WINNING THE SPACE RACE WITH DATA SCIENCE

DILRABO KHIDIROVA

OUTLINE

- Executive summary
- Introduction
- Methodology
- Results
- Conclusion
- Appendix

EXECUTIVE SUMMARY

SUMMARY OF METHODOLIGIES

SUMMARY OF ALL RESULTS

INTRODUCTION

Project background and context

Problems you want to find

METHODOLOGY

- Executive summary
- *Data collection methodology:*
- *Perform data wrangling*
- *Perform exploratory data analysis (EDA) using data visualization and SQL*
- *Perform interactive visual analytics using Folium and Plotly Dash*
- *Perform predictive analysis using classification models.*

MY GITHUB

https://github.com/dilrabonu/IBM_project.git

DATA COLLECTION

Task 1: Request and parse the SpaceX launch data using the GET request

To make the requested JSON results more consistent, we will use the following static response object for this project:

In [9]:
```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_
```

We should see that the request was successfull with the 200 status response code

In [10]:
```
response.status_code
```

Out[10]: 200

Now we decode the response content as a Json using `.json()` and turn it into a Pandas dataframe using `.json_normalize()`

In [36]:
```
# Use json_normalize meethod to convert the json result into a dataframe
import pandas as pd
json_data = response.json()
data = pd.json_normalize(json_data)
```

Using the dataframe `data` print the first 5 rows

In [37]:
```
# Get the head of the dataframe
print(data.head(5))
```

```
      static_fire_date_utc  static_fire_date_unix    net  window  \
0  2006-03-17T00:00:00.000Z           1.142554e+09  False     0.0
1                     None                    NaN  False     0.0
2                     None                    NaN  False     0.0
3  2008-09-20T00:00:00.000Z           1.221869e+09  False     0.0
```

Activate Windows
Go to Settings to activate Windo

https://github.com/dilrabonu/IBM_project.git

DATA SCRAPING COLLECTION

Next, request the HTML page from the above URL and get a `response` object

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.

In [12]:
```
# use requests.get() method with the provided static_url
# assign the response to a object
response=requests.get(static_url)
response
```

Out[12]:
```
<Response [200]>
```

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

In [14]:
```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(response.content, 'html.parser')
```

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

In [17]:
```
# Use soup.title attribute
Title=soup.title
Title
```

Out[17]:
```
<title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```


TASK 2: Extract all column/variable names from the HTML table header

https://github.com/dilrabonu/IBM_project.git

DATA WRANGLING

HTTPS://GITHUB.COM/DILRABONU/IBM_PROJECT.GIT

TASK 1: Calculate the number of launches on each site

The data contains several Space X launch facilities: Cape Canaveral Space Launch Complex 40 **VAFB SLC 4E** , Vandenberg Air Force Base Space Launch Complex 4E **(SLC-4E)**, Kennedy Space Center Launch Complex 39A **KSC LC 39A** .The location of each Launch Is placed in the column `LaunchSite`

Next, let's see the number of launches for each site.

Use the method `value_counts()` on the column `LaunchSite` to determine the number of launches on each site:

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

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

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

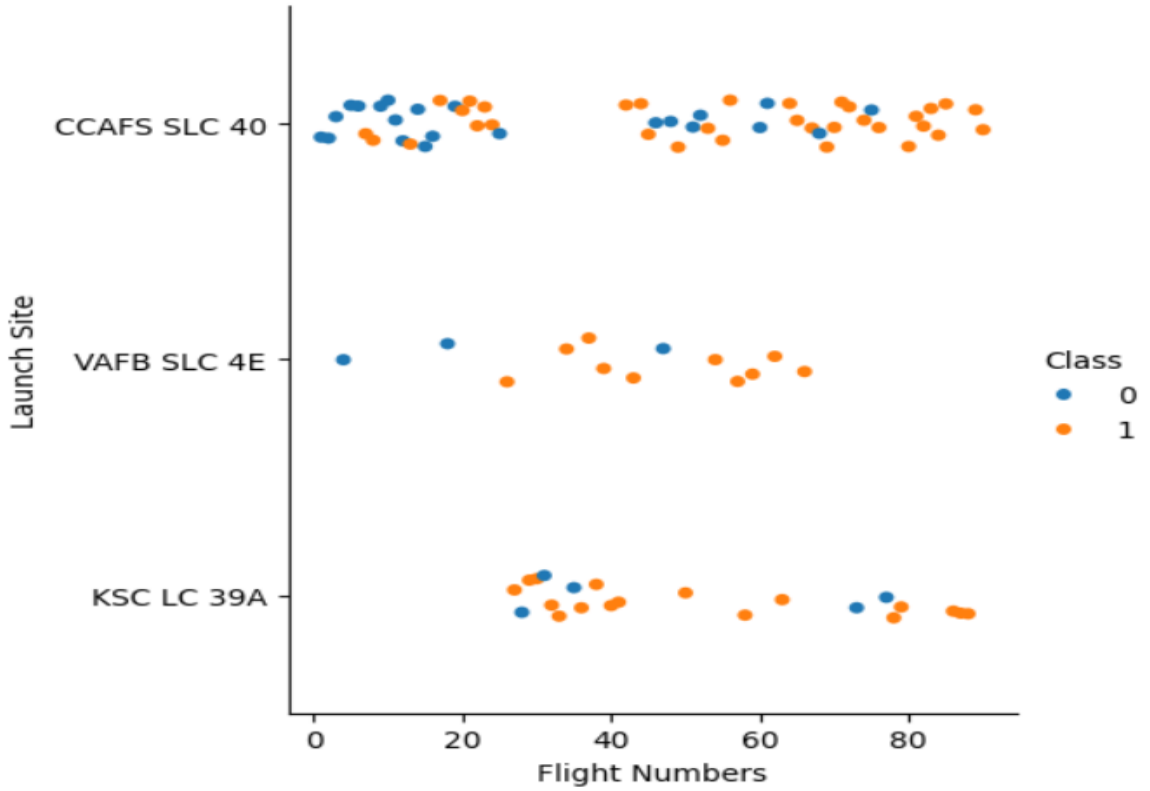
- **LEO**: Low Earth orbit (LEO)is an Earth-centred orbit with an altitude of 2,000 km (1,200 mi) or less (approximately one-third of the radius of Earth),[1] or with at least 11.25 periods per day (an orbital period of 128 minutes or less) and an eccentricity less than 0.25.[2] Most of the manmade objects in outer space are in LEO [1].
- **VLEO**: Very Low Earth Orbits (VLEO) can be defined as the orbits with a mean altitude below 450 km. Operating in these orbits can provide a number of benefits to Earth observation spacecraft as the spacecraft operates closer to the observation[2].
- **GTO** A geosynchronous orbit is a high Earth orbit that allows satellites to match Earth's rotation. Located at 22,236 miles (35,786

DATA VISUALIZATION

https://github.com/dilrabonu/IBM_project.git

In [10]:

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



Task 1

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

In [32]:
```
cursor.execute('SELECT DISTINCT Launch_Site FROM SPACEXTBL')
cursor.fetchall()
```

Out[32]:
```
[('CCAFS LC-40',), ('VAFB SLC-4E',), ('KSC LC-39A',), ('CCAFS SLC-40',)]
```

Task 2

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

In [36]:
```
cursor.execute("SELECT * FROM SPACEXTBL WHERE Launch_Site LIKE 'CCA%' LIMIT 5")
cursor.fetchall()
```

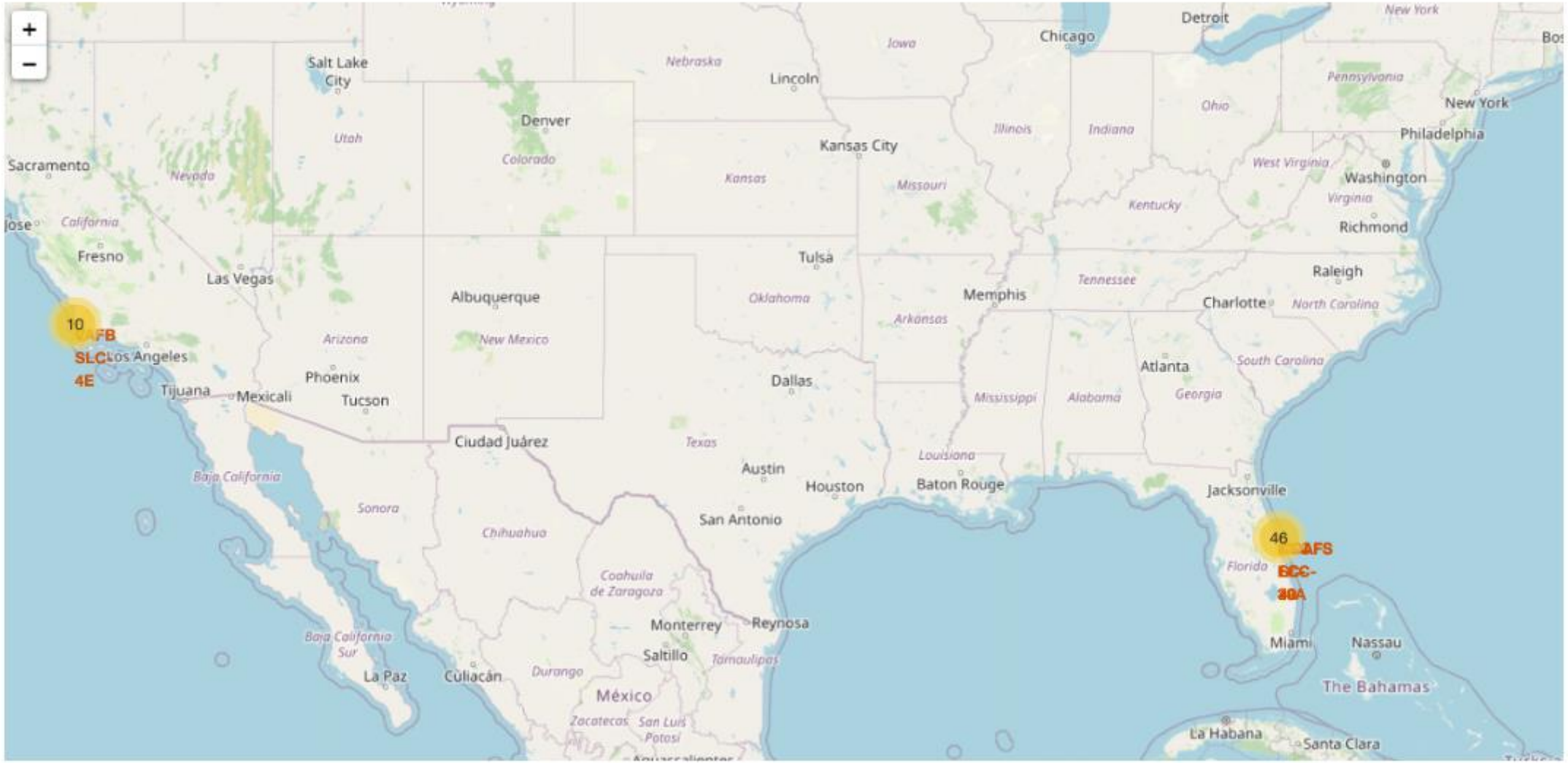
Out[36]:
```
[('2010-06-04',
  '18:45:00',
  'F9 v1.0  B0003',
  'CCAFS LC-40',
  'Dragon Spacecraft Qualification Unit',
  0,
  'LEO',
  'SpaceX',
  'Success',
  'Failure (parachute)'),
 ('2010-12-08',
  '15:43:00',
  'F9 v1.0  B0004',
  'CCAFS LC-40',
  'Dragon demo flight C1, two CubeSats, barrel of Brouere cheese',
```

DATA SQL AND EDA

1. https://github.com/dilrabonu/IBM_project.git

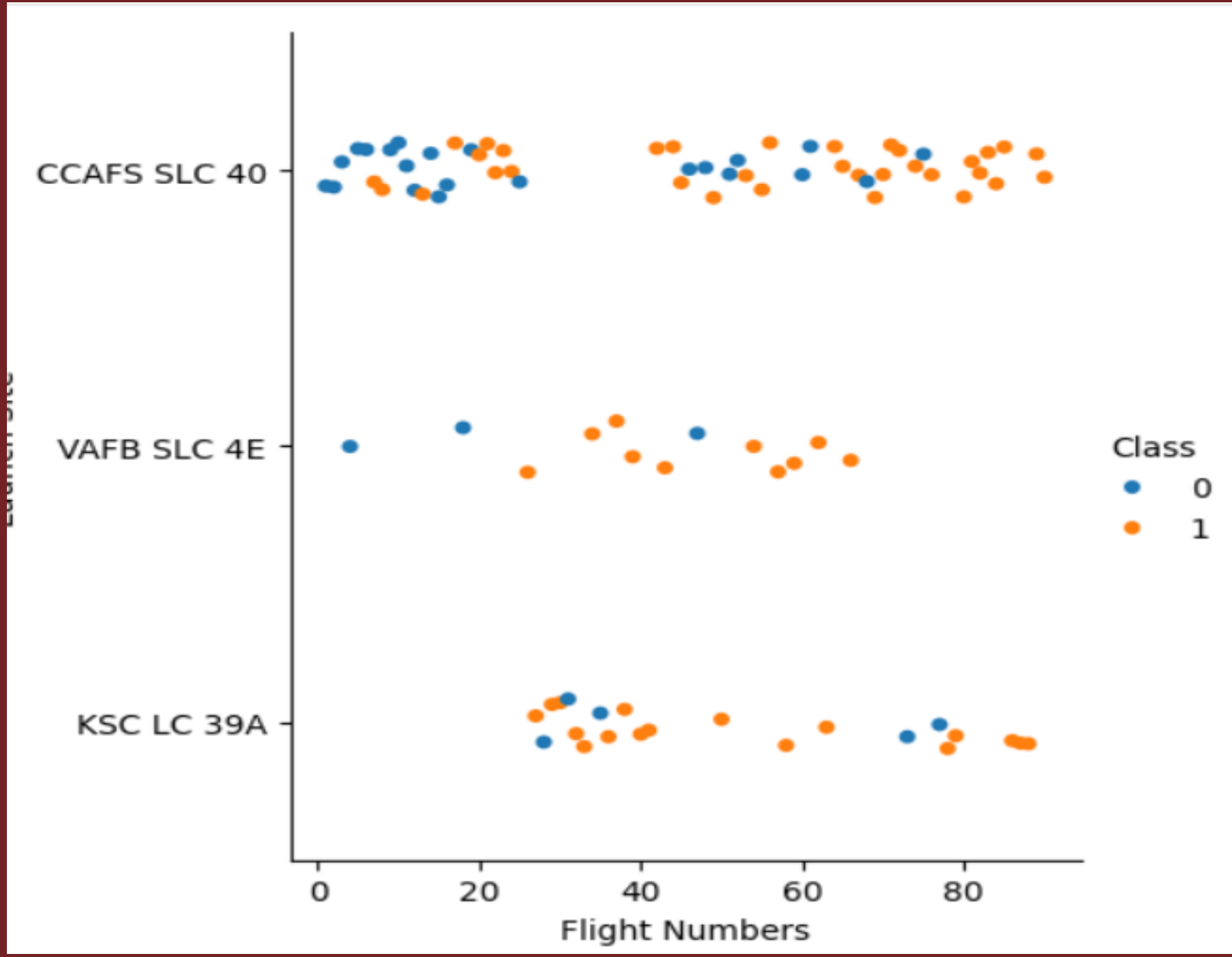
FOLIUM https://github.com/dilrabonu/IBM

Impact factor	Measurement	Target	Achieved



THE MAIN REASON IS TO UNDERSTAND DATA EASILY AND EXPLAINE TO STAKEHOLDERS UNDERSTANDABLE

https://github.com/dilrabonu/IBM_project.git



PREDICTION

https://github.com/dilrabonu/IBM_project.git

TASK 4

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

In [19]:
```
logreg = LogisticRegression()
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
logreg_cv = GridSearchCV(logreg, parameters, cv=10)

logreg_cv.fit(X_train, Y_train)

print("Best parameters:", logreg_cv.best_params_)
```

```
Best parameters: {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
```

In [20]:
```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()
```

We output the `GridSearchCV` object for logistic regression. We display the best parameters using the data attribute `best`
the accuracy on the validation data using the data attribute `best_score_`.

In [21]:
```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8464285714285713
```

PREDICTIVE ANALYSIS

https://github.com/dilrabonu/IBM_Project.git

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

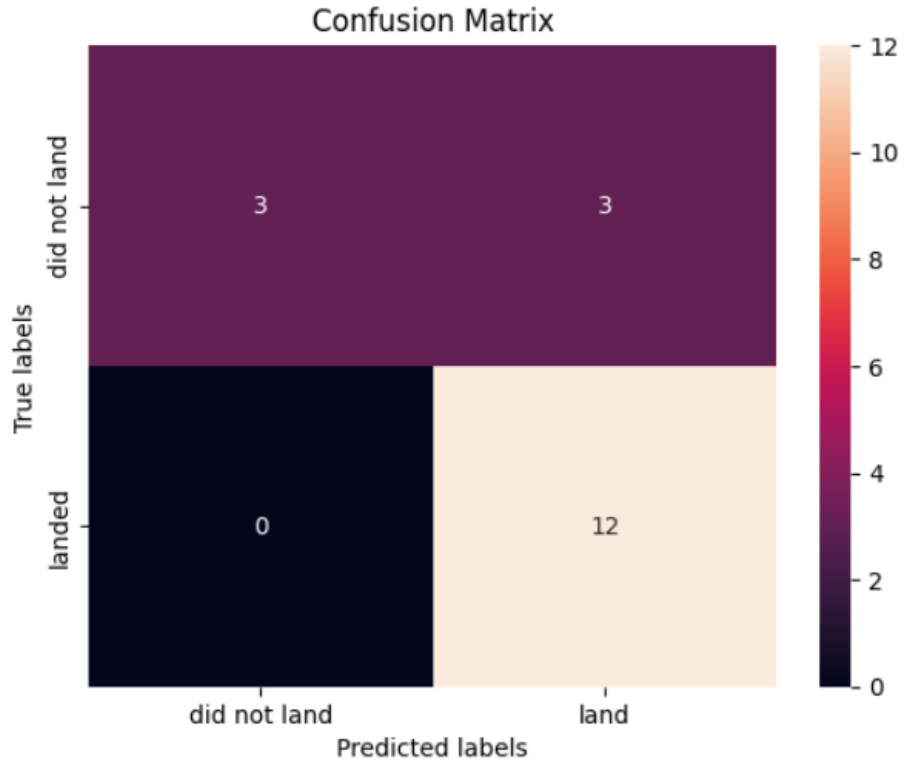
In [22]:
```
accuracy = logreg_cv.score(X_test, Y_test)

print("Accuracy on test data:", accuracy)
```

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

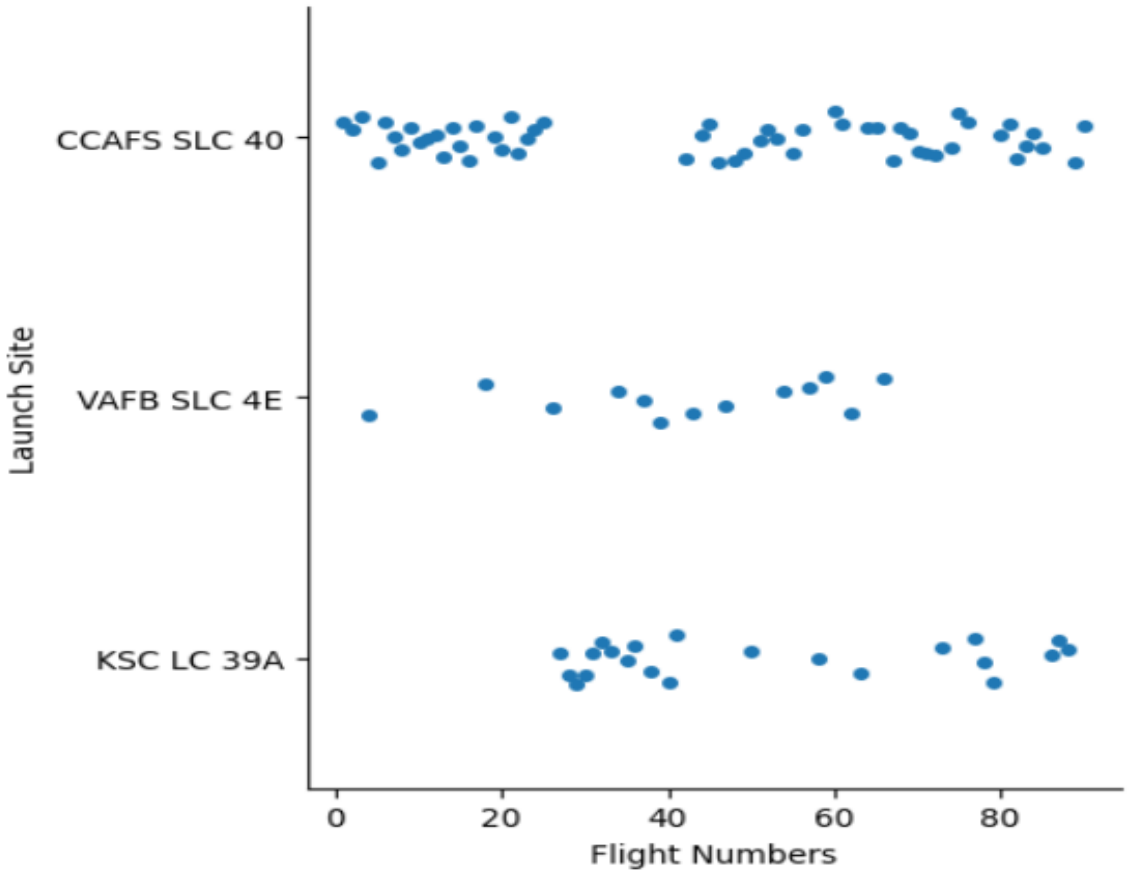
Lets look at the confusion matrix:

In [23]:
```
yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



In [9]:

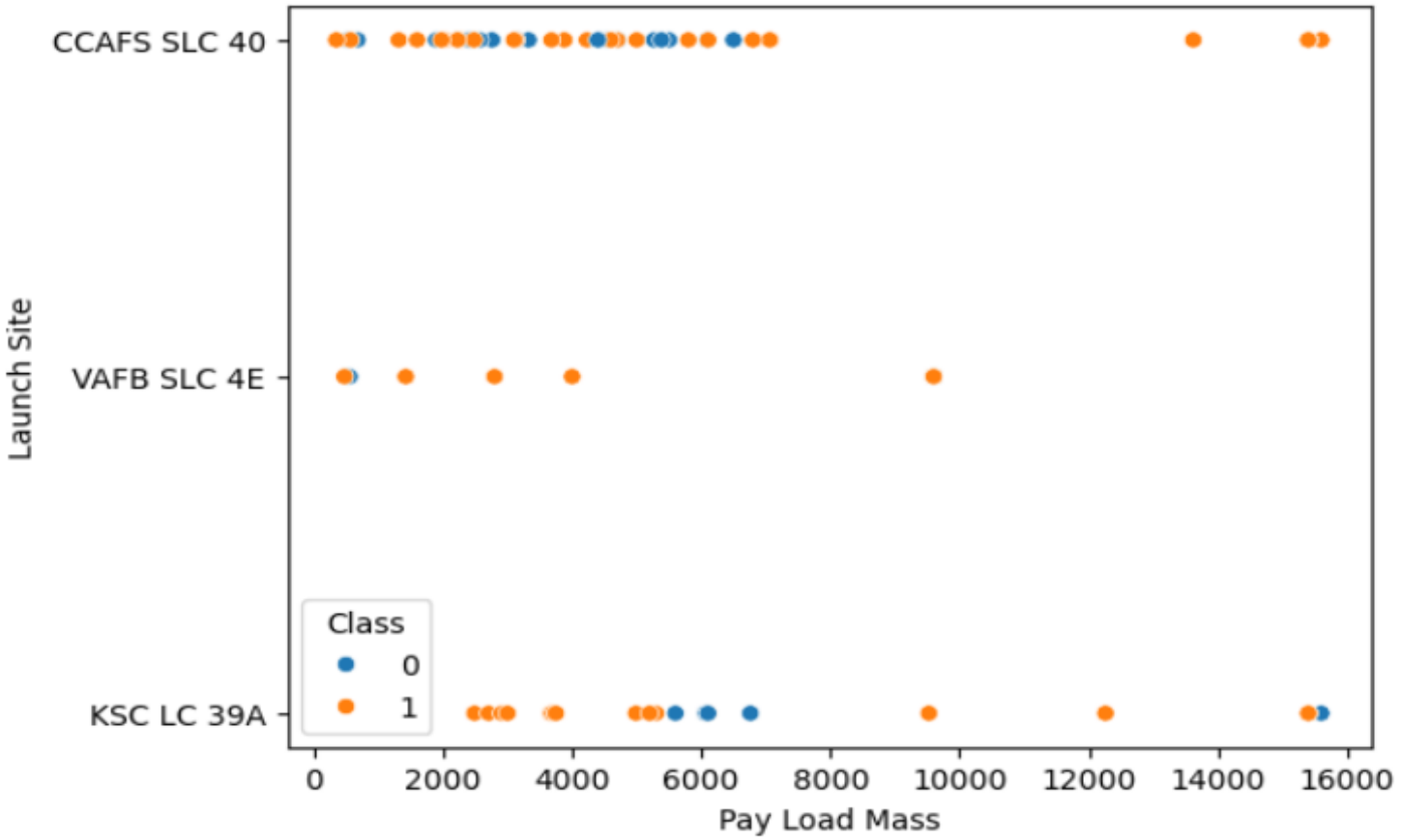
```
### TASK 1: Visualize the relationship between Flight Number and Launch Site
sns.catplot(data=df, y="LaunchSite", x="FlightNumber")
plt.xlabel("Flight Numbers")
plt.ylabel("Launch Site")
plt.show()
```



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

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

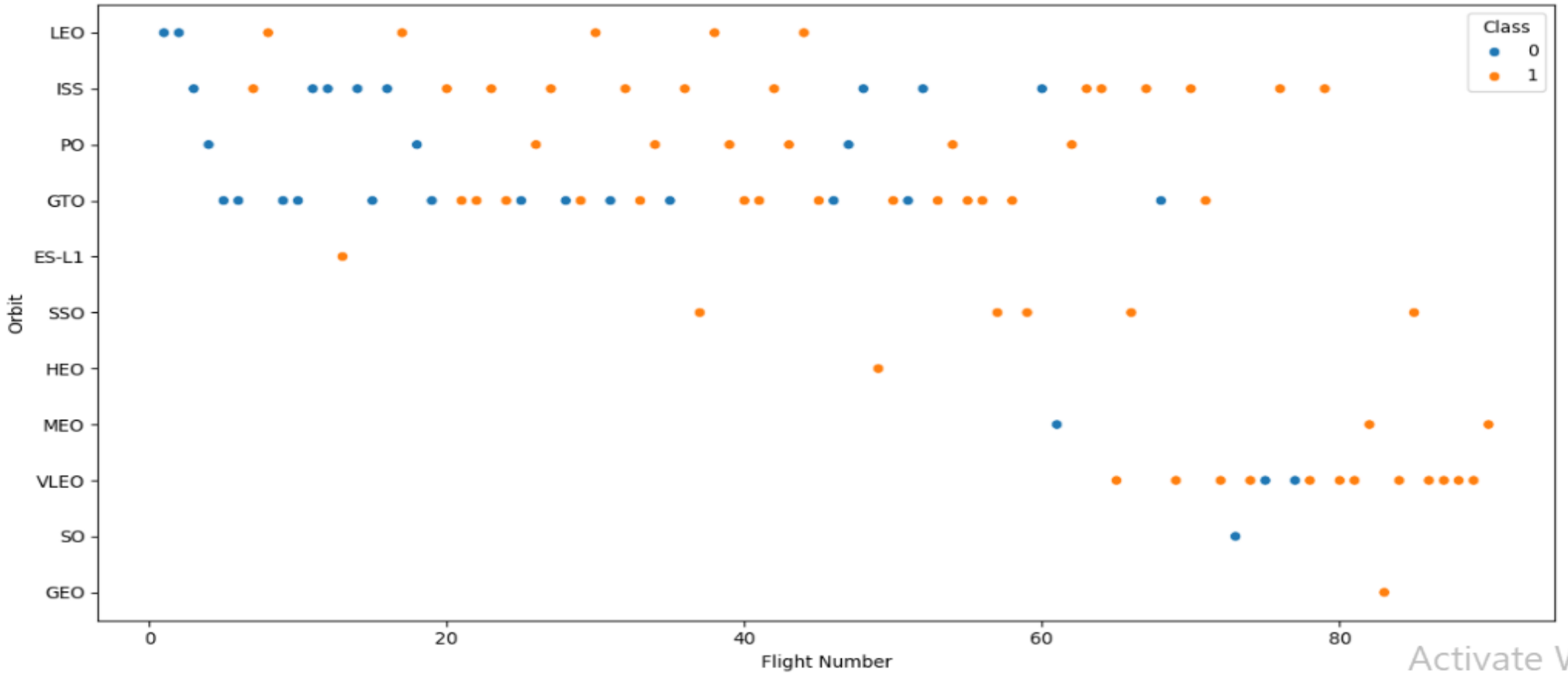
[13]:
```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch site, and hue to be the class
sns.scatterplot(data=df, x="PayloadMass", y="LaunchSite", hue="Class")
plt.xlabel("Pay Load Mass")
plt.ylabel("Launch Site")
plt.show()
```



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

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

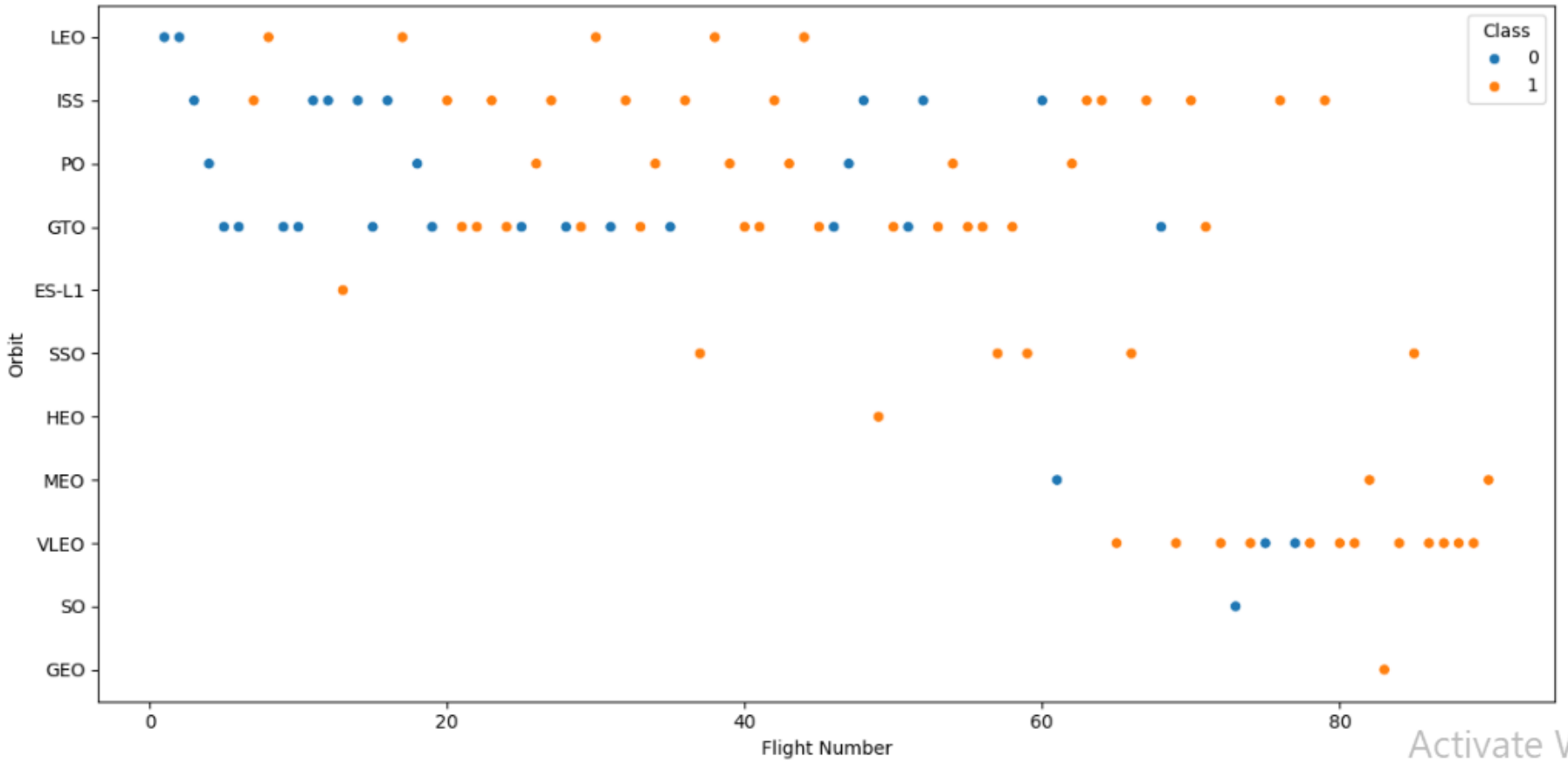
[22]:
```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class value
plt.figure(figsize=(12,6))
sns.scatterplot(data=df, x="FlightNumber", y="Orbit", hue="Class")
plt.xlabel("Flight Number")
plt.ylabel("Orbit")
plt.tight_layout()
plt.show()
```



You should see that in the LEO orbit the Success appears related to the number of flights; on the other hand, there seems to be no relationship between flight number when in GTO orbit.

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

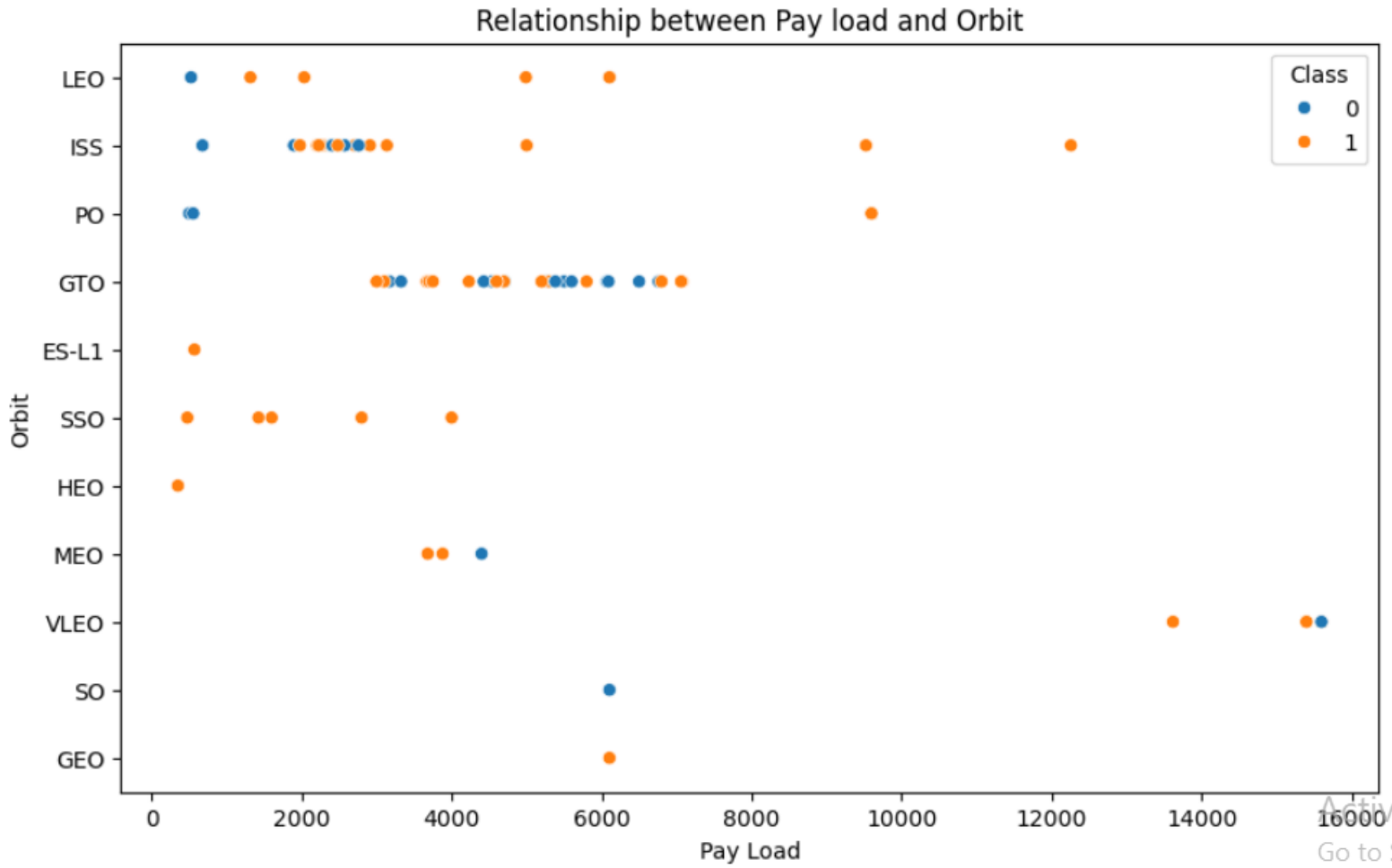
In [22]:
```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class value
plt.figure(figsize=(12,6))
sns.scatterplot(data=df, x="FlightNumber", y="Orbit", hue="Class")
plt.xlabel("Flight Number")
plt.ylabel("Orbit")
plt.tight_layout()
plt.show()
```



You should see that in the LEO orbit the Success appears related to the number of flights; on the other hand, there seems to be no relationship between flight number when in GTO orbit.

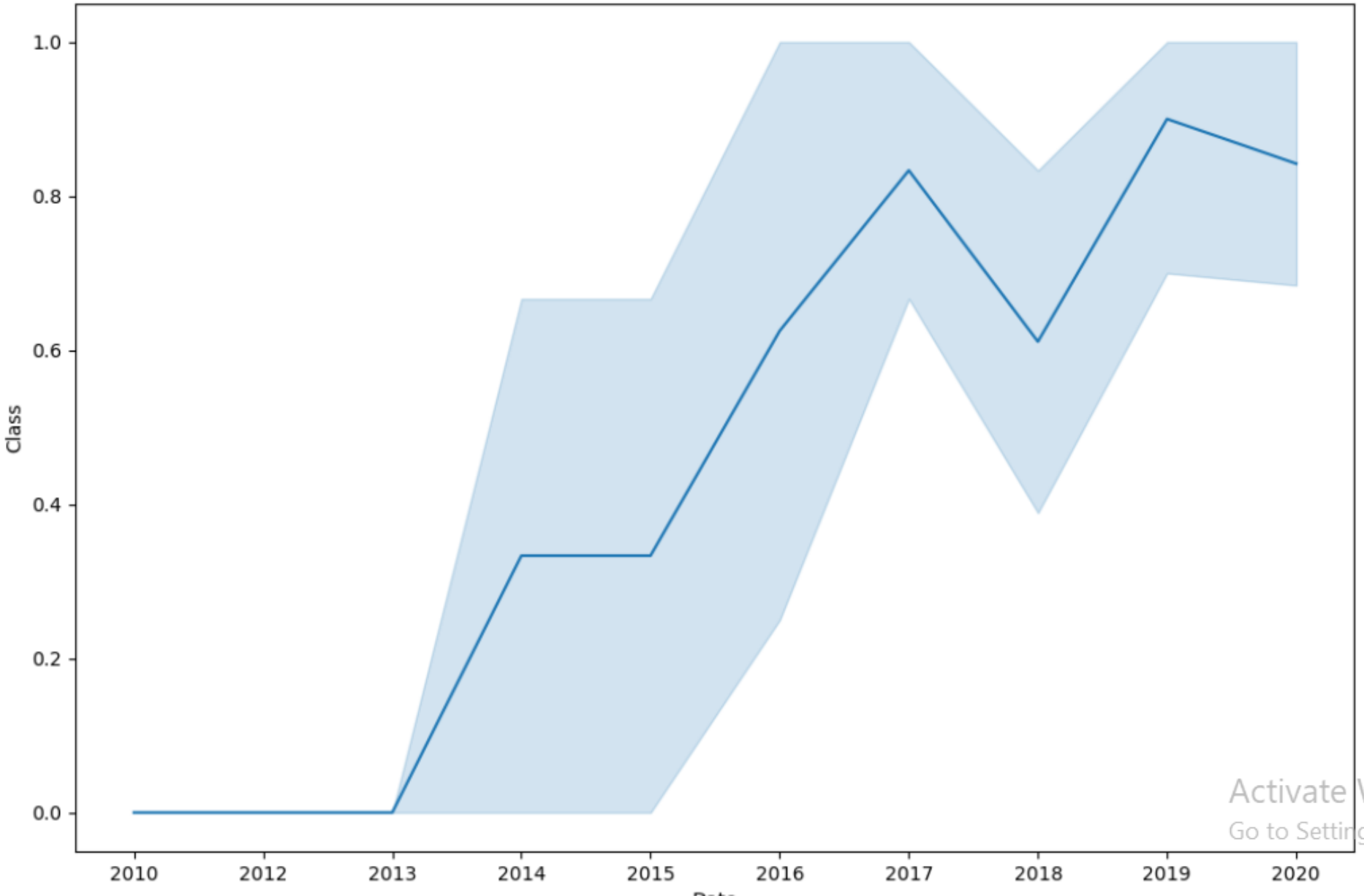
In [27]:

```
### TASK  5: Visualize the relationship between Payload and Orbit type
plt.figure(figsize=(10,6))
sns.scatterplot(data=df, x="PayloadMass", y="Orbit", hue="Class")
plt.title("Relationship between Pay load and Orbit")
plt.xlabel("Pay Load")
plt.ylabel("Orbit")
plt.show()
```



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

```
## Features Engineering
plt.figure(figsize=(12,8))
sns.lineplot(data=df, x="Date", y="Class")
plt.show()
```



UNIQUE LAUNCH SITES

Task 1

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

In [32]:
```
cursor.execute('SELECT DISTINCT Launch_Site FROM SPACEXTBL')
cursor.fetchall()
```

Out[32]:
```
[('CCAFS LC-40',), ('VAFB SLC-4E',), ('KSC LC-39A',), ('CCAFS SLC-40',)]
```

LAUNCH SITE BEGINS WITH "CCA"

Task 2

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

In [36]:
```
cursor.execute("SELECT * FROM SPACEXTBL WHERE Launch_Site LIKE 'CCA%' LIMIT 5")
cursor.fetchall()
```

Out[36]:
```
[('2010-06-04',
  '18:45:00',
  'F9 v1.0  B0003',
  'CCAFS LC-40',
  'Dragon Spacecraft Qualification Unit',
  0,
  'LEO',
  'SpaceX',
  'Success',
  'Failure (parachute)'),
 ('2010-12-08',
  '15:43:00',
  'F9 v1.0  B0004',
  'CCAFS LC-40',
  'Dragon demo flight C1, two CubeSats, barrel of Brouere cheese',
  0,
  'LEO (ISS)',
  'NASA (COTS) NRO',
  'Success',
  'Failure (parachute)'),
 ('2012-05-22',
  '7:44:00',
  'F9 v1.0  B0005',
  'CCAFS LC-40',
  'Dragon demo flight C2',
  525,
  'LEO (ISS)',
  'NASA (COTS)',
  'Success',
  'No attempt'),
 ('2012-10-08',
  '0:35:00',
  'F9 v1.0  B0006',
  'CCAFS LC-40',
  'SpaceX CRS-1',
  500,
  'LEO (ISS)',
  'NASA (CRS)',
  'Success',
  'No attempt'),
 ('2013-03-01',
  '15:10:00',
  'F9 v1.0  B0007',
  'CCAFS LC-40',
  'SpaceX CRS-2',
  677,
  'LEO (ISS)',
  'NASA (CRS)',
  'Success',
  'No attempt')]
```

TOTAL PYLOADS BY NASA

Task 3

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

In [47]:
```
cursor.execute("SELECT sum(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE Customer LIKE 'NASA (CRS)%'")
cursor.fetchall()
```

Out[47]:
```
[(48213,)]
```

AVERAGE PAYLOADS

Task 4

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

In [49]:
```
cursor.execute("SELECT avg(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE Booster_Version LIKE 'F9 v1.1%'")
cursor.fetchall()
```

Out[49]:
```
[(2534.6666666666665,)]
```

FIRST LANDINH DATE SUCCESSFULLY

Task 5

List the date when the first succesful landing outcome in ground pad was acheived.

Hint:Use min function

```
cursor.execute("SELECT MIN(date) AS first_successful_landing_date FROM SPACEXTBL WHERE landing_outcome = 'Success (ground pa
cursor.fetchall()
```

```
[('2015-12-22',)]
```

SUCCESSFUL DRONE SHIP LANDING WITH PAYLOAD BETWEEN 4000 AND 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

In [64]:
```
cursor.execute("SELECT DISTINCT Booster_Version FROM SPACEXTBL WHERE landing_outcome LIKE 'Success (drone ship)' AND PAYLOAD
cursor.fetchall()
```

Out[64]:
```
[('F9 FT B1022',), ('F9 FT B1026',), ('F9 FT  B1021.2',), ('F9 FT  B1031.2',)]
```

TOTAL NUMBER OF SUCCESSFUL AND FAILURE LANDING

Task 7

List the total number of successful and failure mission outcomes

In [68]:
```
cursor.execute("SELECT mission_outcome, COUNT(*) AS total_count FROM SPACEXTBL GROUP BY mission_outcome")
cursor.fetchall()
```

Out[68]:
```
[('Failure (in flight)', 1),
 ('Success', 98),
 ('Success ', 1),
 ('Success (payload status unclear)', 1)]
```

MAXIMUM PAYLOAD MASS

Task 8

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

In [70]:
```
cursor.execute("SELECT DISTINCT Booster_Version FROM SPACEXTBL WHERE PAYLOAD_MASS__KG_ = (SELECT max(PAYLOAD_MASS__KG_) FROM
cursor.fetchall()
```

Out[70]:
```
[('F9 B5 B1048.4',),
 ('F9 B5 B1049.4',),
 ('F9 B5 B1051.3',),
 ('F9 B5 B1056.4',),
 ('F9 B5 B1048.5',),
 ('F9 B5 B1051.4',),
 ('F9 B5 B1049.5',),
 ('F9 B5 B1060.2 ',),
 ('F9 B5 B1058.3 ',),
 ('F9 B5 B1051.6',),
 ('F9 B5 B1060.3',),
 ('F9 B5 B1049.7 ',)]
```

LAUNCH RECORDS

```
('F9 B5 B1049.7 ',)]
```

Task 9

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

Note: SQLLite does not support monthnames. So you need to use substr(Date, 6,2) as month to get the months and substr(Date,0,5)='2015' for year.

In [73]:
```
cursor.execute("SELECT DISTINCT Booster_Version, Launch_Site FROM SPACEXTBL WHERE landing_outcome LIKE 'Failure (drone ship)
cursor.fetchall()
```

Out[73]:
```
[]
```

RANK 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.

In [ ]:

```
cursor.execute("SELECT landing_outcome, COUNT(*) AS outcome_count FROM SPACEXTBL WHERE Date BETWEEN '2010-06-04' AND '2017-0
GROUP BY landing_outcome
ORDER BY outcome_count DESC;
```

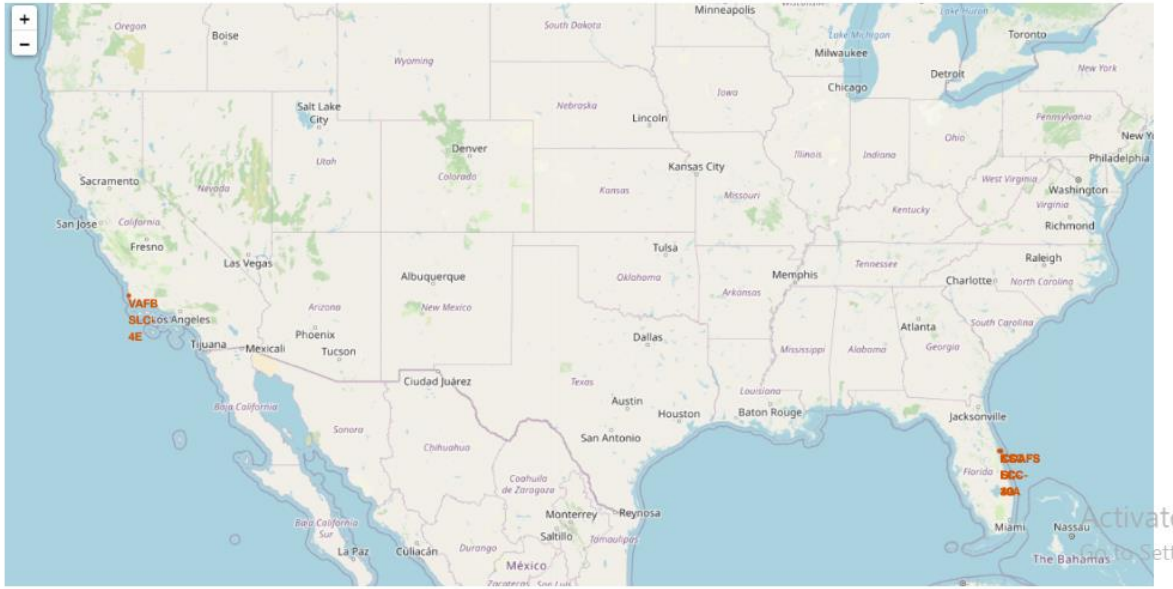
INITIAL MAP

```
12; color:#d35400;"><b>%s</b></div>' % 'label', ))
```

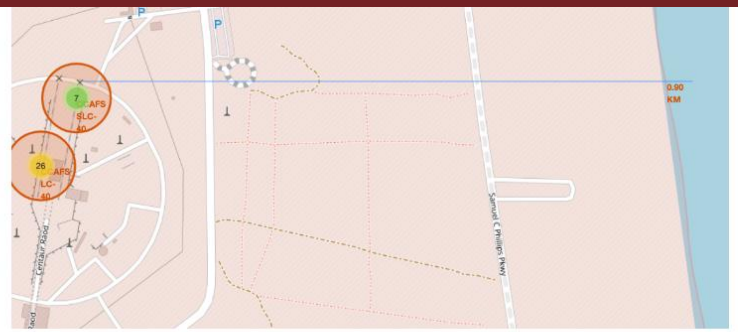
In [9]:

```
# Initial the map
site_map = folium.Map(location=nasa_coordinate, zoom_start=5)
# For each launch site, add a Circle object based on its coordinate (Lat, Long) values. In addition, add Launch site name
```

The generated map with marked launch sites should look similar to the following:



THIRD MAP



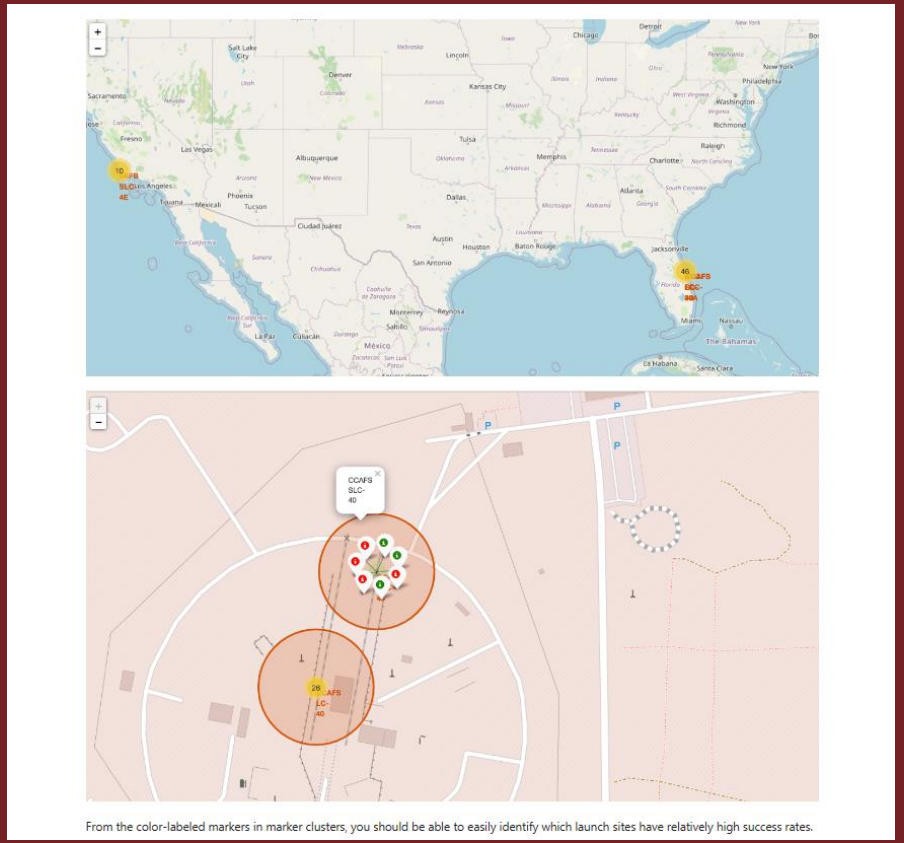
TODO: Similarly, you can draw a line betwee a launch site to its closest city, railway, highway, etc. You need to use `MousePosition` to find the their coordinates on the map first

A railway map symbol may look like this:

A highway map symbol may look like this:

A city map symbol may look like this:

SECOND MAP


CCAFS SLC-40
AFS LC-40
From the color-labeled markers in marker clusters, you should be able to easily identify which launch sites have relatively high success rates.

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`.

In [19]:
```
logreg = LogisticRegression()
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
logreg_cv = GridSearchCV(logreg, parameters, cv=10)

logreg_cv.fit(X_train, Y_train)

print("Best parameters:", logreg_cv.best_params_)
```

```
Best parameters: {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
```

In [20]:
```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()
```

We output the `GridSearchCV` object for logistic regression. We display the best parameters using the data attribute `best_params_` and the accuracy on the validation data using the data attribute `best_score_`.

In [21]:
```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8464285714285713
```

MATRIX OF LOGISTIC REGRESSION

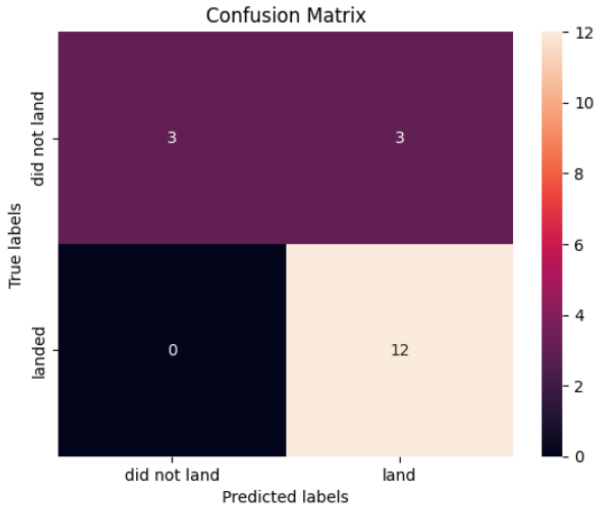
In [22]:
```
accuracy = logreg_cv.score(X_test, Y_test)

print("Accuracy on test data:", accuracy)
```

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

Lets look at the confusion matrix:

In [23]:
```
yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



Examining the confusion matrix, we see that logistic regression can distinguish between the different classes. We see that the major problem is false positives.

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`.

In [25]:
```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

In [29]:
```
svm_cv=GridSearchCV(svm, parameters, cv=10)
svm_cv.fit(X_train, Y_train)
print("Best parameters:", svm_cv.best_params_)
```

```
Best parameters: {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
```

In [30]:
```
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
```

MATRIX OF SVM

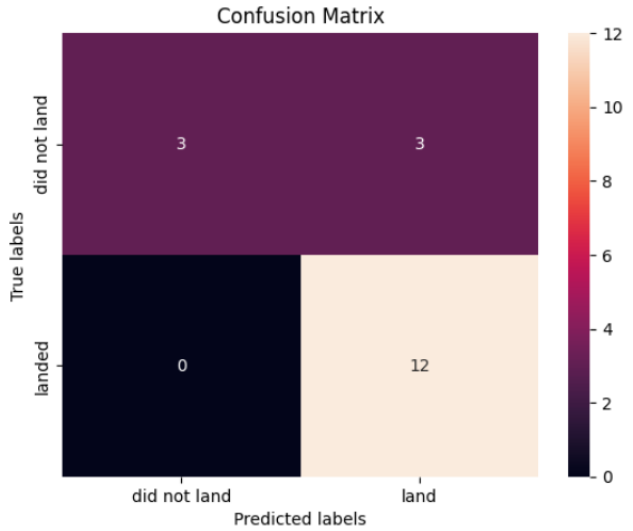
In [31]:
```
accuracy = svm_cv.score(X_test, Y_test)

print("Accuracy on test data:", accuracy)
```

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

We can plot the confusion matrix

In [32]:
```
yhat=svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



DECISION TREE CLASSIFIER

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`.

In [38]:

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

tree = DecisionTreeClassifier()

tree_cv=GridSearchCV(tree, parameters, cv=10)
tree_cv.fit(X_train, Y_train)
print("Best parameters:" , tree_cv.best_params_)
```

```
/lib/python3.11/site-packages/sklearn/model_selection/_validation.py:425: FitFailedWarning:
3240 fits failed out of a total of 6480.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
3240 fits failed with the following error:
Traceback (most recent call last):
  File "/lib/python3.11/site-packages/sklearn/model_selection/_validation.py", line 729, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/lib/python3.11/site-packages/sklearn/base.py", line 1145, in wrapper
    estimator._validate_params()
  File "/lib/python3.11/site-packages/sklearn/base.py", line 638, in _validate_params
    validate_parameter_constraints(
  File "/lib/python3.11/site-packages/sklearn/utils/_param_validation.py", line 95, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' parameter of DecisionTreeClassifier must be an int
in the range [1, inf), a float in the range (0.0, 1.0], a str among {'log2', 'sqrt'} or None. Got 'auto' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/lib/python3.11/site-packages/sklearn/model_selection/_search.py:979: UserWarning: One or more of the test scores are non-fin
ite: [        nan         nan         nan         nan         nan         nan
```

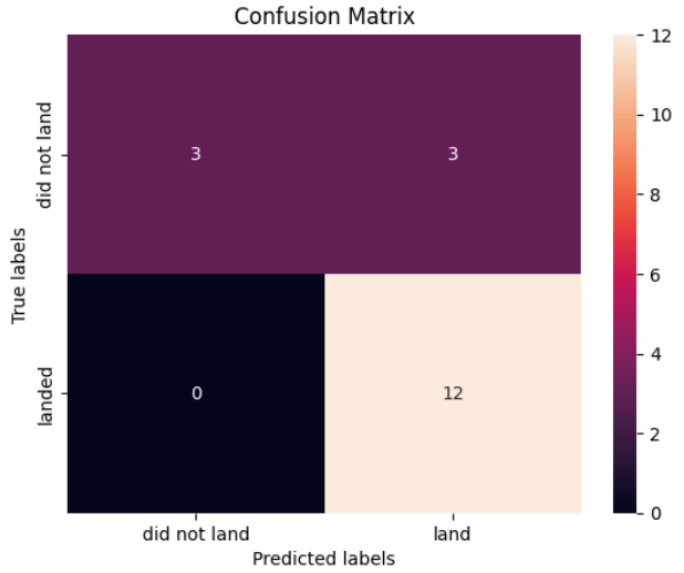
MATRIX OF DECISION TREE

In [40]:
```
accuracy= tree_cv.score(X_test, Y_test)
print("Accuracy on test data:", accuracy)
```

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

We can plot the confusion matrix

In [41]:
```
yhat = tree_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



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`.

In [42]:
```
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()
```

In [42]:
```
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()
```

In [44]:
```
knn_cv=GridSearchCV(KNN, parameters, cv=10)
knn_cv.fit(X_train, Y_train)
print("Best parameters:", knn_cv.best_params_)
```

```
/lib/python3.11/site-packages/threadpoolctl.py:1019: RuntimeWarning: libc not found. The ctypes module in Python 3.11 is maybe too old for this OS.
  warnings.warn(
Best parameters: {'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}
```

In [45]:
```
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
```

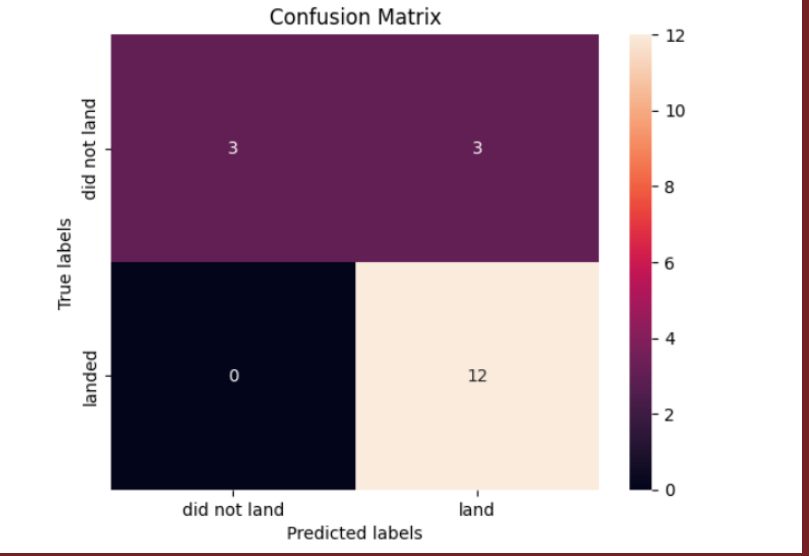
MATRIX OF KNN

[46]:
```
accuracy=knn_cv.score(X_test, Y_test)
print("Accuracy on test data:", accuracy)
```

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

We can plot the confusion matrix

[47]:
```
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



THE BEST RESUL IS 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 pa
from the dictionary `parameters`.

In [19]:
```
logreg = LogisticRegression()
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
logreg_cv = GridSearchCV(logreg, parameters, cv=10)

logreg_cv.fit(X_train, Y_train)

print("Best parameters:", logreg_cv.best_params_)
```

```
Best parameters: {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
```

In [20]:
```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()
```

We output the `GridSearchCV` object for logistic regression. We display the best parameters using the data attribute `best_pa`
the accuracy on the validation data using the data attribute `best_score_`.

In [21]:
```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8464285714285713
```

TASK 5

MATRIX

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

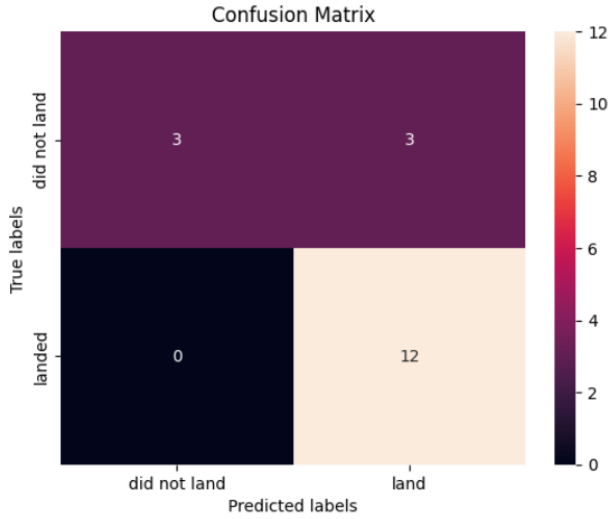
In [22]:
```
accuracy = logreg_cv.score(X_test, Y_test)

print("Accuracy on test data:", accuracy)
```

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

Lets look at the confusion matrix:

In [23]:
```
yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



IN THIS PROJECT I TRIED TO SHOW DIFFERENT VISUALIZATIONS, BAR CHART, TABLE, GRPHS AND MAPS FOR UNDERSTANDING EASILY. IN THE PREDICTION PROCESS I USED DECISION TREE, KNN, LOGISTIC_REGRESSION, SVM MODELS AND DID PREDICTION WITH THE HJELP OF THESE MODELS. THE BEST MODEL WAS LOGISTIC REGRESSION AND IT DISPLAYS 0.84.

THANK YOU FOR YOUR ATTENTION