IBM Data Science Capstone Project | Space X | Falcon 9

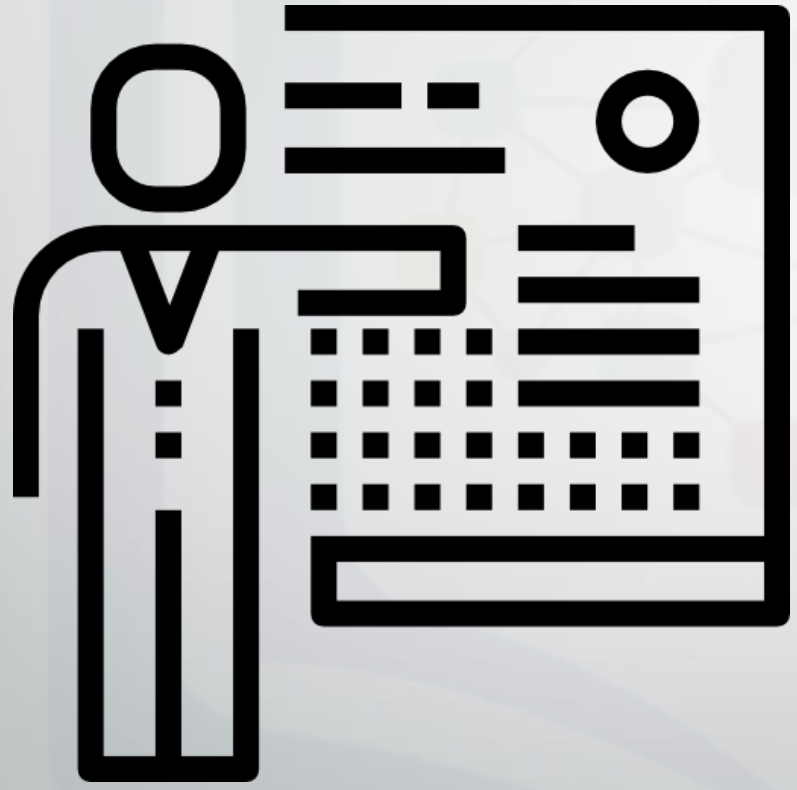
Keamogetsoe Mphuthi

09/01/2024

IBM Developer

SKILLS NETWORK

OUTLINE



- Executive Summary
- Introduction
- Methodology
- Results
  - Visualization – Charts
  - Dashboard
- Discussion
  - Findings & Implications
- Conclusion
- Appendix

EXECUTIVE SUMMARY

Methodologies:

Data collection and wrangling for preprocessing.

Exploratory Data Analysis (EDA) using data visualization and SQL.

Developed an interactive map using Folium.

Created a data dashboard using Plotly Dash.

Conducted predictive analysis with classification techniques.

Results:

Key insights from exploratory data analysis.

Interactive analytics demo presented via screenshots.

Successful predictive analysis outcomes.

IBM Developer

SKILLS NETWORK

PROJECT BACKGROUND & INTRODUCTION

Project Background:

- This project aimed to predict the success of the Falcon 9 first stage landing. SpaceX offers rocket launches at a significantly lower cost ($62 million) compared to competitors ($165 million or more), largely due to its ability to reuse the first stage of its rockets. Predicting the likelihood of a successful landing helps estimate launch costs, providing valuable insights for potential competitors bidding against SpaceX.

Key Challenges:

- Identifying factors influencing the rocket's successful landing.
- Understanding how various rocket-related variables affect the success rate of landings.
- Determining optimal conditions for SpaceX to achieve the highest success rate for rocket landings.

METHODOLOGY

Data Collection and Analysis Methodology:
Extract a Falcon 9 launch records HTML table from Wikipedia
Parsed the table and converted it into a Pandas data frame
Cleaned and prepared data for machine learning, including one-hot encoding and removing irrelevant columns.
Conducted exploratory data analysis (EDA) using visualization techniques and SQL queries, such as scatter plots and bar graphs, to identify patterns and relationships.
Created interactive visualizations using Folium and Plotly Dash.
Built, tuned, and evaluated predictive models using classification techniques.

IBM Developer

SKILLS NETWORK

METHODOLOGY | WEBSCRAPING | DATA EXTRACTION

URL GITHUB REMEMBER

TASK 1: Request the Falcon9 Launch Wiki page from its URL

First, let's perform an HTTP GET method to request the Falcon9 Launch HTML page, as an HTTP response.

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

Create a `BeautifulSoup` object from the HTML `response`

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

Print the page title to verify if the `BeautifulSoup` object was created properly

```
# Use soup.title attribute
soup.title
```

IBM Developer

SKILLS NETWORK

METHODOLOGY | WEBSCRAPING | PARSING

URL GITHUB REMEMBER

TASK 3: Create a data frame by parsing the launch HTML tables

We will create an empty dictionary with keys from the extracted column names in the previous task. Later, this dictionary will be converted into a Pandas dataframe

In [12]:
```
launch_dict = dict.fromkeys(column_names)
launch_dict
```

Out[12]:
```
{'Flight No.': None,
 'Date and time ( )': None,
 'Launch site': None,
 'Payload': None,
 'Payload mass': None,
 'Orbit': None,
 'Customer': None,
 'Launch outcome': None}
```

In [13]:
```
# Remove an irrelvant column
del launch_dict['Date and time ( )']

# Let's initial the launch_dict with each value to be an empty list
launch_dict['Flight No.'] = []
launch_dict['Launch site'] = []
launch_dict['Payload'] = []
launch_dict['Payload mass'] = []
launch_dict['Orbit'] = []
launch_dict['Customer'] = []
launch_dict['Launch outcome'] = []
# Added some new columns
launch_dict['Version Booster'] = []
launch_dict['Booster landing'] = []
launch_dict['Date'] = []
launch_dict['Time'] = []
```

We can now convert the data to a CSV format

After you have fill in the parsed launch record values into `launch_dict`, you can create a dataframe from it.

```
df = pd.DataFrame(launch_dict)
```

We can now export it to a **CSV** for the next section, but to make the answers consistent and in case you have difficulties finishing this lab.

Following labs will be using a provided dataset to make each lab independent.

```
df.to_csv('spacex_web_scraped.csv', index=False)
```

IBM Developer

SKILLS NETWORK

METHODOLOGY | DATA WRANGLING

Introduction to Data Wrangling

URL GITHUB REMEMBER

The dataset includes several scenarios where the booster did not land successfully. For example, "True Ocean" indicates a successful ocean landing, while "False Ocean" denotes an unsuccessful one. Similarly, "True RTLS" and "False RTLS" refer to successful and unsuccessful landings on a ground pad, and "True ASDS" and "False ASDS" represent successful and unsuccessful landings on a drone ship. These outcomes are converted into training labels: 1 for a successful landing and 0 for an unsuccessful one.

Process Conducted:

Created landing outcome labels.

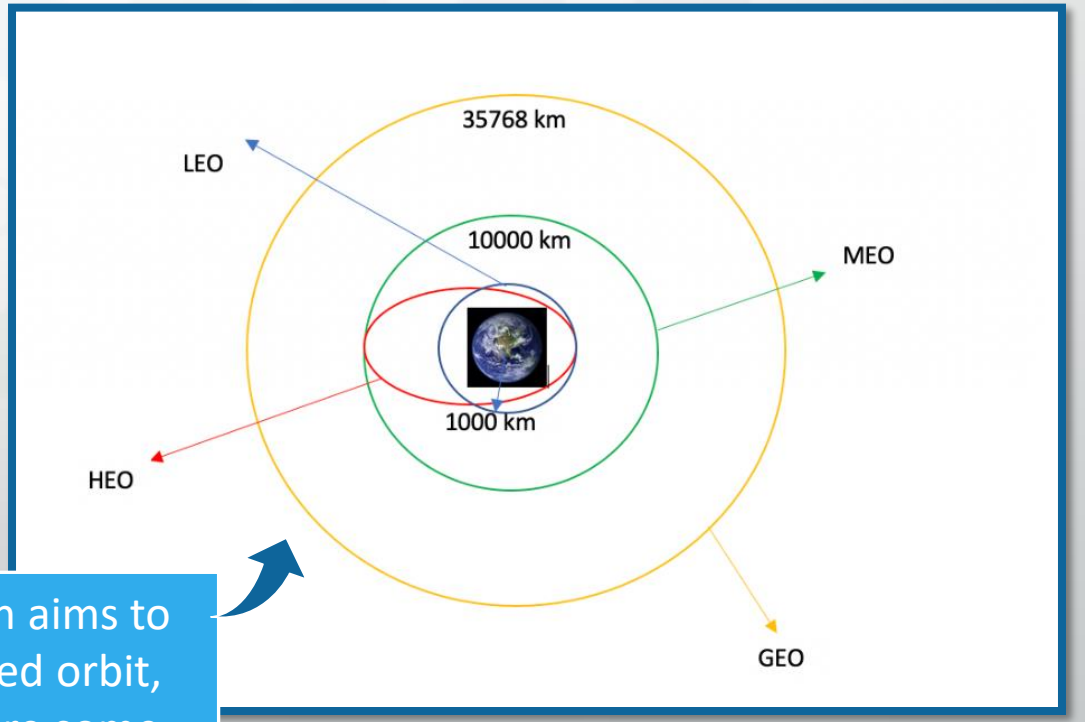
Performed Exploratory Data Analysis (EDA).

Exported dataset to a .CSV file.

Calculated the number of launches per site.

Evaluated mission outcomes by orbit type.

Analyzed the number and frequency of each orbit type.



Each launch aims to an dedicated orbit, and here are some common orbit types:

IBM Developer

SKILLS NETWORK

METHODOLOGY | EDA AND INTERACTIVE VISUAL ANALYTICS

Scatter Plots: Scatter plots illustrate the relationship or correlation between two variables. They are particularly useful when dealing with a large volume of data points, as they provide insights into how changes in one variable may be associated with changes in another, helping to identify trends, clusters, or outliers.

Flight Number vs. Payload Mass: Shows the correlation between the order of launches and the payload weight, indicating trends in SpaceX's payload capacity over time.

Flight Number vs. Launch Site: Maps flight numbers to specific launch sites, highlighting frequency and patterns of site usage.

Payload Mass vs. Launch Site: Examines the variation in payload weights by different launch sites, reflecting site-specific capabilities or mission types.

Orbit vs. Flight Number: Displays the relationship between flight numbers and their targeted orbits, suggesting shifts in SpaceX's strategic priorities or demand for certain orbital destinations.

Payload Mass vs. Orbit: Analyzes the association between payload weight and orbit type, revealing trends related to design constraints or mission preferences for specific orbits.

Orbit vs. Payload Mass: Highlights how payload mass is distributed across different orbital destinations, providing insights into the payload capabilities required for various orbital types.

IBM Developer

URL GITHUB REMEMBER

SKILLS NETWORK

METHODOLOGY | EDA AND INTERACTIVE VISUAL ANALYTICS

URL GITHUB REMEMBER

Bar Graphs: Bar graphs provide a straightforward way to compare data across different groups. By representing categories on one axis and discrete values on the other, they effectively show relationships and can highlight significant changes in data, making them ideal for comparisons across categories or time.

Mean Payload Mass vs. Orbit: This bar graph shows the average payload mass for each type of orbit. By comparing the mean payload masses, it becomes clear which orbits typically host heavier or lighter payloads. This can provide insights into the types of missions usually undertaken for each orbital category.

IBM Developer

SKILLS NETWORK

METHODOLOGY | EDA AND INTERACTIVE VISUAL ANALYTICS URL GITHUB REMEMBER

Line Graphs: Line graphs are effective for displaying data trends over time. They clearly show how variables change, helping to predict future outcomes based on existing data patterns. They are particularly useful when tracking changes over continuous intervals, such as years.

Success Rate vs. Year: This line graph displays the success rate of launches over time, plotted by year. It shows the percentage of successful launches per year and helps visualize trends in launch success. This can be crucial for understanding SpaceX's reliability and performance improvements or challenges over time.

IBM Developer

SKILLS NETWORK

METHODOLOGY | PREDICTIVE ANALYSIS

Methodology Synopsis: Conduct EDA, create a class label column, standardize features, split the data into training and test sets, and use cross-validation for hyperparameter tuning. Evaluate models on various performance metrics to choose the best one.

Exploratory Data Analysis (EDA) and Determining Training Labels: Understand the dataset by summarizing statistics, visualizing distributions, and identifying patterns. Address missing values, outliers, and data types. Define and create the target variable for classification (e.g., churn = 1, no churn = 0).

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1
0	1.0	6104.959412	1.0	1.0	0.0	0.0
1	2.0	525.000000	1.0	1.0	0.0	0.0
2	3.0	677.000000	1.0	1.0	0.0	0.0
3	4.0	500.000000	1.0	1.0	0.0	0.0
4	5.0	3170.000000	1.0	1.0	0.0	0.0

Creating a Class Column: Add a column for the class label based on the target variable, which will be used as the output in machine learning models. Address any class imbalance using techniques like oversampling or undersampling to enhance model performance.

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
transform = preprocessing.StandardScaler()
```

```
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`.

Standardizing the Data: Transform the features to have a mean of 0 and a standard deviation of 1. This ensures that all features are on the same scale, which is crucial for algorithms sensitive to feature scales, like SVM and Logistic Regression.

SKILLS NETWORK

METHODOLOGY | PREDICTIVE ANALYSIS

Splitting into Training and Test Data: Divide the dataset into training (e.g., 70%) and test sets (e.g., 30%). The training set is used to build models, while the test set evaluates their performance to assess generalization.

Hyper Tuning :Optimize model performance by tuning hyperparameters using techniques such as Grid Search or Random Search with cross-validation. Adjust parameters like kernel type for SVM, tree depth for Classification Trees, and regularization strength for Logistic Regression..

Evaluating Model Performance: Compare models based on metrics such as accuracy, precision, recall, F1-score, and ROC-AUC. Consider model interpretability, speed, and scalability to select the most effective model.

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`

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size=0.2, random_state=2)
print ('Train set:', X_train.shape,  Y_train.shape)
print ('Test set:', X_test.shape,  Y_test.shape)
```

```
Train set: (72, 83) (72,)
Test set: (18, 83) (18,)
```

we can see we only have 18 test samples.

```
Y_test.shape
```

```
(18,)
```

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']}
```

```
lr = LogisticRegression()
grid_search = GridSearchCV(lr, parameters, cv=10)
logreg_cv = grid_search.fit(X_train, Y_train)
```

TASK 9

Calculate the accuracy of tree_cv on the t

```
tree_cv.score(X_test, Y_test)
```

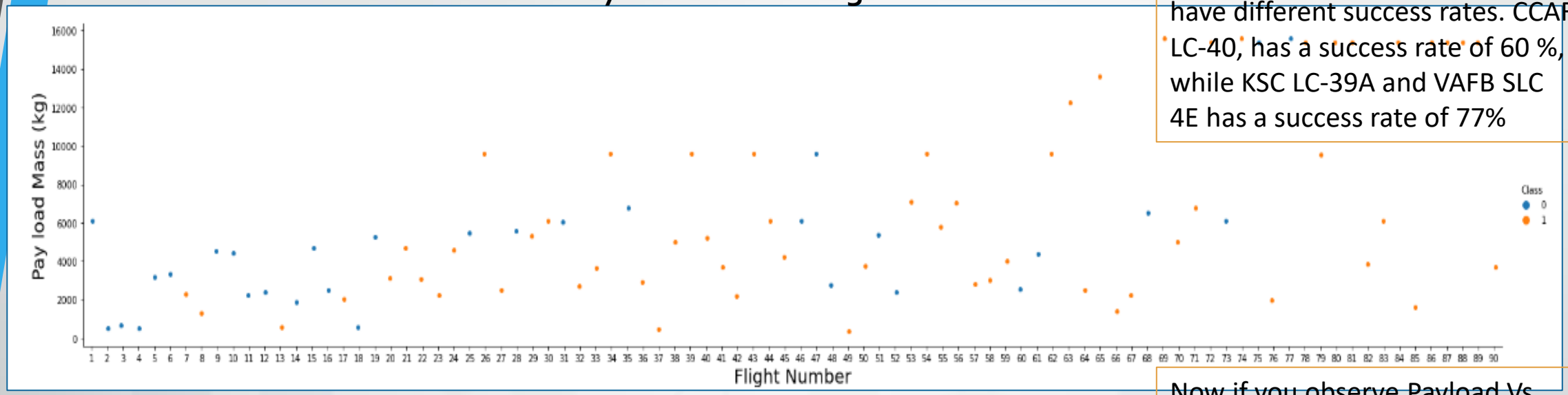
```
0.8333333333333334
```

IBM Developer

SKILLS NETWORK

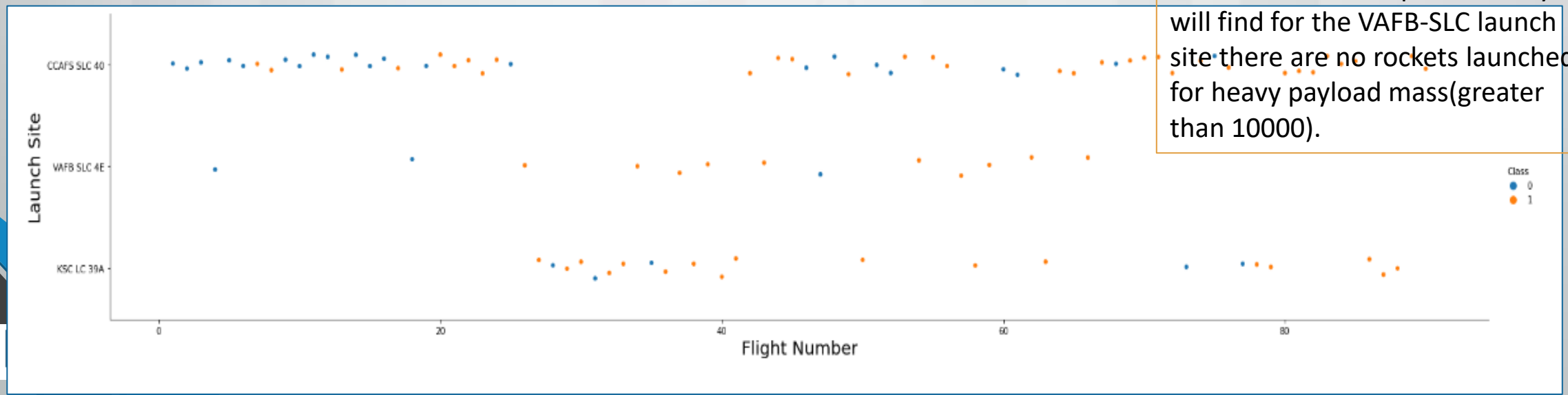
RESULTS | EDA WITH VISUALIZATION

Payload Mass VS Flight Number



We see that different launch sites have different success rates. CCAFS LC-40, has a success rate of 60 %, while KSC LC-39A and VAFB SLC 4E has a success rate of 77%

Launch Site VS Flight Number



Now if you observe Payload Vs. Launch Site scatter point chart you will find for the VAFB-SLC launch site there are no rockets launched for heavy payload mass(greater than 10000).

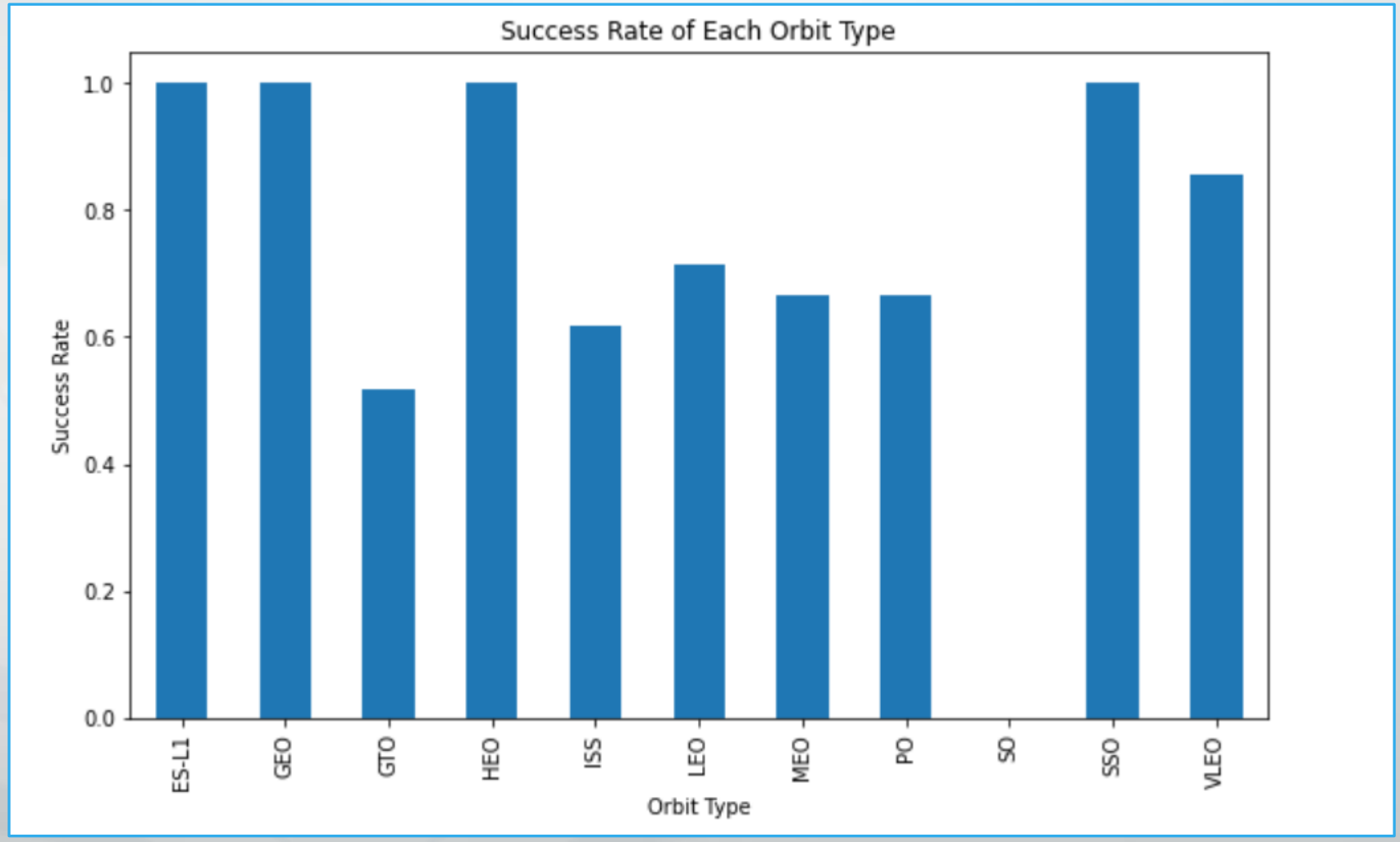
IBM

RESULTS | EDA WITH VISUALIZATION

Orbit:

GEO,
HEO,
SSO,
ES-L1

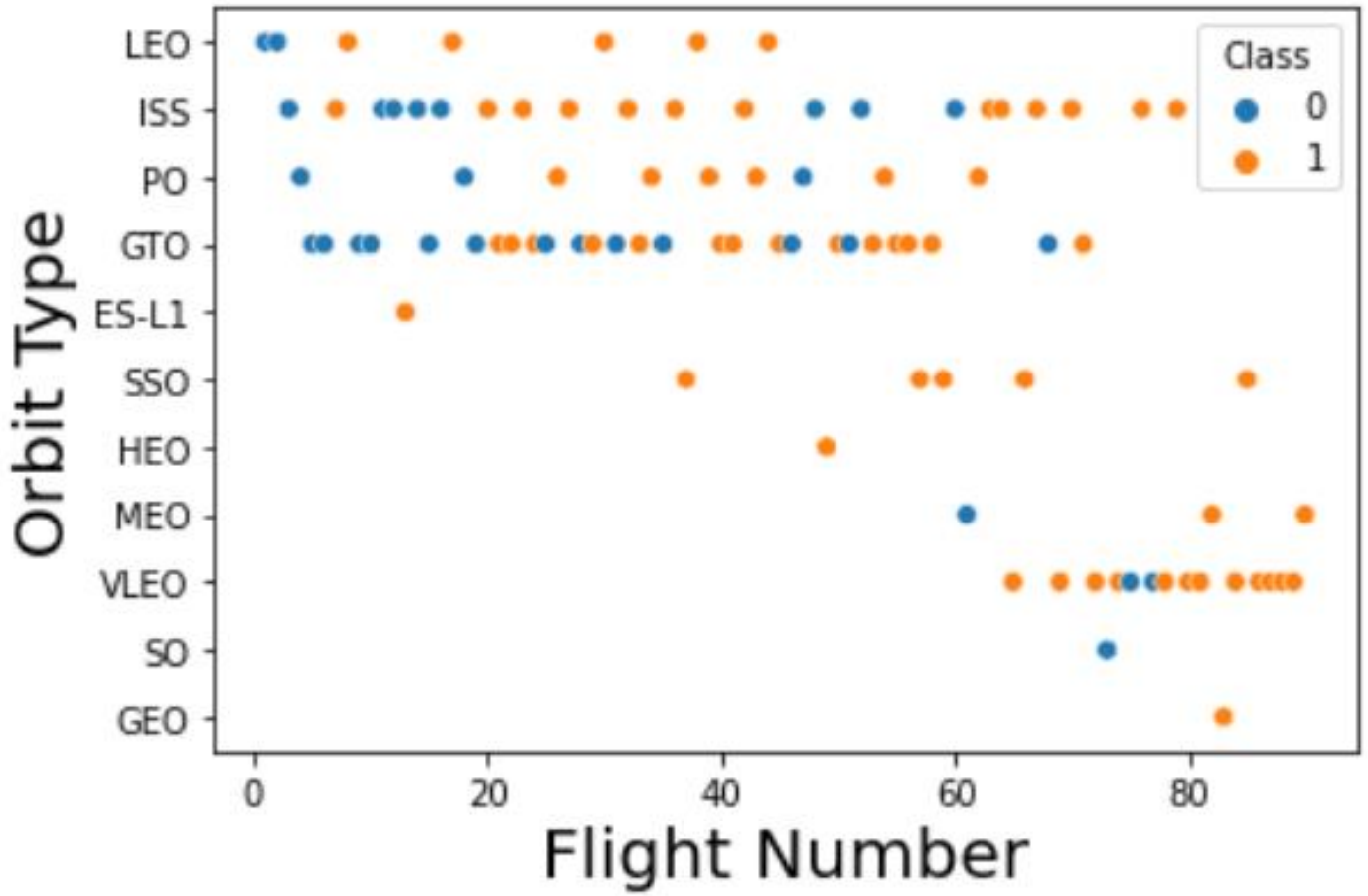
They have the best Success Rate



RESULTS | EDA WITH VISUALIZATION

LEO Success: Related to flight number.
GTO Success: No apparent relationship to flight number.

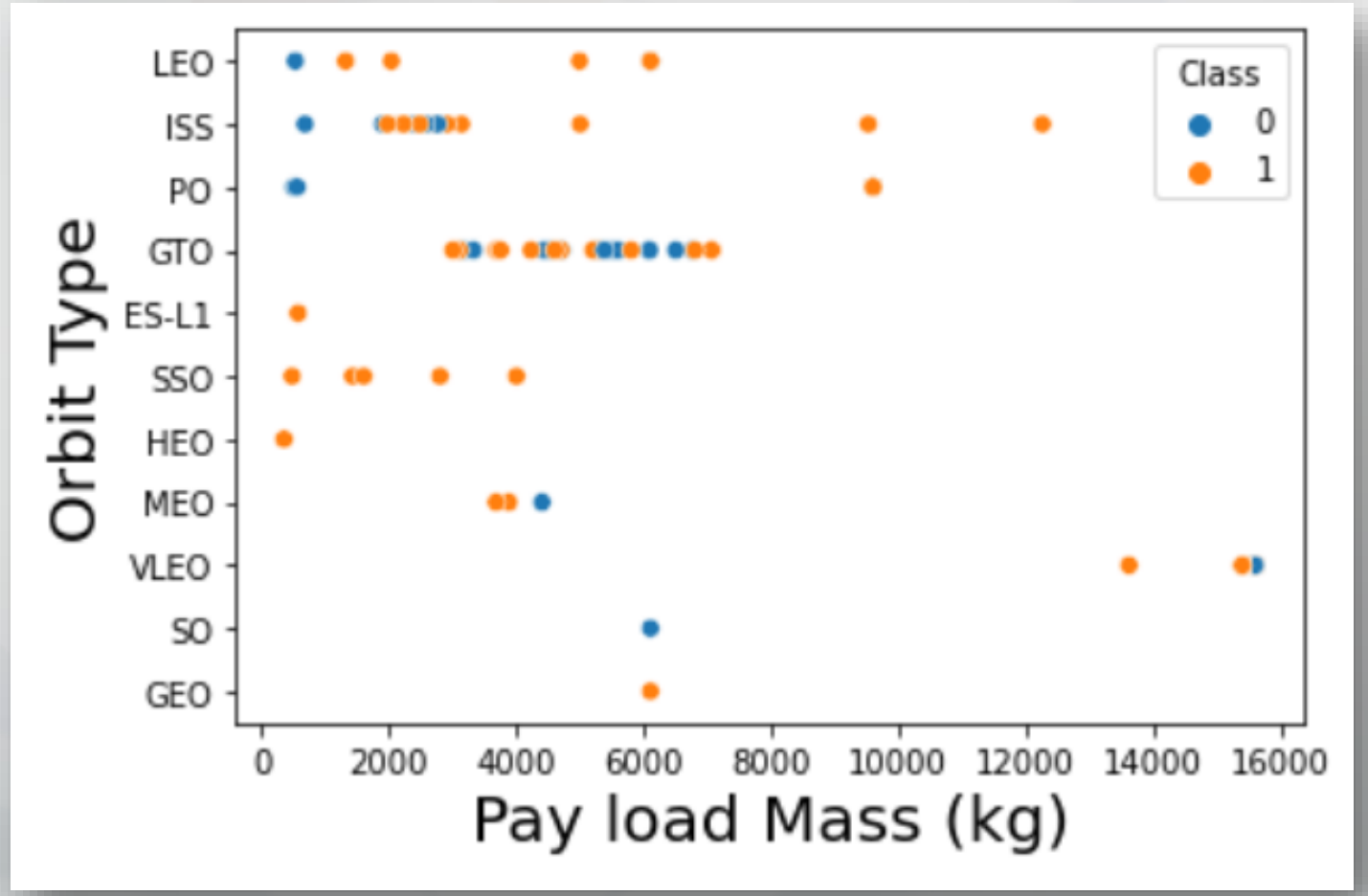
Orbit Type VS Flight Number



RESULTS | EDA WITH VISUALIZATION

Orbit Type VS Pay Load

Heavy payloads: Negative impact on GTO, positive impact on GTO and Polar LEO (ISS).

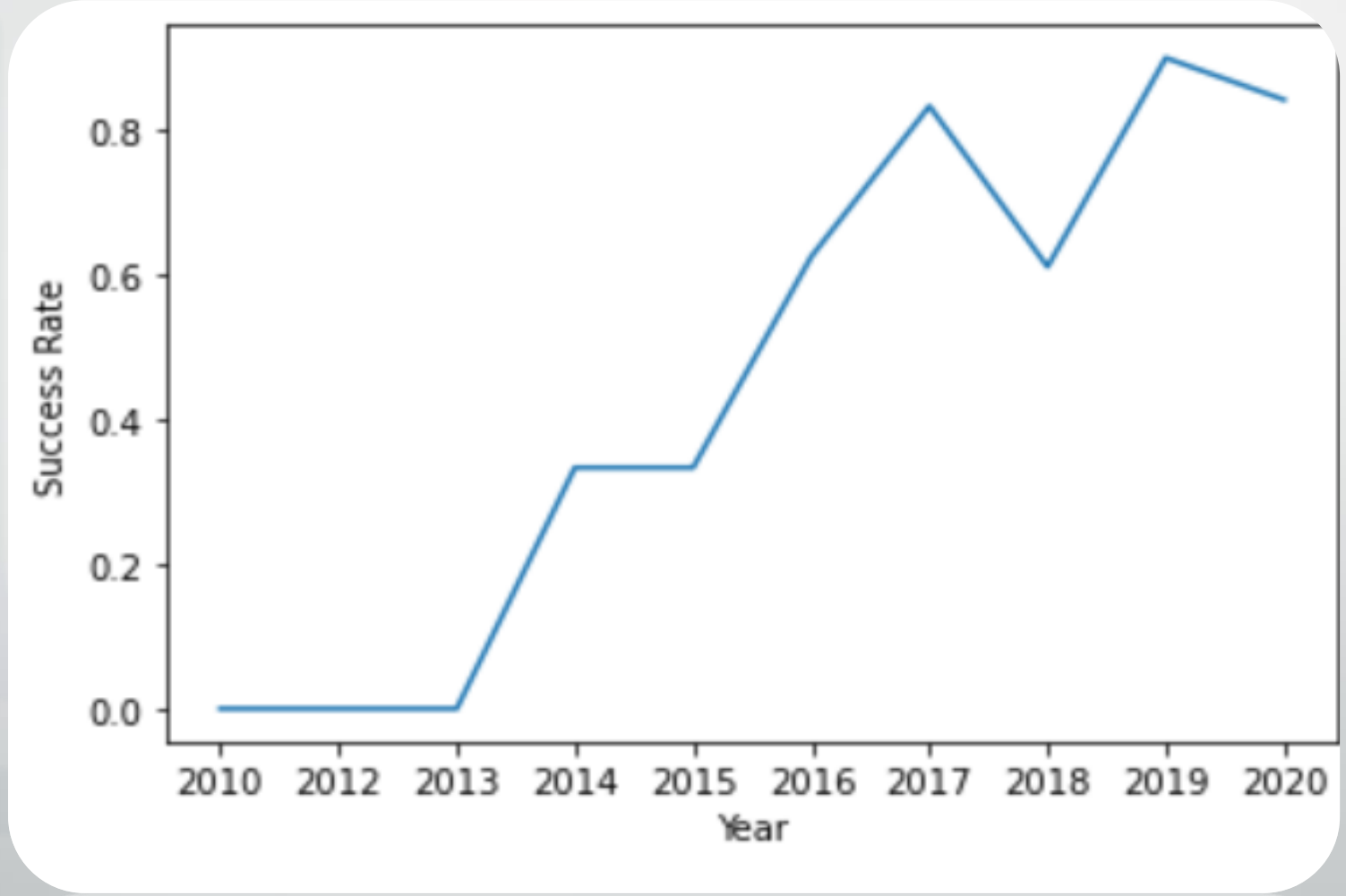


RESULTS | EDA WITH VISUALIZATION

Success Rate VS Year

Yearly Success Rate:

Success rate since 2013 has been increasing till 2020



RESULTS | EDA WITH SQL

SQL QUERY FOR NAMES OF THE UNIQUE LAUNCH SITES

Definition:

The **DISTINCT** keyword in SQL is used to return only unique (different) values from a column or combination of columns in a query result. It removes duplicate records.

Importance:

- Ensures data accuracy by filtering out duplicates.
- Optimizes query results for analysis by providing a clear dataset.
- Improves performance in some cases by reducing the result size.

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

Task 1

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

```
%sql SELECT DISTINCT LAUNCH_SITE FROM SPACEXTBL;
```

RESULTS | EDA WITH SQL

SQL QUERY FOR 5 RECORDS WHERE LAUNCH SITES BEGIN

Launch_Site.

Importance:

Identifying the launch site is crucial in the context of space missions or logistics because it affects the trajectory, payload capacity, costs, and success probability of a launch. The location determines the orbital paths available, weather conditions, regulatory considerations, and operational logistics.

launch_site
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40

Task 2

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

```
%sql SELECT LAUNCH_SITE FROM SPACEXTBL WHERE LAUNCH_SITE LIKE 'CCA%' LIMIT 5;
```

IBM Developer

RESULTS | EDA WITH SQL

SQL QUERY FOR:

Task3

Total payload mass carried by boosters

Task 4

Average payload mass carried by booster version F9 v1.1

Task 3

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

```
%sql SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE CUSTOMER = 'NASA (CRS)';
```

```
 * ibm_db_sa://rlv46682:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:32731/bludb
Done.
```

1
45596

Task 4

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

```
%sql SELECT AVG(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE BOOSTER_VERSION = 'F9 v1.1';
```

```
 * ibm_db_sa://rlv46682:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:32731/bludb
Done.
```

1
2928

IBM Developer

SKILLS NETWORK

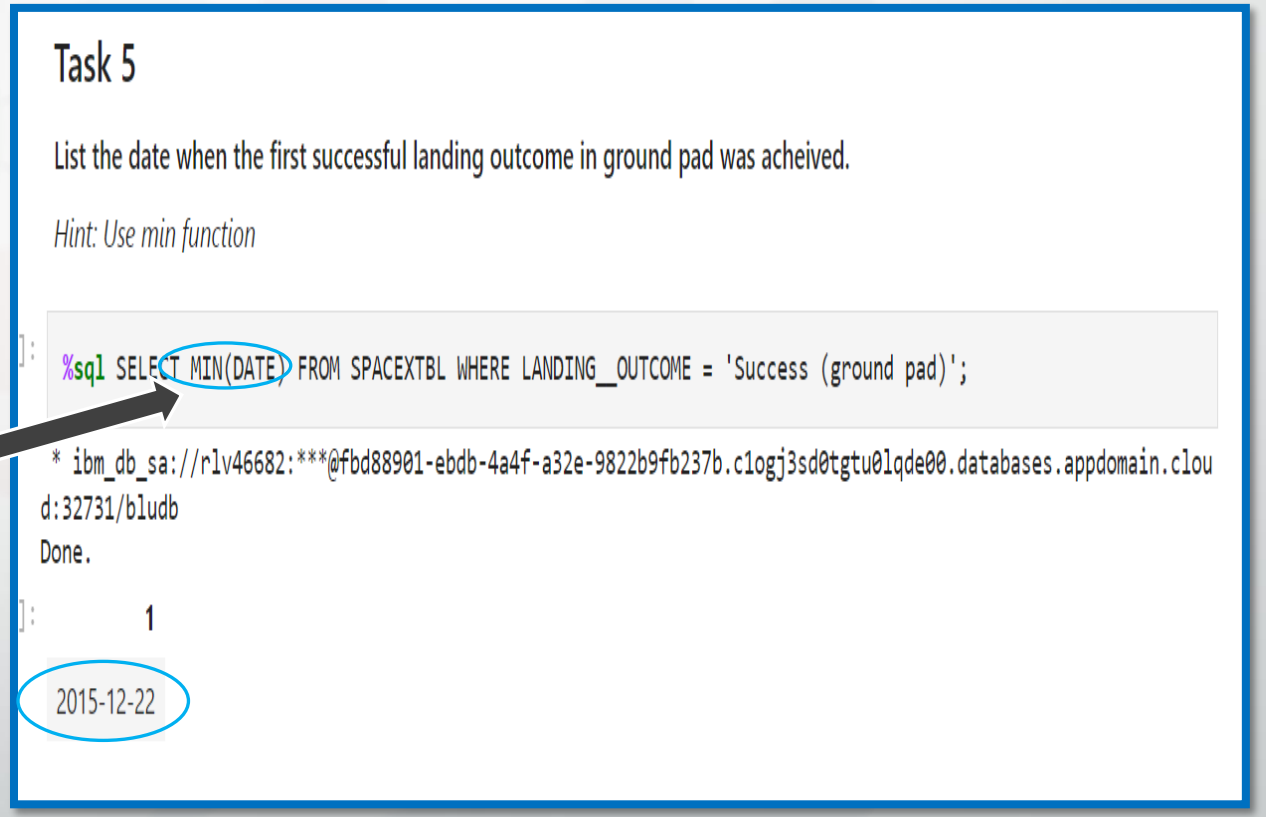
RESULTS | EDA WITH SQL

SQL QUERY FOR LIST THE DATE WHEN THE FIRST SUCCESSFUL LANDING OUTCOME IN GROUND PAD WAS ACHIEVED

Min Function.

Importance:

The **MIN** function in SQL returns the smallest value from a specified column in a dataset. It is important because it allows for quick identification of the minimum value in large datasets, which is useful in analytics, reporting, and data validation tasks



IBM Developer

SKILLS NETWORK

RESULTS | EDA WITH SQL

SQL QUERY FOR LIST THE NAMES OF THE BOOSTERS WHICH HAVE SUCCESS IN DRONE SHIP AND HAVE PAYLOAD MASS GREATER THAN 4000 BUT LESS THAN 6000

Task 6

List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000

```
%sql SELECT DISTINCT BOOSTER_VERSION FROM SPACEXTBL WHERE LANDING__OUTCOME = 'Success (drone ship)' AND (PAY
```

```
 * ibm_db_sa://rlv46682:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.c1ogj3sd0tgtu0lqde00.databases.appdomain.clou
d:32731/bludb
Done.
```

booster_version
F9 FT B1021.2
F9 FT B1031.2
F9 FT B1022
F9 FT B1026

Query selects unique values in the booster column

RESULTS | EDA WITH SQL

SQL QUERY FOR TOTAL NUMBER OF SUCCESSFUL AND FAILURE MISSION OUTCOMES

Define:

The COUNT() function in SQL returns the number of rows that match a specified condition in a query. It is commonly used to determine the number of records in a table or the number of non-null values in a column.

Importance:

Data Analysis: Helps quickly summarize and analyze datasets by counting occurrences.
Performance: Efficiently retrieves counts without needing to load all data.
Validation: Ensures data integrity by verifying expected row counts or conditions.

Task 7

List the total number of successful and failure mission outcomes

```
%sql SELECT MISSION_OUTCOME, COUNT(*) FROM SPACEXTBL GROUP BY MISSION_OUTCOME;
```

```
 * ibm_db_sa://rlv46682:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:32731/bludb
Done.
```

mission_outcome	2
Failure (in flight)	1
Success	99
Success (payload status unclear)	1

IBM Developer

SKILLS NETWORK

RESULTS | EDA WITH SQL

SQL QUERY FOR THE BOOSTER_VERSIONS WHICH HAVE CARRIED THE MAXIMUM PAYLOAD MASS. USE A SUBQUERY

Task 8

List the names of the booster_versions which have carried the maximum payload mass. Use a subquery

```
%sql SELECT DISTINCT BOOSTER_VERSION FROM SPACEXTBL WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTBL)
```

```
* ibm_db_sa://rlv46682:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:32731/bludb
Done.
```

booster_version
F9 B5 B1048.4
F9 B5 B1048.5
F9 B5 B1049.4
F9 B5 B1049.5
F9 B5 B1049.7
F9 B5 B1051.3
F9 B5 B1051.4
F9 B5 B1051.6
F9 B5 B1056.4
F9 B5 B1058.3
F9 B5 B1060.2
F9 B5 B1060.3

RESULTS | EDA WITH SQL

SQL QUERY FOR the FAILED LANDING_OUTCOMES IN DRONE SHIP, THEIR BOOSTER VERSIONS, AND LAUNCH SITE NAMES FOR IN YEAR 2015

Task 9

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

```
%sql SELECT LANDING__OUTCOME, BOOSTER_VERSION, LAUNCH_SITE FROM SPACEXTBL WHERE LANDING__OUTCOME = 'Failure (drone ship)' AND
```

```
 * ibm_db_sa://rlv46682:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:32731/bludb
Done.
```

landing__outcome	booster_version	launch_site
Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

RESULTS | EDA WITH SQL

SQL QUERY FOR RANKING THE COUNT OF LANDING OUTCOMES

Task 10

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(*)
FROM SPACEXTBL
WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY LANDING__OUTCOME
ORDER BY COUNT(*) DESC;
```

```
 * ibm_db_sa://rlv46682:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:32731/bludb
Done.
```

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

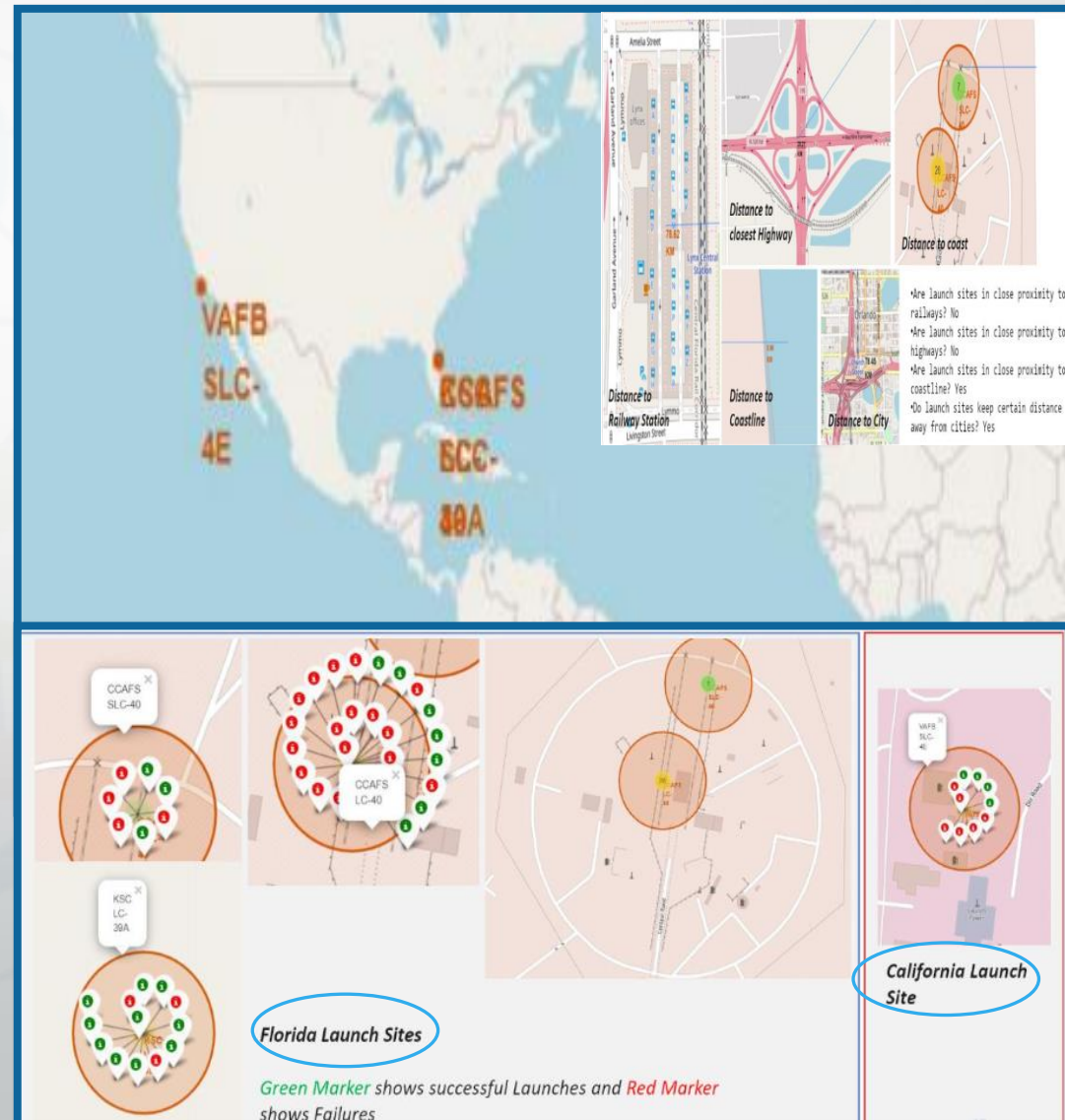
RESULTS | INTERACTIVE MAP WITH FOLIUM RESULTS

Launch sites are in the USA coasts.

Folium is a Python library used for creating interactive maps. It is built on top of **Leaflet.js**, a leading open-source JavaScript library for interactive maps. Folium enables easy visualization of geospatial data by combining Python data handling capabilities with Leaflet's mapping tools.

Importance of Folium:

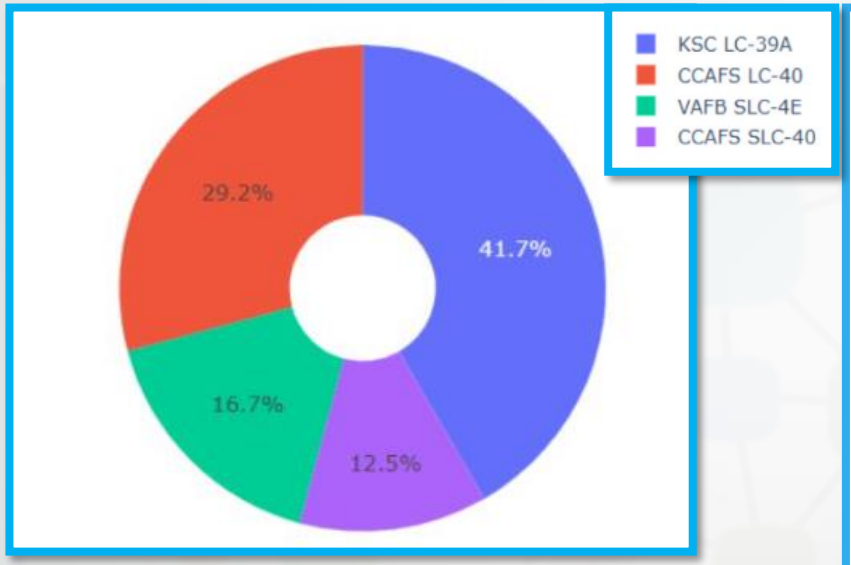
1. **Interactive Maps**: Enables the creation of interactive maps that can be embedded in Jupyter notebooks or web applications.
2. **Data Visualization**: Supports a variety of geospatial data formats, allowing for robust and customizable visualizations.
3. **Integration with Python Ecosystem**: Easily integrates with other Python libraries (e.g., pandas, geopandas) for handling and visualizing complex geospatial data.



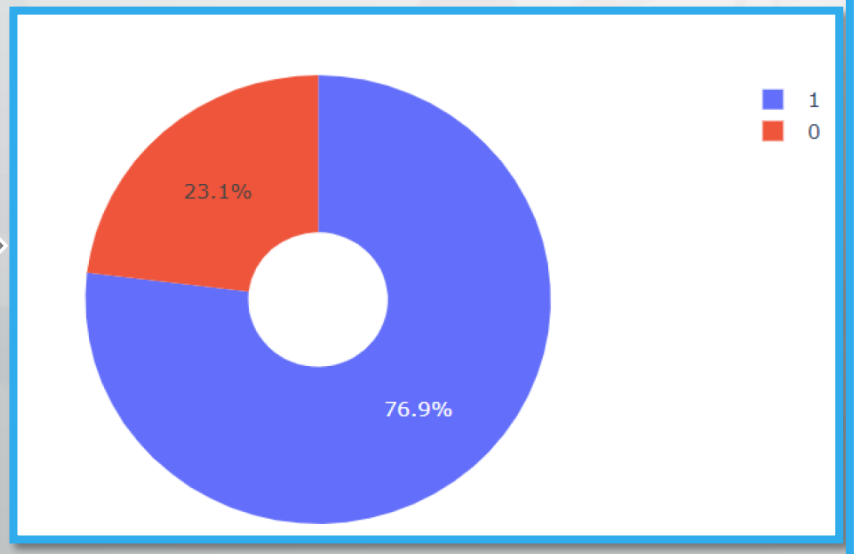
IBM Developer

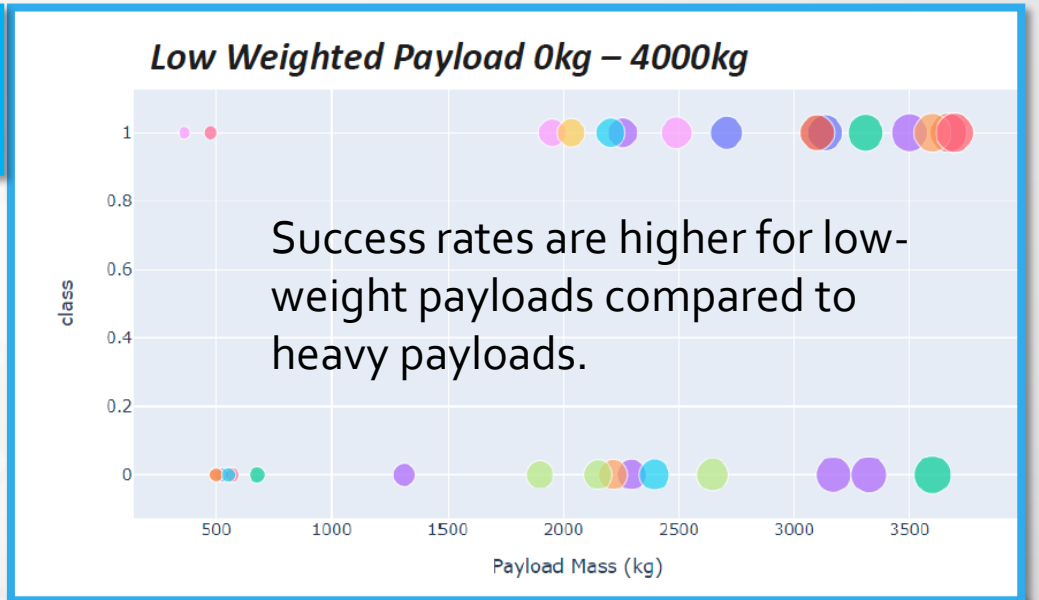
RESULTS | PLOTLY DASH DASHBOARD

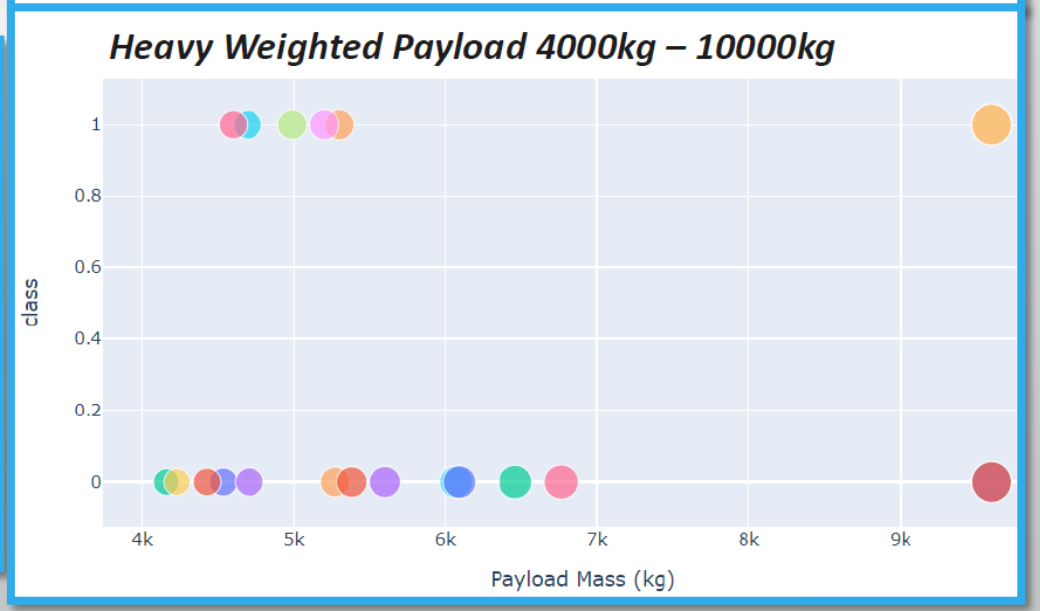
KSC LC-39A achieved most successful launches



KSC LC-39A achieved a 76.9% success rate and a 23.1% failure rate.







IBM Developer

RESULTS | PREDICTIVE ANALYSIS (CLASSIFICATION)

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`.

```
# define hyperparameters to tune
parameters_svm = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}

# define the model
svm = SVC(random_state = 12345)

# define the grid search object
grid_search_svm = GridSearchCV(
    estimator = svm,
    param_grid = parameters_svm,
    scoring = 'accuracy',
    cv = 10
)
# execute search
svm_cv = grid_search_svm.fit(X_train,Y_train)
```

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

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

SVM Outcome

IBM Developer

SKILLS NETWORK

RESULTS | PREDICTIVE ANALYSIS (CLASSIFICATION)

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`.

```
# define hyperparameters to tune
parameters_tree = {'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]}

# define the model
tree = DecisionTreeClassifier(random_state = 12345)

# define the grid search object
grid_search_tree = GridSearchCV(
    estimator = tree,
    param_grid = parameters_tree,
    scoring = 'accuracy',
    cv = 10
)
# execute search
tree_cv = grid_search_tree.fit(X_train, Y_train)
```

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

```
tuned hyperparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 6, 'max_features': 'auto', 'min_samples_leaf':
2, 'min_samples_split': 5, 'splitter': 'random'}
accuracy : 0.8732142857142856
```

Decision Tree Outcome

IBM Developer

SKILLS NETWORK

RESULTS | PREDICTIVE ANALYSIS (CLASSIFICATION)

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`.

```
# define hyperparameters to tune
parameters_knn = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'p': [1,2]}

# define the model
knn = KNeighborsClassifier()

# define the grid search object
grid_search_knn = GridSearchCV(
    estimator = knn,
    param_grid = parameters_knn,
    scoring = 'accuracy',
    cv = 10
)
# execute search
knn_cv = grid_search_knn.fit(X_train, Y_train)
```

KNN Outcome

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

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

IBM Developer

SKILLS NETWORK

RESULTS | PREDICTIVE ANALYSIS (CLASSIFICATION)

TASK 12

Find the method performs best:

```
models = {'KNeighbors':knn_cv.best_score_,
          'DecisionTree':tree_cv.best_score_,
          'LogisticRegression':logreg_cv.best_score_,
          'SupportVector': svm_cv.best_score_}

bestalgorithm = max(models, key=models.get)
print('Best model is', bestalgorithm,'with a score of', models[bestalgorithm])
if bestalgorithm == 'DecisionTree':
    print('Best params is :', tree_cv.best_params_)
if bestalgorithm == 'KNeighbors':
    print('Best params is :', knn_cv.best_params_)
if bestalgorithm == 'LogisticRegression':
    print('Best params is :', logreg_cv.best_params_)
if bestalgorithm == 'SupportVector':
    print('Best params is :', svm_cv.best_params_)
```

Best Performer

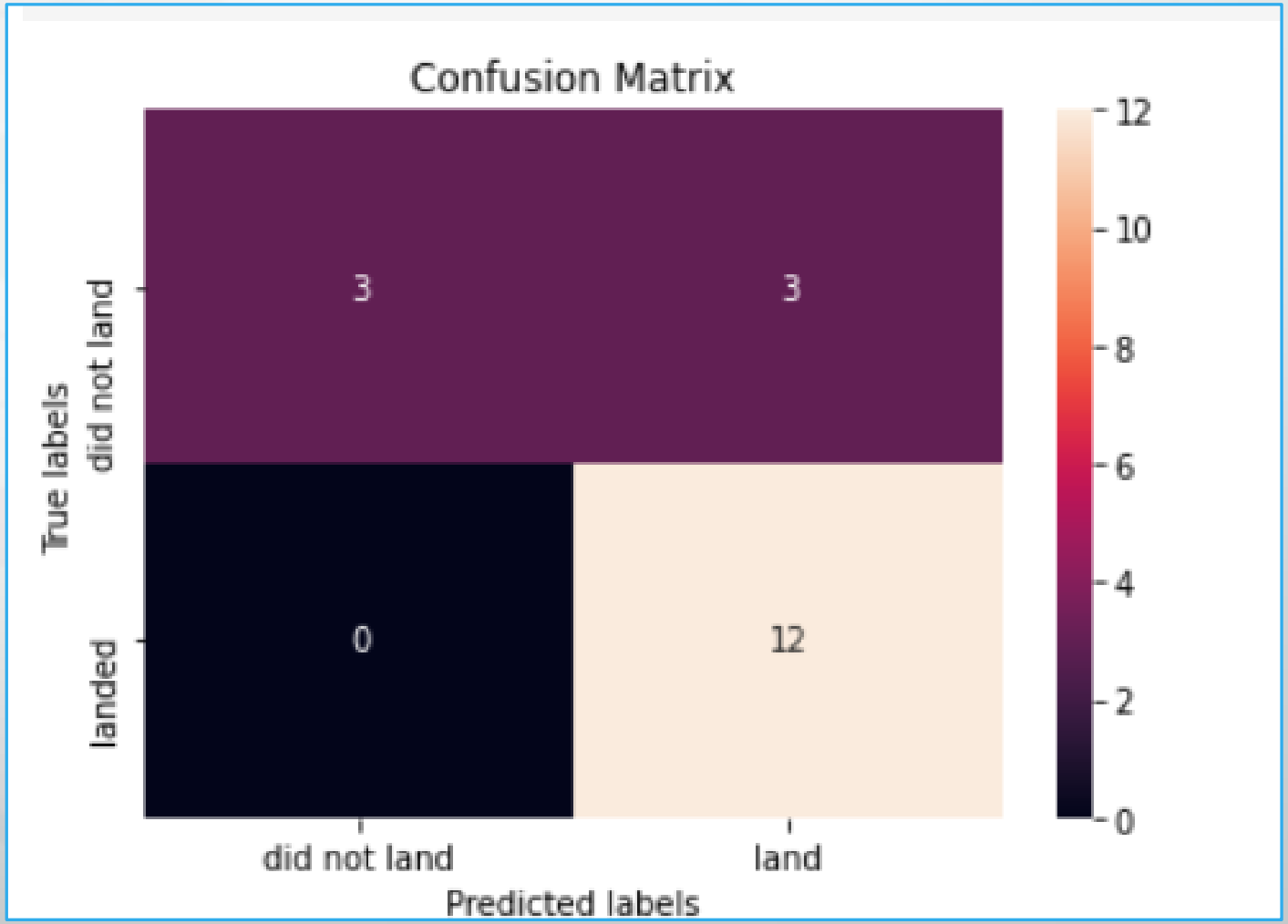
```
Best model is DecisionTree with a score of 0.8732142857142856
Best params is : {'criterion': 'gini', 'max_depth': 6, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_samples_split': 5,
'splitter': 'random'}
```

RESULTS |PREDICTIVE ANALYSIS (CLASSIFICATION)

Best model is Decision Tree with a score of 0.8732142857142856

Note:

The confusion matrix shows that the Tree model can distinguish between classes, but it has a significant issue with false positives.



IBM Developer

SKILLS NETWORK

CONCLUSION

- **Tree Classifier Algorithm**: Most effective for predicting SpaceX launch outcomes, handling complex decision boundaries well.
- **Payload Weight**: Lower-weighted payloads perform better than heavier ones, suggesting heavier payloads face more significant challenges.
- **Success Rates Over Time**: SpaceX's launch success rates improve with time, indicating ongoing refinement of technology and processes.
- **Launch Sites**: Kennedy Space Center LC-39A has the highest success rate, reflecting its critical role and advanced facilities.
- **Optimal Orbits**: GEO, HEO, SSO, and ES L1 orbits have the best success rates, highlighting favorable conditions and technological advantages.

Summary: The Tree Classifier Algorithm is best for predicting launch outcomes, with insights into payload performance, launch site effectiveness, and optimal orbits revealing key factors influencing SpaceX's success and continuous improvements.