Leading Space Race with Data Science: A Capstone Project of SpaceX

Devansh Pathak

September 15, 2023

Outline

- Executive Summary
- Introduction
- Methodology
- Results
- Discussion
- Conclusions

Executive Summary

- This undertaking aims to employ machine learning classification algorithms to forecast the successful landing of Falcon 9 first stage.
- The project encompasses several key phases:
  - Data Gathering: The initial step involves collecting the necessary data.
  - Exploratory Data Analysis (EDA): Thoroughly examining the collected data to understand its characteristics and gain insights
  - Data Visualization: Utilizing visualization techniques to depict the aspects of rocket launches associated with their outcomes, whether they are successful or not.
  - Machine Learning Prediction: Employ machine learning models for predictive analysis.
- Through the use of data visualization, this project illuminates the factors influencing the success or failure of rocket launches. The conclusive findings indicate that the decision tree algorithm stands out as the most effective machine learning method for forecasting the successful landing of the Falcon 9 first stage.

Introduction

- This project is designed to forecast the successful landing of the Falcon 9 first stage. SpaceX promotes Falcon 9 rocket launches on its website at a price of 62 million dollars, while other companies charge as much as 165 million dollars for their launches. Much of SpaceX's cost advantage is attributed to their ability to reuse the first stage. Consequently, the ability to predict the first stage's landing outcome allows for a more accurate estimation of launch costs. This information can prove invaluable should another company seek to compete with SpaceX in the rocket launch industry.

- It's noteworthy that a significant proportion of failed landings are deliberate, as SpaceX occasionally opts for controlled landings in water.

- The central question we aim to address in this project revolves around determining whether, given a specific set of Falcon 9 rocket launch parameters such as cargo tonnage, orbit type, launch site, and more, the first stage of the rocket will successfully achieve a landing.

Section 1

Methodology

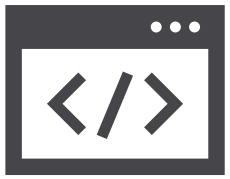
Methodology

- Data Collection and Data Wrangling
  - SpaceX API
  - Web Scrapping
- Exploratory Data Analysis (EDA)
  - Pandas and Numpy
  - SQL
- Data Visualization
  - Matplotlib and Seaborn
  - Folium
  - Dash
- Machine Learning Prediction
  - Logistic Regression
  - SVM
  - Decision Tree
  - K-Nearest Neighbors (KVM)

Data Collection

Data collection involves the systematic acquisition and measurement of information related to specific variables within an established system. This process enables us to address pertinent questions and assess the outcomes of interest. As mentioned earlier, our dataset was sourced from Wikipedia using a combination of REST API and web scraping techniques.

For the REST API approach, we initiated the process with a GET request. Subsequently, we encoded the response content in JSON format and transformed it into a pandas data frame using the json_normalize() function. Following this, we conducted data cleaning, identifying and addressing missing values to ensure data completeness.



In the case of web scraping, BeautifulSoup was employed to extract launch records presented as an HTML table. We then parsed this table and converted it into a pandas dataframe, facilitating further analysis of the data.

Data Collection – SpaceX API

- In this phase, we initiated an API request to obtain SpaceX launch data using the provided API. To refine the dataset, we applied filters to include only Falcon 9 launches, as the API encompasses information regarding a range of rocket launches conducted by SpaceX.
- To ensure data completeness, any missing values within the dataset were replaced with the mean of the respective column to which they belong.
- This image provides a glimpse of the data, highlighting a selection of rows and columns from the dataset.
- Below is a snapshot of the dataset, illustrating a subset of rows and columns:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs
4	1	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False
5	2	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False	False	False
6	3	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False	False	False
7	4	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False	False	False
8	5	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	False	False	False

Create API GET request, normalize data and read into a DataFrame

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

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

```
# Use json_normalize meethod to convert the json result into a dataframe
if response.status_code == 200:
    spacex_data = response.json()
    df = pd.json_normalize(spacex_data)
else:
    print("Failed:", response.status_code)
```

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

Declare global variable lists that will store data returned by helper functions with additional API calls to get relevant data

```
# Call getBoosterVersion
getBoosterVersion(subset_df)
```

```
# Call getLaunchSite
getLaunchSite(subset_df)
```

```
# Call getPayloadData
getPayloadData(subset_df)
```

```
# Call getCoreData
getCoreData(subset_df)
```

Call helper functions to get relevant data where columns have IDs

```
launch_dict = {'FlightNumber': list(subset_df['flight_number']),
'Date': list(subset_df['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}
```

Construct dataset from received data & combine columns into a dictionary

Create DataFrame from dictionary and filter to keep only the Falcon 9 launches

```
# Create a data from launch_dict
launch_df = pd.DataFrame(launch_dict)
```

```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = launch_df[launch_df['BoosterVersion'] == 'Falcon 9']
print(data_falcon9.head())
```

```
   FlightNumber        Date BoosterVersion  PayloadMass Orbit    LaunchSite  \
4             6  2010-06-04       Falcon 9          NaN   LEO  CCSFS SLC 40
5             8  2012-05-22       Falcon 9        525.0   LEO  CCSFS SLC 40
6            10  2013-03-01       Falcon 9        677.0   ISS  CCSFS SLC 40
7            11  2013-09-29       Falcon 9        500.0    PO   VAFB SLC 4E
8            12  2013-12-03       Falcon 9       3170.0   GTO  CCSFS SLC 40
```

Data Collection – Web Scraping

- In this phase, we conducted web scraping to gather historical launch records for Falcon 9 rockets. Our data source was a Wikipedia page titled "List of Falcon 9 and Falcon Heavy launches."
- Here's a breakdown of the process:
  - We initiated a request to access the Wikipedia page specifically dedicated to Falcon 9 launches by using its URL.
  - We utilized BeautifulSoup, a powerful web scraping library, to extract the Falcon 9 launch records directly from the Wikipedia page. This involved parsing the table containing the launch information.
  - The extracted launch records were then converted into a pandas data frame. This transformation facilitated further analysis and manipulation of the data for our project.

```
static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"
```

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

```
<Response [200]>
```

Create API GET method to request Falcon 9 launch HTML page

```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
content = page.text
BeautifulSoup = BeautifulSoup(content, 'html.parser')
```

Create BeautifulSoup Object

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

```
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 i in first_launch_table.find_all('th'):
    if extract_column_from_header(i)!=None:
        if len(extract_column_from_header(i))>0:
            column_names.append(extract_column_from_header(i))
```

Find all the tables on the Wiki page and extract relevant column names from HTML table header

```
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']=[]
```

Create an empty Dictionary with keys from extracted column names

```
def date_time(table_cells):
    return [data_time.strip() for data_time in list(table_cells.strings)][0:2]
def booster_version(table_cells):
    out=''.join([booster_version for i,booster_version in enumerate( table_cells.strings) if i%2==0][0:-1])
    return out
def landing_status(table_cells):
    out=[i for i in table_cells.strings][0]
    return out
def get_mass(table_cells):
    mass=unicodedata.normalize("NFKD", table_cells.text).strip()
    if mass:
        mass.find("kg")
        new_mass=mass[0:mass.find("kg")+2]
    else:
        new_mass=0
    return new_mass
def extract_column_from_header(row):
    if (row.br):
        row.br.extract()
    if row.a:
        row.a.extract()
    if row.sup:
        row.sup.extract()
    colunm_name = ' '.join(row.contents)
    # Filter the digit and empty names
    if not(colunm_name.strip().isdigit()):
        colunm_name = colunm_name.strip()
        return colunm_name
```

Fill up the launch_dict with launch records extracted from table rows by utilizing helper functions to help parse HTML data

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

Convert launch_dict to DataFrame

Data Wrangling

- In the data wrangling phase, we conducted Exploratory Data Analysis (EDA) to uncover patterns within the dataset and establish labels for training supervised models. The dataset encompassed various mission outcomes, which were transformed into training labels with the following definitions:
- **1**: Indicates that the booster successfully landed.
- **0**: Denotes that the booster was unsuccessful in its landing attempt.

Data Wrangling – Label Definition

- To create these labels, we considered several landing scenarios:
  - **True Ocean**: This label represents a mission outcome where the booster successfully landed in a specific region of the ocean.
  - **False Ocean**: Signifies a mission outcome where the booster was unsuccessful in landing within a designated ocean region.
  - **RTLS (Return to Launch Site)**: This label corresponds to a mission outcome where the booster successfully landed on a ground pad.
  - **False RTLS**: Indicates a mission outcome where the booster was unsuccessful in landing on a ground pad.
  - **True ASDS (Autonomous Spaceport Drone Ship)**: This label is assigned to a mission outcome where the booster successfully landed on a drone ship.
  - **False ASDS**: Represents a mission outcome where the booster was unsuccessful in landing on a drone ship.
- By defining these labels, we established a clear framework for training supervised machine learning models to predict the success or failure of Falcon 9 booster landings.

```
from js import fetch
import io

URL = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_1.csv'
resp = await fetch(URL)
dataset_part_1_csv = io.BytesIO((await resp.arrayBuffer()).to_py())
```

```
df=pd.read_csv(dataset_part_1_csv)
df.head(10)
```

Load dataset into a DataFrame

```
# 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
```

```
# 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
```

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

Finding Patterns

```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
# Define the set of bad outcomes
bad_outcomes = {'False Ocean', 'None ASDS', 'True ASDS', 'True Ocean', 'True RTLS'}

# Create a list of labels based on the Outcome column
landing_class = [0 if outcome in bad_outcomes else 1 for outcome in df['Outcome']]

# Assign the list to the 'landing_class' variable
df['landing_class'] = landing_class

# Print the first few rows of the DataFrame to verify the new column
print(df.head())
```

```
   FlightNumber        Date BoosterVersion  PayloadMass Orbit    LaunchSite  \
0             1  2010-06-04       Falcon 9  6104.959412   LEO  CCAFS SLC 40
1             2  2012-05-22       Falcon 9   525.000000   LEO  CCAFS SLC 40
2             3  2013-03-01       Falcon 9   677.000000   ISS  CCAFS SLC 40
3             4  2013-09-29       Falcon 9   500.000000    PO   VAFB SLC 4E
4             5  2013-12-03       Falcon 9  3170.000000   GTO  CCAFS SLC 40

       Outcome  Flights  GridFins  Reused   Legs LandingPad  Block  \
0    None None        1     False   False  False        NaN    1.0
1    None None        1     False   False  False        NaN    1.0
2    None None        1     False   False  False        NaN    1.0
3  False Ocean        1     False   False  False        NaN    1.0
4    None None        1     False   False  False        NaN    1.0

   ReusedCount Serial   Longitude   Latitude  landing_class
0            0  B0003  -80.577366  28.561857              1
1            0  B0005  -80.577366  28.561857              1
2            0  B0007  -80.577366  28.561857              1
3            0  B1003 -120.610829  34.632093              0
4            0  B1004  -80.577366  28.561857              1
```

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

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

Create landing outcome label

EDA with Data Visualization

- During the Exploratory Data Analysis (EDA) phase, we employed various charts to gain deeper insights into the dataset:
  - **Scatter Plots**: Scatter plots are effective for visualizing the relationship or correlation between two variables, making patterns easily observable. We generated the following scatter plots:
    - 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
  - **Bar Chart**: Bar charts are a common choice for comparing the values of a variable at a specific point in time. They provide a clear visual representation of which groups are the most common and how other groups compare. Each bar's length is proportional to the value it represents. We used a bar chart to visualize the relationship between the success rate of each orbit type.
  - **Line Chart**: Line charts are useful for tracking changes over a period of time, allowing us to observe trends over time. We created a line chart to analyze the yearly trend in the average launch success rate.

- These visualizations helped us uncover patterns, relationships, and trends within the dataset, facilitating a deeper understanding of the factors influencing Falcon 9 launch outcomes.

EDA with SQL

- In the Exploratory Data Analysis (EDA) using SQL on an IBM DB2 cloud instance, the following queries and operations were conducted to gain insights into the SpaceX dataset:
  - **Unique Launch Sites**: Displayed the names of the unique launch sites in the space mission.
  - **Launch Sites Starting with 'CCA'**: Displayed 5 records where launch sites begin with the string 'CCA'.
  - **Total Payload Mass by NASA (CRS)**: Calculated and displayed the total payload mass carried by boosters launched by NASA under the Commercial Resupply Services (CRS) program.
  - **Average Payload Mass for F9 v1.1**: Displayed the average payload mass carried by booster version F9 v1.1.
  - **First Successful Ground Pad Landing**: Listed the date when the first successful landing outcome on a ground pad was achieved.
  - **Boosters with Drone Ship Success and Payload Mass**: Listed the names of the boosters that achieved success on a drone ship and had a payload mass greater than 4000 but less than 6000.
  - **Total Successful and Failed Mission Outcomes**: Displayed the total number of successful and failure mission outcomes.
  - **Booster Versions with Maximum Payload Mass (Subquery)**: Listed the names of the booster versions that carried the maximum payload mass using a subquery.
  - **Failed Drone Ship Landings in 2015**: Listed the failed landing outcomes in a drone ship, along with their booster versions and launch site names for the year 2015.
  - **Ranking Landing Outcomes**: Ranked the count of landing outcomes (such as Failure on a drone ship or Success on a ground pad) between the dates 2010-06-04 and 2017-03-20 in descending order.
- These SQL queries and operations provided valuable insights and summaries of the SpaceX dataset, enabling a comprehensive exploration of the data.

Creating an Interactive Map with Folium

- Utilizing Folium's interactive mapping capabilities allowed for the exploration of geospatial data, enabling a more interactive visual analysis and a deeper understanding of factors such as location and proximity that influence launch success rates.

- Key steps involved in building the interactive map with Folium included:
  - **Marking Launch Sites**: All launch sites were marked on the map, providing a visual representation of their locations.
  - **Highlighting Areas**: Folium's 'folium.circle' and 'folium.marker' functionalities were employed to highlight circular areas around each launch site, with text labels indicating their significance.
  - **Clustered Markers**: A 'MarkerCluster()' was added to group launch success (green) and failure (red) markers for each launch site, aiding in visualizing success rates.
  - **Distance Calculation**: Distances between launch sites and their proximity to various features like coastlines, railroads, highways, and cities were calculated.
  - **Mouse Position**: A 'MousePosition()' feature was added to provide coordinate information when hovering the mouse over a point on the map.
  - **Distance Display**: 'folium.Marker()' was used to display distances in kilometers on the map for reference points such as coastlines, railroads, highways, and cities.
  - **Drawing Lines**: 'folium.Polyline()' was employed to draw lines connecting launch sites to proximity features (e.g., coastline, railroad, highway, city).

- By executing these steps, the interactive map built with Folium provided answers to important questions:
  - Are launch sites in close proximity to railways? (YES)
  - Are launch sites in close proximity to highways? (YES)
  - Are launch sites in close proximity to coastlines? (YES)
  - Do launch sites maintain a certain distance from cities? (YES)

- This interactive map served as a valuable tool for analyzing and visualizing geospatial data, aiding in the assessment of factors influencing launch success rates.

Development of a Plotly Dash Web Application for SpaceX Launch Data

- A Plotly Dash web application was created to facilitate interactive visual analytics of SpaceX launch data in real-time. Several components were added to the dashboard to enhance its functionality:
  - **Launch Site Drop-down**: An input component was incorporated into the dashboard, allowing users to filter visualizations by all launch sites or a specific launch site of interest.
  - **Pie Chart**: A pie chart was integrated into the dashboard. When 'All Sites' is selected, it displays the total count of successful launches; when a particular site is chosen, it displays both success and failure counts for that site.
  - **Payload Range Slider**: A slider for payload range selection was added, enabling users to easily choose different payload ranges and identify visual patterns.
  - **Scatter Chart**: A scatter chart was included to explore potential correlations between payload and mission outcomes for selected site(s). Additionally, each scatter point is color-labeled with the booster version, providing insights into mission outcomes with different boosters.

Development of a Plotly Dash Web Application for SpaceX Launch Data

- The dashboard facilitated the exploration of SpaceX launch data and addressed key questions:
  - **Largest Successful Launch Site**: KSCLC-39A had the largest number of successful launches with a count of 10.
  - **Highest Launch Success Rate**: KSC LC-39A boasted the highest launch success rate at 76.9%.
  - **Payload Range with Highest Success Rate**: The payload range between 2000 kg and 5000 kg exhibited the highest launch success rate.
  - **Payload Range with Lowest Success Rate**: Payload ranges of 0-2000 kg and 5500-7000 kg had the lowest launch success rates.
  - **F9 Booster Version with Highest Success Rate**: The booster version FT had the highest launch success rate.
- This Plotly Dash web application proved to be a valuable tool for real-time exploration and visualization of SpaceX launch data, providing insights into various aspects of launch success and payload outcomes.

```
from js import fetch
import io

URL1 = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_2.csv"
resp1 = await fetch(URL1)
text1 = io.BytesIO((await resp1.arrayBuffer()).to_py())
data = pd.read_csv(text1)
```

```
URL2 = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_3.csv'
resp2 = await fetch(URL2)
text2 = io.BytesIO((await resp2.arrayBuffer()).to_py())
X = pd.read_csv(text2)
```

```
Y = data["Class"].to_numpy()
```

Predictive Analysis - Read dataset into DataFrame

```
# students get this
transform = preprocessing.StandardScaler()
x_scaled = transform.fit_transform(X)
col=X.columns
X=pd.DataFrame(x_scaled, columns=col)
X.head()
```

Predictive Analysis – Standardize the data

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
```

Predictive Analysis – Train/Test/Split data

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

...

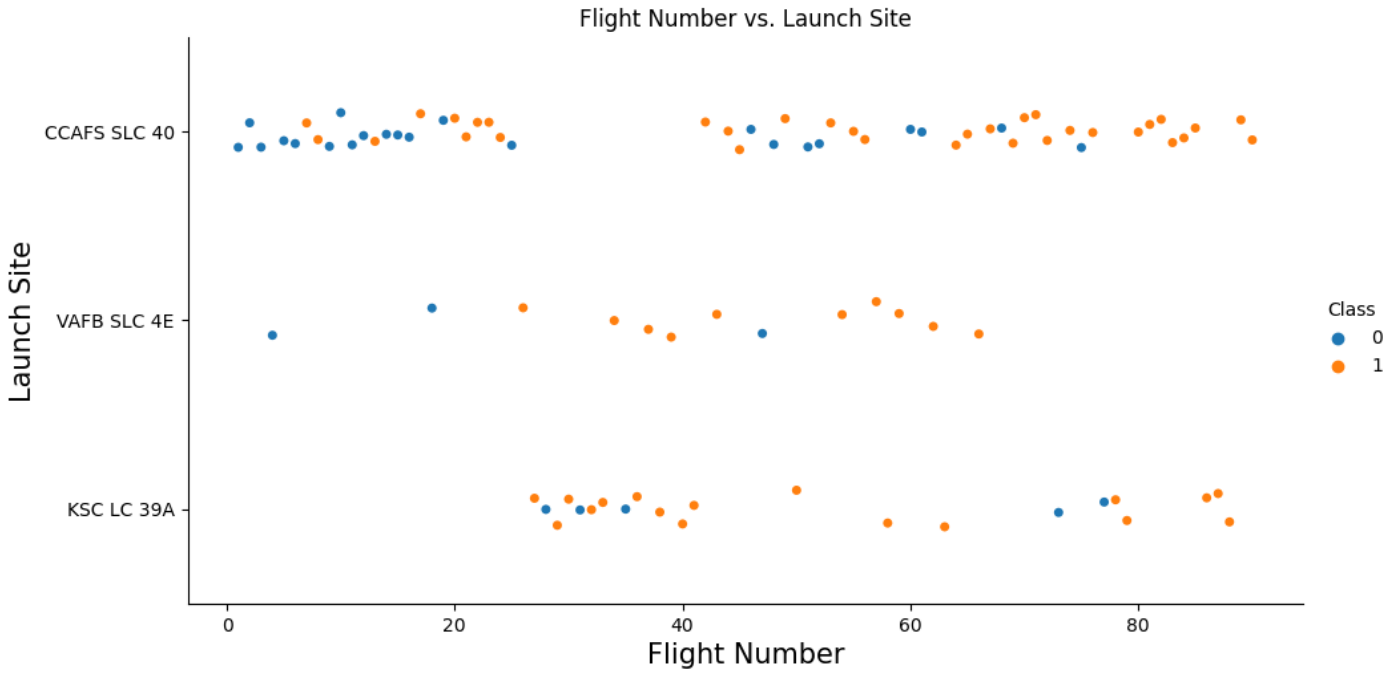
```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

Predictive Analysis – Create & Refine Models

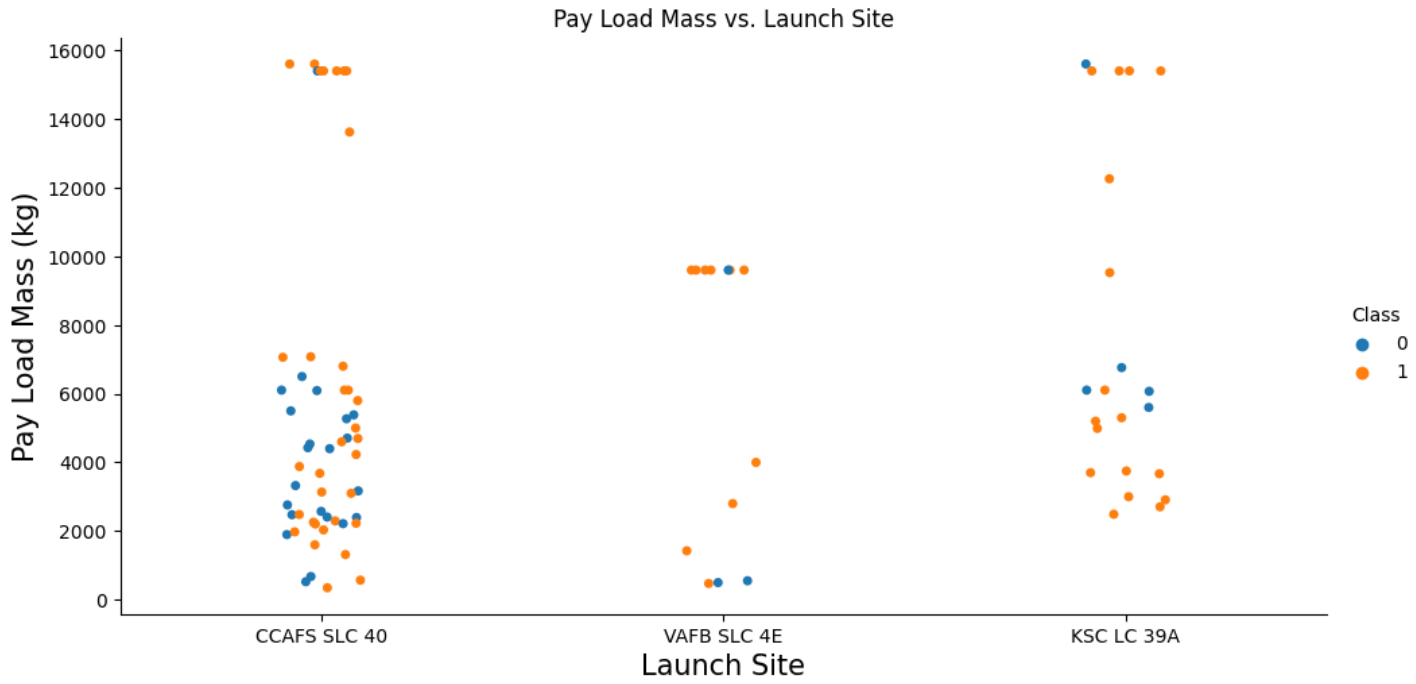
Section 2

Insights drawn from EDA

Flight Number vs. Launch Site

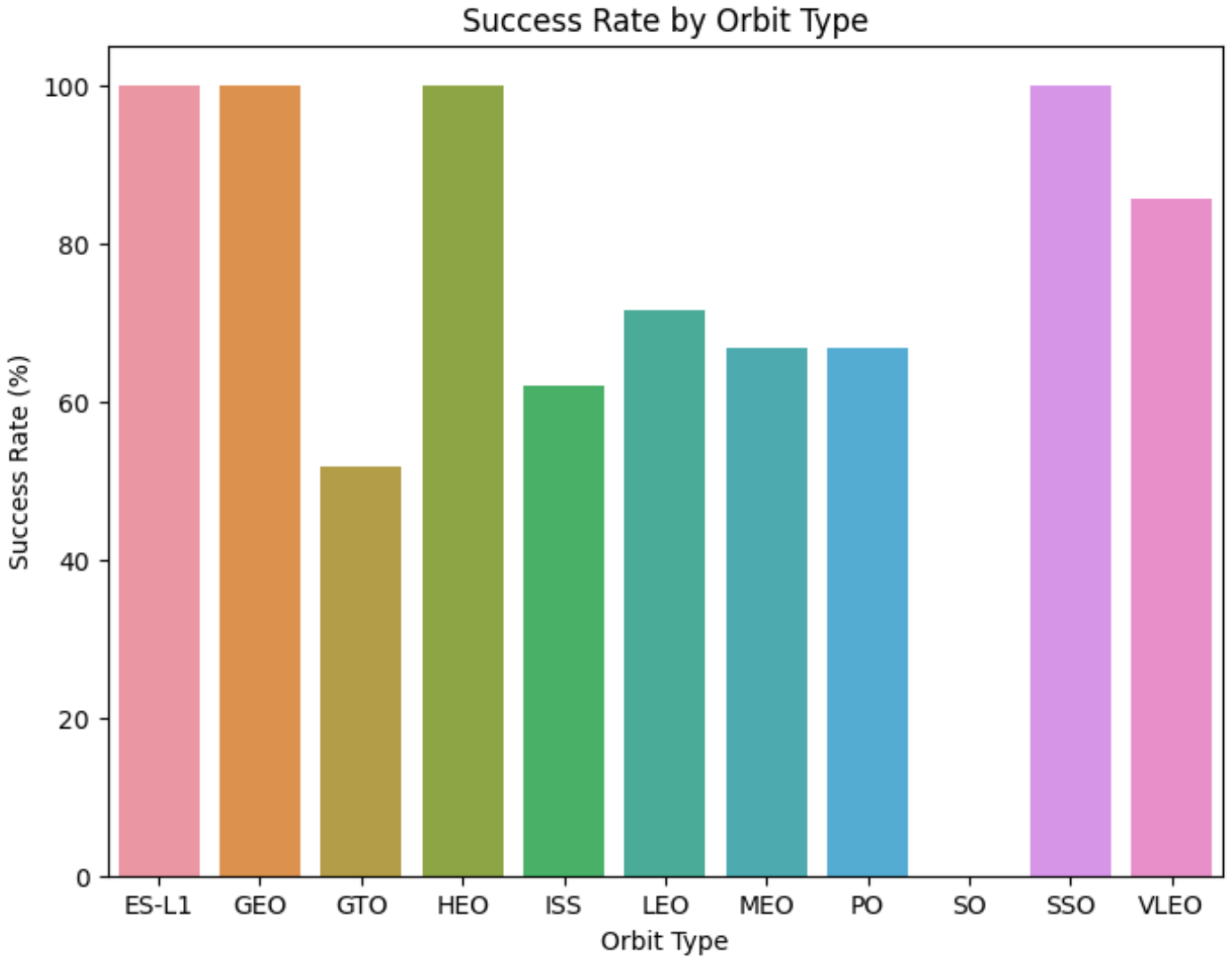


- Success rates (Class = 1) increases as the number of flights increase
- For launch site KSC LC 39A, it takes at least around 25 launches before a first successful launch



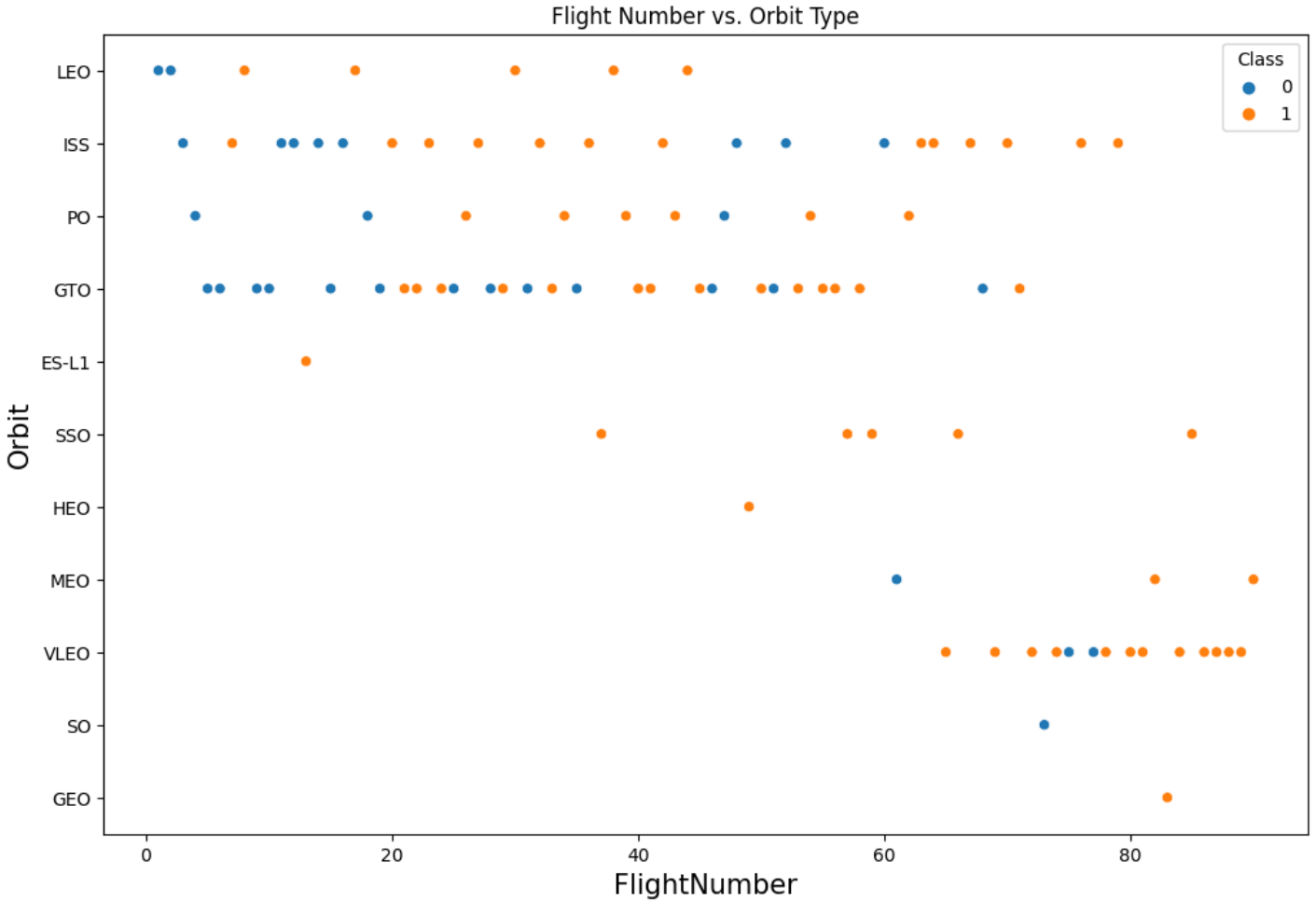
Payload vs. Launch Site

- For lauch site VAFB SLC 4E, there are no rockets launched for payload greater than 10,000 kg
- Percentage of successful launch (Class = 1) increases for launch site VAFB SLC 4E as the payload mass increases
- There is no clear correlation or pattern between launch site and payload



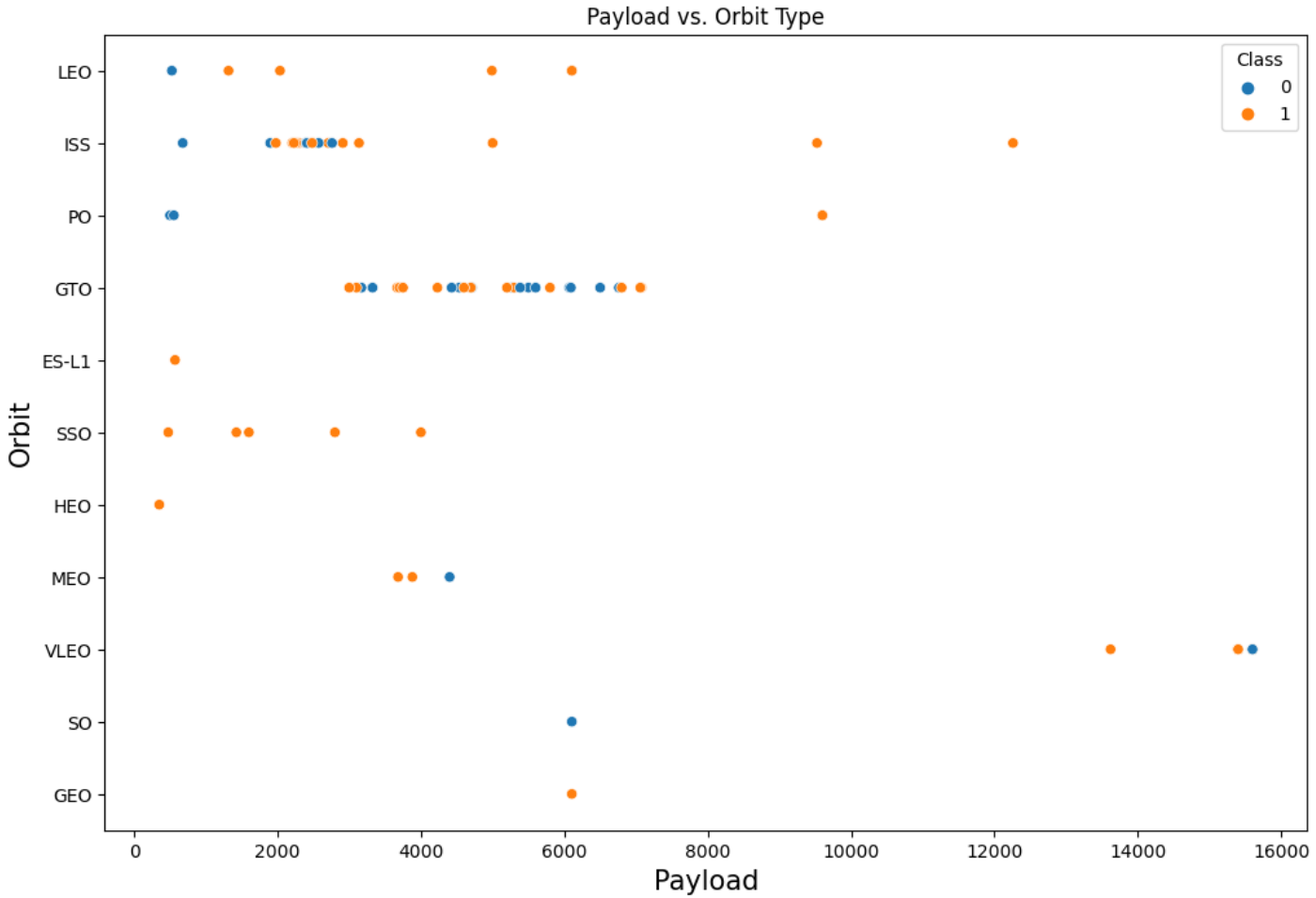
Success Rate vs. Orbit Type

- Orbits ES-L1, GEO, HEO, and SSO have the highest success rates
- GTO orbit has the lowest success rates



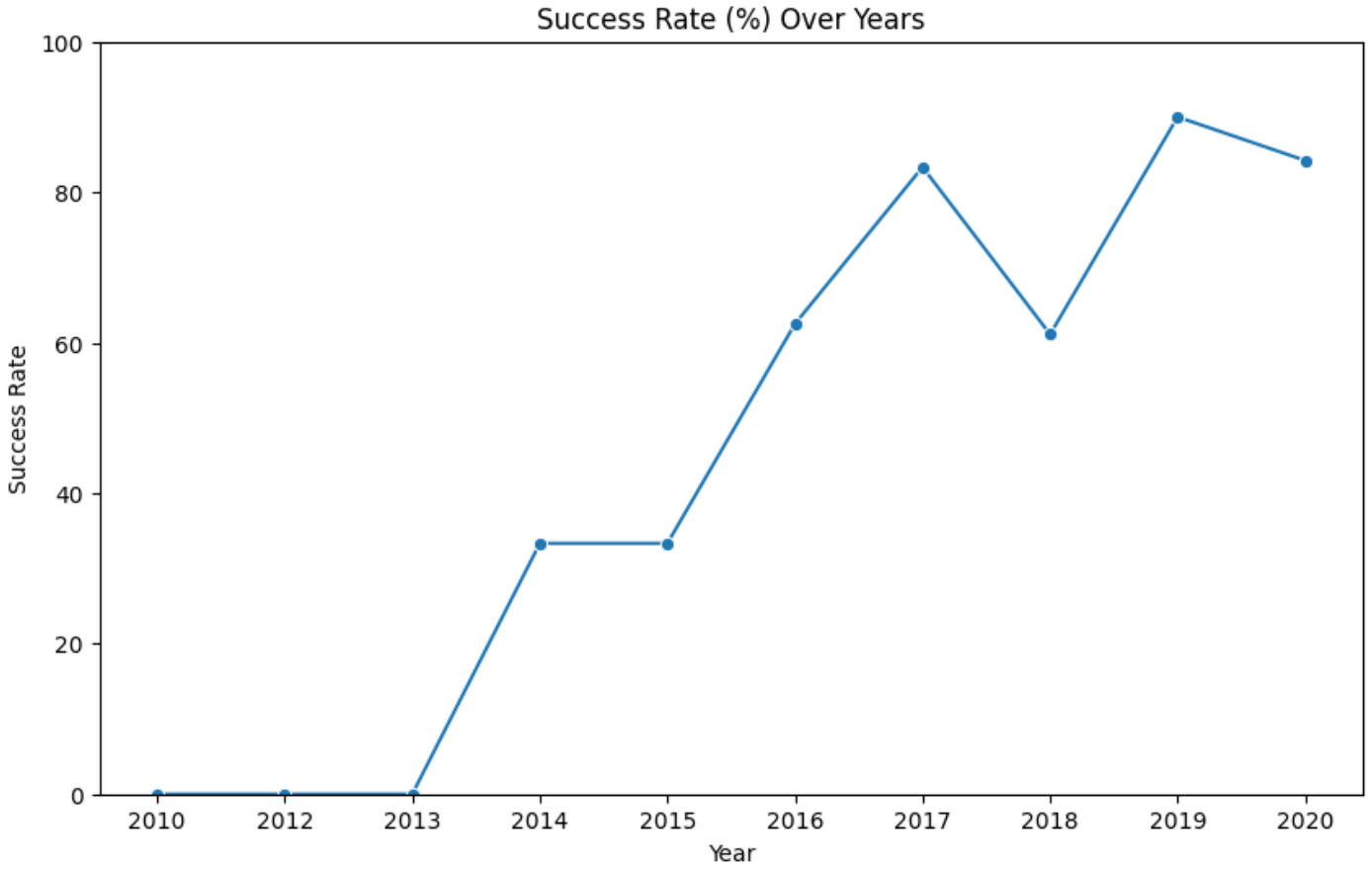
Flight Number vs. Orbit Type

- For orbit VLEO, first successful landing (Class = 1) does not occur until 60+ number of flights
- For most orbits (LEO, ISS, PO, SSO, MEO, VLEO) successful landing rates appear to increase with flight numbers
- There is no relationship between flight number and orbit for GTO



Payload vs. Orbit Type

- Successful landing rates (Class = 1) appear to increase with payload for orbits LEO, ISS, PO, and SSO
- For GEO orbit, there is no clear pattern between payload and orbit for successful or unsuccessful landing



Success Rate Over Years

- Success rate (Class = 1) increased by about 80% between 2013 to 2020
- Success rates remained the same between 2010 and 2013; and between 2014 and 2015
- Success rates decreased between 2017 and 2018; and between 2019 and 2020

```
%sql SELECT DISTINCT Launch_Site FROM SPACEXTABLE;
```

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

```
Done.
```

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

All Launch Site Names

- 'SELECT DISTINCT' returns only unique values from the queries column (launch_site)
- There are 4 unique launch sites

Launch Site Names Begin with CCA

```
%sql SELECT * FROM SPACEXTABLE WHERE Launch_Site LIKE 'CCA%' LIMIT 5;
```

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

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

- Using keywords 'LIKE' and format 'CCA%' returns records where 'launch_site' column starts with 'CCA'
- 'LIMIT 5' limits the number of returned records to 5

Total Payload Mass

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

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

Total_payload_mass
45596

- 'SUM' adds column 'PAYLOAD_MASS_KG' and returns total payload mass for customers names 'NASA (CRS)'

Average Payload Mass by F9 v1.1

```
%%sql
SELECT AVG(PAYLOAD_MASS__KG_) AS Average_Payload_Mass
FROM SPACEXTABLE WHERE Booster_Version = 'F9 v1.1';
```

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

Done.
```

Average_Payload_Mass
2928.4

- 'AVG' keyword returns the average of payload mass in 'PAYLOAD_MASS_KG' column where booster version is 'F9 v1.1'

First Successful Ground Landing Date

```
%%sql
SELECT MIN(DATE) AS First_successful_landing_gound
FROM SPACEXTABLE WHERE Landing_Outcome = 'Success (ground pad)';
```

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

First_successful_landing_gound
2015-12-22

- 'MIN(DATE)' selects the first or the oldest date from the 'Date' column where first successful landing on group pad was achieved
- 'WHERE' defines the criteria to return date for scenarios where 'Landing_Outcome' value is equal to 'Success (ground pad)'

```
%%sql
SELECT DISTINCT "Booster_Version"
FROM SPACEXTABLE
WHERE "Landing_Outcome" = 'Success (drone ship)'
AND "PAYLOAD_MASS__KG_" > 4000
AND "PAYLOAD_MASS__KG_" < 6000;
```

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

Done.
```

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

Successful Done Ship Landing with Payload between 4000 and 6000

- The query finds the booster version where payload mass is greater than 4000 but less than 6000; 'Landing_Outcome' is success in drone ship
- The 'AND' operator in the 'WHERE' clause returns booster versions where both conditions is true

```
%%sql
SELECT "Mission_Outcome", COUNT(*) AS "Count"
FROM SPACEXTABLE
GROUP BY "Mission_Outcome";
```

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

Done.
```

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

Total Number of Successful and Failure Mission Outcomes

- The 'GROUP BY' arranges identical data in a column into groups
- In this case, number of mission outcomes by types of outcomes are grouped in column 'counts'

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

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

Done.
```

Booster_Version
F9 B5 B1048.4
F9 B5 B1049.4
F9 B5 B1051.3
F9 B5 B1056.4
F9 B5 B1048.5
F9 B5 B1051.4
F9 B5 B1049.5
F9 B5 B1060.2
F9 B5 B1058.3
F9 B5 B1051.6
F9 B5 B1060.3
F9 B5 B1049.7

Boosters Carried Maximum Payload

- The sub-query returns the maximum payload mass by using keywords 'MAX' on the payload mass column
- The main query returns booster versions and respective payload mass where payload mass is maximum value of 15600

```sql
%%sql
SELECT
    CASE
        WHEN strftime('%m', "Date") = '01' THEN 'January'
        WHEN strftime('%m', "Date") = '02' THEN 'February'
        WHEN strftime('%m', "Date") = '03' THEN 'March'
        WHEN strftime('%m', "Date") = '04' THEN 'April'
        WHEN strftime('%m', "Date") = '05' THEN 'May'
        WHEN strftime('%m', "Date") = '06' THEN 'June'
        WHEN strftime('%m', "Date") = '07' THEN 'July'
        WHEN strftime('%m', "Date") = '08' THEN 'August'
        WHEN strftime('%m', "Date") = '09' THEN 'September'
        WHEN strftime('%m', "Date") = '10' THEN 'October'
        WHEN strftime('%m', "Date") = '11' THEN 'November'
        WHEN strftime('%m', "Date") = '12' THEN 'December'
    END AS "Month",
    "Landing_Outcome",
    "Booster_Version",
    "Launch_Site"
FROM SPACEXTABLE
WHERE strftime('%Y', "Date") = '2015'
AND "Landing_Outcome" LIKE 'Failure (drone ship)%';
```

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

Done.
```

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

2015 Launch Records

- The query lists landing outcomes, booster version, and the launch site where landing outcome is failed in drone ship in the year 2015
- The 'AND' operator in the 'WHERE' clause returns booster versions where both conditions are met
- The '%Y' keyword extracts the year from the column 'Date'

```
%%sql
SELECT "Landing_Outcome", COUNT(*) AS "Count"
FROM SPACEXTABLE
WHERE "Date" BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY "Landing_Outcome"
ORDER BY "Count" DESC;
```

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

Done.
```

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

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

- The 'GROUP BY' keyword arranges data in column landing outcome into groups
- The 'BETWEEN' and 'AND' keywords returns the date between 2010-06-04 and 2017-03-20
- The 'ORDER BY' keyword arranges the counts column in descending order
- The result of the query is ranked list of landing outcome counts per the specified date range

Section 3

Launch Site Proximities Analysis

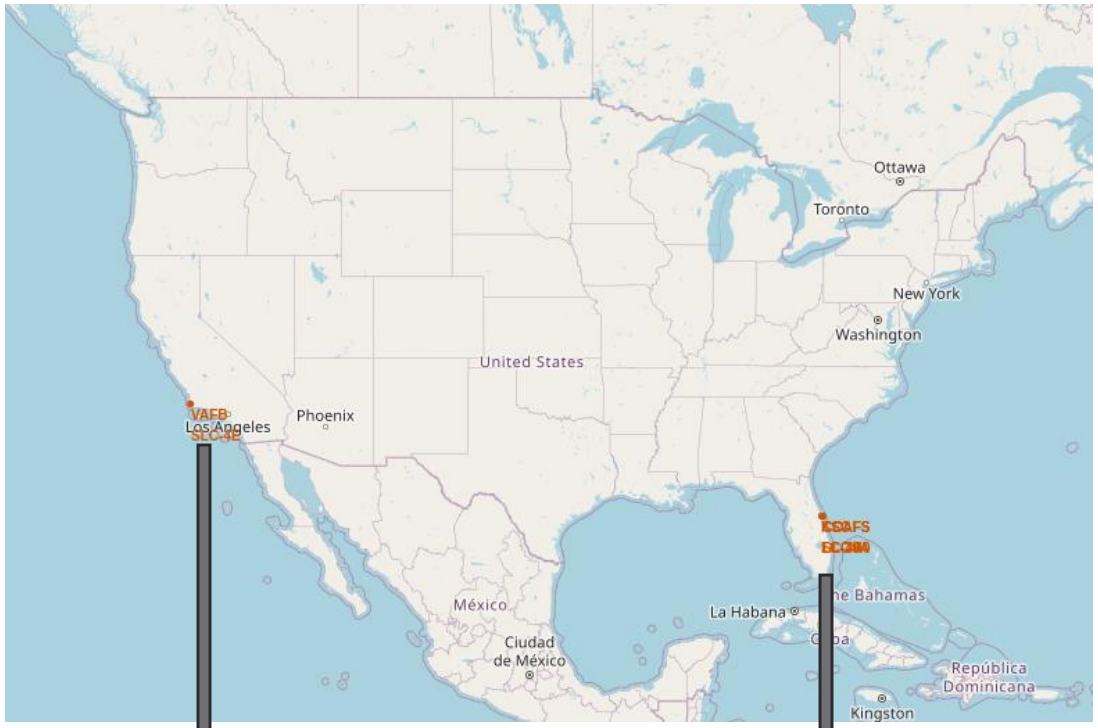


Figure 1.1 - Global Map

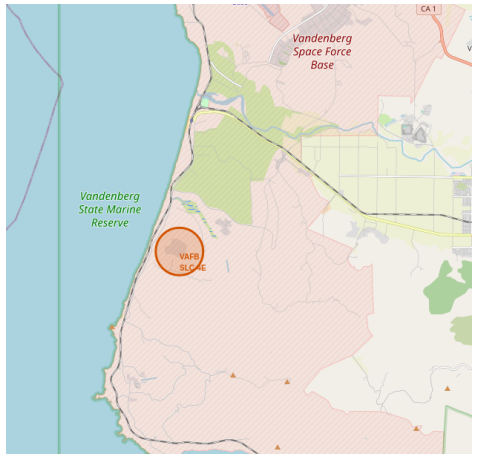


Figure 1.2 - Zoom 1

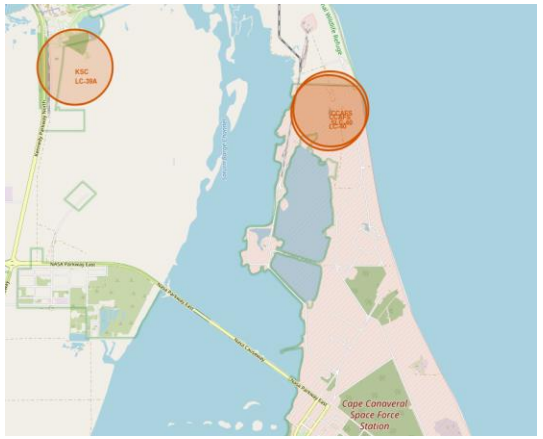
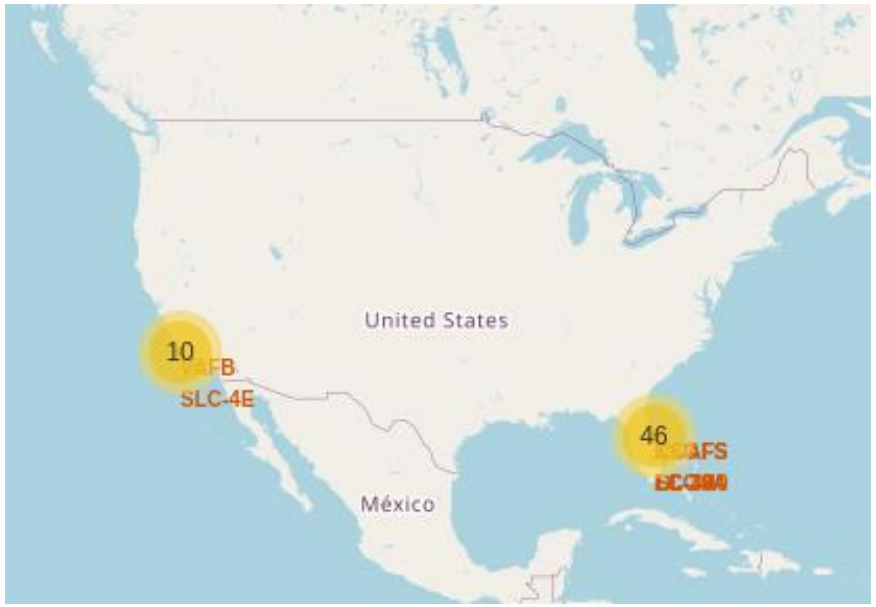


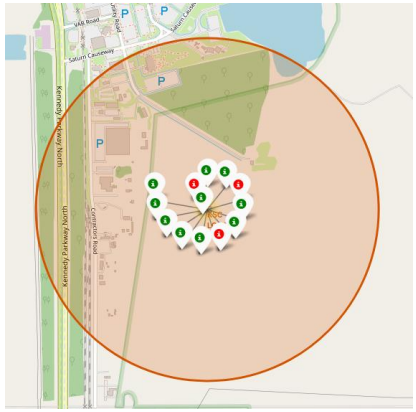
Figure 1.3 - Zoom 2

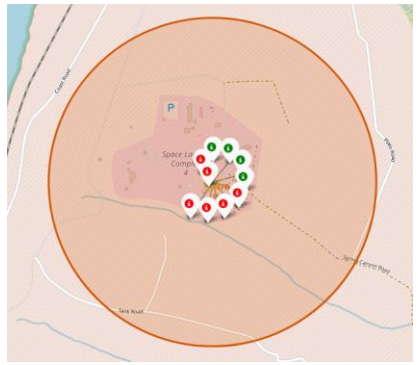
SpaceX Falcon 9 – Launch Sites Map

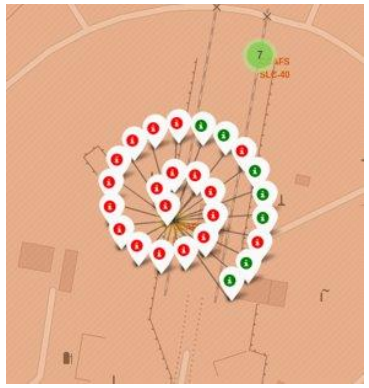
- Figure 1.1 - Global Map with Falcon 9 launch sites located in United States (California & Florida)
- Figure 1.2 & 1.3 - Zoomed launch sites to display 4 launch sites:
  - VAFB SLC-4E (CA)
  - CCAFS LC-40 (FL)
  - KSC LC-39A (FL)
  - CCAFS SLC-40 (FL)

SpaceX Falcon 9 Success/Failed Map





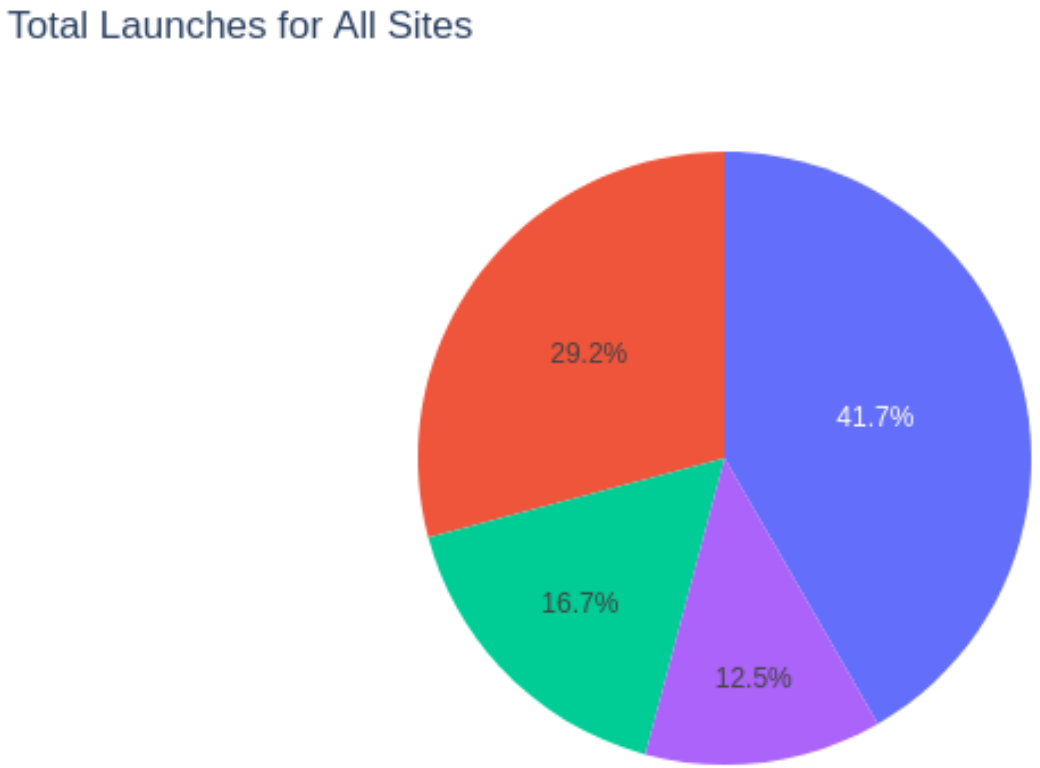




Section 4

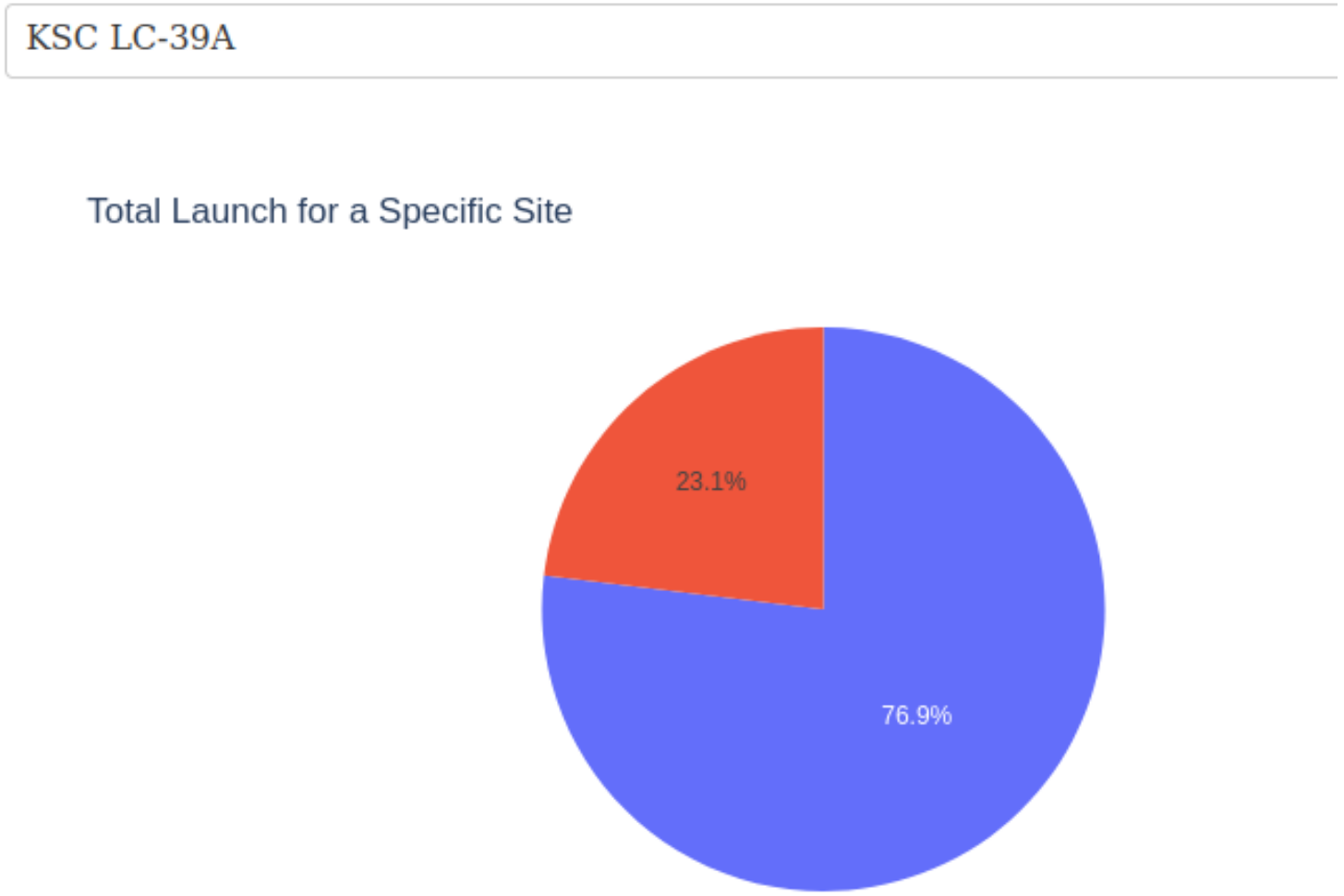
Build a Dashboard with Plotly Dash

ALL SITES



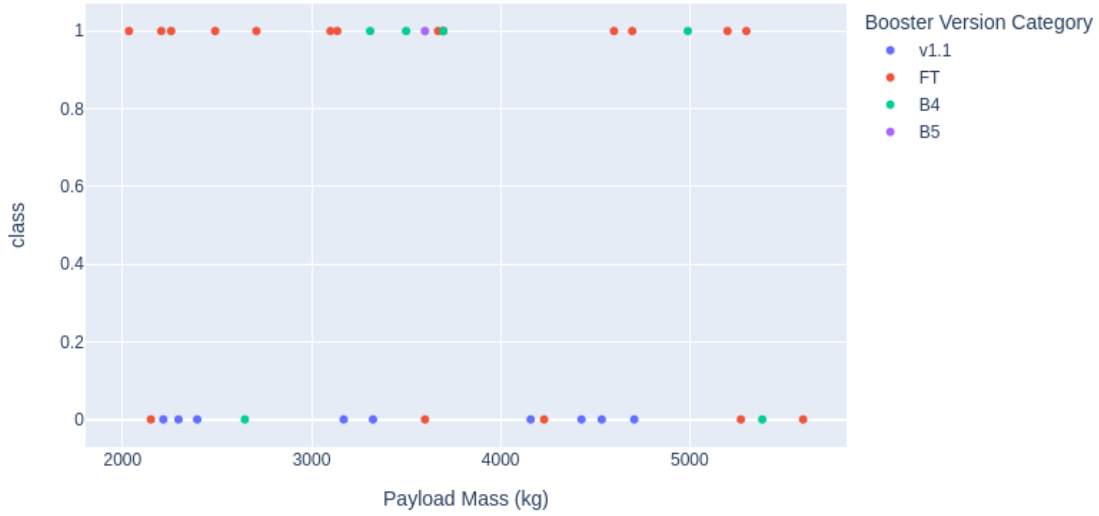
Launch Success Counts For All Sites

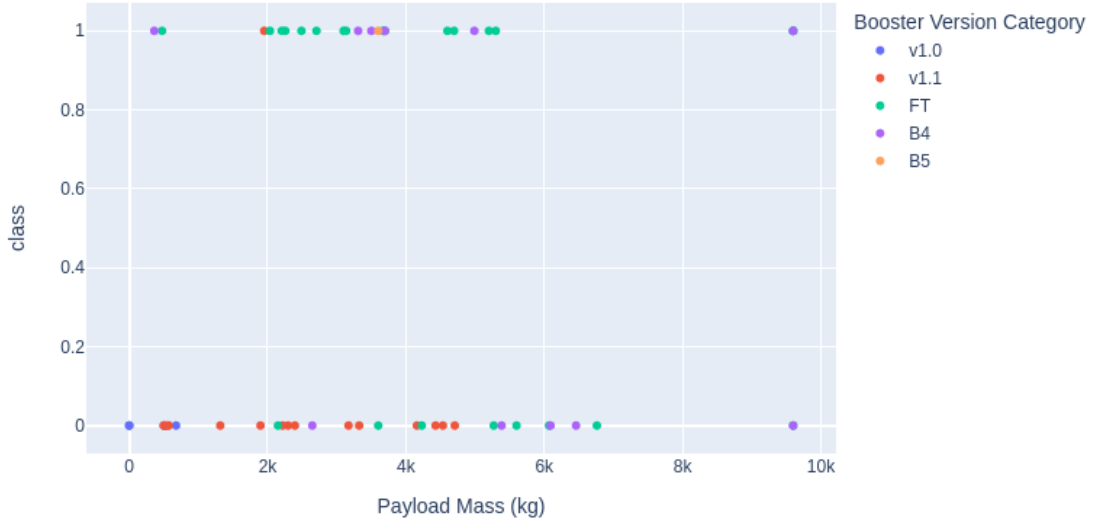
- Launch Site KSC LC-39A has the highest launch success rate
- Launch Site CCAFL SLC-40 has the lowest launch success rate



Launch Site with Highest Success Ratio

- KSC LC-39A launch site has the highest launch success rate
- Launch success rate is 76.9%
- Launch success failure is 23.1%



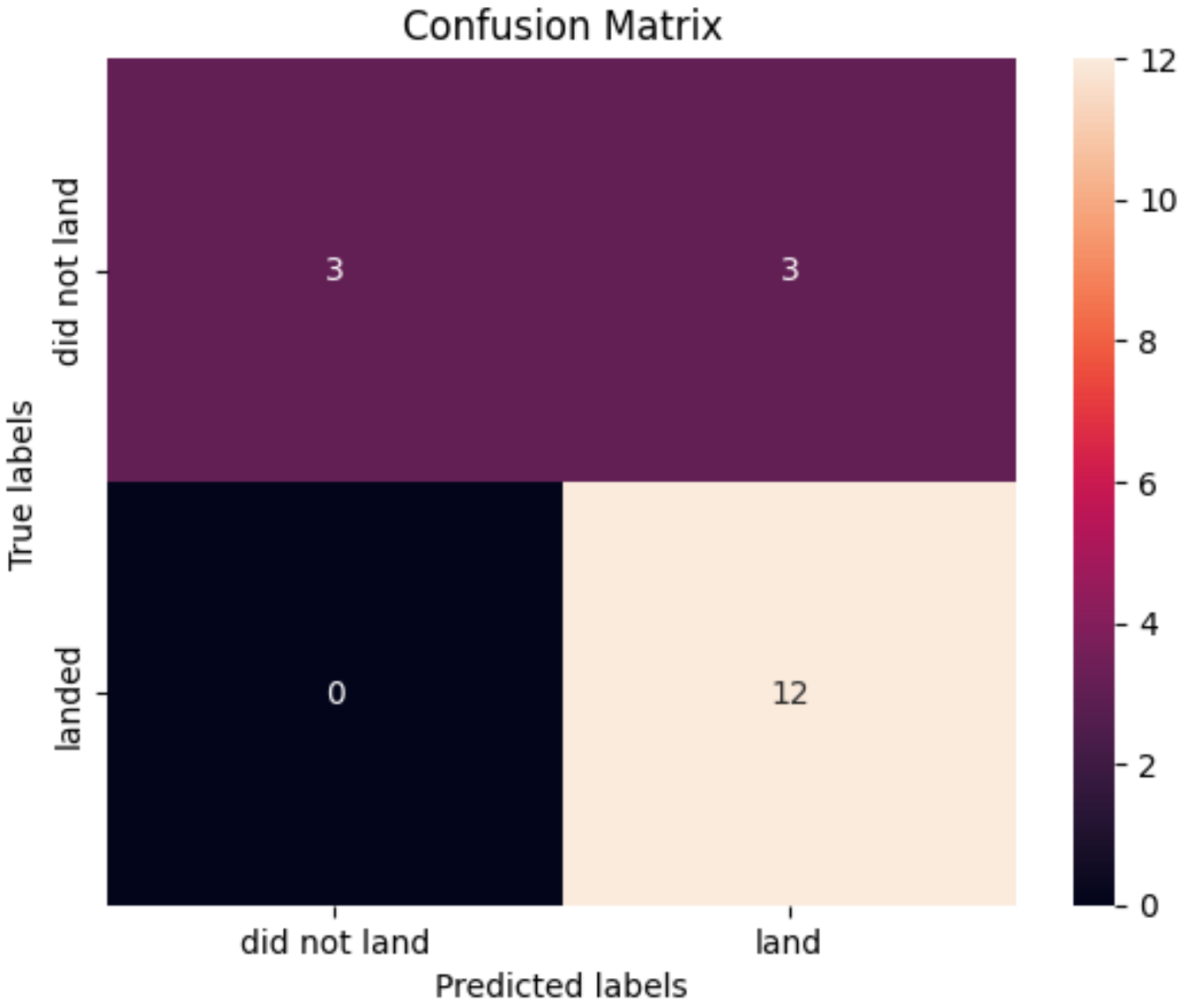


Payload vs. Launch Outcomes

- Most successful launches are in the payload range from 2000 to 5500
- Booster version category 'FT' has the most successful launches
- Only booster with a success launch when payload is greater than 6000 is B4

Section 5

Predictive Analysis



Confusion Matrix

- The confusion matrix is the same for all models (LR, SVM, Decision Tree, KNN).
- According to the confusion matrix, there were 18 predictions made.
- 12 scenarios were predicted as "Yes" for landing, and they successfully landed (True positive).
- 3 scenarios (top left) were predicted as "No" for landing, and they did not land (True negative).
- 3 scenarios (top right) were predicted as "Yes" for landing, but they did not land successfully (False positive).
- Overall, the classifier is correct about 83% of the time ((TP + TN) / Total) with a misclassification or error rate ((FP + FN) / Total) of about 16.5%.

Discussion

- With an increase in the number of flights, the likelihood of a successful landing in the first stage also rises.
- While success rates tend to rise alongside increased payload, there is no evident correlation between payload mass and success rates.
- From 2013 to 2020, there was an approximately 80% increase in the launch success rate.
- Among launch sites, 'KSC LC-39A' boasts the highest launch success rate, while 'CCAFS SLC-40' exhibits the lowest.
- The orbits ES-L1, GEO, HEO, and SSO demonstrate the highest launch success rates, whereas GTO exhibits the lowest.
- Launch sites have been strategically positioned away from urban areas, closer to coastlines, railways, and highways.
- The Decision Tree model outperforms all others in machine learning classification, achieving an accuracy of approximately 87.5%. When evaluated on test data, all models achieve an accuracy score of around 83%. Further data may be necessary for fine-tuning the models and potentially finding a better fit.

Conclusion

- Project objective: Create a machine learning model for SpaceX landing.
- Data sources: Utilized data from a public SpaceX API and conducted web scraping of the SpaceX Wikipedia page.
- Data processing: Generated data labels and stored the collected data in a DB2SQL database.
- Visualization: Developed a dashboard for data visualization.
- Utilization: SpaceX can employ this model to make informed launch decisions by predicting Stage 1 landing success prior to liftoff.
- Data enhancement: To improve the model's accuracy and select the optimal machine learning approach, it is advisable to gather additional data.

Thank you!