

Winning Space Race with Data Science

Jaylen Bailey
January 20, 2025



Outline

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

Executive Summary

- Summary of methodologies
 - Data collection and data wrangling
 - EDA with data visualization/SQL
 - Building an interactive map (Folium)
 - Building a Dashboard (Plotly Dash)
 - Predictive analysis (Classification)
- Summary of all results
 - EDA results
 - Interactive analytics
 - Predictive analysis

Introduction

- Project background and context
 - SpaceX has revolutionized the aerospace industry by prioritizing the development of reusable rockets, such as the Falcon 9, to reduce space exploration costs significantly. Successful landings are vital for reusability, enabling the company to recover and reuse rocket components, optimizing costs, and improving efficiency. This project analyzes historical launch data to understand better the factors contributing to landing success.
- Problems you want to find answers
 - The project addresses key questions: What factors most influence the success of Falcon 9 landings? How can these factors be used to predict future landing outcomes? Specifically, we seek to explore how variables like payload mass, launch site, and orbit type correlate with landing success and leverage machine learning models to forecast outcomes for future launches.

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
 - SpaceX Rest API
 - Web Scrapping from Wikipedia
- Perform data wrangling
 - One Hot Encoding for machine learning and removing any non-pertinent information from dataset.
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
 - Built classification models (Logistic Regression, Decision Tree, SVM, KNN).

Data Collection

- **Primary Data Sources:**
 - **SpaceX REST API:** Retrieved detailed launch records (payload mass, orbit type, launch site, landing outcomes).
 - **Web Scraping:** Supplemented missing details from Wikipedia.
- **Data Wrangling and Preprocessing:**
 - Filtered dataset to include only Falcon 9 launches.
 - Addressed missing values
 - Encoded categorical variables like LaunchSite and Orbit for analysis.

Data Collection – SpaceX API

```

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

To make the requested JSON results more consistent, we will use the following static response object for this project:
static_json_url = "https://api.spacexdata.com/v3/launches"

We should see that the request was successful with the 200 status response code
response = requests.get(static_json_url)
response.status_code
200

Now we decide the response content as a json using `json()` and turn it into a Pandas dataframe using `json_normalize()`.
# Let's make sure the number of columns in the json result into a dataframe
data = pd.json_normalize(response.json())
Using the dataframe, print the first 5 rows
data.head(5)

rocket    payloads   launchpad
cores flight_number date_utc
0  Set1004eadd9950709ebe  Set1004eadd9950709ebe  Set1004eadd9950709ebe  24722 2006-03-24 2006-03-24
1  Set1004eadd9950709ebe  Set1004eadd9950709ebe  Set1004eadd9950709ebe  2007 2007-03-21 2007-03-21
2  Set1004eadd9950709ebe  Set1004eadd9950709ebe  Set1004eadd9950709ebe  28723 2008-09-28 2008-09-28
3  Set1004eadd9950709ebe  Set1004eadd9950709ebe  Set1004eadd9950709ebe  5 2010-06-04 2010-06-04
4  Set1004eadd9950709ebe  Set1004eadd9950709ebe  Set1004eadd9950709ebe  6 2010-06-04 2010-06-04
5  Set1004eadd9950709ebe  Set1004eadd9950709ebe  Set1004eadd9950709ebe  6 2010-06-04 2010-06-04

You will notice that a lot of the data are IDs. For example the rocket column has no information about the rocket just an identification number.
We will now use the API again to get information about the launches using the IDs given for each launch. Specifically we will be using column rocket, payloads, launchpad, and cores.

# Let's take a subset of our dataframe keeping only the features we want and the flight number, and data_utc.
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]
# We will also drop the rows with multiple cores because those are rocket launches with 2 extra rocket boosters and rows that have multiple payloads in a single rocket.
data = data[~data['cores'].duplicated()]
# Since pandas stores the date in UTC, we will also extract the simple value in the list and replace the feature.
data['date'] = data['date_utc'].apply(lambda x: x[:-1])
# We will use the API again to convert the date UTC to a date in America/Edmonton and then extracting the date leaving the time
data['date'] = pd.to_datetime(data['date'], utc=True).dt.tz_localize('America/Edmonton').dt.date
# Using the date we will restrict the dates of the launches
data = data[data['date'] == datetime.date(2020, 11, 13)]
# From the rocket ID we would like to learn the booster name
# From the payload, we would like to know the mass of the payload and the orbit that it is going to
# From the launchpad, we would like to know the name of the launch site being used, the longitude, and the latitude.
# From cores we would like to learn the outcome of the landing, the type of the landing, number of flights with that core, whether gridfins were used, whether the core is reused, whether legs were used, the landing pad used, the block of the core which is a number used to separate version of cores, the number of times this specific core has been reused, and the serial of the core.
The data from these requests will be stored in lists and will be used to create a new dataframe.

```

1. Request and parse the SpaceX launch data using GET request

<https://github.com/jay1enbailey/Applied-Data-Science-Capstone/blob/main/jupyter-labs-spacex-data-collection-api.ipynb>

Task 2: Filter the dataframe to only include Falcon 9 launches

Finally we will remove the Falcon 1 launches keeping only the Falcon 9 launches. Filter the data dataframe using the BoosterVersion column to only keep the Falcon 9 launches. Save the filtered data to a new dataframe called data_falcon9.

```
[51]: # Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = data_launch[data_launch['BoosterVersion'] != 'Falcon 1']
```

Now that we have removed some values we should reset the FlightNumber column

```
[52]: data_falcon9.loc[:, 'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9
```

FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude
0	2006-03-24	None	20.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	Nan	0	Merlin1A	167.743129	9.047721
1	2007-03-21	None	Nan	LEO	Kwajalein Atoll	None None	1	False	False	False	None	Nan	0	Merlin2A	167.743129	9.047721
2	2008-09-28	None	165.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	Nan	0	Merlin2C	167.743129	9.047721
3	2009-07-13	None	200.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	Nan	0	Merlin3C	167.743129	9.047721
4	2010-06-04	None	Nan	MEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0003	-80.577366	28.561857
...
89	2020-09-03	None	15800.0	VLEO	KSC LC 39A	True ASDS	2	True	True	True	Se9e3032383ecbb0234e7c	5.0	12	B1060	-80.603956	28.608058
90	2020-10-06	None	15800.0	VLEO	KSC LC 39A	True ASDS	3	True	True	True	Se9e3032383ecbb0234e7c	5.0	13	B1058	-80.603956	28.608058
91	2020-10-18	None	15800.0	VLEO	KSC LC 39A	True ASDS	6	True	True	True	Se9e3032383ecbb0234e7c	5.0	12	B1051	-80.603956	28.608058
92	2020-10-24	None	15800.0	VLEO	CCSFS SLC 40	True ASDS	3	True	True	True	Se9e3032383ecbb0234e7c	5.0	12	B1060	-80.577366	28.561857
93	2020-11-05	None	3681.0	MEO	CCSFS SLC 40	True ASDS	1	True	False	True	Se9e3032383ecbb0234e7c	5.0	8	B1062	-80.577366	28.561857

94 rows x 17 columns

Data Wrangling

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

```
[53]: data_falcon9.isnull().sum()
```

FlightNumber	0
Date	0
BoosterVersion	94
PayloadMass	6
Orbit	0
LaunchSite	0
Outcome	0
Flights	0
GridFins	0
Reused	0
Legs	0
LandingPad	30
Block	4
ReusedCount	0
Serial	0
Longitude	0
Latitude	0
dtype: int64	

Then, we need to create a Pandas data frame from the dictionary launch_dict.

```
[54]: # Create a dict from launch_dict
# If there is a key in launch_dict that is not in the data, then add it to the data with a maximum value for it
# If there is a key in the data that is not in launch_dict, then add it to launch_dict with a max length
# If there is a key in launch_dict that is not in the data, then add it to the data with a max length - len(launch_dict[key])
# If there is a key in the data that is not in launch_dict, then add it to launch_dict with a max length - len(data[key])
# Using the date we will restrict the dates of the launches
data = data[data['date'] == datetime.date(2020, 11, 13)]
Show the summary of the dataframe
```

```
[55]: # Show the head of the dataframe
data.head(5)
```

```
[56]:
```

Task 3: Dealing with Missing Values

Calculate below the mean for the PayloadMass using the .mean(). Then use the mean and the .replace() function to replace np.nan values in the data with the mean you calculated.

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

You should see the number of missing values of the PayloadMass change to zero.

Now we should have no missing values in our dataset except for in LandingPad.

We can now export it to a CSV for the next section, but to make the answers consistent, in the next lab we will provide data in a pre-selected date range.

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

```
[58]: data_falcon9.isnull().sum()
```

FlightNumber	0
Date	0
BoosterVersion	94
PayloadMass	0
Orbit	0
LaunchSite	0
Outcome	0
Flights	0
GridFins	0
Reused	0
Legs	0
LandingPad	30
Block	4
ReusedCount	0
Serial	0
Longitude	0
Latitude	0
dtype: int64	

3. Dealing with Missing Values

2. Filter the dataframe to only include Falcon 9 launches

Data Collection - Scraping

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

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

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

Create a BeautifulSoup object from the HTML response

[6]: # Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(response.text, 'html')

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

[8]: # Use soup.title attribute
soup.title

[8]: <title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

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

Next, we want to collect all relevant column names from the HTML table header.

Let's try to find all tables on the wiki page first. If you need to refresh your memory about `BeautifulSoup`, please check the external reference link towards end of this lab.

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

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

```
[10]: # 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: 100%;">
<tbody><tr>
<th scope="col">Flight No.</th>
</tr>
<tr>
<th scope="col">Date and/or time (<a href="/wiki/Coordinated_Universal_Time" title="Coordinated Universal Time">UTC</a>)</th>
<th scope="col"><a href="/wiki/List_of_Falcon_9_first-stage_boosters" title="List of Falcon 9 first-stage boosters">Version, <br/>Booster</a> <sup class="reference" id="cite_ref-booster-11-0"><a href="#cite_note-booster-11">span class="cite-bracket">[</span><span class="cite-bracket">]</span></a></sup></th>
```

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

```
[11]: 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 Non-empty column name ('if name is not None and len(name) > 0') into a list called column_names
tc = first_launch_table.find_all('th')
for th in tc:
    name = extract_column_from_header(th)
    if name is not None and len(name) > 0:
        column_names.append(name)
```

Check the extracted column names

```
[12]: print(column_names)
```

1. Request the Falcon9 Launch Wiki page from its URL

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

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

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

```
[13]: launch_dict= dict.fromkeys(column_names)

# Remove an irrelevant 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']= []
```

Next, we just need to fill up the `launch_dict` with launch records extracted from table rows.

Usually, HTML tables in Wiki pages are likely to contain unexpected annotations and other types of noises, such as reference links B0004.1[8], missing values N/A [e], inconsistent formatting, etc.

To simplify the parsing process, we have provided an incomplete code snippet below to help you to fill up the `launch_dict`. Please complete the following code snippet with TODOs or you can choose to write your own logic to parse all launch tables:

```
[14]: 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
        else:
            flight_number=rows.td.string
            flag=True
    if flag==True:
        extracted_row+=1
        print(extracted_row,":",flight_number)

[15]: for k in launch_dict.keys():
    print("Key {} => Len {}".format(k, len(k)))

Key Flight No. => Len 10
Key Launch Date => Len 11
Key Payload => Len 7
Key Payload mass => Len 12
Key Orbit => Len 5
Key Customer => Len 8
Key Launch outcome => Len 14
Key Mission Booster => Len 15
Key Boosted Launch => Len 15
Key Date => Len 4
Key Time => Len 4
```

After you have filled in the parsed launch record values into `launch_dict`, you can create a dataframe from it.

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

We can now export it to a **<CSV>** for the next section, but to make the answers consistent and in case you have difficulties finishing this lab

```
[19]: df.to_csv('spacex_web_scraped.csv', index=False)
```

3. Create a data frame by parsing the launch HTML tables

Data Wrangling

TASK 1: Calculate the number of launches on each site

The data contains several Space X launch facilities: Cape Canaveral Space Launch Complex 40 **VAFB SLC 4E**, Vandenberg Air Force Base Space Launch Complex 4E (**SLC-4E**), Kennedy Space Center Launch Complex 39A **KSC LC 39A**. The location of each Launch is placed in the column `LaunchSite`.

Next, let's see the number of launches for each site.

Use the method `.value_counts()` on the column `LaunchSite` to determine the number of launches on each site:

```
[8]: # Apply .value_counts() on column LaunchSite  
df['LaunchSite'].value_counts()  
  
[8]: 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

Use the method `.value_counts()` to determine the number and occurrence of each orbit in the column `Orbit`

```
[9]: # Apply .value_counts on Orbit column  
df['Orbit'].value_counts()  
  
[9]: 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
```

1. Calculate the number of launches on each site

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

Use the method `.value_counts()` on the column `Outcome` to determine the number of `landing_outcomes`. Then assign it to a variable `landing_outcomes`.

```
[10]: # landing_outcomes = values on Outcome column  
df['Outcome'].value_counts()  
  
[10]: 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  
None count, dtype: int64
```

`True Ocean` means the mission outcome was successfully landed to a specific region of the ocean while `False Ocean` means the mission outcome was unsuccessfully landed to a specific region of the ocean. `True RTLS` means the mission outcome was successfully landed to a ground pad. `False RTLS` means the mission outcome was unsuccessfully landed to a ground pad. `True ASDS` means the mission outcome was successfully landed to a drone ship. `False ASDS` means the mission outcome was unsuccessfully landed to a drone ship. `None ASDS` and `None None`: these represent a failure to land.

```
[11]: # Calculate the value counts of the Outcome column  
landing_outcomes = df['Outcome'].value_counts()  
  
# Iterate over the landing outcomes and print them  
for i, (outcome, count) in enumerate(landing_outcomes.items()):  
    print(i, outcome, count)
```

```
0 True ASDS 41  
1 None None 19  
2 True RTLS 14  
3 False ASDS 6  
4 None None 5  
5 False Ocean 2  
6 None ASDS 2  
7 False RTLS 1
```

We create a set of outcomes where the second stage did not land successfully:

```
[12]: bad_outcomes = set(landing_outcomes.keys())[1,3,5,6,7]  
bad_outcomes  
  
[12]: {'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

3. Calculate the number and occurrence of mission outcome of the orbits

2. Calculate the number and occurrence of each orbit

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`:

```
[13]: # landing_class = 0 if bad_outcome  
# landing_class = 1 otherwise  
landing_class = df['Outcome'].map(lambda x: 0 if x in bad_outcomes else 1)
```

This variable will represent the classification variable that represents the outcome of each launch. If the value is zero, the first stage did not land successfully; one means the first stage landed successfully

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

Class
0
1
2
3
4
5
6
7

```
[14]: df.head(5)  
  
[14]:   FlightNumber  Date  BoosterVersion  PayloadMass  Orbit  LaunchSite  Outcome  Flights  GridFins  Reused  Legs  LandingPad  Block  ReusedCount  Serial  Longitude  Latitude  Class  
0           1  2010-06-04  Falcon 9  6104.959412  LEO  CCAFS SLC 40  None None    1  False  False  NaN  1.0    0  B0003  -80.77366  28.561857  0  
1           2  2012-05-22  Falcon 9  5254.000000  LEO  CCAFS SLC 40  None None    1  False  False  NaN  1.0    0  B0005  -80.77366  28.561857  0  
2           3  2013-03-01  Falcon 9  677.000000  ISS  CCAFS SLC 40  None None    1  False  False  NaN  1.0    0  B0007  -80.77366  28.561857  0  
3           4  2013-09-29  Falcon 9  500.000000  PO  VAFB SLC 4E  False Ocean  1  False  False  NaN  1.0    0  B1003  -120.810829  34.632093  0  
4           5  2013-12-03  Falcon 9  3170.000000  GTO  CCAFS SLC 40  None None    1  False  False  NaN  1.0    0  B1004  -80.77366  28.561857  0
```

4. Create a landing outcome label from Outcome column

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

```
[15]: df['Class'].mean()
```

```
[15]: np.float64(0.6666666666666666)
```

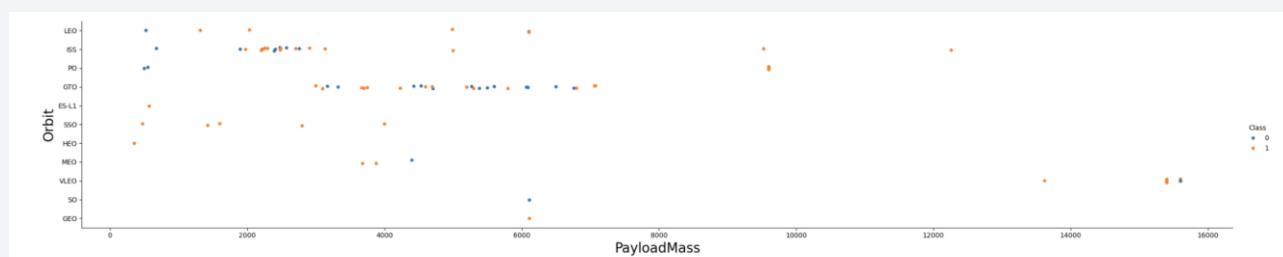
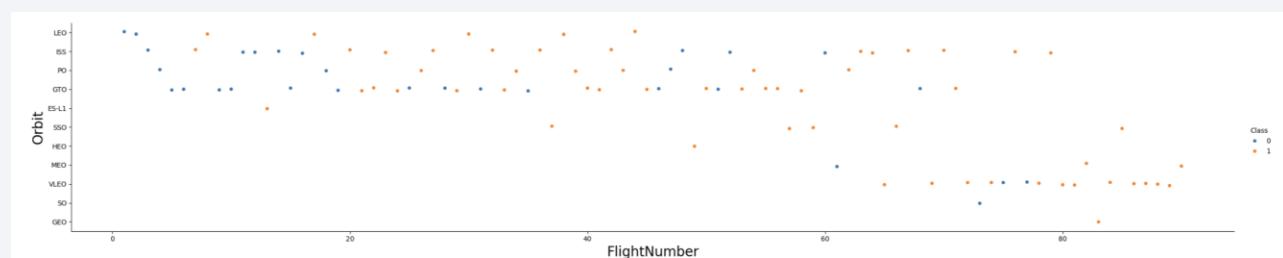
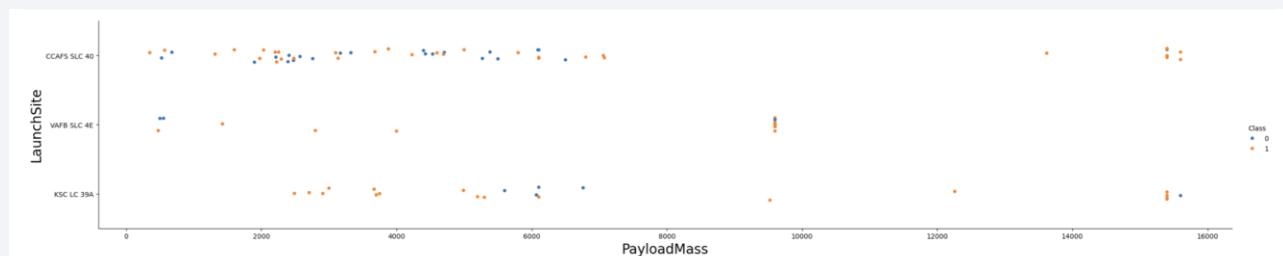
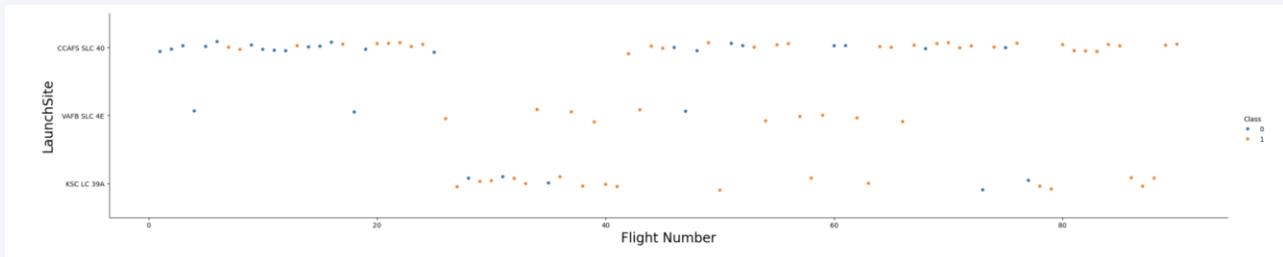
We can now export it to a CSV for the next section, but to make the answers consistent, in the next lab we will provide data in a pre-selected date range.

```
df.to_csv("dataset_part_2.csv", index=False)
```

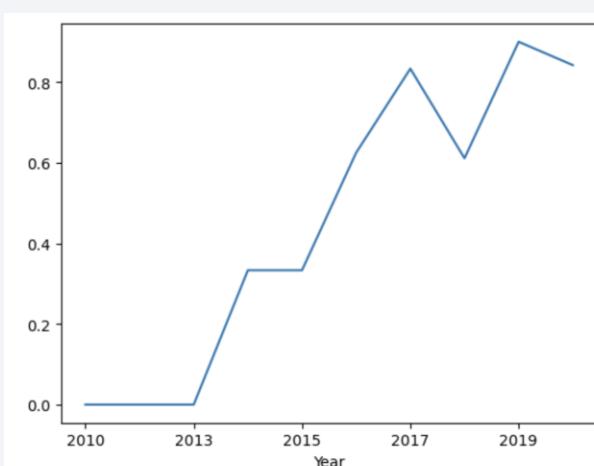
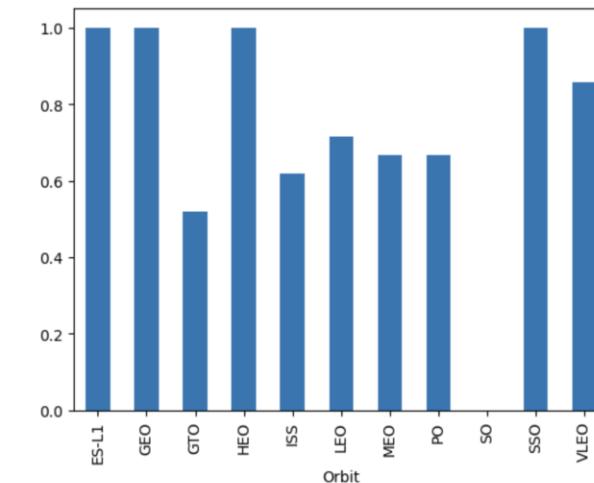
<https://github.com/jay1enbailey/Applied-Data-Science-Capstone/blob/main/labs-jupyter-spacex-Data%20wrangling.ipynb>

EDA with Data Visualization

Visualization of relationship between (1) Flight Number and Launch Site, (2) Payload and Launch Site, (3) Flight Number and Orbit type, (4) Payload and Orbit type



Bar chart was used to visualize the relationship between success rate of each orbit type; Line chart was used to visualize the launch success yearly trend

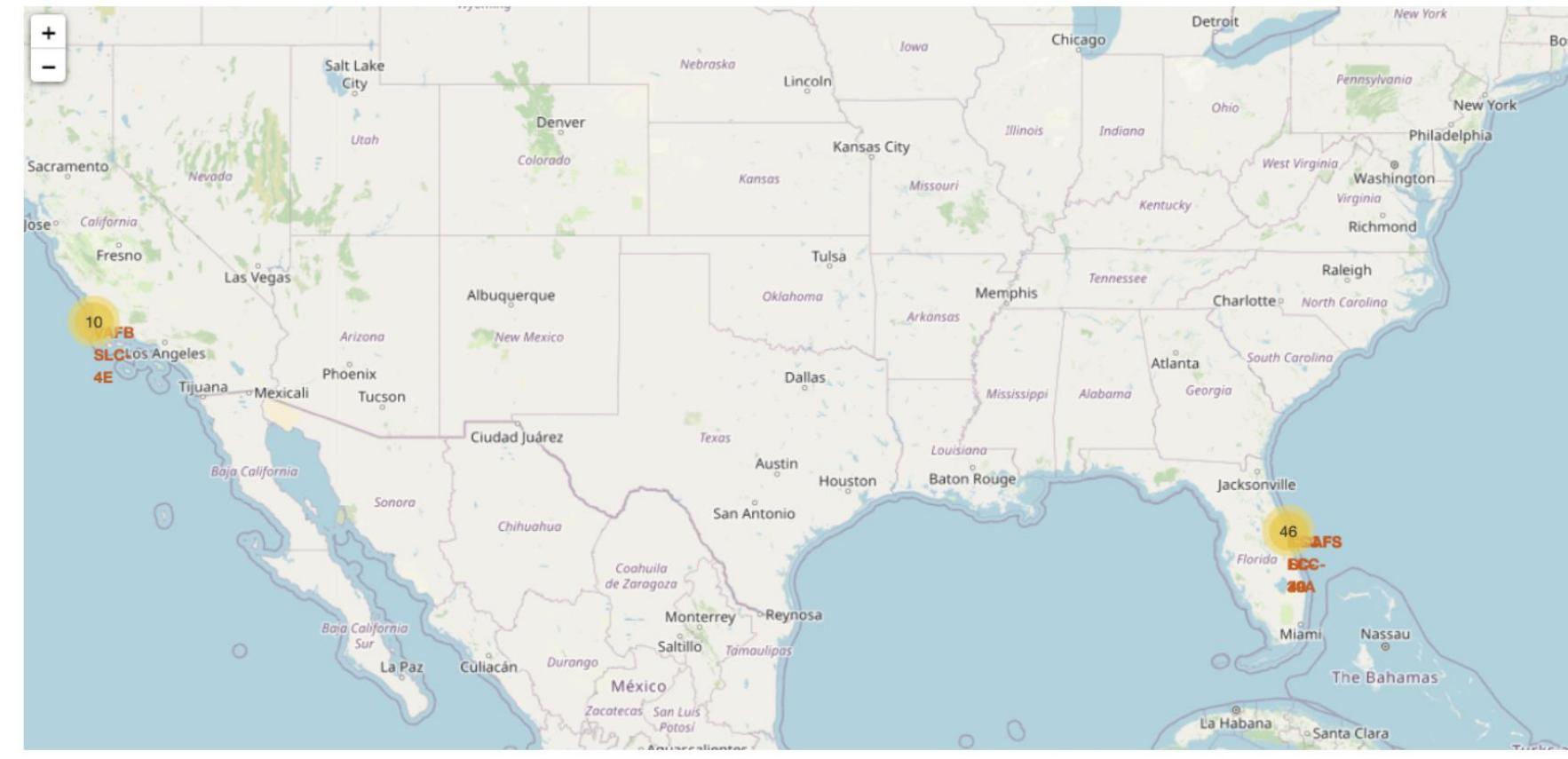


EDA with SQL

https://github.com/jay1enbailey/Applied-Data-Science-Capstone/blob/main/jupyter-labs-eda-sql-coursera_sqlite.ipynb

- Display the names of the unique launch sites in the space mission
- Display 5 records where launch sites begin with the string 'CCA'
- Display the total payload mass carried by boosters launched by NASA (CRS)
- Display average payload mass carried by booster version F9 v1.1
- List the date when the first successful landing outcome in ground pad was achieved.
- List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000
- List the total number of successful and failure mission outcomes
- List the names of the booster_versions which have carried the maximum payload mass using a subquery
- List the records which will display the month names, failure_landing_outcomes in drone ship ,booster versions, launch_site for the months in year 2015.
- Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order.

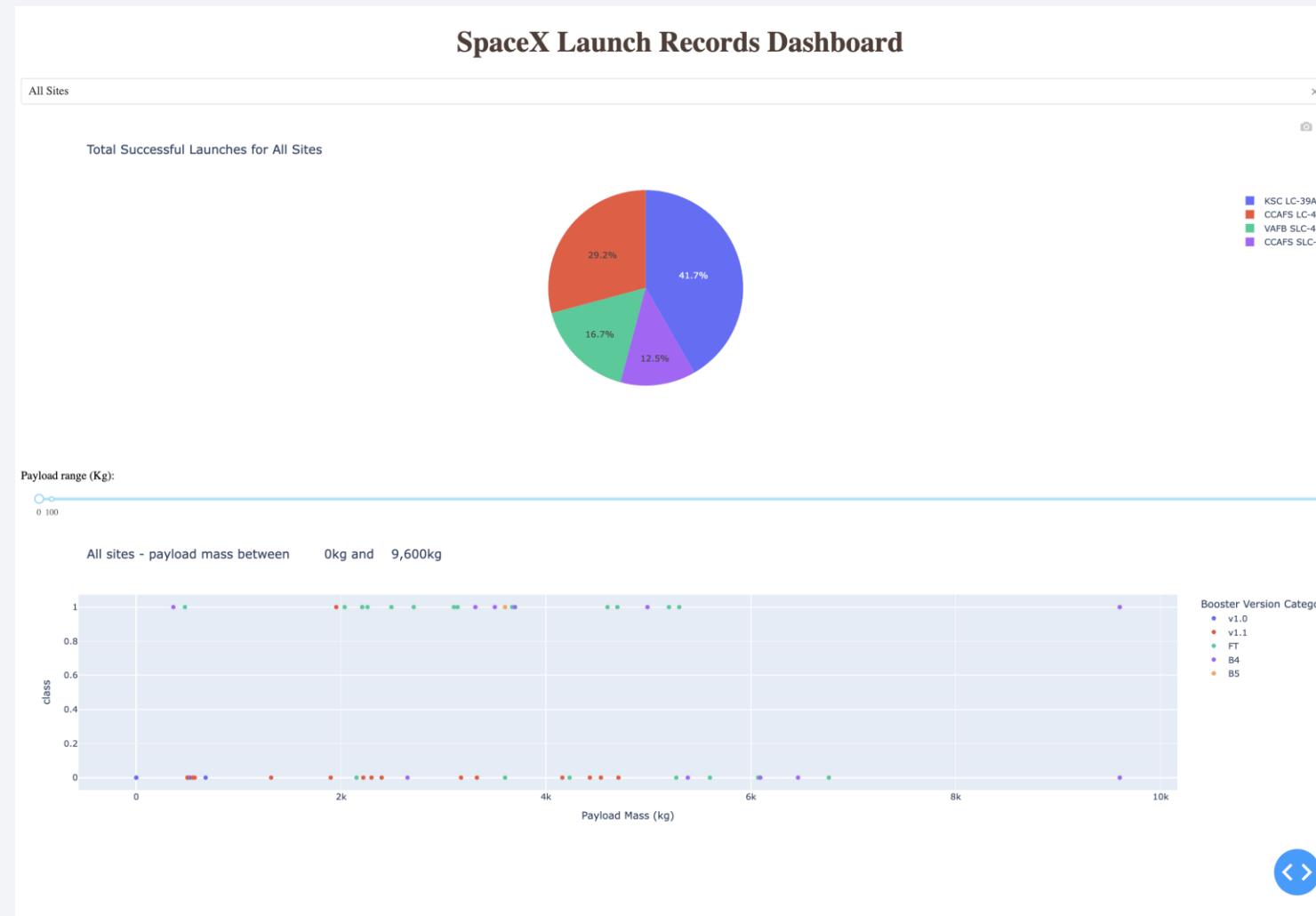
Build an Interactive Map with Folium



Map markers have been added to the map to find the right location for building a launch site

https://github.com/jay1enbailey/Applied-Data-Science-Capstone/blob/main/lab_jupyter_launch_site_location.ipynb

Build a Dashboard with Plotly Dash

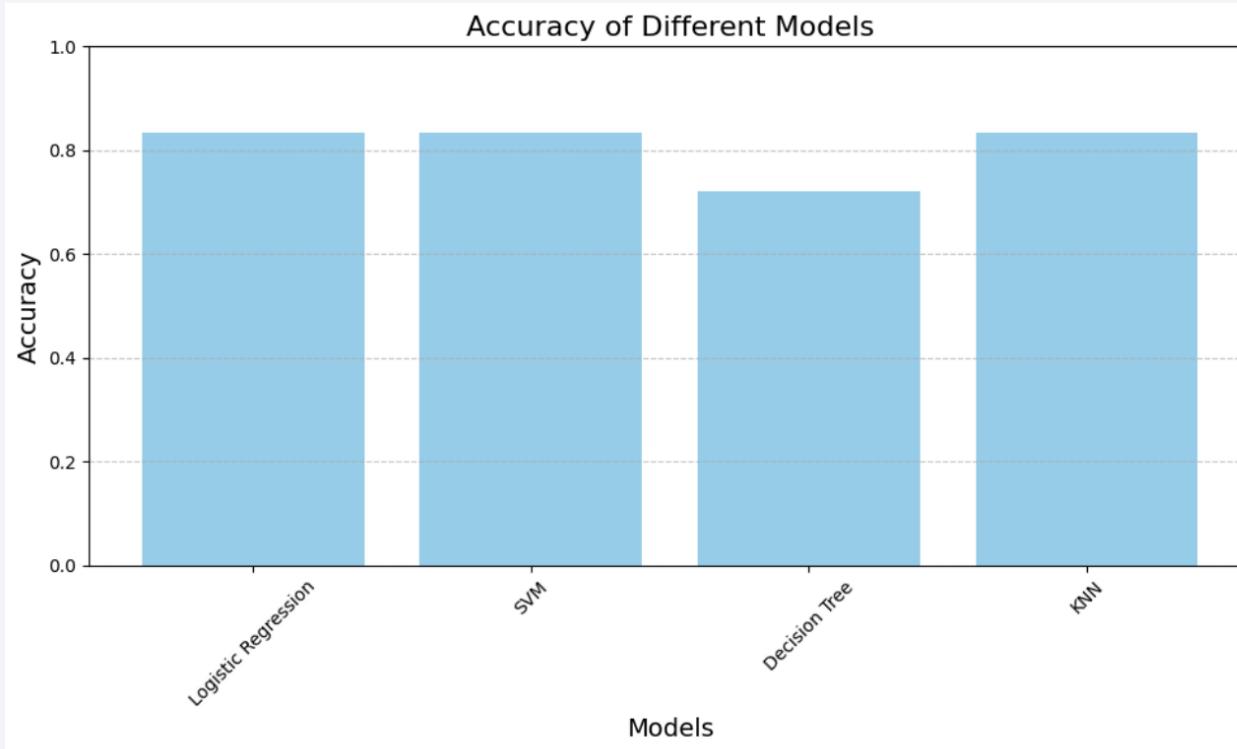


Dropdown option of pie chart created to show the success launches of all / each site

Scatter plot with payload range slider to show the success launches of all / each site by payload mass

Predictive Analysis (Classification)

https://github.com/jay1enbailey/Applied-Data-Science-Capstone/blob/main/SpaceX_Machine%20Learning%20Prediction_Part_5.ipynb



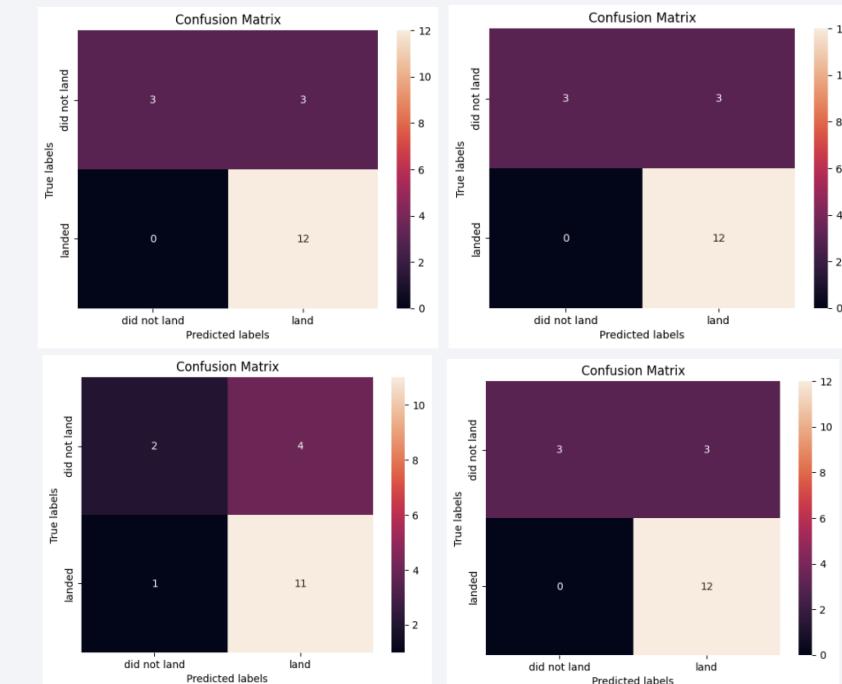
- The methods performed best are LR, SVM, KNN where all 3 achieved the highest accuracy of 83.33%.

TASK 12

Find the method performs best:

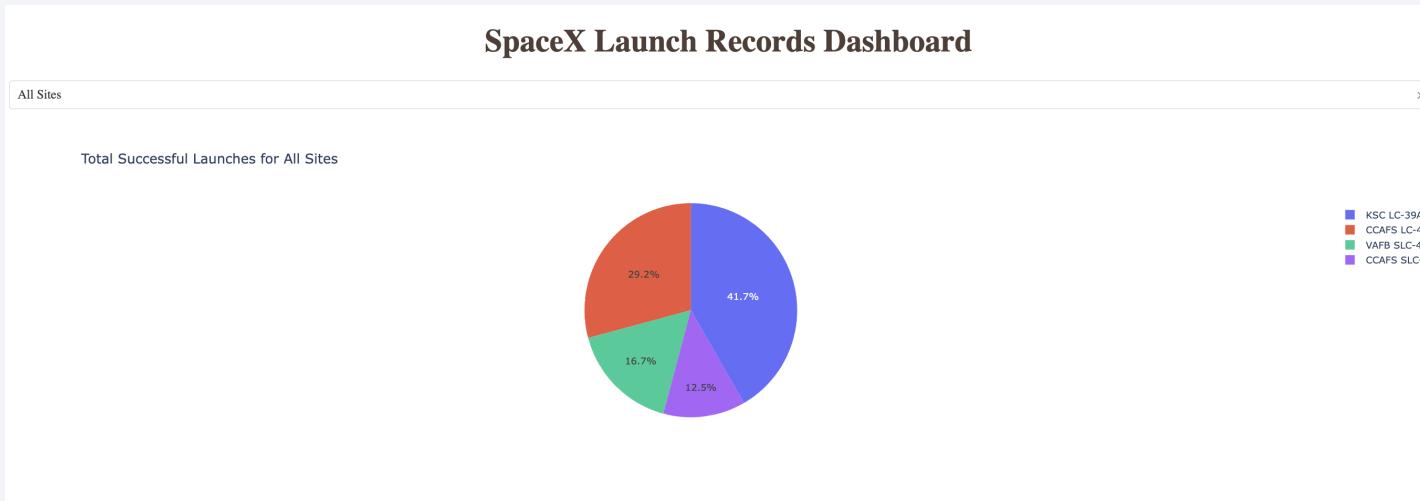
```
print('LR Accuracy:', '{:.2%}'.format(logreg_accuracy))
print('SVM Accuracy:', '{:.2%}'.format(svm_accuracy))
print('Decision Tree Accuracy:', '{:.2%}'.format(tree_accuracy))
print('KNN Accuracy:', '{:.2%}'.format(knn_accuracy))
```

LR Accuracy: 83.33%
SVM Accuracy: 83.33%
Decision Tree Accuracy: 72.22%
KNN Accuracy: 83.33%



Results

- LR, SVM, KNN performed best for forecasting outcomes in this data.
- Lighter payloads have a higher performance compared to heavier ones.
- The likelihood of a SpaceX launch succeeding increases with the number of years of experience, suggesting a trend towards flawless launches over time.
- KSC LC-39A has the highest number of successful launches compared to other launch sites.
- GEO, HEO, SSO, ES L1 exhibit the highest rates of successful launches.



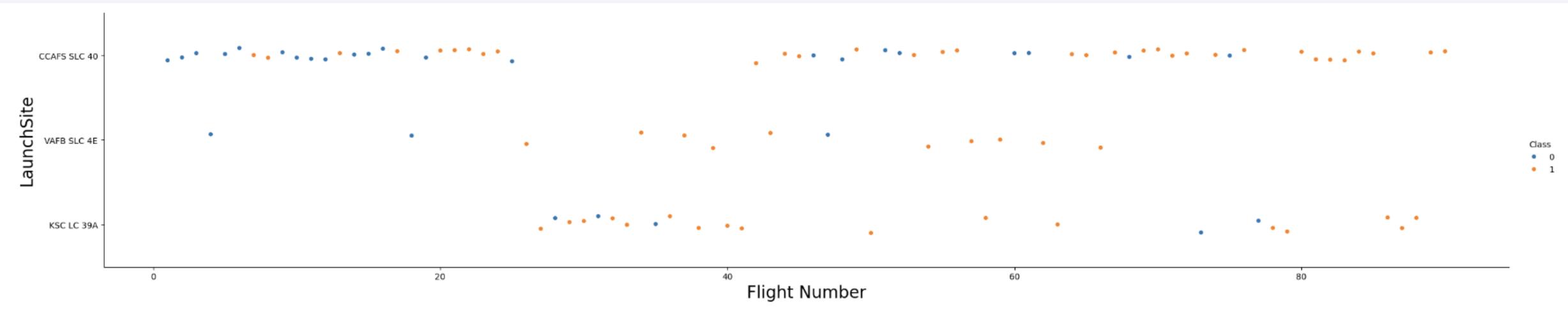
KSC LC-39A has the highest number of successful launches

The background of the slide features a complex, abstract digital visualization. It consists of numerous thin, glowing lines that create a sense of depth and motion. The lines are primarily blue and red, with some green and purple highlights. They form a grid-like structure that curves and twists across the frame, resembling a 3D wireframe or a network of data points. The overall effect is futuristic and dynamic, suggesting concepts like data flow, digital communication, or complex systems.

Section 2

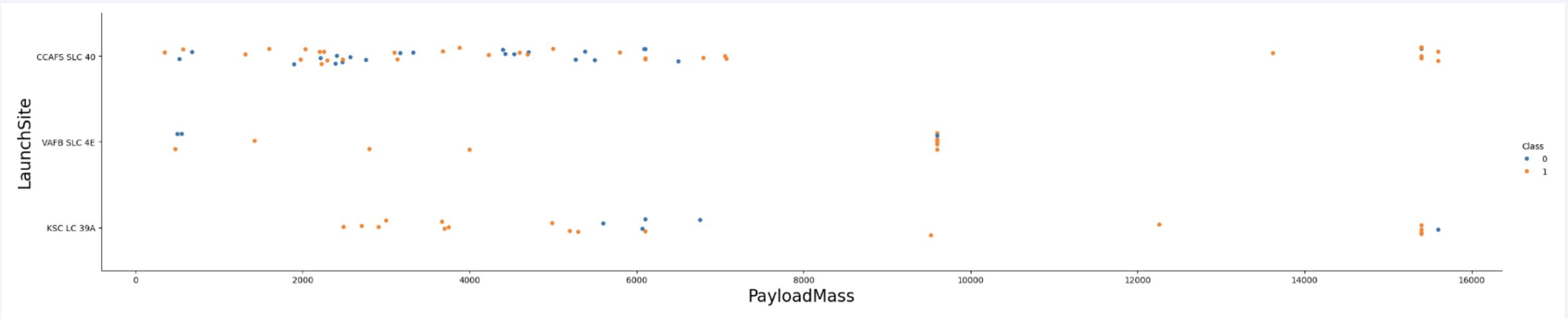
Insights drawn from EDA

Flight Number vs. Launch Site



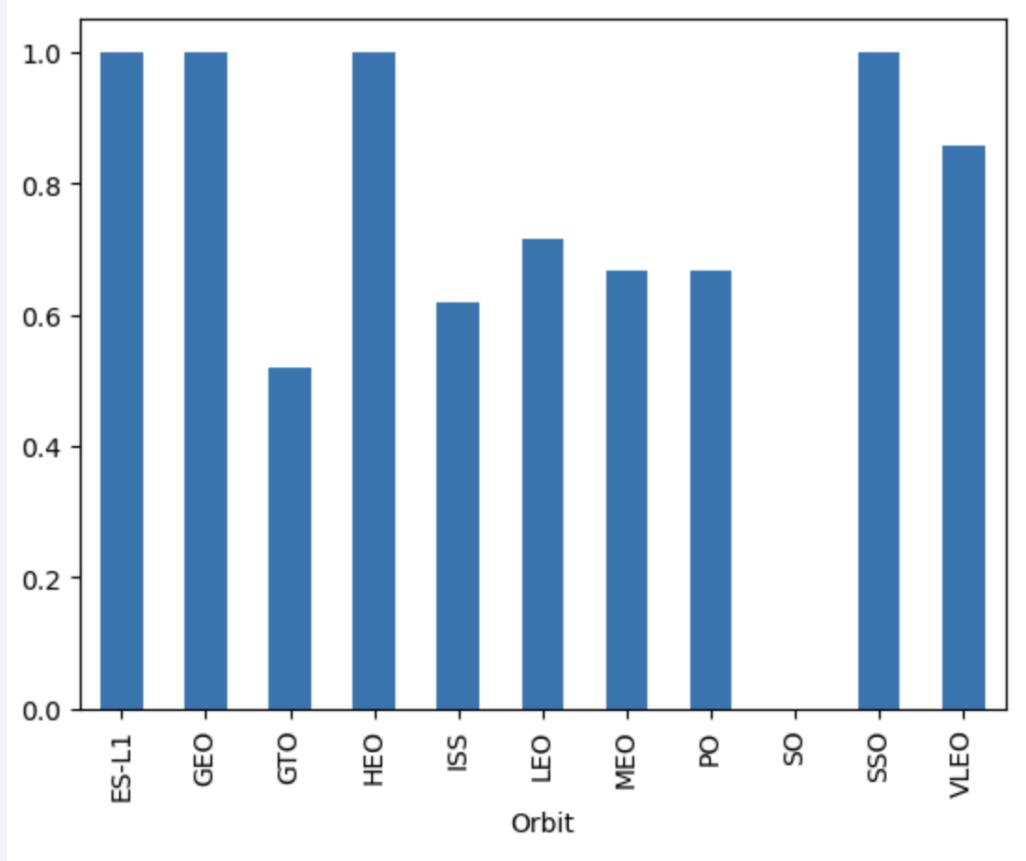
- Total number of launches from launch site CCAFS SLC 40 are higher than the other launch sites.

Payload vs. Launch Site



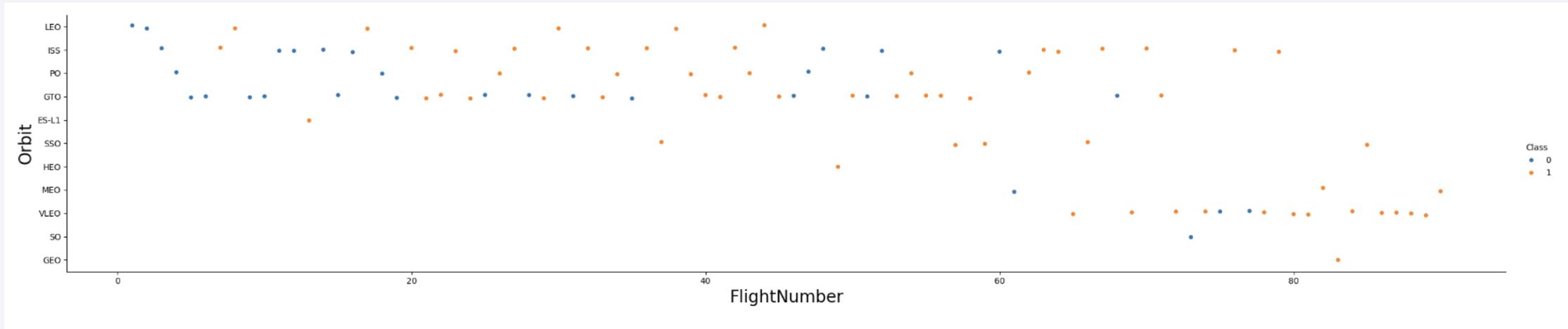
- Payloads with lower mass have more launches compared to those with higher mass across all three launch sites.

Success Rate vs. Orbit Type



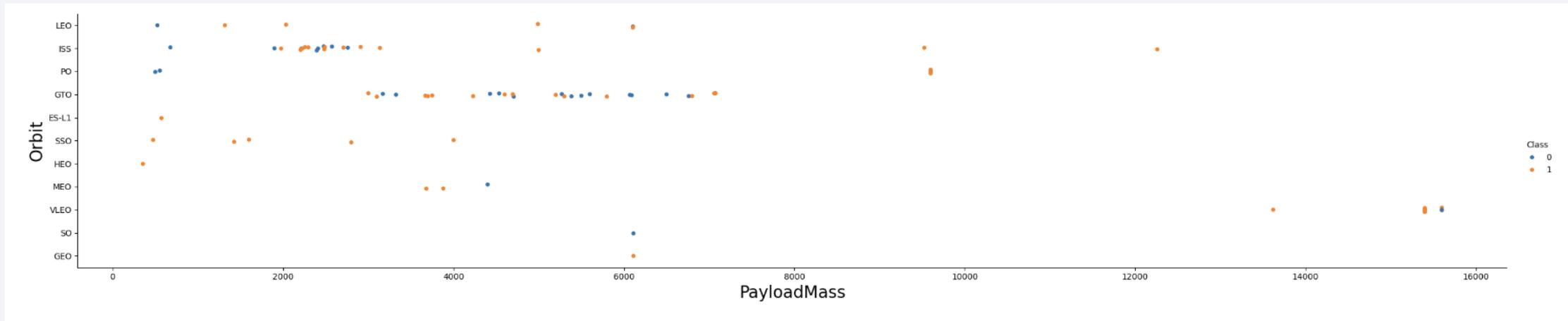
- The orbit types of ES-L1, GEO, HEO, SSO are among the highest success rates.

Flight Number vs. Orbit Type



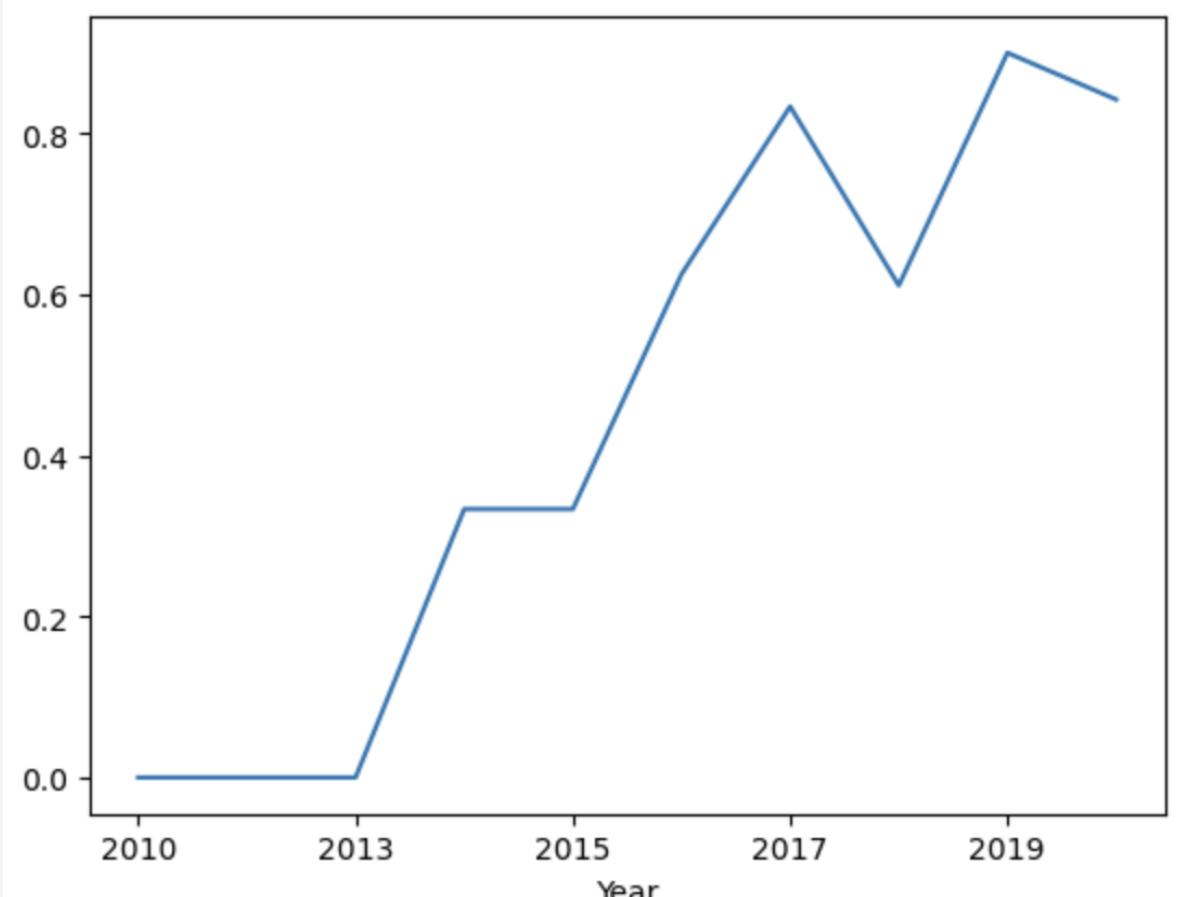
- Trends can be observed of shifting to VLEO launches in recent years.

Payload vs. Orbit Type



- Heavy payload tends to have higher successful landing rates for PO, LEO, and ISS orbits. GTO orbit success is less predictable with a close to equal mix of successes and failures.

Launch Success Yearly Trend



- The launch success rates have been increasing since 2013 till 2020, due to advancements in technology and experience.

All Launch Site Names

Task 1

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

```
[10]: sql SELECT DISTINCT LAUNCH_SITE FROM SPACEXTBL ORDER BY 1;
```

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

```
Done.
```

```
[10]: Launch_Site
```

```
CCAFS LC-40
```

```
CCAFS SLC-40
```

```
KSC LC-39A
```

```
VAFB SLC-4E
```

- Performed an SQL query to obtain all launch site names

Launch Site Names Begin with 'CCA'

Task 2

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

```
[11]: sql SELECT * FROM SPACEXTBL WHERE LAUNCH_SITE LIKE 'CCA%' LIMIT 5;  
* sqlite:///my_data1.db  
Done.
```

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

- Find 5 records where launch sites begin with `CCA`

Total Payload Mass

Task 3

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

```
[13]: sql SELECT SUM(PAYLOAD_MASS__KG_) AS TOTAL_PAYLOAD FROM SPACEXTBL WHERE PAYLOAD LIKE '%CRS%';
      * sqlite:///my_data1.db
Done.  
[13]: TOTAL_PAYLOAD
      111268
```

- Calculate the total payload carried by boosters from NASA

Average Payload Mass by F9 v1.1

Task 4

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

```
[15]: sql SELECT AVG(PAYLOAD_MASS__KG_) AS AVG_PAYLOAD FROM SPACEXTBL WHERE BOOSTER_VERSION = 'F9 v1.1';  
* sqlite:///my_data1.db  
Done.  
[15]: AVG_PAYLOAD  
2928.4
```

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

First Successful Ground Landing Date

Task 5

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

Hint:Use min function

```
[22]: %%sql
SELECT MIN(DATE) AS FIRST_SUCCESS_GP
FROM SPACEXTBL
WHERE "Landing_Outcome" = 'Success (ground pad)';
```

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

```
[22]: FIRST_SUCCESS_GP
2015-12-22
```

- Find the dates of the first successful landing outcome on ground pad

Successful Drone Ship Landing with Payload between 4000 and 6000

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

```
[23]: %%sql  
SELECT DISTINCT BOOSTER_VERSION  
FROM SPACEXTBL  
WHERE PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000  
AND "Landing_Outcome" = 'Success (drone ship)';
```

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

```
[23]: Booster_Version  
F9 FT B1022  
F9 FT B1026  
F9 FT B1021.2  
F9 FT B1031.2
```

- List the names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000

Total Number of Successful and Failure Mission Outcomes

Task 7

List the total number of successful and failure mission outcomes

```
[24]: sql SELECT MISSION_OUTCOME, COUNT(*) AS QTY FROM SPACEXTBL GROUP BY MISSION_OUTCOME ORDER BY MISSION_OUTCOME;  
* sqlite:///my_data1.db  
Done.
```

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

- Calculate the total number of successful and failure mission outcomes

Boosters Carried Maximum Payload

Task 8

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

```
[25]: sql SELECT DISTINCT BOOSTER_VERSION FROM SPACEXTBL WHERE PAYLOAD_MASS_KG_ = (SELECT MAX(PAYLOAD_MASS_KG_) FROM SPACEXTBL) ORDER BY BOOSTER_VERSION;
```

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

```
[25]: Booster_Version
```

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

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

2015 Launch Records

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.

```
[29]: %%sql
SELECT BOOSTER_VERSION, LAUNCH_SITE
FROM SPACEXTBL
WHERE "Landing_Outcome" = 'Failure (drone ship)'
AND substr(DATE, 1, 4) = '2015';
```

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

```
[29]: 

| Booster_Version | Launch_Site |
|-----------------|-------------|
| F9 v1.1 B1012   | CCAFS LC-40 |
| F9 v1.1 B1015   | CCAFS LC-40 |


```

- List the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015

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

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.

```
[30]: %%sql
SELECT "Landing_Outcome", COUNT(*) AS QTY
FROM SPACEXTBL
WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY "Landing_Outcome"
ORDER BY QTY DESC;
```

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

Landing_Outcome	QTY
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

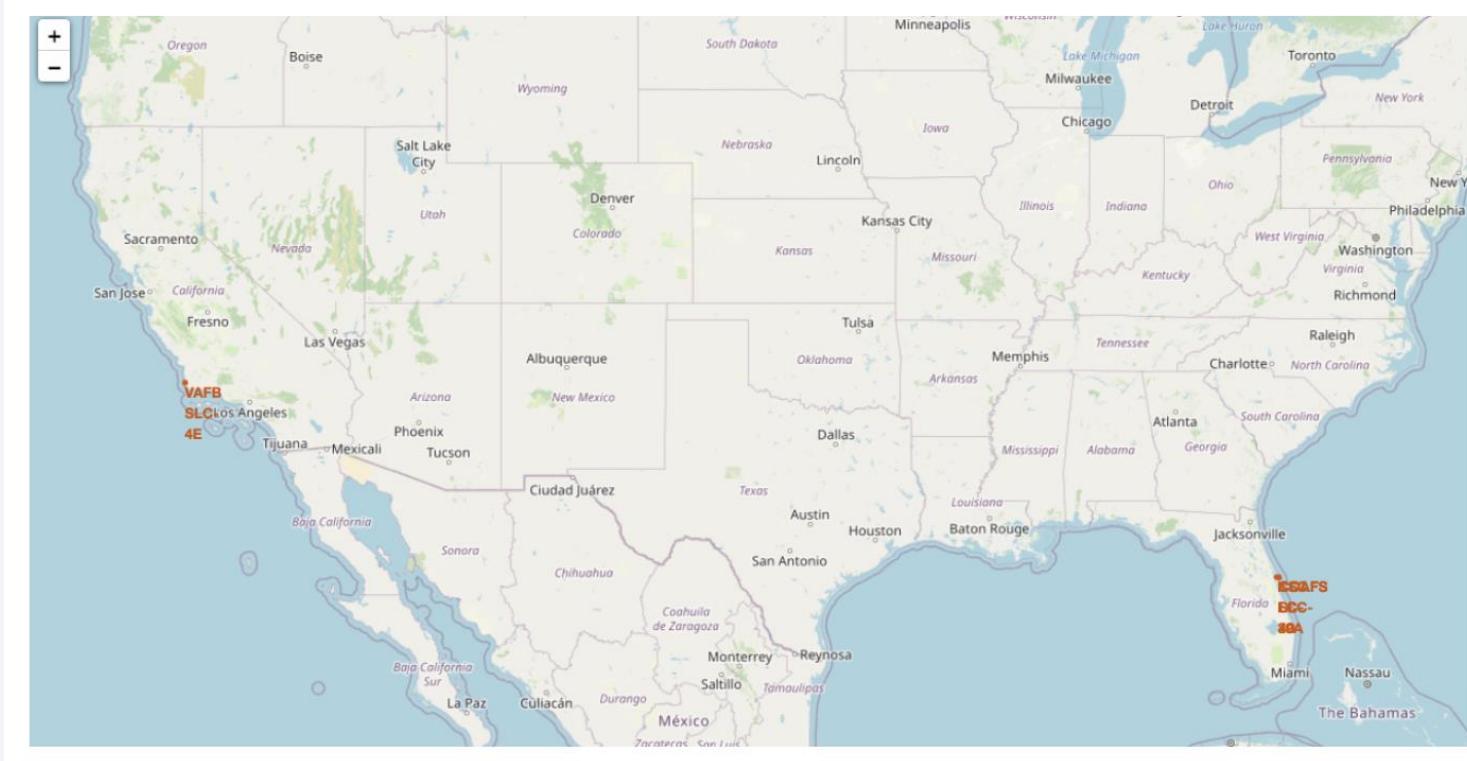
- Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order

The background of the slide is a photograph taken from space at night. It shows the curvature of the Earth's horizon against a dark blue sky. City lights are visible as numerous small white and yellow dots, primarily concentrated in the lower right quadrant where the United States appears. In the upper left quadrant, the green and yellow glow of the Aurora Borealis (Northern Lights) is visible.

Section 3

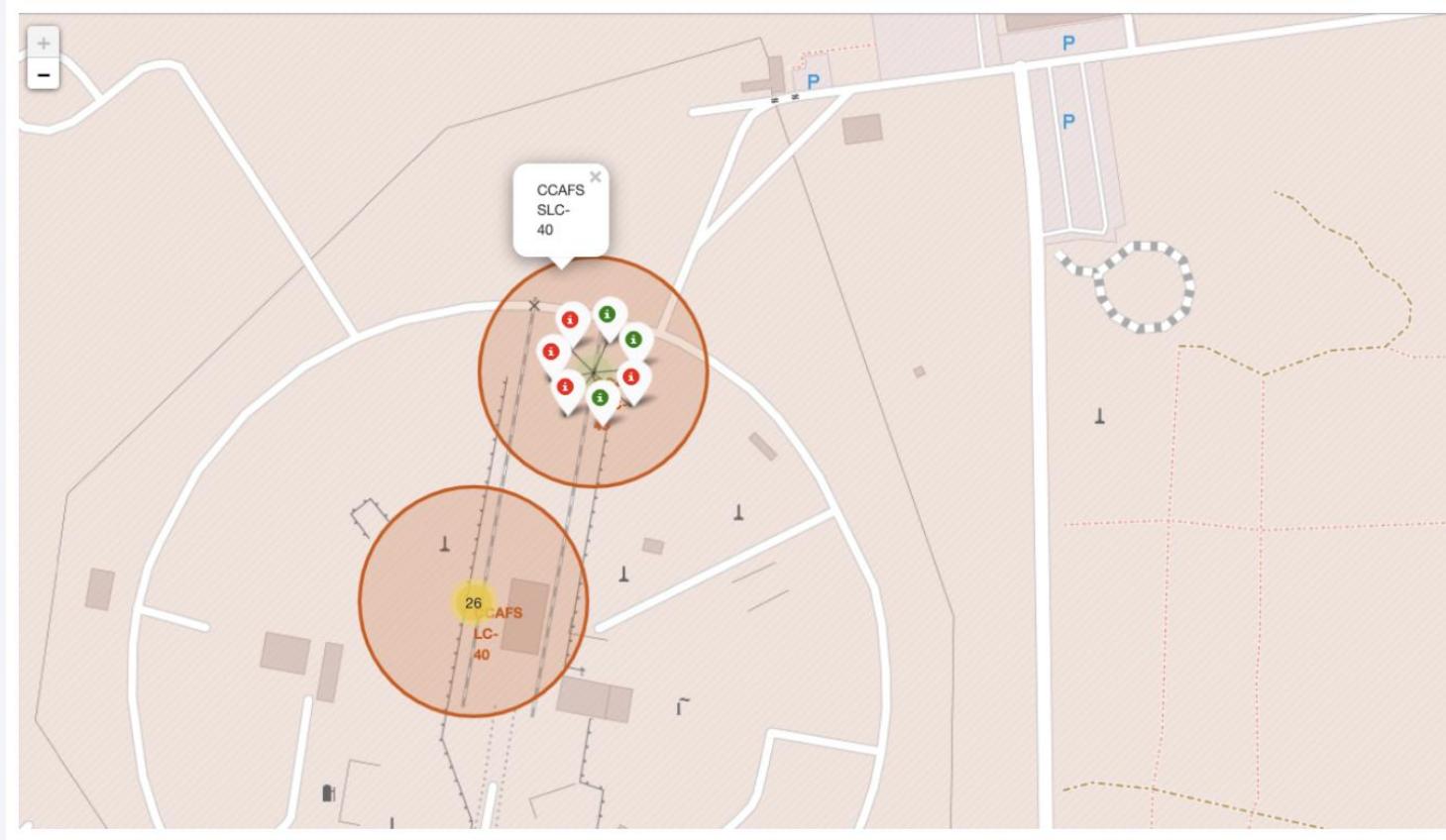
Launch Sites Proximities Analysis

All launch sites on a map



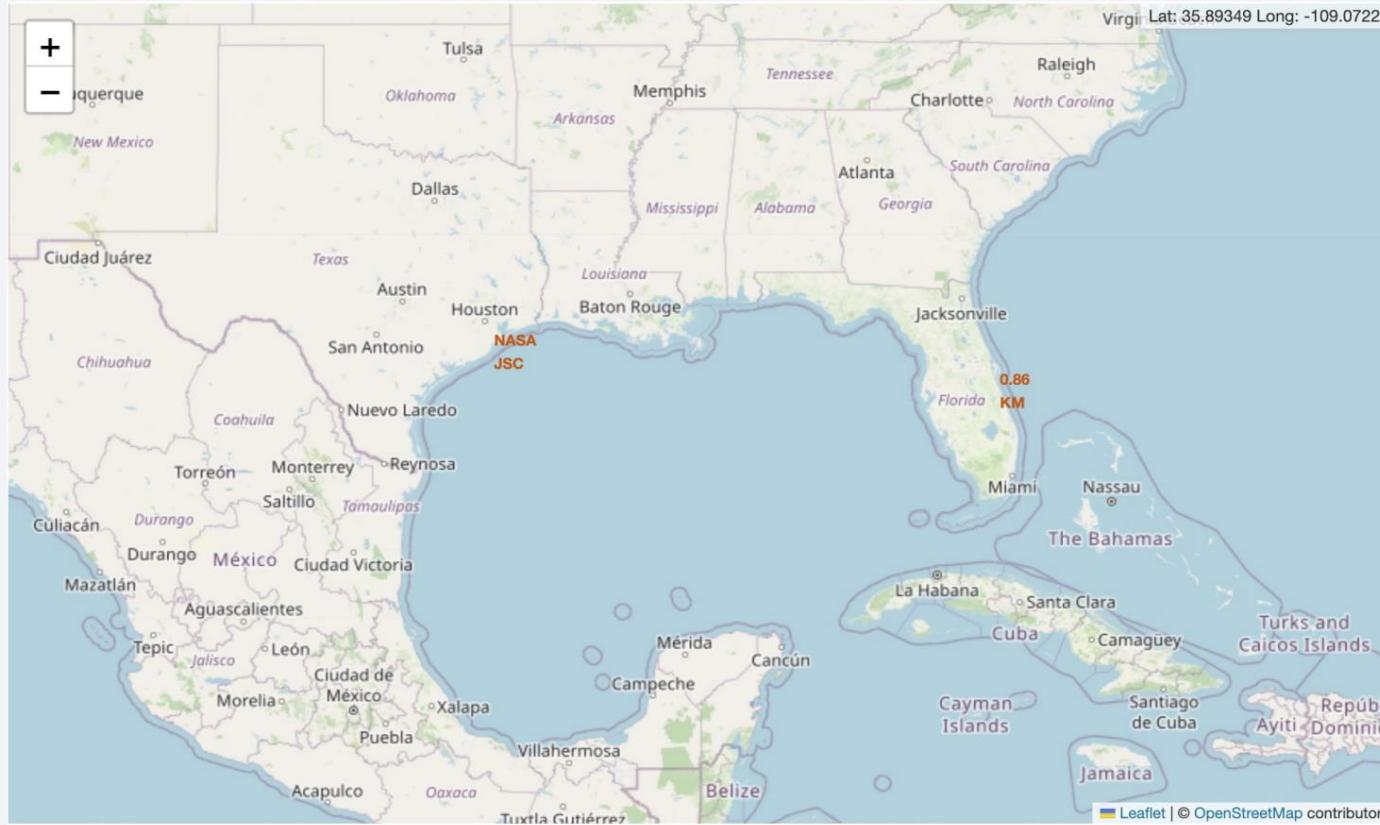
- Launch sites are labeled by a marker with their names on the map.

Success/Failed launches marked on the map



- Grouped in clusters on the map
- Labeled by green markers for successful launches, and red markers for unsuccessful ones.

Distance between a launch site to its proximities



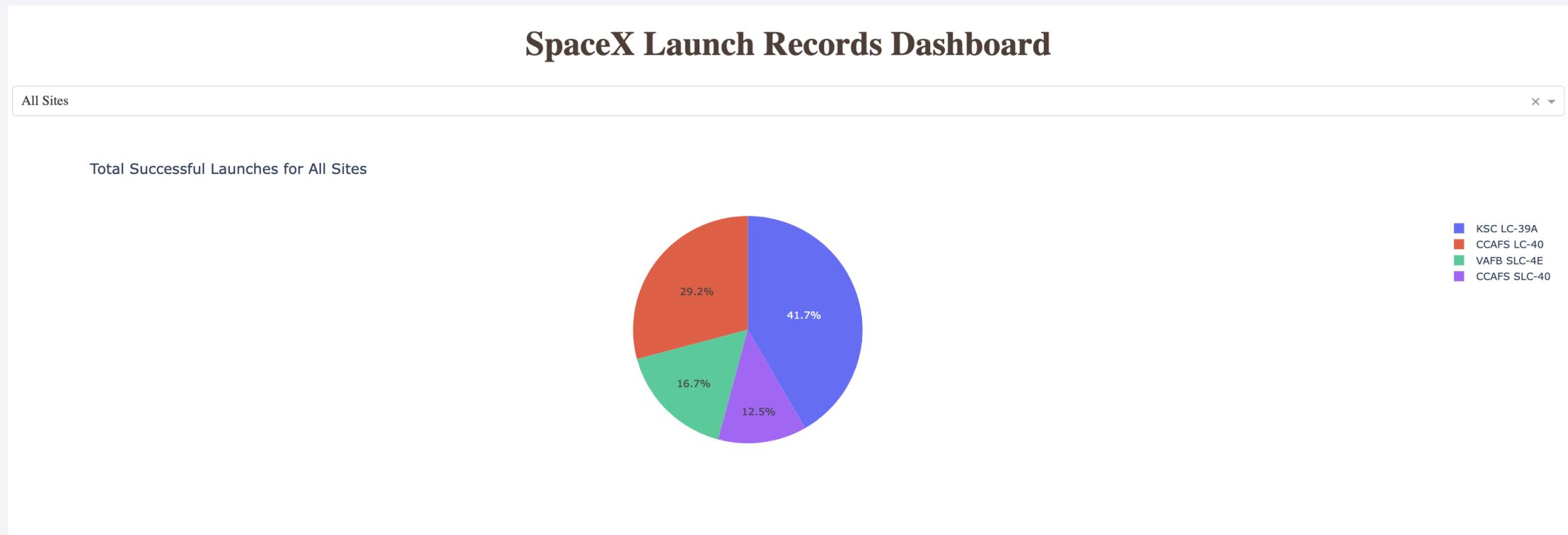
- Closest coastline from NASA JSC is marked as a point using MousePosition
- The distance between the coastline point and the launch site, which is approx. 0.86 km.

The background of the slide features a close-up photograph of a printed circuit board (PCB). The left side of the image has a blue color overlay, while the right side has a red color overlay. The PCB itself is dark grey or black, with numerous red and blue printed circuit lines (traces) connecting various components. Components visible include a large blue integrated circuit package at the top left, several smaller yellow and orange components, and a grid of surface-mount resistors on the left edge.

Section 4

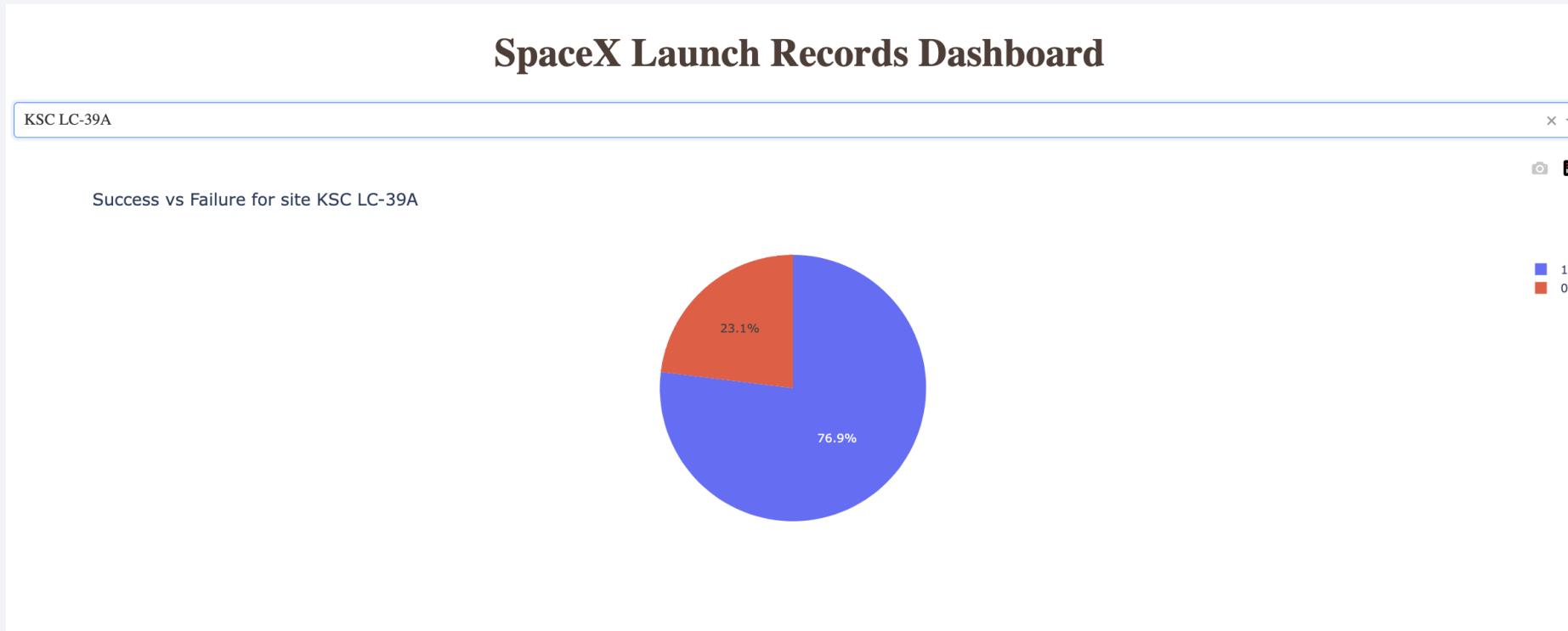
Build a Dashboard with Plotly Dash

Total success launches for all sites



- KSC LC-39A - highest amount of success launches with 41.7% from the entire record
- CCAFS SLC-40 - lowest amount of success launches with only 12.5%.

Most successful launch site



- KSC LC-39A, the launch site with highest amount of success, has a 76.9% success rate, and 23.1% failure rate.

Payload vs. Launch Outcome

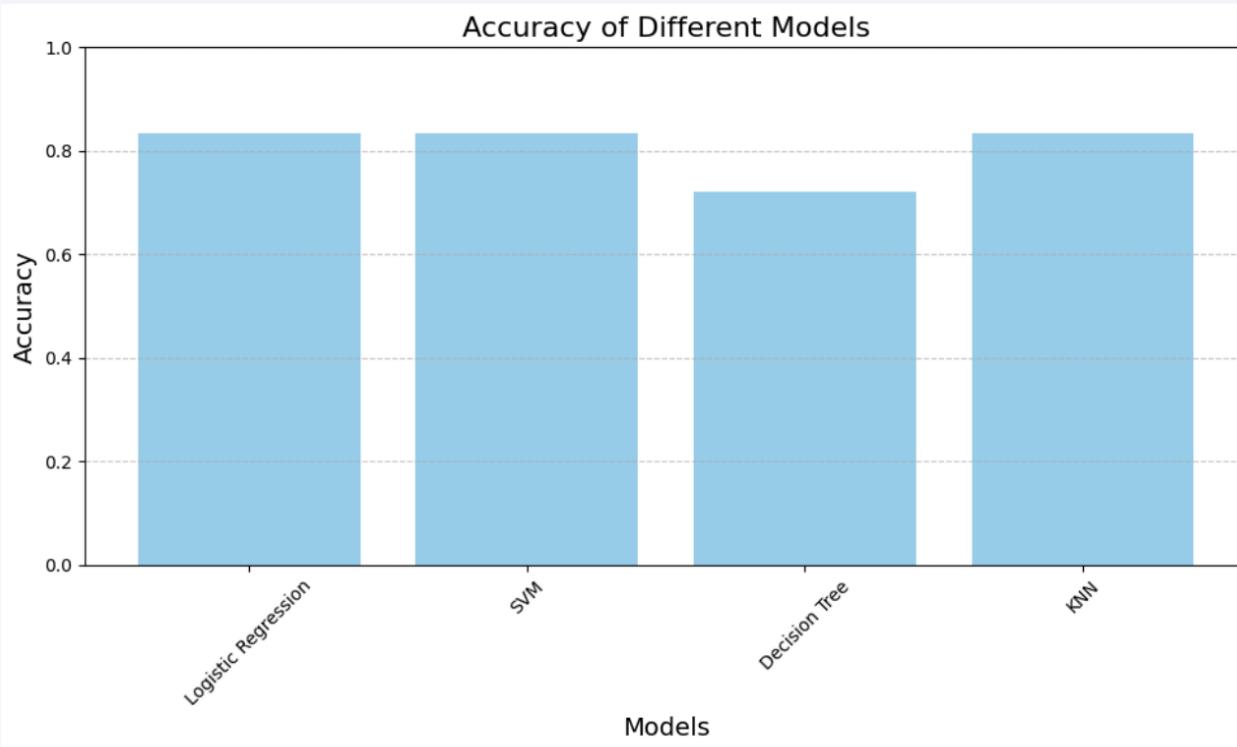


The background of the slide features a dynamic, abstract design. It consists of several thick, curved lines in shades of blue and yellow, creating a sense of motion and depth. The lines curve from the bottom left towards the top right, with some lines being more prominent than others. The overall effect is reminiscent of a tunnel or a high-speed journey through a digital space.

Section 5

Predictive Analysis (Classification)

Classification Accuracy



TASK 12

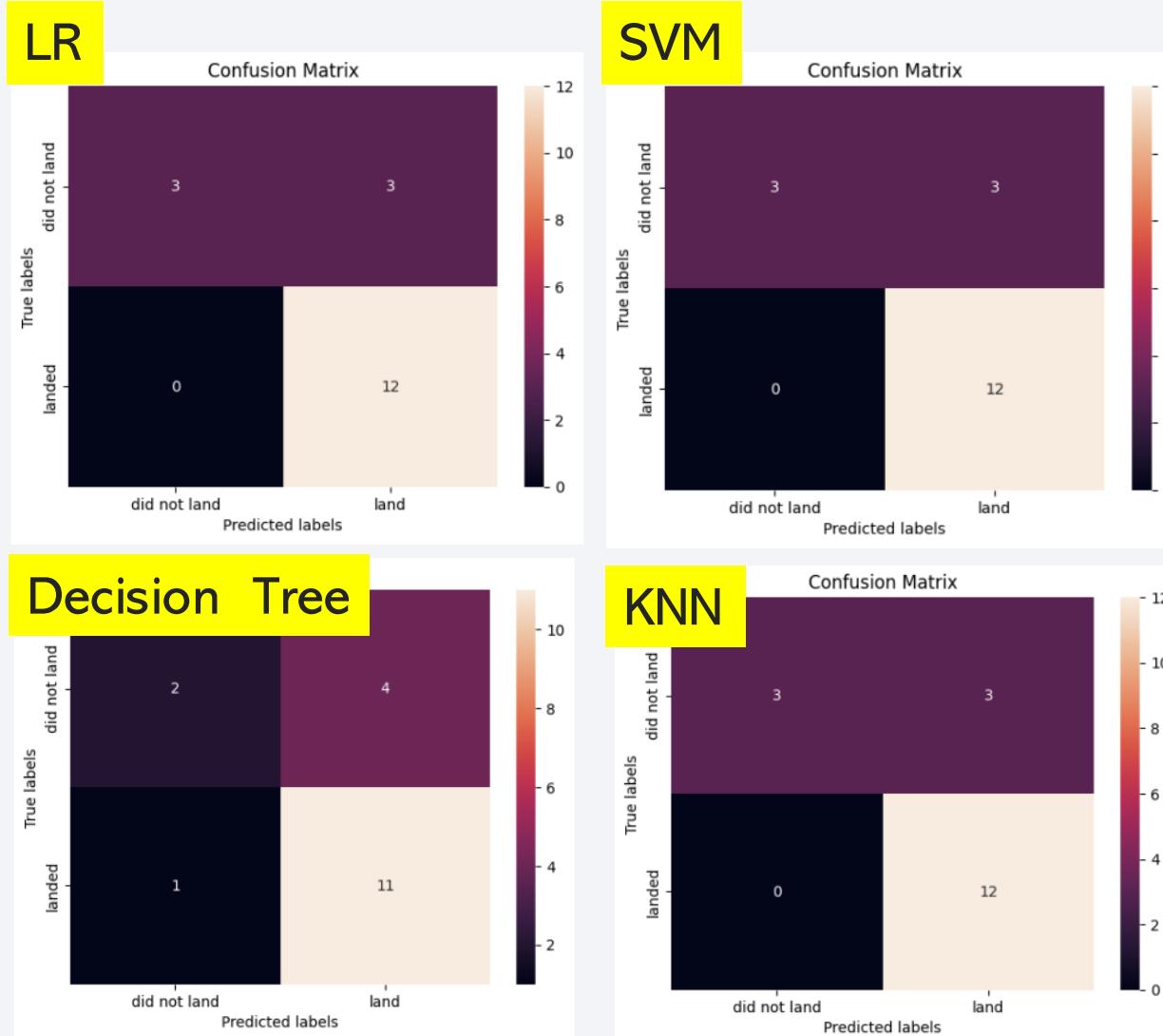
Find the method performs best:

```
print('LR Accuracy:', '{:.2%}'.format(logreg_accuracy))
print('SVM Accuracy:', '{:.2%}'.format(svm_accuracy))
print('Decision Tree Accuracy:', '{:.2%}'.format(tree_accuracy))
print('KNN Accuracy:', '{:.2%}'.format(knn_accuracy))
```

```
LR Accuracy: 83.33%
SVM Accuracy: 83.33%
Decision Tree Accuracy: 72.22%
KNN Accuracy: 83.33%
```

- The methods performed best are LR, SVM, KNN where all 3 achieved the highest accuracy of 83.33%.

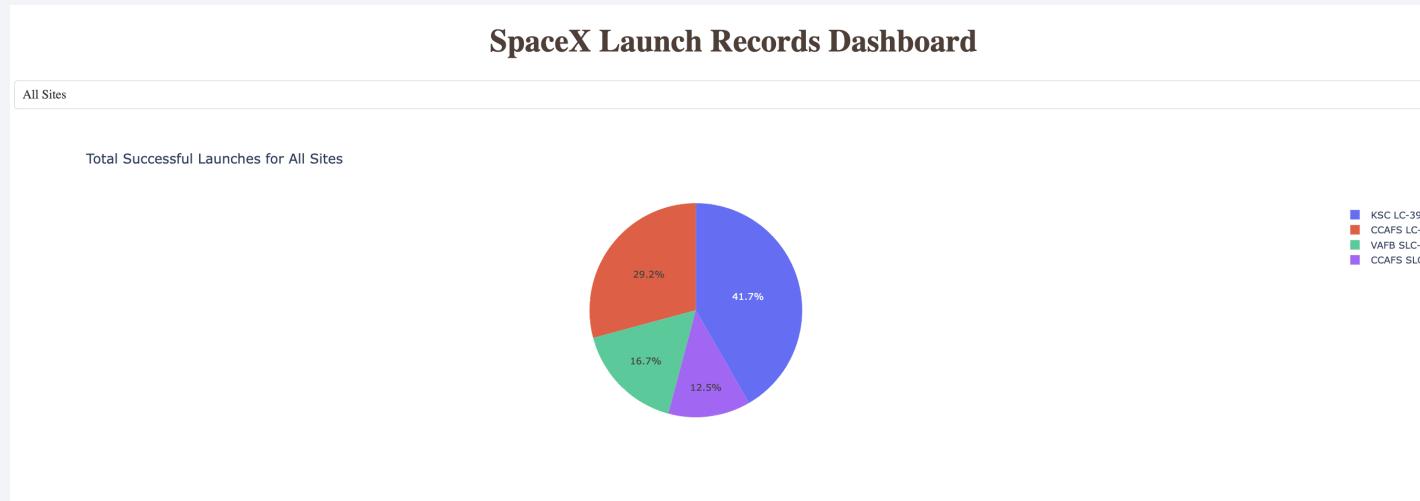
Confusion Matrix



- LR, SVM, KNN models are good
 - Their confusion matrix show that they predicted all 12 successful landing correctly, with 0 error.
- The Decision Tree model only predicted 11 successful landing correctly
 - One of them wrongly predicted as a failed / did not land.
- LR, SVM, KNN models have the same accuracy of 83.33%

Conclusions

- LR, SVM, KNN performed best for forecasting outcomes in this data.
- Lighter payloads have a higher performance compared to heavier ones.
- The likelihood of a SpaceX launch succeeding increases with the number of years of experience, suggesting a trend towards flawless launches over time.
- KSC LC-39A has the highest number of successful launches compared to other launch sites.
- GEO, HEO, SSO, ES L1 exhibit the highest rates of successful launches.



KSC LC-39A has the highest number of successful launches

Thank you!

