Skills Network

Space X Falcon 9 First Stage Landing Prediction

Assignment: Machine Learning Prediction

Estimated time needed: **60** minutes

Space X advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is because Space X can reuse the first stage. Therefore if we can determine if the first stage will land, we can determine the cost of a launch. This information can be used if an alternate company wants to bid against space X for a rocket launch. In this lab, you will create a machine learning pipeline to predict if the first stage will land given the data from the preceding labs.

Several examples of an unsuccessful landing are shown here:

Most unsuccessful landings are planed. Space X; performs a controlled landing in the oceans.

Objectives

Perform exploratory Data Analysis and determine Training Labels

- create a column for the class
- Standardize the data
- Split into training data and test data

-Find best Hyperparameter for SVM, Classification Trees and Logistic Regression

- Find the method performs best using test data

Import Libraries and Define Auxiliary Functions

In [1]:
```
import piplite
await piplite.install(['numpy'])
await piplite.install(['pandas'])
await piplite.install(['seaborn'])
```

We will import the following libraries for the lab

In [2]:
```
# Pandas is a software library written for the Python programming language for d
import pandas as pd
# NumPy is a library for the Python programming language, adding support for lar
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a MatLab like
import matplotlib.pyplot as plt
#Seaborn is a Python data visualization library based on matplotlib. It provides
import seaborn as sns
# Preprocessing allows us to standarsize our data
from sklearn import preprocessing
# Allows us to split our data into training and testing data
from sklearn.model_selection import train_test_split
# Allows us to test parameters of classification algorithms and find the best on
from sklearn.model_selection import GridSearchCV
# Logistic Regression classification algorithm
```

```
from sklearn.linear_model import LogisticRegression
# Support Vector Machine classification algorithm
from sklearn.svm import SVC
# Decision Tree classification algorithm
from sklearn.tree import DecisionTreeClassifier
# K Nearest Neighbors classification algorithm
from sklearn.neighbors import KNeighborsClassifier
```

This function is to plot the confusion matrix.

In [3]:
```
def plot_confusion_matrix(y,y_predict):
    "this function plots the confusion matrix"
    from sklearn.metrics import confusion_matrix

    cm = confusion_matrix(y, y_predict)
    ax= plt.subplot()
    sns.heatmap(cm, annot=True, ax = ax); #annot=True to annotate cells
    ax.set_xlabel('Predicted labels')
    ax.set_ylabel('True labels')
    ax.set_title('Confusion Matrix');
    ax.xaxis.set_ticklabels(['did not land', 'land']); ax.yaxis.set_ticklabels([
    plt.show()
```

Load the dataframe

Load the data

In [4]:
```
from js import fetch
import io

URL1 = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-D
resp1 = await fetch(URL1)
text1 = io.BytesIO((await resp1.arrayBuffer()).to_py())
data = pd.read_csv(text1)
```

In [5]:
```
data.head()
```

Out[5]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Fli
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	

In [6]:
```
URL2 = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-D
resp2 = await fetch(URL2)
```

```
text2 = io.BytesIO((await resp2.arrayBuffer()).to_py())
X = pd.read_csv(text2)
```

In [7]:
```
X.head(100)
```

Out[7]:

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	O
0	1.0	6104.959412	1.0	1.0	0.0	0.0	0.0	
1	2.0	525.000000	1.0	1.0	0.0	0.0	0.0	
2	3.0	677.000000	1.0	1.0	0.0	0.0	0.0	
3	4.0	500.000000	1.0	1.0	0.0	0.0	0.0	
4	5.0	3170.000000	1.0	1.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	
85	86.0	15400.000000	2.0	5.0	2.0	0.0	0.0	
86	87.0	15400.000000	3.0	5.0	2.0	0.0	0.0	
87	88.0	15400.000000	6.0	5.0	5.0	0.0	0.0	
88	89.0	15400.000000	3.0	5.0	2.0	0.0	0.0	
89	90.0	3681.000000	1.0	5.0	0.0	0.0	0.0	

90 rows × 83 columns

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']).

In [8]:
```
Y = data['Class'].to_numpy()
```

TASK 2

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

In [10]:
```
# students get this
transform = preprocessing.StandardScaler()
X = transform.fit(X).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`

In [11]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_
```

we can see we only have 18 test samples.

In [12]:
```
Y_test.shape
```

Out[12]:
```
(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`.

In [13]:
```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

In [16]:
```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2
lr=LogisticRegression()
logreg_cv = GridSearchCV(estimator=lr, cv=10, param_grid=parameters).fit(X_train
```

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 [17]:
```
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

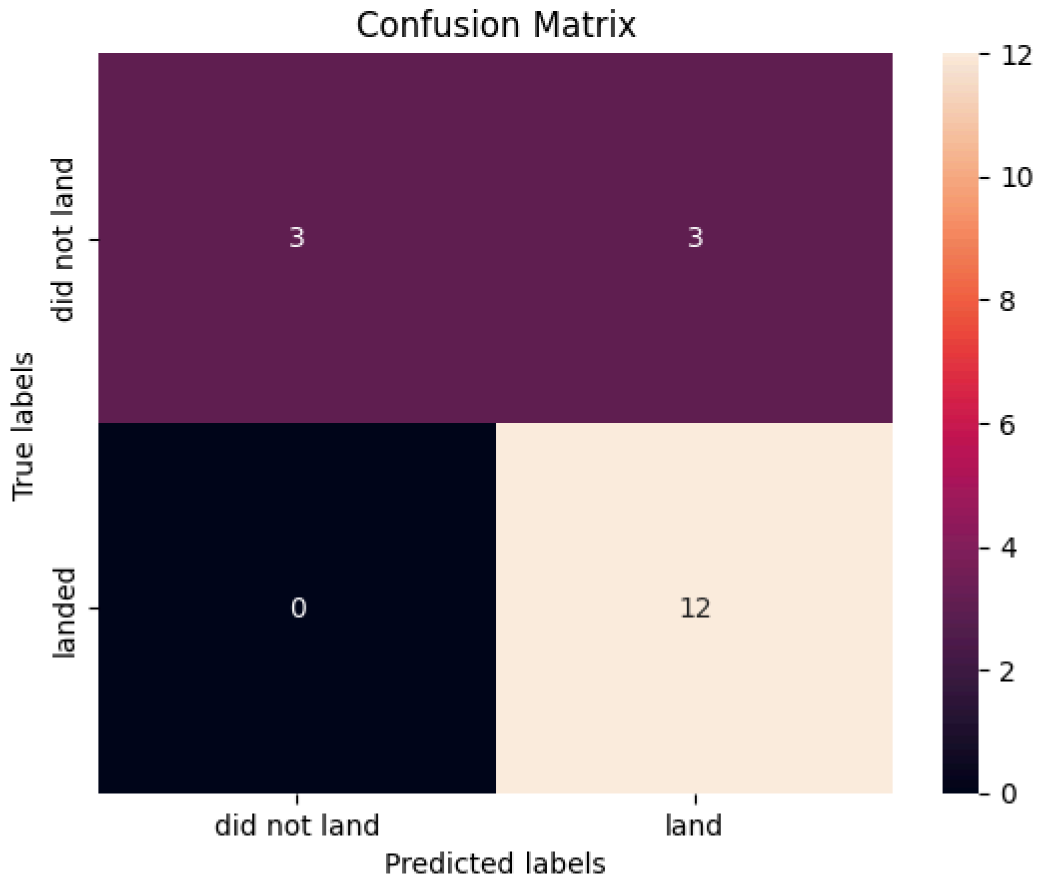
Calculate the accuracy on the test data using the method `score`:

In [18]:
```
logreg_score = logreg_cv.score(X_test, Y_test)
print("score :", logreg_score)
```

```
score : 0.8333333333333334
```

Lets look at the confusion matrix:

In [19]:
```python
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.

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 [20]:
```python
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
svm_cv = GridSearchCV(estimator=svm, cv=10, param_grid=parameters).fit(X_train,
```

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

```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.0316227766016837
9, 'kernel': 'sigmoid'}
accuracy : 0.8482142857142856
```

In [22]:
```python
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
```

TASK 7

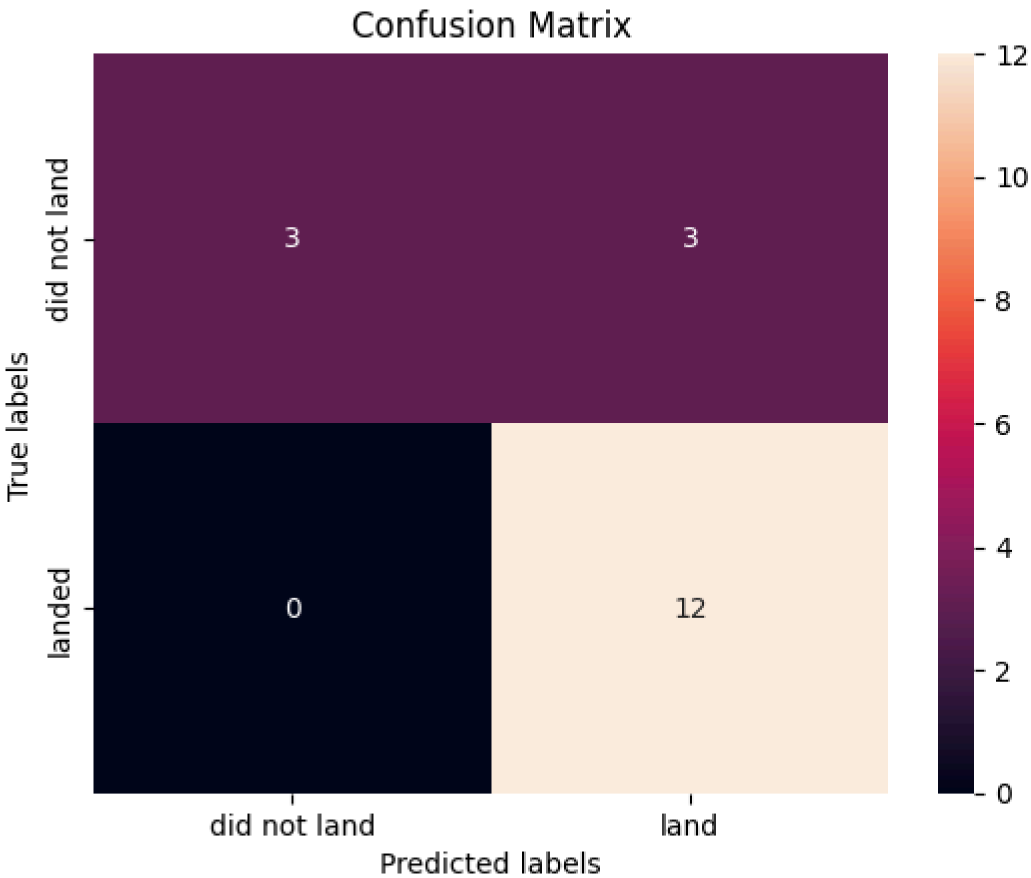
Calculate the accuracy on the test data using the method `score` :

In [23]:
```
svm_cv_score = svm_cv.score(X_test, Y_test)
print("score :",svm_cv_score)
```

```
score : 0.8333333333333334
```

We can plot the confusion matrix

In [24]:
```
yhat=svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



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`.

In [25]:
```
parameters = {'criterion': ['gini', 'entropy'],
     'splitter': ['best', 'random'],
     'max_depth': [2*n for n in range(1,10)],
     'max_features': ['auto', 'sqrt'],
```

```
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 5, 10]}

tree = DecisionTreeClassifier()
tree_cv = GridSearchCV(estimator=tree, cv=10, param_grid=parameters).fit(X_train
```

```
/lib/python3.11/site-packages/sklearn/model_selection/_validation.py:425: FitFail
edWarning:
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_sc
ore='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", li
ne 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_pa
rams
    validate_parameter_constraints(
  File "/lib/python3.11/site-packages/sklearn/utils/_param_validation.py", line 9
5, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' paramet
er 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: UserWarnin
g: One or more of the test scores are non-finite: [       nan        nan        n
an        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.72321429 0.79107143 0.74642857 0.7625     0.77321429 0.79285714
 0.78928571 0.6625     0.78214286 0.81785714 0.80535714 0.77678571
 0.78214286 0.76071429 0.76607143 0.7625     0.77678571 0.66607143
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.77857143 0.84642857 0.74642857 0.76428571 0.87678571 0.75
 0.76071429 0.87321429 0.83392857 0.73392857 0.83392857 0.79107143
 0.84642857 0.83214286 0.775      0.81964286 0.80357143 0.81071429
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.79107143 0.77857143 0.83392857 0.78035714 0.7625     0.77857143
 0.79107143 0.83392857 0.74642857 0.775      0.75178571 0.80535714
 0.77678571 0.83392857 0.775      0.775      0.79107143 0.74821429
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.81964286 0.76428571 0.81964286 0.77857143 0.77857143 0.80535714
 0.7625     0.77678571 0.74464286 0.7625     0.80357143 0.83392857
 0.76071429 0.80357143 0.78928571 0.78928571 0.77678571 0.81785714
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.71964286 0.81964286 0.80535714 0.76428571 0.82142857 0.78928571
 0.78928571 0.80357143 0.73571429 0.81964286 0.83214286 0.83392857
 0.84821429 0.79107143 0.79107143 0.7625     0.7625     0.81964286
        nan        nan        nan        nan        nan        nan
```

```
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.76071429 0.73928571 0.79107143 0.79107143 0.73571429 0.74821429
0.78928571 0.89285714 0.74821429 0.7625     0.81785714 0.76428571
0.71071429 0.83214286 0.78928571 0.80535714 0.73571429 0.83214286
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.79107143 0.74821429 0.77857143 0.7375     0.775      0.7375
0.73214286 0.79107143 0.80178571 0.77857143 0.79107143 0.81964286
0.73928571 0.77678571 0.81785714 0.83214286 0.76071429 0.83392857
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.83392857 0.79107143 0.82142857 0.81785714 0.80892857 0.80357143
0.73214286 0.81785714 0.78928571 0.79107143 0.73214286 0.80535714
0.7625     0.81428571 0.79107143 0.79107143 0.80535714 0.83214286
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.79285714 0.81964286 0.69107143 0.75178571 0.75       0.81785714
0.83214286 0.725      0.76428571 0.75       0.70892857 0.84821429
0.69464286 0.76785714 0.78928571 0.81964286 0.775      0.78928571
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.71785714 0.76607143 0.66785714 0.79464286 0.76071429 0.72142857
0.81964286 0.79107143 0.72142857 0.80535714 0.79107143 0.77678571
0.81964286 0.82321429 0.81964286 0.7375     0.80178571 0.78214286
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.83214286 0.84642857 0.78928571 0.81964286 0.77678571 0.875
0.77678571 0.77857143 0.83214286 0.81964286 0.79107143 0.80535714
0.775      0.76785714 0.81964286 0.83214286 0.775      0.725
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.81964286 0.8        0.81964286 0.83214286 0.80535714 0.78928571
0.81785714 0.76428571 0.775      0.79107143 0.73214286 0.84642857
0.7875     0.83214286 0.77321429 0.77678571 0.74821429 0.75
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.7375     0.83392857 0.7625     0.71964286 0.81964286 0.81964286
0.75       0.83392857 0.79107143 0.775      0.73571429 0.78928571
0.77321429 0.71071429 0.80535714 0.76071429 0.81964286 0.81964286
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.79107143 0.80357143 0.81964286 0.725      0.76071429 0.79464286
0.70714286 0.81785714 0.75       0.81964286 0.80535714 0.83392857
0.7625     0.75535714 0.81785714 0.80535714 0.71964286 0.82142857
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.77678571 0.72142857 0.68035714 0.80535714 0.79107143 0.80714286
0.7625     0.74821429 0.77857143 0.80357143 0.81964286 0.83392857
0.7875     0.74821429 0.78928571 0.80178571 0.77857143 0.79285714
       nan        nan        nan        nan        nan        nan
```

```
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.78928571 0.78214286 0.73571429 0.71964286 0.80357143 0.79285714
0.73392857 0.83214286 0.73392857 0.78928571 0.77857143 0.81071429
0.78928571 0.81607143 0.68214286 0.83214286 0.78928571 0.81964286
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.80178571 0.7625     0.70714286 0.73571429 0.72321429 0.81785714
0.75178571 0.75178571 0.82857143 0.78928571 0.775      0.77678571
0.77678571 0.7375     0.90178571 0.84464286 0.71071429 0.80892857
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.775      0.7625     0.74821429 0.81964286 0.7625     0.80357143
0.71785714 0.75       0.75       0.75       0.7625     0.77857143
0.71071429 0.84821429 0.72142857 0.75357143 0.77857143 0.73571429]
  warnings.warn(
```

In [ ]:

In [26]:
```
print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', 'max_depth': 1
6, 'max_features': 'sqrt', 'min_samples_leaf': 4, 'min_samples_split': 5, 'splitt
er': 'best'}
accuracy : 0.9017857142857144
```

TASK 9

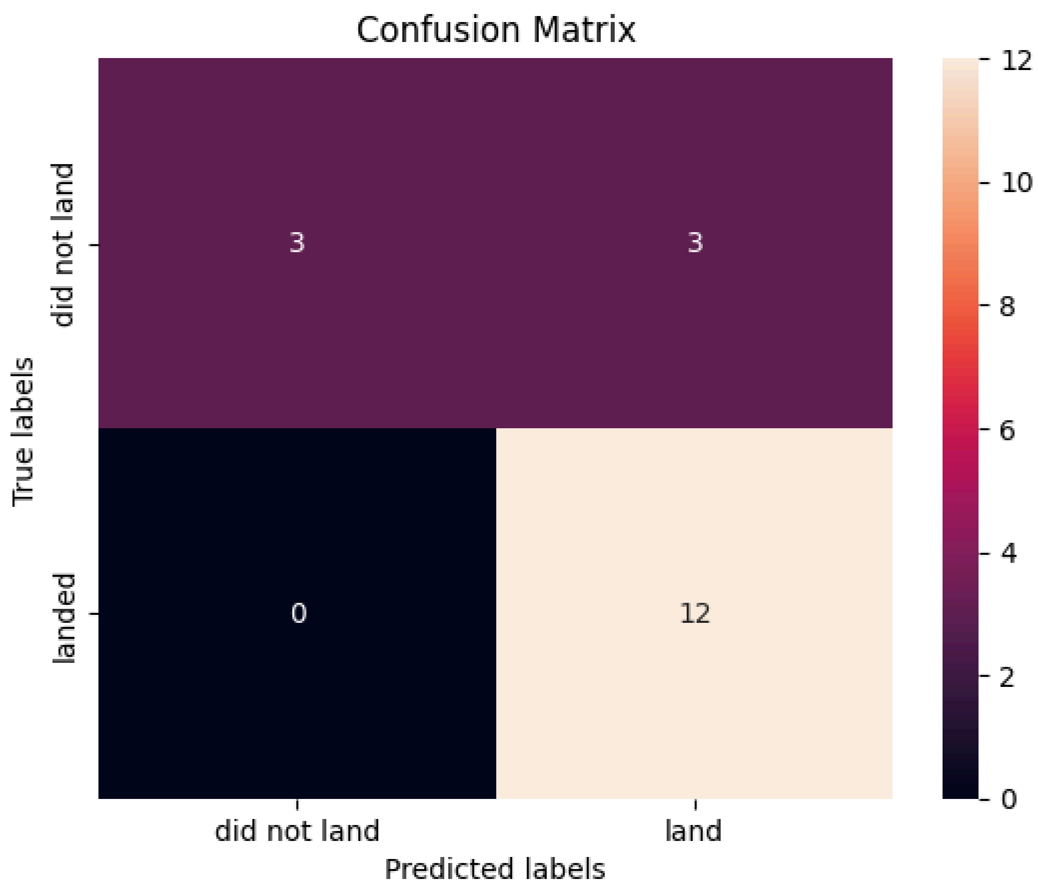
Calculate the accuracy of tree_cv on the test data using the method `score` :

In [27]:
```
tree_cv_score = svm_cv.score(X_test, Y_test)
print("score :",tree_cv_score)
```

```
score : 0.8333333333333334
```

We can plot the confusion matrix

In [28]:
```
yhat = tree_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



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 [29]:
```
parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'p': [1,2]}

KNN = KNeighborsClassifier()
knn_cv = GridSearchCV(estimator=KNN, cv=10, param_grid=parameters).fit(X_train,
```

```
/lib/python3.11/site-packages/threadpoolctl.py:1019: RuntimeWarning: libc not fou
nd. The ctypes module in Python 3.11 is maybe too old for this OS.
  warnings.warn(
```

In [ ]:

In [30]:
```
print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
print("accuracy :",knn_cv.best_score_)
```

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

TASK 11

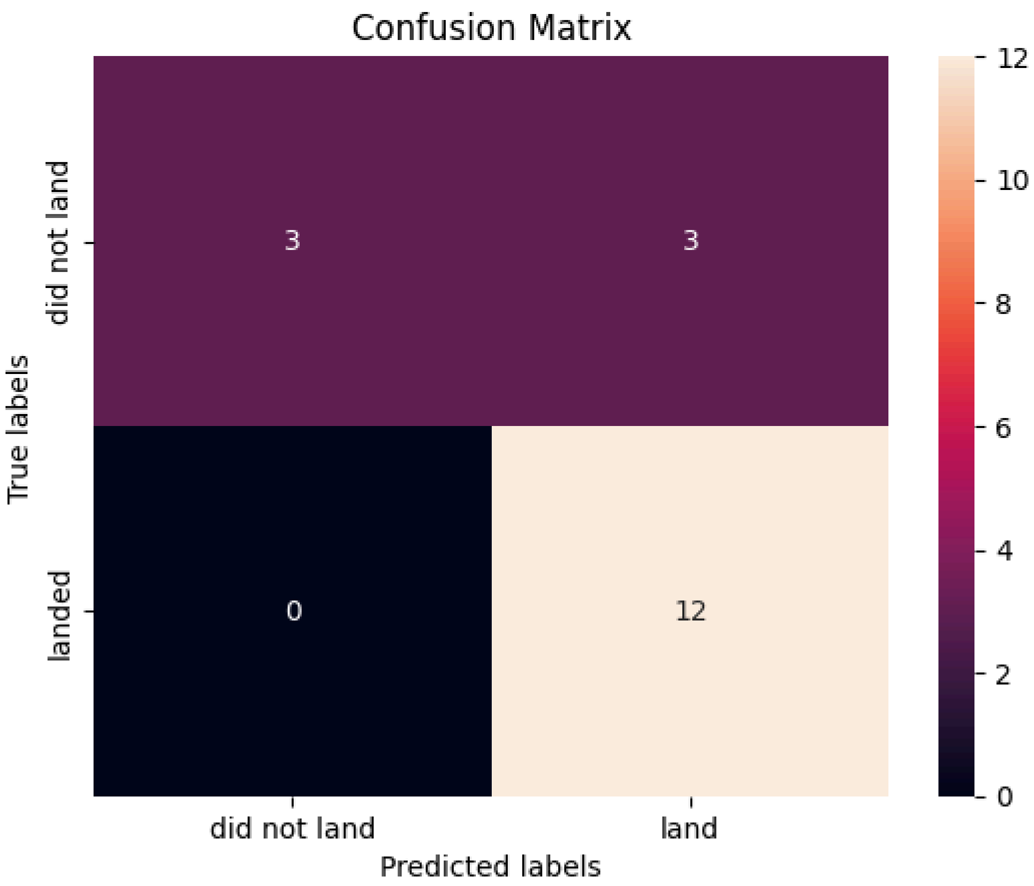
Calculate the accuracy of knn_cv on the test data using the method `score`:

In [31]:
```
knn_cv_score = knn_cv.score(X_test, Y_test)
print("score :",knn_cv_score)
```

```
score : 0.8333333333333334
```

We can plot the confusion matrix

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



TASK 12

Find the method performs best:

In [33]:
```
accuracy = [svm_cv_score, logreg_score, knn_cv_score, tree_cv_score]
accuracy = [i * 100 for i in accuracy]

method = ['Support Vector Machine', 'Logistic Regression', 'K Nearest Neighbour'
models = {'ML Method':method, 'Accuracy Score (%)':accuracy}

ML_df = pd.DataFrame(models)
ML_df
```

Out[33]:

	ML Method	Accuracy Score (%)
0	Support Vector Machine	83.333333
1	Logistic Regression	83.333333
2	K Nearest Neighbour	83.333333
3	Decision Tree	83.333333

In [36]:

```
import pandas as pd
import numpy as np
from sklearn.metrics import jaccard_score, f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

# Example: Assume X_train, X_test, Y_train, Y_test are defined from your data sp

# Initialize classifiers
logreg = LogisticRegression()
svm = SVC()
tree = DecisionTreeClassifier()
knn = KNeighborsClassifier()

# Train classifiers
logreg.fit(X_train, Y_train)
svm.fit(X_train, Y_train)
tree.fit(X_train, Y_train)
knn.fit(X_train, Y_train)

# Make predictions on test set
logreg_yhat = logreg.predict(X_test)
svm_yhat = svm.predict(X_test)
tree_yhat = tree.predict(X_test)
knn_yhat = knn.predict(X_test)

# Calculate scores
jaccard_scores = [
    jaccard_score(Y_test, logreg_yhat, average='binary'),
    jaccard_score(Y_test, svm_yhat, average='binary'),
    jaccard_score(Y_test, tree_yhat, average='binary'),
    jaccard_score(Y_test, knn_yhat, average='binary'),
]

f1_scores = [
    f1_score(Y_test, logreg_yhat, average='binary'),
    f1_score(Y_test, svm_yhat, average='binary'),
    f1_score(Y_test, tree_yhat, average='binary'),
    f1_score(Y_test, knn_yhat, average='binary'),
]

# Accuracy scores (if already calculated)
accuracy = [logreg.score(X_test, Y_test), svm.score(X_test, Y_test), tree.score(

# Creating DataFrame
scores_test = pd.DataFrame(
```

```
    np.array([jaccard_scores, f1_scores, accuracy]),
    index=['Jaccard_Score', 'F1_Score', 'Accuracy'],
    columns=['LogReg', 'SVM', 'Tree', 'KNN']
)

# Transpose the DataFrame for a better view
scores_test = scores_test.transpose()

print(scores_test)
```

```
        Jaccard_Score  F1_Score  Accuracy
LogReg       0.800000  0.888889  0.833333
SVM          0.733333  0.846154  0.777778
Tree         0.846154  0.916667  0.888889
KNN          0.750000  0.857143  0.777778
```

In [37]:
```
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 model is DecisionTree with a score of 0.9017857142857144
Best params is : {'criterion': 'entropy', 'max_depth': 16, 'max_features': 'sqr
t', 'min_samples_leaf': 4, 'min_samples_split': 5, 'splitter': 'best'}
```

Authors

Pratiksha Verma

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2022-11-09	1.0	Pratiksha Verma	Converted initial version to Jupyterlite