# Data science capstone project

RAUL JAIR OSORNO ORTIZ

HTTPS://GITHUB.COM/RAULRULAS

01/11/2024

# Executive Summary

Data was collected from SpaceX's public API and Wikipedia to classify successful landings. Then, they were analyzed with SQL, visualizations, interactive maps and dashboards. The categorical variables were coded and standardized to build machine learning models, optimized using GridSearchCV. Four models were tested (Logistic Regression, SVM, Decision Tree and KNN), all with an accuracy close to 83.33%, although with a slight overestimation of successful landings. To improve accuracy, it is recommended to collect more data and explore other models.

# Introduction

Today, several companies are making the space more accessible. SpaceX leads the sector with economical launches thanks to the reuse of the first stage of its Falcon 9 rockets, generating great savings.

The goal of this project is to help SpaceY, a startup that wants to compete with SpaceX, predict whether the first stage of a launch will land successfully, which could reduce the costs of each mission. In this project, we will take on the role of a data scientist at SpaceY and develop a machine learning model to predict rocket first stage reusability, using public information and data analysis techniques to support decision making and optimize the company's resources.

SPACE Y

Allon Mask

# Methodology

Data Collection:

- Gathered data from the SpaceX public API and SpaceX's Wikipedia page.

Data Wrangling:

- Processed and cleaned data, classifying landings as either successful or unsuccessful.

Exploratory Data Analysis (EDA):

- Conducted EDA using SQL queries and data visualizations to identify key patterns and insights.

Interactive Visual Analytics:

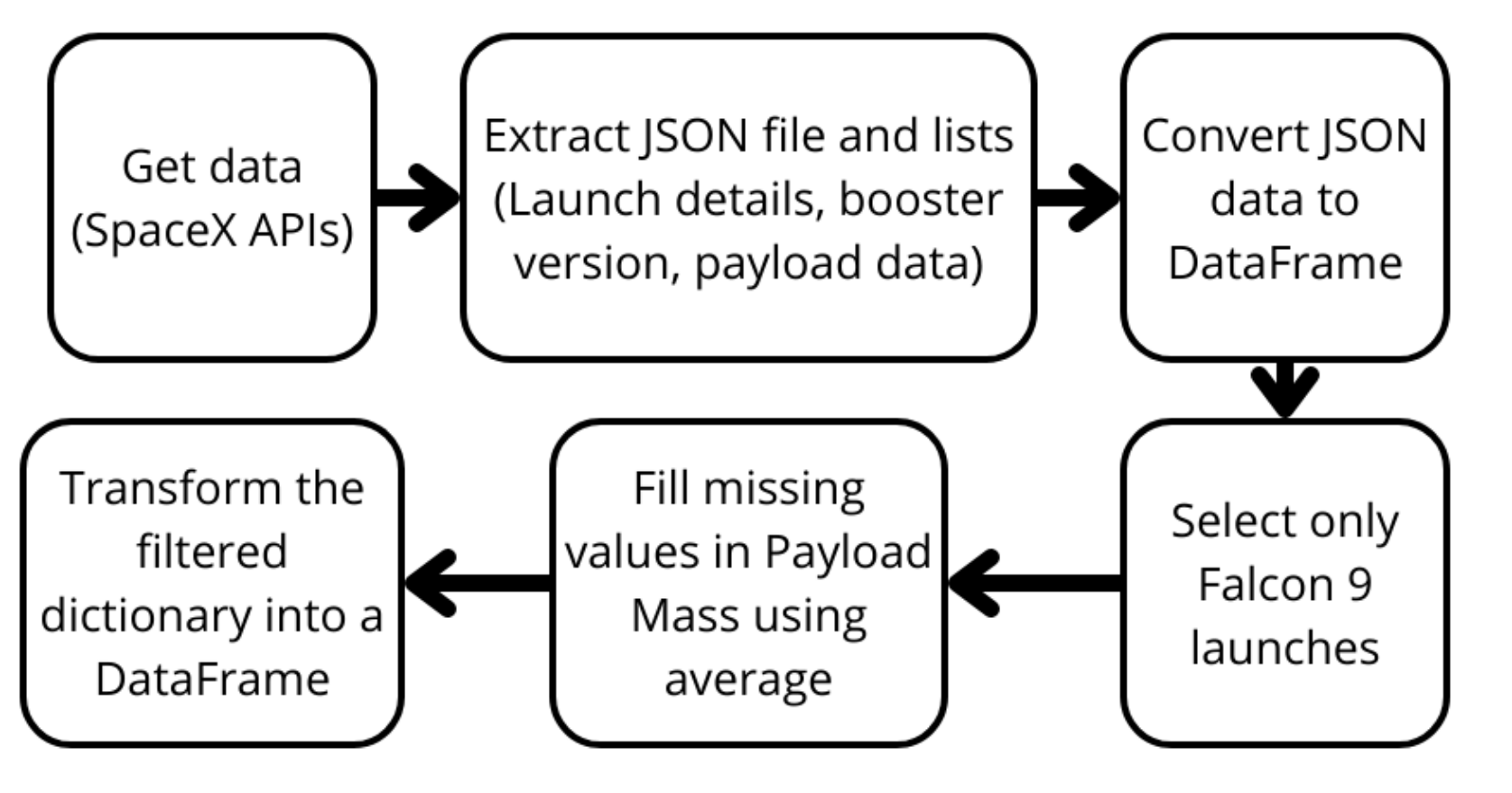
- Used Folium for map-based visualizations and Plotly Dash for interactive dashboards.

Predictive Analysis:

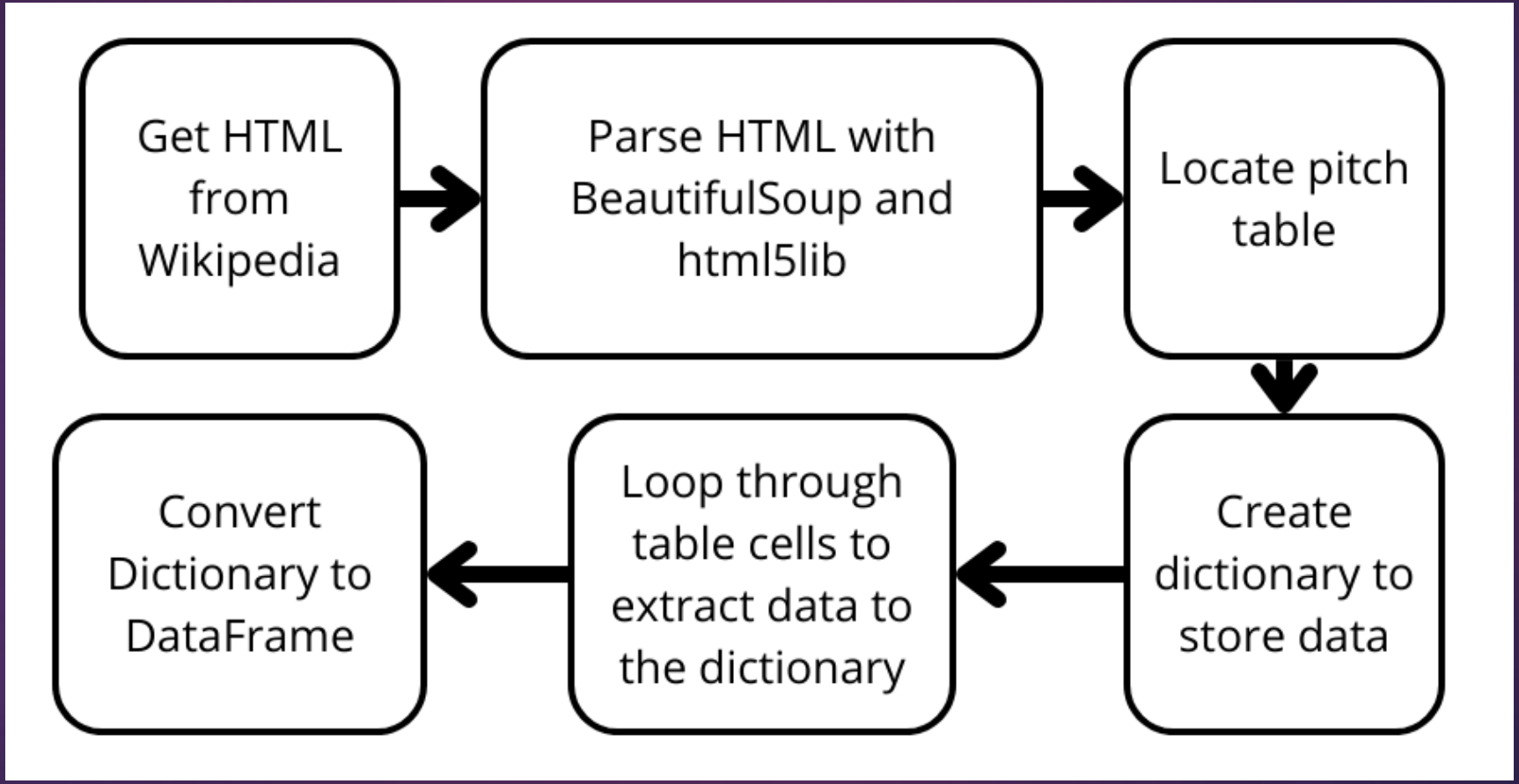
- Built classification models to predict successful landings.Optimized models using GridSearchCV to improve accuracy.

# Outline

# Data collection



# Data wrangling



# EDA with SQL

- EXPLORATORY DATA ANALYSIS WITH SQL

# All launch site names

```python
# Conectar a la base de datos
con = sqlite3.connect("my_data1.db")

# Ejecutar la consulta usando pandas
query = "SELECT DISTINCT LAUNCH_SITE FROM SPACEXTABLE"
result_df = pd.read_sql_query(query, con)

# Ver el resultado
print(result_df)
```

```
    Launch_Site
0   CCAFS LC-40
1   VAFB SLC-4E
2    KSC LC-39A
3  CCAFS SLC-40
```

It appears there are duplicate entries in the database for launch site names, specifically with slight variations like "CCAFS SLC-40" and "CCAFSSLC-40." These likely refer to the same launch site but contain minor data entry inconsistencies. Originally, "CCAFS LC-40" was used as the site name, and upon review, it seems there are only three unique launch sites: "CCAFS SLC-40," "KSC LC-39A," and "VAFB SLC-4E."

# Launch site names beggining with “CCA”

```
# Conectar a la base de datos
con = sqlite3.connect("my_data1.db")

query = "SELECT * FROM SPACEXTABLE WHERE LAUNCH_SITE LIKE 'CCA%' LIMIT 5;"
result_df = pd.read_sql_query(query, con)
display(result_df)
```

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

-First five entries  in database with Launch Site name  beginning with CCA.

# Total payload mass for NASA

```
# Conectar a la base de datos
con = sqlite3.connect("my_data1.db")

query = "SELECT SUM(PAYLOAD_MASS__KG_) AS Total_Payload_Mass FROM SPACEXTABLE WHERE Customer = 'NASA (CRS)';"
result_df = pd.read_sql_query(query, con)
display(result_df)
```

| | Total_Payload_Mass |
|---|---|
| 0 | 45596 |

-This query sums the total payload mass in kg where NASA was the customer.

# Average Payload Mass by F9 v1.1

```
query = "SELECT AVG(PAYLOAD_MASS__KG_) AS Average_Payload_Mass FROM SPACEXTABLE WHERE Booster_Version = 'F9 v1.1';"
result_df = pd.read_sql_query(query, con)
display(result_df)
```

| | Average_Payload_Mass |
|---|---|
| 0 | 2928.4 |

This query determines the average payload mass for launches that utilized the booster version F9 v1.1.

# First Successful Ground Pad Landing Date

```
query = "SELECT MIN(Date) AS First_Successful_Landing_Date FROM SPACEXTABLE WHERE Landing_Outcome = 'Success (ground pad)';"
result_df = pd.read_sql_query(query, con)
display(result_df)
```

| | First_Successful_Landing_Date |
|---|---|
| 0 | 2015-12-22 |

The SQL query shown here searches for the date of the first successful landing on a ground platform from SpaceX missions.

# Successful boosters on drone ship with payload of 4000-6000 kg

```
query = "SELECT Booster_Version FROM SPACEXTABLE WHERE Landing_Outcome = 'Success (drone ship)' AND PAYLOAD_MASS__KG_ > 4000 AND PAYLOAD_MASS__KG_ < 6000;"
result_df = pd.read_sql_query(query, con)
display(result_df)
```

| | Booster_Version |
|---|---|
| 0 | F9 FT B1022 |
| 1 | F9 FT B1026 |
| 2 | F9 FT B1021.2 |
| 3 | F9 FT B1031.2 |

This output shows a list of booster versions that successfully landed on a drone ship and carried a payload weighing between 4000 and 6000 kg.

# Total number of each mission outcome

```
query = "SELECT CASE WHEN Landing_Outcome LIKE 'Success%' THEN 'Success' WHEN Landing_Outcome LIKE 'Failure%' THEN 'Failure' ELSE 'Other' END AS Outcome_Category, COUNT(*) AS Outcome_Count FROM SPACEXTABLE GROUP
result_df = pd.read_sql_query(query, con)
display(result_df)
```

| | Outcome_Category | Outcome_Count |
|---|---|---|
| 0 | Failure | 10 |
| 1 | Other | 30 |
| 2 | Success | 61 |

This allows you to quickly view how many missions were successful, failed, or had other landing outcomes, providing an overview of the various outcomes and their frequency.

# Maximum payload mass

```
query = "SELECT Booster_Version FROM SPACEXTABLE WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTABLE);"
result_df = pd.read_sql_query(query, con)
display(result_df)
```

| | Booster_Version |
|---|---|
| 0 | F9 B5 B1048.4 |
| 1 | F9 B5 B1049.4 |
| 2 | F9 B5 B1051.3 |
| 3 | F9 B5 B1056.4 |
| 4 | F9 B5 B1048.5 |
| 5 | F9 B5 B1051.4 |
| 6 | F9 B5 B1049.5 |
| 7 | F9 B5 B1060.2 |
| 8 | F9 B5 B1058.3 |
| 9 | F9 B5 B1051.6 |
| 10 | F9 B5 B1060.3 |
| 11 | F9 B5 B1049.7 |

The result shows all rocket versions that reached the maximum payload recorded in the database, indicating that multiple launches reached the same maximum payload weight carried.

# “Details of failed drone ship launches for each month of 2015"

```
query = "SELECT substr(Date, 6, 2) AS Month, Landing_Outcome, Booster_Version, Launch_Site FROM SPACEXTABLE WHERE Landing_Outcome LIKE 'Failure (drone ship)'AND substr(Date, 1, 4) = '2015';"
result_df = pd.read_sql_query(query, con)
display(result_df)
```

| | Month | Landing_Outcome | Booster_Version | Launch_Site |
|---|---|---|---|---|
| 0 | 01 | Failure (drone ship) | F9 v1.1 B1012 | CCAFS LC-40 |
| 1 | 04 | Failure (drone ship) | F9 v1.1 B1015 | CCAFS LC-40 |

These data allow us to observe the monthly distribution of failed drone launches in 2015.

# "Count of Successful and Failed Landings on Drone and Fixed Platform between 2010 and 2017"

```
query = "SELECT Landing_Outcome, COUNT(*) AS Outcome_Count FROM SPACEXTABLE WHERE (Landing_Outcome = 'Failure (drone ship)' OR Landing_Outcome = 'Success (ground pad)') AND Date BETWEEN '2010-06-04' AND '2017-03-
result_df = pd.read_sql_query(query, con)
display(result_df)
```

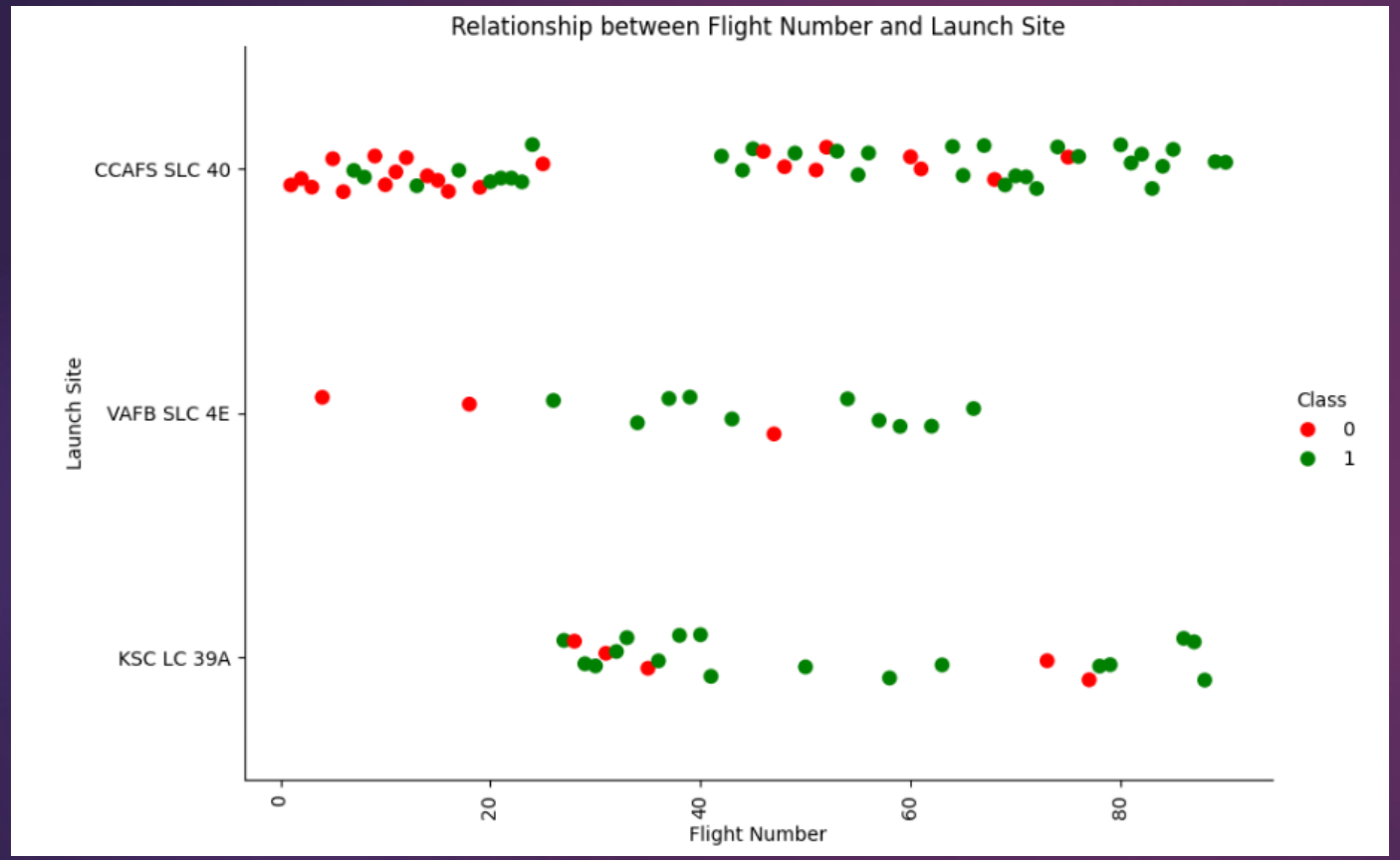
| | Landing_Outcome | Outcome_Count |
|---|---|---|
| 0 | Failure (drone ship) | 5 |
| 1 | Success (ground pad) | 3 |

The time filter restricts the data between June 4, 2010 and March 20, 2017. The result is sorted based on landing type, providing a quick view of performance in that period.

# EDA with Visualization

- EXPLORATORY DATA ANALYSIS WITH SEABORN PLOTS

# Flight Number vs LaunchSite



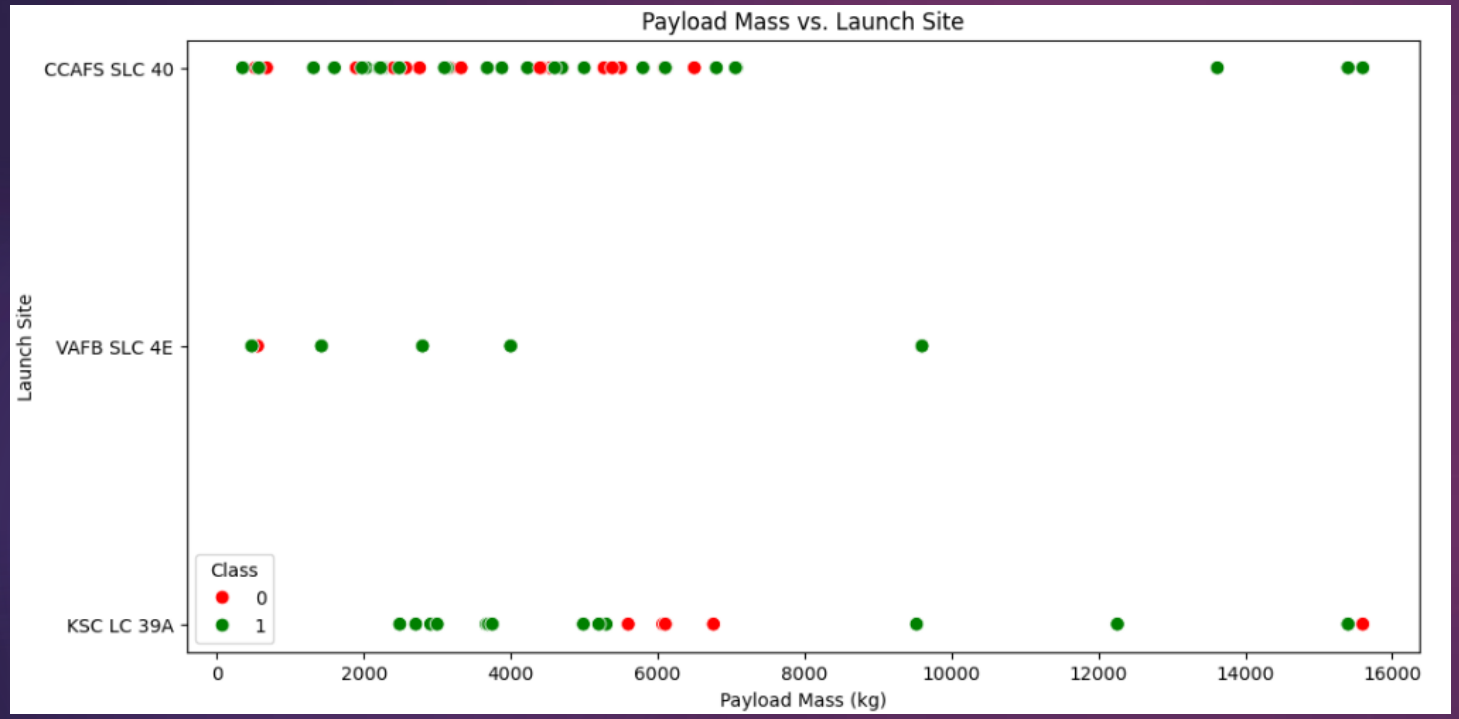

Green dots indicates successful launch;
Red dots indicates unsuccessful launch.

-Most launches occur at CCAFS site SLC 40, followed by KSC LC 39A and VAFB SLC 4E.

-At the CCAFS SLC 40 site, there are a greater number of green dots (success), indicating that launches from this site have a relatively high success rate.

-It appears that as the Flight Number increases, there is a trend towards a greater number of successes (green dots), especially in CCAFS SLC 40.

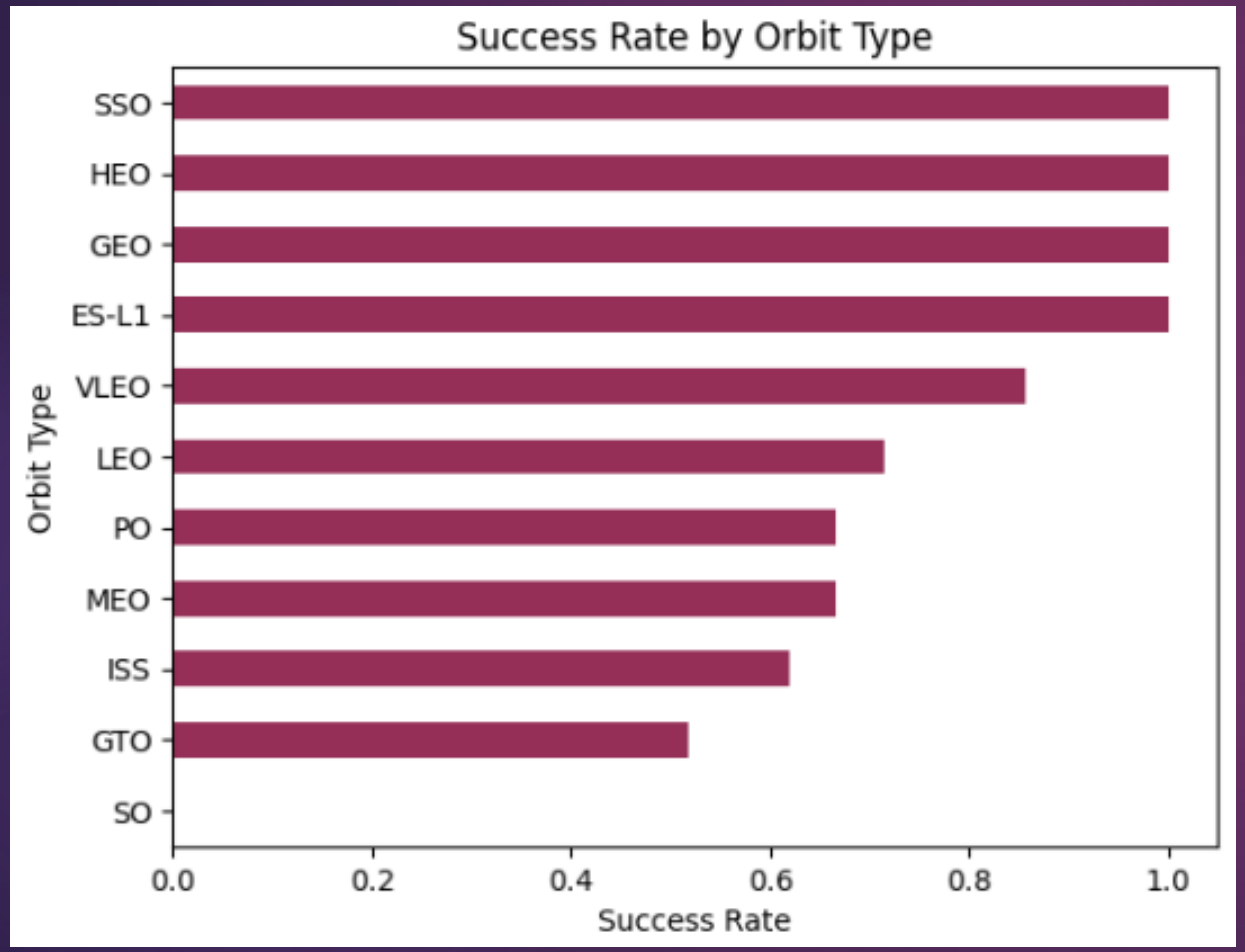
# Payload vs LaunchSite




Green dots indicates successful launch;
Red dots indicates unsuccessful launch.

-At the VAFB SLC 4E site, there are no launches exceeding a payload mass of 10,000 kg. All releases on this site have a lower payload compared to the other sites.
-This may indicate that VAFB SLC 4E is used for lighter missions, which may be related to infrastructure limitations or a preference to use this site for lighter missions.
-KSC LC 39A is where some of the heaviest payloads are seen, reaching up to 16,000 kg. This site appears to be prepared to handle significant loads, possibly due to advanced infrastructure or due to its strategic location.

# Success Rate vs Orbit Type

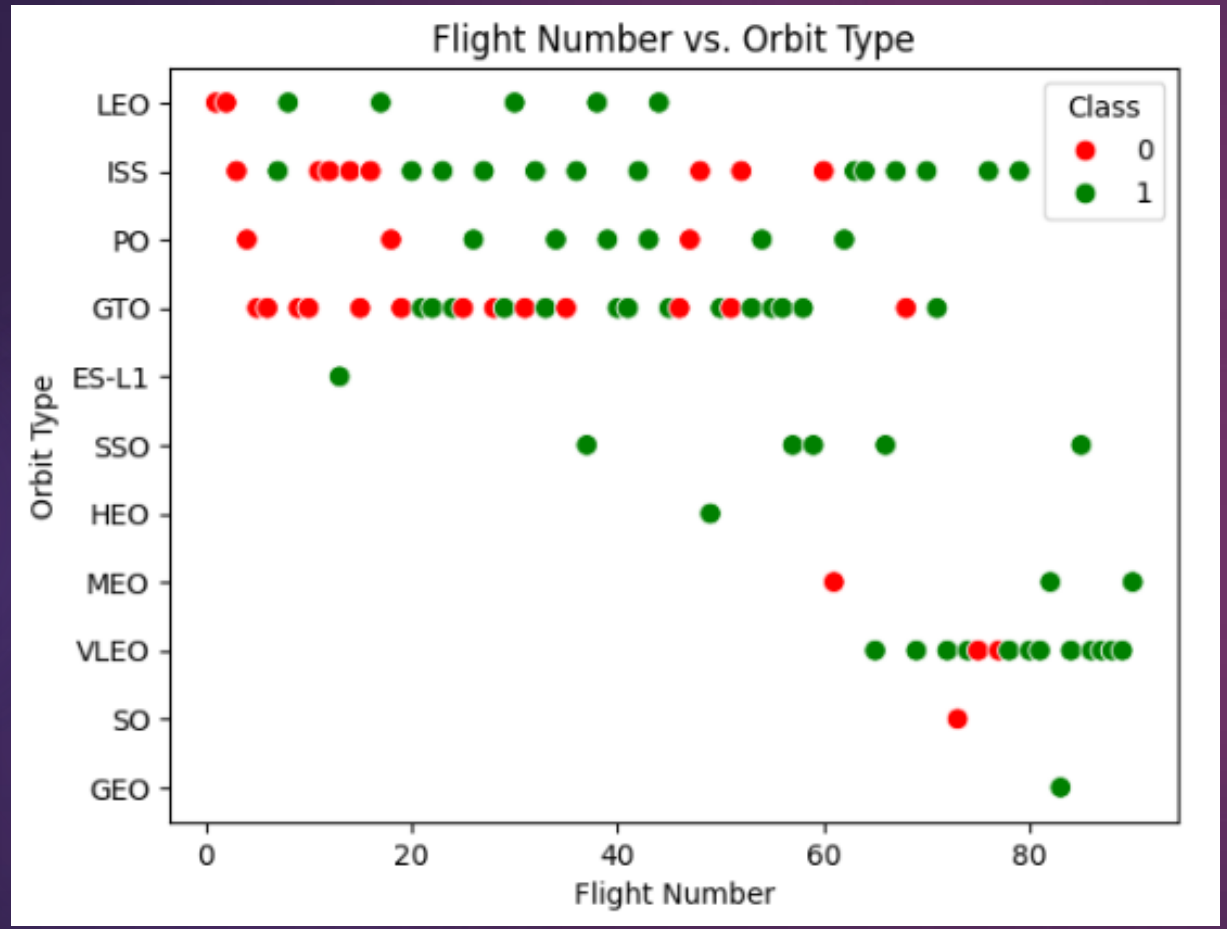



Success Rate Scale with
0 as 0%
0.6 as 60%
1 as 100%

-Orbits with 100% success rates, such as SSO, HEO, GEO and ES-L1, show that missions to these destinations are reliable and stable.

-The low success rate in GTO and SO suggests that these missions may have more stringent technical requirements or operational conditions that make them more susceptible to failure.

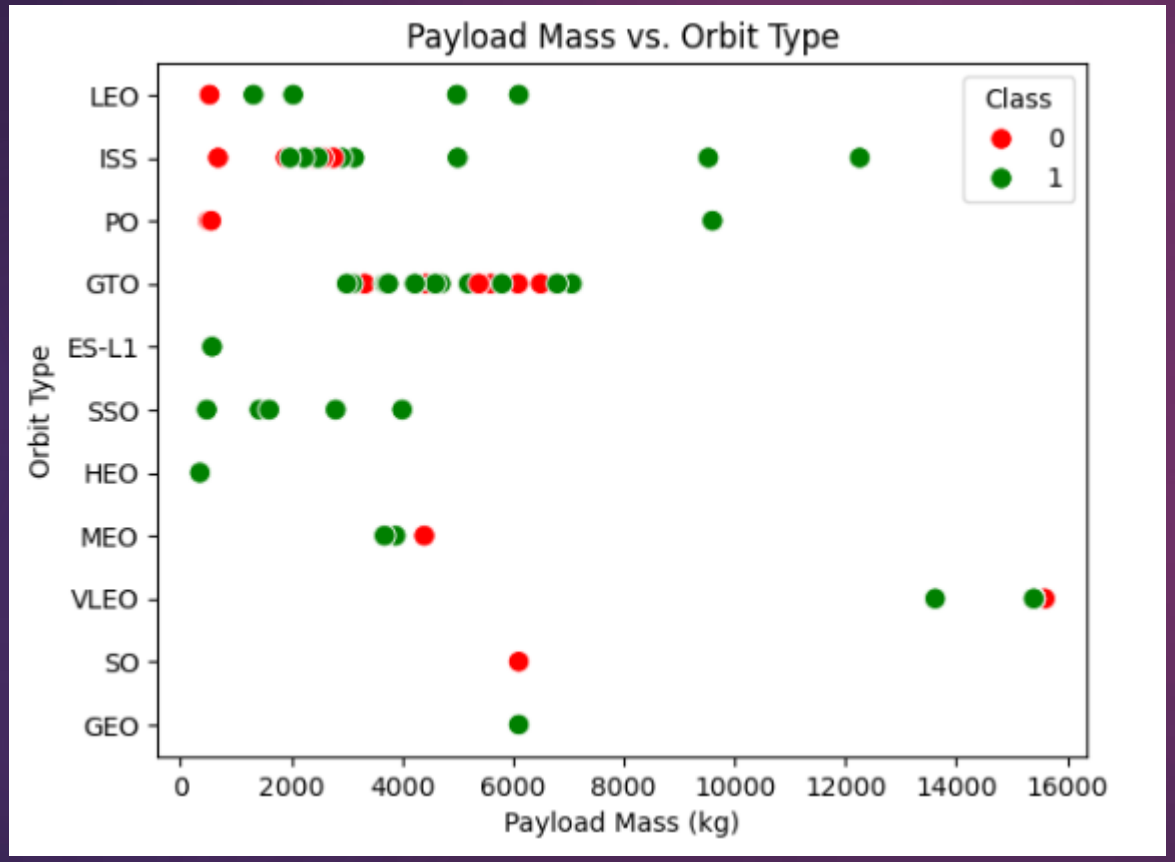
# Flight Number vs Orbit Type




Green dots indicates successful launch;
Red dots indicates unsuccessful launch.

-This graph reinforces the idea that experience and knowledge accumulation in repeated launches to certain orbits (especially LEO and ISS) have significantly improved mission success. Additionally, it indicates that some orbits, such as GTO, remain challenging even with progress in flight experience

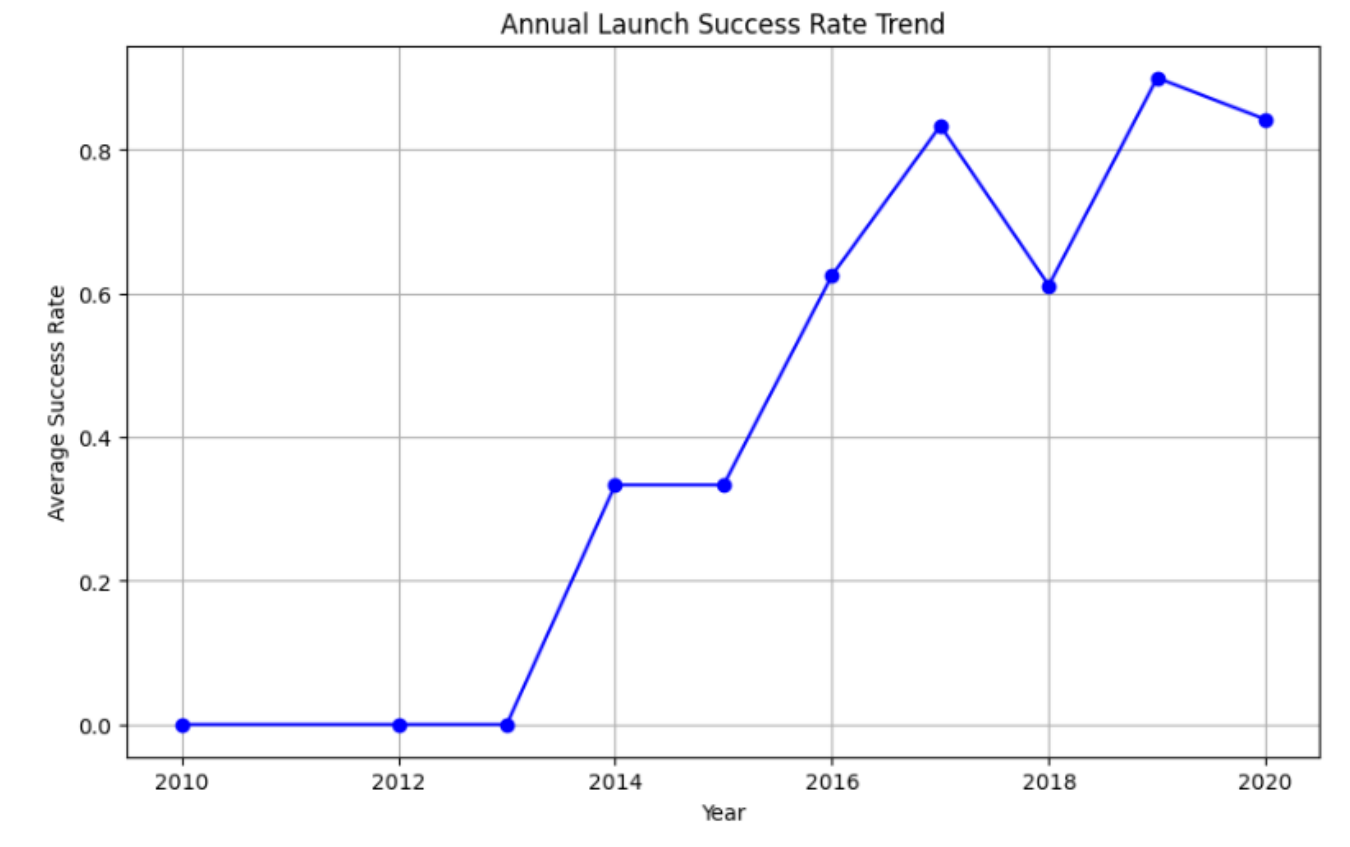
# Payload Mass vs Orbit Type




Green dots indicates successful launch;
Red dots indicates unsuccessful launch.

-This graph suggests that missions with light to medium payloads are frequent and usually successful in most orbits. The ability to successfully handle heavier payloads is notable in GTO and LEO, while more specific orbits (such as SSO and HEO) feature less variety in payload weight but a high success rate.

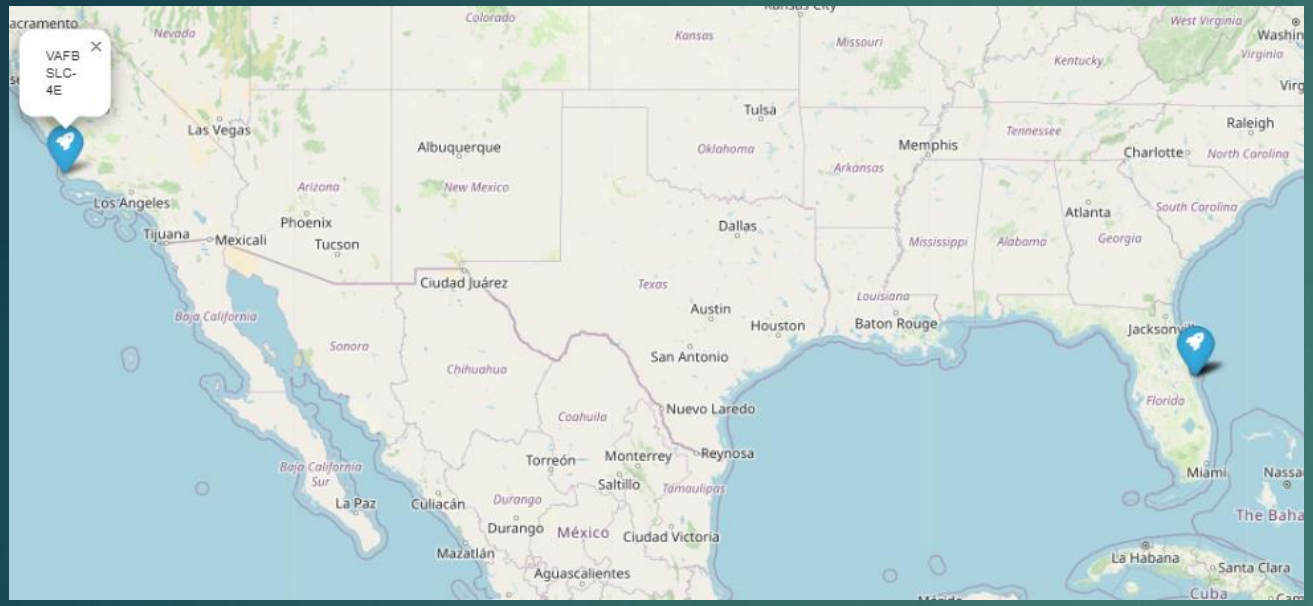
# Launch success yearly trend

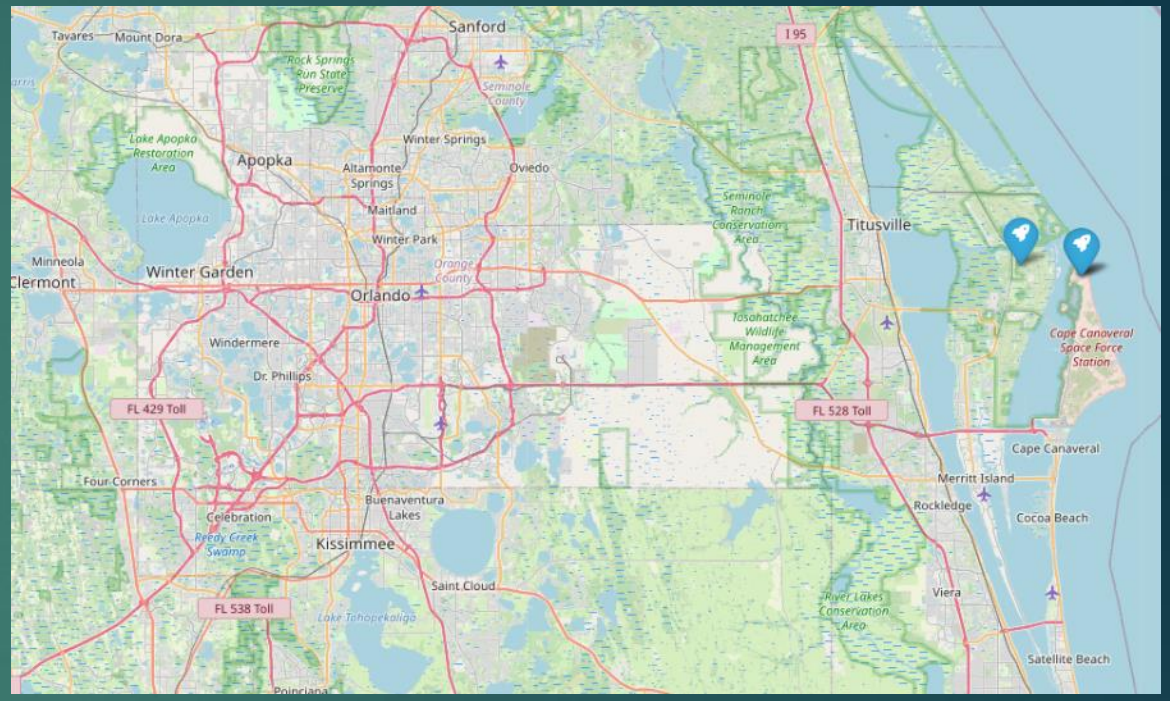



-The graph suggests a clear improvement in launch missions over time. The general trend is positive, with an increase in the success rate as time goes by, particularly notable after 2013 until reaching approximately 80% success.

# Interactive Visual Analytics with Folium

# Launch site locations



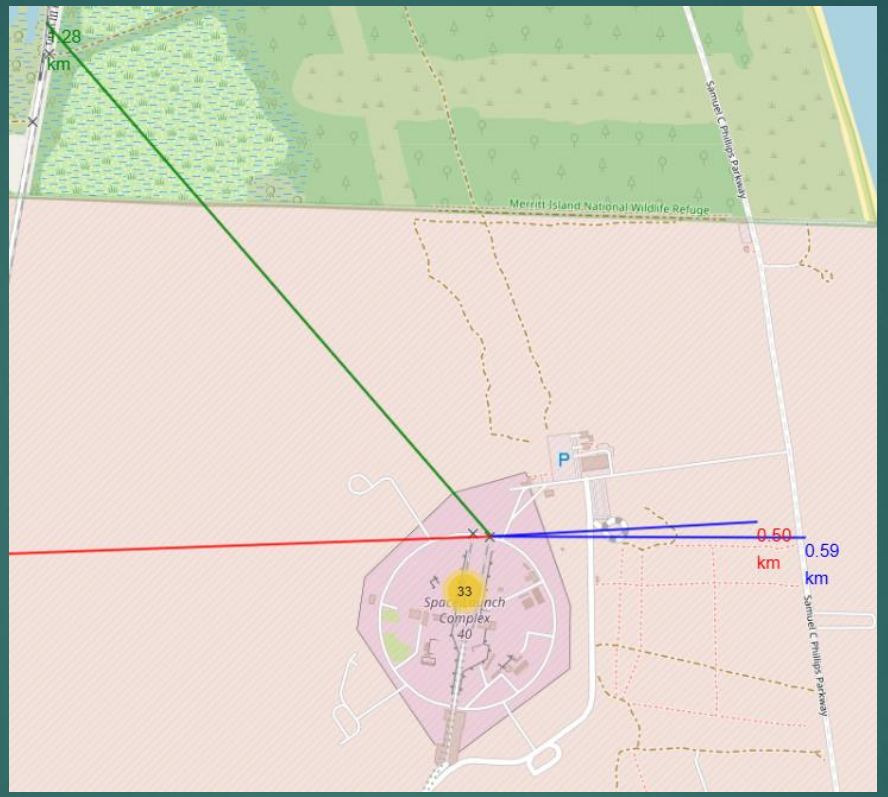

The launch sites are not near the equator, but are strategically located near the coast to minimize the risks associated with rocket launches.

The locations of the launch sites can be seen on the map. The right map is a zoom from the left map.

# Color launch markers


This map shows the launch sites with markers grouped and colored according to the launch outcome: markers in green represent successful launches, while red markers indicate failed launches. The markers are organized in a circular structure within the group, allowing for compact display when multiple pitches share the same coordinates.

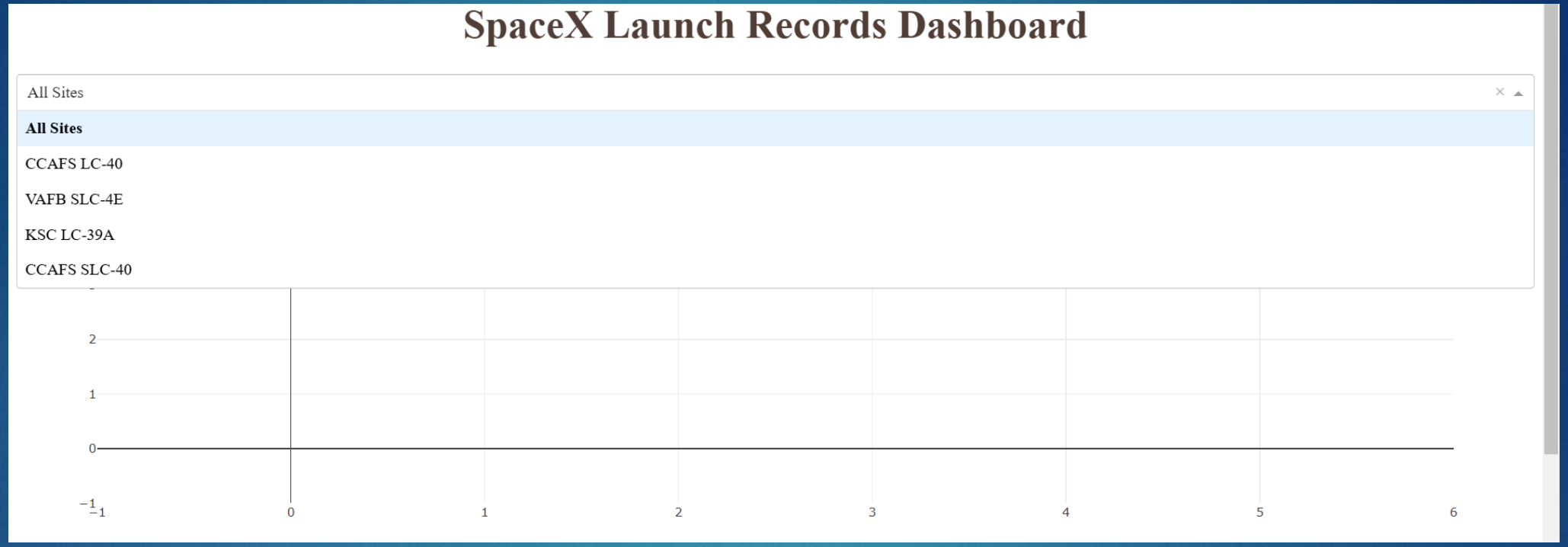
# Key Location Proximites




This map allows you to view distances from the launch site to several nearby points of interest, which is useful for understanding proximity and access from different types of infrastructure near the site.

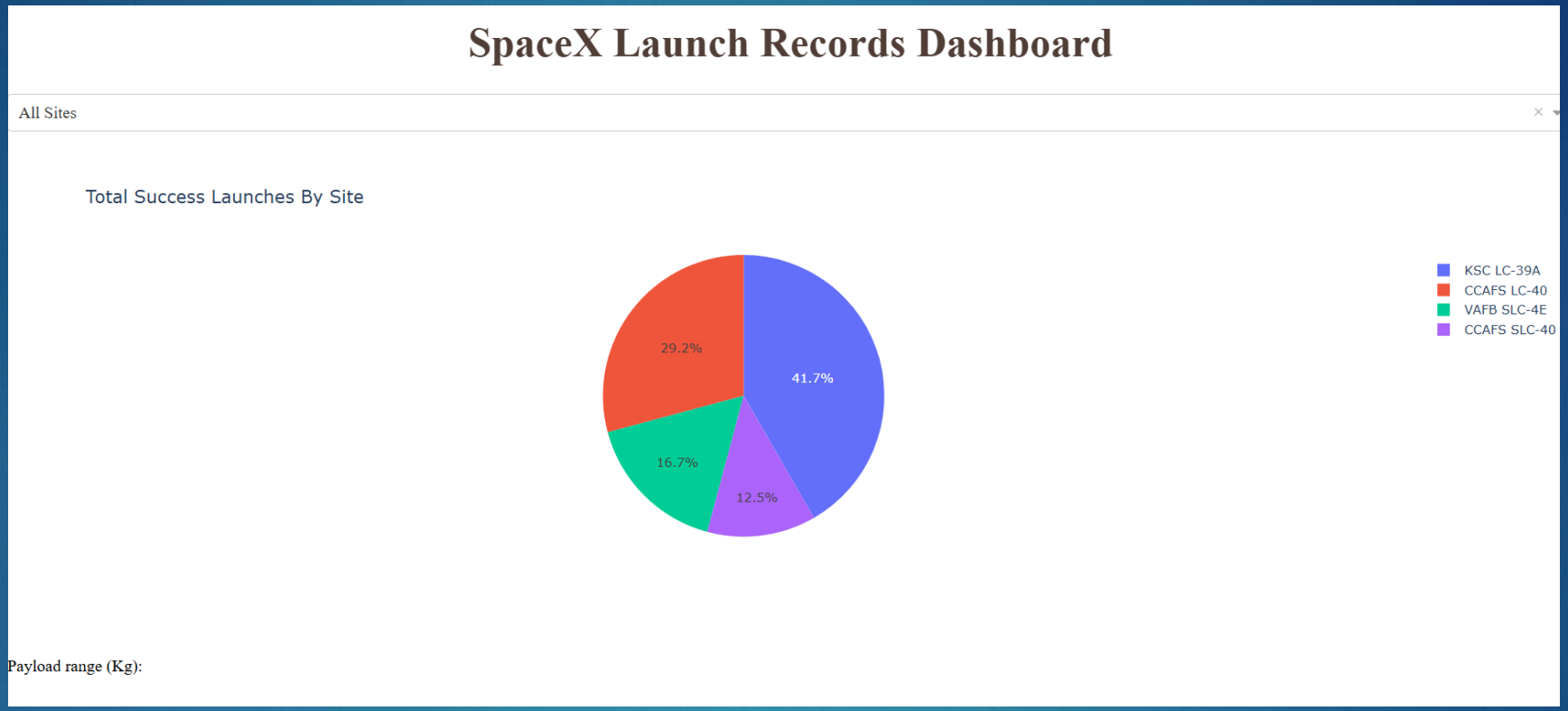
# Dashboard with Plotly Dash

# Launch Site Drop-down Input Component



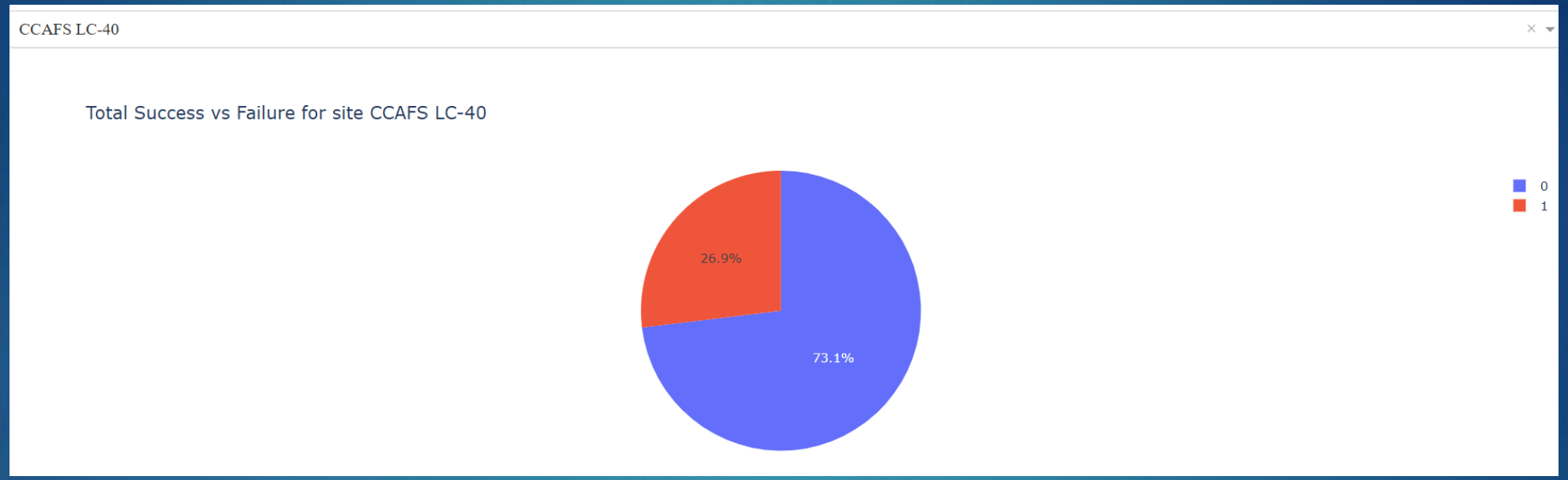

This result shows an interactive dashboard of SpaceX launch logs, in which a drop-down menu has been implemented at the top. This menu allows you to select between different SpaceX launch sites, including an option to view "All Sites." Selecting from the menu will update the graphics below based on the chosen launch site.

# Pie chart of selected site




This visualization is a pie chart showing the percentage of successful launches for each SpaceX launch site in the data set. Selecting the "All Sites" value from the drop-down menu has enabled the display of total launch successes across all sites together.

# Pie chart of selected site




This result shows an interactive dashboard of SpaceX launch logs, in which a drop-down menu has been implemented at the top. In this case the CCAFS LC-40 site has been selected and the program shows the pie chart.

# Range slider and callback function


The Plotly dashboard includes a payload range selector, set from 0 to 10000 instead of the maximum value of 15600. The "class" variable identifies successful landings with 1 and unsuccessful landings with 0. In the scatter plot, the Colors represent the different versions of the booster, while the size of the dots reflects the number of launches. In the specific range of 0 to 6000, it is curious to note that there are two failed landings with payloads of zero kg.

# Predictive analysis classification

# First steps for Machine Learning

## 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`,make sure the output is a Pandas series (only one bracket df['name of column']).

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

## TASK 2

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

```
# students get this
from sklearn import preprocessing

# Creamos el transformador de escalado
transform = preprocessing.StandardScaler()

# Aplicamos la transformación y reasignamos a la variable X
X = transform.fit_transform(X)
```

We split the data into training and testing data using the function `train_test_split`. The training data is divided into validation data, a second set used for training data; then the models are trained and hyperparameters are selected using the function `GridSearchCV`.

## TASK 3

Use 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 following labels.

`X_train, X_test, Y_train, Y_test`

```
# División de los datos en conjuntos de entrenamiento y prueba
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
```

we can see we only have 18 test samples.

```
Y_test.shape
```

# Logistic regression

## 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']}
```

```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

# Definimos los parámetros para GridSearchCV
parameters = {
    "C": [0.01, 0.1, 1],
    "penalty": ["l2"],
    "solver": ["lbfgs"]
}

# Creamos el modelo de regresión logística
lr = LogisticRegression()

# Configuramos GridSearchCV con validación cruzada de 10 pliegues
logreg_cv = GridSearchCV(lr, parameters, cv=10)

# Ajustamos el modelo a los datos de entrenamiento
logreg_cv.fit(X_train, Y_train)

# Mostramos los mejores parámetros y el mejor puntaje
print("Best Parameters:", logreg_cv.best_params_)
print("Best Score:", logreg_cv.best_score_)
```

```
Best Parameters: {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
Best Score: 0.8464285714285713
```

# Accuracy and Confussion matrix

## TASK 5

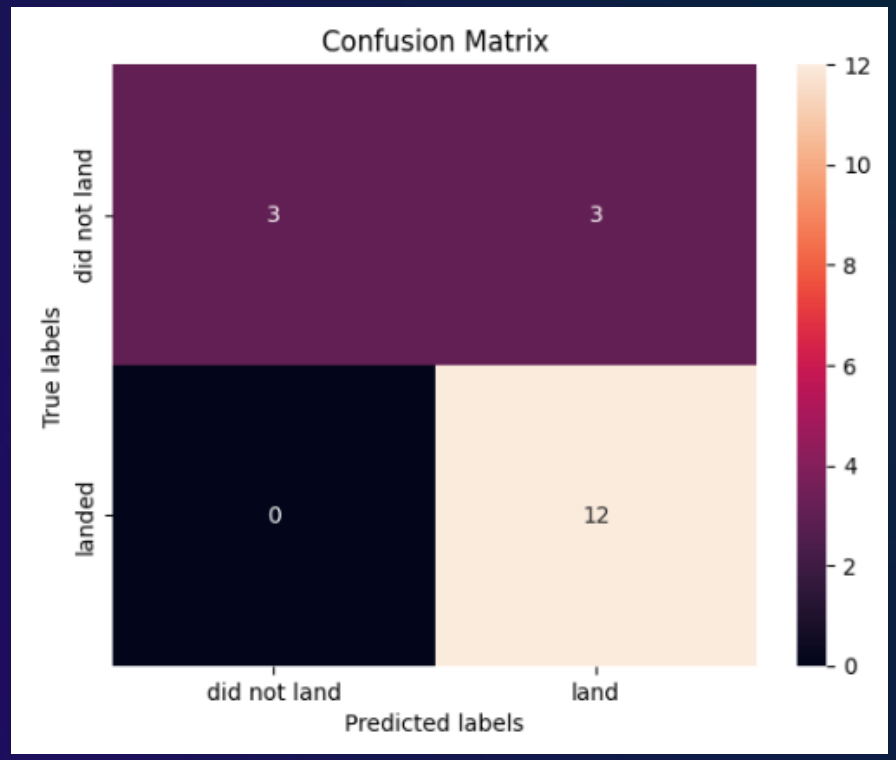
Calculate the accuracy on the test data using the method `score`:

```
# Calcula la precisión en el conjunto de prueba
accuracy = logreg_cv.score(X_test, Y_test)
print("Accuracy on test data:", accuracy)
```

```
Accuracy on test data: 0.8333333333333334
```

Lets look at the confusion matrix:

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




The model accurately predicts the cases where the event actually landed (TP = 12) and also has some successes in the cases where the event did not land (TN = 3). However, it has a moderate number of false positives (FP = 3), that is, incorrect landing predictions when the event did not actually land.
The absence of false negatives (FN = 0) is positive, indicating that the model did not fail to predict "did not land" when the event actually landed.

# Support Vector Machine

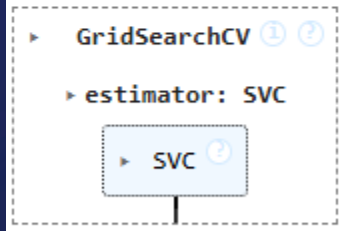
## TASK 6

Create a support vector machine object then create a `GridSearchCV` object `svm_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

```
# Configura GridSearchCV con validación cruzada de 10 pliegues
svm_cv = GridSearchCV(svm, parameters, cv=10)

# Ajusta el modelo a los datos de entrenamiento
svm_cv.fit(X_train, Y_train)
```




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

```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
accuracy : 0.8482142857142856
```

# Confusion matrix

## TASK 7

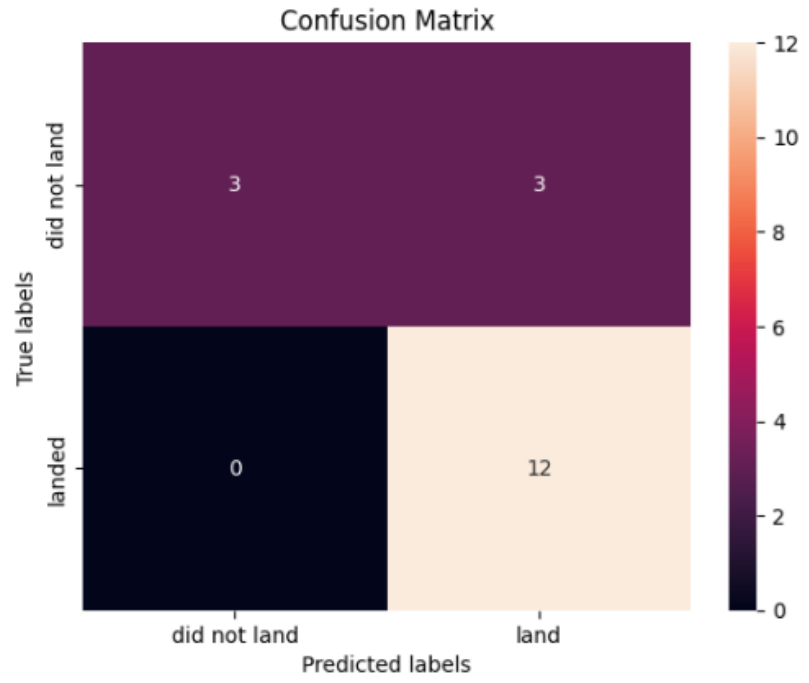
Calculate the accuracy on the test data using the method `score`:

```
# Calcula la precisión del modelo SVM en el conjunto de prueba
svm_accuracy = svm_cv.score(X_test, Y_test)
print("Accuracy of SVM on test data:", svm_accuracy)
```

```
Accuracy of SVM on test data: 0.8333333333333334
```

We can plot the confusion matrix

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




The SVM model has a high precision in landing cases and manages not to miss any cases where a landing actually occurred (FN = 0). However, there are some false positives (FP = 3), indicating that in some cases it predicts landings that do not occur.

# Tree classifier

## TASK 8

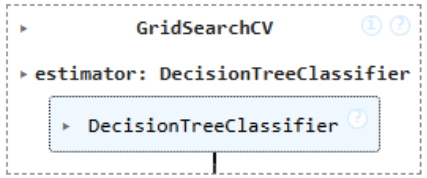
Create a decision tree classifier object then create a `GridSearchCV` object `tree_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

```
parameters = {
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random'],
    'max_depth': [2**n for n in range(1, 10)],
    'max_features': ['sqrt', 'log2'],  # Eliminamos 'auto'
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 5, 10]
}

tree = DecisionTreeClassifier()
```

```
# Configuramos GridSearchCV con validación cruzada de 10 pliegues
tree_cv = GridSearchCV(tree, parameters, cv=10)

# Ajustamos el modelo a los datos de entrenamiento
tree_cv.fit(X_train, Y_train)
```




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

```
tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', 'max_depth': 128, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 10, 'splitter': 'random'}
accuracy : 0.875
```

# Confussion matriz

## TASK 9

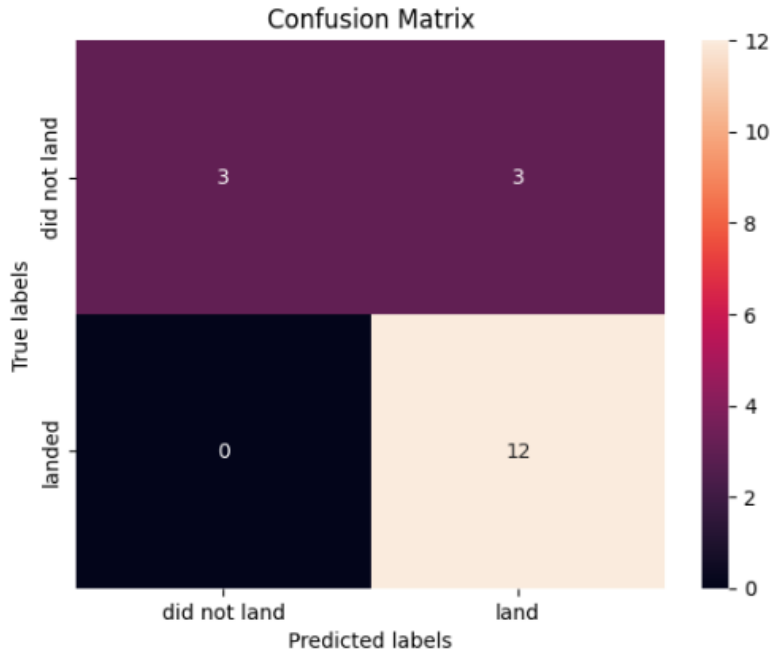
Calculate the accuracy of tree_cv on the test data using the method `score`:

```
# Calcula la precisión del modelo de árbol de decisión en el conjunto de prueba
tree_accuracy = tree_cv.score(X_test, Y_test)
print("Accuracy of Decision Tree on test data:", tree_accuracy)
```

```
Accuracy of Decision Tree on test data: 0.8333333333333334
```

We can plot the confusion matrix

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




Since both the SVM model and the decision tree have similar accuracy (83.33%) and have the same false positive errors, either model could be acceptable in terms of performance.

# KNN

## TASK 10

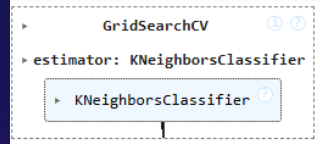
Create a k nearest neighbors object then create a `GridSearchCV` object `knn_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

```
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()
```

```
# Configuramos GridSearchCV con validación cruzada de 10 pliegues
knn_cv = GridSearchCV(KNN, parameters, cv=10)

# Ajustamos el modelo a los datos de entrenamiento
knn_cv.fit(X_train, Y_train)
```




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

```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}
accuracy : 0.8482142857142858
```

## TASK 11

Calculate the accuracy of knn_cv on the test data using the method `score`:

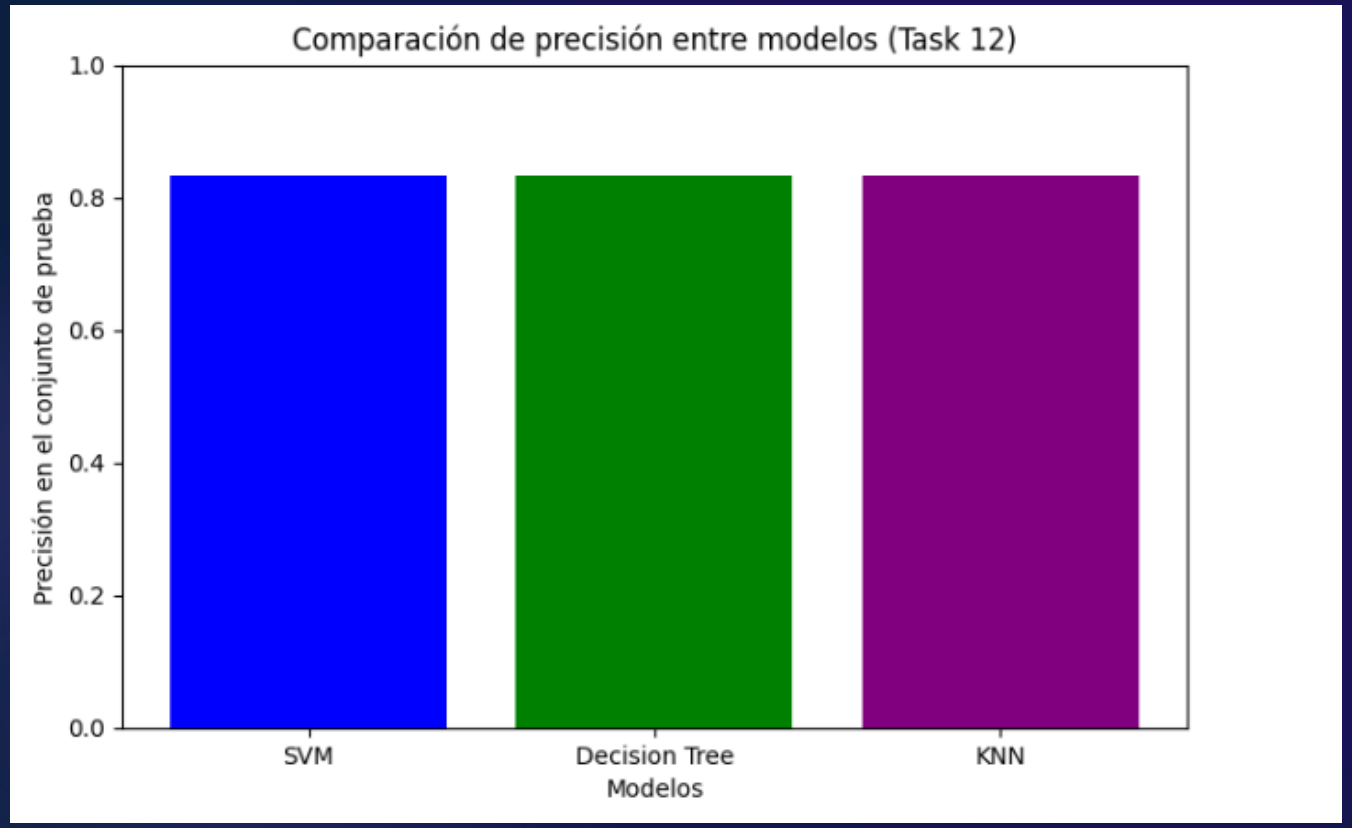
```
# Calcula la precisión del modelo KNN en el conjunto de prueba
knn_accuracy = knn_cv.score(X_test, Y_test)
print("Accuracy of KNN on test data:", knn_accuracy)
```

```
Accuracy of KNN on test data: 0.8333333333333334
```

We can plot the confusion matrix

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

# Best method perform




The three models (SVM, Decision Tree and KNN) have identical accuracy on the test set, 83.33%.
Since there is no difference in terms of accuracy between the models, the choice between them may depend on other factors, such as the speed of prediction, the ease of interpretation of the model, or the specific needs of the context in which they will be used.

# Conclusions

The developed model is a major step forward for SpaceY in competing with SpaceX in the reusable launch market. With 83% accuracy, it aids strategic decision-making for cost savings. Further data collection and model refinement can enhance accuracy, helping SpaceY optimize resources and competitiveness.

Although the current model provides good results, it is recommended to collect more data to further improve the accuracy of the model. With a larger and more diverse data set, it is likely that a machine learning model will be identified that better fits the specific conditions and variables that affect successful landings, optimizing prediction accuracy.

# Appendix

GitHub repository url:
https://github.com/Raulrulas/Applied-Data-Science-Capstone

Instructors:
Instructors: Rav Ahuja, Alex Aklson, Aije Egwaikhide, Svetlana Levitan, Romeo Kienzler, Polong Lin, Joseph Santarcangelo, Azim Hirjani, Hima Vasudevan, Saishruthi Swaminathan, Saeed Aghabozorgi, Yan Luo