# Data Science Capstone Project: SPACEX

By

Sayeda Fatima

# Outline

**Summary of Methodologies**

- Data Collection was carried out through SPACE X Rest API
- Data Wrangling employed One-Hot Coding
- Exploratory Data Analysis (EDA) using SQL and Visualizations
- Interactive Visual Analytics using Folium and Plotly-Dash
- Predictive Analysis using Machine Learning Algorithms

**Summary of Results**

- Data Analysis revealed relationships between variables and conditions  determining success or failure
- Interactive Visualizations achived
- Best Predictive Modelling is Machine Learning

## Project Background and Context

- The aim of this project is to predict success or failure of Falcon 9's landings. SpaceX's website records Falcon 9 rocket's launch cost as $62 million in contrast to $165 million by other providers. The price difference is due to SpaceX's ability to reuse its first stage. Hence, by predicting success of landing, the cost of launch can be estimated. Consequently, this information can be useful for other providers competing with SpaceX for rocket launches.

## Problems requiring resolution

- What are the main characteristics of a successful or failed landing?
- What are the impacts of variable relationships on landing success or failure?
- What are the conditions which will facilitate SpaceX in acquiring best landing success rates?

# Methodology

**Data Collection**

- SPACE X Rest API

**Data Wrangling**

- Unwanted Column Drop
- One-Hot Coding

**Exploratory Data Analysis (EDA)**

- Visualizations
- SQL

**Interactive Visual Analytics**

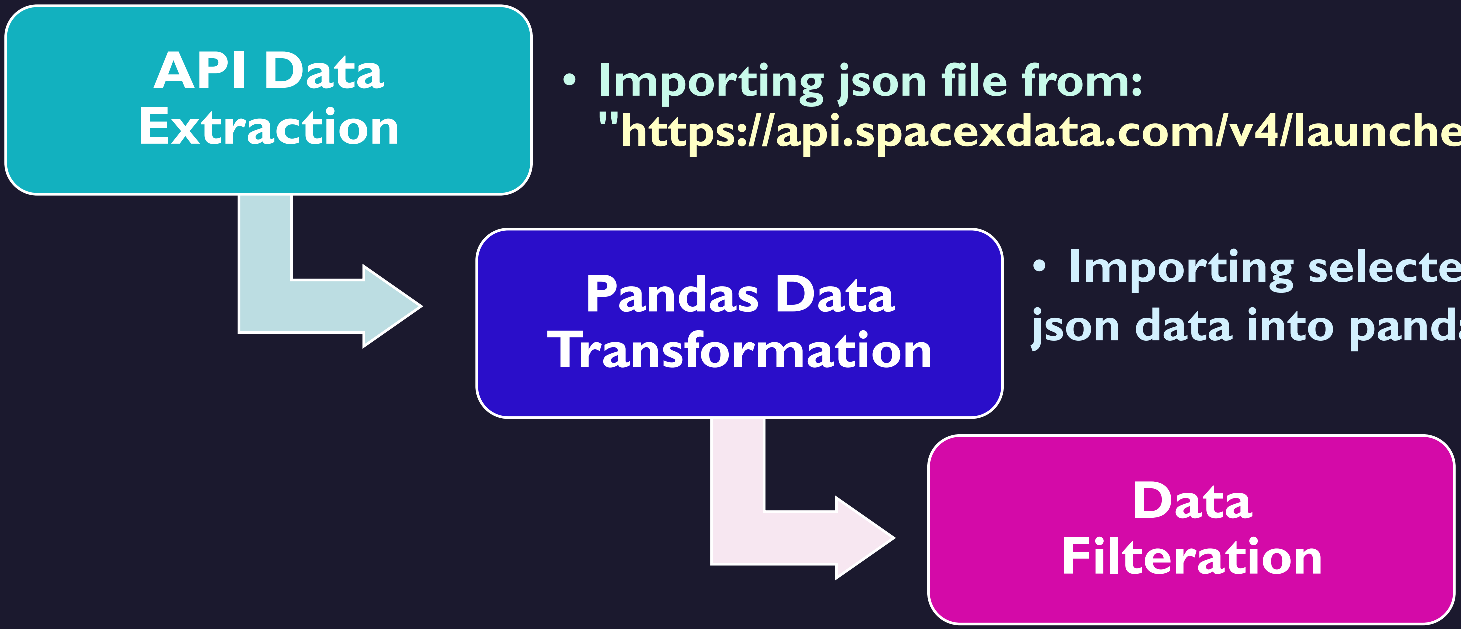
- Folium
- Plotly Dash

**Predictive Analysis**

- Classification Models

## Data Collection Methodology

The following methods have been employed:



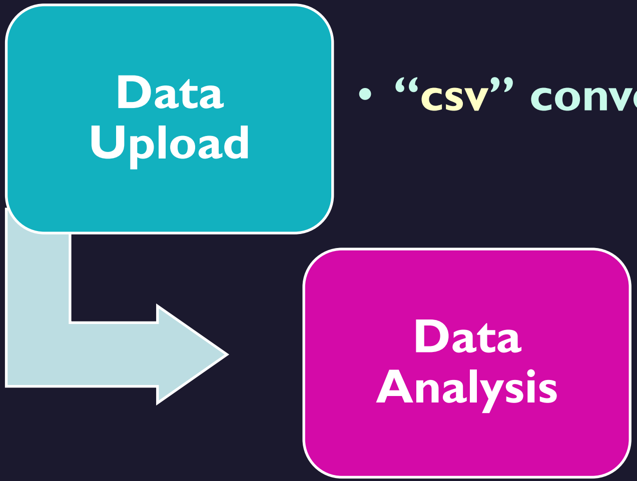
- Importing json file from: "https://api.spacexdata.com/v4/launches/past"

- Importing selected columns from json data into pandas dataframe

- Filtering 'Falcon9' rows
- "Payload" Missing Value replacement with average values

## Data Wrangling Methodology

**The following methods have been employed:**



- "csv" conversion to Dataframe

Task 1: Calculating "Number of Launches"
Task 2: Calculating "Orbit Occurrences"

# EDA & IVA Methodology

The following methods have been employed:

**Folium,**
***an interactive leaflet map***

**The following folium methodology was employed:**
- **Reverse Geo-coding to find State names with Launch Site concentration**
- **Marking of Launch sites**
- **Marking of Successful/Failed launches**
- **Marking distance between launch sites and proximities**

**Plotly Dash,**
***an interactive framework for building ML & data science web apps***

**The following plotly dash method was employed:**
- **Creation of App Layout**
- **Creation of Callback Functions for Pie Chart**
- **Creation of Callback Functions for Scatter Chart with Slider**

# Predictive Analysis Methodology

The following methods have been employed:

**Decision Trees** which employs a top-down approach to a dataset and tries to group data on the basis of dissimilarity till a certain degree of similarity is attained

**K-Nearest Neighbors (KNN)** which classifies a given dataset elements on the basis of similarity or closeness

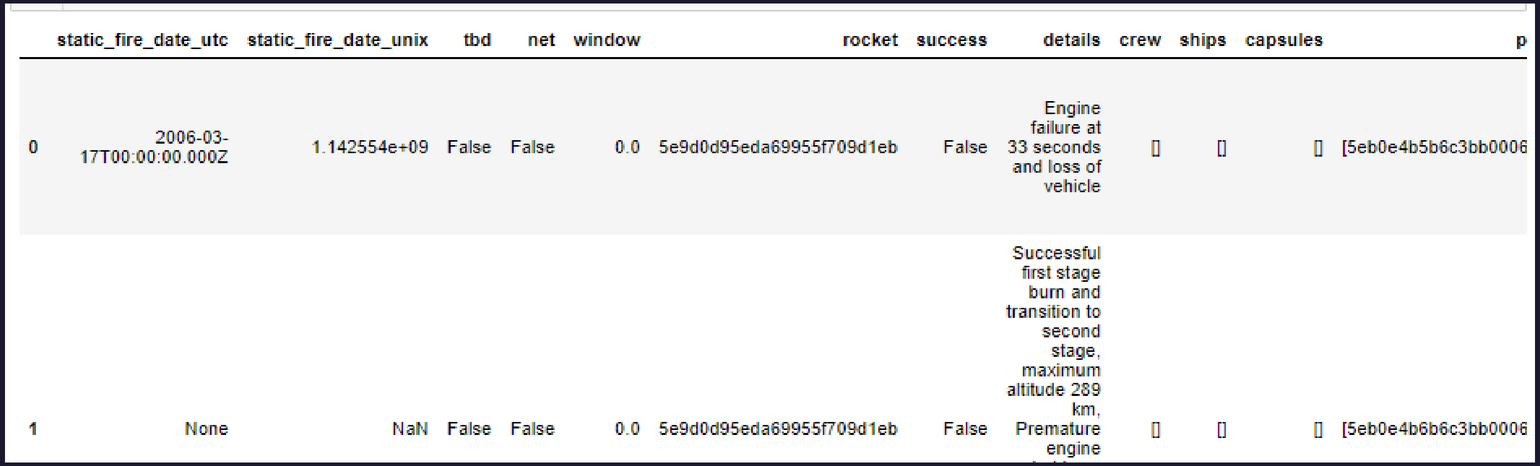
**Support Vector Machines (SVM)** which finds the best hyperplane to divide a given dataset into two classes

**Logistic Regression** which estimates the probability of an event occurring, like yes or no, based on selected dataset of independent variables.

Business Use

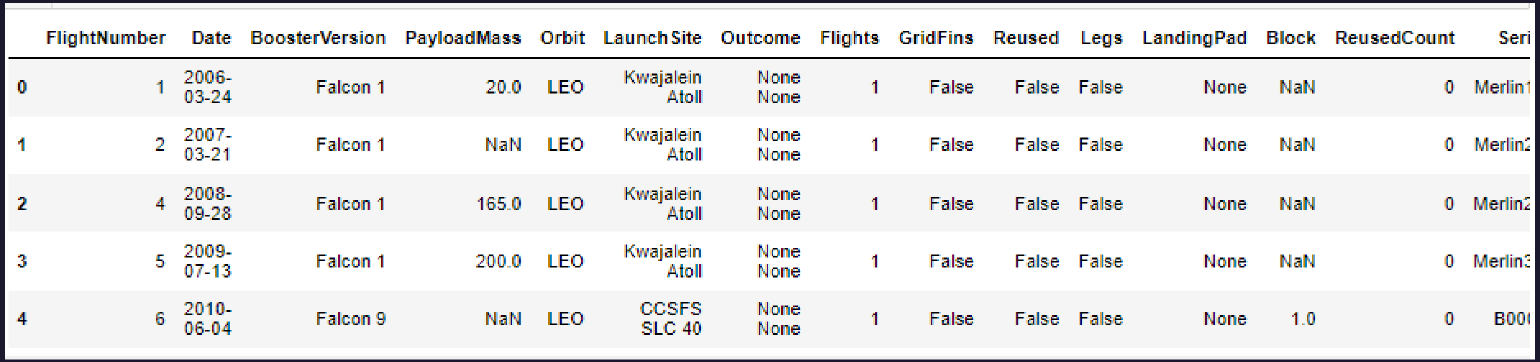
# Data Collection Results

**API Data Extraction**



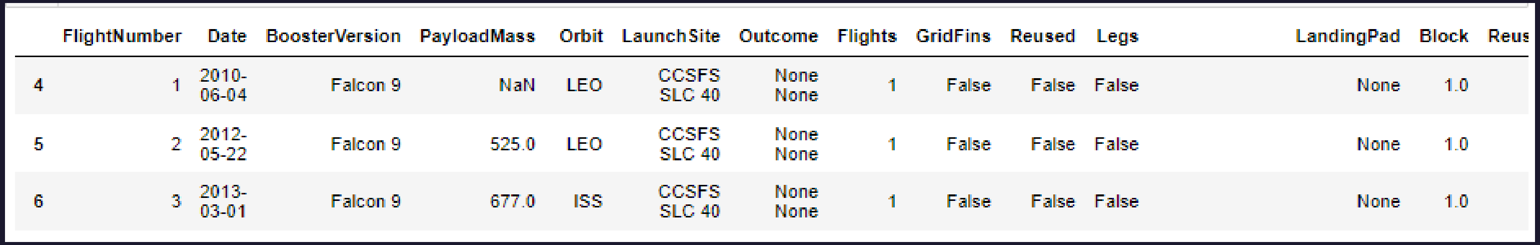
| | static_fire_date_utc | static_fire_date_unix | tbd | net | window | rocket | success | details | crew | ships | capsules | p |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2006-03-17T00:00:00.000Z | 1.142554e+09 | False | False | 0.0 | 5e9d0d95eda69955f709d1eb | False | Engine failure at 33 seconds and loss of vehicle | [] | [] | [] | [5eb0e4b5b6c3bb0006 |
| 1 | None | NaN | False | False | 0.0 | 5e9d0d95eda69955f709d1eb | False | Successful first stage burn and transition to second stage, maximum altitude 289 km, Premature engine | [] | [] | [] | [5eb0e4b6b6c3bb0006 |

**Pandas Data Transformation**



| | FlightNumber | Date | BoosterVersion | PayloadMass | Orbit | LaunchSite | Outcome | Flights | GridFins | Reused | Legs | LandingPad | Block | ReusedCount | Seri |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2006-03-24 | Falcon 1 | 20.0 | LEO | Kwajalein Atoll | None None | 1 | False | False | False | None | NaN | 0 | Merlin1 |
| 1 | 2 | 2007-03-21 | Falcon 1 | NaN | LEO | Kwajalein Atoll | None None | 1 | False | False | False | None | NaN | 0 | Merlin2 |
| 2 | 4 | 2008-09-28 | Falcon 1 | 165.0 | LEO | Kwajalein Atoll | None None | 1 | False | False | False | None | NaN | 0 | Merlin2 |
| 3 | 5 | 2009-07-13 | Falcon 1 | 200.0 | LEO | Kwajalein Atoll | None None | 1 | False | False | False | None | NaN | 0 | Merlin3 |
| 4 | 6 | 2010-06-04 | Falcon 9 | NaN | LEO | CCSFS SLC 40 | None None | 1 | False | False | False | None | 1.0 | 0 | B00 |

**Data Filtration**



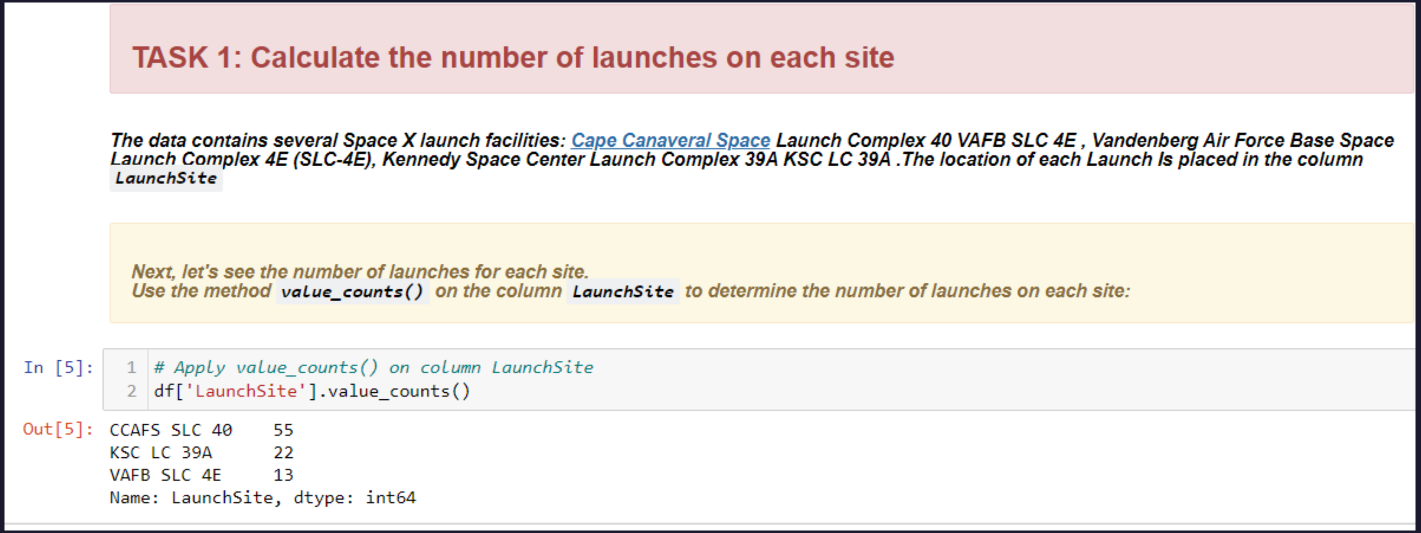
| | FlightNumber | Date | BoosterVersion | PayloadMass | Orbit | LaunchSite | Outcome | Flights | GridFins | Reused | Legs | LandingPad | Block | Reus |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 1 | 2010-06-04 | Falcon 9 | NaN | LEO | CCSFS SLC 40 | None None | 1 | False | False | False | None | 1.0 | |
| 5 | 2 | 2012-05-22 | Falcon 9 | 525.0 | LEO | CCSFS SLC 40 | None None | 1 | False | False | False | None | 1.0 | |
| 6 | 3 | 2013-03-01 | Falcon 9 | 677.0 | ISS | CCSFS SLC 40 | None None | 1 | False | False | False | None | 1.0 | |

Business Use

## Data Wrangling Results I

**Number of Launches**

Maximum launches From "CCAFS SLC 40"



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`

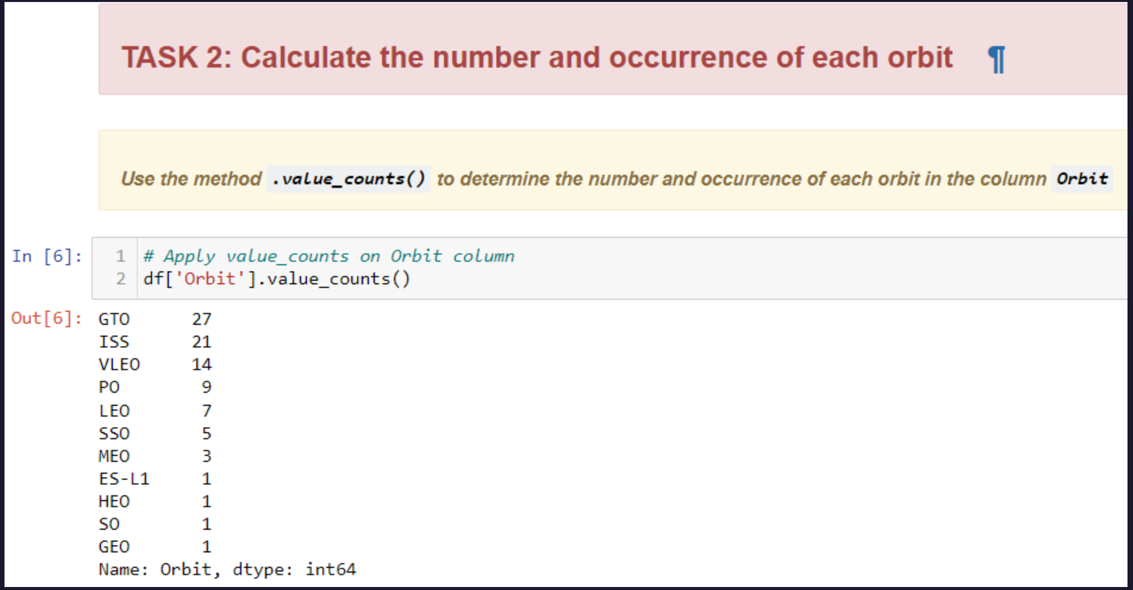
*Next, let's see the number of launches for each site.*
*Use the method* `value_counts()` *on the column* `LaunchSite` *to determine the number of launches on each site:*

```
In [5]:
# Apply value_counts() on column LaunchSite
df['LaunchSite'].value_counts()
```

```
Out[5]: CCAFS SLC 40    55
        KSC LC 39A      22
        VAFB SLC 4E     13
        Name: LaunchSite, dtype: int64
```

**Number of Orbital Occurances**

Orbit "GTO" exhibited Maximum Occurances



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`

```
In [6]:
# Apply value_counts on Orbit column
df['Orbit'].value_counts()
```

```
Out[6]: 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
```

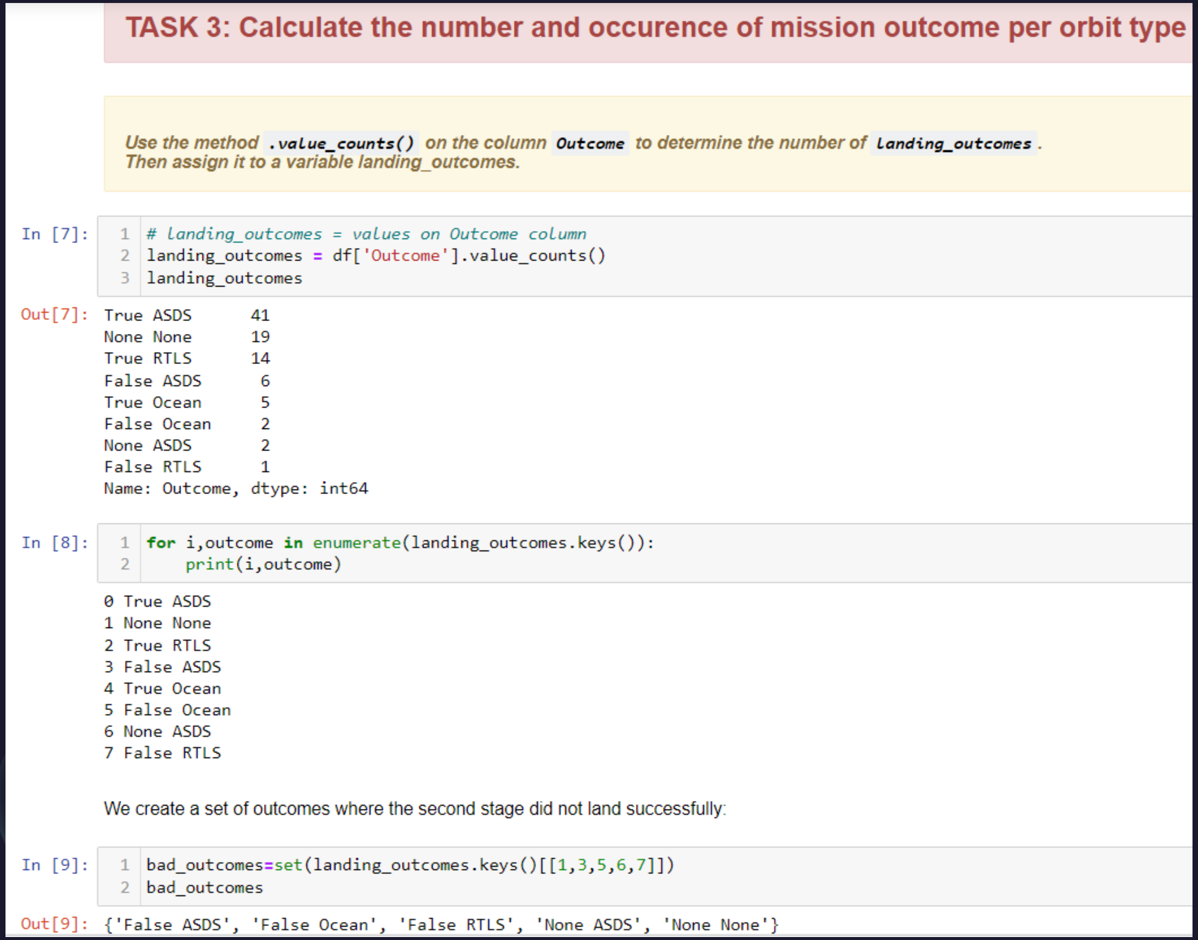
# Data Wrangling Results 2

**Mission Outcome Occurrences**

60 True Outcomes

**New Column Creation for Labelling**

Copied "Outcomes" Column to New "Class" Column



**TASK 3: Calculate the number and occurence of mission outcome per orbit type**

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

In [7]:
```
# landing_outcomes = values on Outcome column
landing_outcomes = df['Outcome'].value_counts()
landing_outcomes
```

Out[7]:
```
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
```

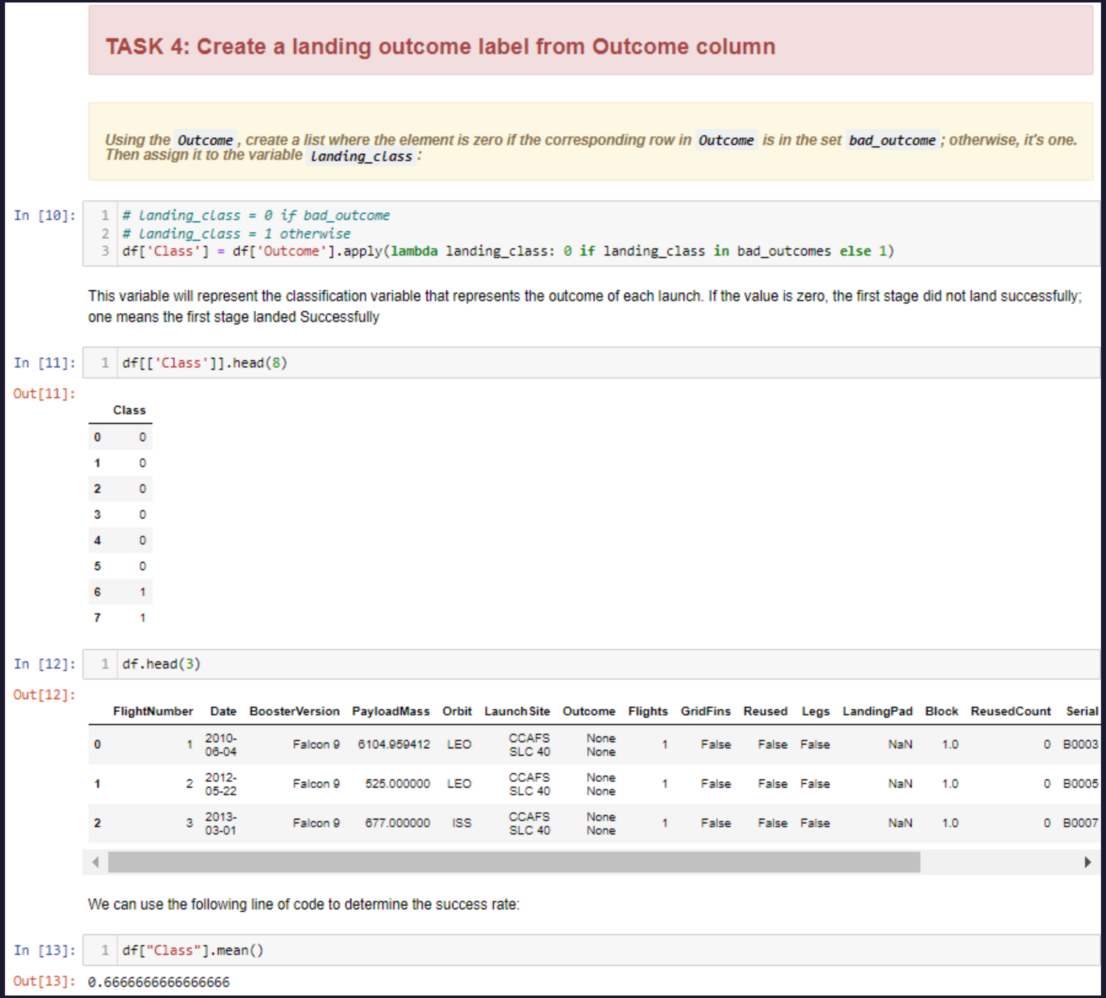
In [8]:
```
for i,outcome in enumerate(landing_outcomes.keys()):
    print(i,outcome)
```

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

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

In [9]:
```
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
bad_outcomes
```

Out[9]:
```
{'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```



**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`:*

In [10]:
```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
df['Class'] = df['Outcome'].apply(lambda landing_class: 0 if landing_class 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

In [11]:
```
df[['Class']].head(8)
```

Out[11]:

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

In [12]:
```
df.head(3)
```

Out[12]:

| | FlightNumber | Date | BoosterVersion | PayloadMass | Orbit | LaunchSite | Outcome | Flights | GridFins | Reused | Legs | LandingPad | Block | ReusedCount | Serial |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2010-06-04 | Falcon 9 | 6104.959412 | LEO | CCAFS SLC 40 | None None | 1 | False | False | False | NaN | 1.0 | 0 | B0003 |
| 1 | 2 | 2012-05-22 | Falcon 9 | 525.000000 | LEO | CCAFS SLC 40 | None None | 1 | False | False | False | NaN | 1.0 | 0 | B0005 |
| 2 | 3 | 2013-03-01 | Falcon 9 | 677.000000 | ISS | CCAFS SLC 40 | None None | 1 | False | False | False | NaN | 1.0 | 0 | B0007 |

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

In [13]:
```
df["Class"].mean()
```

Out[13]:
```
0.6666666666666666
```

# EDA Interactive Visuals Results I

**Interpretation**

Task 1: Increase in success rate for each site

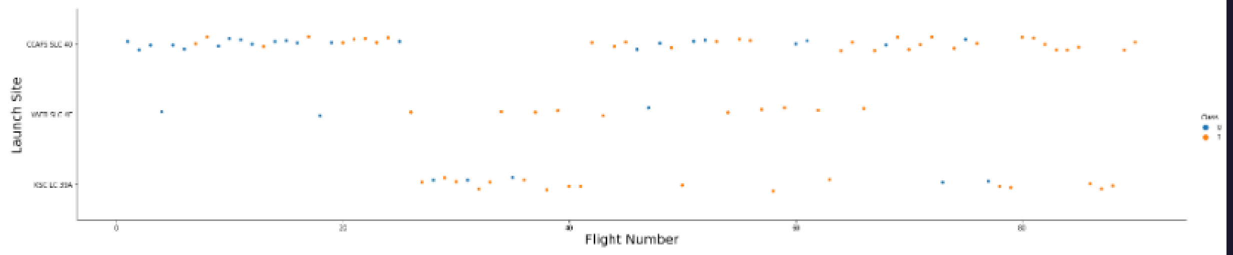
Task 2: Landing Failures seem to be related to excessive payload

Task 3: Best success rate was exhibited by "ES-L1", "GEO", "HEO" & "SSO"

## TASK 1: Visualize the relationship between Flight Number and Launch Site

*Use the function `catplot` to plot `FlightNumber` vs `LaunchSite`, set the parameter `x` parameter to `FlightNumber`, set the `y` to `Launch Site` and set the parameter `hue` to `'class'`*

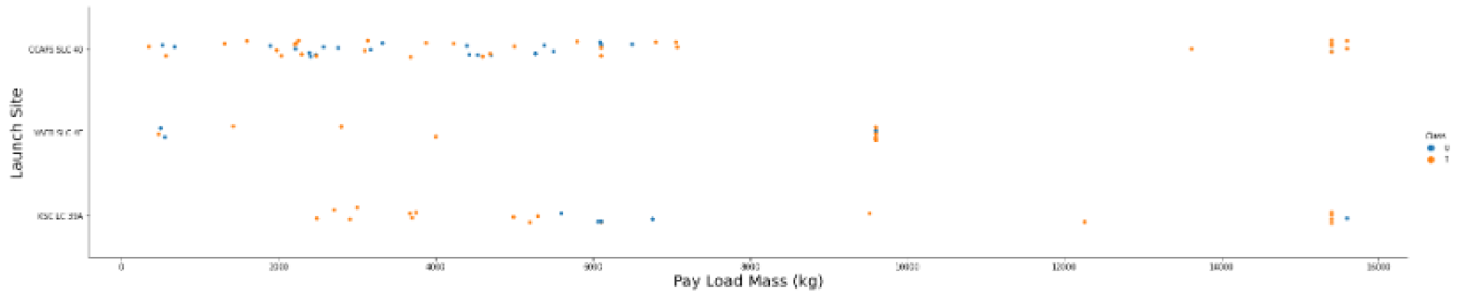
```
# Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to be the class valu
sns.catplot(y="LaunchSite", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Launch Site",fontsize=20)
plt.show()
```



## TASK 2: Visualize the relationship between Payload and Launch Site

*We also want to observe if there is any relationship between launch sites and their payload mass.*

```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch site, and hue to be the class
sns.catplot(y="LaunchSite", x="PayloadMass", hue="Class", data=df, aspect = 5)
plt.xlabel("Pay Load Mass (kg)",fontsize=20)
plt.ylabel("Launch Site",fontsize=20)
plt.show()
```



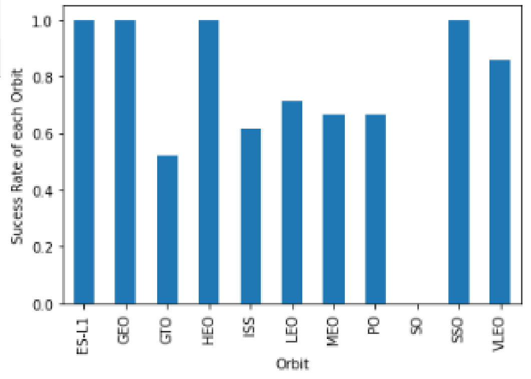
## TASK 3: Visualize the relationship between success rate of each orbit type

*Next, we want to visually check if there are any relationship between success rate and orbit type.*

Let's create a `bar chart` for the sucess rate of each orbit

```
# HINT use groupby method on Orbit column and get the mean of Class column
data = df.groupby('Orbit')['Class'].mean()
ax = data.plot(kind='bar')
ax.set_xlabel("Orbit")
ax.set_ylabel("Sucess Rate of each Orbit")
```

```
Text(0, 0.5, 'Sucess Rate of each Orbit')
```



# EDA Interactive Visuals Results 2

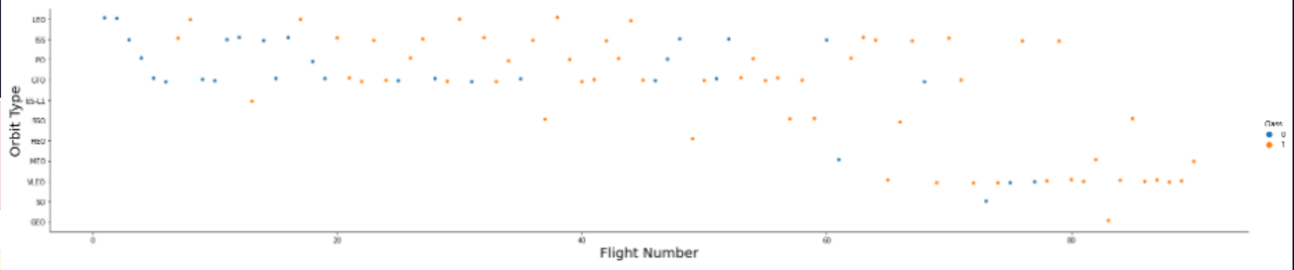
**Task 4**

There is an increase in success rate with number of flights

**TASK 4: Visualize the relationship between FlightNumber and Orbit type**

*For each orbit, we want to see if there is any relationship between FlightNumber and Orbit type.*

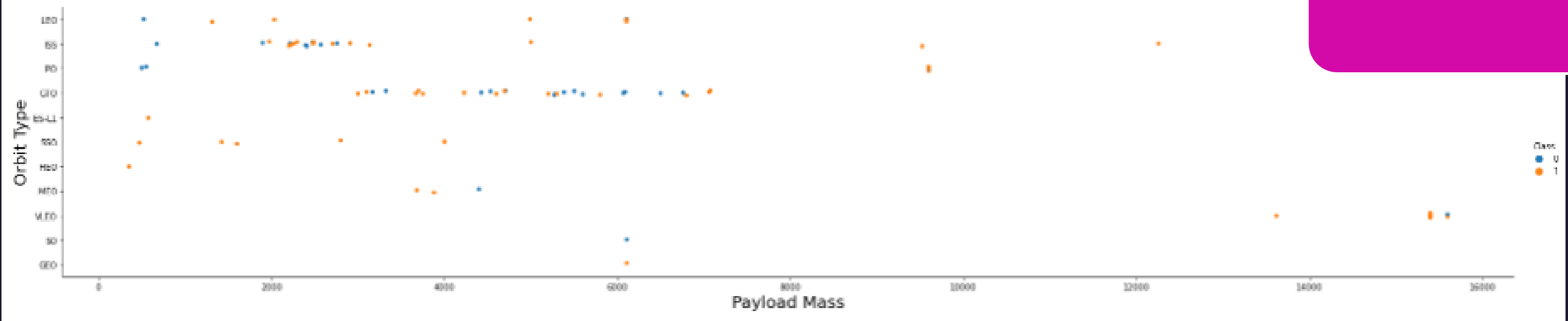
```python
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class value
sns.catplot(y="Orbit", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Orbit Type",fontsize=20)
plt.show()
```



**TASK 5: Visualize the relationship between Payload and Orbit type**

*Similarly, we can plot the Payload vs. Orbit scatter point charts to reveal the relationship between Payload and Orbit type*

```python
# Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, and hue to be the class value
sns.catplot(y="Orbit", x="PayloadMass", hue="Class", data=df, aspect = 5)
plt.xlabel("Payload Mass",fontsize=20)
plt.ylabel("Orbit Type",fontsize=20)
plt.show()
```



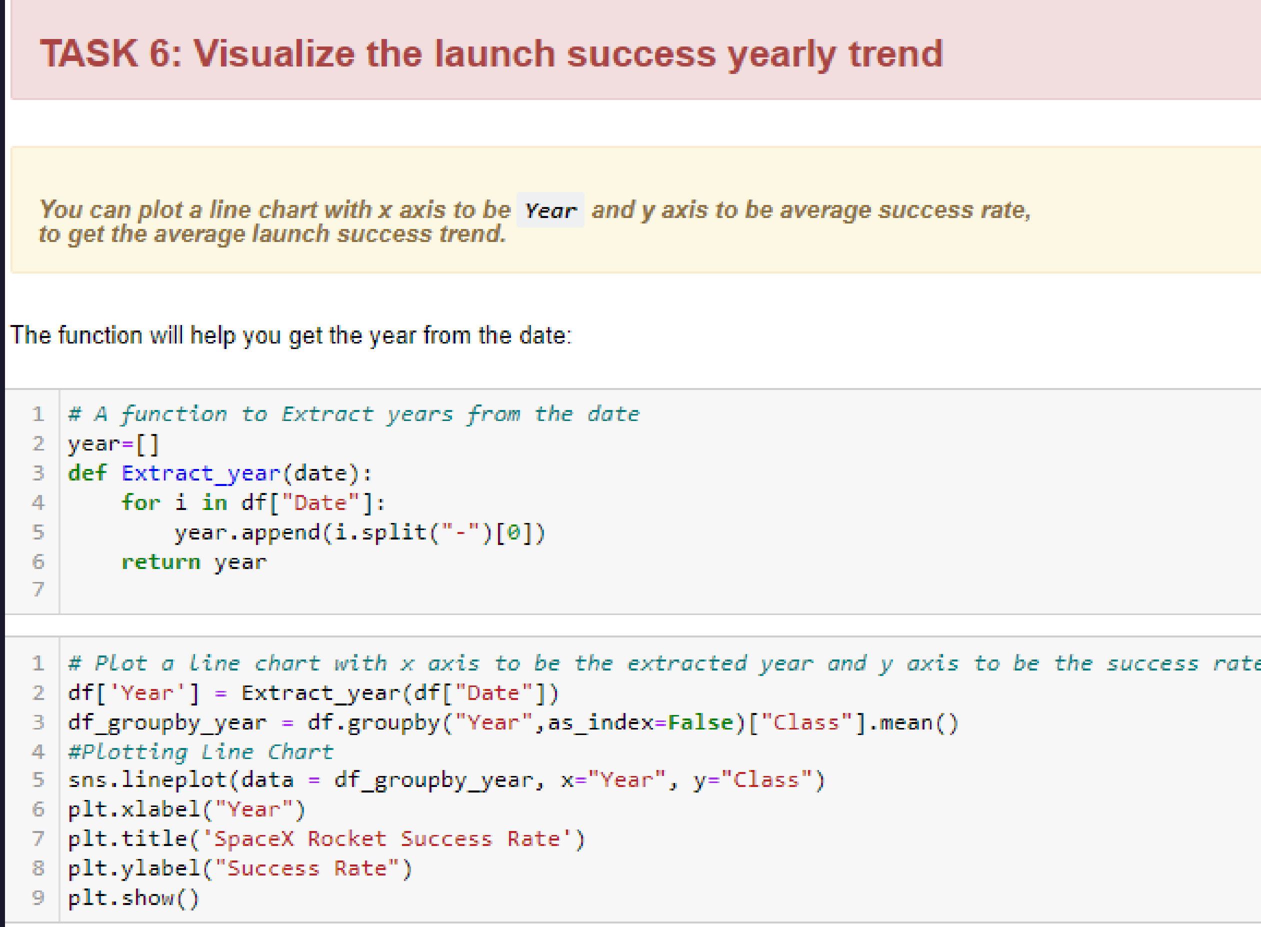
**Task 5**

Increased "Payload Mass" increases success rate in some Orbits like "LEO" while decreasing "Payload Mass" increases success rate in some Orbits like " GTO "

## EDA Interactive Visuals Results 3

**Tasks 6**

There is a general increase in success rate since “2013”



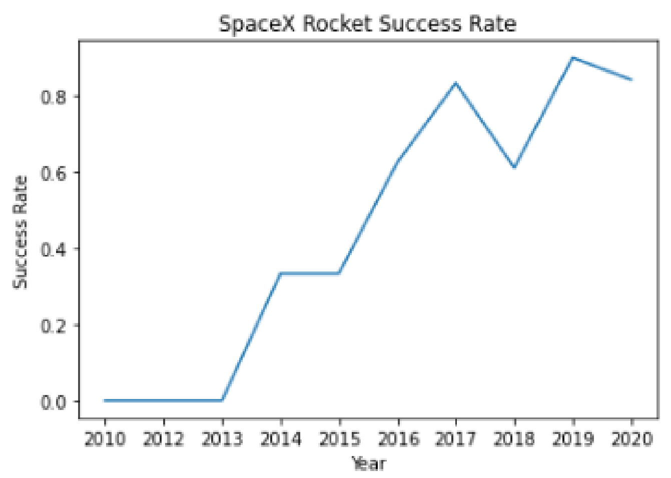
### TASK 6: Visualize the launch success yearly trend

*You can plot a line chart with x axis to be `Year` and y axis to be average success rate, to get the average launch success trend.*

The function will help you get the year from the date:

```
# A function to Extract years from the date
year=[]
def Extract_year(date):
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year

```

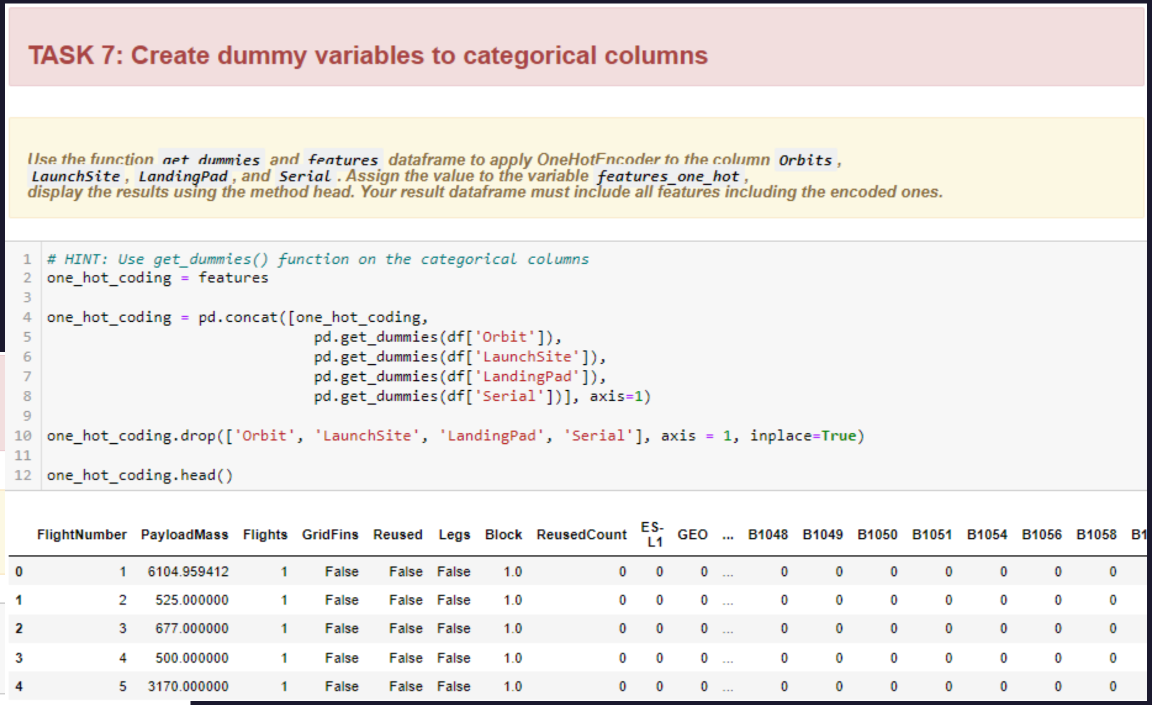
```
# Plot a line chart with x axis to be the extracted year and y axis to be the success rate
df['Year'] = Extract_year(df["Date"])
df_groupby_year = df.groupby("Year",as_index=False)["Class"].mean()
#Plotting Line Chart
sns.lineplot(data = df_groupby_year, x="Year", y="Class")
plt.xlabel("Year")
plt.title('SpaceX Rocket Success Rate')
plt.ylabel("Success Rate")
plt.show()
```



## EDA Interactive Visuals Results 4

**Task 7 & 8**

"One Hot Coding" transposed features columns as float

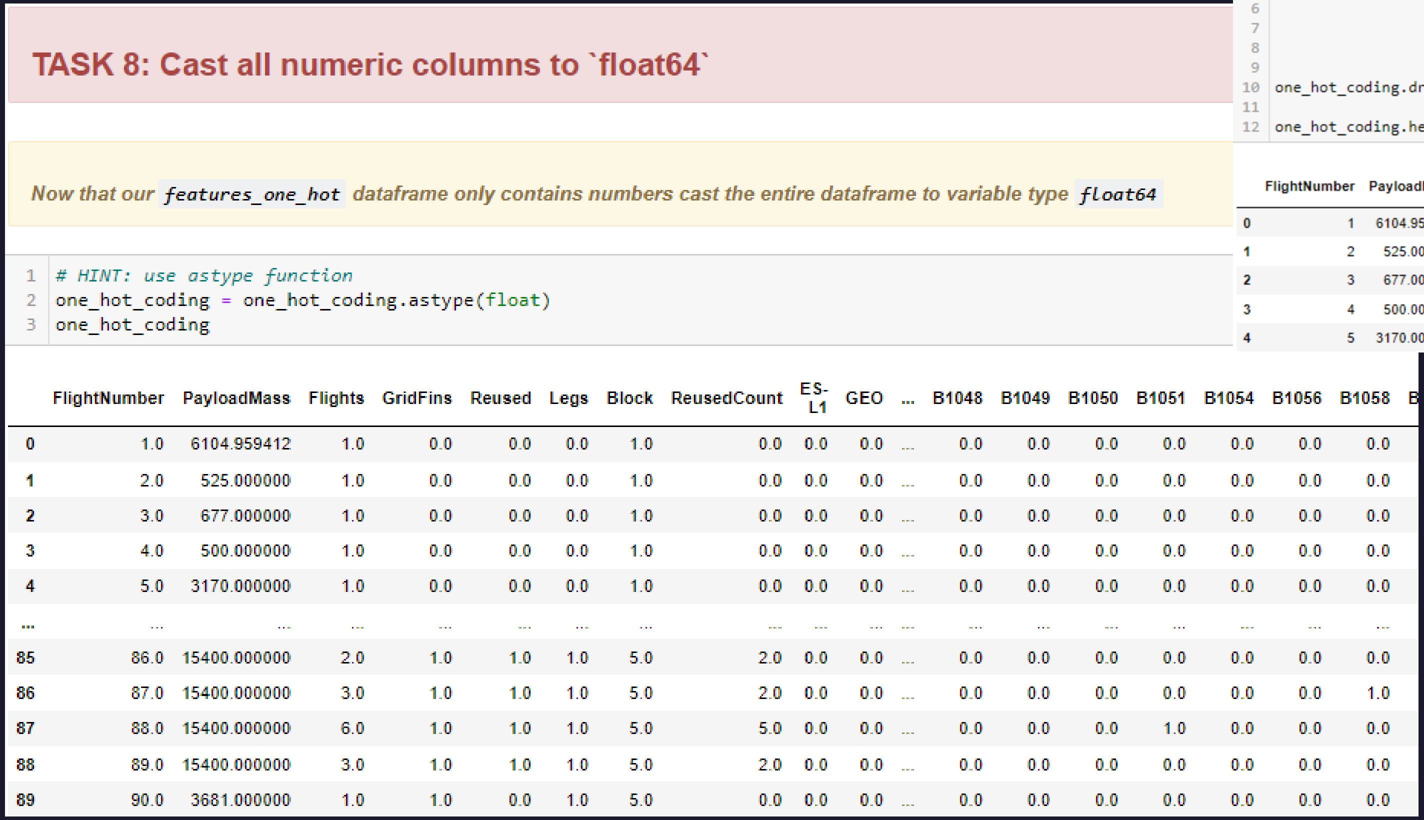


### TASK 7: Create dummy variables to categorical columns

*Use the function `get_dummies` and `features` dataframe to apply OneHotEncoder to the column `Orbits`, `LaunchSite`, `LandingPad`, and `Serial`. Assign the value to the variable `features_one_hot`, display the results using the method head. Your result dataframe must include all features including the encoded ones.*

```python
# HINT: Use get_dummies() function on the categorical columns
one_hot_coding = features

one_hot_coding = pd.concat([one_hot_coding,
                            pd.get_dummies(df['Orbit']),
                            pd.get_dummies(df['LaunchSite']),
                            pd.get_dummies(df['LandingPad']),
                            pd.get_dummies(df['Serial'])], axis=1)

one_hot_coding.drop(['Orbit', 'LaunchSite', 'LandingPad', 'Serial'], axis = 1, inplace=True)

one_hot_coding.head()
```

| | FlightNumber | PayloadMass | Flights | GridFins | Reused | Legs | Block | ReusedCount | ES-L1 | GEO | ... | B1048 | B1049 | B1050 | B1051 | B1054 | B1056 | B1058 | B1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 6104.959412 | 1 | False | False | False | 1.0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 1 | 2 | 525.000000 | 1 | False | False | False | 1.0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 2 | 3 | 677.000000 | 1 | False | False | False | 1.0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 3 | 4 | 500.000000 | 1 | False | False | False | 1.0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 4 | 5 | 3170.000000 | 1 | False | False | False | 1.0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |



### TASK 8: Cast all numeric columns to `float64`

*Now that our `features_one_hot` dataframe only contains numbers cast the entire dataframe to variable type `float64`*

```python
# HINT: use astype function
one_hot_coding = one_hot_coding.astype(float)
one_hot_coding
```

| | FlightNumber | PayloadMass | Flights | GridFins | Reused | Legs | Block | ReusedCount | ES-L1 | GEO | ... | B1048 | B1049 | B1050 | B1051 | B1054 | B1056 | B1058 | B |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1.0 | 6104.959412 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 1 | 2.0 | 525.000000 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 2 | 3.0 | 677.000000 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 3 | 4.0 | 500.000000 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 4 | 5.0 | 3170.000000 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 85 | 86.0 | 15400.000000 | 2.0 | 1.0 | 1.0 | 1.0 | 5.0 | 2.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 86 | 87.0 | 15400.000000 | 3.0 | 1.0 | 1.0 | 1.0 | 5.0 | 2.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | |
| 87 | 88.0 | 15400.000000 | 6.0 | 1.0 | 1.0 | 1.0 | 5.0 | 5.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | |
| 88 | 89.0 | 15400.000000 | 3.0 | 1.0 | 1.0 | 1.0 | 5.0 | 2.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 89 | 90.0 | 3681.000000 | 1.0 | 1.0 | 0.0 | 1.0 | 5.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |

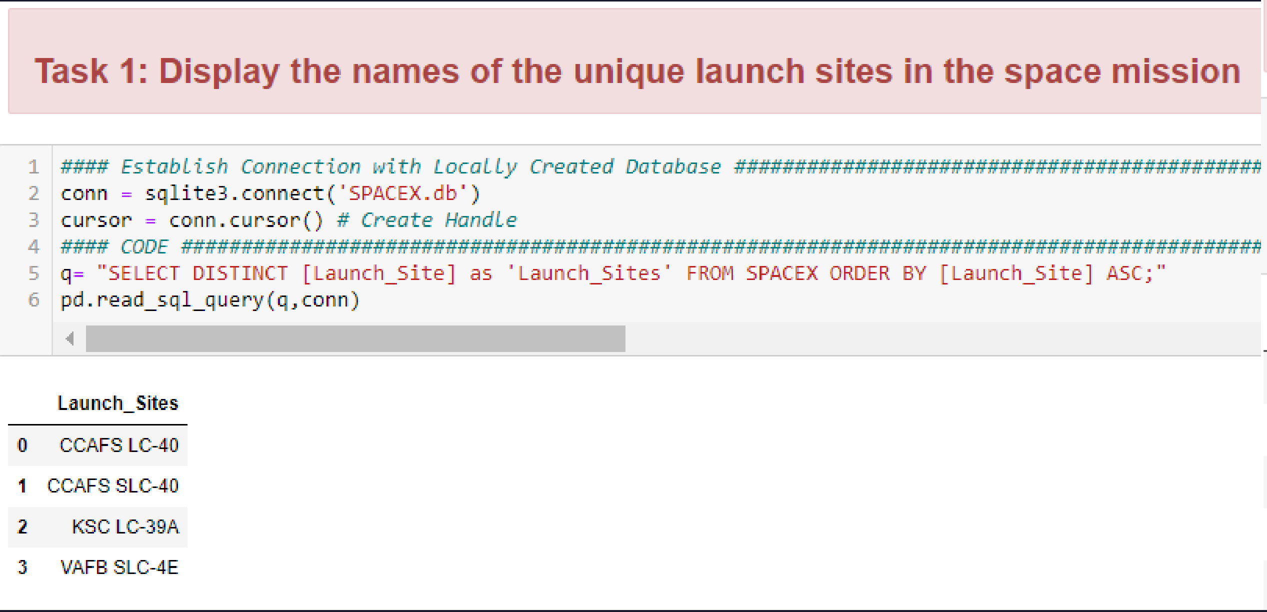
## EDAwith SQL I

**Bonus Task**

**Local SQL Database Creation achieved from csv**

**Tasks**

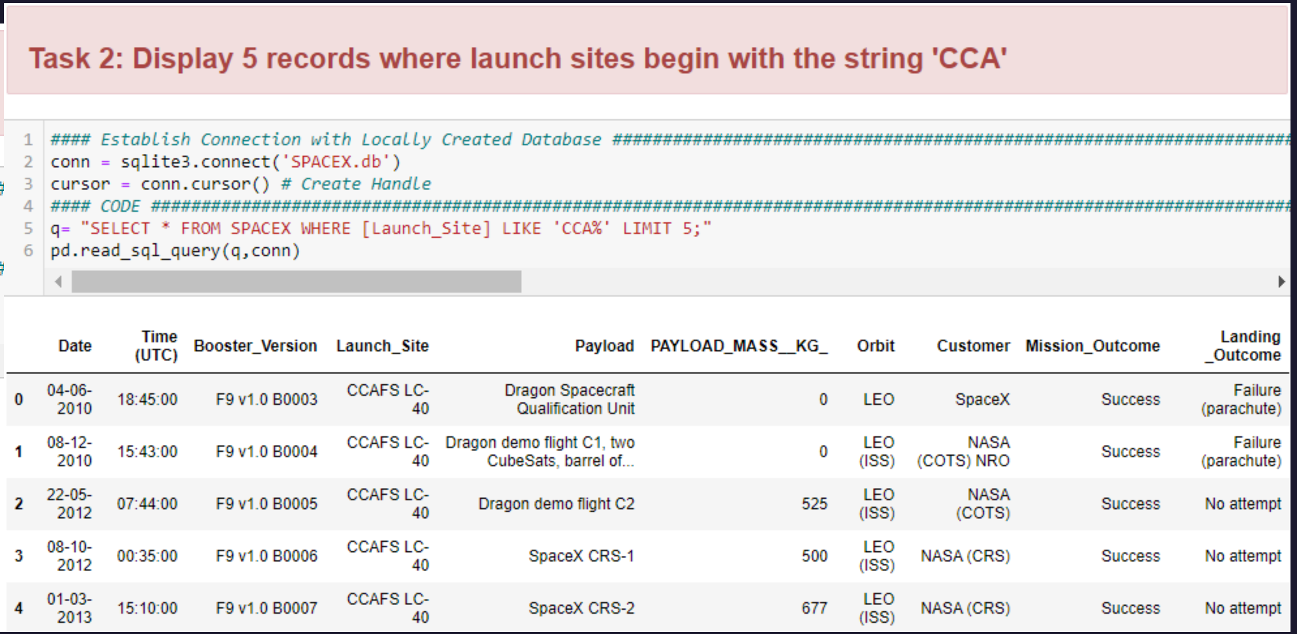
1) 4 Unique Launch Sites were found
2) Records of Launch Sites Like 'CCA' achieved



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

```
#### Establish Connection with Locally Created Database ##############################################
conn = sqlite3.connect('SPACEX.db')
cursor = conn.cursor() # Create Handle
#### CODE ###################################################################################################
q= "SELECT DISTINCT [Launch_Site] as 'Launch_Sites' FROM SPACEX ORDER BY [Launch_Site] ASC;"
pd.read_sql_query(q,conn)
```

| | Launch_Sites |
|---|---|
| 0 | CCAFS LC-40 |
| 1 | CCAFS SLC-40 |
| 2 | KSC LC-39A |
| 3 | VAFB SLC-4E |



**Task 2: Display 5 records where launch sites begin with the string 'CCA'**

```
#### Establish Connection with Locally Created Database ##############################################
conn = sqlite3.connect('SPACEX.db')
cursor = conn.cursor() # Create Handle
#### CODE ###################################################################################################
q= "SELECT * FROM SPACEX WHERE [Launch_Site] LIKE 'CCA%' LIMIT 5;"
pd.read_sql_query(q,conn)
```

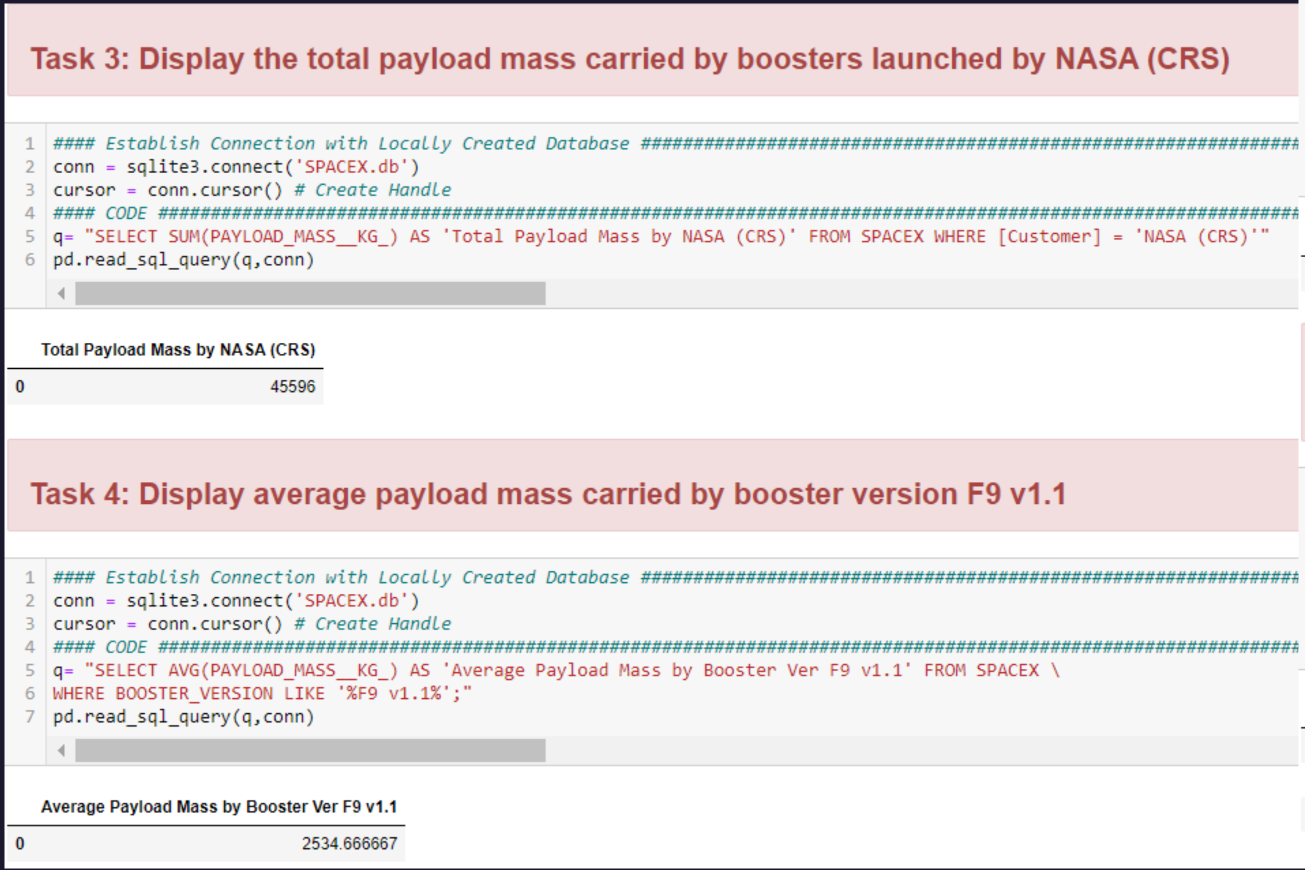
| | Date | Time (UTC) | Booster_Version | Launch_Site | Payload | PAYLOAD_MASS__KG_ | Orbit | Customer | Mission_Outcome | Landing _Outcome |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 04-06-2010 | 18:45:00 | F9 v1.0 B0003 | CCAFS LC-40 | Dragon Spacecraft Qualification Unit | 0 | LEO | SpaceX | Success | Failure (parachute) |
| 1 | 08-12-2010 | 15:43:00 | F9 v1.0 B0004 | CCAFS LC-40 | Dragon demo flight C1, two CubeSats, barrel of... | 0 | LEO (ISS) | NASA (COTS) NRO | Success | Failure (parachute) |
| 2 | 22-05-2012 | 07:44:00 | F9 v1.0 B0005 | CCAFS LC-40 | Dragon demo flight C2 | 525 | LEO (ISS) | NASA (COTS) | Success | No attempt |
| 3 | 08-10-2012 | 00:35:00 | F9 v1.0 B0006 | CCAFS LC-40 | SpaceX CRS-1 | 500 | LEO (ISS) | NASA (CRS) | Success | No attempt |
| 4 | 01-03-2013 | 15:10:00 | F9 v1.0 B0007 | CCAFS LC-40 | SpaceX CRS-2 | 677 | LEO (ISS) | NASA (CRS) | Success | No attempt |

## EDAwith SQL Results 2

**Tasks**

3) Total "Payload Mass" carried by Boosters is 45596
4) Average "Payload Mass" by Booster F9 v1.1 is 2535

5) First Successful Landing was in 2017
6) 4 successful Boosters between "Payload Mass" 4000 and 600



**Task 3: Display the total payload mass carried by boosters launched by NASA (CRS)**

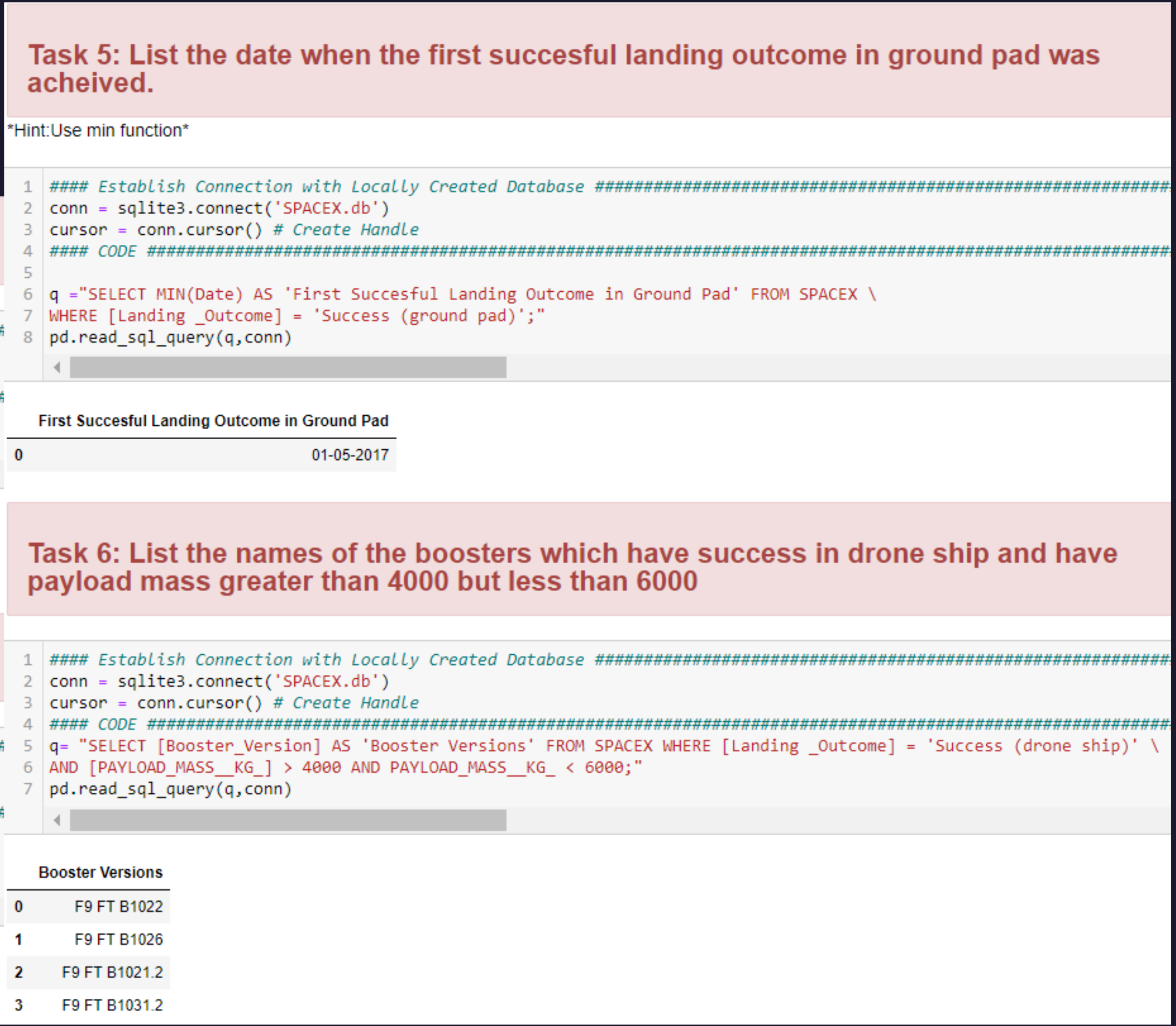
```
#### Establish Connection with Locally Created Database ##############################################################
conn = sqlite3.connect('SPACEX.db')
cursor = conn.cursor() # Create Handle
#### CODE ##################################################################################################
q= "SELECT SUM(PAYLOAD_MASS__KG_) AS 'Total Payload Mass by NASA (CRS)' FROM SPACEX WHERE [Customer] = 'NASA (CRS)'"
pd.read_sql_query(q,conn)
```

```
  Total Payload Mass by NASA (CRS)
0                            45596
```

**Task 4: Display average payload mass carried by booster version F9 v1.1**

```
#### Establish Connection with Locally Created Database ##############################################################
conn = sqlite3.connect('SPACEX.db')
cursor = conn.cursor() # Create Handle
#### CODE ##################################################################################################
q= "SELECT AVG(PAYLOAD_MASS__KG_) AS 'Average Payload Mass by Booster Ver F9 v1.1' FROM SPACEX \
WHERE BOOSTER_VERSION LIKE '%F9 v1.1%';"
pd.read_sql_query(q,conn)
```

```
  Average Payload Mass by Booster Ver F9 v1.1
0                                 2534.666667
```



**Task 5: List the date when the first succesful landing outcome in ground pad was acheived.**

*Hint:Use min function*

```
#### Establish Connection with Locally Created Database ##############################################################
conn = sqlite3.connect('SPACEX.db')
cursor = conn.cursor() # Create Handle
#### CODE ##################################################################################################

q ="SELECT MIN(Date) AS 'First Succesful Landing Outcome in Ground Pad' FROM SPACEX \
WHERE [Landing _Outcome] = 'Success (ground pad)';"
pd.read_sql_query(q,conn)
```

```
  First Succesful Landing Outcome in Ground Pad
0                                    01-05-2017
```

**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**

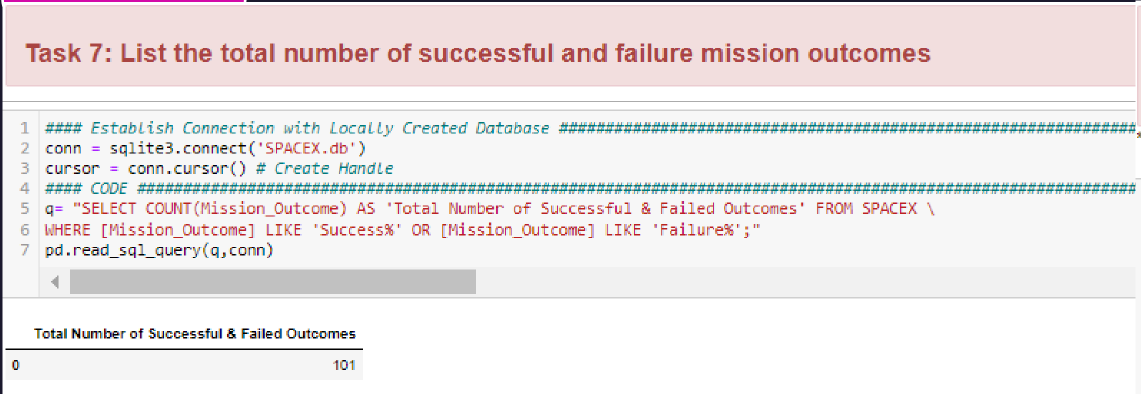
```
#### Establish Connection with Locally Created Database ##############################################################
conn = sqlite3.connect('SPACEX.db')
cursor = conn.cursor() # Create Handle
#### CODE ##################################################################################################
q= "SELECT [Booster_Version] AS 'Booster Versions' FROM SPACEX WHERE [Landing _Outcome] = 'Success (drone ship)' \
AND [PAYLOAD_MASS__KG_] > 4000 AND PAYLOAD_MASS__KG_ < 6000;"
pd.read_sql_query(q,conn)
```

```
  Booster Versions
0     F9 FT B1022
1     F9 FT B1026
2   F9 FT B1021.2
3   F9 FT B1031.2
```

# EDA with SQL Results 3

**Tasks**

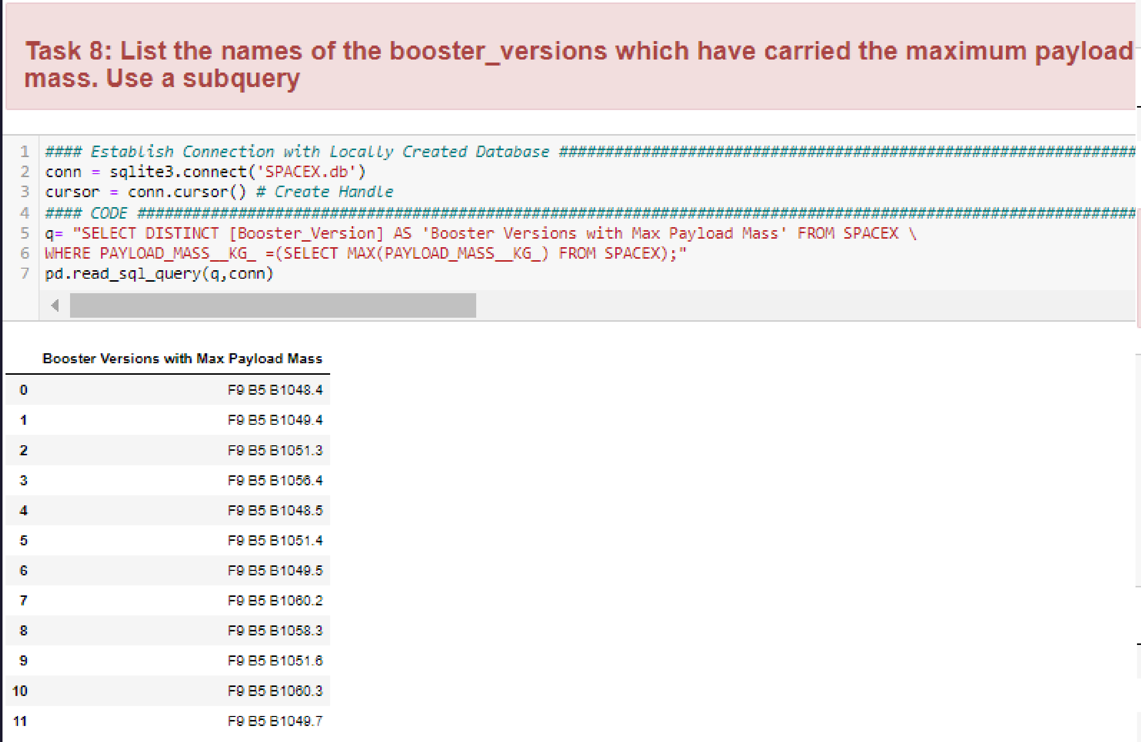
7) Total Successful & Failed Outcomes are 101
8) Sub-query returned 11 Boosters with Max "Payload Mass"
9) "Year 2015" Analysis displayed 2 failed landing outcomes
10) 34 Successful Outcomes between "2010-17"



**Task 7: List the total number of successful and failure mission outcomes**

```
#### Establish Connection with Locally Created Database ###############################################################
conn = sqlite3.connect('SPACEX.db')
cursor = conn.cursor() # Create Handle
#### CODE ##############################################################################################################
q= "SELECT COUNT(Mission_Outcome) AS 'Total Number of Successful & Failed Outcomes' FROM SPACEX \
WHERE [Mission_Outcome] LIKE 'Success%' OR [Mission_Outcome] LIKE 'Failure%';"
pd.read_sql_query(q,conn)
```

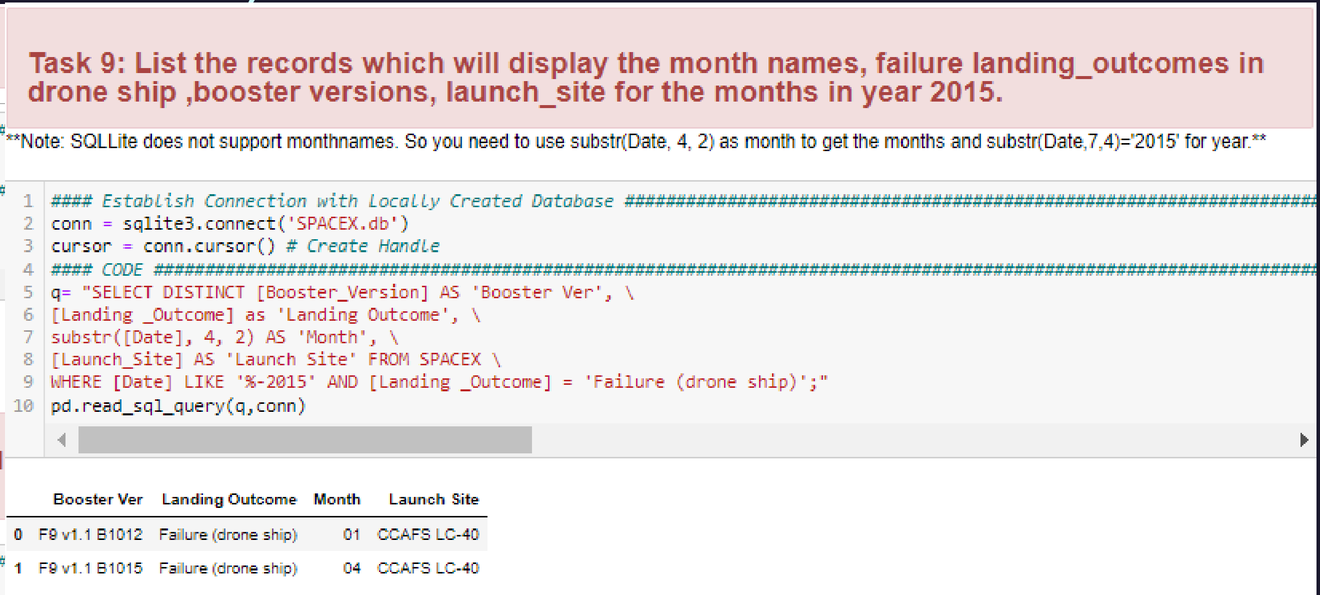
| | Total Number of Successful & Failed Outcomes |
|---|---|
| 0 | 101 |



**Task 8: List the names of the booster_versions which have carried the maximum payload mass. Use a subquery**

```
#### Establish Connection with Locally Created Database ###############################################################
conn = sqlite3.connect('SPACEX.db')
cursor = conn.cursor() # Create Handle
#### CODE ##############################################################################################################
q= "SELECT DISTINCT [Booster_Version] AS 'Booster Versions with Max Payload Mass' FROM SPACEX \
WHERE PAYLOAD_MASS__KG_ =(SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEX);"
pd.read_sql_query(q,conn)
```

| | Booster Versions with Max Payload Mass |
|---|---|
| 0 | F9 B5 B1048.4 |
| 1 | F9 B5 B1049.4 |
| 2 | F9 B5 B1051.3 |
| 3 | F9 B5 B1056.4 |
| 4 | F9 B5 B1048.5 |
| 5 | F9 B5 B1051.4 |
| 6 | F9 B5 B1049.5 |
| 7 | F9 B5 B1060.2 |
| 8 | F9 B5 B1058.3 |
| 9 | F9 B5 B1051.6 |
| 10 | F9 B5 B1060.3 |
| 11 | F9 B5 B1049.7 |

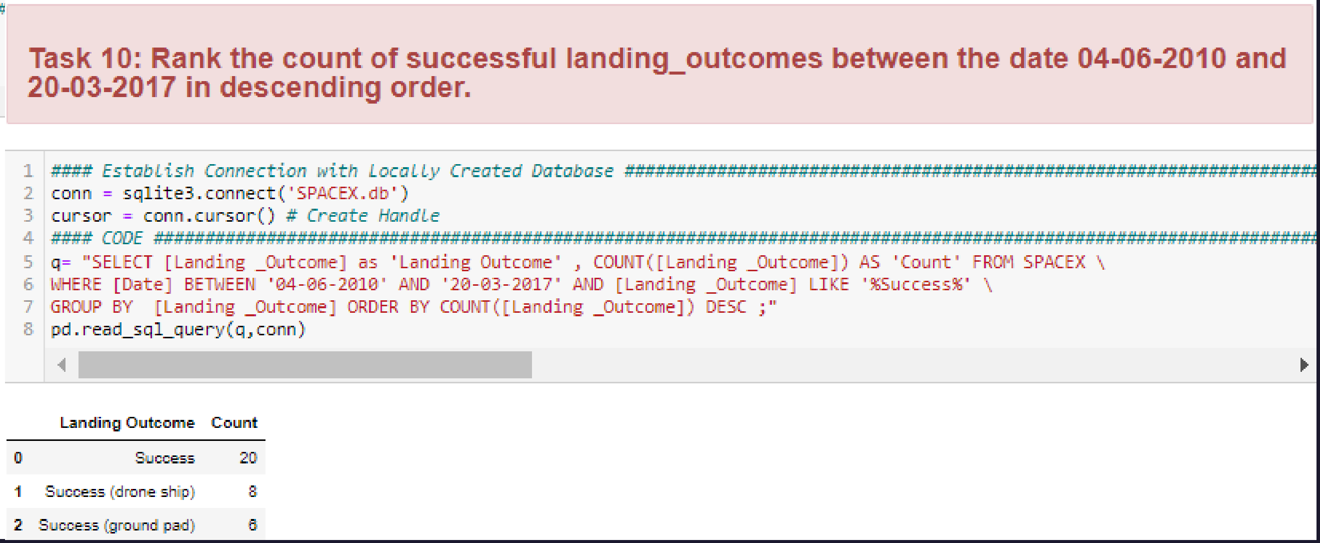


**Task 9: List the records which will display the month names, failure landing_outcomes in drone ship ,booster versions, launch_site for the months in year 2015.**

**Note: SQLLite does not support monthnames. So you need to use substr(Date, 4, 2) as month to get the months and substr(Date,7,4)='2015' for year.**

```
#### Establish Connection with Locally Created Database ###############################################################
conn = sqlite3.connect('SPACEX.db')
cursor = conn.cursor() # Create Handle
#### CODE ##############################################################################################################
q= "SELECT DISTINCT [Booster_Version] AS 'Booster Ver', \
[Landing _Outcome] as 'Landing Outcome', \
substr([Date], 4, 2) AS 'Month', \
[Launch_Site] AS 'Launch Site' FROM SPACEX \
WHERE [Date] LIKE '%-2015' AND [Landing _Outcome] = 'Failure (drone ship)';"
pd.read_sql_query(q,conn)
```

| | Booster Ver | Landing Outcome | Month | Launch Site |
|---|---|---|---|---|
| 0 | F9 v1.1 B1012 | Failure (drone ship) | 01 | CCAFS LC-40 |
| 1 | F9 v1.1 B1015 | Failure (drone ship) | 04 | CCAFS LC-40 |



**Task 10: Rank the count of successful landing_outcomes between the date 04-06-2010 and 20-03-2017 in descending order.**

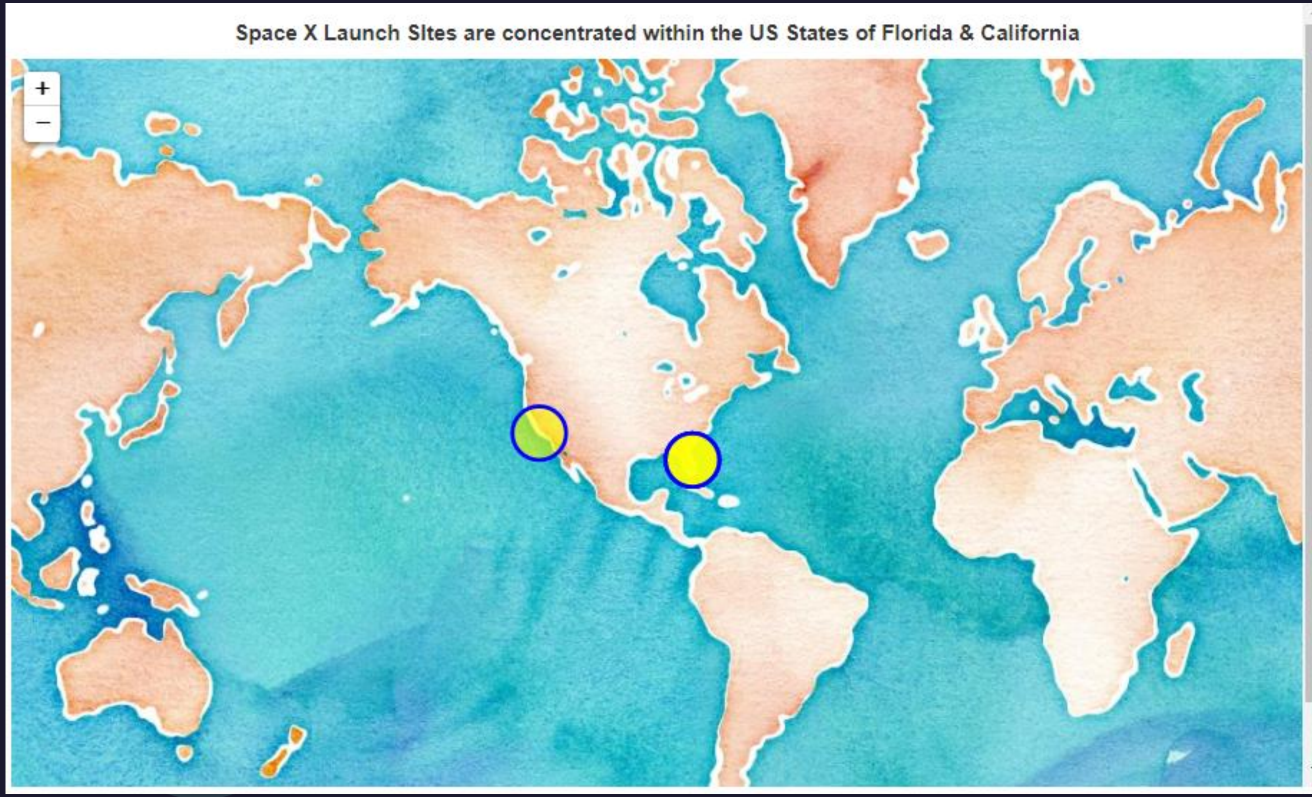
```
#### Establish Connection with Locally Created Database ###############################################################
conn = sqlite3.connect('SPACEX.db')
cursor = conn.cursor() # Create Handle
#### CODE ##############################################################################################################
q= "SELECT [Landing _Outcome] as 'Landing Outcome' , COUNT([Landing _Outcome]) AS 'Count' FROM SPACEX \
WHERE [Date] BETWEEN '04-06-2010' AND '20-03-2017' AND [Landing _Outcome] LIKE '%Success%' \
GROUP BY  [Landing _Outcome] ORDER BY COUNT([Landing _Outcome]) DESC ;"
pd.read_sql_query(q,conn)
```

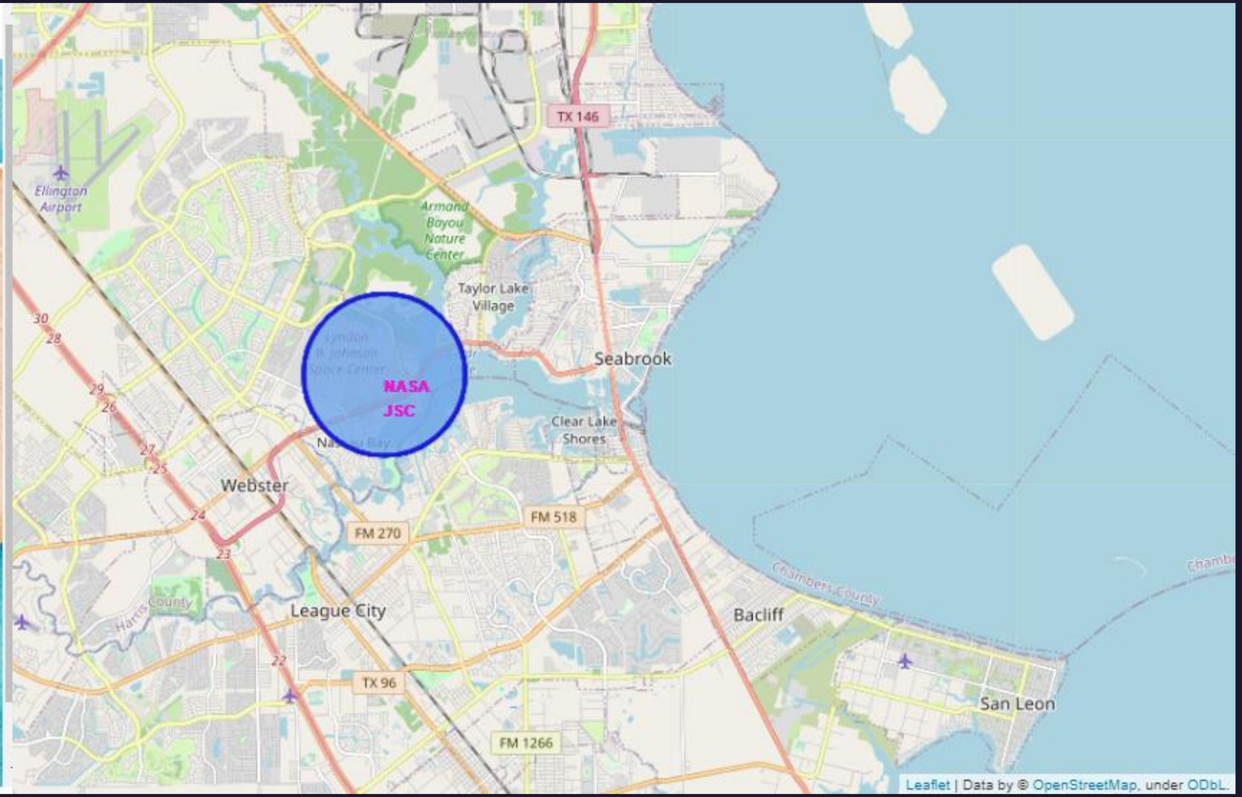
| | Landing Outcome | Count |
|---|---|---|
| 0 | Success | 20 |
| 1 | Success (drone ship) | 8 |
| 2 | Success (ground pad) | 6 |

# Interactive Visual Analysis Folium Results 1

Bonus Task: Displaying Concentration of Space X Launch Sites

Task 1: Displaying 'NASA Johnson Space Center'

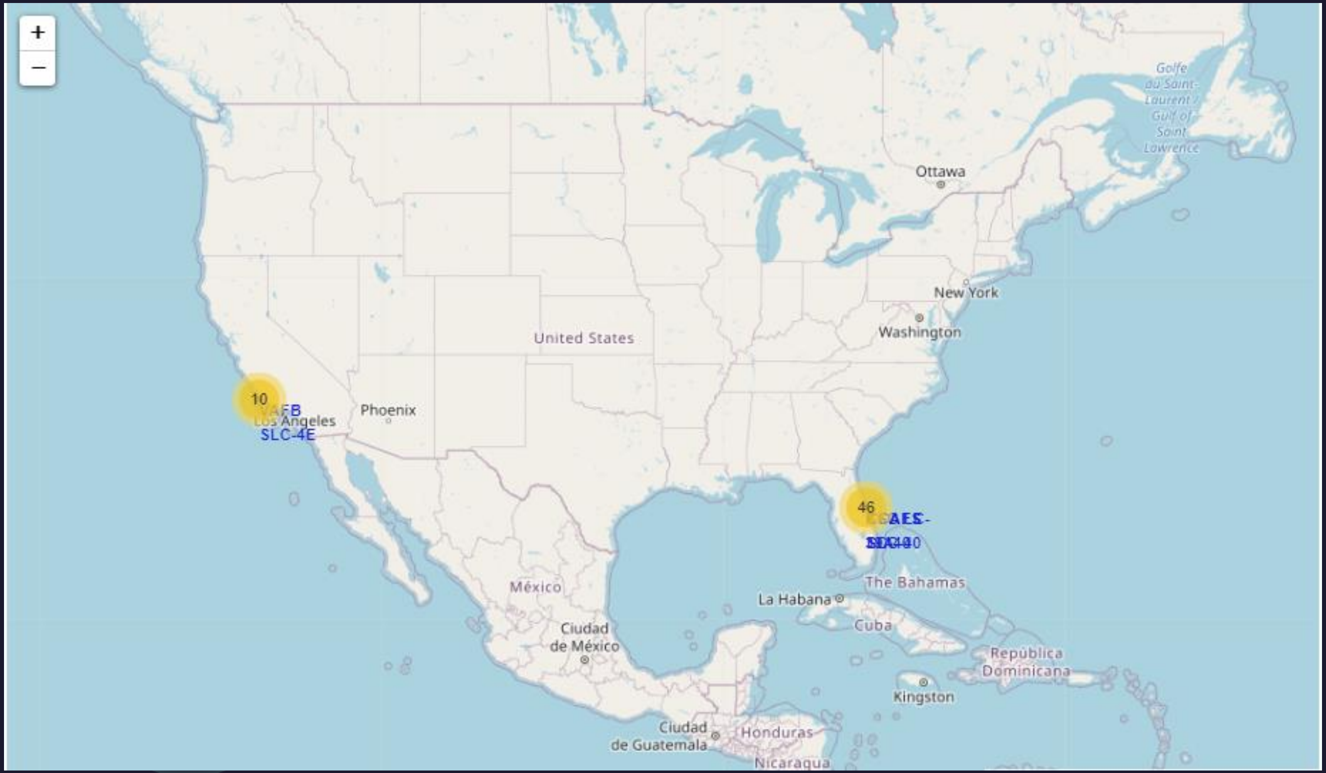

Space X Launch Sites are concentrated within the US States of Florida & California
+
−

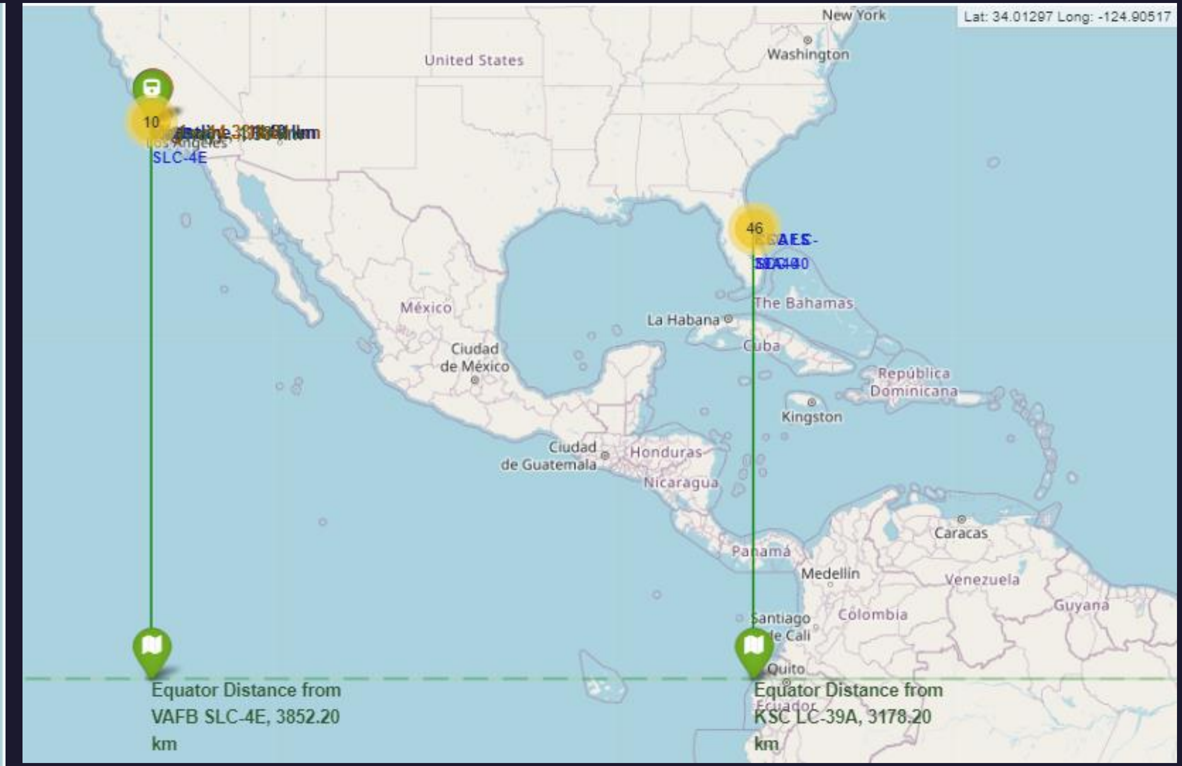


# Interactive Visual Analysis Folium Results 2

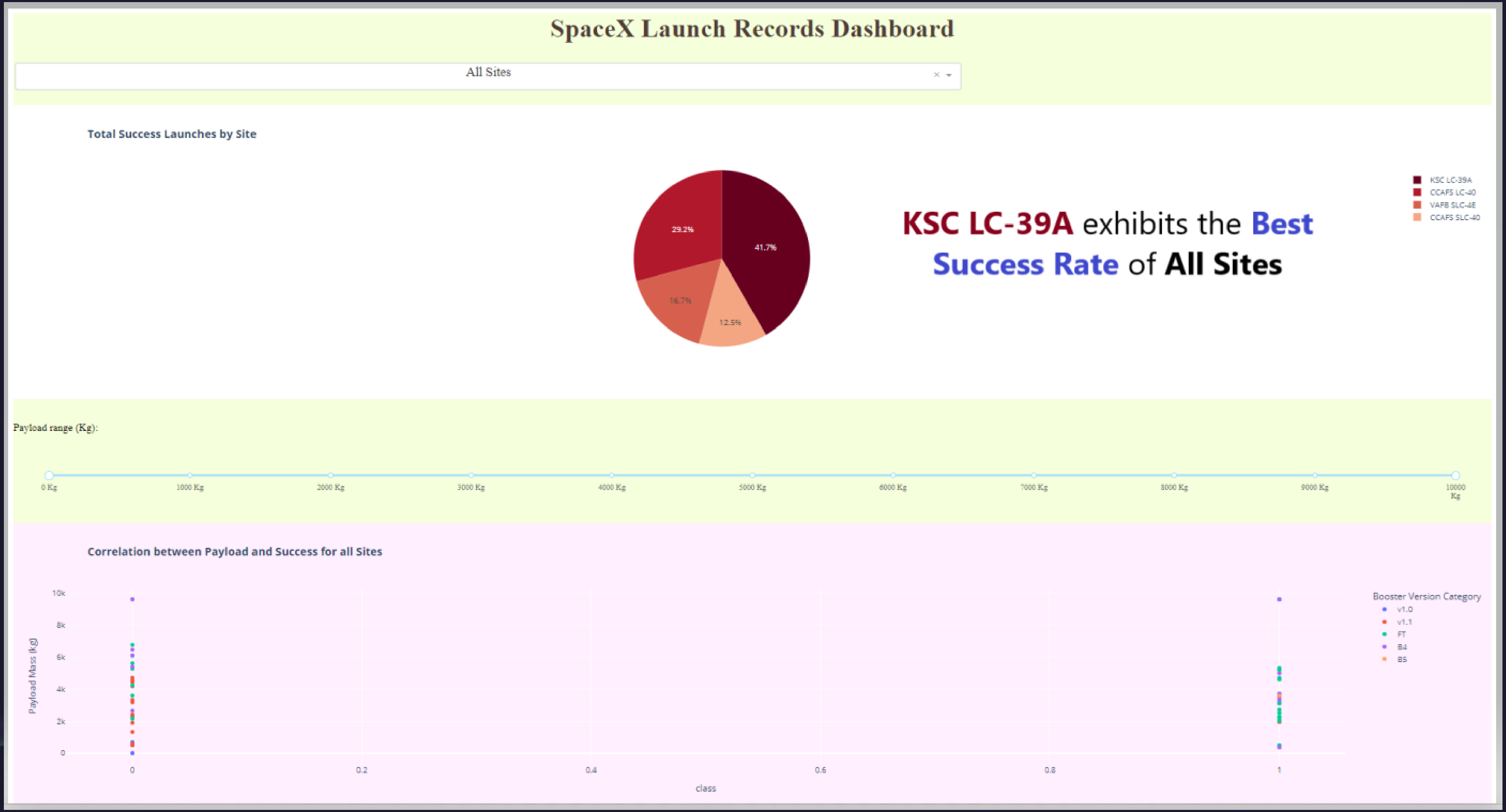
Task 2: Displaying Successful/Failed Launches

Task 3: Displaying Distances between Launch Sites to its Proximities





## Interactive Visual Analysis: Plotly Dash All Sites Results



SpaceX Launch Records Dashboard

All Sites

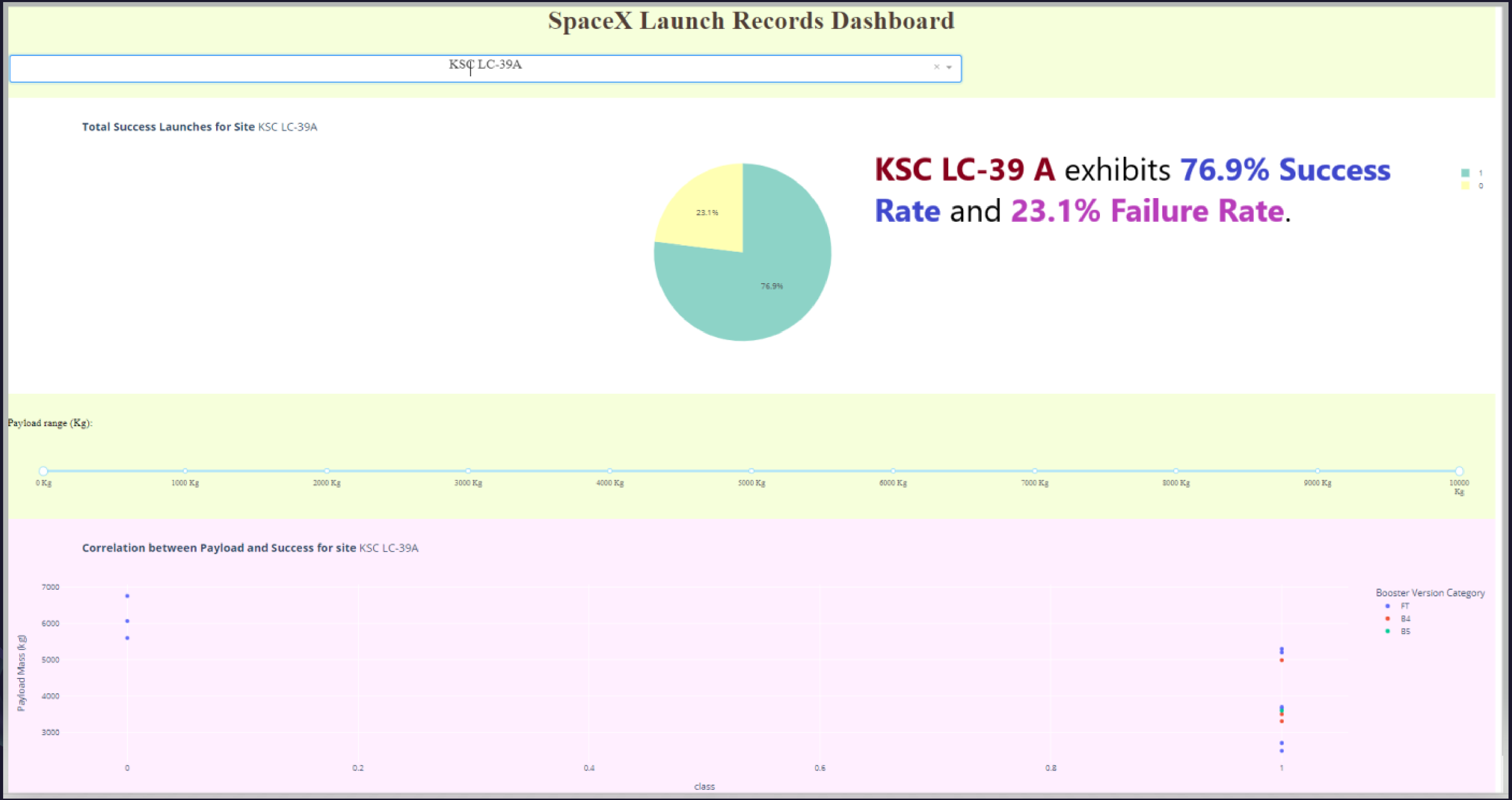
Total Success Launches by Site

**KSC LC-39A** exhibits the **Best Success Rate** of **All Sites**

Payload range (Kg):

Correlation between Payload and Success for all Sites

## Interactive Visual Analysis: Plotly Dash KSC LC-39 A Site Results



SpaceX Launch Records Dashboard

KSC LC-39A

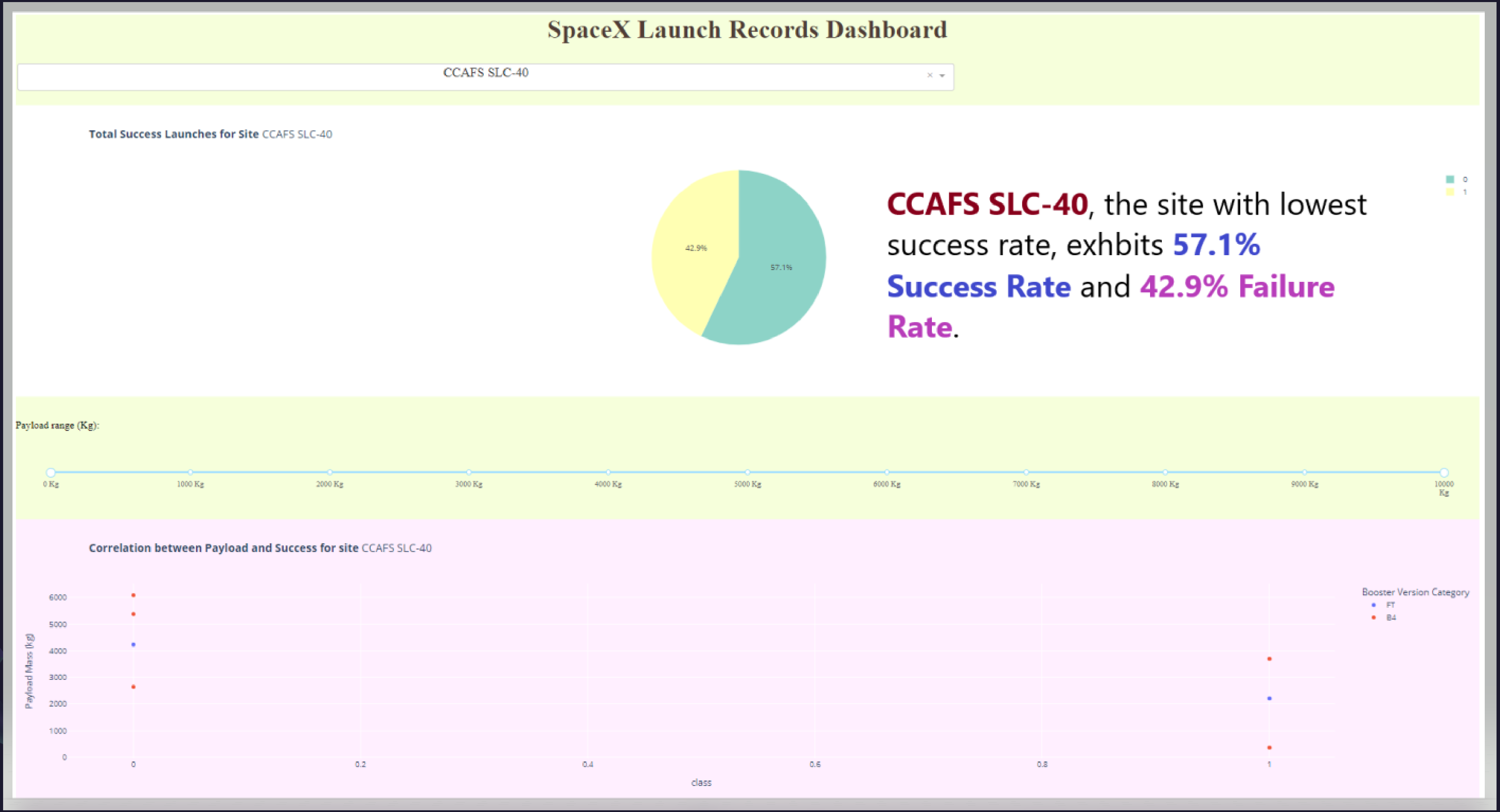
Total Success Launches for Site KSC LC-39A

**KSC LC-39 A** exhibits **76.9% Success Rate** and **23.1% Failure Rate**.

Payload range (Kg):

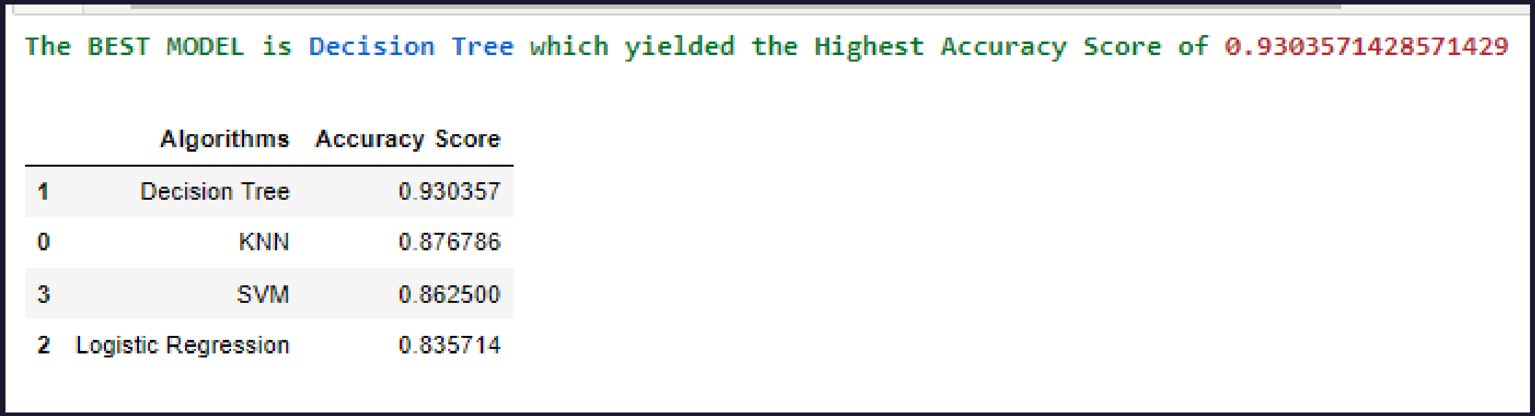
Correlation between Payload and Success for site KSC LC-39A

## Interactive Visual Analysis: Plotly Dash CCAFS SLC-40 Site Results



**CCAFS SLC-40**, the site with lowest success rate, exhbits **57.1% Success Rate** and **42.9% Failure Rate**.

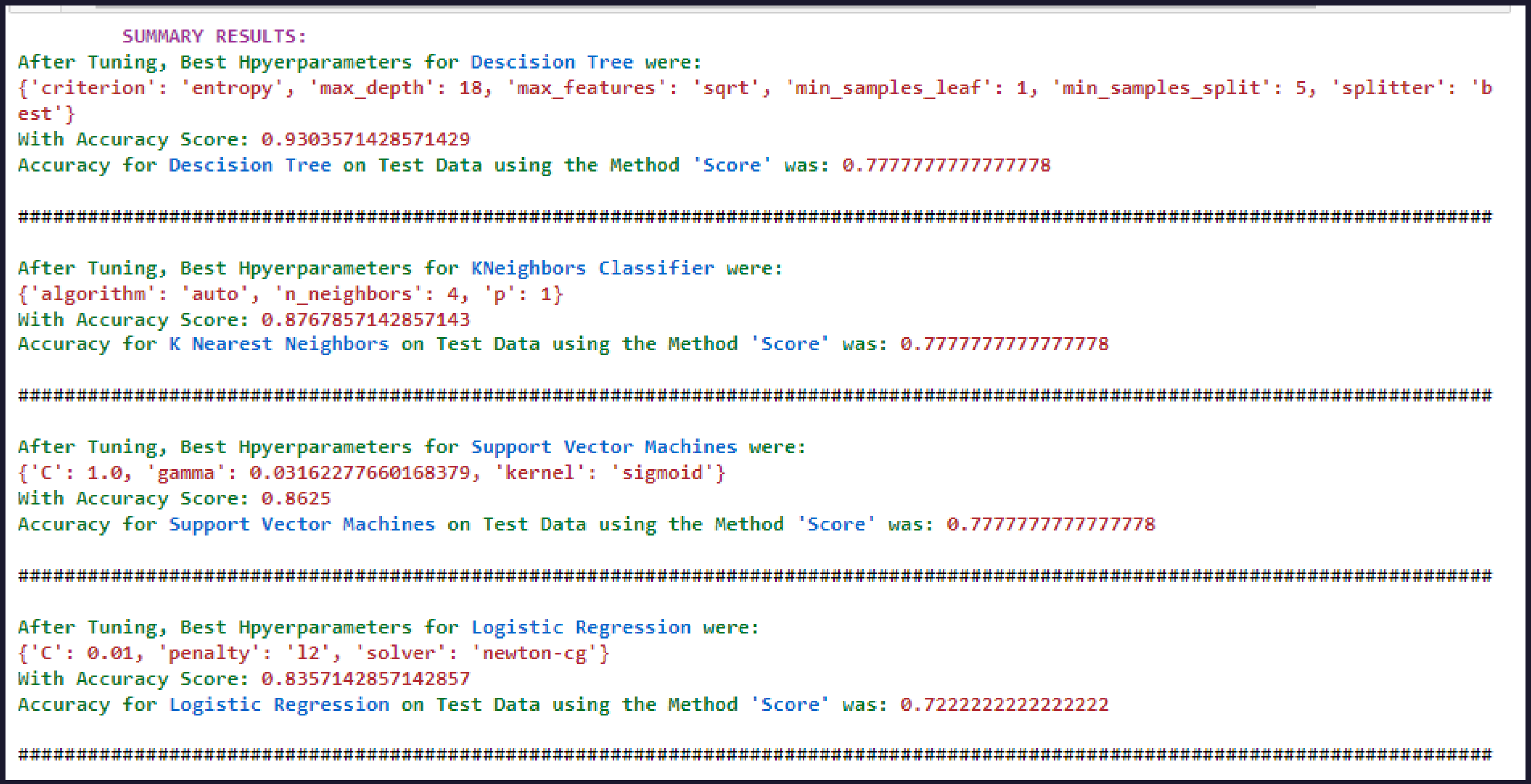
# Predictive Analysis Results



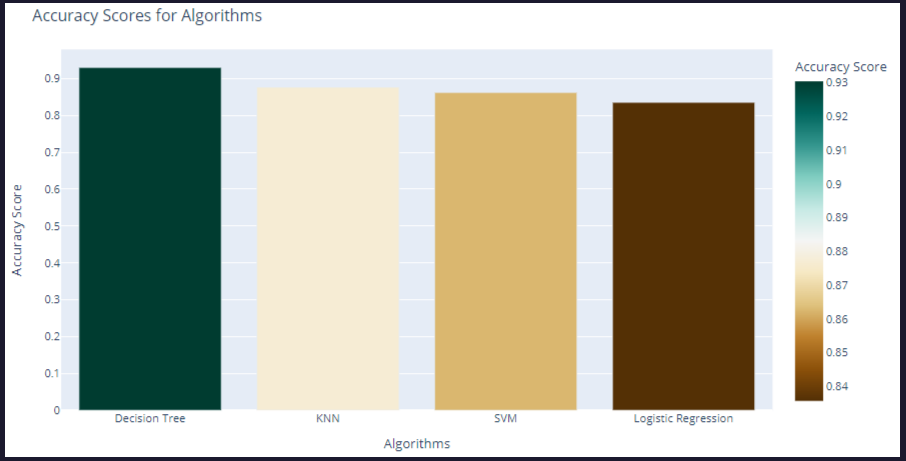
```
The BEST MODEL is Decision Tree which yielded the Highest Accuracy Score of 0.9303571428571429
```

| | Algorithms | Accuracy Score |
|---|---|---|
| 1 | Decision Tree | 0.930357 |
| 0 | KNN | 0.876786 |
| 3 | SVM | 0.862500 |
| 2 | Logistic Regression | 0.835714 |

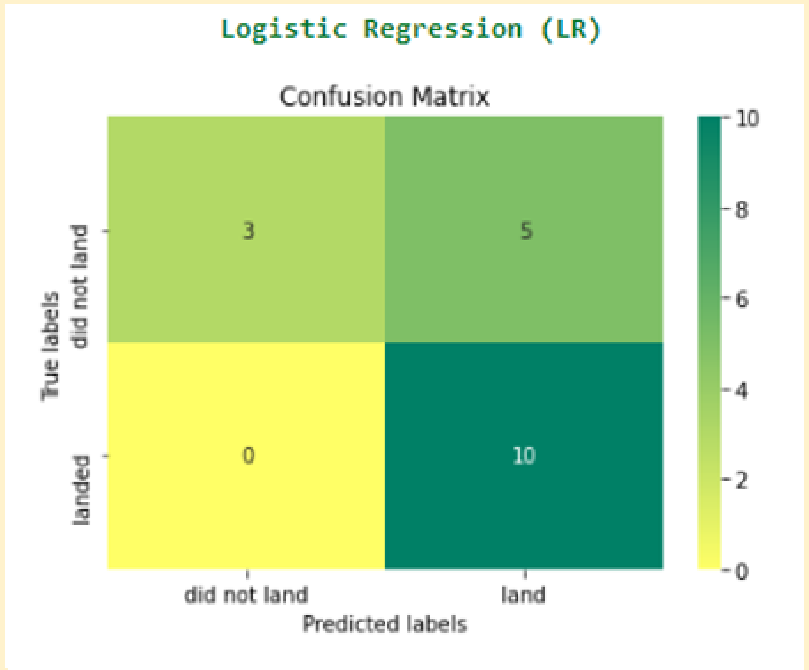
With the Highest Accuracy Score on Train Data, Decision Tree is the winner among all models.

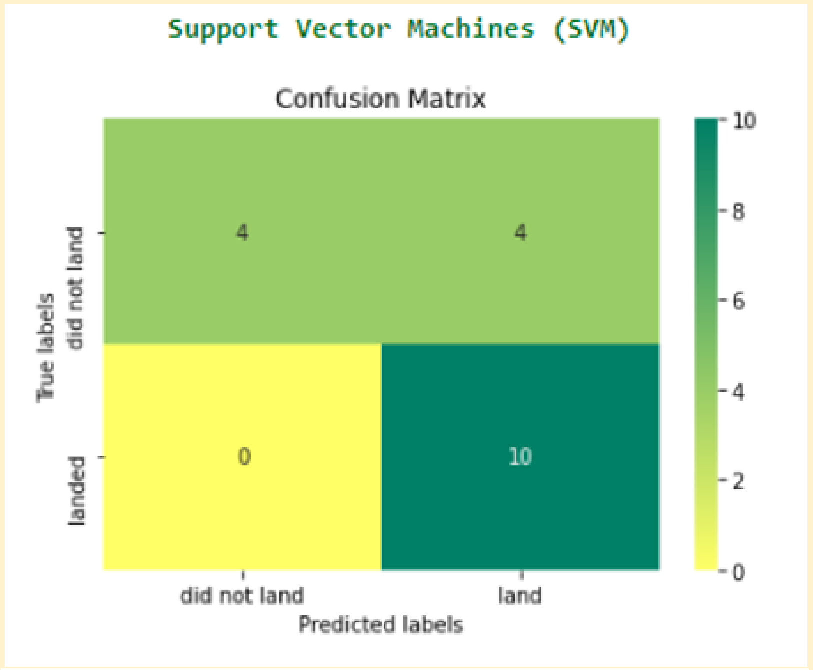


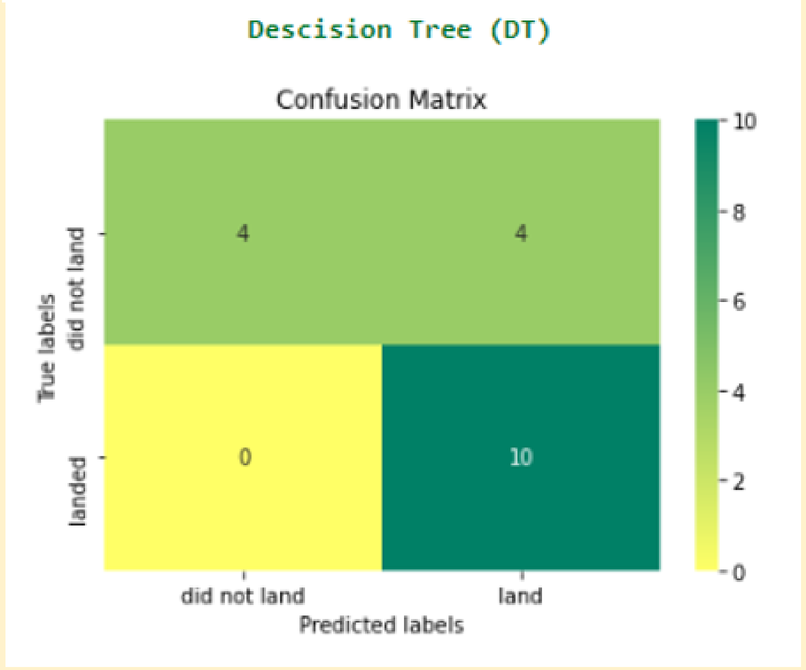
```
         SUMMARY RESULTS:
After Tuning, Best Hpyerparameters for Descision Tree were:
{'criterion': 'entropy', 'max_depth': 18, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 5, 'splitter': 'b
est'}
With Accuracy Score: 0.9303571428571429
Accuracy for Descision Tree on Test Data using the Method 'Score' was: 0.7777777777777778

##################################################################################################################################

After Tuning, Best Hpyerparameters for KNeighbors Classifier were:
{'algorithm': 'auto', 'n_neighbors': 4, 'p': 1}
With Accuracy Score: 0.8767857142857143
Accuracy for K Nearest Neighbors on Test Data using the Method 'Score' was: 0.7777777777777778

##################################################################################################################################

After Tuning, Best Hpyerparameters for Support Vector Machines were:
{'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
With Accuracy Score: 0.8625
Accuracy for Support Vector Machines on Test Data using the Method 'Score' was: 0.7777777777777778

##################################################################################################################################

After Tuning, Best Hpyerparameters for Logistic Regression were:
{'C': 0.01, 'penalty': 'l2', 'solver': 'newton-cg'}
With Accuracy Score: 0.8357142857142857
Accuracy for Logistic Regression on Test Data using the Method 'Score' was: 0.7222222222222222

##################################################################################################################################
```

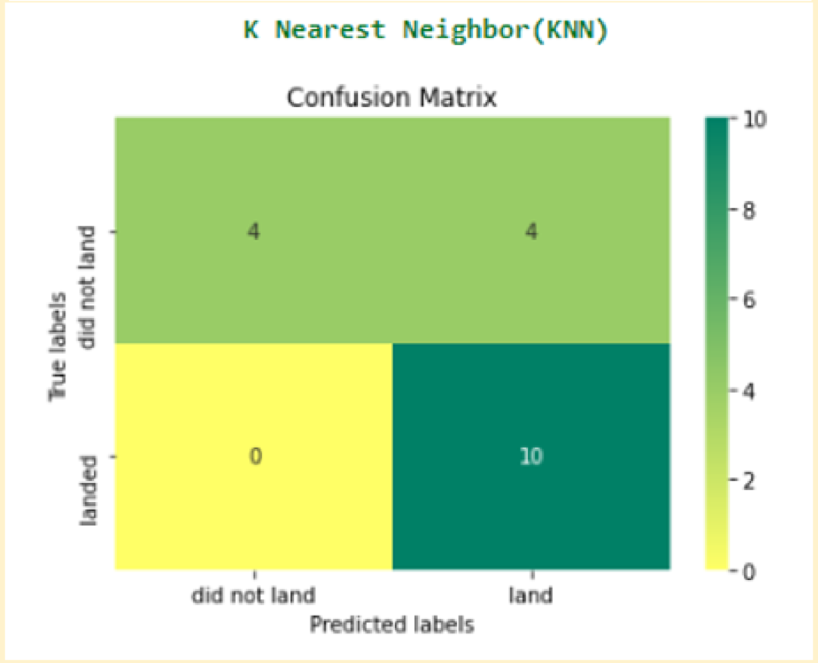


## Predictive Analysis Results







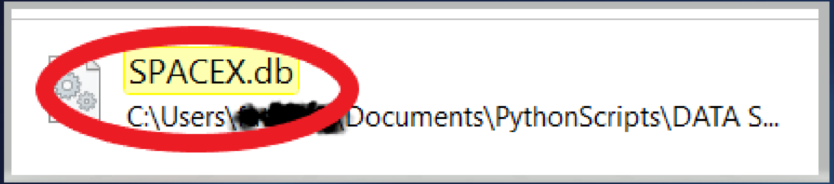


After tuning for best hyper-parameters, accuracy on Test Data using method score was the same for all models except Logistic Regression (LR). This is why, the confusion matrix for LR is different while for the rest it is the same

# Application of Creativity Beyond Template

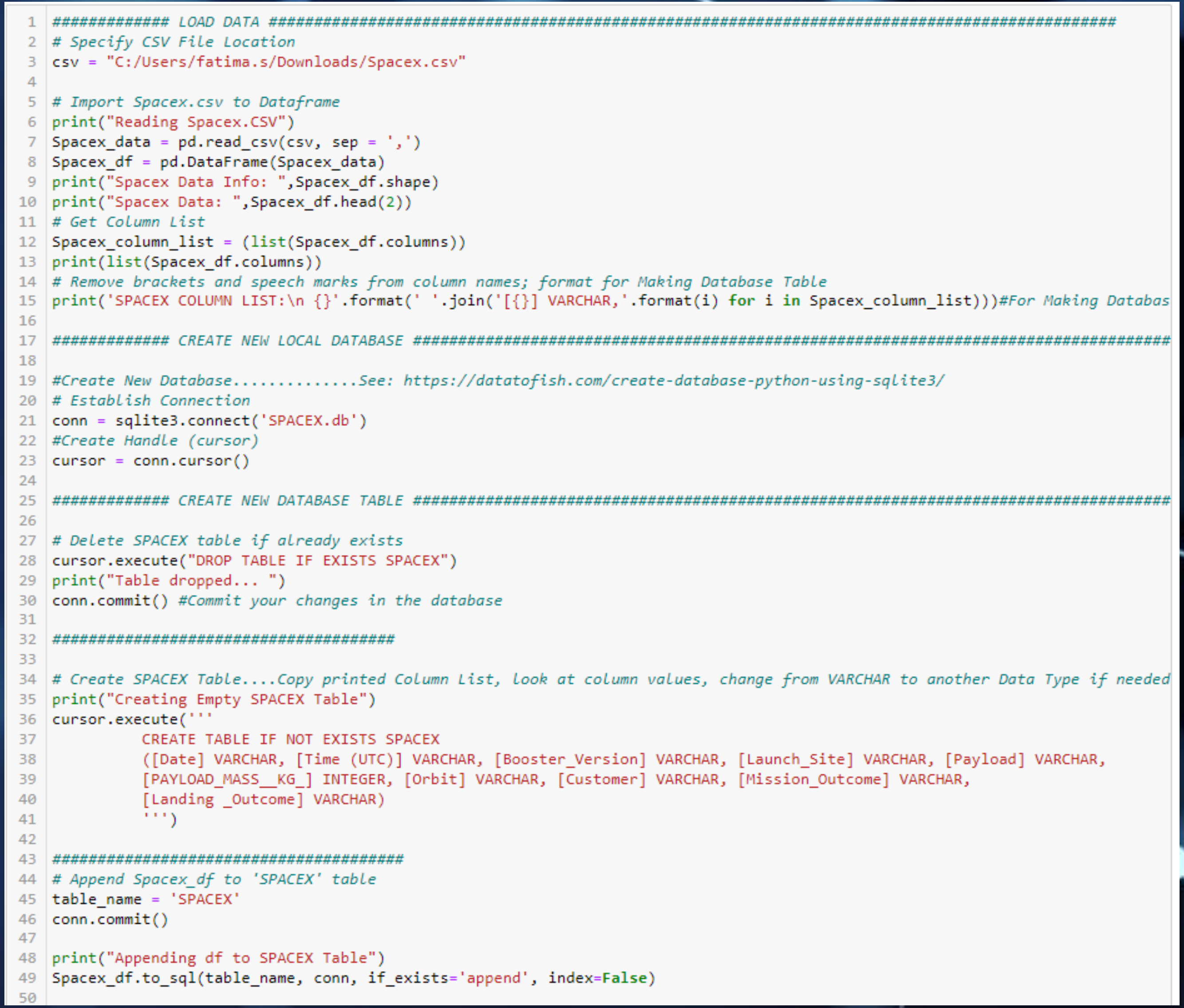
## Exploratory Data Analysis

- **SQL:** Local Database Creation was achieved for subsequent SQL EDA



SPACEX.db

C:\Users\[redacted]Documents\PythonScripts\DATA S...



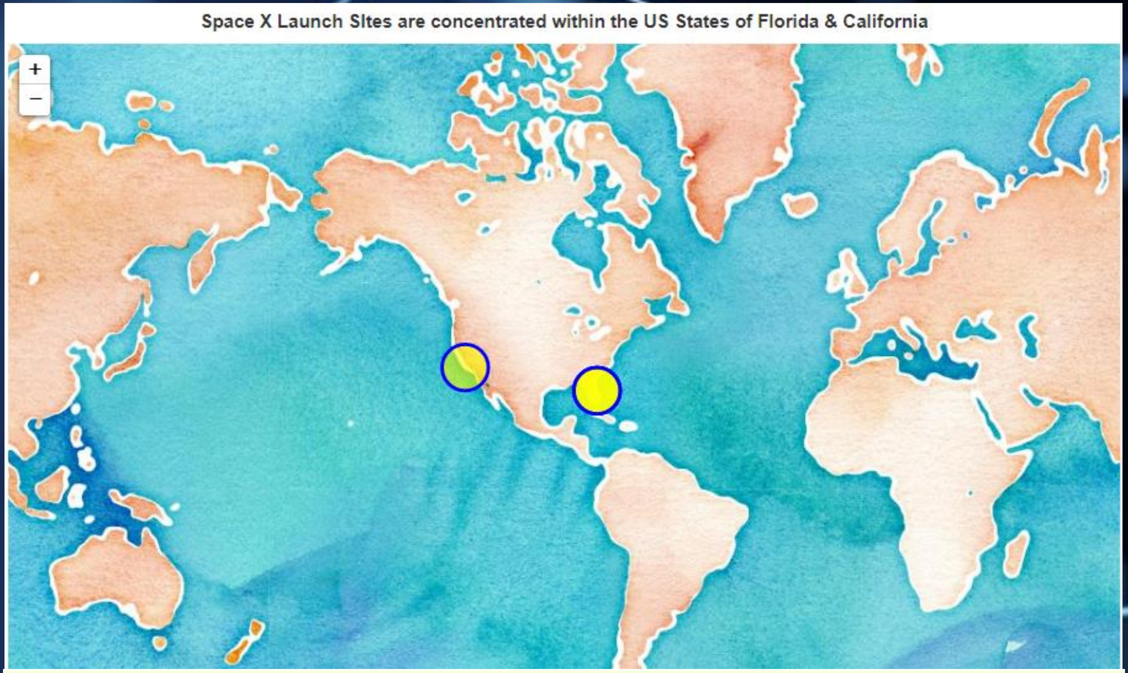
```
############# LOAD DATA ############################################################################################
# Specify CSV File Location
csv = "C:/Users/fatima.s/Downloads/Spacex.csv"

# Import Spacex.csv to Dataframe
print("Reading Spacex.CSV")
Spacex_data = pd.read_csv(csv, sep = ',')
Spacex_df = pd.DataFrame(Spacex_data)
print("Spacex Data Info: ",Spacex_df.shape)
print("Spacex Data: ",Spacex_df.head(2))
# Get Column List
Spacex_column_list = (list(Spacex_df.columns))
print(list(Spacex_df.columns))
# Remove brackets and speech marks from column names; format for Making Database Table
print('SPACEX COLUMN LIST:\n {}'.format(' '.join('[{}] VARCHAR,'.format(i) for i in Spacex_column_list))#For Making Databas

############# CREATE NEW LOCAL DATABASE ##############################################################################

#Create New Database.............See: https://datatofish.com/create-database-python-using-sqlite3/
# Establish Connection
conn = sqlite3.connect('SPACEX.db')
#Create Handle (cursor)
cursor = conn.cursor()

############# CREATE NEW DATABASE TABLE ##############################################################################

# Delete SPACEX table if already exists
cursor.execute("DROP TABLE IF EXISTS SPACEX")
print("Table dropped... ")
conn.commit() #Commit your changes in the database

######################################

# Create SPACEX Table....Copy printed Column List, look at column values, change from VARCHAR to another Data Type if needed
print("Creating Empty SPACEX Table")
cursor.execute('''
          CREATE TABLE IF NOT EXISTS SPACEX
          ([Date] VARCHAR, [Time (UTC)] VARCHAR, [Booster_Version] VARCHAR, [Launch_Site] VARCHAR, [Payload] VARCHAR,
          [PAYLOAD_MASS__KG_] INTEGER, [Orbit] VARCHAR, [Customer] VARCHAR, [Mission_Outcome] VARCHAR,
          [Landing _Outcome] VARCHAR)
          ''')

#######################################
# Append Spacex_df to 'SPACEX' table
table_name = 'SPACEX'
conn.commit()

print("Appending df to SPACEX Table")
Spacex_df.to_sql(table_name, conn, if_exists='append', index=False)

```

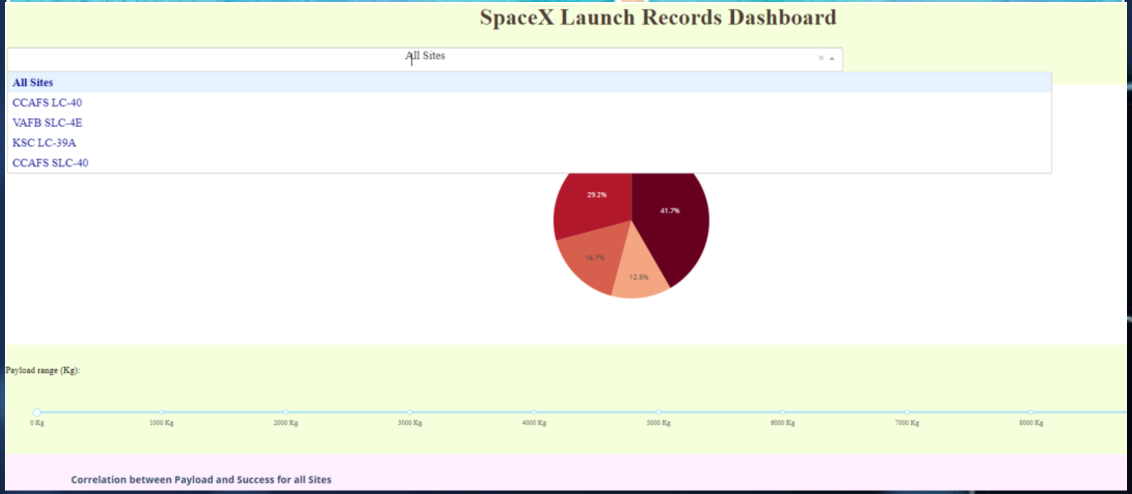
# Application of Creativity Beyond Template

## Interactive Visual Analytics

- **Folium:** Reverse Geo-coding to find State names with Launch Site concentration

- **Plotly Dash:** Colored dropdown list, app background coloring & pie chart color change


Space X Launch SItes are concentrated within the US States of Florida & California
+
−

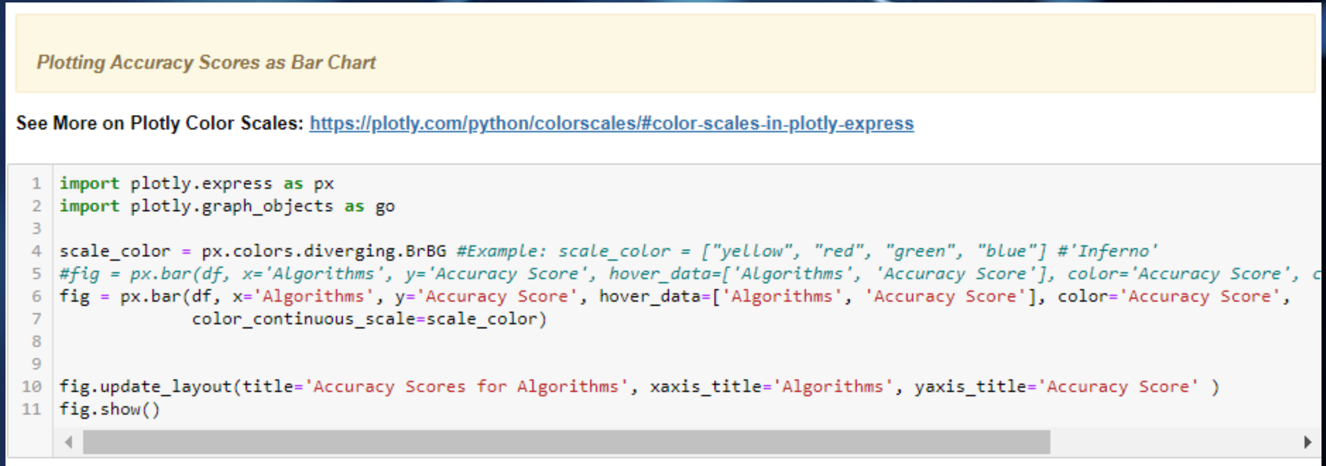

SpaceX Launch Records Dashboard

Correlation between Payload and Success for all Sites

# Application of Creativity Beyond Template

**Predictive Analysis:**

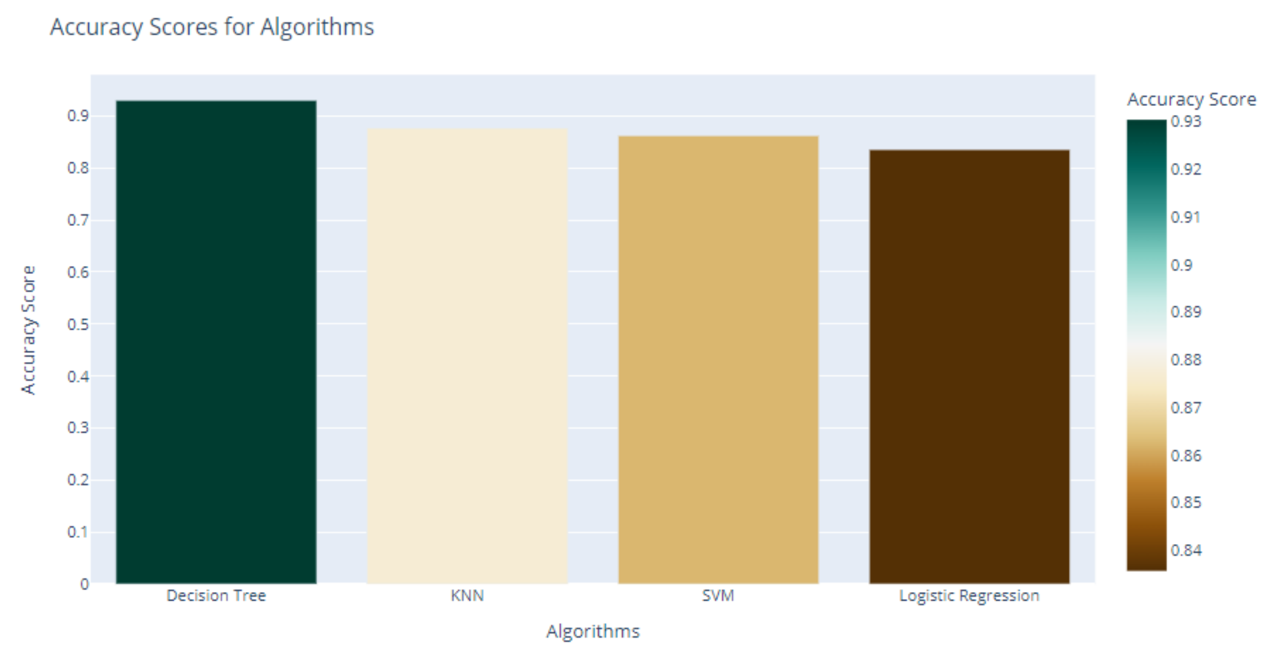
- **Plotting of Accuracy Scores as Bar Chart to furnish visual insight of modelling accuracy results**



*Plotting Accuracy Scores as Bar Chart*

**See More on Plotly Color Scales:** https://plotly.com/python/colorscales/#color-scales-in-plotly-express

```
import plotly.express as px
import plotly.graph_objects as go

scale_color = px.colors.diverging.BrBG #Example: scale_color = ["yellow", "red", "green", "blue"] #'Inferno'
#fig = px.bar(df, x='Algorithms', y='Accuracy Score', hover_data=['Algorithms', 'Accuracy Score'], color='Accuracy Score', c
fig = px.bar(df, x='Algorithms', y='Accuracy Score', hover_data=['Algorithms', 'Accuracy Score'], color='Accuracy Score',
             color_continuous_scale=scale_color)


fig.update_layout(title='Accuracy Scores for Algorithms', xaxis_title='Algorithms', yaxis_title='Accuracy Score' )
fig.show()
```



THANK YOU