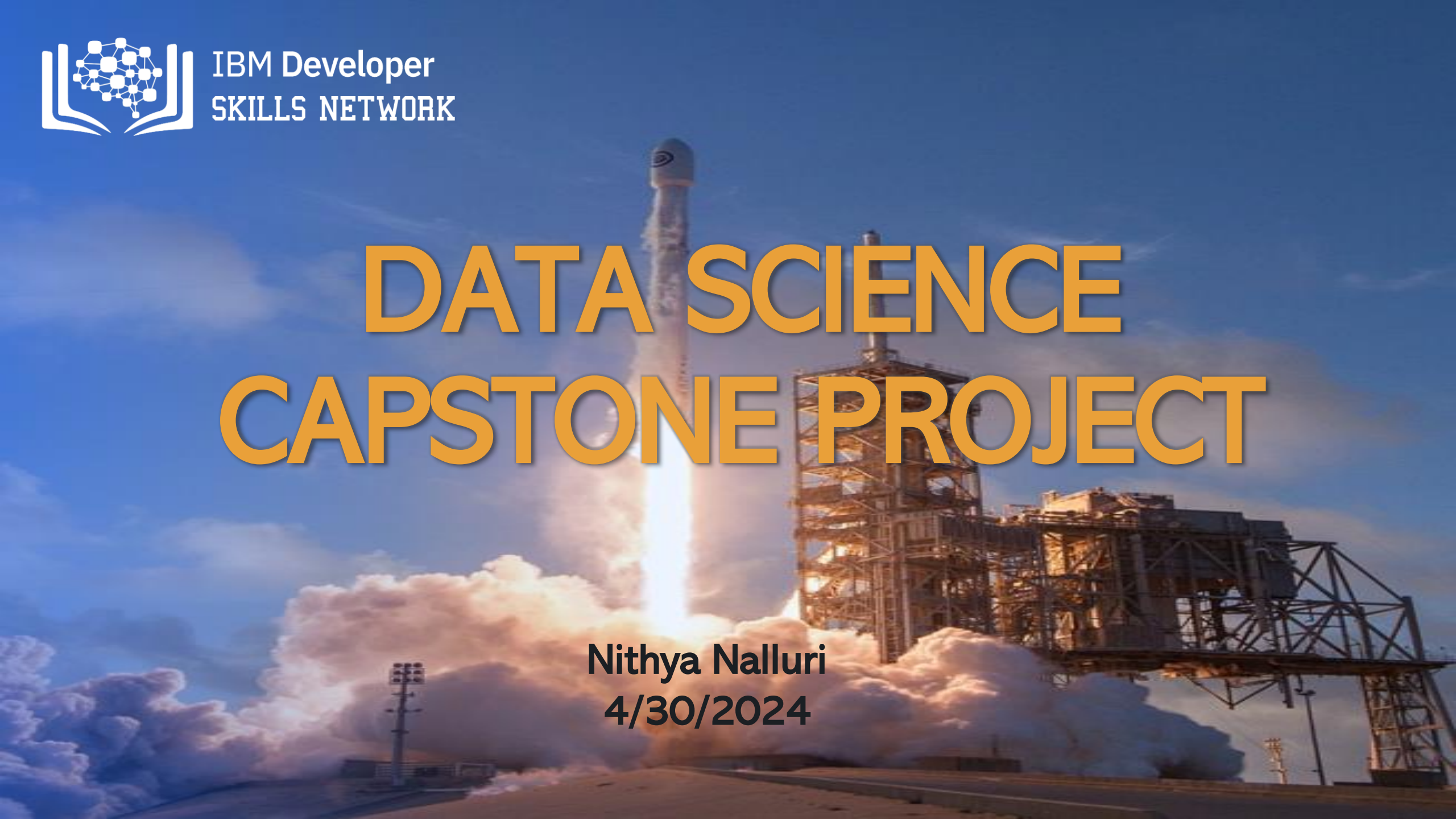
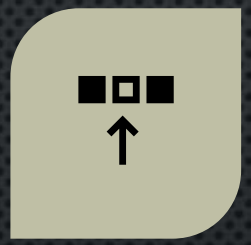


DATA SCIENCE CAPSTONE PROJECT

Nithya Nalluri

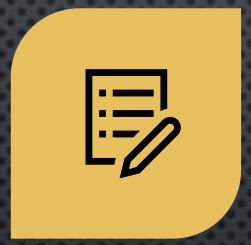
4/30/2024

OUTLINE



EXECUTIVE SUMMARY

INTRODUCTION



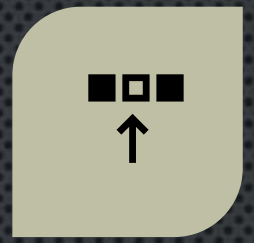
METHODOLOGY

RESULTS

CONCLUSION

APPENDIX

EXECUTIVE SUMMARY



OVERVIEW OF ALL METHODOLOGIES:

- DATA COLLECTION
- DATA WRANGLING
- EXPLORATORING DATA ANALYSIS
- INTERCATIVE VISUAL ANALYSIS
- PREDDICTIVE ANALYSIS

OVERVIEW OF ALL RESULTS:

- EXPLORATORY DATA ANALYSIS RESULTS (EDA)
- GEOSPATIAL ANALYTICS
- INTERACTIVE DASHBOARD
- PREDICITIVE ANAYLSIS OF CLASSFICTION MODES

INTRODUCTION

Project Background:

- SpaceX has launched Falcon rockets at the cost of roughly $62 million. Overall compared other companys thheir luanch was siginatly cheaper as some of their luanchs can be upwards of $165 million. Spacex is able to reduce the cost of their launchs by being able to reuse tier first stage of the rocket

Project Goal:

- If We prediction if launchs will be able to land or not in their first stages, and determine the cost of the launch. Using this information can use when a company wants to bid aganist spacex for a rocket launch

Project Result:

- This project will allow us to predict to perdict if spacex falcon 9 will land sucessfully

METHODOLOGY SUMMARY

1) Data Collection

- Sending GET request to SPACEX REST API and creating new dataframe
- Web scraping

2) Data Wrangling

- Remove all NaN values
- Count number of istnaces for the follow questions
- Add Labels to landing outcomes
    - 0 if bosster does not land sucessfully
    - 1 if booster does land sucessfully

3) Explortatory Data Analysis

- Using Matplotlib and Seaborn to create visualtin that shjow realstionships between differet variables and find any trends of pattersn in the plots
- Used SQL to query through the SPACEX to find certain data

METHODOLOGY SUMMARY (cont.)

4) Interactive Visual Analytics

- Using Folium to do Geospatial analysis
  - Creating interactive maps
  - Displaying data overlaying maps
- Using Plotly Dash to create a interactive dashboard
  - Supports various types of charts and allows for zomming, panning and annotation on the plots

5) Data Modeling and Evaluation

- Using Scikit-learn to do the following:
  - Prepocessing
  - Train-Test Splits
  - Classification Models
  - Hyperparameter Tuning with GridSearchCV
  - Create Confusion Matrices
  - Assessing Accuracy

Data Collection

1) Requesting all the contents of the SPACEX API and convert into dataframe

2) Reconstructing the dataset into a dictionary format and then into a DataFrame using helper functions

3) Filtering dataframe to only include Falcon 9 launch data and remove missing values

SPACEX REST API

1) Extract a Falcon 9 launch records from HTML page

2) Parse the launch HTML table and convert it into a dataframe

WEB SCRAPING

GitHub Link

Data Collection (SPACEX REST API)

GitHub Link

1) Requesting all the contents of the SPACEX API and convert into dataframe

- Step 1: Sending GET requests to the SPACEX REST API and return the GET response
- Step 2: Convert the response to .json then to pandas dataframe

```
# Spacex Rest API link
space_url = “https://...”

# Sending a GET request to the Spacex Rest API
response = requests.get(spacex_url)

# If status response is 200, request of successfull
response.status_code

# .json() to turn response into JSON format
r = response.json()

#n.json_normalize() turns the .json to the Pandas dataframe
data = pd.json_normlize(r)
```

Data Collection (SPACEX REST API) Cont.

2) Reconstructing the dataset into a dictionary format and then into a dataframe using helper functions

- Step 1: Creating global variables for all the column headers to store data returned by the helper functions
- Step 2: Utilizing the helper functions (getLaunchSite, getBoosterVersion, getPayloadData, getCoreData) to fetch data from columns with IDs
- Step 3: Merging the columns into a dictionary format using the received data and defined global variables
- Step 4: Creating a dataframe from the launch_dict dictionary

```
#Global variables
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
ReusedCount = []
Serial = []
Longitude = []
Latitude = []
```

Step 1

```
# Call getLaunchSite
getLaunchSite(data)

# Call getBoosterVersion
getBoosterVersion(data)

# Call getPayload
getPayloadData(data)

# Call getCoreData
getCoreData(data)
```

Step 2

```
launch_dict = {'FlightNumber':
list(data['flight_number']),
'Date': list(data['date']),
'BoosterVersion':BoosterVersion,
'PayloadMass':PayloadMass,
'Orbit':Orbit,
'LaunchSite':LaunchSite,
'Outcome':Outcome,
'Flights':Flights,
'GridFins':GridFins,
'Reused':Reused,
'Legs':Legs,
'LandingPad':LandingPad,
'Block':Block,
'ReusedCount':ReusedCount,
'Serial':Serial,
'Longitude': Longitude,
'Latitude': Latitude}
```

Step 3

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

Step 4

GitHub Link

Data Collection (SPACEX REST API) Cont.

GitHub Link

3) Filtering dataframe to only include Falcon 9 launch data and remove missing values

- Step 1: Using the BoosterVersion cloumn to keep Falcon 9 launches
- Step 2: Calculate the mean of PayloadMass using .mean() and using .replce() to replace any missing values (np.nan) in the data with the PayloadMass mean value

Step 1

```
data_falcon9 =
df[df['BoosterVersion'] !=
'Falcon 1']
print(data_falcon9)

# Count the number of null
values in each column
null_counts =
data_falcon9.isnull().sum()
```

Step 2

```
# 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)

# Check the updated DataFrame
print(data_falcon9)

# Check for any remaining NaN values
print(data_falcon9.isnull().sum())
```

Data Collection (Web Scraping)

GitHub Link

1) Extract a Falcon 9 launch records from HTML page

- Step 1: Using HTTP GET method to request from URL Falcon 9 launch data
- Step 2: Create BeautifulSoup object from HTML response
- Step 3: Extract all the column header names from the HTML table

Step 1

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

# assign the response to a object
obj = r.text
```

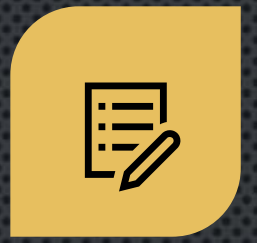
Step 2

```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(obj, 'html')
```

Step 3

```
# Use the find_all function in the BeautifulSoup object, with element type `table`
html_tables = soup.find_all('table’)

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

column_names = []

# Apply find_all() function with `th` element on first_launch_table
for r in first_launch_table.find_all('th'):
    c_name = extract_column_from_header(r)
    if (c_name != None and len(c_name) > 0):
        column_names.append(c_name)
print(column_names)
```

Data Collection (Web Scraping) cont.



2) Parse the launch HTML table and convert it into a dataframe

- Step 1: Create empty dictionary with keys from the column names
- Step 2: Parsing launch record values from the new dictionary
- Step 3: Create dataframe

```
launch_dict= dict.fromkeys(column_names)
print(launch_dict)
# 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']=[]
```

Step 1

```
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable
plainrowheaders collapsible")):
   # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number
corresponding to launch a number
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
        else:
            flag=False
```

Step 2

```
# Assuming launch_dict is a dictionary with
keys as column names and values as lists
df = pd.DataFrame({key: pd.Series(value,
dtype='object') for key, value in
launch_dict.items()})
```

Step 3

GitHub Link

Data Wrangling

GitHub Link

In the next slides the Logic and Code

Loading in dataset into a dataframe → Finding patterns in the data → Define labels to the data for supervised model training

Code

```
df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_1.csv")
df.head(10)
```

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577366	28.561857
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366	28.561857
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.577366	28.561857
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	0	B1003	-120.610829	34.632093
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561857
5	6	2014-01-06	Falcon 9	3325.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1005	-80.577366	28.561857
6	7	2014-04-18	Falcon 9	2296.000000	ISS	CCAFS SLC 40	True Ocean	1	False	False	True	NaN	1.0	0	B1006	-80.577366	28.561857
7	8	2014-07-14	Falcon 9	1316.000000	LEO	CCAFS SLC 40	True Ocean	1	False	False	True	NaN	1.0	0	B1007	-80.577366	28.561857
8	9	2014-08-05	Falcon 9	4535.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1008	-80.577366	28.561857
9	10	2014-09-07	Falcon 9	4428.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1011	-80.577366	28.561857

Data Wrangling (cont.)

1) Finding patterns in the data

- *Task 1: Calculate the number of launches on each site*
  - There are 3 launch sites
- *Task 2: Calculate the number and occurrences of each orbit*
  - There are 11 different orbits
- *Task 3: Calculate the number and occurances of mission outcome of the orbits*
  - There are 8 different landing outcomes
    - True ASDS: Successfully landed on a drone ship
    - False ASDS: Successfully landed on a drone shiip
    - None ASDS: Failed to Land
    - True RTLS: Unsuccessfully landed to a ground pad
    - False RTLS: Unsuccessfully landed to a ground pad
    - True Ocean: Successfully landed to a certain region in ocean
    - False Ocean: Unsuccessfully landed to a certain region in ocean
    - None None: Failed to Land

GitHub Link

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

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

Task 1

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

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

Task 2

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

```
True ASDS      41
None None      19
True RTLS      14
False ASDS      6
True Ocean      5
False Ocean     2
None ASDS       2
False RTLS      1
Name: Outcome, dtype: int64
```

Task 3

Data Wrangling (cont.)

2) Define labels to the data for supervised model training

- Convert the various mission outcomes into binary labels
  - 1 for Successful landing
    - True ASDS: Successfully landed on a drone ship
    - True RTLS: Unsuccessfully landed to a ground pad
    - True Ocean: Successfully landed to a certain region in ocean
  - 0 for Unsuccessful landing
    - False ASDS: Successfully landed on a drone shiip
    - None ASDS: Failed To Land
    - False RTLS: Unsuccessfully landed to a ground pad
    - False Ocean: Unsuccessfully landed to a certain region in ocean
    - None None: Failed To Land

GitHub Link

```
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
```

Step 1

Enumerating all the mission outcomes from 0 to 7 labels instead of using their string name as the key

```
bad_outcome=set(landing_outcomes.keys()[[1,3,5,6,7]])
bad_outcome
```

Step 2

```
{'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

Using the number labels that correspond to the mission outcomes that did not successfully land and adding them all to a variable called `bad_outcome`

```
df['Class']=landing_class
df[['Class']].head(8)
```

	Class
0	0
1	0
2	0
3	0
4	0
5	0
6	1
7	1

Step 4

Output of a new column in the DataFrame called "Class" that stores the binary labels for the mission outcomes

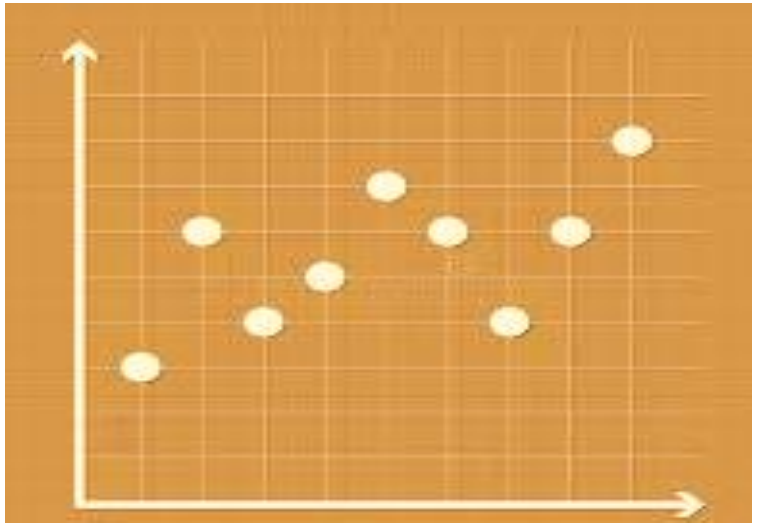
```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise

landing_class = []

for outcome in df["Outcome"]:
    if outcome in bad_outcome:
        landing_class.append(0)
    else:
        landing_class.append(1)
```

Step 3

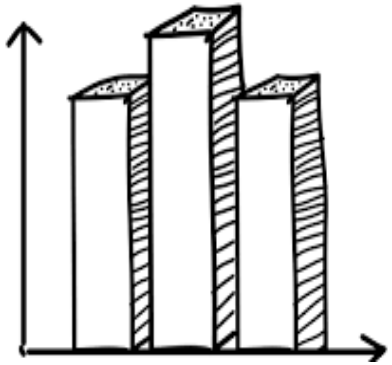
Using the "Outcome" column to check if any values in the column equal values in the "bad_outcome" variable, then set them to 0; otherwise, set them to 1 into a list called "landing_class"

EDA WITH DATA VISUALIZATION

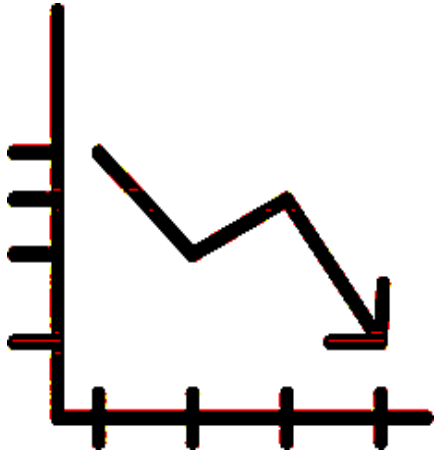
Scatter Plot



Bar Chart



Line Chart



1) SCATTER PLOT: USED TO OBSERVE THE RELATIONSHIP/CORRELATION BETWEEN TWO OR MORE NUMERIC VARIABLES

- RELATIONSHIP BETWEEN FLIGHT NUMBER AND LAUNCH SITE
- RELATIONSHIP BETWEEN PAYLOAD AND LAUNCH SITE
- RELATIONSHIP BETWEEN FLIGHT NUMBER AND ORBIT TYPE
- RELATIONSHIP BETWEEN PAYLOAD AND ORBIT TYPE

2) BAR CHART: USED TO COMPARE NUMERICAL VALUES WITH CATEGORICAL VARIABLES AND OBSERVE THE HIGHEST OCCURRING GROUPS COMPARED TO THE OTHER GROUPS

- RELATIONSHIP BETWEEN SUCCESS RATE OF EACH ORBIT TYPE

3) LINE CHART: USED TO SHOW THE CHANGE OF A CERTAIN NUMERICAL VARIABLE OVER TIME

- RELATIONSHIP BETWEEN SUCCESS RATE AND YEAR

GitHub Link

EDA with SQL

Using SQL queries and operations to better understand the SpaceX dataset:

1. Display the names of the unique launch sites in the space mission
2. Display 5 records where the launch begins with the string "CCA“
3. Display the total payload mass carried by boosters launched by NASA (CRS)
4. Display the average payload mass carried by booster version F9. v1.1
5. List the data when the first successful landing outcome on a ground pad was achieved
6. List the names of the boosters that had successfully landed on a drone ship with a payload mass between 4000 and 6000 kg
7. List the total number of successful and failed mission outcomes
8. List the names of the booster versions that have carried the maximum payload mass
9. List the failed landing outcomes on a drone ship, including their booster versions and launch site names for 2015.
10. Rank the count of landing outcomes between the dates 2010-06-04 and 2017-03-20 in descending order

- Failure: drone ship
- Success: ground pad

GitHub Link

Interactive Map with Folium

Using Folium interactive maps to help analyze the geospatial data to better visualize the location and proximity of the launch sites and how that might impact the launch success rate

- **Step 1:** Mark all the launch sites
  - Initialize the map using a Folium Map object
  - Use `folium.Circle` and `folium.Marker` to highlight each launch site on the map with a circle

- **Step 2:** Mark all the successful/failed launches from each site
  - Use `MarkerCluster()`
  - Green indicates a successful launch, and red indicates a failed launch

- **Step 3:** Calculate the distances between the launch sites and their proximities using latitude and longitude values
  - For coastline, railroad, highway, and cities
  - First, use `MousePosition()` to help mark the latitude and longitude values
  - Create a marker using `folium.Marker()`
  - Draw using `folium.Polyline()`
  - Report the steps above for each proximity from launch sites using markers and drawing lines

- **Step 4:** Answer the following questions
  1. Are launch sites in close proximity to railways?
     - Yes
  2. Are the launch sites in close proximity to highways?
     - Yes
  3. Are the launch sites in close proximity to the coastline?
     - Yes
  4. Do the launch sites keep a certain distance away from cities?
     - Yes

GitHub Link

Interactive Plotly Dashboard

Using Plotly API in Python to create an interactive dashboard consisting of a pie chart and a scatter plot

- *Pie Chart:* Displays successful launches per site
  - Interaction Options:
    - Clicking on the dropdown menu for all sites shows the successful launches for all sites as a percentage in a pie chart.
    - Clicking on the dropdown menu for a specific launch site shows the success versus failed launches from that site as a percentage in the pie chart.

- *Scatter Plot:* Shows the correlation between the outcome of launches and the payload mass
  - Interactive Filtering:
    - Use the filter to select the minimum and maximum range of the payload mass in kilograms.

GitHub Link

Predictive Analysis (Classification)

Below are the steps to create a model, evaluate the model, and find the most optimal classification model

Create Models

Steps:

1. Load the Dataset
2. Perform Data Standardization and Preprocessing
3. Split the Data
   - Training set
   - Testing set
4. Run Different Classification Algorithms with the Data
5. Create an object for each of those algorithms
   - Create a GridSearchCV object and pass the algorithm objects and the required hyperparameters

Evaluate Models

Steps:

1. GridSearchCV Outputs:
   - Best parameters: Hyperparameters
   - Best score: Accuracy on the validation data
2. Create Confusion Matrix
   - For each algorithm
3. Examine Confusion Matrix
   - For each algorithm

Find Optimal Classification Model

Steps:

1. Review the accuracy scores
   - For all classification algorithms
2. Find the highest accuracy
   - Will help identify the best performing model

GitHub Link

RESULTS

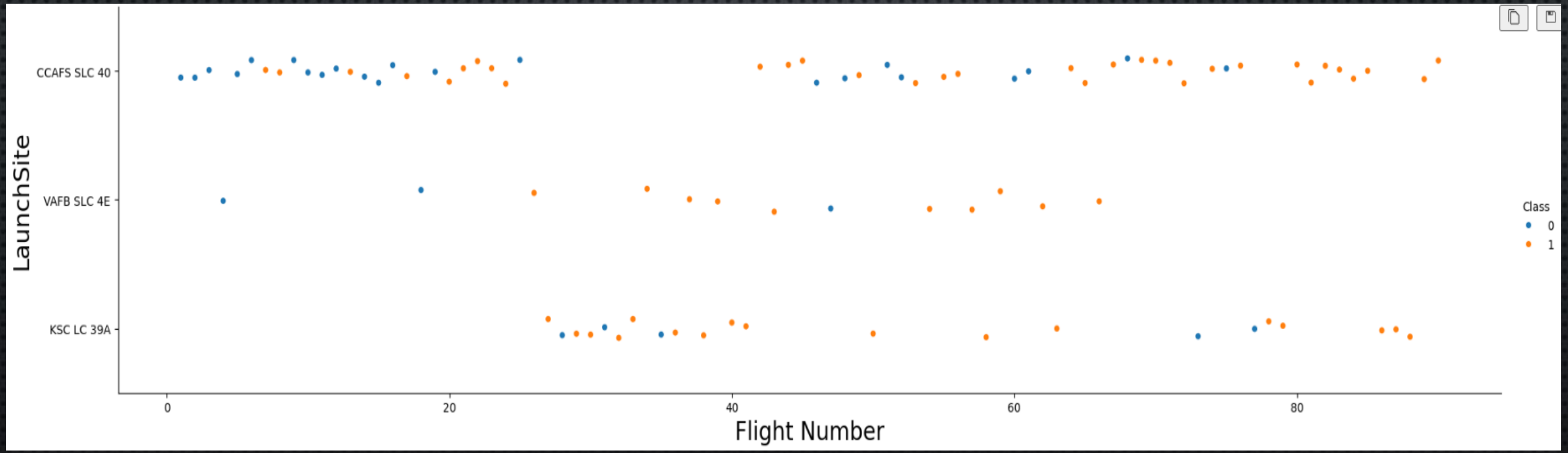
- Overview
  - Exploratory data analysis (EDA) results
  - Interactive analytics demo in screenshots
  - Predictive analysis results

EDA WITH VISUALIZATION

Flight Number vs. Launch Site

Output

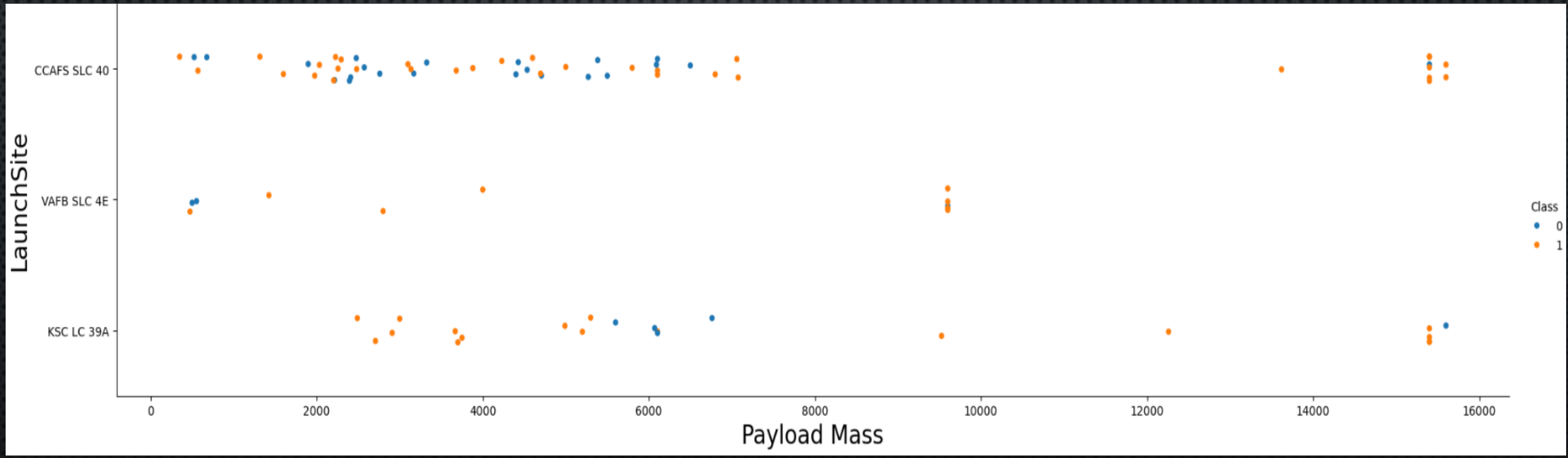
- The plot shows the relationship between the 3 launch sites on the y-axis and the flight number on the x-axis.
- The classes 1 and 0 represent successful and failed missions, respectively.
- The pattern we can observe is that earlier flight numbers, such as those from launch site CCAFS 5LC 40, had more failures initially, while later flight launches tend to be more successful. This trend is evident across the other launch sites as well.



Payload vs. Launch Site

Output

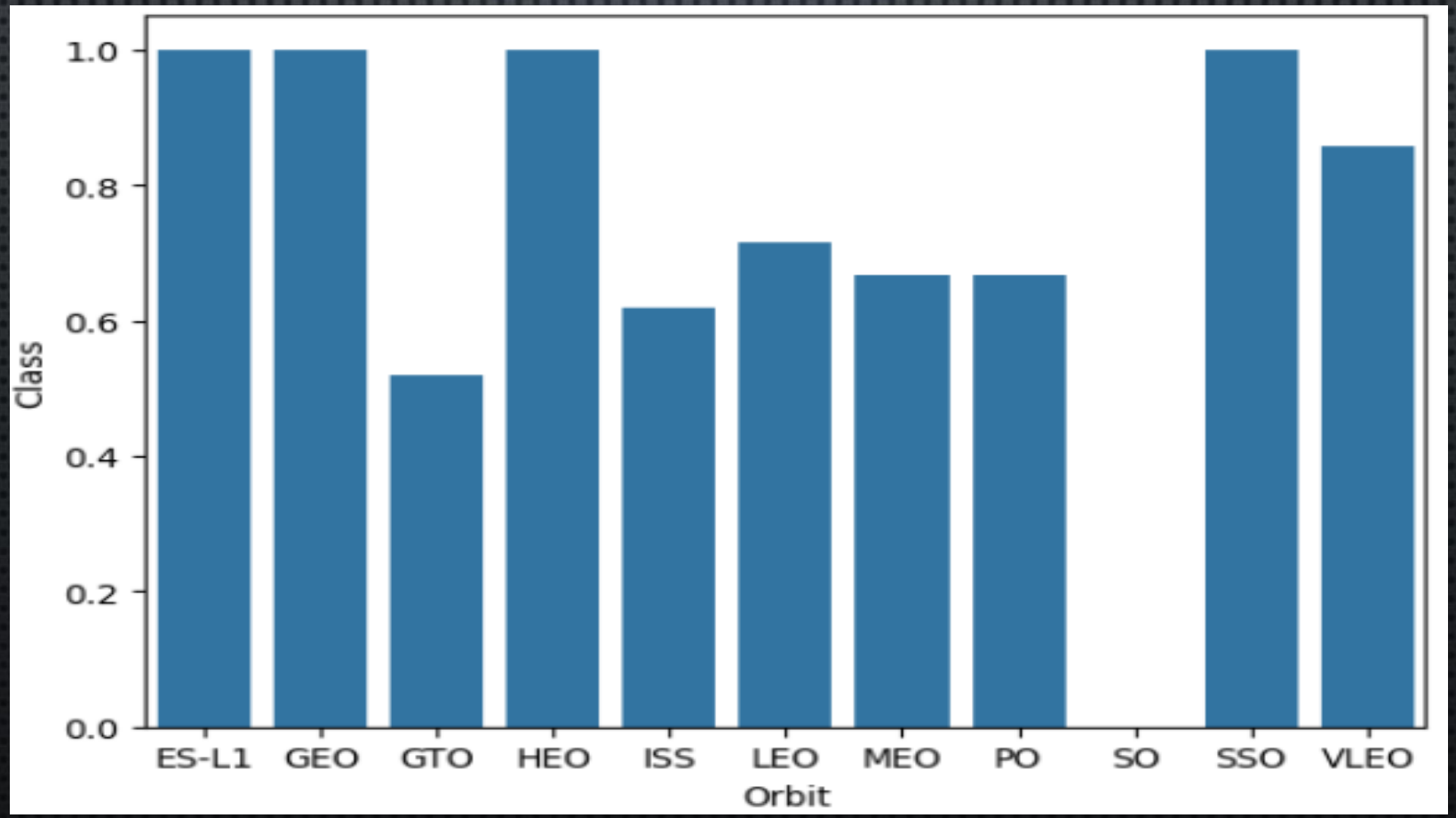
- The plot shows the relationship between the 3 launch sites on the y-axis and the payload mass on the x-axis.
- The classes 1 and 0 represent successful and failed missions, respectively.
- We can observe that the flights tend to have a payload mass between 0kg to 7500kg. There are very few flights with a payload mass between 9000kg to 16000kg. Interestingly, at this higher payload mass range, there were fewer failures in the flight launches.



Success Rate vs. Orbit Type

Output

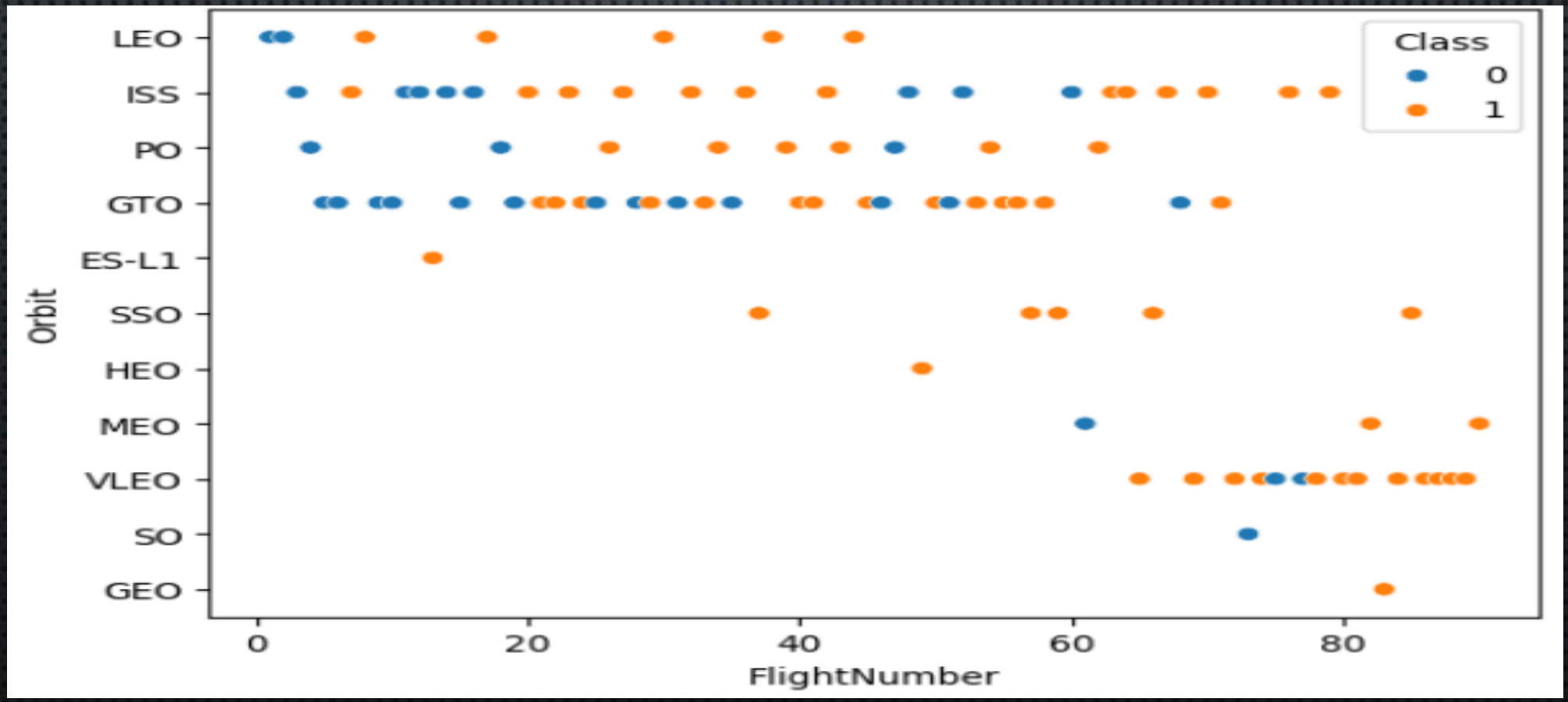
- The plot shows the relationship between the class (succes rate) on the y-axis and the orbit type on the x-axis.



Flight Number vs. Orbit Type

Output

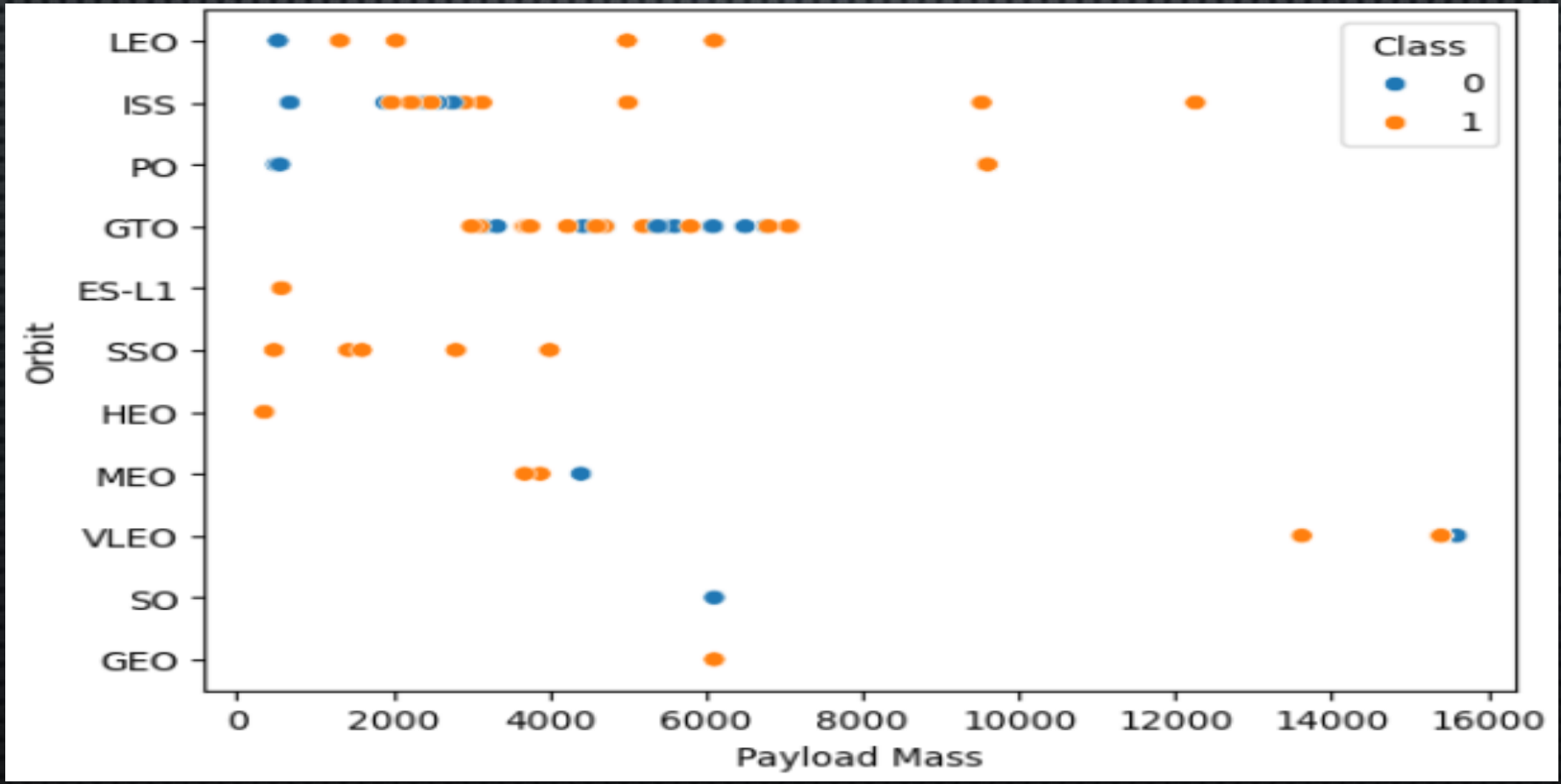
- The plot shows the relationship between the orbit on the y-axis and the flight number on the x-axis.
- The classes 1 and 0 represent successful and failed missions, respectively.



Payload vs. Orbit Type

Output

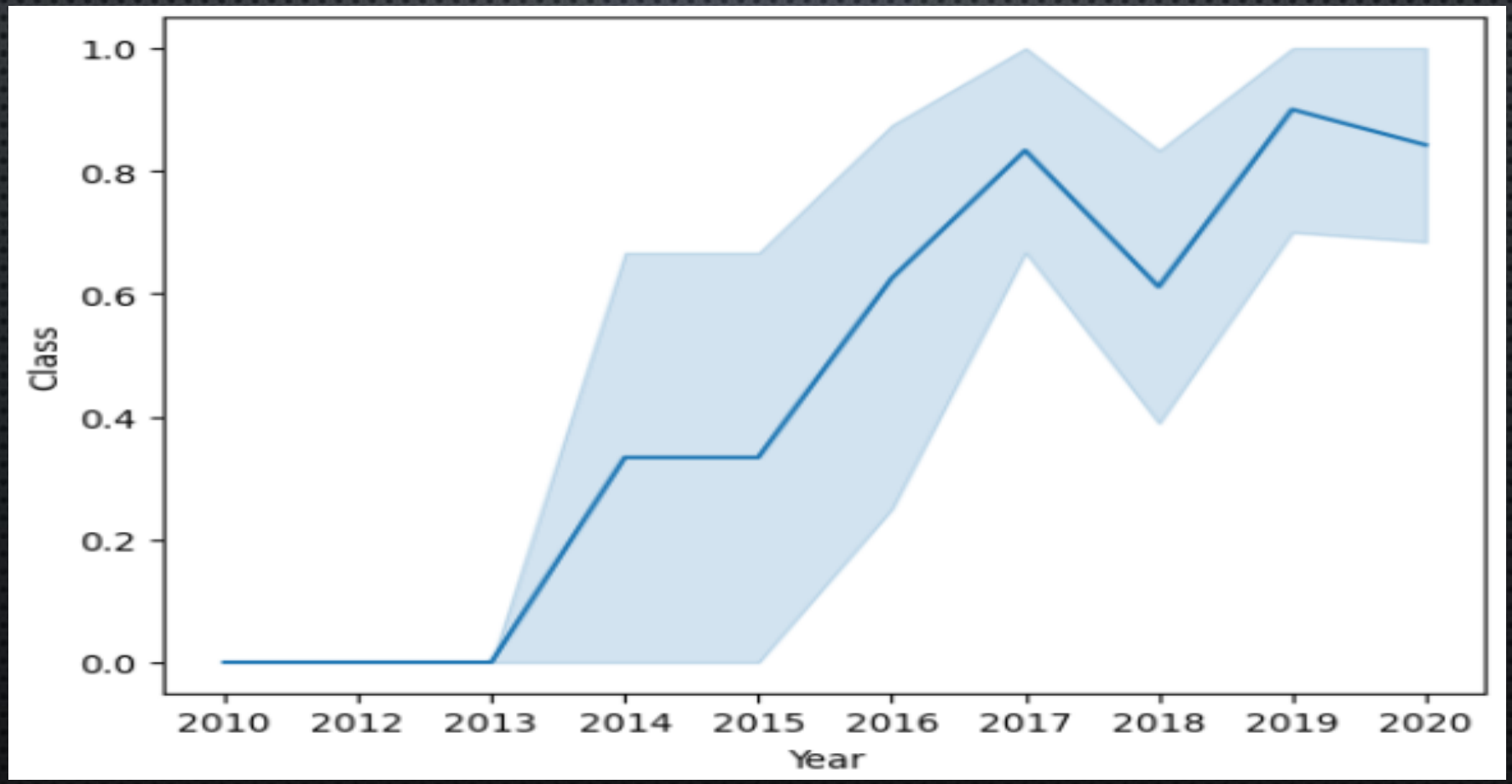
- The plot shows the relationship between the orbit on the y-axis and the flight number on the x-axis.
- The classes 1 and 0 represent successful and failed missions, respectively.



Launch Success Yearly Trend

Output

- The plot shows the relationship between the orbit on the y-axis and the flight number on the x-axis.



EDA WITH SQL

Launch Site Names

Query Task

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

SQL QUERY

- %sql SELECT DISTINCT LAUNCH_SITE FROM SPACEXTABLE

Output

- Shows that there are 4 launch sites and their names

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

Launch Site Names Begin with 'CCA'

Query Task

- Display 5 records where launch sites begin with the string ‘CCA’

SQL QUERY

- %sql SELECT * FROM SPACEXTABLE WHERE launch_site LIKE 'CCA%' LIMIT 5

Output

- Shows five records of launch sites whose names start with 'CCA,' including additional columns of information from the dataset related to each launch site

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

Query Task

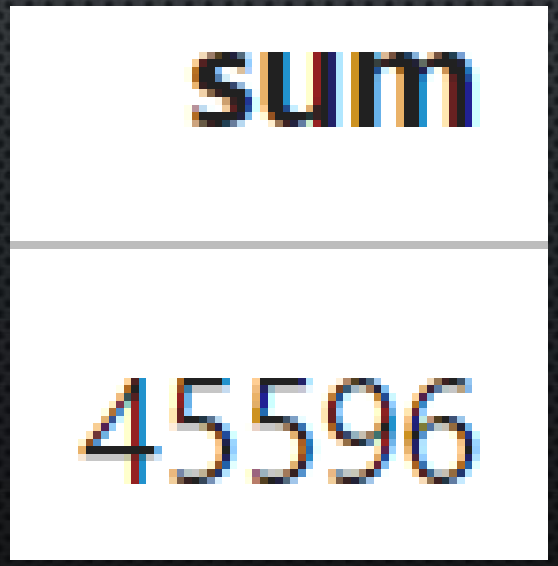
- Display the total payload mass carried by boosters launched by NASA (CRS)

SQL QUERY

- %sql SELECT SUM(payload_mass__kg_) as sum from SPACEXTABLE WHERE customer LIKE 'NASA (CRS)'

Output

- Shows the sum (or total) payload mass carried by bossters launched by NASA (CRS)



Payload Mass by F9 v1.1

Query Task

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

SQL QUERY

- %sql SELECT avg(payload_mass__kg_) as Average FROM SPACEXTABLE WHERE booster_version like 'F9 v1.1%'

Output

- Shows the average of the paylaod mass carried by the bosster version F9 v1.1

Average
2534.6666666666665

First Successful Ground Landing Date

Query Task

- List the date when the first succesful landing outcome in ground pad was achieved

SQL QUERY

- %sql SELECT MIN(date) as Date FROM SPACEXTABLE WHERE mission_outcome LIKE 'Success' AND landing_outcome LIKE 'Success (ground pad)'

Output

- Shows the first successful landing outcome in ground pad date

Date
2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

Query Task

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

SQL QUERY

- %sql SELECT booster_version FROM SPACEXTABLE WHERE (mission_outcome LIKE 'Success') AND (payload_mass__kg_ > 4000 AND payload_mass__kg_ < 6000) AND (landing_outcome LIKE 'Success (drone ship)')

Output

- Shows the 4 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

Total Number of Successful and Failed Mission Outcomes

Query Task

- List the total number of successful and failure mission outcomes

SQL QUERY

- %sql SELECT MIN(date) as Date FROM SPACEXTABLE WHERE mission_outcome LIKE 'Success' AND landing_outcome LIKE 'Success (ground pad)'

Output

- Shows list of the counts of the mission outcomes

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

Boosters Carried Maximum Payload

Query Task

- List the names of the booster_versions which have carried the maximum payload mass

SQL QUERY

- %sql SELECT booster_version FROM SPACEXTABLE WHERE payload_mass__kg_ = (SELECT MAX(payload_mass__kg_) FROM SPACEXTABLE)

Output

- Shows list of names of the booster_version which have carried the max payload mass

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

Query Task

- 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

SQL QUERY

- %sql SELECT CASE WHEN substr(Date, 6, 2) = '01' THEN 'January' WHEN substr(Date, 6, 2) = '02' THEN 'February' WHEN substr(Date, 6, 2) = '03' THEN 'March' WHEN substr(Date, 6, 2) = '04' THEN 'April' WHEN substr(Date, 6, 2) = '05' THEN 'May' WHEN substr(Date, 6, 2) = '06' THEN 'June' WHEN substr(Date, 6, 2) = '07' THEN 'July' WHEN substr(Date, 6, 2) = '08' THEN 'August' WHEN substr(Date, 6, 2) = '09' THEN 'September' WHEN substr(Date, 6, 2) = '10' THEN 'October' WHEN substr(Date, 6, 2) = '11' THEN 'November' WHEN substr(Date, 6, 2) = '12' THEN 'December' ELSE 'Unknown' END AS Month, landing_outcome, booster_version, launch_site FROM SPACEXTABLE WHERE Date LIKE '2015%' AND landing_outcome LIKE 'Failure (drone ship)'

Output

- Shows the Months of when the landing outcome is failure to Land in drone ship, booster versions, launch site for the months in year 2015

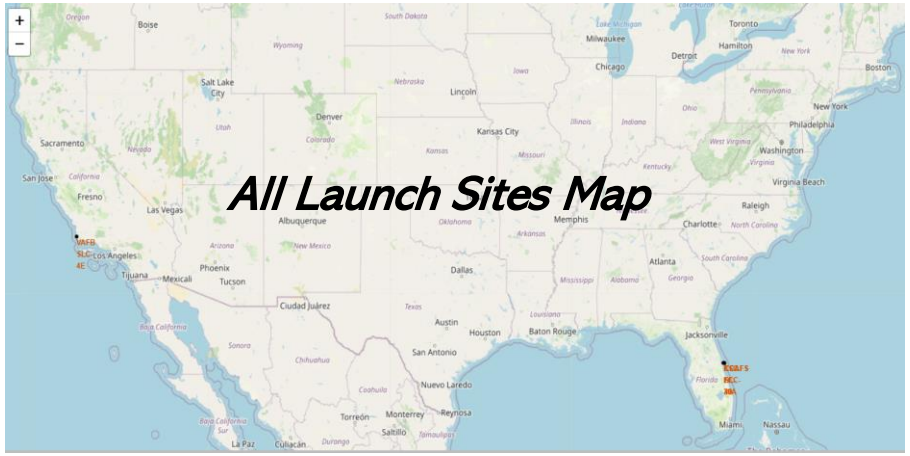
Month	Landing_Outcome	Booster_Version	Launch_Site
January	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
April	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

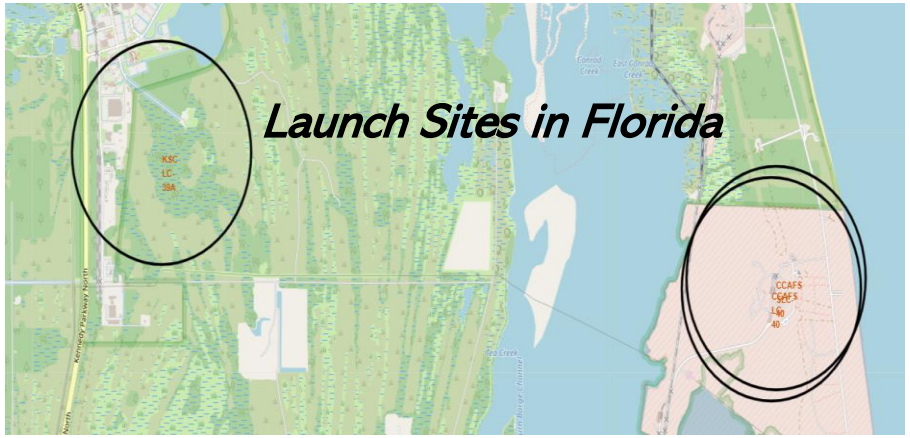
RANK LANDING OUTCOMES BETWEEN 2010-06-04 AND 2017-03-20

- **Query Task**
  - 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 QUERY**
  - %sql SELECT landing_outcome, count(*) as Count FROM SPACEXTABLE WHERE Date >= '2010-06-04' AND Date <= '2017-03-20' GROUP by landing_outcome ORDER BY count DESC
- **Output**
  - Show all landing outcomes along with the number of times each outcome occurred, sorted in descending order of occurrence

Landing_Outcome	Count
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

LAUNCH SITES PROXIMITIES ANALYSIS





MARK ALL LAUNCH SITES ON A MAP

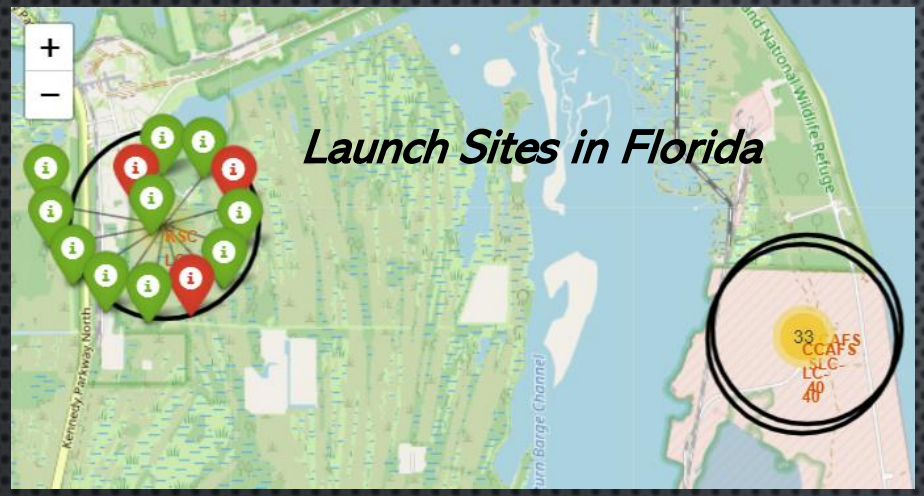
The 4 main rocket launch sites are loacted at:

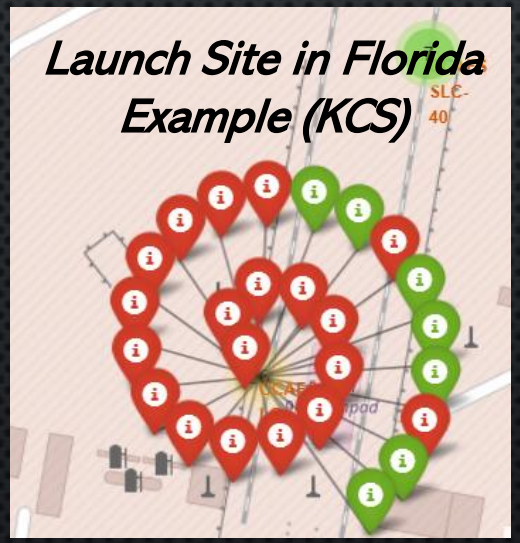
1. **CCAFS LC-40:** Cape Canaveral, Florida
2. **CCAFS SLC-40:** Cape Canaveral, Florida,
3. **KSC LC-39A:** In Kennedy Space Center, Florida
4. **VAFB SLC-4E:** In Vandenberg Air Force Base, California

EDA Questions:

- **Proximity to Equator:** Not all sites are directly on the Equator line.
- **Proximity to Coast:** All sites are near the coast. CCAFS LC-40, CCAFS SLC-40 and KSC are on the east coast in Florida, and VAFB is on the west coast in California

MARK SUCCESS/FAILED LAUNCHES FOR EACH SITE





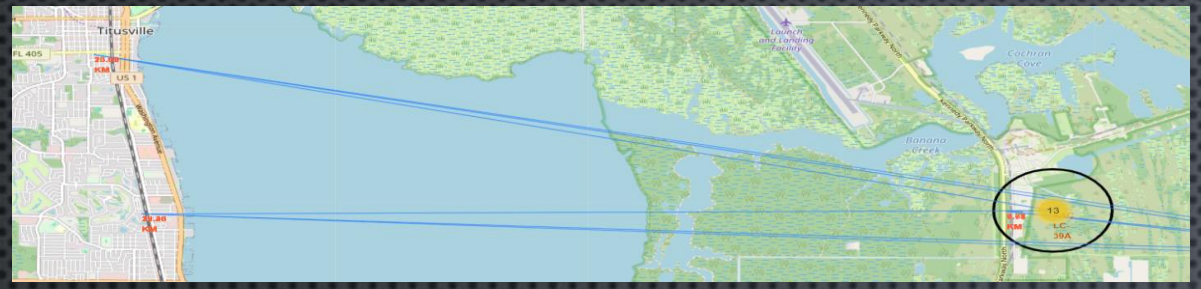
THE 4 MAIN ROCKET LAUNCH SITES ARE LOACTED AT:

- **CCAFS LC-40**: CAPE CANAVERAL, FLORIDA
- **CCAFS SLC-40**: CAPE CANAVERAL, FLORIDA,
- **KSC LC-39A**: IN KENNEDY SPACE CENTER, FLORIDA
- **VAFB SLC-4E**: IN VANDENBERG AIR FORCE BASE, CALIFORNIA

MAP IMAGES:

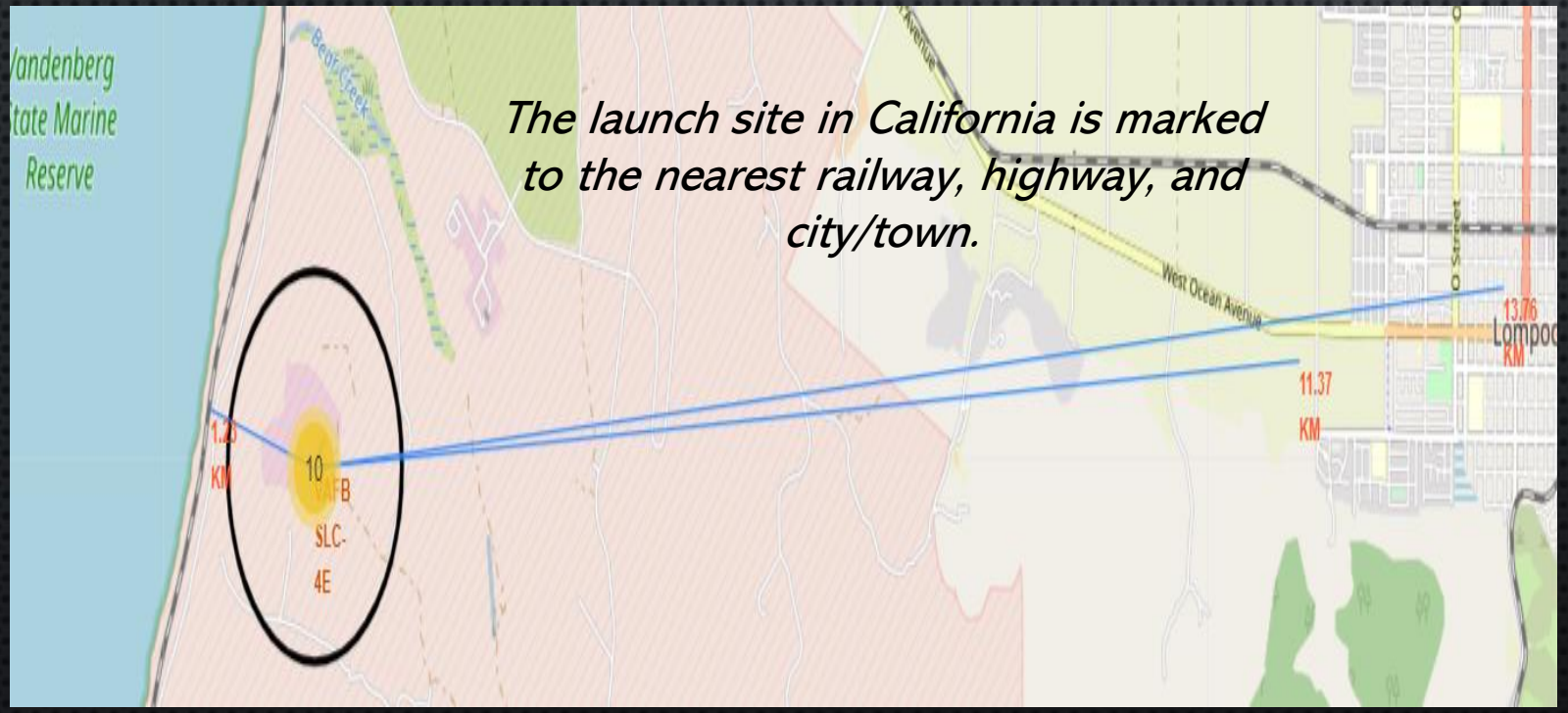
- THE LEFT IMAGE DISPLAYS ALL LAUNCHES AT THE FOUR SITES, WITH RED INDICATING A FAILED LAUNCH AND GREEN INDICATING A SUCCESSFUL LAUNCH
- HOVERING OVER EACH COLORED MARKER REVEALS THE LAUNCH'S NAME

DISTANCES BETWEEN A LAUNCH SITE TO ITS PROXIMITIES



One of the launch sites in Florida is marked to the nearest railway, highway, and city/town.

All three launch sites in Florida are marked to the nearest railway, highway, and city/town



The launch site in California is marked to the nearest railway, highway, and city/town.

DISTANCES BETWEEN A LAUNCH SITE TO ITS PROXIMITIES (CONT.)

QUESTIONS:

1) ARE LAUNCH SITES CLOSE TO THE COASTLINES?

- ALL SITES ARE NEAR THE COAST. CCAFS LC-40, CCAFS SLC-40 AND KSC ARE ON THE EAST COAST IN FLORIDA, AND VAFB IS ON THE WEST COAST IN CALIFORNIA (ALSO ANSWERED ON SLIDE 41)

2) ARE LAUNCH SITES CLOSE TO RAILWAYS?

- NO, ONLY THE LAUNCH SITE IN CA IS CLOSE TO A RAILWAY, WITH A DISTANCE OF 1.33 KM. ALL OTHER LAUNCH SITES ARE AROUND 20 TO 30 MILES AWAY FROM RAILWAY

3) ARE LAUNCH SITES CLOSE TO HIGHWAYS?

- NO, ONLY THE FLORIDA LAUNCH SITES ARE CLOSE TO A HIGHWAY, UNDER 10 KM

4) DO THE LAUNCH SITES KEEP A CERTAIN DISTANCE FROM CITIES/TOWNS?

- YES, THEY DO, AND THE RANGE OF THIS DISTANCE IS 13 TO 27 MILES

MAP IMAGES (SLIDE 43):

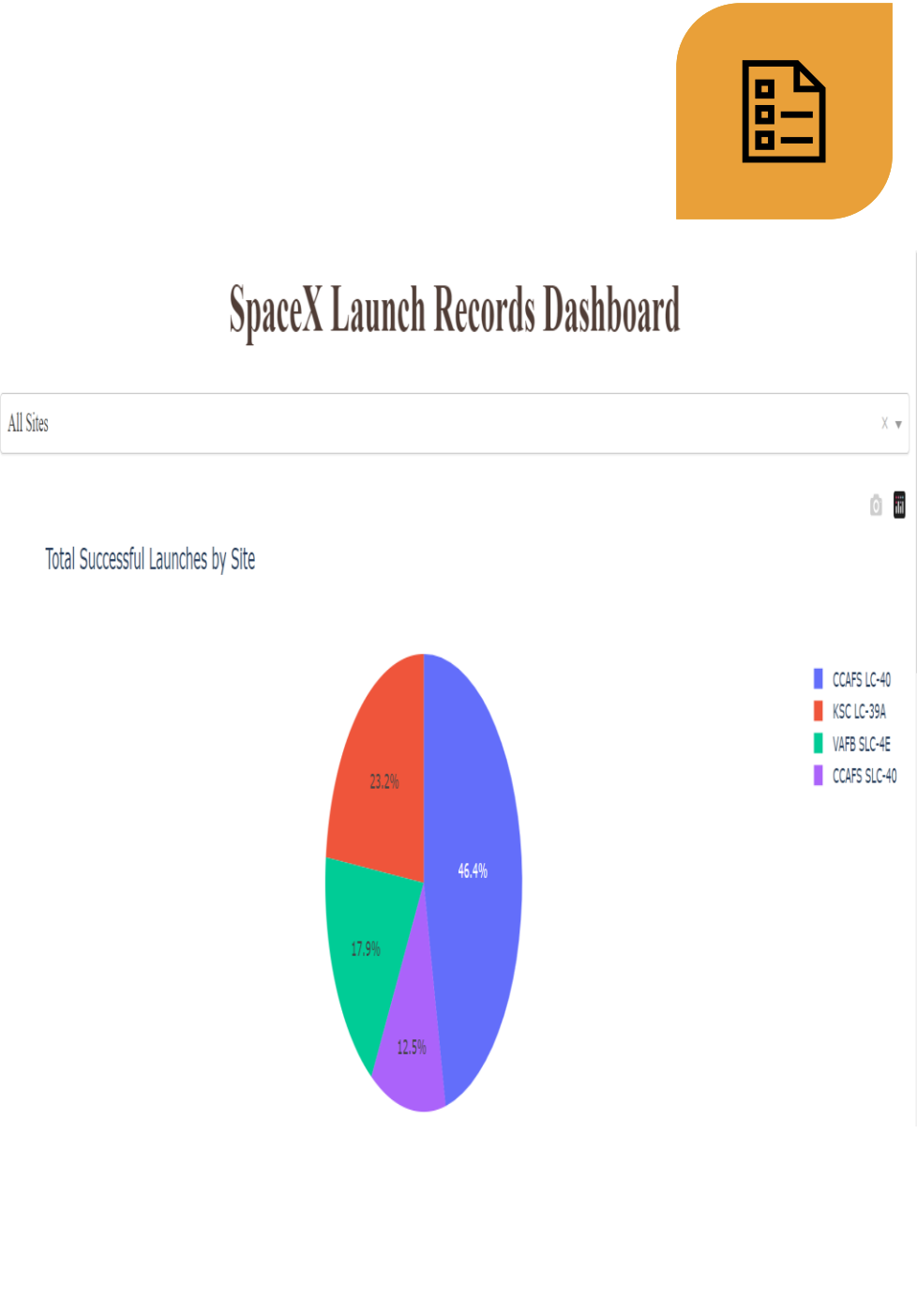
- THE IMAGES ON THE EARLIER SLIDE SHOW ALL THE DISTANCES FROM THE NEAREST RAILWAYS, HIGHWAYS, AND CITIES/TOWNS TO EACH OF THE 4 MAIN LAUNCH SITES IN CA AND FL AND DISPLAY THE DISTANCE IN KILOMETERS

INTERACTIVE DASHBOARD WITH PLOTLY DASH

LAUNCH SUCCESS COUNT FOR ALL SITE

DashBoard compontents

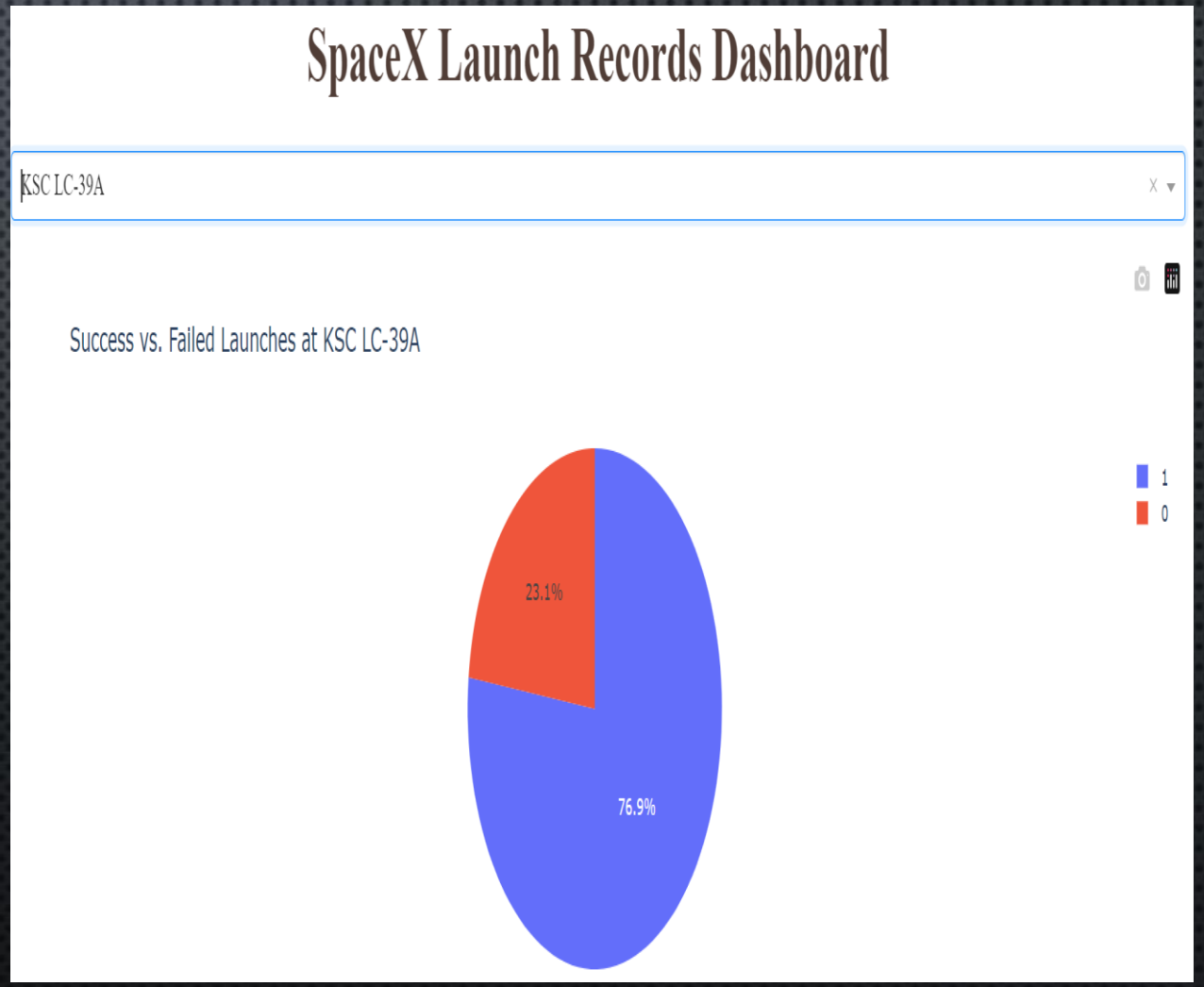
- A Dropdown menu of options to pick all the launch sites or certain ones to show their successful launches based on the selection, represented as a pie chart
- *Which site has the largest successful launches?*
  - This question is to find the total count of successful launches per site
    - Answer: CCAFS LC-40 site has the largest number of successful launches

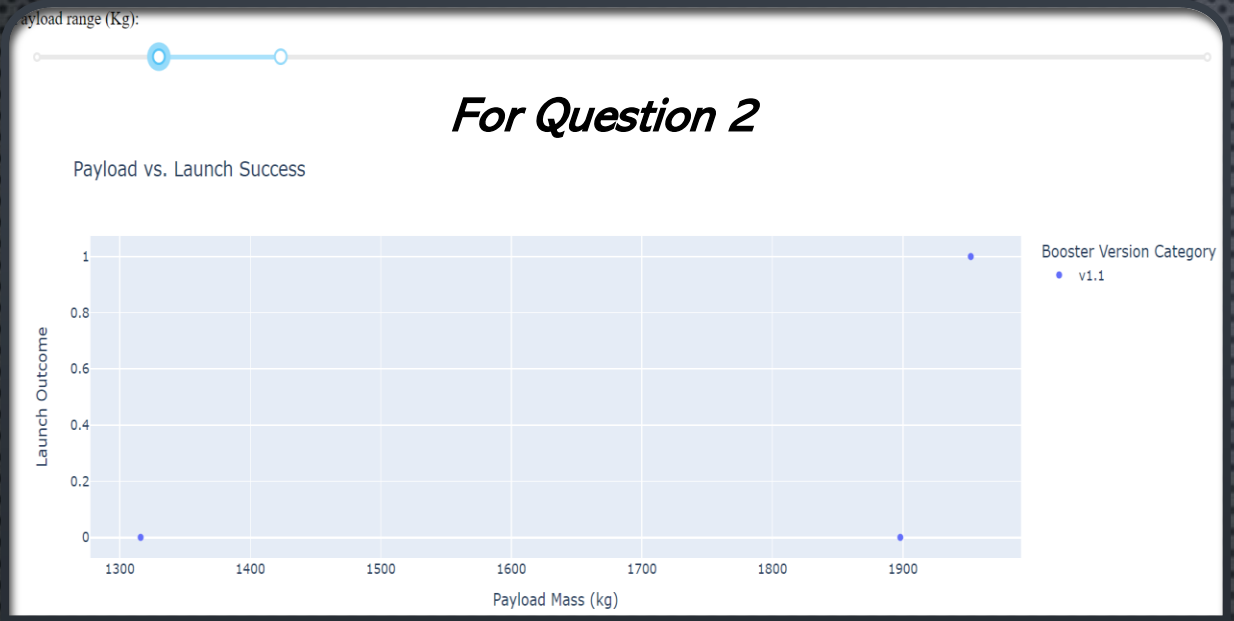


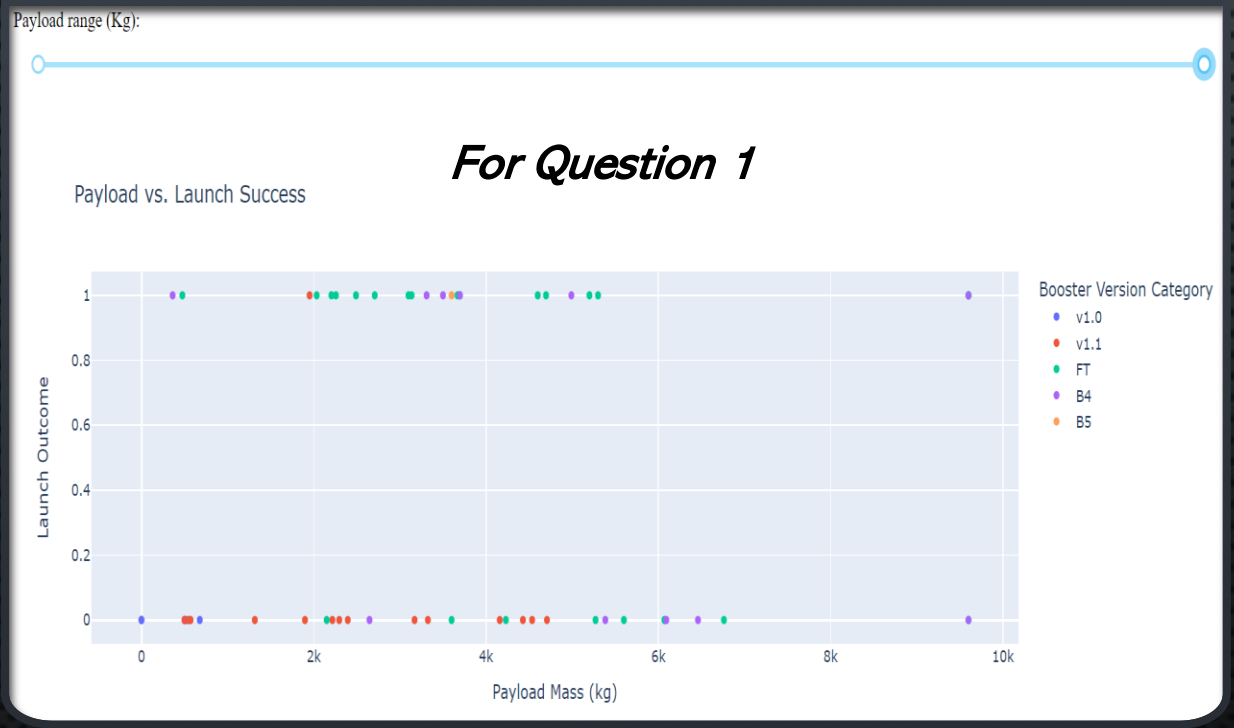
LAUNCH SUCCESS WITH HIGHEST LAUNCH SUCCESS RATIO

DASHBOARD COMPONTENTS

- WHEN SELECTING SPECIFIC LAUNCH SITES FROM THE DROPDOWN MENU, THE PIE CHART SHOWS THE PERCENTAGE OF SUCCESSFUL VS. FAILED LAUNCHES
- *WHICH SITE HAS THE HIGHEST LAUNCH SUCCESS RATE?*
  - THIS QUESTION LOOKS AT THE RATIO OF SUCCESSFUL LAUNCHES TO TOTAL LAUNCHES FOR EACH LAUNCH SITE
    - ANSWER: KSC LC-39A SITE HAS THE HIGHEST LAUNCH SUCCESS RATE. WITH A RATIO OF 76.9% (SUCESSFUL LAUNCHES) TO 23.1% (FAILED LAUNCHS)





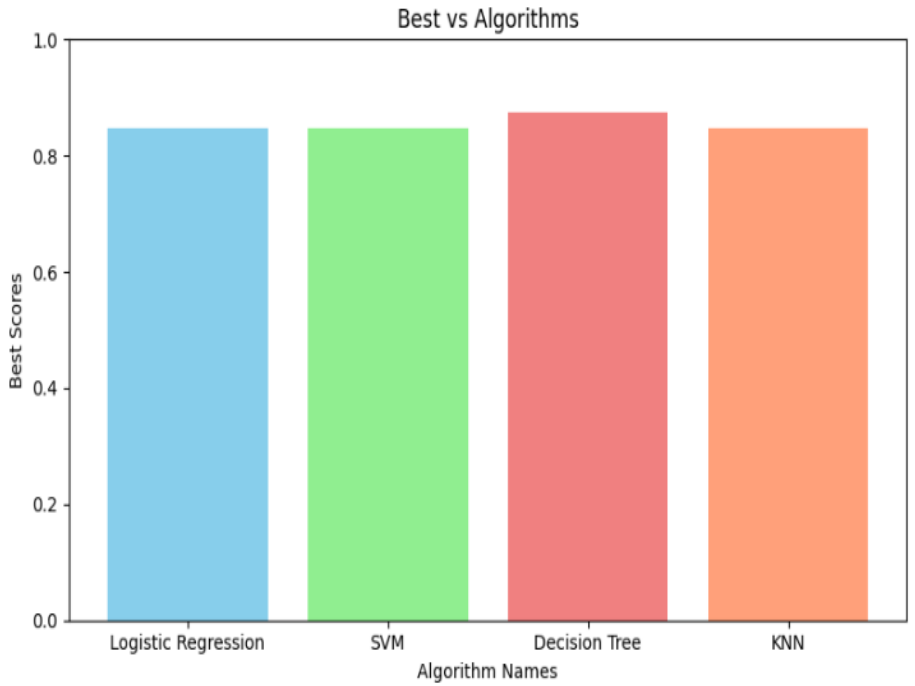


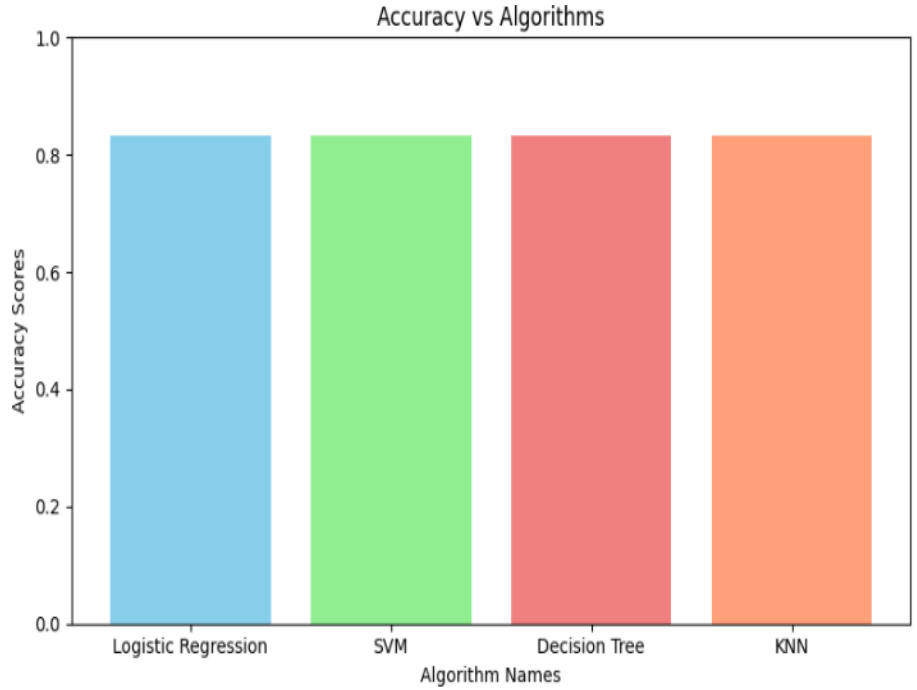
PAYLOAD VS. LAUNCH OUTCOME SCATTER PLOT FOR ALL SITES

Questions and Answers:

1. Which payload range has the highest launch success rate?
   - Answer: The range from 0kg to 10,000kg
     - 21 sucessful launchs
2. Which payload range has the lowest launch success rate?
   - Answer: The range from 1,300kg to 1,900kg
     - 1 Sucessful launch
3. Which Falcon 9 Booster version (v1.0, v1.1, FT, B4, and B5) has the highest launch success rate?
   - Answer: The Falcon 9 version FT

PREDICTIVE ANALYSIS (CLASSIFICATION ALGORITHMS)





CLASSIFICATION ACCURACY

Model with the Highest Classification Accuracy:

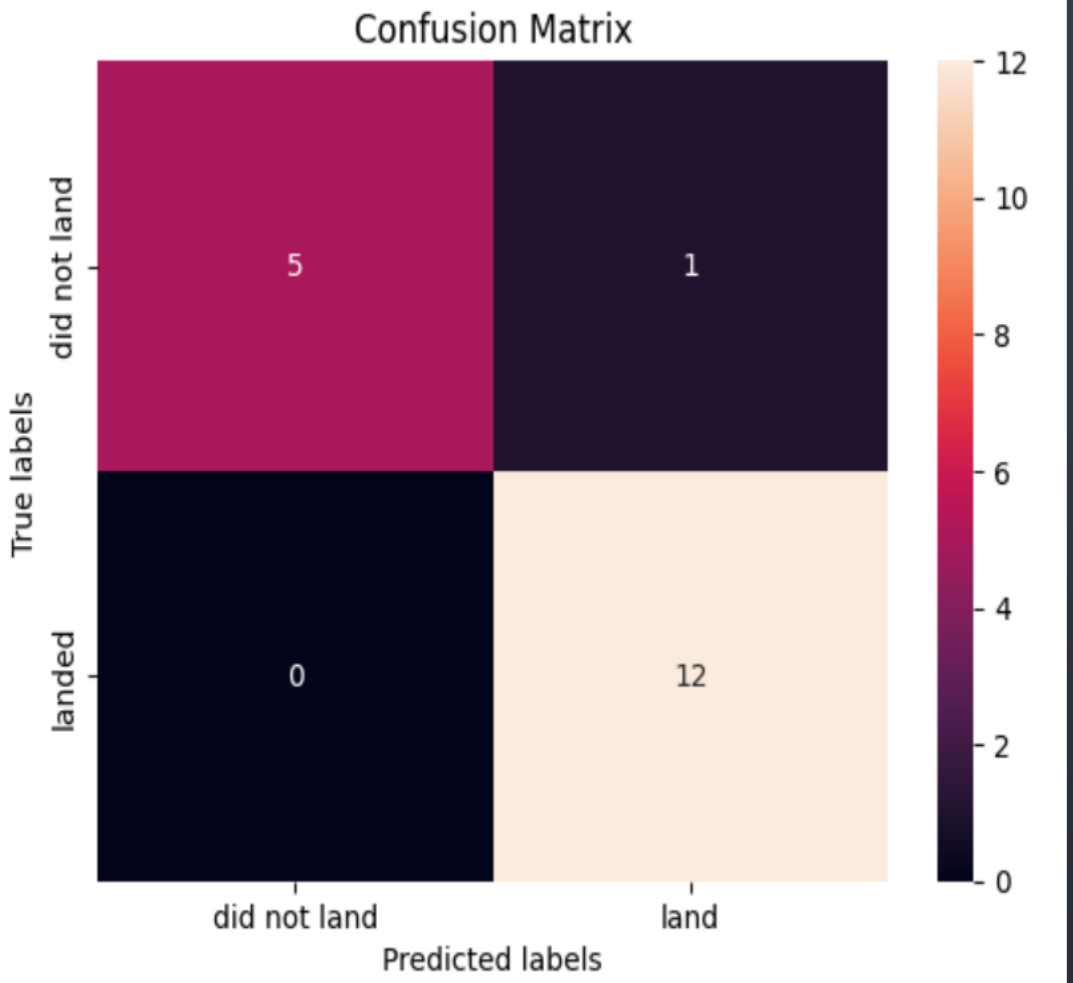
- Decision Tree model has the highest Best Score
  - Best Score: 87.50%
  - Accuracy Score: 83.33%

Bar Charts:

- Best Scores of All Algorithms
- Accuracy Scores of All Algorithms

	Accuracy Scores	Best Scores
Logistic Regression	0.833333	0.846429
SVM	0.833333	0.848214
Decision Tree	0.833333	0.875000
KNN	0.833333	0.848214

CONFUSION MATRIX



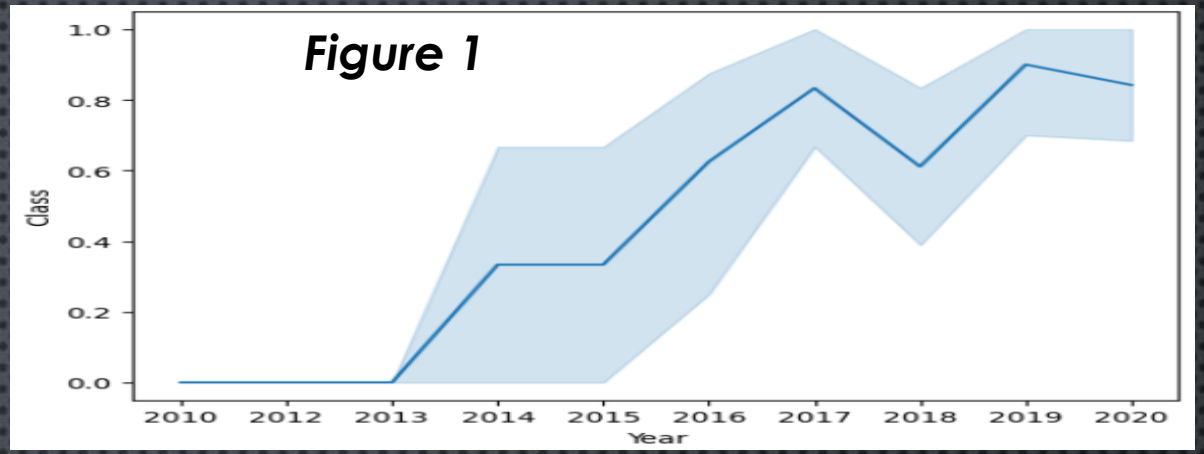
Decision Tree Confusion Matrix Analysis

- The confusion matrix represents a total of 18 results from the best classification algorithm.

1. The model performed well in predicting successful landings, 12 out of 18 correct predictions.
2. There were instances where the model incorrectly predicted a successful landing when it did not occur (1 out of 18).
3. The model's accuracy in predicting failed landings was 5 out of 18 correct predictions.

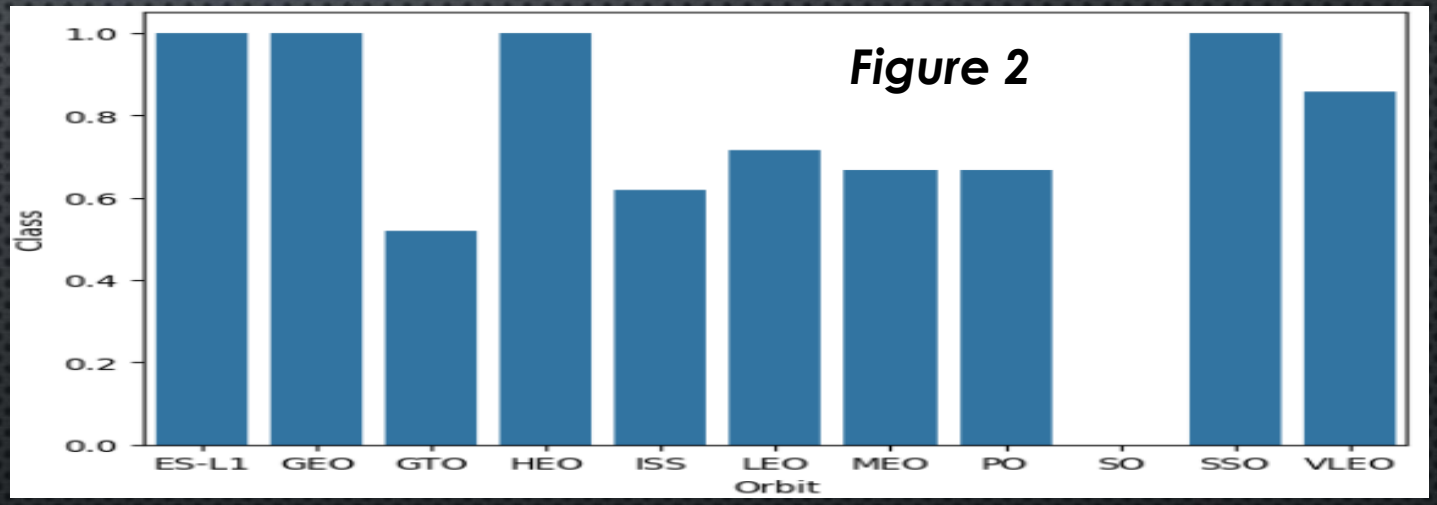
CONCLUSIONS

1) Over the years, there have been fewer failures in launches as the number of launches increased, especially after 2013, as depicted in Figure 1.

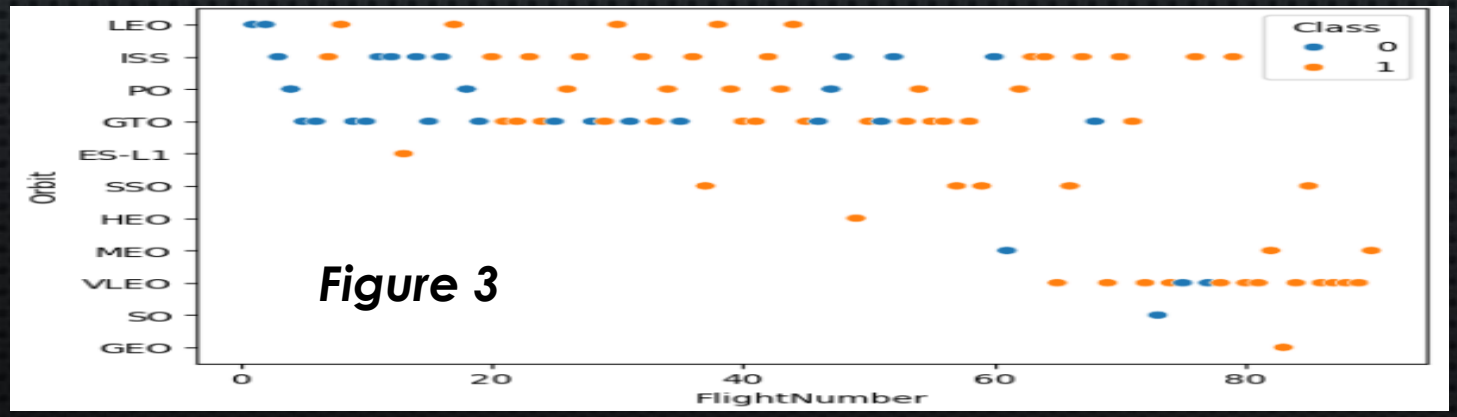


- From 2010 to 2013, there were no successful launches. However, from 2017 onwards, the line graph reveals a trend where SpaceX has consistently achieved a high success rate.
- Occasionally, there are slight dips or increases, suggesting that they have reached the maximum limit of their accuracy in flights.

2) The success of launches into different orbits shows the accuracy of rockets for each orbit, as illustrated in Figure 2.



- The data illustrates that orbits such as ES-L1, GEO, HEO, and SSO all boast a perfect 100% accuracy for their rocket missions. This is because only one rocket has been sent to those orbit levels, except for SSO, as depicted in Figure 3. On the other hand, lower accuracy in orbits like VELO is expected, given its status as a low Orbit Level and the need for a higher payload due to gravity, as shown in Figure 4 (on next slide).



CONCLUSIONS

3) Most rockets are designed for higher orbit missions, resulting in lower payload mass requirements.

- In Figure 4, we can observe that most launches have a payload range from over 0kg to 8000kg.
- Figure 5 provides a clearer explanation of this trend by presenting the average payload (for booster version F9 v1.1), which is 2534.66kg. This indicates a generally low payload for the rockets.

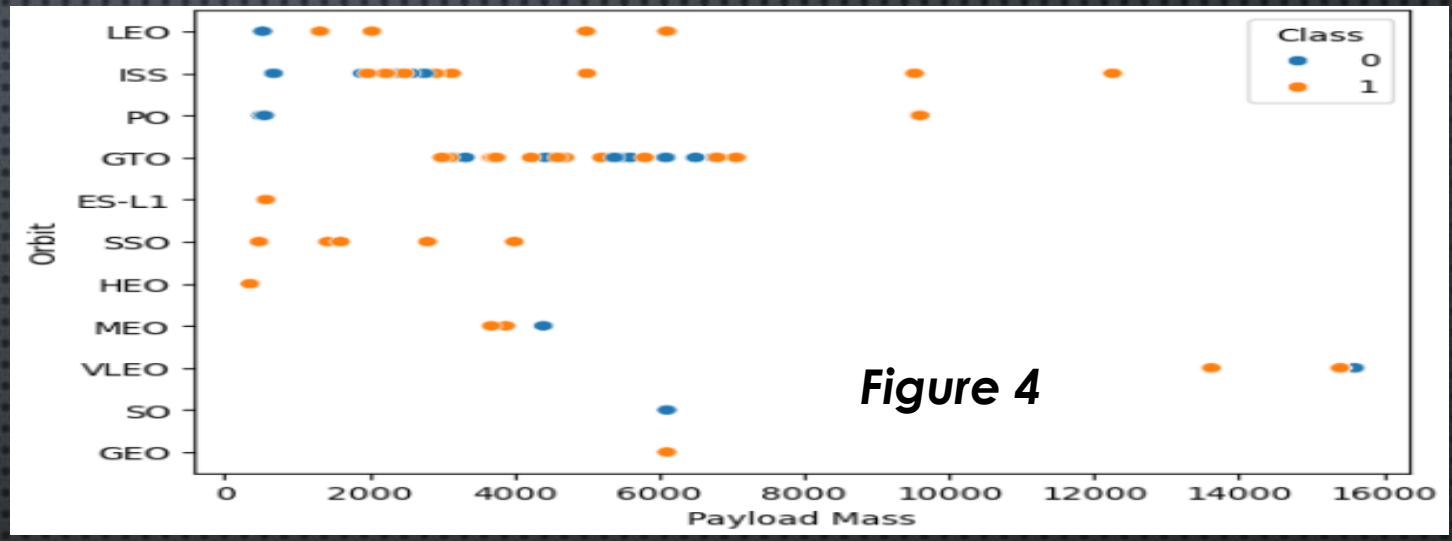


Figure 4

Figure 5

Average
2534.6666666666665

CONCLUSIONS

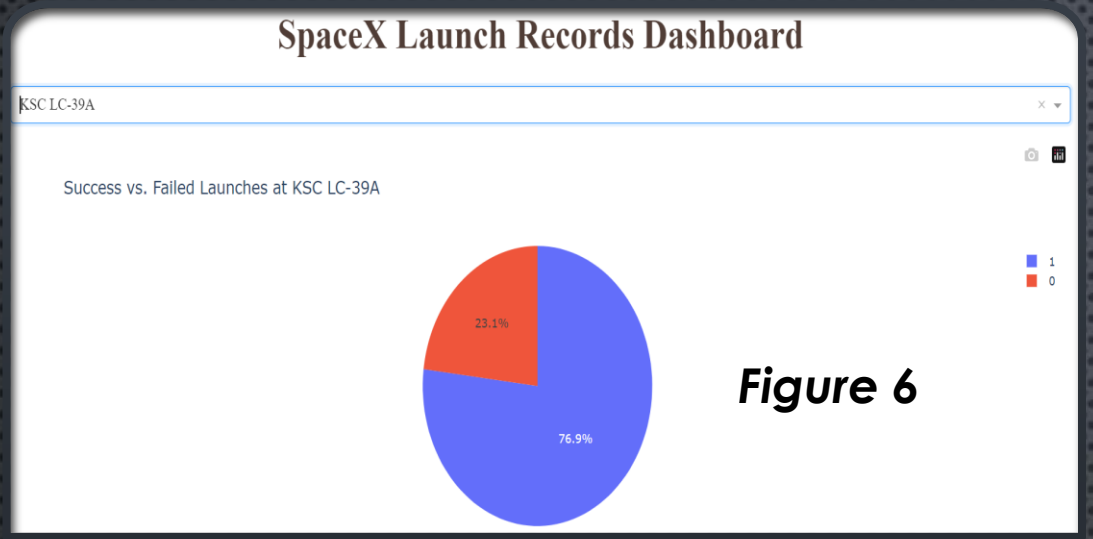


Figure 6

4) Among the four main launch sites, the one in California has the highest success rate for launches, as depicted in Figure 6.

- The ratio of successful to failed launches at the KSC LC39A CA launch site is 76.9% to 23.1%.

5) The best classification algorithm for this dataset is Decision Tree.

- Figure 7 displays the confusion matrix of the best classification algorithm for determining if a launch will be successful or not.

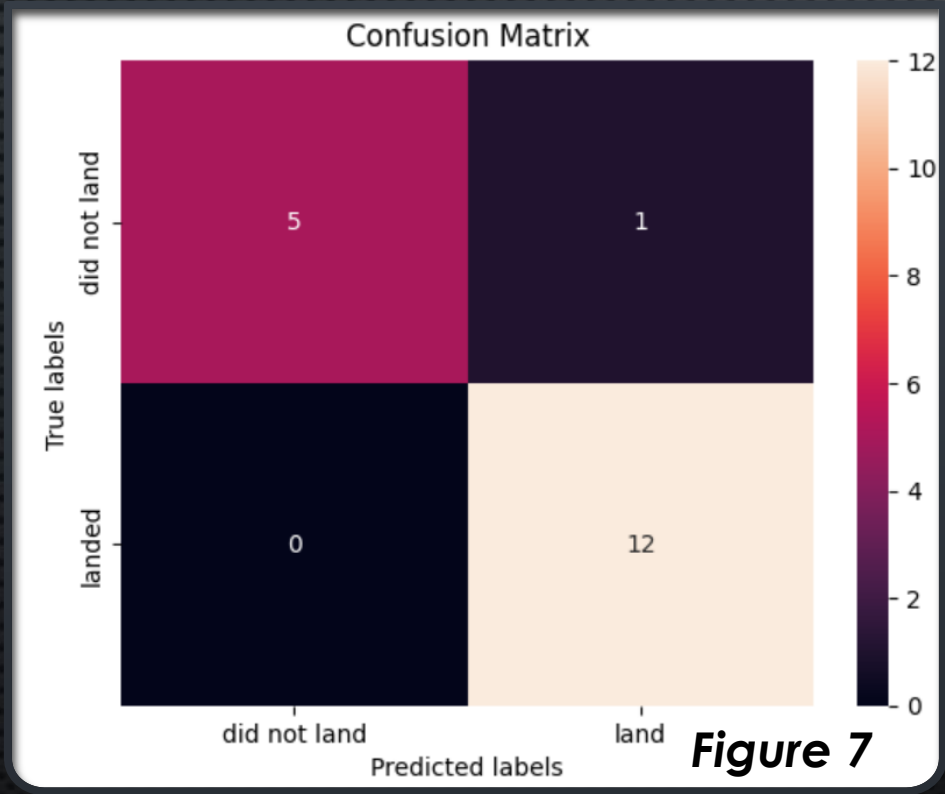


Figure 7

APPENDIX

```python
# All launch sites
launch_sites = [
    {'name': 'CCAFS LC-40', 'lat': 28.562302, 'lon': -80.577356},
    {'name': 'CCAFS SLC-40', 'lat': 28.563197, 'lon': -80.576820},
    {'name': 'KSC LC-39A', 'lat': 28.573255, 'lon': -80.646895},
    {'name': 'VAFB SLC-4E', 'lat': 34.632834, 'lon': -120.610745}
]

# Coordinates for closest locations (railways, highways, and cities) for each launch site
closest_coordinates_CA_launch_site = [
    {'railway': (34.63547, -120.6238), 'highway': (34.63773, -120.48661), 'city': (34.64112, -120.46073)}]

closest_coordinates_FL_launch_sites = [
    {'railway': (28.57246, -80.80512), 'highway': (28.57307, -80.65489), 'city': (28.60743, -80.81334)}]

# Iterate through launch sites and calculate distances
for launch_site in launch_sites:
    launch_site_name = launch_site['name']
    launch_site_lat = launch_site['lat']
    launch_site_lon = launch_site['lon']

    # Calculate distances to closest locations based on launch site name
    if launch_site_name == 'VAFB SLC-4E':
        closest_coords = closest_coordinates_CA_launch_site[0]
    else:
        closest_coords = closest_coordinates_FL_launch_sites[0]

    # Calculate distances
    distance_to_railway = calculate_distance(launch_site_lat, launch_site_lon, *closest_coords['railway'])
    distance_to_highway = calculate_distance(launch_site_lat, launch_site_lon, *closest_coords['highway'])
    distance_to_city = calculate_distance(launch_site_lat, launch_site_lon, *closest_coords['city'])

    # Create distance markers and lines
    for destination_coordinates, distance in zip(
            [closest_coords['railway'], closest_coords['highway'], closest_coords['city']],
            [distance_to_railway, distance_to_highway, distance_to_city]
        ):
        # Create a distance marker
        distance_marker = folium.Marker(
            destination_coordinates,
            icon=folium.DivIcon(
                icon_size=(20, 20),
                icon_anchor=(0, 0),
                html='<div style="font-size: 12; color:#ff5c33;"><b>%s</b></div>' % "{:10.2f} KM".format(distance),
            )
        )

        # Define the distance line
        distance_line = folium.PolyLine(
            locations=[(launch_site_lat, launch_site_lon), destination_coordinates],
            weight=1
        )

        site_map.add_child(distance_marker)
        site_map.add_child(distance_line)

site_map
```

CUSTOM LOGIC FOR FOLIUM MAP


IBM Developer
SKILLS NETWORK
THANK YOU ☺