IBM Developer
SKILLS NETWORK

Winning the Scape Race with Data Science

SpaceX Falcon 9 Landing Analysis

Nelly Vanessa Grillo

February 11th, 2025

Outline

Executive Summary

Summary of Methodologies:

- In this project, we will use different machine learning classification algorithms to predict the successful landing of the SpaceX Falcon 9 first stage.
- The methodologies used are:
    - Data Collection, Data Wrangling and formatting
    - Exploratory Data Analysis
    - Interactive Visualization
    - Machine learning prediction: Logistic Regression, SVM, Decision Tree, and KNN

Results:

Exploratory Data Analysis:

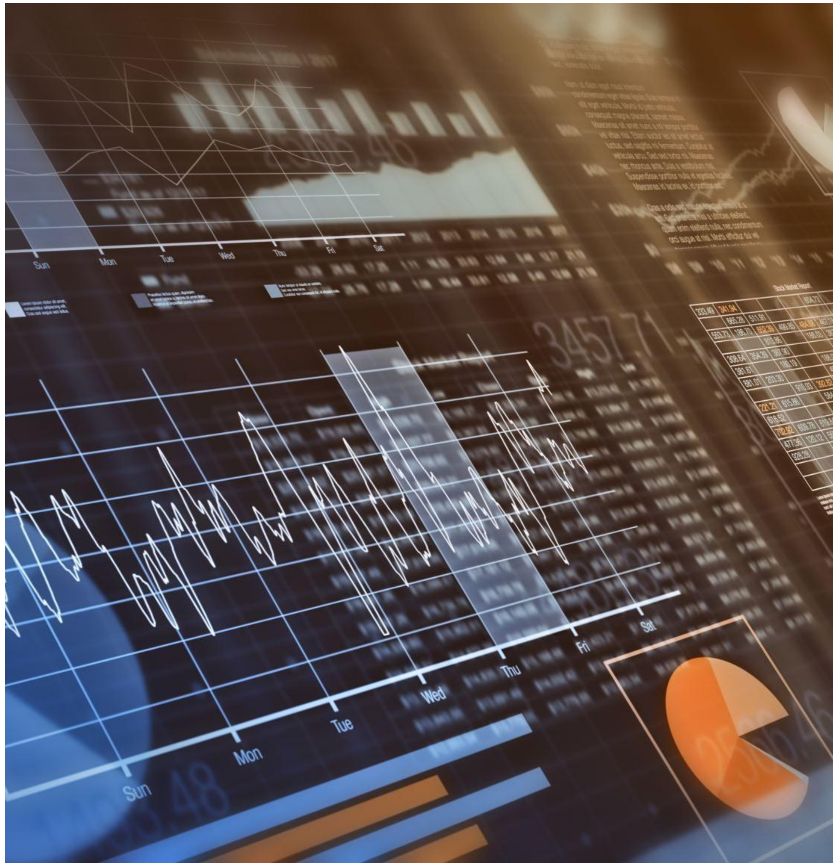
- Launch success progressively improve.
- The landing site KSC LC-39A had the highest success rate.
- The orbits ES-L1, GEO, HEO, and SSO had a 100% success rate.

Data Analysis with Visualization:

- Most launch sites are near the equator and close to the coast.

Predictive Analytics:

- The Decision Tree may be the best machine learning algorithm to predict if the Falcon 9 first stage will land successfully.



Introduction

- SpaceX advertises Falcon 9 launches at a cost of 62 million dollars, significantly lower than the 165 million dollars charged by competing providers. A major contributor to this cost difference is SpaceX's ability to reuse the first stage.
- By accurately forecasting whether the first stage will land successfully, we can estimate the overall launch cost. This predictive capability could provide valuable insights for alternative companies seeking to compete with SpaceX in rocket launch contracts.
- This project aims to predict the likelihood of a successful landing for the first stage of the Falcon 9 rocket.

Section 1

Methodology

Methodology

1. Data Collection

- Request to the SpaceX API
- Web Scraping from Wikipedia

2. Data Wrangling

- Determine the number of launches on each site, number and occurrence of each orbit, number and occurrence of mission outcome per orbit type using .value_counts()
- Create a landing outcome label: 0 (not land successfully) and 1 (land successfully)

Exploratory Data Analysis (EDA)

- Manipulate and evaluate the SpaceX dataset using SQL queries
- Visualize relationships between variables, and determine patterns using Pandas, and Matplotlib

Interactive Visual Analytics

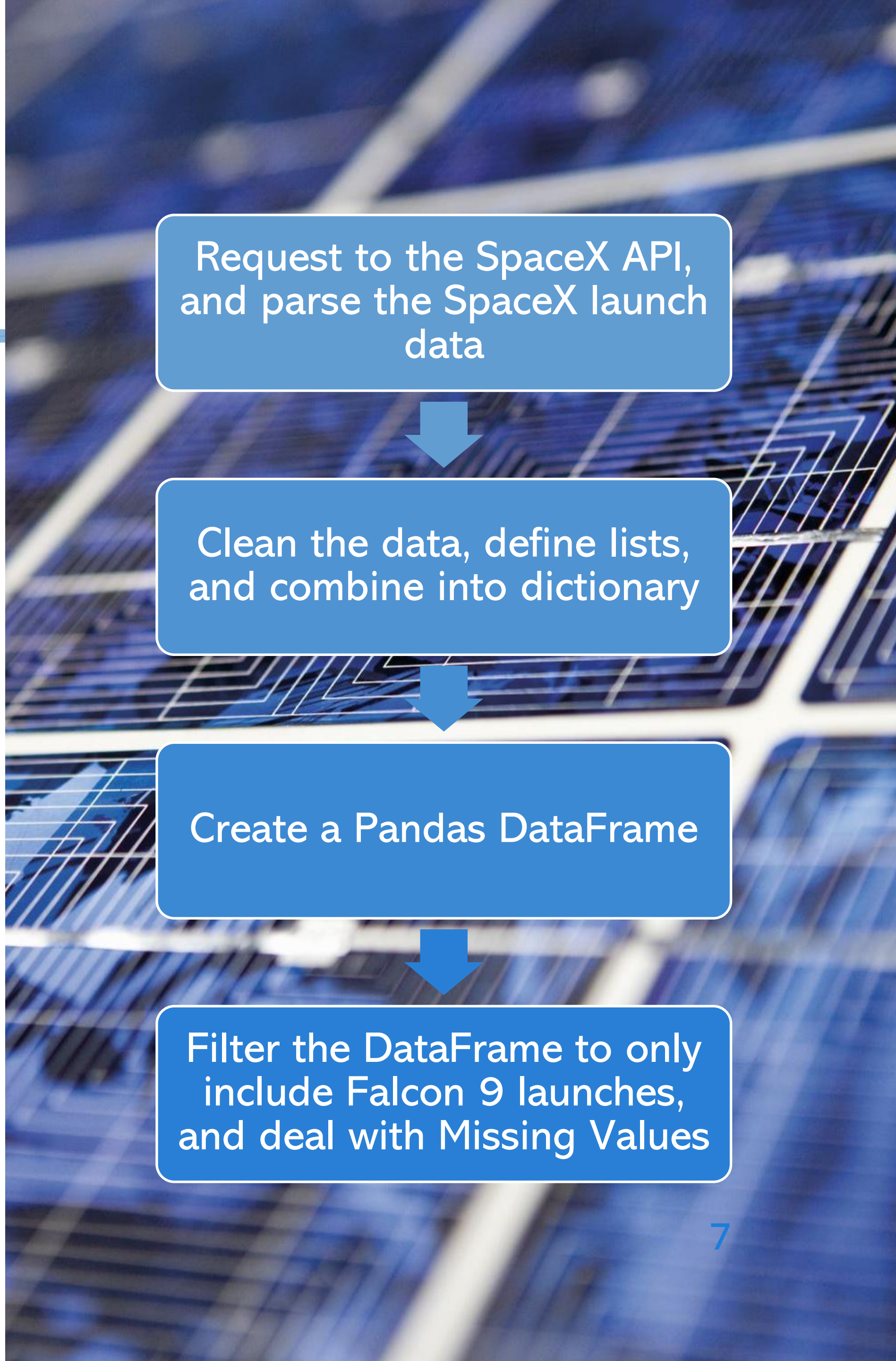
- Geospatial Analytics using Folium
- Interactive Dashboard using Plotly Dash

Data Modelling and Evaluation

- Machine Learning prediction using Logistic Regression, Support Vector Machine (SVM), Decision Tree, K-Nearest Neighbors(KNN)

Data Collection – SpaceX API

- Using the SpaceX API to retrieve data about launches, including information about the rocket used, payload delivered, launch specifications, landing specifications, and landing outcome.
- The information was extracted from a Public API https://api.spacexdata.com/v4/launches/past
- [GitHub URL to Notebook: Data Collection]



Data Collection – SpaceX API

1

- Make a GET response to the SpaceX REST API
- Convert the response to a .json file then to a Pandas DataFrame

2

- Use custom logic to clean the data (Appendix)
- Define lists for data to be stored in
- Call Custom Functions to retrieve data and fill the lists (Appendix)
- Use the lists as values in a dictionary and construct the dataset

3

- Create a Pandas DataFrame from the dictionary dataset

4

- Filter the DataFrame to only include Falcon 9 launches
- Replace missing values of PayloadMass with the mean PayloadMass value

1

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

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

```
# Use json_normalize method to convert the json result into a dataframe
data = pd.json_normalize(response.json())
```

2

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

```
# Call getBoosterVersion
getBoosterVersion(data)
```

```
# Call getLaunchSite
getLaunchSite(data)
```

```
# Call getPayloadData
getPayloadData(data)
```

```
# Call getCoreData
getCoreData(data)
```

```
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}
```

3

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

4

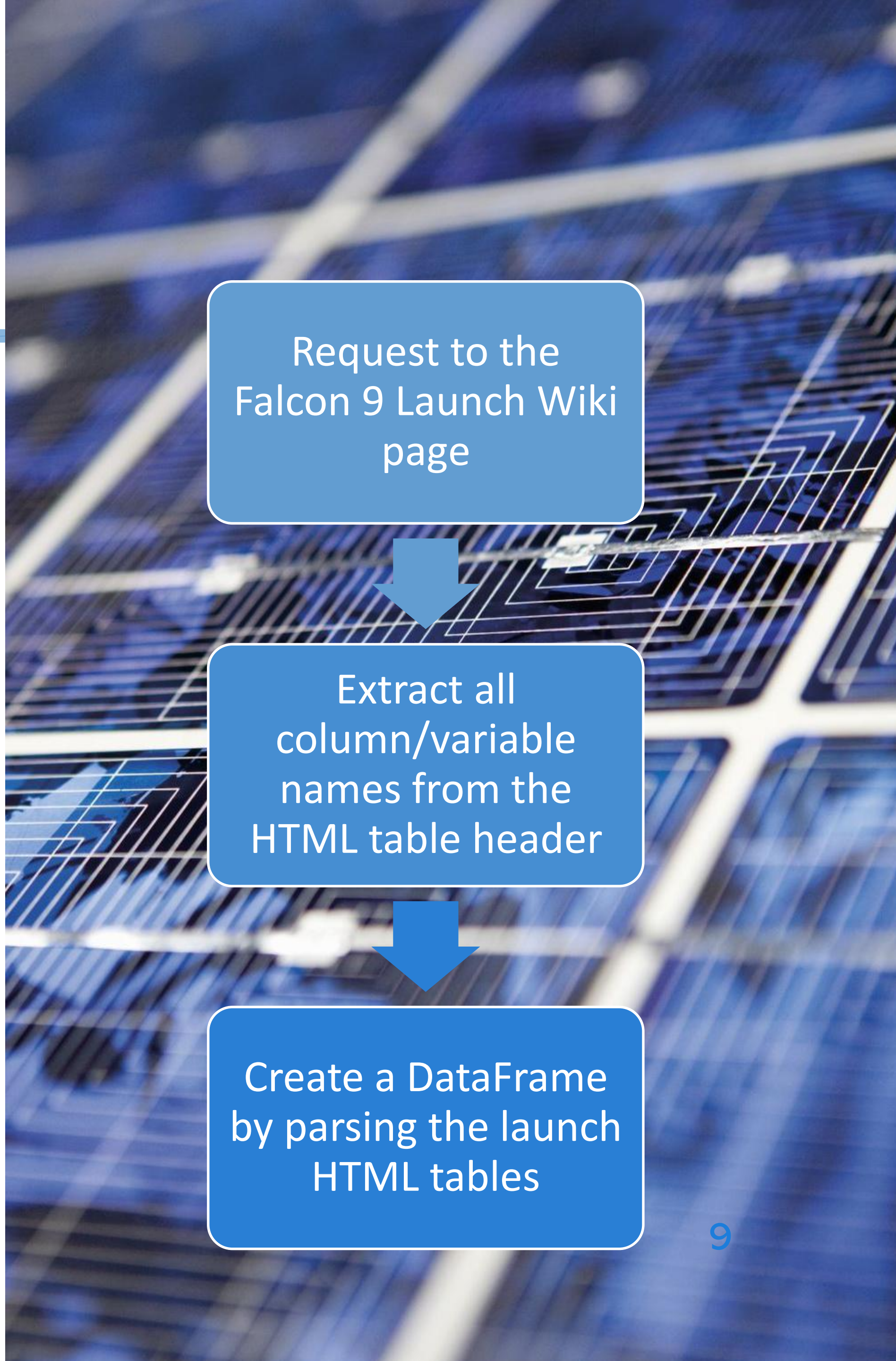
```
data_falcon9 = df[df['BoosterVersion']!= 'Falcon 1']
```

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

```
# 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)
```

Data Collection – Web Scraping

- Web Scraping to collect Falcon 9 historical launch records from Wikipedia page: "List of Falcon 9 and Falcon Heavy launches"
- GitHub URL to Notebook: Web Scraping



Data Collection – Web Scraping

1
- Request the HTML page from the URL
- Assign the response to an object

2
- Create a BeautifulSoup object from a response text content
- Find all tables within the HTML page

3
- Extract all column header names from the tables found within the HTML page

4
- Use the column names as keys in dictionary
- Use custom functions and logic to parse all launch tables to fill the dictionary values (Appendix)

5
- Convert the dictionary to a Pandas DataFrame ready to export

1

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

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

2

```python
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(data, 'html5lib')
```

```python
# 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')
```

3

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

4

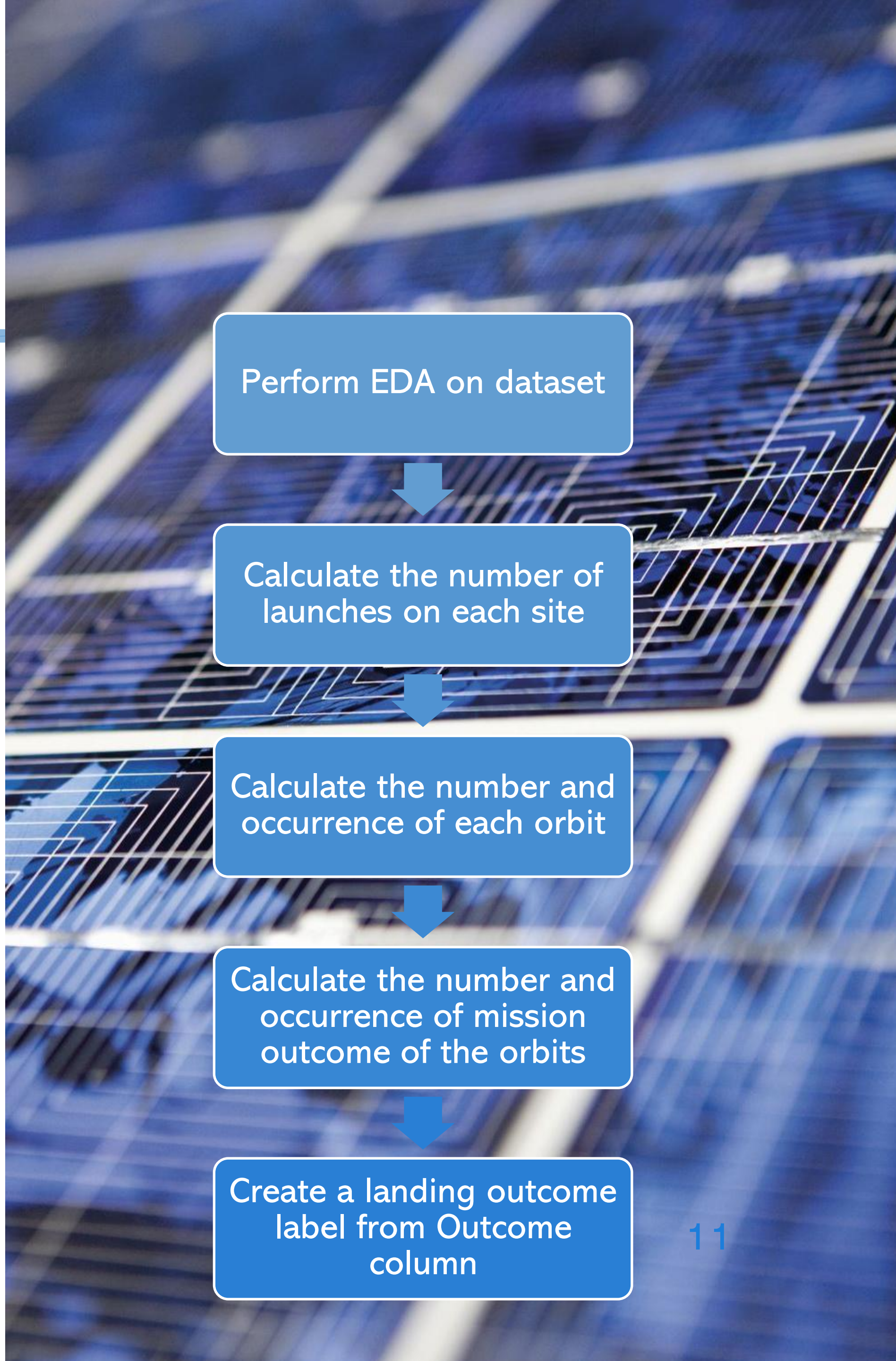
```python
launch_dict= dict.fromkeys(column_names)

# 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']=[]
```

5

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

Data Wrangling

- Perform Exploratory Data Analysis (EDA) to find some patterns in the data and determine what would be the label for training supervised models.
- Convert the outcomes into Training Labels with 1 means the booster successfully landed, and 0 means it was unsuccessful.
- The information was extracted from a Public API https://api.spacexdata.com/v4/launches/past.
- GitHub URL to Notebook: Data Wrangling



Data Wrangling

- The SpaceX dataset contains several SpaceX launch facilities. The location of each Launch is placed in the column LaunchSite.
- Each launch aims to a dedicated orbit. The orbit type is in the Orbit column.
- Initial Exploratory Data Analysis: Using the .value_counts() method to determine the following:

1 — Data Exploration: Calculate the number of launches on each site

2 — Data Exploration: Calculate the number and occurrence of each orbit

3 — Data Exploration: Calculate the number and occurrence of landing outcome per orbit type

1

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

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

2

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

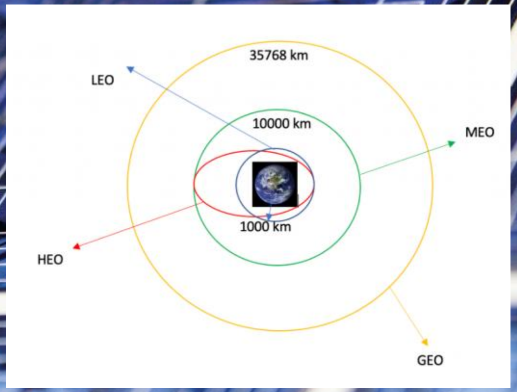
```
Orbit
GTO      27
ISS      21
VLEO     14
PO        9
LEO       7
SSO       5
MEO       3
HEO       1
ES-L1     1
SO        1
GEO       1
Name: count, dtype: int64
```

3

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

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

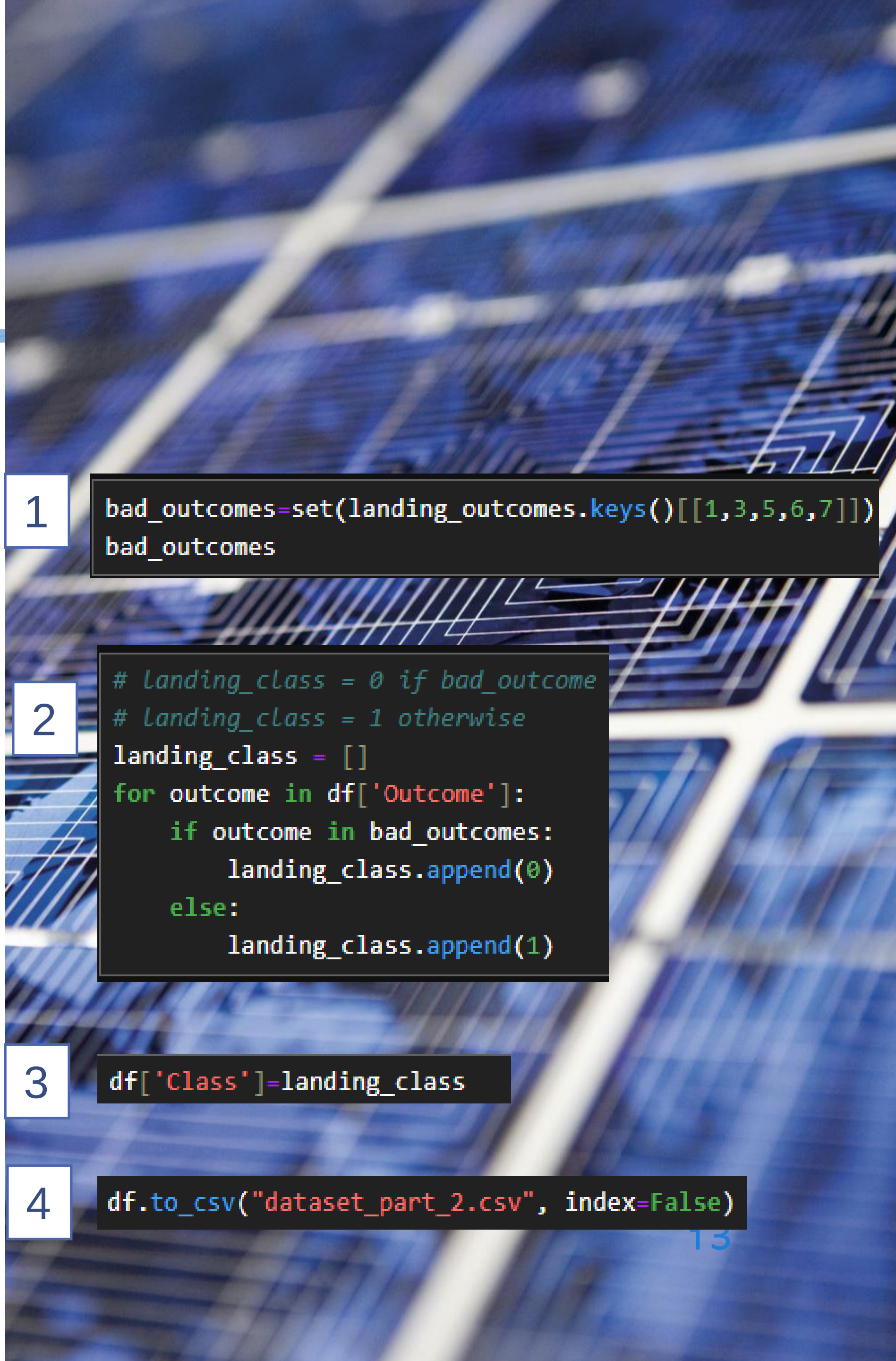
Orbits



Data Wrangling

- To determine Training labels:

1. Identify Unsuccessful Landings: Define bad outcomes
2. Create Landing Outcome Labels: Assign 0 for unsuccessful, 1 for successful
3. Determine Success Rate
4. Export Processed Data: Export to csv



1

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

2

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

3

```
df['Class']=landing_class
```

4

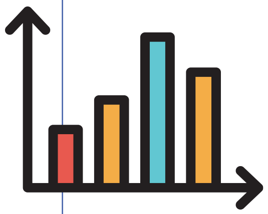
```
df.to_csv("dataset_part_2.csv", index=False)
```

EDA with Data Visualization

Scatter Charts

To visualize the relationship between:

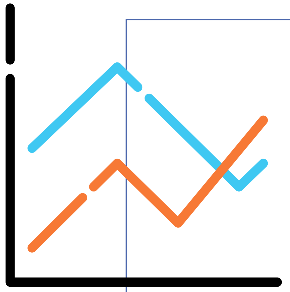
- Flight Number and Launch Site
- Payload Mass and Launch Site
- Flight Number and Orbit type
- Payload Mass and Orbit type



Bar Charts

To visualize the relationship between:

- Success rate and Orbit type



Line Charts

To visualize the relationship between:

- Launch success rate and Year

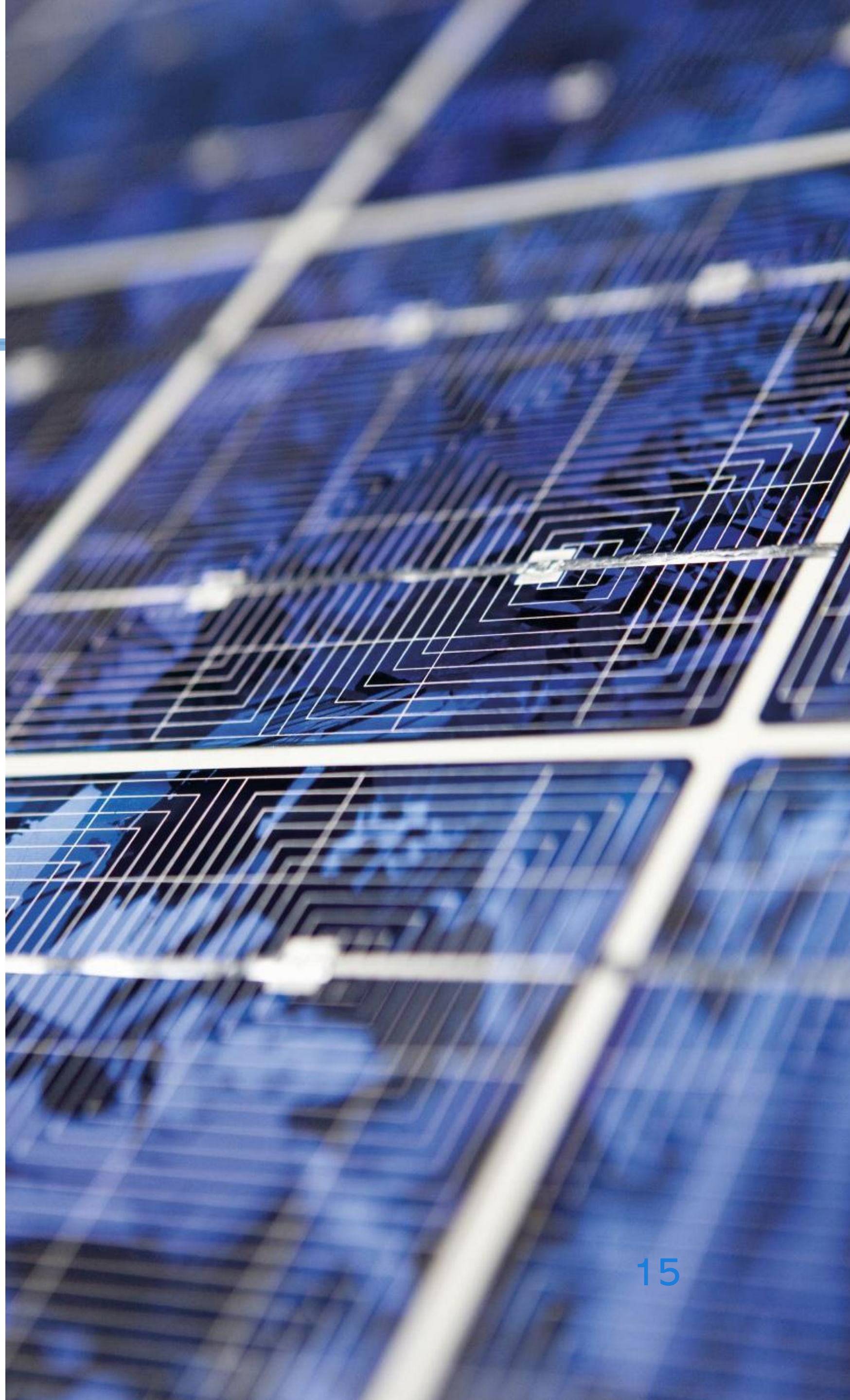
[GitHub URL to Notebook: EDA with Data Visualization]

EDA with SQL

A summary of queries performed:

- 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
- 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

[GitHub URL to Notebook: EDA with SQL]



Build an Interactive Map with Folium

- To visualize the launch data on an interactive map, the following steps were taken:

1. Mark all launch sites on a map:

- Initialize the map using a Folium Map object
- Add a folium.Circle and folium.Marker for each launch site on the launch map

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

- Assign a marker color of successful Class = 1 as green, and failed Class 0 as red.
- Marker cluster helps to simplify a map containing many markers having the same coordinate.
- Add a folium.Marker to marker_cluster for each launch result, to put the launches into clusters.
- Assign the icon_color as the marker_color determined previously.

3. Calculate the distances between a launch site to its proximities

- To explore and analyze the proximities of launch sites, add a MousePosition on the map. The distances between points can be calculated using the Lat and Long values.
- Create a folium.Marker object to show the distance.
- Display the distance between two points by drawing a folium.PolyLine

GitHub URL to Notebook: Interactive Map with Folium

Build a Dashboard with Plotly Dash

- Functions from Dash are used to generate an interactive site where we can toggle the input using a dropdown menu and a range slider
- To visualize the launch data on an interactive Data Dashboard, the following plots were added to a Plotly Dashboard:

1. Pie Chart (px.pie()) showing the Total Successful launches per site:

- To visualize which sites are most successful.
- To visualize the success/failure ratio for an individual site, add a dcc.Dropdown() object

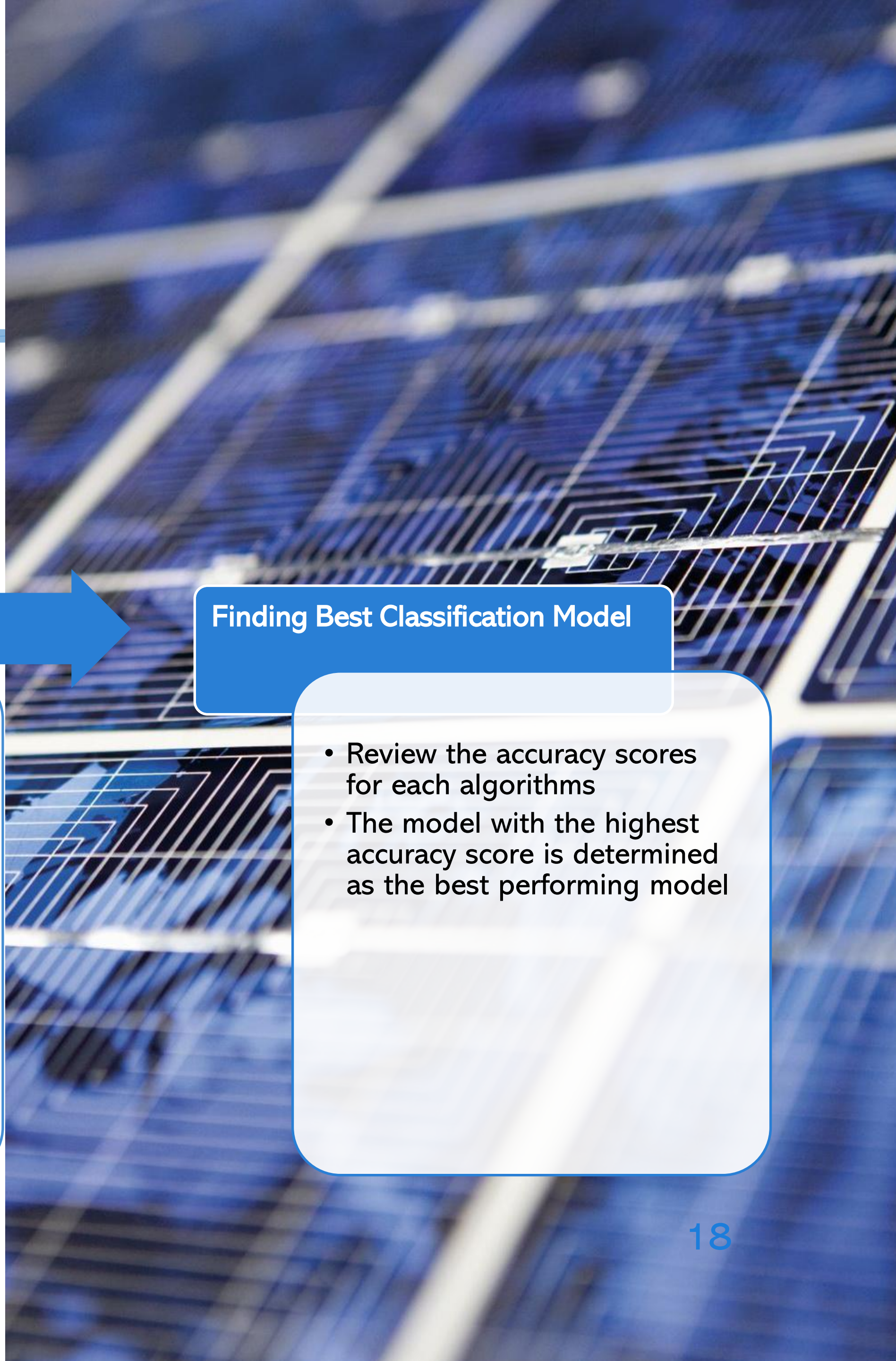
2. Scatter graph (px.scatter()) to show the correlation between Outcome (success or fail) and Payload mass (Kg) for the different Booster versions

- Use a RangeSlider() object to filter by ranges of payload mass.
- Filter by booster version.

[GitHub URL to Notebook: Dashboard with Plotly Dash]()

Predictive Analysis (Classification)

- Summary of steps taken to develop, evaluate, and find the best performing Classification model:



Model Development

- To prepare the dataset for Model Development:
  - Load the SpaceX launch dataset
  - Perform data transformation: standardize the data
  - Split data into training and test dataset using train_test_split()
  - Decide which type of Machine Learning Algorithm is the most appropriate

Model Evaluation

- Fos each Algorithm:
  - Using the output GridSearchCV object:
    - Check the tuned hyperparameters (best_params)
    - Check the accuracy (score and best_score)
  - Plot and examine the Confusion Matrix

Finding Best Classification Model

- Review the accuracy scores for each algorithms
- The model with the highest accuracy score is determined as the best performing model

GitHub URL to Notebook: Predictive Analysis

Results

Exploratory data analysis results

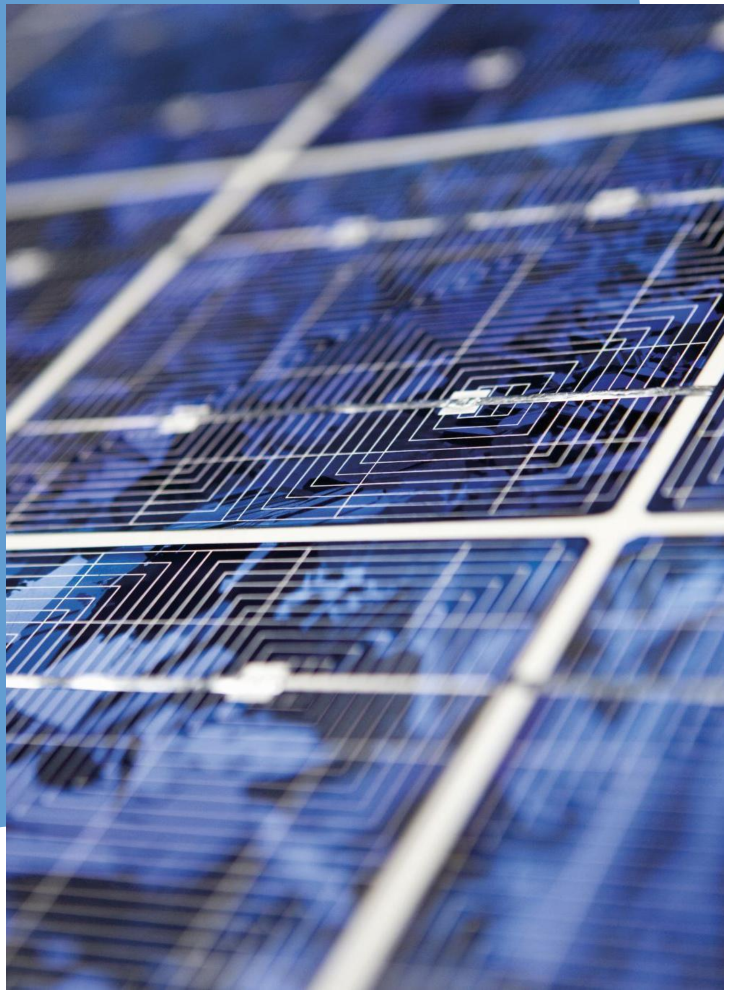
Interactive analytics demo in screenshots

Predictive analysis results

Section 2

Insights drawn from EDA

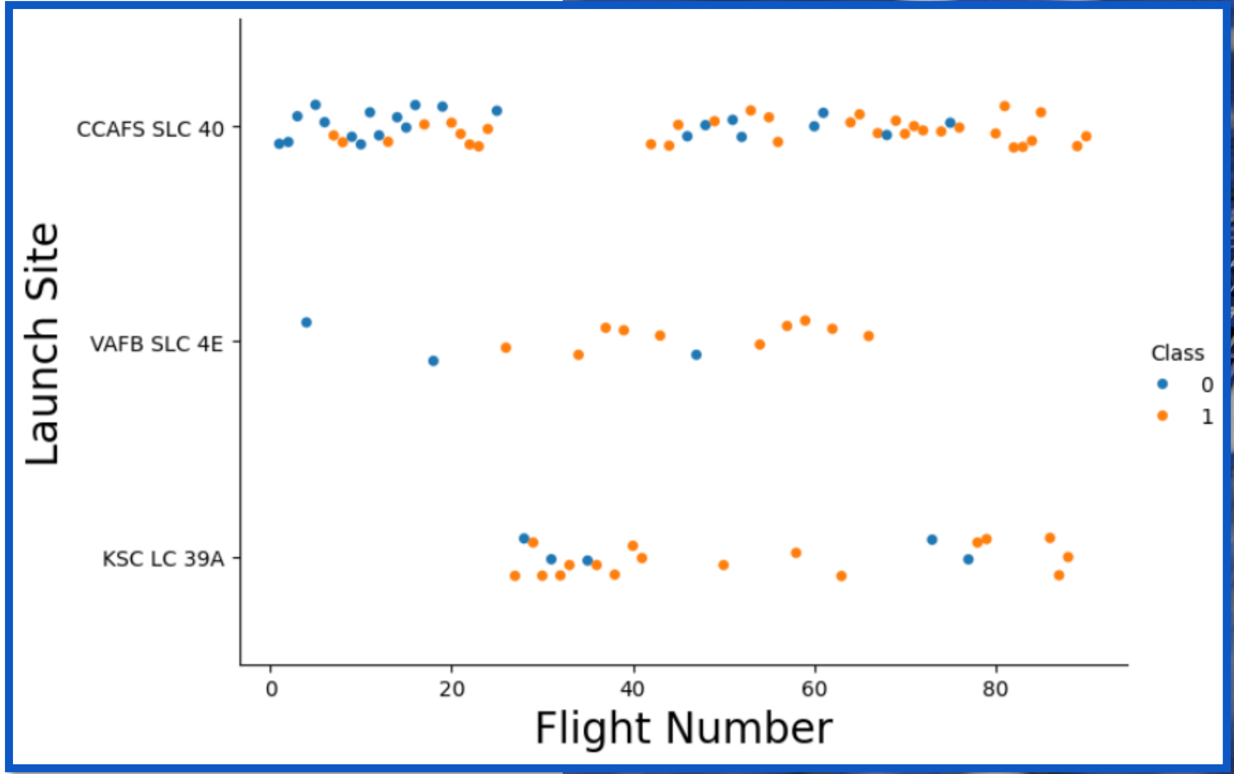
EDA With Visualization Results



Flight Number vs. Launch Site

The Launch Site vs Flight Number scatter plot suggests that:

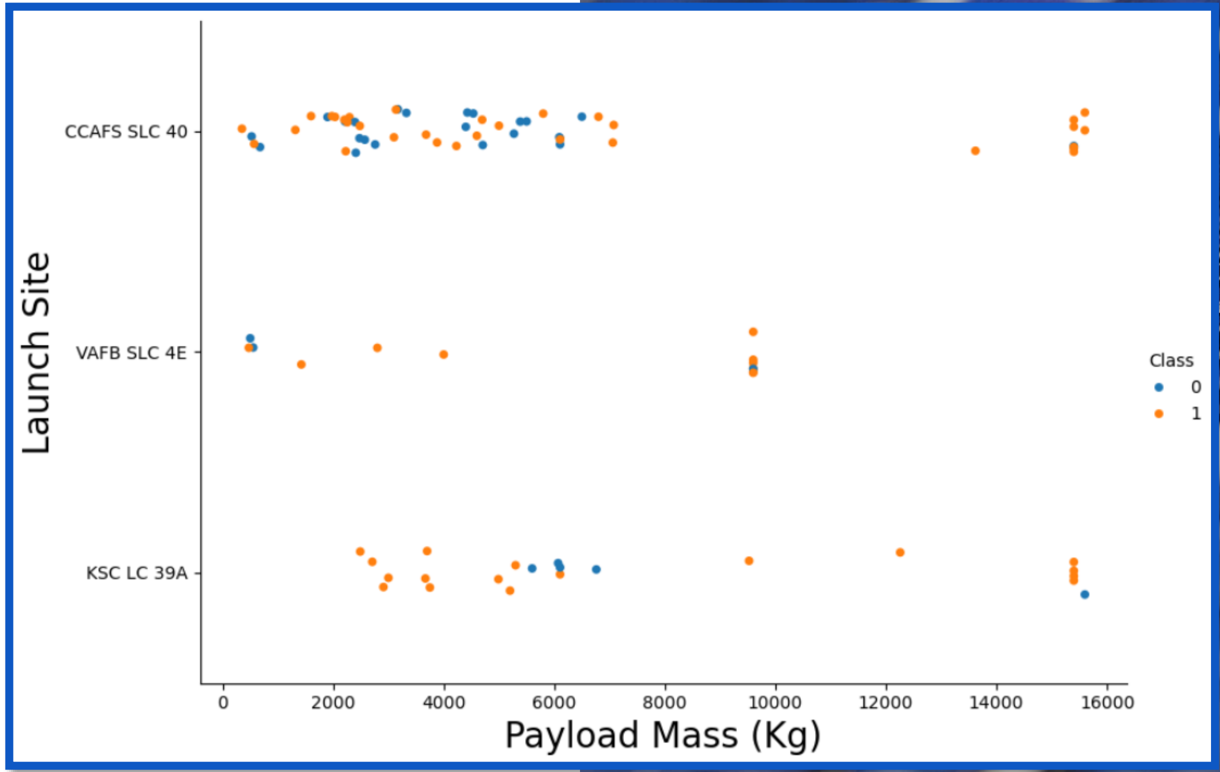
- The success rate in each launch site increases as the number of flights increases.
- Most early flights were unsuccessful and were launched from site CCAFS SLC 40.
- There are no early flights launched from site KSC LC 39A.



Payload Mass vs. Launch Site

The Payload Mass vs. Launch Site scatter plot suggests that:

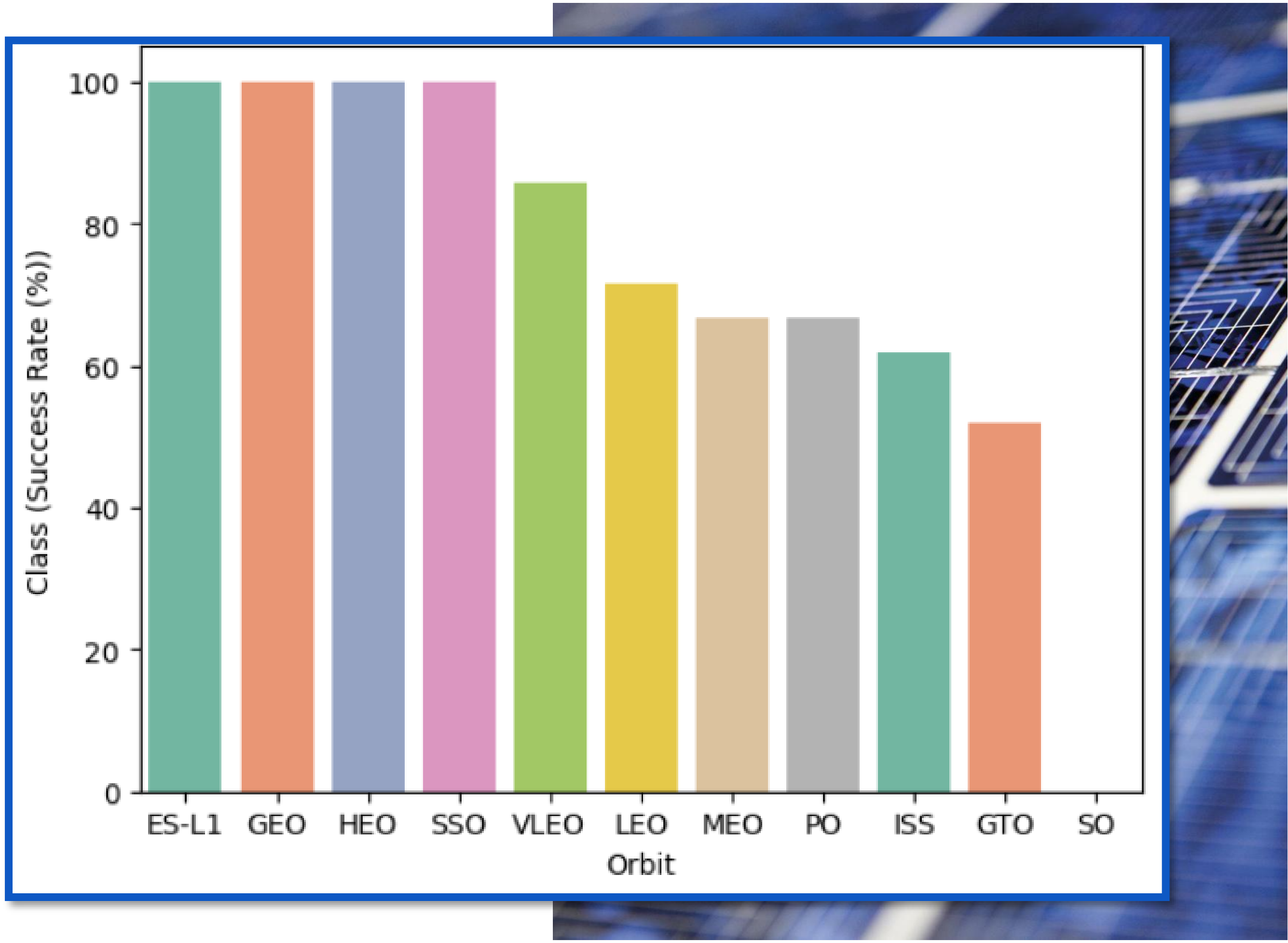
- There are a limited number of unsuccessful landings for Payload Mass above 7000 Kg.
- There are no rockets launched in VAFB-SLC launch site for heavy Payload Mass (above10000 Kg)
- The success rate is higher as the Payload Mass (Kg) increases for CCAFS SLC 40 launch site.



Success Rate vs. Orbit Type

The Success Rate vs. Orbit Type bar chart suggests that:

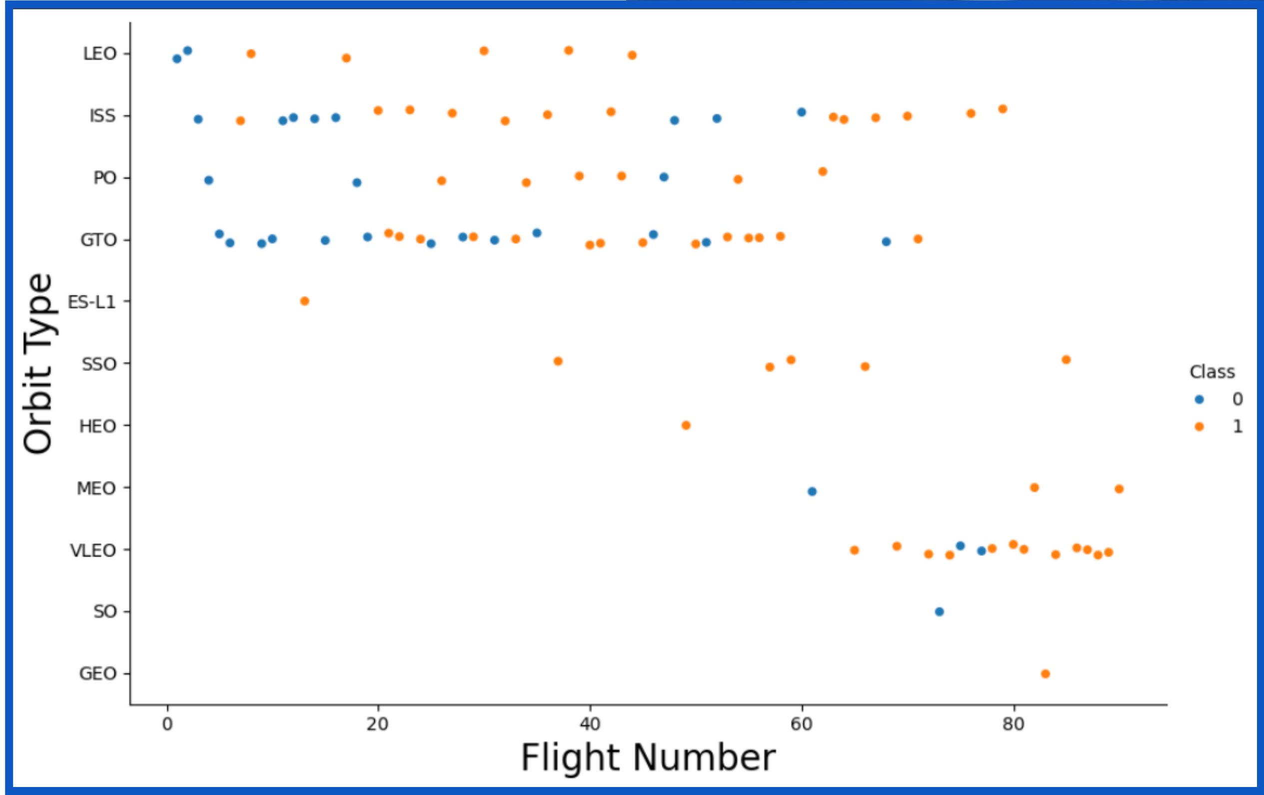
- The orbits ES-L1, GEO, HEO, and SSO have the highest success rate (100%).
- The orbit GTO has the lowest success rate (~50%).
- The orbit SO has 0% success rate.



Flight Number vs. Orbit Type

The Flight Number vs. Orbit Type scatter plot suggests that:

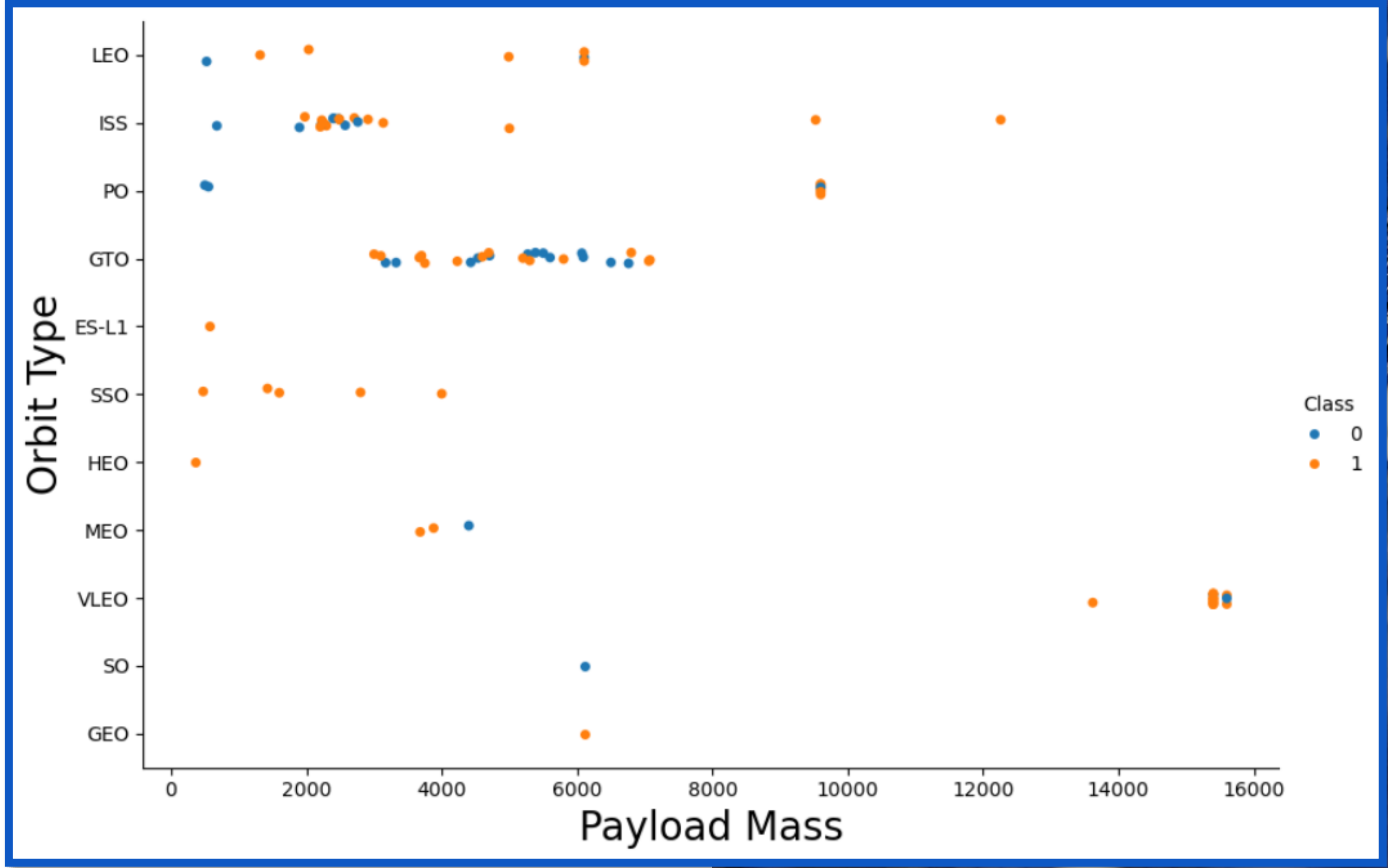
- The success rate increases as the flight number increases.
- The orbits ES-L1, HEO, and GEO have only 1 successful flight in their respective orbits, which can explain the highest success rate.
- The orbit SSO has 5 successful flights.
- The orbit LEO has most unsuccessful landings in the early launches.



Payload Mass vs. Orbit Type

The Payload Mass vs. Orbit Type scatter plot suggests that:

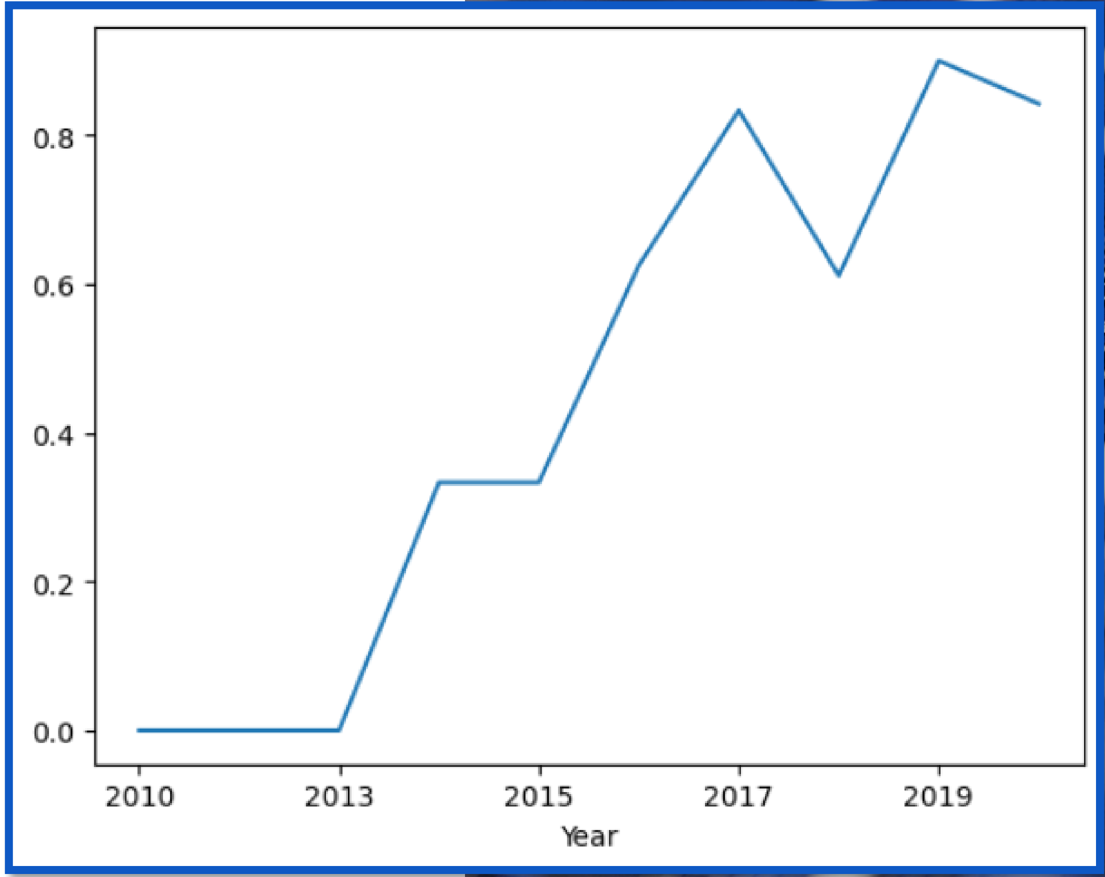
- The orbits ISS, and VLEO have more success with heavy Payload Mass (above 10000 Kg).
- The orbit SSO has the more success with low Payload Mass (below 6000 Kg).



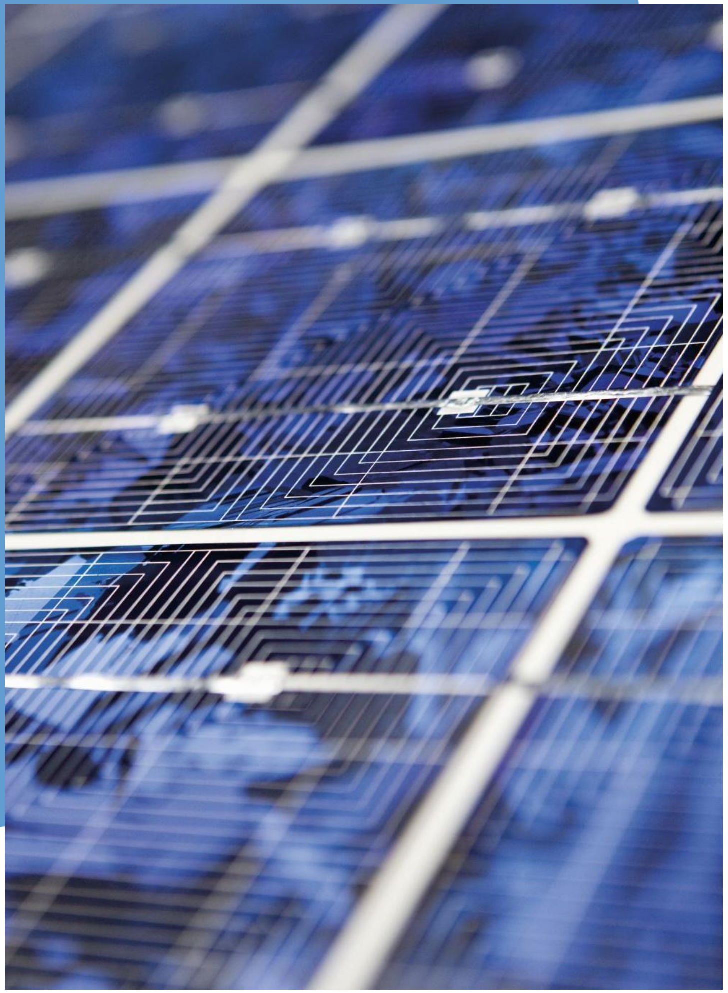
Launch Success Yearly Trend

The line chart of yearly launch success rate suggests that:

- All landings were unsuccessful between 2010 and 2013.
- The success rate generally increased after 2013, despite minor declines in 2018 and 2020.
- The most success rate were in 2017.



EDA
With SQL results



All Launch Site Names

Find the names of the unique launch sites:

SQL Query:

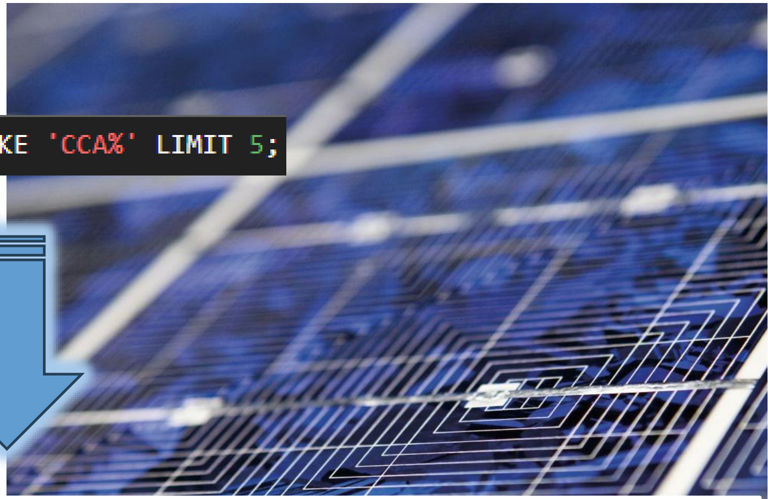
```
%sql SELECT DISTINCT Launch_Site FROM SPACEXTABLE;
```

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

The word DISTINCT in the query shows only the unique values from the Launch_Site column of the SPACEXTABLE.

Launch Site Names Begin with 'CCA'

Find 5 records where launch sites begin with `CCA`



SQL Query:

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

The word LIMIT 5 in the query shows only 5 records, and the LIKE keyword is used with ‘CCA%’ to retrieve values beginning with ‘CCA’.

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

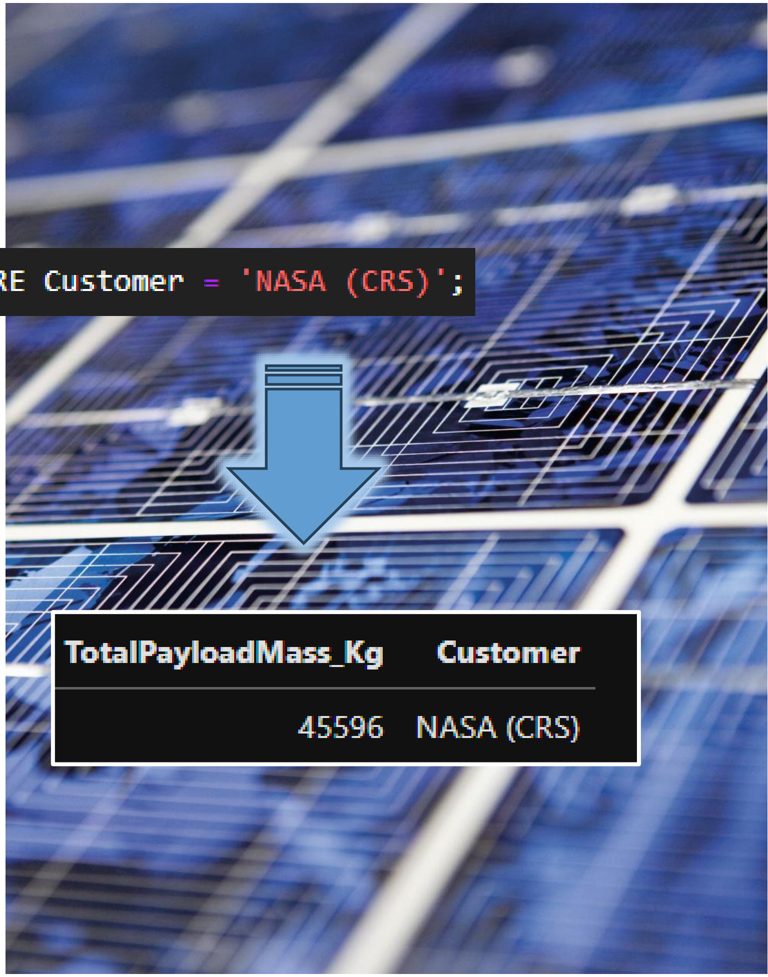
Total Payload Mass

Calculate the total payload carried by boosters from NASA:

SQL Query:

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

The function SUM is used to calculate the total of the Payload Mass (Kg) column, and the WHERE clause filters the dataset to only perform calculations on Customer NASA (CRS).



TotalPayloadMass_Kg	Customer
45596	NASA (CRS)

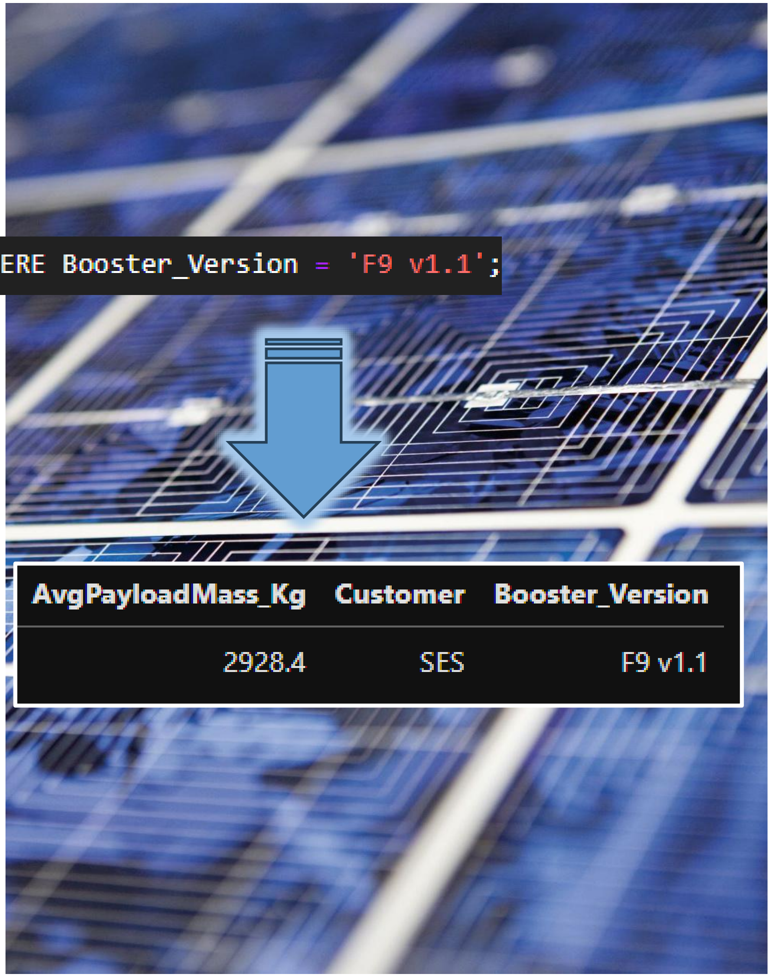
Average Payload Mass by F9 v1.1

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

SQL Query:

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

The function AVG is used to calculate the average of the Payload Mass (Kg) column, and the WHERE clause filters the dataset to only perform calculations on Booster_Version F9 v1.1



AvgPayloadMass_Kg	Customer	Booster_Version
2928.4	SES	F9 v1.1

First Successful Ground Landing Date

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

SQL Query:

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

The function MIN shows the minimum date in the column Date, and the WHERE clause filters the results to only the successful ground pad landings.

First_succsessful_ground_landing
2015-12-22

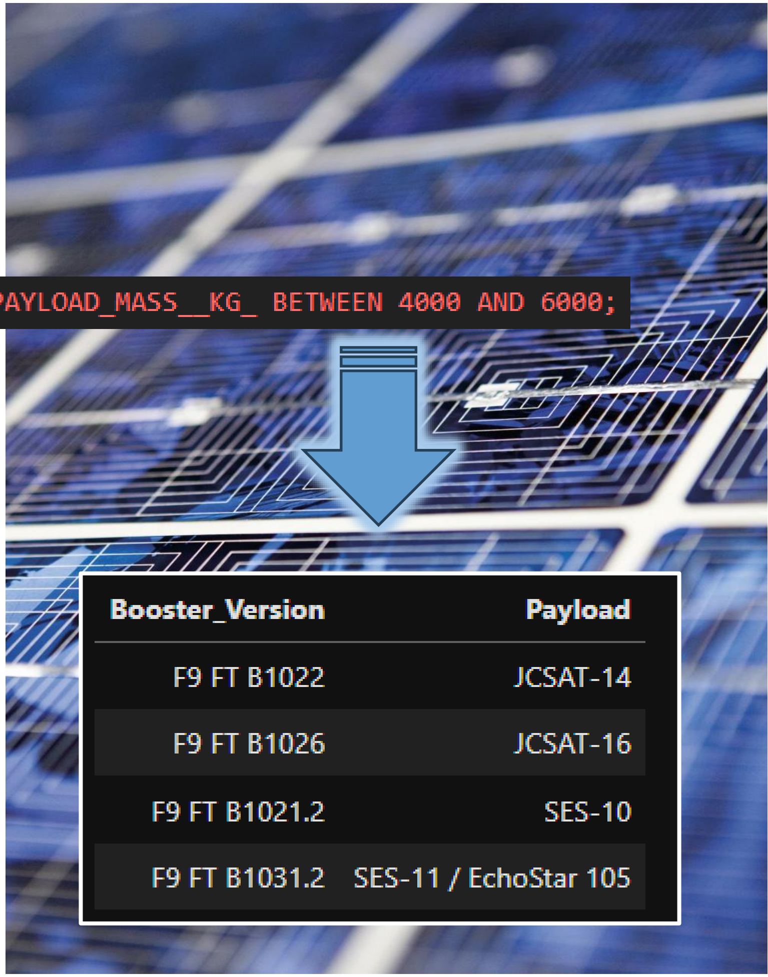
Successful Drone Ship Landing with Payload between 4000 and 6000

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

SQL Query:

```
%sql SELECT Booster_Version, Payload FROM SPACEXTABLE WHERE Landing_Outcome = 'Success (drone ship)' AND PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000;
```

The WHERE clause is used to filter the results to Landing_outcome = Success (drone ship), the AND clause specifies additional filter condition, and the BETWEEN keyword allows to select values 4000 < x < 6000



Booster_Version	Payload
F9 FT B1022	JCSAT-14
F9 FT B1026	JCSAT-16
F9 FT B1021.2	SES-10
F9 FT B1031.2	SES-11 / EchoStar 105

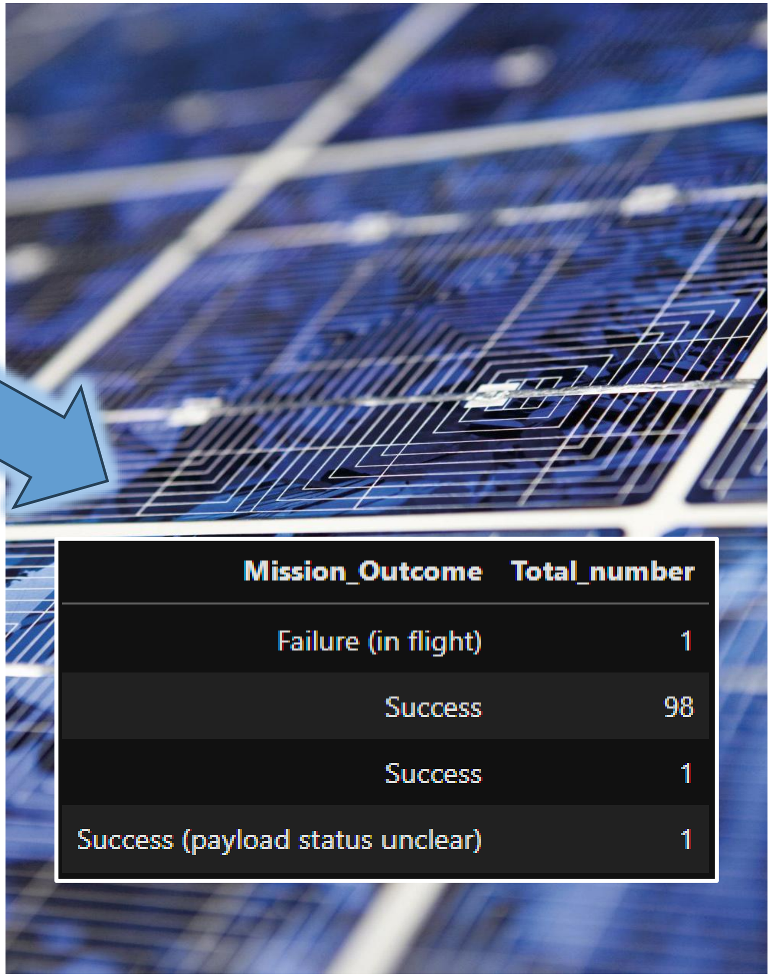
Total Number of Successful and Failure Mission Outcomes

Calculate the total number of successful and failure mission outcomes

SQL Query:

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

The COUNT keyword is used to calculate the total number of mission outcomes, and the GROUP BY keyword is used to group the results by the mission outcome.



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

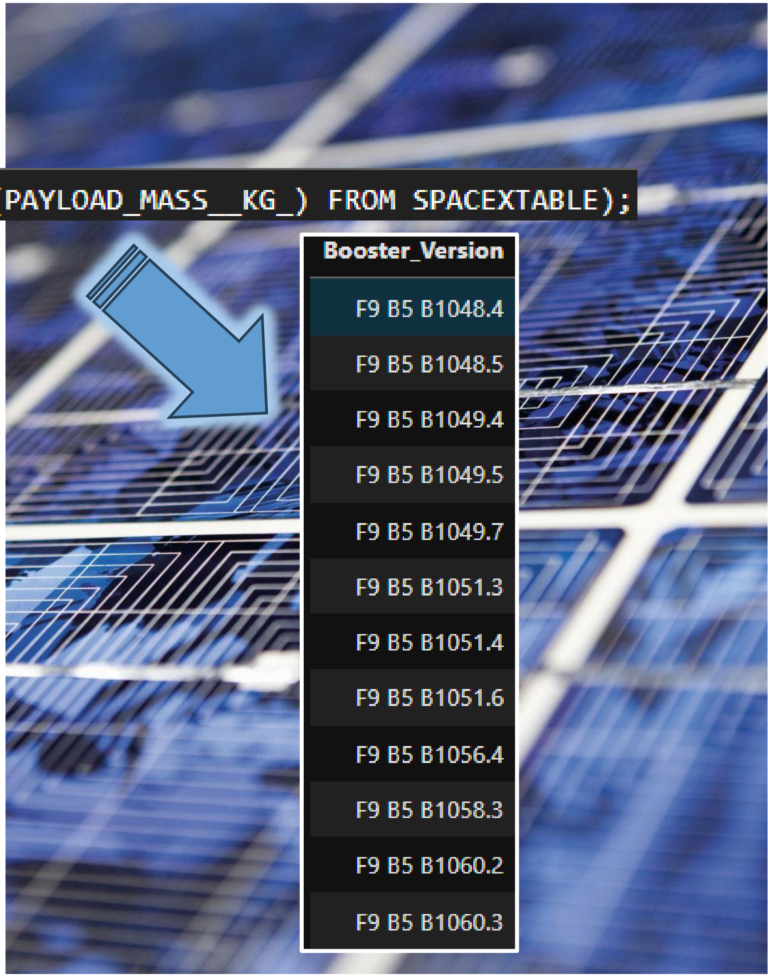
Boosters Carried Maximum Payload

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

SQL Query:

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

The word DISTINCT in the query shows only the unique values from the Launch_Site column of the SPACEXTABLE.



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

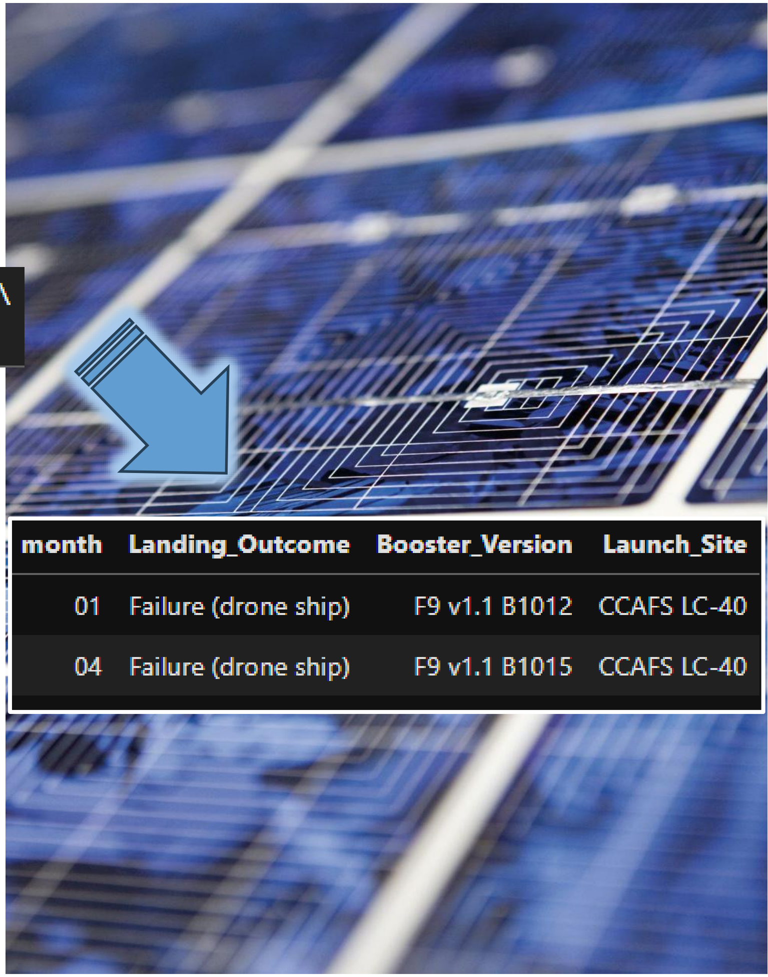
2015 Launch Records

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

SQL Query:

```
%sql SELECT substr(Date,6,2) as month, Landing_Outcome, Booster_Version, Launch_Site from SPACEXTABLE \
where Landing_Outcome = 'Failure (drone ship)' and substr(Date,0,5) = '2015';
```

The substr() keyword is used in the select statement to get the month and year from the date column where substr(Date, 0, 5)='2015' for year, and Landing_outcome was 'Failure (drone ship)'.



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

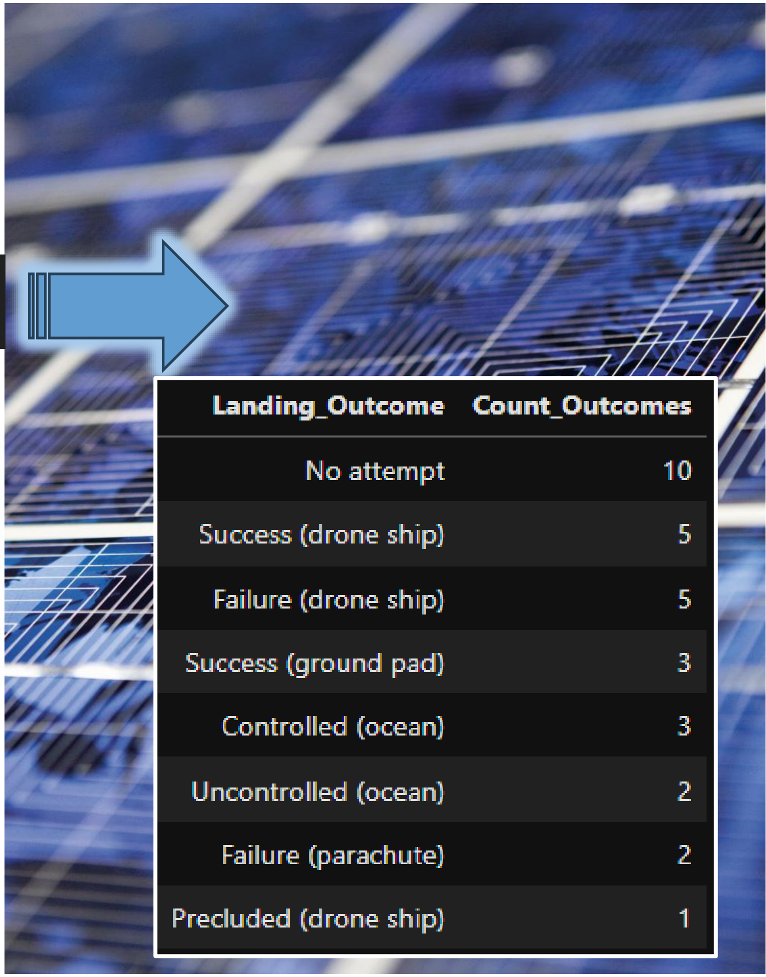
Rank Landing Outcomes Between 2010-06-04 and 2017-03-20

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_Outcomes FROM SPACEXTABLE \
WHERE Date BETWEEN '2010-06-04' AND '2017-03-20' GROUP BY Landing_Outcome ORDER BY Count_Outcomes DESC;
```

The WHERE keyword is used with BETWEEN keyword to filter the results to dates specified. The results are group and ordered using the keywords GROUP BY and ORDER BY, where DESC is used to specify the descending order.

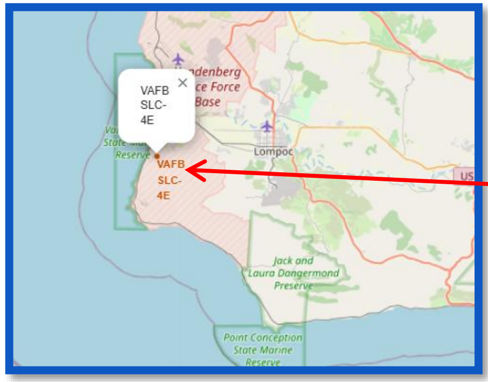


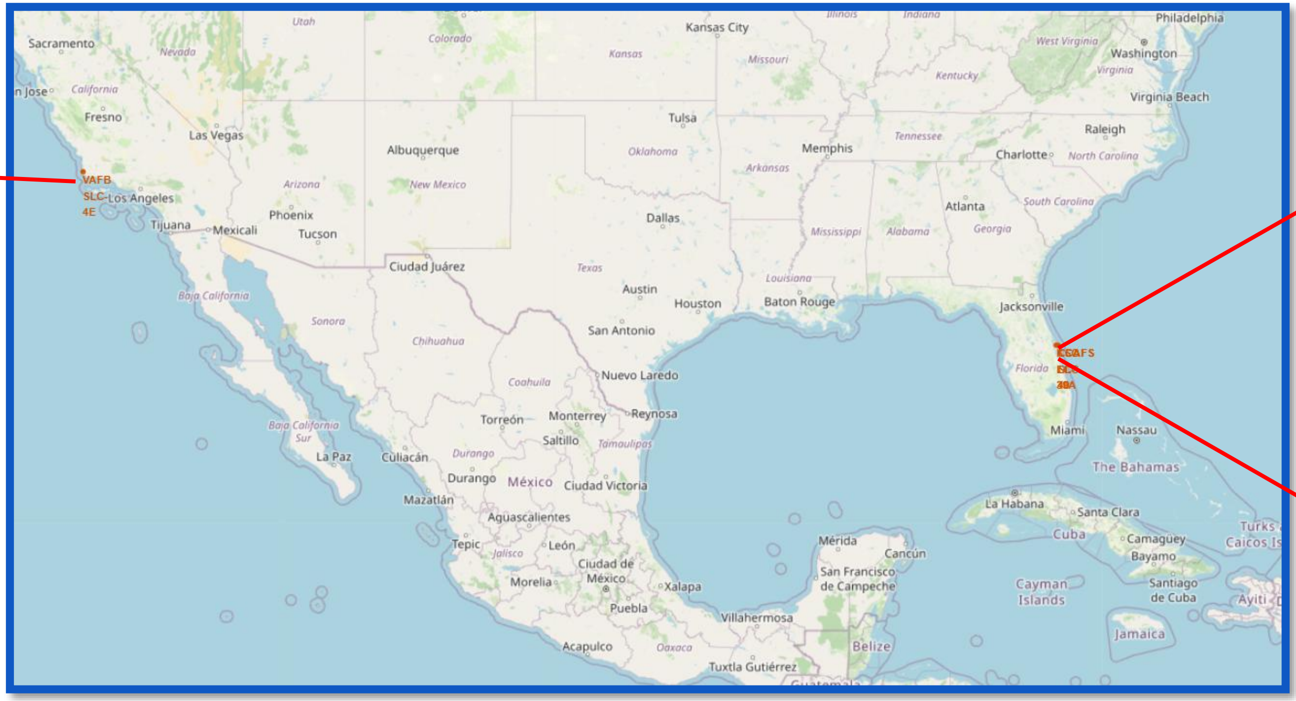
Landing_Outcome	Count_Outcomes
No attempt	10
Success (drone ship)	5
Failure (drone ship)	5
Success (ground pad)	3
Controlled (ocean)	3
Uncontrolled (ocean)	2
Failure (parachute)	2
Precluded (drone ship)	1

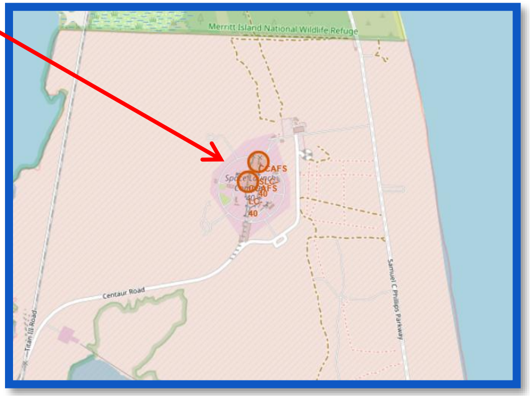
Section 3

Launch Sites Proximities Analysis

Launch sites on a Map







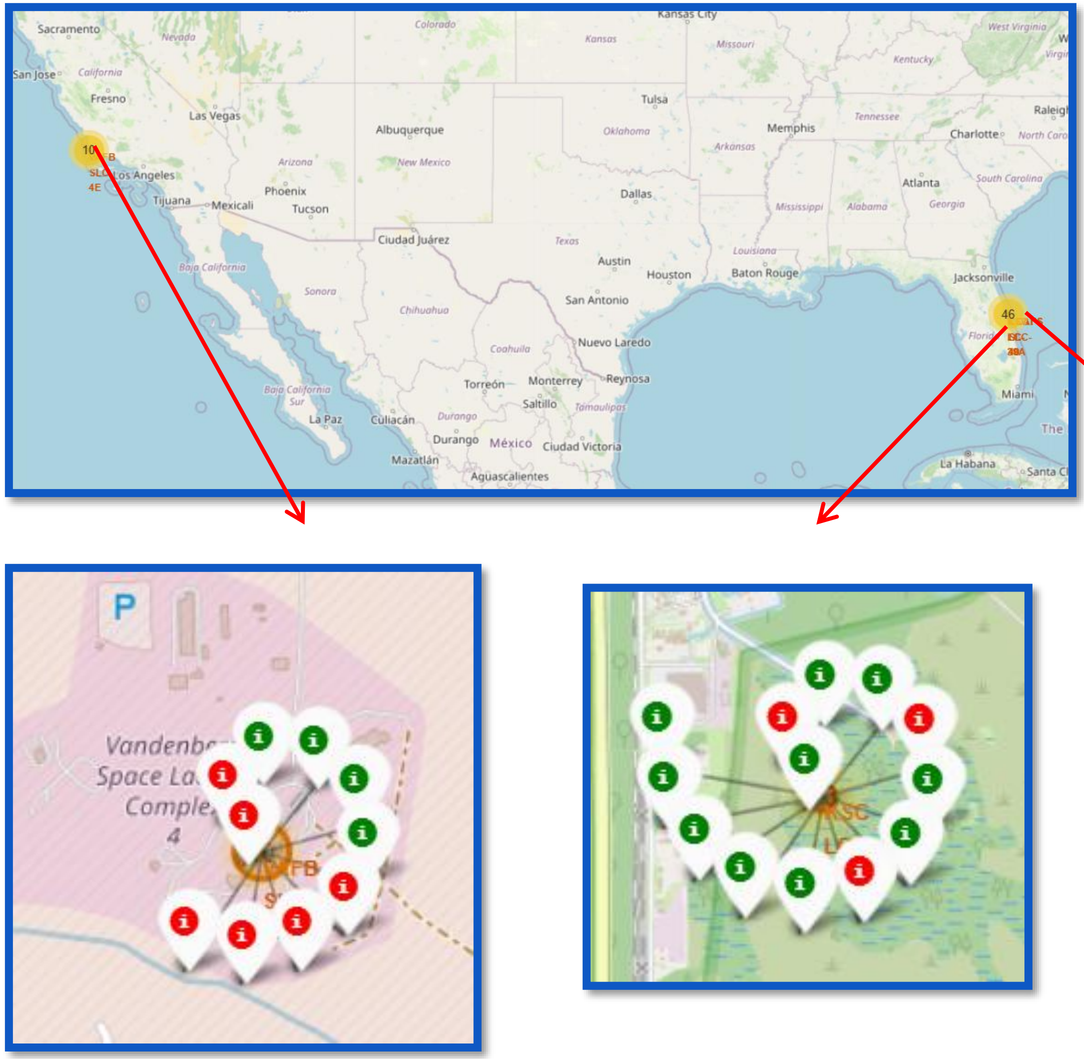
All SpaceX launches sites are located on the coast of the United States of America, in proximity to the Equator.

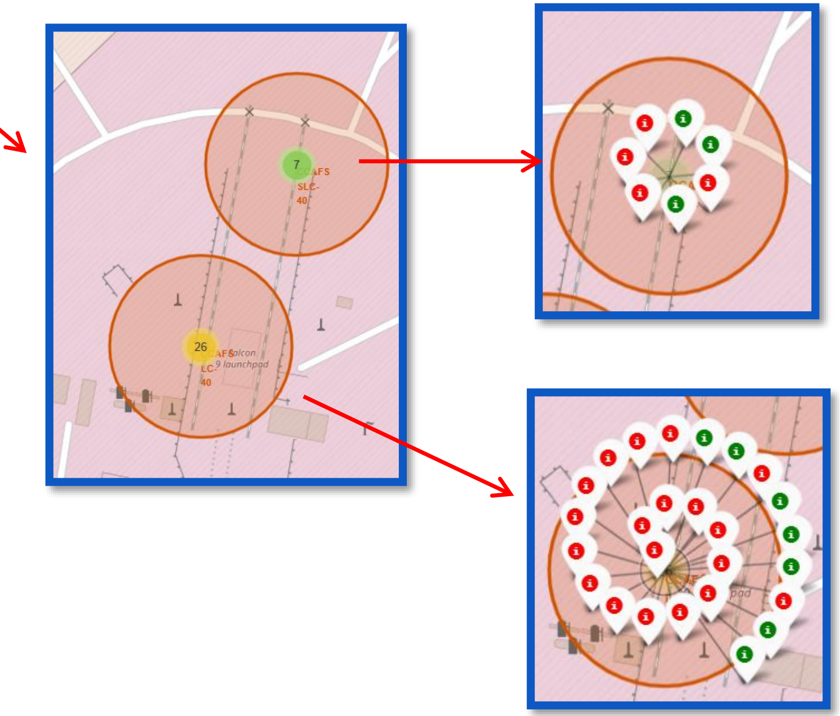
Launch sites on a Map

Launches are grouped into clusters.

<u>Outcomes:</u>

- **Green** markers for successful launches
- **Red** markers for unsuccessful launches





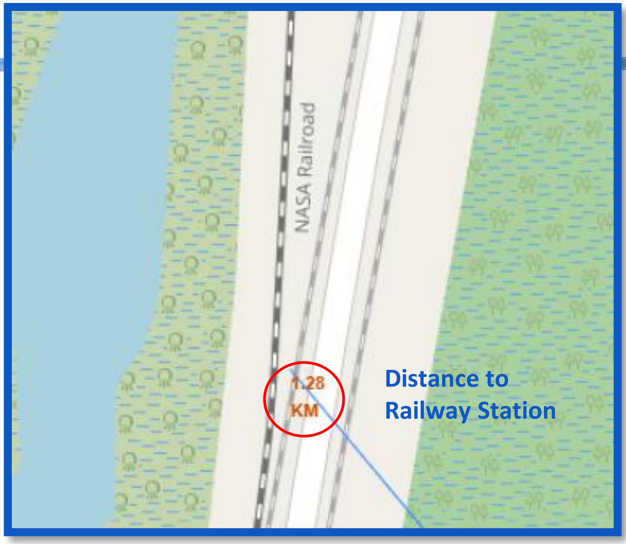
CCAFS SLC-40

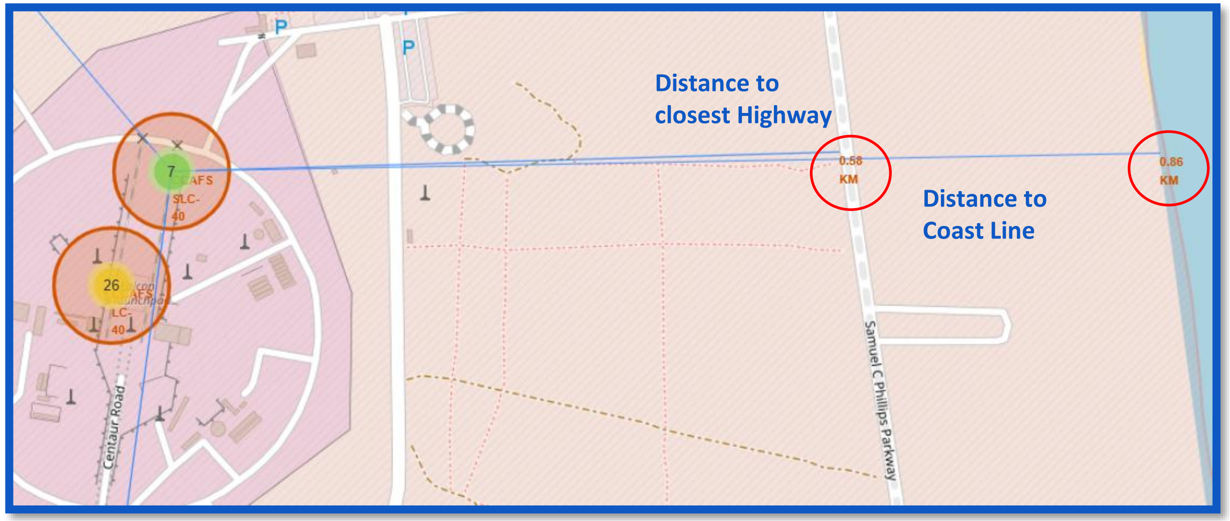
CCAFS LC-40

Distances between a launch site to its proximities

Using the Launch Site CCAFS SLC-40 as a reference, the distance:

- To the closest to highway is 0.58 Km
- To the Coast Line is 0.86 Km
- To the Railway Station is 1.28 Km
- To the closest City 51.43 Km

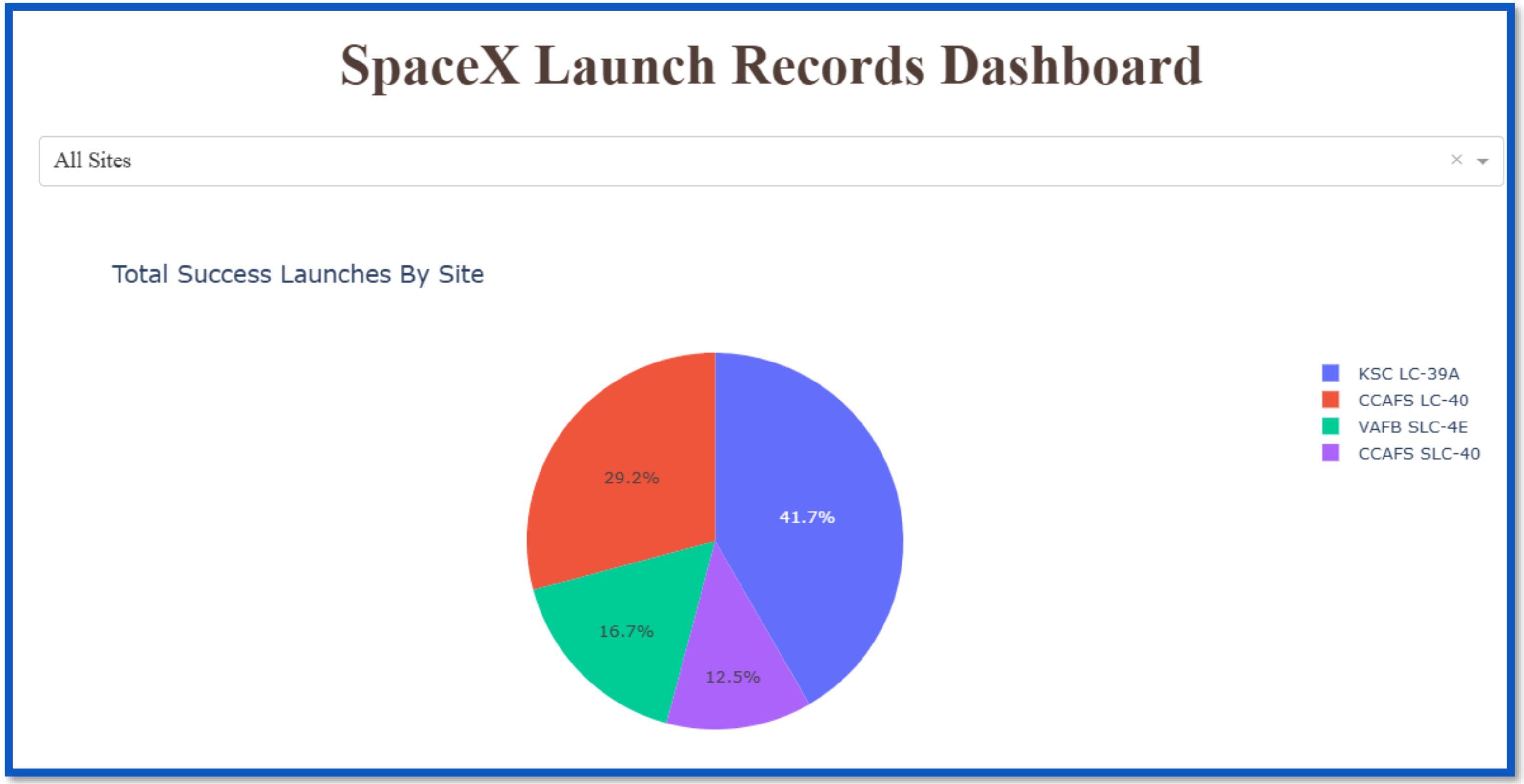




Section 4

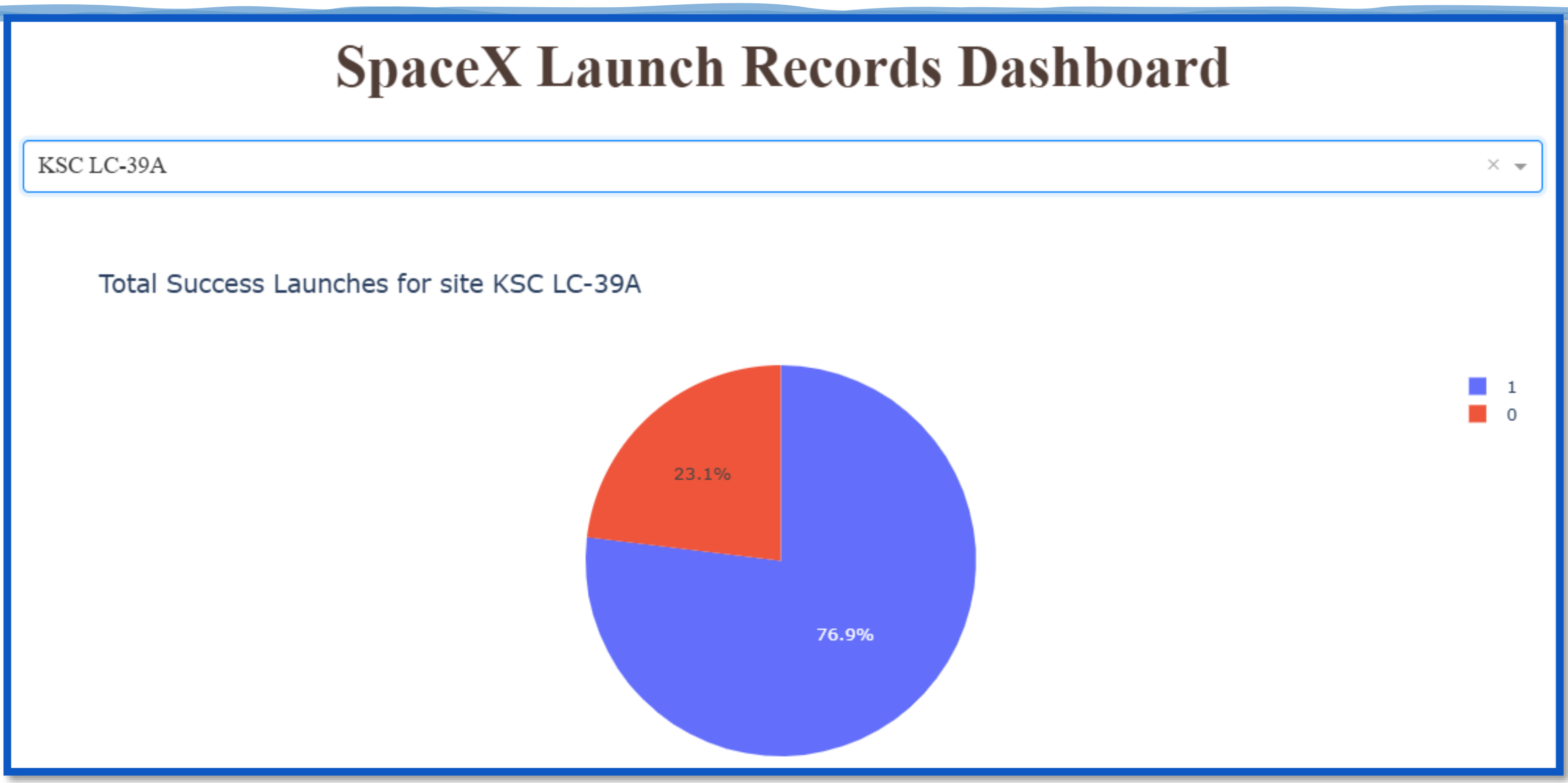
Build a Dashboard with Plotly Dash

Launch success by Site



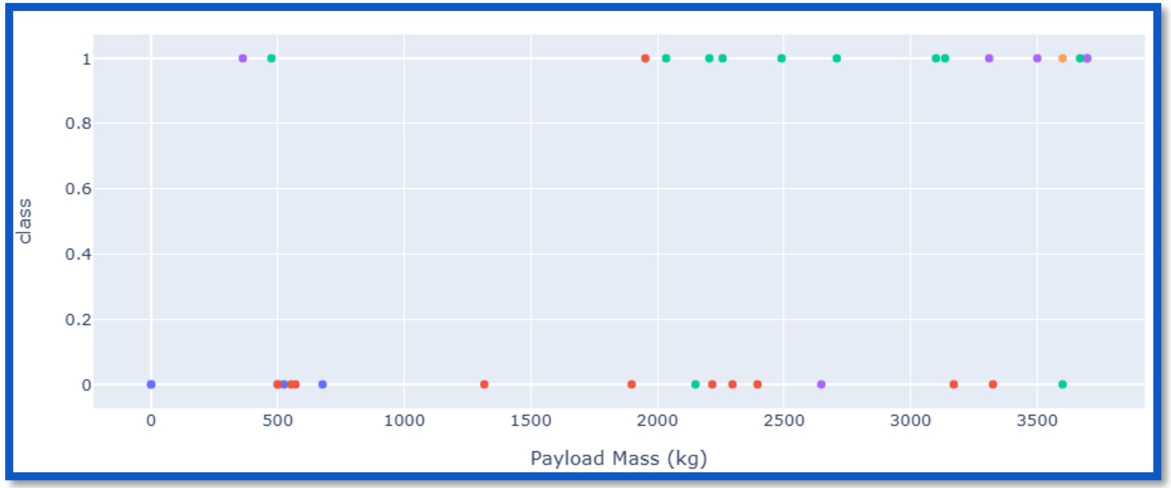
The Launch Site KSC LC-39A had the most successful launches from all sites, with 41.7% of total successful launches.

Launch success by Site

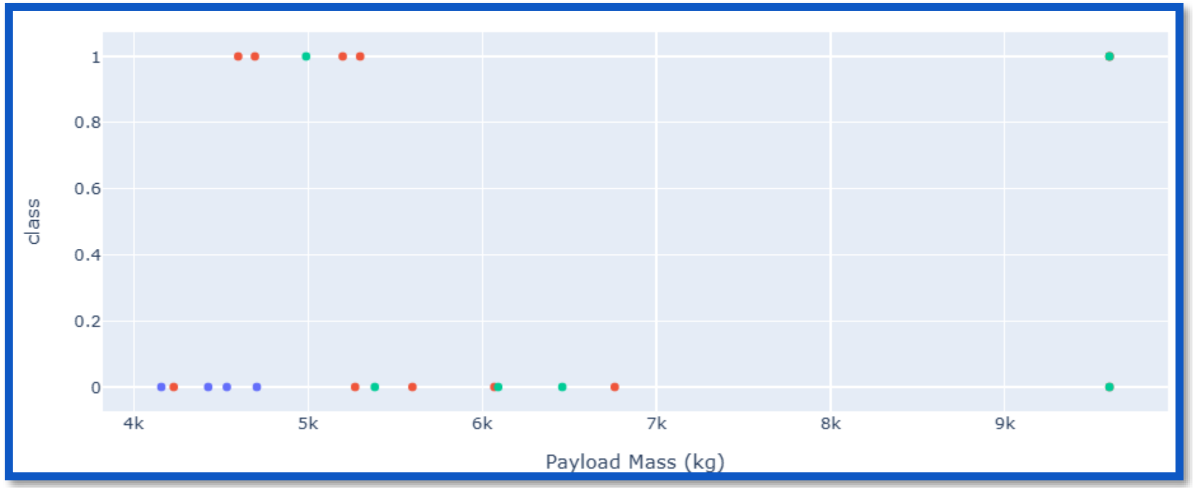


The Launch Site KSC LC-39A achieved the highest rate of successful launches with 76.9% success rate.

Launch success by Site



Low weighted Payload

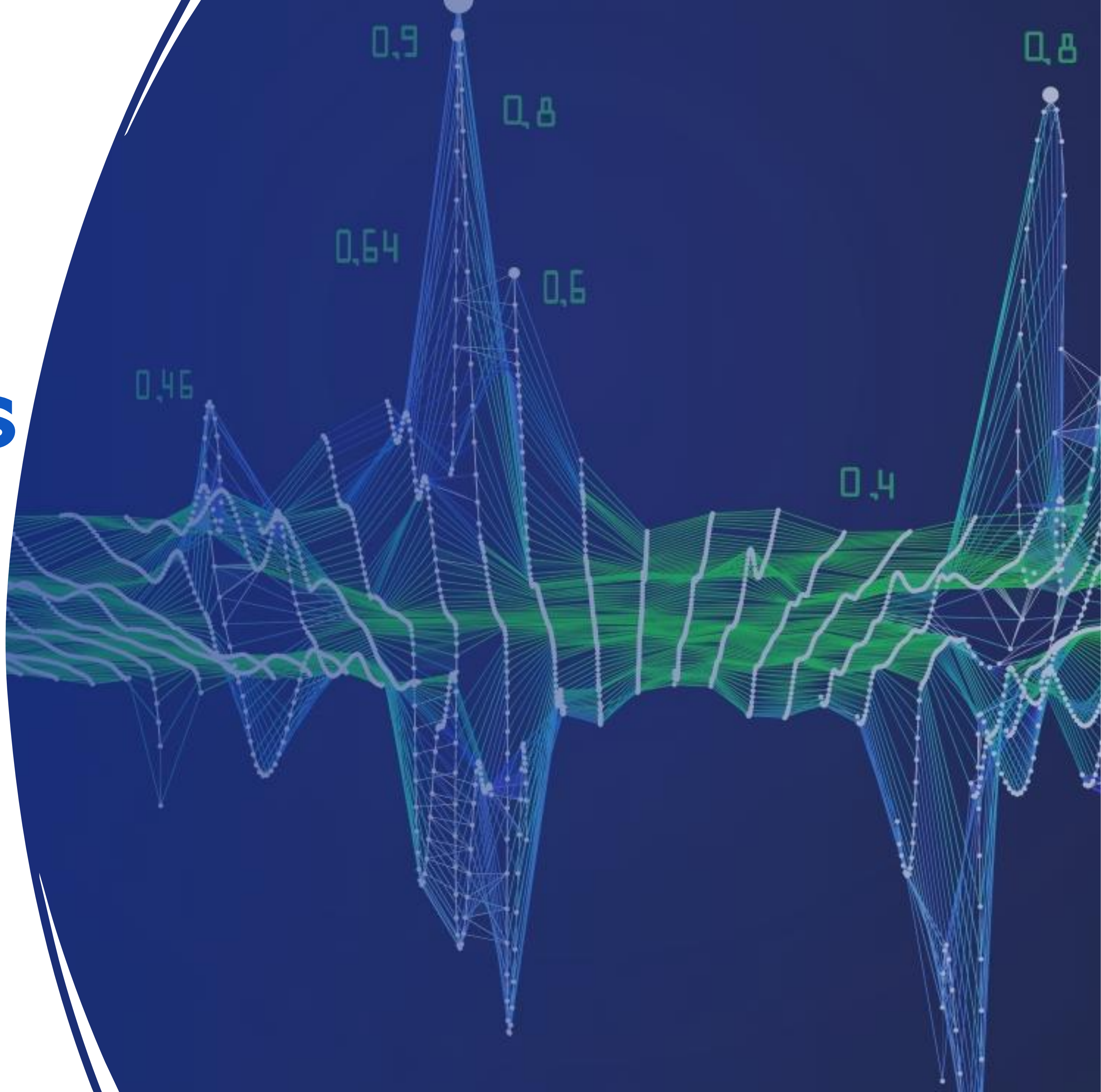


Heavy weighted Payload

- The Success rates for low weighted payloads is higher than the heavy weighted payloads.
- Payloads between 2000 Kg and 5000 Kg have the highest success rate.

Section 5

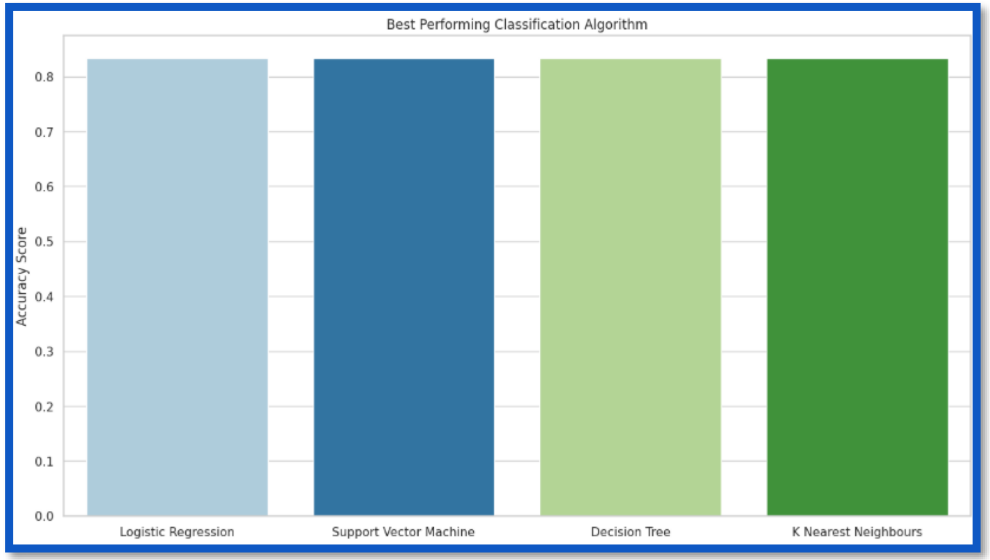
Predictive Analysis (Classification)

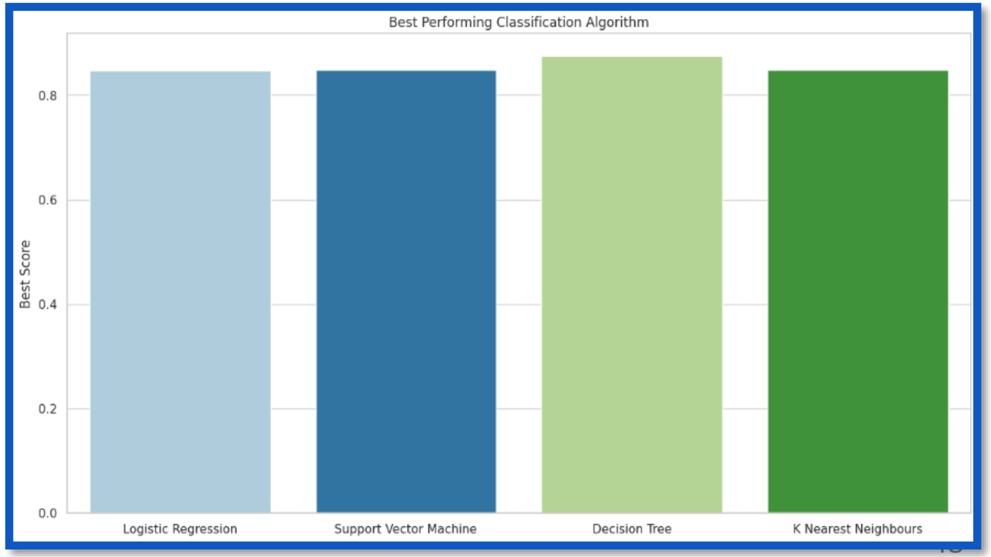


Classification Accuracy

- The Accuracy Score is 83.33% in all Classification Models, which means that all the models perform equally well in terms of correctly classifying the data.
- The most accurate model on the validation set is the Decision Tree model with the highest Accuracy of 87.5%, indicating that it has the potential to outperform other algorithms.

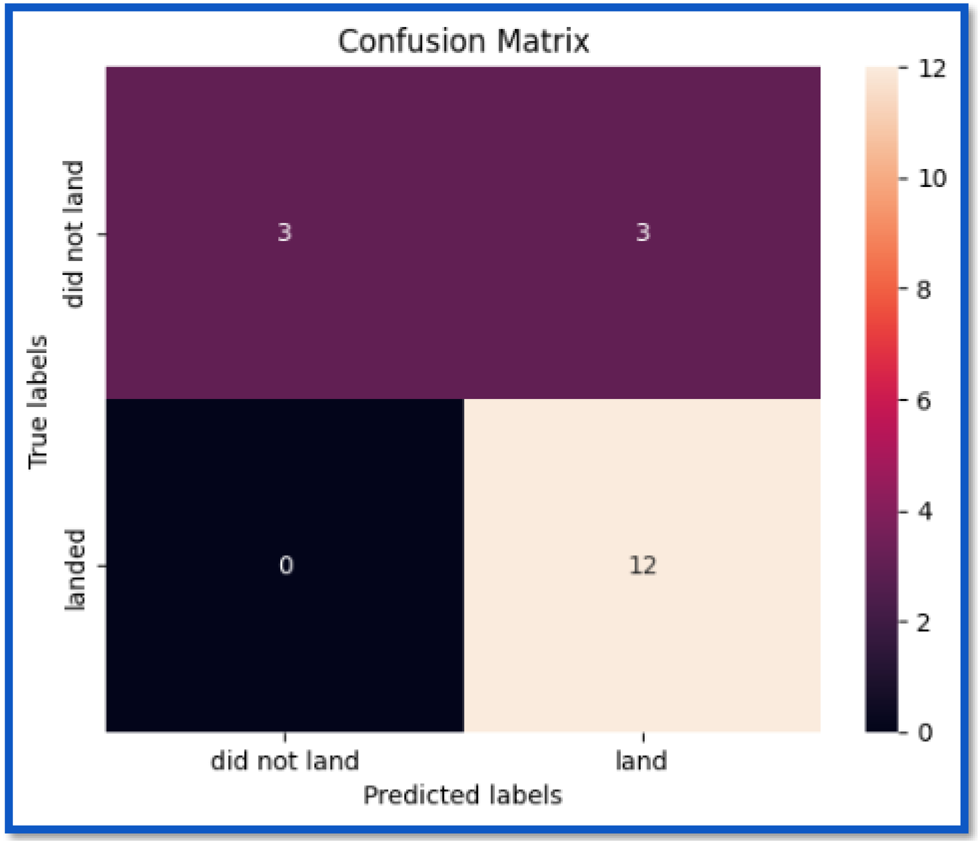
Algorithm	Accuracy Score	Best Score
Logistic Regression	0.833333	0.846429
Support Vector Machine	0.833333	0.848214
Decision Tree	0.833333	0.875000
K Nearest Neighbours	0.833333	0.848214





Confusion Matrix

- The Confusion Matrix for the Decision Tree model shows:
  - True Positives (TP): 12
  - True Negatives (FN): 3
  - False Positives(FP): 3
- The model appears to perform well, with strong ability to correctly predict landings while making a few mistakes on the “did not land” cases.
- The model never fails to identify a landing (False Negative = 0), meaning it has high recall for the “landed” class.



Conclusions

- In this project, we aim to predict whether the first stage of a Falcon 9 launch will successfully land providing insights into launch cost estimation.
- Mission success is influenced by various factors, including the launch site, target orbit, and, most notably, the number of previous launches. This suggests a progressive accumulation of knowledge and experience over successive launches, contributing to the transition from initial failures to successful missions.
- The orbits with best success rates are ES-L1, GEO, HEO, and SSO.
- The Launch Site KSC LC-39A achieved the highest rate of successful launches with 76.9% success rate.
- Payload mass is a key factor in mission success, varying based on the target orbit. Certain orbits require either lighter or heavier payloads; however, in general, missions with lower payload mass tend to achieve higher success rates compared to those with heavier payloads
- The models demonstrated comparable performance on the test set, with the Decision Tree model exhibiting a slight advantage, with an accuracy of 87.5 %.
- Most launch sites are strategically located near the equator to leverage Earth's rotational speed, providing a natural velocity boost that reduces the need for additional fuel and boosters, ultimately lowering launch costs.

Appendix

Data Collection – SpaceX REST API

- Custom functions to retrieve the required information
- Custom logic to clean the data

```
# Lets take a subset of our dataframe keeping only the features we want and the flight number, and date_utc.
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]

# We will remove rows with multiple cores because those are falcon rockets with 2 extra rocket boosters
# and rows that have multiple payloads in a single rocket.
data = data[data['cores'].map(len)==1]
data = data[data['payloads'].map(len)==1]

# Since payloads and cores are lists of size 1 we will also extract the single value in the list and
# replace the feature.
data['cores'] = data['cores'].map(lambda x : x[0])
data['payloads'] = data['payloads'].map(lambda x : x[0])

# We also want to convert the date_utc to a datetime datatype and then extracting the date leaving
# the time
data['date'] = pd.to_datetime(data['date_utc']).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` column we would like to learn the booster name.

```
# Takes the dataset and uses the rocket column to call the API and append the data to the list
def getBoosterVersion(data):
    for x in data['rocket']:
       if x:
        response = requests.get("https://api.spacexdata.com/v4/rockets/"+str(x)).json()
        BoosterVersion.append(response['name'])
```

From the `launchpad` we would like to know the name of the launch site being used, the logitude, and the latitude.

```
# Takes the dataset and uses the launchpad column to call the API and append the data to the list
def getLaunchSite(data):
    for x in data['launchpad']:
       if x:
         response = requests.get("https://api.spacexdata.com/v4/launchpads/"+str(x)).json()
         Longitude.append(response['longitude'])
         Latitude.append(response['latitude'])
         LaunchSite.append(response['name'])
```

From the `payload` we would like to learn the mass of the payload and the orbit that it is going to.

```
# Takes the dataset and uses the payloads column to call the API and append the data to the lists
def getPayloadData(data):
    for load in data['payloads']:
       if load:
        response = requests.get("https://api.spacexdata.com/v4/payloads/"+load).json()
        PayloadMass.append(response['mass_kg'])
        Orbit.append(response['orbit'])
```

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, wheter the core is reused, wheter legs were used, the landing pad used, the block of the core which is a number used to seperate version of cores, the number of times this specific core has been reused, and the serial of the core.

```
# Takes the dataset and uses the cores column to call the API and append the data to the lists
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cores/"+core['core']).json()
                Block.append(response['block'])
                ReusedCount.append(response['reuse_count'])
                Serial.append(response['serial'])
            else:
                Block.append(None)
                ReusedCount.append(None)
                Serial.append(None)
            Outcome.append(str(core['landing_success'])+' '+str(core['landing_type']))
            Flights.append(core['flight'])
            GridFins.append(core['gridfins'])
            Reused.append(core['reused'])
            Legs.append(core['legs'])
            LandingPad.append(core['landpad'])
```

Data Collection – Web Scraping

- Custom functions for web scraping
- Custom logic to fill up the launch_dict values with values from the launch tables

```python
def date_time(table_cells):
    """
    This function returns the data and time from the HTML  table cell
    Input: the  element of a table data cell extracts extra row
    """
    return [data_time.strip() for data_time in list(table_cells.strings)][0:2]

def booster_version(table_cells):
    """
    This function returns the booster version from the HTML  table cell
    Input: the  element of a table data cell extracts extra row
    """
    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):
    """
    This function returns the landing status from the HTML table cell
    Input: the  element of a table data cell extracts extra row
    """
    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):
    """
    This function returns the landing status from the HTML table cell
    Input: the  element of a table data cell extracts extra 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
```

```python
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
        #get table element
        row=rows.find_all('td')
        #if it is number save cells in a dictonary
        if flag:
            extracted_row += 1
            # Flight Number value
            # TODO: Append the flight_number into launch_dict with key `Flight No.`
            launch_dict['Flight No.'].append(flight_number)
            #print(flight_number)
            datatimelist=date_time(row[0])

            # Date value
            # TODO: Append the date into launch_dict with key `Date`
            date = datatimelist[0].strip(',')
            launch_dict['Date'].append(date)
            #print(date)

            # Time value
            # TODO: Append the time into launch_dict with key `Time`
            time = datatimelist[1]
            launch_dict['Time'].append(time)
            #print(time)

            # Booster version
            # TODO: Append the bv into launch_dict with key `Version Booster`
            bv=booster_version(row[1])
            if not(bv):
                bv=row[1].a.string
                launch_dict['Version Booster'].append(bv)
            print(bv)

            # Launch Site
            # TODO: Append the bv into launch_dict with key `Launch Site`
            launch_site = row[2].a.string
            launch_dict['Launch site'].append(launch_site)
            #print(launch_site)

            # Payload
            # TODO: Append the payload into launch_dict with key `Payload`
            payload = row[3].a.string
            launch_dict['Payload'].append(payload)
            #print(payload)

            # Payload Mass
            # TODO: Append the payload_mass into launch_dict with key `Payload mass`
            payload_mass = get_mass(row[4])
            launch_dict['Payload mass'].append(payload_mass)
            #print(payload)
```

```python
            # Orbit
            # TODO: Append the orbit into launch_dict with key `Orbit`
            orbit = row[5].a.string
            launch_dict['Orbit'].append(orbit)
            #print(orbit)

            # Customer
            # TODO: Append the customer into launch_dict with key `Customer`
            if (row[6].a is not None):
                customer = row[6].a.string
            else:
                customer = row[6].string
            launch_dict['Customer'].append(customer)
            #print(customer)

            # Launch outcome
            # TODO: Append the launch_outcome into launch_dict with key `Launch outcome`
            launch_outcome = list(row[7].strings)[0]
            launch_dict['Launch outcome'].append(launch_outcome)
            #print(launch_outcome)

            # Booster landing
            # TODO: Append the launch_outcome into launch_dict with key `Booster landing`
            booster_landing = landing_status(row[8])
            launch_dict['Booster landing'].append(booster_landing)
            #print(booster_landing)
```

Classification Accuracy

- To find which method perform best:

```
lr_score = logreg_cv.score(X_test, Y_test)
svm_score = svm_cv.score(X_test, Y_test)
tree_score = tree_cv.score(X_test, Y_test)
knn_score = knn_cv.score(X_test, Y_test)

lr_best_score = logreg_cv.best_score_
svm_best_score = svm_cv.best_score_
tree_best_score = tree_cv.best_score_
knn_best_score = knn_cv.best_score_
```

```
algorithms = ['Logistic Regression', 'Support Vector Machine', 'Decision Tree', 'K Nearest Neighbours']


scores = [lr_score, svm_score, tree_score, knn_score]

best_scores = [lr_best_score, svm_best_score, tree_best_score, knn_best_score]

column_names = ['Algorithm', 'Accuracy Score', 'Best Score']

df = pd.DataFrame(list(zip(algorithms, scores, best_scores)),columns = column_names)
df
```

- To visualize Best Performing Classification Algorithm (Best Score)

```
sns.set(style="whitegrid")

plt.figure(figsize=(15,8))
sns.barplot(x=algorithms, y=best_scores, palette="Paired")
plt.title("Best Performing Classification Algorithm")
plt.ylabel("Best Score")
plt.show()
```

- To visualize Best Performing Classification Algorithm (Accuracy Score)

```
plt.figure(figsize=(15,8))
sns.barplot(x=algorithms, y=scores, palette="Paired")
plt.title("Best Performing Classification Algorithm")
plt.ylabel("Accuracy Score")
plt.show()
```

Thank you!