IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Maria Carolina Mertens
27 October 2025

Outline

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

Executive Summary

This project predicts the successful landing of SpacesX Falcon 9 first stage by evaluating machine learning models. Predicting successful landing is important as this would mean reusing the first stage thus, saving millions of dollars for space travel.

By collecting historical launch data from SpaceX REST API and webscraping related Wikipages, relevant data were made available for preprocessing, exploratory data analysis (EDA) and feature engineering. Multiple classifiers including Logistic Regression (LogRes), Decision Trees (DT), Support Vector Machine (SVM), and K-Nearest Neighbors (KNN) were used to train and test the data for cross-validation and hyperparameter tuning.

Although the accuracy on the test data across all classification algorithms obtained the same result, the Decision Tree Classifier's best parameters produced the best score in accuracy. Critical factors impacting the results include Payload Mass, Orbit Type, Launch Site and Flight Number.

Introduction

The commercial space industry is rapidly lowering the barrier to space travel, with **SpaceX** leading the way in cost-effective launch solutions. The dramatic difference in pricing—where competing providers charge an estimated **$165 million** per launch, while a Falcon 9 launch by SpaceX costs approximately **$62 million**—is primarily attributed to their innovative **reusable first-stage rocket technology**.

The ability to successfully land and reuse the first stage is the key financial determinant of a SpaceX launch's cost, saving millions of dollars with each recovery. Therefore, predicting this success is crucial for both operational planning and cost estimation.

Section 1

Methodology

Methodology

- Data collection methodology:
  - Request historical data from SpaceX API. The response content was decoded and turned into a Pandas dataframe which was then filtered to keep only Falcon 9 launches.
  - More data were scraped from a snapshot of the List of Falcon 9 and Falcon Heavy Launches Wikipage to extract all relevant columns.
- Perform data wrangling
  - Missing values in Payload Mass were replaced by mean values. However, for the Landing Pad, 'None' values were retained to represent when landing pads were not used.
  - The number of launches on each sites and the number of occurrences in each orbit were counted. Mission outcomes were mapped as Class 1 for success and Class 0 for failure.
  - Orbits, Launch Site, Landing Pad and Serial were converted into numerical features using One-Hot encoding or dummy variables.

Methodology

- Perform exploratory data analysis (EDA) using visualization and SQL
  - Variables were analyzed using SQL queries to derive context on the landing success, and failure in different sites within certain periods.
  - Initial patterns and relationships between variables like Payload Mass and Flight Number were identified using scatterplots, bar chart and line chart to determine their impact on successful outcome.

Methodology

- Perform interactive visual analytics using Folium and Plotly Dash
  - With Folium, all Launch Sites were marked on a map showing their successful and failed launches. Markers were also created to show the distance of a site to its closest city, railway or highway.
  - With Plotly Dash, the total success launches were visualized through a pie chart and the correlation between Payload Mass and Class per Launch Site were analyzed through a scatterplot.

Methodology

- Perform predictive analysis using classification models
  - After data was standardized, feature set (X) and target variable (Y) were split into training and testing sets.
  - Multiple classifiers including Logistic Regression (LogReg), Decision Trees (DTC), Support Vector Machine (SVM), and K-Nearest Neighbors (KNN) were used for evaluation.
  - GridSearchCV with cross-validation (cv=10) was used for each classifier to find the best parameters.
  - Models were evaluated based on Accuracy and Confusion Matrix to understand the model's ability in classifying correctly the successful (True Positives) versus the unsuccessful (True Negatives) landings.

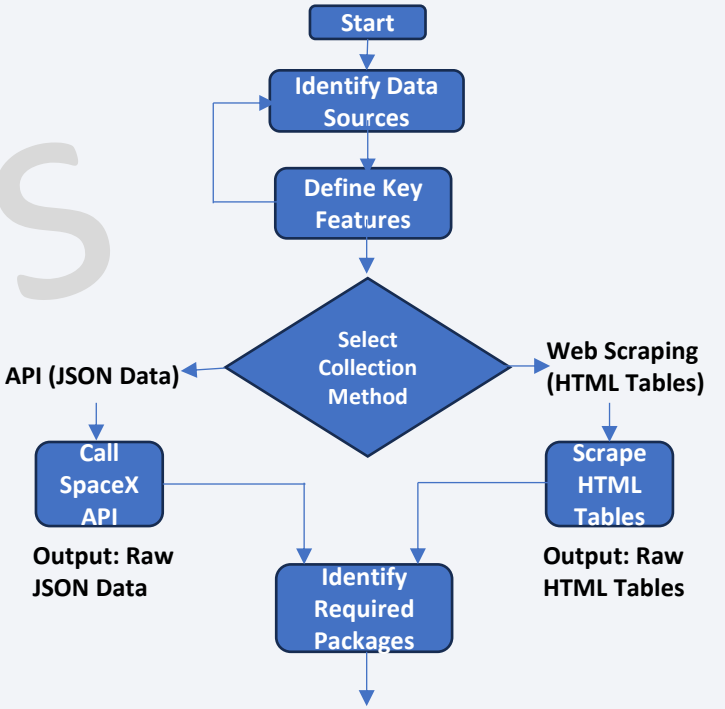
Data Collection

Identify Data Sources. This project used historical mission data from SpaceX Rest API and from snapshots on the List of Falcon 9 and Falcon Heavy Launches Wikipedia updated on 9th June 2021.

Define Key Features. Variables relevant to the landing outcome were identified which included rocket for booster names, launchpad for Launch Sites, payloads for Payload Mass and orbits, and cores for landing details.

Select Data Collection Method. **Call API** was used to get JSON data from SpaceX API and **web scraping** for HTML tables.

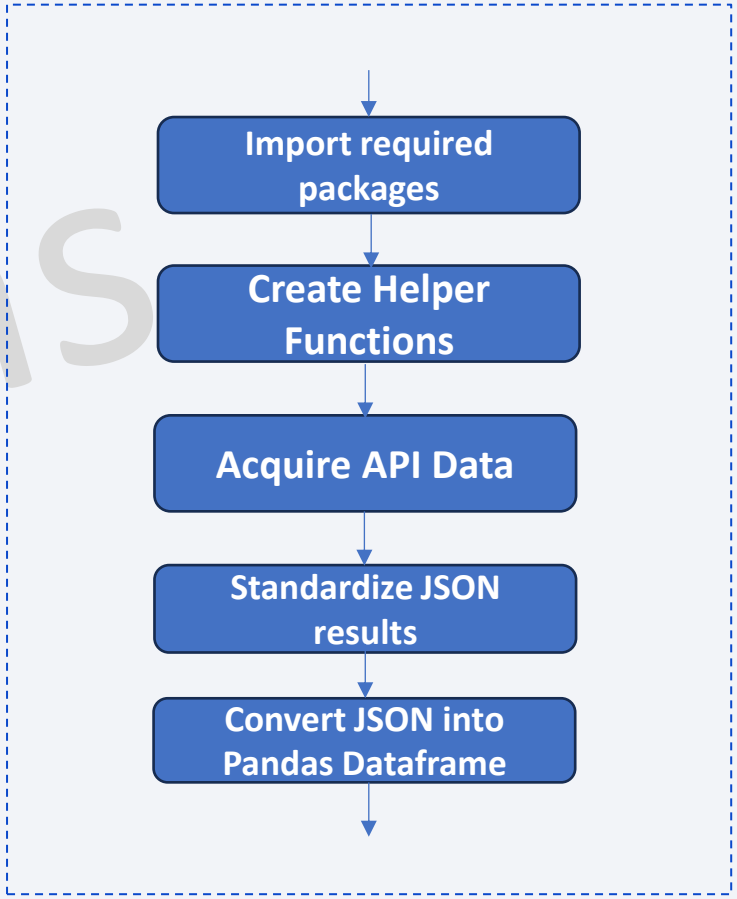
Identify required packages. Important packages not built into the core Python language were identified to access the external modules, libraries and tools required for the tasks.



Data Collection – SpaceX API

For SpaceX JSON data:

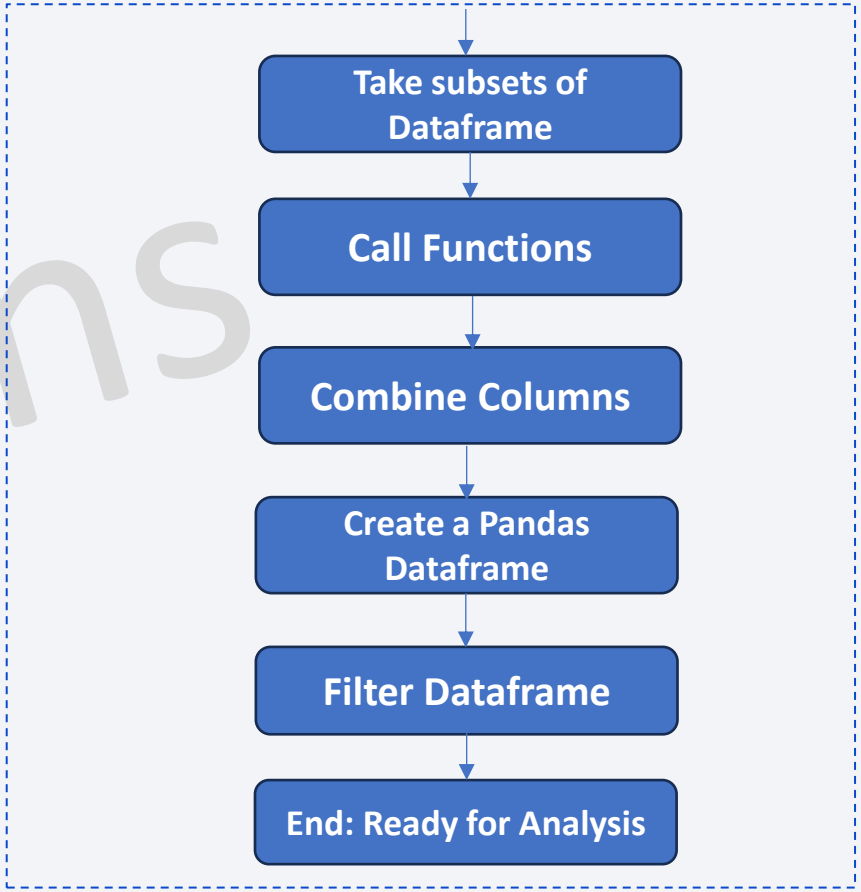
1. **Import required packages.** Important packages to be used were imported.
2. **Create Helper Functions.** A series of **def** functions were created to help extract Identification Numbers, Launch Sites, Payload Mass, Orbit Types and landings using **requests.get(), json()** and **append(response[])**.
3. **Acquire API Data.** Rocket launch data was requested using **requests.get()** from **URL** of SpaceX API.
4. **Standardize JSON results.** To make data reliable and uniform across requests, a s**tatic JSON URL** object was used.
5. **Convert JSON into Pandas Dataframe.** Parse JSON data using **.json()** and flatten it into a dataframe using **json_normalize().**



Data Collection – SpaceX API

6. **Take subsets of Dataframe.** Only important features of the dataframe which include rocket, payloads, launchpad and cores were extracted using **map(len), map(lambda), to_datetime** and **datetime.date(),** which were stored in lists.
7. **Call Functions.** Update the lists by calling the previously created functions**.**
8. **Combine Columns.** Create a dataset using the updated lists and combine them as a dictionary.
9. **Create a Pandas Dataframe.** From the newly created dictionary, create a dataframe using **pd.Dataframe()**
10. **Filter Dataframe.** Keep only Falcon 9 and remove the other Falcon launches then save the new dataframe.

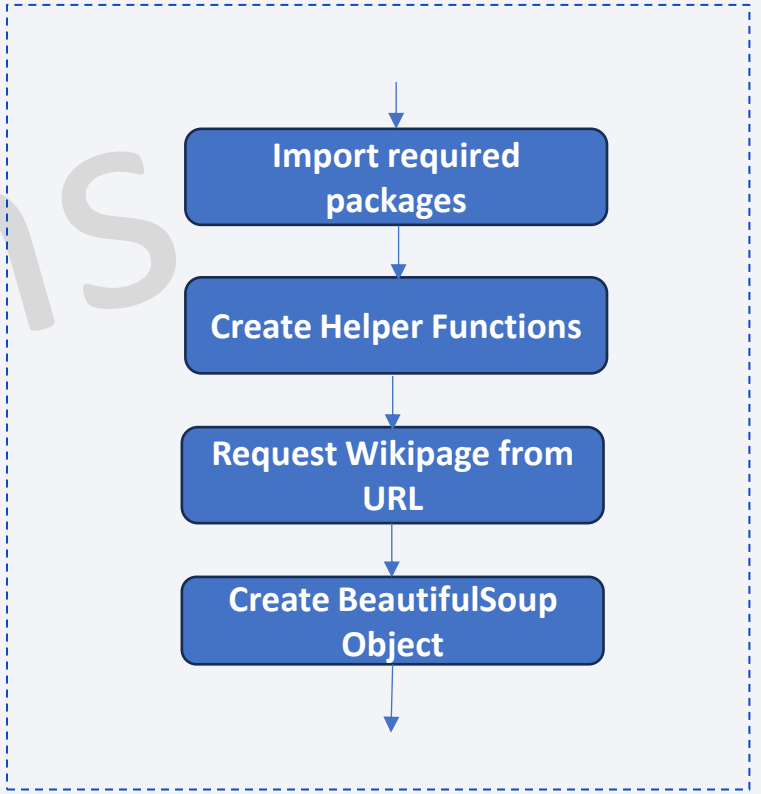
GitHub URL: Carolprojects/jupyter-labs-spacex-data-collection-api.ipynb at 69ab9788cc4df9cfc85c1c6b2c27956cb296b275 · CarolaMertens/Carolprojects



Data Collection - Scraping

For Falcon Launches Wikipedia

1. **Import required packages.** Important packages to be used were imported.
2. **Create Helper Functions.** A series of **def** functions were created to help extract identification numbers, date-times, booster version, landing status, Payload Mass and column headers using strip(), join(), find(), .strings, .text, .br, .a, .sup, extract(), .contents and isdigit().
3. **Request Wikipage from URL.** Perform **HTTP Get** method to **request** a **response** using **requests.get().**
4. **Create BeautifulSoup Object**. To extract data from HTML, create a **BeautifulSoup** object with **html.parser.** Then print title using **.title.**



Data Collection - Scraping

5. **Extract Column Names from HTML Table**. Use **find_all, len(), append()** and iterate through **<th>** elements with **extract_column_from_header** to get column names.
6. **Create an empty dictionary**. Create an empty dictionary with keys (**dict.fromkeys)** from columns.
7. **Fill up dictionary.** Fill up dictionary with cleaned data from HTML table using **find_all(), .string, strip(), isdigit(), data_time()** and **list().**
8. **Create a Pandas Dataframe.** Convert filled-up dictionary into a dataframe using **pd.Dataframe**.

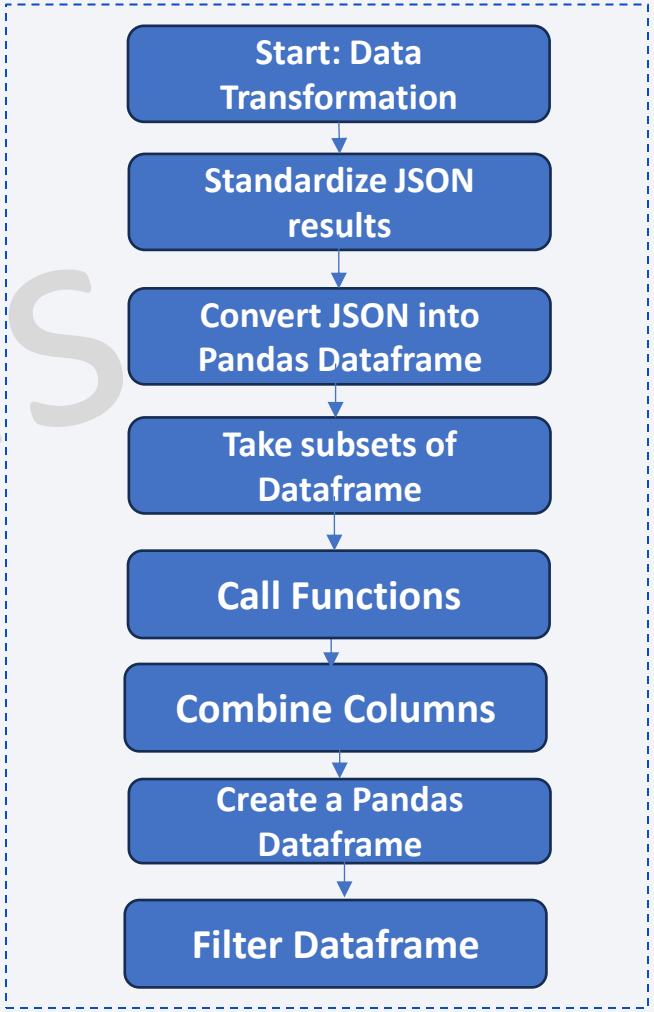
GitHub URL: Carolprojects/jupyter-labs-webscraping (4).ipynb at main · CarolaMertens/Carolprojects

Data Wrangling

Data Transformations on SpaceX API

- Standardize JSON results. (See page 11)
- Convert JSON into Pandas Dataframe. (See page 11)
- Take subsets of Dataframe. (See page 11)
- Call Functions. (See page 12)
- Combine Columns. (See page 12)
- Create a Pandas Dataframe. (See page 12)
- Filter Dataframe. (See page 12)

GitHub URL: Carolprojects/jupyter-labs-spacex-data-collection-api.ipynb at main · CarolaMertens/Carolprojects

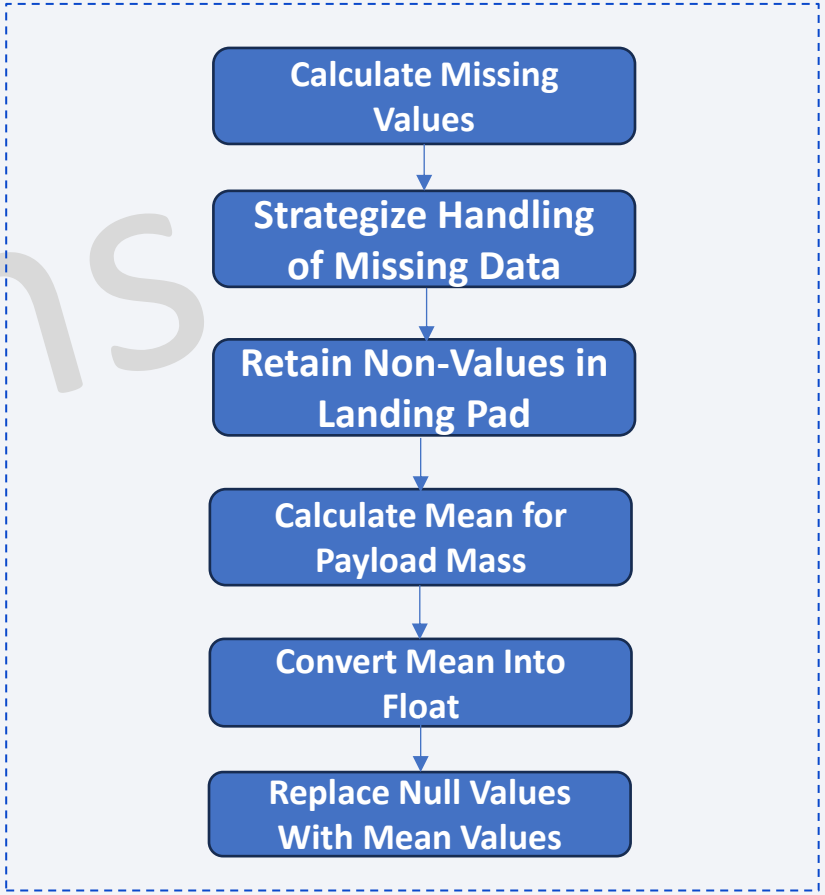


Data Wrangling

Dealing with Missing Values on SpaceX API

- The total number of missing values for each column was calculated using **isnull().sum()** to assess the usability of the reduced dataset and to inform the strategy for handling missing data.
- None values in Landing Pad column were retained to indicate instances where Landing Pads were not utilized.
- Mean for Payload Mass was calculated and converted into float using **astype('float').mean()** to replace null (**np.nan**) values with mean values using **replace().**

GitHub URL: Carolprojects/jupyter-labs-spacex-data-collection-api.ipynb at main · CarolaMertens/Carolprojects



Data Wrangling

Data Transformation on Scraped Data from Wikipedia

- Create Helper Functions. See page 13
- Create BeautifulSoup Object. See page 13
- Extract Column Names from HTML Table. See page 14
- Create an empty dictionary. See page 14
- Fill up dictionary. See page 14
- Create a Pandas Dataframe. See page 14

GitHub URL: Carolprojects/jupyter-labs-webscraping (4).ipynb at main · CarolaMertens/Carolproject s

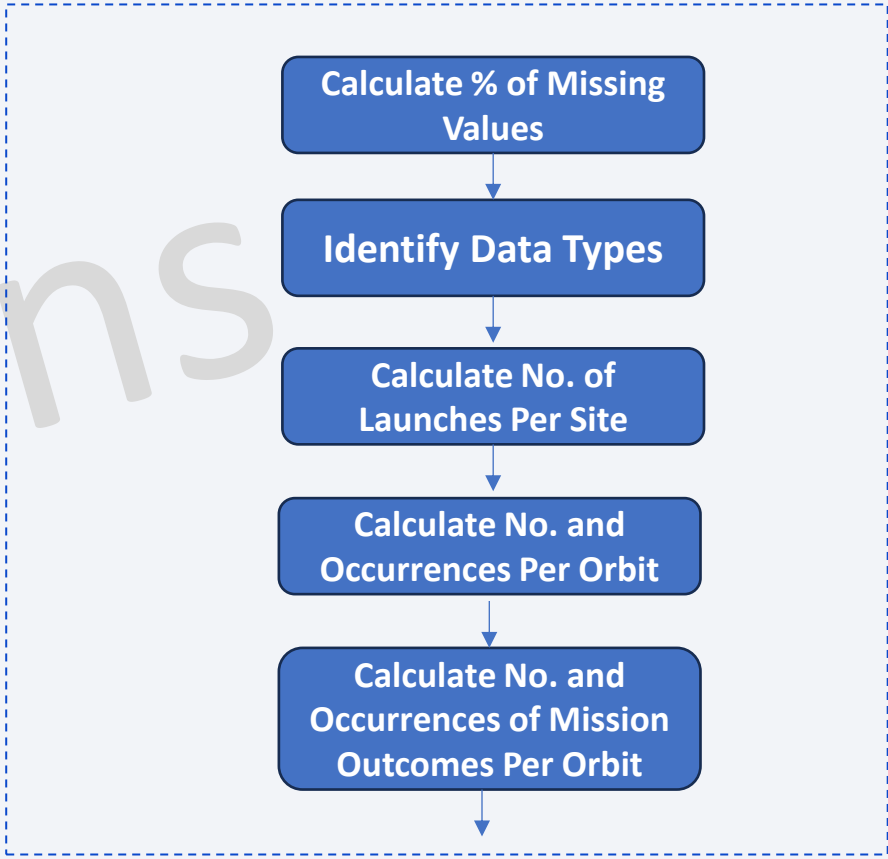
Remove blank rows from tables. Using SQL queries, blank rows were deleted using **is not null**.

GitHub URL: Carolprojects/jupyter-labs-eda-sql-coursera_sqllite.ipynb at bd4fcb225082ed97bb3e6173596aee5580ac7e0a · CarolaMertens/Carolprojects

Data Wrangling & EDA

The following Exploratory Data Analysis procedures were performed to get context on the dataset:

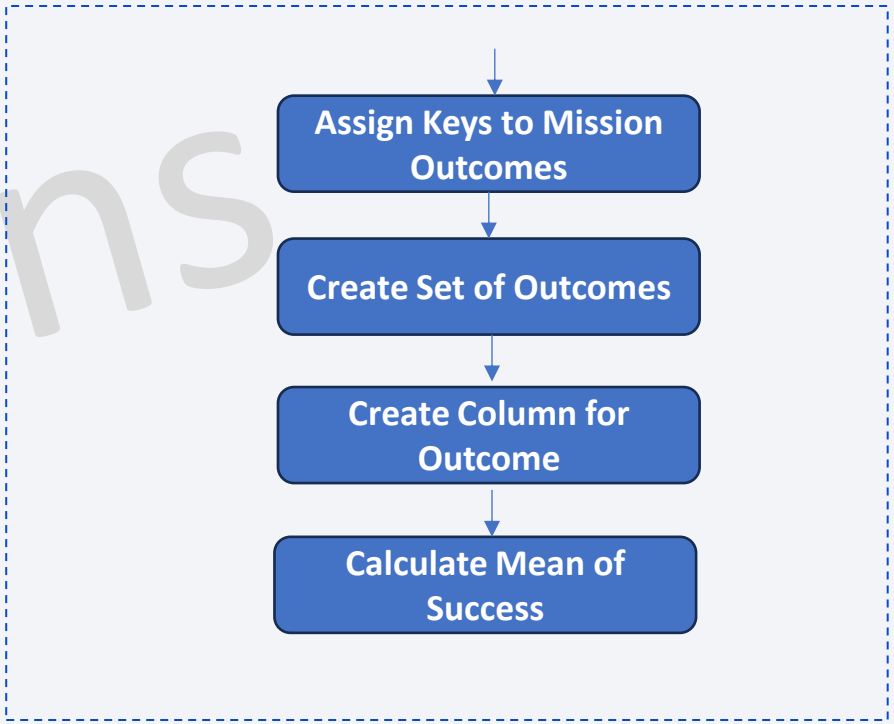
- Percentage of missing values was calculated on each attribute: **df.isnull().sum()/len(df)*100.**
- Data types were identified using **.dtypes** to determine which columns are numerical and categorical.
- Number of launches on each site was calculated: **df.value_counts(['LaunchSite'])**
- Number and occurrences of each orbit was calculated excluding the GTO orbit by filtering Orbit column: **df[df['Orbit'] != 'GTO']** then use: **value_counts(['Orbit'])**
- Calculated the number and occurrences of mission outcomes per orbit: **df.value_counts('Outcome')**



Data Wrangling & EDA

- Mission outcomes were assigned with keys: **enumerate(landing_outcomes.keys())** for easy reference
- A set of outcomes were created for unsuccessful landings: **set(landing_outcomes.keys()[[1,3,5,6,7]]).**
- A column was created to list outcomes where 0 means unsuccessful and 1 means successful: **[0 if outcome in bad_outcomes else 1 for outcome in df['Outcome']]**
- Mean of success rate was calculated: **df["Class"].mean()**

GitHub URL: [Carolprojects/labs-jupyter-spacex-Data wrangling (1).ipynb at main · CarolaMertens/Carolprojects]



EDA with Data Visualization

EDA was conducted using different visualizations for initial patterns and relationships:

- To easily identify the density of the data and its overall distribution across the range of variables, scatterplot was used between:
    1. Flight number and Payload Mass
    2. Flight number and Launch Site
    3. Payload mass and Launch Site
    4. Flight number and Orbit Type
    5. Payload mass and Orbit Type
- To easily understand how success rates vary across different Orbit Types, a bar chart was used.
- To analyze trends and changes of success rates over years, a line chart was used.

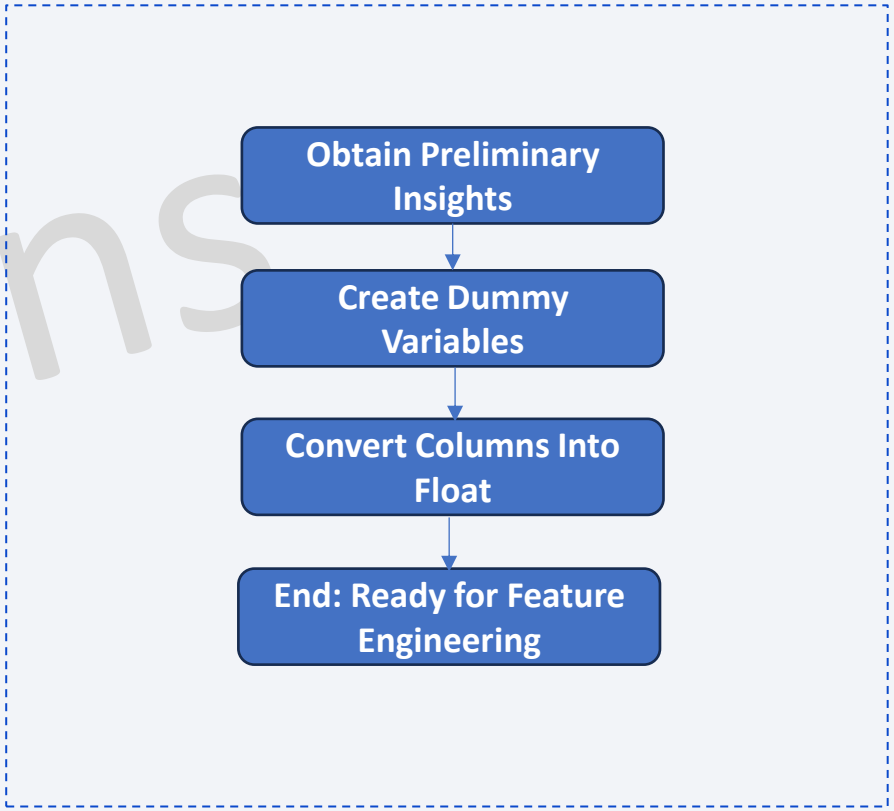
GitHub URL: Carolprojects/jupyter-labs-eda-dataviz.ipynb at bd4fcb225082ed97bb3e6173596aee5580ac7e0a · CarolaMertens/Carolprojects

EDA & Feature Engineering

After obtaining preliminary insights about the impact of different variables on success rate, some features were selected to be used for successful future predictions.

- Dummy variables were created to represent categorical data using **get_dummies()** to apply **OneHotEncoder**
- After creating the one-hot dataframe, its numeric columns were converted into float64 using **astype()**

GitHub URL: Carolprojects/jupyter-labs-eda-dataviz.ipynb at bd4fcb225082ed97bb3e6173596aee5580ac7e0a · CarolaMertens/Carolprojects



EDA with SQL

The following SQL queries were performed in Python to gain insight on the dataset:

- First SQL extension was loaded and a connection with the database was established using **sqlite3.connect(), cursor() and %sql sqlite:///my_data1.db** before reading the csv data using **read_csv()** and **to_sql().**
- **Drop Table if Exists** was executed and blank rows were removed from the table.
- Names of unique Launch Sites were displayed using **%sql select distinct.**
- 5 records of Launch Sites beginning with ‘CCA’ were displayed using **%sql…where Launch_Site like 'CCA%' limit 5**
- Total Payload Mass by NASA(CRS) was displayed using **%sql select sum(PAYLOAD_MASS__KG_)…where Customer = 'NASA (CRS)’**

EDA with SQL

- Average Payload Mass by F9 v1.1 was displayed using **%sql select avg(PAYLOAD_MASS__KG_)… where Booster_Version = 'F9 v1.1'**
- Listed date of first successful outcome in ground pad: **%sql select min(Date)… where Landing_Outcome='Success (ground pad)'**
- Listed successful booster in drone ship with specific payload: **%sql… where Landing_Outcome = 'Success (drone ship)' and PAYLOAD_MASS__KG_ > 4000 and PAYLOAD_MASS__KG_ <6000**
- Listed successful and failed outcomes. Since there were spaces in some rows, they were first removed for correct calculation: **%sq update SPACEXTABLE set Mission_Outcome = trim(Mission_Outcome) where true.** Then count total: **%sq select count(*)… group by Mission_Outcome**

EDA with SQL

- Used subquery to list boosters with maximum loads: **%sq …where PAYLOAD_MASS__KG_ in (select max(PAYLOAD_MASS__KG_) as Max_Payload_Mass from SPACEXTABLE)**
- Listed month names, and other details on failed outcomes in drone ship in 2015: **%sq select substr(Date,6,2) as Month_2015, … where Landing_Outcome='Failure (drone ship)' and substr(Date,0,5)='2015'**
- Ranked the total of landing outcomes between specified periods in descending order: **%sq select Landing_Outcome, count(Landing_Outcome) …where Date between '2010-06-04' and '2017-03-20' group by Landing_Outcome order by count(Landing_Outcome) desc**

GitHub URL: Carolprojects/jupyter-labs-eda-sql-coursera_sqllite.ipynb at bd4fcb225082ed97bb3e6173596aee5580ac7e0a · CarolaMertens/Carolprojects

Build an Interactive Map with Folium

Launch sites were located and marked on the map to perform interactive visual analytics using Folium. Below is the summary of objects created and added:

- A map with NASA Johnson Space Center as initial center location using

```
site_map = folium.Map(location=nasa_coordinate, zoom_start=10)
```

- A red circle with a text label NASA JSC was added around its coordinates to easily spot its location on the map using:

```
circle = folium.Circle(location=nasa_coordinate, radius=1000, color='#d35400',
        fill=True).add_child(folium.Popup('NASA Johnson Space Center'))
marker = folium.map.Marker(location=nasa_coordinate, icon=DivIcon(icon_size=(20,20),
        icon_anchor=(0,0), html='<div style="font-size: 12;
        color:#d35400;"><b>%s</b></div>' % 'NASA JSC', ))
site_map.add_child(circle)
site_map.add_child(marker)
```

Build an Interactive Map with Folium

- Located all Launch Sites and added circles around their labels on the map by iterating through them to explore their proximity to each other, the coast and the Equator

  for lat, lng, site in zip(launch_sites_df.Lat, launch_sites_df.Long, launch_sites_df['Launch Site']):

- Launch outcomes were added to each site to see which sites have high success rates. To easily identify the outcome on the map, classifications were made where class=1 in green marker means successful and class=0 in red marker means failure. **MarkerCluster()** was used to simplify a map containing markers having the same coordinates.

  spacex_df['marker_color'] = np.where(spacex_df['class'] == 1, 'green', 'red ')

Build an Interactive Map with Folium

- Marked down a point using **MousePosition()** on the map. With the mouse on a certain point of interest like the closest coastline, its distance to the Launch Site could be automatically calculated using **sin(), cos(), sqrt()** and **atan2().**

  a = sin(dlat / 2)2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2**
  c = 2 * atan2(sqrt(a), sqrt(1 - a))
  distance = R * c

- A blue polyline was drawn between a Launch Site and a selected point to easily locate the direction and the endpoints of the distance on the map.
- Additional lines were drawn from Launch Sites to their nearest city and railway to calculate their distance and to identify distance patterns and their impact to the success of the landings.

GitHub URL: [Carolprojects/lab_jupyter_launch_site_location folium_vscode.ipynb at main · CarolaMertens/Carolprojects]

Build a Dashboard with Plotly Dash

Plots/graphs and interactions were added to the dashboard for dynamic visualizations:

- **Implemented a Launch Site drop-down component** so users can easily choose between all available sites or a single specific location. This streamlines filtering and guarantees data accuracy for Launch Site names.
- **Integrated a success-rate pie chart** enhanced with a legend that automatically renders when a Launch Site is selected from the dropdown. This provides users with an easy-to-read, percentage-based view of the selected site's performance outcome.
- **Introduced a range slider** for Payload Mass selection, giving users precise control to filter missions by specific weight thresholds. This functionality helps users investigate the correlation between Payload Mass and mission outcome.

Build a Dashboard with Plotly Dash

- **Generated scatter plot** for visual comparison between Payload Mass and mission outcomes to see if a relationship exists at the chosen site(s).
- **Enhanced the scatter plot with colors and labels** to differentiate mission outcomes by booster. (See page 72 in Appendix A for the entire code)

GitHub Link: Carolprojects/dash_spacex VSCode.ipynb at main · CarolaMertens/Carolprojects

Predictive Analysis

1. A function was first created to plot the confusion matrix:

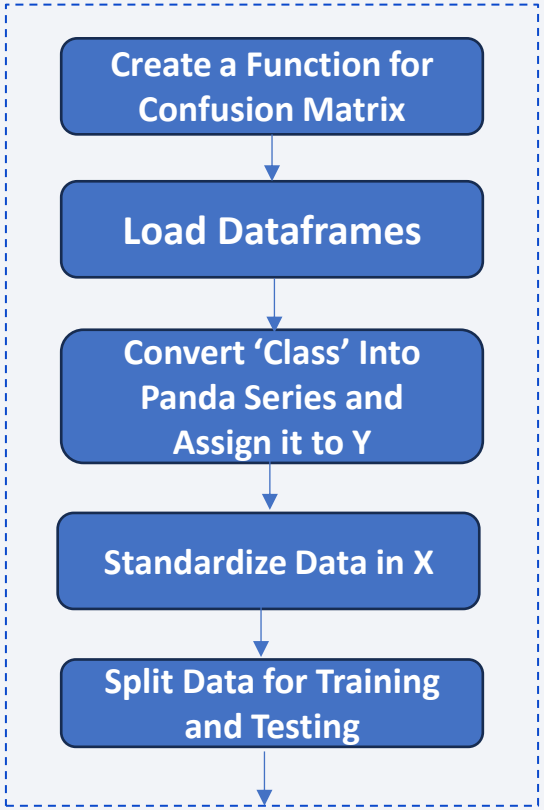
```
def plot_confusion_matrix(y,y_predict):
    cm = confusion_matrix(y, y_predict)
    ax= plt.subplot()
    sns.heatmap(cm, annot=True, ax = ax)   …
```

2. Dataframes were loaded using **requests.get(), io.BytesIO** and **pd.read_csv()**
3. Assigning the column ‘Class’ to variable Y making sure that the output is a Panda series: **Y = data['Class'].to_numpy()**
4. Standardize the data in X:

```
transform = preprocessing.StandardScaler()
X=transform.fit_transform(X)
```

5. Split the data for training and testing:

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



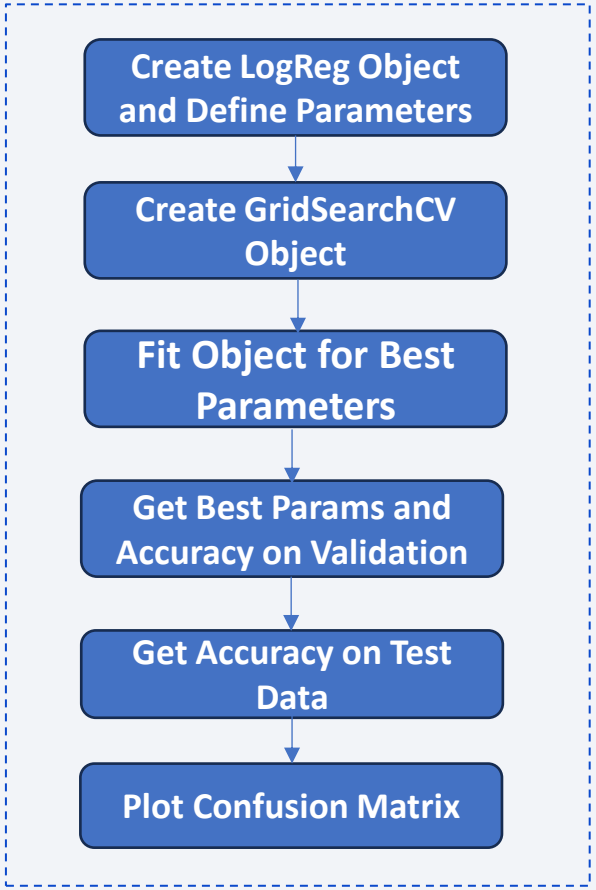
GitHub Link: Carolprojects/SpaceX_Machine Learning Prediction_Part_5.ipynb at main · CarolaMertens/Carolprojects · GitHub

Predictive Analysis (Logistic Regression)

1. Create Logistic Regression object by defining its parameter grid, creating a grid search object with 10-fold cross-validation, then fitting the object to find the best parameters from the dictionary:

   parameters = {'C': [0.01, 0.1, 1], 'penalty': ['l2'], 'solver': ['lbfgs']}
   lr = LogisticRegression()
   logreg_cv = GridSearchCV(estimator = lr, param_grid = parameters, cv = 10)
   logreg_cv.fit(X_train, Y_train)

2. Get the best parameters using the data attribute **best_params_** and the accuracy on the validation data using **best_score_**
3. Calculate accuracy on the test data using **score().**
4. Plot the confusion matrix: **yhat = logreg_cv.predict(X_test)**
   plot_confusion_matrix(Y_test, yhat)



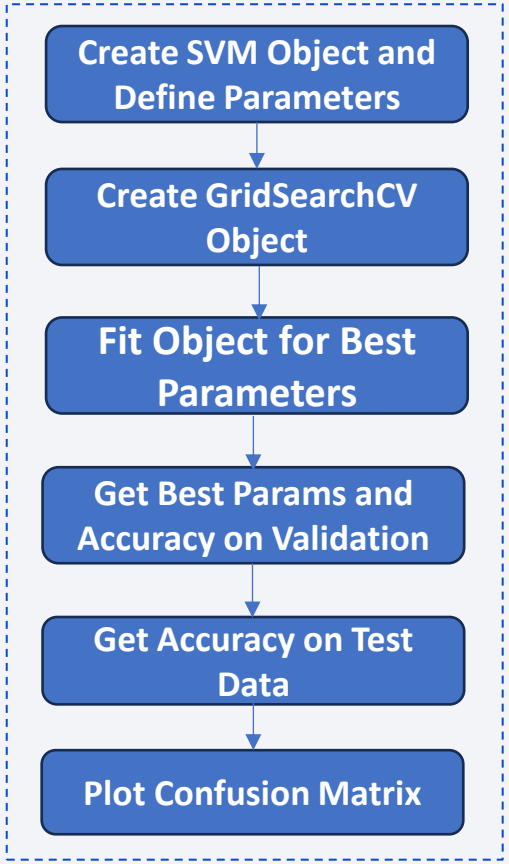
GitHub Link: Carolprojects/SpaceX_Machine Learning Prediction_Part_5.ipynb at main · CarolaMertens/Carolprojects · GitHub

Predictive Analysis (Support Vector Machine)

1. Create a Support Vector Machine object by defining its parameters and creating a GridSearchCV object with 10-fold cross-validation. Then fit the object to find the best parameters from the dictionary:

   parameters = {'kernel': ('linear', 'rbf', 'poly', 'rbf', 'sigmoid'), 'C': np.logspace(-3, 3, 5), 'gamma': np.logspace(-3, 3, 5)}
   svm = SVC()
   svm_cv = GridSearchCV(estimator=svm, param_grid=parameters, cv=10)
   svm_cv.fit(X_train, Y_train)
2. Get the best parameters using the data attribute **best_params_** and the accuracy on the validation data using **best_score_**
3. Calculate accuracy on the test data using **score().**
4. Plot the confusion matrix:

   yhat = svm_cv.predict(X_test)
   plot_confusion_matrix(Y_test, yhat)



GitHub Link: Carolprojects/SpaceX_Machine Learning Prediction_Part_5.ipynb at main · CarolaMertens/Carolprojects · GitHub

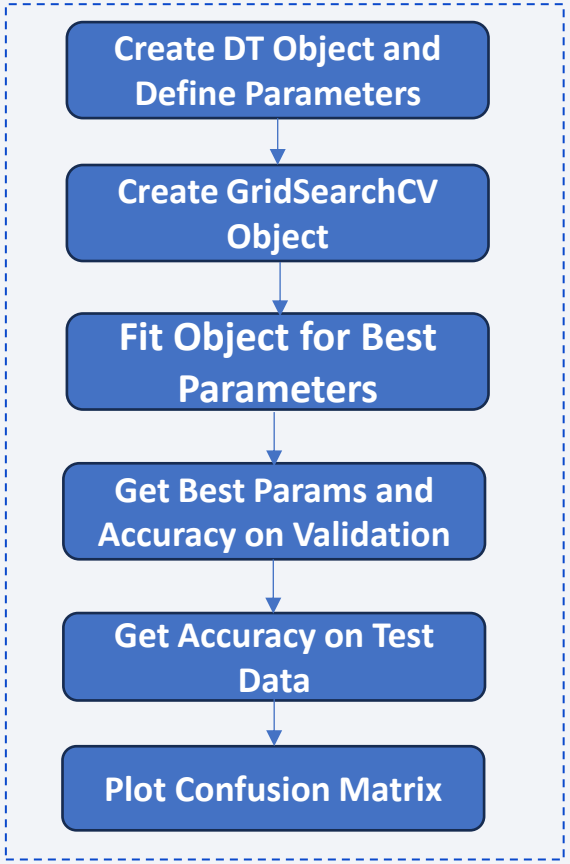
Predictive Analysis (Decision Tree)

1. Create a Decision Tree classifier object by defining its parameters and creating a GridSearchCV object with 10-fold cross-validation. Then fit the object to find the best parameters from the dictionary:

```
parameters = {'criterion': ['gini', 'entropy'], 'splitter': ['best', 'random'],
'max_depth': [2*n for n in range(1,10)], 'max_features': ['auto', 'sqrt'],
'min_samples_leaf': [1, 2, 4], 'min_samples_split': [2, 5, 10]}
tree = DecisionTreeClassifier()
tree_cv = GridSearchCV(estimator=tree, param_grid=parameters, cv=10)
tree_cv.fit(X_train, Y_train)
```

2. Get the best parameters using the data attribute **best_params_** and the accuracy on the validation data using **best_score_**
3. Calculate accuracy on the test data using **score()** method.
4. Plot the confusion matrix:

```
yhat = tree_cv.predict(X_test)
plot_confusion_matrix(Y_test, yhat)
```

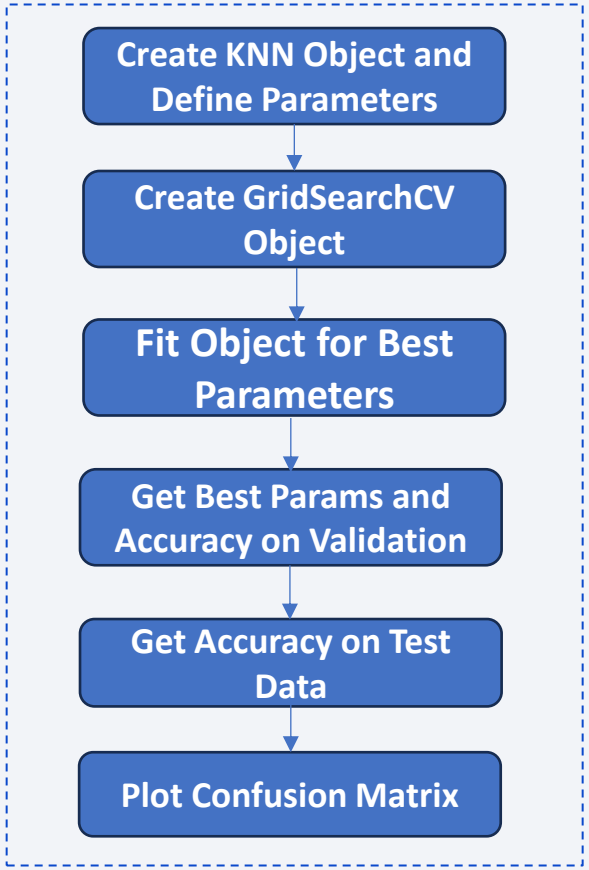


GitHub Link: Carolprojects/SpaceX_Machine Learning Prediction_Part_5.ipynb at main · CarolaMertens/Carolprojects · GitHub

Predictive Analysis (K-Nearest Neighbors)

1. Create a K-Nearest Neighbors object by defining its parameters and creating a GridSearchCV object with 10-fold cross-validation. Then fit the object to find the best parameters from the dictionary:

   parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'], 'p': [1,2]}
   KNN = KNeighborsClassifier()
   knn_cv = GridSearchCV(estimator=KNN, param_grid=parameters, cv=10)
   knn_cv.fit(X_train, Y_train)
2. Get the best parameters using the data attribute **best_params_** and the accuracy on the validation data using **best_score_**
3. Calculate accuracy on the test data using **score()**
4. Plot the confusion matrix: **yhat = knn_cv.predict(X_test)**
   plot_confusion_matrix(Y_test, yhat)



GitHub Link: Carolprojects/SpaceX_Machine Learning Prediction_Part_5.ipynb at main · CarolaMertens/Carolprojects · GitHub

Exploratory Data Analysis Results

Here is a summary of the **Exploratory Data Analysis results**:

- For Flight Number vs. Launch Site: success rate seemed to increase as the Flight Number increases in CCAFS SLC 40, VAFB SLC 4E and KSC LC 39A sites.
- For Payload Mass vs. Launch Site, almost all rockets beyond 7000 Payload Mass were successful in landing in all 3 abovementioned sites.
- For Success Rate vs. Orbit Type: ES-L1, SSO, HEO and GEO have 100% success rate while GTO has the lowest reliability rate. But for SO there is 0% due to a direct consequence of physics which forces Falcon 9 to meet the Solar Orbit (SO) mission requirement rather than its reusability.
- For Flight Number vs. Orbit Type: the higher the Flight Number, the higher the success rate in the orbit except for GTO and others with only 1 landing. GTO reduces first stage reusability since it requires higher energy requirement leaving less fuel for crucial landing maneuver.
- For Payload Mass vs Orbit Type: there are more successful landing rates for PO, LEO and ISS with heavy payloads.
- For Success Rate vs. Year, success rate continuously increased from 2013 to 2020

Exploratory Data Analysis Results

Interactive analytics demo in screenshots results:

- KSC LC-39A Launch Site has the highest success rate, while CCAFS SLC-40 has the lowest.
- The payload range of 2500-5000 kg has the highest launch success rate with 11 successful landings from various Booster Versions.
- B4 Booster Version has the highest Payload Mass with 9600 kg while v1.0 has the lowest Payload Mass with 0 kg.
- The payload range of 7500-1000 kg has the lowest launch success rate with only 1 successful landing from BS Booster version.
- FT Booster Version has the highest launch success rate with 13 successful landings while v1.0 Booster Version has the lowest with 0 successful landing.

Predictive Analysis Results

Standard Accuracy Comparison (Method Score)

- The score() method was used to quickly calculate each model's **accuracy** on the fraction of correctly predicted samples.
- When calculating test accuracy, all four classifiers yielded the **identical result** of approximately **0.8333** (or 83.33%)
- These same results suggest that when running the score() method on the default and basic version of the test set, all four models achieved the **exact same accuracy**. This could indicate the test set is simple and that the models, *before* hyperparameter tuning (optimization) was used, performed similarly.

Predictive Analysis Results

Optimization Accuracy Comparison (Best Score)

- The Best Score Accuracy represents the **highest performance** that each classification model was able to achieve after having its hyperparameters optimized. This gives the most realistic measure of each model's capability to perform best when it is operating at its peak potential.
- Using this evaluation metric with hyperparameter tuning and cross-validation, the models showed differentiation, with the **Decision Tree Classifier** achieving the highest score:

Classifier	Best Score Accuracy	Percentage	Rank
Decision Tree Classifier	**0.8768**	**87.68%**	**1**
K-Nearest Neighbors (KNN)	0.8482	84.82%	2
Support Vector Machine (SVM)	0.8482	84.82%	3
Logistic Regression	0.8464	84.64%	4

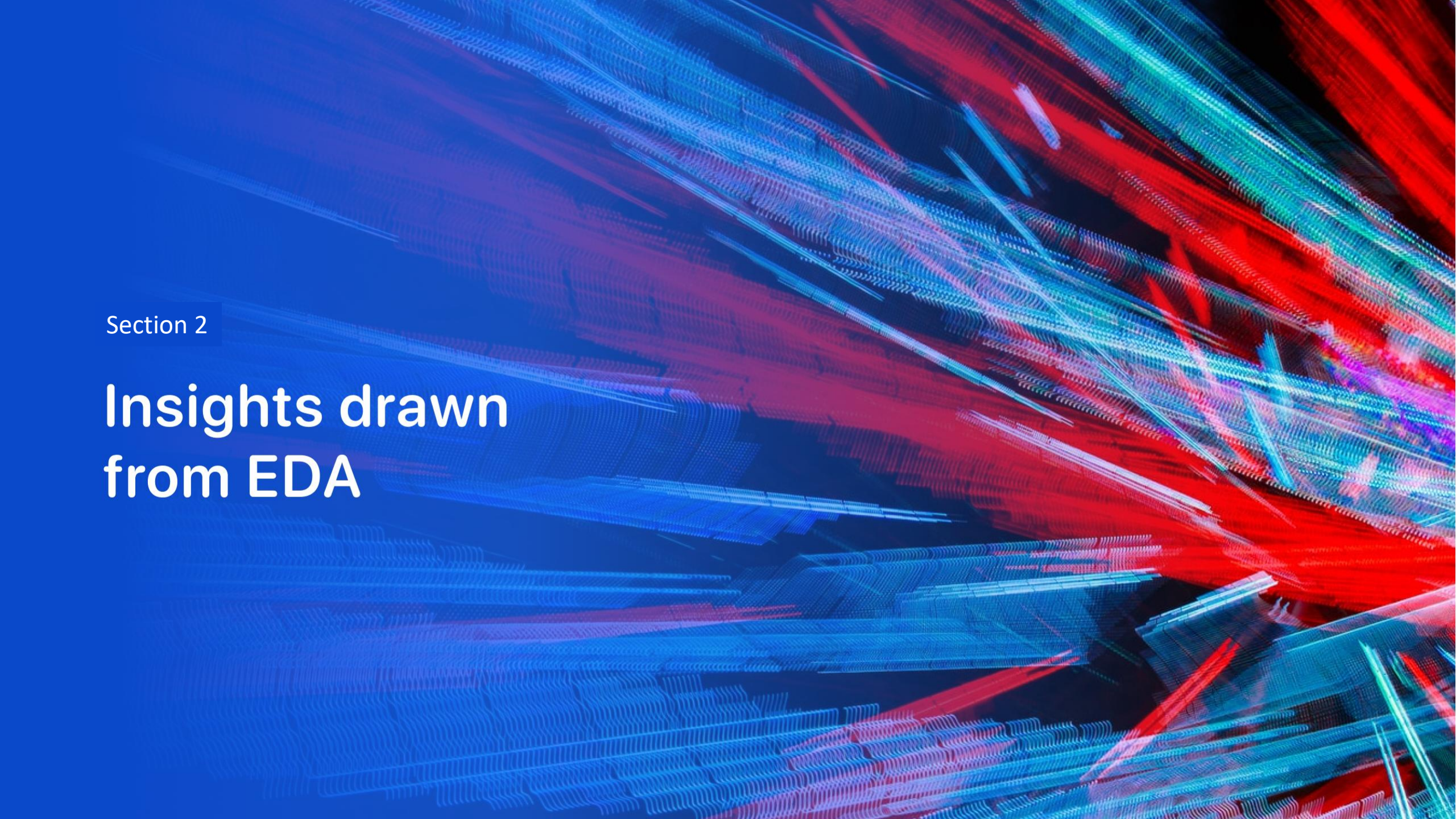
Predictive Analysis Results

Confusion Matrix Analysis

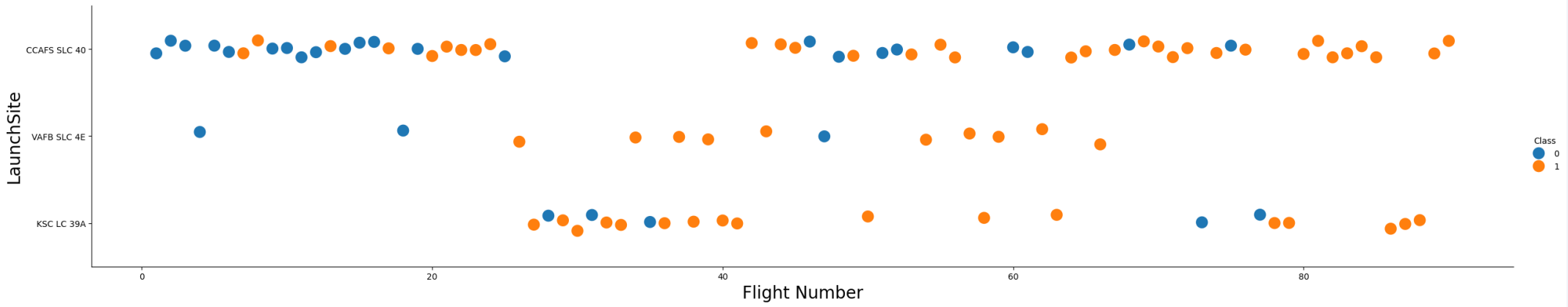
- A review of the **Confusion Matrix** revealed that, for the test set used in the standard score, **all four classifiers generated identical performance metrics** concerning class predictions:

Metric	Count	Interpretation
True Positive (TP)	12	The model **correctly predicted 12 instances** that truly **landed** (Actual: Landed, Predicted: Landed).
False Positive (FP)	3	The model **incorrectly predicted 3 instances** to have **landed** when they truly **did not land** (Actual: Not Landed, Predicted: Landed).

- The reason all classifiers produced the same results (identical confusion matrices) is due to characteristics of the **data itself**, indicating that the classification problem is exceptionally **simple.**


Section 2
Insights drawn from EDA

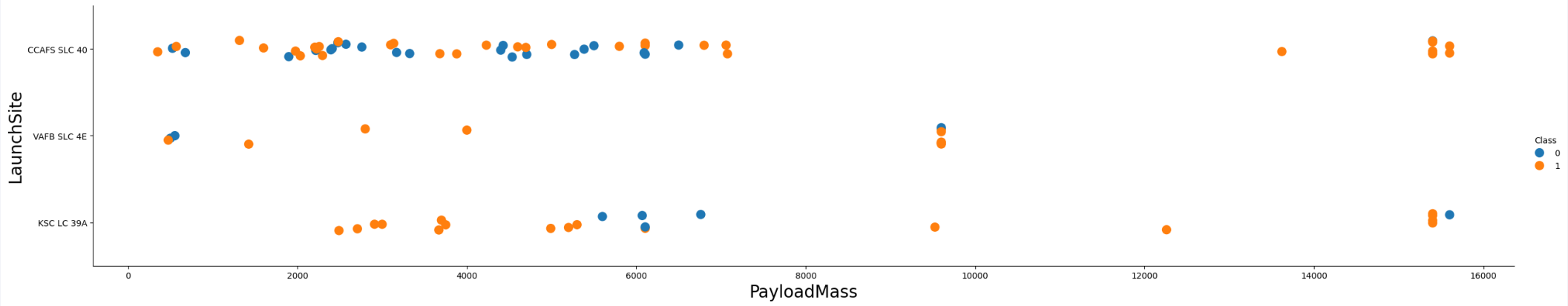
Flight Number vs. Launch Site



In the scatterplot you could see that the landings of the last few flights in every Launch Site were all successful.

It also shows that the higher the Flight Number is, the more likely the success rate is.

Payload vs. Launch Site



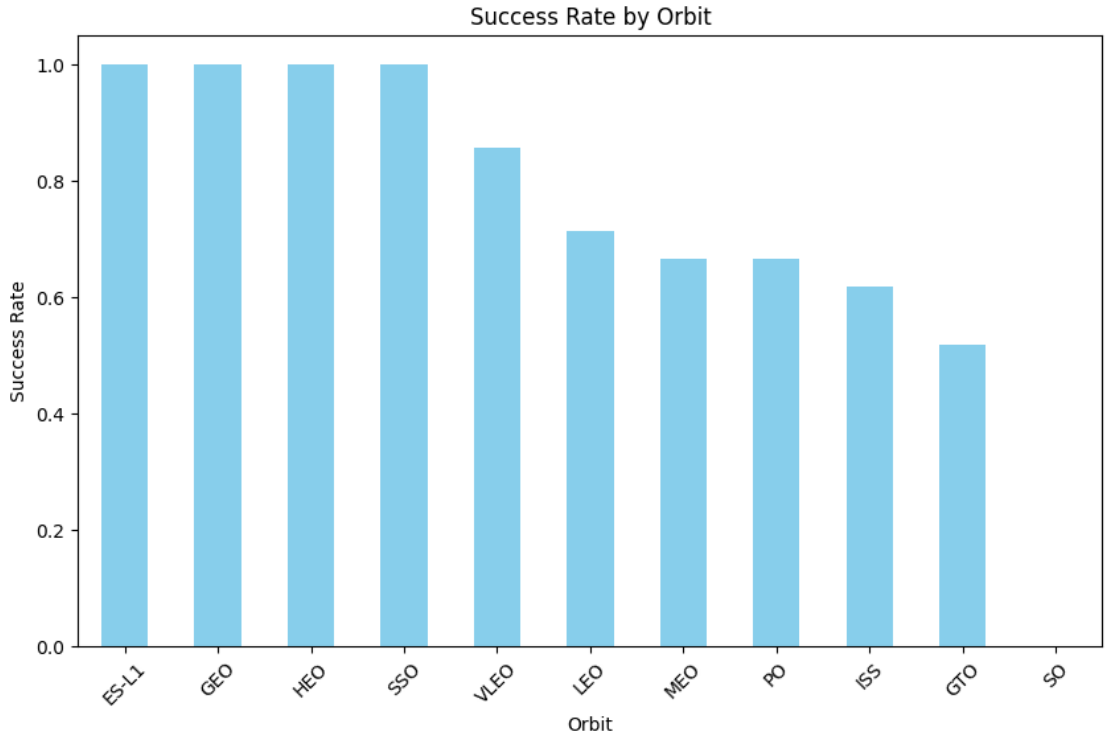
Here you could see that VAFB-SLC 4E and KSC-LC 39A have more success rates than CCAFS-SLC 40 however, for VAFB-SLC Launch Site, no rockets for heavy Payload Mass (greater than 10000) were launched.

For all 3 Launch Sites, almost all rockets beyond 7000 Payload Mass were successful in landing.

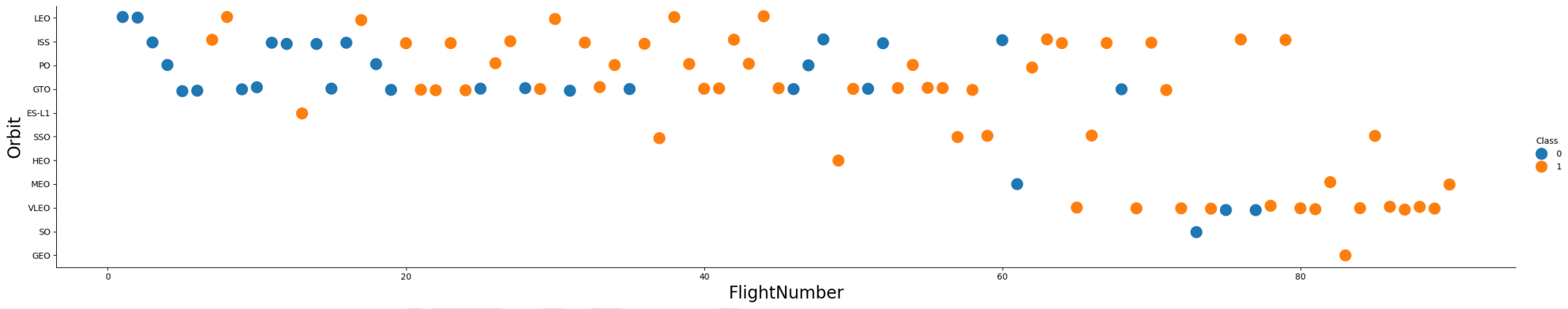
Success Rate vs. Orbit Type

As seen in the bar chart, ES-L1, GEO, HEO and SSO have 100% success rate while GTO has the lowest reliability rate for successful landing.

For SO there is 0% due to the fact that Falcon 9 commits all its available fuel to pushing the payload as fast and as far as possible to make a Solar Orbit (SO) mission successful rather than make its first stage reusable.



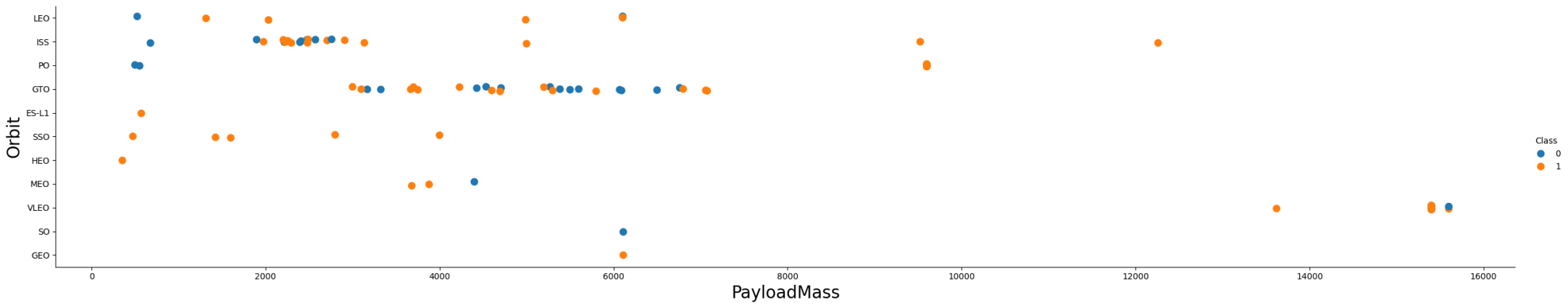
Flight Number vs. Orbit Type



It could be observed in the chart that success rate in the orbit seemed to be related to the Flight Number except for GTO and other orbits with only 1 landing.

This is understandable since GTO's booster separates from the second stage at a higher altitude and faster velocity that no fuel is reserved for landing.

Payload vs. Orbit Type



GEO and SO appeared to have only one Payload Mass orbiting around it.

For VLEO, there appeared to be no Payload Mass lower than 12000 which orbited around it.

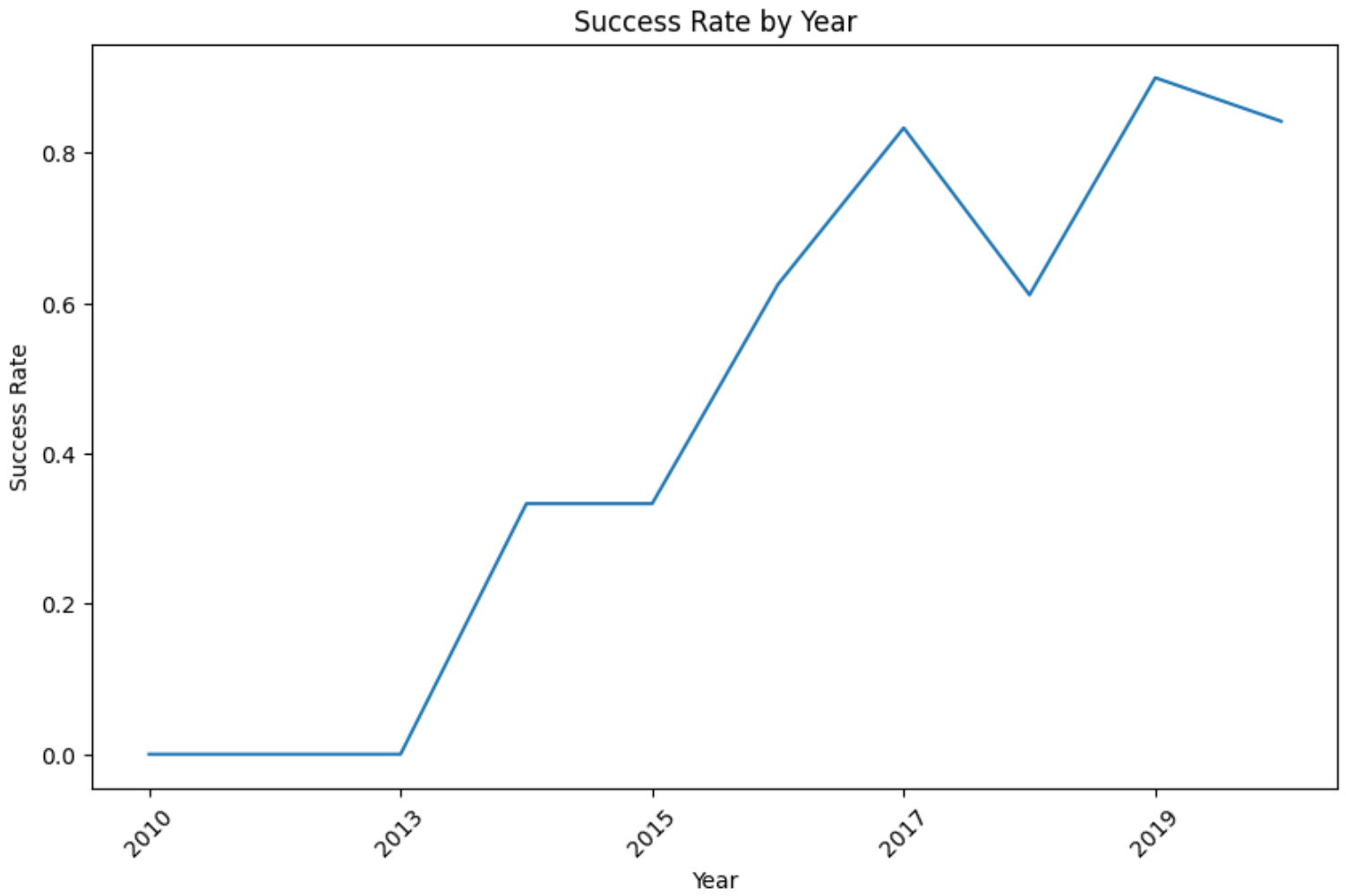
SSO appeared to have 100% successful landings. No Payload Mass higher than 4000 orbited around it.

For LEO, ISS and PO, there appeared to be more successful landings the heavier the rockets' loads were.

However, for GTO, there appeared to have no relationship between Payload Mass and Orbit Type since successful and unsuccessful outcomes were both present in the chart.

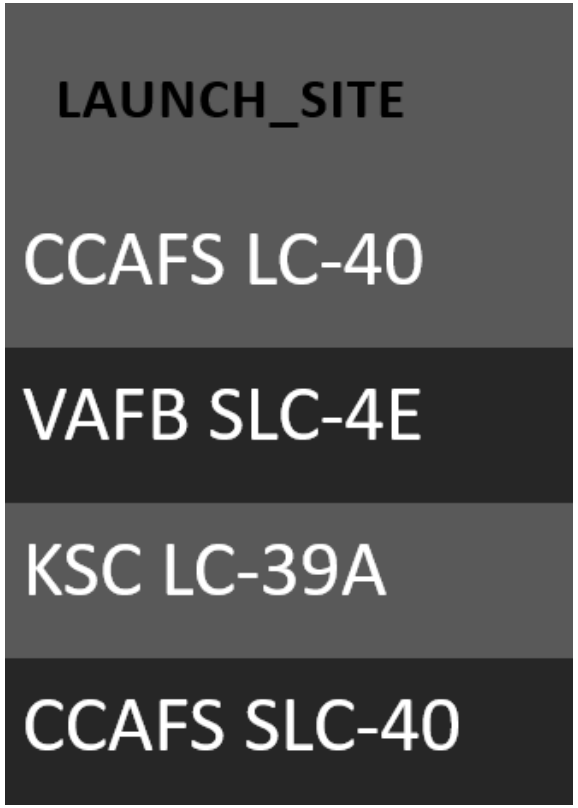
It could be observed that success rate since 2013 kept increasing till 2020.

Launch Success Yearly Trend



All Launch Site Names

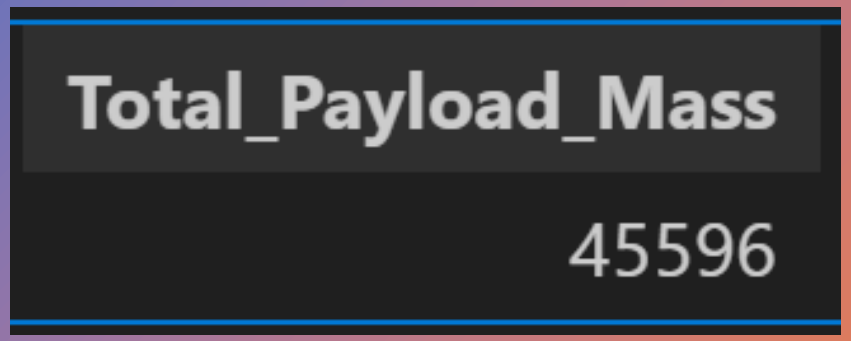
By using the **distinct** keyword in SQL query to get the names of the Launch Sites produced these unique names.



Launch Site Names Begin with 'CCA'

The SQL query ..**where Launch_Site like 'CCA%'** filters the Launch Site column which starts with CCA characters while **with limit 5** returns only 5 records.

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

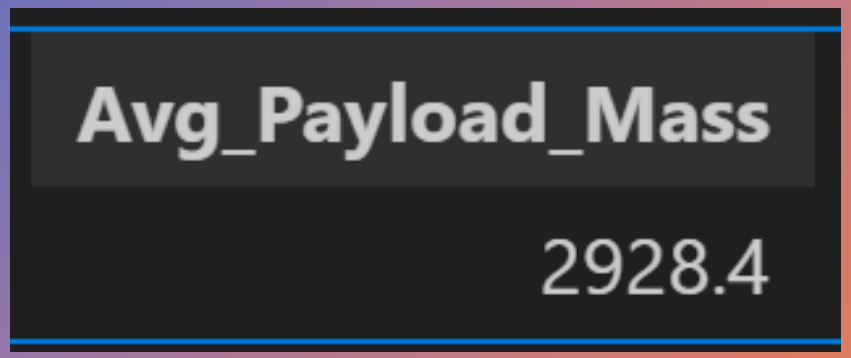


Total Payload Mass

The following query calculates the total payload carried by boosters from NASA

```
%sql select sum(PAYLOAD_MASS__KG_) as
Total_Payload_Mass
from SPACEXTABLE
where Customer = 'NASA (CRS)'
```

Sum() computes the total number of payloads while **where** filters the column 'Customer' to include only the NASA (CRS) in the calculation.



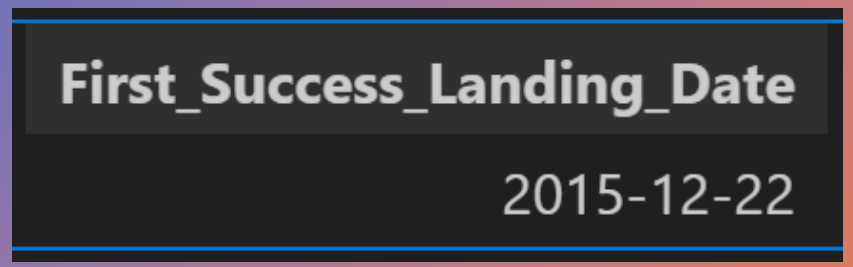
Average Payload Mass by F9 v1.1

To calculate the average Payload Mass carried by booster version F9 v1.1:

```
%sql select avg(PAYLOAD_MASS__KG_) as
Avg_Payload_Mass
from SPACEXTABLE
where Booster_Version = 'F9 v1.1’
```

Where filters the column booster version for ‘F9 v1.1’ and gets its average of Payload Mass using **AVG().**

First Successful Ground Landing Date



To find the dates of the first successful landing outcome on ground pad:

%sql select min(Date) as First_Success_Landing_Date
from SPACEXTABLE
where Landing_Outcome='Success (ground pad)'

Min() returns the oldest date in the dataset and **where** filters the column landing outcome to only include success on the ground pad as basis of the dates.

Successful Drone Ship Landing with Payload between 4000 and 6000

Booster_Version	PAYLOAD_MASS__KG_
F9 FT B1022	4696
F9 FT B1026	4600
F9 FT B1021.2	5300
F9 FT B1031.2	5200

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

```
%sql select Booster_Version, PAYLOAD_MASS__KG_
from SPACEXTABLE
where Landing_Outcome = 'Success (drone ship)' and
PAYLOAD_MASS__KG_ > 4000 and
PAYLOAD_MASS__KG_ <6000
```

Where filtered 2 conditions; 1 for successful landing in a drone ship and another for a range of Payload Mass using **and** to combine conditions.

count	Mission_Outcome
1	Failure (in flight)
99	Success
1	Success (payload status unclear)

Total Number of Successful and Failure Mission Outcomes

To calculate the total number of successful and failure mission outcomes

First, remove the spaces from the mission outcome using **trim()** since it was found to contain spaces on 'Success' values thereby producing an incorrect result

```
%sql update SPACEXTABLE
set Mission_Outcome = trim(Mission_Outcome)
where true
```

Then count the mission outcome by grouping itself:

```
%sql select count(*) as count, Mission_Outcome from
SPACEXTABLE group by Mission_Outcome
```

Booster_Version	Max_Payload_Mass
F9 B5 B1048.4	15600
F9 B5 B1049.4	15600
F9 B5 B1051.3	15600
F9 B5 B1056.4	15600
F9 B5 B1048.5	15600
F9 B5 B1051.4	15600
F9 B5 B1049.5	15600
F9 B5 B1060.2	15600
F9 B5 B1058.3	15600
F9 B5 B1051.6	15600
F9 B5 B1060.3	15600
F9 B5 B1049.7	15600

Boosters Carried Maximum Payload

List the names of the booster which have carried the maximum Payload Mass

```
%sql select Booster_Version,
PAYLOAD_MASS__KG_ as Max_Payload_Mass
from SPACEXTABLE
where PAYLOAD_MASS__KG_ in
      (select max(PAYLOAD_MASS__KG_) as
      Max_Payload_Mass from SPACEXTABLE)
```

The subquery calculates the highest value in Payload Mass across the entire table. Then the outer query **where** filters every row to see if its value is equal to the result of the subquery.

2015 Launch Records

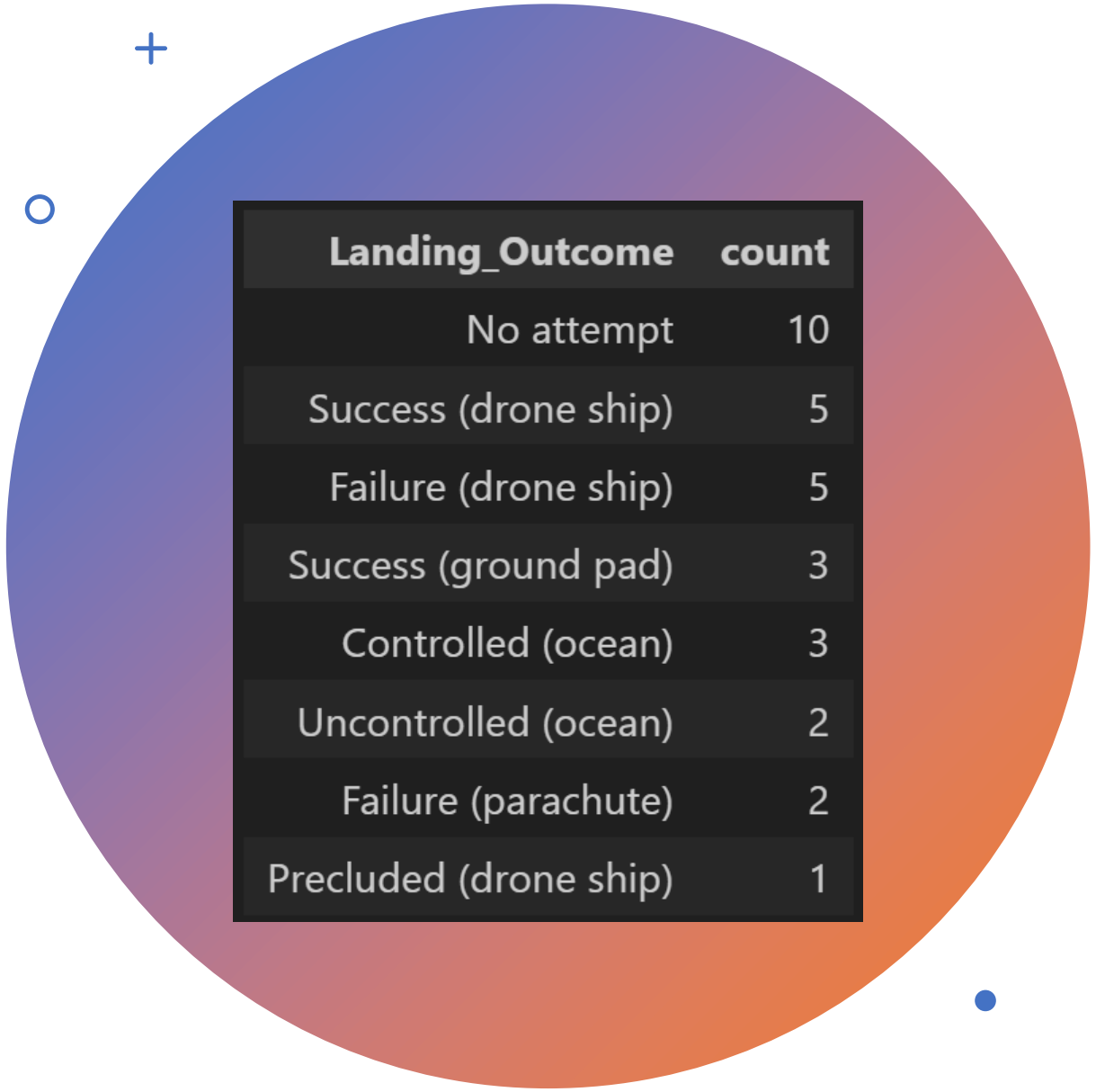
Month_2015	Landing_Outcome	Booster_Version	Launch_Site
01	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
04	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

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

```
%sql select substr(Date,6,2) as Month_2015,
Landing_Outcome, Booster_Version, Launch_Site
from SPACEXTABLE
where Landing_Outcome='Failure (drone ship)' and
substr(Date,0,5)='2015’
```

Where filters to include only failed landing on drone ship combined with the year equivalent to 2015.

- substr(Date,6,2) extracts the 6th position from YYYY-MM-DD which refers to the month while 2 refers to the 2 digits of that month
- substr(Date,0,5) extracts the year on a length of 5 (to capture 2015).



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

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 select Landing_Outcome,
count(Landing_Outcome) as count
from SPACEXTABLE
where Date between '2010-06-04' and '2017-03-20'
group by Landing_Outcome
order by count(Landing_Outcome) desc
```

The query counts the total number of each type of landing outcomes for all launches between the given dates then present the result from the most to the least frequent.

Section 3

Launch Sites Proximities Analysis

Folium Map Sites Location

Figure 1.1

Figure 1.2

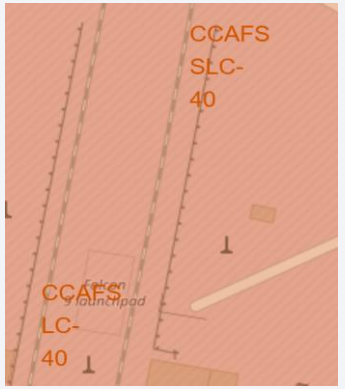
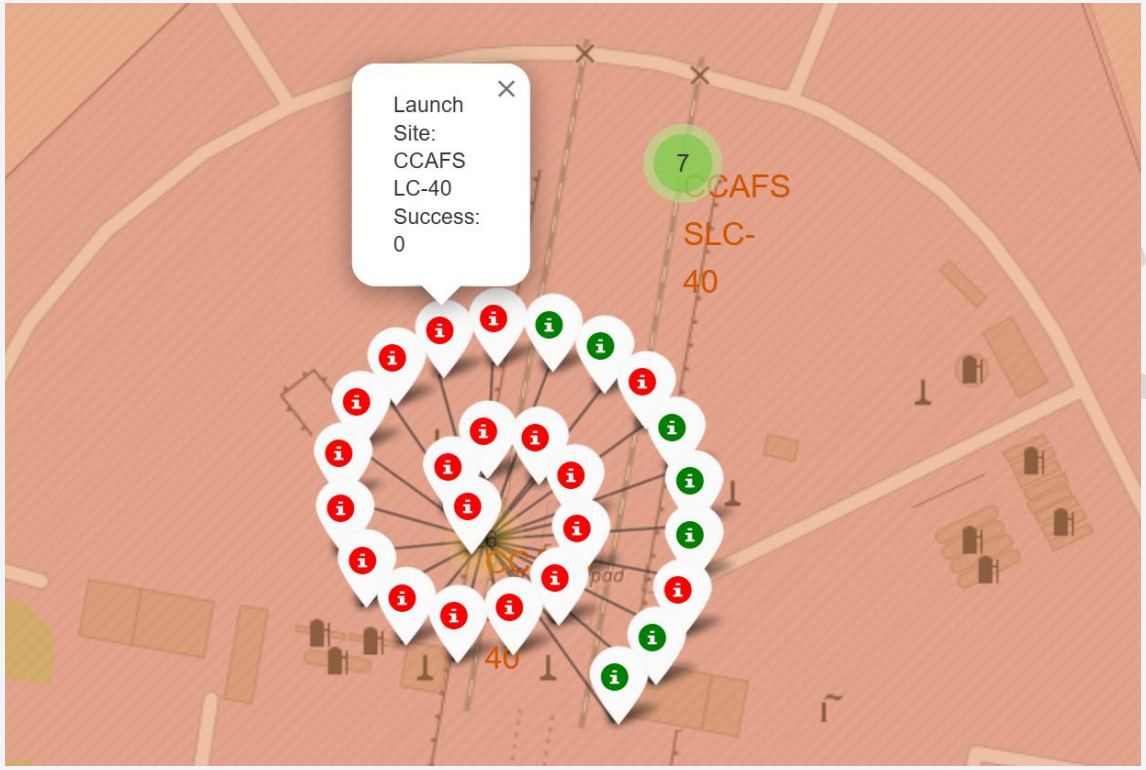


Figure 1.3

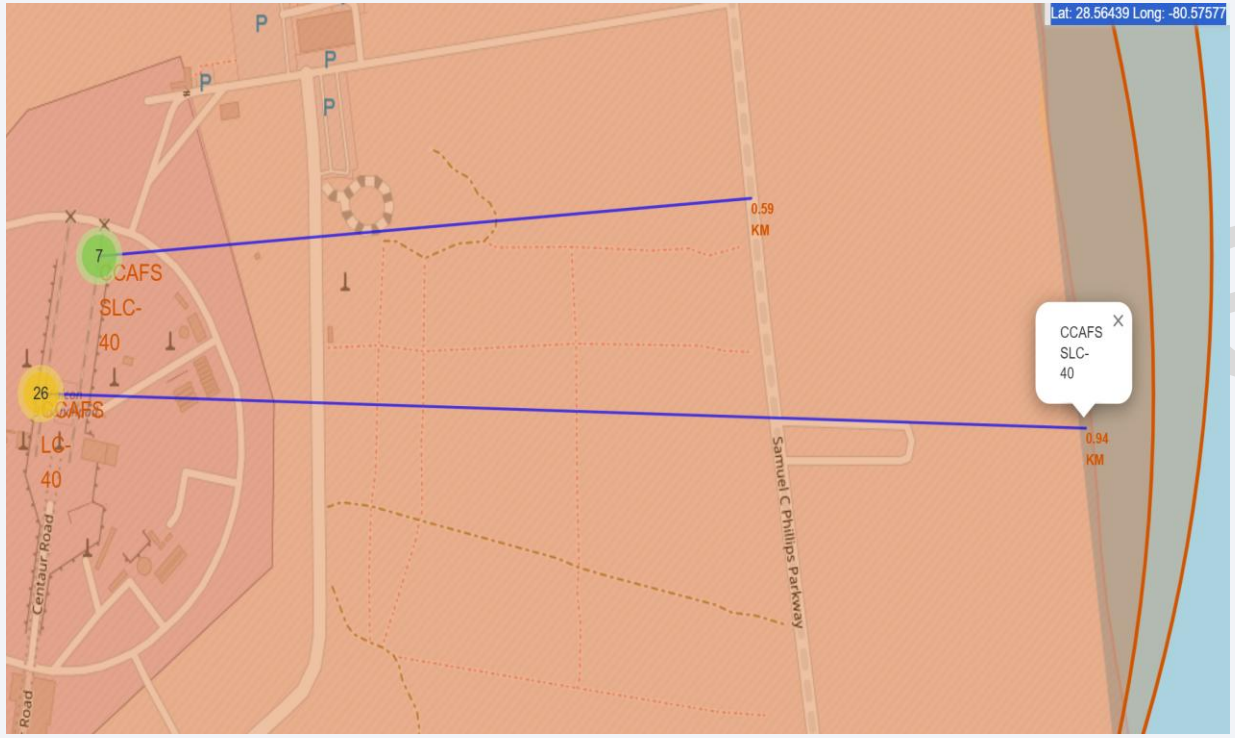
- Figure 1.1 shows the folium map of the Launch Sites represented by red circle markers. The red circles are positioned based on the Launch Site's coordinates. Text labels were also included to identify the name of each site.
- From the left you can find the VAFB-SLC-4E Launch Site while on the right you can hardly read the overlapping texts. This means that there are 2 or more Launch Sites located near each other which can be examined by zooming in the map. Let us maximize the right side of the map to find out how many Launch Sites are there.
- Figure 1.2 shows the zoomed-in overlapping Launch Sites of Figure 1.1. Here you can see that there are 3 Launch Sites with KSC-LC-39A on the left and another overlapping Launch Sites on the right.
- Figure 1.3 shows CCAFS-SLC-40 and CCAFS-LC-40 Launch Sites that are overlapping in Figure 1.2.
- The total number of Launch Sites shown on the map is four. All of them are located near the coast since this is a safety requirement to get rockets move to space with minimal risk.

Folium Map – Launch Outcomes

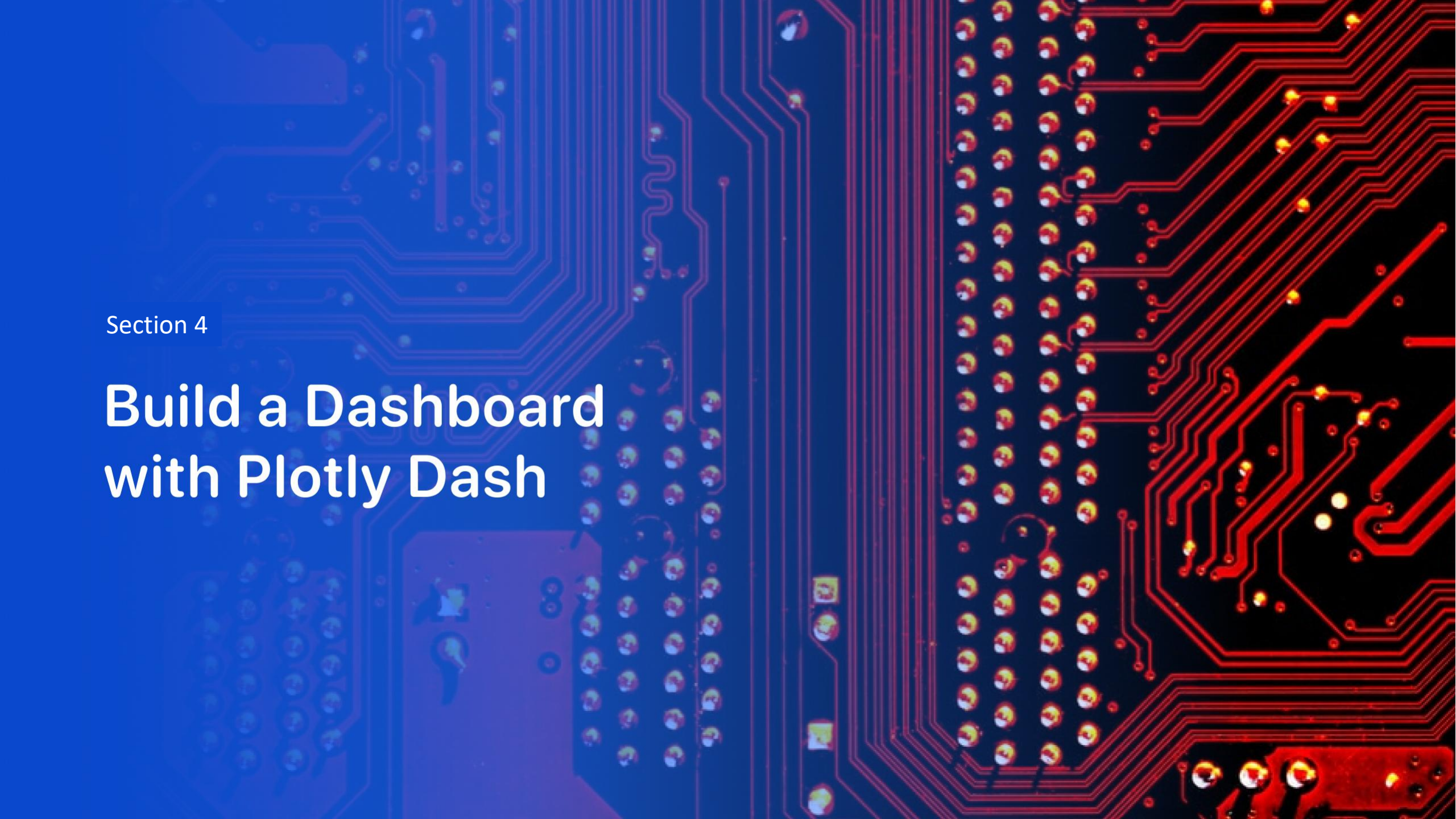


- This figure shows the launch outcomes of CCAFS-LC-40. A successful launch is identified as 1 in green marker while a failed one is classified as 0 in red. Marker clusters simplify a map having many markers in the same coordinate.
- The green circle marks the location of CCAFS-SLC-40. Clicking it would show how many successful and unsuccessful landings it had.
- Through color-labeled markers in marker clusters we can easily identify 7 successful launches from the site and 19 unsuccessful launches.

Folium Map – Launch Site Proximities



- This figure shows the nearest highway to the Launch Site CCAFS-SLC-40 with a distance of 0.59k, and the nearest coastline which is 0.94km away from CCAFS-LC-40.
- A Mouse Position on the map was used to get the coordinates.
- A Polyline between the Launch Site and the selected point was drawn to mark the distance between these endpoints.
- Close proximities of Launch Sites to highways, railways and coastlines give substantial benefits for logistical challenges in launching rockets since rockets are too large and heavy to be transported.

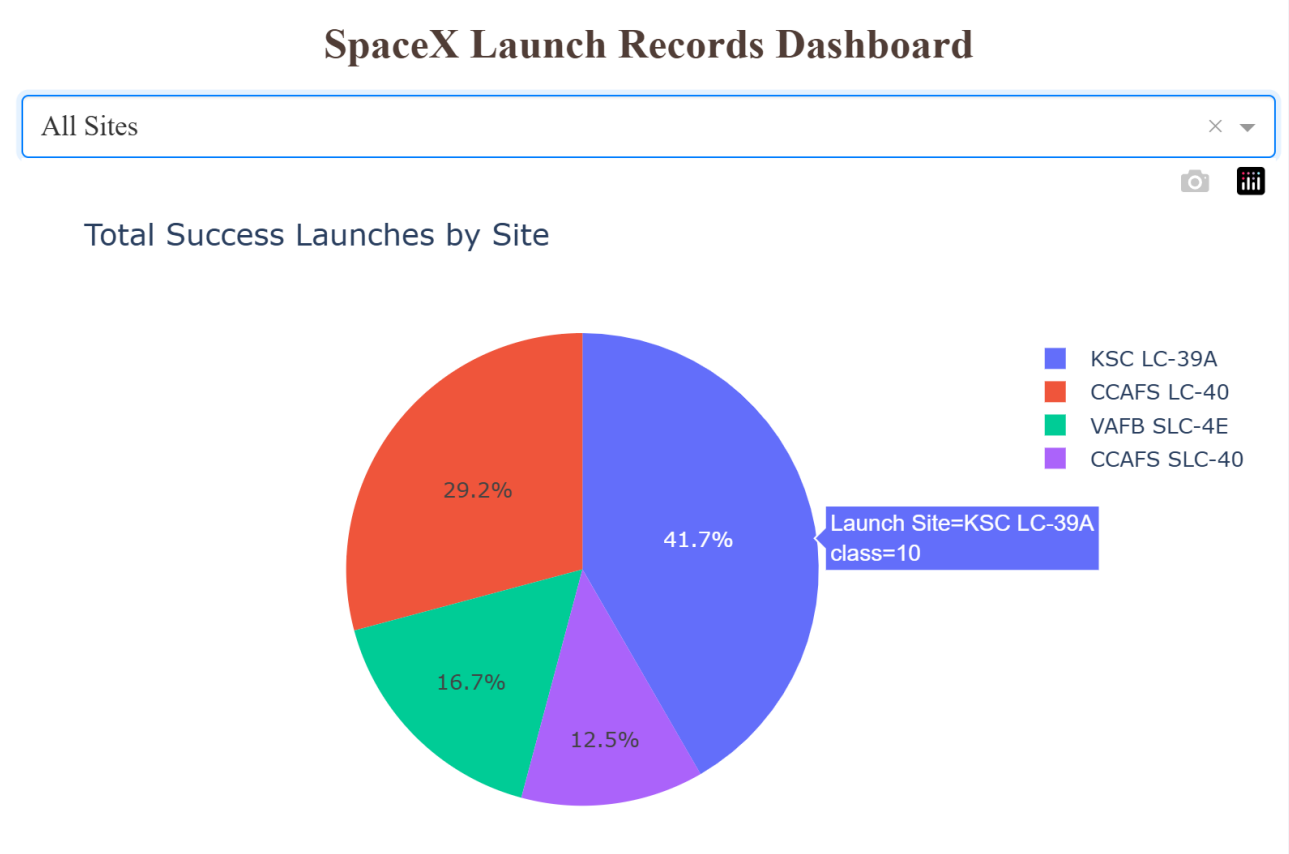

Section 4
Build a Dashboard with Plotly Dash

Plotly Dash - All Sites Dashboard

The pie chart is configured to display the percentage of successful launches across all sites by selecting "All Sites" from the drop-down menu.

- Interactive Tooltip: Hovering over a slice reveals the site's name and the total number of successful landings achieved there.
- Legend Clarity: The accompanying legend clearly shows which site corresponds to each color.

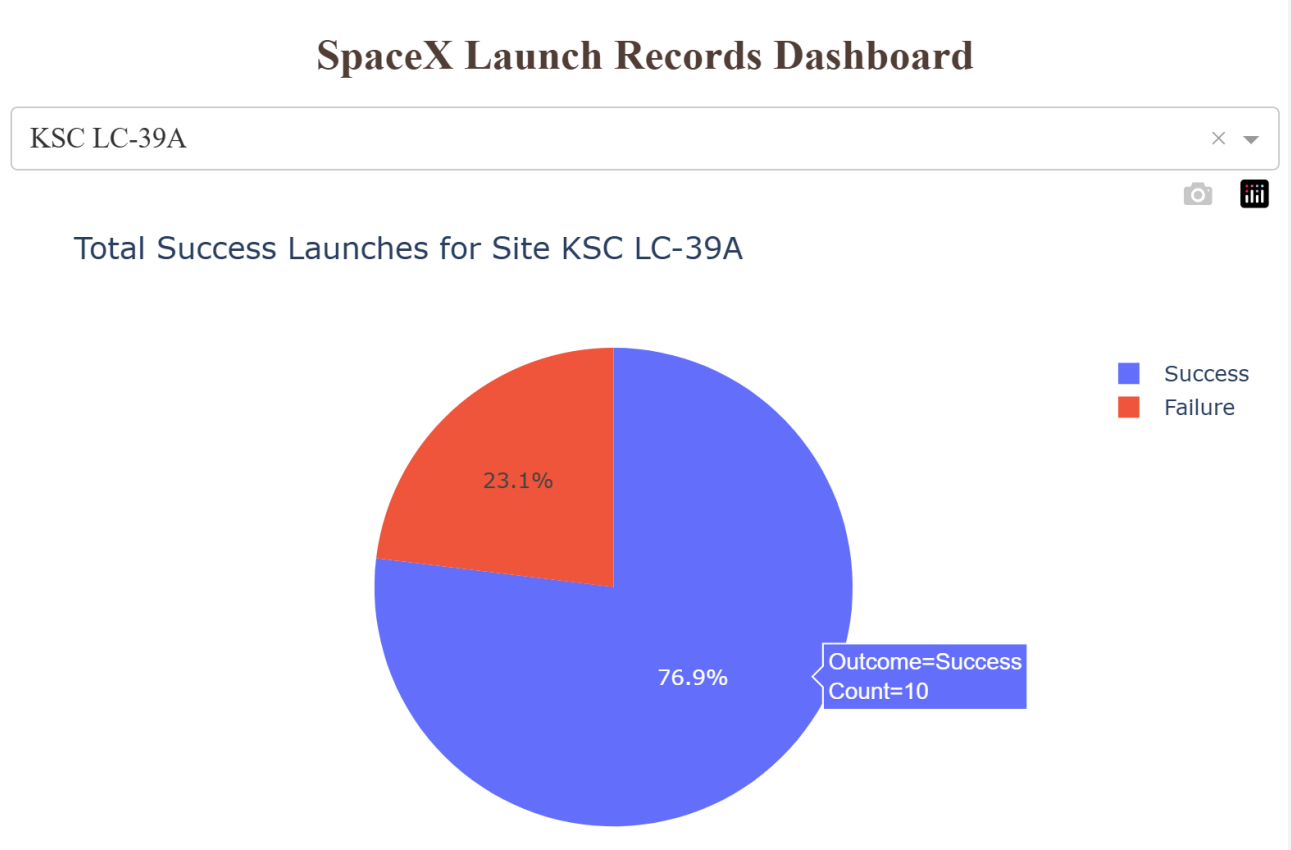
Key Finding: The chart shows KSC LC-39A has the highest success rate of 41.7%, while CCAFS SLC-40 has the lowest with only 12.5%.



Plotly Dash - Highest Launch Dashboard

Based on the overall success analysis from the previous chart, we've isolated KSC LC-39A for a detailed look.

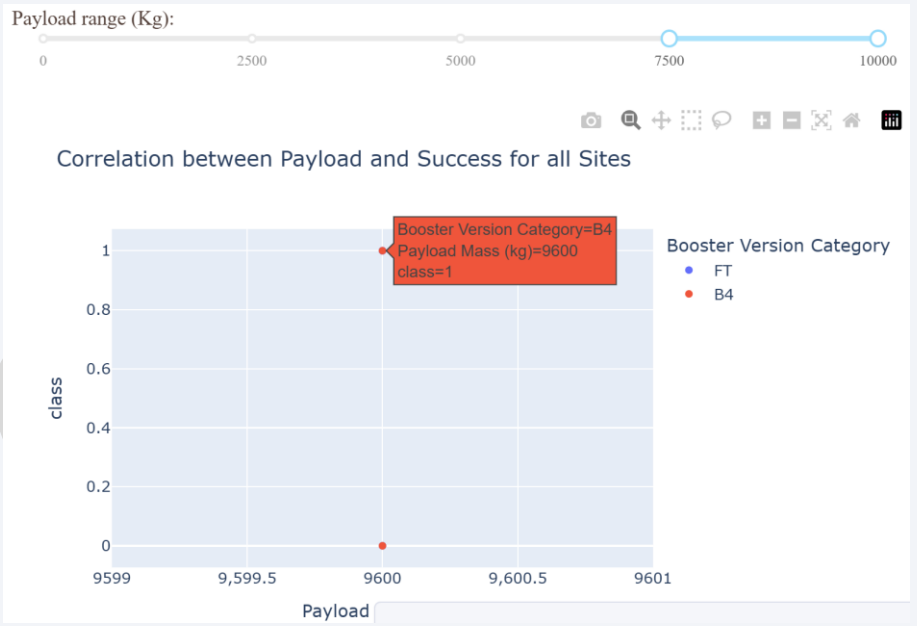
- By selecting KSC LC-39A from the drop-down menu, the pie chart now displays this site's specific mission outcomes:
  - It boasts a 76.9% success rate (10 successful launches).
  - The failure rate is 23.1%.
- The chart includes a title, labels, and a legend, and the total count for success or failure is shown when hovering over the respective slice.
- This high success rate is a key metric for planning future missions.



Plotly Dash - Different Payloads Dashboard

- The scatter plots show the relationship between payload and launch outcome (class) for all sites in different mass payloads selected in the range slider
- The payload range slider can be adjusted to show the desired mass of boosters. The upper line (1) represents success while the lower line (0) represents failure landings. Hovering over the scatter points reveals the booster version, the Payload Mass (kg) and the outcome. The legend helps identify the points of their Falcon 9 booster versions.
- For all ranges, FT booster version appears to have the highest success rate among all other versions of Falcon 9.

Plotly Dash - Different Payloads Dashboard



2500-5000 range has the highest launch success rate:

- For B4, the higher the payload, the higher its success rate is.
- For FT, the rate of success is higher than failure and its payload over 4500 are highly successful.
- For B5, only one mission appears in this range which is successful.
- For v1.1, all missions in this range appear to be a failure.

7500-10000 range has the lowest launch success rate:

- B4 booster's successful landing with a Payload Mass of 9600
- Only 1 failure landing also from B4 booster.

Section 5

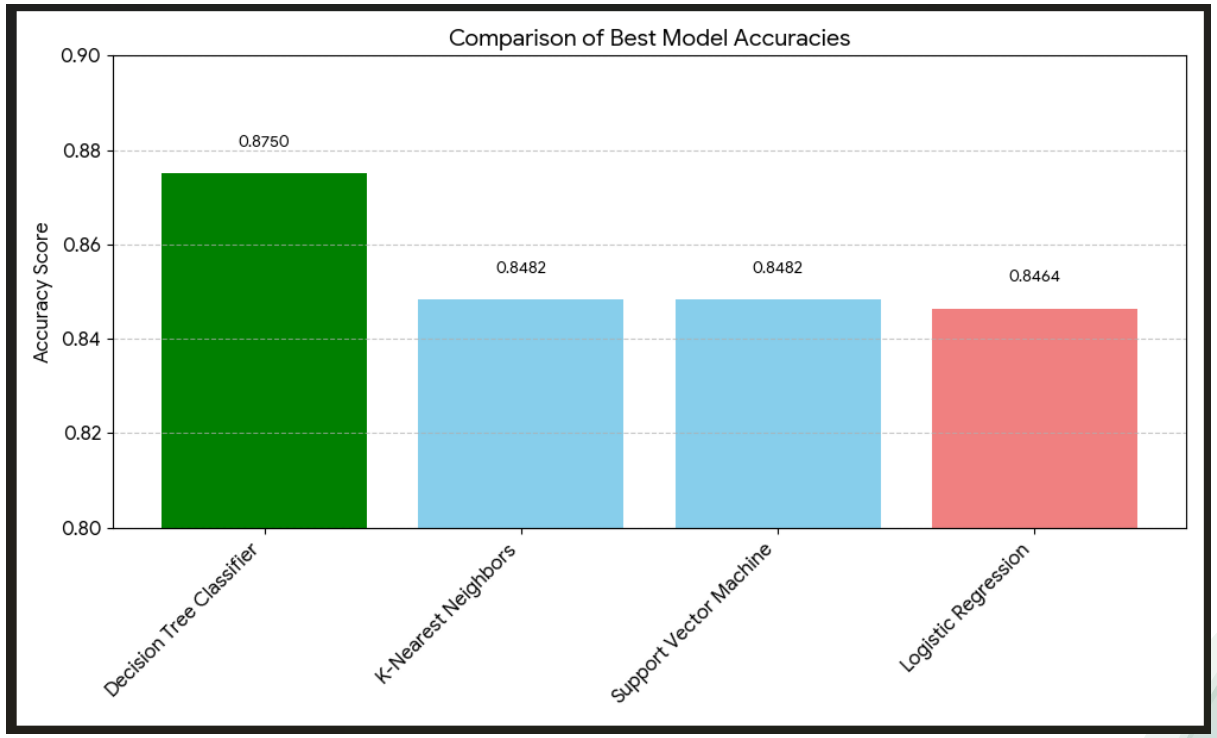
Predictive Analysis (Classification)

Classification Accuracy

The visualization clearly shows the **Decision Tree Classifier** with the highest bar, confirming its superiority based on the optimized, best score accuracy.

Next from the top are K-Nearest Neighbors and Support Vector Machine having the same accuracy results.

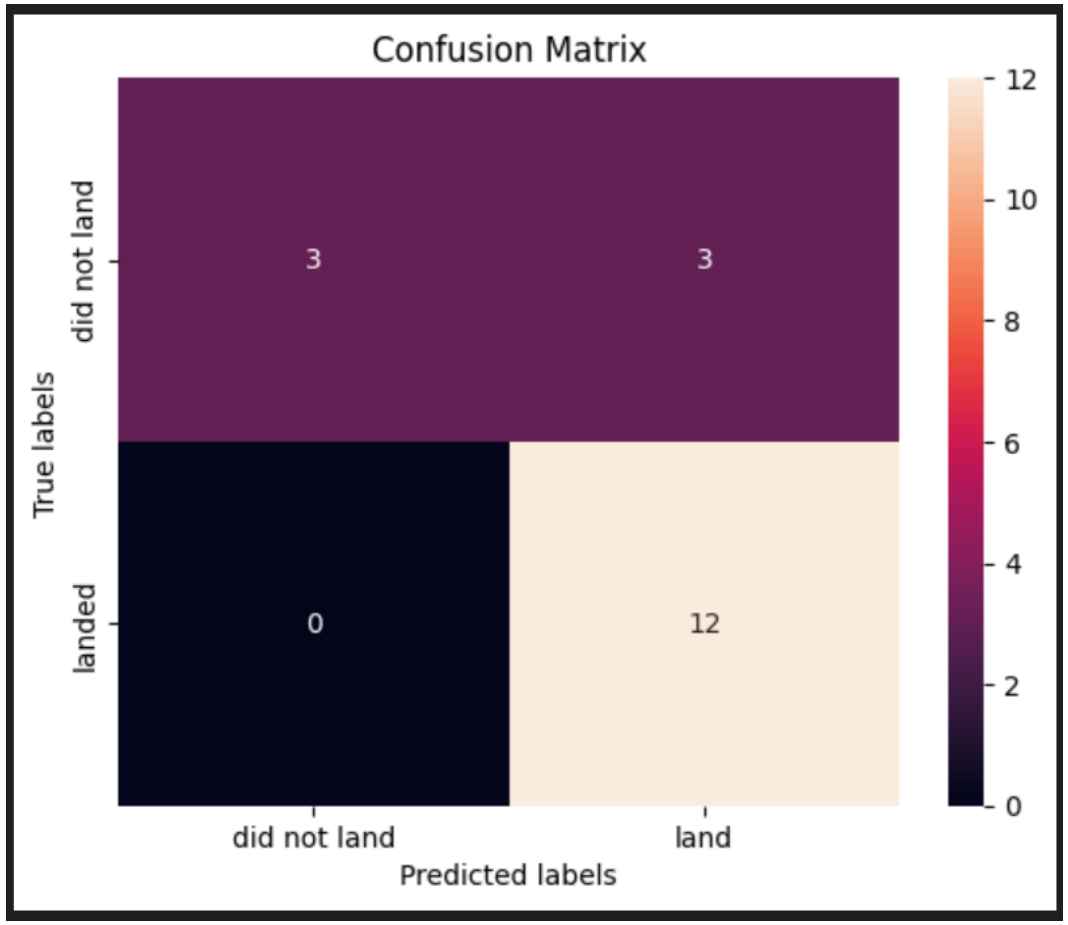
The lowest bar belongs to Logistic Regression.



Confusion Matrix

This is the confusion matrix of the best performing model, Decision Tree Classification.

The primary problem is the high number of **False Positives (3)**. This means three instances that were predicted to be successful landings but actually failed.



Conclusions

This case study successfully demonstrates the application of machine learning to predict the reusability outcome of successful landing of SpaceX Falcon 9 first stage boosters.

The analysis revealed that several mission parameters such as Orbit Type, Launch Site and Payload Mass are the most significant factors influencing landing success.

- **Best Model:** The **Decision Tree Classification** model emerged as the top performer.
  - The accuracy result of 0.83 using the **score()** method for all models means that their default, un-optimized versions performed identically.
  - But once the models were optimized using the **best_score_** accuracy with hyperparameters and cross-validations, the **Decision Tree Classifier** was the **best-performing model**, achieving an accuracy of **87.68%**. The other three models performed very similarly, all in the range of 84.64% to 84.82% accuracy.
  - All classification models generated the same results in the **confusions matrix**. Because the test set is small, only very few data points are available, leading to a small number of possible outcomes. With a small sample size, the probability of different algorithms making the same few errors is much higher.

Conclusions

- **Feature Importance:** Payload Mass, Orbit Type and Launch Site are the most important features identified by models.
  - **Payload Mass** is a direct measure of the total mass the rocket must accelerate (e.g. higher mass generally reduces propellant for landing)
  - **Orbit Type** dictates the energy required for the mission (e.g. higher energy orbits like GEO transfer make recovery harder)
  - **Launch Site** determines the geographical and operational characteristics impacting the landing method (e.g. proximity to drone ship affects fuel)
  - **Flight Number** is a non-causal factor which correlates with success in historical data due to the improvements over time in hardware and software features of the rockets.
- **Practical Utility:** The predictive capability of the model provides a quantitative basis for **risk assessment** and **cost estimation**, allowing companies to forecast the expected launch service cost with greater precision.

Conclusions/Recommendations

To further enhance this predictive capability, future work should focus on:

- Integrating **more data into the training set** to make the task challenging enough to differentiate their performance.
- **Employing other methods** or deep learning techniques like Random Forest to improve generalization and stability.
- **Maximize precision** to mitigate false positive issues to avoid wasted resources and poor decision-making regarding future launches or investments.

Appendix

Appendix A: Build Interactive Dashboard With Plotly Dash

```
min_value = spacex_df['Payload Mass (kg)'].min()
max_value = spacex_df['Payload Mass (kg)'].max()
app.layout = html.Div(children=[html.H1("SpaceX Launch Records Dashboard",
    style={'textAlign': 'center','color': '#503D36', 'font-size': 24}),
html.Div([dcc.Dropdown(id='site-dropdown', options=[
            {'label': 'All Sites', 'value': 'ALL'},
            {'label': 'CCAFS LC-40', 'value': 'CCAFS LC-40'},
            {'label': 'VAFB SLC-4E', 'value': 'VAFB SLC-4E'},
            {'label': 'KSC LC-39A', 'value': 'KSC LC-39A'},
            {'label': 'CCAFS SLC-40', 'value': 'CCAFS SLC-40'},],
        value='ALL', placeholder="Select a Launch Site here", searchable=True),
html.Div([dcc.Graph(id='success-pie-chart')]),
#Add a Range Slider to Select Payload
html.Div([html.Label("Payload range (Kg):"),
    dcc.RangeSlider(id='payload-slider', min=0, max=10000, step=1000,
        marks={0: '0', 2500: '2500', 5000: '5000', 7500: '7500', 10000: '10000'},
        value=[min_value, max_value])], style={'padding': '20px 0px', 'color': '#503D36'}),
html.Div([dcc.Graph(id='success-payload-scatter-chart')]),])])
```

Appendix

```
#Add a Callback Function to render success-pie-chart based on selected site dropdown
@app.callback(
    Output(component_id='success-pie-chart', component_property='figure'),
    Input(component_id='site-dropdown', component_property='value'))
def get_pie_chart(entered_site):
    filtered_df = spacex_df
    if entered_site == 'ALL':
        fig = px.pie(filtered_df, values='class',
        names='Launch Site',
        title='Total Success Launches by Site')
    else:
        # Filter for the selected site
        site_df = filtered_df[filtered_df['Launch Site'] == entered_site]
        # Count success (1) and failure (0)
        site_counts = site_df['class'].value_counts().reset_index()
        site_counts.columns = ['Outcome', 'Count']
        site_counts['Outcome'] = site_counts['Outcome'].map({1: 'Success', 0: 'Failure'})
        fig = px.pie(site_counts, values='Count', names='Outcome',
            title=f'Total Success Launches for Site {entered_site}')
return fig
```

Appendix

```
@app.callback(
        Output(component_id='success-payload-scatter-chart', component_property='figure'),
        [Input(component_id='site-dropdown', component_property='value'),
        Input(component_id='payload-slider', component_property='value')])
def get_scatter_chart(entered_site, payload_range):
        low, high = payload_range
        mask = (spacex_df['Payload Mass (kg)'] >= low) & (spacex_df['Payload Mass (kg)'] <= high)
        filtered_df = spacex_df[mask]
        if entered_site == 'ALL':
                fig = px.scatter(filtered_df, x='Payload Mass (kg)', y='class',  color='Booster Version Category',
                        title='Correlation between Payload and Success for all Sites')
        else:
                site_df = filtered_df[filtered_df['Launch Site'] == entered_site]
                fig = px.scatter(site_df,  x='Payload Mass (kg)', y='class', color='Booster Version Category',
                        title=f'Correlation between Payload and Success for site {entered_site}')
return fig
app.run(debug=True, port=8051)
```

Appendix

Appendix B: Reference for Data Sources:

- Falcon 9. (2025, October 28). In *Wikipedia*. Retrieved October 28, 2025, from https://en.wikipedia.org/wiki/Falcon_9
- List of Falcon 9 and Falcon Heavy Launches. (2025, October 28). In *Wikipedia*. Retrieved October 28, 2025, from

https://en.wikipedia.org/wiki/List_of_Falcon_9_and_Falcon_Heavy_launches

Thank you!