statninja/IBM-DataScience-SpaceX-Capstone/tree/main/Machine%20Learning%20Pred

OVERALL FINDINGS & IMPLICATIONS

Findings

- Model Performance: The models performed similarly on the test set with the decision tree model slightly better performance
- All the launch sites are near the coast
- Success Rate of Landings: 66.67%
- KSC LC-39A: Has the highest success rate among launch sites.
- Orbits: ES-L1, GEO, HEO, and SSO have a 100% success rate
- Implication 3

CONCLUSION

- Larger datasets may support in getting more findings
- Conduct further analysis with different Machine Learning algorithms to see if accuracy can get improved