Predicting Success: A Data-Driven Analysis of SpaceX Falcon 9 Landings

Using Machine Learning

IBM *Certification Course Project*
By: *James Montoya*
Github Repo

Image source: https://www.google.com.co/url?sa=i&url=https%3A%2F%2Fwww.environmentenergyleader.com%2F2024%2F04%2Fspacex-reuses-falcon-9%2F&psig=AOvVaw2nbyyOzCOolCbO9VKHDdgF&ust=1725847008515000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCJD0qKSfsogDFQAAAAAdAAAAABAE

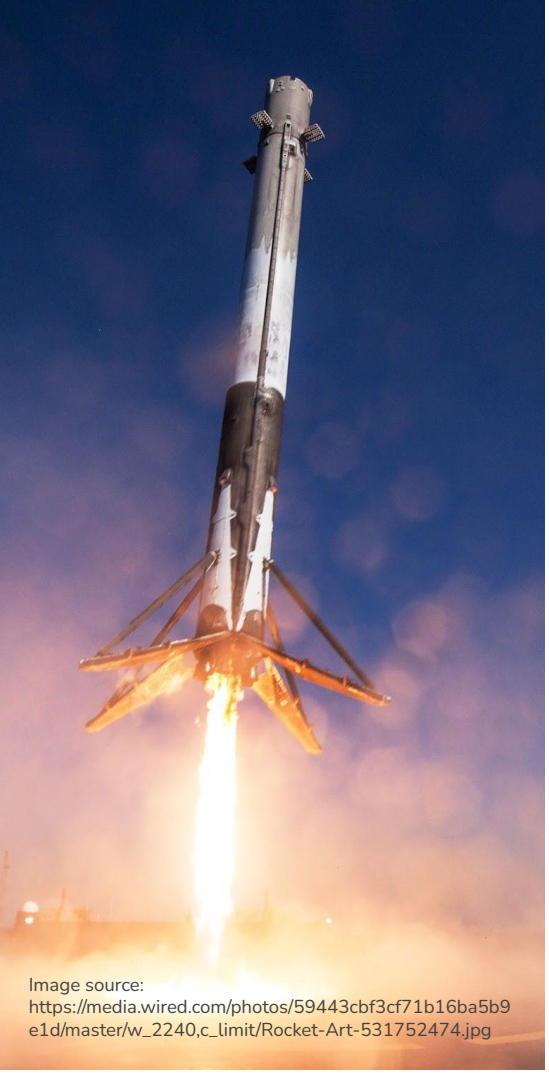


Image source: https://media.wired.com/photos/59443cbf3cf71b16ba5b9e1d/master/w_2240,c_limit/Rocket-Art-531752474.jpg

Executive Summary

SpaceX has revolutionized the commercial space industry with its cost-effective Falcon 9 rocket launches, made affordable by reusing the rocket's first stage. To compete in this market, predicting whether the Falcon 9's first stage will land successfully is crucial, as it directly impacts launch costs.

This analysis adopts a data science approach to predict landing success. We collected and cleaned public data on Falcon 9 launches, performed exploratory data analysis (EDA) to identify key factors influencing landings, and developed predictive models using machine learning techniques. The resulting insights can guide new competitors like "Space Y" in strategic decision-making and cost optimization.

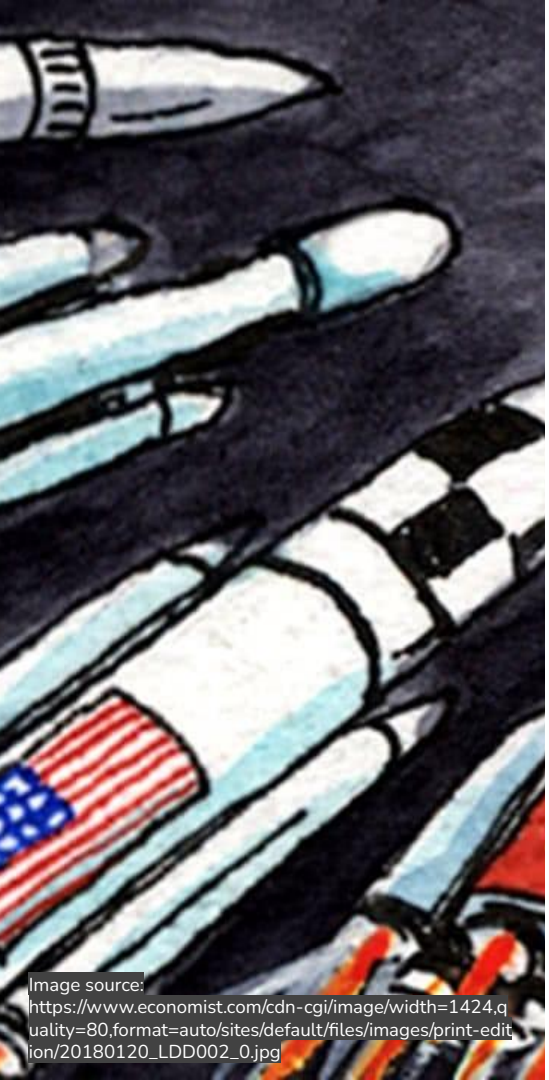


Image source: https://www.economist.com/cdn-cgi/image/width=1424,quality=80,format=auto/sites/default/files/images/print-edition/20180120_LDD002_0.jpg

Introduction

The commercial space age has begun, with private companies making space travel more accessible and affordable. Companies like Virgin Galactic, Rocket Lab, and Blue Origin are pioneering different aspects of space exploration. However, SpaceX stands out as one of the most successful players in the field. SpaceX's notable achievements include missions to the International Space Station, the creation of the Starlink satellite internet constellation, and launching manned missions into space.

One of the reasons for SpaceX's success is its relatively low-cost rocket launches. The Falcon 9 rocket, for example, is offered at $62 million per launch, while other providers charge upwards of $165 million. A significant part of these savings comes from SpaceX's ability to reuse the first stage of its rockets. This first stage does most of the heavy lifting to propel the payload toward orbit. Its recovery is crucial to maintaining low launch costs, but it is not always guaranteed due to mission parameters or technical difficulties.

In this presentation, we will analyze data from **SpaceX** for a the new competitor **Space Y.** Our goal is to estimate the cost of rocket launches and predict whether the Falcon 9's first stage will successfully land and be reusable. Instead of relying solely on rocket science, we will use machine learning models trained on public data to make these predictions and help our team develop competitive strategies.

Image source: *Image created by DALL-E, September 2024.*

Data Collection and Data Wrangling

The data for this project was collected from publicly available sources, including API SpaceX's sources, cvs files and web scraping methods using beautifulsoup in wikipedia website. The data includes information on each Falcon 9 launch, such as launch date, landing site, payload mass, weather conditions, and landing success among others.

Data Acquisition

Collecting data from various sources, ensuring data quality, and performing initial cleaning.

Data Cleaning and Transformation

Addressing null values, filtering values, and transforming data into a suitable format for machine learning algorithms.

Data Exploration

Analyzing the collected data to identify patterns, relationships, and potential features for the model.

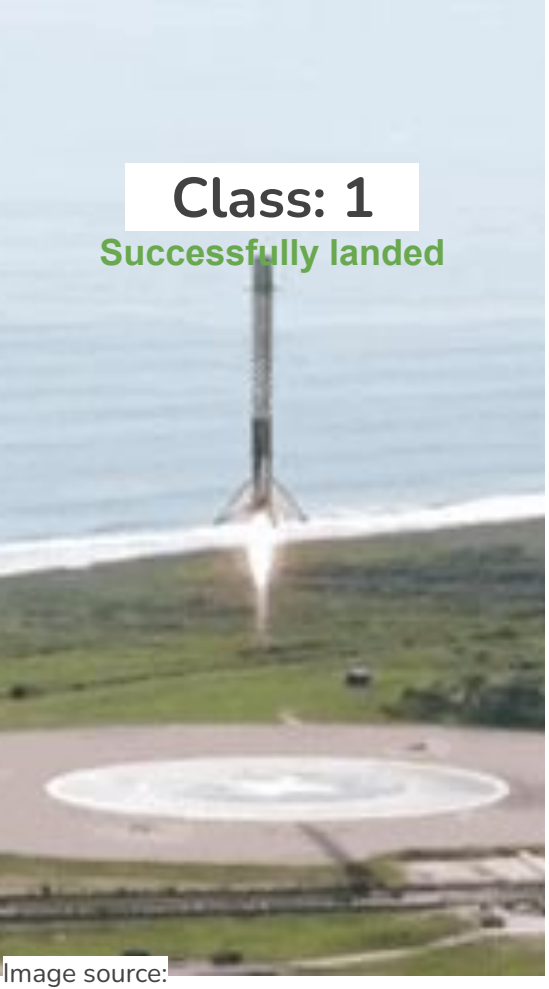


Image source: https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DS0701EN-SkillsNetwork/api/Images/landing_1.gif

Outcomes predicted

The goal is to classify the landing outcomes of the Falcon 9 boosters into two distinct categories:

- **Class 0:** Represents a *bad outcome* where the booster did not successfully land.
- **Class 1:** Represents a *good outcome* where the booster successfully landed.

The classification variable, **Y**, will be used to indicate the outcome of each launch, with values of either 0 or 1, providing a clear, binary distinction between successful and unsuccessful landings. This classification will help in analyzing and predicting future landing success rates for cost optimization and competitive positioning.

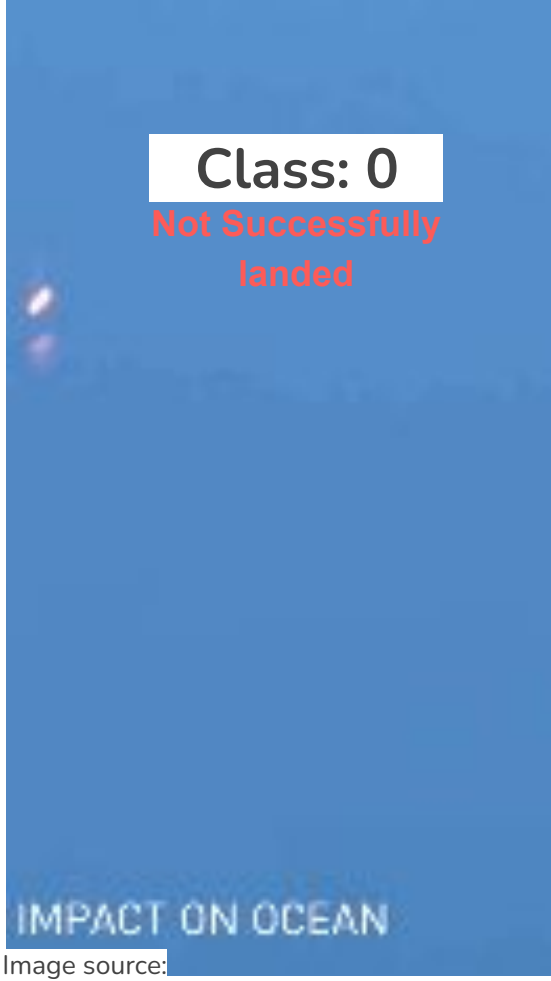


Image source: https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DS0701EN-SkillsNetwork/api/Images/crash.gif

EDA and interactive visual analytics methodology

Exploratory Data Analysis

Image source: local python file running in Visual Studio Code (1.92.2) in Macbook Pro M1

First, let's read the SpaceX dataset into a Pandas dataframe and print its summary

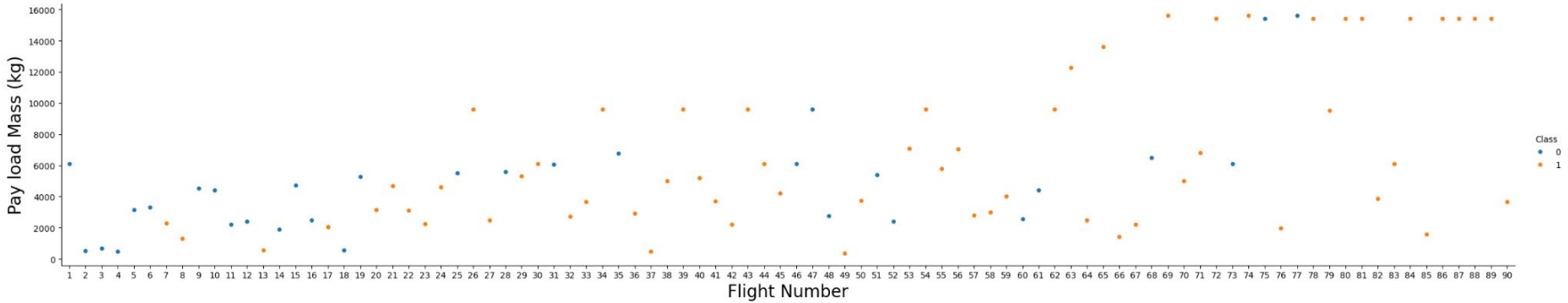
```
from js import fetch
import io

URL = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_2.csv"
resp = await fetch(URL)
dataset_part_2_csv = io.BytesIO((await resp.arrayBuffer()).to_py())
df=pd.read_csv(dataset_part_2_csv)
df.head(5)
```

Python

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577366	28.561857	0
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366	28.561857	0
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.577366	28.561857	0
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	0	B1003	-120.610829	34.632093	0
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561857	0

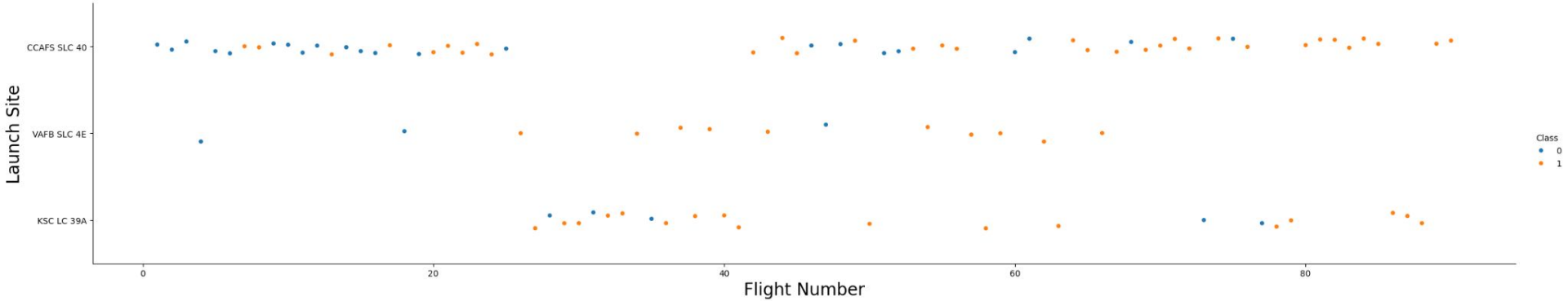
This data is intended to help analyze the factors influencing the success or failure of SpaceX's rocket landings. The columns provide comprehensive details about each launch, such as the payload mass, launch site, and booster specifications, which can be used to build predictive models and gain insights into the landing outcomes.

Relationship between Flight Number (x-axis) and Payload Mass (kg) (y-axis)



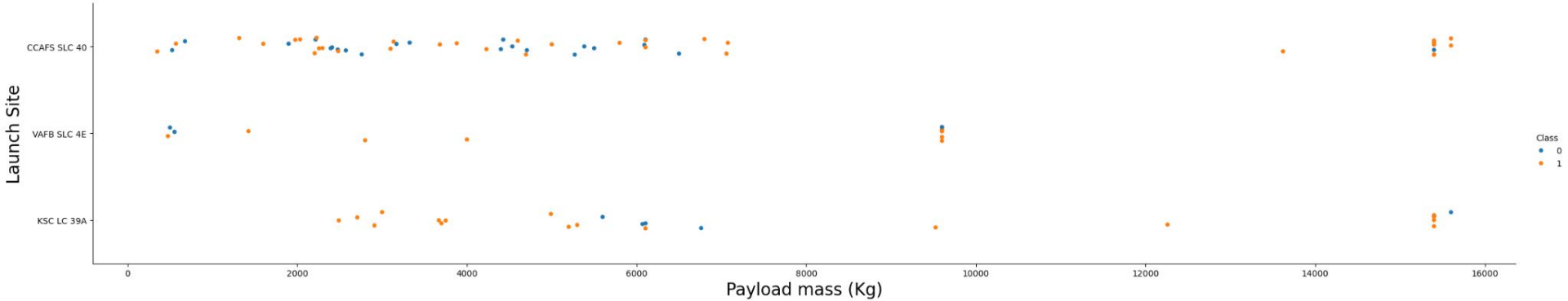
The chart shows that SpaceX’s landing success rate improves over time, regardless of payload mass. While payload mass varies, it does not directly determine success, suggesting other factors also influence landing outcomes.

Relationship between Flight Number and Launch Site



The success of the Falcon 9 first stage landings tends to improve over time, with variations between different launch sites. The data suggests that some sites, particularly KSC LC 39A, may offer conditions or configurations more conducive to successful landings in recent flights.

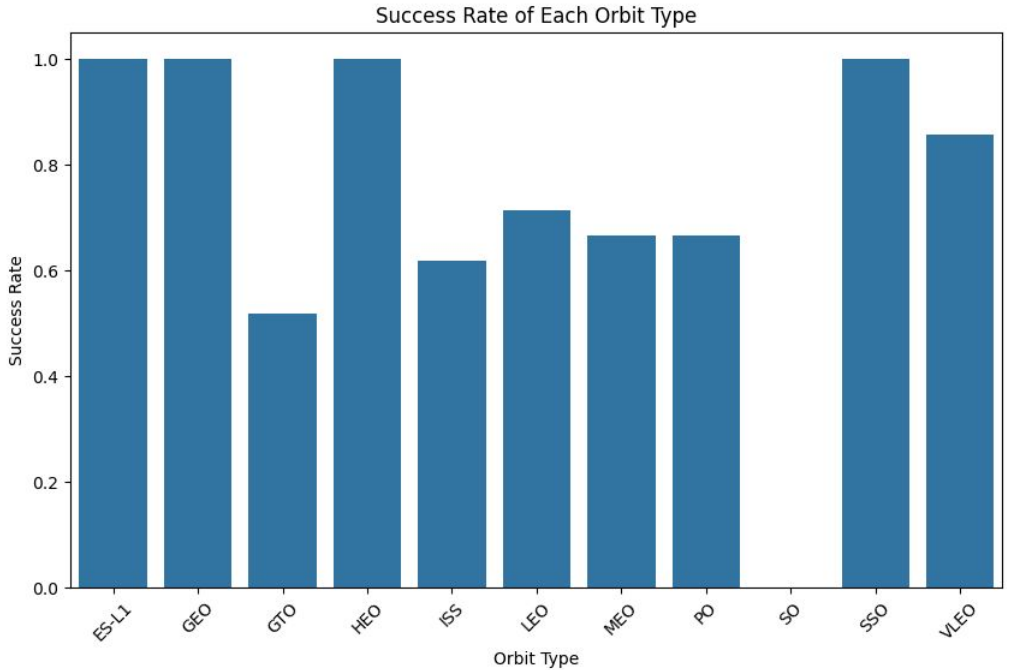
Relationship between Payload Mass and Launch Site



Launch sites are chosen partly based on payload mass, with heavier payloads launching from specific sites. Landing success appears to be independent of payload mass, indicating that other factors contribute to the outcomes.

Observing Payload Mass Vs. Launch Site scatter point chart we can see that the VAFB-SLC launchsite there are no rockets launched for heavy payload mass(greater than 10000).

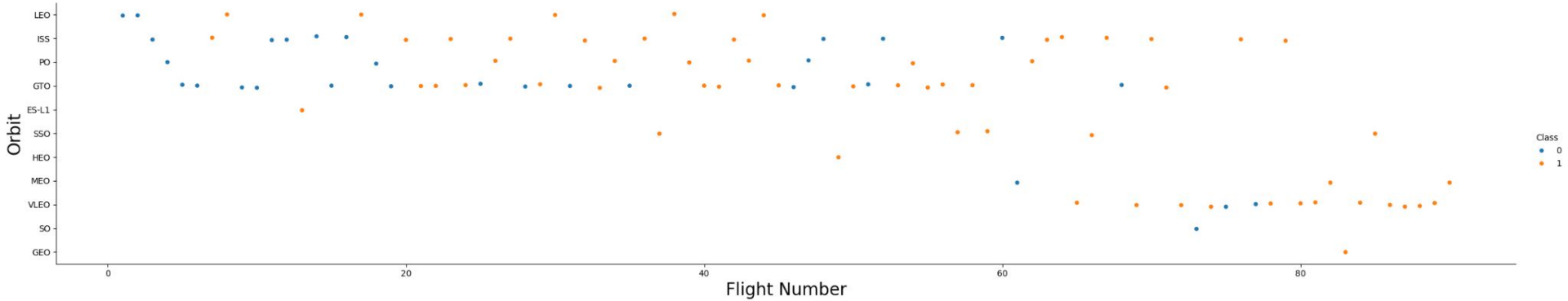
Relationship between success rate of each orbit type



ES-L1, SSO, and VLEO have the highest success rates, close to or at 100%. This suggests that launches targeting these orbits have been very reliable in achieving successful outcomes.

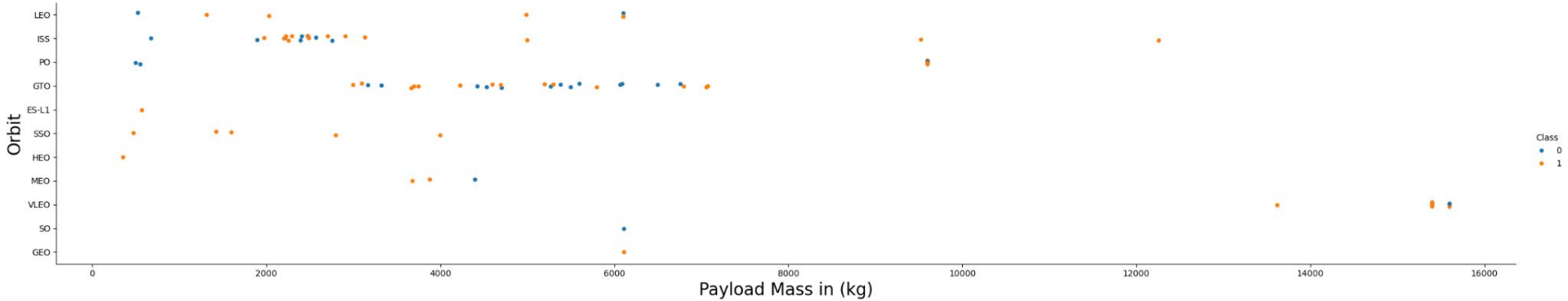
Orbit type plays a critical role in the success rate of launches. Some orbits like ES-L1, SSO, and VLEO are more consistently successful, while others, such as GEO and HEO, present more challenges.

Relationship relationship between FlightNumber and Orbit type



We can observe that in the LEO orbit, success seems to be related to the number of flights. Conversely, in the GTO orbit, there appears to be no relationship between flight number and success.

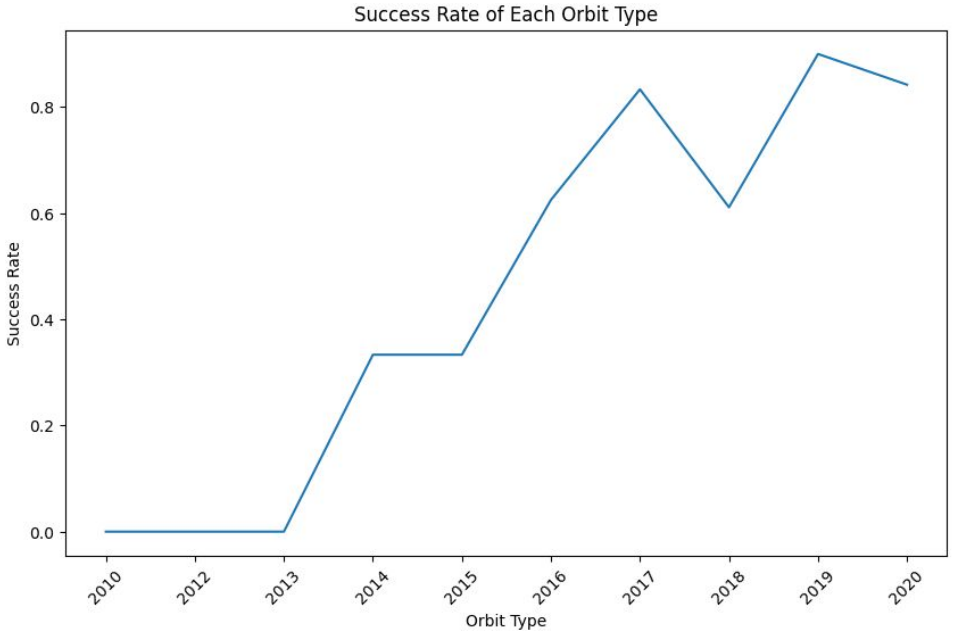
Relationship between Payload Mass and Orbit type



With heavy payloads the successful landing or positive landing rate are more for Polar,LEO and ISS.

However, for GTO, it's difficult to distinguish between successful and unsuccessful landings as both outcomes are present.

Launch success yearly trend



We can observe that the success rate since 2013 kept increasing till 2020

Exploratory Data Analysis (EDA) Using SQL Queries

Another effective method for data analysis in a notebook environment is using SQL queries. This approach enables us to execute SQL statements directly within the notebook cells, leveraging a simple wildcard to interact with the data seamlessly.

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

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

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

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

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

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

Date	Time (UTC)	Booster_Version	Launch_Site	
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	

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

Exploratory Data Analysis (EDA) Using SQL Queries

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

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

Customer	total_payload_mass
NASA (CRS)	45596

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

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

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

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

Booster_Version	average_payload_mass
F9 v1.1 B1003	2534.6666666666665

Exploratory Data Analysis (EDA) Using SQL Queries

```
%%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) ;
```

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

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

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

```
%%sql
SELECT
    CASE strftime('%m', "Date")
        WHEN '01' THEN 'January'
        WHEN '02' THEN 'February'
        WHEN '03' THEN 'March'
        WHEN '04' THEN 'April'
        WHEN '05' THEN 'May'
        WHEN '06' THEN 'June'
        WHEN '07' THEN 'July'
        WHEN '08' THEN 'August'
        WHEN '09' THEN 'September'
        WHEN '10' THEN 'October'
        WHEN '11' THEN 'November'
        WHEN '12' THEN 'December'
    END AS monthname,
    Landing_Outcome,
    Booster_Version,
    Launch_Site
FROM SPACEXTABLE WHERE substr(Date,0,5)='2015' AND Landing_Outcome = 'Failure (drone ship)';
```

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

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

Displaying the List of records which will display the month names, failure landing_outcomes in drone ship ,booster versions, launch_site for the months in year 2015.

Image source: *Image created by DALL-E, September 2024.*

Features Engineering

Getting preliminary insights about how each important variable would affect the success rate, we will select the features that will be used in success prediction in the future module.

```
features = df[['FlightNumber', 'PayloadMass', 'Orbit', 'LaunchSite', 'Flights', 'GridFins', 'Reused', 'Legs', 'LandingPad', 'B
features.head()
```

	FlightNumber	PayloadMass	Orbit	LaunchSite	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial
0	1	6104.959412	LEO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B0003
1	2	525.000000	LEO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B0005
2	3	677.000000	ISS	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B0007
3	4	500.000000	PO	VAFB SLC 4E	1	False	False	False	NaN	1.0	0	B1003
4	5	3170.000000	GTO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B1004

Now we apply the one hot encode to create dummies columns in order to prepare the data for the machine learning algorithms.

```
# HINT: Use get_dummies() function on the categorical columns
# Create dummy variables for the categorical columns
features_one_hot = pd.get_dummies(features, columns=['Orbit', 'LaunchSite', 'LandingPad', 'Serial'])

# Display the results to verify
features_one_hot.head()
```

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	...	Serial_B1048	Serial_B1049
0	1	6104.959412	1	False	False	False	1.0	0	False	False	...	False	False
1	2	525.000000	1	False	False	False	1.0	0	False	False	...	False	False
2	3	677.000000	1	False	False	False	1.0	0	False	False	...	False	False
3	4	500.000000	1	False	False	False	1.0	0	False	False	...	False	False

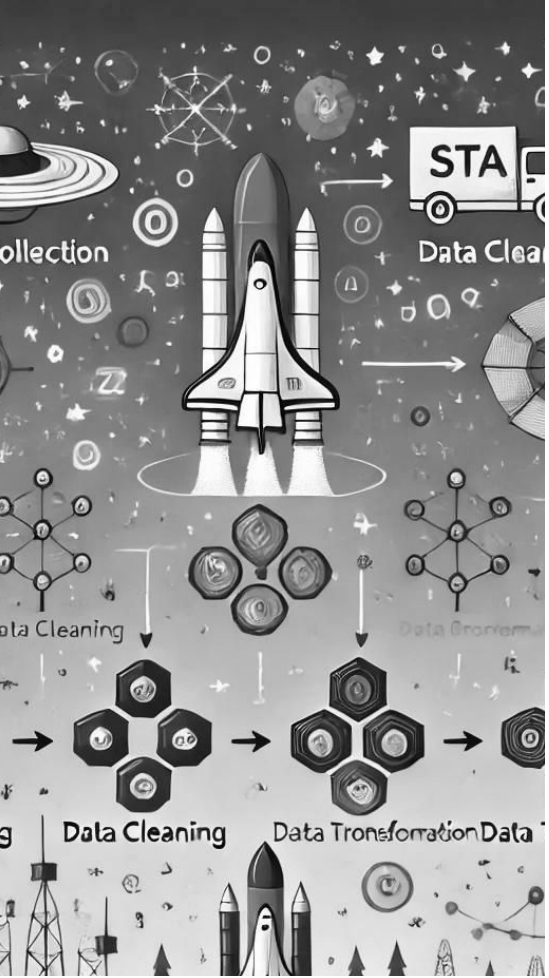


Image source: *Image created by DALL-E, September 2024.*

Predictive analysis methodology

Objective: To develop a machine learning model that predicts the success of Falcon 9 first-stage landings.

Data Preparation: Collected and cleaned SpaceX launch data, including features like flight number, payload mass, launch site, orbit type, and booster specifications.

Feature Engineering: Selected key features influencing landing outcomes, such as weather conditions, payload mass, and launch configurations, and transformed them into a suitable format for model training. Use One hot encoding.

Data Standardization: Standardized the dataset to ensure all features are on a comparable scale, improving model performance and convergence speed.

Model Selection: Evaluated various classification algorithms, including Logistic Regression, Support Vector Machine, Decision Trees, and k nearest neighbors KNN, to find the most accurate predictor, using a GridSearchCV object to optimize hyperparameters for each model.

Model Training and Evaluation: Split the data into training and test sets, trained the models, and evaluated their performance using metrics like accuracy score and the confusion matrix to assess true vs. false predictions.

Outcome: Identified the two best-performing model for predicting landing success, providing actionable insights to optimize future SpaceX missions.

Predictive analysis (classification)

Getting (y) Class

```
#from js import fetch
import requests
import io

# URL of the CSV file
URL1 = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_2.csv"

# Fetch the data from the URL
response = requests.get(URL1)

# Convert the content to a pandas DataFrame
data = pd.read_csv(io.BytesIO(response.content))

# Display the first few rows of the data
print(data.head())
```

✓ 1.3s

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

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

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

Fetching data ready from url1 in order to get the predicted class that we want to predict with the algorithms of ML.

TASK 1

Create a NumPy array from the column `Class` in `data`, by applying the method `to_numpy()` then assign it to the variable `Y`,mak

```
y = data['Class'].to_numpy()
y
```

[150] ✓ 0.0s

```
array([0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1,
       1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1])
```

Predictive analysis (classification)

Getting (X) Features

Fetching data ready from url2 to get X or the features that will help us to predict the y class

```
# URL of the CSV file
URL2 = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_3.csv'

# Fetch the data from the URL
response2 = requests.get(URL2)

# Convert the content to a pandas DataFrame
X = pd.read_csv(io.BytesIO(response2.content))
```

[148] ✓ 1.4s Python

```
X.head(100)
```

[149] ✓ 0.0s Python

dMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	...	Serial_B1058	Serial_B1059	Serial_B1060	Serial_B1062	GridFins_False	GridFins_True	Reused_False	Reused_True	Legs_False	Legs_True
59412	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
00000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
00000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
00000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
00000	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
00000	2.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0
00000	3.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0
00000	6.0	5.0	5.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0
00000	3.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0
00000	1.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0

Predictive analysis (classification)

Data Standardization

TASK 2

Standardize the data in `X` then reassign it to the variable `X` using the transform provided below.

```
# students get this
from sklearn.preprocessing import StandardScaler
#transform = preprocessing.StandardScaler()
transform = StandardScaler().fit_transform(X)
transform
```

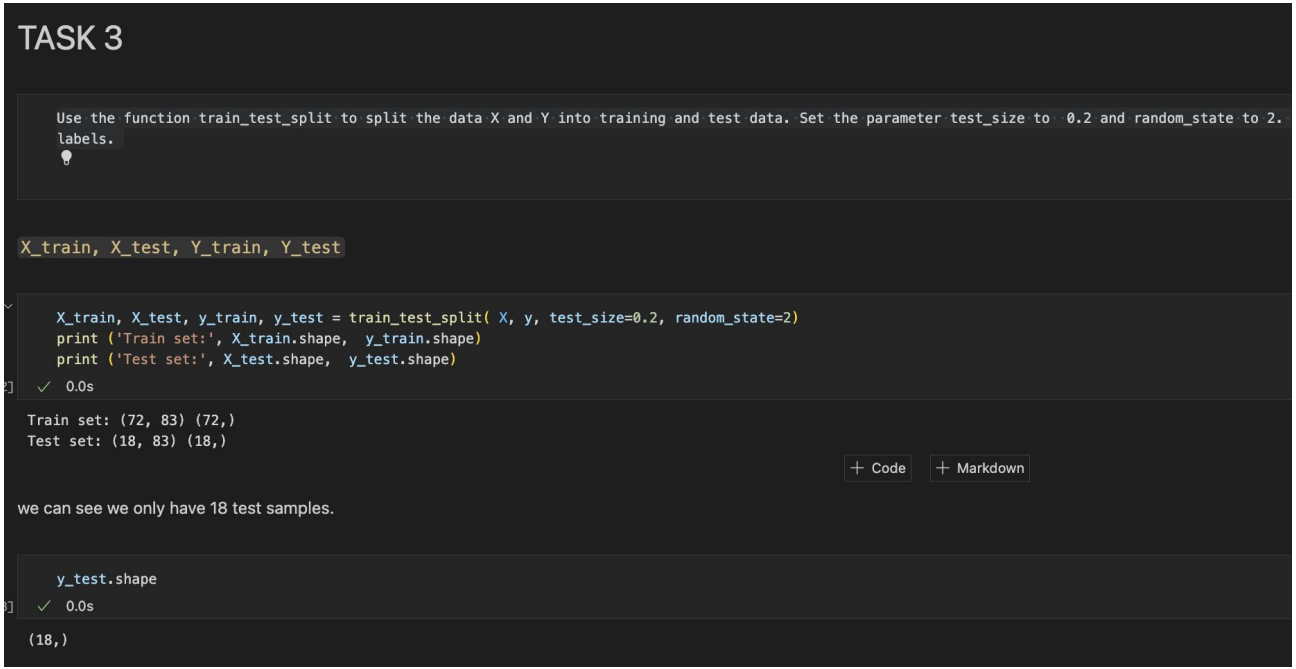
[151] ✓ 0.0s

```
array([[-1.71291154e+00, -1.94814463e-16, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       [-1.67441914e+00, -1.19523159e+00, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       [-1.63592675e+00, -1.16267307e+00, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       ...,
       [ 1.63592675e+00,  1.99100483e+00,  3.49060516e+00, ...,
         1.19684269e+00, -5.17306132e-01,  5.17306132e-01],
       [ 1.67441914e+00,  1.99100483e+00,  1.00389436e+00, ...,
         1.19684269e+00, -5.17306132e-01,  5.17306132e-01],
       [ 1.71291154e+00, -5.19213966e-01, -6.53912840e-01, ...,
        -8.35531692e-01, -5.17306132e-01,  5.17306132e-01]])
```

Standardizing (X) to prepare the data for using machine learning algorithms.

Predictive analysis (classification)

Splitting Data



Using the function train_test_split to split the data X and Y into training and test data. Set the parameter test_size to 0.2 and random_state to 2. The training data and test data should be assigned to the variables.

- 80% data for Training 72 rows
- 20% data for Testing 18 rows

Predictive analysis (classification)

Using GridSearchCV to fine-tune hyperparameters for each model, ensuring optimal performance.

GridSearchCV is used to optimize the hyperparameters of machine learning models to achieve the best possible performance.

This object will be runned for these algorithms: Logistic Regression, Support Vector Machine, Decision Trees, and k nearest neighbors KNN

Logistic Regression

Using GridSearchCV

TASK 4

Create a logistic regression object then create a GridSearchCV object `logreg_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

[154] ✓ 0.0s

```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()
# Create a GridSearchCV object with logistic regression and parameters, set cv = 10
logreg_cv = GridSearchCV(estimator=lr, param_grid=parameters, cv=10)
# Fit the GridSearchCV object to find the best parameters
logreg_cv.fit(X_train, y_train)
```

[155] ✓ 0.1s

After run GridSearchCV we get the best parameters for LR in the variable **`logreg_cv.best_params_`**

GridSearchCV
estimator: LogisticRegression
LogisticRegression

We output the `GridSearchCV` object for logistic regression. We c… the best parameters using the data attribute `best_para`

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

[156] ✓ 0.0s

```
tuned hpyerparameters :(best parameters)  {'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8196428571428571
```

Logistic Regression

Accuracy and Confusion matrix

TASK 5

Calculate the accuracy on the test data using the method sc

```
from sklearn.metrics import accuracy_score

# Assume you have a GridSearchCV object for logisti
# Get the best parameters from the GridSearchCV res
best_lr_pa = logreg_cv.best_params_

# Initialize a new Logistic Regression model with t
lr2 = LogisticRegression(**best_lr_pa)

# Train the new Logistic Regression model on the fu
lr2.fit(X_train, y_train)

# Predict using the test dataset
y_pred_lr = lr2.predict(X_test)

# Calculate the accuracy
accuracy_lr = accuracy_score(y_test, y_pred_lr)
print("Accuracy on the test data for Logisitc Regression:", accuracy_lr)
```

57] ✓ 0.0s

```
Accuracy on the test data for Logisitc Regression: 0.8333333333333334
```

With the best parameters:

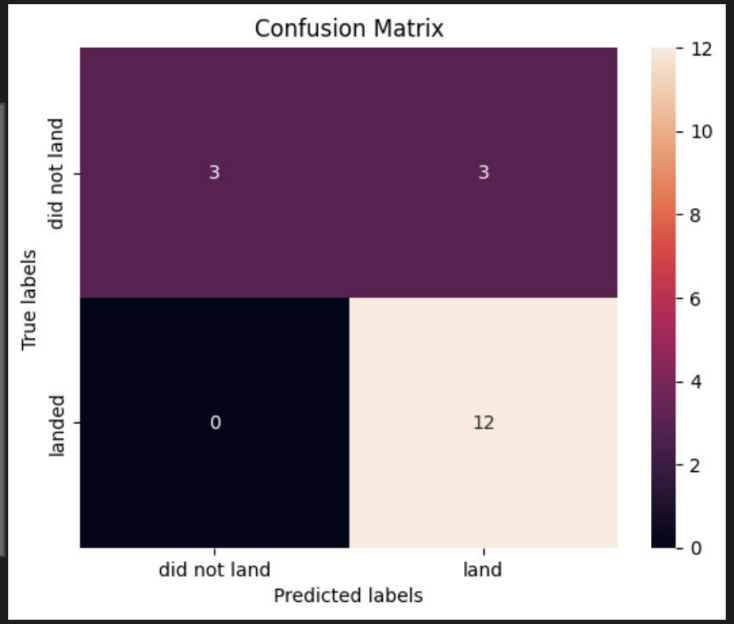
`logreg_cv.best_params_`

we set the model up and now we can train it with the train set. Then we can predict using the test set to get the accuracy of the model:

Accuracy: 83%_

```
yhat=lr2.predict(X_test)
plot_confusion_matrix(y_test,yhat)
```

✓ 0.0s



Examining the confusion matrix, we see that logistic regression can distinguish between the

Overview:

True Postive - 12 (True label is landed, Predicted label is also landed)

False Postive - 3 (True label is not landed, Predicted label is landed)

Support Vector Machine

Using GridSearchCV

TASK 6

Create a support vector machine object then create a `GridSearchCV` object `svm_cv` with cv = 10. Fit the o

```
parameters = {'kernel':('linear', 'rbf', 'sigmoid'), #
              'C': np.logspace(-3, 3, 3),
              'gamma':np.logspace(-3, 3, 3)}
svm = SVC()
```

59] ✓ 0.0s

```
from sklearn.model_selection import RandomizedSearchCV

svm_cv = RandomizedSearchCV(estimator=svm, param_distributions=parameters, cv=3, n_iter=5, n_jo
svm_cv.fit(X_train, y_train)
```

60] ✓ 3.2s

▸ RandomizedSearchCV
▸ estimator: SVC
▸ SVC

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

61] ✓ 0.0s

```
tuned hpyerparameters :(best parameters)  {'kernel': 'sigmoid', 'gamma': 1.0, 'C': 1000.0}
accuracy : 0.6666666666666666
```

After run GridSearchCV we get the best parameters for SVM in the variable **`svm_cv.best_params_`**

Support Vector Machine

Accuracy and Confusion matrix

TASK 7

Calculate the accuracy on the test data using the method sc

```
from sklearn.metrics import accuracy_score

# Assume you have a GridSearchCV object for SVM cal
# Get the best parameters from the GridSearchCV res
best_params_svm = svm_cv.best_params_

# Initialize a new SVM model with the best paramete
svm_best = SVC(**best_params_svm)

# Train the new SVM model on the full training data
svm_best.fit(X_train, y_train)

# Predict using the test dataset
y_pred_svm = svm_best.predict(X_test)

# Calculate the accuracy
accuracy_svm = accuracy_score(y_test, y_pred_svm)
print("Accuracy on the test data for SVM:", accuracy_svm)
```

62] ✓ 0.0s

Accuracy on the test data for SVM: 0.6666666666666666

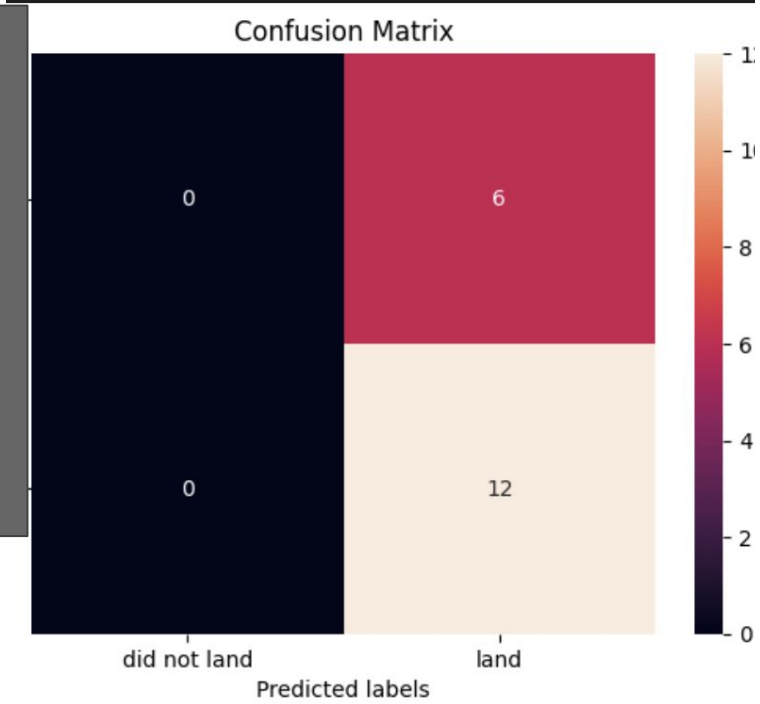
With the best parameters:

`svm_cv.best_params_`

we set the model up and now get train it with the train set. Then we can predict using the test set to get the accuracy of the model:

Accuracy: 66%

```
t=svm_best.predict(X_test)
t_confusion_matrix(y_test,yhat)
1s
```



Decision Tree

Using GridSearchCV

TASK 8

Create a decision tree classifier object then create a `GridSearchCV` object `tree_c`

```
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()

# Create a GridSearchCV object
tree_cv = GridSearchCV(estimator=tree,

# Fit the GridSearchCV object to find t
tree_cv.fit(X_train, y_train)
```

After run GridSearchCV we get the best parameters for SVM in the variable **`tree_cv.best_params_`**

Decision Tree

Accuracy and Confusion matrix

TASK 9

Calculate the accuracy of tree_cv on the test data us

```
# Assume you have a GridSearchCV object for
# Get the best parameters from the GridSear
best_params_tree = tree_cv.best_params_

# Initialize a new Decision Tree model with
tree_best = DecisionTreeClassifier(**best_pa

# Train the new Decision Tree model on the
tree_best.fit(X_train, y_train)

# Predict using the test dataset
y_pred_tree = tree_best.predict(X_test)
```

[104] ✓ 0.0s

```
from sklearn.metrics import accuracy_score
# Calculate the accuracy
accuracy_tree = accuracy_score(y_test, y_pred_tree)
print("Accuracy on the test data for Decision Tree:", accuracy_tree)
```

[166] ✓ 0.0s

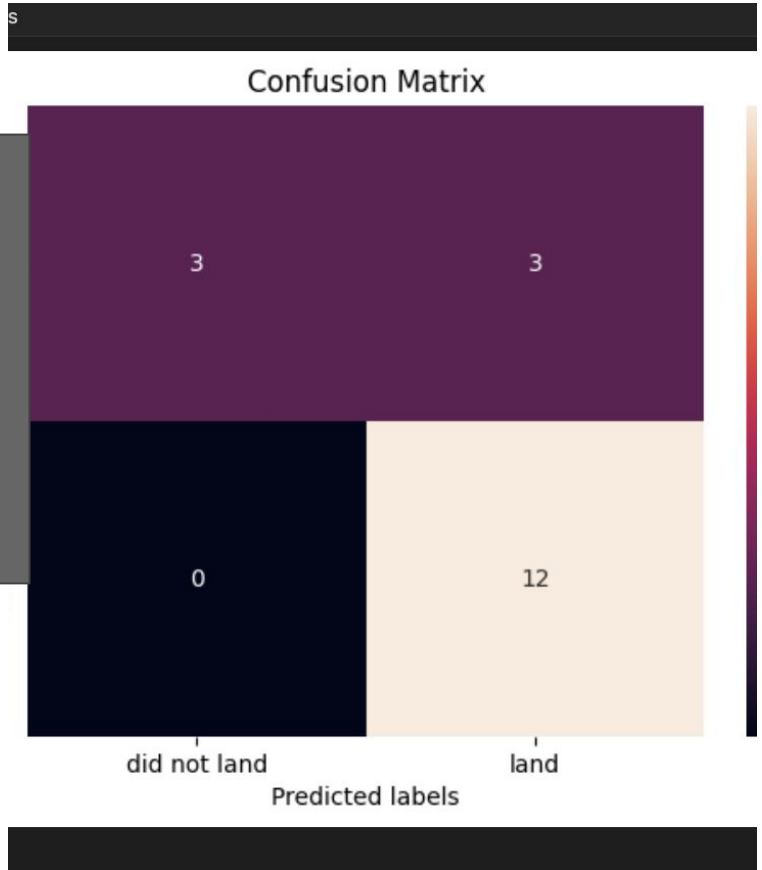
```
Accuracy on the test data for Decision Tree: 0.8333333333333334
```

With the best parameters:

`tree_cv.best_params_`

we set the model up and now get train it with the train set. Then we can predict using the test set to get the accuracy of the model:

Accuracy: 83%



K Nearest Neighbors KNN

Using GridSearchCV

TASK 10

Create a k nearest neighbors object then create a `GridSearchCV` object `knn_cv` with cv = 10. Fit the object t

```
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()
```

[168] ✓ 0.0s

```
# Create a GridSearchCV object
KNN_cv = GridSearchCV(estimator=KNN, param_grid=parameters, cv=10)

# Fit the GridSearchCV object to find the best parameters
KNN_cv.fit(X_train, y_train)
```

[169] ✓ 1.4s

GridSearchCV
estimator: KNeighborsClassifier
KNeighborsClassifier

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

[170] ✓ 0.0s

```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 3, 'p': 1}
```

After run GridSearchCV we get the best parameters for SVM in the variable `KNN_cv.best_params_`

K Nearest Neighbors KNN

Accuracy and Confusion matrix

TASK 11

Calculate the accuracy of knn_cv on the test data

```
from sklearn.metrics import accuracy_scor

# Assume you have a GridSearchCV object f
# Get the best parameters from the GridSe
best_params_knn = KNN_cv.best_params_

# Initialize a new KNN model with the bes
knn_best = KNeighborsClassifier(**best_pa

# Train the new KNN model on the full tra
knn_best.fit(X_train, y_train)

# Predict using the test dataset
y_pred_knn = knn_best.predict(X_test)

# Calculate the accuracy
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print("Accuracy on the test data for KNN:", accuracy_knn)
```

71] ✓ 0.0s

```
Accuracy on the test data for KNN: 0.6111111111111112
```

With the best parameters:

`KNN_cv.best_params_`

we set the model up and now get train it with the train set. Then we can predict using the test set to get the accuracy of the model:

Accuracy: 61%

Image source: *Image created by DALL-E, September 2024.*

Best Model predictive Classification

TASK 12

Find the method performs best:

```
# Store the accuracy scores in a dictionary
accuracy_results = {
    'Logistic Regression': accuracy_lr,
    'SVM': accuracy_svm,
    'Decission Tree': accuracy_tree,
    'KNN': accuracy_knn
}


# Sort the dictionary by accuracy
sorted_accuracy = sorted(accuracy_results.items(), key=lambda x: x[1], reverse=True)

# Get the top two models
top_two_models = sorted_accuracy[:2]

print("Top Two Models:")
for model, accuracy in top_two_models:
    print(f"Model: {model}, Accuracy: {accuracy}")
```

[175] ✓ 0.0s

```
Top Two Models:
Model: Logistic Regression, Accuracy: 0.8333333333333334
Model: Decission Tree, Accuracy: 0.8333333333333334
```

Logistic Regression, and Decision Tree, perform well predicting the successful landing

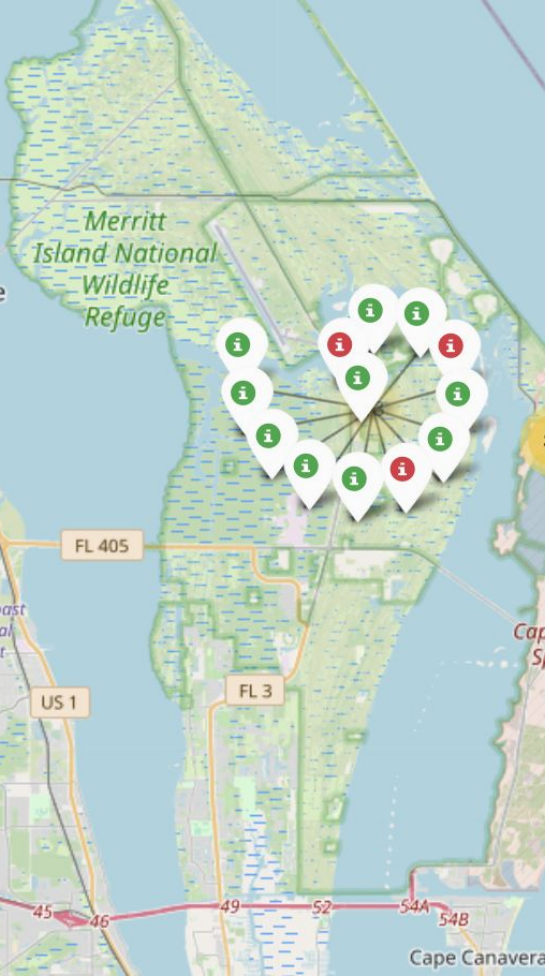


Image source: local python file running in Visual Studio Code (1.92.2) in Macbook Pro M1

Interactive maps with Folium

To analyze and visualize SpaceX launch sites geographically, we used Folium, a powerful Python library for creating interactive maps directly within the notebook.

Approach

- Mapped all SpaceX launch sites, including CCAFS SLC 40, VAFB SLC 4E, and KSC LC 39A.
- Added markers and tooltips to display site names, coordinates, and launch details.
- Analyzed geographical factors and their potential impact on launch success.

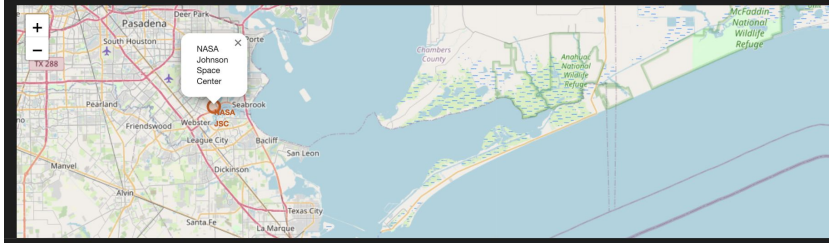
Benefits

- Provides intuitive, interactive insights into spatial data.
- Highlights patterns and trends related to launch site selection and outcomes.

Interactive maps with Folium

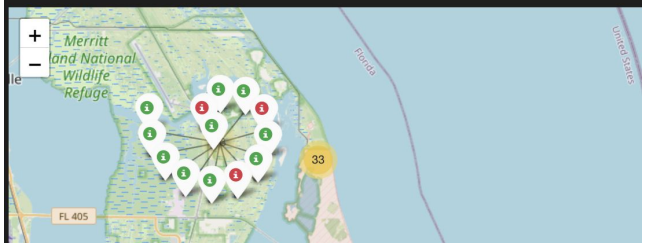
Creating maps in this project is crucial for visualizing the spatial relationships between launch sites and their outcomes, revealing patterns and trends influenced by location. For instance, **mapping shows how proximity to water and isolation of launch sites are strategic factors in minimizing risk to people in case of accidents.**

```
# Create a blue circle at NASA Johnson Space Center's coordinate with a popup label showing its name
circle = folium.Circle(nasa_coordinate, radius=1000, color='#d35400', fill=True).add_child(folium.Popup('NASA Johnson Space Center'))
# Create a blue circle at NASA Johnson Space Center's coordinate with a icon showing its name
marker = folium.map.Marker(
    nasa_coordinate,
    # Create an icon as a text label
    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)
```



Displaying locations and adding text label on a specific coordinates

```
# Initialize the map
site_map = folium.Map(location=[28.5632, -80.5768], zoom_start=5)

# Create a MarkerCluster
marker_cluster = MarkerCluster().add_to(site_map)

# Iterate over the DataFrame and create markers
for index, record in spacex_df.iterrows():
    # Create an icon with the appropriate color
    icon = folium.Icon(color='white', icon_color=record['marker_color'])

    # Create a marker with the launch site and the icon
    marker = folium.Marker(
        location=[record['Lat'], record['Long']],
        popup=record['Launch Site'],
        icon=icon
    )

    # Add the marker to the marker cluster
    marker_cluster.add_child(marker)

# Display the map
site_map
```



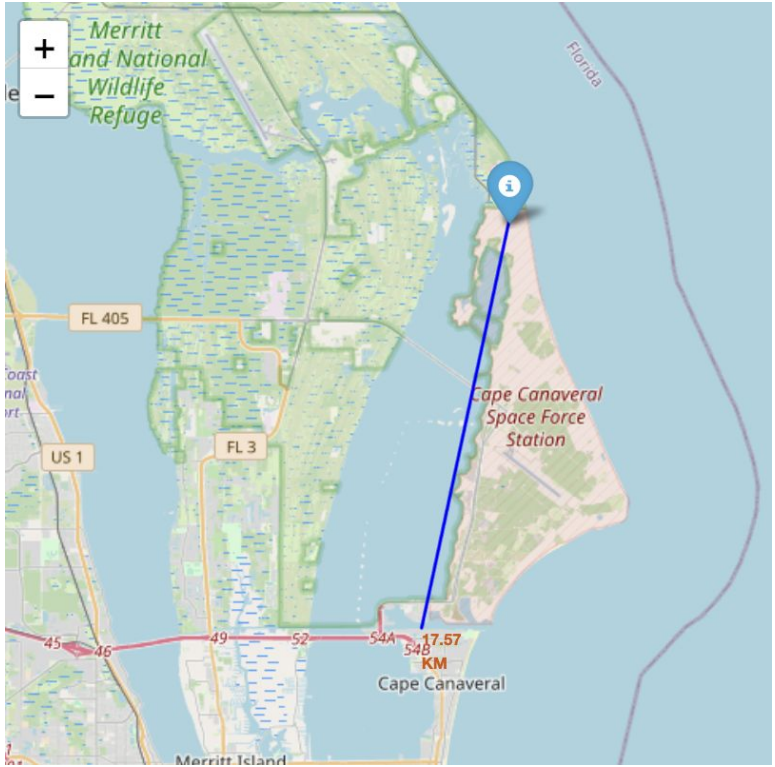
Displaying market clusters for each launch sites and showing success and fail landing rockets site.

Interactive maps with Folium

Showing the distance of **17.57km** from a launch site **(CCAFS SLC-40)** to the closer city **(Cape Canaveral).**

We are able to answer:

- Are launch sites in close proximity to railways?
- Are launch sites in close proximity to highways?
- Are launch sites in close proximity to coastline?
- Do launch sites keep certain distance away from cities?



Plotly Dash dashboard

This Plotly Dash dashboard provides an interactive platform to visualize key metrics and trends related to SpaceX Falcon 9 launches. It features various elements, such as the success rate by launch site, which highlights site-specific performance, and an analysis of how payload mass affects the likelihood of a successful landing. The dashboard offers a dynamic, user-friendly way to explore data, identify patterns, and gain valuable insights, ultimately supporting data-driven decision-making for future launches.

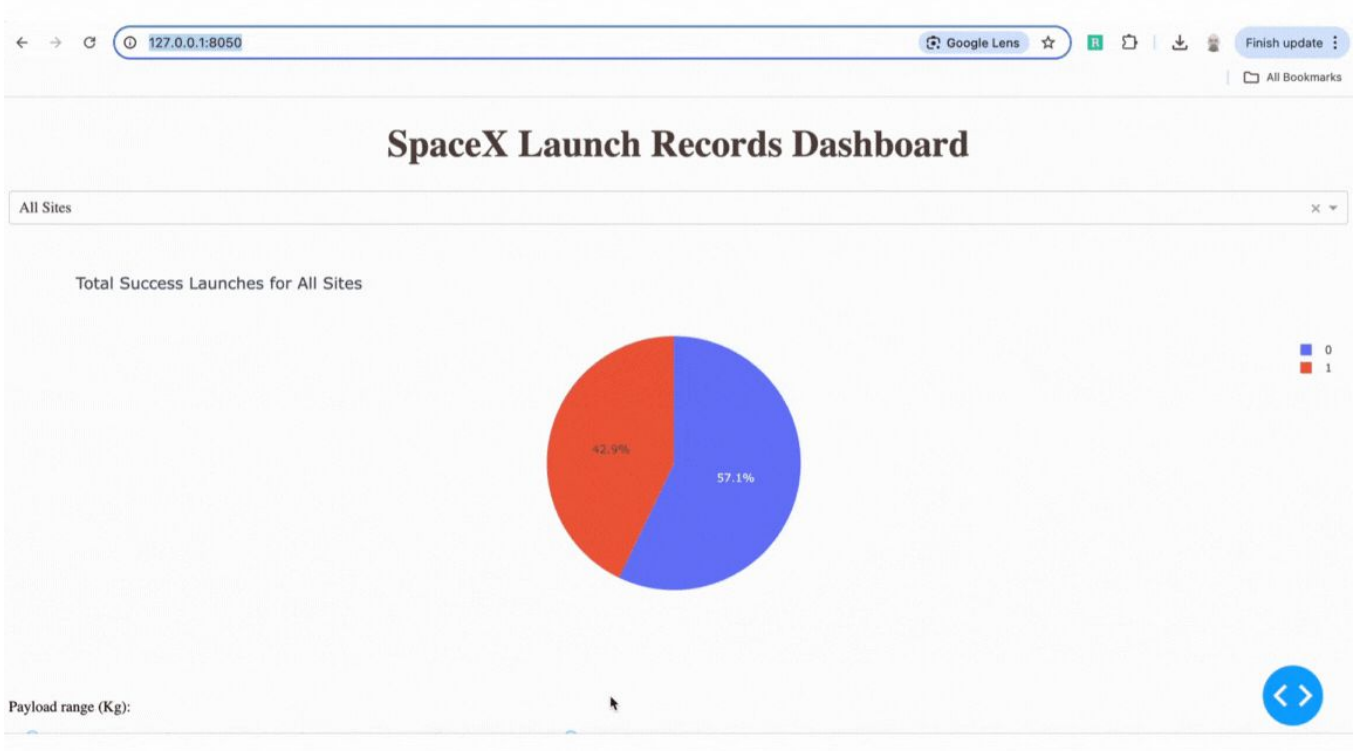


Image source: local python file running in Visual Studio Code (1.92.2) in Macbook Pro M1. Gif created in Canva

Conclusion

Model Performance: The Logistic Regression and Decision Tree models demonstrated the highest accuracy and balanced performance across key metrics, making it the best choice for predicting Falcon 9 first-stage landing success.

Key Insights:

The Support Vector Machine (SVM) and K-Nearest Neighbors (KNN) models had lower accuracy and struggled with prediction errors, particularly false positives and false negatives.

Enhanced Data Visualization: Interactive maps and dashboards effectively visualize complex data, making it easier to identify patterns and trends in SpaceX launch outcomes.

Implications for Space Y: Accurate predictions of landing success can optimize mission planning, reduce costs, and enhance decision-making processes for future launches.

Future Recommendations:

- Further model tuning and exploration of additional features could improve prediction accuracy.
- Consider using ensemble methods or advanced techniques such as deep learning to enhance model performance.